INTERNATIONAL JOURNAL OF LATEST TECHNOLOGY IN ENGINEERING, MANAGEMENT & APPLIED SCIENCE (IJLTEMAS)

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XIV, Issue IX, September 2025

Buffer Overflow Vulnerabilities in the Age of AI: Challenges and Mitigation Strategies

Hritik Sharma[1], Prof. (Dr.) Seema Gupta[2]

[1]Assistant Professor, IIMT

[2]Associate Professor IIMT

DOI: https://doi.org/10.51583/IJLTEMAS.2025.1409000015

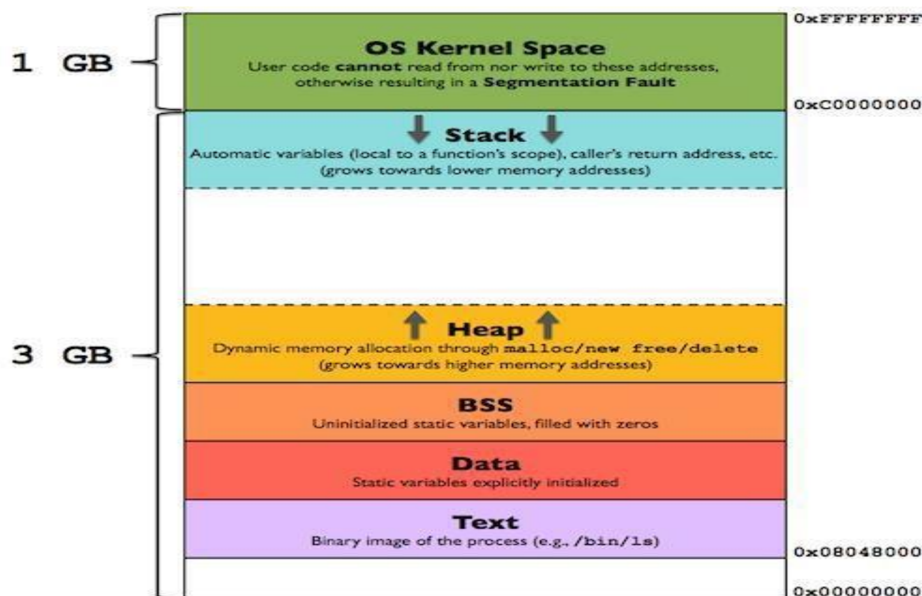
Abstract: Buffer overflow vulnerabilities have plagued software systems for decades and continue to pose a significant security risk. My paper provides a comprehensive analysis of buffer overflows, detailing their mechanisms, historical context, and the challenges they present to modern systems. It explores traditional and contemporary defense strategies, including compile-time and run-time defenses, and examines the potential role of emerging technologies like Artificial Intelligence (AI) in mitigating these vulnerabilities. The paper emphasizes the ongoing need for robust security practices and continuous research to address this enduring threat.

Keywords: Buffer overflow, buffer overrun, cybersecurity, software vulnerabilities, artificial intelligence, security mitigation, stack overflow, heap overflow, memory corruption, code injection, exploit techniques, return address overwriting, Address Space Layout Randomization (ASLR).

I. Introduction

The rapid advancement of technology has brought about transformative changes across various domains. However, this progress has also been accompanied by evolving security challenges. Among these challenges, buffer overflow vulnerabilities stand out as a persistent threat. Buffer overflows, rooted in programming errors, have been exploited to compromise systems for many years, highlighting a fundamental flaw in software development practices. This paper aims to provide an in-depth exploration of buffer overflow vulnerabilities, analyzing their characteristics, impact, and mitigation strategies in the context of modern computing.

Understanding Buffer Overflow Vulnerabilities



Defining Buffer Overflow

A buffer overflow, also known as a buffer overrun, occurs when a program attempts to write more data to a fixed-size buffer than the buffer is allocated to hold. This leads to overwriting adjacent memory locations, potentially corrupting program data, control flow information, and other critical data. The National Institute of Standards and Technology (NIST) defines a buffer overflow as "A condition at an interface under which more input can be placed into a buffer or data holding area than the capacity allocated, overwriting other information. Attackers exploit such a condition to crash a system or to insert specially crafted code that allows them to gain control of the system".

Mechanisms of Buffer Overflow

Buffer overflows are fundamentally programming errors. They arise when a program does not properly validate the size of input data before writing it to a buffer. This oversight can occur in various memory areas, including:

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XIV, Issue IX, September 2025

- Stack: Used for storing local variables, function arguments, and return addresses. Stack overflows are a common type of buffer overflow.
- Heap: Used for dynamic memory allocation. Heap overflows can be more complex to exploit than stack overflows.
- Data section: Used for storing global and static variables.
- BSS section: Used for uninitialized static variables.

Consequences of Buffer Overflow

The consequences of a successful buffer overflow exploit can be severe, including:

- Corruption of program data.
- Unexpected transfer of control.
- Memory access violations.
- Execution of arbitrary code.

Historical Perspective

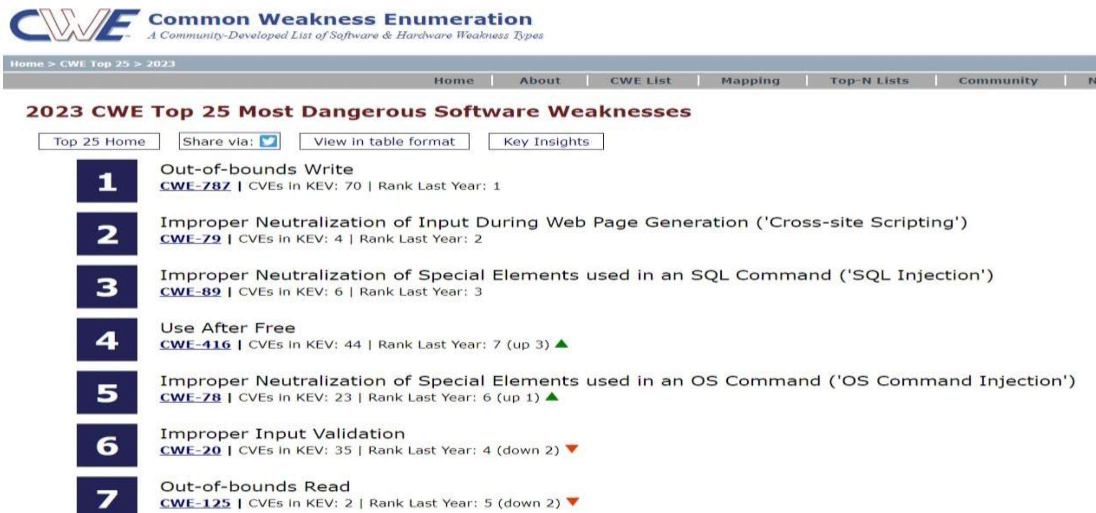
Buffer overflow attacks are not a new phenomenon. Their history dates back several decades, with significant attacks demonstrating their enduring impact.

- Early Exploits: The Morris Internet Worm in 1988 utilized a buffer overflow exploit in the "fingerd" program.
- Landmark Publications: The publication of "Smashing the Stack for Fun and Profit" in Phrack magazine in 1996 provided detailed information on exploiting stack-based buffer overflows.
- Worm Attacks: Several high-profile worm attacks, such as the Code Red worm (2001), the Slammer worm (2003), and the Sasser worm (2004), exploited buffer overflows in widely used software like Microsoft IIS and SQL Server.

This history underscores that buffer overflows have been a persistent threat, exploited in various attacks with significant consequences.

Prevalence and Impact

Despite the availability of prevention techniques, buffer overflows remain a major concern.



- CWE Top 25: Buffer overflow variants are consistently listed among the most dangerous software weaknesses in the Common Weakness Enumeration (CWE) Top 25.
- Continued Exploitation: Buffer overflows continue to be exploited in attacks against operating systems and applications and are prevalent in exploit toolkits.
- Root Causes: The continued prevalence can be attributed to factors such as legacy code, careless programming practices, and the complexity of modern software systems.

The impact of buffer overflows extends from system crashes and data corruption to complete system compromise and unauthorized code execution, making them a critical security concern.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XIV, Issue IX, September 2025

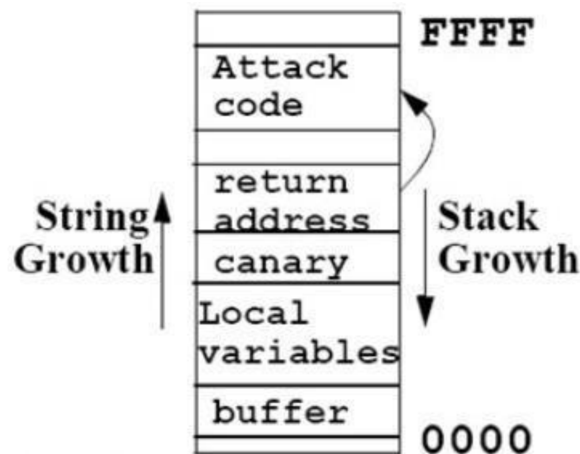
Traditional Buffer Overflow Defenses

Traditional approaches to mitigating buffer overflows involve both compile-time and run-time defenses.

Compile-Time Defenses

Compile-time defenses aim to prevent buffer overflows during the software development process.

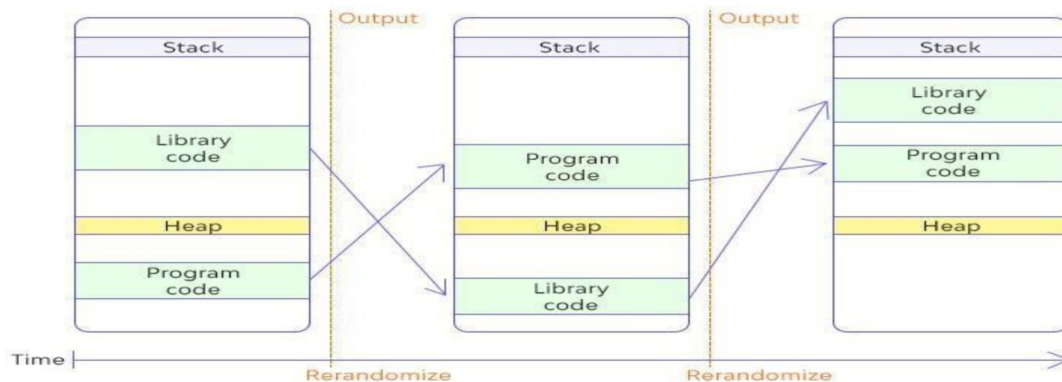
- Programming Language Choice: Using modern high-level languages with built- in bounds checking can help prevent buffer overflows. However, this may come at the cost of performance overhead and reduced control over system resources.
- Safe Coding Practices: Emphasizing secure coding practices, such as thorough input validation and avoiding unsafe functions, is crucial. Projects like OpenBSD demonstrate the effectiveness of rigorous code auditing and rewriting.
- Language Extensions and Safe Libraries: Replacing unsafe standard library functions with safer alternatives (e.g., Libsafe) and using language extensions to enforce bounds checking can provide added protection.
- Stack Protection Mechanisms: Techniques like StackGuard and StackShield are compiler extensions that introduce mechanisms to detect and prevent stack-based buffer overflows by using canaries or copying return addresses.



Run-Time Defenses

Run-time defenses aim to detect and prevent buffer overflows in running programs.

- Executable Address Space Protection: Preventing the execution of code on the stack or heap by marking those memory regions as non-executable. This relies on memory management unit (MMU) support.
- Address Space Layout Randomization (ASLR): Randomizing the memory layout of key data structures (stack, heap, libraries) to make it harder for attackers to predict target addresses.
- Guard Pages: Placing guard pages (memory regions marked as invalid) around sensitive memory areas to detect out-of-bounds accesses.

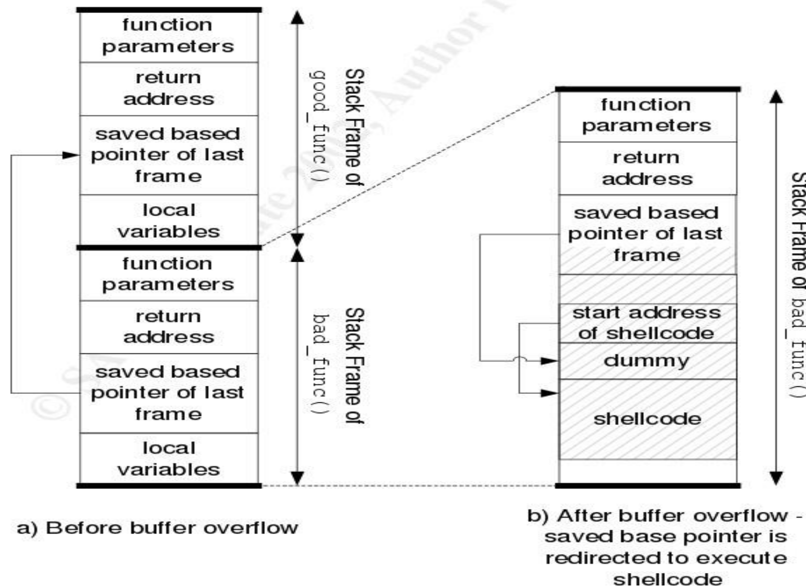


ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XIV, Issue IX, September 2025

Beyond Traditional Buffer Overflows

The document also mentions other forms of overflow attacks that go beyond the basic stack overflow:

- Replacement Stack Frame: Overwriting the buffer and saved frame pointer to redirect control flow.



- Return-to-libc Attacks: Exploiting vulnerabilities even with non-executable stacks by overwriting the return address to point to existing library functions (e.g., system()).
- Heap Overflows: Overflows that occur in the heap memory region, potentially corrupting heap metadata or function pointers.

These variations demonstrate the adaptability of attackers and the need for comprehensive defense strategies.

The Role of AI in Buffer Overflow Mitigation

Emerging technologies, particularly AI, offer new opportunities to enhance buffer overflow defenses.

AI-Powered Fuzzing: AI can optimize fuzzing by generating more effective test cases, increasing the likelihood of discovering vulnerabilities.

AI-Driven Static Analysis: Machine learning models can be trained to identify vulnerable code patterns with greater accuracy.

AI-Enhanced Intrusion Detection: AI can improve the detection of buffer overflow attacks by analyzing system behavior and network traffic.

Automated Vulnerability Patching: AI could potentially automate the process of generating and deploying patches for buffer overflow vulnerabilities.

Challenges and Future Directions

While AI offers potential benefits, challenges remain:

- AI Complexity: Developing and deploying AI-based security solutions can be complex.
- Data Requirements: Training AI models requires large and diverse datasets.
- Adversarial Attacks: AI-based defenses may be vulnerable to adversarial attacks designed to evade detection.
- Explainability: Ensuring the explainability of AI-driven security decisions is crucial for trust and accountability.

Future research should focus on addressing these challenges and exploring new AI techniques to create more robust and adaptive buffer overflow defenses.

II. Conclusion

Buffer overflow vulnerabilities continue to be a significant security concern, demanding ongoing attention from researchers, developers, and security professionals. A combination of traditional and AI-driven defense strategies is essential to effectively mitigate this threat. By improving software development practices, leveraging technological advancements, and fostering

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XIV, Issue IX, September 2025

collaboration, it is possible to create more secure and resilient systems.

References

1. Black, Paul E., and Irena Bojanova. "Defeating buffer overflow: A trivial but dangerous bug." IT professional 18, no. 6 (2016): 58-61.
2. Maroš, B., Homoliak, I., Kacic, M. and Petr, H., 2013, October. Detection of network buffer overflow attacks: A case study. In 2013 47th International Carnahan Conference on Security Technology (ICCST) (pp. 1-4). IEEE.
3. Piromsopa, Krerk, and Richard J. Enbody. "Buffer-overflow protection: the theory." 2006 IEEE International Conference on Electro/Information Technology. IEEE, 2006.
4. Day, David Jonathan. Mitigating the Risk of Buffer Overflow Attacks Against Forked Daemon Servers Using Network Intrusion Detection Systems. University of Derby (United Kingdom), 2010.
5. Alam, Shahid. "Cybersecurity: Past, present and future." arXiv preprint arXiv:2207.01227 (2022).
6. Gu A, Jain N, Li WD, Shetty M, Shao Y, Li Z, Yang D, Ellis K, Sen K, Solar-Lezama Challenges and paths towards ai for software engineering. arXiv preprint arXiv:2503.22625. 2025 Mar 28.
7. Xu H, Wang S, Li N, Wang K, Zhao Y, Chen K, Yu T, Liu Y, Wang H. Large language models for cyber security: A systematic literature review. arXiv preprint arXiv:2405.04760. 2024 May 8.