

Overview Over Attack Vectors and Countermeasures for Buffer Overflows

Valentin Brandl Wissenschaftliches Seminar Faculty of Computer Science and Mathematics OTH Regensburg

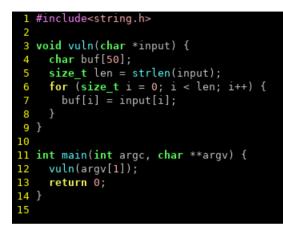


- Motivation
- Technical Overview
- Ways of Exploiting Buffer Overflows
- Analyzed Countermeasures
- Discussion



- 14% of CVEs in 2018 were BOF
- Concerns languages with manual memory management (C, C++, Fortran)
- Second most used programming language: C (2019)





buf	0xC8	\leftarrow SP (vuln)
buf		
buf	0xFA	\leftarrow BP (vuln)
[old IP]	0xFB	
[BP (main)]	0xFC	
[*input]	0xFD	
argc	0xFE	
argv	0xFF	

[payload]	0xC8	\leftarrow SP (vuln)
[payload]		
[payload]	0xFA	\leftarrow BP (vuln)
[controlled IP]	0xFB	
[BP (main)]	0xFC	
[*input]	0xFD	
argc	0xFE	
argv	0xFF	

argc	0xFE	\leftarrow SP (main)
argv	0xFF	\leftarrow BP (main)



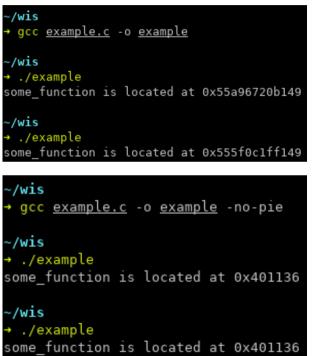
- Attacker overwrites any kind of function pointer (return address, VMT, ...)
- Attacker places payload in memory or reuses existing code
- When function pointer is used, attacker gains code execution
- DoS is also possible by accessing invalid memory



- Randomize location of program in memory
- Attacker doesn't know where payload is located
- Prevents code execution
- Information leak allows exploitation
- Brute-force of 32 bit addresses possible
- Does not prevent DoS
- Compile-time mitigation, no code changes needed









- Memory can be either writable or executable
- Attacker cannot supply shellcode directly
- Code reuse still possible
- Compile-time mitigation, no code changes needed

NX



- Markers at the end of a stack frame
- Invalid marker → Buffer overflow occurred
- No code changes required
- Only mitigates stack-based BOF
- Knowledge of canary allows bypassing



- Read-only stack for return addresses
- Compared before return
- Compiler extension
- Only against stack-based BOF



- Each indexing operation is checked
- 100% effective (where applied)
- Non-trivial runtime overhead
- Used in languages with runtimes (Java, C#, Python, ...)



- Value (size) is associated with a buffer
- Only allow indexing with validated values
- Language extension
- Lot of work to use, but type inference helps



- Major OS implement ASLR
- Compilers implement PIE, NX, Stack Canaries (discussable defaults)

Mitigation	GCC?	clang?
PIE	No	No
NX	Yes	Yes
Stack Canary	No	No



- Most techniques only prevent exploitation (code execution)
- DoS might be just as critical (aviation, autonomous driving, ...)
- Only dependent typing and RBC actually prevent BOF



- Use C, C++ and Fortran only if unavoidable and enable compiler mitigations
- Viable alternatives exist (Rust, Go, Java, ...)



• Thank you for listening