Secure Computing Architecture:

A Direction for the Future -- The OS Friendly Microprocessor Architecture

Patrick Jungwirth, PhD [1]; Philip Chan, PhD [2]
US Army Research Lab
[1] Computational and Information Sciences Directorate
[2] Survivability/Lethality Analysis Directorate
Aberdeen Proving Ground, MD, USA
patrick.w.jungwirth.civ@mail.mil
philip.w.chan2.civ@mail.mil

Hameed Badawy, PhD
Klipsch School of Electrical and Computer Engineering
US Army High Performance Computing Center
New Mexico State University, Las Cruces, NM, USA
badawy@nmsu.edu

***Abstract* — We present a short historical review of computer security covering computer architectures and operating systems. Tools and techniques for secure architectures have been well researched; however, attention has only focused on microprocessor performance.**

A new direction in computer security is microprocessor and OS co-design. The co-design approach needs to leave the insecure von Neumann bus architecture in the past. Co-design also needs to consider the application space. Embedded systems do not require extensive sharing rules.

Co-design approach has renewed interest in tagged computer architectures from the 60's for computer security. Multics OS pioneered advanced computer security in the late 60's. Multics was considered too complicated for general computing applications. An "object oriented" microprocessor, i432 (with hardware protection features), was developed in the 70's. As a research processor, i432 was a great success; however, 80's semiconductor design rules doomed the performance. The mindset of 'too complicated,' from past computer generations, is holding computer security back.

Computer security needs to continue researching co-design and rediscover past wisdom. We will explore a new research direction: the OS Friendly Microprocessor Architecture (OSFA). We present three innovations that will help solve some of the security issues facing cybersecurity. OS Friendly Microprocessor Architecture is a thread-safe architecture. The architecture's pipeline memory also addresses the context switch overhead problem and helps reduce OS complexity.

Keywords — Computer Security, Cybersecurity, Microprocessor, OS, Co-Design, Tagged Memory, Memory Pipeline, OS Friendly Microprocessor Architecture

I. Computer System History

What is old is new again. Many of the cyber security problems of today, were studied and solved back in the '70's. As illustrated by Multics, circa 1969-1973, and the i432 microprocessor, circa 1981, excellent computer security has been demonstrated; however, until recently, the only marketing hype was muscle car top speed.

The Rice Computer, R1, circa 1959, used a 2 bit tagged memory architecture for debugging [1]. Other notable tagged memory computers include: Rice R2, 1972, [1], Burroughs B6500, 1969, employed a 3-bit type tag field [2], and Telefunken TR440, 1970, used a 2-bit memory type tag [3]. Today, there has been a renewed interest in tagged architectures for computer security.

In the classic 1975 computer security paper, Saltzer and Schroeder [4] defined the properties for a secure computer system. The most significant property is the *principle of least privilege*: only give the user or application the absolute minimum privileges required. From the '80's until recently, 1/4 mile top speed was the only metric of interest. Today, there is renewed interest in Saltzer and Schroeder's security properties.

Computer security requires a negative proof mindset. According to Dijkstra, "[Software] testing shows the presence, not the absence of bugs" [5]. Testing cannot prove software is perfect. A formal proof-of-correctness [6]-[7] is required to demonstrate the highest level of computer security assurance.

Computer security requires a leave-no-stone-unturned approach (negative proof mindset). A strong defense against one class of cyber-attacks does not guarantee good computer security. The Greek army overcame a decade-long stalemate of Troy using a social engineering attack. The city of Troy welcomed the Trojan horse and its "attack package" [8]; thereby giving the Greek army victory. A cyber attacker only needs to find one security vulnerability to enter a castle and take the kingdom. In the cyber realm, you must understand cyber-attacks to defend against them: "the best defense is a good offense."

> *In my view, a defender who doesn't know how to attack is no defender at all.*
>
> W. Earl Boebert, Computer Security Pioneer [9]

Information assurance consists of the five properties: integrity, authenticity, confidentiality, traceability and availability. The properties define privacy and auditability of information.

A. 1970's Telephone Network

Poor network security, "security through obscurity" (do not publish any technical documents), is illustrated by 1970's era in-

This work was partially supported through funding provided by the U.S. Army Research Laboratory (ARL) under contract No. W911NF-07-2-0027.

band signaling telephone network. Without any authentication, it was impossible to tell the difference between a prankster and the phone company. Two separate publications, Weaver et al. 1954 [10], and Breen et al. 1960 [11] provided the technical details to control the "open door" telephone network. The security issues were compounded since any hobbyist could easily build a "bluebox" [12] to control the telephone system. In-band signaling gave everyone administrator privileges.

In the cryptographic world today, open source algorithms are considered essential for peer review, and only the encryption key is undisclosed. NIST has published the Advanced Encryption Standard (AES) [13] algorithm, and anyone can review the algorithm and software codes.

B. von Neumann Architecture

The von Neumann bus architecture has its origins back in the 1950's. In a von Neumann architecture, there is no difference between program instructions and data. Instructions and data share the same bus architecture and memory. As early as 1972, Feustel [1] points out the security flaw.

> *In the von Neumann machine, program and data are equivalent in the sense that data which the program operates on may be the program itself. The loop which modifies its own addresses or changes its own instructions is an example of this. While this practice may be permissible in a minicomputer with a single user, it constitutes gross negligence in the case of multi-user machine where sharing of code and/or data is to be encouraged.*
>
> E. Feustel 1972 [1]

Cowan 2000 [14] illustrates the ease of buffer overflow attacks in a von Neumann architecture: "By overflowing the buffer, the attacker can overwrite the adjacent program state with a near-arbitrary [2] sequence of bytes …" Wagner et al. 2000 [15] states "Memory safety is an absolute prerequisite for security, and it is the failure of memory safety [protection] that is most often to blame for insecurity in deployed software." In a von Neumann architecture, programs and instructions share the same computer resources. The sharing violates Saltzer and Schroeder's security principles.

Podebrad, et al. 2009 [16] analyzed information assurance attributes (integrity, authenticity, confidentiality, traceability, and availability) for the von Neumann architecture. Podebrad, et al. concluded: a no-execute bit was insufficient and "Only a fundamental re-evaluation of the objectives i.e. the optimization of performance without compromising security would lead to significant improvement."

A Harvard architecture completely separates program instructions and data at the hardware level. Program instructions and data do not share resources. The Harvard architecture enforces "least privilege principle" at the hardware level. The two separate memory buses allow for reading a program instruction and data at the same time.

C. Tagged architecture

In 1973, Feustel [17] examined the advantages of tagged memory. He points out the vast economic disparity between system hardware and developing software.

> *Hardware costs have dropped radically while software cost and complexity has grown [63], [66]. We must now reconsider the balance of hardware and software and to provide more specialized function[s] in hardware than we have previously, in order to drastically simplify the programming process [1] - [4].*
>
> Feustel 1973 [17]

Gehringer and Keedy in 1985 [18] present a rebuttal of tagged computing architectures. Gehringer and Keedy did not anticipate using hardware dynamic typing (memory tagging) for computer and cyber security applications. Memory tagging provides for real-time type checking and least privilege enforcement across computations.

> *… fundamentally [memory tagging] it is just a mechanism for the architectural implementation of dynamic typing. … The important point is that all the aims of tagging can be efficiently achieved without hardware tagging …"*
>
> Gehringer and Keedy in 1985 [18]

Forty plus years after Feustel's 1973 paper and billions of transistor per chip, software costs are still climbing! Tagged computer architectures have emerged as a solution to the software cannot secure software problem afflicting the computer world today. As Yogi Berra would have said: "It's déjà vu all over again." Tagged memory architecture papers [19]-[28] cover computer architectures, security policies, secure pointers, information flow verification, and dynamic type checking for computer and cyber security applications.

D. i432 Microprocessor

The i432 microprocessor introduced several innovations to the microprocessor world; including, an "object-oriented"[1] approach to microprocessor hardware. The semiconductor design rules from the early '80's led to trade-offs that drastically reduced performance. Commercially, the i432 was a failure; however, as a research processor, the i432 pioneered some hardware based security concepts [29].

> *As a research effort the 432 was a remarkable success. It proved that many independent concepts such as flow-of-control, program modularization, storage hierarchies, virtual memory, message-passing, and process/processor scheduling could all be subsumed under a unified set of ideas.*
>
> Coldwell and Jensen 1988 [29]

Coldwell and Jensen addressed the shortcomings in the i432 architecture. Surprisingly, the "object-oriented" approach did not reduce the performance. At the time, the i432 was an enormous design requiring a multi-chip solution. Today with billion plus transistor integrated circuits, the concepts pioneered by the i432 need to be revisited for more advanced hardware based cybersecurity.

E. Co-Design: CPU and Operating system

The TIARA [30] co-design project at MIT developed a microprocessor and OS as a system to take advantage of metadata security tagging in hardware to strongly type and compartmentalize data. To achieve high computer security assurance, TIARA decomposes the OS into small components

[1] Today, object oriented is synonymous with the software development methodology of the same name.

suitable for machine formal verification. TIARA implements least privilege at the OS level using mutually suspicious OS components (zero kernel OS). A zero kernel operating system rejects the root level access monolithic kernel concept, distributes authority across several threads, and adheres to least privilege across the entire OS.

> *Today's computer systems are practically and fundamentally insecure. … The roots of this problem are not a mystery: (1) Our hardware is based on outdated design compromises … (2) the core of our software systems deliberately compromise sound security principles … and (3) the computer system artifacts we build are beyond our science and engineering to analyze and manage.*
> TIARA [30] 2009.

TIARA enforces least privilege across calculations using metadata (security tags) and label propagation. Metadata and a security lattice determine least privilege. For example, untrusted data is security tagged as external, integer. The result of any calculation with a security tag of external, integer is the least privilege security tag of external, integer. Hardware security tagging and computation label propagation make Saltzer and Schroeder's security principles practical [30]: "Metadata-driven hardware interlocks make it practical to take the security principles of Saltzer and Schroeder [59] seriously." TIARA strongly enforces Saltzer and Schroeder principles of (1) complete mediation (authority is verified for all operations); (2) least privilege; and (3) privilege separation.

Separation kernels use hardware assisted virtualization (hypervisor techniques) to create secure compartments (execution sandboxes) to completely isolate the separation kernel from all applications. Previous systems [31-36] have used hardware assisted virtual memory and separation kernels to enforce computer security. Computer security for von Neumann architectures is limited to ingenious implementations utilizing the available system hardware. For example, Grace, et al. in "Transparent Protection of Commodity OS Kernels Using Hardware Virtualization" [35] creates a virtual Harvard architecture using hardware assisted virtualization and shadow page tables to isolate the "guest" OS from the hypervisor.

The TIARA ecosystem does not consider a root-of-trust. For secure transaction servers, security certificates are required to establish trust to a central authority and generate/verify digital signatures. For TIARA [30], a secure boot operation traceable to a certificate authority also needs to be considered.

DARPA released the Clean-slate design of Resilient, Adaptive, Secure Hosts (CRASH) [37] request for proposals in 2010. The program goals were to create systems highly resistant to cyber-attack, able to learn from cyber-attacks, and self-repair. A team led by BAE Systems supported by University of Pennsylvania, Northeastern University, and Harvard University proposed SAFE based on its previous TIARA project. The goal of "SAFE: A Clean-Slate Architecture for Secure Systems," [38] is a formally verified, clean-slate co-design for the entire computer ecosystem covering system hardware, OS, compilers, and development tools. SAFE just like TIARA is a security tagged architecture. To reduce the high cost of context switching, monolithic OS's place too much functionality in the kernel. SAFE uses the same distributed mutually suspicious OS components to limit context switching.

> *Conventional systems have two (or some fixed, small number of) security regimes—kernel and user. Kernel mode has complete access to all resources, … Furthermore, domain crossing — switching from kernel to user mode or vice versa — is considered expensive, which encourages pushing more functionality into single system calls rather than separating system functionality into lots of little functions.*
> S. Chiricescux, et al.: 2013 [38]

II. Computer Security Lessons Learned and Future Directions

We presented a short review of past cyber lessons learned and emphasize the importance of a clean-slate, co-design covering microprocessor, OS, and design for machine formal verification (proof-of-correctness). The seL4 (secure extended L4) operating system's proof-of-correctness used a machine generated formal proof [39]-[40]. The SAFE [38] project has also proposed machine formal verification.

We must leave the insecure von Neumann bus architecture behind and look beyond the Harvard architecture. We also need to consider past microprocessor architecture features. For example, the Zilog Z80 was developed in 1976 [41], and it incorporated a register file and an alternate register file for fast interrupt context switching.

We truly need to take an outside-the-box approach. With almost 10 billion transistors per chip, transistors are "free." High cost and difficulty developing software were pointed out way back in 1973 [17]. Fundamental architecture characteristics need to be set aside and take a block diagram approach and focus on co-design and machine formal verification.

> *The analysis of currently available computer architectures has shown that such systems offer a lot of security gaps. This is due to the fact that in the past hardware has only been optimized for speed - never for security.*
> Podebrad, et al. 2009 [16]

Figure 1 shows an example protected pointer [27], [42] with a 4-bit security tag field, a 24-bit limit field, and a pointer base field. The pointer base field provides the starting address to an array. The limit field specifies the length of the array. The protected pointer, called a fat pointer, provides protection against buffer overflows and illegal range pointer operations. Unfortunately, the fat pointer requires that we trust the compiler to generate the right limit field value. If a malicious user alters program code for a protected pointer type, the buffer overflow attack reappears.

We are faced with questions about our trust model. What do we trust and how much do we trust each part?

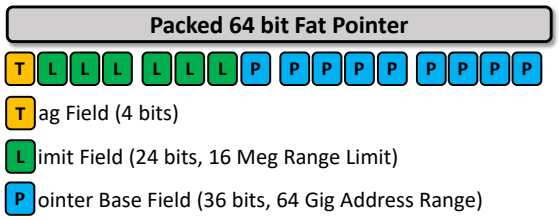

Figure 1. Example Fat Pointer

We know the following are **poor** security design decisions:

- Software cannot protect software
- von Neumann architecture is inherently insecure
- Do we trust compiler generated safe types?
- Monolithic OS
- Single point security failures
- Speed at all costs kills security

We know the following are **good** security design decisions:

- Hardware security tagging and computation label propagation make Saltzer and Schroeder's security principles practical [30].
- Least privilege for all software, OS, and applications.
- Security containers, where any operation outside the computer security container is prohibited. All operations inside the container cannot affect anything in other containers.
- With a clean-slate co-design approach; we need to:
  - Review the historical lessons learned and
  - Look beyond traditional computer architectures –
    - (a) We are not limited to von Neumann, and Harvard bus architectures; and
    - (b) Register and cache memory sizes should be determined by a system-level design approach, not transistor count, chip area, or maximum clock speed requirements.
- Design symmetry for microprocessor hardware.
- Root of trust, secure boot, secure OS, and secure shutdown, secure auditing, and log files.
- Multiple failures to break security
- Programmer does not have to be a security engineer.

III. OS Friendly Microprocessor Architecture

Here, we will focus on three OSFA innovations [43]-[45]: (1) computer security using multi-level tagged memory, (2) single cycle context switch, and (3) protected I/O channels. The cache bank memory pipeline provides for single cycle context switching. The unique features in the OSFA include.

- Cache Bank Memory Pipeline (Figure 2),
- Multi-level security tagging (Figure 3),
- Multiple Register Files (Figures 2 and 4), and
- High-Speed Context Switching.

A. seL4 Operating System

We are interested in a co-design approach for a microprocessor, OS, and machine formal verification. The seL4 (secure extended L4) operating system is formally proven (machine generated proof-of-correctness), open source operating system. seL4 is a small microkernel with only 8,700 lines of C code and 600 lines of assembly code [40]. To better understand microprocessor-OS co-design, we recommend a study of formally proven OS's to understand where the complicated and privileged code areas are. We believe microprocessor hardware security primitives can significantly reduce large code sections. These provide a two-fold advantage: easier formal verification, and higher performance operating system. The goal is to create a high performance, secure computer system and not follow [16]: "… past hardware has only been optimized for speed - never for security" design philosophy.

With the chicken and egg problem, what came first, the processor or the OS? Historically speaking, both developed over time. Here, we are going to revisit the clean-slate approach and recommend a more open-minded, out-of-the-box approach. We will focus on three innovations in the OS Friendly Microprocessor Architecture and how they can help solve security limits performance mindset.

B. OSFA Cache Bank Architecture

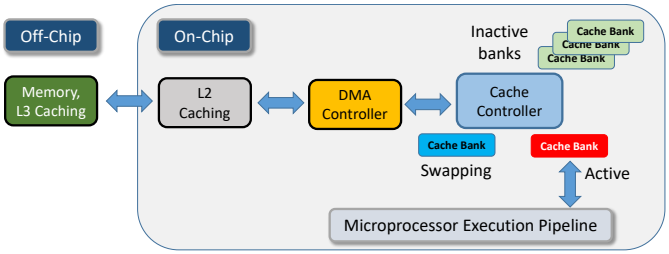

Figure 2. Cache Bank Memory Pipeline

Tag memory for computer security applications has been limited to a bit array of an integer plus tag bits. Fat memory pointers [27] pack a type "safe" pointer plus security tags into the same memory word. Figure 2 illustrates the cache bank memory pipeline for the OSFA. The cache controller manages a set of cache banks. The active cache bank is connected to the execution pipeline. The DMA controller copies the swapping cache bank to and from L2 (level 2) caching. The inactive cache banks are not in use and can be run at a low clock speed and lower voltage to save power. The cache controller provides arbitration to ensure the active cache bank and swapping cache bank are not the same bank.

In Figure 3, a cache bank consists of a cache bank security header tag and memory cell tags. Hardware restricts access to tags. The two level tag structure provides more flexibility than a single tag field for each memory cell. For a 1 k-word cache bank, a 64-byte security header is small. For each 64-bit memory cell, a 64-bit tag field is large. In the OSFA, we propose to use "security locality" (compare to data locality) to reduce the number of tag bits. Each process will have similar security settings, and we propose to use a small 1 k entry to ~16 k entry tag cache to reduce the size of the tag field and provide more flexibility.

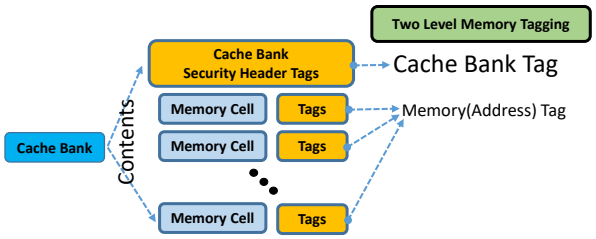

Figure 3. Cache Bank/Memory Cell Tags

C. OSFA Block Diagram

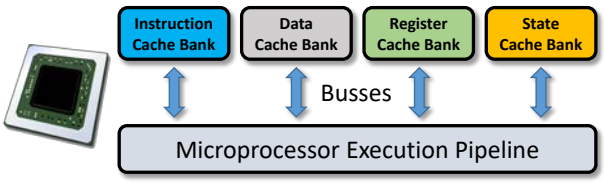

Figure 4. OSFA Architecture

Figure 4 shows the symmetry of the OSFA processor architecture. The OSFA is an extended Harvard architecture with four cache bank memory pipelines from Figure 2. The instruction and data pipelines are similar to a Harvard architecture. The register pipeline allows register files to be swapped in a single cycle; a more advanced version of the Z80 from 1976. The state pipeline allows the execution pipeline latches to be bank switched in a single cycle (see [44] for more details). Hardware provides complete separation of instructions and data. If it were possible to maliciously change the tag bits, the hardware prevents a data stack overflow from "inserting" executable code into the instruction memory pipeline. The combination of hardware and tag fields provides two levels of protection against data stack overflows.

D. OSFA Stack

Stack space memory has the Stack tag set. Only a register with the Stack_Pointer tag set has permission to push and pop objects from a stack. The instruction cache bank memory pipeline and data cache bank memory pipeline both contain stack space areas. The protected stack pointers in [27] can be maliciously changed by editing the binary execution file. In the OSFA, a process requests the trusted microkernel to create a stack area. The stack space has the Stack tag set, and only stack operations are allowed. The base and limit fields from a fat pointer can be used; however, we cannot completely trust the

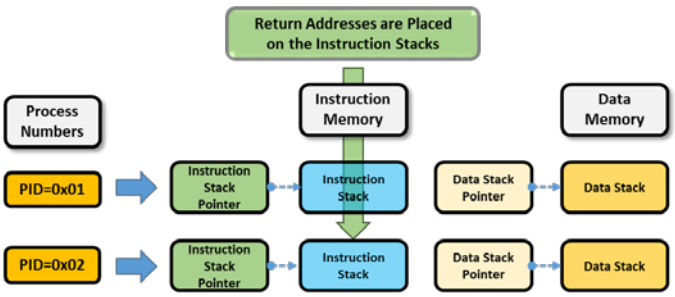

Figure 5. Independent Instruction and Data Stacks

limit field. If the process attempts to access stack space outside the address range set by the trusted microkernel, a memory exception is raised. The address range provides one more level of protection against malicious software. Figure 5 shows an example instruction and data stack frames for two processes. Return addresses are placed in the instruction stack frames and cannot be overwritten by the data stack frames. More details about the operation of security tags will be presented in a cyber security example later on.

E. OSFA Supports 1 Cycle Context Switch

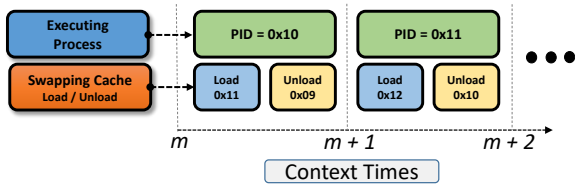

Figure 6. Parallel Context Switch

One limitation with standard operating systems is the high cost of a context switch. The current architecture mindset of von Neumann and Harvard bus architectures limits our thinking to create a single cycle context switch. The cache bank memory pipeline architecture introduced in Figure 2 provides for single cycle context switching. Figure 6 illustrates the parallelism provided by the cache bank memory pipeline from Figure 2. The cache bank controller allows for the DMA controller to copy a cache bank to and from L2 caching while the microprocessor execution pipeline in Figure 4 is executing the active cache bank (running process) from Figure 2. Figure 6 shows process ID 0x10 is executing while process ID 0x11 is loading into a cache bank and process ID 0x09 is copied to L2 caching. A more detailed discussion covering context switching is found in [43]. A trusted boot for the OSFA is described in [46].

F. Computer Security Example

We present a simple OSFA computer security example in Figure 7. The example code creates a pointer to a 16-bit integer array, Array16[ ], with Length elements. The OSFA creates an array pointer where the security tags are set to Read-Write-Modify not allowed, and I/O is set. The memory page referenced by the pointer, Array16, has its tag field set to an integer array. The memory page tags and memory cell tags illustrate two level security tags. The only operations allowed on the array memory page are array element operations. Accessing the memory cells marked Read-Write-Modify not allowed will cause a hardware exception. The Length field was configured by the trusted microkernel and set to Read-Write-Modify are not allowed. The running process cannot access Length's memory address. Any attempt for the executing process to read the address of Array16 results in a hardware generated exception. The tag fields allow the pointer to Array16[ ], and the Length field to be trusted. Only the trusted microkernel can access these fields. The function call IOPort(Array16[ ]); calls the microkernel. The microkernel can trust the Array16[ ] pointer

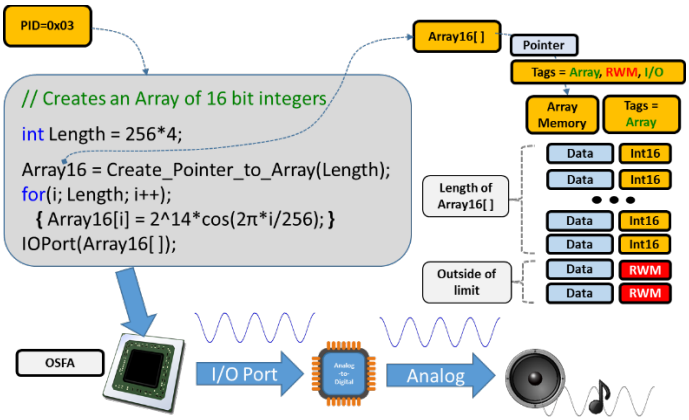

Figure 7. Computer Security Example

and Length field. The microkernel simply calls a DMA I/O processor and the DMA uses the trusted array pointer, Array16[ ], and the Length field to complete the I/O operation. The OSFA tags provide good security will little to no overhead.

G. *Some Architecture Limitations*

Context switching on the OSFA can be compared to stream data processing. If there is not a sufficient number of processes to utilize the parallelism provided by the OSFA architecture, the overhead for a context switch may be larger; however, keep in mind, this is where the processor utilization is low. The two level tagged security may provide too much security and make communicating between shared memory pages more difficult; however, the large number of security settings provide flexibility. For high speed I/O streaming, cache bank size limits the length for an I/O burst.

H. *Cyber Security Impact*

Computer security threats require better hardware solutions. Adding security features and robust design to computer hardware will be a useful and necessary step to mitigate security threats. A typical hacker's cyber exploits, like Heartbleed bug [47], recklessly attack business and cause untold damages to both software and hardware. Many security vendors are researching ways to increase privacy and prevent future Heartbleed-like bugs from spreading and evolving. The current insecure "security model" is collapsing under its own weight. The cat-and-mouse computer virus, and anti-virus signature/patch software approach has failed and cannot be fixed.

Users require greater expectations, individual control of social networking security and privacy, online protection, theft, and abuse of personal information. Users want to have open and safe internet, protected by computer security technologies that are more intuitive, interactive, and most of all invisible. Users want an improved security model with simple to manage and secure credentials. In summary, privacy and security that is easy to use and really works.

So, it's obvious; the answer is, the operating systems resides in the system hardware (computer chips). Silicon computer security would be a substantial cost savings and more secure strategy [47]-[50]. The references discuss security and security issues in wireless computer communication architectures, security policies, and cyber security applications.

In contrast, hardware-based security is the first to boot and operates independently even after the boot process. The OSFA with hardware-based security can shield against potential malware and other threats; including initial boot of a thick operating system like Linux. The dedicated cyber security features within the architecture also operate without burdening the processor, sacrificing security or loss in productivity.

A friendly open-platform, standards-based operating system approach to computer security will allow transparent and collaborative solutions, rapid response to emerging cyber threats, and the potential for broad industry acceptance and long-term success.

IV. Conclusion

We have presented a short review of computer and cyber security history. We have shown that good cybersecurity practices are well understood; however, "… past hardware has only been optimized for speed - never for security." [16]. We have introduced hardware cyber security innovations in the OS Friendly Microprocessor Architecture: cache bank memory pipeline architecture, and multi-level hardware memory tagging. We have illustrated how the cache bank memory pipeline architecture provides for single cycle context switching. We are interested in receiving feedback about the benefits and the limitations of the new architecture (design).

V. Acknowledgment

The authors would like to thank RDECOM community for the opportunity to develop the OSFA. This work was partially supported through funding provided by the U.S. Army Research Laboratory (ARL) to Army High Performance Computing Center at Stanford University and New Mexico State University under contract No. W911NF-07-2-0027.

VI. Disclaimer

Reference herein to any specific commercial, private or public products, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement, recommendation, or favoring by the United States Government. The views and opinions expressed herein are strictly those of the author(s) and do not represent or reflect those of the United States Government.

Distribution statement A: Approved for public release; distribution is unlimited. 11626.

VII. REFERENCES

[1] E. Feustel: "The Rice Research Computer – A tagged architecture*," ACM AFIPS Proceedings of the spring joint computer conference, pp. 369-377, Atlantic City, New Jersey — May 16 - 18, 1972.

[2] Burroughs: *Burroughs B6500 Information Processing Systems Reference Manual*, Burroughs Corp., Detriot MI, 1969.

[3] AEG Telefunken: *TR441: Characteristics of the RD441*, (German), AEG Telefunken Manual, DBS 180 0470 Konstanz, Germany, 1970.

[4] J. Saltzer and M. Schroeder: "The protection of information in computer systems," Proceedings of the IEEE, 63(19):1278-1308, Sept. 1975.

[5] J. Buxton and B. Randell, eds: *Software Engineering Techniques*, April 1970, p. 16. NATO Science Committee Conference, Rome, Italy, 27–31 October 1969.

[6] K. Schaffer, and J. Voas: "What Happened to Formal Methods for Security?" Computer, pp. 70-79, August 2016.

[7] J. Voas: "Insights on Formal Methods in Cybersecurity," Computer, pp. 102-105, May 2015.

[8] Wikipedia.org: "Trojan Horse," (Accessed May 2017). https://en.wikipedia.org/wiki/Trojan_Horse

[9] R. Smith, "A Contemporary Look at Saltzer and Schroeder's 1975 Design Principles", IEEE Security & Privacy, Vol. 10, No. 5, pp. 20-25, Nov.-Dec. 2012.

[10] A. Weaver and N. Newall: "In-Band Single Frequency Signaling," Bell System Technical Journal, 33(6), 1309-1330, (November 1954). https://archive.org/details/bstj33-6-1309 (May 2017)

[11] C. Breen. and C. Dahlbom: "Signaling Systems for Control of Telephone Switching," Bell System Technical Journal, 39(6), 1381-1444 (November 1960). https://archive.org/details/bstj39-6-1381 (May 2017)

[12] Computer History Museum, "Blue Box," (December 2014). http://www.computerhistory.org/collections/catalog/102713487 (May 2017).

[13] FIPS 197: "Advanced Encryption Standard (AES)," November 2001. http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf

[14] C. Cowan, et al.: "Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade*," SANS 2000 (System Administration and Network Security) Conference, 1-11, 2000. http://www.cs.utexas.edu/~shmat/courses/cs380s_fall09/cowan.pdf (May 2017)

[15] D. Wagner, et al.: "A first step toward automated detection of buffer overrun vulnerabilities." In Proceedings of the Network Distributed Systems Security Symposium, pp. 1–15, 2000.

[16] I. Podebrad, et al.: "List of criteria for a secure computer architecture," IEEE Third International Conference on Emerging Security Information, Systems and Technologies, Secureware '09, pp. 76-80, June 2009.

[17] E. Feustel: "On The Advantages of Tagged Architecture," IEEE Transactions on Computers, Vol. C-22, No. 7, JULY 1973.

[18] E. Gehringer, and J. Keedy: "Tagged Architecture: How Compelling Are its Advantages?," Proceedings of the 12th annual international symposium on Computer architecture, Boston, Massachusetts, pp. 162-170, June 17 - 19, 1985.

[19] J. Song: Security Tagging for a Real-Time Zero-Kernel Operating System, Dissertation, University of Idaho, October 2014.

[20] U. Dhawan, et al.: "Hardware Support for Safety Interlocks and Introspection," IEEE Adaptive Host and Network Security Workshop, September 14, 2012.

[21] N. Zeldovich, et al.: "Hardware Enforcement of Application Security Policies Using Tagged Memory," Proceedings of the 8th USENIX conference on Operating systems design and implementation, pp. 225-240, San Diego, California, December, 2008.

[22] A. de Amorim, et al.: "A Verified Information-Flow Architecture (Long version)," crash-safe.org, access 5-10-2017. http://www.crash-safe.org/assets/verified-ifc-long-draft-2013-11-10.pdf

[23] S. Chiricescu, et al.: "SAFE: A Clean-Slate Architecture for Secure Systems," IEEE International Conference on Technologies for Homeland Security (HST), pp. 570-576, 12-14 Nov. 2013.

[24] U. Dhawan, et al.: "Architectural Support for Software-Defined Metadata Processing," crash-safe.org, accessed 5-10-2017 www.crash-safe.org/assets/PUMP-ASPLOS-2015.pdf

[25] J. Alves-Foss, et al.: A new Operating System for Security Tagged Architecture Hardware in Support of Multiple Independent Levels of Security (MILS) Compliant Systems, AFRL Technical Report: AFRL-RI-RS-TR-2014-088, April 2014. www.dtic.mil/dtic/tr/fulltext/u2/a602198.pdf

[26] H. Shrobe, et al.: "Trust-Management, Intrusion Tolerance, Accountability, and Reconstruction Architecture (TIARA)," Massachusetts Institute of Technology, AFRL Final Technical Report AFRL-RI-RS-TR-2009-271, June 2009. www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA511350

[27] A. Kwon, et al.: "Low-Fat Pointers: Compact Encoding and Efficient Gate-Level Implementation of Fat Pointers for Spatial Safety and Capability-based Security," CCS'13, Berlin, Germany, November 4–8, 2013.

[28] G. Venkataramani, et al.: "FlexiTaint: A Programmable Accelerator for Dynamic Taint Propagation," IEEE 14th International Symposium on High Performance Computer Architecture, Salt Lake City, UT, pp. 173-184, 16-20 Feb. 2008.

[29] R. Colwell and E. Jensen: "Performance Effects of Architectural Complexity in the Intel 432," ACM Transactions on Computer Systems, Vol. 6, No. pp.296-339, August 1988.

[30] J. Alves-Foss,et al.: *A New Operating System for Security Tagged Architecture Hardware In Support of Multiple Independent Levels of Security (MILS) Compliant Systems*, University of Idaho, Center Secure and Dependable Systems, Air Force Research Lab Tech Report AFRL-RI-RS-TR-2014-088, APRIL 2014.

[31] Keegan, W. "Separation Kernels Enable Rapid Development of Trustworthy Systems," COTS Journal, pp. 1-3, | February 2014. www.lynx.com/pdf/Separation_Kernels_Enable_Rapid_Development.pdf

[32] Kulkarni, O., et al.: "Virtualization Technology: A Leading Edge," International Journal of Computer Application, Issue 2, Volume 2, pp. 272-287, April 2012. rspublication.com/ijca/april%2012%20pdf/32.pdf

[33] Fannon, R. An analysis of hardware-assisted virtual machine based rootkits, Thesis , Naval Postgraduate School, June 2014. calhoun.nps.edu/handle/10945/42621

[34] Irvine, C., et. al.: "The Trusted Computing Exemplar Project," Proceedings of the 5th IEEE Systems, Man and Cybernetics Information Assurance Workshop, Military Academy, West Point, NY, pp. 109-115 June 2004.

[35] Grace, M., et al.: "Transparent Protection of Commodity OS Kernels Using Hardware Virtualization," Security and Privacy in Communication Networks, pp 162-180, 2010.

[36] Xiong, X., et al.: "Practical Protection of Kernel Integrity for Commodity OS from Untrusted Extensions," NDSS Symposium, 2011. http://www.internetsociety.org/sites/default/files/xipdf.pdf

[37] DARPA, "Clean-slate design of Resilient, Adaptive, Secure Hosts (CRASH)," DARPA-BAA-10-70, June 2010. https://www.fbo.gov/index?s=opportunity&mode=form&id=4022d960a15e87bcaf0fb70101ab53b8&tab=core&_cview=1

[38] S. Chiricescux, et al.: "SAFE: A Clean-Slate Architecture for Secure Systems," IEEE International Conference on Technologies for Homeland Security (HST), Waltham, MA, 12-14 Nov. 2013.

[39] T. Murray, et al.: "seL4: From General Purpose to a Proof of Information Flow Enforcement,"IEEE Symposium on Security and Privacy, Berkeley, CA, USA, pp. 415-429, 19-22 May 2013.

[40] G. Klein, et al.: "seL4: Formal Verification of an OS Kernel," Proceedings of the ACM SIGOPS symposium on Operating systems principles, pp. 207-220, Big Sky, Montana, October, 2009. http://web1.cs.columbia.edu/~junfeng/09fa-e6998/papers/sel4.pdf

[41] J. Crisp: *Introduction to Microprocessors and Microcontrollers*, Newnes, Boston, MA, 1998.

[42] S. McPeak, et al.: "CCured: Type-Safe Retrofitting of Legacy Software," ACM Transactions on Programming Languages and Systems, Vol. 27, No. 3, pp. 477–526, May 2005.

[43] P. Jungwirth and P. La Fratta: *OS Friendly Microprocessor Architecture*, Army Research Lab Report ARL-SR-0370, April 2017. http://www.arl.army.mil/arlreports/2017/ARL-SR-0370.pdf

[44] P. Jungwirth abd P. La Fratta: "OS Friendly Microprocessor Architecture," US Patent 9122610, September 2015.

[45] P. Jungwirth and P. La Fratta: "OS friendly microprocessor architecture: Hardware level computer security", *Proc. SPIE* 9826, Cyber Sensing 2016, 982602, 12 May 2016.

[46] P. Jungwirth, et al.: "Rootkits and the OS friendly microprocessor architecture", Proc. SPIE 10185, Cyber Sensing, April 2017.

[47] Z. Durumeric, et al.: "The matter of heartbleed." Proceedings of the 2014 Conference on Internet Measurement Conference. ACM, 2014.

[48] P. Chan. and Y. Qu: "Assessing Vulnerabilities in Bluetooth Low Energy (BLE) Wireless Network based IoT Systems," IEEE Second International Conference on Big Data Security Cloud, High Performance and Smart Computing, Intelligent Data and Security, pp. 42-48, April, 2016.

[49] P. Chan, et al.: "Modeling mobile network connectivity in the presence of jamming," IEEE Military Communictions Conference, pp. 1-4, October, 2012.

[50] P. Chan, et al.: "System Engineering Approach in Tactical Wireless RF Network Analysis," INTECH Open Access Publisher, 2012.