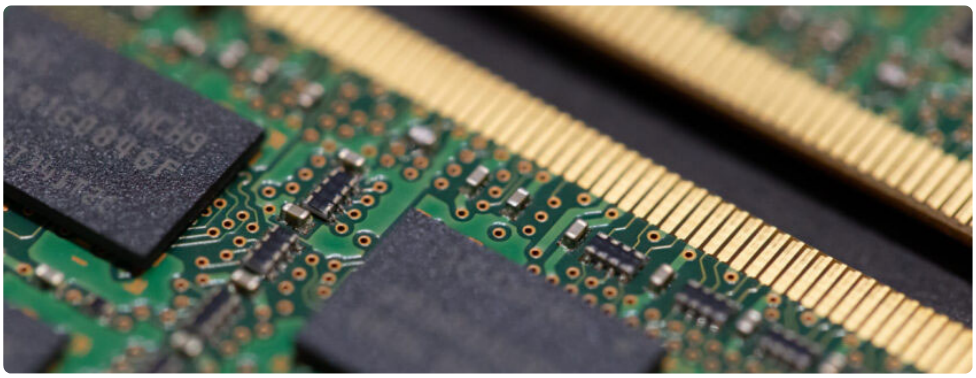
SocietyByte

BFH-Magazin für die Humane Digitale Transformation

Memory-Safety: Chancen und Herausforderungen für die Software-Sicherheit der Zukunft, Teil 1

Von Yann Herren (BFH Technik und Informatik) , Pascal Mainini (BFH Technik und Informatik) | 0 Kommentare



Software ist aus unserem Alltag nicht mehr wegzudenken: ob im neuesten Elektroauto, beim E-Banking oder in lebensrettenden Systemen. Eine der am weitesten verbreiteten Bedrohungen für die Sicherheit solcher Systeme ist die fehlerhafte Speicherverwaltung. Man spricht bei Schwachstellen dieser Art von sogenannten «Memory-Safety-Schwachstellen». Laufend werden neue Schwachstellen in diesem Bereich entdeckt. Die zugrundeliegende Problematik zu beheben, stellt jedoch für Entwickler*innen und Unternehmen eine grosse Herausforderung dar.

Was ist Memory-Safety?

Memory-Safety beschreibt, wie sicher und korrekt eine Software mit der Verwaltung, Zuweisung und Freigabe von **Arbeitsspeicher (RAM)** umgeht. Sie ist ein zentraler Bestandteil der Softwaresicherheit und bleibt trotz jahrelanger Forschung eine der häufigsten Schwachstellen in moderner Software [1] [#_ftn1] .

Arbeitsspeicher wird in einem Computer verwendet, um **Daten** wie Benutzereingaben oder Resultate von Berechnungen zu speichern. Beim Start eines Programms wird neben den Daten (den Informationen, die verarbeitet werden) auch der **Programmcode** (die Anweisungen, die der Computer ausführt) in den Arbeitsspeicher geladen. Wie Abbildung 1 zeigt, gibt es im Arbeitsspeicher keine strikte Trennung zwischen Programmcode und Daten. [2] [#_ftn2]

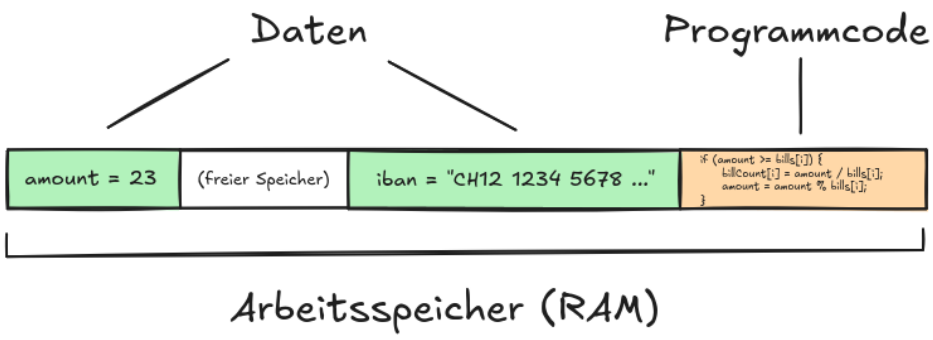


Abbildung 1: Daten und Programmcode im Arbeitsspeicher

Weit verbreitete Programmiersprachen wie C oder C++ ermöglichen den Entwickler*innen die manuelle Verwaltung des verwendeten Arbeitsspeichers. Diese gilt allgemein als äusserst fehleranfällig: Programme können dabei unbeabsichtigt auf unzulässige Speicherbereiche zugreifen, was wiederum von Angreifer*innen ausgenutzt werden kann, um beispielsweise Schadcode auszuführen oder vertrauliche Daten auslesen. Das häufige Auftreten von Memory-Safety-Schwachstellen hat mitunter damit zu tun, dass die manuelle Speicherverwaltung in diesen Programmiersprachen nach wie vor intensiv genutzt wird. [2] [#_ftn2]

Doch wie äussern sich Memory-Safety-Schwachstellen in der Praxis? Ein Blick auf reale Beispiele zeigt die Auswirkungen dieses Problems.

Memory-Safety-Schwachstellen in der Praxis

In einem kürzlich veröffentlichten Report entdeckte *Kaspersky Security Services* [3] [#_ftn3] mehrere Schwachstellen im neuesten Infotainment-System von Mercedes-Benz. Darunter befanden sich auch Memory-Safety-Schwachstellen, Schwächen im Anti-Diebstahl-System sowie die Möglichkeit zur Manipulation der **Firmware** (spezielle Software, die in die Hardware eingebettet ist und grundlegende Funktionen ermöglicht). Dieses Beispiel verdeutlicht, dass Memory-Safety-Probleme selbst in alltäglich genutzten Systemen auftreten. Für Nutzer*innen kann dies spürbare Folgen haben, etwa durch einen Ausfall wichtiger Funktionen oder den Verlust persönlicher Daten. Weiter sind laut Berichten von Microsoft [4] [#_ftn4] und Google [5] [#_ftn5] anhaltend ungefähr 70% der gefundenen Sicherheitsprobleme in ihrer eigenen Software auf Memory-Safety-Schwachstellen zurückzuführen.

Im Rahmen einer Semesterarbeit im Bachelor Informatik an der BFH/TI wurde eine Auswertung von Daten der **National Vulnerability Database (NVD)** [6] [#_ftn6] vorgenommen. Die NVD ist eine öffentlich zugängliche Datenbank über bekannte Sicherheitslücken in Software und Hardware. Dabei wurden der Anteil an Sicherheitslücken der Kategorie «Memory-Safety» [7] [#_ftn7] der gesamten Anzahl gegenübergestellt. Die erstellte Statistik in Abbildung 2 bietet ein umfassenderes Bild von Memory-Safety-Schwachstellen in der gesamten IT-Landschaft. Der Anteil variiert in den letzten 10 Jahren zwischen 15 und 20%. Dies ist ein beachtlicher Anteil und die Dunkelziffer dürfte noch deutlich höher liegen.

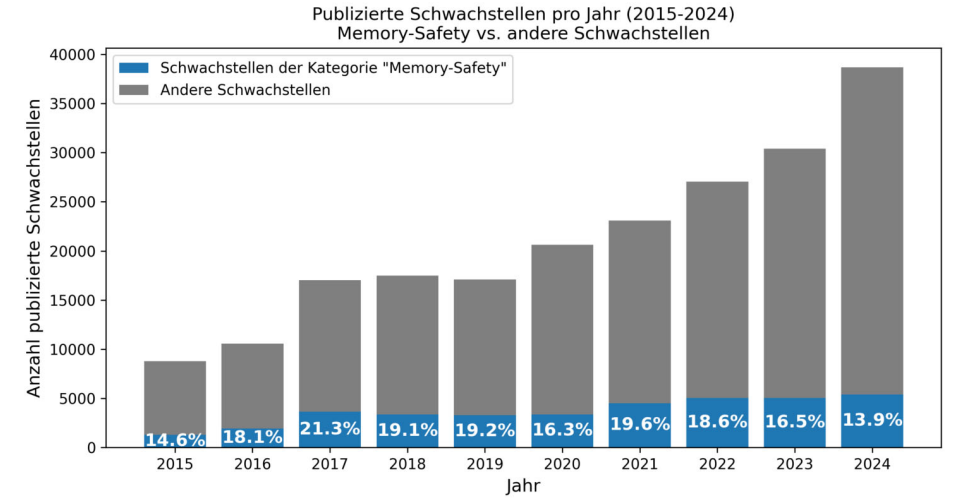


Abbildung 2: Auswertung der NVD über Memory-Safety-Schwachstellen

Doch welche Möglichkeiten gibt es, dieses Problem zu lösen – oder zumindest einzuschränken?

Ansätze zur Verbesserung der Memory-Safety

Um Memory-Safety-Schwachstellen zu vermeiden, wird empfohlen, dass Programmcode möglichst in **Memory-Safe-Sprachen** geschrieben wird. Dass das Thema als wichtig wahrgenommen wird, zeigt z.B. auch eine Ankündigung vom weissen Haus. [8] [#_ftn8] Sprachen wie z.B. Python, Java und C# verwalten den Speicher automatisch mithilfe eines **Garbage Collectors** (ein Mechanismus, der ungenutzten Speicher im Hintergrund aufräumt und verwaltet). Dies benötigt jedoch zusätzliche Ressourcen, was ihren Einsatz für auf Effizienz angewiesene und hardwarenahe Systeme wie z.B. **Embedded Systems** (spezielle Computersysteme, die in grössere technische Produkte integriert sind, wie Autos oder Haushaltsgeräte) ungeeignet macht. In solchen Fällen ist die direkte Kontrolle über den Speicher, wie sie die Sprachen C/C++ ermöglichen, unentbehrlich.

Hier kommt die Programmiersprache **Rust** ins Spiel. Sie bietet einen interessanten Ansatz, Memory-Safety-Probleme zu verhindern.

Rust to the rescue?

Die im Jahr 2015 veröffentlichte Programmiersprache Rust [9] [#_ftn9] findet in vielen Anwendungsgebieten Einsatz. Sie ermöglicht hardwarenahe Programmierung, *garantiert* Memory-Safety zur Entwicklungszeit und funktioniert ohne zusätzliche Mechanismen wie ein Garbage Collector. Sie bietet moderne Sprachkonzepte und definiert bestimmte Regeln, mit denen Memory-Safety-Schwachstellen gar nicht erst auftreten können.

Rust gewinnt als Programmiersprache in verschiedenen Bereichen der IT zunehmend an Bedeutung und könnte Memory-Safety-Schwachstellen weiter einschränken. Der Hardware-Security-Experte Pascal Mainini erläutert im Interview, wie Rust künftig die Memory-Safety verbessern könnte und in welchen Bereichen die Sprache aktuell besonders relevant ist:

> [Herren]: «In welchen Branchen oder Bereichen wird Rust an Bedeutung gewinnen?»
>
> [Mainini]: «Rust ist als Sprache grundsätzlich universell einsetzbar, aber es gibt Bereiche, in denen die Adoption schon weiter fortgeschritten ist. Einerseits natürlich HW-nahe Programmierung, aber auch z.B. der Cloud-Bereich. Da Rust eine sehr effiziente Sprache ist, trägt sie auch zur Energieeffizienz bei, wenn Applikationen auf tausenden von Servern laufen. Last but not least ist Rust auch die einzige Sprache, die mittlerweile neben C zur Entwicklung von Treibern für den Linux-Kernel genutzt werden kann.»

Hier das ganze Interview zu Rust und dessen Entwicklung lesen. [https://www.societybyte.swiss/2025/12/04/memory-safety-ein-interview-ueber-rust-und-die-zukunft-sicherer-softwareentwicklung-teil-2/]

Bilanz und Ausblick

Memory-Safety-Schwachstellen können Systeme jeglicher Art betreffen. Über die gesamte IT-Landschaft hinweg schwankt der Anteil der bekannten Schwachstellen dieser Kategorie zwischen 15 und 20%, deutlich weniger als die häufig zitierten 70% von Microsoft und Google. Dennoch handelt es sich um ein ernstzunehmendes Problem. Ein vollständiger Schutz vor Memory-Safety-Schwachstellen erfordert für Unternehmen und Entwickler*innen viel Aufwand und Ressourcen. Eine kurzfristige Lösung ist anspruchsvoll, da bestehender Code entweder in eine Memory-Safe-Sprache portiert oder umfassend abgesichert werden müsste. Zudem ist die Erkennung solcher Schwachstellen oft schwierig. Verschiedene Ansätze, wie neue Detektionsmethoden, etwa durch KI, oder der Einsatz von Sprachen wie Rust, könnten das Risiko in Zukunft verringern. Eines steht jedoch fest: Programmcode sollte zukünftig möglichst in Memory-Safe-Sprachen geschrieben werden. Des Weiteren muss das Bewusstsein bei Unternehmen und Entwickler*innen für dieses Problem deutlich geschärft werden.

Referenzen

[1] [#_ftnref1] H. Okhravi, “Memory Safety,” IEEE Security & Privacy, Bd. 22, Nr. 4, pp. 13–15, Jul. 2024 [Online]. Zugriff: doi:10.1109/MSEC.2024.3409849

[2] [#_ftnref2] C. Rohlf (2023, Sep. 26). Memory Safety: An Explainer [Webseite]. Zugriff: https://cset.georgetown.edu/article/memory-safety-an-explainer/, 18. Okt. 2025

[3] [#_ftnref3] Kaspersky Security Services (2025, Jan. 17). Mercedes-Benz Head Unit security research report [Website]. Zugriff: https://securelist.com/mercedes-benz-head-unit-security-research/115218/, 19. Okt. 2025

[4] [#_ftnref4] Microsoft Security Response Center (2019, Jul. 16). A proactive approach to more secure code [Webseite]. Zugriff: https://msrc.microsoft.com/blog/2019/07/a-proactive-approach-to-more-secure-code/, 18. Okt. 2025

[5] [#_ftnref5] The Chromium Projects (o.D.). Memory safety [Webseite]. Zugriff: https://www.chromium.org/Home/chromium-security/memory-safety/, 18. Okt. 2025

[6] [#_ftnref6] B. Harold (2015). National Institute of Standards and Technology: National Vulnerability Database [Webseite]. Zugriff: https://nvd.nist.gov/, 18. Okt. 2025

[7] [#_ftnref7] The Mitre Corporation (o.D.) Comprehensive Categorization: Memory Safety [Webseite]. Zugriff: https://cwe.mitre.org/data/definitions/1399.html#Vulnerability_Mapping_Notes_1399, 18. Okt. 2025

[8] [#_ftnref8] The White House: PRESS RELEASE: Future Software Should Be Memory Safe [Archiviert]. Zugriff: https://web.archive.org/web/20240226175250/https://www.whitehouse.gov/oncd/briefing-room/2024/02/26/press-release-technical-report/, 31. Okt. 2025

[9] [#_ftnref9] Rust Team (o.D.) Rust: A language empowering everyone to build reliable and efficient software [Webseite]. Zugriff: https://rust-lang.org/, 19. Okt. 2025

AUTHOR: YANN HERREN

Yann Herren studiert Informatik im Bachelor an der Berner Fachhochschule. Er interessiert sich für die Entwicklung moderner und sicherer Software-Systeme sowie Anwendungen im Bereich von Embedded Systems. Im Rahmen einer Semesterarbeit beschäftigte er sich ausführlich mit der Thematik von Memory-Safety und Memory-Safe-Sprachen wie Rust.

Posts from Yann Herren | Website

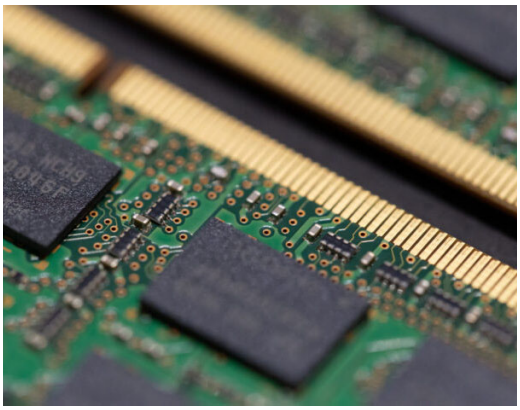
AUTHOR: PASCAL MAININI

Pascal Mainini ist Tenure Track Dozent am Institute for Cybersecurity and Engineering ICE der Berner Fachhochschule. Als Experte für angewandte Kryptographie, sichere Soft- und Hardware sowie Datenschutz und Privatsphäre setzt er sich dafür ein, die Integrität, Vertraulichkeit und Sicherheit moderner digitaler Systeme zu gewährleisten.

Posts from Pascal Mainini | Website

Create PDF

Ähnliche Beiträge



Memory Safety: Ein Interview über Rust und die Zukunft sicherer Softwareentwicklung, Teil 2

0

COMMENTS