Memory Safety:
A CALL FOR STRATEGIC ADOPTION

BSA | The Software Alliance has always prioritized improving software security. That is why BSA developed the BSA Framework for Secure Software and the first priority in the BSA 2023 Global Cyber Agenda is software security. Adopting memory-safe languages presents an opportunity to improve software security, but only if done thoughtfully, which is why BSA supports a policy of strategic adoption.

Background & Context

Programming Languages

Software that businesses and government agencies rely on to deliver services to their customers and citizens is written in programming languages. Software developers use different languages for different applications and programs. For example, software developers often use Python for artificial intelligence and machine learning; JavaScript for web development; Java or C# for enterprise applications; R for data analysis; and C, C++, or Rust for systems programming.

Decisions about which programming language to use are complicated and require the consideration of numerous variables, including performance, scalability, maintainability, developer availability, compatibility, and cost. Moreover, many software projects are built upon existing code bases that may be years or decades old. These projects often must adopt programming languages used in the existing code base. Further, many of these code bases draw upon third-party (often open-source) libraries, in which programming language decisions are made by those managing the software projects.

Memory Safety

When DARPA created the Internet, it could not know and address all potential security challenges. Similarly, when software developers created programming languages, they could not know and address all future security challenges, let alone appreciate that nearly every person and billions of devices would connect to the Internet. The result is that some languages, unfortunately, are not "memory safe"—that is, malicious actors have developed ways to access the memory of software developed using these languages.

Software developers address the risks associated with programming languages that are not memory safe in multiple ways, including using secure development practices generally, applying static and dynamic security analysis and testing tools, and enabling compiler features. Although no security control can guarantee absolute security, these efforts can significantly improve the security of software that uses languages that are not memory safe.

Strategic adoption will require action by the entire software ecosystem, including the open source software community, software producers, and software customers.

Acknowledging Our Current Situation

Today, if a software developer were starting with a blank screen, the software developer would strongly consider using a memory-safe language. But that is not the world we live in. We are not starting from a blank screen. We use billions, if not trillions, of lines of code written in languages that are not memory safe. Additionally, many of these lines of code are in some of the most foundational pieces of software, like the Linux kernel that serves as a bridge between a computer's hardware and the applications and programs it runs, further complicating the process of converting programming languages.

We must develop policies that acknowledge this situation, while charting a path toward a more secure future. Regarding memory-safe languages, the policy most likely to result in that future is a policy of strategic adoption.

Challenges to Adopting Memory-Safe Languages

If adopting memory-safe languages is an opportunity to improve security, why not simply require all software producers and government agencies to convert code? Simply put, broad conversion requirements may be impractical and will likely be suboptimal. Despite the long-term benefits of converting to memory-safe languages, those benefits may be outweighed if not done strategically.

Further, many software producers that use secure software development practices have already scanned and mitigated risks associated with memory safety.

Converting Software Creates Risks

The process of converting software to a memory-safe language might, inadvertently, introduce new bugs. As the US National Cybersecurity Strategy states, "even the most advanced software security programs cannot prevent all vulnerabilities." Consequently, policymakers should expect that converting trillions of lines of code to memory-safe languages will reduce vulnerabilities associated with memory safety but create risks associated with other vulnerabilities in the new code.

Another challenge when converting software is that broadly used languages benefit from a wide and deep set of analysis tools that memory-safe languages may not have today. The consequence is that code written in memory-safe languages may be memory safe, but software developers may face other challenges related to analyzing and securing the new code.

Other Activities May Offer a Better Return on Investment

Products and services that have not yet implemented other cybersecurity best practices would likely benefit more from adopting those best practices than converting to a memory-safe language. For example, implementing multifactor authentication or encrypting data at rest and in transit would likely be better investments than converting to a memory-safe language. Another consideration is compensating controls. For example, using C++ compiler memory safety protections (even if only in the short run) may be a better investment in cybersecurity than converting to a memory-safe language. Similarly, a threat model may demonstrate that different uses, for example a mobile application or a cloud service, face different threats.

Ultimately, a software producer should make a risk-based decision about where to invest cybersecurity resources, whether it be implementing best practices, using compensating controls, or adopting memory-safe languages.

Successful Adoption Requires the Entire Software Ecosystem

Strategic adoption will require action by the entire software ecosystem, including the open source software community, software producers, and software customers. Open source projects that use languages that are not memory safe will need to actively manage risks to

memory safety. And if a software producer adopts a memory-safe language for an application, a customer may need to update its version of the application. This challenge may seem minimal, but experience tells us that customers are often slow to update software—and sometimes for good reasons (e.g., operational constraints to a customer's systems dictate that the customer can only update the software periodically).

Resources Are Finite

Every organization, including government agencies, should understand that resources to improve cybersecurity are finite. Resources an organization uses to adopt memory-safe language are then not available to address known exploitable vulnerabilities in an application, implement multi-factor authentication, or invent the next security technology needed to protect against evolving threats.

This challenge might be even more pervasive when considering human resources. As the Office of the National Cyber Director stated in its Requests Insight and Expertise on Cyber Workforce, Training, and Education, we continue "to face a significant shortfall in cyber talent." Today, there are simply not enough software developers, let alone software developers trained in secure software development practices or programming languages appropriate to replace languages that are not memory safe. Policymakers should expect that software developers switching languages will, at least for a time, be less productive and more likely to write suboptimal (i.e., less secure, reliable, and performant) code.

Strategic Adoption

Given the challenge of converting trillions of lines of code, policymakers should adopt a policy of strategic adoption, or a policy that:

Requires Active Risk Management

It is simply not possible to convert all software immediately or simultaneously. This fact, in conjunction with the challenges identified above, highlight the need for software producers, as well as government agencies that develop software, to make risk-based decisions about how to prioritize adopting memory-safe languages. Software producers and government agencies should consider the benefits of adopting memory-safe languages, with the goal of using memory-safe languages or otherwise addressing vulnerabilities associated with memory safety.

Policymakers should require software producers and government agencies to take a risk-based approach to implementing compensating controls like using secure development practices generally, applying static and dynamic memory-safety tools, and enabling compiler features, as well as converting software to memory-safe languages lest the conversion create new and worse cybersecurity vulnerabilities.

Sets a Bold but Achievable Path to a More Secure Future

Policymakers should set the bold vision of ultimately using only memory-safe languages for new software programs. Because the digital ecosystem currently uses trillions of lines of code that software producers have not yet converted to memory-safe languages, software producers and government agencies will need to continue using those languages both to update security of old code and for interoperability with new code. But policymakers can set a bold vision of a software ecosystem that responsibly moves away from languages that are not memory safe.

Prioritizes New Code

Decisions about how to manage the risks associated with languages that are not memory safe should be risk based. Prioritizing writing new programs in memory-safe languages over transitioning existing programs into memory-safe languages is likely to produce better security for the same investment. Prioritizing newly written code will allow software producers to make risk-based decisions about applying compensating controls to software written in languages that are not memory safe, while training more software developers, and building more tools to improve coding in memory-safe languages.

Decisions about how to manage the risks associated with languages that are not memory safe should be risk based.

Invests in Research and Development

Research projects aimed at automating the transition from one programming language to another show promise. If these programs can effectively transition from one programming language to another without introducing either security or functionality issues, the entire digital ecosystem would benefit. Governments around the world should both increase investment and direct existing investment toward projects aimed at accomplishing this important goal.

Provides Training and Support

Today, many software developers have neither trained in nor have gained experience with memory-safe languages, which will make coding slower and errors more likely. Government support of private sector and academic training can provide the workforce necessary to manage and execute the policy of strategic adoption. For example, governments should consider requiring and incentivizing educational institutions that train software developers to ensure that they train their students in memory-safe languages.

Deploys Incentives

Many software producers are already managing the risk associated with using languages that are not memory safe. But policymakers can and should incentivize companies and government agencies to implement the policy of strategic adoption. Policymakers should also consider how they will incentivize customers to update software after a software producer has adopted a memory-safe language or otherwise managed the risk associated with using a language that is not memory safe. They should also consider how they might recognize software producers that are successfully implementing the policy of strategic adoption, including through procurement preferences.

Government support of private sector and academic training can provide the workforce necessary to manage and execute the policy of strategic adoption.

Leads by Example

Government agencies should lead by example by, for example, taking a risk-based approach and prioritizing new code that they or their contractors write. Some programs written by agencies and their contractors may be low-risk, whereas others, like software used in critical infrastructure systems like the healthcare and public health or water and wastewater sectors, may be high-risk and require prioritization. Lawmakers should require government agencies to lead, and should implement the same policy of strategic adoption that private sector software producers will undertake.

Conclusion

None of the challenges identified above justify unreasonable delay in government agencies and private sector software producers adopting memory-safe languages. However, imposing fixed timelines that do not support strategic adoption risks harming the digital ecosystem such a policy would hope to help. Rather, policymakers should support a policy of strategic adoption that requires active risk management; sets a bold but achievable path to a more secure future; prioritizes new code; invests in research and development; provides training and support; deploys incentives; and directs government agencies to lead by example.