Lecture 6
Exploit Protection Mechanisms

Computer and Network Security
November 07, 2022

Computer Science and Engineering Department

Outline

Why Do Exploiting?

- money
- fame
- money
- challenge
- money
- politics
- fun
- career

- penetration testing
- security consulting
- security auditing
- prepare for defense
- knowledge base

Attack Objectives

- gain control (root access)
- leak information (privacy leaks, passwords)
- cripple infrastructure (denial of service, shut down)

Runtime Application Attacks

- exploit applications while running
- alter application behavior
- exploiting vulnerabilities and misconfigurations
- focus is controlling the system (root account)
- an intermediary step is gaining shell access to user
- privilege escalation

Steps in Runtime Application Attacks

- goal is alter the application control flow
- either use existing functionality (in some new way) or inject new functionality
- rewrite/rewire configuration/code to create a new execution path
- it starts with a buffer overflow and an overwrite of data

- buffer overflow overwrites a code pointer
- find a suitable address to point new code pointer to
- execute new path
- code pointer may be: return address (on stack), function pointer

Buffer Overflow

- write beyond buffer limits
- stack-based overflow: overwrite variable, return address or function pointer
- heap overflow: corrupt dynamically allocated memory

Shellcode

- sequence of machine level instructions
- stored in memory at a convenient address
- executed when requested by jumping at the start address

Using the Local Stack

- place shellcode in local buffer on stack
- rewrite return address to point to beginning of the buffer on the stack
- may need NOPs if exact address is not known
- unable to be done if stack is non-executable

Shellcode Attack Vector

- find vulnerability: buffer overflow
- determine offset from buffer to code pointer
- determine address of buffer storing the shellcode
- build shellcode
- create payload: injected shellcode + padding + return address overwrite in payload
- trigger attack: send data as argument, standard input or environment variable; jump to shellcode address
- attack is executed by executing shellcode

- (from Jonathan Katz, CMSC 414, Computer and Network Security)
- prevent existence
  - safe programming/secure coding
  - input validation
  - static/dynamic analysis
- prevent exploitation
  - ASCII armored address space
  - stack guard/stack protection
  - non executable stack/data execution prevention
  - address space layout randomization (ASLR)

Outline

Safe Programming

- string management
- integer management
- buffer management
- bounds checking
- safe typing, data conversions
- code auditing
- ...

Input Validation

- sanitizing input
- all are printable characters in case of string functions
- proper data type, proper sign
- may incur overhead

- analyze source code without running the program
- coverity, cppcheck, splint, clang static analyzer
- may also be done on binary files: Veracode, CodeSonar

Dynamic Analysis

- run program and find vulnerabilities
- fuzz testing: send random data as input
- slows program, may find more problems
- valgrind, purify, dmalloc

Outline

At a Glance

- code integrity protection
- randomize address space
- stack guard

ASCII Armored Address Space

- place code, data and libraries at addresses starting with 0x00
- disables attacks that require the NUL byte to be absent
- certain attacks may work even if the NUL byte is present

Code Integrity

- do not modify code, do no execute writable zone
- stack and other zones of memory that are writable are marked non-executable
- any jump to the stack or heap would result in access violation

W xor X

- page/segment is either writable or executable

NX Bit

- page is marked as non-executable
- CPU bit, set by OS
- may be bypassed using `mprotect()`

Bypassing DEP

- jump to existing executable code
- return-to-libc (call system)
- use return oriented programming
- for testing purposes, disable using `-z execstack` as argument to `ld`
- use `mprotect` to update page permissions

DEP Bypass Demo

- call `system("/bin/sh")`
- use `mprotect()` to force writable executable stack

Address Space Layout Randomization

- randomize address space, place code, libraries and stack and random addresses
- a buffer will use a different address each time the program is run

PIC and PIE

- Position Independent Code (in libraries)
- Position Independent Executable (in executable files)

Bypassing ASLR

- brute forcing (32 bit systems)
- use format string or other vulnerabilities to learn stack layout
- use huge NOP sled
- bug: use `ulimit -s unlimited` – the stack fills all available space
- for testing purposes, you may disable it

Disable ASLR

```
echo 0 | sudo tee /proc/sys/kernel/randomize_va_space
setarch $(uname -m) -R /bin/bash
```

ASLR Bypass Demo

- use brute force to bypass ASLR

Stack Guard/Stack Protection

- place a value between buffer and frame pointer/return value
- canary value
- in case of buffer overflow, value gets written and an exception handler is run

Canary Values

- may contain terminator characters (0x00, 0x0a, 0x0d, 0xff)
- may be a random string or a XOR function

Stack Guard Demo

- demo

Bypassing Canaries

- does not protect internal variables or buffer from another buffer
- you may use string formatting attacks to find the canary value and rewrite with itself
- in case the process forks, you may trigger multiple forks and then try guessing one byte at a time
- for test purposese, disable using `-fno-stack-protector` as argument to gcc

Outline

- ASCII-armored address spaces
- code integrity protection, data execution prevention
- address space layout randomization
- stack protection/stack guard, canary value

Other Runtime Attacks

- code reuse
  - return to libc
  - return oriented programming
- data oriented attacks: do not target the alteration of the control flow, but overwrite data

Keywords

- exploit
- buffer overflow
- shellcode
- input validation
- static analysis
- dynamic analysis
- code integrity
- DEP
- ASLR
- PIC, PIE
- canary value
- stack guard

References

- The Art of Exploitation, 2nd Edition
  - Chapter 0x600. Countermeasures

Useful Links

- http://security.stackexchange.com/questions/20497/stack-overflows-defeating-canaries-aslr-dep-nx
- http://security.stackexchange.com/questions/18556/how-do-aslr-and-dep-work
- http://www.phrack.org/issues.html?id=13&issue=67&mode=txt
- http://www.cs.umd.edu/~jkatz/security/s12/lecture22.ppt
- http://www.cs.bham.ac.uk/~covam/teaching/2012/secprog/10-more-defenses.pdf