Cybersecurity and OSFA Architecture

Patrick Jungwirth
US Army Research Lab
Aberdeen Proving Ground,
MD, 21005
001.410.278.6174
patrick.w.jungwirth@mail.mil

Abdel-Hameed Badawy
New Mexico State University
1125 Frenger Mall
Las Cruces, NM 88003-8001
001.575.646.6476
badawy@nmsu.edu

ABSTRACT
In this paper, we will present an overview of the patented OS Friendly Microprocessor Architecture (OSFA) and introduce its cyber security features. We are interested in feedback from cyber security professionals, microprocessor developers, and operating system (OS) developers about the hardware computer security features in the OSFA.

The OS Friendly Microprocessor Architecture is based on a cache bank memory pipeline. The cache banks and memory cells incorporate Unix-like file permission bits to create a hardware level computer security mechanism. A trusted OS and OSFA hardware system form a separation kernel. The trusted OS configures the security tags (access permission bits) for each cache bank. Any attempt to access a resource outside the bounds of the sandbox defined by the security tags raises a hardware exception handled by the trusted OS.

We present our plans for developing a field programmable gate array (FPGA) softcore OSFA based on the RISC-V architecture for test and evaluation.

Categories and Subject Descriptors
C.5.3 Computer Systems Organization – Computer Systems Implementation

General Terms
Performance, Design, Security.

Keywords
Cyber security, computer architecture, hardware computer security, trust, privacy.

1. INTRODUCTION
The 1970's telephone network used in-band signaling where control information (tones) and data (voice) where combined together. There was no authentication of control information. The papers by Weaver et al. [1], and Breen et al. [2] provided the technical details of the telephone network control system. Any electronics hobbyist could easily build a blue box [3] to control the telephone network. A blue box built by Steve Wozniak is on display at the Computer History Museum [4]. The 70's telephone network is a good example (oxymoron?) of how *not* to build a secure system. Control information must be separated and isolated from untrusted software and untrusted users.

In the current architectures from well-known vendors, such as Intel [5], AMD, ARM, *et al.*, there are no mechanisms to separate shared resources across executing threads and processes. A virtual machine (hypervisor) can provide a security layer; however, traditional hypervisors are slow and have limitations.

The goal of the OSFA [6-7] is to create an architecture where control information is isolated from untrusted users and untrusted software. As long as the untrusted software does not exceed the limits set by the hardware security mechanism, the software will run with no performance overhead.

Microprocessor designers and OS developers have not worked closely enough together to develop a best of both worlds system [8-9]. Microprocessor hardware always leads operating system development. We need to find simple hardware features that can significantly reduce the amount of software required for an operating system. In "Separation Kernels Enable Rapid Development of Trustworthy Systems," Keegan [10] wrote: "Compared to trying to separate applications in an operating system, a separation kernel can remove roughly a million lines or more of complex code to achieve the same capabilities." We would like to add < 10 % more hardware to save thousands to orders of magnitude more lines of computer code.

2. OSFA INTRODUCTION
The OSFA architecture consists of 4 cache bank memory pipelines connected to an instruction execution pipeline. For light-weight threads, the on-chip cache hierarchy provides near instantaneous context switching times. The cache banks provide fast pipelined and parallel read and write operation to the caches while the microprocessor's execution pipeline is running instructions. The cache bank controllers provide arbitration to prevent the memory pipeline and microprocessor's execution pipeline from accessing the same cache bank at the same time. This separation or virtual fence allows the cache memory pages to transfer to and from the L1 caches while the microprocessor pipeline is executing instructions. Figure 1 introduces the key features of the cache bank memory pipeline architecture in the OSFA [7-9]. The cache controller and cache banks block contains 3 groups of cache banks.

Figure 2 illustrates the OSFA microprocessor architecture. The OSFA is an extended Harvard architecture. There are 4 cache bank memory pipelines. The active cache bank is connected to the microprocessor's execution pipeline. The swapping cache bank is connected to the DMA controller. Idle cache banks are not connected. The cache bank controller manages the cache bank

NCS'17, June 7-8, 2017, Huntsville, AL, USA.

connections. The cache bank controller manages connecting and disconnecting cache banks from the execution pipeline and DMA controller. The cache bank configuration allows for running a process from the active cache bank and copying or loading a swapping cache bank in parallel. The cache bank controller provides for single cycle context switching. The register cache bank pipeline and pipeline state cache bank pipeline save the process state proving for a single cycle context switching. A more detailed context switch description is presented in [7-9].

3. Hardware Security Framework

As illustrated in Figure 1, the OSFA's hardware security mechanism [6-7] consists of cache bank security headers and memory cell permission bits (security tags) similar to Unix file permission bits. The OSFA security architecture is divided into access level similar to Unix protection rings. Figure 3 (left side) shows the OSFA's security hierarchy. The security hierarchy [9] is a directory like structure where inside security containers, access level *n* has visibility to the bottom level.

A trusted OS configures the cache bank security headers and memory cell permission bits. The security headers and memory cell permission bits define the limits for a hardware level sandbox. In Figure 3, a process sends a malicious pointer to an OS DLL function call. The malicious pointer tries to access a memory page with cache bank header is set to **RWM** (Read, Write and Modify are not allowed). A hardware level exception is raised and the trusted OS handles the exception and cleans up the malicious process.

4. OSFA Shared Memory Examples

In Figure 4, we present some shared memory page examples for the OSFA architecture. There are 4 processes shown as process IDs (PID) of `PID = 0x1a`, `0xbc`, `0x3d`, and `0x41`. The shared memory page security header shows the access rights for each PID. Each memory cell also has permission bit fields (security tags). The shared memory operations are allowed as long as the PID has access rights. For a process to access a memory cell, it must have access rights in the security header and the memory cell permission bits must allow the operation (read, write, modify, *etc.*). When the PID does not have access rights (lower level diagrams), a hardware exception occurs and is handled by the trusted OS. For example, the process, `PID = 0x41`, has memory page access rights: **RWM** (Read = allowed, Write and Modify = not allowed). When the process, `PID = 0x41`, attempts to read the third memory cell with permission bits **RWM** (Read, Write and Modify = not allowed), a hardware level exception is raised and handled by the trusted OS.

5. FPGA Softcore Implementation Effort

In Figure 5, to evaluate and compare the OSFA to a conventional microprocessor, we are developing a softcore OSFA implementation based on the well-established RISC-V [11] processor. We will use the RISC-V development tool chain (compiler, linker, and debugger) to create code for both the conventional CPU and OSFA processor. A benchmark suite consisting of CPU performance tests and security tests based on DARPA Cyber Grand Challenge efforts will be developed to compare the OSFA to a conventional processor. We will run the same binaries on both CPUs for an unbiased performance comparison. We will also be developing a full system cycle-accurate simulator [12-13].

6. Conclusion

The OSFA is a step towards cybersecurity in hardware. In the past, more resilient cyber security has required virtual machines (hypervisors) to create execution sandboxes for untrusted software. The OSFA uses a trusted OS to configure the hardware security mechanism to define the bounds of the execution sandbox. Software is allowed to run with no security overhead as long as it does not exceed the limits of the sandbox. At run time, an operation that exceeds the limits of the execution sandbox, instantly raises a hardware level exception preventing the malicious process from accessing anything outside the sandbox.

We have outlined our plans to compare the performance of the OSFA to a conventional processor. Performance and security testing results will be published in the future.

7. ACKNOWLEDGMENTS

The authors wishes to thank RDECOM community for the opportunity to develop the OS Friendly Microprocessor Architecture and improvements. The authors also would like to thank our colleagues at the Army High Performance Computing Research Center at Stanford University.

8. DISCLAIMER AND COPYRIGHT

9. REFERENCES

[1] Weaver, A. and Newall, N., "In-Band Single Frequency Signaling," Bell System Technical Journal, 33(6), 1309-1330, (Nov 1954). https://archive.org/details/bstj33-6-1309

[2] Breen, C. and Dahlbom, C., "Signaling Systems for Control of Telephone Switching," Bell System Technical Journal, 39(6), 1381-1444 (November 1960). https://archive.org/details/bstj39-6-1381

[3] Wikipedia.org, "Blue Box," (March 2017). http://en.wikipedia.org/wiki/Blue_box.

[4] Computer History Museum, "Blue Box," (March 2017). http://www.computerhistory.org/collections/catalog/102713487

[5] Intel.com: Enclave. https://software.intel.com/en-us/blogs/2016/06/06/overview-of-intel-software-guard-extension-enclave

[6] Jungwirth, P., and La Fratta, P. OS Friendly Microprocessor Architecture, US Patent 9,122,610, September 2015.

[7] Jungwirth, P., and La Fratta, P. OS Friendly Microprocessor Architecture: Hardware Level Computer Security, SPIE Defense + Security – Cybersensing Conference, Baltimore, MD. 2016.

[8] Jungwirth, P., Computer Security Framework and Hardware Level Computer Security in an Operating System Friendly Microprocessor Architecture, US Patent Application, April 2017.

[9] Jungwirth, P., and Barnett, T. Computer Security and the OS Friendly Microprocessor Architecture, SPIE Defense + Security – Cybersensing Conference, Anaheim, CA. 2017.

[10] Keegan, W. "Separation Kernels Enable Rapid Development of Trustworthy Systems," COTS Journal, pp. 1-3, | February 2014.

[11] RISC-V. www.risc-4.org.

[12] Magnusson, P., et al. "Simics: A full system simulation platform." Computer 35.2 (2002): 50-58.

[13] Binkert, N., et al. "The gem5 simulator." ACM SIGARCH Computer Architecture News 39.2 (2011): 1-7.

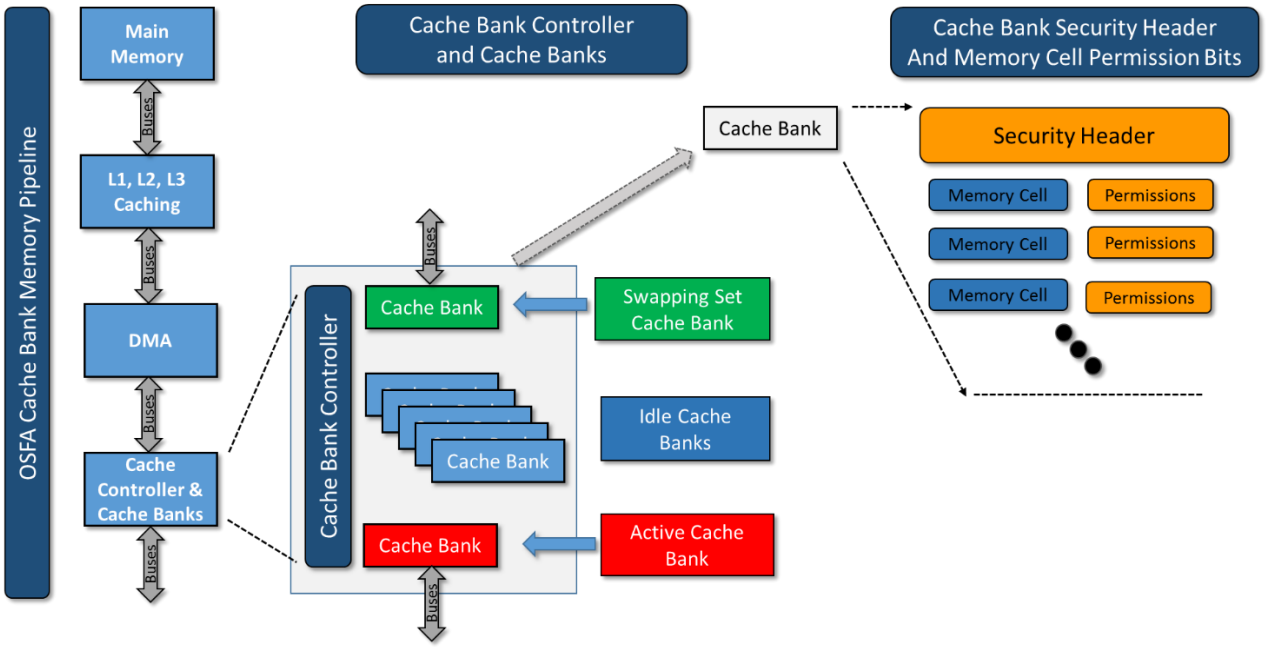


Figure 1. Cache Controller and Cache Bank Memory Pipeline Architecture

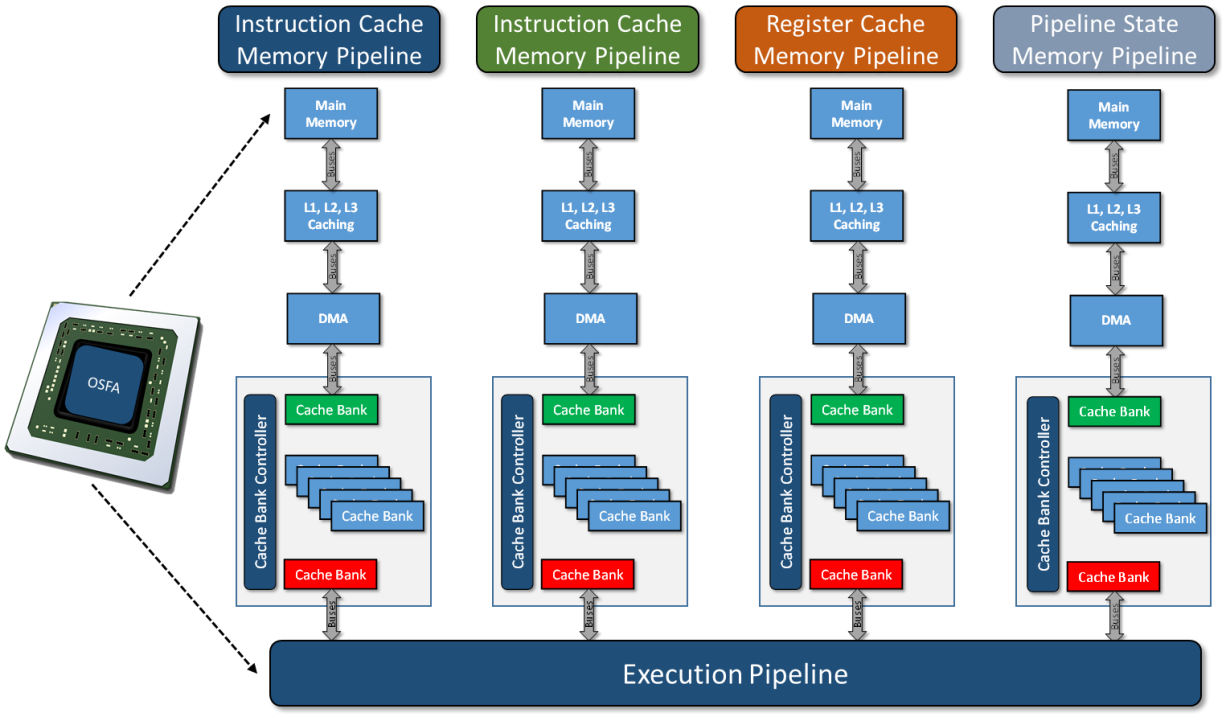


Figure 2. OS Friendly Microprocessor Architecture

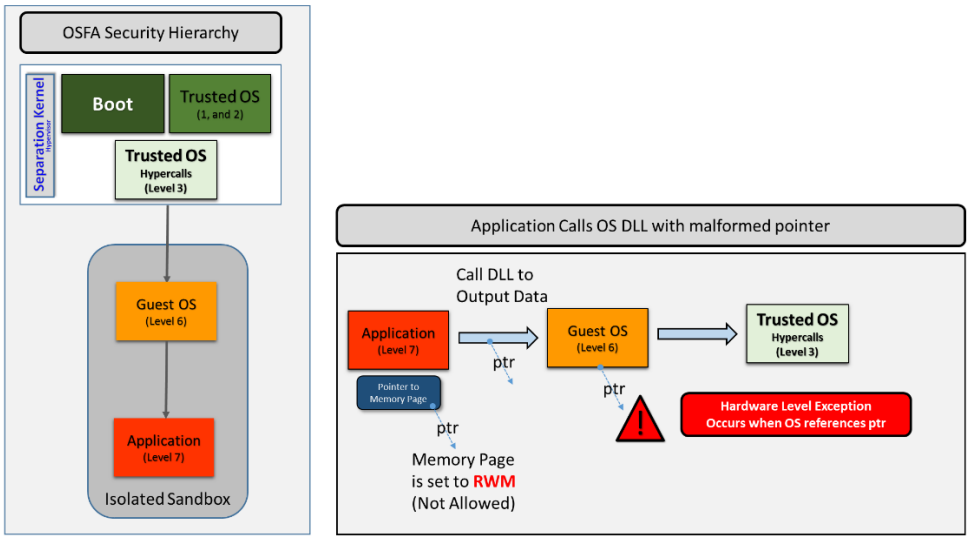


Figure 3. OSFA Security Hierarchy and Hardware Exception Handling.

We illustrate 4 shared memory examples. Each shared memory page contains a security header which contains the access rights for each process. Each memory cell also contains permission bits. For a process to access a memory cell, it must have access rights in the security header and the memory cell permission bits must allow the operation (read, write, *etc.*).

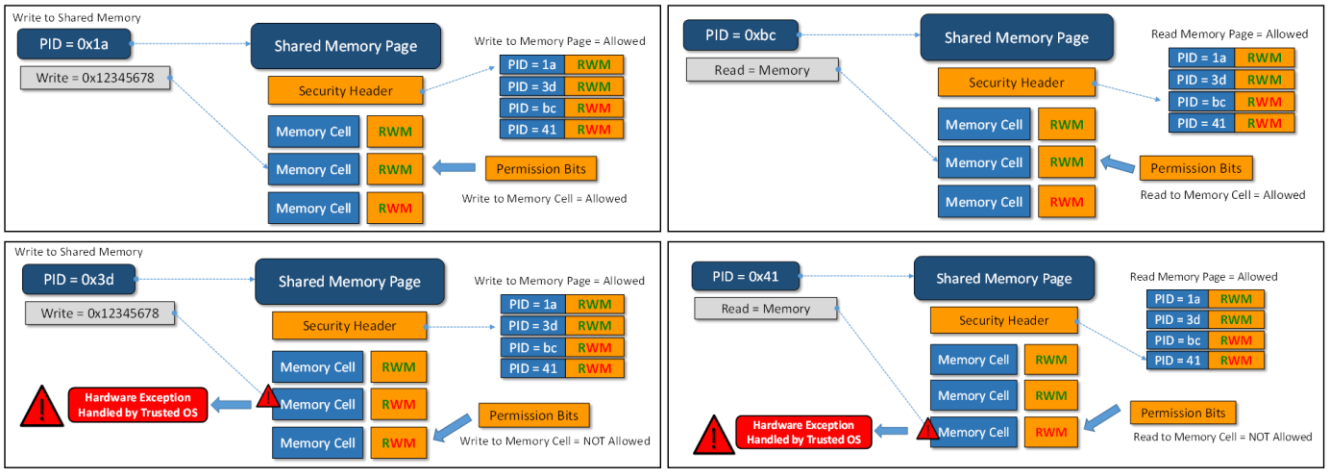
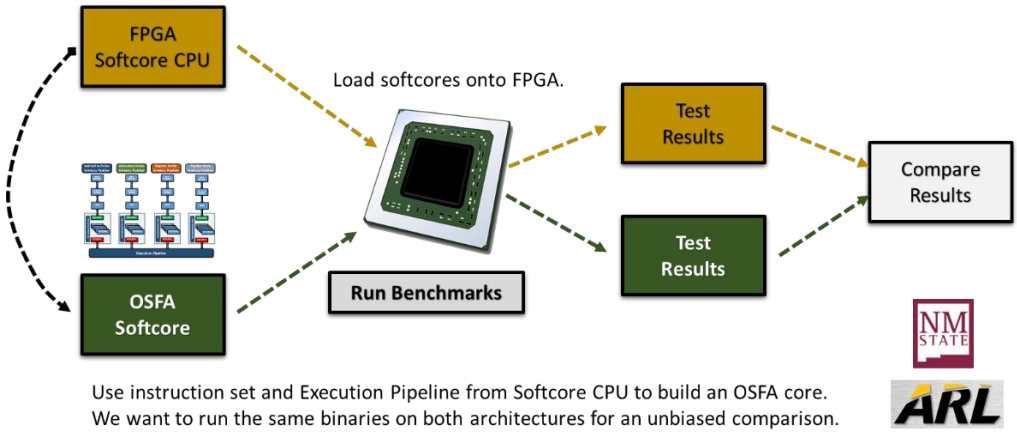


Figure 4. OSFA Shared Memory Security



Use instruction set and Execution Pipeline from Softcore CPU to build an OSFA core.
We want to run the same binaries on both architectures for an unbiased comparison.

Figure 5. OSFA Softcore Development and Performance Testing