4-28-2024

Smart Contracts in Blockchain File Sharing

Megan Elizabeth Salmons
University of Alabama in Huntsville

Smart Contracts in Blockchain File Sharing

by

Megan Elizabeth Salmons

An Honors Capstone

submitted in partial fulfillment of the requirements

for the Honors Diploma

to

The Honors College

of

The University of Alabama in Huntsville

04/28/2024

Honors Capstone Project Director: Dr. Earl Wells

[illegible] 04/28/2024

Student Date

Earl Wells 05/04/2024

Project Director Date

Aleksandar Milenkovic Digitally signed by Aleksandar Milenkovic Date: 2024.05.05 20:10:18 -05'00'

Department Chair Date

Sean Lane Digitally signed by Sean M Lane Date: 2024.05.07 10:25:56 -05'00'

Honors College Dean Date

HONORS COLLEGE
THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Honors College

Frank Franz Hall

+1 (256) 824-6450 (voice)

+1 (256) 824-7339 (fax)

honors@uah.edu

Honors Thesis Copyright Permission

This form must be signed by the student and submitted with the final manuscript.

__Megan Salmons________________

Student Name (printed)

[illegible]

Student Signature

__04/28/2024__________________

Date

Table of Contents:

Abstract

The purpose of this honors project was to demonstrate the development and implementation of new smart contract technology in a locally deployed distributed blockchain system. This project also intended to demonstrate the feasibility of application of this technology through the consideration of a use-case representing the demands of a real world file sharing environment. This project investigated the merits and practicality of utilizing smart contracts for access control integration in a secure file sharing system. This work was conducted as a part of The University of Alabama in Huntsville's CPE 488/498 Cybersecurity Capstone course during the 2023-2024 academic year.

Smart contracts are pieces of code deployed on a blockchain to automatically perform or automate functionality. In this case, the focus of the smart contract is on access control features to promote greater security on a file sharing blockchain network within a medical scenario. The blockchain uses the InterPlanetary File System (IPFS) for off-chain storage and Proof-of-Stake for consensus but due to the implementation of these features additional access control is required in order to ensure proper access is maintained. The smart contract deployed in this project focuses on associating users with an address, user account, and role, allowing for login, access list, and role based controls to occur on each instance of the chain. Through integration and deployment of these smart contract access control features, the security of the blockchain can be increased, continuing to improve the overall confidence in the blockchain and data sharing.

Introduction

Over the last decade blockchain technology has experienced many developments, making huge strides in product deployment and research. Some commonly seen applications of blockchains include cryptocurrencies, non-fungible tokens (NFTs), or use as an immutable ledger. Although these applications are among the most common, work continues to determine the potentially far reaching applications for blockchain technology. In order to assist in further development, additional features and associated technologies must be applied to help extend the functionality of blockchain. One such feature is a smart contract.

A smart contract is not a "contract" in the traditional sense of the word but rather, a function (or functions) in the blockchain code that aims to execute automatically in order to automate functionality in the blockchain without third party involvement (Khan 2021, 2901). When a smart contract executes, the blockchain it is deployed on is updated to maintain a record of the execution (IBM, n.d.). Since the smart contract is deployed with the blockchain, all of the nodes connected to the network have a copy of its functions, allowing for an equal right of execution among all nodes (Nguyen 2019, 66793).

Smart contracts can be deployed on a variety of blockchain platforms, however each implementation incorporates this functionality differently. Ethereum is one of the major blockchain platforms commonly used in research and development. It is the first (Khan 2021, 2903) and most commonly used platform for smart contract deployment (Zou 2021 , 2085), and thus was a primary point of focus in this project. When deployed on Ethereum, each smart contract is given its own address where the functions are stored and can then be executed when conditions are met (Ethereum 2024) (when a user performs a certain transaction, for example). As previously mentioned, this transaction is tracked by adding a block to the blockchain. A major feature of blockchain (and by association, smart contracts) is immutability; once a block is added to the blockchain, it cannot be altered. The same is true with smart contracts. Since smart contracts are deployed with the blockchain code, once they have been implemented, they cannot be changed.

The application of smart contracts in blockchain can be far reaching and vary greatly between implementations due to the fact that the developer is able to write the code from scratch to achieve the desired functionality. A few example applications of smart contracts include vulnerability detection, transactional privacy, data management (access, sharing), and tracking assets (Khan 2021, 2906-2914).

In this project, the smart contract is implemented in a blockchain that is designed to securely share files in a medical scope. This blockchain uses the InterPlanetary File System to store files off-chain, however there is little to no access control built into IPFS or the base blockchain deployed. The smart contract focuses on access control for users attempting to access the blockchain as well as user file permissions. This access control is automated within the blockchain by being deployed on each node rather than a central server, allowing nodes to validate the transactions using blocks in the chain. The value of implementing access control in this way, for this project, is that with validation occurring within each node and blockchain, there is no additional server need which both allows the network to remain decentralized but also prevents a lockout for all users should the central login server fail.

Development Process

The requirements for this project were described within the requirements that were set for the secure file sharing blockchain project developed in the CPE 488/498 Cybersecurity Senior Capstone courses. The smart contract access control requirements included the implementation of access control, specifically utilizing a smart contract to verify user access roles. These requirements were required to pass four tests, verifying that users could be given roles or have them removed, as well as only access files with the proper permissions. These tests included a user with a valid role successfully accessing a file they are permitted to access, a user without the valid role *not* successfully accessing a file they are not permitted to access, and the repetition of these tests after the users roles have been changed to a valid and invalid role respectively. These were the requirements and tests that guided the work of this project.

The first step of this project was planning. The initial scope of the CPE 488/498 project did not include the smart contract access control requirements. Once these were added, additional time was spent in order to research and plan the best way to go about implementing the smart contract concept. As previously discussed, smart contracts are a newer addition to blockchain technology, and as a result, there is minimal concrete support or documentation available. At the time, the CPE 488/498 group had planned to use Ethereum as the foundation for the development of the blockchain. This prompted the initial selection of Solidity as the language to be used for this project. While there are other smart contract languages, Solidity is the most commonly used and is the language of choice for Ethereum, however in a survey conducted of professional smart contract and blockchain developers, it was noted that even it has issues due to its immaturity and community support limitations (Zou 2021, 2092).

Despite the lack of support, development began using Solidity and the Remix Ethereum Integrated Development Environment, focusing on role based access control. The initial work in

Solidity was focused on attempting to create, assign, remove, change, and check roles. Additional functionality was attempted to specify the access level required for specific files used with IPFS and to trigger alerts when a “bad access” had been detected. Without a blockchain to deploy on during the development process (as the CPE 488/498 blockchain was still being designed and developed at this time), all work was performed in Remix in order to check for compilation and attempt deploying on a sample Ethereum blockchain. Unfortunately, due to the limits of Remix, community support and documentation, and the nature of the smart contract code that was being tested, development was forced to halt until the project blockchain was able to be deployed.

Another unfortunate setback early on in the completion of this project was the overhaul change of the target blockchain. After initial creation of the Solidity contract was ready for testing but halted by the lack of blockchain to test on, the team decided to change from an Ethereum implementation to a locally designed and implemented blockchain. This change overall benefited the team, however due to the vast difference in the deployment of Ethereum from the structure and implementation of the locally designed blockchain, the use of Solidity was no longer possible. Due to this fact, the access control features needed to be reworked to be deployed on the new chain.

When reevaluating the plan for the smart contract access control after the blockchain changes, Python was selected as the language for the smart contract implementation. This choice was made as Python was also chosen as the language to be used to implement the blockchain itself. While not typically used for smart contracts deployed on a platform like Ethereum, Python is a fairly mature, well documented and supported language that could interact with the blockchain in order to automate functionalities on each node. All of these reasons would allow for a more streamlined integration of the access control features and avoid incompatibility of both blockchain structure and language interactions. Since both elements were written in Python, the smart contract features could simply be called by the main blockchain functionality. This would prevent additional steps, dependencies, or language installs for the deployment of the final project. While other smart contract languages (beyond Solidity) were options, the CPE 488/498 group felt that the choice to use Python would be the most straightforward given this reasoning.

Project Functionality

After adjusting the plans for this project, functions to address account creation, login, logout, change role, and file permissions were developed. In addition to these functions, Validator and Account classes were created so data such as username and password could be kept up with and stored in the blockchain. Another consideration was the creation and function of different menus for each role, ensuring that only the proper options would be allowed for each user role. The details of each implemented function are discussed throughout the following section.

It is important to note that in the CPE 488/498 course, the scenario considered in the design of the blockchain is that of a hospital network, and as a result, the options for roles in this implementation are “admin,” “doctor,” and “patient.” If this project were to be applied in a different situation, these roles and their access permissions could be altered, however for the sake of this project and paper, the work will describe the access of the medical scenario roles specifically. In this case, the admin role is primarily designed to allow the assigned user access to manage user accounts as one might in a system administrator type position. The doctor role is designed to allow for the download and upload of files with selection of a (patient) user to give access permissions to for each upload. The patient role allows for the user to download and upload files. Upon login to the network of the main blockchain program, a check is run for the user's role, ensuring that the proper menu gets launched as different menu option buttons will be available based on the returned role.

List Users and getAccounts

The “Lists Users” function is only available for admin users. It allows the admin user to view a list of all users that exist in the system. This functionality is accomplished through a function called getAccounts. This function has no parameters or returns, but instead it updates two global lists maintained by the node based on users listed in the blockchain. The two lists that are updated in getAccounts are accountNames and accounts. AccountNames maintains just the username attribute of all the users found in the blockchain and is used for graphical elements where only the names need to be shown (such as with “List Users”). The accounts list maintains a list of all the account objects found within the blockchain. These are maintained separately as a result of integration and for the ease of processing for use in the graphical user interface. In getAccounts, the blockchain is traversed in reverse order (starting from the most recent block). During this traversal, any block with a transaction type of “Update_User” or “Create_Account” is noted and the relevant information is recorded into the respective lists. When updating these lists, a check is performed prior to addition to ensure that the user has not already been listed to prevent duplicates due to one or more updates to the account role. Creating the list this way also ensures the correct role is selected since the blockchain is immutable and it is important that only the most recently assigned role is applied to the user. The getAccounts function is used to support other functionality as described in the Create Account, Change User Role, Login, and Upload sections.

Create Account

The “Create Account” function is only available for admin users. It allows the admin user to enter information for a new user. This information includes a username, password, legal name, and a role. When creating this account, it is ensured that no user can be without a role through the use of a dropdown menu in the graphical user interface. This makes sure the created user (and any user in the network) will always have explicit permissions and only have access to what is permitted. It is also verified that no other user already exists with the entered username by calling getAccounts and comparing the input. An error message is returned if the username is already assigned. Once the information is successfully and correctly entered, an Account object is created and passed to the blockchain to be added to the record. The transaction type of this action is “Create_Account”.

Change User Role

The “Change User Role” function is only available for admin users. It allows the admin user to edit the account object for a selected user by changing the user’s role. The admin selects the name of a user from a list found by parsing the blockchain with the getAccounts function, then selects a role option from a drop down menu (to ensure a valid selection and simplify the selection process). The role choices are the same as in the rest of the system (admin, doctor, patient). The admin user is able to change the role for any user in the system except themself. This is done to ensure there is always at least one admin user in the system and prevent a lockout, since there must be a second admin in order to change the role of the first. Once the choices have been made and checks performed, the information is sent to the blockchain as an account object. The transaction type of this action is “Update_User.”

Login

The login function is available to users of all role types. Successfully completing the login process allows a user to access the blockchain and file features seen in the rest of the project. Login begins by calling getAccounts to get a recent list of users listed in the blockchain. This is dependent on the design of the CPE 488/498 blockchain as the blockchain was designed to be sent as soon as a user connects to another node in the blockchain network. By adding this smart contract functionality to be triggered immediately after connecting to the blockchain network, user authentication is able to protect unauthorized users or malicious actors from accessing the blockchain. It also allows for the use of the blockchain rather than a central server to perform credential validation. After updating the list of users, the login credentials are compared to those in the accounts list that were identified. If the entered username and password match those found in a singular account object, a validLogin flag is set. This flag indicates a successful login and grants the user access to the rest of the program features.

Logout

The logout function is available to users of all role types. Once a successful login has been completed, the logout option is available to a user. When a user attempts to trigger a logout, a check is performed to ensure the user does not have any “candidate blocks” waiting to be added to the blockchain. Candidate blocks are blocks that still require validation to be added to the blockchain. The designers in the CPE 488/498 team set forth the unofficial requirement that all candidate blocks must have been dealt with prior to a user logging out, and thus this check is performed first. After it is determined that there are no outstanding candidate blocks, the user is removed from the list of active validators to ensure that the inactive user’s system is not tasked with adding or validating blocks (another requirement from the designers). Once this has occurred, a flag is set indicating the validator has been removed and the user can be returned to the login menu.

File Access

File access has two main focuses: upload and download. The blockchain is designed to use the InterPlanetary FIle System to store files off the chain, however the upload and download occur within the chain’s code. Each upload transaction stores a hash that can be used by the download transaction to retrieve a specified file from IPFS, however there is no built in access control to this functionality. As a result, it must be ensured that only users who are authorized to access each file are permitted to do so. The first way to ensure proper file access is in only allowing doctor and patient role users to interact with files. Admin roles have no reason in the medical scenario to interact with files and thus are not allowed to. This is initially done by a role check completed prior to presenting menu options for the user interface. Additionally, file access is enforced by maintaining an “accessList” for each file object.

Upload

The upload process consists of selecting the file to be shared, uploading it to IPFS, and creating a file object to be given to the blockchain for recording with a transaction type of “Upload.” A file object has four attributes: file hash (the IPFS hash), file name, author name (username of user who uploaded the file), and accessList. When uploading a file, patient users simply pick the file they wish to upload at which time the file is uploaded and the user’s username is added to the accessList attribute for the file. When uploading a file as a user with the doctor role, the user must select a user from a provided menu to grant access. The options provided to the doctor user are other users in the system returned by getAccounts who have the role of patient.

The reasoning behind this choice of access list implementation was based primarily off of the medical scenario the CPE 488/498 project was based in. Since patient files must be kept confidential under HIPAA laws, it follows that a doctor user should only be able to select one user to share the file with (essentially attaching to that patient’s record). It also follows that any doctor should be able to access a patient's medical record as the medical professional working with a patient may not always be the same within the hospital. For this reason each user with a doctor role should not have to be listed explicitly, rather any user with a doctor role should be allowed to access any patient files. This resembles a medical system with an electronic record system. A record of file interaction is maintained through the blockchain.

Download

The previously mentioned record of file access is maintained in the blockchain through the download functionality. Unlike the upload process, the download transaction process looks the same on the front end for both patient and doctor users. The user will be able to select a file to access from a provided list. In the backend, when a user triggers a download transaction, if they are found to be a patient user, a check is done to compile a menu list of each file for which that user’s username is listed in the accessList attribute of the file. If the user has the doctor role, then a list of every file in the blockchain will be provided. After the completion of this transaction, the download function will provide the downloaded file information to the blockchain for the record with a transaction type of “Download.”

This ensures that users are only able to potentially interact with files that they are allowed to, reducing the risk of improper access to extra files. If a user's role is changed, their access will change as well. For example, if a patient user has their role changed to doctor, they will gain access to all of the files in the system. If that user’s role is then changed again to be a patient, they will only have access to the files that they uploaded prior to having the doctor role, or any files for which a doctor user selected their username as the patient. If the user was changed to have an admin role, they would lose access to all files through the blockchain until returned to the patient or doctor role.

Integration

After the creation of the discussed functionality, these features were integrated into the main CPE 488/498 blockchain project. Due to the blockchain code design, a majority of these features were built into the graphic user interface directly. Despite being worked directly into the GUI, the smart contract access control concept maintains functionality on individual nodes and is triggered by user interactions with GUI elements and blockchain interactions throughout the network.

Conclusion

Self Assessment

Overall, the completion of this project is satisfactory. All of the requirements originally set have been tested and met. It would have been interesting to implement these features as originally intended on an Ethereum based blockchain using Solidity in order to gain experience in such a niche and new technology, however with the several setbacks and changes to the original plan, the result is about what was expected. The CPE 488/498 blockchain project group is also satisfied with the impact this portion of the project has had on the overall function of the system. The tasks set out by this project could have been completed somewhat earlier but remained mostly on track with the CPE 488/498 timeline. Despite the main blockchain development changes, the timeline for completion of this portion was begun on time and was ready for integration in a timely manner. Better communication with the CPE 488/498 group in terms of progress, specific element design and implementation, and expectations of each element could have improved the timeliness and overall quality of the project especially in terms of the sophistication of portions such as the smart contracts. This team member feels that all the group members should have been more involved in the initial blockchain development in order to ensure decisions such as the shift from Ethereum to a locally designed and developed system had been better understood by all members, as well as the impacts such a choice would have on the overall implementation.

Lessons Learned

As far as knowledge specific to this project, lots was learned about blockchain development, structure, implementations, and other relevant concepts. The use of Solidity in the beginning of the project was very interesting in terms of learning more about the complicated inner workings of blockchains and their smart contracts. As far as other less technical lessons learned, there was lots to discover about working with a team while still developing a separate portion of the project. As discussed in the Self Assessment section of this paper, this team member feels that many issues could have been resolved sooner had the CPE 488/498 team learned to better communicate as a group and made informed decisions together. This is not the fault of any particular individual but a skill the team had to develop as a whole. The communication of decisions that would impact other members' implementations (and communication in general) greatly improved over the course of the project. Overall this project was highly beneficial for learning about the process and details involved in a major team development project.

An additional lesson learned over the course of this project was flexibility. One must be flexible when working with code, new technology, and a team. Due to the obscurity of blockchain development and concepts, this project was very challenging. With so much information being hard to find or interpret, if it existed publicly at all, this project had to shift to using different libraries, features, languages, or ways of completing a task then were often intended. While often frustrating, these shifts were to alleviate issues that would completely halt progress, making them a necessary choice. Without being flexible in terms of the way code was written or when working with a team, this project may not have been as successful.

Future Work

If more time were allowed or work were to continue on this project, there are many improvements or additional functions that could be added. A major feature that should be added is a penalty system for the Proof-of-Stake consensus mechanism implemented in the main blockchain program. While Proof-of-Stake was implemented in the main blockchain project, one extra feature of this consensus mechanism is a penalty system that can be implemented in order to deter or punish bad actors. Due to the constraints of the main system and course, this was not a priority or goal for this project, however going forward would be a great security measure to add through smart contracts that would benefit the whole blockchain. Since each validator has a certain amount of stake when they join the network, this could be associated with the user and be updated by the smart contract when triggered. A scheme for detecting malicious actors or actions as well as an appropriate penalty value could be developed.

An additional feature that would benefit this project may be the ability to update additional user information. In the current implementation, only the role is able to be updated for a user and it can only be changed by an admin. The ability for a user to be able to update their own password information would more closely resemble a typical login system as well as increase security as the admin user would not need to set each user’s password themself. In addition to this, maintaining additional information about a user such as their email, phone number, home address, or insurance information would be necessary in a hospital system. Adding the ability to maintain and update this information would be vital in a real world scenario as well as give more markers to check for duplicate accounts.

In general, this implementation is vague in terms of the required functionality. When designing this project only basic features for a file sharing system with access control were defined. As a result, only a few roles were created, tested, or given a purpose. In future works, more roles with additional functionalities could be added to account for other roles in a hospital scenario, or the system could be adapted to a different context with different roles entirely. If this were the case, more smart contract features would be needed to account for the permissions of these new roles.

Finally, if continuing with this work, great consideration should be given to the security concerns present with the use of smart contracts in blockchain. Like any other technology, especially those in their infancy, smart contracts have a number of security concerns. These concerns vary by the purpose and integration of a contract and could be exploited by malicious actors. Despite the setbacks, challenges, and frustration of this project moving from Solidity to Python, a number of professionals and documents refer to the insecurity of development using smart contract languages due to their immaturity, complexity with minimal documentation, lack of "good code practice and procedure," and more (Zou 2021, 2092-2093). This may indicate a less insecure or at the very least an easier to secure smart contract in the Python implementation over that of the Solitiy. Further adaptations of this work should investigate the specific security risks associated with this implementation and how such risk could be mitigated or accepted. An analysis of the risks associated with a Python implementation versus those of a Solidity implementation could be used to justify a shift to a smart contract language as necessary for application on other blockchains.

Works Cited

Ethereum. 2024. “Introduction to Smart Contracts.” Last modified April 22, 2024. https://ethereum.org/en/developers/docs/smart-contracts/.

IBM. n.d. “What are Smart Contracts on Blockchain?” Accessed April 10, 2024. https://www.ibm.com/topics/smart-contracts.

Khan, Shafaq et al. 2021. “Blockchain Smart Contracts: Applications, Challenges, and Future Trends.” *Peer Peer Netw Appl*. vol. 14: 2901-2925. http://doi.org/10.1007/ s12083-021-01127-0.

Nguyen, Dinh et al. 2019. "Blockchain for Secure EHRs Sharing of Mobile Cloud Based E-Health Systems." *IEEE Access*, vol. 7: 66792-66806. https://doi.org/10.1109 /ACCESS.2019.2917555.

Zou, W. et al. 2021. “Smart Contract Development: Challenges and Opportunities.” *IEEE Transactions on Software Engineering*, vol. 47, no. 10: 2084-2106. https://doi.org/ 10.1109/TSE.2019.2942301.