# Buffer Overflow / Heap Overflow

A Buffer Overflow occurs when a program writes more data to a buffer (temporary memory allocation) than it can hold, causing adjacent memory corruption. A Heap Overflow is a special case where the overflow occurs in the heap region (dynamically allocated memory at runtime).

These flaws often lead to arbitrary code execution, crashes, or privilege escalation. 

**Example**: Input longer than the allocated buffer.

## Memory Basics

| Memory Segment | Purpose |
| --- | --- |
| Stack | Function calls, local variables (LIFO) |
| Heap | Dynamically allocated memory (malloc, new) |
| Data | Global/static variables |
| Code/Text | Program instructions |

## Types of Buffer Overflows

| Type | Definition | Affected Memory | Common Language |
| --- | --- | --- | --- |
| 1. Stack Overflow | Overflows a buffer on the stack | Stack | C, C++ |
| 2. Heap Overflow | Overflows a buffer in the heap | Heap | C, C++, some Java |
| 3. Integer Overflow + Buffer Overflow | Incorrect size calculation leads to unsafe memory allocation | Heap or Stack | C |
| 4. Off-by-One Overflow | Writing one byte past the buffer boundary | Stack | C |
| 5. Format String Attack | Uses %x, %s, etc. to read/write arbitrary memory | Stack or heap | C |

## Examples

**Stack Buffer Overflow (Classic):**   
char buf[8];  
strcpy(buf, "AAAAAAAAAAAAAAAA");

Input is 16 bytes, but buffer is only 8 bytes → overflows adjacent memory.

**Heap Overflow**char\* buf = malloc(8);  
strcpy(buf, "AAAAAAAAAAAAAAAA");  
Overwrites heap metadata → can lead to arbitrary memory write.

## Real-World Exploit

MS03-026: Windows RPC buffer overflow (Blaster Worm).

Heartbleed Bug: Buffer over-read in OpenSSL’s heartbeat extension.

## Impact

| Impact Type | Description |
| --- | --- |
| Remote Code Execution (RCE) | Attacker injects and executes shellcode |
| Privilege Escalation | Exploit buffer overflow in SUID binaries |
| Denial of Service (DoS) | Crash the application or service |
| Bypass of Security Controls | Overwrite variables, bypass logic, authentication |

## Exploitation Process

Find vulnerable input (e.g., user input copied without length check)  
Craft payload to overflow buffer  
Overwrite saved return address (stack)  
Redirect execution to injected shellcode

# Tools for Detection / Exploitation

| Tool | Use |
| --- | --- |
| GDB | Debugging & memory analysis |
| pwndbg, gef, peda | GDB enhancements for exploit dev |
| fuzzing (AFL, LibFuzzer) | Find overflow vulnerabilities |
| valgrind / AddressSanitizer | Detect memory corruption |
| Metasploit | Exploits with payloads |
| Immunity Debugger / OllyDbg | Windows memory analysis |

# Mitigation Techniques

| Mitigation | Description |
| --- | --- |
| Use Safe Functions | Use strncpy, snprintf, avoid gets, strcpy |
| Bounds Checking | Validate input size before copying |
| Stack Canaries | Special values placed before return addresses (detected if overwritten) |
| ASLR (Address Space Layout Randomization) | Randomizes memory layout at runtime |
| DEP / NX Bit (Data Execution Prevention) | Prevents code execution in data sections |
| Control Flow Integrity (CFI) | Prevents hijacking control flow |
| Fuzz Testing | Automated input mutation to find overflows |
| Memory-Safe Languages | Use Rust, Go, or Python when possible |

# Points

“Buffer overflow remains one of the **most dangerous vulnerabilities**, enabling full system compromise.”

“During VAPT, simulate overflow risks using **fuzzing, static analysis, and manual source code review.**”

“Ensure that team avoids unsafe functions, uses **ASLR, DEP**, and integrates **memory error detection** tools in CI/CD.”   
  
ASLR randomizes the memory addresses used by system components (stack, heap, libraries, etc.) every time a program is run. This prevents attackers from reliably predicting the memory location of injected code or known functions (like libc, system, shellcode).

DEP marks certain regions of memory (like the **stack**, **heap**, or **data segments**) as **non-executable**, meaning even if an attacker injects code there, **the CPU will not execute it.**

# Compliance Impact

| Standard | Relevance |
| --- | --- |
| PCI-DSS | Secure coding guidelines for buffer control |
| ISO/IEC 27001 | Application layer protection |
| CWE-120 | Common Weakness Enumeration ID for Buffer Overflow |
| OWASP Top 10 (2017 - A9) | Known Vulnerabilities (overflow-based CVEs) |

# Real Breach Cases

| Case | Description |
| --- | --- |
| Blaster Worm (2003) | RPC DCOM Buffer Overflow – global RCE |
| Heartbleed (2014) | OpenSSL overflow → sensitive memory leak |
| Slammer Worm (2003) | MS SQL buffer overflow in UDP packet parsing |