Session 03

Exploiting. Part 1: Applications

Security of Information Systems (SIS)

Computer Science and Engineering Department

October 18, 2023

Attacking a System

1. steal (information leak, information disclosure)
2. control (access, privileges)
3. cripple (crash, denial of service, sabotage)

Paths to Controlling a System

- breaking authentication
- side channel attacks
- bypass checks (misconfigurations)
- exploit vulnerabilities

Breaking Authentication

- guess passwords (or other credentials)
- crack passwords (or other credentials)
- social engineering
- impersonate

Side Channel Attacks

- do not alter or attack system directly
- covert channel
- infer information (passwords, keys, messages) from error messages, power dissipation, electromagnetic signals, sounds etc.
- system-centric attack not application-centric attack: you may have a perfect app but a flawed system

Misconfigurations

- mostly wrong ACL checks
- restricted information is available
- caused by system complexity and/or programmer/designer/administrator lack of complete view of the system

Exploiting

- system/application has a vulnerability: can be used for attacker benefit
- unintended behavior (not known or not checked by designer)
- can get inside the system/application, can control the system/application
- issue created by the designer/developer of the application/system

Papers

- Smashing the Stack for Fun and Profit (Phrack Magazine)
- Beyond Stack Smashing: Recent Advances in Buffer Overrun Attacks (IEEE Security & Privacy 2004)

Attacking a System

- find a "way in": misconfiguration, exploit
- get as much power as possible (look for privilege escalation, go for complete privileges)
- extract information
- control the system
- hide presence
- make it persistent

Attack Vector

- steps for an attack
- do reconnaissance, do information leak, get access, escalate, make permanent
- different vulnerabilities or flaws are exploited in an exploit chain

Malware

- software with malicious intent
- it's implanted on the target system, it runs on the target system
- an exploit may be exploited remote or locally by malware
- a separate attack must be used to implant the malware

Types of Malware

- `http://www.malwaretruth.com/the-list-of-malware-types/`
- adware
- spyware
- virus
- worm
- trojan
- rootkit
- backdoor
- keylogger
- ransomware

System/Component Flows

- input → attack surface
- input processing by applications → input validation
- application uses internal control flow to process data
- flaws/vulnerabilities may appear inside the app, or in the component interaction (access control lists, configuration files, message passing)
- control flow vs. data flow

Application Exploiting

- vulnerability in app allows leak or control of app
- generally related to memory exploiting: memory disclosure, memory overwrite
- goals
  - critical data (read or overwrite)
  - code pointers (overwrite and alter control flow)

Buffer Overflow

- most basic vulnerability
- go past the buffer boundary and overwrite data
- look for code pointers: return address on stack, function pointers

Runtime Binary Application Attacks

- exploit running application
- identify vulnerability and corrupt memory
- generally aim to control the app, run arbitrary code, get shell
- ideal step is to get privileged access to the system

Attack Steps

- identify vulnerability (usually buffer overflow)
- determine offset from the start of the buffer to target to overwrite (usually a code pointer)
- determine value used to overwrite target (points to “useful” attacker code)
- craft payload
  1. initial padding (size offset)
  2. overwrite value
  3. possible other values (function arguments, code gadgets)
  4. possible initial shellcode
- inject payload in vulnerable application
- profit

Control-Flow Hijacking

- goal is to alter the control flow and take control of the program
- we can create new edges in the control flow graph: code reuse (ROP)
- we can add new vertices in the control flow graph: code injection (shellcode)
- CFI (*Control Flow Integrity*) is used to prevent control-flow highjacking: expensive

Data-Oriented Attacks

- overwrite data (no code pointers) and do not alter the control flow
- use existing/valid paths in the control flow to get control of the program or leak information
- Hong Hu, Shweta Shinde, Sendroiu Adrian, Zheng Leong Chua, Prateek Saxena, Zhenkai Liang: Data-Oriented Programming: On the Expressiveness of Non-Control Data Attacks, IEEE S&P 2016

Keywords

- system components
- exploit
- vulnerability
- malware
- attack vector
- attack surface
- input validation
- code pointer
- code reuse
- code injection
- shellcode
- Return Oriented Programming (ROP)
- Data Oriented Programming (DOP)
- control flow
- control flow hijacking
- control flow integrity