EC ENGR M117 – Computer System Security

Fall 2024 – Professor Yuan Tian

Lecture 1: Computer System Security

When should we trust computer systems?

- Ken Thompson: Creator of UNIX
    - There was a backdoor to the UNIX compiler

Why Own Client Machines?

1. IP Address and Bandwidth Stealing
    a. Click Fraud
    b. Denial of Service
    c. Spam
2. Steal User Credentials and Inject Ads
3. Ransomware
    a. Worm spreads via vulnerabilities in SMB
4. Spread to isolated Systems
    a. Stuxtnet
5. Bitcoin Mining

Server-side Attacks

- Financial Data Theft
    - Equifax
- Political Motivation
    - DNC Attack
    - Ukraine attacks (election, power grid)
- Infect Visiting Users

Server-side Breaches

- Typical attack steps (Kill chain)
    - Reconnaissance

    - Foothold
    - Lateral Reconnaissance
    - Lateral Movement
    - Data Extraction
    - Exfiltration

Mpack:

- PHP based tools installed on compromised websites
    - Infects browsers which visit site
    - Embedded as an iframe on infected page

Sysadmin for city of SF government

- Changed passwords

Marketplaces for Vulnerabilities:

- Bug bounty Programs
- Zero Day Initiatives
- Black Market

Marketplace for owned machines:

- Pay-per-Install (PPI) Services
    - Own victim's machine
    - Download and install client

- Vulnerabilities are inevitable
    - There will always be bugs
        - Qualys discovered heap overflow in sudo
    - Safe Languages have bugs
- Systems must be designed to be resilient in the face of both software vulnerabilies

<u>Security Principles</u>

- Defense in Depth

- Systems should be built with security protections at multiple layers
- If there is a bug in Chrome’s interpreter, CHome, OS, HW, and Network should have securities

- Principle of least Privilege
    - User should only have access to the data and resources needed to perform routine authorized tasks
- Privilege Separation
    - Dividing a system into parts which we can limit access
    - Segmentation prevents attackers from taking over a whole system
- Open Design
- Keep it Simple

Security Policies

- Subject (Who?)
- Object (What?)
- Operation (How?)

UNIX Security Model

- Subjects
    - Users, processes
    - User owner, group owner, other
- Objects
    - Files, directories
    - Sockets, pipes, HW devices
- Access Operations
    - Read, write, edit
- Users
    - Each user has unique ID
    - Service accounts
    - User Accounts
- Groups

- File Ownership
    - All files and directories have file owner and shared ownder
- Access Control:

ACLs: Identifies what operations subjects can perform

Role Based Access Control (RBAC)

UNIX Processes

- Processes are isolated
    - cannot access each other's memory
    - runs as a specific user
    - can access any files which UID has access to

Process User IDs

- Effective User ID
- Real User ID
- Saved User ID

SETUID Bit: Elevating Privileges

UNIX Privilege separation is not very good

- Many processes depend on setuid
- Privileged processes bypass all kernel permission checks

UNIX

- Pros
    - Simpel model that protects in most cases
    - Flexible to be used in many simple systems
- Cons
    - ACLs do not account for enterprise complexity
    - All system operations require root access

<u>Lecture 2: Windows Security Model</u>

- Flexible Access Control Lists (ACLs):
    - Objects have full ACLs
    - Users can be members of multiple groups
    - ACLs support Allow and Deny ules
- Object Security Descriptor:
    - Every object has a security descriptor
    - Contains:
        - Security Identifiers: For owner and primary group of object
        - Discretionary ACL: Access rights allowed for users or groups
            - Controls rights users have
            - Includes all accesses for object
        - System ACL: Generates audit reports
            - Tracks users that modify objects, sensitive behaviors
- Each process in Windows has <u>Tokens</u>/Security contexts
    - Windows checks if object has access using tokens
- Capabilities vs ACLs
    - ACL: System checks whether subject is on list of users with access to the object
    - Capabilities: Does not care who subjects are, just focuses on access
- Weak Protection on Desktops
    - Malicious Applications can run as you and access all files

Mac OS

- App Sandbox: Mediates access to HW, Network, App Data, User Files

Android:

- Android uses Linux and its own kernel application for isolation
- Each application runs with its own UID in its own VM
    - Reference monitor checks permissions on intercomponent communication
- First time users have large scale permissions

Chrome Security Models

- Architecture:
    - Past: One giant browser process
        - Issue: One tab can read the memory and cookies of other tabs
    - Modern: Division into components
        - Pro: More isolation when running different websites
        - Each iframe has a separate renderer process
    - Components
        - Render Process: Controls everything inside of a tab
        - Browser Process: Controls “chrome” part of application (address bar, etc)
        - GPU process: handles GPU requests
        - Broker (Main Browser): Privileged Controller of activities of the sandboxed processes
            - Renderer’s only access to network is via parent browser process
            - File system Access can be restricted
        - Plugin Process: Handles Plugins

Open Design

- Security of a mechanism should not depend on the secrecy of its design or implementation
    - If the secrets are abstracted from the mechanism, than the damage is individual

Browser Security Models:

- Goals
    - Safely browse the web
    - Support secure web apps
    - Support secure mobile apps
- Web Security Threat Model
    - Web Attacker sets up malicious website visited by victim
        - No control of network

- Network Security Threat Model
    - Network Attacker intercepts and control network communication
        - Modifies network packets
- Malware attacker
    - Attackers escape browser isolation mechanisms and run separately under the OS

URLs

- Global Identifiers of network-retrievable documents
- URL sends a HTTP Request
    - Set-cookie remembers the session
- Basic Browser Execution Model
    - Each browser window/frame:
        - Loads content
        - Renders it
        - Responds to events
    - Events can be:
        - User actions
        - Rendering
        - Timing
- Document Object Model (DOM)
    - Object-oriented interface used to read and write docs
        - Similar to an OS
        - Provides representation of this data structure
    - Includes Browser Object Model (BOM)
        - Window, document, frames()
    - Can change HTML

Rendering Content: Think like an attacker

- HTML Image Tags: A faulty website can spoof another website, hide its image, and communicate with other sites
- Frames

    - Frame: Rigid Division as part of a frameset
    - iFrame: Floating inline frame

Policy Goals:

- Safe to visit an evil website
    - In terms of the OS is not affected
- Safe to visit two pages at the same time
- Allow safe delegation

Browser security mechanism

- Each frame of a page has an origin
- Frame can access data on the frame with the same origin

Components of Browser Security Policy

- Frame-Frame relationships

Domain Relaxation:

- Cross-origin network requests: Allows set of domains access
- Cross-origin client side communication:

Cookies

- Used to store state on user's website
- Secured Cookies: Can only be sent by HTTPS
    - Provides confidentiality against network attackers
- httpOnly Cookies: Cookie sent over HTTPs, but not accessible to scripts

Frames and Frame Busting

- Embed HTML documents in other documents

<u>Lecture 3</u>

<u>Cross Site Request Forgery</u>

- Cookies are not a good way to secure websites
- CSRF based on stronger session management
    - Secret Token Embedded in Page
    - Referrer validation
- Browser sends authenticated cookie to server
    - Server responds with content (as cookies act somewhat like passwords)

Basic CSRF

- Suppose I have a tab with Gmail open
    - My cookies will be sent without my discretion
    - Website will accept my cookie as if it is my password
        - This assumes that the cookie persists for a period of time
- Attack server will send malicious code (forgery request) to run code on my browser
    - Browsers directly execute the code it receives

CSRF Defenses:

- Secret Validation Token
    - Ex: Secret Token just for website (most of the time a form)
    - Requests include a hard-to-guess secret
        - Randomly Generated is optimal
- Referrer Validation
    - HTTP Referrer header
    - Lenient Referrer Validation does not work if the Referrer is missing
    - Privacy Issues
        - Referrer may leak privacy sensitive information
- Custom HTTP Header

How to avoid CSRF:

- HTTPS sites

- Origin Header

Cross Site Scripting (XSS)

- Attacker's malicious code executed on the victim's browser
- When an attacker can inject scripting code into pages generated by a web application
- The Scenario:
    - Reflected XSS Attack
        - Attack Script is reflected back to the user as a part of a page from the victim site
            - Victim visits website and receives malicious link
            - Victim clicks on link, user input is echoed
            - Victim sends valuable data (via cookie)
    - Stored XSS
        - Attacker stores the malicious code in a resource managed by the web application (ex: Database)

Lecture 5: Browser Code Isolation and Content Security Policy

Modern Web Sites:

- Handles sensitive information

Same-Origin Policy

- Two websites can communicate with each other using postMessage API
  - It is assumed that there is a specific reason/call
- Servers interact with websites using JSON queries (network requests)

Limitations

- Libraries are included using tags
  - No isolation, may execute arbitrary code
- Developers may not modify policies
- Does not prevent information leaks

Browsing Context:

- Each frame is its own browsing context

HTML Sandbox:

- Restrict frame actions
  - Directive sandbox ensures iframe has unique origin and cannot execute JavaScript
  - Developer can control what permissions embedded features can have

Content Security Policy

- Use HTTP header to specify policy
- Whitelist instructing browser to only execute or render resources from specific sources
  - Prohibits inline scripts embedded in script tags

Cross Origin Resource Sharing (CORS)

- Relaxation of Same Origin Policy
  - With SOP, a.com can send HTTP request to b.com, but cannot read the response

    - With CORS, a.com can whitelist b.com to read cross-origin

Password-strength checker

- Confine the checker with COWL (Confinement with Origin Web Labels)
    - Express sensitivity of data (Checker context label must be as sensitive as password)

HTTPS

- Threat Model: Network Attacker
    - Controls network infrastructure

Quiz 1 Notecard

Lecture 1

Why Own Client Machines

- IP Address Stealing & Bandwidth Stealing
- Stealing User Credentials
- Ransomware
- Spread to isolated systems
- Bitcoin Mining

Marketplace for Vulnerabilities

- Bug Bounties
- Zero Day Initiatives
- Black Market
  - Pay-per-Install (PPI) Services

Security Principles

- Vulnerabilities are Inevitable
  - Every Safe Language Has Bugs
  - Systems must be designed to be resilient in both software vulnerabilities and malicious users
- Zero Trust
  - Verify explicitly
  - Least Privileged Access
  - Assume Breach

Principles of Secure Systems:

- Defense in Depth
  - Systems should be built with security protections at multiple layers
- Principle of Least Privilege
  - Users should only have access to the data and resources they need to perform routine tasks

- Privilege Separation
    - Dividing a system into parts to limit access
- Open Design
    - Security should not rely on secrecy
- Keep it Simple

Security Subjects

- Applies to Users, Processes, Apps, Domains, and Josts

Security Policies

- Subject
- Object
- Operation

UNIX Security Model

- Subjects (Who)
    - Users, processes
    - Each user has unique integer ID
        - 0 reserved for root
    - Groups are collections of users who share files
- Objects (What)
    - Files, directories
        - All files have single user owner and group owner
- Access Operations (How)
    - Read, Write, Execute

Access Control Lists (ACLs): Identifies what operations subjects can perform

Role Based Access Control (RBAC)

UNIX Processes:

- Processes are isolated

- Processes run as a specific user
- Process User IDs
  - Effective User ID: Determines permissions for process
  - Real User ID: Determines the user that started the process
  - Saved User ID: EUID prior to change

Reducing Privilege

- Root changes all UIDs to arbitrary values
- Unprivileged users can change EUID to RUID or SUID
- Setuid can give unrestricted server access

Elevating Privileges

- Become root user
- Setuid

UNIX

- Pros:
  - Provides common protections
  - Flexible
- Cons:
  - ACLs do not account for enterprise complexity
  - Cannot handle different applications wihtihn a single user account
  - All system operations require root access

Window Security Model

- Flexible ACLs
  - Supports Allow and Deny Rules
- Object Security Descriptors
  - Specifies who can perform hat and audit rules
    - SIDs
    - DACL

        - SACL
- Tokens: Security context
    - Impersonation tokens can be used temporarily yto adapt a different context
- Access Request
    - Windows checks if security context has access to the object
- Capabilities: Subject presents an unforgeable ticket that grants access to the object
    - System doesn't care who the subject is

Process Isolation

- Mac OSX Sandbox
- Android kernel application sandbox
    - Each app runs with own UID in own VM

Chrome Security

- Modern Chrome Architecture is more modular
- Chrome controls the search bar
    - Renderer Process controls anything inside of the tab
- Process based site isolation using iframes
- Broker (Main Program) has privilege and supervises activities of sandboxed processes

<u>Browser Security Model</u>

Goals of Web Security:

- Safely browse the web
- Support secure web apps
- Support secure mobile apps

Web Security Threat Model:

- Web Attacker sets up malicious site visited by the victim
- No control of the network
- Can obtain SSL/TLS certificates for their website

Network Security Threat Model:

- Network Attacker intercepts and controls network communication
- Wireless eavesdropper, or Evil router with DNS poisoning

Malware Attacker

- Attacker escapes browser isolation mechanism and runs separately under OS

HTTP

- URLs: Global identifiers of retrievable documents
- Requests: Get and Post
- Response: Cookies

Rendering Content:

- Basic Browser Execution Model
  - Each browser window loads content and renders it

Document Object Model (DOM)

- Object-oriented interface used to read and write docs

Isolation

- Windows may contain frames from different sources

Web Policy Goals

- Make it safe to visit an evil website
- Make it safe to visit two pages at the safe time
- Allow for safe delegation

Browser Security Mechanism

- Each frame of page has origin
- Frame can access data on frame with same origin
- Frame cannot access data associated with different origin

Cookies: Used to store state on user’s machine

- Used for user authentication, personalization, and user tracking

Slide Deck 3

CSRF: Attack site uses login cookie

- Defenses
    - Secret Validation Token: Requests include a hard-to-guess secret
- Referer Validation
    - Lenient: Does not work if the referrer is missing
    - Strict: Secure but sometimes not available
- Custom HTTP Header
    - Does not work across domains

Cross Site Scripting (XSS): Attacker's malicious code is executed on the victim's browser

- Reflected: Attack script is reflected back to the user as part of a page from the victim site
- Stored: Attacker stores malicious code in a resource managed by the web application

Defenses:

- Check inputs and outputs to client server
- Echo user input from server
- Input data validation: Remove/encode special characters
- Output filtering: Allow only safe commands, encode XHTML special chars

Slide Deck 4

How do we isolate code from different sources?

Same-origin Policy (SOP)

- Isolate content from different origins
- We cannot use DOM to communicate between frames, but we can still send postMessages
- Cant access documents of cross origin pages
- Cannot inspect cross origin responses

SOP Limitations:

- Libraries: Runs in same frame and origin as rest of page
- Does not prevent information leaks
- Cannot relax policy
- Cross Origin scripts run with privilege within page

Browsing Context:

- Frame with its own DOM
- Thread which does not have a DOM
- Consists of:
  - Has origin determined by protocol/host
  - Is isolated from others via SOP
  - Can communicate via postMessage
  - And make network messages using XHR

Restricting execution:

- HTML5 iframe Sandbox: Restrict frame actions
- CSP: (Whitelist instructing browser)

Cross Origin Resource Sharing (CORS): Method that allows a whitelist for cross origin requests

- Issue: A website cannot visit a neighboring website without violating SOP

<u>Slide Deck 5</u>

Threat Model: Network Attacker

- Controls network infrastructure

Quiz 2 Notes

Format String Attacks

Bug: Real execution deviates from expected behavior

Exploit: Input that gives attacker advantage

Control Hijacking: Take over target machine to execute arbitrary code

- Buffer Overflow
- Format String

Execution/Control Flow:

- Processor fetches, decodes, and executes instruction
- Processer reads and writes onto stack and heap

Buffer Overflow:

- Data is written outside of the space allocated for the buffer

Non-Variadic functions:

- Compiler knows number and types of arguemnts
- Emits instructions for caller to push arguents left and right

Control Flow Hijack Defenses (Canaries, DEP, ASLR)

StackGuard: Canary word between retrun address and local vars

- Epilogue checks canary before function returns
- Canary should be hard to forge (randomized)

Bypass: Data Pointer Subterfuge:

- Overwrite a data pointer first

Data Execution Prevention (DEP):

- Mark Stack as non-executable using NX bit
- Make memory page either exclusively writable or executable

Ret2libc Attack

- Overwrite return address by address of a libc function
  - Setup a fake return address

ASLR

- Randomize addresses of program memory
- Can be defeated using brute force

Control Flow Integrity

- Protects against powerful adversary
- Widely-Applicible
- Provably-correct and Trustworthy

CFI Model:

- We can:
  - Overwrite any data memory at any time
  - Overwrite registers in current context
- We cannot:
  - Execute Data
  - Modify Code
  - Write to %ip
  - Overwrite registers in other contexts

Control Flow Graph:

- Each vertex is a basic block
- Edges represent potential transfers between connected blocks

Call Graph

- Nodes are functions
- Edges are function calls

Super Graph

- Superimpose CFGs of all procedures over the call graph

Context Sensitive: Different calling contexts are distinguished

Flow Sensitive: Considers order/flow of statements

- Insensitive means linear algorithms

Path Sensitive: Maintains branch conditions along each execution path

- Insensitive means local variables are not global

Fuzzing

- Blackbox fuzzing
    - Feed random inputs in and see if it exhibits incorrect behavior
        - Inefficient
- Fuzzing
    - Automatically generate test cases (random)
    - Application is monitored for errors
- Regression
    - Run valid inputs, look for errors

Mutation-Based Fuzzing

- Take well formed input, randomly perturb (flip bit, etc)
- Not efficient for protocols which use checksums

Generation-Based Fuzzing

- Test cases are generated from some description of the input format
- Better for completeness, but hard to create

Code coverage

- Measures how many paths, branches, and lines have been taken
- Can ensure initial file
- Does not Guarantee that the bug is found

Coverage Guided Gray Box Fuzzing

- Run mutated inputs on instrumented program and measure code coverage
    - Use genetic algorithms

Dataflow-Guided Fuzzing

- Intercept data flow, analyze the inputs of comparisons