Eye tracking technologies to analyze and visualize the behavior of secure coders

Daniel Kyle Davis

Follow this and additional works at: https://louis.uah.edu/uah-dissertations

Recommended Citation
Davis, Daniel Kyle, "Eye tracking technologies to analyze and visualize the behavior of secure coders" (2023). Dissertations. 375.
https://louis.uah.edu/uah-dissertations/375

This Dissertation is brought to you for free and open access by the UAH Electronic Theses and Dissertations at LOUIS. It has been accepted for inclusion in Dissertations by an authorized administrator of LOUIS.
EYE TRACKING TECHNOLOGIES TO
ANALYZE AND VISUALIZE THE
BEHAVIOR OF SECURE CODERS

Daniel Kyle Davis

A DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in
Computer Science
to
The Graduate School
of
The University of Alabama in Huntsville
December 2023

Approved by:
Dr. Feng Zhu, Research Advisor/Committee Chair
Dr. Haeyong Chung, Committee Member
Dr. Jacob Hauenstein, Committee Member
Dr. Chaity Banerjee, Committee Member
Dr. Jodi Price, Committee Member
Dr. Letha Etzkorn, Department Chair
Dr. Rainer Steinwandt, College Dean
Dr. Jon Hakkila, Graduate Dean
Abstract

EYE TRACKING TECHNOLOGIES TO ANALYZE AND VISUALIZE THE BEHAVIOR OF SECURE CODERS

Daniel Kyle Davis

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Computer Science
The University of Alabama in Huntsville
December 2023

Secure coders’ experiences and their proficiency vary greatly, and any overlooked software security flaws in code can lead to costly repercussions in deployed software applications. The techniques that secure coders utilize to analyze source code and develop mitigation strategies for security flaws are not well understood. Gaining a proper understanding of how coders approach finding and mitigating security flaws can help us efficiently and accurately discover and resolve such issues. One potentially beneficial technique is to collect, analyze, and visualize eye gazes that capture their coding patterns and behaviors. Our systematic literature survey focused on published methods for multiple types of static and dynamic changing eye tracking stimuli, with a particular emphasis on techniques using multiple participant-editable types of stimuli presented simultaneously to simulate a realistic software coding experience. Our work proposes an eye tracking design and analysis framework that breaks down the various stages of software coding. Our decision matrix maps objectives for software programming to analyze techniques for comparing eye gazes among software developers. This involved investigating the limitations of current visualization methods, specifically for user-controlled dynamic stimuli. Our investigation involved using
eye tracking technologies to capture how developers write code, use tools, and read natural language documents and instructions. The study encompassed a wide range of tasks, including simultaneously reading documentation, writing code, and using security source coding analysis tools. Software developer tasks and individual actions create complexity in designing eye tracking experiments and analyzing the collected eye gazes. Our approach allows us to explore behaviors across a range of tasks for a single secure coder and among different coders. New visualization techniques were developed to investigate behaviors during secure coding tasks including methods to present transitions among components within and between applications, as well as present coders’ attention levels during secure coding. Our contributions include a literature survey, framework design, secure coding learning modules, scrollable and modifiable eye tracking stimuli analysis, pupil diameter changes analysis, and stimuli presented in different sequences based on individual participants’ behavior. Our contributions focus on comparing and contrasting multiple visualization methods for eye tracking stimuli.
Acknowledgements

I would like to express my appreciation and gratitude to Dr. Feng Zhu for providing fundamental guidance, continuous encouragement and endless opportunities while working on my graduate research at The University of Alabama in Huntsville (UAH). I have had the great honor to work under his supervision for my Master’s in Computer Science as well as my Doctor of Philosophy in Computer Science. His creativity in research and development as well as his enthusiasm regarding cybersecurity education has advanced my knowledge in performing and presenting innovative research in multiple conferences and peer-reviewed journal publications. His influential mentoring and motivation during my research and experiments as well as his passion for research and exploration have allowed me to successfully fulfill my graduate studies at UAH. His guidance allowed the research to endure during the COVID-19 pandemic with advising via remote conferencing.

I am truly grateful for the guidance towards the successful achievement of the Doctor of Philosophy in Computer Science from the professors in the Computer Science department at UAH as well as the Psychology, Math and Industrial & Systems Engineering departments. I am grateful to my graduate panel members, Dr. Haeyong Chung, Dr. Jacob Hauenstein, Dr. Chaity Banerjee and Dr. Jodi Price, for their encouragement and generous time spent while working with me on the Doctor of Philosophy in Computer Science. Furthermore, Dr. Letha Etzkorn, Dr. Harry Delugach, Dr. Huaming Zhang, Dr. Timothy Newman and Dr. Vineetha Menon provided support towards the successful completion of my graduate coursework.

I would like to express thanks to Betty Nelson, Maryann Bierer, Tamara (Tami) Lang and Joshua Jones of the Computer Science Department faculty who helped me whenever I needed it while working on my Doctor of Philosophy in Com-
puter Science. Dr. Jodi Price of the Psychology Department at UAH for providing her expertise and guidance. Dr. Zhu and Dr. Price experience was vital to the success of my Doctor of Philosophy in Computer Science. Several classmates in the Computer Science department provided me with support and encouragement in multiple courses during my graduate studies. Thanks to all students in the Computer Science Computer Security course in the Fall 2019 semester for their participation in our secure coding eye tracking research data collection study. I am extremely grateful for the staff and computing resources available at the Alabama Supercomputer Authority (ASA) for the usage of the High-Performance Computing clusters for analyzing data collected during my dissertation research.
# Table of Contents

Abstract ................................................................. ii  

Acknowledgements ..................................................... v  

Table of Contents .................................................... xii  

List of Figures .......................................................... xiii  

List of Tables ............................................................ xxvi  

List of Acronyms ........................................................ xxvii  

Epigraph ................................................................. xxix  

Chapter 1. Introduction ................................................ 1  

Chapter 2. Background and Related Work ........................... 15  
  2.1 Investigate Secure Coder Behaviors’ to Analyze and Under-
      stand Their Secure Coding Skills ............................... 15  
  2.2 What are Eye Tracking Technologies? ............................ 19  
  2.3 Eye Tracking Visualization Literature Review .................. 31  
      2.3.1 Literature Survey Methodology ............................ 35
<table>
<thead>
<tr>
<th>2.3.2</th>
<th>Existing Discoveries with Eye Tracking</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3.3</td>
<td>Eye Tracking Discoveries for Software Development</td>
<td>41</td>
</tr>
<tr>
<td>2.3.4</td>
<td>Heat Map</td>
<td>49</td>
</tr>
<tr>
<td>2.3.5</td>
<td>Scan Path</td>
<td>54</td>
</tr>
<tr>
<td>2.3.6</td>
<td>Clustering</td>
<td>60</td>
</tr>
<tr>
<td>2.3.7</td>
<td>Fixation Table</td>
<td>64</td>
</tr>
<tr>
<td>2.3.8</td>
<td>Transition Matrix</td>
<td>67</td>
</tr>
<tr>
<td>2.3.9</td>
<td>Radial Transition Graph</td>
<td>75</td>
</tr>
<tr>
<td>2.3.10</td>
<td>Scarf Plot</td>
<td>79</td>
</tr>
<tr>
<td>2.3.11</td>
<td>Summary of Visualization Techniques</td>
<td>81</td>
</tr>
<tr>
<td>2.3.12</td>
<td>Limitations of Current Visualization Techniques</td>
<td>86</td>
</tr>
</tbody>
</table>

**Chapter 3. Eye Tracking Design and Analysis Framework** | 90
| 3.1 | Eye Tracking Design and Analysis Framework | 90 |
| 3.2 | Software Developer Tasks | 101 |
| 3.3 | Types of Stimuli Elements | 103 |
| 3.4 | Viewing Dimensions | 105 |
| 3.5 | Software Developer Control of Stimuli | 105 |
| 3.6 | Areas of Interest (AOIs) | 107 |
| 3.7 | Quantitative vs Qualitative Analysis | 108 |
3.8 Analysis and Comparison of Developers .................................. 110

Chapter 4. Secure Coding Learning Module Methodology ........ 112

4.1 Designing Learning Modules for Secure Coding .................. 112
4.2 Designing Eye Tracking Stimuli from Secure Coding Learning Modules ......................................................... 116
4.3 Static Stimuli for Reading of Documentation ...................... 121
4.4 Static Stimuli for Reading of Source Code ......................... 122
4.5 Dynamic Stimuli for Reading of Source Code ...................... 123
4.6 User Interactive Stimuli for Patching of Vulnerabilities ........ 124
4.7 Secure Coding Learning Modules in an Eye Tracking Timeline 126

Chapter 5. Eye Gaze Data Collection and Processing .............. 128

5.1 Procedure and Tasks Step-by-Step ................................. 128
5.2 Participants in Our Secure Coding Eye Tracking Study .......... 138
5.3 Eye Tracking Filtering Techniques ................................. 142
5.4 Manually Creating Areas of Interest (AOIs) ....................... 144
5.5 Analyzing with Tobii Studio Metrics ............................. 147
5.6 Exporting Tobii Studio Eye Gaze Data ........................... 149
5.7 Creation of Custom Analysis Scripts ............................. 150

Chapter 6. Visualization Analysis of Secure Coding Behavior .... 152
6.1 Data Analysis and Visualization Methodologies .......................... 153
6.2 Analyzing Static Stimuli with Simple Areas of Interest (AOIs) 154
6.3 Comparing Static Stimuli for Multiple Participants with Simple 
Areas of Interest (AOIs) ...................................................... 163
6.4 Analyzing Static Stimuli with Detailed Areas of Interest (AOIs) 168
6.5 Comparing Static Stimuli for Multiple Participants with De-
tailed Areas of Interest (AOIs) ............................................. 172
6.6 Analyzing Multiple User Interactive Stimuli Content ............... 174
6.7 Comparing User Interactive Stimuli of Multiple Participants . 184
6.8 Visualization of our Eye Pupil Diameter Adjustments .............. 192
6.9 Other Approaches .......................................................... 202
6.10 Summary of Visualization Methods ................................. 202
6.11 Framework Summary ..................................................... 214

Chapter 7. Eye Tracking Metrics Analysis of Secure Coder 
Behaviors ............................................................................. 217
7.1 Secure Coders’ Programming Assignment Results .............. 217
7.2 Comparing Eye Movements in Secure Coding .................... 225
7.3 Statistical Analysis .......................................................... 229

Chapter 8. Insights into Secure Coder Behaviors ....................... 236
8.1 Secure Coders’ Behaviors Analysis ..................................... 236
8.2 Insight Captured from the Analysis of Secure Coders’ Behaviors 238

8.2.1 Did Secure Coders Struggle to Utilize the RIPS Static Code Analysis Tool for Their First Use Case of the RIPS Tool? 239

8.2.2 Did Secure Coders Answer Correctly Because They Found the Coding Flaw? 240

8.2.3 Did Secure Coders Look at the Coding Flaw for CWE-73 but Still Answered Incorrectly That CWE-73 is Not an Issue in ShareAlbum? 241

8.2.4 Did Secure Coders Respond Incorrectly Due to Not Examining the Code Containing the Security Flaw? 242

8.2.5 Do Secure Coders Practice Efficient Coding by Reviewing the Whole Code Before Focusing on a Specific Function, or Do They Inefficiently Jump Straight to the Function? 243

Chapter 9. Conclusions 244

Chapter 10. Future Work 253

References 255

Appendix A: Institutional Review Board and Student Consent 273

Appendix B: Setup of the Eye Tracking Equipment 285

Appendix C: Student Questionnaire Survey 287
Appendix D: Secure Coding Exercises ........................................... 363

D.1 CWE-311: Missing Encryption of Sensitive Data ..................... 363
D.2 CWE-434: Unrestricted Upload of File with Dangerous Type .......... 365
D.3 CWE-862: Missing Authorization ........................................... 367
D.4 CWE-22: Improper Limitation of a Pathname to a Restricted Directory (‘Path Traversal’) ........................................... 369
D.5 CWE-443: HTTP Response Splitting ....................................... 372
D.6 CWE-73: External Control of File Name or Path ....................... 374
D.7 CWE-79: Improper Neutralization of Input During Web Page Generation (‘Cross-site Scripting’) ....................................... 379

Appendix E: Secure Coder Behaviors ........................................... 383

E.1 High-level Summary of Secure Coders’ Behaviors ..................... 383
E.2 Secure Coders’ Behaviors When Reading Learning Content for CWE-73 ........................................... 393
E.3 Secure Coders’ Behaviors When Working Exercise for CWE-73 398
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Fixation and Saccade Mock-up</td>
<td>22</td>
</tr>
<tr>
<td>2.2</td>
<td>Mock-up of Eye Tracking Setup with Monitor and User</td>
<td>23</td>
</tr>
<tr>
<td>2.3</td>
<td>Tobii Pro X2-60 Eye Tracker Mounted on Desktop Monitor or Laptop</td>
<td>25</td>
</tr>
<tr>
<td>2.4</td>
<td>Tobii Studio Design and Record Tab</td>
<td>26</td>
</tr>
<tr>
<td>2.5</td>
<td>Tobii Studio Visualizations Tab</td>
<td>27</td>
</tr>
<tr>
<td>2.6</td>
<td>Tobii Studio Areas of Interest Tab</td>
<td>28</td>
</tr>
<tr>
<td>2.7</td>
<td>Tobii Studio Statistics Tab</td>
<td>28</td>
</tr>
<tr>
<td>2.8</td>
<td>Selection of Papers from the Literature Survey</td>
<td>37</td>
</tr>
<tr>
<td>2.9</td>
<td>Heat Map for Static Stimuli for Single Software Developer for 57 milliseconds</td>
<td>51</td>
</tr>
<tr>
<td>2.10</td>
<td>Heat Map for Static Stimuli for Single Software Developer</td>
<td>52</td>
</tr>
<tr>
<td>2.11</td>
<td>Heat Map for Static Stimuli for Multiple Software Developers Example 1</td>
<td>53</td>
</tr>
<tr>
<td>2.12</td>
<td>Heat Map for Static Stimuli for Multiple Software Developers Example 2</td>
<td>54</td>
</tr>
<tr>
<td>2.13</td>
<td>Scan Path for Static Stimuli for Single Software Developer for 57 milliseconds</td>
<td>57</td>
</tr>
<tr>
<td>2.14</td>
<td>Scan Path for Static Stimuli for Single Software Developer for full recording</td>
<td>58</td>
</tr>
<tr>
<td>2.15</td>
<td>Scan Path for Static Stimuli for Multiple Software Developers for 57 milliseconds</td>
<td>59</td>
</tr>
<tr>
<td>2.16</td>
<td>Scan Path for Static Stimuli for 29 Software Developers for full recording</td>
<td>60</td>
</tr>
<tr>
<td>2.17</td>
<td>Clustering with High Spatial Distance Threshold for All Software Developers viewing Static Stimuli Content</td>
<td>62</td>
</tr>
</tbody>
</table>
2.18 Clustering with Low Spatial Distance Threshold for All Software Developers viewing Static Stimuli Content ........................................... 63
2.19 Clustering with Low Spatial Distance Threshold for All Software Developers viewing Static Stimuli Content with Mouse Clicked Locations Overlayed ........................................... 64
2.20 Fixation Table of a Single Developer Shown as a Count and a Percentage when Reading Source Code displayed using Static Stimuli Content ........................................................................ 66
2.21 Fixation Table of a Single Developer as a Count and a Percentage when Reading and Writing Source Code as well as reviewing reports from Software Scanning Software displayed using Non-Linear Dynamic Stimuli Content ........................................................................ 67
2.22 Transition Matrix Shown as a Transition Count and a Transition Percentage when Reading Source Code presented as Static Stimuli Content ........................................................................ 70
2.23 Programming Solution Option A when Reading Source Code presented as Static Stimuli Content ........................................................................ 73
2.24 Programming Solution Option B when Reading Source Code presented as Static Stimuli Content ........................................................................ 74
2.25 Programming Solution Option C when Reading Source Code presented as Static Stimuli Content ........................................................................ 75
2.26 Radial Transition Graph Example for three Software Developers when Reading Documentation presented as Static Stimuli Content ........................................................................ 78
2.27 Radial Transition Graph Grid View for Multiple Software Developers when Reading and Writing Source Code as well as reviewing reports from Software Scanning Software viewed using Static Stimuli Content ........................................................................ 78
2.28 Scarf Plots for a Software Developer presented with a Time-based Measurement when viewed using Non-Linear Dynamic Stimuli Content Example 1 ........................................................................ 82
2.29 Scarf Plots for a Software Developer presented with a Transition-based Count when viewed using Non-Linear Dynamic Stimuli Content Example 1 ........................................................................ 83
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.10</td>
<td>Static Code Analysis Tool Learning Module Participant Demographics</td>
<td>141</td>
</tr>
<tr>
<td>5.11</td>
<td>Creating AOIs for Static Stimuli Content</td>
<td>145</td>
</tr>
<tr>
<td>5.12</td>
<td>Creating AOIs for Dynamic Stimuli Content</td>
<td>147</td>
</tr>
<tr>
<td>5.13</td>
<td>Tobii Studio Statistics Tab for Table Representation</td>
<td>148</td>
</tr>
<tr>
<td>5.14</td>
<td>Tobii Studio Statistics Tab for Chart Representation</td>
<td>148</td>
</tr>
<tr>
<td>5.15</td>
<td>Data Export Tab in Tobii Studio</td>
<td>150</td>
</tr>
<tr>
<td>5.16</td>
<td>Example of a Raw Eye Gaze Datafile from Tobii Studio</td>
<td>151</td>
</tr>
<tr>
<td>6.1</td>
<td>Example of a Static Stimuli Presented to a Participant using Tobii</td>
<td>155</td>
</tr>
<tr>
<td></td>
<td>Studio PDF Stimuli</td>
<td></td>
</tr>
<tr>
<td>6.2</td>
<td>Example of a Static Stimuli with Overlay for a Single AOI</td>
<td>156</td>
</tr>
<tr>
<td>6.3</td>
<td>Static Stimuli Data Content (Page 1 of 4) for CWE-434</td>
<td>158</td>
</tr>
<tr>
<td>6.4</td>
<td>Static Stimuli Data Content (Page 2 of 4) for CWE-434</td>
<td>159</td>
</tr>
<tr>
<td>6.5</td>
<td>Static Stimuli Data Content (Page 3 of 4) for CWE-434</td>
<td>160</td>
</tr>
<tr>
<td>6.6</td>
<td>Static Stimuli Data Content (Page 4 of 4) for CWE-434</td>
<td>161</td>
</tr>
<tr>
<td>6.7</td>
<td>Questionnaire for CWE-434 for multiple choice answer</td>
<td>162</td>
</tr>
<tr>
<td>6.8</td>
<td>AOI Transition Line Plot for Static Stimuli Learning Content</td>
<td>164</td>
</tr>
<tr>
<td></td>
<td>Example 1</td>
<td></td>
</tr>
<tr>
<td>6.9</td>
<td>AOI Transition Line Plot for Static Stimuli Learning Content</td>
<td>165</td>
</tr>
<tr>
<td></td>
<td>Example 2</td>
<td></td>
</tr>
<tr>
<td>6.10</td>
<td>Fixation Durations and Number of Fixations Plot for Static Stimuli</td>
<td>167</td>
</tr>
<tr>
<td>6.11</td>
<td>Comparison of four participants for CWE-79 using the Radial Transition Graph</td>
<td>169</td>
</tr>
<tr>
<td>6.12</td>
<td>Example from CWE-434 of creating a Single AOI for Software Code for Static Stimuli</td>
<td>170</td>
</tr>
<tr>
<td>6.13</td>
<td>Example from CWE-434 of creating Multiple AOIs for Software Code for Static Stimuli</td>
<td>171</td>
</tr>
</tbody>
</table>

xvi
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.14</td>
<td>Static PDF Stimuli Data Analysis for CWE-434 with a detailed AOI</td>
<td>172</td>
</tr>
<tr>
<td>6.15</td>
<td>AOI Transition Plot for Static Stimuli for CWE-434 with a multiple AOI</td>
<td>175</td>
</tr>
<tr>
<td>6.16</td>
<td>AOI Transition Plot for Static Stimuli for CWE-434 with a multiple AOI</td>
<td>176</td>
</tr>
<tr>
<td>6.17</td>
<td>AOI Transition Plot for Static Stimuli for CWE-434 with a multiple AOI</td>
<td>177</td>
</tr>
<tr>
<td>6.18</td>
<td>AOI Transition Plot for Static Stimuli for CWE-434 with a multiple AOI</td>
<td>178</td>
</tr>
<tr>
<td>6.19</td>
<td>AOI Transition Plot for Static Stimuli for CWE-434 with a multiple AOI</td>
<td>179</td>
</tr>
<tr>
<td>6.20</td>
<td>AOI Transition Plot for Static Stimuli for CWE-434 with a multiple AOI</td>
<td>180</td>
</tr>
<tr>
<td>6.21</td>
<td>Example of Fixation Matrix and Transition Plot for CWE-79</td>
<td>183</td>
</tr>
<tr>
<td>6.22</td>
<td>PlantUML Swimlane Representation Example 1</td>
<td>186</td>
</tr>
<tr>
<td>6.23</td>
<td>PlantUML Swimlane Representation Example 2</td>
<td>187</td>
</tr>
<tr>
<td>6.24</td>
<td>Transition Matrix for CWE-79 for One Student</td>
<td>189</td>
</tr>
<tr>
<td>6.25</td>
<td>Complex PlantUML State Representation</td>
<td>191</td>
</tr>
<tr>
<td>6.26</td>
<td>Simple PlantUML State Representation</td>
<td>192</td>
</tr>
<tr>
<td>6.27</td>
<td>Boxplot of Eye Pupil for One Student with Number of Fixations Example 1</td>
<td>194</td>
</tr>
<tr>
<td>6.28</td>
<td>Boxplot of Eye Pupil for One Student with Number of Fixations Example 1</td>
<td>195</td>
</tr>
<tr>
<td>6.29</td>
<td>Boxplot of Eye Pupil for One Student with Number of Fixations Example 2</td>
<td>196</td>
</tr>
<tr>
<td>6.30</td>
<td>Boxplot of Eye Pupil for One Student with Number of Fixations Example 2</td>
<td>197</td>
</tr>
<tr>
<td>6.31</td>
<td>Eye Pupil Data for Developers at 250 Millisecond Interval Example 1</td>
<td>199</td>
</tr>
</tbody>
</table>
6.32 Eye Pupil Data for Developers at 250 Millisecond Interval Example 1 (Right Pupil) .................................................. 200
6.33 Eye Pupil Data for Developers at 250 Millisecond Interval Example 2 (Left Pupil) .................................................. 200
6.34 Eye Pupil Data for Developers at 250 Millisecond Interval Example 2 (Right Pupil) .................................................. 201
6.35 Simple Moving Average Mean of Pupil Diameter for a Developer .......................................................... 203
6.36 Swimlane Diagram for Single Software Developer Reading Documentation .................................................. 209
6.37 State Transition Diagram for Single Software Developer Reading Documentation .................................................. 211
6.38 Simple Moving Average (SMA) Mean of Left Pupil Diameter .......................................................... 213

7.1 Manual Code Analysis Learning Module (CWE-311) Participant Questionnaire Results .................................................. 220
7.2 Manual Code Analysis Learning Module (CWE-434) Participant Questionnaire Results .................................................. 220
7.3 Manual Code Analysis Learning Module (CWE-862) Participant Questionnaire Results .................................................. 222
7.4 Manual Code Analysis Learning Module (CWE-22) Participant Questionnaire Results .................................................. 222
7.5 Static Code Analysis Tool Learning Module (CWE-443) Participant Questionnaire Results .................................................. 224
7.6 Static Code Analysis Tool Learning Module (CWE-73) Participant Questionnaire Results .................................................. 224
7.7 Static Code Analysis Tool Learning Module (CWE-79) Participant Questionnaire Results .................................................. 225
7.8 Learning Content for CWE-311 from Five Students that Answered Incorrectly .................................................. 226
7.9 Learning Content for CWE-311 from Five Students that Answered Correctly .................................................. 227
C.16  Student Questionnaire Survey Page 5 of 5 ................. 291
C.17  Manual Code Analysis Module Question 1 ...................... 292
C.18  Static Code Analysis Module Question 1 ...................... 292
C.19  Manual Code Analysis Module Question 2 ...................... 293
C.20  Static Code Analysis Module Question 2 ...................... 293
C.21  Manual Code Analysis Module Question 3 ...................... 294
C.22  Static Code Analysis Module Question 3 ...................... 294
C.23  Manual Code Analysis Module Question 4 ...................... 295
C.24  Static Code Analysis Module Question 4 ...................... 295
C.25  Manual Code Analysis Module Question 5 ...................... 296
C.26  Static Code Analysis Module Question 5 ...................... 296
C.27  Manual Code Analysis Module Question 6 ...................... 297
C.28  Static Code Analysis Module Question 6 ...................... 297
C.29  Manual Code Analysis Module Question 7 ...................... 298
C.30  Static Code Analysis Module Question 7 ...................... 298
C.31  Manual Code Analysis Module Question 8 ...................... 299
C.32  Static Code Analysis Module Question 8 ...................... 299
C.33  Manual Code Analysis Module Question 9 ...................... 300
C.34  Static Code Analysis Module Question 9 ...................... 300
C.35  Manual Code Analysis Module Question 10 ..................... 301
C.36  Static Code Analysis Module Question 10 ..................... 301
C.37  Manual Code Analysis Module Question 11 ..................... 302
C.38  Static Code Analysis Module Question 11 ..................... 302
C.39  Manual Code Analysis Module Question 12 ..................... 303
C.40  Static Code Analysis Module Question 12 ..................... 303
C.41  Manual Code Analysis Module Question 13 ..................... 304

xx
C.42 Static Code Analysis Module Question 13 . . . . . . . . . . . . . 304
C.43 Manual Code Analysis Module Question 14 . . . . . . . . . . . . . 305
C.44 Static Code Analysis Module Question 14 . . . . . . . . . . . . . 305
C.45 Manual Code Analysis Module Question 15 . . . . . . . . . . . . . 306
C.46 Static Code Analysis Module Question 15 . . . . . . . . . . . . . 306
C.47 Manual Code Analysis Module Question 16 . . . . . . . . . . . . . 307
C.48 Static Code Analysis Module Question 16 . . . . . . . . . . . . . 307
C.49 Manual Code Analysis Module Question 17 . . . . . . . . . . . . . 308
C.50 Static Code Analysis Module Question 17 . . . . . . . . . . . . . 308
C.51 Manual Code Analysis Module Question 18 . . . . . . . . . . . . . 309
C.52 Static Code Analysis Module Question 18 . . . . . . . . . . . . . 309
C.53 Manual Code Analysis Module Question 19 . . . . . . . . . . . . . 310
C.54 Static Code Analysis Module Question 19 . . . . . . . . . . . . . 310
C.55 Manual Code Analysis Module Question 20 . . . . . . . . . . . . . 311
C.56 Static Code Analysis Module Question 20 . . . . . . . . . . . . . 311
C.57 Manual Code Analysis Module Question 21 . . . . . . . . . . . . . 312
C.58 Static Code Analysis Module Question 21 . . . . . . . . . . . . . 312
C.59 Manual Code Analysis Module Question 22 . . . . . . . . . . . . . 313
C.60 Static Code Analysis Module Question 22 . . . . . . . . . . . . . 313
C.61 Manual Code Analysis Module Question 23 . . . . . . . . . . . . . 314
C.62 Static Code Analysis Module Question 23 . . . . . . . . . . . . . 314
C.63 Manual Code Analysis Module Question 24 . . . . . . . . . . . . . 315
C.64 Static Code Analysis Module Question 24 . . . . . . . . . . . . . 315
C.65 Manual Code Analysis Module Question 25 . . . . . . . . . . . . . 316
C.66 Static Code Analysis Module Question 25 . . . . . . . . . . . . . 316
C.67 Manual Code Analysis Module Question 26 . . . . . . . . . . . . . 317
<table>
<thead>
<tr>
<th>Question Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>C.68</td>
<td>Static Code Analysis Module Question 26</td>
<td>318</td>
</tr>
<tr>
<td>C.69</td>
<td>Manual Code Analysis Module Question 27.1</td>
<td>319</td>
</tr>
<tr>
<td>C.70</td>
<td>Static Code Analysis Module Question 27.1</td>
<td>319</td>
</tr>
<tr>
<td>C.71</td>
<td>Manual Code Analysis Module Question 27.2</td>
<td>320</td>
</tr>
<tr>
<td>C.72</td>
<td>Static Code Analysis Module Question 27.2</td>
<td>320</td>
</tr>
<tr>
<td>C.73</td>
<td>Manual Code Analysis Module Question 27.3</td>
<td>321</td>
</tr>
<tr>
<td>C.74</td>
<td>Static Code Analysis Module Question 27.3</td>
<td>321</td>
</tr>
<tr>
<td>C.75</td>
<td>Manual Code Analysis Module Question 27.4</td>
<td>322</td>
</tr>
<tr>
<td>C.76</td>
<td>Static Code Analysis Module Question 27.4</td>
<td>322</td>
</tr>
<tr>
<td>C.77</td>
<td>Manual Code Analysis Module Question 27.5</td>
<td>323</td>
</tr>
<tr>
<td>C.78</td>
<td>Static Code Analysis Module Question 27.5</td>
<td>323</td>
</tr>
<tr>
<td>C.79</td>
<td>Manual Code Analysis Module Question 27.6</td>
<td>324</td>
</tr>
<tr>
<td>C.80</td>
<td>Static Code Analysis Module Question 27.6</td>
<td>324</td>
</tr>
<tr>
<td>C.81</td>
<td>Manual Code Analysis Module Question 28.1</td>
<td>325</td>
</tr>
<tr>
<td>C.82</td>
<td>Static Code Analysis Module Question 28.1</td>
<td>325</td>
</tr>
<tr>
<td>C.83</td>
<td>Manual Code Analysis Module Question 28.2</td>
<td>326</td>
</tr>
<tr>
<td>C.84</td>
<td>Static Code Analysis Module Question 28.2</td>
<td>326</td>
</tr>
<tr>
<td>C.85</td>
<td>Manual Code Analysis Module Question 28.3</td>
<td>327</td>
</tr>
<tr>
<td>C.86</td>
<td>Static Code Analysis Module Question 28.3</td>
<td>327</td>
</tr>
<tr>
<td>C.87</td>
<td>Manual Code Analysis Module Question 28.4</td>
<td>328</td>
</tr>
<tr>
<td>C.88</td>
<td>Static Code Analysis Module Question 28.4</td>
<td>328</td>
</tr>
<tr>
<td>C.89</td>
<td>Manual Code Analysis Module Question 28.5</td>
<td>329</td>
</tr>
<tr>
<td>C.90</td>
<td>Static Code Analysis Module Question 28.5</td>
<td>329</td>
</tr>
<tr>
<td>C.91</td>
<td>Manual Code Analysis Module Question 28.6</td>
<td>330</td>
</tr>
<tr>
<td>C.92</td>
<td>Static Code Analysis Module Question 28.6</td>
<td>330</td>
</tr>
<tr>
<td>C.93</td>
<td>Manual Code Analysis Module Question 28.7</td>
<td>331</td>
</tr>
<tr>
<td>Question</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>C.94</td>
<td>331</td>
<td></td>
</tr>
<tr>
<td>C.95</td>
<td>332</td>
<td></td>
</tr>
<tr>
<td>C.96</td>
<td>332</td>
<td></td>
</tr>
<tr>
<td>C.97</td>
<td>333</td>
<td></td>
</tr>
<tr>
<td>C.98</td>
<td>333</td>
<td></td>
</tr>
<tr>
<td>C.99</td>
<td>334</td>
<td></td>
</tr>
<tr>
<td>C.100</td>
<td>334</td>
<td></td>
</tr>
<tr>
<td>C.101</td>
<td>335</td>
<td></td>
</tr>
<tr>
<td>C.102</td>
<td>335</td>
<td></td>
</tr>
<tr>
<td>C.103</td>
<td>336</td>
<td></td>
</tr>
<tr>
<td>C.104</td>
<td>336</td>
<td></td>
</tr>
<tr>
<td>C.105</td>
<td>337</td>
<td></td>
</tr>
<tr>
<td>C.106</td>
<td>337</td>
<td></td>
</tr>
<tr>
<td>C.107</td>
<td>338</td>
<td></td>
</tr>
<tr>
<td>C.108</td>
<td>338</td>
<td></td>
</tr>
<tr>
<td>C.109</td>
<td>339</td>
<td></td>
</tr>
<tr>
<td>C.110</td>
<td>339</td>
<td></td>
</tr>
<tr>
<td>C.111</td>
<td>340</td>
<td></td>
</tr>
<tr>
<td>C.112</td>
<td>340</td>
<td></td>
</tr>
<tr>
<td>C.113</td>
<td>341</td>
<td></td>
</tr>
<tr>
<td>C.114</td>
<td>341</td>
<td></td>
</tr>
<tr>
<td>C.115</td>
<td>342</td>
<td></td>
</tr>
<tr>
<td>C.116</td>
<td>342</td>
<td></td>
</tr>
<tr>
<td>C.117</td>
<td>343</td>
<td></td>
</tr>
<tr>
<td>C.118</td>
<td>343</td>
<td></td>
</tr>
<tr>
<td>C.119</td>
<td>344</td>
<td></td>
</tr>
</tbody>
</table>

xxiii
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Visualization Techniques for Graphically Analyzing Eye Gazes and Software Developer Behaviors Set 1</td>
<td>99</td>
</tr>
<tr>
<td>3.2</td>
<td>Visualization Techniques for Graphically Analyzing Eye Gazes and Software Developer Behaviors Set 2</td>
<td>100</td>
</tr>
<tr>
<td>5.1</td>
<td>Total Duration Recording Times per Secure Code Learning Module</td>
<td>132</td>
</tr>
<tr>
<td>6.1</td>
<td>Eye Tracking Visualization Methods Set 1</td>
<td>205</td>
</tr>
<tr>
<td>6.2</td>
<td>Eye Tracking Visualization Methods Set 2</td>
<td>206</td>
</tr>
<tr>
<td>7.1</td>
<td>Statistics Table for Learning Content AOI</td>
<td>234</td>
</tr>
</tbody>
</table>
## List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACM</td>
<td>Association for Computing Machinery</td>
</tr>
<tr>
<td>AOI</td>
<td>Area of Interest</td>
</tr>
<tr>
<td>AOIs</td>
<td>Areas of Interest</td>
</tr>
<tr>
<td>ASA</td>
<td>Alabama Supercomputer Authority</td>
</tr>
<tr>
<td>CERIAS</td>
<td>Center for Education and Research in Information Assurance and Security</td>
</tr>
<tr>
<td>CSV</td>
<td>Comma Separate Values</td>
</tr>
<tr>
<td>CWE</td>
<td>Common Weakness Enumeration</td>
</tr>
<tr>
<td>CWEs</td>
<td>Common Weakness Enumerations</td>
</tr>
<tr>
<td>EMIP</td>
<td>Eye Movements in Programming</td>
</tr>
<tr>
<td>ETRA</td>
<td>Eye Tracking Research and Applications Symposium</td>
</tr>
<tr>
<td>HTML</td>
<td>HyperText Markup Language</td>
</tr>
<tr>
<td>NVD</td>
<td>National Vulnerability Database</td>
</tr>
<tr>
<td>OWASP</td>
<td>Open Web Application Security Project</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>PHP</td>
<td>PHP: Hypertext Preprocessor</td>
</tr>
<tr>
<td>PNG</td>
<td>Portable Network Graphics</td>
</tr>
<tr>
<td>RIPS</td>
<td>Research and Innovation to Promote Security</td>
</tr>
<tr>
<td>SDLC</td>
<td>Software Development Life Cycle</td>
</tr>
<tr>
<td>SEI</td>
<td>Software Engineering Institute</td>
</tr>
<tr>
<td>SMA</td>
<td>Simple Moving Average</td>
</tr>
<tr>
<td>SVG</td>
<td>Scalable Vector Graphics</td>
</tr>
<tr>
<td>UAH</td>
<td>The University of Alabama in Huntsville</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
</tbody>
</table>
You can teach a student a lesson for a day; but if you can teach him to learn by creating curiosity, he will continue the learning process as long as he lives.

- Clay P. Bedford
Chapter 1. Introduction

Secure coding is a mission that cannot be neglected as computing devices continue to increase. Every year, thousands of new software vulnerabilities are identified. Education is a crucial factor and a significant challenge to counter cyber threats. There is a considerable need for improved methodologies with active hands-on educational techniques for programmers to learn practical strategies to mitigate software vulnerabilities; to protect private data; and ultimately write secure code in the first place. But gaining insight into how people learn has always been challenging. To the best of our knowledge, this is the first usage of eye tracking technologies to understand secure coding practices and gain insight into the behaviors of secure coders. Our research focused on exploring the ways that students comprehended and learned to develop secure software. This included recording their eye gaze movements while they studied our hands-on secure coding learning modules and mitigated various security weaknesses within source code.

Software security vulnerabilities can exist in several aspects of source code: the lack of focus on security threats when an application is developed, using open source or third-party components, being more interconnected, and sharing user data over network devices and accounts [1]. Security vulnerabilities dam-
age integrity, confidentiality, and availability. Secure coding is a major aspect to software assurance and vulnerability management [2, 3]. Secure coders often examine various aspects of a programming code including: syntax, semantics, program flow, and overall program execution. They typically focus on finding security weaknesses and then determining the appropriate mitigation techniques. With diverse skills, secure coders may approach discovering and fixing vulnerabilities very differently and inefficiently. It is better to develop source code that is secure during the software development life-cycle before publication as the cost to mitigate after the software is published is typically higher. Resources that help with secure coding includes vulnerability databases [4–6], code reviews [7, 8], secure coding standards [9–12] and as well as tool suites [13–15]. While secure coding becomes increasingly important, secure coding behavior is not well understood. There is a lack of understanding of secure coder behavior when the coder applies the various techniques to mitigate flaws or utilizes secure coding tools and resources. Understanding secure coding behavior and where challenges occur during coding can help us understand where to improve learning content and therefore improve secure coders’ abilities. These methods could potentially be improved by an understanding of secure coding behavior and how coders approach secure coding problems [16]. Our research is focused on studying the behavior of secure coders in order to bring improvements to the secure coding process.

The creation of learning content and hands-on learning modules to help facilitate teaching is a fundamental and critical component of learning [17, 18]. Various industry researchers and university scholars have contributed towards
developing content for learning secure software development principles. The Software Engineering Institute (SEI) at Carnegie Mellon University has created several curricula based upon the experience level of participants. Each curriculum contains a set of tools, procedures, and recommended tactics for how to develop learning material to help secure coders discover and mitigate security weaknesses in software applications [19]. Researchers at Purdue University have created the Center for Education and Research in Information Assurance and Security (CERIAS) with several principal investigators researching secure programming education [20]. Researchers at Towson University have developed workshops aimed at security injections in source code [21]. The Georgia Institute of Technology has created the Secure Software Development course that is offered as a three-day online course [22]. At Syracuse University, researchers have developed the SEED-LABS 2.0 to provide tasks in labs and workshops, video lectures, and references for common vulnerabilities in software [23]. In 2021, Open Web Application Security Project (OWASP) released their secure coding training platform identified as Dojo after it was given to OWASP from Trend Micro [24]. SAFECode provides security training courses that are publicly available as webcast [25]. Other companies provide secure coding training; however, several companies typically have a fee for the complete suite of training and tools [26–29].

Our approach is to educate students with free and publicly available learning modules that guide participants step-by-step, but still provide the opportunity to mitigate security coding flaws with hands-on experience. We developed our modules to help coders learn practical strategies by first identifying and then
mitigating vulnerabilities in source code. Our hands-on approach involved using a photo and video web sharing application called ShareAlbum. We used vulnerabilities from the Common Weakness Enumeration (CWE) repository to facilitate participants learning about weaknesses and vulnerabilities that are determined by industry to have a high importance [6]. The CWE repository provides a list of common weaknesses and vulnerability types as well as example code and exploits that could apply to several programming languages [30]. We also introduced specific CWEs from the OWASP Top 10 list and the National Vulnerability Database (NVD) [31, 32]. Our learning modules illustrated coding instances of specific CWEs and showed exploitation methods and security mitigation techniques. Finally, each secure coding learning module concluded with an overall discussion.

One of our objectives is to understand how participants use our secure coding learning modules when they work on secure coding exercises.

Several techniques could be utilized to understand what students are truly learning while working on secure coding. Often tests or exams are given that attempt to gauge the content that students comprehend from lectures. Essays or projects are another ordinary form in some fields to gauge student understanding. Interviews can provide more information about students’ thinking processes and their level of understanding that might be difficult to collect with paper exams or essays [33]. A traditional approach of reviewing security flaws in a PowerPoint presentation does not provide the students with hands-on experience to gain knowledge and then immediately apply their knowledge [34]. Programming assignments are commonly used in software development coursework [35]. Sev-
eral of these methods require designers of the material to correctly set up the questions and assignments to determine the level of understanding [36]. These methods are often specific to certain fields of study and depend upon the group of participants. These methods also lack the ability to objectively collect the behavior of participants [34]. Additional methods to objectively study learning behaviors are needed to provide a more in-depth understanding.

The purpose of our study is to examine the effectiveness of the teaching material by studying students and their behaviors when they locate and mitigate security flaws in code. Our research is focused on the collection and analysis techniques of participants’ behaviors as they learn how to develop secure software. The objective of understanding students’ learning behavior is critical to developing effective learning modules and education methodologies.

Researching the behavior and patterns of individuals performing tasks can allow for a greater understanding of how individuals learn new concepts and the methods individuals use to solve problems. Existing methods of investigating behaviors often utilize think-aloud sessions, questionnaires, or verbal reviews, but secure coders’ ability is stored in skill memories, which cannot always be verbalized [37]. A limitation of commonly used techniques such as questionnaires and verbal reviews is the potential to have a bias in the feedback individuals provide on how they view the various stimuli content. A concurrent think-aloud study may limit the ability of participants to focus on the stimuli content and tasks. Consequently disturbing their attention and focus on the stimuli content [38]. Furthermore, individuals may not remember each aspect of the detail that is
needed to gain a full understanding of their behavior. Therefore, it is difficult to collect and analyze behavior and patterns via questionnaires or verbal reviews in an objective manner. Direct observations do not require participants to recall or verbally explain their techniques during secure coding, but they are unnatural and distracting to participants [39]. Researchers are examining if social psychologists could benefit from using eye tracking as a method to objectively collect feedback but prevent the challenges from using individual describing and reporting techniques such as interviews, self-assessment reports or verbal and written documentation [40, 41]. Several common statistical and various visualization methods have been utilized significantly in the basic reading of paragraphs from books as well as math problems [42, 43]. A discreet and unobtrusive technique is needed to observe secure coders’ behaviors while not requiring participants to recall or verbally explain their techniques during secure coding. Our research focuses on using technology to assist in the data collection of eye gazes over stimuli content to gain an understanding of how individuals approach learning and problem-solving tasks that require their attention to switch rapidly between different software applications.

One of our research objectives is to determine if eye tracking can be utilized to help understand the eye gaze patterns, behavioral patterns, and learning techniques of software developers as they work to solve software problems [44]. Our particular interest is in determining if this can help to provide an objective method to understand how individuals approach secure coding problem solving and how they learn new concepts. Specifically, we concentrated on researching
how software developers perform coding activities to learn, discover, and mitigate software flaws and coding bugs in software applications. Applying eye tracking technology when developers work on secure coding exercises can allow us to visually investigate the areas of source code and toolsets that participants utilized to resolve software flaws in source code.

Eye tracking technologies permit researchers to collect, analyze, and visualize the viewing behavior of participants (e.g., read the text and browse the Internet). Eye tracking technologies allow researchers to observe participants’ eye gazes objectively [45]. The analysis of eye tracking results allows for investigating participants’ reading patterns and behavior across distinct types of visual stimuli without requiring participants to verbally communicate their actions. One of our objectives is to determine whether eye tracking will allow for investigating software developers’ behavior and patterns without requiring the developer to verbally communicate or recall from memory their techniques.

Our eyes are comprised of various structures such as our pupil, retina, fovea, and various other components to help us visualize objects and colors [46]. Studying changes of pupil sizes, gaze location, pattern of eye movements allows researchers to examine participants’ driving behavior [47, 48], to determine attention in advertising media [49], to predict attention in playing chess [50], to study skills of flying and landing airplanes [51–53], and to examine software Unified Modeling Language (UML) class diagrams [54]. If participants are required to provide real-time verbal feedback, then that could be a considerable increase in the cognitive load while working on tasks [55]. Prior research has found that eye
tracking alone cannot directly determine the amount of cognitive load on individuals [55, 56]. However, research is ongoing in using eye tracking in combination with questionnaires and various stimuli to examine the behavior and patterns of participants [57]. Our research study uses eye tracking technology to study the behavior of secure coders. In addition, participants were presented with questions and secure coding exercises while their eye gazes were recorded.

We investigated secure coders’ eye fixations and transitions with visualizations. A fixation occurs when a participant’s eye is relatively fixed at one point and their eye has no movement. When viewed on a computer monitor, then a fixation is recorded as an X-point and a Y-point for a 2-dimensional viewing scene along with a timestamp. This allows researchers to study the eye gaze of participants and determine when in time and where in the scene the eye gaze occurred. This can allow researchers to observe the viewing pattern of participants which is useful to determine the attention that participants have given to specific parts of a reading or visualization stimuli [55]. Transitions and order of stimuli visited can give insight into the behavior and methods that participants used when reading and watching stimuli. When the eyes are rapidly moving between two fixations, a saccade occurs [58]. Fixations typically last for milliseconds to seconds [57]. The length of each saccade is noticeably short and is in the order of a few milliseconds. Saccades are often utilized to help determine the amount of attention or changes in attention [59]. Raw eye tracking data that are collected during an experiment including fixations and saccades can be very time-consuming to process and complex to understand. A commonly adopted technique includes creating Areas
of Interest (AOIs) on presented stimuli. This allows a research analyst to add specific regions of interest to the stimuli that were presented to participants [54]. The transitions between AOIs are important to analyze as well as the fixations that occur inside of an AOI. An AOI is generally created manually by drawing on top of the stimuli content [59]. This can help to identify patterns and tactics that participants may utilize to approach a problem but could also help to identify anomalies.

Visualization techniques of eye gazes can provide valuable insight into the techniques that individuals use to approach problem solving such as secure coding. We determined after reviewing numerous research publications on eye tracking for software application coding that the published literature did not provide significant details on using eye tracking to study the behavior and patterns of developers performing detailed software coding activities such as secure coding. Many existing eye tracking technologies focus on top-down and left-to-right reading patterns but reading source code and performing secure coding activities do not follow the standard text reading patterns or even general software coding patterns. Existing visualization methods for eye gaze analysis have limitations for eye tracking stimuli that is user driven and when multiple types of eye tracking stimuli are presented at the same time. Specifically, our findings discovered little information on techniques to visualize the behavior of developers performing various coding activities such as utilizing multiple applications at once and controlling the flow of events by modifying the software code presented in the stimuli content. Many of the visualization techniques that are used in eye tracking research cannot ef-
effectively provide insight into the behavior given the number of AOIs and the high number of transitions between AOIs. Our research focused on methods to analyze the eye gaze of developers performing coding tasks by using existing techniques but also exploring additional visualization algorithms. One such approach is the usage of swimlane diagrams to study the pattern that each developer utilized when working on coding activities. Furthermore, we have explored changes in the pupil diameter of developers as they worked and mitigated security flaws in source code. While our research is focused on examining the behavior of software developers, we believe that other disciplines can also benefit from using eye tracking technologies as well as our visualization techniques.

The fundamental challenge stems from user-controlled eye tracking stimuli. Secure coders may read paragraphs of documentation and guidelines; they may read and write source code; and they may utilize security scanning tools. Often secure coders perform several of these tasks concurrently with an application visually adjacent to other applications and rapidly switch focus between applications. Furthermore, the process and order of finding and mitigating security weaknesses may vary for each participant. Our study provides a realistic software development environment. Secure coders can scroll through documents and source code files, switch among multiple source code files, use a static analysis tool, and conduct web searches. Our design allows each secure coder to control the flow of fixing vulnerabilities. This dynamic design setup is essential for the objective of providing a natural software coding environment that is needed for
evaluating the value of eye gaze visualization techniques for secure coders and their coding behavior.

One of our research goals is to understand the tactics that are utilized to comprehend security issues in source code and approaches that are used to implement mitigation techniques. Another research goal is to understand secure coding behaviors and overcome the limitations of existing eye tracking technologies. That is, we enable:

1. complex tasks and long sessions that are driven by user-controlled tasks.

2. multiple types of stimuli/applications simultaneously.

3. visualization and comparison of multiple participants.

In this dissertation, we also compare the existing techniques and propose our own eye gaze visualization approaches to observe participants’ secure coding behavior. We utilize some existing visualization methods, but we utilize our own dataset to determine the ability to utilize the visualization methods for secure coding. This ensures a uniform comparison of various visualization methods, as the visual images in our manuscript are derived from eye-tracking data gathered from participants engaged in the same source code project. Visualizing and understanding the behavior of secure coders may allow us to discover barriers in our hands-on learning modules and improve the learning content for educating secure coders.

Our main contribution is the understanding of secure coders’ behaviors with multiple types of visualizations of distinct aspects in secure coding and over
a timeline. Our approach considers the linear interval of time and sequence of patterns that secure coders’ transition between stimuli and remain within stimuli. Our visualization techniques, especially those that present different stimuli content concurrently, are essential to understanding secure coding behavior. At the low level, we process eye movements, the speed of movement, the duration of eye fixation, and changes in pupil sizes. At the medium level, we examine participants’ eye gaze at the applications and source code files or function level. At the high level, we present participants’ secure coding patterns and strategies.

Besides the adaptation of existing eye tracking analysis methods for secure coding, we propose swimlane diagrams, state transition diagrams, and pupil size fluctuation diagrams. These allowed us to investigate the transitions within multiple components of source code files and other applications. The swimlane diagrams and state transition diagrams provided the ability to group common software elements together and therefore visually determine and objectively quantify when secure coders have significant changes to different groupings of AOIs over a timeframe. In addition, we captured coders’ attention (the changes in eye gaze) over a timeline to understand and compare their efforts. In this dissertation, we propose a progressive methodology to educate students on secure coding and we evaluate the behavior of our secure coders training with our Manual Code Analysis Learning Module and our Static Code Analysis Tool Learning Module using eye tracking technologies.

The remainder of this dissertation discusses background information, presents each of our learning modules and implementations, the results from our eye track-
ing study and explores future work. In Chapter 2, our research aims to investigate and analyze the behaviors of secure coders, placing special emphasis on their secure coding skills and eye tracking patterns. This involves an extensive literature review, which thoroughly explores existing research pertaining to software development and eye tracking technologies. Chapter 3 includes our Eye Tracking Design and Analysis Framework for software development with a focus on secure coding. This provides a structured methodology that researchers can utilize to plan studies, collect data, and analyze information obtained from eye tracking technology. This helps researchers collect data to understand the visual behavior and gaze patterns of participants while they interact with various stimuli or engage in specific tasks. Our framework is valuable for comprehending software developers’ visual behavior and gaze patterns as they interact with different stimuli or undertake specific coding tasks. Chapter 4 focuses on secure coding stimuli elements and tasks. It is crucial for researchers to decide on the type of visual stimuli or interactive tasks that participants will be presented during a study. Eye tracking stimuli are typically categorized into three main types: static, linear dynamic, and non-linear dynamic stimuli. Chapter 5 covers how our secure coders were presented with visual stimuli including static images and dynamic interactive content. This includes data processing that involves filtering and cleaning the raw data to remove imperfections and ensure data accuracy. This includes identifying specific regions of interest within the stimuli, called AOIs. Chapter 6 provides an overview of eye tracking visualization methods that help to represent the eye gaze data in a visual format. This helps researchers to make sense of the
eye gaze data and gain insight into participants’ visual behavior. In Chapter 7, our eye tracking research involves applying statistical analysis to draw meaningful conclusions from the eye tracking studies. This includes descriptive statistics and non-parametric tests. Chapter 8 includes a description of our current ongoing work. In Chapter 9, we discuss our findings and conclusions. Chapter 10 covers research that could be researched in the future.
Chapter 2. Background and Related Work

This chapter provides background information on these topics and highlights related work on secure coding and eye tracking technologies. Additionally, we provide a detailed and systematic literature survey encompassing various eye gaze visualization techniques, with a specific emphasis on their application in software development. By collecting data using eye tracking technology, we aimed to uncover the patterns and behaviors of participants while engaging with our secure coding learning modules.

2.1 Investigate Secure Coder Behaviors’ to Analyze and Understand Their Secure Coding Skills

Secure coding is an ever-increasing necessity in software development as cyber threats continue posing significant challenges. According to the NVD [32], 4,000 to 8,000 new vulnerabilities have been reported every year since 2005. Over 40% of the reported vulnerabilities are high severity. Exploitation of the vulnerabilities cost $60 Billion every year in the United States [60]. Sensitive data are often exposed in cyber-attacks where data leakage and theft from websites and server breaches are not well prevented. Discovering security and privacy flaws in
existing source code as well as reducing the number of security flaws introduced with new source code is a critical objective for academia and industry.

Software development involves the creation of software applications through many activities including requirements gathering, designing modules, coding, testing, deploying and finally maintaining a software application. The computing instructions contained in a software application are typically coded by a software developer to meet the requirements and design of the software application while supporting long-term maintainability of the software. However, often the source code is not developed and tested to prevent threats from cyber-attacks [9–12].

There are numerous resources of industry best practices available: code review methods [7, 61–63], testing guides [64], secure coding standards [11, 12, 65], vulnerability databases [66, 67], the dictionary of attacks [5], framework for prioritizing weaknesses [68, 69], and software tools and learning modules [13–15, 70–72]. The available materials may not be as effective as they can potentially achieve because there is a lack of understanding of students’ or developers’ secure coding behavior. Students are often overwhelmed to identify and fix security weaknesses [73].

Researchers have examined the effectiveness of manual reviewing of source code to find security weaknesses for web applications and determined it did not produce significant value for finding and mitigating security weaknesses. A study conducted with 30 web developers focused on finding security weaknesses in a PHP application resulted in twenty percent of the developers being unable to find a single vulnerability. In that study, no developer found all of the vulnerabil-
ities [74]. Potential improvements in the ability to find software weaknesses and security flaws in source code is the usage of analysis tools. These types of tools often help software developers find and mitigate security flaws that are missed in a manual searching method. Our research seeks to determine if additional improvements can be beneficial to developers by studying their behaviors and reading patterns when conducting both manual code analysis and when a code analysis tool is used.

Software developers improve their ability and skills to develop quality software applications with more experience as they learn patterns and techniques of writing and debugging software throughout their career. Software developers may include algorithm coders, software testers, and software security developers. A developer may work in a single stage or multiple stages of the Software Development Life Cycle (SDLC) [75]. However, the patterns and techniques that a secure coder utilizes when searching, writing, and testing software code are not easily collected by researchers in an objective and non-intrusive manner. Questionnaires, verbal interview reviews, think-aloud sessions, behavioral retrospectives and other self-reporting methods are helpful, but they cannot provide the objective and detailed information that is needed to study the techniques and approaches developers utilize in software development activities or secure coding activities [44]. Self-reporting techniques cannot directly monitor where participants look on stimuli content and, therefore, can lead to incorrect assumptions about what aspects of a stimuli participants actually examined. Secure coders may rapidly switch between different files and tools, may have multiple types of
stimuli content on a computer display at the same time, and may modify their code. Typical tasks include reading multiple source code files, using web browsers, code editing tools, and code analysis tools. The analysis of their behavior may require examining the stimuli at the programming statement line level, function level, the file level, and the application level. The details cannot simply be portrayed in questionnaires, retrospectives, or think-aloud sessions. Techniques that can objectively collect and measure secure coders’ behavior while remaining unobtrusive can allow a researcher to observe and discover where and when software developers focus on key components in the SDLC such as in the phase of software coding and software testing for coding flaws.

Program comprehension is when software developers are reading source code with the objective of understanding the data structures and algorithms that exist in a software application. This allows developers to construct the data flows and behaviors of an application. Developers are often not the original authors of the code and spend significant time reading and navigating to gain understanding of the code [76]. Often security issues in the source code are not considered when the source code is originally developed [1, 8]. Secure coders may work with existing applications that they have little familiarity with the code and data structures. Consequently, secure coders may benefit from strategies and tools to facilitate their ability to improve their understanding of the code in order to correctly mitigate secure vulnerabilities without introducing additional software flaws [55]. Our goal is to determine if we can improve the learning process by
understanding the behaviors and patterns that secure coders use to approach mitigation of security flaws in software.

Our desire is to engage the participants with hands-on learning where the participants work with an actual application and have step-by-step instructions on how to approach secure coding. A critical goal for providing the ability to gain realistic experience in secure coding problems is to utilize an actual software application that contains security flaws. This is to permit participants the opportunity to gain hands-on learning with an actual software application and not simply a basic function with a small number of executable lines of source code often referred to as a code snippet.

Therefore, there is a need to systematically study and improve the understanding of secure coding behavior. Our goal is to develop effective hands-on educational modules to actively engage software developers to learn practical cybersecurity strategies. Our hands-on learning techniques are designed to guide software developers in a step-by-step process of manually discovering, ranking, and mitigating security weaknesses in application source code. Furthermore, our research is focused on using eye trackers to gain insight and understanding to be able to improve secure coding learning.

2.2 What are Eye Tracking Technologies?

The human eye consists of the pupil, the retina, the fovea and various cones and rods to allow for light signals to be transmitted over the optic nerve and then to be processed by the visual cortex. The movement of our eyes including the
location, direction, momentum, and duration provides insight into our behavior of how we view scenes in the world. Eye tracking technologies record the eye gaze locations and eye movements as well as our eye pupil sizes [77]. Visual stimuli can range from static imagery content to dynamic web pages as well as video content or scenes in nature. Furthermore, stimuli may be interactive and therefore allow each participant to control the order of events in the stimuli. Stimuli are utilized to help guide participants in completing a task.

An eye tracker setup consists of physical eye tracker hardware and software to record participants’ eye gazes. Eye tracking hardware utilizes infrared light that is reflected off the retina of our eyes to identify where an individual is looking. This type of technology can also estimate the size of the pupil [78]. Depending upon the eye tracking hardware specifications, the raw eye gaze data points can be collected for both the left and the right eye [79]. The eye tracker records the eye gaze data points along with metadata that is needed to relate the eye gaze data with stimuli, time, and location. A fixation occurs when the eye is relatively stationary [80]. A fixation can consist of many gaze points that are relatively close together [81, 82], as shown in Figure 2.1. Specifically, a fixation is when our eyes’ fovea, which is the middle region of our retina and the part of the eye that provides the best scene perception, is fixated on an object. A fixation will typically last for milliseconds to seconds [83]. The length of fixations often depends upon the participant as well as the stimuli content utilized [43]. A saccade is the rapid eye movement that occurs when our eyes are moving from one fixation point to another fixation point [80]. Our eyes move to bring objects of curiosity into view so that the
foveal region can increase the amount of data that our brain receives versus the lower quality of data we receive through our parafoveal region or peripheral vision. Saccades are typically shorter than fixations and last for only 30 to 80 milliseconds [84]. Researchers have determined that new visual data are not digested when saccades occur [43]. Another type of eye movement that may be examined is called smooth pursuits. Saccades are typically examined for stationary stimuli content whereas smooth pursuits are examined for dynamic moving stimuli content [85]. A microsaccade is another type of eye movement that occurs when we fixate on a particular object to help focus our eyes to account for the occasional eye drifting [86]. To rephrase, microsaccades are rapid eye movements when our eyes are fixating on an object since our eyes are never completely stationary on an object [87, 88]. Researchers may group microsaccades together as part of a fixation rather than examining the microsaccades individually [43]. Our research does not directly consider microsaccades as a measurement for our reading and software coding tasks as our research is primarily focused on analyzing fixations and saccade transitions. Researchers have previously examined fixations and saccades as a way to study the attention of participants [59, 89].
Eye gaze can be evaluated as coordinates on a computer monitor and can therefore be utilized to determine how participants’ eyes move as they review stimuli on a monitor. One type of eye tracking device can be mounted directly below a computer monitor as shown in the mock-up in Figure 2.2. Eye trackers generally have micro-projectors and optical sensors to create reflection patterns on our eyes and then utilize image processing to find our eyes. The eye tracker is also responsible for processing logic to calculate the exact position of our eyes and therefore our gaze point on the computer monitor. The participant eyes and projection patterns are monitored in real-time. Eye tracking has been utilized in previous studies to investigate participants’ eye gaze to determine where they are looking as they watch various visualizations [90]. This can help to understand what people are visually observing by monitoring their eye gaze patterns.
A critical objective when researching and selecting the appropriate eye tracking hardware and software is the determination of stimuli content that will be presented to participants. This includes determining if only static stimuli content will be utilized or if the eye tracking experiment may also include types of dynamic stimuli media. Furthermore, if participants are capable of changing the stimuli content in real-time, then we classify this type of stimuli content as interactive content or non-linear dynamic content. Linear Dynamic stimuli category includes videos, animations, and moving graphics. However, non-linear dynamic stimuli is interactive content that allows each individual participant to control their viewing experience. Non-linear dynamic stimuli are used when software developers are allowed at their autonomy to change the ordering of stimuli elements and scrolling and switching of applications is allowed by participants. This is beneficial to
create a natural software development environment in order to provide the ability for software developers to control which stimuli elements are viewed as well as the order in which each developer views the stimuli. Analysis of eye tracking eye gaze data encounters distinctive challenges when analyzing non-linear dynamic stimuli. This is because each participant has the freedom to explore the eye tracking stimuli in a non-linear manner. This results in less predictable eye gaze patterns and more individualized behaviors.

Each developer should complete at least one calibration cycle per each session to ensure that their eye gazes and pupil diameters are recorded correctly. This is especially important when the stimuli content is presented on a computer monitor. Depending upon the length of the session, researchers can optionally perform more than one eye calibration cycle. For our secure coding research study, we completed two calibration settings per each recording session as a precaution.

We evaluated various manufacturers (LC Technologies, EyeLink, ISCAN, Tobii Technologies, SensoMotoric Instruments) to find the eye tracking hardware and software that would best record the student behavior [91]. Based on the application (secure coding), the eye tracking timing, accuracy, precision, latencies, and sampling frequencies [92], we selected the Tobii Technologies eye tracker for our secure coding research study. Tobii is currently the leader with the number of publications utilizing their product line [91, 93]. We utilized the Tobii Pro X2-60 hardware as shown in Figure 2.3 and the Tobii Studio software as shown in Figure 2.4. This Tobii Pro product can collect data samples every 16.7 milliseconds and records both left and right eye gaze data points [94]. The Tobii Studio software
provides the ability to utilize multiple types of stimuli content and support our need to have a split-screen arrangement of stimuli content viewable at a time. Additionally, the Tobii software allows participants to control stimuli content by scrolling and switching between files. Our research found that multiple literature publications used the Tobii technology suite of products [91].

Figure 2.3: Tobii Pro X2-60 Eye Tracker Mounted on Desktop Monitor or Laptop.

The Tobii Studio software provides a rich feature set of stimuli options as well as a robust analysis capability [90]. Figure 2.4 presents the “Design and Record” tab that allows eye tracking researchers to design and record participant eye gaze. The tab contains various stimuli elements that the Tobii Studio application supports. This includes instructions, images, websites, movies, questionnaires, and PDF elements. After recording sessions are completed, then eye
tracking researchers can progress into the “Replay” tab to watch recording sessions individually or move onto the “Visualization” tab (as shown in Figure 2.5) to consider multiple eye tracking participants at once. An eye tracking researcher has options to turn on or off specific participant recordings per stimuli element while working in the “Visualization” tab.

Figure 2.4: Tobii Studio Design and Record Tab.
After completing the analysis in the “Replay” tab and the “Visualization” tab, then the eye tracking researcher could move onto the “Areas of Interest” tab for creation of their own defined AOIs as shown in Figure 2.6. The individual stimuli elements that were used in the recording sessions are presented for the eye tracking researchers to manually create their own AOIs. Finally, eye tracking researchers can move onto the “Statistics” tab (shown in Figure 2.7) or the “Data Export” tab to analyze the recording data for selected AOIs and participants.
Figure 2.6: Tobii Studio Areas of Interest Tab.

Figure 2.7: Tobii Studio Statistics Tab.
Researchers have developed many algorithms over the years to determine what is a fixation and saccade from the raw data points. Processing and filtering the eye tracking data to remove noise is a critical first step after collecting the raw eye gazes. Examples include the I-VT Filter [80], ClearView Filter, and Tobii Fixation Filter [58, 80, 92]. The filtering and transforming of the raw eye gaze data points into informative measurements and meaningful insight is an essential phase of using eye tracking technologies. This allows researchers to potentially determine which stimuli content participants found interesting or confusing [59]. Researchers may define one or more AOIs on top of stimuli. A simple stimuli element may only have one AOI defined; however, more complex stimuli may contain several AOIs particularly if dynamic content or scrolling content is contained in the stimuli. Multiple AOIs may be created per each stimuli to facilitate analysis of key features of interest [95]. Often the analysis phase requires examining several stimuli content at a time and therefore several AOIs are often analyzed collectively. As the number of AOIs increases, then this adds additional complexity to visually view the patterns and behavior of participants. This is also known as an increase in the visual clutter [96]. This is also further compounded if the analysis requires investigating by comparing and contrasting several participants.

The simplest stimuli content that is used in eye tracking research is a static page of visualization that contains no scrolling or updating content. Examples of static stimuli include images, photographs, static web pages with no scrolling. Other researchers have commonly utilized stimuli of source code with no scrolling.
allowed [97–102]. However, this often limits the type of content that can be presented to developers as software applications may consist of multiple lines of code across many source code files and directories. Presenting content with the ability to scroll complicates the ability to analyze the eye gaze data in an automated mode. Dynamic media as well as interactive media can allow for a more natural experience for software developers; however, the post-processing required to gain insight from the eye gaze data is more complicated than compared to static stimuli content. Software developers need a natural setting for their software coding sessions in order to investigate their behaviors and techniques in an objective procedure. Our eye tracking experiment and eye gaze analysis focused on secure coding; however, the other phases of the SDLC such as design of software may also benefit from our type of stimuli and analysis techniques. While our focus is on software coding tasks and stimulus content for secure coding, we believe that non-linear interactive stimuli and various different types of stimuli presented simultaneously could be applicable to other fields of eye tracking research.

Often it is possible to examine each participant’s eye gazes separately from other participants. However, researchers may want to compare the eye gazes between multiple participants, especially when comparing and contrasting the behavioral patterns of software developers. Research studies that involve multiple participants may have limitations with some of the visualization methods when attempting to compare and contrast the behaviors of participants. Multiple visualization methods work well for single individuals; however, some of the
visualizations are too complex to provide meaning when several participants are included.

Improving our understanding of the cognitive processes that developers utilize when working software development exercises may allow us to understand how software developers view the design of software instructional material and potentially allow us to improve secure coding learning content [103–105]. Our focus was to investigate the behavior and patterns of software developers learning and working software coding tasks. Our approach utilized eye tracking technologies to unobtrusively collect eye gazes of software developers and then use post-processing visualization analysis techniques to examine the behavior of software developers. The following sections contain an overview of eye tracking technologies and prior published research findings to elaborate on the benefits and current limitations of eye tracking research for secure coding activities.

2.3 Eye Tracking Visualization Literature Review

Our research involved systematically reviewing research publications focused on examining visualization and analysis techniques for eye gazes using eye tracking technology with a focus on software coding and software testing. These were published in various journals and conference papers between the late 1960s and 2021. Articles were reviewed based upon title and abstract for the initial review. A full discussion on our keywords and selection criteria is discussed in the next section. The Eye Movements in Programming (EMIP) and the Eye Tracking Research and Applications Symposium (ETRA) conferences provided
multiple related eye tracking research articles [50, 55, 100, 106–114]. Several literature survey papers on eye tracking have been published in recent years with a focus on software development [44, 56, 81, 82, 99, 115]. A literature review forms an integral aspect of research, as the literature review provides a comprehensive survey of previous publications findings to identify gaps and research questions as well as help researchers understand the state-of-the-art methodologies already published. This helps to avoid duplication with existing studies. Our emphasis is on existing visualization techniques and defining new methods that focus on non-linear dynamic eye tracking stimuli. This means that the stimuli can be modified in real-time by a software developer as well as support for scrolling content in the eye tracking stimuli. Our interest is in providing a software developer a natural coding environment allowing multiple applications to be viewed on a computer monitor at the same time and with scrolling support in each application and providing a participant the ability to modify source code in real-time. To the best of our knowledge, these aspects in combination are not well researched in the current published literature.

Our literature survey of existing eye tracking visualization was utilized to evaluate the existing visualization methods with eye tracking data from software developers performing secure coding activities. Our literature review serves to inform and support our research with relevant background knowledge. Our survey includes a comprehensive and systematic literature survey of existing eye gaze visualization approaches for software coding tasks and objectives in the SDLC. This involved reviewing numerous research publications that contained techniques
for visualization of eye gazes and systematically evaluated the visualization ca-
pabilities for eye tracking stimuli used for software coding tasks such as secure
coding. Our manuscript does not include the exact illustrations that already exist
in the published literature. Instead, our manuscript uses the techniques found in
the published literature to generate new unpublished visualization illustrations
by using our own eye tracking dataset that we collected from software developers.
That allows for a consistent comparison between the different visualization tech-
niques since the visualization images we include in our manuscript are generated
from eye tracking data collected from participants working in the same source
code project. Using the same eye tracking dataset across all the visualization
methods allows for a baseline in the eye tracking dataset to be established to
objectively compare and contrast the visualization techniques. We discuss the
benefits and limitations of the current visualization methods when applied to
stimuli content used in software coding and testing. We provide an overview of
techniques used in the post-processing visualization of eye gaze data of software
developers working coding problems.

Researchers have utilized eye tracking to understand human behavior and
physiological effects [116]. Eye tracking technologies allow researchers to record
and investigate the eye gazes and patterns of participants, and therefore observe
the attention and cognitive activities of participants [56, 89]. Areas of investiga-
tion include determining if a link exists between attention and eye movements and
what features in stimuli attract attention [59, 117–119]. This allows researchers
to overcome some of the limitations of indirect measurement techniques by col-
lecting eye gazes as well as where and how long attention is focused over time [120]. Our goal is to evaluate the applicability of existing methods to visualize the gaze patterns and behaviors collected from eye tracking technologies. Existing eye tracking studies typically present static stimuli that limit the participants’ ability to modify in real-time the presented eye tracking stimuli [38, 97, 106, 108]. Our literature survey and research suggest that the current published eye tracking literature is lacking in using visualization techniques for non-linear dynamic stimuli content that is modifiable by participants. This dissertation considers the complexities of non-linear dynamic stimuli and illustrates several visualization methods when applied to non-linear dynamic stimuli. Non-linear dynamic eye tracking stimuli are not presented in the same sequence for all participants, but from the eye tracking’s perspective of what developers’ do. This allows each developer to control the eye tracking stimuli at their own autonomy. Supporting non-linear dynamic eye tracking is beneficial to study a natural software coding environment, to improve our understanding of the cognitive processes of developers, and thus to improve software coding learning and education [103–105].

Our goal is to evaluate whether existing methods can visualize the gaze patterns and behaviors of software developers performing secure coding tasks. It is especially important to allow a software developer to use multiple applications in parallel and transition frequently between them. Our research initially focuses on static stimuli content analysis before transitioning to eye tracking content that is non-linear dynamic stimuli. We initially present static stimuli content to participants in order to provide simpler eye tracking stimuli before using dynamic
stimuli content. This approach allows for establishing the benefits of non-linear
dynamic content for eye tracking with software developers. This means that our
eye tracking experiment design allows a developer to scroll within web pages,
source code files, documents, and transition among different applications. Thus,
we will be able to investigate, visualize, and understand the natural behavior of
software developers in multiple stages of the SDLC without imposing limits that
would be hampered if using only static eye tracking stimuli. Our work focuses
primarily on secure coding which can be a part of the software coding and software
testing phase of the SDLC. We do not directly consider the numerous stimuli
elements that can often be found in design and requirement gathering phase;
however, our types of stimuli elements, analysis techniques and framework may
facilitate researchers’ ability to analyze eye gaze collected for tasks focused on
design and requirements.

2.3.1 Literature Survey Methodology

We started our literature survey by searching and reading publications in
the ACM Digital Library database, the IEEE Xplore database and the ScienceDi-
rect database. Furthermore, we included findings from a Google Scholar search
and an EBSCO Host search. We only focused on papers written in the English
language. These databases provided several published articles that matched our
search criteria and were reviewed to further determine their applicability to our
research. These databases provided the initial list of articles before we identified
additional articles from the bibliography of the reviewed papers and papers with related terminology.

For the ACM Digital Library and the IEEE Xplore, the search criteria were set to search on the abstract. For the ScienceDirect database, the search criteria were limited to the title, abstract, and author-specific keywords. The Google Scholar search feature allowed us to limit the search based upon the title of the manuscript and not include the abstract. The EBSCO Host database allowed for limiting the search to the abstract only. The end date for the search was set for December 2021. Figure 2.8 contains an overview summary of the literature survey search.

The following search phrase and Boolean operators were utilized to search the above-mentioned databases for the initial set of papers to review:

\[
\text{("Eye Tracking" OR "Eye Movement" OR "Eye Gaze") AND} \\
\text{("Programming" OR "Developer" OR "Visualization" OR} \\
\text{"Code Reading" OR "Software Engineering")}
\]
The authors defined the following inclusion aspects for the papers:

1. visualization methods for static, linear dynamic, and non-linear dynamic eye tracking stimuli;

2. methods that could potentially allow for analyzing eye gazes of stimuli used for software development activities, including the ability to scroll in applications and modify source code file;

3. must include a visualization approach that could be reproducible or online resources are available to reproduce the analysis technique.

The authors defined the following exclusion aspects for the papers:

1. papers focused on providing strictly numerical non-visual techniques for analyzing eye gazes;
2. manuscripts that focused on eye tracking stimuli that were very basic and did not have the complexity of software coding (simple math-based stimuli, general English text reading stimuli).

To determine the suitability of the discovered papers, first the title and abstract were examined by the authors. If the title and abstract did not provide a clear indication that the paper contained some type of visualization technique for eye tracking, then the full text was reviewed for visualization approaches to analyze eye gaze data. When reading the full text, if the paper were found to contain no visualization method that could be explored for the types of stimuli content that we set as the criteria, then the manuscript was removed from further examination.

We found 19 articles via the Association for Computing Machinery (ACM) Digital Library database that had applicable research to match the requirements for inclusion. We found 4 articles on the ScienceDirect database that had applicable manuscripts for our research interest. We found 11 articles in the IEEExplore database that had related research to visualization of a software developer behavior. We found 1 article on Google Scholar that met the inclusion criteria. This provided 35 publications that had potential for visualization analysis of eye gazes collected from software coding tasks. When reviewing the initial set of articles, we found citations to additional papers not previously found that appeared worthy by meeting the inclusion criteria for including in the literature survey. Some of these articles are preprint submissions, but the visualization techniques found in
these articles were worth consideration. We identified an additional 56 articles for a total of 91 articles and the full content in all 91 articles was reviewed.

The type of visualization method that was utilized in these papers was the primary component of most manuscripts that were extracted. Any benefits and limitations of the visualization method were also extracted from the discovered manuscripts. In the following sections, we discuss several of the benefits and limitations but with our own dataset collected from software developers. We also provide references to the published manuscripts that also state the limitations when discussing each visualization technique. This manuscript not only lists the findings from the literature review papers but also utilizes the visualization techniques found with our own dataset collected from software developers to examine the visualization methods using a consistent code base. This method ensures a standardized comparison among the visualization techniques, as the data used to produce these visualizations came from participants all working with the same software code. Several of the visualization techniques are derived from the findings in the literature survey. The published literature does not investigate in detail the non-linear dynamic software coding stimuli that are presented to users with multiple different applications on the monitor at once in a split-screen fashion and allow the participants to modify the code presented in the stimuli content.

The following sections are not a comprehensive list of all visualizations and statistical techniques in the published literature. The limitations are listed with each visualization technique and examples that show the limitations. By taking the approach of showing the limitations of each visualization technique
individually then this allows readers to see the limitations by the visualization


techniques.

2.3.2 Existing Discoveries with Eye Tracking

Researchers use eye tracking to study participant behavior in several do-

doms including airplane pilot attention [52], playing chess [50], and debugging

software [97]. Prior research using eye tracking has examined the eye gaze of

participants driving vehicles [121, 122], playing video games [123], social media

usage [124], weather forecasting [125] and product decision aid recommenders

[126]. Researchers have utilized eye tracking to study if road-side objects such

as advertisements and traffic signs can affect driver’s ability to safety navigate

in a urban environment [121]. Additionally, researchers have used eye tracking
to determine if the mood of the driver has any effect on their ability to safety drive [122]. Video gamers are also considering using eye tracking as a potential

input device to control a game [123]. For basic math problems, researchers have
determined that generally the pupil diameter of participants increases as more
difficult arithmetic problems are presented to the participants. The researchers
determined that when the cognitive load started to decrease then the pupil con-
stricts. Researchers have examined the changes in the pupil diameter based upon
participants’ experience level [127–129]. Prior research publications have focused
on investigating the spatio-temporal eye gaze correlation for gaining insight into
the transition patterns of participants [82]. Researchers have previously discov-
ered relationships between eye movements and attention when reading natural
language text [43]. Researchers have determined that when participants are listening to digits, then the pupil generally dilates. Afterwards, when participants are reporting the number that they heard, then the pupil diameter constricts [130].

Eye tracking technologies allow researchers to collect the eye gazes of participants without disturbing their natural workflow and thereby allow for an investigation into the participants’ cognitive processes and their visual attention [115]. Some eye tracking devices are wearable but can be intrusive [54]. The Tobii X2-60 eye tracker that was selected for our research is designed to be attached to the bottom of a computer monitor and does not require participants to wear any device on their physical body in order to be less intrusive than wearable head-mounted eye tracking devices [54, 79].

Researchers often focus on task relevant problems to investigate differences in their behavior based upon each participant’s background and expertise [131, 132]. They examine eye gaze regressions as a measurement of determining if participants have difficulties reading text [43, 89]. The findings suggest that a reader that has few regressions in natural language text is a good reader. However, software programs are syntactically and semantically different than standard text reading [131].

2.3.3 Eye Tracking Discoveries for Software Development

Our focus is to investigate the behavior and patterns of software developers’ learning and working software coding tasks such as secure coding. Prior studies reveal differences in behaviors for reading source code and reading natural
language text [38, 131]. Our approach utilized eye tracking technologies to unobtrusively collect eye gazes of software developers and then use post-processing visualization analysis techniques to examine the behavior of software developers. Researchers often do not address software coding applications that provide the ability for participants to scroll over eye tracking stimuli content or utilize different software applications at the same time in a split-screen fashion or even modify the eye tracking stimuli real-time as it is viewed [97–100].

Recent work has started exploring the benefits of permitting horizontal and vertical scrolling for software development tasks while an eye tracker records the eye movements of a participant. Researchers have created an initial Eclipse based tool called iTrace which allows for scrollable Java code, HTML files and XML files to be utilized as eye tracking stimuli within the Eclipse IDE. The iTrace tool provides value since it allows scrolling and can account for eye tracking fixations down to the individual source code lines of a software application [133, 134]. The iTrace tool has been utilized by several researchers for conducting eye tracking studies in the software development domain to study the behavior of developers’ [76, 135–137]. The owners of this iTrace tool are continuing to add additional features for more IDEs and software applications as well as support for more hardware eye tracking devices [107]. Since this tool supports a plugin architecture for extending functionality, then this tool is a significant development in this field as it allows for scrolling and switching file behavior within specific IDEs, such as Eclipse [76, 138]. Recent publications indicate that the iTrace tool now has support for the Visual Studio IDE in addition to the original Eclipse IDE.
support [139]. Furthermore, recent work on an iTrace extension for the Atom IDE called iTrace-Atom has been proposed along with Gazel for analyzing collected eye gaze data [140]. Recent work has extended the iTrace tool for support with the Google Chrome web browser [88, 107]. The iTrace tool has significant potential with the plugin architecture as more functionality and software applications are integrated into iTrace. Current published literature indicates that additional work is needed for the iTrace tool to support source code that a participant can modify [88, 107, 140, 141].

A few of our objectives in designing our eye tracking experiment is that we did not want to impose a specific application or web browser on a software developer. Furthermore, we wanted to provide the developer with the ability to modify the source code real time with their programming solution to the security coding flaw. Additionally, not all software development activities occur in an IDE or web browser such as a software developer utilizing a software scanning tool to automatically search for software bugs. Our approach using Tobii Studio allowed for a software developer to modify the actual source code that they examine in order to test their understanding of finding and mitigating software bugs in existing source code. Furthermore, our approach allowed a developer to have multiple different application windows open at once, such as a text editor and a web browser, in a side-by-side comparison view. This adds additional complexity in the post-processing phase to create AOIs as the dynamic stimuli are updated. Currently, the literature is still lacking in examining the eye gazes of developers when they utilize multiple types of modifiable stimuli content on the screen at the
same time, particularly with scrollable source code and when arranged in a split-
screen layout. Furthermore, the techniques are lacking in allowing researchers to
discover when developers frequently transition between multiple computer appli-
cations such as source code files, integrated development environments, software
scanning tools and web browsers. Allowing an eye tracking participant to have the
ability to modify the actual source code real-time is a critical aspect to providing
a natural software coding environment; however, this is often not addressed in
the literature. The ability to visualize a developers’ behavior when a developer is
controlling the flow of eye tracking stimuli is therefore still lacking.

Research using eye tracking have discovered that pupillary responses can
be related to the difficulty of tasks as well as the overall processing time [142].
The researchers discovered that cognitive load and pupil diameter sizes are cor-
related for interactive assignments [42]. Researchers have determined that as the
task difficulty increases, then our pupils dilate in response to the increase in cog-
nitive load [127, 143]. Prior research has revealed that after a participant answers
a question, then that participant has a significant shrinking in the diameter of
their pupils [144]. Researchers has examined if the pupil data could be utilized
for examining the cognitive workload of participants [53, 54]. There is still a lack
of content in investigating the changes in pupil diameter size when utilizing dy-
namic non-linear and participant-controlled content, particularly for the software
engineering field such as secure coding.

Using eye trackers, it may be possible to determine if a software developer
utilizes a top-down code review approach or a trial-and-error programming ap-
proach or a procedural control flow approach. This can be useful to determine if specific areas of code are looked at more than other areas [145–147]. Prior research using eye tracking to examine when participants read source code has discovered that initial portions of source code tend to have long dwell durations with the code segments viewed later having shorter dwell durations [43, 54]. Another study found that the verification of ideas and thoughts on how source code transforms input data is often determined by software developers jumping among functions in the code [98]. Previous research has discovered that participants scan paths and attention can help to uncover the cognitive processes that participants use when reading source code [43, 54].

Prior research has examined eye tracking fixation data from participants at a line-by-line level when reviewing source code. Their research revealed that reading source code is not exactly similar to reading standard paragraph text such as found in reading books or news articles. Instead, readers of source code often jump from line to line or compare and contrast several lines of source code. The authors verify using quantitative analysis that when participants spend more time on scanning source code, then the participant is likely to take less time finding defects in the source code. Their analysis was performed on programs written in the C programming language that reached a maximum length of 23 lines of code. Additionally, the authors identified two behavior patterns which they identified as Retrace Declaration Pattern and Retrace Reference Pattern. Using this analysis approach, the researchers were able to determine that the
participants often referenced or retraced declared C variables when the program source code utilized an existing declared variable [108].

In a research study conducted recently also applying this visualization approach for line-by-line analysis of developers reading source code discovered specific areas of a function that developers spent more time examining. The work revealed that when a developer is within a method, then the method invocations and variable declarations had the more significant time spent reading while the method signatures received less reading time by the developers. Their research revealed that when a developer is going between methods then developers only read a few lines per method and that the developers are often assessing the flow of the data represented by the variables [76]. This finding was also confirmed by other researchers in that developers transition more often between AOIs that are adjacent in the same source code file, particularly with previous AOIs [98]. Researchers found that developers explore the code by scanning over the source code but the developers that examined the code more meticulously had better likelihood of solving a task associated with changing of the code base [76].

Researchers applied eye tracking technologies to record the fixations of professional software developers and then replayed those to novice developers. Their research revealed that for the software bugs that novices viewed the behavior of professional developers, then the novices found those software bugs faster than for bugs that they just attempted to find without viewing the eye gaze of experts [97]. Research conducted also discovered that novices take more time to find flaws in source code and potentially have a greater likelihood of introducing
new bugs in a software application. The researchers conclude that the ability to uncover software bugs was related to the ability to comprehend the software code [148, 149]. Researchers that examined the dwell time or fixation time of novices versus non-novices discovered that novices had lower percentage of average total dwell time when compared to the non-novices [111]. A study conducted in recent years indicates that reading source code sectionally might be an indication of a developer who is a beginner [113]. Studying differences of novice and experts has provided insights into expert behavior who focus more on relevant parts of source code than novices [150]. Researchers found that experts read source code in a less linear way when compared to novices. They also found novices read source code less linearly than natural language text [102, 131]. Researchers have discovered that when developers scan the entire source code of an application, then those developers could more efficiently find the defects in the software code. This is potentially due to developers identifying potential defects that may exist while the developer is building a mental model of the overall source code application [108].

Researchers have recently completed detailed literature survey reports that contain comprehensive descriptions of using eye tracking technologies for general software development activities [44, 56, 81, 82, 89, 115, 151]. Conferences such as The Eye Movements in Programming Conference and the Eye Tracking Research and Application Symposium now exist that directly focus on eye tracking related research with software programming [50, 55, 100, 106–114]. Eye tracking studies have examined and determined that differences exist when developers read source
code than when they read natural language text [131]. Reading and writing source code for general algorithm comprehension is not directly the same as working to discover and resolve security vulnerabilities. However, there are similarities with how developers approach software coding and secure coding [97].

Our focus was researching visualization methods for multiple types of stimuli content including static eye tracking stimuli as well as non-linear dynamic stimuli. This allowed a software developer not only to scroll in long source code files but also to modify the source code that is presented as stimuli elements. Furthermore, we do not want to limit the design to only a specific software development integrated development environment. This provided a more natural environment for a software developer than limiting the stimuli to only lines of code that fit on a single monitor [115]. The current literature is lacking with visualization and analysis methods for activities involved in secure coding particularly for non-linear dynamic stimuli. Many existing eye tracking publications limit participants’ ability to scroll in the eye tracking stimuli which also includes limiting the ability of a participant to actually modify the eye tracking stimuli that are presented [38, 97, 106, 108]. Our primary focus was directly observing secure coding and learning activities, and then determined the ability to analyze secure coder patterns and behavior.

Writing source code and performing secure coding tasks should not mandate a specific step-by-step order of stimuli element content that eye tracking participants must follow, if the desire is to capture the unique self-motivated behavior that each secure coder uses to find and mitigate security flaws. Thus,
a need exists to improve the methodologies used to investigate and analyze eye tracking data for software development and secure coding. A potential pitfall with some methods is that they can allow for the unique individual differences in secure coders to be obscured from view. For example, this can occur if the data are simplified as an arithmetic mean for several participants over an extended timeframe [55]. Therefore, the visualization approach may depend upon the type of stimuli content and the task given to coders.

Software development includes many aspects for designing, developing, and maintaining software applications. The areas of software requirements gathering and documentation [152], software traceability [136, 138, 153], UML [154–157], user interface designs, database interfaces, and software unit testing is beyond the scope of this manuscript. While our work focused on the secure coding aspects of software development, it will be interesting to explore other aspects of the software development domain in the future. Our manuscript includes references to the manuscripts of multiple other researchers’ discoveries in these other software development areas for interested readers.

2.3.4 Heat Map

A heat map allows researchers to determine the area of a stimulus that participants had the longest dwell times. They utilize color gradients to highlight areas on stimuli that attracted the most visual attention either as an individual or a group of participants. The heatmap color palette allows the areas with longer dwell times to be distinguished from the areas that had shorter dwell times
Prior research has determined that longer fixation durations indicate more involved cognitive processing \([43, 59]\). Heat maps work well for static imagery content and for datasets with minimum stimuli screen content presented at one time. However, the stimuli content used by software developers often includes content that requires scrolling of multiple web pages and multiple source code files as well as participants switching back and forth between multiple files. Also, software engineering tasks include stimuli content in which not all developers visit every aspect of the overall content provided and each developer may visit the content in different sequences from other developers. This complicates the data analysis and limits the usage of heat maps. Nevertheless, dynamic participant-controlled stimuli content is required to allow software developers a natural flow of their natural fashion of approaching software code writing. Researchers have determined that another limitation is that heatmaps can allow single data points to appear as a lighter color that can be misinterpreted if viewers are not familiar with the color map schema used on the heat map \([158]\). Overall, a heat map can provide an overview for analysis of static imagery eye tracking stimuli content; however, a heat map has limitations for stimuli content containing scrollable content or participant interactive content.

A heatmap can be generated as a static imagery snippet as shown in Figure 2.9. It is also possible to generate a video recording of a heatmap that shows the update over time. In Figure 2.9, a single developer eye gaze is shown for a static stimuli content for the first fifty-seven milliseconds of the recording session. In Figure 2.10, the same software developer eye gaze is displayed but for their entire
recording session. In Figure 2.11 and Figure 2.12, we show the heatmap for multiple developers for different static stimuli content containing software code. The darker red color indicates longer fixation times for those regions on the stimuli while lighter green color indicates shorter fixations times by the participants.

**Figure 2.9:** Heat Map for Static Stimuli for Single Software Developer for 57 milliseconds.
Figure 2.10: Heat Map for Static Stimuli for Single Software Developer.
Figure 2.11: Heat Map for Static Stimuli for Multiple Software Developers Example 1.
2.3.5 Scan Path

A scan path or gaze plot provides researchers with the ability to examine a participant’s eye movements as their eyes move from one fixation point to the next fixation point [159]. This can provide a time-based aspect that heat maps alone cannot provide to researchers. A scan path provides a visual representation of the fixations, gaze location and frequency changes along a linear timeframe.
This may be beneficial for determining a search pattern and behavior of participants. A circle is typically utilized to represent a fixation and a connecting line to represent saccades between the fixations [81]. Scan paths work well for single participant analysis as the build-up of many connected fixations points for multiple participants can complicate the readability of scan paths [158]. Even with a single participant, the scan path can build up over a long recording session and complicate the data analysis. Some versions of scan paths attempt to solve this problem by providing a sliding window timeframe to remove previous fixations after a specified timeframe has occurred. This works well when watching a video recording overlayed with scan paths of a participant’s eye gaze. However, this approach has limitations if attempting to generate a static image of a scan path. Researchers have also investigated scan path visualization techniques as a way to establish the patterns that occur for software coding by identifying common scanpath events. The scan path visualization technique works well for single participants; however, when overlaying several participants then the visualization of scan paths provides reduced value. A potential improvement to the scan path for multiple participants is the ability to set a sliding window timeframe to remove fixation points after a specific time period for static stimuli. Often these scanpath events for software coding are identified as reading, look ahead, look back, return, forward jump, verify events. Our research did not focus on an in-depth analysis of scanpath events as other researchers have examined scanpath events in detail [84, 98, 102, 108].
A scan path is generated as a static imagery snippet as shown in Figure 2.13 and Figure 2.14 and Figure 2.15 and Figure 2.16. The scan path can also be generated in a video format for watching during the analysis phase. In Figure 2.13, a single developer eye gaze is shown for a static stimuli content for the first fifty-seven milliseconds of the recording session. In Figure 2.14, the same developer eye gaze is displayed but for the entire recording session for that specific software developer. In Figure 2.15, we show the scan path for two developers for a static stimuli content for the first seventy-five milliseconds. In Figure 2.16, we demonstrate a non-cut version of the scan path for twenty-nine software developers for their full-length video recordings.
Figure 2.13: Scan Path for Static Stimuli for Single Software Developer for 57 milliseconds.
Figure 2.14: Scan Path for Static Stimuli for Single Software Developer for full recording.
**Figure 2.15:** Scan Path for Static Stimuli for Multiple Software Developers for 57 milliseconds.
Figure 2.16: Scan Path for Static Stimuli for 29 Software Developers for full recording.

2.3.6 Clustering

A cluster map allows for a spatial threshold to be supplied in order to separate fixations into separate regions for visual analysis. Different clusters are created when fixations are spatially separated based upon an analyst supplied limitation value. Clustering has limitations that are similar to heat maps and scan paths when used for dynamic video recorded content as well as scrollable web pages or stimuli with user interactive content. Clustering does work well for
static imagery content to determine which areas of the stimuli content are viewed by several participants. This is possible since clustering allows for showing the percentage of participants that fixated in a cluster [58]. As showcased in Figure 2.17, the cluster identified as Cluster 6 had 97% of developers fixated in that cluster compared to 84% fixated in Cluster 5 and 65% of developers fixated in Cluster 4. Changing the spatial threshold of clustering can create additional or fewer clusters. When decreasing the spatial threshold, then this restricts the gap that fixations can be before a new cluster is generated. This is shown in Figure 2.18 for the same developers but with a lower spatial threshold. The disadvantage of decreasing the spatial distance threshold can increase the visual clutter in a cluster image [58]. The Tobii Studio software clustering visualization tool also allows researchers to add a mouse icon in the locations where participants clicked the mouse button when viewing the stimuli as shown in Figure 2.19.
Figure 2.17: Clustering with High Spatial Distance Threshold for All Software Developers viewing Static Stimuli Content.
Figure 2.18: Clustering with Low Spatial Distance Threshold for All Software Developers viewing Static Stimuli Content.
2.3.7 Fixation Table

A fixation matrix is a table of fixations grouped based upon stimuli content or if needed a table of AOIs if more concentrated aspects of the stimuli content should be examined. This table is often presented as a count or as a percentage for the total number of fixations per participant. Another common name for transitions over stimuli or AOIs is called the gaze shift [59]. For example, a
fixation table can be generated to examine fixations when participants utilize
the e-learning software known as Canvas to determine where students are looking
when they are taking an online exam [83]. An example of a fixation table is shown
in Figure 2.20 and another example with more AOIs is shown in Figure 2.21. It
can be determined from Figure 2.20 that the developer viewed two AOIs with a
fixation count greater than forty. From Figure 2.21, one can visually understand
that with an increase in the number of AOIs, then there is also an increase in the
complexity of the fixation table. A limitation of the fixation table is that it does
not indicate the chronological order or the transitions when a participant moves
from one AOI to another AOI or from one stimuli content to another stimuli
content. Furthermore, if comparing participants’ patterns and behaviors is of
interest, then a new fixation table is needed for every participant.
**Figure 2.20:** Fixation Table of a Single Developer Shown as a Count and a Percentage when Reading Source Code displayed using Static Stimuli Content.

<table>
<thead>
<tr>
<th>AGI Name</th>
<th>Number of Fixation (count)</th>
<th>Fixation Duration (ms)</th>
<th>Total Duration (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AG1[ICME 311 Option A Start Headers and Session Variables]Hit</td>
<td>35</td>
<td>5350.0</td>
<td>7017.0</td>
</tr>
<tr>
<td>AG1[ICME 311 Option A SQL Database Connection Variables]Hit</td>
<td>21</td>
<td>3810.0</td>
<td>7017.0</td>
</tr>
<tr>
<td>AG1[ICME 311 Option A Included Function Files]Hit</td>
<td>16</td>
<td>2634.0</td>
<td>4916.0</td>
</tr>
<tr>
<td>AG1[ICME 311 Option B Execute SQL Query]Hit</td>
<td>23</td>
<td>6000.0</td>
<td>17417.0</td>
</tr>
<tr>
<td>AG1[ICME 311 Option B Create SQL Variable]Hit</td>
<td>49</td>
<td>12116.0</td>
<td>24733.0</td>
</tr>
<tr>
<td>AG1[ICME 311 Option B Escape Input Variable]Hit</td>
<td>33</td>
<td>2885.0</td>
<td>6219.0</td>
</tr>
<tr>
<td>AG1[ICME 311 Option B Function Definitions]Hit</td>
<td>1</td>
<td>100.0</td>
<td>110.0</td>
</tr>
<tr>
<td>AG1[ICME 311 Option B Return SQL Results]Hit</td>
<td>14</td>
<td>3550.0</td>
<td>9350.0</td>
</tr>
<tr>
<td>AG1[ICME 311 Option C Escape Input Variable]Hit</td>
<td>29</td>
<td>5385.0</td>
<td>13385.0</td>
</tr>
<tr>
<td>AG1[ICME 311 Option C Function Definitions]Hit</td>
<td>14</td>
<td>3433.0</td>
<td>7837.0</td>
</tr>
<tr>
<td>AG1[ICME 311 Option C Password Encryption]Hit</td>
<td>21</td>
<td>4184.0</td>
<td>10215.0</td>
</tr>
<tr>
<td>AG1[ICME 311 Option C Create SQL Variable]Hit</td>
<td>47</td>
<td>9399.0</td>
<td>24998.0</td>
</tr>
<tr>
<td>AG1[ICME 311 Option C Execute SQL Query]Hit</td>
<td>16</td>
<td>3291.0</td>
<td>6466.0</td>
</tr>
<tr>
<td>AG1[ICME 311 Option C Return SQL Results]Hit</td>
<td>4</td>
<td>534.0</td>
<td>1034.0</td>
</tr>
</tbody>
</table>

**Student MCA_14_F_19 for Recording Rec 21**

<table>
<thead>
<tr>
<th>AGI Name</th>
<th>Number of Fixation (count)</th>
<th>Fixation Duration (ms)</th>
<th>Total Duration (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AG1[ICME 311 Option A Start Headers and Session Variables]Hit</td>
<td>05.300</td>
<td>08.554</td>
<td>04.725</td>
</tr>
<tr>
<td>AG1[ICME 311 Option A SQL Database Connection Variables]Hit</td>
<td>07.420</td>
<td>06.104</td>
<td>04.725</td>
</tr>
<tr>
<td>AG1[ICME 311 Option A Included Function Files]Hit</td>
<td>05.653</td>
<td>04.241</td>
<td>03.310</td>
</tr>
<tr>
<td>AG1[ICME 311 Option B Execute SQL Query]Hit</td>
<td>08.127</td>
<td>09.593</td>
<td>11.729</td>
</tr>
<tr>
<td>AG1[ICME 311 Option B Create SQL Variable]Hit</td>
<td>37.314</td>
<td>19.376</td>
<td>16.657</td>
</tr>
<tr>
<td>AG1[ICME 311 Option B Escape Input Variable]Hit</td>
<td>04.593</td>
<td>04.612</td>
<td>04.188</td>
</tr>
<tr>
<td>AG1[ICME 311 Option B Function Definitions]Hit</td>
<td>06.353</td>
<td>06.159</td>
<td>06.698</td>
</tr>
<tr>
<td>AG1[ICME 311 Option B Return SQL Results]Hit</td>
<td>04.946</td>
<td>05.676</td>
<td>06.296</td>
</tr>
<tr>
<td>AG1[ICME 311 Option C Escape Input Variable]Hit</td>
<td>10.247</td>
<td>08.610</td>
<td>09.014</td>
</tr>
<tr>
<td>AG1[ICME 311 Option C Function Definitions]Hit</td>
<td>04.946</td>
<td>05.489</td>
<td>04.946</td>
</tr>
<tr>
<td>AG1[ICME 311 Option C Password Encryption]Hit</td>
<td>07.420</td>
<td>06.690</td>
<td>13.210</td>
</tr>
<tr>
<td>AG1[ICME 311 Option C Execute SQL Query]Hit</td>
<td>05.653</td>
<td>05.118</td>
<td>04.354</td>
</tr>
<tr>
<td>AG1[ICME 311 Option C Return SQL Results]Hit</td>
<td>01.413</td>
<td>00.853</td>
<td>01.063</td>
</tr>
<tr>
<td>TOTAL</td>
<td>99.992</td>
<td>99.993</td>
<td>99.993</td>
</tr>
</tbody>
</table>
2.3.8 Transition Matrix

There is often a need to analyze and visualize the transitions that occur between various stimuli content or AOIs and even within the same stimuli content. The previously mentioned visualization methods can help to provide a
A high-level overview of participants’ viewing behavior of stimuli; however, additional visualization techniques are needed to graphically represent the behaviors and patterns in eye gaze transitions. Transitions show the switches in attention between stimuli or AOIs and this can help researchers understand the visual strategies over a period of time that participants used when inspecting the stimuli presented. A limitation of a transition matrix is that it does not show the order in which participants transition between the various stimuli or AOIs. Basically, the chronological changes in the transitions cannot be viewed in a transition matrix. If multiple participants’ results are aggregated together into a single transition matrix it is difficult to determine which participant contributed to which transitions [112, 160]. Recently, the transition matrix visualization approach has been examined when researchers are studying the behavior of individuals reading UML Class Diagrams [54].

An example of a transition matrix is shown in Figure 2.22 for one of our eye tracking participants. This task in our eye tracking experiment consisted of each participant reviewing a source code defect description concerning encrypting plaintext passwords before storing passwords in a database. Then, each participant was given three programming functions but only one of the three programming functions contained the acceptable programming code that would encrypt a plaintext password before it is stored into a database. Each software developer was asked to indicate the correct programming function that would mitigate an encryption flaw to prevent the source code from containing a security weakness. “Programming Option C” contained the encryption for the user password before
storing it in a database. The source code shown in “Programming Option A” and “Programming Option B” did not contain any code related to storing a plain-text password in a database. This experiment was designed to examine if each participant could discover the source code that could potentially have a specific software flaw but with proper programming checks in the source code to prevent the security flaw from occurring. The objective was to ascertain if participants correctly understood how to locate flaws associated with password saving and the needed encryption mitigation techniques in existing code.

In our transition matrix, each row represents the current AOI that a software developer viewed. Therefore, the column in our transition matrix represents the next AOI visited after leaving the current AOI. Furthermore, we provide a legend at the top of our transition matrix picture to represent the abbreviated AOIs for easier readability. The transition matrix is valuable to researchers in order to ascertain the eye gaze adjustments from one AOI to another AOI. The prior mentioned visualization techniques do not provide the detailed overview of transitions between AOIs that a transition matrix illustrates. This type of matrix provides a high-level synopsis of the patterns and behavior of the eye gaze transitions utilized by each participant. This type of visualization demonstrates what areas of source code developers transition between when reading and writing source code.
Figure 2.22: Transition Matrix Shown as a Transition Count and a Transition Percentage when Reading Source Code presented as Static Stimuli Content.

In Figure 2.22, we provide an example of a transition matrix for a single software coding exercise that contained three software program functions. Each programming function is presented as a single stimulus that is displayed in Figure
Each programming solution had multiple AOIs that we created based upon the programming logic in each programming function. Our transition matrix includes a legend at the top of Figure 2.22 that is designed to create an “AOI ID” mapping to the “AOI Name” that was used as the overlay on top of the programming function. This mapping from a shorter AOI designation, “AOI ID”, to the fully labelled AOI designation, “AOI Name”, was designed to allow for an abbreviated naming convention to construct a cubical-shaped matrix for transition count and transition percentage as shown in Figure 2.22. Adding our legend with a shorter AOI name mapping was designed to facilitate easier reading and tracking of the eye gaze shifts shown in the transition matrix. The first AOI mapping shown in Figure 2.23, is identified as AOI ID “A” with the AOI Name “Option A Start Headers and Session Variables”. The example of this AOI ID “A” with the programming code is shown in Figure 2.23 with the first AOI box in Figure 2.23. The rest of the AOIs shown in the legend of Figure 2.22 are shown sequentially in Figure 2.23 and Figure 2.24 and Figure 2.25. The transition count shown in the middle of Figure 2.22 and the transition percentage shown at the bottom of Figure 2.22 relies upon the abbreviated naming conventions for the AOIs to facilitate displaying a cubical-shaped matrix. The AOI abbreviated name shown in the rows of Figure 2.22 indicate the current AOI while the AOI abbreviated name shown in the columns of Figure 2.22 indicate the AOI that was transitioned towards after leaving the AOI shown in each row.

One can determine from Figure 2.22 that this specific developer had fourteen transitions from Area of Interest (AOI) identified as L (Option C Execute
SQL Query) to the AOI identified as M (Option C Return SQL Results). The correct solution for this software flaw was in programming Option C. However, the specific AOI with the software flaw resolved was located in the AOI identified as N (Option C Password Encryption). The developer had the most transitions to the AOI identified as N (Option C Password Encryption) from the AOI identified as J (Option C Escape Input Variable). Specifically, this developer entered the AOI N (Option C Password Encryption) a total of eight times from the AOI identified as J (Option C Escape Input Variable). The specific programming logic for those AOIs is shown in Figure 2.24 and Figure 2.25.
Figure 2.23: Programming Solution Option A when Reading Source Code presented as Static Stimuli Content.
**Figure 2.24:** Programming Solution Option B when Reading Source Code presented as Static Stimuli Content.
2.3.9 Radial Transition Graph

One group of researchers created the ability to study the patterns and transitions of multiple participants using radial transition graphs. Their graphing capability shows the transitions between defined AOIs as well as the amount of time participants spent in each AOI. This allows research analysts to visualize the dwell times or sequence of fixations per each AOI per participant at the same time as the transitions from an AOI to a different AOI. The transition from one
AOI to another AOI is symbolized by arc lines in the graph. Each AOI has separate lines for the incoming and outgoing transitions. The researchers developed the capability to view transitions per participant as well as the ability to compare transition and timing differences in multiple participants. Their developed capability is shared via an online website http://www.rtgct.fbeck.com/ that allows researchers to format their datasets in a predefined format and upload to their hosted website [160].

Researchers have studied the reading behavior of three software programs written using the C++ programming language. Their analysis method included examining fixations line-by-line that occur when developers examine source code. Their findings suggest that it is possible to utilize eye tracking technologies to determine where the transitions occur when developers are reading source code. These researchers utilized the radial transition graphs to assist in their analysis of eye gazes over software programming code. They concluded that transitions between lines of source code are close to the present line and associated lines are often viewed nearby in time. The C++ programs that the researchers analyzed had lengths of 12 to 29 lines of code [106]. Furthermore, by using the radial transition graphs, researchers have discovered patterns of linear code reading as well as patterns representing the execution of source code [111].

In Figure 2.26 and Figure 2.27, we demonstrate the radial transition graph with our eye tracking dataset for our software coding exercises. We used the online website http://www.rtgct.fbeck.com/ with our dataset in the expected format and generated Figure 2.26 and Figure 2.27 [160]. Figure 2.26 shows a basic exam-
ple of the visualization technique for three participants with seven AOIs. Figure 2.27 demonstrates the same visualization technique but for sixteen participants and sixteen AOIs. As Figure 2.27 shows, as the number of AOIs continue increasing and the number of transitions between AOIs continue increasing, then this increases the complexity of this visualization method. However, the inventors of this visualization method provide additional analysis techniques to continue performing analysis. One example is the ability to hover over an AOI and compare two participants. As can be seen in Figure 2.27, it is possible to determine that participant labeled $MCA_{19}.F.19$ had 12.4 seconds of eye gaze duration in the AOI labeled $AOI \ [CWE \ 434 \ Option \ A \ IF \ Incorrect \ File \ Extension] \ Hit$. However, participant labeled $MCA_{12}.F.19$ only had 3.7 seconds of eye gaze duration for the same AOI. For this specific programming task, this $AOI \ [CWE \ 434 \ Option \ A \ IF \ Incorrect \ File \ Extension] \ Hit$ AOI had the software coding flaw and participant $MCA_{19}.F.19$ correctly identified the software flaw while participant $MCA_{12}.F.19$ did not identify the software flaw.
Figure 2.26: Radial Transition Graph Example for three Software Developers when Reading Documentation presented as Static Stimuli Content.

Figure 2.27: Radial Transition Graph Grid View for Multiple Software Developers when Reading and Writing Source Code as well as reviewing reports from Software Scanning Software viewed using Static Stimuli Content.
Benefits of the radial transition graph include the ability to compare visited AOIs as well as the transitions between the AOI that are visited. The radial transition graph also provides the temporal aspect that some visualization methods cannot provide to the level of radial transition graphs. However, a limitation that we found with the radial transition graph is when the number of AOI starts to increase, then the graph starts to become more difficult to analyze. Also, the number of unique easily identifiable colors is reduced. Another issue occurs as the number of transitions per graph increases, then the diagram is more difficult to understand. This same conclusion was also found by researchers who studied the application of radial transition graphs with three eye tracking experts evaluating the method [160]. The radial transition graphs have limitations given the number of transitions and number of AOIs that are needed for software coding in a natural environment for large scale software code bases. Some studies limit the size of the source code that is examined using radial transition graphs [106, 111].

2.3.10 Scarf Plot

Scarf plots allow for visualization of the AOIs along with the associated transitions displayed over a linear timeline. Researchers have investigated the usage of viewing multiple participant scan paths next to each other and color-coding specific AOIs that are of particular interest. This allows for showing the transitions between specific AOIs for multiple participants. The x-axis can be utilized to list the various AOIs and the y-axis used to show the transitions over time for each participant [106, 161]. This method provides the ability to examine
the series of visited AOIs in the chronological order in which they are visited by each participant. This could be beneficial to allow for comparison between several participants as each participant has their own linear timeline. However, a limitation with scarf plots is the difficulty to determine specific AOIs when the number of AOIs continue to increase [81, 160]. As the number of participants increases, then that also increases the complexity to compare and find patterns in scarf plots [162]. When the number of transitions also increases, then the x-axis can also become difficult to analyze. An example, the scarf plot has limitations as the number of transitions between AOI increase for each participant. An interactive version of the scarf plots does provide some ability to overcome this limitation.

Scarf plots can be generated from a time-based measurement as well as a transition-based system of measurement. Scarf plots that are time-based illustrate the AOIs that are fixated on longer than other AOIs and also when in the sequence of transitions, a specific AOI was fixated on for a longer period of time. Scarf plots that are transition-based are useful when comparing participants for patterns and common behaviors. This is because each box in the plot is equal length allowing easier comparison of the transitions regardless of how long participants remained in any of the AOIs.

As exhibited in Figure 2.28 and Figure 2.30, is time-based measurement for two different software developers’ and Figure 2.29 and Figure 2.31 is transition-based for two different software developers. The developer for Figure 2.28 and Figure 2.29 did not have many transitions and did not visit many different AOIs.
However, the software developer for Figure 2.30 and Figure 2.31 visited many AOIs and changed AOIs frequently. As shown in Figure 2.28 and Figure 2.29, scarf plots work well when the number of AOIs in the analysis are small. If participants have a considerable number of transitions or visit multiple AOIs, then scarf plots have limitations for visually examining the behavior and strategies of participants as shown in Figure 2.30 and Figure 2.31.

2.3.11 Summary of Visualization Techniques

Our survey concentrated on the examination of published techniques for visualizing eye gaze data of various stimulus types including static, linear dynamic, and non-linear interactive stimuli. A summary of the visualization techniques is presented in Figure 2.32 with references to existing published literature containing additional visual examples of several graphical technique. Our focus was determining current visualization techniques of eye gaze data for developers performing activities related to software coding.
Figure 2.28: Scarf Plots for a Software Developer presented with a Time-based Measurement when viewed using Non-Linear Dynamic Stimuli Content Example 1.
Figure 2.29: Scarf Plots for a Software Developer presented with a Transition-based Count when viewed using Non-Linear Dynamic Stimuli Content Example 1.
Figure 2.30: Scarf Plots for a Software Developer presented with a Time-based Measurement when viewed using Non-Linear Dynamic Stimuli Content Example 2.
Figure 2.31: Scarf Plots for a Software Developer presented with a Transition-based Count when viewed using Non-Linear Dynamic Stimuli Content Example 2.
2.3.12 Limitations of Current Visualization Techniques

Our literature survey found minimal focus on examining and visualizing the eye gazes of developers while they work on the types of tasks required to
learn how to develop software code, which includes scrolling, using multiple source code files, and working with diverse types of software applications. The current literature lacks visualization and analysis methods specifically tailored to secure coding activities. One of our focuses is directly observing secure coders as they perform secure coding learning and practice activities to determine the potential to analyze secure coder patterns and behavior.

Several of the published papers focused on source code and eye tracking have particularly used source code snippets of small coding functions with no scrolling [97–102]. The published literature is lacking in examining visualization methods for non-linear dynamic stimuli that is caused from each developer changing the stimuli as well as the usage of multiple source code files, web browser, and coding analysis tools. During our literature review, we determined that several existing visualizations had limitations for analysis of eye gazes within the software engineering field. Limitations particular existed for when the number of AOIs increased beyond five to ten AOIs. Limitations also exist when attempting to examine the transitions between AOIs along with limitations with the dynamic nature of scrolling in source code files and web browsers.

Writing source code and performing secure coding tasks is an example where researchers should not mandate a specific step-by-step order of stimuli element content that eye tracking participants must follow if the desire is to capture the unique self-motivated behavior that each secure coder uses to find and mitigate security flaws. Therefore, a need exists to improve the methodologies used to investigate and analyze eye tracking data for software development and
secure coding. A potential pitfall with some methods is that they can allow for the unique individual differences in secure coders to be obscured from view. An example of this can be seen when the data are simplified as an arithmetic mean for multiple participants over a prolonged period [55]. Therefore, the visualization approach may depend upon the type of stimuli content and the task given to coders.

Eye tracking technologies offer the potential to collect information on cognitive load while participants’ work intensive tasks without requiring participants to provide real-time verbal feedback. In our search of existing data visualization techniques, we found several graphical techniques that help visually demonstrate participants’ fixations, their transitions and viewing patterns as well the amount of time that developers focused on stimuli content. The heatmap, scan path and clustering techniques are commonly utilized algorithms that eye tracking researchers initially use to visualize fixations. Researchers have discovered limitations with these approaches and created techniques such as the fixation table to provide an overall summary in numerical format. However, this method does not provide any indication of transitions between the various stimuli content or various AOIs. A transition matrix can provide numerical results of the transitions in the stimuli. The radial transition graphs and scarf plots can provide information on the transitions and durations and often allow for comparison between various developers. Several of these methods present problems for comparing and contrasting the patterns and techniques of several developers. As the number of AOI increases for each participant then the complexity in the transition matrix
also increases. As an option, researchers can convert the column headers into shorter names and utilize a legend to improve the readability of the transition matrix as the number of AOI increases. Similarly, the radial transition graph and the scarf plot also start to increase in complexity when the number of AOI increases. A benefit to helping both the radial transition graph and the scarf plot are interactive versions that help to improve the readability when the number of AOI increases significantly. Many of the visualization techniques work well for a small number of stimuli content or a limited number of AOIs that are investigated. This is further complicated when developers are working tasks that span across several stages of the SDLC and utilize different stimuli media in different fashions than other developers.
Chapter 3. Eye Tracking Design and Analysis Framework

Our Eye Tracking Design and Analysis Framework encompasses several key aspects that are crucial for understanding secure coders’ interactions with stimuli elements during their coding tasks. Our framework explores a range of eye tracking stimuli elements that are presented to developers for their observation and interaction. The level of control that secure coders have over their eye tracking stimuli is examined in order to investigate their ability to interact with and control the presented stimuli. Our framework offers the ability to gain insight into the processes and visual patterns of software developers during their coding tasks.

3.1 Eye Tracking Design and Analysis Framework

The type of stimuli classification listed in our framework is well-defined in the eye tracking community [81, 82, 95, 160]. However, it is often not straightforward to determine what type of stimuli to use to gain insight into the behavior of participants. Our findings suggest that the type of tasks as well as the hypotheses of the researcher will influence the decision as to the stimuli type that are needed for each eye tracking study. We developed a framework to provide an overview of the design and analysis decisions that will help researchers understand how
to utilize visualizations for various types of eye tracking stimuli in order to gain insight into the behaviors and patterns of participants.

The type of tasks for software developers will vary depending on if a developer is writing new source code, searching for the location of flaws by reading existing source code, or if they want to mitigate a bug by modifying existing source code. Furthermore, developers may utilize other applications to get assistance searching for coding function or finding flaws in source code such as static or dynamic code analysis tools. An eye-tracking study of software developers may involve various stimuli content to capture the behavior of how each developer approaches diverse types of tasks. By using different stimuli content, this allows us to examine the unique behavior of developers working on different problems. Furthermore, some activities, such as writing code to mitigate a software flaw, will require a developer to modify the stimuli presented to them at their own determination and pace. Coding developers may have multiple types of stimuli displayed on a computer monitor simultaneously. A developer’s ability to control the stimuli presented, change the flow of events, and potentially modify the stimuli content is a unique category in our framework. Our findings suggest that the types of tasks will influence the decision as to type of stimuli utilized and if each secure coding developer is allowed to have any control over the stimuli.

Our framework categories are defined based upon the type of tasks and content used in many of the stages of the SDLC. Eye tracking researchers have commonly grouped the analysis in two major groups such as the quantitative statistical metrics or graphical visualization techniques [44, 56, 82, 95]. Researchers
also utilize AOIs to create boundaries on stimuli content that can be utilized in the analysis process to limit or group elements within stimuli content. Our research primary consisted of a single AOI or multiple AOIs on stimuli. Another type is overlaying or overlapping a AOI on top of another AOI [95]. This requires consideration in the analysis to avoid misrepresenting the eye tracking data with overlapping AOIs. Another group in our framework is based upon the need to analyze developers individually or analyze a group of developers.

Our framework was developed specifically for software development tasks; however, many of the classifications could be applicable to research areas beyond software development. The literature is currently lacking with detailed approaches on how to handle stimuli content that is participant controlled and modifiable during the eye tracking recording session. This type of control over the stimuli is especially critical for us to understand the behavior of software developers working secure coding activities in a natural software programming environment. We believe that this is also applicable to other research areas as well that utilize multiple diverse types of stimuli content and eye tracking when participants are allowed to control the flow of events or modify the stimuli.

The design of experiments for eye tracking research involves several steps before any eye gaze data are collected. Determining the type of stimuli that will be presented to participants is a critical aspect to having a successful eye tracking study. Often pilot studies can provide critical and useful feedback to improve the design and setup. Our framework includes multiple categories to help eye tracking researchers determine the type of stimuli that should be used
based upon the type of tasks of objectives in an eye tracking study. In post-
processing, different analysis methods may be used depending upon the stimuli
content or the tasks that participants completed in the eye tracking study. We
developed our framework based upon the design of the eye tracking experiment
setup, post-processing transformations and how analysts can approach gaining
insight from the collected eye gaze data.

Figure 3.1 provides an overview of the SDLC with specific objectives that
are often focused on the various stages of software design, writing source code,
testing software code as well as general tasks that span multiple stages of the
SDLC. Our framework illustrates that the various stages of software coding and
testing that have specific types of tasks that can be examined with eye tracking
technologies. The software design stage often involves natural language reading
while the software coding stage generally contains a focus on reading and writing
of source code in a specific programming language. As a result of both our research
as well our literature review, this has revealed that the type of tasks can influence
the type of eye tracking stimuli and how the stimuli are presented to an eye
tracking participant. This can also impact the type of analysis that is performed
on the eye gaze data after collection is completed. Our literature survey focused
on analyzing and comparing several eye gaze visualization techniques with the
intention of developing a guideline that will assist researchers in determining the
types of stimuli to use for specific types of software development tasks in addition
to providing details on what type of analysis diagrams can provide value based
upon the software development tasks. Our focus is on the software coding and

93
software testing phase of the SDLC for secure coding; however, our findings with our stimuli analysis techniques may also provide value to the other phases of the SDLC.

We utilized eye gaze data from an experiment that we conducted where each participant was given various software reviews and coding tasks for secure coding problems. The software tasks in our experiment included the following objectives from the SDLC:

- reading problem statements and short English paragraphs about programming flaws that could exist in source code;
- reading existing source code that contained software flaws, specifically security flaws;
- reading existing source code that contained mitigation techniques for specific security bugs;
- writing actual source code by modifying existing source code;
- utilizing web browsers and software code scanning tools;
- testing of software modifications;
- reading online resources via the web browser for assistance in coding solutions to software programming flaws.

Our decision matrix presented in Figure 3.2 allowed us to determine how to design the eye tracking stimuli content and perform the analysis based upon
the software tasks. Our approach was to examine other researchers’ visualization techniques but utilize our own eye gaze dataset to illustrate the visualization methods we discovered in our literature review. This approach allows for a consistent comparison of the visualization techniques since the dataset utilized for the generation of the visualization was collected with all participants using the same software code. The mapping of the visualization techniques to our categories in the design and analysis of eye tracking data is shown in Table 3.1 and Table 3.2. In Table 3.1 and Table 3.2, we provide recommendations to other eye tracking researchers when specific types of visualization methods work well based upon the design and analysis categories defined in our framework in Figure 3.1 and Figure 3.2. Table 3.1 and Table 3.2 create the linkage between the visualization techniques utilized for analysis and our eye tracking framework.

Our framework briefly mentions the commonly identified stages of the SDLC to acknowledge that the stages exist; however, our focus with eye tracking visualization analysis is concentrated on the software coding and software testing phases of the SDLC since that is where secure coding problems should be examined. Eye tracking could potentially provide benefits in the requirements definition, software design, software coding, software testing, and software maintenance; but we have only examined software coding and software testing and therefore cannot validate these eye gaze visualization methods for other stages. Our work does not cover every potential usage in the SDLC as that would significantly increase the amount of visualization analysis. Our work briefly considers the natural language reading of documentation and normal reading instructions.
for participants on how to find and resolve software flaws in existing source code. The majority of our eye tracking stimuli are focused on existing source code while providing each participant the ability to modify the source code that is used as eye tracking stimuli. Other aspects such as UML, software requirements documentation reading/writing and database structure definition is beyond the scope of this dissertation. Furthermore, aspects such as front-end graphical user interface design, automated software deployment, cloud server storage and database management, while are aspects in the SDLC, those areas are also beyond the scope of this dissertation. Some of the design and visualization techniques may provide value in these areas as well, but we have not fully explored these areas to fully validate and endorse these visualization methods for every single stage of the SDLC.
Figure 3.1: Software Development Lifecycle (left side) and our Eye Tracking Framework (right side) for Investigating Developers’ Behaviors and Patterns in the Software Development Lifecycle.
<table>
<thead>
<tr>
<th>Software Development Life Cycle Objective</th>
<th>Types of Tasks</th>
<th>Type of Stimuli</th>
<th>View Dimensions</th>
<th>Developer Interactive</th>
<th>Quantitative vs Qualitative</th>
<th>Comparison Perspective</th>
<th>Areas of Interest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reading Problem Statement</td>
<td>Natural Language Reading</td>
<td>Static Content</td>
<td>Single Stimuli</td>
<td>Non-Changeable</td>
<td>Descriptive Statistical</td>
<td>Single Developer</td>
<td>Single AOi</td>
</tr>
<tr>
<td></td>
<td>Reading Source Code</td>
<td></td>
<td></td>
<td></td>
<td>Statistical Graphic</td>
<td>Multiple Developers</td>
<td>Multiple AOIs</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Infographics</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Answering Problem Statement</td>
<td>Natural Language Reading</td>
<td>Static Content</td>
<td>Single Stimuli</td>
<td>Non-Changeable</td>
<td>Descriptive Statistical</td>
<td>Single Developer</td>
<td>Single AOi</td>
</tr>
<tr>
<td></td>
<td>Linear Dynamic Content</td>
<td>Side-by-side Multiple Stimuli</td>
<td>Statistical Graphic</td>
<td>Multiple Developers</td>
<td>Overlapping AOIs</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Non-Linear Dynamic Content</td>
<td>Overlapping Multiple Stimuli</td>
<td>Statistical Graphic</td>
<td>Multiple Developers</td>
<td>Overlapping AOIs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reading Software Code</td>
<td>Reading Source Code</td>
<td>Static Content</td>
<td>Single Stimuli</td>
<td>Non-Changeable</td>
<td>Descriptive Statistical</td>
<td>Single Developer</td>
<td>Single AOi</td>
</tr>
<tr>
<td></td>
<td>Using IDEs/Tools</td>
<td>Linear Dynamic Content</td>
<td>Side-by-side Multiple Stimuli</td>
<td>Statistical Graphic</td>
<td>Multiple Developers</td>
<td>Multiple AOIs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Writing Source Code</td>
<td>Non-Linear Dynamic Content</td>
<td>Overlapping Multiple Stimuli</td>
<td>Statistical Graphic</td>
<td>Multiple Developers</td>
<td>Multiple AOIs</td>
<td></td>
</tr>
<tr>
<td>Writing Software Source Code</td>
<td>Reading Source Code</td>
<td>Non-Linear Dynamic Content</td>
<td>Single Stimuli</td>
<td>Interactable</td>
<td>Descriptive Statistical</td>
<td>Single Developer</td>
<td>Single AOi</td>
</tr>
<tr>
<td></td>
<td>Using IDEs/Tools</td>
<td>Linear Dynamic Content</td>
<td>Side-by-side Multiple Stimuli</td>
<td>Statistical Graphic</td>
<td>Multiple Developers</td>
<td>Multiple AOIs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Writing Source Code</td>
<td>Non-Linear Dynamic Content</td>
<td>Overlapping Multiple Stimuli</td>
<td>Statistical Graphic</td>
<td>Multiple Developers</td>
<td>Multiple AOIs</td>
<td></td>
</tr>
<tr>
<td>Utilizing IDEs</td>
<td>Reading Source Code</td>
<td>Non-Linear Dynamic Content</td>
<td>Single Stimuli</td>
<td>Interactable</td>
<td>Descriptive Statistical</td>
<td>Single Developer</td>
<td>Single AOi</td>
</tr>
<tr>
<td></td>
<td>Using IDEs/Tools</td>
<td>Linear Dynamic Content</td>
<td>Side-by-side Multiple Stimuli</td>
<td>Statistical Graphic</td>
<td>Multiple Developers</td>
<td>Multiple AOIs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Writing Source Code</td>
<td>Non-Linear Dynamic Content</td>
<td>Overlapping Multiple Stimuli</td>
<td>Statistical Graphic</td>
<td>Multiple Developers</td>
<td>Multiple AOIs</td>
<td></td>
</tr>
<tr>
<td>Code Reviews / Walkthroughs</td>
<td>Reading Source Code</td>
<td>Non-Linear Dynamic Content</td>
<td>Single Stimuli</td>
<td>Non-Changeable</td>
<td>Descriptive Statistical</td>
<td>Single Developer</td>
<td>Single AOi</td>
</tr>
<tr>
<td></td>
<td>Using IDEs/Tools</td>
<td>Linear Dynamic Content</td>
<td>Side-by-side Multiple Stimuli</td>
<td>Statistical Graphic</td>
<td>Multiple Developers</td>
<td>Multiple AOIs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Writing Source Code</td>
<td>Non-Linear Dynamic Content</td>
<td>Overlapping Multiple Stimuli</td>
<td>Statistical Graphic</td>
<td>Multiple Developers</td>
<td>Multiple AOIs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Using IDEs/Tools</td>
<td>Linear Dynamic Content</td>
<td>Side-by-side Multiple Stimuli</td>
<td>Statistical Graphic</td>
<td>Multiple Developers</td>
<td>Multiple AOIs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Writing Source Code</td>
<td>Non-Linear Dynamic Content</td>
<td>Overlapping Multiple Stimuli</td>
<td>Statistical Graphic</td>
<td>Multiple Developers</td>
<td>Multiple AOIs</td>
<td></td>
</tr>
<tr>
<td>Testing Code</td>
<td>Reading Source Code</td>
<td>Non-Linear Dynamic Content</td>
<td>Single Stimuli</td>
<td>Interactable</td>
<td>Descriptive Statistical</td>
<td>Single Developer</td>
<td>Single AOi</td>
</tr>
<tr>
<td></td>
<td>Using IDEs/Tools</td>
<td>Linear Dynamic Content</td>
<td>Side-by-side Multiple Stimuli</td>
<td>Statistical Graphic</td>
<td>Multiple Developers</td>
<td>Multiple AOIs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Writing Source Code</td>
<td>Non-Linear Dynamic Content</td>
<td>Overlapping Multiple Stimuli</td>
<td>Statistical Graphic</td>
<td>Multiple Developers</td>
<td>Multiple AOIs</td>
<td></td>
</tr>
<tr>
<td>Reading Documentation</td>
<td>Natural Language Reading</td>
<td>Static Content</td>
<td>Single Stimuli</td>
<td>Non-Changeable</td>
<td>Descriptive Statistical</td>
<td>Single Developer</td>
<td>Single AOi</td>
</tr>
<tr>
<td></td>
<td>Linear Dynamic Content</td>
<td>Side-by-side Multiple Stimuli</td>
<td>Statistical Graphic</td>
<td>Multiple Developers</td>
<td>Overlapping AOIs</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Overlapping Multiple Stimuli</td>
<td>Statistical Graphic</td>
<td>Multiple Developers</td>
<td>Overlapping AOIs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Researching Online Guidance / Solutions</td>
<td>Natural Language Reading</td>
<td>Static Content</td>
<td>Single Stimuli</td>
<td>Interactable</td>
<td>Descriptive Statistical</td>
<td>Single Developer</td>
<td>Single AOi</td>
</tr>
<tr>
<td></td>
<td>Using IDEs/Tools</td>
<td>Linear Dynamic Content</td>
<td>Side-by-side Multiple Stimuli</td>
<td>Statistical Graphic</td>
<td>Multiple Developers</td>
<td>Multiple AOIs</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 3.2:** Decision Matrix for Designing Eye Tracking Stimuli and Analyzing the Eye Gaze Data when Investigating Software Developers’ Behaviors using Eye Tracking.
Table 3.1: Visualization Techniques for Graphically Analyzing Eye Gazes and Software Developer Behaviors Set 1.

<table>
<thead>
<tr>
<th>Types of Tasks</th>
<th>Types of Stimuli</th>
<th>Viewing Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Natural Language Reading</td>
<td>Reading Source Code</td>
<td>Writing Source Code</td>
</tr>
<tr>
<td>Heat Map</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Scan Path</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Clustering</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Fixation Table</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Transition Matrix</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Icicle Plots</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Radial Transition Graphs</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Scarf Plots</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Swimlanes</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Transition State Diagram</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Pupil Boxplots</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Pupil SMA Window</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Table 3.2: Visualization Techniques for Graphically Analyzing Eye Gazes and Software Developer Behaviors Set 2.

<table>
<thead>
<tr>
<th>Developer Interactive</th>
<th>Areas of Interest (AOIs)</th>
<th>Comparison Perspective</th>
<th>Quantitative vs Qualitative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developer Non-Changeable</td>
<td>Developer Interactable</td>
<td>Single AOI</td>
<td>Multiple AOIs (&lt; 10)</td>
</tr>
<tr>
<td>Heat Map</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Scan Path</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Clustering</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Fixation Table</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Transition Matrix</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Icicle Plots</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Radial Transition Graphs</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Scarf Plots</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Swimlanes</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Transition State Diagram</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Pupil Boxplots</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Pupil SMA Window</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

100
Our approach for design and analysis of eye gazes for software coding and software testing is agnostic of the project management techniques that are commonly utilized such as Agile, Waterfall, Scrum, or Kanban. Our approach of reading documentation, reading source code, modifying source code, and using software scanning tools is agnostic of the project management techniques. Our focus was on the software coding and software testing phase since this is where the secure coding activities typically occur. However, the eye tracking stimuli including natural language text commonly found in software design and software requirements documentation could potentially be analyzed in a similar fashion. Our research has not explored the various types of stimuli or tasks in the software design phase or the software requirements gathering phase to fully evaluate the visualization methods for those areas.

3.2 Software Developer Tasks

Software developers may need multiple diverse types of stimuli while their eye gaze is recorded with an eye tracker. Their tasks may involve reading paragraphs of documentation text, reading software source code, and writing source code as well as using software tools to analyze source code. Developers can be presented with documentation discussing software coding best practices, software design patterns, algorithm design, software coding examples and code with software flaws. Furthermore, developers can be presented with questions and hands-on coding exercises related to phases in the SDLC. A participant may be presented with diagramming models, such as UML models, of how a software sys-
tem is designed if the diagramming models were created and maintained as the software programming code was developed [154, 155, 163]. Each type of task may have slightly different methods of presentation for the stimuli element. This will influence the different methods of analyzing the eye gaze based on the different tasks.

We conducted a literature survey to investigate the published methods that are used to visualize eye gazes over several types of stimuli content that could be used for software tasks. Our investigation focused on finding techniques to visualize the behavior and determine patterns in eye gazes for both static and dynamic changing stimuli as well as developer modifiable and interactive software coding tasks. Our framework design also includes analysis of single and multiple developer comparison techniques. Researchers have previously conducted literature reviews of using eye tracking technologies and performing various analysis techniques [44, 56, 99]. Several researchers limit the ability for a developer to scroll over stimuli and often limit the size and scope of the stimuli displayed to a developer [97–100]. Reading source code as well as writing source code for software applications do not exactly follow a natural left to right and top to bottom reading approach as often found in textbooks and novels [111, 131]. Software development activities are different than standard reading tasks and the types of stimuli used in many eye-tracking studies [38]. Our research focused on exploring existing techniques when applied within the software development domain, in particular the software coding and software testing phase in order to discover the strengths and weaknesses of existing visualization techniques. Furthermore,
we researched the ability to create new visualization methods for applying in the software domain.

The following are software coding tasks from our framework that we included in our eye tracking experiment:

- Reading short English paragraphs about programming flaws
- Reading existing source code with software programming flaws
- Reading existing source code with checks to prevent software crashes
- Writing source code by modifying existing code with flaws
- Using web browsers to search for coding guidelines and assistance
- Testing of source code modifications for software flaw resolution
- Running software scanning application to automatically discover software flaws

3.3 Types of Stimuli Elements

The stimuli type categorization is a common classification defined by many eye tracking manufacturers [58, 164], and researchers in eye tracking [81, 82, 115]. Static stimuli content is defined as content that does not change [82]. Examples include images or motionless visualizations. An example is the usage of a stationary photo of a scene for static stimuli content. Dynamic stimuli content is defined as stimuli that is animated. Dynamic stimuli can be further defined as linear or
non-linear stimuli. Linear dynamic stimuli content is defined as animated content that is the same for all participants. An example is a prerecorded video that is shown to participants. Each participant would watch the prerecorded video for the same length of time. The participants cannot change the flow of events in linear dynamic stimuli. Whereas non-linear dynamic stimuli content is defined as animated content that could potentially be different for each participant based upon the choices of each participant. An example of non-linear dynamic content is web-based stimuli or participant interactive stimuli where the participant modifies the flow of stimuli presented in real-time [81, 82]. The integration of the participant into the flow of stimuli elements adds additional complexity to the creation of AOIs. The type of stimuli will influence the creation and editing of AOIs as well as the type of visualization algorithm needed to transform the raw eye gaze data into insight revealing each participants’ behaviors.

The following are types of eye tracking stimuli from our framework that we included in our eye tracking experiment:

- Static stimuli for reading short English paragraphs about programming flaws, reading existing source code with software programming flaws and for reading existing source code with checks to prevent software crashes

- Non-linear dynamic stimuli for writing source code by modifying existing code with flaws, using web browser to search for coding guidelines and assistance, testing of source code modifications for software flaw resolutions, and running software scanning application to scan for software flaws
3.4 Viewing Dimensions

Participants may be presented a single type of stimuli content at a time or potentially be provided multiple types of stimuli content. The activities required for software development may require a developer to utilize code editing applications, web browsers, and code analysis tools to find and mitigate software flaws. A developer may often utilize multiple diverse types of applications side-by-side on their computer monitor. Therefore, our research explores how to utilize eye tracking when a developer switches between multiple diverse types of stimuli that are displayed on a computer monitor at the same time. Our research and visualization findings are not limited to only software coding tasks but are applicable to other areas that rely upon stimuli elements that are non-linear dynamic and participant controllable.

The following are the viewing dimensions from our framework that we included in our eye tracking experiment:

- Single view static stimuli completely filling monitor display

- Split-screen non-linear dynamic stimuli for source code viewing and editing source code on half of display and web browser for internet programming assistance searching on second half of monitor

3.5 Software Developer Control of Stimuli

A simple form of stimuli that can be utilized in an eye tracking study is participant non-changeable stimuli that is the same for all participants. Dynamic
content that changes every few frames and the same change occurs for all participants at the same time step is termed linear dynamic stimuli. An example of linear dynamic stimuli is a prerecorded video. Interactive or user modifiable content is also considered dynamic stimuli. However, since each participant can control the flow of events then each participant presented stimuli will not exactly match the ordering of other participant stimuli. This type of non-linear dynamic interactive and user-controlled stimuli content requires additional complexity to analyze than linear dynamic content as each participant will have a unique recording [95, 164]. An example of dynamic stimuli that updates based upon user interaction is the scrolling of a website or multiple page document. Another example for non-linear dynamic stimuli content is when each participant modifies the stimuli presented therefore causing each participant to have a different ordering of events with interactive content.

The eye tracking experiment that we conducted to perform this visualization analysis study did permit software participants the ability to scroll in the source code as well as modify the actual source code presented as eye tracking stimuli. Our research study was conducted by showing the simpler static stimuli to the participants at the start of the session. The non-linear dynamic stimuli that we allowed each developer to change the eye tracking stimuli real-time were presented towards the end of their recording session for the eye tracking study.
3.6 Areas of Interest (AOIs)

A post-processing phase that research analysts utilize to examine subsets of the collected eye gaze data points is the creation and usage of AOIs. This is often created on top of the stimuli that are presented [54]. These are typically manually drawn on stimuli content using an AOI editor tool [59]. Eye tracking applications may provide the software as a built-in tool for drawing and modifying AOIs [58]. Specifically, the creation of AOIs allows for select regions of the stimuli to be separated out and then the eye gaze analyzed separately from the entire raw gaze dataset. The creation of AOIs allows for the inspection of transitions between regions of the same stimuli. The creation of an AOI can simply be a total encompassing AOI around an entire stimuli content. A single AOI may be beneficial for general analysis of a stimulus that is simple. When a stimulus content contains several regions of interest, then multiple AOIs on a single stimuli element is often better for analyzing the content in the stimuli. It is typically recommended that researchers create AOIs based upon their hypothesis [82]. Generally, it is better to not have a single large AOI that encompasses several aspects of a stimulus [59]. Overlapping AOI can also be created for complex scenes with popup windows. The creation and usage of AOIs are not absolutely required to study the eye gazes of participants as many visualizations and statistical techniques can provide insight without AOIs. AOIs simply allows for eye tracking metrics to be calculated on sub-regions of a single stimuli content.
The stimuli content, the research questions and hypotheses developed are a contributing factor in deciding to utilize AOIs or simply the full stimuli alone.

 Often with static stimuli, the AOIs can be created before any data recording is performed. However, with non-linear dynamic stimuli content, then it is not possible to create AOIs until after the eye tracking recording is completed. Participants may control the flow of stimuli presented and therefore control the flow of AOIs. Interactive content allows each participant to have variations in the specific ordering of AOIs events. The creation of AOIs for video recording content that changes based upon user interactions is very time-consuming as each participant has a different recording that must be annotated with an AOI editor tool. The specific AOI editor tool that allowed us to process both the static and non-linear dynamic stimuli and generate our AOIs is identified as Tobii Studio [58]. This can be handled by stepping through each frame of each participant recording and then updating, moving or turning on/off the AOIs [95]. It is a priority to maintain consistency in creating AOIs in order to keep from potentially affecting any hypotheses post-hoc of data collection.

3.7 Quantitative vs Qualitative Analysis

Several statistical measurements exist for eye tracking that are often applied for transforming raw eye gaze into meaningful insight. These are often calculated as either a count, a mean, a maximum value, a minimum value, or a summation value [58] that provide a quantitative measurement [81]. Often the metrics provide a time-based or temporal aspect; however, some of the metrics
provide a location or spatial-focused measurement. Prior literature review publications indicate that more researchers have focused on temporal metrics and less publications on spatial metrics. Researchers discovered that spatial metrics were used more for experiments involving learning techniques and decision making problems [44].

A common eye tracking metric is the number of fixations or fixation count. This is a count for each fixation for a given timespan or a given stimuli or a AOI in a scene. This is useful to understand which stimuli a participant viewed more than other stimuli content. Other researchers have found that fixation count has potential in software maintenance tasks [114]. Another common measurement is the fixation duration which also gives similar meaning but with a time measurement. The fixation duration is a measure for how long the eye remained stationary in a position [59]. The fixation duration is often determined as an average but can also be calculated for the first fixation duration or total fixation duration. The average can be expressed per each AOI or across all AOIs. Prior researchers have shown that participants with longer mean fixation durations may have greater difficulty processing the stimuli content [43]. The first fixation duration can be used when examining specific areas of stimuli content. Another common time calculation metric is the measurement of the time until the first fixation. This is useful to provide information to researchers about which stimuli was attended first by each participant and for how much time has passed until each participant looked at the stimuli [165]. Researchers have found that fixation based measurements have
been explored by several eye tracking researchers for software development tasks [87].

Another frequently used technique of understanding eye gaze is to utilize visualizations and graphical techniques. This is occasionally used in a stand-alone fashion but often is observed alongside the stimuli that were presented to participants. Visualizations can provide additional insight into the eye gaze such as the ability to understand transitions and sequences that viewers performed when looking at stimuli content. Graphical visualizations are capable of providing a spatiotemporal viewpoint of the eye gaze [81]. The sequences of fixations can help to determine patterns in the behaviors of participants. The transitions between stimuli content are useful to understand the steps that participants used to solve problem. Researchers have examined repeated transitions between content as an implication of greater cognitive demand [95]. The ability to graphically represent eye gaze revisits to previous stimuli content is possible with visualization methods. Our work focused on visualization methods for eye tracking with non-linear dynamic stimuli content activities as well as visualization of participants using multiple types of stimuli content.

3.8 Analysis and Comparison of Developers

Eye tracking research studies involve many developers that participate in the overall study and may come from various backgrounds and experience levels. Depending upon the research study and stimuli, this may involve splitting developers into control or experimental groups with various independent variables.
applied. The analysis of the eye gaze data may be examined at an individual
developer level or by comparing and contrasting several developers at once. The
number of developers may determine how the analysis of eye gaze data is accom-
plished. Some visualization methods work well for single developers while other
methods have limitations when the number of developers increase, specifically to
visually understand the eye gaze data points or patterns for multiple developers.
Other visualization techniques work well for multiple developers; however, the
techniques may provide incomplete analysis for a single developer [82].
Chapter 4. Secure Coding Learning Module Methodology

Our research focused on crafting a step-by-step approach to teach participants various cybersecurity software programming flaws. Specifically, we aimed to illustrate how these software coding flaws could manifest within a photo and video sharing website called ShareAlbum that was created at UAH. This chapter provides an overview of how the eye tracking stimuli represented the content from our secure coding learning modules in order to evaluate how participants progressed through the learning modules using eye tracking technologies. Our secure coding learning modules were thoughtfully designed to build a student’s confidence in solving secure coding problem by increasing the complexity of the problem in tandem with a student’s progress as a secure coder.

4.1 Designing Learning Modules for Secure Coding

We needed a software application with security flaws in order for the students to gain hands-on experience learning abouts CWEs and how to find the flaws in source code and mitigate the vulnerabilities. We also determined that we should provide students with a realistic software application with security flaws. One main objective of this realistic software application requirement is that we wanted the students to work with a meaningful length of source code that is used
to build an entire application. The reason was to provide students a more realistic code base that they might work with in industry. We did not want students to examine a single programming function that is pulled out of an entire application. This removes the context associated when a function is pulled out of an application and the context surrounding the method is removed. A single method with a few lines of source code is not as realistic as they may find in a daily working job and therefore, we wanted a code base designed for a full running application. Additionally, we elected to select an application with security flaws already existing in the source code. Fortunately, we had access to a custom web-based photo and video sharing project that has security flaws existing in the source code. This ShareAlbum application source code has similar functionality that is similar to the weaknesses associated with CWEs.

The ShareAlbum application’s primary user focus is for the uploading and sharing of photos and videos through user albums via a web browser. This provides many opportunities for security flaws as the application is written in the PHP: Hypertext Preprocessor (PHP) and HyperText Markup Language (HTML) programming languages with a MySQL backend for data storage. Many of the weaknesses in the source code for ShareAlbum are listed in the CWE repository and NVD database [2]. The photo and video sharing web-application consists of approximately thirty files of source code. We have full access to the source code and can set up the photo and video sharing website as a local hosted application. The ShareAlbum project can be utilized to create and share albums, upload photos and upload videos. Registered users can tag photos, videos, and
albums as well as send notifications with other registered users. The *ShareAlbum* application was a project at our university. Two student developers, who studied cybersecurity and were on a coding competition team that won locally, and a global award, created the *ShareAlbum* application. The software has many secure coding errors as security of the application was not a focus when it was developed. We believe that our web-based photo and video sharing project provides students a more realistic example of mitigating security flaws in an application versus a single program function.

We elected to utilize the *ShareAlbum* application for our eye tracking research study due to the close similarities between examples shown in the CWE repository and security flaws in the *ShareAlbum* PHP and HTML source code. Nearly 80% of server-side website processing utilizes PHP and PHP is still in active development [166, 167]. Once students are aware of how to identify security flaws and develop mitigation strategies, then they can deploy that experience to other programming languages. Some of the learning content requires our secure coders to utilize third-party security scanning software to help facilitate the secure coders finding of the security flaws.

In secure coding tasks, distinct types of source code files, security vulnerabilities, questions and tools are presented in several stages. The secure coding learning modules we utilized in our research focused on identification and mitigation techniques that are discussed in the CWE repository. The Top 25 list was first released in 2009 and updated in 2011 and again in 2019. The weaknesses are based upon vulnerabilities found in the NVD [168]. This list of the Top 25
software errors provides a detailed description of common software security design and coding flaws. The overall CWE repository can be helpful to software engineers, project managers, customers, and academia researchers to understand common cybersecurity weaknesses and vulnerabilities. Details of common programming weaknesses, sample code of weaknesses, sample code of exploitation techniques, and countermeasures are provided within the CWE repository [6, 68].

Our secure coding learning modules are designed to utilize a progressive approach to teach students secure coding by identifying and fixing security weaknesses. We started with an introduction identifying the CWE repository both in class and hand out material. Countermeasures and mitigation strategies are provided for each CWE based upon information in the CWE repository. Then, our learning modules provided a sample program with source code that has known security flaws that allow for cyber exploitation. Exploitation techniques and finally mitigation strategies for the security weaknesses are discussed. This includes sample code of weaknesses and sample code of exploitation techniques. We demonstrate how to develop secure coding to prevent specific weaknesses. We finish with an overall discussion on the material presented. We provide videos of our learning modules as guides to help provide hands-on learning. This provides video demonstrations of the source code so that participants can experiment with the material presented in the learning modules and help demonstrate the expected outcome from the secure coding learning modules.

There are many software applications within which these common security errors could exist. Therefore, we had to narrow the focus to create our
progressive-styled learning module for secure coding. We designed our modules to actively engage a participant to learn practical cybersecurity strategies through secure coding. Overall, our learning modules provide background information, description of common security issues and mitigation strategies. All material is available online to be utilized by our students as well as other universities.

4.2 Designing Eye Tracking Stimuli from Secure Coding Learning Modules

The secure coding programming problems presented in our eye tracking research study increased in complexity and difficulty. In our research study, participants progressed through our hands-on secure coding learning exercises, in which the security weaknesses that were easier to find and mitigate were presented first and presented as multiple choice or true/false questions. More challenging security weaknesses was presented afterwards, and participants had to modify actual source code as part of the task. For the Manual Code Analysis Learning Module, our effort was focused on the creation of a coding exercise that involved a manual code review without the usage of any code analysis tool suites. This involved a line-by-line manual reviewing of the code. It is recommended that an organization utilize manual and automated methods for secure code [169, 170]. For the Static Code Analysis Tool Learning Module, our effort focused on creating an exercise that utilized the Research and Innovation to Promote Security (RIPS) static code analysis tool. This involved running the RIPS analysis tool on the source code.
and examining the reported finding from the tool to determine if a vulnerability existed in the ShareAlbum application.

Our eye tracking experiment setup and design had to be able to manage the different types of stimuli content presented. Frequently, a participant may have multiple types of stimuli on their computer monitor at once and transition repeatedly between stimuli content of distinct types. Figure 4.1 provides an example split screen of our photo and video sharing login webpage alongside the instructions guide. An example is shown in Figure 4.2 of how participants used both the source code editor and the web browser in a split screen fashion. Figure 4.3 provides an example split screen of our photo and video sharing home web navigator alongside the instructions guide. Participants switched between multiple source code files and webpages at their own determination. The goal of this approach is to allow participants a natural environment to find and mitigate security weaknesses and at the same time allow the eye tracker to capture their behavior and approaches. Secure coders were asked to keep the split screen view in order to allow for analysis of their behavior.
Figure 4.1: Split Screen between ShareAlbum Login and Instructions / Guidance Text.

Figure 4.2: Split Screen between RIPS Tool in Web Browser and Code Editor.
Our Manual Code Analysis Learning Module focused on reading documentation of static stimuli content for our secure coders to gain a general understanding of secure coding flaws with the CWE repository. Furthermore, our Manual Code Analysis Learning Module also provided the opportunity to manually review our web-based photo and video sharing project and the associated source code. This manual review of the webpages and files of source code requires utilizing dynamic non-linear eye tracking stimuli content. This permits the participants the ability to switch between multiple source code files and scroll through the various code functions defined in the source code of the web-based project. Participants may switch attention between multiple source code functions, source code files as well as other applications such as web pages. Furthermore, our secure coders were
given the opportunity to actually write new source code versus simply reading
source code. Therefore, this type of stimuli content is actually modified by each
coder during the eye tracking recording session. There are various levels of user
interactive content that must be examined for secure coders. Our Manual Code
Analysis Learning Module requires the secure coder to interact with multiple
distinct types of stimuli content to accomplish all of the objectives.

Our Static Code Analysis Tool Learning Module allowed each secure coder
the ability to utilize a static code analysis tool to scan source code to find vari-
ous security weaknesses. It is recommended by security specialists for secure
coders to have the ability to conduct manual code reviews but also utilize secu-
code reviews may work adequately for small software applications; however, for
large-scale software applications or limitations due to hidden software compo-
nents, then additional assistants are needed to help uncover defects. The RIPS
static code analysis tool was selected for our Static Code Analysis Tool Learn-
ing Module [173]. The static code analysis tool adds even more user interactive
stimuli content that further complicates the eye tracking analysis. However, the
static code analysis tool can help secure coders find security weaknesses and re-
ceive a report on potential vulnerabilities in the software application. How secure
coders analyze the report from the static code analysis tool can be beneficial to
determine what aspects they examine in the tool report.

Our two learning modules allow our participants to gain experience with
both manual code reviews and static code analysis applications procedures. A
progressive approach allowed us to initially present simple material, then progress to examples and challenging coding problems. The study was designed with step-by-step stages:

- Start students with basic concepts of secure coding
- Then students work two multiple choice programming questions
- Then to conclude students modify source code files for secure coding

Various stimuli were utilized to present the material to the participants. We utilized instruction stimuli to help guide the participants and PDFs to cover the learning content. After specific learning content was displayed, we utilized a questionnaire to ask which programming solution best addressed the weaknesses. Participants provided their responses to the multiple-choice questions directly in Tobii Studio. Then we continued with participants examining source code in Notepad++ and writing either source code, pseudocode, and/or comments. This was designed to determine if participants could correctly identify if a specific vulnerability existed and then could each student provide a mitigation technique to secure the code.

4.3 Static Stimuli for Reading of Documentation

Both of our learning modules start with a description of the CWE repository and the benefits it provides to secure coders. Both of our learning modules provided a brief introduction of several CWEs and then provided example source code of several weaknesses and finally provided several demonstration source code
snippets that could allow for exploiting a weakness in our application. This content is presented as short paragraphs that are similar to standard reading of a book or research article. Our eye tracking technology recorded the eye gaze of each secure coder while they looked through our learning content. As part of our analysis phase, we explored each coder eye gaze with various visualization methods and common eye tracking metrics to gain an understanding of their reading behavior and patterns of the ordinary documentation paragraphs of text.

4.4 Static Stimuli for Reading of Source Code

An important aspect of our research is to study the behavior and methodologies of secure coders while they perform secure coding tasks. Therefore, we presented each student several CWEs that allowed them to read existing source code functions and determine if a vulnerability existed and how to mitigate the vulnerability. This approach allowed us to introduce coding examples to the participants and present multiple choice or true/false questions as a way to introduce secure coding problems to our participants.

For our Manual Code Analysis Learning Module, after reviewing the learning content for CWE-311 [174] and CWE-434 [175], then each student was presented several static screenshot images of snippet of source code and asked which code snippet correctly mitigated the security weaknesses. No scrolling of the source code was allowed for these tasks. This was designed to allow participants who may not be familiar with writing source code to get a general understanding of reading existing source code. This allowed us to analyze the eye gaze of source
code in a similar fashion as the reading of standard document text. We presented each participant with a multiple-choice question in which the participant was presented with options to pick the source code function that correctly mitigated the weakness in the *ShareAlbum* source code. For CWE-311 each participant had three multiple choice functions to choose the correct solution and for CWE-434 each participant had two multiple choice functions to pick the correct mitigation.

### 4.5 Dynamic Stimuli for Reading of Source Code

For our Static Code Analysis Tool Learning Module, participants were required to answer if the RIPS tool provided false positives for flagging CWE-73 [176] and CWE-443 [177, 178] as potential vulnerabilities. This involved the participants reading source code, potentially using the vulnerability scanner, and answering true or false if each weakness actually existed in the code snipped shown. The Static Code Analysis Tool Learning Module did permit each secure coder the ability to scroll through source code when they are reading source code. During our analysis phase, we explored existing visualization methods and designed our own visualization approaches for analyzing eye gazes for secure coders reading dynamic source code stimuli content. For existing visualization methods, our methodology ensures a stable comparison of visualization techniques, as the dataset used to create the visualizations was compiled with all participants employing the same *ShareAlbum* software code.
4.6 User Interactive Stimuli for Patching of Vulnerabilities

The ability to read source code and discover weaknesses is a major aspect of secure coding; however, the developers should also have the ability to create a mitigation strategy and modify the source code to reduce or eliminate the vulnerability. Therefore, our research also examined participants’ eye gaze as they write source code to secure our software application. This user interactive stimuli content not only allows participants to control the flow of events by scrolling or switching between files and applications, but additionally it actually allowed a participant to modify the stimuli content that was presented. We believe this is a critical aspect to truly capture the behavior of secure coders performing secure coding exercises.

Our Manual Code Analysis Learning Module included tasks for the participants to provide programming solutions for CWE-862 [179] and CWE-22 [180]. The participants were required to manually scan the source PHP and HTML code files as well as the SQL queries to determine where weaknesses could be exploited. Then, their writing source code tasks involved the participants having the requirement to modify source code to mitigate the security flaw associated with those CWEs. Each participant modified the source code to provide their unique solution to mitigate the security flaw that they discovered in the source code. Each student started with the same baseline source code and provided their own solution to mitigate the vulnerabilities. Students were permitted to provide PHP and HTML code to mitigate the vulnerability or pseudocode if they could
not provide the actual PHP and HTML syntax correct code. However, not all students in the experiment are familiar with the PHP and HTML programming language. Therefore, we allowed the students to provide pseudocode to be able to determine if they could correctly identify where in the code the vulnerability existed and then inform us if they understand how to mitigate the weakness. Finally, if students could not provide either solution notation, then we did permit the student to at least provide an English written comment that they found in the code where the vulnerability existed and how they could recommend mitigating the vulnerability. Our grading of their solution reflected if the student got the programming aspect completely correct, partially correct, or incorrectly answered the programming assignment.

For our Static Code Analysis Tool Learning Module, participants had to write programming solutions for CWE-79 [181] in addition to using the RIPS tool to analyze the vulnerability report. Participants also had the option to use the local web-based photo and video sharing website called ShareAlbum to determine if weaknesses existed and allows participants to attempt a security exploit. Participants were permitted the ability to utilize a search engine or additional websites as needed to further research the CWE or potential solutions. In our analysis phase, we utilized existing eye tracking visualization methods but discovered limitations with existing approaches as the number of AOIs and participants increased for user interactive and user modifiable stimuli content. Therefore, we researched and designed additional visualization approaches to handle user inter-
active stimuli content for when each secure coder had the tasks of modifying the stimuli content real-time as it was presented.

4.7 Secure Coding Learning Modules in an Eye Tracking Timeline

Our secure coding learning modules were initially designed in PDF document format and the source code for ShareAlbum was provided via an online website for students to access to learn secure coding. In order to utilize the secure coding learning module material for eye tracking we decided to convert the content from the learning module into stimuli elements for tracking the eye gazes of participants. While it could be possible to simply allow participants to just openly and freely work the secure coding learning module without any guidance or restrictions; this makes it challenging to design the research study to collect meaningful results as participants progress through the learning module. The Tobii Studio software application provides a Design and Record tab that allows eye tracking researchers to design and build out their plan for the eye tracking stimuli that are presented to eye tracking participants.

An example for our Manual Code Analysis Learning Module is shown in Figure 4.4. This approach allows for presenting different types of stimuli elements such as Instructions, Images, Web, Questionnaire, PDF documents, or simply record the entire computer monitor content. We initially presented participants with an instruction overview stimulus in order to brief the participants on secure coding. This material was mostly natural language text and presented using half-page PDF stimuli elements. Next, we presented problems focused on specific
CWEs using the PDF stimuli elements and the questionnaire stimuli elements. Afterwards, we relied upon the screen recording feature to open Notepad++ and a web browser for the problems where participants were expected to review and update the actual source code that was presented in the recording session.

Figure 4.4: Tobii Studio Design and Record Tab for our Manual Code Analysis Learning Module.
Chapter 5. Eye Gaze Data Collection and Processing

In our research study, we conducted a thorough eye gaze data collection procedure for secure coding exercises. Our participants were enrolled in undergraduate and graduate computer security courses to ensure they had a basic understanding of software coding principles and a brief introduction to software security flaws. Furthermore, we manually identified and created AOIs in the collected eye gaze data to focus on specific elements of the source code. Multiple eye tracking metrics allowed for further analysis of the collected data in order to gain valuable insights into visual attention patterns. The eye gaze data was also processed using custom Python analysis scripts tailored to our research study specific requirements. This approach enabled a robust examination of the eye gaze patterns of our participants.

5.1 Procedure and Tasks Step-by-Step

Additional details on our study approval and forms provided to each participant can be found in Appendix A. Each participant received an emailed copy of the consent form to read when they scheduled their eye tracking session appointment with the lead investigator. When each participant entered the research study lab, then each participant was given a copy of the same consent form that
required their individual hand-written signature. The lead investigator asked each participant if they had any questions or concerns about being a participant in the eye tracking study. All scheduled participants agreed and signed the consent form. Only one participant was allowed in the research lab at a time and the only other individual in the lab at the same time was the lead investigator. The research lab never had two eye tracking participants present in the lab at the same time. Each participant received an overview from the lead investigator that explained the Tobii X2-60 eye tracking device and the Tobii Studio software application that would drive the experiment and display the stimuli elements. The lead investigator watched each participant from ten feet away to ensure that each participant remained at a proper distance from the eye tracking device. Each participant was allowed to have scrap paper on the tabletop next to the keyboard; however, no participant used any of the scrap paper.

Each participant was allowed to read the material at their own pace while the eye tracker recorded their eye movements across the various stimuli elements. It is critical to perform an accurate calibration for each participant as the eye gaze information will not be useful if the eye tracker is not calibrated for each participant [95]. A 5-point calibration or a 9-point calibration is a common calibration setting that is utilized to have participants fixate on that specific number of dots to find the boundary of the viewing area [58]. Initially, a participant completed the first nine-point calibration using the Tobii Studio software calibration tool. Next, the first phase of the eye recording session was started. In the first phase, participants were first presented with a welcome and introduction content
using static stimuli elements that contained natural language text about security vulnerabilities that exist in software code. A participant was shown specific types of coding flaws that could exist in source code based upon the CWE database. A participant was shown static code snippets of source code alongside some of the natural language text to introduce to participants examples of the source code. Then, a participant was presented with static code snippets of software code that either allowed a cybersecurity vulnerability to exist in the software application or a software solution that mitigated a cybersecurity vulnerability. Each participant was asked to provide their answer choice directly in Tobii Studio using Tobii Studio’s Questionnaire Element. The answer choice was provided as a multiple-choice option between two programming code snippets or three programming code snippets. The first phase of the eye tracking study was designed to provide static stimuli of natural language text, static stimuli of source code using the PHP and HTML programming languages and ask basic questions about the source code flaws. After the first recording session was completed each participant was offered the opportunity to take a break of around 10 to 15 minutes to rest.

For the second phase of each eye tracking session, a participant then completed their second nine-point calibration using the Tobii Studio software calibration tool. In the second phase, a participant was asked to review existing source code in HTML, PHP and PostgreSQL that contained a software flaw and provide their solution in the source code text editor that would resolve the software flaw. This phase of the study presented entire source code files to a participant instead
of code snippets and therefore scrolling was allowed by each participant. Fur-
thermore, unlike in the first phase when a participant answered a multiple-choice
question, this phase required a participant to modify the actual source code.
Therefore, this phase contained the non-linear dynamic eye tracking stimuli. For
example, one problem was presented containing a software flaw that could allow
for malicious input in an online form to be provided via a web page. Each partic-
ipant received an overview in natural language text and then was presented with
source code and a web browser to test out changes to the source code. The web
browser was presented on half of the monitor and the source code presented on
the other half of the monitor. Another problem was presented that was associated
with checking for valid file paths in source code when writing a file to the hard
disk to ensure that the save path exists in the operating system. Each participant
had the option of saving their source code changes and refreshing the web browser
to evaluate their code modifications. After each participant finished their second
recording session, they received a thank you message on the computer monitor
and were informed with a message on the computer monitor that the study was
completed. The source code that each participant could have potentially modified
was saved in a different directory on an encrypted hardware drive. No participant
was allowed to see the code modifications performed by any other participant.

The total recording session had a 2-hour maximum time allocation per par-
ticipant and each participant was informed before the recording session started
about the maximum time limit. The break time that some participants took
halfway through each learning module was not counted in these calculations for
the total recording duration. Table 5.1 provides the summary for both of our learning modules. Furthermore, each individual participant’s total recording duration per learning module is shown in Figure 5.1. The specific names of the participants are hidden due to privacy concerns. The participants are not provided in any specific order on the x-axis in Figure 5.1.

**Table 5.1:** Total Duration Recording Times per Secure Code Learning Module.

<table>
<thead>
<tr>
<th></th>
<th>Manual Code Analysis Learning Module</th>
<th>Static Code Analysis Tool Learning Module</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Average (minutes)</strong></td>
<td>49.66</td>
<td>54.10</td>
</tr>
<tr>
<td><strong>Minimum (minutes)</strong></td>
<td>23.97</td>
<td>23.77</td>
</tr>
<tr>
<td><strong>Maximum (minutes)</strong></td>
<td>86.15</td>
<td>93.57</td>
</tr>
<tr>
<td><strong>Median (minutes)</strong></td>
<td>49.00</td>
<td>56.10</td>
</tr>
<tr>
<td><strong>Standard Deviation</strong></td>
<td>13.33</td>
<td>19.83</td>
</tr>
<tr>
<td><strong>Standard Error</strong></td>
<td>2.47</td>
<td>3.56</td>
</tr>
</tbody>
</table>
The following are the total recording duration per each problem that was worked by our participants in the Manual Code Analysis Learning Module. Figure 5.2 contains the recording duration for CWE-311 and Figure 5.3 for CWE-434 that was presented as multiple choice options in the Manual Code Analysis Learning Module. Figure 5.4 contains the recording duration for CWE-862 and Figure 5.5 for CWE-22 that was presented as programming problems in the Manual Code Analysis Learning Module.

**Figure 5.1:** Participant Total Recording Duration per Secure Code Learning Module.
Figure 5.2: CWE-311 Recording Duration per Participant.

Figure 5.3: CWE-434 Recording Duration per Participant.
Figure 5.4: CWE-862 Recording Duration per Participant.

Figure 5.5: CWE-22 Recording Duration per Participant.
The following are the total recording duration per each problem that was worked by our participants in the Static Code Analysis Tool Learning Module. Figure 5.6 contains the recording duration for CWE-73 and Figure 5.7 for CWE-443 that was presented as true/false options in the Static Code Analysis Tool Learning Module. Figure 5.8 contains the recording duration for CWE-79 that was presented as a programming problem in the Static Code Analysis Tool Learning Module.

![Figure 5.6: CWE-73 Recording Duration per Participant.](image)

136
Figure 5.7: CWE-443 Recording Duration per Participant.

Figure 5.8: CWE-79 Recording Duration per Participant.
5.2 Participants in Our Secure Coding Eye Tracking Study

During the Fall 2019 semester at UAH, undergraduate students from UAH’s Introduction to Computer and Software Security (CS 485), at the senior level and graduate students from UAH’s Introduction to Computer and Software Security (CS 585), a first-year graduate level course, were invited to participate in our secure coding eye tracking study. We limited our search for participants to individuals in Computer Science courses since our eye tracking stimuli required a basic understanding of software coding. This research study used our two secure coding learning modules and utilized eye tracking technologies. As a benefit of attending an eye tracking recording session, the problems that they worked during the eye tracking study would not have to be worked as a course take-home assignment. No participant was monetarily compensated for their recording session. Each participant completed a post-questionnaire that requested their demographic information. Students were informed of the eye tracking research study in their coursework and were given the chance to have one extra team member on their semester project if they attended the research study. The students had an introduction to CWEs in class lectures prior to attending. All participants in the study were above the age of 18. There was no physical or psychological risk associated with this eye tracking study. Participants were not forced to provide any information that they did not want to disclose even with the information being collected anonymously.
We crafted the study to take roughly 1 to 2 hours with a break approximately at the half-way point per secure coding learning module. The eye tracker was calibrated for each student. This involved plotting and moving a small red dot on the monitor to help find the border of the display while the student fixated on the small red dot. To ensure we collected suitable quality data we also performed a second calibration cycle at the half-way point. The Tobii Studio software provided the calibration capability. We utilized the enhanced calibration option with nine points. Participants worked at their own pace while their eye gaze was recorded. At the conclusion, participants were presented with our paper questionnaire that asked the participants specific questions related to the content, their experiences and demographic information. Our questionnaire also asked participants to rank their programming experience levels. We provided four categories for participants to pick from to limit the range of responses.

The eye tracking recording sessions for the Manual Code Analysis Learning Module had 29 participants; and the eye tracking recording sessions for the Static Code Analysis Tool Learning Module had 31 participants. Our study had 26 participants that were part of both the Manual Code Analysis Learning Module and the Static Code Analysis Tool Learning Module. In Figure 5.9, for our Manual Code Analysis Learning Module, we include the following information about our participants: their identified gender, general information about their age, their identified ethnicity, their current degree, and their current experience level. In Figure 5.10, for our Static Code Analysis Tool Learning Module, we include the same information about our participants. Every participant was presented the
same learning material about security flaws in source code, and each was presented the same questions in the same order. No participant was ever allowed to see the responses or decision-making processes of any other participant. Participants were only allowed to complete one learning module per day to reduce the amount of time participants spent in one session.

Figure 5.9: Manual Code Analysis Learning Module Participant Demographics.
Given that each learning module contained 3 to 4 questions for participants to review and answer and the number of participants that volunteered for our study, our eye tracking design did not include separate groups where some participants received some questions while other participants received a different set of questions. Our research was focused on trying to understand what type of stimuli to present to participants for security coding and what type of visualizations were possible to understand where the participants were looking at. Our

Figure 5.10: Static Code Analysis Tool Learning Module Participant Demographics.
finding suggests that which visualization technique to utilize depends upon the stimuli that is presented to participants as well as the question that is asked to the participants. Our research was focused on finding what type of visualizations are even useful for dynamic non-linear eye tracking stimuli and the benefits and limitations of these various visualization methods.

In Figure 5.9 and Figure 5.10, we provide the experience level that participants answered in their questionnaire. For the senior level experience, a participant was expected to have 10 years or more of programming experience. For the mid-level experience, a participant was anticipated to have 5 to 10 years of programming experience. For the junior-level, we asked that participants mark this category if they were between 3 to 5 years of experience. For the entry-level, we recorded this was 0 to 3 years of experience.

5.3 Eye Tracking Filtering Techniques

A commonly utilized technique in eye tracking research is creating AOIs which provides significant benefits to analyzing the eye gaze data points of sub-areas of stimuli. However, with increasing the number of AOIs that must be analyzed, then this also increases the complexity to understand the eye gaze data. A similar limitation to increasing number of AOIs is a limitation when participants change AOIs frequently. The frequencies that any participant changes from one AOI to another AOI can depend upon the stimuli used in an eye tracking study, the tasks given to a participant and the individual participant experience and tendencies. Switches in attention between and within stimuli or AOIs can help
researchers understand the visual strategies over a period that participants used when inspecting the stimuli presented. Furthermore, the approaches and techniques that each participant uses when reading stimuli content can determine how frequently a participant transitions to a different AOI. Nevertheless, AOI transition rates can provide insight into the behavioral patterns of a participant.

A critical step in processing is the creation of these customized AOIs. These steps classify grouping of fixations, AOIs that participants focused on, the transition frequencies, and sequences in moving between stimuli elements or AOIs over time. They allow researchers to examine the patterns and approaches those participants used when approaching problem solving. The creation of AOIs is applicable to multiple types of stimuli. The creation of AOIs is a critical milestone for comparing and contrasting participants’ behaviors when they use multiple applications concurrently and transition often between distinct types of stimuli contents.

Most eye tracking software provides various types of filters that can be applied to the raw data to reduce the noise and to classify fixations and saccades [80, 92]. Filters are applied to categorize fixations and saccades from the raw eye tracking data. We applied Tobii’s I-VT Filter to the raw data. This is a commonly applied filter that determines the eye movement classification based upon the velocity of the directional shift of the eye [84, 92]. If the velocity of the eye movement is below a specific threshold, then the data point sample is marked as being a fixation. If it is above the threshold, then it is marked as being a saccade. This velocity is measured in visual degrees per second. For
our application of the filter, we relied upon a threshold of 30 degree per second velocity [80]. It is recommended that visual degrees are a better measurement standard than using pixels [84, 92]. The I-VT filter has additional settings such as the ability to discard very short fixations, merge adjacent fixations and select if both eyes are to be averaged. Upon examining the design and parameters of each of the three filters that Tobii Studio offers, we utilized the default settings for the parameters with the velocity threshold set to 30 degrees per second [80, 92].

5.4 Manually Creating Areas of Interest (AOIs)

As part of our processing, we created static and dynamic AOIs to investigate the eye gazes at the application level, source code file or function level as well as the programming statement level. For static AOIs we relied upon AOIs tab in Tobii Studio to create specific AOIs for our static stimuli elements. An example is shown in Figure 5.11 for one of our multiple-choice programming problems presented to participants. In this example, we created multiple AOIs for different aspects of a software programming function. The individual stimuli elements for the project are listed in the “Media” area on the left-hand side of the AOI tab. The specific AOIs that are manually created are listed in the AOI area on the left-hand side as well. Different colors can be assigned to different AOI. The AOI tool in Tobii Studio provides various shape tools that a researcher can utilize to manually create their AOI based upon their determination. There is no time component associated with static stimuli element when creating AOIs.
Figure 5.11: Creating AOIs for Static Stimuli Content.

For dynamic AOIs, this was accomplished by advancing over a video recording of a single participant and marking each frame. Each frame provided the ability to modify existing AOIs or create new AOIs per each participant. These specific frames are equivalent of markers per each participant recording over the linear time. Our experience with dynamic media and AOIs with frames has allowed for more flexibility in the content we present as stimuli; however, it does increase the time required for processing. This allows researchers to create a linear transformation of various AOIs over each recorded session. This provides the ability to overcome the limitations of some studies that limit the number of lines of source code displayed to one screen display size and limit the ability of participants to scroll [76]. It is, however, especially time consuming to draw, transform, move, turn on and off AOIs for a long duration session.
An example of a dynamic stimuli recording and creating AOI for that specific recording is shown in Figure 5.12. The layout is similar to the static stimuli element; however, a timeline is provided at the bottom of the window that allows a researcher to move back and forward over the dynamic stimuli recording to create and edit AOIs. In this case, specific AOIs can be enabled or disabled over the timeline. When an AOI is selected, the recording timeline will show a solid white line when that specific AOI is enabled and white dots when the AOI is disabled. Keypoints are provided as blue circles when the AOI is either enabled or disabled. The frame count is shown at the top center of the window to allow researchers to advance for each frame. This allows researchers to advance for the frames and add, remove, or resize any AOI. This feature of Tobii Studio was essential in our using of non-linear dynamic stimuli content. Each recording per participant is shown in the “Media” area of the AOI Tab.
5.5 Analyzing with Tobii Studio Metrics

Tobii Studio provides several commonly utilized eye tracking metrics to evaluate the eye gaze of participants. One metric is called Fixation Count and we provide an example in Figure 5.13 for multiple participants. The participant names are not directly shown to protect their identity from public release. As shown in Figure 5.13, researchers can select specific participants and specific AOIs that the researchers have created to analyze using the Statistics tab in Tobii Studio. The statistical metric can be shown in a table format as shown in Figure 5.13 or in a chart format as shown in Figure 5.14.
Figure 5.13: Tobii Studio Statistics Tab for Table Representation.

Figure 5.14: Tobii Studio Statistics Tab for Chart Representation.
5.6 Exporting Tobii Studio Eye Gaze Data

The eye gaze data that are collected in Tobii Studio can be exported for additional analysis beyond the statistical analysis methods that currently exist in Tobii Studio as shown in Figure 5.15. If a raw eye gaze data filtering method is utilized, then that filtering technique is applied to the exported data. Researchers have the option to turn off the eye gaze filtering algorithm and export the raw eye gaze data. The data that we exported from Tobii Studio had the same filter applied that was utilized when creating the AOI. The Data Export tab allows researchers to select specific participants as well as various fields that the Tobii Studio maintains per participant. For example, the \textit{AOI [Name of AOI] Hit} is an option under the section “Gaze event data and AOI activity information”. This exports all AOIs in the test set and provides a Boolean indication per participant over time when they viewed content in each AOI. The left and right pupil data can also be exported in this tab for further analysis. This ability led us to design and create our own analysis script to create our various visualization images that are included in this dissertation.
5.7 Creation of Custom Analysis Scripts

After creating the AOIs in Tobii Studio and exporting the eye gaze data to Comma Separate Values (CSV) files, then we created our custom data analysis script using the Python programming language to create visualizations of our participants’ fixations, gaze shifts, AOI transitions and behaviors. An example of the raw text-based data is shown in Figure 5.16 as well as the output from our Python data transformer and our Python AOI selector to down select to specific problems of interest to create various visualization diagrams.
Figure 5.16: Example of a Raw Eye Gaze Datafile from Tobii Studio.
Chapter 6. Visualization Analysis of Secure Coding Behavior

In our research, we extensively explored various visualization methodologies concerning eye tracking and software development. Initially, static stimuli were analyzed using simple AOIs that allows for basic observations. Then, this was followed by comparisons across multiple eye tracking participants for simple AOIs. Subsequently, a more detailed analysis was performed using specific AOIs to gain deeper insights into participants’ eye gaze patterns. The study extended to analyzing developer interactive stimuli content, evaluating multiple developers’ responses to secure coding problems. Furthermore, eye pupil diameter changes were visualized, providing valuable information on visual focus changes while our participants work secure coding exercises. The research culminated in a summary of various visualization methods employed, showcasing their effectiveness in presenting eye tracking data. Overall, our framework employed throughout our study demonstrates the significance of eye tracking as a powerful tool for understanding visual behavior and yielded valuable findings for secure coding.
6.1 Data Analysis and Visualization Methodologies

In this section, we present visualization techniques that we utilized to investigate the eye gazes of our secure coders while they performed secure coding activities. This enabled us to view eye gazes when participants read documentation, examine multiple source code functions and files, write in source code files, use the web browser, scroll through multiple pages of content. Our approach also examined their transitions between stimuli content and areas of stimuli that are of particular interest. Furthermore, this allowed us to capture the behaviors of participants viewing these types of stimuli while participants learn and work on secure coding problems. Finally, our techniques allowed us to compare and contrast participants’ strategies to discover and mitigate security weaknesses in source code. The usage of these types of stimuli content, tasks and user interactive control flow of events did increase the complexity of analysis and visualizations. However, this approach permitted the ability to offer the participants a natural software development and secure code development environment for secure coding.

We start with the simplest stimuli content, tasks and analysis approaches and then progress to the approaches used in the dynamic non-linear interactive tasks. First, we present a basic analysis of reading documentation and how to compare multiple participants. Then, our analysis progresses to the programming content and the visualizations at a more detailed level of the source code. Afterwards, we present our analysis of user interactive content and split screen
applications along with a comparison of several secure coders. Finally, we discuss our visualization techniques for graphically viewing changes in a secure coder’s pupil diameter.

6.2 Analyzing Static Stimuli with Simple Areas of Interest (AOIs)

Our initial methodology focused on analyzing the learning content that is presented to the secure coders. This learning content was presented as stimuli content using Tobii Studio PDF element feature and we did not allow any scrolling. This simplified the processing analysis, especially when creating multiple AOIs. AOIs were created for each page of learning content stimuli presented to the participants. An example is shown in Figure 6.1 for the learning content associated with CWE-311 and CWE-798 as it was presented to each participant. Figure 6.2 contains the same stimuli but shows a single AOI overlay for analysis of participant eye gazes. Stimuli were either examined with a single AOIs overlay as shown in Figure 6.2 or multiple AOIs overlays.
CWE-311 Missing Encryption of Sensitive Data. We need to store or transmit sensitive or critical information in some cases. For example, in ShareAlbum, we need to store users' name, email, information and password in a database. If this information is in plaintext, user's login information will be exposed if their computer is compromised by an attacker. So that we need to use encrypt algorithm to encrypt this information. The encryption algorithms we suggest using are scrypt, bcrypt and PBKDF2.

CWE-798, use hard-code credentials may be convenient to our coding. However, this kind of credentials will allow an attacker to bypass the authentication that has been configured by software administrator, which is hard to detect and mitigate by software administrator[7]. Encrypt these credentials by using an encryption algorithm is essential. The encryption algorithms we suggest using are scrypt, bcrypt and PBKDF2.
CWE-311 Missing Encryption of Sensitive Data. We need to store or transmit sensitive or critical information in some cases. For example, in ShareAlbum, we need to store users’ name, email, information and password in a database. If this information is in plaintext, user’s login information will be exposes if their computer is compromised by an attacker. So that we need to use encrypt algorithm to encrypt this information. The encryption algorithms we suggest using are scrypt, bcrypt and PBKDF2.

CWE-798, use hard-code credentials may be convenient to our coding. However, this kind of credentials will allow an attacker to bypass the authentication that has been configured by software administrator, which is hard to detect and mitigate by software administrator[7]. Encrypt these credentials by using an encryption algorithm is essential. The encryption algorithms we suggest using are scrypt, bcrypt and PBKDF2.

**Figure 6.2:** Example of a Static Stimuli with Overlay for a Single AOI.

We used static stimuli for CWE-434 when coders were asked to answer a multiple-choice question. Figure 6.3 presents the instructions to each participant informing each participant of the specific type of vulnerability and what type of question is going to be asked in the eye tracking stimuli that is presented after this specific slide. In Figure 6.4, we present “Programming Solution Option A” and Figure 6.5 contains “Programming Solution Option B”. The correct programming solution is “Programming Solution Option A” since that program code checks for allowed file extension while programming code in “Programming Solution Option
B" does not have a check for file extension before storing into the database. The last slide of each question presented to each eye tracking participant contained an instruction slide that once the participant moves past that specific stimuli element, then they cannot return to any previous stimuli elements. Figure 6.6 contains this stop slide that is shown in each section to indicate that a participant should review any prior material before moving to the next stimuli element which contains the questionnaire. In Figure 6.7, we present the questionnaire that is presented to each participant following the stop slide shown in Figure 6.6. Tobii Studio collects and stores the data for each participant when they answer multiple choice questionnaires like shown in Figure 6.7.
Next you will be presented with potential coding solutions to the CWE-434 for Unrestricted Upload of a File with Dangerous Types. Examine the source code file to determine which option best addresses the vulnerability. Then you will select which programming solution fixed the CWE-434 vulnerability.

Choices of programming solutions for CWE-434 that would apply:

A. Programming Solution Option A
B. Programming Solution Option B
Programming Solution Option A

Figure 6.4: Static Stimuli Data Content (Page 2 of 4) for CWE-434.
Programming Solution Option B

Figure 6.5: Static Stimuli Data Content (Page 3 of 4) for CWE-434.
PROCEEDING BEYOND THIS SLIDE,

YOU WILL NOT BE ABLE TO RETURN
to the content in this section of the learning module.

Please ensure you have completed all necessary reading information before proceeding to the next section.

Figure 6.6: Static Stimuli Data Content (Page 4 of 4) for CWE-434.
Our Python programming language script iterates over the comma separated files and stores the data in a Pandas DataFrame. The script iterates row by row over the data and determines when a participant switched from one AOI to a different AOI. We down selected to specific AOIs that are specified as input parameters. This allows us to take all of the AOIs for each participant and indicate a specific CWE that should be analyzed. Furthermore, our Python script uses the AOIs and fixation data to generate various plotting visualizations using the Python package Matplotlib to generate Portable Network Graphics (PNG) chart images [182]. Our script includes the ability to use Python Multiprocessing capability to allow each participant data processing to operate in a different Python interpreter process. This significantly improved the processing time as
the script scales out to the number of threads available on a system and allows for the processing of multiple secure coders at once using parallel processing.

6.3 Comparing Static Stimuli for Multiple Participants with Simple Areas of Interest (AOIs)

One chart that we generated is an AOI transition line. This provided the ability to observe objectively the transitions from one AOI to another AOI associated with a specific CWE. An example of four AOI transition line graphs is displayed in Figure 6.8 for two different participants and another two participants presented in Figure 6.9 while each were presented questions to CWE-434. The X-axis contains the start time when the participant moved into a specific AOI. The Y-axis contains the AOI unique name. This plot demonstrates the various transitions that each participant took when reading the stimuli content. It can be determined from this diagram that each participant has a unique reading pattern and visible transitions between the various AOIs.
Figure 6.8: AOI Transition Line Plot for Static Stimuli Learning Content Example 1.
Another plot chart that we generated is a plot of the fixation duration and number of fixations per each AOI over a linear timeline. This provided the
ability to observe not only the transition from an AOI to another AOI but also includes the fixation duration totals and number of fixations in each AOI until a transition to another AOI occurs. This combines the linear nature of eye tracking occurring over a timeline, the transitions associated when moving from one AOI to another AOI as well as an eye tracking metric such as fixation duration or number of fixations.

Prior research has determined that when participants have longer fixation durations then this typically represents deeper cognitive processing than shorter fixation durations [59]. An example is displayed in Figure 6.10 for two participants. The X-axis contains the AOI name in the order in which they occur for that specific secure coder and changes when the coder changed AOI. The X-axis is different per participant as different secure coders visit AOIs in the order that they determined viewing the stimuli content. The first X-axis data point is the starting AOI, and the last X-axis data point is the ending AOI. The Y-axis contains the duration of all fixations in that AOI until a transition occurs for the top portion in Figure 6.10. The bottom portion in Figure 6.10 is for the number of fixations metric. This secure coder had a high number of continuous fixations for Page 1 and Page 2 of the stimuli content associated with CWE-434. The other two pages (Page 3 and Page 4) associated with CWE-434 did not receive as much attention before the secure coder switched to a different page.
Figure 6.10: Fixation Durations and Number of Fixations Plot for Static Stimuli.

We also included the ability to generate data files in the correct format needed to generate the Radial Transition Graph for our eye tracking data as well. We utilized the online graphing capability for Radial Transition Graphs to generate the visualization [183]. Figure 6.11 is an example of using another researcher
Radial Transition Graph to visualize our collected data [106, 111, 160, 183]. This can allow one to visualize that Student identified as SCA.112.F.19 only had a duration of 25.5 seconds in an AOI while Student identified as SCA.104.F.19 had a duration of 1.04 minute in that same AOI.

6.4 Analyzing Static Stimuli with Detailed Areas of Interest (AOIs)

Our initial endeavor focused on creating large single AOIs around the entire stimuli content as shown in Figure 6.12. Creating a single AOI that contains an entire block of software code may not allow as detailed view as needed to determine what the coder really looked at within a block of software code. Therefore, we created new AOIs on the static pages that had software programming code functions. We created several AOIs that group sections of a software function. An example is shown in Figure 6.13 for the questionnaire associated with CWE-434. Our AOIs creation in this more detailed examination of the source code focused on programming logic components that are grouped together to build a programming function. A programming function consists of several key elements including function definition, declaration of variables, programming control statements and programming saving internal variable statements. This analysis technique relied upon the creation of multiple AOIs focused on these more detailed programming statements as shown in Figure 6.13. This is different than our prior examination of creating an entire overarching AOI as shown in Figure 6.12. This allowed a supplemental detailed analysis of specifically what aspects of a programming
Figure 6.11: Comparison of four participants for CWE-79 using the Radial Transition Graph.
function each coder viewed and also the order in which the coder viewed the sections of a programming source code function.

**Figure 6.12**: Example from CWE-434 of creating a Single AOI for Software Code for Static Stimuli.
Figure 6.13: Example from CWE-434 of creating Multiple AOIs for Software Code for Static Stimuli.

In Figure 6.14, our diagram view of the AOIs for a specific secure coder is presented to help show which aspects of a programming function the coder viewed more than other AOIs. We can immediately determine that this specific coder had a greater focus on the AOI associated with the SQL Query execution and the Function Definitions AOI in the first programming option (Option A) compared to other AOIs. This diagram view provides a quick overview of which AOIs each student viewed more than other AOIs; however, it does not demonstrate the
transition or behaviors the coder used when working our secure coding problems. This is a limitation of the fixation matrix that simply summarizes eye tracking statistical results based upon the individually visited stimuli elements or AOIs. No transitions in the stimuli elements or AOIs can be determined from a fixation matrix alone.

Figure 6.14: Static PDF Stimuli Data Analysis for CWE-434 with a detailed AOI.

### 6.5 Comparing Static Stimuli for Multiple Participants with Detailed Areas of Interest (AOIs)

We examined the transitions from each AOI over a timeline in order to examine the techniques that secure coders utilized. Examples of multiple participants are presented in Figure 6.15, Figure 6.16, Figure 6.17, Figure 6.18, Figure 6.19, and finally a participant in Figure 6.20. The X-axis contains the order of when the participant moved into the AOI. The exact start time on the X-axis was removed to improve the readability of the plot. When used in the interactive
program, users can hover over the transition points and a popup will occur with the AOI name and the actual start time. The Y-axis contains the AOI unique name for both programming option A and programming option B.

In Figure 6.15, we determined that the coder viewed several AOIs in programming option A before transitioning to programming option B. The participant also had several transitions back to option A while viewing option B. In Figure 6.16, we can establish that another participant viewed several AOIs in programming option A before going to option B. However, after a few transitions in option B, the participant went back to option A and performed twenty-eight transitions within option A before returning to option B for a second viewing.

In Figure 6.17, we demonstrates that a third coder had several transitions between the function definition and the declaration of variables. This chart also indicates that the third participant mostly focused on one option at a time and made few transitions between the two options. In Figure 6.18, we can determine that the transition pattern of a fourth secure coder appears to transition between the two options more rapidly than our prior participants.

In Figure 6.19, we can ascertain that a fifth participant examined option A for a great amount of time before transitioning to option B. We can also determine that the participant had several transitions between the AOI associated with a programming check to determine if input fields are missing and a check for incorrect file extension. In Figure 6.20, we can determine a sixth participant viewed option A before transitioning over to option B. Then, the coder reviewed both programming options. The data indicate that the sixth participant had sev-
eral transitions between the function definition AOI and the variable declaration
AOI along with the AOI associated with checking for incorrect file extension.

For CWE-434, the programming solution that contained the security flaw
was located in programming option A. This approach to analyzing the AOI tran-
sitions allows researchers to establish a transition pattern and revisit pattern of
participants working the problem. We also examined the fixation duration and
number of fixations for these same participants working the problem for CWE-
434. This view allows for examining which AOIs the participants focused more on
but also when those fixations occurred over a timeline. This is not possible with
the fixation matrix that simply provides an overall summary of fixation count or
fixation duration per each stimuli content or AOI. This demonstrates that some
secure coders have more fixations in the same programming option before switch-
ing to a different programming option while other secure coders have fixations
stretched throughout the programming options when working a secure coding
problem. This type of view demonstrates each participant’s return or revisit to a
previous stimuli content element or AOI showing both the count of revisits as well
as the order of revisits [59]. It can be determined from that each secure coder has
a unique reading pattern and the order of transition between the various AOIs
can be visually analyzed per each participant.

6.6 Analyzing Multiple User Interactive Stimuli Content

The previous analysis methods have focused on static stimuli content that
did not allow any secure coder participating in the eye tracking experiment to
Figure 6.15: AOI Transition Plot for Static Stimuli for CWE-434 with a multiple AOI Example 1.
Figure 6.16: AOI Transition Plot for Static Stimuli for CWE-434 with a multiple AOI Example 2.
Figure 6.17: AOI Transition Plot for Static Stimuli for CWE-434 with a multiple AOI Example 3.
Figure 6.18: AOI Transition Plot for Static Stimuli for CWE-434 with a multiple AOI Example 4.
Figure 6.19: AOI Transition Plot for Static Stimuli for CWE-434 with a multiple AOI Example 5.
Figure 6.20: AOI Transition Plot for Static Stimuli for CWE-434 with a multiple AOI Example 6.
scroll in the text. The inability to scroll restriction allowed for simpler post-
processing of the eye tracking data. This is a common restriction researchers 
have faced with eye tracking research [58, 97]. However, we believe that this 
also restricts the ability of the participant to follow a natural flow of problem 
solving that is typical for reading and writing software program code [76]. There-
fore, our experiments also included dynamic interactive stimuli in both of our 
learning modules. The post-processing data analysis is more complicated and 
time-consuming for dynamic user interactive stimuli content compared to static 
stimuli; however, our belief is that it is a necessary aspect of eye tracking and 
software development.

The eye tracking software application that we utilized allows for re-watching 
of the dynamic interactive stimuli as a video recording for each secure coder. Each 
secure coder’s raw data points are saved with a video-recorded file that contains 
the stimuli content and, furthermore, metadata for the eye gazes in relationship 
to the video frames. We used the Tobii Studio application to create, move, trans-
form, and disable AOIs as needed for each coder for the dynamic content. This 
was accomplished by advancing the video recording at an interval of 0.2 seconds 
for each coder and annotating AOIs for each timestep. Depending upon where 
the participant scrolled and which source files or websites the coder viewed, then 
that altered which AOIs were active and the location of the AOIs. The AOIs for 
the dynamic interactive content were then exported from Tobii Studio for further 
data analysis. For our dynamic stimuli content, particularly the source code con-
taining multiple code files with several programming functions per each file, our
AOI creation was focused on the function level due to the considerable number of AOIs that was required. This entailed the creation of as few as 10 AOIs to as high as 120 AOIs depending upon the student and the specific CWE. The dynamic interactive stimuli significantly increased the number of AOIs that was generated to cover all participants and revealed some shortcomings of our initial analysis approaches when using more limited static stimuli content.

Our first investigation was with the fixation matrix and the transition plot similar to our examination of the static stimuli content. As shown in Figure 6.21, it can be visually verified that the secure coder focused significantly more on the AOI associated with the source code function for getting notifications and another AOI associated with the source code function for sending notifications. The security vulnerability for CWE-79 did occur in the AOI for sending notification. We can also see when and how often the specific secure coder utilized the RIPS static code analysis tool as well as the Google search engine to look for additional information. As can be visually established from Figure 6.21, as the number of AOIs increases, then the analysis of the transitions becomes more difficult to visualize and develop meaningful results.
Figure 6.21: Example of Fixation Matrix and Transition Plot for CWE-79.
6.7 Comparing User Interactive Stimuli of Multiple Participants

Our swimlane approach was developed to visually determine the transitions between various high-level components and specific AOIs. This allowed us to group AOIs together into categories that have common source files or usages. For example, we grouped AOIs associated with instructions together and AOIs associated with using the static code analysis tool. Then, AOIs for each source code function contained in the same source file were also grouped together. The swimlane diagram approach allows us to clearly recognize the differences in the multiple stimuli and applications that each secure coder utilized when working the secure coding learning modules. The transitions between the multiple stimuli contents can also be easily visualized with swimlane diagrams. This diagramming technique allows for the comparison of multiple participants and visually ascertain their behavior. The width of each swimlane column is not an indication of time spent in that grouping but is instead the smallest size possible based on the text contained in each swimlane column.

PlantUML allowed for the generation of swimlane diagrams using an automated script processing approach from text strings [184]. This removed the need to create the swim lane diagrams manually for each secure coder. The width of each column is strictly dependent on its content and has no meaning to the time the secure coder spent in that grouping. An example of one participant can be viewed in Figure 6.22, and an example of a second participant can be visualized in Figure 6.23. The swimlane diagram provides a visualization method of grouping
AOIs together to help visualize when the coders make a more defined transition between various components such as source code, tools, instructions, and web searching. However, a limitation of the swimlane diagram is that often, all of the swimlanes cannot fit on a typical single sheet of paper. The PlantUML web server provides the capability of generating Scalable Vector Graphics (SVG) of the data to allow for better viewing of swimlanes that contain significant transitions between stimuli components [184].
Figure 6.22: PlantUML Swimlane Representation Example 1.
Figure 6.23: PlantUML Swimlane Representation Example 2.

A transition matrix can provide an analyst with a high-level overview summary of the transitions between stimuli content or AOIs. Our transition matrix provides a legend for a shorter AOI unique identifier that is matched to a specific AOI name. This provides for easier readability in the matrix as the number of AOIs increased. The rows in the matrix represent the current AOI and the columns represent the next AOI that was transitioned to after leaving the current AOI. We provided a matrix diagram representing the transition count as well as the transition ratio for each secure coder. In Figure 6.24, we can visu-
ally determine for one secure developer that for the AOI Web RIPS Tool Home Page the only transition is to the Instruction AOI. Similarly, we can determine that the same developer’s only transitions from the Web Search Engine Google AOI to another AOI is the AOI associated with the send notification to another user function located in the notify file source code. The cyber vulnerability for CWE-79 existed inside of the send notification to another user function. A transition matrix provides an overall summary of transitions that allows for comparing multiple participants.
The transition matrix approach provides the ability to view the transitions per each stimuli element or AOI and the swimlane diagrams also provide that ability but allow for grouping of specific AOIs together. We wanted to explore additional diagramming methods to visualize the data with a different diagram but maintain the grouping components selected in the swimlane diagram. Therefore, we created a state transition diagram that uses the PlantUML state
diagramming feature. Directly using all of the transitions from all of the AOIs may result in a considerable number of arrows in a state diagram and thus become unreadable. This can be shown in Figure 6.25. One solution is to reduce the state diagram lines to the major grouping of AOIs. For example, this could be used to match up with the column headers in our swimlane diagram. As shown in Figure 6.26, this improves the readability for the same secure coder that is shown in Figure 6.25. It is our determination that state diagrams have limitations as the number of AOIs increase as well as if participants have several transitions between stimuli content or AOIs.
Figure 6.25: Complex PlantUML State Representation.
Figure 6.26: Simple PlantUML State Representation.

6.8 Visualization of our Eye Pupil Diameter Adjustments

Prior research has discovered correlations between the changes in our pupil diameter and mental effort or cognitive load when working a problem. Researchers have observed that an increase in pupil dilation occurs when the task complexity increases [110, 127, 129, 130, 185]. Researchers have examined changes in the pupil diameter for air traffic controllers [186] and participants contributing in a driving simulator [109]. However, to our knowledge, research in pupil changes in the area of secure coding hands-on training modules has not been investigated. Our
research examined various techniques to visualize changes in the pupil diameter size for when our secure coders worked our secure coding exercises.

Our custom Python program parsed the eye diameter that was recorded. This allowed for the generation of various visualization plots of the pupil data. Our initial attempt involved examining the changes in pupil diameter for each AOI. This allowed us to generate a boxplot per each AOI. An example of one secure coder for their left pupil and right pupil is shown in Figure 6.27 and Figure 6.28. A second example for another secure coder is shown in Figure 6.29 and Figure 6.30. The AOI visited is shown on the X-axis with the pupil diameter shown on the Y-axis. The number of fixations is then overlayed as a second Y-axis.

In our analysis of several secure coders, it was observed that some participants had higher pupil diameter sizes in some AOIs than compared to other AOIs that the secure coder visited. For example, in Figure 6.27, is for a secure coder’s left eye and Figure 6.28 is for the same secure coder’s right eye. Figure 6.27 and 6.28 indicates that several of the peaks in pupil size occurred in the Instruction AOI, the source code function for Getting User Notifications AOI and in the source code function for Sending Notifications to Another User AOI. In Figure 6.29 and Figure 6.30, a second secure coder also had peaks in their pupil diameter for the same AOIs as well as when the second secure coder used the Google Search Engine. However, a concern with this visualization approach is that when a secure coder enters an AOI that has many fixations, then this technique could potentially miss certain peaks and dips in the pupil diameter since
Figure 6.27: Boxplot of Eye Pupil for One Student with Number of Fixations Example 1 (Left Pupil).
Figure 6.28: Boxplot of Eye Pupil for One Student with Number of Fixations Example 1 (Right Pupil).
Figure 6.29: Boxplot of Eye Pupil for One Student with Number of Fixations Example 2 (Left Pupil).
Figure 6.30: Boxplot of Eye Pupil for One Student with Number of Fixations Example 2 (Right Pupil).
this approach only resets when the secure coders transition from one AOI to another AOI. Therefore, our analysis was expanded to examine the pupil diameter at a constant rate of time regardless of changing to a different AOI or remaining in the same AOI.

The examination of the pupil diameter for secure coding is a challenging problem due to the potential to miss sudden adjustments in the pupil diameter when viewing eye tracking stimuli that span multiple AOIs. Our research involved visual techniques to examine the changes in pupil diameters. Future research is needed to determine the correlation between the changes in pupil diameter and participants solving secure coding problems. Our focus in this manuscript is focused on what type of visualization techniques could potentially benefit the analysis of eye pupil changes over non-linear dynamic stimuli content.

Another technique for examining the left and right pupil data could potentially provide insight when participants do not make frequent transitions between AOIs. This method focused on analyzing the eye tracking data at specific intervals of time. Our algorithm utilized a timestamp in milliseconds that sampled the left and right pupil diameters at a fixed time interval. There is a minimal time spacing that must occur before a new pupil diameter sample is taken from the full recorded dataset. If the sample time window occurs in an AOI that is not of interest, then the data sample is not taken until the participant enters into any AOI that is of interest. Since we analyzed specific AOIs that are based upon specific CWEs, our pupil sampling algorithm had to manage the occurrences of secure coders viewing AOIs of other CWEs or unrelated stimuli content. This is
to eliminate the potential of unwanted AOIs being used in the data sampling for pupil data. Therefore, the smallest interval in time when pupil data is sampled from the full dataset is at least the size of the selected time step size parameter. Our experiments have examined time step parameter sizes of 250 milliseconds, 500 milliseconds, 750 milliseconds, and 1000 milliseconds. In Figure 6.31 and Figure 6.32, we present the left and right pupil measurements for a software developer at an interval of 250 milliseconds for CWE-79. In Figure 6.33 and Figure 6.34, we present another software developer at the same interval for CWE-79.

![Figure 6.31: Eye Pupil Data for Developers at 250 Millisecond Interval Example 1 (Left Pupil).](image-url)
Figure 6.32: Eye Pupil Data for Developers at 250 Millisecond Interval Example 1 (Right Pupil).

Figure 6.33: Eye Pupil Data for Developers at 250 Millisecond Interval Example 2 (Left Pupil).
Another approach examined the changes in secure coders’ pupil diameter by utilizing a Simple Moving Average (SMA) and calculating the mean of the pupil diameter for specific time-periods. In our research, we examined small time-periods such as counting as little as two fixations to as high as counting thirty fixations for the SMA algorithm. An example of a software developer left pupil diameter over a linear timeline with the SMA algorithm and a count of five fixation points is shown in Figure 6.35. This specific developer had a steady increase in their left pupil when they examined the software programming function associated with deleting an album from a database. For this specific hands-on programming exercise, a software vulnerability existed in the Delete Album software function. The software vulnerability would allow a security attacker to delete an album.
even if the security attacker did not create the album. This software developer had multiple reductions in their pupil size after the steady increase levels off. This reduction in the pupil diameter size may indicate a reduction in the cognitive load and processing of the secure coder as a potential indication that the developer believes they have correctly found the software vulnerability [130].

6.9 Other Approaches

Eye tracking researchers created the ability to view the transition patterns of several participants as a transition tree or as an icicle plot. This allows the researchers to examine the transition patterns as well as the frequencies of those transitions. The researchers designed their analysis method to work with stimuli that are not bound to the same timeframe for each participant [162]. This provides a closer representation of the type of stimuli used to investigate the behaviors of software developers since software developers need to have the option to interact and control the flow of eye tracking stimuli events just as they would naturally for software coding tasks. A limitation of this method is that a publicly available resource for generating the diagrams with an independent dataset was not published.

6.10 Summary of Visualization Methods

During our research, we explored published literature to determine existing methods and techniques that eye tracking researchers have explored for analyzing eye tracking data. Researchers have created tools for the visualization of eye
Figure 6.35: Simple Moving Average Mean of Pupil Diameter for a Developer.
fixations using heat maps, gaze plots and clustering over geographic map views [158]. Researchers often explore the usage of a fixation matrix [83] or transition matrix to provide an overall summary of participant behaviors [112, 160]. Some studies focused on analyzing stimuli that is not bound to the same timeframe for each participant [162]. Scarf plots allow for visualization of the AOIs and the associated transitions displayed over a linear timeline [59, 160, 162]. Radial transition graphs allow for the transition from one AOI to another AOI to be symbolized by arc lines in a graph [160]. Using the radial transition graphs, researchers have discovered patterns of linear code reading as well as patterns representing the execution of source code [106, 111]. Researchers have investigated the usage of viewing multiple participant scan paths next to each other and color-coding specific AOIs that are of particular interest [106]. In Table 6.1 and Table 6.2, we summarize the major eye tracking visualization methods that are currently published by researchers.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Benefits</th>
<th>Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heat Map</td>
<td>Color palette allows for the areas with more fixations to be distinguish from the areas with less fixations.</td>
<td>Quickly determine focus area over a stimuli element or AOI.</td>
<td>Does not work well with dynamic video stimuli. Limitations when used for multiple participants.</td>
</tr>
<tr>
<td>Scan Path or Gaze Path</td>
<td>Plot eye movements from one fixation to the next fixation forming a path.</td>
<td>Helps to form a time-based aspect to the graphics of viewed stimuli.</td>
<td>Difficult to examine for multiple participants or with a long session for a single participant.</td>
</tr>
<tr>
<td>Clustering</td>
<td>Different clusters are created when fixations are spatial separated based upon an analyst supplied parameter.</td>
<td>Works well to give an overview of several participants.</td>
<td>Does not work very well for dynamic video stimuli.</td>
</tr>
<tr>
<td>Transition Matrix</td>
<td>A table indicating transitions to other stimuli elements/AOIs usually presented as a count of transitions to and from stimuli elements/AOIs or a percentage of the total.</td>
<td>Provides an overall summary of which stimuli/AOI had higher transitions from other stimuli elements/AOIs.</td>
<td>Does not show the order of transitions between the various stimuli elements/AOIs. Limits when the number of stimuli/AOIs increases.</td>
</tr>
</tbody>
</table>
Table 6.2: Eye Tracking Visualization Methods Set 2.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Benefits</th>
<th>Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixation Matrix</td>
<td>A table of fixations usually presented as a count or as a percentage per stimuli/ÀOI.</td>
<td>An overall summary of fixation count or percentage.</td>
<td>Does not indicate the chronological order in eye gaze transitions.</td>
</tr>
<tr>
<td>Radial Transition Graph</td>
<td>A circular graph representation of the visited stimuli/ÀOI. The transition from one stimuli/ÀOI to another stimuli/ÀOI is symbolized by arc lines.</td>
<td>Visualize the transitions to and from stimuli/ÀOI along with the dwell times per each stimuli/ÀOI.</td>
<td>As number of stimuli elements/ÀOIs increases and/or the number of transitions increases, then the graph is difficult to understand.</td>
</tr>
<tr>
<td>Scarf Plot</td>
<td>Stimuli elements/ÀOI are marked on a timeline per each participant and can be transition-based or time-based. A color-code is often used for unique identification.</td>
<td>Useful for analyzing the stimuli/ÀOI sequences or dwell times in the order that they occur.</td>
<td>Difficult to determine differences in patterns as the number of participants increase. Difficult to understand with different temporal ordering of stimuli/ÀOI.</td>
</tr>
<tr>
<td>Parallel Scan Path</td>
<td>Stimuli elements/ÀOIs are plotted on the X-axis with time on the Y-axis.</td>
<td>Ability to see the transitions over stimuli/ÀOI for a small number of participants.</td>
<td>Difficult to analyze as the number of stimuli/ÀOI increases or considerable number of transitions analyzed.</td>
</tr>
</tbody>
</table>
Several existing and common visualization techniques work well for static stimuli content or with stimuli that have a small number of objects to distinctly observe. Eye tracking data for secure coding often includes interactive content that requires scrolling multiple web pages and multiple source code files as well as participants switching back and forth between multiple files. User interactive content complicates the data analysis phase; however, we believe that for secure coding activities to be realistic, we must allow each secure coder to control the flow of events as each coder works to discover and mitigate security vulnerabilities. We created a custom analysis program to generate additional eye tracking visualization methods that generate new types of images for visualization of eye gaze.

In our research, we evaluated our own custom visualization techniques that can be applied to help facilitate the analysis of behaviors and strategies of software developers. Our focus was to examine visualization methods that could provide meaningful insight as the number of AOIs continue increasing for non-linear dynamic stimuli content that allows for participant user interaction. To our knowledge, swimlane diagrams have not been utilized to study the eye gaze behaviors of software developers. Swimlane diagrams allow categorizing AOIs with a common relationship. For example, in software coding, each source code file contains several function or methods of software source code. Swimlanes allow us to group the source code functions into individual columns based upon the source code file the functions or methods reside inside. This can also be extended to work with the algorithm and source code statements that are inside of a source
code function or if a source code file contains several class objects definitions. A swimlane diagram provides the ability to group into individual lanes for organizing several AOIs. This grouping organization helps when visualizing the transitions between the various AOIs as the number of AOIs continue increasing. SVG for our generation of swimlane diagrams helped to improve the readability of developers with a considerable number of transitions and visited AOIs.

An example of a swimlane diagram is shown in Figure 6.36. This example contains three major groupings of multiple AOIs. Each major group is represented as columns and each AOI is represented in a tan square. The width of each swimlane column is strictly based upon the content of each column and has no relationship to the time the secure coder viewed content for that grouping. In this example, we can visually determine that a developer moved from the instructions to view a website before examining the source code. The swimlane allows for the grouping of several AOIs as can be seen in the notify helper file column with several AOIs representing programming function logic.
We explored other graphical diagramming techniques to study the transitions that software developers use when working on software coding exercises. We investigated the ability to use state diagrams as another method to visualize the transitions. Similar to the swimlane diagram approach, there is a need to group
based on the source code functions and files, source code analysis tools, websites, and instruction documentation. A state diagram allows researchers to examine the transitions within several stimuli content as well as several AOIs. A state diagram can be designed to indicate the amount of time that developers spend in an AOI as well as the number of transitions between the various AOIs. A limitation of the transition state diagrams is the inability to handle a significant number of transitions. We explored the ability to group several AOIs into a high-level AOI to help improve the readability of the graphic. This is especially critical when the transition state diagram is anticipated to be used as printed content.

Figure 6.37 shows a state transition diagram for a developer reading instruction material, a website, and a sole source code file. We can visually determine from this example that the developer did not visit the source code file with the software programming flaw. We can also visually determine that the developer never utilized our selected software scanning tool. This diagramming technique provided a quick overview of the major elements that a developer viewed as well as a count of the transitions that occur within the major groups.
Figure 6.37: State Transition Diagram for Single Software Developer Reading Documentation, utilizing a Web-browser and Reading a Source Code file with multiple coding functions when viewed using Non-Linear Dynamic Stimuli Content.

An additional aspect of eye tracking research is the ability to objectively collect the sizes of participants’ left and right pupils’ diameter during the recording session. The literature is currently lacking in visualization methods to transform raw pupil data into meaningful insight for software development activities. We explored the benefits and limitations of examining the eye gaze pupil data for software developers for software coding and testing of code.

One method of visualization is using boxplots to plot the minimum and maximum as well as the quantiles for each AOI. Our research explored plotting a boxplot for each AOI over a timeline of transitions between AOIs. A limitation of this approach is that we could potentially miss critical peaks and reductions in pupil diameter size if developers are focused in an AOI for an extended period of time without changing to another AOI. Therefore, we explored the creation of
an analysis visualization plot that directly selects pupil diameter size at specific time intervals and counts of fixations. This may result in plotting the same AOI for several occurrences over a linear timeline. This helps show the pupil diameter size at a constant rate over a linear time. In Figure 6.38, is an example of using a SMA by calculating the mean pupil diameter for a software developer for a specific number of fixations. We examined a threshold as low as two fixations to as high as thirty fixations. This approach helps to minimize the loss of information if developers remain in a single AOI for a significant amount of time by taking the mean measurement at specific intervals. In order to help viewers clearly visualize the various AOIs we utilized a color mapping scheme that is similar to scarf plots. An example is shown in Figure 6.38 for a single software developer examining source code for software flaws.
Researchers have examined the relationships between software development and software maintenance tasks in relation to cognitive load for several decades [35, 43, 187]. Studying the changes in pupil diameter as software developers work software learning and coding exercises may allow us to understand the cognitive load of software developers’ [40]. Prior research has determined that p-
Participants have a correlation between cognitive effort and eye pupil size [110, 185]. Research has found that increases in pupil size can indicate an increase in cognitive load [108, 149]. However, these findings are mostly associated with reading stimuli and simple mathematical tasks. Finding and fixing software coding issues consisting of multiple types of stimuli that are more complex than left-to-right natural language text stimulus and often take considerably more time to review [55, 98]. One study was conducted examining the pupil diameter of operators during various load conditions for naval vessel combat stations that were utilized to identify and track threats as well as potentially engage enemy threats with weapons. Their findings suggest that operators with an unloaded scenario had lower pupil diameter size when compared to participants with a normal scenario especially for the participants with an overloaded combat scenario [45]. Other researchers have found similar results with changes in pupil sizes and cognitive load for driving simulations [109]. Still, the current literature is lacking in understanding pupil diameter changes for software coding activities in the SDLC [98, 113].

6.11 Framework Summary

An overall summary of the visualization and statistical analysis techniques is presented in Table 3.1 and Table 3.2 for several categories in our framework that was shown in Figure 3.1 and Figure 3.2. In Table 3.1 and Table 3.2, we provide an overview of several statistical or visualization techniques and relate the techniques in regard to the classification of techniques that we have developed. Table 3.1
and Table 3.2 focus on the graphical visualization techniques that can be used for analyzing participants’ eye gazes by providing a check-mark if the graphical technique is beneficial for the defined categories. We propose a framework that helps researchers design an eye tracking experiment and analyze the recorded eye gazes using visualization diagrams. We also present a summary of existing visualization methods for eye tracking and references to articles containing visual examples of the visualization techniques.

Our key contributions include the following:

- Classification of the goals, objectives, participant tasks, and visualization techniques in distinct stages of the SDLC for eye tracking

- A decision matrix for mapping objectives/tasks in the SDLC to specific aspects of eye tracking design, analysis, and comparison

- Guide on the type of software tasks and eye tracking stimuli to present to participants

- Comparison of the visualization techniques based on participants, stimuli content, and software development task

Researchers need to understand when to apply specific visualization techniques in order to find the patterns, behaviors, and strategies that participants utilized when viewing stimuli content. Various visualization techniques are useful based on the tasks and stimuli content utilized in eye tracking. Applying the correct analysis method will allow researchers to understand how participants
approach problem solving and how participants utilize tools. We believe that our eye tracking technologies offer the ability to objectively observe and measure participants. Our research has examined the limitations of existing approaches and explored new visualization techniques to help overcome the limitations. One of our goals was to create the ability to capture students’ behaviors as they used multiple types of stimuli content when learning, finding, and mitigating security coding weaknesses. Our research has enabled the ability to analyze participants’ behaviors when they use multiple applications concurrently. Furthermore, our research allows for investigating the techniques that our participants utilized to find and mitigate security bugs in software codes.
Chapter 7. Eye Tracking Metrics Analysis of Secure Coder Behaviors

Our eye tracking study allowed us to objectively study and gain insight in order to understand and improve students’ learning behavior. Our analysis indicates that there is a distinction in the learning phase for participants who answered correctly compared to participants who did not provide the correct mitigation strategy. Specifically, our research indicates the most effective and efficient way to learn secure coding is to fully understand coding errors before working on the source code.

7.1 Secure Coders’ Programming Assignment Results

Our Manual Code Analysis Learning Module as well as our Static Code Analysis Tool Learning Module that utilizes the static code analysis tool RIPS had three overall phases. The first phase was the presentation of critical cybersecurity flaws that can exist in source code as determined by the CWE repository and NVD database [2]. This included a brief explanation of how the vulnerability could exist in our photo and video sharing web-based application known as ShareAlbum. The second phase had either multiple-choice programming options or true/false questions that each participant was required to answer after
reviewing the static source code presented in Tobii Studio. The multiple-choice questions allowed each participant to select what they believed would best mitigate a security vulnerability. The final phase had full source code files presented in the Notepad++ text editor and each participant was expected to directly modify. Each participant was asked the same security flaw question; however, no participant was ever allowed to see the response from any other participant.

The verification of each participant’s response depended upon the type of stimuli that was utilized. For the multiple-choice and true/false questions, the verification of each participant’s response was immediate and decisive using Tobii Studio questionnaire stimuli. The programming problems presented in the study that could not be automatically graded with Tobii Studio were manually analyzed afterward by manual code review. The programming problems were more complex than the true/false or multiple-choice questions. Our review process of the participants’ programming problems involved examining the source code that each participant modified and deciding if new source code had been written, if pseudocode was provided by the participant, or if just a comment was left in the source code explaining the location and mitigation strategy that should be utilized. Each response was determined to be correct in fully mitigating the vulnerability, partially mitigating the vulnerability or unsuccessful in determining a mitigation technique for the vulnerability. The programming problems were designed to determine if a participant could determine the correct location in the source code files where a vulnerability existed and then, if possible, could develop a mitigation plan to reduce or eliminate the vulnerability. Since not all partic-
participants had domain knowledge in the HTML and PHP programming languages, participants were allowed the option to provide just pseudocode or a comment if they could not develop the actual source code to mitigate the vulnerability. Each participant was encouraged to attempt to write the source code if possible and only as an alternative option to write pseudocode or comments. The benefit of at least allowing participants to provide some feedback is the ability to examine if the participant found the location in the source code where the software security flaws exist. This is useful even if they can’t develop the source code to resolve the issue. No participant was ever allowed to see the answers of any other participant.

The following presents the findings from our Manual Code Analysis Learning Module which consisted of a manual code review of the photo and video sharing web-based application. The cybersecurity coding flaw associated with CWE-311 (Figure 7.1) and CWE-434 (Figure 7.2) was presented as multiple-choice questions. For CWE-311, the correct solution was Programming Solution C. As shown in Figure 7.2, no student answered Solution A; 5 answered Solution B; and 24 answered Solution C. For CWE-434, 28 answered Solution A; and 1 student answered Solution B. The correct solution for CWE-434 was Programming Solution A. Overall, the participants performed well in picking the correct choice that resolved the security flaw.
Figure 7.1: Manual Code Analysis Learning Module (CWE-311) Participant Questionnaire Results.

Figure 7.2: Manual Code Analysis Learning Module (CWE-434) Participant Questionnaire Results.
The two typed programming problems addressing CWE-862 and CWE-22 required participants to write code. These were more difficult than multiple-choice. This increased complexity enabled us to determine if they successfully mitigated the exposed vulnerability. The participants were encouraged to write actual code. If they were not able to, they might provide written pseudocode or written comment. The participants were graded on if they:

- fully corrected the weakness,
- partially corrected the weakness,
- did not correct the weakness.

For the coding flaw associated with CWE-862, as shown in Figure 7.3, the participants overall were able to correctly provide source code that mitigated the vulnerability. For the CWE-22 vulnerability, as shown in Figure 7.4, only 14 out of the 29 participants correctly determined the flaw location and were able to mitigate the coding flaw.
Figure 7.3: Manual Code Analysis Learning Module (CWE-862) Participant Questionnaire Results.

Figure 7.4: Manual Code Analysis Learning Module (CWE-22) Participant Questionnaire Results.
The Static Code Analysis Tool Learning Module results are presented in the following images. In this learning module, we presented two true/false questions associated with CWE-443 (Figure 7.5) and CWE-73 (Figure 7.6). For these two questions, each participant was asked to answer:

- True, if the vulnerability existed in the source code as identified by the RIPS static code analysis tool
- False, if the vulnerability did not exist in the source that was identified by the RIPS tool as a potential security flaw

For the coding problem in the Static Code Analysis Tool Learning Module, we focused on CWE-79. The result from our participants is shown in Figure 7.7. Overall, most participants got this programming problem partially correct; very few completely mitigated the vulnerability or completely incorrectly answered the problem.
Figure 7.5: Static Code Analysis Tool Learning Module (CWE-443) Participant Questionnaire Results.

Figure 7.6: Static Code Analysis Tool Learning Module (CWE-73) Participant Questionnaire Results.
Our verification of what solutions each participant provided was not directly performed using eye tracking technologies for every question that we presented in the eye tracking study. Some of the questions, like multiple choice and true/false, were presented to each participant in the Tobii Studio questionnaire stimuli and the results were collected by Tobii Studio. However, the programming problems required manual code analysis review by researchers and expert coders and documentation of the participant response.

7.2 Comparing Eye Movements in Secure Coding

We present our analysis of the fixations via graphical visualization techniques, specifically heatmaps. The longer a participant looked at an area, the redder the color in the heatmap. We present two groups of participants: all 5

![Figure 7.7: Static Code Analysis Tool Learning Module (CWE-79) Participant Questionnaire Results.](image)
participants who did not answer correctly (see Figure 7.8), and 5 participants who did answer correctly (see Figure 7.9). We randomly selected 5 out of the remaining 24 participants. It should be directly stated that it is not a reasonable comparison to generate heatmaps with all 24 participants who did answer correctly because there are significantly more data points, and the heat maps would not allow for meaningful conclusions.

Figure 7.8: Learning Content for CWE-311 from Five Students that Answered Incorrectly.
Figure 7.10 and Figure 7.11 represent the heatmaps while participants were reading the Manual Code Analysis Learning Module content related to CWE-311. It can be visually observed that there were fewer fixations from the group of participants that did not answer correctly when compared to the group of participants that did answer correctly. This would indicate that the participants who answered incorrectly did not spend as much time learning about the review material related to CWE-311. For all participants, the time spent on the learning portion was about a quarter of the total time on this task. Thus, it is costly if one does not spend enough time learning.

The next set of heatmaps represents Programming Solution Option C for CWE-311 for the same two groups. Figure 7.10 is from the 5 participants who did not correctly pick the solution. Figure 7.11 is of the 5 participants who did correctly for CWE-311. It would not be an equal comparison of Figure 7.10 to

![CWE-311 Missing Encryption of Sensitive Data. We need to store or transmit sensitive or critical information in some cases. For example, in ShareAlbum, we need to store users' names, email information and password in a database. If this information is in plaintext, user's login information will be exposed if their computer is compromised by an attacker. So that we need to use encrypt algorithm to encrypt this information. The encryption algorithms we suggest using are encrypt, bcrypt and PBKDF2.](image)
Figure 7.11 if we show the data of all 24 participants. The key solution here is the usage of Password.Hash with Password.Bcrypt to solve the problem of missing encryption of sensitive data. This visualization appears to indicate that the 5 participants spent more time on the code but still did not get the correct answer. These visualizations appear to indicate the participants who did not answer the question correctly glanced over the learning content related to the CWE-311 problem. Whereas the participants that did answer the CWE-311 problem correctly, appeared to have more fixations when studying the learning content as shown in Figure 7.11. The metrics provided in the next section will elaborate on how much time participants spend examining the source code solution for option C.

![Programming Solution Option C](image)

**Figure 7.10:** Programming Solution Option C for CWE-311 from Five Students that Answered Incorrectly.
7.3 Statistical Analysis

We examined dozens of statistical metrics for our collected data. In addition, each metric is generated with the mean, maximum, minimum, sum, median, count and standard deviation. The statistical metrics are based upon the AOIs that are generated or drawn by the researcher [58, 94].

A few of the potential metrics for our analysis include [58, 84]:

1. First Fixation Duration [seconds]

2. Fixation Duration [seconds]

3. Total Fixation Duration [seconds]
4. Fixation Count [count]

5. Visit Duration [seconds]

6. Total Visit Duration [seconds]

7. Visit Count [count]

We selected the Total Fixation Duration and the Count of Fixations as well as the Total Visit Duration because they best represented participants’ behavior for the Manual Code Analysis Learning Module. AOI was generated for each page displayed to the participants that worked the Manual Code Analysis Learning Module. For a comparison to the previous figures mentioned, two additional AOIs were generated in direct relationship to the learning content for CWE-311 as well as the Programming Solution Option A and B and C. All 29 participants from the Manual Code Analysis Learning Module are included in the statistical results below as our results are provided using the mean for the selected metrics. Programming Solution Option C was the correct answer for the multiple-choice response to CWE-311. No student selected Option A for an answer.

The mean for the total fixation duration is shown in Figure 7.12 for CWE-311. Figure 7.12 illustrates that for the 5 participants that incorrectly answered, those participants had a lower mean total fixation duration for reviewing the content related to learning about CWE-311. The 24 participants who worked the Manual Code Analysis Learning Module that did answer CWE-311 correctly had a higher mean total fixation duration for reviewing the content. The results indicate that all 29 participants who worked the Manual Code Analysis Learning
Module had a low total fixation duration for Option A. The participants that answered Option C for CWE-311 had several more seconds of fixation duration when reviewing that Option C solution. A similar conclusion can be visualized with the data for the mean fixation count as shown in Figure 7.13. The average for participants who correctly answered had 26 more fixations for the learning content than participants who got the question wrong. In addition, the mean total visit duration (see Figure 7.14) indicates that participants who spent more time analyzing the review material were able to correctly identify the correct solution.

![CWE 311 Total Fixation Duration - Mean](image)

**Figure 7.12:** Total Mean Fixation Duration for CWE-311 AOIs.
Figure 7.13: Mean Fixation Count for CWE-311 AOIs.

Figure 7.14: Mean Total Visit Duration for CWE-311 AOIs.
We conducted an independent two-sample one-tailed T-Test to compare participants who answered correctly to the participants who did not answer correctly for CWE-311 that was in the Manual Code Analysis Learning Module. Our particular interest was the learning content AOI associated with CWE-311. We used a significance level of 0.05 and assumed that the variance was not equal. For the learning content AOI, there was a significant difference for the 24 participants that answered correctly compared to the 5 participants that did not answer correctly, as shown in Table 7.1.
For total fixation duration, fixation count, and total visit duration, the participants that answered correctly compared to the participants that did not answer correctly had a two-sample t-test as shown in Table 7.1. Therefore, we reject our null hypothesis that no difference in the mean exists and conclude that participants who answered the question correctly spent more time and had a higher fixation count on the learning content than participants who answered incorrectly. The t-test for the programming options did not reveal a significant
difference between the two groups of participants. This can be visualized in the previous charts. Educators and students should emphasize that understanding the coding errors before working on the code is critical for success and efficiency.
Chapter 8. Insights into Secure Coder Behaviors

Our study seeks to gain a deeper understanding of developers’ capabilities in writing secure code by examining their behaviors. Our objectives are to identify patterns, strengths, and weaknesses in their secure coding skills by observing and analyzing their coding practices. The ultimate goal is to contribute towards findings that are valuable to the field of behavioral analysis of secure coders, which would lead to improvements in training modules for secure software development and improving the coding behaviors and skills of secure coders. This chapter focuses on interpreting the data in the context of secure coding objectives and drawing insights from our findings. Our initial findings suggest that visualizations and analysis of eye gaze data are critical aspects of understanding the behaviors of secure coders.

8.1 Secure Coders’ Behaviors Analysis

By utilizing eye tracking technologies to monitor where secure coders focus their attention during the coding process, we can uncover valuable insights into the cognitive processes involved in their coding exercises. We aim to gain insights into how participants visually process and attend to the presented content. Acquiring this understanding will enable us to enhance the educational modules we employ
for teaching secure coders. Additionally, we can enhance the practices of secure coders by simplifying the learning process, making it easier for them to identify and address security vulnerabilities.

It is challenging to achieve an objective assessment by only performing post-study surveys or interviews conducted after secure coders have finished the exercises. This is because without having a preemptive perception of the specific questions to ask or potential difficulties that could arise, then researchers may struggle to write survey questions that adequately capture the nuances of participants’ experiences. Furthermore, if researchers only perform post-study surveys or interviews, then it can be challenging to verify responses are genuinely representative of the experiences of everyone. Some participants may indicate that the study went well with no significant difficulties. They eventually overcame the problems, even though some aspects could be improved to reduce their cognitive load or frustrations. Secure coders may differ in their perceptions and willingness to give candid feedback on their experiences in a research study. Without real-time observations, it is challenging to determine these subtle aspects of the learning process. Eye-tracking technologies enable us to monitor, in real-time, how secure coders read and interact with software coding tools, thus allowing us to gain insights into their techniques and frustrations related to secure coding. Existing published literature lacks comprehensive insights derived from eye-tracking concerning secure coding, especially regarding dynamic and constantly changing source code stimuli.
Our prior chapters examined visualization methods and statistical analysis techniques to compare and contrast the behavior of secure coders. This chapter focuses on manually watching and analyzing the behaviors of secure coders working on a secure coding exercise. Specifically, we analyzed the behaviors of secure coders, determining if CWE-73 (File Manipulation) is a vulnerability in the ShareAlbum code base. This problem required secure coders to study learning material related to CWE-73 and then analyze the ShareAlbum code base with assistance from the RIPS static code analysis tool. This specific problem was selected for manual analysis and the results presented in this chapter because 15 secure coders answered the question correctly. In comparison, 16 secure coders did not provide the correct response. This CWE-73 exercise had more participants answer incorrectly compared to other CWEs exercises in our study. Moreover, it allowed us to study the actions of secure coders during their initial encounter with the RIPS static code analysis tool. The RIPS tool indicated that four potential errors exist in the ShareAlbum source code for the File Manipulation vulnerability. The specific question required that each secure coder answer “YES, this is a vulnerability as correctly determined by the RIPS tool” or “NO, the RIPS tool results are a false positive”.

8.2 Insight Captured from the Analysis of Secure Coders’ Behaviors

Below are the questions and insights we have acquired from our investigation into their coding behaviors. Examples of each coding problem provided to participants for eye-tracking analysis are presented in Appendix D. Further
information regarding the eye gaze patterns of each participant related to the below-mentioned insights can be found in Appendix E.

8.2.1 Did Secure Coders Struggle to Utilize the RIPS Static Code Analysis Tool for Their First Use Case of the RIPS Tool?

Yes, our examination of the eye gaze recordings indicated that several secure coders struggled with the specific file path that should be provided in the RIPS web interface to search for vulnerabilities in the ShareAlbum code base. The exercise was presented with an Instruction Guide that was specific to each CWE exercise and a generic RIPSHelp Instruction Guide. The Instruction Guide contained a relative file path (\Home\helpers\album.helper.php) to the ShareAlbum file (album.helper.php) that was specific to the CWE that was being worked. The RIPSHelp Instruction Guide contained a file path that was absolute (C:\wamp64\www\ShareAlbum\Home) to the ShareAlbum code base. The specific file (album.helper.php) was not required to be provided in the RIPS tool. However, the home webpage absolute path to the ShareAlbum code base was required for the RIPS tool.

Our analysis indicates that improving the instruction set for the RIPS tool file path is required to reduce secure coder confusion for future exercises. We believe that improving the instruction guide to help secure coders learn how to utilize software code analysis tools would be beneficial to improving their learn-
ing module experience. Several secure coders struggled for the initial part of this experiment by not knowing the exact file path that RIPS required to find the ShareAlbum code base. Enhancing the Instructional Guide stimuli would improve the learning module. It should also enhance secure coding practices by streamlining the process of promptly providing vulnerability reports to secure coders. This, in turn, will enable them to identify and address actual coding flaws more efficiently without getting frustrated that they cannot get a code analysis tool that should help them to work properly.

8.2.2 Did Secure Coders Answer Correctly Because They Found the Coding Flaw?

No, the analysis of their behaviors revealed that some secure coders answered the True or False question for CWE-73 due to the limited response options and not because they analyzed the source code containing the vulnerability. Two examples are the secure coder identified as S14 and the secure coder identified as S26. These secure coders got the correct answer; however, they did not look at the fourth reported vulnerability in RIPS, which is the one that contained the security flaw for CWE-73. Secure coder S14 did not direct their gaze towards the source code function add_a_new_file_to_album, even though it contains the security flaw associated with CWE-73. Nonetheless, they correctly identified that CWE-73 represents a security flaw in the ShareAlbum software. Similarly, secure coder S26 correctly identified that CWE-73 is a security flaw in ShareAlbum; how-
ever, the secure coder never looked into the source code containing the security flaw. Both of these secure coders faced significant challenges while attempting to initiate their work with the RIPS software. Their difficulties stemmed from the fact that they inadvertently provided an incorrect file path when launching their sessions. These experiences underscore the critical need for comprehensive training to ensure that secure coders can effectively leverage such tools to identify security vulnerabilities in software code. Additional details on the behaviors of secure coder S14 and S26 can be found in Appendix E.

8.2.3 Did Secure Coders Look at the Coding Flaw for CWE-73 but Still Answered Incorrectly That CWE-73 is Not an Issue in *ShareAlbum*?

Yes, our analysis revealed that some secure coders looked at the source code function and RIPS static code analysis tool report for CWE-73; however, the secure coders, for some unknown reason, did not correctly answer that the CWE-73 vulnerability existed in the *ShareAlbum* code base. Two examples are the secure coder identified as S30 and the secure coder identified as S18. Secure coder S30 started gazing at 560 seconds into their working exercise section at the software code function *add_a_new_file_to_album*, but specifically within that function at the internal function call to *move_uploaded_file*, which is the security flaw. This secure coder even opened the HELP guide in RIPS associated with this security flaw to examine the example code and proof of concept provided
by RIPS. However, they did not correctly determine that CWE-73 is an issue in *ShareAlbum*. For secure coder S18, the secure coder examined the fourth reported issue by RIPS at 339 seconds into their working exercise. At 343 seconds, the secure coder returned to the `album.helper.php` source code file. At 350 seconds, the secure coder opened the RIPS source code popup box for the fourth reported security flaw. The secure coder closed the popup box after 5 seconds. Therefore, one can visually determine that the secure coder did not spend a significant amount of time reading the fourth reported security flaw which is the one that contains the vulnerability for CWE-73. For further insights into the behaviors of secure coders S30 and S18, please consult Appendix E.

### 8.2.4 Did Secure Coders Respond Incorrectly Due to Not Examining the Code Containing the Security Flaw?

Yes, our analysis found that some secure coders answered incorrectly, and based on their reading behaviors, they could not have answered correctly. This is because they did not examine the source code function with the CWE-73 security flaw. An example is the secure coder identified above as S24. Secure coder S24 never examines the second, third, or fourth reported flaw by RIPS, and the fourth reported issue is the one that contains the flaw for CWE-73. Therefore, the secure coder could not have possibly answered the question correctly because they did not have an informed understanding of the security flaw. This secure coder also struggled to get the RIPS tool to work correctly to start their session.
For additional information regarding the behaviors of secure coder S24, please refer to Appendix E.

8.2.5 Do Secure Coders Practice Efficient Coding by Reviewing the Whole Code Before Focusing on a Specific Function, or Do They Inefficiently Jump Straight to the Function?

Yes, in some cases, several secure coders glanced over the majority of the source code file `album.helper.php` before focusing on specific programming lines in the source code. They commonly scrolled and glanced over several of the functions without spending significant time on any one software function. Several examples are secure coders: S10, S20, and S26. Furthermore, some secure coders, such as secure coder S14 and secure coder S10, utilized the Notepad++ FIND feature to search for specific strings in the source code files. For a more in-depth understanding of the behaviors exhibited by these specific secure coders, direct your attention to Appendix E.
Chapter 9. Conclusions

In summary, our research endeavors to combine behavioral analysis of secure coders and eye tracking to assess and comprehend secure coders’ software development skills. Our hands-on learning modules are designed to facilitate the learning process with step-by-step secure coding techniques and strategies. We helped participants by generating a list of weaknesses, prioritizing those weaknesses by risk, and targeting specific security vulnerabilities, as well as testing the mitigation solutions that were developed. Furthermore, we used eye trackers to study secure coding behaviors.

Our main focus is advancing cybersecurity education methodologies by understanding secure coding behavior using eye trackers. Understanding the behavior of participants’ as they learn secure coding practices and solve secure coding problems can facilitate the needed improvements that will allow for the creation of constructive learning content. This is an objective in order to gain insight into how to educate individuals on secure coding to improve data security and protect data privacy. We have previously developed dozens of learning modules that focused on cybersecurity. Our experience suggests that learning modules need continuous iterations and improvements. To the best of our knowledge, this is the first usage of eye tracking technologies to understand human behavior for se-
cure coding practices and investigate improving cybersecurity education. Our eye tracking research study that utilized our two secure coding learning modules had up to 31 undergraduate and graduate students participate. Our analysis indicates that one of the reasons that some participants performed poorly was because they spent less time reviewing the learning material, even though they spent about the same amount of time on the source code developing the solutions. As educators, understanding the behavior of participants while they learn in order to improve our understanding of their learning patterns is a critical step to improving the content utilized to teach secure coding.

One of our aspirations for our research was to determine the effectiveness and helpfulness of our secure coding learning modules to assist students in developing secure software applications. The purpose of this research was to study the effectiveness of the teaching material in the secure coding learning modules by studying the behavior of fixing security bugs in programming projects designed for pervasive computing environments. The teaching material was designed to facilitate participants’ understanding of the security and privacy challenges to the pervasive computing devices and communication by providing training on how to write secure software code and find security software flaws in existing source code. It was important to ensure that our two learning modules were concise yet informative to the participants in the course in order to decrease the overwhelming tasks of developing secure software applications. Using eye tracking allowed us to understand if participants read specific areas while perhaps disregarding other areas. Furthermore, we examined participants’ coding patterns and determined
what they really examined when secure coding by using eye tracking stimuli. This required converting our secure coding learning module into various types of eye tracking stimuli. We developed a progressive approach that will guide participants in identifying common coding flaws that allow for cyberattacks to then hands-on implementation to mitigate the discovered weaknesses in code.

We performed a literature survey of existing eye tracking analysis techniques including statistical as well as pictorial visualization methods, specifically focused on methods for stimuli and tasks that software developers use in various stages of the SDLC. Many techniques for post-processing data analysis exist for static stimuli content and limited time span eye gaze data collection [43, 44, 129]. Researchers have been examining eye tracking methods for general reading since the 1980s [89]. This is a commonly identified usage by Tobii Pro [188, 189]. The literature survey revealed the different types of visualization analysis techniques that have been examined for eye gaze data collected from eye trackers in the software development and software coding domain. However, we conducted our own eye tracking study to present new illustrations that utilize the same code base rather than reusing the same visualization illustrations that are already published. The literature survey identified several existing visualization techniques, but the specific illustrations in this section have not been previously published. Therefore, we present several of the standard pictorial visualization techniques commonly found in the published literature papers but generated visualizations from our eye tracking study. Our research has examined the visualization methods for the benefits and the limitations of the type of tasks and stimuli utilized in
software coding activities. We evaluated multiple statistical and graphical visualization techniques in relationship to several aspects of the design of an experiment with the goal of allowing each developer to read and write source code in a natural software coding environment.

In the Manual Code Analysis Learning Module, the students were presented with stimuli of selected functions from our *ShareAlbum* application. Then, students were asked multiple choice questions concerning those images and specific CWEs. The students were presented with source code to review and make a recommendation for a solution to mitigate specific weaknesses. Finally, students were presented with the source code files from ShareAlbum and asked to locate and mitigate specific CWE security vulnerabilities by fixing the source code. For each CWE in the Static Code Analysis Tool Learning Module, the students were presented with stimuli of the source code split screened with the web browser. The web browser had tabs for the RIPS web-based interface, the ShareAlbum web-interface, and a web search engine. This split-screen between the web browser and the source code allowed students to view the source code alongside the RIPS tool and the actual running ShareAlbum application. Students were presented with two true/false questions followed by a CWE that required the students to rewrite source code to mitigate a security vulnerability.

Our research is focused on examining existing visualization methods and crafting new visualization diagramming techniques that allow for examining the behavior of participants for secure coding projects and objectively visualizing secure coders’ behavior and patterns. This is critical in order to understand how
secure coders approach discovering security weaknesses, their usage of security analysis tools and their approaches to security mitigation techniques.

An essential phase after the data collection is the data analysis. The methodologies applied to transform the raw eye fixations and saccades into meaningful insight are lacking with certain stimuli content [58]. Common techniques are appropriate for reading paragraphs of natural reading text; however, these approaches do not work well for stimuli that are more dynamic media-based in nature such as source code reading or scrolling through web pages [111]. When the order of stimuli is not the same for all participants in the eye tracking recording sessions due to the unique individual behavior of the participant, then existing approaches of data analysis do not work well because the eye movement patterns are much more complex than simpler stimuli content [162]. Furthermore, it is often necessary to determine the visual scan path that participants take to solve a problem in the chronological order in which participants viewed the stimuli [54]. This can be especially beneficial to determining patterns between participants.

Our research study utilized heat maps, scan paths, and clusters which are commonly utilized techniques to investigate eye gazes of participants. Then, we explored visual plotting of transitions and AOIs to compare the transition patterns between AOIs. Our approach allowed us to study the reading patterns of participants as they reviewed source code and utilized the security analysis tools. These additional visualization methods allowed another viewpoint to overcome some of the limitations of the simpler techniques to manage several AOIs as well as the transitions between several AOIs. This allowed for exploring the dynamic
nature of scrolling and jumping in and out of source code that typically limits the ability to use common eye tracking visualization methods. Our techniques allowed for investigating participants’ eye gaze at the application level, and source code level and then visualizing their techniques and strategies that were used to learn and work secure coding exercises.

An AOI to Fixation Duration diagram allowed us to view the specific AOI that had a higher or a lower duration of fixations before transitioning to a different AOI. This provided benefits over the simpler fixation duration matrix that simply provided a sum of fixation duration per each AOI. This permitted us to determine over a time period when the participants had higher or lower fixation duration in an AOI. The combination of fixation duration, AOIs and a timeline provided significantly more insight of their behavior versus viewing this information separately. This diagramming technique provided insight into when in time and for which AOIs each secure coder had a higher number of fixations.

We determined that applying a grouping to several of the AOIs could be beneficial to visually examine when participants switch between major applications or subtasks such as reading instructions, reading source code, using static code analysis tools, and searching the web for assistance. Collecting and examining how developers find and mitigate software flaws can provide the ability to understand their behavior and approaches. Secure coders may not find every security weakness in an application even if using code analysis tools over simply manually reviewing the source code. It is also possible to correctly find a security flaw but not necessarily correctly mitigate the weakness or keep from introducing
new security or software bugs. Therefore, we believe it is important to examine when participants discover as well as when they mitigate security flaws. It is important to understand when individuals struggle to discover or mitigate a source code flaw allowing a security vulnerability. Understanding what aspects of a security tool the secure coders find useful and rely upon can allow for improvements to the security assessment tools. We created a swim lane representation of the major components using PlantUML. This allowed us to automate the diagram drawing process using Python scripting. We also created a state representation of the major components and created another diagramming method using PlantUML as well. This technique is designed to give a high-level overview of the transitions between major components.

We examined the left and right pupil data that were recorded during our eye recording sessions. Following a similar approach to visualize the pupil data, our work focused on creating custom scripts to examine the minimum, maximum and average pupil diameter changes over a timeline and for our AOIs. We also created the ability to sample the eye pupil size at specific intervals in time to keep from losing data points when participants do not transition out of an AOI for an extended period of time. To our knowledge, the literature is lacking in research that examines pupil changes in regard to secure coding.

In our eye tracking study, participants reviewed learning content related to secure coding, answered multiple-choice programming problems and developed mitigation strategies in source code. We evaluated the behavior of participants and examined their performance in solving security weaknesses in a web-based
application using eye tracking technology. Our results indicate that participants that answered a problem incorrectly, did not spend as much time on the material associated with learning about the problem when compared to participants that did answer correctly.

Our research indicates that visualization methods can allow for the ability to study eye gaze for participants conducting secure coding activities. The visualization of the data can allow for gaining insight into participants’ behavior and patterns among participants. The current literature has limited visualization methods for examining secure coders’ eye gazes while they learn and work on secure coding problems. Questionnaires and review sessions cannot impartially provide adequate feedback on software engineers’ learning techniques and their approaches to problem-solving. Eye tracking technologies offer the ability to objectively observe and measure software developers’ learning patterns and techniques in how they approach secure coding and software development.

Our research work is focused on gaining insight by examining the patterns in the behaviors of software developers who participated in our research study. Specifically, our interest is in examining the behavior when participants utilize multiple applications and multiple stimuli content in secure coding activities. Understanding user behaviors and patterns when they utilize multiple applications and when users control the ordering of stimuli events for eye tracking has not been extensively examined. Our belief is that by visualizing the various eye gaze transitions between multiple applications, then we can gain insight into the behavioral techniques and gaze patterns that secure coders utilize to solve problems.
Our research was able to draw meaningful insight into the behaviors of our secure coders and guide us on potential improvements to our secure coding learning modules. Our research also revealed that secure coders need clear instructions when utilizing new code-scanning tools for the first time.
Chapter 10. Future Work

Researchers and designers can utilize the behavior patterns and insight gained from eye tracking to enhance user experiences, improve the effectiveness of communication via visual aids and gain a more in-depth understanding of human cognition and perception of visual content. At this time, our focus is mainly on the software coding and software testing phase of the SDLC since many security flaws are fixed in those stages and we utilized an existing software application that has software flaws in the code. Software development, in general, also includes requirements gathering and documentation as well as design and architecture that we do not directly consider in our eye tracking experiment. Our research does consider the natural language text for instructions and learning content to help facilitate our participants’ learning about coding problems before starting on coding tasks. However, we do not directly explore stimuli containing UML diagrams, UX diagrams, database structure or cloud architecture deployment diagrams. Other researchers have already started exploring the design phase of software applications by investigating the eye gazes of participants as they view UML diagrams [154–157]. Software traceability with the requirements and code is another area of research that could benefit from eye tracking [136, 138, 153]. Other researchers indicate that dynamic stimuli content could be useful for this aspect of the SDLC
as well [141]. Eye tracking has the potential to facilitate the investigation of participants reading software requirements specification documents as well [152]. This could potentially be utilized to examine readers’ approaches to user manuals before products are released to the market in order to find the locations in the user manual where readers struggle. We have not fully explored each type of task that could be explored in software development due to the numerous types of tasks in every stage of the SDLC. Several of the visualization methods mentioned in this paper may provide value for those phases as well. We leave the exploration of specific stimuli elements related to the other software stages as a future research effort as we have focused primarily on software code reading and editing as well as software testing. Our framework provides an approach to ensure the validity of the experiment design and help facilitate the extraction of meaningful insights from the collected secure coders’ eye gaze data.
References


[171] The MITRE Corporation, “SECURE CODE REVIEW.”


[184] PlantUML, “PlantUML in a nutshell.”


Appendix A: Institutional Review Board and Student Consent

The Principal Investigator created a request to conduct an eye tracking research study and submitted it to the The University of Alabama in Huntsville Institutional Review Board. The research study requested authorization to conduct an eye tracking research study with students only from UAH. The UAH Institutional Review Board approved the research study for the Fall 2019 semester. Written consent from every student was mandatory before any eye tracking sessions commenced. The raw eye gaze data for every student is confidential and has not been publicly released.
Date: 29 Sept 2019

PI: Daniel Kyle Davis
PI Department: Computer Science
The University of Alabama in Huntsville

Dear Daniel,

The UAH Institutional Review Board of Human Subjects Committee has reviewed your proposal titled: *Teach Students Security and Privacy for Pervasive Computing Environments* and found it meets the necessary criteria for approval. Your proposal seems to be in compliance with these institutions Federal Wide Assurance (FWA) 00019998 and the DHHS Regulations for the Protection of Human Subjects (45 CFR 46).

Please note that this approval is good for one year from the date on this letter. If data collection continues past this period, you are responsible for processing a renewal application a minimum of 60 days prior to the expiration date.

No changes are to be made to the approved protocol without prior review and approval from the UAH IRB. All changes (e.g. a change in procedure, number of subjects, personnel, study locations, new recruitment materials, study instruments, etc) must be prospectively reviewed and approved by the IRB before they are implemented. You should report any unanticipated problems involving risks to the participants or others to the IRB Chair.

If you have any questions regarding the IRB’s decision, please contact me.

Sincerely,

Ann L. Bianchi
IRB Chair
Associate Professor, College of Nursing

OFFICE OF THE VICE PRESIDENT FOR RESEARCH
Von Braun Research Hall M-17  Huntsville, AL 35899  T 256.824.6100  F 256.824.5783

Figure A.1: IRB Approval Page 1 of 3.
Expedited: form 2

☐ Clinical studies of drugs and medical devices only when condition (a) or (b) is met. (a) Research on drugs for which an investigational new drug application (21 CFR Part 312) is not required. (Note: Research on marketed drugs that significantly increases the risks or decreases the acceptability of the risks associated with the use of the product is not eligible for expedited review. (b) Research on medical devices for which (i) an investigational device exemption application (21 CFR Part 812) is not required, or (ii) the medical device is cleared/approved for marketing and the medical device is being used in accordance with its cleared/approved labeling.

☐ Collection of blood samples by finger stick, heel stick, ear stick, or venipuncture as follows: (a) from healthy, nonpregnant adults who weigh at least 110 pounds. For these subjects, the amount drawn may not exceed 550 ml in an 8 week period and collection may not occur more frequently than 2 times per week; or (b) from other adults and children, considering the age, weight, and health of the subjects, the collection procedure, the amount of blood to be collected, and the frequency with which it will be collected. For these subjects, the amount drawn may not exceed the lesser of 50 ml or 3 ml per kg in an 8 week period and collection may not occur more frequently than 2 times per week.

☐ Prospective collection of biological specimens for research purposes by noninvasive means. Examples: (a) hair and nail clippings in a nondisfiguring manner; (b) deciduous teeth at time of exfoliation or if routine patient care indicates a need for extraction; (c) permanent teeth if routine patient care indicates a need for extraction; (d) excised and external secretions (including sweat); (e) unstimulated saliva collected either in an unstimulated fashion or stimulated by chewing gumbase or wax or by applying a dilute citric solution to the tongue; (f) placenta removed at delivery; (g) amniotic fluid obtained at the time of rupture of the membranes prior to or during labor; (h) supra- and subgingival dental plaque and calculus, provided the collection procedure is not more invasive than routine prophylactic scaling of the teeth and the process is accomplished in accordance with accepted prophylactic techniques; (i) nasooral and skin cells collected by boccal scraping or swab, skin swab, or mouth washings; (j) sputum collected after saline mist nebulization.

☐ Collection of data through noninvasive procedures (not involving general anesthesia or sedation) routinely employed in clinical practice, excluding procedures involving x-rays or microwaves. Where medical devices are employed, they must be cleared/approved for marketing. (Studies intended to evaluate the safety and effectiveness of the medical device are not generally eligible for expedited review, including studies of cleared medical devices for new indications).

☐ Research involving materials (data, documents, records, or specimens) that have been collected, or will be collected solely for nonresearch purposes (such as medical treatment or diagnosis).

☐ Collection of data from voice, video, digital, or image recordings made for research purposes.

☐ Research on individual or group characteristics or behavior (including, but not limited to, research on perception, cognition, motivation, identity, language, communication, cultural beliefs or practices, and social behavior) or research employing survey, interview, oral history, focus group, program evaluation, human factors evaluation, or quality assurance methodologies.
Exempt form 3:

☐ Research conducted in established or commonly accepted educational settings, involving normal educational practices, such as (a) research on regular and special education instructional strategies, or (b) research on the effectiveness of or the comparison among instructional techniques, curricula, or classroom management methods. The research is not FDA regulated and does not involve prisoners as participants.

☐ Research involving the use of educational tests (cognitive, diagnostic, aptitude, achievement), survey procedures, interviews, or observation of public behavior in which information is obtained in a manner that human subjects cannot be identified directly or through identifiers linked to the subjects and any disclosure of the human subject’s responses outside the research would not place the subjects at risk of criminal or civil liability or be damaging to the subject’s financial standing, employability, or reputation. The research is not FDA regulated and does not involve prisoners as participants.

☐ Research involving the use of educational tests (cognitive, diagnostic, aptitude, achievement) survey procedures, interview procedures, or observation of public behavior if (a) the human subjects are elected or appointed public officials or candidates for public office, or (b) Federal statute(s) require(s) without exception that the confidentiality of the personally identifiable information will be maintained throughout the research and thereafter. The research is not FDA regulated and does not involve prisoners as participants.

☐ Research involving the collection or study of existing data, documents, records, pathological specimens, or diagnostic specimens, if these sources are publicly available or if the information is recorded by the investigator in such a manner that subjects cannot be identified, directly or through identifiers linked to the subjects. The research is not FDA regulated and does not involve prisoners as participants.

☐ Research and demonstration projects which are conducted by or subject to the approval of department or agency heads, and which are designed to study, evaluate, or otherwise examine: (i) public benefit or service programs; (ii) procedures for obtaining benefits or services under those programs; (iii) possible changes in or alternatives to those programs or procedures; or (iv) possible changes in methods or levels of payment for benefits or services under those programs. The protocol will be conducted pursuant to specific federal statutory authority; has no statutory requirement for IRB review; does not involve significant physical invasions or intrusions upon the privacy interests of the participant; has authorization or concurrent by the funding agency and does not involve prisoners as participants.

☐ Taste and food-quality evaluation and consumer acceptance studies, (i) if wholesome foods without additives are consumed or (ii) if a food is consumed that contains a food ingredient at or below the level and for a use found to be safe, or agricultural chemical or environmental contaminant at or below the level found to be safe, by the Food and Drug Administration or approved by the Environmental Protection Agency or the Food Safety and Inspection Service of the U.S. Department of Agriculture. The research does not involve prisoners as participants.

☐ Surveys, interviews, or observation of public behavior involving children cannot be exempt.
Date: 12 October 2019

PI: Daniel Davis
PI Department: Computer Science
The University of Alabama in Huntsville

Dear Daniel,

The UAH Institutional Review Board of Human Subjects Committee has reviewed your proposal titled, Teach Students Security and Privacy for Pervasive Computing Environments and found it meets the necessary criteria for approval. Your proposal seems to be in compliance with these institutions Federal Wide Assurance (FWA) 00019998 and the DHHS Regulations for the Protection of Human Subjects (45 CFR 46).

Please note that this approval is good for one year from the date on this letter. If data collection continues past this period, you are responsible for processing a renewal application a minimum of 60 days prior to the expiration date.

No changes are to be made to the approved protocol without prior review and approval from the UAH IRB. All changes (e.g. a change in procedure, number of subjects, personnel, study locations, new recruitment materials, study instruments, etc) must be prospectively reviewed and approved by the IRB before they are implemented. You should report any unanticipated problems involving risks to the participants or others to the IRB Chair.

If you have any questions regarding the IRB’s decision, please contact me.

Sincerely,

[Signature]

Ann L. Bianchi
IRB Chair
Associate Professor, College of Nursing

Figure A.4: Revised IRB Approval Page 1 of 3.
Expedited:

- Clinical studies of drugs and medical devices only when condition (a) or (b) is met: (a) Research on drugs for which an investigational new drug application (21 CFR Part 312) is not required. (Note: Research on marketed drugs that significantly increases the risks or decreases the acceptability of the risks associated with the use of the product is not eligible for expedited review. (b) Research on medical devices for which (i) an investigational device exemption application (21 CFR Part 812) is not required, or (ii) the medical device is cleared/approved for marketing and the medical device is being used in accordance with its cleared/approved labeling.

- Collection of blood samples by finger stick, heel stick, arm stick, or venipuncture as follows: (a) from healthy, nonpregnant adults who weigh at least 110 pounds. For these subjects, the amounts drawn may not exceed 550 ml in an 8 week period and collection may not occur more frequently than 2 times per week; or (b) from other adults and children, considering the age, weight, and health of the subjects, the collection procedure, the amount of blood to be collected, and the frequency with which it will be collected. For these subjects, the amount drawn may not exceed the lesser of 56 ml or 3 ml per kg in an 8 week period and collection may not occur more frequently than 2 times per week.

- Prospective collection of biological specimens for research purposes by noninvasive means. Examples: (a) hair and nail clippings in a noninvasive manner; (b) deciduous teeth at time of exfoliation or if routine patient care indicates a need for extraction; (c) permanent teeth if routine patient care indicates a need for extraction; (d) excreta and external secretions (including sweat); (e) uncultured saliva collected either in an unstimulated fashion or stimulated by chewing gumbase or wax or by applying a dilute citric solution to the tongue; (f) placenta removed at delivery; (g) amniotic fluid obtained at the time of rupture of the membranes prior to or during labor; (h) supra- and subgingival dental plaque and calculus; provided the collection procedure is not more invasive than routine prophylactic scaling of the teeth and the process is accomplished in accordance with accepted prophylactic techniques; (i) mucosal and skin cells collected by buccal scraping or swab, skin swab, or mouth washings; (j) sputum collected after saline mist nebulization.

- Collection of data through noninvasive procedures (not involving general anesthesia or sedation) routinely employed in clinical practice, excluding procedures involving x-rays or microwaves. Where medical devices are employed, they must be cleared/approved for marketing. (Studies intended to evaluate the safety and effectiveness of the medical device are not generally eligible for expedited review, including studies of cleared medical devices for new indications).

- Research involving materials (data, documents, records, or specimens) that have been collected, or will be collected solely for nonresearch purposes (such as medical treatment or diagnosis).

- Collection of data from voice, video, digital, or image recordings made for research purposes.

- Research on individual or group characteristics or behavior (including, but not limited to, research on perception, cognition, motivation, identity, language, communication, cultural beliefs or practices, and social behavior) or research employing survey, interview, oral history, focus group, program evaluation, human factors evaluation, or quality assurance methodologies.
Exempt

☐ Research conducted in established or commonly accepted educational settings, involving normal educational practices, such as (a) research on regular and special education instructional strategies, or (b) research on the effectiveness of or the comparison among instructional techniques, curricula, or classroom management methods. The research is not FDA regulated and does not involve prisoners as participants.

☐ Research involving the use of educational tests (cognitive, diagnostic, aptitude, achievement), survey procedures, interviews, or observation of public behavior in which information is obtained in a manner that human subjects cannot be identified directly or through identifiers linked to the subjects and any disclosure of the human subject’s responses outside the research would not place the subjects at risk of criminal or civil liability or be damaging to the subject’s financial standing, employability, or reputation. The research is not FDA regulated and does not involve prisoners as participants.

☐ Research involving the use of educational tests (cognitive, diagnostic, aptitude, achievement) survey procedures, interview procedures, or observation of public behavior if (a) the human subjects are elected or appointed public officials or candidates for public office, or (b) Federal statute(s) require(s) without exception that the confidentiality of the personally identifiable information will be maintained throughout the research and thereafter. The research is not FDA regulated and does not involve prisoners as participants.

☐ Research involving the collection or study of existing data, documents, records, pathological specimens, or diagnostic specimens, if these sources are publicly available or if the information is recorded by the investigator in such a manner that subjects cannot be identified, directly or through identifiers linked to the subjects. The research is not FDA regulated and does not involve prisoners as participants.

☐ Research and demonstration projects which are conducted by or subject to the approval of department or agency heads, and which are designed to study, evaluate, or otherwise examine: (i) public benefit or service programs; (ii) procedures for obtaining benefits or services under those programs; (iii) possible changes in or alternative to those programs or procedures; or (iv) possible changes in methods of or levels of payment for benefits or services under those programs. The protocol will be conducted pursuant to specific federal statutory authority; has no statutory requirement for IRB review; does not involve significant physical invasion or intrusion upon the privacy interests of the participant; has authorization or consent by the funding agency and does not involve prisoners as participants.

☐ Taste and food-quality evaluation and consumer acceptance studies, (i) if wholefood foods without additives are consumed or (ii) if a food is consumed that contains a food ingredient at or below the level and for a use found to be safe; or agricultural chemical or environmental contaminant at or below the level found to be safe by the Food and Drug Administration or approved by the Environmental Protection Agency or the Food Safety and Inspection Service of the U.S. Department of Agriculture. The research does not involve prisoners as participants.

☐ Surveys, interviews, or observation of public behavior involving children cannot be exempt.
Recruitment Form: Teach Students Security and Privacy for Pervasive Computing Environments

Researchers in the Computer Science Department at the University of Alabama in Huntsville (UAH) are conducting a study that will examine learning modules designed for teaching students security and privacy for pervasive computing environments.

The purpose of this research is to study the effectiveness of the teaching material on the UAH PSP (Pervasive Security and Privacy) website by studying the behavior of fixing security bugs in programming projects designed for pervasive computing environments. The teaching material is designed to facilitate students’ understanding of the security and privacy challenges to the pervasive computing devices and communication. Furthermore, the teaching materials is designed to help instructors achieve expected learning goals of security and privacy for pervasive computing. The PSP learning modules assignments are designed at a suitable challenging level and will not contain nonessential and time-consuming tasks.

UAH students who are taking the following courses will be allowed to participate in the study:

- (CS-485) Computer & Software Security
- (CS-585) Computer Security
- (CS-685) Advanced Computer Security

Please be advised that this experiment is for students above the age of 18. There is no physical or psychological risk associated with this study. Your participation in this Research Study is VOLUNTARY!

For ALL students enrolled in CS-485 and CS-585 and CS-685

The course professor will assign a class programming project from the PSP website that is mandatory.

For students that do NOT wish to participate in an Eye Tracking Research Study

If you decide NOT to participate in the research study, you will not lose any points for the class programming project and in no ways will affect your grade for declining to participate in the research. The maximum group size is 2 students for the class programming project assignment.

For students that wish to participate in an Eye Tracking Research Study

If you decide to participate in the research study, you are still responsible for the class programming project that Dr. Zhu assigns for CS485 and CS585 and CS685. As an incentive for joining the research study, more students will be allowed to form a group in the class project that Dr. Zhu assigns for CS485 and CS585 and CS685. This is to compensate for your time spent in the study, as each of you in the larger group will have less workload for the class project which compensates you for the time required for the research study. For students who participate in the study, you may have a group of up to 3 students for the class project to compensate for the time required for the eye tracking study. All members of the group must participate in the research study. Students who participate can add to their resume that they have experience in participating in a research study. You will not receive any financial pay for participating.

If you agree to join the research study, once written consent is given using the approved consent form, you will schedule a day and time slot with the Principal Investigator to work on the projects from the PSP website. This involves working secure coding projects in a Cyber Security Lab in the Computer Science Department at UAH. This includes reading the teaching material and working on software programs presented in the PSP learning modules. We will use an eye tracker (Tobi Pro X2-60 with Tobi Studio or

Figure A.7: Recruitment Form: Teach Students Security and Privacy for Pervasive Computing Environments Page 1 of 3.
Eye Tribe Tracker with the software that we developed) to record your eye movements, while you study the teaching material and work the learning module lessons.

If you decide to participate in the research, the only cost to you is the time you spend during the study. It may take 30 minutes to 2 hours to complete the eye tracking component of the research study with an additional 15 minutes to 30 minutes for the questionnaire. There should be no more eye strain than the normal computer usage. The PI, Daniel Davis, will help research participants calibrate the eye tracker in the beginning of their work and remain available during the session for any issues or questions. Participants in the research study will work on the PSP projects on their own paces while their eye gaze is recorded.

Unique participant numbers will be generated to record your data, and these numbers will be made available only to those researchers directly involved with this study, thereby ensuring strict confidentiality. The researchers will not utilize your name for the data collection. The student should not share their unique participant number with any other individual.

After the learning module is worked, you will be asked a series of questions in a questionnaire about your opinion and experiences of the PSP framework and the teaching material, as well as about demographic information (e.g., age, gender). You will be offered the opportunity to see your own eye gaze recorded by eye trackers. This eye gaze review is optional by the individual student preferences.

A statistical analysis of the collected data cannot be provided the same day as students complete the research study. This is due to the complexity of analyzing the data and the post-processing of the data that is required from the raw eye data points to generate the statistical metrics commonly utilized for eye tracking. Therefore, the PI, Daniel Davis, will present the latest statistical results as a class presentation for the students towards the end of the class semester. The instructor of the course, Dr. Feng Zhu, will schedule a specific day during normal course schedule time that Daniel Davis will present the latest statistical results to the students as a class presentation. This in-class presentation by Daniel Davis will be provided to all students of Dr. Zhu’s class if they participated in the research study or declined the study.

If you accept the research study, you are free to withdraw from the study at any time. You will not be penalized because of withdrawal in any form. Investigators reserve the right to remove any participant from the session without regard to the participant’s consent.

If you have any questions about the study, you may contact the Principal Investigator at any point in the research process. You may contact the PI, Daniel Kyle Davis, graduate student in the Department of Computer Science, Technology Hall, The University of Alabama, Huntsville, AL, USA, 35899, dkd0004@uah.edu; (256) 937-8139. You may also contact the faculty supervisor, Dr. Feng Zhu from the Department of Computer Science, Technology Hall, The University of Alabama, Huntsville, AL, USA, 35899, fz0001@uah.edu; (256) 824-6255. If you have questions about your rights as a research participant, or concerns or complaints about the research, you may contact the Office of the IRB (IRB) at 256.824.2465 or email Dr. Ann Bianchi, IRB Chair, at irb@uah.edu.

If you would like to learn more about the opportunity to be a part of this study, please contact the Principal Investigator. You may contact the PI, Daniel Kyle Davis, graduate student in the Department of Computer Science, Technology Hall, The University of Alabama, Huntsville, AL, USA, 35899, dkd0004@uah.edu; (256) 937-8139.

Figure A.8: Recruitment Form: Teach Students Security and Privacy for Pervasive Computing Environments Page 2 of 3.
This study was approved by the Institutional Review Board at UAH and will expire in one year from September 29, 2019.

Figure A.9: Recruitment Form: Teach Students Security and Privacy for Pervasive Computing Environments Page 3 of 3.
Consent Form: Teach Students Security and Privacy for Pervasive Computing Environments

You are invited to participate in a study. The purpose of this research is to study the behavior of fixing security bug in pervasive security and privacy projects. If you decide not to participate in the study, you will not lose any points for the class project and declining will not affect your grade. If you participate, as an incentive, more numbers of students will be allowed to form a group for the class project. That is to compensate for your time spent in the study; each of you in the larger group will have less workload. If you finish the study, you will be offered to see your own eye gaze patterns recorded by eye trackers.

The Primary Investigator (PI) is Daniel Kyle Davis, a PhD candidate from the Department of Computer Science, Technology Hall, The University of Alabama, Huntsville, AL, USA, 35899, (256) 937-8139. Please be advised that this experiment is for students above age 18.

PROCEDURE TO BE FOLLOWED IN THE STUDY: Once written consent is given for the study, you will be asked to work on one of the projects on the PSP website. We will use an eye tracker (Tobi Pro X2-60 with Tobii Studio or Eye Tribe Tracker with the software that we developed) to record your eye movements, while you study the teaching material. After the eye tracking recording, you will be asked a series of questions about your opinion and experience of the PSP framework and the teaching material, as well as about demographic information (e.g., age, gender). There is no physical or psychological risk associated with this study. The only cost to you is the time you spend during the study. There should be no more eye strain than the normal computer usage. It may take 30 minutes to 2 hours to complete the eye tracking recording with an additional 15 minutes to 30 minutes for the questionnaire.

CONFIDENTIALITY OF RESULTS: Unique participant numbers will be generated and will be used to record your data, and these numbers will be made available only to those researchers directly involved with this study, thereby ensuring strict confidentiality. Consent forms will be destroyed after 3 years. The data from your session will only be released to those individuals who are directly involved in the research and only using your unique participant number. The researchers will not utilize your name for the data. The student should not share their unique participant number with any other individual.

FREEDOM TO WITHDRAW: You are free to withdraw from the study at any time. You will not be penalized because of withdrawal in any form. Investigators reserve the right to remove any participant from the session without regard to the participant’s consent.

CONTACT INFORMATION: If you have any questions about the study, you may contact the Principal Investigator at any point in the research process. You may contact the PI, Daniel Davis, graduate student in the Department of Computer Science, Technology Hall, The University of Alabama, Huntsville, AL, USA, 35899. dkld0004@uah.edu, (256) 937-8139. You may also contact the faculty supervisor, Dr. Feng Zhu from the Department of Computer Science, Technology Hall, The University of Alabama, Huntsville, AL, USA, 35899. fz0001@uah.edu, (256) 824-6255. If you have questions about your rights as a research participant, or concerns or complaints about the research, you may contact the Office of the IRB (IRB) at 256.824.2465 or email Dr. Ann Bianchi, IRB Chair, at irb@uah.edu.

If you agree to participate in our research, please sign and date below. This study was approved by the Institutional Review Board at UAH and will expire in one year from September 29, 2019.

Name (Please Print) ________________________________  Signature ________________________________  Date ________________________________

Figure A.10: Consent Form: Teach Students Security and Privacy for Pervasive Computing Environments.
Debriefing Form: Teach Students Security and Privacy for Pervasive Computing Environments

Thank you for participating in our research study on security and privacy. The purpose of this research is to study the effectiveness of the teaching material on the UAH PSP (Pervasive Security and Privacy) website by studying the behavior of fixing security bugs in programming projects designed for pervasive computing environments. Your eye gaze data will be further analyzed by the researchers, Daniel Davis and Dr. Feng Zhu, in order to determine if any improvements in the educational content can be completed by the researchers. The eye tracked data will help to understand whether the teaching material facilitates the understanding of security and privacy challenges and students’ behavior. Also, this will help to identify nonessential tasks and important details that are missing. This will enable researchers to design and develop adaptive content based on an individual’s need. Your contributions today are greatly appreciated.

Now, that you have concluded the eye tracking component of the study, you will complete a questionnaire about your demographic information (e.g., age, gender), questions related to the content in the study and any suggested improvements for future studies based upon your experience with the PSP framework and the teaching material just presented. This questionnaire should take no more than 15 minutes to 30 minutes. The questionnaire will be anonymous to protect your confidentiality and anonymity.

You are offered the opportunity to see your own eye gaze patterns recorded by the eye trackers. This is a voluntary component that you may choose to view your results or skip. If you decide to view your eye gaze patterns, it will not affect your results. The only cost is the time to view the results. If you wish to view your results at a later date, you must have the unique participant number that was provided at the start of the study. Again, you should not share your unique participant number with any other individual.

A statistical analysis of your collected data cannot be provided the same day. This is due to the complexity of analyzing the data and the post-processing of the data that is required. Therefore, the PI, Daniel Davis, will present the latest statistical results as a class presentation for the students towards the end of the class semester. The instructor of the course, Dr. Feng Zhu will schedule a specific day during normal course schedule time that Daniel Davis will present the latest statistical results to the students as a class presentation.

Thank You for Participating in the Eye Tracking Research Study! Do not forget to update your resume indicating you have participated in a research study.

CONTACT INFORMATION: If you have any questions about the study, you may contact the Principal Investigator at any point in the research process. You may contact the PI, Daniel Kyle Davis, graduate student in the Department of Computer Science, Technology Hall, The University of Alabama, Huntsville, AL, USA, 35899, dkd0004@uah.edu, (256) 937-8139. You may also contact the faculty supervisor, Dr. Feng Zhu from the Department of Computer Science, Technology Hall, The University of Alabama, Huntsville, AL, USA, 35899, fz0001@uah.edu, (256) 824-6255. If you have questions about your rights as a research participant, or concerns or complaints about the research, you may contact the Office of the IRB (IRB) at 256.824.2465 or email Dr. Ann Blanch, IRB Chair, at irb@uah.edu.

Note that this study was approved by the Institutional Review Board at UAH and will expire in one year from September 29, 2019.

Figure A.11: Debriefing Form: Teach Students Security and Privacy for Pervasive Computing Environments.
Appendix B: Setup of the Eye Tracking Equipment

We used the Tobii X2-60 eye tracking hardware and the Tobii Studio software for the recording application. The Tobii Studio application provides the ability to present several types of stimuli content, manage participate data collection and provide data filtering of the raw data points. Each student used a single twenty-four-inch 1080p monitor with a refresh rate of 144Hz to view our stimuli content. One Dell G7 2018 Laptop was connected to the ASUS monitor. The laptop built-in monitor was not used in the recording session. A single external wired keyword and an external wired mouse was used as the interface device for human input. The Tobii X2-60 eye tracker has a tracking frequency of 60Hz that collected the gaze data points and pupil diameters [79]. The eye tracker is capable of recording both the left and right pupil sizes of participants [58]. The Tobii X2-60 eye tracker was fixated below the monitor for our research study. This eye tracker does not require a chin rest; therefore, it allows for a participant to move their head around when not in the calibration mode. All participants were located approximately 50 to 70 cm from the monitor and eye tracker. Typically, eye tracking hardware can provide more accurate results if the lighting conditions is stable [190]. Generally, results in outdoor environments are not as reliable as an indoor setting [53]. Depending upon the research study, participants may also benefit from a location that has reduced external noise to allow better concentration. A single eye tracking collection laboratory room with
no windows was utilized and signs indicating that it was a useability laboratory was posted. The signs informed individuals to remain quiet in the surrounding areas. Our room was chosen to minimize noise and bright lights as it is important to make the student feel as comfortable as possible [84]. We conducted a small exploratory pilot study to obtain a general overview of the tasks and the study layout. Feedback was positive but our exploratory pilot participants asked for more helpful guidance in the flow of material and to allow for a half-way point break. We implemented the recommendations before conducting the actual study.

We utilized the Tobii Studio (Subscription Version 3.4.8) software to record the raw eye gazes as the Tobii Studio software has the capability of presenting the various stimuli elements while eye gazes are recorded [58]. Furthermore, the Tobii Studio software was used to create the AOIs on static and non-linear dynamic stimuli as well as perform our initial analysis as discussed in the following sections. Participants had the option of using the RIPS static code analysis tool [173]. Additional details in the procedure were discussed when a participant was allowed to use this tool. The Google Chrome web browser was the only web browser that was allowed to be utilized. The Python programming language was utilized for data analysis and diagramming plot generation with the eye gaze data exported from Tobii Studio in CSV files [58]. The Notepad++ application was utilized for code reading and editing as it provides basic functionality for text editing and source code syntax coloring [191].
Appendix C: Student Questionnaire Survey

Figure C.12: Student Questionnaire Survey Page 1 of 5.
14. It is mentioned that the code analysis tool often produces what type of errors?  
[A] False, positive errors  
[B] False, negative errors  
[C] No code analysis tool was mentioned  

15. The CWE-79 Improper Neutralization of Input During Web Page Generation was determined to be a weakness due to which of the following HTTP methods?  
[A] HTTP PUT  
[B] HTTP GET  
[C] HTTP POST  
[D] HTTP HEAD  
[E] None of the above  

16. The CWE-89 Improper Neutralization of Special Elements used in an SQL Command is extremely costly and difficult to mitigate.  
[A] True  
[B] False  

17. According to the scoring weight provided in the learning module, which of the following had the lowest "weight" that is used when calculating the S-Value?  
[A] Ease of Detection  
[B] Attacker Awareness  
[C] Remediation Cost  
[D] Attack Frequency  
[E] None of the above  

18. A potential mitigation technique for an SQL injection is to ensure that input strictly conforms to specifications as expected.  
[A] True  
[B] False  

19. The sample programming project only provides the capability of sharing user uploads with a public view.  
[A] True  
[B] False  
[C] View not mentioned in the learning module  

20. It is mentioned in the learning module that attackers could utilize a weakness with improper pathname limitations on file uploads to inject malicious files or executables.  
[A] True  
[B] False  
[C] File uploading weaknesses not mentioned in the learning module  

21. How many electronic devices with sensitive data do you routinely use?  
(A) None  
(B) 1-2  
(C) 3-4  
(D) 5-6  
(E) > 6  

22. Which of the following mobile devices do you use?  
(Check all that apply)  
(A) iOS Phone/Tablet  
(B) Windows Phone/Tablet  
(C) Android Phone/Tablet  
(D) Other  

23. Are these mobile devices for ________ use?  
(A) Personal  
(B) Professional  
(C) Both  

24. How often do you use the mobile device(s)?  
(A) Several times a day  
(B) Daily  
(C) Weekly  
(D) Occasionally  

25. What types of apps do you utilize on your mobile device?  
(Check all that apply)  
(A) Productivity (notetaking, calendar, file synchronization, document creation, etc.)  
(B) Financial (banking, credit cards, budget, etc.)  
(C) Shopping (Ebay, Amazon, etc.)  
(D) Video Games  
(E) Entertainment (YouTube, Hulu, Netflix, etc.)

Figure C.13: Student Questionnaire Survey Page 2 of 5.
Figure C.14: Student Questionnaire Survey Page 3 of 5.
<table>
<thead>
<tr>
<th>Question</th>
<th>Options</th>
<th>Answer</th>
</tr>
</thead>
</table>
| 30. The following questions ask about the steps that you may or may not take to maintain your security. How often do you engage in the following behaviors? | [A] Never  
[B] Sometimes  
[C] Very Often |        |
| [A] Reveal personal information on the Internet or filling out paper forms | ANSWER:        | ANSWER: |
| [B] Carefully read privacy policies | ANSWER:        | ANSWER: |
| [C] Have multiple email accounts to protect privacy | ANSWER:        | ANSWER: |
| [D] Maintain anti-malware applications | ANSWER:        | ANSWER: |
| [E] Revealing personal information if it will allow the provider to give you better service or price | ANSWER:        | ANSWER: |
| [F] Finding out how a company or organization plans to use your identities or private information | ANSWER:        | ANSWER: |
| [G] Falsifying (lying) information about yourself on a website or paper forms | ANSWER:        | ANSWER: |
| [H] Downloading security patches for your personal computer | ANSWER:        | ANSWER: |
| 31. Compared to projects in other courses, what features do you like most? (Please check all that apply) | [A] Projects descriptions and instructions  
[B] PowerPoint Presentation  
[C] Video of PowerPoint Presentation  
[D] Video of executed demo  
[E] Sample projects |        |
| [A] The projects help me to secure my devices | ANSWER:        | ANSWER: |
| [B] The projects help me to securely use the Internet | ANSWER:        | ANSWER: |
| [C] The projects help me to securely write code | ANSWER:        | ANSWER: |
| [D] I like the security projects | ANSWER:        | ANSWER: |
| [E] I will recommend the website and projects to others | ANSWER:        | ANSWER: |
| [F] No One | ANSWER:        | ANSWER: |

Figure C.15: Student Questionnaire Survey Page 4 of 5.
37. Have you presented any of the CVE's that we presented during this SECURITY CODING PRACTICES RESEARCH STUDY as part of the CS-485/CS-585 course? Mark all CVE's that apply.
   [A] CWE-22 : Improper Neutralization of a Pathname to a Restricted Directory (Path Traversal)
   [B] CWE-79 : Improper Neutralization of Input During Web Page Generation (Cross-Site Scripting)
   [C] CWE-89 : Improper Neutralization of Special Elements Used in an SQL Command (SQL Injection)
   [D] CWE-311 : Missing Encryption of Sensitive Data
   [E] CWE-327 : Use of a Broken or Risky Cryptographic Algorithm
   [F] CWE-434 : Unrestricted Upload of File with Dangerous Types
   [G] CWE-759 : Use of a One-Way Hash without a Salt
   [H] CWE-798 : Use of Hard-coded Credentials
   [I] CWE-862 : Missing Authorization
   [J] I have presented NONE of these CVE's for the CS-485/CS-585 course

38. How would you rate your programming experience level for software development?
   [A] Trainee/Entry Developer/Engineer
   [B] Junior Developer/Engineer
   [C] Middle Developer/Engineer
   [D] Senior Developer/Engineer

39. Did the Code Analysis Tool help you to discover and mitigate the vulnerabilities easier than manually finding and mitigating as completed without using a Code Analysis Tool?
   [A] Strongly agree
   [B] Agree
   [C] Don’t know
   [D] Disagree
   [E] Strongly disagree

40. Did the Code Analysis Tool help you to discover and mitigate the vulnerabilities quicker than manually finding and mitigating as completed without using a Code Analysis Tool?
   [A] Strongly agree
   [B] Agree
   [C] Don’t know
   [D] Disagree
   [E] Strongly disagree

Figure C.16: Student Questionnaire Survey Page 5 of 5.
• What is your age?

**Figure C.17:** Manual Code Analysis Module Question 1.

**Figure C.18:** Static Code Analysis Module Question 1.
• What is your gender?

**Figure C.19:** Manual Code Analysis Module Question 2.

**Figure C.20:** Static Code Analysis Module Question 2.
• What is your ethnicity?

**Figure C.21:** Manual Code Analysis Module Question 3.

**Figure C.22:** Static Code Analysis Module Question 3.
What is the technique discussed in the module that is for rating the weaknesses and severity in a trustworthy and compliant manner while accommodating for the various business domains that security weaknesses exist in?

**Figure C.23:** Manual Code Analysis Module Question 4.

**Figure C.24:** Static Code Analysis Module Question 4.
• This source code analysis is usually performed during a code review and is also known as a ———— test.

**Figure C.25:** Manual Code Analysis Module Question 5.

**Figure C.26:** Static Code Analysis Module Question 5.
This analysis technique is performed in a non-runtime environment by inspecting the source code.

**Figure C.27:** Manual Code Analysis Module Question 6.

**Figure C.28:** Static Code Analysis Module Question 6.
• What is the primary purpose of the sample programming project mentioned in the learning module?

Figure C.29: Manual Code Analysis Module Question 7.

Figure C.30: Static Code Analysis Module Question 7.
The sample programming project supports the ability to send messages to other registered users.

**Figure C.31:** Manual Code Analysis Module Question 8.

**Figure C.32:** Static Code Analysis Module Question 8.
• The sample programming project supports the ability to leave comments on uploaded content.

**Figure C.33:** Manual Code Analysis Module Question 9.

**Figure C.34:** Static Code Analysis Module Question 9.
• The primary programming language for the sample programming project is [_____].

**Figure C.35:** Manual Code Analysis Module Question 10.

**Figure C.36:** Static Code Analysis Module Question 10.
• The name of the code analysis software tool used in this experiment is called __________.

**Figure C.37:** Manual Code Analysis Module Question 11.

**Figure C.38:** Static Code Analysis Module Question 11.
• According to the scoring weight provided in the learning module, which of the following had the highest “weight” that is used when calculating the S-Value?

![Figure C.39: Manual Code Analysis Module Question 12.](image)

![Figure C.40: Static Code Analysis Module Question 12.](image)
• A lower S-Value implies that the weakness should be at a higher priority to be mitigated than a weakness with a higher S-value.

**Figure C.41:** Manual Code Analysis Module Question 13.

**Figure C.42:** Static Code Analysis Module Question 13.
• It is mentioned that the code analysis tool often produces what type of errors?

**Figure C.43:** Manual Code Analysis Module Question 14.

**Figure C.44:** Static Code Analysis Module Question 14.
• The CWE-79 Improper Neutralization of Input During Web Page Generation was determined to be a weakness due to which of the following HTTP methods?

Figure C.45: Manual Code Analysis Module Question 15.

Figure C.46: Static Code Analysis Module Question 15.
• The CWE-89 Improper Neutralization of Special Elements used in an SQL Command is extremely costly and difficult to mitigate.

**Figure C.47**: Manual Code Analysis Module Question 16.

**Figure C.48**: Static Code Analysis Module Question 16.
According to the scoring weight provided in the learning module, which of the following had the lowest “weight” that is used when calculating the S-Value?

**Figure C.49:** Manual Code Analysis Module Question 17.

**Figure C.50:** Static Code Analysis Module Question 17.
• A potential mitigation technique for an SQL injection is to ensure that input strictly confirms to specifications as expected.

**Figure C.51:** Manual Code Analysis Module Question 18.

**Figure C.52:** Static Code Analysis Module Question 18.
The sample programming project only provides the capability of sharing user uploads with a public view.

**Figure C.53:** Manual Code Analysis Module Question 19.

**Figure C.54:** Static Code Analysis Module Question 19.
- It is mentioned in the learning module that attackers could utilize a weakness with improper pathname limitations on file uploads to inject malicious files or executables.

**Figure C.55:** Manual Code Analysis Module Question 20.

**Figure C.56:** Static Code Analysis Module Question 20.
• How many electronic devices with sensitive data on the devices do you routinely use?

**Figure C.57:** Manual Code Analysis Module Question 21.

**Figure C.58:** Static Code Analysis Module Question 21.
• Which of the following mobile devices do you use? (Check all that apply)

**Figure C.59:** Manual Code Analysis Module Question 22.

**Figure C.60:** Static Code Analysis Module Question 22.
• Are these mobile devices for ———— use?

**Figure C.61:** Manual Code Analysis Module Question 23.

**Figure C.62:** Static Code Analysis Module Question 23.
• How often do you use the mobile device(s)?

Figure C.63: Manual Code Analysis Module Question 24.

Figure C.64: Static Code Analysis Module Question 24.
- What types of apps do you utilize on your mobile device? (Check all that apply)

**Figure C.65:** Manual Code Analysis Module Question 25.

**Figure C.66:** Static Code Analysis Module Question 25.
What types of activities do you perform on your device using publicly accessible Wi-Fi networks?

**Figure C.67: Manual Code Analysis Module Question 26.**
Figure C.68: Static Code Analysis Module Question 26.
• Do you feel secure to access Internet via free Wi-Fi at the following places? (Airport)

**Figure C.69:** Manual Code Analysis Module Question 27.1.

**Figure C.70:** Static Code Analysis Module Question 27.1.
• Do you feel secure to access Internet via free Wi-Fi at the following places? (Fast Food Restaurant)

**Figure C.71:** Manual Code Analysis Module Question 27.2.

**Figure C.72:** Static Code Analysis Module Question 27.2.
Do you feel secure to access Internet via free Wi-Fi at the following places? (Bookstore (Barnes and Noble, Books a Million, etc.))

**Figure C.73:** Manual Code Analysis Module Question 27.3.

**Figure C.74:** Static Code Analysis Module Question 27.3.
• Do you feel secure to access Internet via free Wi-Fi at the following places? (Public Library)

**Figure C.75:** Manual Code Analysis Module Question 27.4.

**Figure C.76:** Static Code Analysis Module Question 27.4.
Do you feel secure to access Internet via free Wi-Fi at the following places? (School (university, high school))

**Figure C.77**: Manual Code Analysis Module Question 27.5.

**Figure C.78**: Static Code Analysis Module Question 27.5.
• Do you feel secure to access Internet via free Wi-Fi at the following places? (Religious organization (Church, mosque, etc.))

![Figure C.79: Manual Code Analysis Module Question 27.6.](image)

![Figure C.80: Static Code Analysis Module Question 27.6.](image)
The following questions ask you to identify the type of information that you think are important to keep private. In this context, ‘privacy’ refers to information about yourself that you think should not be accessed without your consent or control. (Gender)

![Figure C.81: Manual Code Analysis Module Question 28.1.](image)

![Figure C.82: Static Code Analysis Module Question 28.1.](image)
• The following questions ask you to identify the type of information that you think are important to keep private. In this context, 'privacy' refers to information about yourself that you think should not be accessed without your consent or control. (Day, Month, Year of Birth)

**Figure C.83**: Manual Code Analysis Module Question 28.2.

**Figure C.84**: Static Code Analysis Module Question 28.2.
The following questions ask you to identify the type of information that you think are important to keep private. In this context, 'privacy' refers to information about yourself that you think should not be accessed without your consent or control. (Zip code)

**Figure C.85**: Manual Code Analysis Module Question 35.

**Figure C.86**: Static Code Analysis Module Question 35.
The following questions ask you to identify the type of information that you think are important to keep private. In this context, ‘privacy’ refers to information about yourself that you think should not be accessed without your consent or control. (Credit card number)

**Figure C.87:** Manual Code Analysis Module Question 28.4.

**Figure C.88:** Static Code Analysis Module Question 28.4.
The following questions ask you to identify the type of information that you think are important to keep private. In this context, 'privacy' refers to information about yourself that you think should not be accessed without your consent or control. (Monthly Income)

**Figure C.89:** Manual Code Analysis Module Question 28.5.

**Figure C.90:** Static Code Analysis Module Question 28.5.
The following questions ask you to identify the type of information that you think are important to keep private. In this context, 'privacy' refers to information about yourself that you think should not be accessed without your consent or control. (Driver's License)

**Figure C.91:** Manual Code Analysis Module Question 28.6.

**Figure C.92:** Static Code Analysis Module Question 28.6.
• The following questions ask you to identify the type of information that you think are important to keep private. In this context, 'privacy' refers to information about yourself that you think should not be accessed without your consent or control. (Photos)

**Figure C.93:** Manual Code Analysis Module Question 28.7.

**Figure C.94:** Static Code Analysis Module Question 28.7.
- The following questions ask you to identify the type of information that you think are important to keep private. In this context, 'privacy' refers to information about yourself that you think should not be accessed without your consent or control. (Year of Birth)

**Figure C.95:** Manual Code Analysis Module Question 28.8.

**Figure C.96:** Static Code Analysis Module Question 28.8.
- The following questions ask you to identify the type of information that you think are important to keep private. In this context, ‘privacy’ refers to information about yourself that you think should not be accessed without your consent or control. (Age)

**Figure C.97:** Manual Code Analysis Module Question 28.9.

**Figure C.98:** Static Code Analysis Module Question 28.9.
The following questions ask you to identify the type of information that you think are important to keep private. In this context, 'privacy' refers to information about yourself that you think should not be accessed without your consent or control. (Home Address)

Figure C.99: Manual Code Analysis Module Question 28.10.

Figure C.100: Static Code Analysis Module Question 28.10.
The following questions ask you to identify the type of information that you think are important to keep private. In this context, ‘privacy’ refers to information about yourself that you think should not be accessed without your consent or control. (Phone number)

**Figure C.101:** Manual Code Analysis Module Question 28.11.

**Figure C.102:** Static Code Analysis Module Question 28.11.
• The following questions ask you to identify the type of information that you think are important to keep private. In this context, ‘privacy’ refers to information about yourself that you think should not be accessed without your consent or control. (Monthly Expenses)

**Figure C.103:** Manual Code Analysis Module Question 28.12.

**Figure C.104:** Static Code Analysis Module Question 28.12.
The following questions ask you to identify the type of information that you think are important to keep private. In this context, 'privacy' refers to information about yourself that you think should not be accessed without your consent or control. (Birthplace)

**Figure C.105:** Manual Code Analysis Module Question 28.13.

**Figure C.106:** Static Code Analysis Module Question 28.13.
The following questions ask you to identify the type of information that you think are important to keep private. In this context, ‘privacy’ refers to information about yourself that you think should not be accessed without your consent or control. (Current Location)

**Figure C.107:** Manual Code Analysis Module Question 28.14.

**Figure C.108:** Static Code Analysis Module Question 28.14.
• Indicate how concerned you are about each of the following possible threats to your security. In this context, ‘security’ refers to a concern about someone or a company being able to potentially harm you (financially, socially or legally) (Private information being used against you later)

Figure C.109: Manual Code Analysis Module Question 29.1.

Figure C.110: Static Code Analysis Module Question 29.1.
• Indicate how concerned you are about each of the following possible threats to your security. In this context, ‘security’ refers to a concern about someone or a company being able to potentially harm you (financially, socially or legally) (Transfer or sale of your identity or private information to other companies)

**Figure C.111:** Manual Code Analysis Module Question 29.2.

**Figure C.112:** Static Code Analysis Module Question 29.2.
• Indicate how concerned you are about each of the following possible threats to your security. In this context, ‘security’ refers to a concern about someone or a company being able to potentially harm you (financially, socially or legally) (Price discrimination (offer best prices to VIP members))

**Figure C.113:** Manual Code Analysis Module Question 29.3.

**Figure C.114:** Static Code Analysis Module Question 29.3.
• Indicate how concerned you are about each of the following possible threats to your security. In this context, ‘security’ refers to a concern about someone or a company being able to potentially harm you (financially, socially or legally) (Identity theft))

**Figure C.115:** Manual Code Analysis Module Question 29.4.

**Figure C.116:** Static Code Analysis Module Question 29.4.
• Indicate how concerned you are about each of the following possible threats to your security. In this context, 'security' refers to a concern about someone or a company being able to potentially harm you (financially, socially or legally) (A store, a company, or a website collects your private information))

**Figure C.117**: Manual Code Analysis Module Question 29.5.

**Figure C.118**: Static Code Analysis Module Question 29.5.
• Indicate how concerned you are about each of the following possible threats to your security. In this context, 'security' refers to a concern about someone or a company being able to potentially harm you (financially, socially or legally) (Knowing your financial situation))

**Figure C.119:** Manual Code Analysis Module Question 29.6.

**Figure C.120:** Static Code Analysis Module Question 29.6.
- The following questions ask about the steps that you may or may not take to maintain your security. How often do you engage in the following behaviors? (Reveal personal information on the Internet or filling out paper forms)

**Figure C.121:** Manual Code Analysis Module Question 30.1.

**Figure C.122:** Static Code Analysis Module Question 30.1.
• The following questions ask about the steps that you may or may not take to maintain your security. How often do you engage in the following behaviors?

(Have multiple email accounts to protect privacy)

Figure C.123: Manual Code Analysis Module Question 30.2.

Figure C.124: Static Code Analysis Module Question 30.2.
The following questions ask about the steps that you may or may not take to maintain your security. How often do you engage in the following behaviors? (Revealing personal information if it will allow the provider to give you better service or price)

**Figure C.125**: Manual Code Analysis Module Question 30.3.

**Figure C.126**: Static Code Analysis Module Question 30.3.
The following questions ask about the steps that you may or may not take to maintain your security. How often do you engage in the following behaviors? (Falsifying (lying) information about yourself on a website or paper forms)

**Figure C.127:** Manual Code Analysis Module Question 30.4.

**Figure C.128:** Static Code Analysis Module Question 30.4.
The following questions ask about the steps that you may or may not take to maintain your security. How often do you engage in the following behaviors? (Carefully read privacy policies)

**Figure C.129:** Manual Code Analysis Module Question 30.5.

**Figure C.130:** Static Code Analysis Module Question 30.5.
The following questions ask about the steps that you may or may not take to maintain your security. How often do you engage in the following behaviors? (Maintain anti-malware applications)

Figure C.131: Manual Code Analysis Module Question 30.6.

Figure C.132: Static Code Analysis Module Question 30.6.
The following questions ask about the steps that you may or may not take to maintain your security. How often do you engage in the following behaviors? (Find out how a company or organization plans to use your identities or private information)

**Figure C.133:** Manual Code Analysis Module Question 30.7.

**Figure C.134:** Static Code Analysis Module Question 30.7.
• The following questions ask about the steps that you may or may not take to maintain your security. How often do you engage in the following behaviors? (Download security patches for your personal computer)

**Figure C.135:** Manual Code Analysis Module Question 30.8.

**Figure C.136:** Static Code Analysis Module Question 30.8.
• Compared to projects in other courses, what features do you like most?

(Please check all that apply)

Figure C.137: Manual Code Analysis Module Question 31.
Figure C.138: Static Code Analysis Module Question 31.
• Whom will you educate on the secure coding that you have learned? (Please check all that apply)

**Figure C.139:** Manual Code Analysis Module Question 32.

**Figure C.140:** Static Code Analysis Module Question 32.
Pervasive Security and Privacy (PSP) Website and projects help me to secure my devices

**Figure C.141:** Manual Code Analysis Module Question 36.1.

**Figure C.142:** Static Code Analysis Module Question 36.1.
Pervasive Security and Privacy (PSP) Website and projects help me to securely write code

Figure C.143: Manual Code Analysis Module Question 36.2.

Figure C.144: Static Code Analysis Module Question 36.2.
• Pervasive Security and Privacy (PSP) Website and projects I will recommend the website and projects to others

Figure C.145: Manual Code Analysis Module Question 36.3.

Figure C.146: Static Code Analysis Module Question 36.3.
• Pervasive Security and Privacy (PSP) Website and projects help me to securely use the Internet

![Figure C.147: Manual Code Analysis Module Question 36.4.](image)

![Figure C.148: Static Code Analysis Module Question 36.4.](image)
• Pervasive Security and Privacy (PSP) Website and projects like the security projects

![Manual Code Analysis Module Question 36.5.](image1)

**Figure C.149:** Manual Code Analysis Module Question 36.5.

![Static Code Analysis Module Question 36.5.](image2)

**Figure C.150:** Static Code Analysis Module Question 36.5.
• Have you presented any of the CWE’s that we presented during this SECURE CODING PRACTICES RESEARCH STUDY as part of the CS-485/CS-585 course? Mark all CWE’s that apply.

**Figure C.151:** Manual Code Analysis Module Question 37.

**Figure C.152:** Static Code Analysis Module Question 37.
- How would you rate your programming experience level for software development?

**Figure C.153:** Manual Code Analysis Module Question 38.

**Figure C.154:** Static Code Analysis Module Question 38.
Appendix D: Secure Coding Exercises

The following are the CWEs included in this eye tracking secure coding research study.

D.1 CWE-311: Missing Encryption of Sensitive Data

Problem: We sometimes need to store or transmit sensitive or critical information. For example, in ShareAlbum, we must store users’ names, email, information, and passwords in a database. If this information is in plaintext, the user’s login information will be exposed if an attacker compromises their computer. So, we need to use an encrypted algorithm to encrypt this information.

```php
function user_register($email, $fname, $lname, $password, $about) {
    $email = mysql_real_escape_string($email);
    $fname = mysql_real_escape_string($fname);
    $lname = mysql_real_escape_string($lname);
    $about = mysql_real_escape_string($about);
    $query = "INSERT INTO user VALUES ('', '$fname', '$lname', '$email', '$password', '$about')";
    $result = mysql_query($query) or die($query."<br/>" . mysql_error());
    return mysql_insert_id();
}
```
Solution: Encrypting these credentials by using an encryption algorithm is essential. The encryption algorithms we suggest to use are scrypt, bcrypt and PBKDF2.

```php
function user_register($email, $fname, $lname, $password, $about) {

    $email = mysql_real_escape_string($email);
    $fname = mysql_real_escape_string($fname);
    $lname = mysql_real_escape_string($lname);
    $about = mysql_real_escape_string($about);

    // Create a hash of the password to store
    $hashToStoreInDb = password_hash($password, PASSWORD_BCRYPT);

    // Example of how to check for lookup purposes
    // $isPasswordCorrectFlag = password_verify($password, $existingHashFromDb);

    $query = "INSERT INTO user VALUES ('', '$fname', '$lname', '$email', '$hashToStoreInDb', '', '$about')";
    $result = mysql_query($query) or die($query."<br/>" . mysql_error());
    return mysql_insert_id();
}
```
D.2 CWE-434: Unrestricted Upload of File with Dangerous Type

Problem: In ShareAlbum, users could upload video and image files onto the server. If an extension of these files was not restricted, attackers could upload or transfer files of dangerous types, such as .php or .exe, that can be automatically processed within the product’s environment.

ShareAlbum Source Code:

```php
function upload_image($image_temp,$image_name,$album_id,$location
                    ,$about,$description){

    $album_id=(int)$album_id;
    mysql_query("INSERT INTO photo VALUES ('','$description','
                    $location','".$_SESSION['user_id'].'','$album_id',
                    FROM_UNIXTIME(UNIX_TIMESTAMP()),'$about',' '$image_name')");
    $image_id=mysql_insert_id();
    $image_file=$image_id.'.'.$image_name;
    echo $image_temp,'<br />',$image_name,'<br />',$album_id,'<br />

    move_uploaded_file($image_temp,'uploads/').'/'.$album_id.'/'.
                    $image_file);
}
```

Solution: Define the allowed extensions of upload files and limit the size of upload files.

```php
function upload_image($image_temp,$image_name,$album_id,$location
                    ,$about,$description){

    $album_id=(int)$album_id;
```

365
$allowed_ext = array('jpg', 'jpeg', 'png', 'gif');
$image_parts = explode('.', $image_name);
$image_ext = strtolower(end($image_parts));
$image_size = getimagesize($image_name);
$errors = [];

if (empty($image_name) || empty($album_id)) {
    $errors[] = 'something is missing';
}

if (in_array($image_ext, $allowed_ext) === false) {
    $errors[] = 'File type not allowed';
}

if ($image_size > 2097152) {
    $errors[] = 'Maximum file size is 2mb';
}

if (album_check($album_id) === false) {
    $errors[] = 'Could not upload to that album';
}

if (!empty($errors)) {
    foreach ($errors as $error) {
        echo $error, '<br />
    }
}
else {

D.3 CWE-862: Missing Authorization

Problem: Users are often assigned different privileges. In the ShareAlbum project, we separate photos and albums into private and public views. We want private photos to be only viewable to the owner, whereas public photos are available to everyone registered. The owner of a photo or album is the only one who can delete the photo or album. If the authorization is incorrect, an attacker may take advantage of it and execute unauthorized actions (coded CWE-862). For instance, the following software code is used to delete all albums by album_id.

We find out that by using $album_id=(int)$album_id, the programmer prevents the SQL injection error (CWE-89), but we did not check the user_id to make sure that the user is authenticated to delete the album that they created.

ShareAlbum Source Code:
function delete_album($album_id) {
    $album_id = (int) $album_id;

    $query_1 = "DELETE FROM album WHERE albumID=$album_id";
    $result_1 = mysql_query($query_1) or die($query_1."<br/>" . mysql_error());

    $query_2 = "DELETE FROM photo WHERE albumID=$album_id";
    $result_2 = mysql_query($query_2) or die($query_2."<br/>" . mysql_error());
}

Solution: Check the user ID to make sure that the logged-in user owns the album. For example, in the following code, check the user ID by using userID="SESSION['user_id'] in the delete query. $_SESSION['user_id'] stored the user ID who logged in now. Missing Authorization or improper authorization happens at a high frequency and has a high attacker awareness. This error is not too complicated to detect and remediate. Designers and developers should divide applications into anonymous, regular, privileged, and administrative areas. Furthermore, one may use role-based access control (RBAC) to enforce the roles at the appropriate boundaries.

function delete_album($album_id) {
    $album_id = (int) $album_id;

    $query_1 = "DELETE FROM album WHERE albumID=$album_id AND userID=" . $_SESSION['user_id'] ";";
D.4 CWE-22: Improper Limitation of a Pathname to a Restricted Directory (‘Path Traversal’)

Problem: In the below code, the external input is used to construct a pathname that is intended to identify a file or directory. However, the code does not correctly neutralize particular elements within the pathname which can cause the pathname to resolve to a location that is outside of the restricted directory. The below code demonstrates the unrestricted upload of a file with a php move_uploaded_file() and a path traversal vulnerability. The code doesn’t check the filename. So, attackers can use “../” sequences to write to files outside of the intended directory. Together with the CWE-434 error, attackers may be able to inject malicious files into the target directory, which leads to a wide variety of consequences from malicious code execution to system crashes.

ShareAlbum Source Code:
function upload_image($image_temp, $image_name, $album_id, $location, $about, $description) {
    $album_id = (int) $album_id;
    mysql_query("INSERT INTO photo VALUES ("', $description, ","', $location, ","'.$_SESSION['user_id']."', $album_id, "," FROM_UNIXTIME(UNIX_TIMESTAMP())), $about, ', $image_name");
    $image_id = mysql_insert_id();
    $image_file = $image_id .'.' . $image_name;
    echo $image_temp, '<br />', $image_name, '<br />', $album_id, '
→ <br />';
    move_uploaded_file($image_temp, 'uploads/' . $album_id . '/' . $image_file);
}

Solution: Check the image file’s filename to ensure it does not include any unexpected characters. For example, only allow a single “.” in the filename or separate filenames and extensions that are assigned to different variables such as $filename and $ext. Do not allow any “.” or “/” in the $filename.
$image_filenames_2 = array_map('strtolower',
    $image_filename_parts_2);

$errors = [];

if (empty($image_name) || empty($album_id)) {
    $errors[] = 'something is missing';
}

foreach ($image_filenames_2 as $filename_part) {
    if (strpos($filename_part, '.') !== false) {
        $errors[] = 'Cannot change to unrestricted directories using .';
    }
}

foreach ($image_filenames_1 as $filename_part) {
    if (strpos($filename_part, '/') !== false) {
        $errors[] = 'Cannot change to unrestricted directories using /';
    }
}

if (album_check($album_id) === false) {
    $errors[] = 'Could not upload to that album';
}

if (!empty($errors)) {
    foreach ($errors as $error){
        echo $error;
    }
}
D.5 CWE-443: HTTP Response Splitting

Problem: An attacker can inject arbitrary headers into the HTTP response header. This can be abused for a redirect when injecting a "Location:" header or help within a session fixation attack when the "SetCookie:" header is added. Additionally, the HTTP response can be overwritten, and JavaScript can be injected, leading to Cross-Site Scripting attacks. The mitigation strategy is to update PHP to prevent header injection or implement a whitelist.

ShareAlbum Source Code:

```php
if (!check_if_user_isLogged_in())
```
```php
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    if (isset($_POST['markAlbumTagForDeletion']) &&
        $_POST['markAlbumTagForDeletionHidden'] === $tagID) {
        mark_tag_for_deletion($_POST['markAlbumTagForDeletionHidden'], "video_album");
        $_SESSION['albumID'] = $albumID;
        header('Location: video.album.viewing.album.php');
        exit();
    }
}

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    if (isset($_POST['markVideoCommentForDeletion']) &&
        $_POST['markVideoCommentForDeletionHidden'] ===
        $_POST['video_commentID']) {
        mark_comment_for_deletion($_POST['markVideoCommentForDeletionHidden'], "video");
        $_SESSION['albumID'] = $albumID;
        header('Location: video.album.viewing.album.php');
```
Solution: Update PHP software to prevent header injection or implement a whitelist of available sites to help remediate this vulnerability. Limit the use of user-specific input for HTTP-response redirection. No update to the source in ShareAlbum is required for this specific example. This is because the above headers function calls are not getting user input from ‘$_GET’; therefore, no vulnerability exists with the current implementation. This was a false positive incorrectly identified by RIPS.

D.6 CWE-73: External Control of File Name or Path

Problem: An attacker might write to arbitrary files or inject arbitrary code into a file. User-tainted data is used when creating the file name that will be opened or the string that will be written to the file. An attacker can try to write arbitrary PHP code in a PHP file to compromise the server entirely.

ShareAlbum Source Code:

```php
function add_a_new_file_to_album($file_user_typed_name,
                                  $file_user_location,$file_user_description,$albumID,
                                  $album_data_struct,$file_user_file_name,
                                  $file_user_file_extension,$actual_file)
{

  $albumID = intval($albumID);
```
$query = build_sql_add_a_new_file_to_album(
    $file_user_typed_name, $file_user_location, 
    $file_user_description, $albumID, $album_data_struct, 
    $file_user_file_name, $file_user_file_extension);

$result = mysqli_query(Database::$connection, $query) or die($query."<br/>
        
    $value = mysqli_insert_id(Database::$connection);

mkdir('..//Data/.'.$album_data_struct.'_album/AlbumID'.
    '.'.FileID'.$.value, 0744, true);

// mkdir('..//Data/thumbs/.'.$album_data_struct.'_album/AlbumID'.
    '.'.$albumID.'//FileID'.$.value, 0744, true);

$storedLocation = '..//Data/.'.$album_data_struct.'_album/
    AlbumID'.$albumID.'//FileID'.$value.'//'.$file_user_file_name
    ;

move_uploaded_file($actual_file, $storedLocation);

//generate_thumbnail($actual_file, $storedLocation);

$query = build_sql_update_file_stored_location($value,
    $storedLocation, $album_data_struct);

$result = mysqli_query(Database::$connection, $query) or die($query."<br/>
        
    $value2 = ($result === true) ? true : false;
Solution: The mitigation strategy is to build a white list for positive file names. Do not only limit the file name to specific paths or extensions. Use a white list with arrays or regular expressions (e.g., alphanumeric) or rename the file to ensure custom paths that are safe are utilized.

```php
function add_a_new_file_to_album($file_user_typed_name, $file_user_location, $file_user_description, $albumID, $album_data_struct, $file_user_file_name, $file_user_file_extension, $actual_file)
{
    $albumID = intval($albumID);
    
    $allowed_ext = array('jpg', 'jpeg', 'png', 'gif', 'mp4', 'mov', 'avi');
    $image_filename_parts_1 = explode('.', $image_name);
    $image_filename_parts_2 = explode('/', $image_name);
    $image_filenames_1 = array_map('strtolower', $image_filename_parts_1);
    $image_filenames_2 = array_map('strtolower', $image_filename_parts_2);
    $errors = [];

    foreach ($image_filenames_2 as $filename_part) {
        if (strpos($filename_part, '.') !== false) {
            // Code snippet
        }
    }

    return $value;
} // end add_a_new_file_to_album
```
foreach ($image_filenames_1 as $filename_part) {
    if (strpos($filename_part, '/') !== false) {
        $errors[] = 'Cannot change to unrestricted directories using /';
    }
}

if (in_array($file_user_file_extension, $allowed_ext) === false) {
    $errors[] = 'File type not allowed';
}

$value = false;
if (!empty($errors)) {
    foreach ($errors as $error) {
        echo $error, '<br '/>;
    }
} else {
    $file_user_file_name = 'whitelisted_path_'.
    $file_user_file_name;
    $query = build_sql_add_a_new_file_to_album( 
    $file_user_typed_name, $file_user_location,

$file_user_description,$albumID,$album_data_struct,
$file_user_file_name,$file_user_file_extension);
$result = mysqli_query(Database::$connection, $query) or
die($query."<br/>".$query ."\n".mysqli_error(Database::$connection))
);
$value = mysqli_insert_id(Database::$connection);

mkdir('../Data/'. $album_data_struct.'_album/AlbumID'.
$albumID.'/FileID'.$value, 0744, true);
// mkdir('../Data/thumbs/'. $album_data_struct.'_album/
AlbumID'.$albumID.'/FileID'.$value, 0744, true);

$storedLocation = '../Data/'. $album_data_struct.'_album/
AlbumID'.$albumID.'/FileID'.$value.'/'.$file_user_file_name
;
move_uploaded_file($actual_file,$storedLocation);
//generate_thumbnail($actual_file,$storedLocation);

$query = build_sql_update_file_stored_location($value,
$storedLocation,$album_data_struct);
$result = mysqli_query(Database::$connection, $query) or
die($query."<br/>".$query ."\n".mysqli_error(Database::$connection))
);
$value = ($result === true) ? true : false;
}

return $value;
} // end add_a_new_file_to_album
D.7 CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

Problem: An attacker might embed a fake login screen to steal user passwords. One potential way to exploit Cross-Site Scripting is to attack the GET Method used in passing variables or user input between web pages. With this security vulnerability, an attacker might execute arbitrary HTML/JavaScript Code in the client’s browser context. User-tainted data is embedded into the HTML output by the application and rendered by the user’s browser, thus allowing an attacker to embed and render malicious code. Preparing a malicious link will lead to an execution of this malicious code in another user’s browser context when clicking the link. This can lead to local website defacement, phishing or cookie stealing, and session hijacking. In ShareAlbum, an attacker may add lines in the URL for the variable retrieved from ‘$_GET’ to embed a fake login box after submitting a new notification to another user.

ShareAlbum Source Code:

```php
if( isset($_GET['submitNewNotification']))
{
    $message = $_GET['message'];
    $userID = $_GET['userID'];
    $errors = array();

    if(empty($message))
    {
        $errors[] = 'A blank notification is not allowed';
    }
}"
```
Solution: To mitigate this threat, you should encode all user-tainted data with PHP built-in functions before embedding the data into the output to ensure that the user provided valid input data.

```php
if (isset($_GET['submitNewNotification']))
```
```php
{
    $message = $_GET['message'];
    $userID = $_GET['userID'];
    $errors = array();

    // Define a regular expression pattern to match HTML tags
    $pattern = '/<[^>]*>/';

    if (empty($message))
    {
        $errors[] = 'A blank notification is not allowed';
    }
    if (preg_match($pattern, $message))
    {
        $errors[] = 'The notification message contains HTML code tags that are not allowed for a notification';
    }

    // output any errors to the user
    if (!empty($errors))
    {
        echo '<h3> ERROR: <br/>
        
        foreach($errors as $error)
        {
            echo $error, '<br />
        }
        echo '</h3>';
    }
    else
```
{  
    // Save the notification in the database  
    $value = create_notification_to_user($message, $userID);  
    if ($value)  
    {  
        echo '<h3> Successfully sent notification of : ', $message, '</h3>';  
    }  
}
Appendix E: Secure Coder Behaviors

E.1 High-level Summary of Secure Coders’ Behaviors

The following presents a high-level summary per secure coder for working on the CWE-73 problem. The specific names of our secure coders have been redacted to protect their identity and their performance results as secure coders.

*Secure Coder S14* took a total of 1113.6 seconds during their session. This secure coder spent 247 seconds on general information about secure coding, which makes up 22.2% of the total 1113 seconds of the session. Their focus on CWE-73 lasted for 624 seconds, of which 46 seconds were dedicated to learning content, 574 seconds to examining source code and RIPS, and the remaining 4 seconds to answer the question if RIPS correctly determined the code to have a security flaw. This student spent 56.0% of their session on CWE-73 and made 112 mouse clicks. Their answer to the question was correct.

*Secure Coder S2* took a total of 1147.3 seconds to generate their eye gaze recording. This secure coder spent 348 seconds or 30.3% of the session learning general information on secure coding. Their time on CWE-73 totaled 457 seconds, with 43 seconds for learning, 403 seconds for examining source code and RIPS, and 11 seconds to answer a question. This amounted to 39.8% of their session total session. The student made 57 mouse clicks for CWE-73 but answered the question incorrectly.
Secure Coder S30 spent 1764.4 seconds on their recording session. This secure coder dedicated 590 seconds (33.4% of the session) to general secure coding information learning. Their engagement with CWE-73 spanned 747 seconds, subdivided into 107 seconds for learning content, 624 seconds for examining source code and RIPS, and 16 seconds for answering a question, making up 42.3% of the total session. Their answer was incorrect despite making 99 mouse clicks while working on the CWE-73 problem.

Secure Coder S28 spent 2578.7 seconds in total for their recording session. This secure coder dedicated 1107 seconds to general information about secure coding, accounting for 42.9% of their session. Their time focused on CWE-73 summed up to 770 seconds, broken down into 212 seconds for learning content, 543 seconds for examining source code and RIPS, and 15 seconds to answer the question if RIPS correctly classified the four security flaws for CWE-73. This problem consumed 29.9% of their entire session. Even with 42 mouse clicks, their answer was incorrect.

The next session was completed by Secure Coder S16. The total duration was 2373.2 seconds. This secure coder spent 1047 seconds (44.1% of the session) on general secure coding information. For CWE-73, the coder took 793 seconds, with 176 seconds spent on learning, 613 seconds on source code and RIPS examination, and a mere 4 seconds to answer the question. The student work on CWE-73 covered 33.4% of their recording session. The student recorded 80 mouse clicks while working CWE-73 and answered the question correctly.
Secure Coder S3 spent a total recording duration of 1898.9 seconds. Their time spent on general secure coding information was 873 seconds, which represented 46.0% of the session. Their attention to CWE-73 lasted 578 seconds: 144 seconds for learning, 427 seconds for examining source code and RIPS, and 7 seconds for answering a question. The student dedicated 30.4% of their recording session to this problem. Interestingly, despite no mouse clicks being recorded, their answer was correct.

Secure Coder S31 took a total of 2272.9 seconds. The student allocated 1168 seconds to general information on secure coding, representing 51.4% of the session. For CWE-73, the coder spent 779 seconds: 251 seconds on learning content, 516 seconds on examining the source code and RIPS, and 12 seconds to answer if the RIPS tool correctly determined that CWE-73 is a problem for the ShareAlbum code base. Working CWE-73 took up 34.3% of the session. The student made 60 mouse clicks, but unfortunately, their answer was incorrect.

Secure Coder S29 spent a total of 3410.1 seconds. This secure coder dedicated 1061 seconds, or 31.1% of the session, to general secure coding information. Their focus on CWE-73 lasted 844 seconds: 177 seconds for learning, 647 seconds for examining source code and RIPS, and 20 seconds for answering the question. Working CWE-73 took 24.8% of their session. With 51 mouse clicks, while working CWE-73, the student’s answer was incorrect.

Secure Coder S23 took a total of 2413.3 seconds. This secure coder invested 832 seconds in general secure coding information, which amounted to 34.5% of the session. Their attention to CWE-73 took up 807 seconds, divided into 119
seconds for learning, 679 seconds for source code and RIPS examination, and 9 seconds for answering the question. Their working this CWE covered 33.4% of their total recording session. The student recorded 37 mouse clicks and responded to the question correctly.

Secure Coder S9 had an extensive 4168 seconds for their recording session. Out of this, the coder devoted 1136 seconds to secure coding’s general information, which made up 27.3% of their session. Their interaction with CWE-73 was particularly notable, consuming 2388 seconds. This was broken down into 134 seconds of learning, a significant 2229 seconds scrutinizing the source code and RIPS, and 25 seconds answering the question if the RIPS tool correctly classified the source code as having a weakness for CWE-73. This problem occupied 57.3% of the session. Their comprehensive review resulted in a correct answer, accompanied by 204 mouse clicks.

This recording featured Secure Coder S26. Their session lasted 1101.1 seconds, with 318 seconds (or 28.9% of the time) dedicated to general secure coding information. Their attention to CWE-73 spanned 463 seconds, divided into 107 seconds of learning, 343 seconds reviewing source code and RIPS, and 13 seconds answering the associated question. This represented 42.0% of their session. With 55 mouse clicks, the student successfully answered the question correctly.

Secure Coder S32 spent a total of 2072.6 seconds. This secure coder allocated 485 seconds to general information on secure coding, equating to 23.4% of their time. For the topic of CWE-73, the coder spent 815 seconds, which was
split between 70 seconds for learning, 738 seconds for examining source code and RIPS, and 7 seconds for question answering. This subject consumed 39.3% of the session. The student made 78 mouse clicks but, unfortunately, did not answer the question correctly.

Secure Coder S10 had a significant duration of 4093.6 seconds. This secure coder dedicated a substantial 1434 seconds to understanding the general aspects of secure coding, accounting for 35.0% of the entire session. In regards to the topic of CWE-73, the coder allocated 1322 seconds, subdivided into 231 seconds of learning, an extensive 1079 seconds for examining the source code and RIPS, and a brief 12 seconds for answering the associated question. This component took up 32.3% of their session. Their meticulous review culminated in a correct response and was accompanied by 109 mouse clicks.

This session features Secure Coder S21, who spent 1733.9 seconds, with a notable 780 seconds (or 45.0% of the time) dedicated to the general concepts of secure coding. Their interactions with CWE-73 were concise, consuming 468 seconds. This comprised 91 seconds of learning, 367 seconds exploring source code and RIPS, and 10 seconds for question answering, which collectively represented 27.0% of the session. Their attentive approach resulted in a correct answer, along with 62 mouse clicks.

Secure Coder S20 spent a total of 3416.8 seconds. Interestingly, the coder only spent 407 seconds on general secure coding information, which made up just 11.9% of the session. However, their attention to CWE-73 was pronounced, totaling 1287 seconds. This time was broken down into 77 seconds for learning,
a prominent 1206 seconds scrutinizing the source code and RIPS, and a swift 4 seconds answering the question if the RIPS tool correctly classified the threat, which together occupied 37.7% of the session. Their comprehensive review, along with 115 mouse clicks, ensured their correct answer to the question.

Secure Coder S12 studied for a duration of 1497.8 seconds. This secure coder dedicated 247 seconds, or 16.5% of their session, to the general information on secure coding. Their focus on CWE-73 was typical, with 732 seconds spent on it. This was divided into 40 seconds of learning, 687 seconds examining the source code and RIPS, and 5 seconds for question answering, making up 48.9% of the session. The student’s diligent approach led to a correct response, which was marked by 28 mouse clicks.

Secure Coder S34 spent a total of 1235.1 seconds. This secure coder allocated 400 seconds (32.4% of the session) to general secure coding information. Their interaction with CWE-73 amounted to 606 seconds, segmented into 92 seconds for learning, 512 seconds for source code and RIPS examination, and a brief 2 seconds for answering the question, cumulatively accounting for 49.1% of their study time. Despite their effort, the student answered incorrectly and made no mouse clicks.

Secure Coder S8 spent a total recording duration of 1793.1 seconds. This secure coder dedicated 363 seconds (or 20.2% of the session) to the broad aspects of secure coding. Regarding CWE-73, the coder spent 885 seconds, distributed into 285 seconds of learning, 593 seconds exploring the source code and RIPS,
and 7 seconds for the question’s response. This detailed review, making up 49.4% of the session, along with 66 mouse clicks, resulted in a correct answer.

Secure Coder S5 spent a duration of 1307.1 seconds. This secure coder spent 489 seconds, making up 37.4% of their session, on general information about secure coding. For CWE-73, the coder invested 497 seconds, divided as 64 seconds on learning content, 415 seconds on source code and RIPS examination, and 18 seconds on answering the question. This represented 38.0% of their entire session. Their efforts paid off with a correct answer and were accompanied by 111 mouse clicks.

Secure Coder S1 had a total recording duration of 1678.5 seconds. Their attention to general secure coding was limited, dedicating only 180 seconds or 10.7% of the session. For CWE-73, the coder spent 588 seconds, which included 15 seconds of learning, 568 seconds reviewing the source code and RIPS, and 5 seconds answering the associated question. This focus covered 35.0% of their session. Unfortunately, despite their efforts and 80 mouse clicks, their answer was incorrect.

Secure Coder S7 spent 1344.5 seconds studying. This secure coder dedicated 566 seconds (or 42.1% of the session) to general secure coding topics. Their interaction with CWE-73 amounted to 496 seconds, broken down into 99 seconds for learning, 391 seconds for code and RIPS analysis, and 6 seconds for the question’s response. This detailed interaction made up 36.9% of their total study time, and despite making no mouse clicks, the student secured a correct answer.
Secure Coder S24 studied for a total duration of 1312.9 seconds. This secure coder dedicated 567 seconds or 43.2% of their session to general secure coding information. Their engagement with the CWE-73 problem lasted 414 seconds, which was further broken down into 91 seconds for learning content, 322 seconds for examining the source code and RIPS, and a mere 1 second for answering the relevant question. This problem covered 31.5% of their entire session. Despite their efforts, the student’s answer was incorrect, and no mouse clicks were recorded during the session.

Secure Coder S27 dedicated 2242.5 seconds to the session. Of this, 828 seconds or 36.9% were spent on general secure coding knowledge. For CWE-73, the coder invested 827 seconds, split into 131 seconds for learning, 687 seconds on source code and RIPS analysis, and 9 seconds for answering the associated question. This represented 36.9% of their study time. Even with 73 mouse clicks, the student’s answer turned out to be incorrect.

Secure Coder S18 spent 1772.9 seconds in the study session. This secure coder allocated 512 seconds, which is 28.9% of the time, to general secure coding topics. Their interaction with CWE-73 spanned 614 seconds, with 230 seconds devoted to learning, 377 seconds to code and RIPS assessment, and 7 seconds for the question’s response. This represented 34.6% of their entire session. Regrettably, despite 53 mouse clicks, the student’s answer was incorrect.

Secure Coder S15 engaged for a total recording duration of 1251.3 seconds. Out of this, the student allocated 403 seconds or 32.2% of the session to general secure coding principles. When studying the CWE-73 problem, the coder spent
a cumulative 567 seconds, dividing their time between 84 seconds of learning content, 477 seconds scrutinizing source code and RIPS, and 6 seconds to address the related question. This problem consumed 45.3% of their session. Despite their effort and 54 mouse clicks, the student responded incorrectly.

*Secure Coder S17* worked their secure coding session for 1891.5 seconds, with 831 seconds (or 43.9% of the total time) dedicated to general secure coding data. Their interaction with the CWE-73 subject lasted 501 seconds. This was further detailed as 134 seconds of learning, 353 seconds of code and RIPS analysis, and 14 seconds for answering the topic-specific question. CWE-73 occupied 26.5% of their study time. Even after 26 mouse actions, the student’s answer was unfortunately incorrect.

*Secure Coder S6* spent a total recording duration of 1209.7 seconds. Here, the coder set aside 444 seconds (or 36.7% of the session) for overarching secure coding themes. For the CWE-73 problem, *Secure Coder S6* used 400 seconds, consisting of 75 seconds of content reading, 321 seconds reviewing code and RIPS, and a mere 4 seconds for the related question’s response. This took 33.1% of their entire session. With 41 mouse clicks, this student’s answer was correct.

*Secure Coder S33* dedicated a session of 1849.7 seconds. A considerable chunk of this time, 910 seconds or 49.2% of the session, was focused on absorbing general secure coding knowledge. Regarding the specific topic of CWE-73, the student committed 742 seconds. This time was divided into 198 seconds of content learning, 536 seconds for analyzing source code and RIPS, and 8 seconds to work
on the pertinent question. In all, the CWE-73 topic took up 40.1% of the student’s session. However, even after 131 mouse clicks, their answer was incorrect.

**Secure Coder S19** spent a total recording duration of 1946.1 seconds. The majority of their time, 1067 seconds (54.8% of the total), was dedicated to overarching secure coding concepts. Their study of the CWE-73 subject lasted for 605 seconds, which was broken down into 224 seconds of content engagement, 350 seconds for code and RIPS scrutiny, and a longer 31 seconds to answer the related question. This problem occupied 31.1% of their recording duration. Despite their dedication and 47 mouse interactions, their efforts did not pay off, as their response was incorrect.

**Secure Coder S11** had a duration of 1803.7 seconds. Here, the coder allocated 732 seconds (or 40.6% of the session) to general secure coding lessons. Delving into the CWE-73 area, **Secure Coder S11** used 421 seconds, splitting it between 77 seconds for content understanding, 331 seconds reviewing code and RIPS, and 13 seconds for addressing the associated question. This subject represented 23.3% of their study span. However, in spite of the effort and 80 mouse actions, the student’s answer was incorrect.

**Secure Coder S13** had a study session for a total of 1416.4 seconds. This secure coder dedicated a notable portion of this time, 578 seconds or 40.8% of the entire session, to understanding the general concepts of secure coding. For the specific topic of CWE-73, the student spent 482 seconds. This time was further categorized into 70 seconds for reading the learning content, 402 seconds for closely examining source code and RIPS associated with CWE-73, and a
duration of 10 seconds for answering the relevant question. The CWE-73 topic accounted for 34.0% of the student’s entire session. The student had 46 mouse clicks during the CWE-73 problem, and their answer was correct.

E.2 Secure Coders’ Behaviors When Reading Learning Content for CWE-73

The following presents a detailed summary per secure coder for reading the learning content associated with CWE-73. The specific names of our secure coders have been redacted to protect their identity and their performance results as secure coders.

Secure Coder S14 dedicated time to reading the introductory paragraphs and gaining insights into how attackers might exploit vulnerabilities. However, there was a notable omission in their learning path; the coder neglected the paragraph discussing mitigation techniques, which implies a potential gap in their knowledge regarding preventive strategies. Furthermore, their approach towards the RIPS tool and the source code with possible security flaws suggests a focus on tools and practical application. An intriguing observation is their choice to read the multiple-choice options prior to the foundational question sentence, indicating a possible strategy of gauging answer options before fully understanding the question’s context.

Secure Coder S2 demonstrated a varied approach towards the learning content, selectively absorbing material that encompasses both the threat and preventive measures. Specifically, the coder read information from the paragraphs
about the introduction, potential utilization by attackers, and mitigation strategies, thereby ensuring a balanced understanding of both problem and solution aspects. A potential learning gap, however, emerged in their decision to skip the paragraph concerning the RIPS tool and related vulnerable source code.

Secure Coder S30 adopted a meticulous and sequential approach towards the learning content. This secure coder ensured comprehensive coverage by reading all the paragraphs in a top-down manner, reflecting a systematic strategy and perhaps an intent to absorb all available information. A notable point of focus was their lengthy concentration on the filename that supposedly contains vulnerabilities, as indicated by the 0.776 seconds fixation spent on it.

Secure Coder S16 adopted a systematic approach, ensuring complete content coverage by reading all the paragraphs from start to finish. Their methodology was marked by a particular emphasis on the paragraph detailing mitigation strategies. This student read the mitigation section multiple times before progressing further, highlighting their intent to deeply comprehend or clarify potential measures to counter vulnerabilities.

Secure Coder S3 embarked on their learning journey by sequentially reading the initial paragraphs that introduce CWE-73, outline the attacker’s strategy, and detail the mitigation strategy. A notable shift in their reading behavior is observed at the 95-second mark. The student revisited the first three paragraphs, however, with a more rapid reading pace. This could suggest a few possibilities: the coder might have felt the need to reinforce their understanding, or Secure Coder S3 might have been skimming the content to locate specific details or to
review the content in anticipation of subsequent questions. The difference in reading speed on their second pass might indicate either a growing familiarity with the content or a strategic skim to highlight critical points.

Secure Coder S31 demonstrated a comprehensive, repetitive, and meticulous approach toward the learning material. The student read the content in a top-down fashion, not once but twice, spending roughly the same duration on each pass. This consistent time investment suggests a deliberate attempt to reinforce comprehension or ensure no details were overlooked. Their behavior exudes thoroughness and possibly a methodical learning technique. Their third, targeted revisit to the mitigation strategy paragraph is a standout observation.

Secure Coder S23 displayed behavior that was not in line with active engagement with the learning material. Instead of reading the content provided, the student’s focus was consistently diverted to the right side of the monitor. The nature of these repeated fixations to the side suggests the possibility of distraction, misunderstanding, or perhaps technical issues. Whatever the reason, it’s clear that this student did not appear to actively engage with the learning content, which could have implications for their understanding and performance on the topic of CWE-73.

Secure Coder S26 demonstrated a systematic and comprehensive approach to the learning content. This secure coder began by reading all the content sequentially from the top down, ensuring that all information was initially covered. What stands out in their behavior is a revisit to the section on the mitigation strategy, suggesting an emphasis or need for clarity on this particular aspect.
This repeated focus might indicate that the student either sees the importance of understanding mitigation strategies or that the coder needs additional time to comprehend the content in this section thoroughly.

Secure Coder S10’s engagement with the content reveals a more iterative and repetitive reading pattern. Starting with the introduction to CWE-73, followed by the attacker’s strategy, the coder frequently went back to reread sections before advancing. This recursive pattern, especially returning to the introductory paragraph on CWE-73 multiple times, might suggest a need for reinforcement or a methodical approach to ensure complete comprehension. Additionally, the student’s behavior highlights their deliberate pace, taking time to read paragraphs several times and thoroughly reviewing the questions in detail. This deep engagement with the content could reflect a meticulous learning style or perhaps the complexity and importance the coder assigns to CWE-73.

Secure Coder S20 commenced by sequentially reading the introduction to CWE-73 and the attacker’s strategy. However, by 23 seconds, the student advanced to the mitigation strategy and soon after to the section on the RIPS tool and related source code to examine. Interestingly, the coder circled back at the 42-second mark to review the mitigation method and revisited the paragraph about the RIPS tool.

Secure Coder S12 demonstrated a linear and systematic approach, reading all the paragraphs sequentially from the start. However, there was a marked discrepancy in their engagement towards the end. This secure coder did not thoroughly review the last paragraph detailing the specific source code file linked
to the CWE-73 security flaw. This potential oversight may indicate either a pre-existing familiarity with the subject or a lapse in attention.

*Secure Coder S24* initiated their reading journey with a straightforward top-down approach, covering the introduction, attacker’s strategy, and mitigation measures within the first 52 seconds. Notably, in the subsequent 16 seconds, the student rapidly revisited the first three paragraphs, emphasizing a reinforcement strategy. This behavior might suggest a desire to confirm or strengthen their understanding before progressing further into the content.

*Secure Coder S18* began by sequentially reading the first two paragraphs, revisiting the second paragraph for a brief moment before proceeding to the third and fourth paragraphs. Notably, at 138 seconds, the coder returned to the very beginning. However, this second pass was distinct from the first, characterized by a swifter reading pace and selective skipping of words. This pattern suggests a combination of thoroughness on the first pass and a faster revision, possibly for reinforcement or to locate specific information.

*Secure Coder S15* spent the initial 50 seconds methodically reading all four paragraphs. After this, the coder exhibited a particular interest in the second paragraph, revisiting it to delve deeper into how an attacker might exploit the security flaw in the source code. The student also displayed a top-down reading pattern when engaging with the question section, indicating systematic engagement.

*Secure Coder S6* showcased a direct and systematic approach, reading the first three paragraphs in a linear fashion. Their focus was predominantly on
understanding the vulnerability, the attacker’s technique, and the strategies for its mitigation.

E.3 Secure Coders’ Behaviors When Working Exercise for CWE-73

The following presents a detailed summary per secure coder for working the exercise associated with CWE-73. This includes using the RIPS static code analysis tool and reviewing the actual source code files for ShareAlbum. The specific names of our secure coders have been redacted to protect their identity and their performance results as secure coders.

Secure Coder S14 briefly read the first paragraph of instructions (in less than 10 seconds) and then moved into the source code file album.helper.php (at around 13 seconds). This coder briefly examined the other source code files in Notepad++ and the RIPSHelp Instruction Guide. The secure coder started with the RIPS tool around 39 seconds into the session. The individual utilized the RIPS Instruction Guide to fill in the form on the RIPS tool. The coder modified the search in RIPS to only scan the specific file the coder was told contained potential security flaws. The person examined the settings for the RIPS tool. The student changed the RIPS verbosity level as indicated in the Instruction Guide (at around 72 seconds). The student received the first results from RIPS back around 130 seconds. The student struggled with using the RIPS tool because the coder kept attempting to input the specific file name to scan rather than the entire ShareAlbum codebase. At 187 seconds, the student got the RIPS tool to scan just the one source code file the coder was interested in. The student utilized the
Notepad++ file shortcut to find the correct path location for the specific source code file. The secure coder examined the results given by RIPS and then scanned source code files to find the issue. This secure coder utilized the FIND feature of Notepad++. After seeing the create_a_new_album function in the source code in Notepad++, the coder returned to RIPS and examined the SQL Injection help from RIPS (around 227 seconds). The secure coder returned and read the source code for the create_a_new_album function. The secure coder started with the comments for the function and then the body of that function. The secure coder began modifying the source code in that function around 270 seconds, even though this problem was a True/False problem. The student examined the album.helper.sql.php file at around 304 seconds to work on this problem. The student modified the source code in the function for build_sql_create_a_new_album to add an intval check around the database query. This student went back to the RIPS tool for additional scan results around 323 seconds. Around 343 seconds into working on this problem, this student returned to the Instruction Guide that was provided to them. The secure coder finished reading the instructions around 373 seconds. The student examined the album.helper.php source code file again. The secure coder quickly went through this source code file, searching for something. The secure coder reached the end of the file around 398 seconds and then returned to RIPS. The student examined the edit_existing_album function around 434 seconds. This secure coder added 3 more intval in album.helper.sql.php and this time around the user id session token. The secure coder returned to the instructions around 507 seconds to reread the question. The student returned to
the RIPS tool around 547 seconds and attempted to search for CWE-73. The student didn’t appear to find the issue for File Manipulation in the source code.

Secure Coder S2 started the session by examining the ShareAlbum web interface and clicked on Register for a user account at 16 seconds. This secure coder successfully created a ShareAlbum web account at 62 seconds. The developer examined the Instruction Guide provided for CWE-73 at 80 seconds, focusing on the part that described the specific question and the source code file name with potential security flaws. The coder switched from the ShareAlbum webpage to the RIPS web interface tool at 84 seconds, then switched back to the ShareAlbum web page 3 seconds later and then back to the RIPS web interface 10 seconds later. The student switched to the album.helper.php source code file for the first time at 105 seconds and then switched to the RIPSHelp Instruction Guide at 108 seconds. The student copied the provided path for ShareAlbum from the RIPSHelp Instruction Guide into the RIPS web interface. The student received the first results from RIPS at 172 seconds. The student switched back to the Instruction Guide at 181 seconds and examined the paragraph on what the RIPS tool was supposed to provide to them. This secure coder switched to the album.helper.php source code file 210 seconds into this part of the session. The students scanned the entire source code file from the top down before really looking at any of the results from the RIPS tool. The student selected the File Manipulation results in RIPS 241 seconds into the session and examined the first File Manipulation vulnerability reported in the RIPS web interface. At 251 seconds, the student switched back to the source code and scrolled to the create_a_new_album function
that was mentioned in the first RIPS *File Manipulation* report. At 265 seconds, the student switched to the wrong area of RIPS for the bug (SQL Injection) that was not associated with CWE-73. At 270 seconds, the student switched to the source code in *album.helper.php*. At 282 seconds, the student switched back to the Instruction Guide. At 312 seconds, the student switched to the RIPS web interface but selected the *File Injection* vulnerability list instead of the *File Manipulation* list. At 319 seconds, the student switched to the source code in *album.helper.php*. The student then switched back to the Instruction Guide at 328 seconds, and then at 330, the student correctly selected the *File Manipulation* filter in the RIPS web tool. At 338 seconds, the student examined the 1st reported security flaw associated with CWE-73 within the RIPS results output. At 350, the student switched over to the *album.helper.php* source code file and examined the `create_a_new_album` function that was reported in the first *File Manipulation* report from RIPS. The student made several transitions between the 1st reported *File Manipulation* bug in RIPS and the source code for `create_a_new_album`. The student never examined the 2nd, 3rd, or 4th *File Manipulation* report provided by RIPS. The 4th report contained the security flaw for CWE-73.

*Secure Coder S30* started the problem by reading the Instruction Guide provided to them and began at the top of that file. This secure coder moved down in a top-to-bottom fashion, reading the Instruction Guide for CWE-73. This student completed reading the Instruction Guide around 58 seconds into that part of the session. This secure coder then moved to the RIPS web interface. The student tried to determine what path file to give to the RIPS scanning tool.

401
This secure coder chose to select the local host address of the *ShareAlbum* webpage displayed to them. This secure coder downselected the vulnerability type in RIPS to the *File Manipulation* error. The RIPS didn’t return any results (due to using the wrong file scan path in the RIPS tool), and the student went back to examine the instruction sheet. The student tried to copy another path file name from the Instruction Guide into the RIPS scan file path option. The student examined the open source code files in Notepad++ to determine their location on the hard drive to provide that as the search location for the RIPS tool to scan around 170 seconds. The student had several extensive fixation durations during that time (several fixations that were 2 seconds long). The student received the first results from RIPS around 187 seconds into that session. The student examined each vulnerability reported in the RIPS tool. After reviewing the overall results page from RIPS, the student read the 1st vulnerability help guide provided in RIPS. The student switched back to the source code for *ShareAlbum* around 308 seconds and examined the source code for `create_a_new_album` in `album.helper.php`. The student spent 57 seconds looking into that one function, `create_a_new_album` in `album.helper.php`. The student returned to the RIPS tool and examined the 2nd vulnerability listed by RIPS. The student began scanning over the source code file `album.helper.php` around 335 seconds and looked for the `delete_this_file` function as indicated in RIPS as the 2nd vulnerability. The student made several transitions between the source code for this function and the RIPS tool. The student moved on to the 3rd vulnerability reported in RIPS around 435 seconds during that part of the session. This RIPS report dealt with the `add_a_new_file_to_album` function
in *album.helper.php*. The student moved onto the 4th RIPS issue around time 560 seconds, and this dealt with `add_a_new_file_to_album`, but specifically with `move_uploaded_file`, which was the security flaw. The student returned to the source code associated with this flaw and examined the code with the CWE-73 flaw. The student opened the HELP guide in RIPS associated with this security flaw to examine the example code and proof of concept provided by RIPS. It was interesting that the student found the source code associated with the security flaw but did not get the correct answer.

*Secure Coder S16* started the problem by immediately switching to the web interface for the RIPS tools and the *ShareAlbum* web interface. This secure coder entered their email address and some type of password and then realized the coder didn’t have an account. It was unknown what password the student entered into the webpage. The student next clicked the link to register for a new *ShareAlbum* account. The student completed the form (email, name, and password) for a *ShareAlbum* account and successfully logged into their new *ShareAlbum* account. The student switched to the Instruction Guide provided in Notepad++ and began reading from the top down for around 50 seconds into that part of the session. The student examined the specific question in the Instruction Guide and then moved into the *album.helper.php* file for a brief period before going back to the Instruction Guide. The student attempted to copy and paste the file location in the Instruction Guide into the RIPS tool to scan for vulnerabilities around 78 seconds. The student changed the vulnerability type in RIPS to the *File Manipulation*. The students got no results as the file path the coder provided was not
correct. The student returned to examine the Instruction Guide again around 100 seconds into the session. After 140 seconds into the recording session, the student found the RIPS Help Instruction Guide that provided additional instructions on using the RIPS tool. This secure coder returned their first results from the RIPS tool around 156 seconds into the session. This secure coder switched back to the Instruction Guide and then transitioned over to the RIPS tool and began examining the results provided in the RIPS tool. This person switched over to the album.helper.php file around 176 seconds into the session but went back to the RIPS tool and continued examining the report. This secure coder examined the source code associated with the 1st reported flaw by RIPS at 210 seconds. When the coder began looking at the create_a_new_album function, the coder focused on starting with the function parameters passed into it and the line of code in that function reported by RIPS. This secure coder returned to the Instruction Guide to review the problem again before switching back to the source code for create_a_new_album. This secure coder mostly used a top-down approach to examine this function at 240 seconds. When the coder got to the line that was also reported in the RIPS tool for the 1st flaw, the coder returned to reference what the RIPS tool reported to them earlier. This secure coder expanded the Help guide in RIPS at 267 seconds. This secure coder switched over to the album.helper.sql.php file to see the specific SQL query that was used to create a new album before switching back to the album.helper.php file. Around 296 seconds into the session, the student moved onto the 2nd reported flaw by the RIPS tool. This secure coder also examined the RIPS help tool for this flaw as well. The student
switched back to the source code for the 2nd reported security flaw in RIPS and began studying the delete_this_file function at 302 seconds into the session. This secure coder switched back to the Instruction Guide and then back to the album.helper.php code. The student switched back to the 1st reported bug in RIPS and scrolled in the source code back to the function named create_a_new_album and reviewed that source code again. The student wrapped up reviewing the create_a_new_album function around 497 seconds. After that, the student moved on to the 3rd vulnerability reported by RIPS. Then, at 508 seconds, the student returned to the Instruction Guide and reviewed the mitigation strategy mentioned in the Instruction Guide. The student switched back to the RIPS tool around 520 seconds. The student started scanning the album.helper.sql.php source code file before switching back to the create_a_new_album in the album.helper.php file. The student examined the last reported flaw in RIPS at 557 seconds and scrolled in the source code to the add_a_new_file_to_album function. This secure coder began examining that function code at 563 seconds, focused on the specific line provided by the RIPS tool. The student then examined the parameters passed into this add_a_new_file_to_album function. This secure coder reviewed the 4th reported bug in RIPS again.

Secure Coder S3 spent the first 10 seconds reading the Instruction Guide before switching to the album.helper.php source code file at 11 seconds. At 15 seconds, the student switched to the RIPSHelp Instruction Guide. At 17 seconds, the student changed the web browser tab to the RIPS tool web interface. At 26 seconds, the student copied the file path provided in the RIPSHelp Instruction
Guide. At 33 seconds, the student copied those into the RIPS web interface. At 33 seconds, the student inserted the file path to the ShareAlbum code base into the RIPS tool. This secure coder utilized the instructions in the RIPS Help Instruction Guide to set the verbosity level in RIPS at 37 seconds. This secure coder set the vulnerability type in RIPS to File Manipulation at 45 seconds. The student clicked scan at 49 seconds and got the 4 potential vulnerabilities reported by RIPS back in the web browser. At 77 seconds, the student scrolled in the RIPS reported list, glanced over the 4 reported issues, and scrolled back to the top of the RIPS report. At 86 seconds, the student switched to the album.helper.php source code file and then switched to the album.helper.sql.php source code file at 87 seconds before changing again back to the album.helper.php file at 88 seconds. The student examined the coding body of the create_a_new_album function. At 102 seconds, the student returned to the Instruction Guide and reviewed the introduction paragraph. At 123 seconds, the student shifted their attention to the RIPS report. At 227 seconds, the coder opened the RIPS help popup box for the 1st reported vulnerability. At 132 seconds, the coder shifted their attention to the album.helper.php source code file and examined the body of the create_a_new_album function. At 139 seconds, the coder returned to the 1st reported issue by RIPS. At 163 seconds, the student shifted their attention to the album.helper.php source code file, initially looking at the wrong function, get_albums, then the student corrected and looked at the body of the function create_a_new_album. The student mainly focused on the source code in album.helper.php but the coder looked back a few times at the popup box window in RIPS for the 1st reported flaw. At
190 seconds, the student scrolled in the popup box with help from RIPS. At 207 seconds, the student returned to the body of the *album.helper.php* source code file, specifically the line mentioned in the RIPS tool. At 221 seconds, the student looked again at the popup box help in RIPS for the 1st reported security flaw. The student returned to the source code in *album.helper.php* 4 seconds later. The student continued examining the programming line `mkdir` that was mentioned in the 1st reported issue by RIPS. At 240 seconds, the student returned to the RIPS report and immediately returned to the *album.helper.php* source code file. This time, however, the students focused on the function definition and function header comment block. At 251 seconds, the student returned to the RIPS tool and closed the popup help box associated with the 1st reported security flaw. Next, at 265 seconds, the student began examining the 2nd reported flaw in RIPS, and at 267 seconds, the student began scrolling in the *album.helper.php* source code file before stopping and returning to the RIPS tool at 270 seconds. This secure coder returned to the *album.helper.php* source code file at 277 seconds and continued scrolling. At 285 seconds, the coder stopped scrolling in the source code and began scrolling in the report generated by RIPS. This secure coder moved to the 3rd and 4th reported potential issues in RIPS. This secure coder resumed scrolling in the *album.helper.php* source code file and stopped scrolling at 295 seconds once the coder reached the `add_a_new_file_to_album` function (which was the function associated with the 3rd and 4th reported issue by RIPS). At 304 seconds, the coder opened the help popup box for the 4th reported issue by RIPS. This secure coder returned to the source code in *album.helper.php* file at 320 seconds.
This secure coder returned to the function `create_a_new_album` at 328 seconds. At 336 seconds, the student opened the popup box for the 1st reported issue again in the RIPS tool. At 342 seconds, the student returned to the source code for `create_a_new_album`. At 348 seconds, the student returned to the RIPS tool. The student returned to the source code file at 355 seconds. At 356 seconds, the student switched to the Instruction Guide and reviewed the question again. At 374 seconds, the student reviewed the usage of the RIPS tool as documented in the Instruction Guide. At 378 seconds, the student opened the `album.helper.sql.php` source code file. At 385 seconds, the student changed back to the Instruction Guide. At 396 seconds, the student reopened the `album.helper.sql.php` source code file. At 401 seconds, the student changed to the `album.helper.php` source code file and read the function `create_a_new_album`.

Secure Coder S31 started reading the Instruction Guide that was presented for this specific CWE. This secure coder spent their first 58 seconds reviewing the Instruction Guide before switching to the ShareAlbum web interface. This secure coder started the process of registering for a ShareAlbum account (username, email, first and last name, and password). This secure coder completed the registration process in around 88 seconds and examined the user logged in to the ShareAlbum homepage. The student clicked the different web links within the ShareAlbum website. This secure coder successfully visited each ShareAlbum webpage by 119 seconds. Next, the student returned to the Instruction Guide provided for CWE-73. The student read in detail the question provided in the Instruction Guide from 128 seconds to 141 seconds. The student then switched to
the RIPSHelp Instruction Guide and fixated on the path/file instruction provided to them. The student, at 149 seconds, switched to the RIPS web interface and entered the path file provided to them in the RIPSHelp Instruction Guide. The student switched to the \textit{album.helper.php} file at 175 seconds briefly before going back to the instruction for CWE-73 and rereading the question. The student attempted to modify the RIPS search criteria for the specific file 'album.helper.php'. This secure coder got the correct results from RIPS at 209 seconds in this part of the session. The student glanced over several of the security flaws reported by RIPS (some associated with \textit{File Manipulation} as well as other security CWEs). At 250 seconds, the student modified the filter for RIPS to limit to only \textit{File Manipulation} results. The student then went back to the Instruction Guide and reviewed the content again. At 288 seconds, the student reviewed the guide on the RIPSHelp Instruction Guide again. At 309 seconds, the student reviewed the 1st \textit{File Manipulation} security flaw reported by RIPS. The student reviewed the help popup provided by RIPS for the 1st reported flaw for \textit{File Manipulation}. At 370, the student examined the 2nd reported bug by the RIPS tool. The student did not look in the source code file for the 1st reported bug by RIPS. At 378 seconds, the student examined the third security flaw reported by RIPS for \textit{File Manipulation}. The student then reviewed all 4 reported flaws by RIPS, quickly comparing them. The student opened the Help guide to the 4th reported bug in RIPS at 470 seconds. This student never spent significant time looking at the source code files.
Secure Coder S23 displayed very few fixations for the first 93 seconds of this part of the recording. The first recorded fixations appeared in the Instruction Guide. After a few seconds, their fixation stopped appearing. At 146 seconds, the student switched from the Instruction Guide to the RIPSHelp Instruction Guide. This secure coder copied and pasted the ShareAlbum file path to the RIPS web tool and adjusted the options as indicated by the RIPSHelp Instruction Guide. The student switched back to the Instruction Guide and then began scrolling through the results of the RIPS tool at 187 seconds. This student found the filter in RIPS for File Manipulation at 238 seconds, and the coder received a new report from RIPS that contained only the security flaws associated with this problem. The student returned to the Instruction Guide and reviewed the content again. At 276 seconds, the student returned to the RIPS tool results. At 292 seconds, the student switched to the album.helper.sql.php source code file and 2 seconds later, changed to the album.helper.php file. The student seemed to examine the create_a_new_album function in album.helper.php. At 324 seconds, the student switched back to the Instruction Guide. At 358 seconds, the student returned to the album.helper.php source code file and began reading the create_a_new_album function again. The student continued to show relatively few fixations. At 390 seconds, the student started scrolling in the album.helper.php file. At 402 seconds, the student turned to the 2nd reported flaw in RIPS. At 424 seconds, the student moved to the 3rd flaw reported by RIPS. At 448 seconds, the student scrolled to the correct location in the source code that contained the security flaw for the add_a_new_file_to_album function. At 460, the student resumed their search.
in the source code file `album.helper.php`. At 470, the coder switched back to the Instruction Guide and seemed to read from the top down. At 540 seconds, the student switched back to the `album.helper.php` source code file, and at 544 seconds, the student returned to the RIPS tool. The student then went back to the code and began examining the `create_a_new_album` function. At 574 seconds, the student opened a new tab in the web browser, searched for CWE-73, and visited the Mitre Corporation webpage. The study reviewed part of the webpage before switching back to the RIPS tool at 622 seconds. This secure coder reviewed the RIPS tool report before ending their session.

Secure Coder S26 spent the first 19 seconds reviewing the Instruction Guide provided to them. The student quickly glanced through the file names for the source code (the coder did not actually look at the source code). At 39 seconds, the student used the RIPSHelp Instruction Guide to insert the file scan path in the RIPS web interface. This secure coder followed the instructions in the Instruction guide to modify the RIPS search file path to the specific file that was mentioned in the instructions containing the potential vulnerability for CWE-73. However, the coder did not get the RIPS path right, so the coder turned to the RIPSHelp Instruction Guide and found the correct path. The first results from RIPS came back at 121 seconds. This secure coder initially clicked on the vulnerability reported by RIPS that was not associated with the CWE-73 (File Manipulation). At 139 seconds, the student returned to the Instruction guide and started rereading the specific question. At 149 seconds, the student switched to the `album.helper.php` file and started reading the `create_a_new_album`
function. At 172 seconds, the student switched to the `album.helper.sql.php` file and started reading the function `build_sql_create_a_new_album`. The student returned to the `album.helper.php` file at 177 seconds. At 180 seconds, the student started reading the `get_albums` function. The student then quickly scrolled and glanced over several methods in the `album.helper.php` file. The student seemed to read the names of the functions in the `album.helper.php` file. At 205 seconds, the student shifted their attention back to the RIPS web interface tool and started scrolling through the report it provided. At 221 seconds, the student examined a cross-site scripting vulnerability reported by RIPS. At 226 seconds, the student examined the help popup given by RIPS but for the Cross-Site Scripting vulnerability (not for CWE-73). The student stopped examining the Cross-Site Scripting help report provided in RIPS at 262 seconds. The student then returned to the `album.helper.php` source code file and glanced over it. At 270 seconds, the student switched back to the RIPSHelp Instruction Guide. At 273 seconds, the student switched to the `album.helper.sql.php` file and started reading the code in the function `build_sql_create_a_new_album`. At 280 seconds, the student started scrolling in the `album.helper.sql.php` source code file, quickly glancing at sections of the code, primarily focusing on the comments description for each function. At 294 seconds, the student returned to the RIPS tool report and examined the File Injection vulnerability reported by RIPS. The student then quickly glanced over the report provided by RIPS for multiple different types of vulnerabilities. The student never examined the `File Manipulation` vulnerability report in RIPS or looked at the source code in the `album.helper.php` associated with CWE-73.
Secure Coder S10 started by reading the paragraphs in the CWE 73 Instruction guide. This secure coder did so in a top-down fashion, reading each paragraph sequentially. The student finished the initial read-through of the Instruction guide in 113 seconds. Then the student returned to reading the specific question mentioned in the Instruction guide, finished this reading at 158 seconds, and then switched to the RIPSHelp Instruction Guide. This secure coder copied the file path tip from the RIPSHelp Instruction Guide to the RIPS tool web interface portal at 170 seconds. At 204 seconds, the student appeared confused about which vulnerability type filter to apply in RIPS, and the coder switched back to the Instruction guide. This secure coder highlighted the vulnerability type in the Instruction guide and read through the Instruction guide again (this time more quickly than the first time). At 245 seconds, the student went back to the RIPS tool, selected the RIPS filter and applied the filter for File Manipulation (the correct one). At 259 seconds, however, the student reviewed the RIPSHelp Instruction Guide and noticed that the instructions indicated setting the filter to ALL. So, the student undid their recent changes in the RIPS tool and changed the filter to ALL vulnerabilities. The student received the first results from RIPS at 264 seconds. The student then went back to the filter option in RIPS and changed from ALL to File Manipulation to narrow the results from the RIPS tool. The student received the finalized results from RIPS for File Manipulation only at 279 seconds. The student reviewed the question in the Instruction guide again and then switched to the album.helper.php source code at 297 seconds. The student quickly looked through the entire album.helper.php source code file and
then went back to the top of this file. Next, at 304 seconds, the student examined
the comments associated with the function create_a_new_album. At 315 seconds,
the student began examining the source code associated with the function create_a_new_album in a top-down fashion. The student went back to the Instruction
guide at 363 seconds and reviewed the question again. At 378 seconds, the student
went back to the RIPS tool and began examining the 1st reported vulnerability
by RIPS for File Manipulation. The student chose to minimize all 4 reported bugs
in the RIPS tool to collapse their explanations. Then, at 390 seconds, the student
expanded the 1st vulnerability reported by RIPS and began reading the content in
the bug report. This secure coder switched to the album.helper.php source code
file at 408 seconds and examined the code in the function create_a_new_album,
which was associated with the 1st security flaw reported by RIPS. The student
highlighted the specific line in the source code that RIPS indicated as a potential
flaw, and the student returned to the RIPS tool and examined the 1st bug report.
At 420 seconds, the student examined the overall logic in the create_a_new_album
function. This secure coder started with the function definitions and parameters
before moving into the function’s body. At 448 seconds, the coder returned to
the RIPS tool and expanded the 2nd collapsed vulnerability listed by RIPS. This
secure coder reviewed the content listed by RIPS for the 2nd vulnerability. At 480
seconds, the student expanded the 3rd vulnerability list by RIPS and reviewed
its content. At 493 seconds, the student returned to the album.helper.php source
code, and by 500 seconds, the student found the source code associated with the
3rd vulnerability list by RIPS. At 510 seconds, the student returned to the 2nd
vulnerability in the RIPS tool, and at 516 seconds, the student scrolled to the location in the album.helper.php source code that contained that line. The student read the line in the source code indicated by RIPS and then read several coding lines around that line in the function delete_this_file. At 533 seconds, the student returned to the RIPS tool and reviewed the report again. At 540, the student returned to the source code in the function delete_this_file. At 566 seconds, the student returned to the RIPS tool and reviewed the 2nd reported security flaw. The student went back to the source code at 570 seconds. At 593 seconds, the student returned to the create_a_new_album function and read the comment header for that function. Next, the student reviewed the content of the function create_a_new_album. At 630 seconds, the student attempted to search using Notepad++ FIND feature for the function build_sql_create_a_new_album, which is called inside of the create_a_new_album function. The student couldn’t find the function (because it was in a different source code file), and the student continued reading the create_a_new_album function. The student wrapped up their reading of the function at 775 seconds. The student returned to the RIPS tool and went to the 2nd reported vulnerability. The student scrolled in the album.helper.php source code to the delete_this_file function that was associated with the 2nd reported security flaw by RIPS. At 794 seconds, the student began examining the function’s name for delete_this_file and then examined the source code contained in the function. The student finished reading this function at 925 seconds. At 936 seconds, the student began reviewing the source code associated with the function add_a_new_file_to_album (this was associated with the 3rd flaw reported by RIPS).
Again, the student started with the function’s name and then progressed through the function source code. At 1036 seconds, the student finished reviewing the source code in *add_a_new_file_to_album*. The student returned to the Instruction guide and reviewed the question again.

*Secure Coder S20* began by quickly switching between all open tabs in Notepad++ and the Web browser. At 23 seconds, the student switched to the RIPSHelp Instruction Guide and copied the path/file needed for RIPS into the RIPS web interface. This secure coder set the verbosity level for RIPS, adjusted the filter, and received their 1st report from RIPS 44 seconds in. The student switched back to the Instruction guide for CWE-73 at 47 seconds. After reviewing the Instruction Guide, the student modified the vulnerability type filter in RIPS software to *File Manipulation* only at 97 seconds. The student received results from RIPS that focused only on *File Manipulation* 120 seconds into the recording. Next, the student examined the 1st reported vulnerability given by RIPS. At 142 seconds, the student returned to the Instruction guide and began rereading the specific task. The student quickly scrolled through all 4 of the vulnerabilities provided in the RIPS report. At 152 seconds, the student went back to the 1st reported vulnerability and continued reading the RIPS report on this problem. At 160 seconds, the student reviewed the 2nd security flaw reported by RIPS. This secure coder examined the RIPS Help *File Manipulation* popup box, including example code, proof of concept, and vulnerability mitigation strategy. At 185 seconds, the student reviewed the third vulnerability reported by RIPS for CWE-73. At 200 seconds, the student went back to the 1st vulnerability re-
ported by RIPS. At 210 seconds, the student went back to the 3rd vulnerability reported by RIPS. At 221 seconds, the student opened the `album.helper.sql.php` file as their main source code file in view and went back to the 1st vulnerability report in RIPS. At 226 seconds, the student switched to the `album.helper.php` file. At 233 seconds, the student went back to the Instruction guide before switching again to the `album.helper.php` file at 240 seconds. The student began reading the source code in the function `create_a_new_album`. At 251 seconds, the student went back to the 1st vulnerability reported in RIPS and began reading the report generated by RIPS. At 260 seconds, the student read the RIPS Help File Manipulation popup box for the 1st vulnerability reported by RIPS. At 266 seconds, the student went back to the source code for `create_a_new_album` and read the body of that function. The student remained in that function until 293 seconds, then switched to the `album.helper.sql.php` file and began reading the logic in the `build_sql_create_a_new_album` function. This function was called from the `album.helper.php` file. At 300 seconds, the student switched back to the `album.helper.php` file and continued reading the `create_a_new_album` function. At 308 seconds, the student went back to the RIPS tool and made repeated transitions between it and the `create_a_new_album` function for 18 more seconds. At 356 seconds, the student went back to the Instruction guide before returning to the `album.helper.php` file 4 seconds later. Then, the student scrolled through the `album.helper.php` source code file until reaching the `delete_this_file` function at 378 seconds (this function was associated with the 2nd vulnerability reported by RIPS). The student stayed in the body of this `delete_this_file` func-
tion until 396 seconds. The student then switched to the `album.helper.sql.php` file and began scrolling, stopping upon reaching the `build_sql_delete_this_album` function at 407 seconds. The student switched back to the `album.helper.php` file at 413 before switching again to the `album.helper.sql.php` file. There, the student located the function `build_sql_prepare_to_delete_this_file`, which was called in the `album.helper.php` file. At 439 seconds, the student switched back to the `album.helper.php` file. Then, the student went to the RIPS tool and read the 1st reported vulnerability by RIPS for File Manipulation. The student scrolled through the `album.helper.php` file again and revisited the `create_a_new_album` function at 469 seconds. At 521 seconds, the student looked at the 1st reported flaw by RIPS and read the source code used to determine if a vulnerability existed. The student finished reviewing the RIPS help for the 1st vulnerability at 546 seconds. At 558 seconds, the student minimized the 1st vulnerability report in RIPS and examined the 2nd reported vulnerability by RIPS. At 591 seconds, the student located the `delete_this_file` function in the `album.helper.php` file associated with the 2nd security flaw reported by RIPS and looked over the body of this `delete_this_file` function. This student was focused several times on the line of code for the function call `build_sql_delete_this_file` before switching to the `album.helper.sql.php` file at 612 seconds. At 616 seconds, the student switched to the `album.helper.sql.php` file. This secure coder quickly switched back to the `album.helper.php` file at 619 seconds and then again to the `album.helper.sql.php` file at 621 seconds. The student switched back to the `album.helper.php` file, specifically to the `delete_this_file` function, at 632 seconds. At 663 seconds, the student
revisited the RIPS tool, examined the 3rd vulnerability reported by RIPS, and then moved to the 2nd vulnerability reported by RIPS. This secure coder revisited the source code of `delete_this_album` at 709 seconds. This secure coder changed to the `album.helper.sql.php` file at 734 seconds. At 754 seconds, the student returned to the `album.helper.php` file and began reading the function body for `delete_this_file`. The student looked at the function call `build_sql_delete_this_file` in `album.helper.php` at 787 seconds. At 789 seconds, the student changed to the `album.helper.sql.php` file and began examining the function `build_sql_delete_this_file` location in the `album.helper.sql.php` file. At 795 seconds, the student returned to the `album.helper.php` file. At 810 seconds, the student went to the RIPS tool and read the 2nd reported vulnerability. At 842 seconds, the student collapsed the 2nd vulnerability report by RIPS. Then, the student examined the 3rd vulnerability flaw reported by RIPS for *File Manipulation*. At 856 seconds, the student opened and read the RIPS help popup box for the 3rd reported flaw. At 880 seconds, the student revisited the 2nd vulnerability reported by RIPS, opened its RIPS help popup box, and began reading. At 884 seconds, the student clicked a link to the RIPS tool to seek more assistance from the PHP website (a new tab in the web browser opened). At 888 seconds, the student went back to the RIPS tool. At 908 seconds, the student revisited the source code `album.helper.php` file and made several transitions between the 2nd reported flaw and the source code in `album.helper.php` associated with that reported potential security risk. At 940 seconds, the student quickly scanned the `album.helper.php` file as if searching for a specific method. At 958 seconds, the student paused
at the function `delete_this_file`, which was the function associated with the 3rd and 4th vulnerabilities reported by RIPS. At 965 seconds, the student went back to the 1st reported security flaw by RIPS and at 973 seconds, revisited the function `create_a_new_album`. At 1004 seconds, the student switched to the `album.helper.sql.php` file and then to the `album.helper.php` file. The student made several transitions between the two source code files over the next few seconds. At 1035 seconds, the student revisited the Instruction guide. At 1050 seconds, the student switched again to the `album.helper.php` file and began reading the body of the function `add_a_new_file_to_album`. At 1088 seconds, the student switched to the `album.helper.sql.php` file and then back to the `album.helper.php` file 3 seconds later, reviewed the comments header for the function `add_a_new_file_to_album`, and then delved back into the code for this function. There were several fixations at time 1138 seconds with the actual security flaw. At 1150 seconds, the student went back to the RIPS tool to examine the 1st reported flaw. Afterward, the student revisited the Instruction Guide.

For `Secure Coder S12`, in the first 15 seconds, the student began reading the first paragraph of the Instruction guide and switched the web browser to the RIPS main web interface. The student then quickly glanced at the other paragraphs in the Instruction guide, and then at 36 seconds, the student examined the vulnerability type filter in the RIPS tool (but the student didn’t change the default of ‘all’). At 42 seconds, the student switched to the RIPSHelp Instruction Guide. At 48 seconds, the student changed the verbosity level as indicated in the RIPSHelp guide. At 59 seconds, the student copied the path/file hint provided
in the RIPSHelp Instruction Guide into the RIPS tool. Then, the student appeared to select the vulnerability type filter again in RIPS but didn’t change it and instead returned to the RIPSHelp Instruction Guide and read the statement on the filter setting. At 70 seconds, the student switched back to the Instruction guide, and at 74 seconds, updated the RIPS tool filter to be set to File Manipulation (the specific CWE-73 that was being worked). Next, the student went back to the Instruction guide and began reading the mitigation strategy and the potential vulnerabilities mentioned in the Instruction guide. At 109 seconds, the student switched to the album.helper.php source code file and began reading the create_a_new_album function. At 170 seconds, the student shifted their attention to the RIPS web interface and appeared to start comparing the 1st reported vulnerability report by RIPS for File Manipulation and the create_a_new_album function source code in album.helper.php. The student mainly focused on the source code in album.helper.php rather than the RIPS tool’s 1st reported vulnerability. At 204 seconds, the student scrolled in the album.helper.php file to the get_albums function. At 208 seconds, the student scrolled in the RIPS web interface tool to the 2nd reported vulnerability by RIPS, and at 210 seconds, scrolled back to the top of the album.helper.php file where the create_a_new_album function was located. At 212 seconds, the student changed the source code view from the album.helper.php file to the album.helper.sql.php file, but 5 seconds later, switched back to the album.helper.php file. At 227 seconds, the student looked at the 2nd report by RIPS and started scrolling in the album.helper.php file until arriving at the function delete_this_file at 232 seconds. This function was associated with the
2nd reported vulnerability by RIPS. At 240 seconds, the student scrolled in the
album.helper.php file again until reaching the create_a_new_album function at 249
seconds. The student then compared the 1st reported vulnerability in RIPS with
the create_a_new_album function in album.helper.php. At 283 seconds, the student
scrolled in the album.helper.php file until reaching the delete_this_file function
again. At 360 seconds, the student compared the source code in the delete_this_file
function in the album.helper.php file with the 2nd reported vulnerability by RIPS.
At 410 seconds, the student briefly examined the 3rd and 4th vulnerability re-
ported by RIPS. The student returned to the delete_this_file function at 420 sec-
onds. The student frequently looked at the body of the source code, the function
comments, and the function definition with input parameters. At 451 seconds,
the student examined the 3rd reported vulnerability by RIPS. At 463 seconds,
the student examined the 4th vulnerability reported by RIPS. At 500 seconds, the
student went back to the 3rd reported issue by RIPS. The student scrolled in the
album.helper.php file until reaching the function add_a_new_file_to_album. This
function was associated with the 3rd and 4th vulnerabilities reported by RIPS.
The student began reading the comments on this function and then moved to the
function's body. Reading of the function add_a_new_file_to_album continued until
557 seconds. The student then switched to the RIPS web interface and examined
the 4th reported flaw by RIPS. At 568 seconds, the student returned to the source
code in add_a_new_file_to_album. At 653 seconds, the student revisited the 4th re-
ported flaw by RIPS and then returned to the source code in album.helper.php 5
seconds later. At 672 seconds, the student switched back to the Instruction guide
before ending the session.

Secure Coder S24 started out the session by reading the first part of the
first sentences of the Instruction guide, and then at 16 seconds in, the student
changed to the album.helper.php source code file. The student quickly looked at
the body of the first function, create_a_new_album before changing to the RIPS-
SHelp Instruction Guide at 22 seconds. The student brought into view the RIPS
web interface at 26 seconds. At 33 seconds, the student changed back to the
Instruction Guide provided and read about the specific question that was being
asked for this exercise. At 51 seconds, the student changed the RIPS vulnerability
type to File Manipulation. At 54 seconds, the student selected SCAN in RIPS,
but the coder received no results (the coder did not give a path/file name in the
RIPS tool). At 59 seconds, the coder changed the vulnerability type in RIPS to
ALL. At 67 seconds, the student changed the vulnerability type in RIPS back
to File Manipulation. At 76 seconds, the students returned to the Instruction
Guide and reviewed the question again. At 84 seconds, the student returned to
the RIPS tool and examined the available options in the RIPS tool web interface.
At 100 seconds, the student returned to the Instruction guide and focused on the
question paragraph. At 103 seconds, the student copied the relative path for the
album.helper.php and pasted it into the RIPS tool at 108 seconds. The student
scan returned no results (because it was a relative path). At 114 seconds, the
student changed the filter in RIPS to ALL and got back no results. The student
continued trying different file paths until 133 seconds, and then the student
changed to the RIPSHelp Instruction Guide. At 137 seconds, the student copied
the file path as indicated in the RIPSHelp Instruction Guide to the RIPS tool,
and the coder got back their first RIPS result set at 148 seconds. At 150 seconds,
the student changed the filter from type *ALL* to type *File Manipulation* and got
back the four potential vulnerabilities associated with this CWE. The student
changed back to the Instruction Guide and reread the question. At 167 seconds,
the student changed to the *album.helper.php* source code file. At 172 seconds, the
student reviewed the report on the 1st vulnerability type returned by the RIPS
tool. At 180 seconds, the student returned to the *album.helper.php* file and started
reading the source code for the function *create_a_new_album*. The student added a
Notepad++ marker on line 20 at 181 seconds, which was the line that RIPS indi-
cated has a security flaw. The student continued reading the *create_a_new_album*
function until 255 seconds, and the student returned to the 1st vulnerability re-
ported in RIPS and reviewed the initial statement in the RIPS report. At 263
seconds, the student switched to the RIPSHelp Instruction Guide and read from
the top down. At 268 seconds, the student changed to the *album.helper.sql.php*
file. At 271 seconds, the student changed back to the Instruction Guide. This
student read the instructions on what to do when they finished. The student
never examined the 2nd, 3rd, or 4th reported flaw by RIPS. The student couldn’t
get the correct answer for the right reasons.

*Secure Coder S18*, in the first 47 seconds, read the paragraphs in the
Instruction guide that discussed the CWE-73 problem and how an attacker could
take advantage of it. This also included reading the paragraph on the mitigation

424
strategy. At 47 seconds, the student changed the web browser to open the RIPS web interface. At 50 seconds, the student returned to the Instruction guide, and at 56 seconds, the student started selecting the file path (relative) given to the album.helper.php source code file. The student pasted that into the RIPS web interface tool and changed the filter type from ALL to File Manipulation at 66 seconds. At 67 seconds, the student received their 1st report from the RIPS tool that did not contain any security issues (because the relative path to the source code was used). The student returned to the Instruction Guide at 73 seconds. The student changed to the album.helper.php source code file at 76 seconds and briefly examined the create_a_new_album function. At 88 seconds, the student changed back to the Instruction Guide. The student returned to the RIPS tool at 93 seconds. At 107 seconds, the student returned to the Instruction guide and examined the file path again. The student copied it again into the RIPS tool and searched again with no results from RIPS (relative path is what the coder inserted when it needed an absolute path). This secure coder switched to the RIPSHelp Instruction Guide at 125 seconds. This secure coder copied the correct file path and pasted it into the RIPS tool. This secure coder got back their first search results from RIPS at 139 seconds, containing the four potential vulnerabilities that RIPS found for CWE-73. This secure coder switched back to the Instruction Guide at 143 seconds. The student changed to the album.helper.php source code file at 144 seconds and back to the Instruction Guide at 146 seconds. This secure coder read the attacker plan and mitigation strategy again in the Instruction Guide. This secure coder switched to the album.helper.php source code file at
172 seconds. At 175 seconds, the coder switched their attention to the RIPS tool 1st reported flaw. At 180 seconds, the coder opened the RIPS popup box to help with the 1st reported vulnerability. At 200 seconds, the coder opened the popup box for the source code in RIPS for the 1st discovered vulnerability. This secure coder closed this popup box and returned to the source code file album.helper.php and started looking into the function create_a_new_album. This secure coder primarily focused on the body of this programming function. At 215 seconds, the coder returned to the RIPS tool and examined the RIPS popup box to help for the 1st reported vulnerability. This secure coder returned to the source code for create_a_new_album at 236 seconds. This secure coder returned to the RIPS tool at 238 seconds and examined the RIPS popup box to help for the 1st reported issue by RIPS. The student returned to the source code file album.helper.php at 259 seconds and continued examining the body of the function create_a_new_album. At 265 seconds, the student returned to the RIPS tool and started looking at the 2nd reported issue by the RIPS tool. At 270 seconds, the student returned to the source code file album.helper.php and started scrolling in this source code file. The student reached the function delete_this_album at 279 seconds and stopped scrolling. At 295 seconds, the student opened the RIPS help popup box associated with the 2nd reported security flaw. At 322 seconds, the student examined the 3rd reported security flaw by the RIPS tool. At 330 seconds, the student returned to the Instruction Guide. At 339 seconds, the student examined the 4th reported issue by RIPS. At 343 seconds, the student returned to the album.helper.php source code file. At 350 seconds, the student
opened the RIPS source code popup box for the 4th reported security flaw. This secure coder closed the popup box after 5 seconds. Next, the coder returned to the Instruction Guide and reread the specific question.

Secure Coder S15 took the first 28 seconds to read the Instruction guide from the top down, reading the first 4 paragraphs focused on the CWE, the attacker plan, and the mitigation strategy. At 28 seconds, the student switched to the album.helper.php source code file. At 30 seconds, the student changed back to the Instruction Guide. Next, the student changed the web browser to point to the RIPS web interface tool. At 40 seconds, the student changed their attention back to the Instruction Guide, reading the paragraph on the RIPS tool and the specific question associated with this problem. At 48 seconds, the student switched to the album.helper.php source code file and started scrolling in that file. By 54 seconds, the student reached the end of that source code file and returned to the top of the album.helper.php file. Next, the student switched to the RIPSHelp Instruction Guide and read the instructions on the file path. By 68 seconds, the student typed in the path/file search field of the RIPS tool. By 96 seconds, the students got the first results back from the RIPS web tool. The student initially scrolled in the RIPS report log and then returned to the top of it. At 105 seconds, the student returned to the Instruction Guide and reread the specific question. At 109 seconds, the student switched to the album.helper.php source code file. At 114 seconds, the student changed the filter type in RIPS to File Manipulation and returned to the Instruction guide to reread the specific question. At 125 seconds, the student received a new scan report from the RIPS tool, which only
contained the security flaws associated with this problem. The student took a
glance through the length of the report returned by RIPS (only contained 4 po-
tential risks then). At 136 seconds, the student read the 1st reported vulnerability
report RIPS provided for *File Manipulation*. At 140 seconds, the students opened
the Help popup box in RIPS for the 1st vulnerability. The student closed that
popup box a few seconds later. At 159 seconds, the student opened the popup box
for the source code in RIPS for the 1st vulnerability. At 166 seconds, the student
opened the source code popup box in RIPS for the 2nd vulnerability reported by
RIPS for *File Manipulation*. At 175 seconds, the student opened the popup box
for the source code inside RIPS that was associated with the 3rd vulnerability.
At 179 seconds, the student opened the popup box for the 4th vulnerability. At
186 seconds, the student returned to the top of the RIPS web tool page and be-
gan reading the RIPS tool report on the 1st vulnerability for *File Manipulation.*
At 200 seconds, the student switched from the Instruction Guide to the source
code file *album.helper.php*. The student began reading the body of the function
*create_a_new_album*. The student looked at the body of this function until 258
seconds. At 258 seconds, the student looked back at the RIPS tool with the
1st reported vulnerability. At 261 seconds, the student returned to reading the
function’s body *create_a_new_album*. At 268 seconds, the student returned to the
vulnerability reports listed in RIPS. At 285 seconds, the student briefly returned
to the source code in *album.helper.php* before returning to the RIPS tool and
reading the 2nd vulnerability report. The student returned to the source code in
*album.helper.php* at 300 seconds. The student scrolled through approximately half
of the `album.helper.php` source code file before switching to the Instruction Guide at 307 seconds. The student read the specific question in the Instruction Guide. The student returned to the `album.helper.php` source code file at 313 seconds and scrolled until the coder reached the function `delete_this_file` at 321 seconds. This `delete_this_file` function was associated with the 2nd reported vulnerability by RIPS. The student studied the function body until 326 seconds and then looked at the function parameters for 327 seconds. This secure coder returned to the reports in the RIPS tool at 336 seconds. The student began reading the third vulnerability report RIPS provided at 344 seconds. By 346 seconds, the student resumed scrolling in the `album.helper.php` source code file, stopped scrolling in the `create_a_new_album` function, and reviewed the body of that function. At 372 seconds, the student switched to the `album.helper.sql.php` file and began reading the `build_sql_create_a_new_album` function. At 376 seconds, the student returned to the file `album.helper.php` and read the code in `create_a_new_album`. At 384 seconds, the student briefly returned to RIPS and examined the 1st reported vulnerability by RIPS and then returned to the source code for `create_a_new_album`. At 397 seconds, the student began scrolling in the `album.helper.php` source code file but stopped with little changes in the content displayed. At 401 seconds, the student returned to the RIPS tool and examined the last reported vulnerability. At 432 seconds, the student looked at the source code in `album.helper.php`, specifically the `create_a_new_album` function. The student then scrolled in the `album.helper.php` file to the function `add_a_new_file_to_album` (this was the one associated with the 4th reported flaw by RIPS). At 442 seconds, the student
began scrolling in the *album.helper.php* source code file; however, the student returned to the function *add_a_new_file_to_album* by 458 seconds.

For *Secure Coder S6*, the first 16 seconds of the student were the initial part of the Instruction. At 17 seconds, the student changed the web browser from the *ShareAlbum* website to the RIPS tool web interface. The student then resumed their reading of the Instruction Guide. At 42 seconds, the student copied the relative path from the Instruction Guide containing the path file name of the *album.helper.php* source code file. The student pasted that into the RIPS tool and returned to the Instruction guide in 49 seconds. At 69 seconds, the student changed to the *album.helper.sql.php* source code file. At 72 seconds, the student changed back to the Instruction Guide. At 78 seconds, the student briefly changed to the *album.helper.php* file before switching to the RIPSHelp Instruction Guide at 79 seconds. At 100 seconds, the student copied the path provided in the RIPSHelp Instruction Guide and pasted that into the RIPS web interface tool (not the relative path). By 120 seconds, the student changed the verbosity level and vulnerability type filter in RIPS to show only the issues for *File Manipulation*. The student got the first results back from RIPS at 128 seconds. The student reviewed the RIPSHelp Instruction Guide again and then switched to the *album.helper.php* file at 137 seconds. At 140 seconds, the student changed to the Instruction Guide. At 150 seconds, the student began reading the 4 reported security flaws given by the RIPS tool. The student changed to the *album.helper.php* source code file at 163 seconds and began reading the body of the function *create_a_new_album*. At 172 seconds, the student reviewed the 1st vulnerability reported by RIPS. At 180
seconds, the student reviewed the 2nd reported flaw by RIPS. At 306 seconds, the student returned to the `album.helper.php` source code file and began reading the `create_a_new_album` function. At 210 seconds, the student completed reading the `create_a_new_album` function and scrolled in the `album.helper.php` source code file. At 223 seconds, the student stopped scrolling at the function `delete_this_file`. This function was associated with the 2nd reported vulnerability by RIPS for *File Manipulation*. At 236 seconds, the student returned to the RIPS web interface. The student went to the 3rd vulnerability reported by RIPS. At 246 seconds, the student returned to the `album.helper.php` source code file and began scrolling. By 268 seconds, the student reached the function `add_a_new_file_to_album`, which was associated with the 3rd and 4th vulnerabilities reported by RIPS. The student reviewed the body of this function before exiting the session.