It is time to standardize principles and practices for software memory safety

Robert N. M. Watson[*†], John Baldwin[††], Tony Chen[‡], David Chisnall[**], Jessica Clarke[*], Brooks Davis[¶], Nathaniel Wesley Filardo[**‡], Brett Gutstein[*], Graeme Jenkinson[†], Ben Laurie[*#†], Alfredo Mazzinghi[†], Simon W. Moore[†*], Peter G. Neumann[¶], Hamed Okhravi[‡‡], Alex Rebert[#], Alex Richardson[#], Peter Sewell[*], Laurence Tratt[¶¶], Muralidaran Vijayaraghavan[#], Hugo Vincent[##], and Konrad Witaszczyk[*]

* University of Cambridge
† Capabilities Limited
¶ SRI International
# Google, Inc
‡ Microsoft, Inc
** SCI Semiconductor
†† Ararat River Consulting
¶¶ King's College London
Arm Limited
‡‡ MIT Lincoln Laboratory

This article appeared in Communications of the ACM, vol. 68, no. 2, pp. 40-45, February 2025.

Twenty-one coauthors, spanning academia and industry, with expertise in memory-safety research, deployment, and policy, argue that standardization is an essential next step to achieving universal strong memory safety.

Introduction

For many decades, endemic memory-safety vulnerabilities in software Trusted Computing Bases (TCBs) have enabled the spread of malware and devastating targeted attacks on critical infrastructure, national-security targets, companies, and individuals around the world. Over the last two years, the information-technology industry has seen increasing calls for the adoption of memory-safety technologies, framed as part of a broader initiative for *Secure by*

Design, from government[1] [2] [3] [4], academia[5], and within the industry itself[6] [7]. These calls are grounded in extensive evidence that memory-safety vulnerabilities have persistently made up the majority of critical security vulnerabilities over multiple decades, and have affected all mainstream software ecosystems and products – and also the growing awareness that these problems are almost entirely avoidable by using recent advances in strong and scalable memory-safety technology.

In this article, we explore *memory-safety standardization*, which we argue is an essential step to promoting *universal strong memory safety* in government and industry, and, in turn, to ensure access to more secure software for all. Over the last two decades, a set of research technologies for *strong memory safety* – memory-safe languages, hardware and software memory protection, formal approaches, and software compartmentalization – have reached sufficient maturity to see early deployment in security-critical use cases. However, there remains no shared, technology-neutral terminology or framework with which to **specify memory-safety requirements**. This is needed to enable reliable specification, design, implementation, auditing, and procurement of strongly memory-safe systems. Failure to speak in a common language makes it difficult to understand the possibilities or communicate accurately with each other, limiting perceived benefits and hence actual demand. The lack of such a framework also acts as an impediment to potential future policy interventions, and as an impediment to stating requirements to address observed market failures preventing adoption of these technologies. Standardization would also play a critical role in improving industrial best practice, another key aspect of adoption.

This article is a distilled version of a longer technical report published by the same authors, which includes further case studies and applications, as well as considering the potential implications of various events and interventions on potential candidate adoption timelines[8].

---

[1] The White House, *Back to the Building Blocks: A Path Towards Measurable Security*, February 2024, https://www.whitehouse.gov/wp-content/uploads/2024/02/Final-ONCD-Technical-Report.pdf.
[2] CISA, NSA, FBI, ASD's ACSC, CCCS, NCSC-UK, NCSC-NZ, and CERT-NZ, *The Case for Memory Safe Roadmaps Why Both C-Suite Executives and Technical Experts Need to Take Memory Safe Coding Seriously*, December 2023, https://www.cisa.gov/sites/default/files/2023-12/The-Case-for-Memory-Safe-Roadmaps-508c.pdf.
[3] NSA, *Software Memory Safety*, Cybersecurity Information Sheet, April 2023, https://media.defense.gov/2022/Nov/10/2003112742/-1/-1/0/CSI_SOFTWARE_MEMORY_SAFETY.PDF.
[4] Department for Business, Energy & Industrial Strategy, *Confronting cyber threats to businesses and personal data*, October 2019, https://www.gov.uk/government/news/confronting-cyber-threats-to-businesses-and-personal-data.
[5] H. Okhravi, *Memory Safety*, IEEE Security & Privacy, vol. 22, no. 4, pp. 13-15, July-August 2024, https://ieeexplore.ieee.org/document/10621922.
[6] Alex Rebert and Christoph Kern, *Secure by Design: Google's Perspective on Memory Safety*, 2024, https://storage.googleapis.com/gweb-research2023-media/pubtools/7665.pdf.
[7] Satya Nadella, *Prioritizing security above all else*, May 2024, https://blogs.microsoft.com/blog/2024/05/03/prioritizing-security-above-all-else/.
[8] Robert N. M. Watson, John Baldwin, Tony Chen, David Chisnall, Jessica Clarke, Brooks Davis, Nathaniel Wesley Filardo, Brett Gutstein, Graeme Jenkinson, Ben Laurie, Alfredo Mazzinghi, Simon W. Moore, Peter G. Neumann, Hamed Okhravi, Alex Rebert, Alex Richardson, Peter Sewell, Laurence Tratt, Murali Vijayaraghavan, Hugo Vincent, and Konrad Witaszczyk, *It is time to standardize principles and practices for software memory safety (extended version)*, February 2025, https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-996.pdf.

Background

For over two decades, memory-safety vulnerabilities have consistently made up around two thirds of critical security vulnerabilities in every major open-source and proprietary software TCB, including Windows,[9] Linux, Android, iOS, Chromium,[10] OpenJDK, FreeRTOS, and others. These problems primarily originate from an existing multi-billion line-of-code C/C++ code corpus that is difficult (probably impossible in practice) to entirely replace due to its scale. Of particular importance within this are the language runtimes of many type-safe and/or memory-safe programming languages, such as Java, JavaScript, and Python, which are often implemented in (or depend heavily on) C and C++.

Memory-safety vulnerabilities are particularly important because, when combined with network communications or other malignant data, they can enable an attacker to escalate (via a multi-step exploit chain) to arbitrary code execution, operating outside the confines of the programming language[11]. These vulnerabilities have proven impossible to completely prevent with conventional engineering, and are especially dangerous because a single error (perhaps one line in a multi-million line-of-code system) is sufficient to achieve total control of a vulnerable system.

Defensive techniques have not stood still – a series of incremental (and reactive) mitigation techniques have (in the short term) complicated work for attackers – but in the longer term these simply contributed to an evolving arms race with attack techniques that are able to bypass them[12]. Despite countless hours of manual source-code auditing, and significant investments in static analysis tooling and fuzzing, the rate of memory-safety vulnerabilities has remained roughly constant for over two decades.

Mitigation and sanitization techniques frequently fail in the longer term because they are **incomplete** (e.g., PAC or CFI, which defend against only a narrow range of attack techniques, or a limited set of vulnerability types identifiable with specific static analysis tools) and/or because they are **probabilistic** (e.g., because they utilize secrets or keys that can be leaked or guessed, such as ASLR or MTE). These increasingly widely deployed techniques, which reflect *current industry best practice* in software TCBs, include:

Table 1: Current industry best practice.

Category	Description	Examples
Development-time techniques (static and dynamic)	Automated static and dynamic bug finding	Coverity and Fortify; fuzzing combined with dynamic techniques such as the Valgrind, ASAN, MSAN, and UBSAN sanitizers; subsets of otherwise

[9] David Weston, *Windows 11: The journey to security by default*, BlueHat IL, 2023, See slide 38: https://github.com/dwizzzle/Presentations/blob/master/David%20Weston%20-%20Windows%2011%20Security%20by-default%20-%20Bluehat%20IL%202023.pdf.
[10] Google, *Memory safety*, The Chromium Projects' documentation. Originally published 2020. https://www.chromium.org/Home/chromium-security/memory-safety/.
[11] Haroon Meer, *Memory Corruption Attacks: The Almost Complete History*, BlackHat 2010.
[12] László Szekeres, Mathias Payer,Tao Wei, and Dawn Song, *SoK: Eternal War in Memory*, 2013 IEEE Symposium on Security and Privacy, Berkeley, CA, USA, May 2013.

		unsafe languages that can reduce exposure to memory-safety issues, such as MISRA C/C++
Run-time techniques	Systems that handle violations of memory safety at run time, coercing them into fail stops, masking their effects, or limiting their exploitability	Software-only techniques such as stack canaries, ASLR and CFI, and also hardware-enabled techniques such as PAC, MTE, W^X (a.k.a. DEP), and architectural "safe stacks"

Fortunately, the last decade has seen the maturation of practically deployable research technologies that have a realistic chance of breaking that arms race in favor of the defending side, introducing **strong memory safety** that **non-probabilistically prevents** a broad set of memory-safety vulnerabilities and attack techniques in critical software TCBs. Broadly, these technologies, now seeing *early industrial adoption* in software TCBs, fall into four categories:

Table 2: Strong memory-safety techniques.

Category	Description	Examples
Memory-safe and type-safe languages	Fully memory-safe and/or type-safe languages; statically checkable safe subsets of otherwise unsafe languages	Rust, Python, Swift, Java, C#, and SPARK, OCaml - excluding code written in their unsafe fragments (e.g., Unsafe Rust); memory-safe C++ subsets[13]
Formal methods	Mathematically rigorous formal verification of memory safety, and broader safety/correctness properties for memory-safety TCBs themselves	Machine-checked formal proofs, in tools such as Coq, Isabelle, or Lean, of systems such as CompCert or seL4; formal verification of code in unsafe language fragments, such as RustBelt
Hardware memory protection	Systems that deterministically detect violations of memory safety at run time, coercing them into fail stops, masking their effects, or preventing their exploitation	CHERI C/C++ memory safety[14]

[13] LLVM Project, *C++ Safe Buffers*, Retrieved December 2024, https://clang.llvm.org/docs/SafeBuffers.html.

[14] Robert N.M. Watson, David Chisnall, Jessica Clarke, Brooks Davis, Nathaniel Wesley Filardo, Ben Laurie, Simon W. Moore, Peter G. Neumann, Alexander Richardson, Peter Sewell, Konrad Witaszczyk, and Jonathan Woodruff. *CHERI: Hardware-Enabled C/C++ Memory Protection at Scale*. IEEE Security & Privacy, vol. 22, no. 04, pp. 50-61, July-August 2024.

Software fault isolation and software compartmentalization	Systems that allow continued operation through privilege minimization despite effective exploitation of memory unsafety, limiting further rights and attack surfaces exposed to attackers. These systems are frequently built on the above techniques, and add the further ability to constrain attacks that have already achieved arbitrary code execution.	Deterministic sandboxing using processes or virtual machines as found in iOS and Android, software-only techniques such as eBPF and WASM, and hardware-enabled techniques such as CHERI compartmentalization

This work has not happened in isolation: concepts such as hardware memory protection and type-safe programming languages have existed almost since the inception of computer systems. However, these current technologies are incrementally adoptable within current hardware or software stacks, and their growing maturity comes alongside an increasingly critical need for memory safety.

Industrial best practices and market failure

Universally deployed strong memory safety enabled by new memory-safety protection technologies presents a remarkable opportunity. However, our excitement is tempered by the understanding that it will require a substantial change in approach by an industry that may see little economic incentive to change the status quo – and, in fact, the real risk of market disadvantage in doing so.

Today, common industrial best practices consist of the widespread use of memory-unsafe languages and coding practices in even our most sensitive computing environments, albeit with some adoption of incomplete or probabilistic memory-safety mitigations such as those described in Table 1. Changes such as the widespread deployment of CHERI hardware, the Rust language, or formal verification are challenging in several ways. They would involve immediate (perceived or real) deployment or development costs, due to disruption of existing software ecosystems, and also (and more importantly) require vendors to potentially divert engineering resources from other areas of development – a high **opportunity cost**[15]. Some vendors may find these costs difficult to justify when the immediate benefits of strong memory safety are not clearly expressed through market signals or when customer demand appears focused on other features.

[15] National Academies of Sciences, Engineering, and Medicine. *Workshop on Secure Building Blocks for Trustworthy Systems*, panel discussion with Robert Watson and Richard Grisenthwaite, Seattle, Washington, USA, 31 July 2024. https://www.nationalacademies.org/event/43213_07-2024_workshop-on-secure-building-blocks-for-trustworthy-systems.

We suggest that the slow adoption of strong memory safety in spite of its clear security benefits may reflect a potential **market failure**[16]: society, as a whole, pays an extremely high cost for memory-safety vulnerabilities[17], as well as taking on a very high risk as these vulnerabilities are present in essentially all critical infrastructure, national security applications, and systems protecting financial and privacy-sensitive data. The history of catastrophic failure associated with these vulnerabilities can be traced at least as far back as the Morris Worm in 1988[18], and many recent examples include ransomware spread[19] or widespread denial of service[20] originating from memory-safety issues.

This analysis is consistent with many other past economic analyses of security, in which **negative security impact is an externality**[21] uncaptured by production costs, sales of products, or (beyond the short term) market cap. The potential cost savings of avoiding deployment of strong memory safety are immediate, tangible and concentrated, while the costs arising from failures are often delayed or dispersed. The market thus provides little immediate pressure on vendors to prioritize strong memory safety, and the financial implications of failing to do so tend to be externalized from vendors, through mechanisms like after-market solutions, disaster recovery, and national security implications, with billions of dollars in damage arising from even a small number of high-profile incidents and data breaches.

This disconnect between the cost of insecurity and the responsibility for mitigating that risk, compounded by two-sided incomplete information, can result in under-investment in robust security measures. The knowing, continued use of memory-unsafe technologies with serious consequences for both individuals and society may be enabled, in part, by a **lack of direct feedback from the market, lack of liability for the impact of product defects, and the challenges all market participants face in accurately assessing the risks and benefits of memory safety.**

Enabling business processes and market interventions

A detailed analysis of this apparent market failure, and potential interventions affecting incentives, is beyond the scope of this paper. However, we observe that a common requirement for many conceivable interventions (and a gap in current thinking) – for example, in regulating consumer electronics or in informing government procurement – is the ability to **concisely express strong memory-safety requirements or guarantees in a technology-neutral manner**. This becomes obvious when trying to imagine how one might:

---

[16] Kopp, Emanuel, Lincoln Kaffenberger, and Nigel Jenkinson. *Cyber risk, market failures, and financial stability*. International Monetary Fund, 2017.
[17] Andy Greenberg, *Ransomware Payments Hit a Record $1.1 Billion in 2023*, Wired Magazine, Published February 2024, https://www.wired.com/story/ransomware-payments-2023-breaks-record/.
[18] U.S. v. Morris, 928 F.2d 504 (2d Cir. 1991). https://scholar.google.com/scholar_case?case=551386241451639668.
[19] NHS Digital, *WannaCry Ransomware Using SMB Vulnerability*. Originally published 2017. https://digital.nhs.uk/cyber-alerts/2017/cc-1411.
[20] CISA, *Widespread IT Outage Due to CrowdStrike Update*. Originally published 2024. https://www.cisa.gov/news-events/alerts/2024/07/19/widespread-it-outage-due-crowdstrike-update.
[21] Ross Anderson and Tyler Moore, *The Economics of Information Security*, 2006. https://www.science.org/doi/abs/10.1126/science.1130992

- Improve industrial best practice to utilize strong memory-safety solutions in all areas
- Enable concise acquisition requirements that incorporate memory safety
- Enable reliable and meaningful procurement of strongly memory-safe systems
- Inform product liability legislation and insurance
- Enable review and audit of systems for strong memory safety
- Enable test and evaluation (T&E) for memory safety
- Enable Common Criteria Certification Requirements to include lab-certifiable memory safety requirements
- Enable subsidies, tax incentives, or other mechanisms to encourage the rapid adoption of strong memory safety
- Support regulatory interventions to mandate the use of security best practices including strong memory safety in specific classes of products or use cases
- Define safe harbor provisions in a potential software liability regime

We see a set of closely linked problems that must be resolved in order to lift industrial best practices, enable business changes (such as expressing strong memory-safety requirements during procurement), or support potential market interventions (such as regulation of critical infrastructure technologies to ensure use of strong memory safety):

- Develop an **intellectual framework** that allows these diverse technologies and approaches to be consistently described, with their benefits and costs documented in common language that can be used in reasoning about potential use cases
- Develop and document **improvements to current industrial practices**, based on these technologies, able to support the development and composition of strongly memory-safe systems in a manner acceptable to industry
- Enable the clear **enunciation of technology-neutral memory-safety requirements** facilitated by these technologies, and of improved practices for the purposes of acquisition, compliance, regulation, composition, and so on.

The memory-safety standardization gap

When designing, implementing, test and evaluating, certifying, and procuring systems able to resist attacks on memory-safety vulnerabilities, it is easy to imagine a broad range of desirable policies enabled by new memory-safety technologies and accompanying improved industrial best practices; for example:

- A smartphone's general-purpose OS and all of its network-facing applications must be implemented with at least non-deterministic data and control-flow pointer protections within five years, and strong memory safety within fifteen years. Mobile device management (MDM) systems must support enterprises administratively prohibiting installation of memory-unsafe applications.
- All data-center TCBs responsible for isolating hosted government systems from each other, and from other customers, must be implemented with strong memory safety within fifteen years.
- All networking infrastructure (such as wireless access points) or cyber-physical systems where software interacts with the physical environment (such as automobiles or certain IoT devices such as smart locks, security cameras, smart

thermostats, etc.) shipped after 2034 must be implemented with strong memory safety.
- Smart phones and IoT devices using machine-learning models on sensitive personal data, such as inputs from cameras, microphones, GPS and other sensors as well as stored data preserved in order to answer questions such as “where are my glasses?” must, by 2040, be strongly isolated using compartmentalization in order to preserve the privacy and integrity of the data, particularly from the large number of other applications typically running on the same device.

Today, however, there is no consistent and widely adopted means to signal these types of general requirements for memory safety, nor even specific choices (such as a requirement for deterministic memory safety).

Audiences for memory-safety standardization

Two closely related goals of standards are to (a) allow the clear and practical communication of requirements between consumers of systems and those providing or implementing them, and (b) similarly allow those providing or implementing systems to describe conformance of systems to consumers. Important audiences for this work would include:

- Those specifying requirements for acquisition (e.g., US DoD, US GSA, UK MoD, and UK NCSC).
- Memory-safety system designers and implementers (e.g., the authors of Rust or OCaml, or those adapting operating systems to support CHERI).
- Application software designers and developers (e.g., the authors of Firefox or Chrome).
- Industrial bodies specifying approaches and technologies to be used within specific sectors (e.g., AutoSAR for automotive systems).
- Government and/or regulatory bodies seeking to incentivize rapid adoption of strong memory safety, or limit the use of memory-unsafe systems through laws, liability, tax incentives, or other mechanisms
- End system designers, implementers, and integrators (e.g., designers of a smartphone product).
- Test and evaluation (T&E), certification, and accreditation bodies (e.g., Common Criteria testing laboratories, DOT&E, external security auditors, system integrators, and administrators).
- Those educating future designers, engineers, and others (e.g., those teaching computer science in universities, or [re-]training staff within companies).

Goals for memory-safety standardization

We argue that **standardizing memory safety** is an essential step to widespread adoption of strong memory-safety technologies. Currently, those technologies are seeing early use in selected critical use cases in government and industry – especially in roots of trust and prototypes of more secure IoT or cloud infrastructure. Examples include the use of Rust in an increasing number of “from-scratch” software components, and Microsoft’s CHERIoT-Ibex processor seeing early deployment across multiple key industry players. We believe that there are multiple gaps, which this work would aim to fill through the development of

both a technology-neutral framework for memory safety, and technology-specific mappings of that framework alongside guidance for their use:

- Develop broad, cross-sector technical consensus on a **practical systemization of strong memory-safety properties and a clear intellectual framework** in which to explain their strengths and weaknesses, appropriate use cases, and so on. This would include classifying sets of technologies based on properties such as coverage of attacks, contributions to abstract memory-safety goals (such as spatial or temporal safety), being probabilistic/secrets-based or deterministic, support for compartmentalization, the potential need for total software rewrites or ABI changes, dependencies on new underlying hardware, potential costs in use and deployment, and so on. It would also explore the tension between design principles underlying memory-protection technologies (e.g., definitions and implementations of topics such as “spatial safety”, “temporal safety”, etc.) versus a vulnerability-oriented perspective (in which memory safety is defined in terms of known forms of memory unsafety).
- Define **best practices for the use of specific memory-safety technologies**, with respect to this framework, such as when and to what extent dependence on unsafe Rust code is suitable within larger Rust software systems, guidelines on structuring such dependencies to support compositional reasoning about safety, the uses of CHERI C and C++ that maximize safety, how to validate whether the implemented hardware-software stack correctly makes use of the memory-safety features, etc.
- Consider the **implications of composing multiple technologies**, which will frequently be present in complete computer systems or products – for example, a C-language OS kernel and C++ language run-time (protected weakly by current mitigation techniques or more strongly with CHERI C/C++ in the future), and an application stack written in a type-safe and memory-safe language.

To be successful, we believe that a memory-safety standardization framework must:

- **Incorporate existing weaker protection technologies**, enabling their specification while also making clear that they are points on a longer-term – and escalating – roadmap for memory safety. It is essential to recognize current industry leaders' efforts in creating and deploying weaker but more accessible technologies within industry.
- **Focus on enabling approaches that are technology and vendor neutral**, which will avoid hampering future procurement processes that require independent competing proposals. For example, a clear request for “strong memory safety” in a requirements statement might be satisfied by either Rust or CHERI C/C++ in a responding proposal.
- **Make clear the boundaries between industrial best practice and ongoing research** to: (a) prevent premature engagement with still immature aspects of memory-safety technologies, (b) reassure implementers that likely extensions to current strong memory-safety technologies will be incrementally adoptable, and (c) lay out a long-term roadmap for future memory-safety technology improvements.
- **Establish tiered safety assurance levels** to guide technology selection based on requirements and constraints, acknowledging their varying costs.
- **Provide distinct guidance for new systems and existing codebases**, recognizing that different strategies may be necessary depending on the context.

Potential structures for one or more standards or documents

It would be premature to try to fix the best structure for the results of this effort. It seems likely that it could include some combination of standards, engineering best practices, and/or technical reports written for a specific audience (e.g., application software designers). However, we expect that they should, in some form address the following:

- Define, in a technology- and vendor-neutral form, a **standard terminology** and an **intellectual framework** for discussing and specifying memory-safety principles and impacts.
- Define **engineering practices in a technology- and vendor-neutral form**, considering topics such as TCB minimization, interoperability with legacy memory-unsafe components to be deprecated or adapted to memory safety in the future, management of weaknesses or omissions in memory-safety technologies, identification of potential performance and/or power efficiency changes, composition of multiple parts utilizing different memory-safety technologies, and documentation practices aimed to support review and assessment.
- Define, per-technology, **engineering best practices specific to each technology** (e.g., for use of CHERI, Rust, etc.)
- Define a methodology for reasoning about the **composition of multiple forms of memory safety** within a single system.
- Provide guidance on memory-safety **T&E**, **review, and assessment practice**.

Adoption narratives and timelines

A key function of this work will be to enable longer-term adoption narratives for memory-safe systems. Of particular interest to us are two classes of widely used systems:

- **Industrial best-practice systems** utilize rigorous and engaged practices employed for commodity software at well-funded companies such as Microsoft, Apple, and Google in developing platforms for application writers. Today, these vendors are aggressively adopting memory-safety mitigation technologies such as ASLR and hardware-enabled cryptographic pointer protections.
- **Security- and privacy-critical systems** reflect engineering used specifically for essential TCBs in mission-critical systems such as those that are used in critical and national infrastructure, defenses, and aerospace. Today, vendors of such systems are already engaging with selective deployment of strong memory-safety technologies such as Rust and CHERI.

We also differentiate new systems from legacy ones – it is easiest to deploy these technologies in the design of a fresh system, especially when new software ecosystems may be created, than it is to deploy them into existing ones. A clear challenge with this narrative is that entirely new systems are only built infrequently, and even where they could be written with a memory-safe language from scratch, they will be created within a large pre-existing memory-unsafe ecosystem that would also need to be migrated or have suitable interfaces

created[22]. To facilitate a gradual transition, new components built with memory-safety technologies must be able to interoperate with existing unsafe legacy components.

Candidate timeline

Establishing potential timelines for adoption is challenging given the potential for enabling interventions of research combined with historically strong industrial reluctance to adopting disruptive technologies with less clear translation into concrete consumer demand. The following candidate timeline has been developed based on what we see as realistic timelines given the state of the technology, combined with evolving thinking on potential interventions including the growing appetite for regulation of technologies that have strong impacts on personal data privacy, especially around machine learning, as well as in growing interest in software liability, which might help motivate improvements in industrial practice:

Period	Sector	Narrative
2018-2027 (current period)	Industry best practices	Industry leaders widely deploy **probabilistic protection** techniques such as ASLR and PAC in well engineered, non-critical applications and devices.
	Security-critical applications	Newly designed critical devices and software systems from industry leaders and national security system acquisition are adopting **deterministic memory-safety** technologies such as Rust and CHERI, and selected use of **formal methods**.
2028-2037 (coming decade)	Industry best practices	Over the course of this decade, newly designed non-critical devices and software systems will increasingly ship with partial or complete **deterministic memory-safety**. The use of branding and certification schemes to clearly signpost less-safe systems as damaging to security and privacy; organizations increasingly require policy exemptions for use of memory-unsafe systems in more security-sensitive environments. Legacy systems and applications continue to use probabilistic protection where it is economically infeasible to transition, but at potentially growing cost due to a **shift in industry best practice** leaving vendors open to product liability claims or regulatory problems. Component deprecation and support stoppage may act as force functions to retire legacy components. Toward the tail end of this period, it becomes reasonable for **insurers to incentivise the use of memory safety**, both with respect to software development (professional indemnity insurance) and software procurement and deployment (cybersecurity insurance), as well as for regulators setting standards for next generations of devices to require the use of strong memory safety

[22] Samuel Mergendahl, Nathan Burow, and Hamed Okhravi. *Cross-Language Attacks*. NDSS. 2022.

	Security-critical applications	**Near universal adoption of deterministic memory safety** in newly deployed systems by the end of the decade, with significant regulatory, acquisition requirement, insurance efforts to ensure memory safety in new systems, and to de-certify non-memory-safe systems.
2038-2047 (longer term)	Industry best practices	**Near universal adoption of deterministic memory safety** in newly deployed systems is achieved in this decade. Small pools of remaining non-memory-safety in long-lived products such as deeply embedded, non-network-connected devices; long lived legacy software stacks that must run only in highly protected environments that impose limitations on their casual use. Successful completion of the long-term project to ground memory safety in formally verified designs and implementations increase confidence in strong memory-safety technologies, and in particular that their TCBs are vulnerability-free.
	Security-critical applications	**Elimination of non-memory safety** outside of very small pools of long-lived, fielded devices, but with significant effort made to totally eliminate them as well. No new security-critical systems without memory safety are created during this decade.

Conclusion

We believe that contemporary language-based, hardware-based, formal, and compartmentalized techniques for achieving memory safety are now of sufficient maturity to allow a path to be planned towards *universal memory safety*, the adoption of strong memory-safety techniques throughout all forms of computer systems. The timeline for such an adoption path is long – likely multiple decades – requiring the deployment of a combination of new hardware, software, and formal techniques serving different adoption paths and catering to differing tolerances for disruption. However, to achieve these goals, industry requires a clear definition of memory safety, accompanied by improvements in engineering practice.

Memory-safety standardization will therefore play an essential role in allowing requirements to be framed in design and procurement, engineering of systems to be tailored to those requirements, and suitable implementation to be auditable. Today, attempts to request memory safety in acquisition, regulation, or liability contexts would be hampered by a lack of a clear set of definitions and practice. Filling this gap requires building industrial consensus on technical approaches, but also a collaborative effort with government and academia to bring such effort to fruition. Despite the need for research to further improve aspects of these technologies, and especially to understand their composition, it is urgent that an effort to appropriately define memory safety begin as quickly as possible based on current technologies and understandings, to feed not just into research, but also improvements in training and delivery.

Acknowledgements

We gratefully acknowledge Graeme Barnes (Arm), Ron Black (Codasip), Mike Eftimakis (Codasip), Andy Frame (VyperCore), John Goodacre (UKRI), Richard Grisenthwaite (Arm), Alice Hutchings (University of Cambridge), William Martin (NSA), Ed Nutting (VyperCore), Anjana Rajan (ONCD), Jonathan Ring (ONCD), Carl Shaw (Codasip), Howie Shrobe (DARPA), Domagoj Stolfa (University of Cambridge), Dan Wallach (DARPA), and Paul Waller (NCSC) for their thoughtful comments and detailed conversations with us about these ideas. Finally, we acknowledge, and remember – Professor Ross Anderson (University of Cambridge) – who developed key ideas in the application of economic principles to computer security, and whose advice helped inspire this work.

Distribution Statement A: Approved for public release. Distribution is unlimited. This material is based in part upon work supported by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under contract FA8750-24-C-B047 (“DEC”), and in part 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 Department of Defense, Under Secretary of Defense for Research and Engineering, or the U.S. Government.

This work was supported in part by Innovate UK projects 105694 (“DSbD”) and 10027440, by EPSRC grants EP/V000292/1 (“CHaOS”) and EP/V000373/1 (“CapableVMs”), by UKRI (ERC-AdG-2022 funding guarantee) grant EP/Y035976/1 “SAFER”, and by ERC-AdG-2017 grant 789108 “ELVER”. Additional support was received from Arm, Google, and Microsoft.