AOS-RISC-V: Towards Always-On Heap Memory Safety

Yonghae Kim
yonhae@gatech.edu
Georgia Institute of Technology
USA

Anurag Kar
anurag.kar@gatech.edu
Georgia Institute of Technology
USA

Siddant Singh
ssingh484@gatech.edu
Georgia Institute of Technology
USA

Ammar A. Ratnani
aratnani7@gatech.edu
Georgia Institute of Technology
USA

Jaekyu Lee
jaekyu.lee@arm.com
Arm Research
USA

Hyesoon Kim
hyesoon@cc.gatech.edu
Georgia Institute of Technology
USA

ABSTRACT

Despite its notoriety for a long time, achieving robust yet practical memory safety remains challenging. In a plethora of memory-safety proposals, we observe that many recent proposals adopted the idea of pointer tagging, which places a pointer tag in the unused upper bits of a pointer and utilizes the pointer tag for a security purpose. With their promising results, such methods give us hope for the end of the eternal war in memory safety.

In this paper, we revisit one of the first pointer-tagging methods and examine its feasibility as a runtime solution. To this end, we prototype AOS-RISC-V, a RISC-V-based full-system framework that ensures heap memory safety. For realistic evaluation, we base our design on the open-source RISC-V BOOM core, one of the most sophisticated RISC-V out-of-order processors. We then implement lightweight hardware extensions for new ISA support and bounds-checking mechanisms. To enable AOS-RISC-V in a real system, we further design new compiler passes in the LLVM compiler framework and add operating system support in the Linux kernel, both of which are compatible with RISC-V. All together, we demonstrate our prototype running under Linux on FPGAs and conduct a full-system-level evaluation. Our evaluation results report a 20% average slowdown for the selected SPEC 2006 workloads.

1 INTRODUCTION

Despite the long-term notoriety, memory safety vulnerabilities are still problematic, accounting for almost 70% of vulnerabilities found in wild [6, 13]. Memory safety problems occur when a spatially or temporally illegal memory access is performed. Buffer overflows and out-of-bounds accesses are well-known examples of spatial memory safety problems. Temporal safety problems include use-after-free (UAF) and uninitialized use.

To protect against such vulnerabilities, numerous defense mechanisms have been proposed, including both software- and hardware-based approaches. However, their runtime deployment in commodity systems is still questionable because of performance, security, or compatibility issues. Even with their proficiency, software solutions [1, 4, 5, 14, 15, 17] typically incur significant performance overhead, and such overhead has limited their applicability only to debugging and testing purposes. On the other hand, hardware-based mechanisms [8, 10, 12, 16, 20–24] achieve moderate runtime overhead but tend to offer partial security guarantees or lose compatibility with legacy code.

Meanwhile, we observe that pointer-tagging methods [8, 10, 12, 16, 24] attract a great attention from research communities. Given

```
1 int main(void) {
2   char *buf = (char *) malloc(10);
3   scanf("%s", buf);
4   printf("buf: %s\n", buf);
5   ... }
```

Figure 1: A simple buffer overflow example.

that the effective virtual memory address size is less than 64 bits under typical virtual address schemes, pointer-tagging methods utilize the unused high-order bits of a pointer to store a pointer tag and use the pointer tag to look up security metadata associated with the pointer. Among the prior work, AOS [10] proposes an efficient bounds-checking mechanism that implements lightweight hardware extensions for heap memory safety. Using Arm pointer authentication (PA) primitives, AOS generates a pointer authentication code (PAC) and embeds it into a pointer address, i.e., *signs* a pointer. AOS then uses the PAC as a pointer tag to look up bounds information upon memory accesses by the signed pointer.

While AOS exhibits its potential of low overhead (an 8.4% average slowdown), we see that its evaluation is conducted based on system emulation (SE) mode in the gem5 simulator [3] and does not execute whole programs for the sake of reasonable simulation time. Given that memory allocation and memory access behavior can vary over time during program execution, a full-system level evaluation with the entire program execution might produce more realistic results.

The nature of open-source RISC-V instruction set architecture (ISA) allows researchers in both academia and industry to easily prototype new architecture or system designs [19]. In addition, the advancement of the RISC-V technology enables full-system evaluation in a real system, e.g., executing benchmarks under the Linux kernel running on FPGAs, bridging the gap between evaluation results from simulation and real products.

Motivated by the maturity of the RISC-V ecosystem, we investigate the feasibility of AOS as a runtime solution through full-system evaluation. To this end, we present AOS-RISC-V, a full-system level framework for memory safety. Based on the RISC-V BOOM core, one of the most sophisticated open-source out-of-order processors, we implement lightweight hardware extensions for new ISA support and bounds-checking mechanisms proposed in AOS. To enable AOS-RISC-V in a real system, we design new compiler passes in the LLVM compiler framework and add necessary operating system (OS) support in the Linux kernel, both of which are compatible with RISC-V. Finally, we prototype AOS-RISC-V running under the

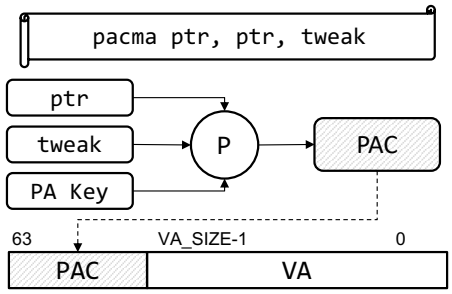

Figure 2: `pacma` instruction using QARMA to generate a PAC.

```
ptr = malloc(size);
pacma ptr, ptr, sp;
bndstr ptr, ptr, size;
```

(a) AOS-malloc.

```
bndclr ptr, ptr;
xpacm ptr, ptr;
free(ptr);
```

(b) AOS-free.

Figure 3: Data-pointer signing proposed in AOS [10].

Linux kernel on FPGAs and conduct performance evaluation. Our results show a 20% average slowdown across selected SPEC 2006 workloads. To contribute to research communities, we open-source AOS-RISC-V.[1]

2 BACKGROUND

2.1 Memory Safety

In computer systems, memory safety vulnerabilities have been persistent over a long period of time. Memory safety problems are generally classified into two types. Spatial memory safety errors occur when a memory instruction accesses outside of its designated boundary, e.g., buffer overflows and out-of-bounds accesses. Temporal safety errors occur when a memory access is performed to a memory region that has been freed, e.g., use-after-free (UAF).

Figure 1 shows a simple heap buffer overflow example where a user provides an input string via `scanf()`. In this example, an attacker can simply inject a long input sequence whose size exceeds the size of the target buffer (`buf`) to invoke a buffer overflow. Since the unsafe `scanf()` has no knowledge of the array size, it will just store the given input to the target memory address and end up overwriting the adjacent memory space.

2.2 Open-source RISC-V CPU Cores

Open-source RISC-V CPU cores [2, 18, 25] have gained traction lately in academia and industry. Since the basic RISC-V ISA is barebones and provides only essential functionality, researchers can add their own ISA extensions and integrate bespoke hardware units to the design, making it highly modular and extensible.

RISC-V also inherits the power efficiency and performance of the RISC architecture, making it an excellent choice for low-power computing with good performance. Moreover, the open-source nature of RISC-V helps make it accessible to a larger audience. For this reason, we choose the RISC-V architecture as our evaluation platform, provide hardware security features to the RISC-V ecosystem, and make it available to a wider audience.

3 AOS-RISC-V

While various proposals have been proposed to achieve memory safety, we choose the prior work, AOS [10], as our target mechanism to implement because of its lightweight hardware extensions as well as efficient metadata management methods. As a hardware-based bounds-checking mechanism, as can be seen in Figure 2, AOS utilizes Arm pointer authentication (PA) primitives to generate a pointer authentication code (PAC). AOS then places the PAC into a pointer address, i.e., *signs* a pointer, and uses the PAC as a pointer tag dedicated to the corresponding pointer. Since AOS recognizes the heap memory vulnerabilities as the most critical attack vector, it signs and protects data pointers returned by dynamic memory allocations, e.g., `malloc()` and `new`, as shown in Figure 3. To handle possible PAC collisions due to the limited PAC size available in a pointer address, AOS maintains a hashed bounds table (HBT) in memory, which accommodates multiple bounds for each PAC and performs an iterative bounds search when necessary.

3.1 ISA Extensions

To enable pointer-signing and bounds management operations, new instructions are introduced. Since AOS is originally based on the AArch64 architecture, we newly find unused instruction encoding reserved for custom instructions in the RISC-V base opcode map [19] and use it to define the following new instructions.

- `pacma rd, rs1, rs2`: computes a PAC using QARMA that takes a pointer address (`rs1`) as a plaintext, a tweak (`rs2`), and PA key.[2] Then, it returns a signed pointer address (`rd`) where the computed PAC is embedded into its high-order bits.
- `xpacm rd, rs1`: strips a signed pointer address (`rs1`) by masking its high-order bits and returns the resulting address (`rd`).
- `bndstr rd, rs1, rs2`: encodes 8-byte bounds information using a base address (`rs1`) and a size (`rs2`), stores the computed bounds in the HBT, and returns the pointer address (`rd`).
- `bndclr rd, rs1`: clears the bounds information associated with a pointer address (`rs1`) by storing an 8-byte zero value in the HBT and returns the pointer address (`rd`).

While the `pacma` instruction is originally proposed to take three source operands in AOS, including additional size information for address hashing code (AHC) generation, only floating-point instructions are supposed to take a third source operand in the RISC-V architecture. As such, we drop the third operand of the `pacma` as well as the use of AHCs in our design and check the *signess* of a pointer by looking for a nonzero PAC in a pointer address.

[1] https://github.com/yonghaekim/AOS-RISC-V

[2] Arm PA provides PA keys stored in hardware registers and invisible to a user process.

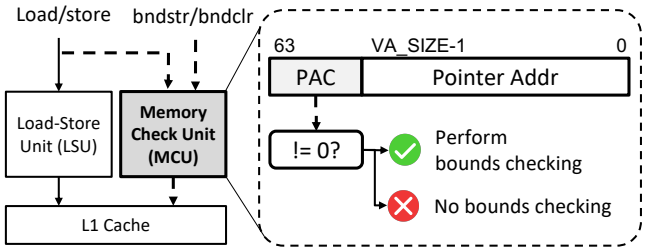

Figure 4: Memory check unit (MCU) structure.

3.2 Memory Check Unit (MCU)

To process new instructions, AOS adds a memory check unit (MCU) responsible for bounds checking and metadata management. Being located next to a load-store unit (LSU), the MCU takes all memory instructions, i.e., loads and stores, as well as bounds instructions, i.e., `bndstr` and `bndclr`. Depending on the instruction type, the MCU generates memory requests to load or store bounds from the HBT in memory. Since the HBT is indexed by PACs (embedded in memory addresses), the locations of bounds are calculated using the base address of the HBT and PACs, i.e., `HBT[PAC]`.

To adapt the MCU design to a real processor design, we decide to break it into two separate queues, namely memory check queue (MCQ) and bounds queue (BDQ). This design choice is based on the following two observations: 1) only bounds instructions require the bounds metadata field and 2) the number of bounds instructions is much less than the number of memory instructions, so the MCU mostly gets full with memory instructions, causing backpressure to the issue stage. For better resource utilization and performance, we choose to size the MCQ sufficiently large such that it can hold as many inflight memory instructions as possible and size the BDQ reasonably small.

3.3 Compiler Support

Since the LLVM 9.0.0 release, the RISC-V target became no-long experimental, and the backend started to support full codegen for the RV32I and RV64I base RISC-V instruction set variants. As such, we design new compiler passes to the optimizer and the RISC-V backend in the LLVM 9.0.1 [11]. First, the `aos-riscv-opt` optimizer pass is designed to detect dynamic memory allocation and deallocation calls and insert new intrinsic functions at the LLVM intermediate representation (IR) level. The inserted intrinsic functions are detected at the `aos-riscv` backend pass and are replaced with new instructions, as shown in Table 1. Additionally, the `aos-riscv-opt` pass inserts a custom system call to a program entry, which enables the AOS mode and configures hardware registers. We introduce more details of the custom system call in Section 3.4.

3.4 OS Support

Along with the hardware modification and the compiler support, we provide kernel support for the hardware configuration.

Configuration. In the user-mode (U-mode) of the standard RISC-V ISA [19], we define the new control and status registers (CSRs) (see Table 2). Specifically, the `enableAOS` CSR is used to enable the hardware-based memory checks by AOS-RISC-V. The base address of an HBT is stored in the `baseAddrOfHBT` CSR, and the number of ways of an HBT is configured via the `numWaysOfHBT` CSR. To interface with such CSRs, we provide a custom system call, namely `__aos_set()`, that is inserted into the entry of a user program at compilation and sets the CSRs with given argument values. Besides to those CSRs for configuration, we also add extra CSRs to count the number of failures of bounds operations; `numBndstrFails`, `numBndclrFails`, and `numBndchkFails`. After a program is terminated, the kernel reads and prints those CSRs to let a user know whether any bounds operation has failed during program execution.

Process management. Besides the hardware configurations through our system call, the kernel needs to keep track of the information of each user process. To do so, we add new fields to the process structure in the linux kernel, i.e., `task_struct`. Those fields are initialized upon process creation and are properly set by our custom system call. During a context switch, if the current process is enabled with AOS-RISC-V, the kernel saves its configuration information in the process structure, including the base address and the number of ways of an HBT assigned to the process. Then, the kernel checks if the next process to execute is also enabled with AOS-RISC-V. If so, the kernel overwrites the CSRs with the configuration information of the next process before it begins its execution. Otherwise, the kernel initializes the CSRs with zero to disable AOS features for the next process.

Table 1: Code examples in C, LLVM IR, and assembly code.

	Code Examples
C code	char *ptr = (char *) malloc(10);
LLVM IR code (frontend)	%3 = call noalias i8* @malloc(i64 10) #3 %4 = call i8* @llvm.aos.pacma.p0i8(i8* %3, i64 0) %5 = call i8* @llvm.aos.bndstr.p0i8(i8* %4, i64 10)
Assembly Code (backend)	call malloc@plt pacma a0, a0, a1 bndstr a0, a0, a1

Table 2: New control and status registers in AOS-RISC-V.

CSR Name	Permission	Description
enableAOS	R/W	Switch to enable AOS-RISC-V
baseAddrOfHBT	R/W	Base address of an HBT
numWaysOfHBT	R/W	Number of ways of an HBT
numBndstrFails	R/W	Number of bounds-store failures
numBndclrFails	R/W	Number of bounds-clear failures
numBndchkFails	R/W	Number of bounds-check failures

4 METHODOLOGY

We prototype AOS-RISC-V on top of the RISC-V BOOM core [25], which is one of the most sophisticated open-source processors. We then evaluate our design on Amazon EC2 F1 using FireSim [9], an open-source FPGA-accelerated hardware platform for full-system simulation. To create instrumented binaries running on AOS-RISC-V, we design custom passes in LLVM-9.0.1 [11]. For our kernel

Table 3: BOOM core configurations for evaluation.

Clock	75 MHz	**L1-I cache**	32KB, 8-way
LLC	4MB	**L1-D cache**	64KB, 16-way
DRAM	16 GB DDR3	**L2 cache**	512KB, 8-way
Front-end	8-wide fetch 16 RAS & 512 BTB entries gshare branch predictor		
Execution	3-wide decode/dispatch 96 ROB entries 100 int & 96 floating point registers		
Load-store unit	24 load & 24 store queue entries		
Memory check unit	36 memory check & 8 bounds queue entries		

support, we modify the Linux distribution (`Linux 5.7-rc3`) provided by FireSim. For performance evaluation, we instrument and compile SPEC 2006 workloads [7] and run instrumented binaries using test inputs.

5 EVALUATION

Figure 5 illustrates the normalized execution time of AOS-RISC-V across the SPEC 2006 workloads. Most benchmarks show moderate runtime overhead (20% on average) in our evaluation. Our analysis reveals that the performance overhead is mainly derived from 1) increased cache port contentions due to additional memory accesses for bounds checking and 2) the cache pollution due to the extra bounds metadata. As the number of memory accesses requiring bounds checking increases, the increased cache port contentions can delay regular memory accesses, slowing down normal program execution. In addition, as the memory footprint of bounds metadata increases, useful cache lines that could have been accessed by regular memory accesses in the near future can be evicted from caches, leading to increased memory latency for subsequent accesses.

Notably, we observe that `sjeng` has near-zero runtime overhead. As shown in Figure 6, the ratio of signed loads and stores over the total memory accesses is only 1%. This result indicates that only 1% of the entire memory accesses require bounds checking, and 99% of memory accesses do not cause extra overhead. In contrast, `bzip2` and `hmmer` exhibit the high ratios of signed memory accesses close to 95% and 56%, respectively. Table 4 shows the number of signing and bounds instructions executed. Note that we insert a `pacma` and a `bndstr` after each`malloc()` and a `bndclr` and `xpacm` before each `free()`, as shown in Figure 3. While most applications execute a marginal number of additional instructions, `hmmer` is shown to be the most malloc-intensive application among the evaluated applications.

6 DISCUSSION

In our current design, we observe a higher runtime overhead than that of AOS. This discrepancy seems to be caused by the limited data fetch width supported in the BOOM core. AOS assumes the data fetch width of 64 bytes supported in modern processors, and therefore up to eight sets of bounds metadata can be brought into the CPU pipeline with a single memory request. However, our baseline BOOM core supports at most 8-byte data fetch width, so more bounds access requests need to be generated during iterative bounds search in the HBT.

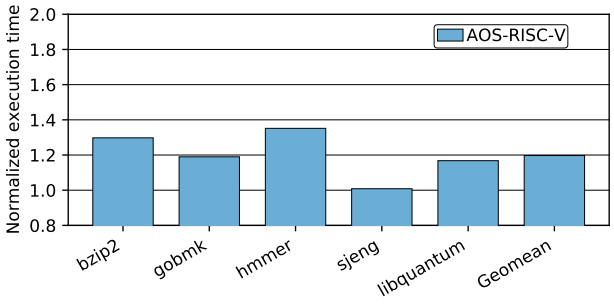

Figure 5: Execution time of AOS-RISC-V across SPEC 2006 workloads, normalized to the baseline.

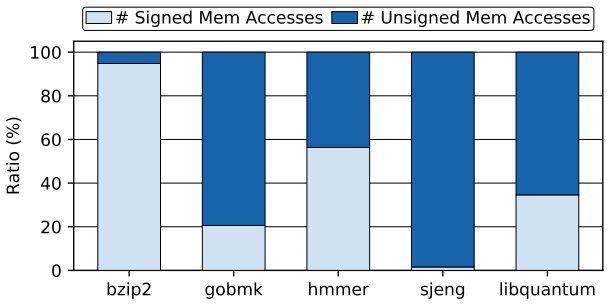

Figure 6: The ratio of signed loads and stores requiring bounds checking over the total memory accesses.

7 FUTURE WORK

AOS-RISC-V is currently under active development, and we leave several tasks as future work.

Dynamic bounds-table resizing. In AOS, the set-associative HBT structure is introduced to handle possible PAC collisions and to accommodate multiple bounds metadata for each PAC. Nevertheless, the HBT can still overflow when a certain application creates numerous bounds metadata at runtime. AOS addresses this concern by adopting the dynamic bounds-table resizing method. In our current design, we only allocate a fixed-size HBT and leave the implementation of dynamic bounds-table resizing as future work.

Exception handling for bounds-operation failure. To support precise debugging or promptly prevent malicious attacks at runtime, a new class of exception would need to be defined to alert the user of a memory safety violation case. Currently, we only count the number of bounds-operation failures and report the number after a user process is terminated.

Enhancing security guarantees. As mentioned, AOS considers the heap exploitation as the most prevalent and problematic attack vector and focus on heap memory safety. To achieve complete memory safety, more sophisticated compiler techniques could be

Table 4: Number of additional signing and bounds instructions executed.

Name	pacma	xpacm	bndstr	bdclr
bzip2	28	24	24	24
gobmk	4181	4172	4181	4172
hmmer	90138	90138	90138	90138
sjeng	4	0	4	0
libquantum	95	95	95	95

developed to extend the security coverage to other memory types, such as stack and global memory.

8 CONCLUSIONS

In this paper, we presented AOS-RISC-V, a full-stack memory safety framework. Based on the open-source RISC-V BOOM core, we prototyped AOS-RISC-V, a full-system level framework for heap memory safety, with our modifications encompassing architecture, compiler, and OS support. Under the Linux kernel running on Amazon EC2 F1 instances, we conducted performance evaluation and showed that AOS-RISC-V incurred a 20% average slowdown across the selected SPEC 2006 workloads.

REFERENCES

[1] Periklis Akritidis, Manuel Costa, Miguel Castro, and Steven Hand. 2009. Baggy Bounds Checking: An Efficient and Backwards-Compatible Defense against out-of-Bounds Errors. In *Proceedings of the 18th USENIX Security Symposium (Security)* (Montreal, Canada). USENIX Association, USA, 51–66.

[2] Krste Asanović, Rimas Avizienis, Jonathan Bachrach, Scott Beamer, David Biancolin, Christopher Celio, Henry Cook, Daniel Dabbelt, John Hauser, Adam Izraelevitz, Sagar Karandikar, Ben Keller, Donggyu Kim, John Koenig, Yunsup Lee, Eric Love, Martin Maas, Albert Magyar, Howard Mao, Miquel Moreto, Albert Ou, David A. Patterson, Brian Richards, Colin Schmidt, Stephen Twigg, Huy Vo, and Andrew Waterman. 2016. *The Rocket Chip Generator.* Technical Report UCB/EECS-2016-17. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html

[3] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The Gem5 Simulator. *SIGARCH Computer Architecture News* 39, 2 (Aug. 2011), 1–7.

[4] Baozeng Ding, Yeping He, Yanjun Wu, Alex Miller, and John Criswell. 2012. Baggy Bounds with Accurate Checking. In *2012 IEEE 23rd International Symposium on Software Reliability Engineering Workshops.* IEEE, 195–200. https://doi.org/10.1109/ISSREW.2012.24

[5] Gregory J. Duck and Roland H. C. Yap. 2018. EffectiveSan: Type and Memory Error Detection Using Dynamically Typed C/C++. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)* (Philadelphia, PA, USA). Association for Computing Machinery, New York, NY, USA, 181–195.

[6] Google. 2017. Google Queue Hardening. https://security.googleblog.com/2019/05/queue-hardening-enhancements.html.

[7] John L. Henning. 2006. SPEC CPU2006 Benchmark Descriptions. *SIGARCH Computer Architecture News* 34, 4 (Sept. 2006), 1–17.

[8] Mohamed Tarek Ibn Ziad, Miguel A. Arroyo, Evgeny Manzhosov, Ryan Piersma, and Simha Sethumadhavan. 2021. No-FAT: Architectural Support for Low Overhead Memory Safety Checks. In *Proceedings of the 48th Annual International Symposium on Computer Architecture (ISCA)* (Virtual Event, Spain). IEEE Press, Piscataway, NJ, USA, 916–929. https://doi.org/10.1109/ISCA52012.2021.00076

[9] Sagar Karandikar, Howard Mao, Donggyu Kim, David Biancolin, Alon Amid, Dayeol Lee, Nathan Pemberton, Emmanuel Amaro, Colin Schmidt, Aditya Chopra, Qijing Huang, Kyle Kovacs, Borivoje Nikolic, Randy Katz, Jonathan Bachrach, and Krste Asanović. 2018. FireSim: FPGA-accelerated Cycle-exact Scale-out System Simulation in the Public Cloud. In *Proceedings of the 45th Annual International Symposium on Computer Architecture (ISCA)* (Los Angeles, California). IEEE Press, Piscataway, NJ, USA, 29–42. https://doi.org/10.1109/ISCA.2018.00014

[10] Yonghae Kim, Jaekyu Lee, and Hyesoon Kim. 2020. Hardware-based Always-on Heap Memory Safety. In *Proceedings of the 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO).* IEEE Computer Society, Los Alamitos, CA, 1153–1166.

[11] Chris Lattner and Vikram Adve. 2004. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proceedings of the International Symposium on Code Generation and Optimization (CGO): Feedback-Directed and Runtime Optimization* (Palo Alto, California). IEEE Computer Society, USA, 75–86.

[12] Michael LeMay, Joydeep Rakshit, Sergej Deutsch, David M. Durham, Santosh Ghosh, Anant Nori, Jayesh Gaur, Andrew Weiler, Salmin Sultana, Karanvir Grewal, and Sreenivas Subramoney. 2021. Cryptographic Capability Computing. In *Proceedings of the 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)* (Virtual Event, Greece). Association for Computing Machinery, New York, NY, USA, 253–267. https://doi.org/10.1145/3466752.3480076

[13] Matt Miller. 2019. Trends, challenges, and strategic shifts in the software vulnerability mitigation landscape. https://github.com/microsoft/MSRC-Security-Research/blob/master/presentations/2019_02_BlueHatIL/2019_01%20-%20BlueHatIL%20-%20Trends%2C%20challenge%2C%20and%20shifts%20in%20software%20vulnerability%20mitigation.pdf.

[14] Santosh Nagarakatte, Jianzhou Zhao, Milo M.K. Martin, and Steve Zdancewic. 2009. SoftBound: Highly Compatible and Complete Spatial Memory Safety for c. In *Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)* (Dublin, Ireland). Association for Computing Machinery, New York, NY, USA, 245–258. https://doi.org/10.1145/1542476.1542504

[15] J. Newsome and D. Song. 2005. Dynamic Taint Analysis for Automatic Detection, Analysis, and SignatureGeneration of Exploits on Commodity Software. In *NDSS.* The Internet Society, USA.

[16] Hiroshi Sasaki, Miguel A. Arroyo, M. Tarek Ibn Ziad, Koustubha Bhat, Kanad Sinha, and Simha Sethumadhavan. 2019. Practical Byte-Granular Memory Blacklisting Using Califorms. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)* (Columbus, OH, USA). Association for Computing Machinery, New York, NY, USA, 558–571. https://doi.org/10.1145/3352460.3358299

[17] Konstantin Serebryany, Derek Bruening, Alexander Potapenko, and Dmitriy Vyukov. 2012. AddressSanitizer: A Fast Address Sanity Checker. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference (ATC).* USENIX, 309–318.

[18] Blaise Tine, Krishna Praveen Yalamarthy, Fares Elsabbagh, and Kim Hyesoon. 2021. Vortex: Extending the RISC-V ISA for GPGPU and 3D-Graphics. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture* (Virtual Event, Greece) *(MICRO '21).* Association for Computing Machinery, New York, NY, USA, 754–766. https://doi.org/10.1145/3466752.3480128

[19] Andrew Waterman and Krste Asanović. 2019. The RISC-V Instruction Set Manual. https://riscv.org/wp-content/uploads/2019/12/riscv-spec-20191213.pdf.

[20] Nathaniel Wesley Filardo, Brett F. Gutstein, Jonathan Woodruff, Sam Ainsworth, Lucian Paul-Trifu, Brooks Davis, Hongyan Xia, Edward Tomasz Napierala, Alexander Richardson, John Baldwin, David Chisnall, Jessica Clarke, Khilan Gudka, Alexandre Joannou, A. Theodore Markettos, Alfredo Mazzinghi, Robert M. Norton, Michael Roe, Peter Sewell, Stacey Son, Timothy M. Jones, Simon W. Moore, Peter G. Neumann, and Robert N. M. Watson. 2020. Cornucopia: Temporal Safety for CHERI Heaps. In *2020 IEEE Symposium on Security and Privacy (SP).* IEEE, Piscataway, NJ, USA, 608–625. https://doi.org/10.1109/SP40000.2020.00098

[21] J. Woodruff, A. Joannou, H. Xia, A. Fox, R. M. Norton, D. Chisnall, B. Davis, K. Gudka, N. W. Filardo, A. T. Markettos, M. Roe, P. G. Neumann, R. N. M. Watson, and S. W. Moore. 2019. CHERI Concentrate: Practical Compressed Capabilities. *IEEE Transactions on Computers (TC)* 68, 10 (2019), 1455–1469.

[22] Jonathan Woodruff, Robert N.M. Watson, David Chisnall, Simon W. Moore, Jonathan Anderson, Brooks Davis, Ben Laurie, Peter G. Neumann, Robert Norton, and Michael Roe. 2014. The CHERI Capability Model: Revisiting RISC in an Age of Risk. In *Proceedings of the 41st Annual International Symposium on Computer Architecture (ISCA)* (Minneapolis, Minnesota, USA). IEEE Press, Piscataway, NJ, USA, 457–468.

[23] Hongyan Xia, Jonathan Woodruff, Sam Ainsworth, Nathaniel W. Filardo, Michael Roe, Alexander Richardson, Peter Rugg, Peter G. Neumann, Simon W. Moore, Robert N. M. Watson, and Timothy M. Jones. 2019. CHERIvoke: Characterising Pointer Revocation Using CHERI Capabilities for Temporal Memory Safety. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)* (Columbus, OH, USA). Association for Computing Machinery, New York, NY, USA, 545–557.

[24] Shengjie Xu, Wei Huang, and D. Lie. 2021. In-fat pointer: hardware-assisted tagged-pointer spatial memory safety defense with subobject granularity protection. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS).* Association for Computing Machinery, New York, NY, USA, 224–240.

[25] Jerry Zhao, Ben Korpan, Abraham Gonzalez, and Krste Asanovic. 2020. SonicBOOM: The 3rd Generation Berkeley Out-of-Order Machine. (May 2020).