

Protection

Thierry Sans

Why are we still vulnerable to buffer overflows and other memory corruption vulnerabilities?

Why code written in assembly code or C are subject to buffer overflow attacks?

- ➔ Because C has primitives to manipulate the memory directly (pointers ect ...)

Choosing a better programming language

Some languages are type-safe (i.e memory safe)

- ➔ Pure Lisp, pure Java, ADA, ML ...

Some languages isolate potentially unsafe code

- ➔ Modula-3, Java with native methods, C# ...

Some languages are hopeless

- ➔ Assembly languages, C ...

Type-Safe Programs

- ➔ Cannot access arbitrary memory addresses
- ➔ Cannot corrupt their own memory
- ✓ Do not crash

So why are we still using unsafe programming languages?

If other programming languages are “memory safe”, why are we not using them instead?

- ➔ Because C and assembly code are used when a program requires high performances (audio, video, calculus ...)
or when dealing with hardware directly (OS, drivers)

How to write better programs with unsafe programming languages?

- Defensive Programming
- Penetration testing
- Formal testing
- Formal development

How to run programs written with unsafe programming languages?

- Fortify Source Functions
- Stack Canaries
- DEP/NX - Non Executable Stack
- ASLR - Address Space Layout Randomization
- PIC/PIE - Position Independent Executables

How to run programs written with
unsafe programming languages?

Fortify Source Functions

- ➔ GCC macro `FORTIFY_SOURCE` provides buffer overflow checks for unsafe C libraries

`memcpy, mempcpy, memmove, memset, strcpy, stpcpy, strncpy, strcat, strncat, sprintf, vsprintf, snprintf, vsnprintf, gets`

Checks are performed

- some at compile time (compiler warnings)
- other at run time (code dynamically added to binary)

Canaries

- The compiler modifies every function's prologue and epilogue regions to place and check a value (a.k.a a canary) on the stack
- When a buffer overflows, the canary is overwritten. The programs detects it before the function returns and an exception is raised
- Different types:
 - random canaries
 - xor canaries
- Disabling Canary protection on Linux
`$ gcc ... -fno-stack-protector`
- Bypassing canary protection : *Structured Exception Handling (SEH)* exploit overwrite the existing exception handler structure in the stack to point to your own code

DEP/NX - Non Executable Stack

- The program marks important structures in memory as non-executable
- The program generates an hardware-level exception if you try to execute those memory regions
- This makes normal stack buffer overflows where you set `eip` to `esp+offset` and immediately run your shellcode impossible
- Disabling NX protection on Linux
`$ gcc ...-z execstack`
- Bypassing NX protection : *Return-to-lib-c* exploit
return to a subroutine of the lib C that is already present in the process' executable memory

ASLR - Address Space Layout Randomization

- The OS randomize the location (random offset) where the standard libraries and other elements are stored in memory
- Harder for the attacker to guess the address of a lib-c subroutine
- Disabling ASLR protection on Linux
`$ sysctl kernel.randomize_va_space=0`
- Bypassing ASLR protection : Brute-force attack to guess the ASLR offset
- Bypassing ASLR protection : *Return-Oriented-Programming (ROP)* exploit use instruction pieces of the existing program (called "gadgets") and chain them together to weave the exploit

PIC/PIE - Position Independent Code/Executables

- **Without PIC/PIE**

code is compiled with absolute addresses and must be loaded at a specific location to function correctly

- **With PIC/PIE**

code is compiled with relative addressing that are resolved dynamically when executed by calling a function to obtain the return value on stack

Confined execution environment - Sandbox

A sandbox is tightly-controlled set of resources for untrusted programs to run in

- ➔ Sandboxing servers - virtual machines
- ➔ Sandboxing programs
 - Chroot and AppArmor in Linux
 - Sandbox in MacOS Lion
 - Metro App Sandboxing in Windows 8
- ➔ Sandboxing applets - Java and Flash in web browsers

Intrusion Detection/Prevention Systems

- Host-based Intrusion Detection Systems (IDS)
- Host-based Intrusion Prevention systems (IPS)
- ✓ Based on signatures (well known programs)
- ✓ Based on behaviors (unknown programs)
- ➡ Example : Syslog and Systrace on Linux
- ⦿ Vulnerable to malicious programs residing in the kernel called “rootkits”

How to write better programs with
unsafe programming languages?

Defensive programming (I)

Adopting good programming practices

Modularity

- ➔ Have separate modules for separate functionalities
- ✓ Easier to find security flaws when components are independent

Encapsulation

- ➔ Limit the interaction between the components
- ✓ Avoid wrong usage of the components

Information hiding

- ➔ Hide the implementation
- ⦿ Black box model does not improve security

Defensive programming (2)

Being security aware programmer

- ✓ Check the inputs, even between components that belongs to the same application (mutual suspicion)
- ✓ Be “fault tolerant” by having a consistent policy to handle failure (managing exceptions)
- ✓ Reuse known and widely used code by using design patterns and existing libraries

Penetration Testing

Testing the functionalities

- ✓ Unit test, Integration test, Performance test and so on ...

Testing the security

- ✓ Penetration tests
- ➔ Try to make the software fail by pushing the limits of a “normal” usage i.e test what the program is not supposed to do

Using formal methods to verify a program

Static analysis - analyzing the code to detect security flaws

- Control flow - analyzing the sequence of instructions
 - Data flow - analyzing how the data are accessed
 - Data structure - analyzing how data are organized
- ➔ Abstract interpretation [Cousot]
- ✓ Verification of critical embedded software in Airbus aircrafts

Using formal methods to generate a program

Mathematical description of the problem



Executable code
or hardware design

A large, thick green arrow points downwards from the text 'Proof of correctness' to the final text 'Executable code or hardware design'.

Examples

Hardware design (VHDL, Verilog)

- ✓ Used by semi-conductor companies such as Intel

Critical embedded software (B/Z, Lustre/Esterel)

- ✓ Urban Transportation
(METEOR Metro Line 14 in Paris by Alstom)
- ✓ Rail transportation (Eurostar)
- ✓ Aeronautic (Airbus, Eurocopter, Dassault)
- ✓ Nuclear plants (Schneider Electric)

Pros and cons of using formal methods

✓ Nothing better than a mathematical proof

➡ A code “proven safe” is safe

⊙ Development is time and effort (and so money) consuming

➡ Should be motivated by the risk analysis

⊙ Do not prevent from specification bugs

➡ Example of network protocols