{P is for Potato}

Potato Academy

Preamble

- So that we can learn better with a more formalised approach
- More focussed on C/C++ code (which works for decompiled IDA code too!)
- https://www.amazon.com/Art-Software-Security-Assessment-Vulnerabilities/dp/0321444426

Vulnerability Research

- Attack surface analysis
- Audit strategies
- Bug classes
- Hooking
- Fuzzing
- Reverse engineering
- Kernel vulnerabilities

Attack Surface Analysis

Attack Surface Analysis

- Enumerate functions where input is collected and used
  - Input collection:
    - User input
    - External data used by the application (config, data files, registry keys etc)
  - Where data is used:
    - System commands (system(), eval()…)
    - Some form of processing (eg copying data from a file into a struct in memory)
  - Dependencies
    - What dependencies does it require that can introduce bugs?
      - Databases
      - Video/image processing libraries…
- Authenticated/Unauthenticated/ACL boundary
  - Any flaws that allow a user to access more than he should?
- https://cheatsheetseries.owasp.org/cheatsheets/Attack_Surface_Analysis_Cheat_Sheet.html

Attack Surface Analysis – Example

- What would be the attack surface for an app like Jira?
- What are the user inputs?
- What kind of processing does it do on uploaded files?
- Boundaries between authenticated users, public, different user roles

Attack Surface Analysis – Example

- What would be the attack surface for a router?
- What services are started by default and accessible?
- Input to web application, mobile application, additional services (afpd, ntp, samba…)
- How is authentication handled?

Audit Strategies

Audit Strategies

- Strategy 1 – Trace malicious input
  - Map out functions that handle input – eg: login(name,password), setDeviceName(name)
  - Very focussed on finding bugs regarding user input
  - Limited understanding of the entire program
  - How to enumerate functions that take in user input?
- Strategy 2 – Module based analysis
  - Analyse parts of the program – eg: util classes, login modules
  - Understand program's design but not directed at finding exploitable bugs (more for understanding the program)
  - Used by experienced code reviewers
- Strategy 3 – Lexical/Pattern search
  - Search for vulnerable calls – eg: memcpy, strcpy, free (double frees)
  - Search for type misuse – eg: signed int compared against unsigned int

Case Study: OpenSSH – Preassessment

- Scope: Can an unauthenticated user log into a system via SSH?
- Preassessment (attack surface analysis):
  - Enumerate types of users:
    - ~~Administrator – start/stop SSH server and modify configuration files~~
    - Authenticated users
    - Unauthenticated users
  - SSH protocol
  - Identify subsystems in OpenSSH code base
    - Buffer management
    - Packet management
    - Crypto
    - Privilege separation – OpenSSH forks 2 processes for each connection: an unprivileged child deals with network data and a privileged parent to authenticate users based on requests from child
  - Focus on code that is triggered by unauthenticated users

Case Study: OpenSSH – Implementation Analysis

- Lexical search:
  - Check for functions that can cause OOB – memcpy, strcpy, sprintf, free
  - Check for signed-ness comparisons and data types (eg int compared with size_t)
- Error conditions and clean-ups:
  - Can error clean-ups leave the program in an inconsistent state? Eg stale data
  - Are exceptions handled properly?
  - Are important function return values checked? Eg return of setuid() not checked
- Unexpected values:
  - Authentication success is treated as 1 or 0. Are there other functions that return other values?
    Eg
    func return_auth_state() { if (error) {return 2} else if (check_auth) {return 1} else {return 0} }
    ….
    auth_access = return_auth_state()
    if (auth_success) { let user in } ← this will pass if return_auth_state returns 1 or 2

Case Study: OpenSSH – Attack Vectors

- Protocol:
  - Sniffing
  - MITM
  - Protocol quirks – backwards compatibility, design issues
  - Protocol state – how does the server handle messages sent to it at unexpected states? Eg server expects an authentication request packet but gets a logout packet
- Login:
  - Brute force
  - Multistage authentication (OTP)
  - File based authentication
  - …

Thought Experiment – What is the strategy for auditing sudo?

- What is the scope? Privilege Escalation
- Determine the attack surface given the scope
- What are the attack vectors?

Thought Experiment – What is the strategy for auditing sudo?

- What is the scope? Privilege Escalation
- Determine the attack surface given the scope
  - Plugins
  - Environment variables
  - Configuration files
  - Command line arguments
- What are the attack vectors?
- Find previous bugs related to PE and see if the same pattern occurs
- Audit patches for bugs to see if new bugs are introduced or improper fixes