Why is memory safety still a concern?

PhD Candidacy Exam

Mohamed (Tarek Ibn Ziad) Hassan
mtarek@cs.columbia.edu

Department of Computer Science
Columbia University

April 9th, 2020
10:30 A.M.
Location: TBD

1 Candidate Research Area Statement

Languages like C and C++ are the gold standard for implementing a wide range of software systems such as safety critical firmware, operating system kernels, and network protocol stacks for performance and flexibility reasons. As those languages do not guarantee the validity of memory accesses (i.e., enforce memory safety), seemingly benign program bugs can lead to silent memory corruption, difficult-to-diagnose crashes, and most importantly; security exploitation. Attackers can compromise the security of the whole computing ecosystem by exploiting a memory safety error with a suitably crafted input. Since the spread of Morris worm in 1988 and despite massive advances in memory safety error mitigations, memory safety errors have risen to be the most exploited vulnerabilities.

As part of my research studies in Columbia, I have worked on designing and implementing hardware primitives [1, 2] that provide fine-grained memory safety protection with low performance overheads compared to existing work. My ongoing work in this research include extending the above primitives to system level protection in addition to addressing their current limitations.

2 Faculty Committee Members

The candidate respectfully solicits the guidance and expertise of the following faculty members and welcomes suggestions for other important papers and publications in the exam research area.

- Prof. Simha Sethumadhavan
- Prof. Steven M. Bellovin
- Prof. Suman Jana

3 Exam Syllabus

The papers have broad coverage in the space of memory safety vulnerabilities and mitigations, which are needed to (1) make a fair assessment of current defensive and offensive techniques and (2) explore new threats and defensive opportunities.

- I begin with a brief overview of memory safety [3–5] showing why it is still a concern.
- To motivate the need for complete memory safety solutions, I go over a timeline of prior exploitation techniques and mitigations, as shown in Figure 1. I start with the earliest documented memory corruption attack from 1972. Then, I describe various memory attacks ranging from code injection (e.g., Morris Worm [6]) and code reuse [7–9] to data-oriented attacks [10]. I also cover immediate mitigations, such as ASLR [11], CFI [12], DFI [13], Data Randomization [14, 15], and the recent ARM pointer authentication (PAC) [16].
- Next, I provide an overview of defensive techniques that aim at enforcing spatial and temporal memory safety. First, I group prior techniques, which offer spatial memory safety guarantees, into three main categories (Whitelisting, Blacklisting, and Randomized Allocators). Each group is further divided by the way it manages its own metadata, as shown in Table 1. Second, I explain the different approaches, which are used to guarantee temporal memory safety in Table 2. My goal is to show the strengths and limitations of each technique.
- Finally, I conclude by highlighting opportunities for future work, given the restrictions of current solutions.

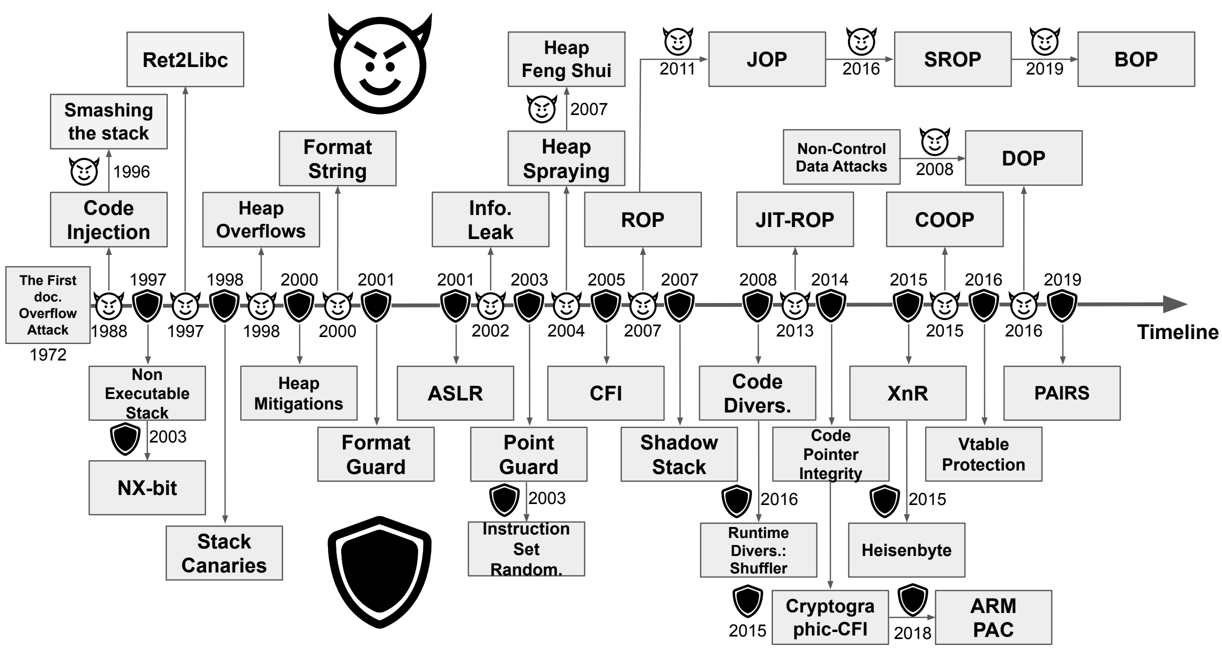

Figure 1: Timeline for memory safety exploitation techniques (marked with demons) and mitigations (marked with shields).

Table 1: Spatial memory protection techniques. Tools with no separate citations are covered in the Systematization of Knowledge papers [3–5] and/or Background section of the PhD dissertations [17, 18].

	Whitelisting		**Blacklisting**	**Randomized Allocators**
	Per-Object	Per-Pointer		
Disjoint Metadata	Compatible C [19] Baggy Bounds [22]	Mondrian [18](Ch2) M-machine [18](Ch2) Softbound [17](Ch4) Hardbound [23] Watchdog [17](Ch5) CUP [25] Intel MPX [26]	Purify Valgrind Dr. Memory Electric Fence ASan [24]	Diehard [20] FreeGuard Guarder [21]
Inlined Metadata	EffectiveSan [27]	CHERI [18, 28] Cyclone [30] CheckedC [32]	SafeMem [29] REST [31] Califorms [1]	
Co-joined Metadata	ARM Memory Tagging [33] SPARC ADI [34]			
No Metadata	Lowfat S/W [35]	Lowfat H/W [36]		

Table 2: Temporal memory protection techniques. Tools with no separate citations are covered in the Systematization of Knowledge papers [3–5] and/or Background section of the PhD dissertations [17, 18].

Solution Category			**Example**
Garbage Collection (GC)	Regular		Hardware Accelerated GC [37]
	Conservative		MarkUs [38]
Memory Quarantining			Valgrind, ASan [24], REST [31], CHERIvoke, and Califorms [1]
Lock & Key	Explicit	Change Lock	CETS [17](Ch4), CUP [25]
	Implicit	Change Lock	Electric Fence, Oscar [39]
		Revoke key	DangNull, DangSan, and BOGO [40]

References

[1] Hiroshi Sasaki, Miguel A. Arroyo, **Mohamed Tarek Ibn Ziad**, Koustubha Bhat, Kanad Sinha, and Simha Sethumadhavan. Practical byte-granular memory blacklisting using Califorms. In *Proceedings of the 52Nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '52, pages 558–571, Columbus, OH, USA, 2019. URL http://doi.acm.org/10.1145/3352460.3358299.

[2] **Mohamed Tarek Ibn Ziad**, Miguel A. Arroyo, Evgeny Manzhosov, Vasileios P. Kemerlis, and Simha Sethumadhavan. PAIRS: Control flow protection using phantom addressed instructions. *arXiv.org*, 2019. URL https://arxiv.org/abs/1911.02038v1.

[3] Victor van der Veen, Nitish dutt-Sharma, Lorenzo Cavallaro, and Herbert Bos. Memory errors: The past, the present, and the future. In *Proceedings of the 15th International Symposium on Research in Attacks, Intrusions, and Defenses RAID*, pages 86–106, Amsterdam, The Netherlands, September 2012. Springer. URL https://doi.org/10.1007/978-3-642-33338-5_5.

[4] Laszlo Szekeres, Mathias Payer, Tao Wei, and Dawn Song. SoK: Eternal war in memory. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, S&P '13, pages 48–62, May 2013. URL https://ieeexplore.ieee.org/document/6547101.

[5] Dokyung Song, Julian Lettner, Prabhu Rajasekaran, Yeoul Na, Stijn Volckaert, Per Larsen, and Michael Franz. SoK: Sanitizing for security. In *Proceedings of the 2019 IEEE Symposium on Security and Privacy*, S&P '19, May 2019. URL https://ieeexplore.ieee.org/document/8835294.

[6] Peter J. Denning. The internet worm. *RIACS Technical Report TR-89.3*, Feb. 1989. URL https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19900014594.pdf.

[7] Hovav Shacham. The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86). In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pages 552–561, Alexandria, Virginia, USA, 2007. URL https://dl.acm.org/citation.cfm?id=1315313.

[8] Kevin Z. Snow, Fabian Monrose, Lucas Davi, Alexandra Dmitrienko, Christopher Liebchen, and Ahmad-Reza Sadeghi. Just-in-time code reuse: On the effectiveness of fine-grained address space layout randomization. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, S&P '13, pages 574–588, Berkeley, CA, USA, May 2013. URL https://ieeexplore.ieee.org/abstract/document/6547134.

[9] Felix Schuster, Thomas Tendyck, Christopher Liebchen, Lucas Davi, Ahmad-Reza Sadeghi, and Thorsten Holz. Counterfeit object-oriented programming: On the difficulty of preventing code reuse attacks in C++ applications. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, S&P '15, pages 745–762, Oakland, CA, USA, 2015. URL https://ieeexplore.ieee.org/document/7163058.

[10] Long Cheng, Hans Liljestrand, Md Salman Ahmed, Thomas Nyman, Trent Jaeger, N. Asokan, and Danfeng Yao. Exploitation techniques and defenses for data-oriented attacks. In *Proceedings of the 2019 IEEE Cybersecurity Development (SecDev)*, pages 114–128, Tysons Corner, VA, USA, Sep. 2019. URL https://ieeexplore.ieee.org/abstract/document/8901549.

[11] PaX-Team. PaX address space layout randomization. 2003. URL http://pax.grsecurity.net/docs/aslr.txt.

[12] Nathan Burow, Scott A Carr, Joseph Nash, Per Larsen, Michael Franz, Stefan Brunthaler, and Mathias Payer. Control-flow integrity: Precision, security, and performance. *ACM Computing Surveys (CSUR)*, 50(1):16, 2017. URL https://dl.acm.org/citation.cfm?id=3054924.

[13] Miguel Castro, Manuel Costa, and Tim Harris. Securing software by enforcing data-flow integrity. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, OSDI '06, pages 147–160, Seattle, Washington, USA, 2006. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1298455.1298470.

[14] Sandeep Bhatkar and R. Sekar. Data space randomization. In *Proceedings of the 5th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, DIMVA '08, pages 1–22, Paris, France, 2008. Springer-Verlag. URL https://link.springer.com/chapter/10.1007/978-3-540-70542-0_1.

[15] Jonghwan Kim, Daehee Jang, Yunjong Jeong, and Brent Byunghoon Kang. POLaR: Per-allocation object layout randomization. In *Proceedings of the 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 505–516, June 2019. URL https://ieeexplore.ieee.org/document/8809488.

[16] Hans Liljestrand, Thomas Nyman, Kui Wang, Carlos Chinea Perez, Jan-Erik Ekberg, and N. Asokan. PAC it up: Towards pointer integrity using ARM pointer authentication. In *Proceedings of the 28th USENIX Security Symposium*, (USENIX Security 19), pages 177–194, Santa Clara, CA, USA, August 2019. URL https://www.usenix.org/system/files/sec19fall_liljestrand_prepub.pdf.

[17] Santosh Ganapati Nagarakatte. *Practical Low-Overhead Enforcement of Memory Safety for C Programs*. PhD thesis, University of Pennsylvania, 2012. URL https://www.cs.rutgers.edu/~santosh.nagarakatte/santosh-nagarakatte-dissertation.pdf.

[18] Alexandre Jean-Michel Procopi Joannou. *High-performance memory safety Optimizing the CHERI capability machine*. PhD thesis, University of Cambridge, 2019. URL https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-936.pdf.

[19] Richard WM Jones and Paul HJ Kelly. Backwards-compatible bounds checking for arrays and pointers in C programs. In *Proceedings of the 3rd International Workshop on Automatic Debugging (ADEBUG-97)*, pages 13–26. Linköping University Electronic Press, 1997. URL https://www.doc.ic.ac.uk/~phjk/Publications/BoundsCheckingForC.pdf.

[20] Emery D. Berger and Benjamin G. Zorn. DieHard: Probabilistic memory safety for unsafe languages. In *Proceedings of the 27th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '06, pages 158–168, Ottawa, Ontario, Canada, 2006. URL https://dl.acm.org/citation.cfm?id=1134000.

[21] Sam Silvestro, Hongyu Liu, Tianyi Liu, Zhiqiang Lin, and Tongping Liu. Guarder: A tunable secure allocator. In *Proceedings of the 27th USENIX Security Symposium*, Security '18, pages 117–133, Baltimore, MD, USA, 2018. USENIX Association. URL https://www.usenix.org/system/files/conference/usenixsecurity18/sec18-silvestro.pdf.

[22] Periklis Akritidis, Manuel Costa, Miguel Castro, and Steven Hand. Baggy bounds checking: An efficient and backwards-compatible defense against out-of-bounds errors. In *Proceedings of the 18th Conference on USENIX Security Symposium*, SSYM '09, pages 51–66, Berkeley, CA, USA, 2009. USENIX Association. URL https://www.usenix.org/legacy/events/sec09/tech/full_papers/akritidis.pdf.

[23] Joe Devietti, Colin Blundell, Milo M K Martin, and Steve Zdancewic. HardBound: architectural support for spatial safety of the C programming language. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XIII, 2008. URL https://dl.acm.org/citation.cfm?id=1346295.

[24] Konstantin Serebryany, Derek Bruening, Alexander Potapenko, and Dmitry Vyukov. AddressSanitizer: a fast address sanity checker. In *Proceedings of the 2012 USENIX Annual Technical Conference*, USENIX ATC '12, 2012. URL https://www.usenix.org/system/files/conference/atc12/atc12-final39.pdf.

[25] Nathan Burow, Derrick McKee, Scott A. Carr, and Mathias Payer. CUP: Comprehensive user-space protection for C/C++. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, ASIACCS 18, pages 381–392, Incheon, Republic of Korea, 2018. URL https://doi.org/10.1145/3196494.3196540.

[26] Oleksii Oleksenko, Dmitrii Kuvaiskii, Pramod Bhatotia, Pascal Felber, and Christof Fetzer. Intel MPX explained: A cross-layer analysis of the intel MPX system stack. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2(2):28:1–28:30, June 2018. URL http://doi.acm.org/10.1145/3224423.

[27] Gregory J Duck and Roland H C Yap. 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 '18, 2018. URL https://dl.acm.org/citation.cfm?id=3192388.

[28] Brooks Davis, Khilan Gudka, Alexandre Joannou, Ben Laurie, A Theodore Markettos, J Edward Maste, Alfredo Mazzinghi, Edward Tomasz Napierala, Robert M Norton, Michael Roe, Peter Sewell, Robert N M Watson, Stacey Son, Jonathan Woodruff, Alexander Richardson, Peter G Neumann, Simon W Moore, John Baldwin, David Chisnall, James Clarke, and Nathaniel Wesley Filardo. CheriABI: enforcing valid pointer provenance and minimizing pointer privilege in the POSIX C run-time environment. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, 2019. URL https://dl.acm.org/citation.cfm?id=3304042&dl=ACM&coll=DL.

[29] Feng Qin, Shan Lu, and Yuanyuan Zhou. SafeMem: exploiting ECC-memory for detecting memory leaks and memory corruption during production runs. In *Proceedings of the IEEE 11th International Symposium on High Performance Computer Architecture*, HPCA '05, 2005. URL https://ieeexplore.ieee.org/document/1385952.

[30] Trevor Jim, J. Greg Morrisett, Dan Grossman, Michael W. Hicks, James Cheney, and Yanling Wang. Cyclone: A safe dialect of C. In *Proceedings of the General Track of the Annual Conference on USENIX Annual Technical Conference*, ATEC '02, pages 275–288, Berkeley, CA, USA, 2002. USENIX Association. URL http://trevorjim.com/papers/usenix2002.pdf.

[31] Kanad Sinha and Simha Sethumadhavan. Practical memory safety with REST. In *Proceedings of the 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture*, ISCA '18, pages 600–611, June 2018. URL https://dl.acm.org/citation.cfm?id=3276598.

[32] Archibald Samuel Elliott, Andrew Ruef, Michael Hicks, and David Tarditi. Checked C: Making C safe by extension. In *Proceedings of the 2018 IEEE Cybersecurity Development*, SecDev'18, pages 53–60, Cambridge, MA, USA, Sep 2018. URL https://ieeexplore.ieee.org/document/8543387.

[33] Joe Bialek, Ken Johnson, Matt Miller, and Tony Chen. Security analysis of memory tagging. Technical report, Microsoft Security Response Center (MSRC), March 2020. URL https://github.com/microsoft/MSRC-Security-Research/blob/master/papers/2020/Security%20analysis%20of%20memory%20tagging.pdf.

[34] Kostya Serebryany, Evgenii Stepanov, Aleksey Shlyapnikov, Vlad Tsyrklevich, and Dmitry Vyukov. Memory tagging and how it improves C/C++ memory safety. *arXiv.org*, February 2018. URL https://arxiv.org/pdf/1802.09517.pdf.

[35] Gregory J. Duck and Roland H. C. Yap. Heap bounds protection with low fat pointers. In *Proceedings of the 25th International Conference on Compiler Construction*, CC '16, pages 132–142, Barcelona, Spain, 2016. URL http://doi.acm.org/10.1145/2892208.2892212.

[36] Albert Kwon, Udit Dhawan, Jonathan M. Smith, Thomas F. Knight, Jr., and Andre DeHon. Low-fat pointers: Compact encoding and efficient gate-level implementation of fat pointers for spatial safety and capability-based security. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 721–732, Berlin, Germany, 2013. URL http://doi.acm.org/10.1145/2508859.2516713.

[37] Martin Maas, Krste Asanoviundefined, and John Kubiatowicz. A hardware accelerator for tracing garbage collection. In *Proceedings of the 45th Annual International Symposium on Computer Architecture*, ISCA 18, pages 138–151, Los Angeles, CA, USA, 2018. IEEE Press. URL https://doi.org/10.1109/ISCA.2018.00022.

[38] Sam Ainsworth and Timothy M Jones. Markus: Drop-in use-after-free prevention for low-level languages. In *Proceedings of the 2020 IEEE Symposium on Security and Privacy*, S&P '20, Oakland, CA, USA, 2020. URL https://www.cl.cam.ac.uk/~tmj32/papers/docs/ainsworth20-sp.pdf.

[39] Thurston H.Y. Dang, Petros Maniatis, and David Wagner. Oscar: A practical page-permissions-based scheme for thwarting dangling pointers. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 815–832, Vancouver, BC, August 2017. USENIX Association. URL https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/dang.

[40] Tong Zhang, Dongyoon Lee, and Changhee Jung. BOGO: buy spatial memory safety, get temporal memory safety (almost) free. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, 2019. URL https://dl.acm.org/citation.cfm?id=3304017.