Memory Safety

Hamed Okhravi, MIT Lincoln Laboratory (hamed.okhravi@ll.mit.edu)

"You will observe with concern how long a useful truth may be known and exist, before it is generally received and practiced on." -- Benjamin Franklin

In 1972, the Computer Security Technology Planning Study report of the US Air Force reads: “The code performing this function does not check the source and destination addresses properly, permitting portions of the monitor to be overlaid by the user. This can be used to inject code into the monitor that will permit the user to seize control of the machine” [1]. This is an example of what later became known as memory corruption attacks, which were famously popularized by Phrack’s “Smashing the Stack for Fun and Profit” [2] in 1996. Despite being one of the earliest classes of vulnerabilities studied, memory safety bugs continue to one of the most widespread in modern systems. Studies by Microsoft [3] and Google [4] report that consistently around 70% of cyber vulnerabilities are memory safety bugs. This is despite decades of research into sanitizers and exploit mitigation techniques [5] that attempt to detect such vulnerabilities and complicate their exploitation. The persistence of memory safety vulnerabilities can perhaps be attributed to widespread usage of memory-unsafe programming languages such as C/C++ in commodity applications, libraries, and operating systems, and lack of hardware support for efficient enforcement of memory safety in commodity processors.

Recent advancements in system-level languages with strong memory safety guarantees (e.g., Rust) and tools and techniques to make large-scale, legacy C/C++ code bases partially or fully memory safe have created a new hope in this area. This is perhaps best exemplified in its acknowledgement in the highest national-level strategies. The 2024 White House Report on the Cybersecurity Posture of the United States [6], for example, mentions the goal to promoted the adoption of “memory-safe programing languages”. In addition, in 2023, NSA joins CISA and other partners in releasing a report [7], making the case for a memory-safe roadmap, in which they “urge senior executives at every software manufacturer to reduce customer risk by prioritizing design and development practices that implement memory-safe languages”.

This special issue of IEEE Security & Privacy aims to highlight recent advancements in memory safety research with an emphasis on solutions. Perhaps the most principled approach for addressing memory safety is using safe programming languages such as Rust. However,

This material is based upon work supported by the Under Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Under Secretary of Defense for Research and Engineering.

redevelopment of existing codebases in such safe languages is a very costly and time-consuming proposition.

Wallach and Lord [A1] share their viewpoint for automated translation of existing C codebases into Rust.

Larsen[A2] describes the state-of-the-art in automated translation of C to Rust using a tool called C2Rust and explains the nuances and challenges of generating safe, idiomatic Rust code automatically. He also proposes a number of ideas including the usage of ML to facilitate such translations.

Before such tools are fully matured, however, existing C/C++ codebases need to be secured against memory corruption attacks. This goal can be achieved either in software or by leveraging hardware extensions. Software-only approaches have the advantage of not requiring new hardware, but may impose larger runtime overheads. Hardware-based approaches, on the other hand, are more efficient, but require the deployment of new hardware.

Nagarakatte[A3] proposes full spatial and temporal memory safety for C by maintaining and runtime checking of metadata. Such runtime enforcement can be further optimized by additional support from hardware.

Huang et al.[A4] propose comprehensive memory safety validation that identifies the subset of memory objects that require runtime protection and isolating them from the rest of the objects. This significantly reduces the overhead of memory safety enforcement at runtime.

Watson et al.[A5] propose the CHERI system, a hardware-software co-design available on top of ARM Morello and Microsoft CHERIoT Ibex cores, that allows efficient enforcement of memory safety for existing C/C++ codebases.

Robertson and Egele[A6] share their viewpoint on how architectural security primitives allow efficient enforcement of heap safety.

Memory safety does not always have to be deterministic, however. Probabilistic techniques can provide effective ways of enforcing safety with low overhead through randomization, replication, and diversification. Such techniques are collectively known as *moving target defenses* [8].

Andre et al.[A7] propose a probabilistic memory safety technique known as Multi-Variant Execution (MVX). In MVX, multiple variants of a program are run concurrently and are cross-checked. Divergence in execution among the variants would be indicative of an attack.

While hardware support allows for efficient enforcement of memory safety, hardware itself is not immune from vulnerabilities. The complexity of modern hardware gives rise to a wide range of architectural and micro-architectural vulnerabilities. These vulnerabilities can exist in the hardware itself or in the interfaces between hardware and software.

Rostami et al.[A8] describe a fuzzing technique for detecting and mitigating architectural and microarchitectural memory safety vulnerabilities in hardware.

Cloosters et al.[A9] describe memory safety vulnerabilities that arise at the interface of software and hardware, in particular in Trusted Execution Environments (TEEs), and proposes approaches for building memory-safe enclaves.

Finally, while there is work to converge these research thrusts, a unified, holistic vision for memory safety remains an open research problem. Liljestrand and Ekberg[A10] share their viewpoint for harmonizing diverse memory safety fronts.

Acknowledgement:

DISTRIBUTION STATEMENT A. Approved for public release. Distribution is unlimited.

This material is based upon work supported by the Under Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Under Secretary of Defense for Research and Engineering.

Bios:

Hamed Okhravi is a senior staff member at the Massachusetts Institute of Technology (MIT) Lincoln Laboratory, Lexington, Massachusetts, 02421, USA. Okhravi received a Ph.D. in electrical and computer engineering from the University of Illinois at Urbana-Champaign. He is also the recipient of two Best Paper Awards, three R&D 100 Awards, the FLC Excellence in Technology Transfer Award, MIT Lincoln Laboratory's Best Invention and Early Career Technical Achievement Awards, and the NSA's Best Scientific Cybersecurity Paper Award. He is a Senior Member of IEEE. Contact him at hamed.okhravi@ll.mit.edu.

Appendix: Related Articles

A1. D. Wallach and B. Lord, "A viewpoint on Memory Safety Manifesto," IEEE Security Privacy, …

A2. P. Larsen, "Migrating C to Rust for Memory Safety," IEEE Security Privacy, …

A3. S. Nagarakatte, "Full Spatial and Temporal Memory Safety for C," IEEE Security Privacy, …

A4. K. Huang et al., “Comprehensive Memory Safety Validation: An Alternative Approach to Memory Safety,” IEEE Security Privacy, …

A5. R. Watson et al., “CHERI: Hardware-Enabled C/C++ Memory Protection at Scale,” IEEE Security Privacy, …

A6. W. Robertson and M. Egele, “A viewpoint on Safer Heaps with Practical Architectural Security Primitives,” IEEE Security Privacy, …

A7. A. Rosti et al., “The Astonishing Evolution of Probabilistic Memory Safety: From Basic Heap-Data Attack Detection towards Fully Survivable Multi-Variant Execution,” IEEE Security Privacy, …

A8. M. Rostami et al., “Fuzzerfly Effect: Hardware Fuzzing for Memory Safety,” IEEE Security Privacy, …

A9. T. Cloosters et al., “Memory Corruption at the Border of Trusted Execution,” IEEE Security Privacy, …

A10. H. Liljestrand and J. Ekberg, “A viewpoint on Harmonizing the Diverse Memory Safety Fronts,” IEEE Security Privacy, …

References:

[1] J. Anderson, “Computer Security Technology Planning Study”, Electronic Systems Division, US Air Force, Report ESD-TR-73-51, 1972
https://csrc.nist.rip/publications/history/ande72.pdf

[2] One, Aleph. "Smashing the stack for fun and profit." Phrack magazine 7.49 (1996): 14-16.

[3] Microsoft® (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

[4] Google (2021), "An update on Memory Safety in Chrome".
https://security.googleblog.com/2021/09/an-update-on-memory-safety-in-chrome.html

[5] Song, Dokyung, et al. "SoK: Sanitizing for security." 2019 IEEE Symposium on Security and Privacy (SP). IEEE, 2019.

[6] The White House, 2024 Report on The Cybersecurity Posture of the United States, May 2024 https://www.whitehouse.gov/wp-content/uploads/2024/05/2024-Report-on-the-Cybersecurity-Posture-of-the-United-States.pdf

[7] CISA, NSA, and other partners, The Case for Memory Safe Roadmaps, December 2023 https://media.defense.gov/2023/Dec/06/2003352724/-1/-1/0/THE-CASE-FOR-MEMORY-SAFE-ROADMAPS-TLP-CLEAR.PDF

[8] H. Okhravi, T. Hobson, D. Bigelow and W. Streilein, "Finding Focus in the Blur of Moving-Target Techniques," in IEEE Security & Privacy, vol. 12, no. 2, pp. 16-26, Mar.-Apr. 2014, doi: 10.1109/MSP.2013.137.