

Operating Systems: Design and Security

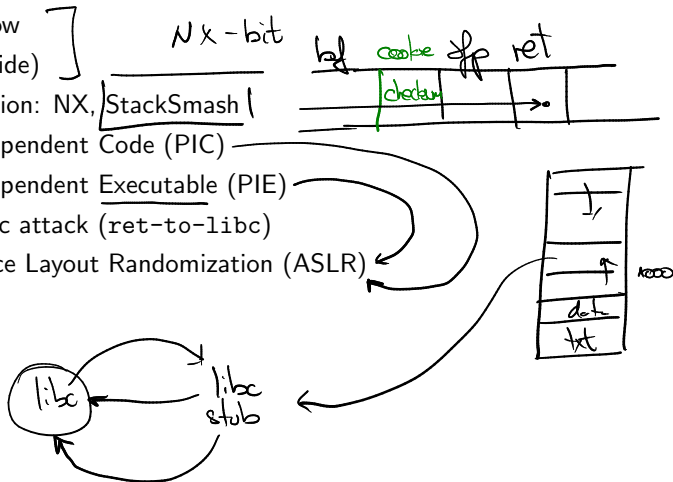
Address Space Randomization

Paul Irofti

Security and Applied Logic Master's
1st Year, 1st Semester, 2019-2020

Concepts

- buffer overflow
- NOP sled (slide)
- stack protection: NX, StackSmash!
- Position Independent Code (PIC)
- Position Independent Executable (PIE)
- Return to libc attack (ret-to-libc)
- Address Space Layout Randomization (ASLR)



Buffer Overflow Example

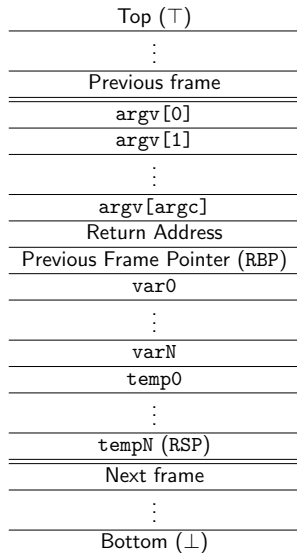
The following code produces a buffer overflow:

```
char buf[10];  
i = 0;  
while (i < 20) {  
    buf[i] = i;  
    i = i + 1;  
}
```

Questions:

- How can I exploit it?
- Where is the interesting code?
- How do I get from the point of entry to the interesting code?

Stack Frame



NOP Sled

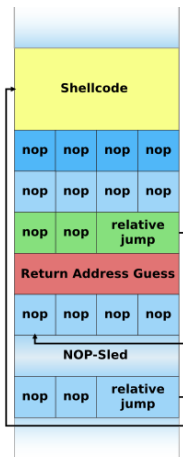


Figure: https://en.wikipedia.org/wiki/Buffer_overflow

Stack Protection

Solutions:

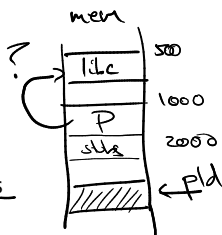
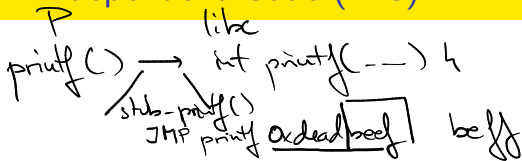
- canaries
 - terminator: enforce RET+ NUL to stop shellcode reads
 - random: created at load, guarded by unmapped pages
 - random XOR: extra layer of xor-key protection
- bounds checking: automated runtime checks
- tagging: W \wedge X, non-executable (NX) bit for pages

Implementations:

- StackGuard
- ProPolice
 - AddressSanitizer
- StackGhost

Supported by all compilers and operating systems. All strategies are enabled by default only on OpenBSD.

Position Independent Code (PIC)



- just like shellcode, but for program sections
- PIC code can run no matter of its absolute load address
- primary users: shared libraries
- program calls library stubs that call into the library (see past laboratory work)
- indirect data access through Global Offset Tables (GOTs)
- one GOT per compilation unit
- GOT address is fixed but unknown until link time
- extra: `ld.so` linker randomizes the GOT address

CALL addr
CALL v[addr]
CALL *EAX

Position Independent Executable (PIE)

- executable where all sections are PIC
- recipe for address space layout randomization (ASLR)
- easy to implement for dynamic executables
- harder for static executables
- most operating systems have support, few enable it by default (OpenBSD, Mac-based, Android, and few Linux distros)

Return to libc attack (ret-to-libc)

- buffer overflow return address is a libc function (not a shellcode)
- gets around tagging, NX-bit, W^X protection
- avoids shellcode design issues (addresses, offsets, NOP-sleds)
- attack example:
 - load libc
 - find the address of the string /bin/sh !
 - find the function address for system(3) nextave(2)
 - fill the payload (shellcode) with calls to system("/bin/sh")

Address Space Layout Randomization (ASLR)

- ASLR represents a suite of protection strategies
 - makes it harder on the attacker to locate memory addresses of various objects (strings, functions, offsets)
 - PAGEEXEC: emulates the NX-bit
 - SEGMEEXEC: emulates the NX-bit on 32-bit x86 through mirroring
- Restrict `mprotect(2)`: enforce W^X behaviour for PROT flags `mmap(2)`
- Randomize stack base `get_spd`
- Randomize `mmap(2)` base
 - randomize ET_EXEC base
- KASLR is the userland equivalent for the kernel
 - 2002: initial design and implementation by PaX Team as a patch for Linux
 - 6 months later: OpenBSD adapted, re-implemented and enabled by default; first OS to do that
- today: still not accepted in mainline Linux, maintained by grsecurity group