Increase the cybersecurity of SCADA and IIoT devices with secure memory management

Andrii Nyzhnyk[1,*,†], Andrii Partyka[1,†] and Michal Podpora[2,†]

[1] Lviv Polytechnic National University, 12 Stepan Bandera str., 79013 Lviv, Ukraine
[2] Opole University of Technology, 76 Proszkowska str., 45-758 Opole, Poland

Abstract

Secure memory management issues are very common in SCADA device software. Systems that are integrated with SCADA and IIoT are often included in the list of critical infrastructures in many countries. Therefore, ensuring the security of these devices is important for national security. Despite the implementation of certain cybersecurity tools and measures, these devices often become targets for attacks. Memory errors remain one of the most common sources of software vulnerabilities. Attackers are actively using them to gain unauthorized access to systems, steal data, disrupt software operations, and perform other criminal acts. These types of vulnerabilities are very difficult to reproduce and fix. This paper discusses how to increase the security of SCADA and IIoT devices using secure memory management. The relevance of the problem of secure memory management in SCADA and IIoT devices makes it a subject of careful study and search for effective solutions. The purpose of this paper is to study the impact of dynamic memory errors on cybersecurity and provide practical recommendations for their elimination. Various sectors, including energy, water treatment, manufacturing, transportation, oil and gas exploration, telecommunications, environmental monitoring, aerospace, and medical facilities, rely heavily on SCADA and IIoT systems. Given the widespread use of these systems in critical infrastructure, addressing memory management vulnerabilities is crucial. This paper presents an overview of the most common memory management issues, such as null pointer dereferencing, use-after-free, and buffer overflow, and highlights notable cyberattacks that exploited these vulnerabilities. The effectiveness of different methods to prevent and mitigate memory management issues, including the use of sanitizers, static code analysis, and programming languages with secure memory management like Rust, is analyzed. The study concludes that a comprehensive approach combining these methods is essential for enhancing the cybersecurity of SCADA and IIoT devices. The findings aim to help software developers and cybersecurity professionals better understand the risks associated with dynamic memory in SCADA and IIoT devices and improve application security.

Keywords

SCADA, IIoT, memory management issues, Rust, buffer overflow, use-after-free, cybersecurity, secure programming, sanitizers, static code analysis

1. Introduction

Imagining modern life without process automation is challenging, as it is crucial for both industry and homes. Most devices can now connect to the Internet, a small but significant percentage of devices can be controlled by voice and even contain artificial intelligence to communicate with humans. The SCADA (Supervisory Control and Data Acquisition) system is a centralized control system that allows monitoring and control of industrial processes. In essence, SCADA has dual functionality—supervision of operations and data acquisition from remote locations, which is critical for efficient and safe operation in various industries.

SCADA systems, alongside the Industrial Internet of Things (IIoT), play a vital role in numerous sectors, including energy, water treatment, manufacturing, transportation, oil and gas exploration, telecommunications, environmental monitoring, aerospace, and medical facilities. The market for industrial control systems, including SCADA, is expected to exceed $181.6 billion by the end of 2024, highlighting their growing importance.

Despite the essential role of SCADA and IIoT systems, they face significant cybersecurity threats, primarily due to memory management issues. According to Google, 59% of the vulnerabilities found in the Android project in 2021 were related to memory issues [1]. Microsoft reported that in 2019, 70% of all vulnerabilities in their projects were memory-related [2]. Similarly, in the Chromium project, almost 70% of critical security bugs are associated with memory security issues, and Mozilla has reported that incorrect memory management can cause up to 73.9% of vulnerabilities [3]. These vulnerabilities often lead to severe

CSDP-2024: Cyber Security and Data Protection, June 30, 2024, Lviv, Ukrain
*Corresponding author.
†These authors contributed equally.
andrii.o.nyzhnyk@lpnu.ua (A. Nyzhnyk); andrii.i.partyka@lpnu.ua (A. Partyka); m.podpora@po.opole.pl (M. Podpora)
0009-0003-9094-0740 (A. Nyzhnyk); 0000-0003-3037-8373 (A. Partyka); 0000-0002-1080-6767 (M. Podpora)

security breaches, such as unauthorized access to systems, data theft, and operational disruptions.

In their research, Oorschot et al. (2023) highlighted the challenges of memory safety in system programming languages like C and C++ [4]. While these languages are powerful, they are prone to memory errors such as null pointer dereferencing, use-after-free, and buffer overflow. These types of errors are particularly prevalent in the software used for SCADA and IIoT devices, which are often written in these languages due to their low-level capabilities and performance requirements.

Also, in many research studies (Altaleb, Haya & Rajnai, Zoltan (2024) or Fall, Moustapha & Chuvalas, Chris & Warning, Nolan & Rabiee, Max & Purdy, Carla (2020) and many more) related to security SCADA and IIoT, secure memory management is not covered [5]. These studies focus on the more common cyber security threats, like "OWASP Top Ten" and others. In this context, secure memory management belongs to low-level system control that significantly impacts cybersecurity [6].

The objective of this paper is to address the impact of dynamic memory errors on the cybersecurity of SCADA and IIoT devices and to provide practical recommendations for their mitigation. By examining various studies and existing methods, we aim to identify effective solutions to enhance the security of these critical systems. Our analysis includes the use of sanitizers, static code analysis, and the adoption of programming languages with built-in memory safety mechanisms, such as Rust, which offers a robust alternative to traditional system programming languages. The goal is to provide insights that will help software developers and cybersecurity professionals better understand and manage the risks associated with memory management in SCADA and IIoT devices.

2. Ensuring cybersecurity of SCADA and IIoT devices

To better understand how SCADA and IIoT devices can be protected, it is necessary to analyze where and how these devices are used. The adaptability of SCADA and IIoT devices to scenarios that require remote monitoring, control, and data collection has led to its widespread adoption in various sectors such as:

- Energy sector. SCADA is widely used in the energy sector to monitor and control the production and distribution of electricity, ensuring efficient and reliable power supply.
- Manufacturing industry. SCADA is used in manufacturing to monitor and control production processes, optimize efficiency, minimize downtime, and improve overall production quality.
- Transportation and traffic management. SCADA systems are used in transportation to monitor and control traffic lights in real time, helping to manage traffic, reduce congestion, and improve road safety.
- Telecommunications. SCADA systems contribute to the development of the telecommunications sector by monitoring and managing network infrastructure, ensuring reliable and uninterrupted communication services.
- Environmental monitoring. SCADA is used in environmental monitoring to track pollution levels, air quality, and other environmental parameters, supporting efforts to solve environmental problems.
- Aerospace industry. SCADA is integrated into the aerospace industry to monitor and control production processes, ensuring the accuracy and quality of aircraft component production.
- Medical facilities. SCADA systems are used in healthcare facilities to monitor and control critical infrastructure, including power distribution, heating, ventilation, air conditioning systems, and medical equipment, ensuring the uninterrupted provision of healthcare services.

To summarize, these systems are widely used in various areas of critical infrastructure. The main task of SCADA and IIoT is to control the system, which helps to manage and achieve the set goals with optimal use of resources.

2.1. The most common memory management issues

Despite its long history and significant economic consequences, the problem of secure memory management remains relevant. According to unofficial estimates, as recently as 2004, memory-related errors cost the industry about $250,000, and this amount is only growing over time.

Memory management vulnerabilities can occur even in well-known projects with millions of users and professional development teams [5]. Different companies conducted the research:

- In 2021, Google reported that 59% of the vulnerabilities found in the Android project were related to memory issues. Different types of vulnerabilities found in the Android project are shown in Fig. 1.
- In 2019, Microsoft researched its projects and found that 70% of all vulnerabilities in program code are related to memory.
- Chromium (the basis of the Google Chrome browser): almost 70% of critical security bugs are related to memory security issues.
- Mozilla: the level of vulnerabilities caused by incorrect memory management can reach 73.9%.

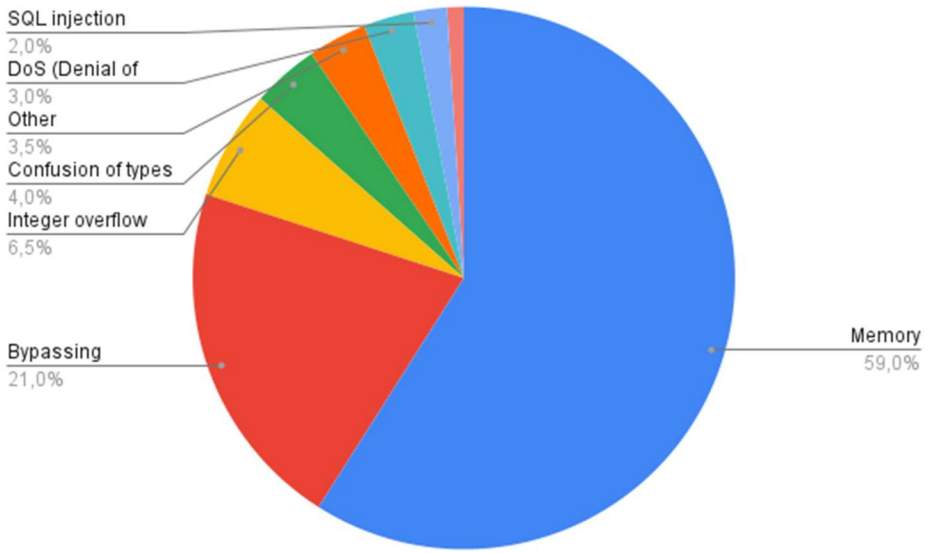

Figure 1: Types of critical and high vulnerabilities in the Android project

According to research by various IT companies, the total percentage of memory-related vulnerabilities is shown in Fig. 2.

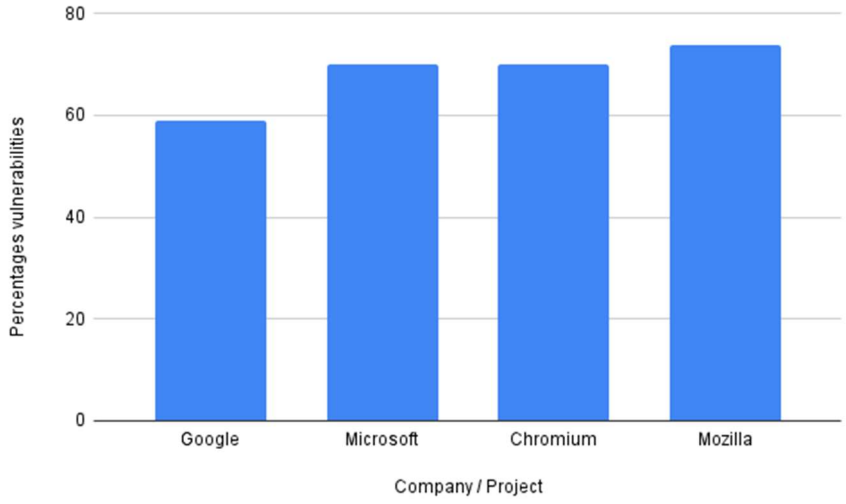

Figure 2: Percentage of vulnerabilities in projects related to memory management

Memory management bugs are a common source of misbehavior in many programming languages [7], but they can be especially prevalent in system programming languages such as C and C++ [8]. C and C++ are the programming languages most commonly used to write SCADA and IIoT software and are the languages in which the largest number of vulnerabilities are found.

The most common problems are related to memory management:

1. **Null pointer dereferencing** is a problem that occurs when a program tries to access memory that has not been allocated or has already been freed.
2. **Use-after-free use** is a problem that occurs when a program tries to access memory that has already been freed.
3. **Buffer overflow** is also a common problem when a program writes data outside the buffer, potentially overwriting other data or executing arbitrary code.

While memory management can cause a vulnerability in an application, other security issues such as misconfiguration of role-based access control, SQL injection, and other well-known vulnerabilities should not be overlooked [9]. Despite this, memory management issues remain the most common in SCADA and IIoT devices.

Securing low-level devices such as SCADA and IIoT differs significantly from traditional approaches to securing cloud infrastructure. [10]. Programs that control SCADA and IIoT mostly run without an operating system or any antivirus [11].

2.2. Cyberattacks that were carried out with the help of memory vulnerabilities

Attacks that use hanging pointers:

- Heartbleed (2014). This vulnerability, which exploited a hanging pointer, was discovered in OpenSSL, a cryptographic library used by millions of web servers. Attackers could have exploited this vulnerability to steal sensitive information, including passwords, encryption keys, and credit card information. The damage from this attack is estimated at billions of dollars [12].
- CVE-2021-45046 (2021). This vulnerability was discovered in the Windows Print Spooler driver. Attackers could exploit this vulnerability to gain full control over vulnerable systems [13].

Attacks that were carried out through uninitialized variables:

- Buffer overflow (2001). This attack led to the theft of 170 million credit card numbers from TJX Companies' systems.
- Stack overflow (2019). This attack, which exploits a stack overflow, led to the outage of Cloudflare services. Cloudflare is a large American company that provides network services for content delivery, protection against DDoS attacks, and other network services.

In addition, there have been thousands of other cyberattacks using all types of memory management vulnerabilities. For example, the WannaCry ransomware virus, which in 2017 infected more than 200,000 computers in 150 countries. WannaCry exploited a "double free" vulnerability in Windows.

These examples highlight cyberattacks related to memory management issues. Such attacks can result in significant consequences, including data theft, service outages, and financial and reputational losses. Implementing secure programming practices and thorough software testing can help prevent these incidents.

2.3. Methods to ensure cybersecurity in Socio-Cyber-Physical Systems (SCPS)

it is crucial to consider the broader context of cybersecurity within Socio-Cyber-Physical Systems (SCPS). According to Yevseiev et al. (2023), integrating cybersecurity into SCPS involves developing comprehensive models that account for the complex interactions between social, cyber, and physical components [14]. These models help in understanding vulnerabilities and developing strategies to mitigate risks in critical infrastructures like SCADA and IIoT systems.

One effective approach is the use of mathematical models and simulations to analyze potential security threats and their impacts. This method allows for the identification and mitigation of vulnerabilities before they can be exploited by attackers. Simulations can model various attack scenarios, enabling researchers and engineers to develop robust defense mechanisms and response strategies.

Enhancing SCADA security with advanced memory management techniques is another method, as discussed by Kim and Lee (2024), who emphasized the importance of adopting modern memory safety mechanisms in SCADA systems [15]. The implementation of such techniques can significantly reduce the risk of memory-related vulnerabilities, which are often targeted by cyber attackers.

Additional Approaches:

1. **Behavioural Analysis and Anomaly Detection:**
   a. Implementing behavioral analysis and anomaly detection tools can help identify unusual activities that may indicate a security breach. These tools analyze the normal behavior of SCADA and IIoT systems and alert administrators to deviations that could signify an attack.
   b. Machine learning algorithms can be employed to improve the accuracy of anomaly detection, learning from historical data to distinguish between legitimate and malicious activities.
2. **Collaborative Security Frameworks:**
   a. Developing collaborative security frameworks that involve multiple stakeholders, including government agencies, private sector companies, and academic institutions, can enhance the overall cybersecurity posture of SCPS. Such frameworks facilitate the sharing of threat intelligence and best practices, fostering a collective defense approach.
   b. Public-private partnerships can play a vital role in advancing cybersecurity research and development, ensuring that SCADA and IIoT systems are equipped with the latest security innovations.
3. **Resilience Engineering:**
   a. Focusing on resilience engineering can help ensure that SCADA and IIoT systems continue to operate effectively even in the face of cyber-attacks. This involves designing systems with built-in redundancy, failover mechanisms, and robust recovery procedures.
   b. Regularly conducting resilience testing, such as cyber wargames and penetration testing, can help identify and address potential weaknesses in the system.
4. **Cybersecurity Education and Training:**
   a. Investing in cybersecurity education and training programs for employees at all levels can significantly improve the security of SCADA and IIoT systems. Ensuring that staff are aware of the latest threats and understand best practices for cybersecurity can reduce the risk of human error, which is often a critical factor in security breaches.

b. Certification programs and ongoing professional development can help maintain a high standard of cybersecurity expertise within organizations.

Ensuring the cybersecurity of socio-cyber-physical systems requires a multifaceted approach that addresses both technical and human factors. By integrating advanced memory management techniques, leveraging behavioral analysis and anomaly detection, fostering collaborative security frameworks, focusing on resilience engineering, and investing in cybersecurity education and training, organizations can significantly enhance the security and resilience of their SCADA and IIoT systems.

3. Methods to prevent and reduce the impact of memory management issues

The negative impact of memory management issues can range from minor crashes to data theft, system disruption, and other criminal activity. There is now a wide range of methods and mechanisms that can be used to prevent or reduce the impact of these problems.

3.1. The use of sanitizers

Sanitizers are essential tools for detecting memory management issues. They help to identify a potential problem at the development stage and thus warn the developer of a potential problem.

The use of sanitizers:

- Sanitizers can detect memory management errors at the early stages of development, allowing developers to fix them at the development and testing stages.
- Sanitizers can detect memory leaks when a program fails to free memory it no longer uses. This helps to reduce the overall memory usage of the program and prevent the program from suddenly terminating due to lack of memory.
- Using sanitizers can improve code quality by detecting and correcting memory management errors and using them efficiently.
- Sanitizers can be used as a learning tool for young developers to better understand how memory management works and how to avoid common mistakes in their professional careers.
- Using sanitizers is an important part of modern software development and can significantly reduce the impact of memory management issues.

A list of the most popular sanitizers and their capabilities:

- Address Sanitizer detects errors related to accessing invalid memory.
- Leak Sanitizer detects memory leaks.

Undefined Behavior Sanitizer detects undefined behavior that can lead to errors.

3.2. Static code analysis

Static analysis tools are software tools that examine code without executing it to identify potential memory management issues and other vulnerabilities. This method of analysis is used during program development, serving as a kind of independent code verification.

Advantages of static analysis:

- Efficiency: static analysis can examine code much faster than it can be tested.
- Proactivity: static analysis can help identify problems in the early stages of development when they are still easy to fix.
- Accuracy: static analysis can identify issues that may be missed during testing.

Examples of static analysis tools:

- Clang Static Analyzer: a free open-source tool that supports C, C++, Objective-C, and Swift.
- Coverity: a commercial tool that supports C, C++, Java, and JavaScript.
- Cppcheck: a free open-source tool that supports C++.
- PVS-Studio: a commercial tool that supports C, C++, and C#.
- SonarQube: an open-source platform that supports many programming languages.

Disadvantages of static analysis:

- **False positives:** Static analysis can sometimes generate error warnings that are not present.
- **Incomplete coverage:** static analysis cannot guarantee to detect all problems.
- **Complexity:** Some static analysis tools can be difficult to set up and use.

Static analysis can be used to detect issues such as memory leaks, buffer overflows, null pointers, unused variables, dead code, and unsafe coding patterns. Static analysis can be integrated into the development environment, which makes it even more convenient to use. Using static analysis tools is an important part of the process of developing secure software.

Beyond the methods above, it is also important to regularly update software, implement firewall and antivirus solutions, and ensure the creation of data backup copies.

3.3. Use languages with secure memory management

One approach is to use safe programming languages. Programming languages with a high level of abstraction and built-in memory safety mechanisms, such as Java, Python, Go, C#, and JS/TS, significantly reduce the risk of memory management issues. The disadvantage of this approach is the inability to rewrite existing programs within a short time and the decrease in program performance. For some systems, such a transition is simply not possible because it requires very low-level work with memory and registers [16].

However, with the advent of programming languages such as Rust, you can solve the problem of low-level access without losing program performance. Rust is a programming language that combines high performance with memory safety. It is becoming an increasingly popular choice for developing system software and other programs where it is critical to avoid memory issues [17].

Here are some of the key benefits of using Rust to prevent memory management issues:

- Ownership system. Rust uses an ownership system to keep track of who owns data in memory. This makes errors such as use-after-free and memory leaks impossible.
- Compile-time checking. Most memory management issues in Rust are detected at compile time, not runtime. This saves time and resources and makes the code more error-resistant.
- No garbage collection. Rust does not use garbage collection, which gives developers more control over memory. This can lead to more economical memory usage and better performance.

Nguyen and Pham (2023) highlighted that secure programming practices for embedded systems, with a focus on memory safety, can greatly benefit from languages like Rust, especially for SCADA and IIoT applications [18].

3.4. Security in cloud infrastructures

A comprehensive approach to developing and maintaining secure cloud infrastructures is essential for modern enterprises, including those utilizing SCADA and IIoT systems. Ensuring security in cloud environments involves multiple layers of protection, including secure configuration of services, continuous monitoring, and the implementation of advanced security practices [19]. This approach helps mitigate risks associated with data breaches and unauthorized access in cloud-based systems.

Connection to SCADA and IIoT Systems: SCADA and IIoT devices are increasingly being integrated into cloud infrastructures to enhance their functionality and scalability. By leveraging cloud services, these devices can benefit from advanced analytics, remote monitoring, and improved data storage capabilities. However, this integration also introduces new security challenges.

Ensuring the security of cloud-based SCADA and IIoT systems is crucial to protect against potential cyber threats that could exploit vulnerabilities in the cloud infrastructure.

Key Security Measures:

- **Secure Configuration**: Proper configuration of cloud services is essential to prevent unauthorized access. This includes setting up strong authentication mechanisms, implementing role-based access control, and ensuring that all data is encrypted both in transit and at rest. Misconfigurations can lead to significant security breaches, as seen in numerous high-profile attacks [20].
- **Continuous Monitoring**: Implementing continuous monitoring solutions helps detect and respond to security incidents in real-time. This includes using intrusion detection systems, security information and event management systems, and regular vulnerability assessments. Continuous monitoring is vital for identifying and mitigating potential threats before they can cause significant damage [21].
- **Advanced Security Practices:** Utilizing advanced security practices such as zero-trust architecture, micro-segmentation, and automated threat intelligence can further enhance the security of cloud infrastructures. Zero-trust architecture ensures that no entity, whether inside or outside the network, is trusted by default. Micro-segmentation divides the network into smaller segments to limit the spread of potential attacks. Automated threat intelligence uses machine learning and AI to identify and respond to threats more effectively [22].

Challenges and Solutions:

- **Data Privacy and Compliance:** Ensuring data privacy and compliance with regulatory requirements is a major challenge in cloud environments. Organizations must implement robust data protection measures and ensure compliance with standards such as GDPR, HIPAA, and NIST. Regular audits and compliance checks are necessary to maintain adherence to these regulations.
- **Integration with Legacy Systems:** Integrating cloud-based solutions with existing legacy SCADA and IIoT systems can be complex. Organizations need to ensure that security measures are compatible with both old and new systems to prevent potential vulnerabilities. This may involve updating legacy systems or using middleware solutions to facilitate secure integration [23].
- **Cost and Resource Management:** Implementing comprehensive security measures in the cloud can be resource-intensive and costly. Organizations must balance the cost of security solutions with the potential risks and impacts of security breaches. Investing in scalable and efficient security tools can help manage costs while ensuring robust protection [24].

In conclusion, securing cloud infrastructures is integral to the overall security of SCADA and IIoT systems. By adopting a layered security approach, implementing continuous monitoring, and leveraging advanced security practices, organizations can significantly enhance the resilience of their cloud-based SCADA and IIoT systems against cyber threats. The following section will demonstrate the effectiveness of various approaches in preventing memory problems.

4. Analyzing the effectiveness of methods to prevent memory management problems

The main disadvantage of static analyzers and sanitizers is that these tools need to be integrated with the existing code base. That is, software developers need to research and integrate these tools into the project. In addition, there is a risk of incorrect integration and misuse of the analyzer or sanitizer [25].

Also, most professional analyzers and sanitizers are not free, which in turn imposes certain restrictions on the development of projects with a small budget. For example, let's consider one of the most popular code analyzers sonar, this tool has different tariff plans, but most companies choose the Enterprise plan [26].

Figure 3: Different tariffs provided by Sonar

According to the tariff plans, the cost of using code analyzers can vary from several hundred to hundreds of thousands of dollars per year.

Tools such as sanitizers and static code analyzers help to improve code quality and prevent other known problems quite significantly. Nevertheless, the most reliable way to deal with memory usage issues is to use the Rust programming language and similar ones. In addition, this approach does not require any additional settings on the part of developers, and the use of the Rust language is free, which makes this approach quite optimal.

Fig. 4 shows a C++ program that simulates a buffer overflow. This program creates an array of five elements of type uint32_t (this is an unsigned integer that takes 32 bits), and all elements of the array are initialized to 0. After that, the program iterates over this array, but the iteration interval was chosen incorrectly and the program will go beyond the buffer.

Figure 4: An example of a program to overflow a buffer

This program compiles successfully. But there is an error in it, the loop will iterate over an array of 13 elements when only 5 elements are needed. The resulting values from the execution of this program are shown in Fig. 5.

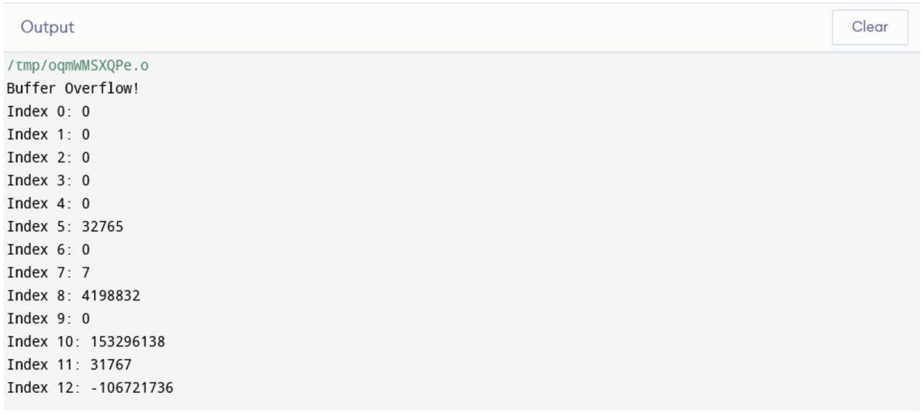

Figure 5: The result of the program execution

Fig. 5 shows that the first 5 elements of the array are zeros, but the sixth element (at index 5) has a value of 32765. Further iterations also show other numbers. If you look at the code listing more closely, you will see that there is no mention of these numbers. That is, it has just been demonstrated how the program went beyond the buffer and accessed data that is outside the executing context.

Along with causing crashes and other problems, these errors can also create security vulnerabilities that can be exploited by attackers to gain unauthorized access to the system. Debugging memory-related errors can be difficult because they often result in subtle errors that are difficult to reproduce. This can lead to lengthy debugging sessions and delayed release cycles, which can be particularly problematic in time-sensitive applications [27].

Rust's ownership model and borrowing system make it virtually impossible to introduce many of these common memory-related errors, which is one of the reasons it has become a popular choice for system programming [28]. Similar approaches are described in [29-32]

Using the Rust programming language (Fig. 6) makes it impossible to prevent buffer overflows and access to other data, as demonstrated in Fig. 4. The results are shown in Fig. 7.

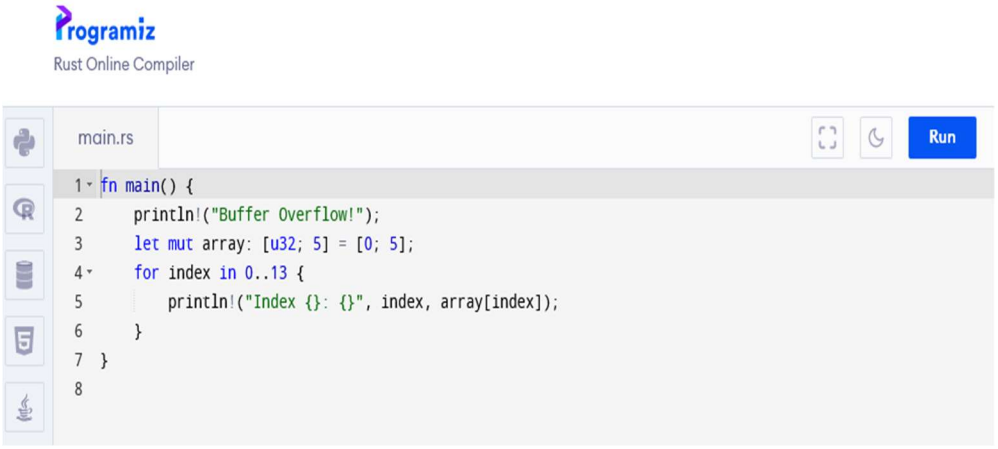

Figure 6: Rust code sample for buffer overflow

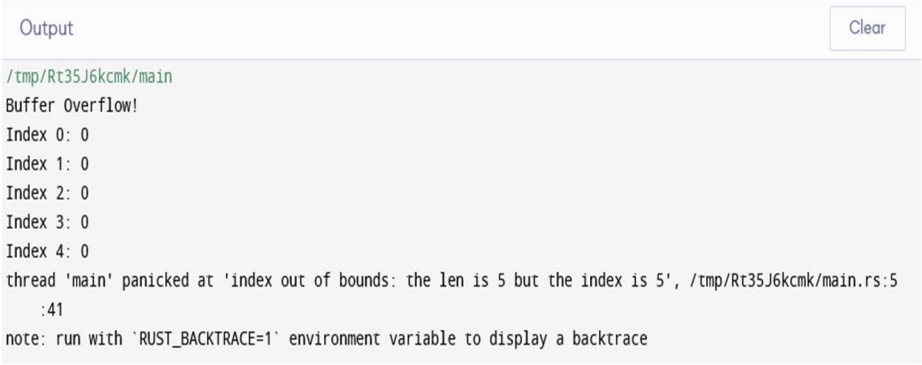

Figure 7: The result of executing the Rust code for a buffer overflow

From Fig. 7, it is clear that the program crashed without allowing it to go beyond the buffer, thus making it impossible to access other information, unlike what happened when executing the C++ program, the results of which are shown in Fig. 5.

While Rust's approach to memory safety offers many advantages, there are also some drawbacks and limitations that should be considered. One potential limitation is that Rust's ownership and borrowing system can be difficult to understand for developers who have not worked with it before. This can make writing Rust code more difficult than languages with simpler memory models, such as Python or JavaScript.

Although Rust's ownership system enhances code safety, it can also complicate working with cyclic data structures. This can lead to a 20–30% increase in the time required to write and debug such code. Rust's memory safety features, which make the code more resistant to errors, slightly reduce its performance. The performance loss can reach 5–10% compared to C/C++ code. It is important to remember that Rust does not guarantee 100% security. Incorrect code can bypass the system's guarantees, and vulnerabilities in third-party libraries remain dangerous.

5. Conclusions

Dynamic memory errors pose a significant cybersecurity threat. Attackers are actively exploiting such vulnerabilities to steal data, disrupt systems, and commit other criminal acts. The damage they cause is estimated at several tens of billions of dollars, and reputational losses are manifested long after the cyberattack and may result in further lawsuits and compensation.

The safe programming practices described here can prevent memory issues. However, it should be noted that none of them is universal and it is better to use a combination of them for maximum protection.

To prevent and reduce the negative impact of dynamic memory management, the best solution is to use a comprehensive approach that includes the following:

Use of safe programming languages.

Use of sanitizers.

Static code analysis.

Professional development of developers.

This paper has demonstrated a memory buffer overflow vulnerability associated with a violation of security rules when working with it. The results were obtained using the C++ language, which is one of the most commonly used languages for writing applications for SCADA and IIoT devices.

As an optimization and solution to such vulnerabilities, it was proposed to use the Rust language. This helped to avoid errors related to memory management. The peculiarity of using this language helped to avoid the vulnerability described above by preventing memory buffer overruns. Rust mechanisms help to avoid a dozen more memory management vulnerabilities.

In general, although Rust's approach to memory safety provides many advantages, it is important to note its limitations and potential drawbacks. Rust's memory safety features can sometimes lead to some performance degradation, and a more comprehensive approach is needed to solve other memory-related problems. Despite these limitations, Rust remains a powerful language for system programming, and its memory protection features provide a significant advantage over many other programming languages and significantly help improve cybersecurity.

References

[1] Data Driven Security Hardening in Android. URL: https://security.googleblog.com/2021/01/data-driven-security-hardening-in.html

[2] A Proactive Approach to More Secure Code. URL: https://msrc.microsoft.com/blog/2019/07/a-proactive-approach-to-more-secure-code/

[3] Memory Safety. URL: https://www.chromium.org/Home/chromium-security/memory-safety/

[4] P. Oorschot, Memory Errors and Memory Safety: C as a Case Study, IEEE Security & Privacy 21 (2023) 70–76. doi: 10.1109/MSEC.2023.3236542.

[5] H. Altaleb, Z. Rajnai, A Comprehensive Analysis and Solutions for Enhancing SCADA Systems Security in Critical Infrastructures, IEEE 11th International Conference on Computational Cybernetics and Cyber-Medical Systems (ICCC) (2024). doi: 10.1109/ICCC62278. 2024.10582956.

[6] M. Fall, et al., Enhancing SCADA System Security (2020) 830–833. doi: 10.1109/MWSCAS48704.2020. 9184532.

[7] Project Zero. URL: https://googleprojectzero.blogspot.com/2022/04/the-more-you-know-more-you-know-you.html

[8] Memory Unsafety in Apple's Operating Systems. URL: https://langui.sh/2019/ 07/23/apple-memory-safety/

[9] D. Shevchuk, et al., Designing Secured Services for Authentication, Authorization, and Accounting of Users, in: Cybersecurity Providing in Information and Telecommunication Systems II, vol. 3550 (2023) 217–225.

[10] Y. Martseniuk, et al., Automated Conformity Verification Concept for Cloud Security, in: Cybersecurity Providing in Information and Telecommunication Systems, vol. 3654 (2024) 25–37.

[11] National Vulnerability Database. URL: https://nvd.nist.gov/vuln/detail/CVE-2021-45046

[12] The Heartbleed Bug. URL: https://heartbleed.com/

[13] P. Oorschot, Memory Errors and Memory Safety: A Look at Java and Rust, IEEE Security & Privacy 21 (2023) 62–68. doi: 10.1109/MSEC.2023.3249719.

[14] S. Yevseiev, et al. Models of Socio-Cyber-Physical Systems Security: monograph, PC TECHNOLOGY CENTER. (2023). doi: 10.15587/978-617-7319-72-5.

[15] H. Kim, J. Lee, Enhancing SCADA Security with Advanced Memory Management Techniques, Int. J. Critical Infrastruct. Prot. 38 (2024) 100493.

[16] G. Saileshwar, et al., HeapCheck: Low-cost Hardware Support for Memory Safety, ACM Transactions on Architecture and Code Optimization 19 (2022) 1–24. doi: 10.1145/3495152.

[17] S. Rajasekaran, V. Kumar, Mitigating Memory Vulnerabilities in IoT Systems: Best Practices and

Case Studies, Future Generation Comput. Syst. 137 (2024) 146–159.
[18] J. Zhou, R. Liu, Cybersecurity in Industrial Control Systems: A Survey, IEEE Transactions on Industrial Informatics 19(3) (2023) 1621–1632.
[19] F. Lomio, S. Moreschini, V. Lenarduzzi, A Machine and Deep Learning analysis among SonarQube rules, Product, and Process Metrics for Faults Prediction, Empirical Software Eng. 27 (2022). doi: 10.1007/s10664-022-10164-z.
[20] V. Khoma, et al., Comprehensive Approach for Developing an Enterprise Cloud Infrastructure, in: CEUR Workshop Proceedings, vol. 3654 (2024) 201–215.
[21] T. Nguyen, D. Pham, Secure Programming Practices for Embedded Systems: A Focus on Memory Safety, Embedded Systems Letters 15(4), (2023) 145–150.
[22] D. Schmidt, Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects. Volume 2. Wiley (2006).
[23] Sonar. URL: https://www.sonarsource.com/plans-and-pricing/
[24] S. Zhang, et al., A Survey of Memory Management Techniques in Embedded Systems, ACM Computing Surveys 55(2) (2022).
[25] M. Jones, P. Smith, Advancements in Secure Memory Management for Critical Systems, J. Syst. Archit. 121 (2023) 102384.
[26] S. Turner, D. Harris, A Comprehensive Review of Memory Safety Mechanisms in IoT Devices, Sensors, 23(6) (2023) 1428.
[27] L. Zhang, Y. Wang, Static and Dynamic Analysis Tools for Memory Safety: A Comparative Study, Software: Practice and Experience 53(7) (2023) 1298–1312.
[28] K. Huang, et al., Comprehensive Memory Safety Validation: An Alternative Approach to Memory Safety, IEEE Security & Privacy (2024) 2–11. doi: 10.1109/MSEC.2024.3379947.
[29] O. Solomentsev, et al., Data Processing Method for Deterioration Detection during Radio Equipment Operation, IEEE Microwave Theory and Techniques in Wireless Communications, MTTW (2019) 1–4.
[30] O. Solomentsev, M. Zaliskyi, Correlated Failures Analysis in Navigation System, IEEE 5th International Conference on Methods and Systems of Navigation and Motion Control, MSNMC 2018 – Proceedings (2018) 123–126.
[31] O. Solomentsev, et al., Efficiency of operational data processing for radio electronic equipment, Aviation 23(3) (2020) 71–77.
[32] O. Solomentsev, et al., Efficiency of data processing for UAV operation system, IEEE 4th International Conference on Actual Problems of Unmanned Aerial Vehicles Developments, APUAVD 2017 - Proceedings (2018) 27–31.