Web Security - Lecture Notes

Overview

1. Malware Recap

Trojan Horses

- Programs that appear useful but contain hidden malicious code. Three operational models:
- Perform original function + separate malicious activity
- Perform the original function but modify it maliciously
- Completely replace the original function with a malicious one

Malware Payloads

System Corruption:

- Data destruction triggered by specific conditions
- Ransomware: encrypts data and demands payment
- BIOS rewriting attacks
- Logic bombs: code that "explodes" when conditions are met

Attack Agents:

- Bots/zombies/drones: compromised computers used for attacks
- Botnets: coordinated collections of bots
- Uses: DDoS attacks, spamming, keylogging, malware spreading

Remote Control:

- Distinguishes bots from worms (worms self-propagate; bots are controlled)
- Implemented via IRC servers, HTTP, or P2P protocols
- Allows attackers to activate bots and issue update commands

Information Theft:

- Keyloggers: capture keystrokes for credentials
- Spyware: monitors wide range of system activity
- Phishing: masquerades as trusted communication
- Spear-phishing: targeted, personalized attacks

Stealthing:

- Backdoors/trapdoors: secret entry points bypassing security
- Rootkits: programs maintaining covert admin access while hiding presence
- Types: Persistent, Memory-based, User mode, Kernel mode, Virtual machine-based, External mode
- Countermeasures

Prevention Elements:

- Keep systems updated with all patches
- Set appropriate access controls
- User awareness and training against social engineering
- Threat Mitigation (when prevention fails):
- Detection, Identification, Removal
- Requirements: Generality, Timeliness, Resiliency, Minimal denial-of-service costs, Transparency, Global and local coverage

Antivirus Generations:

- Simple scanners (signature-based)
- Heuristic scanners (rule-based probability)
- Activity traps (behaviour-based, memory-resident)
- Full-feature protection (multiple techniques combined)

Additional Approaches:

- Host-based behaviour-blocking software
- Perimeter scanning (ingress/egress monitors)
- Worm countermeasures (6 classes: signature filtering, filter-based containment, payload classification, TRW scan detection, rate limiting, rate halting)

2. Web Security Fundamentals

What is Web Security?

- Protecting websites and web applications from cyber threats while ensuring:
- Confidentiality
- Integrity
- Availability of data and services
- Core Principles
- Authentication & Authorization: Verify identity and control access
- Input Validation: Prevent malicious data entry
- Secure Communication: Use HTTPS and encryption
- Session Management: Protect user sessions from hijacking

The Web Security Model

Three Components:

- Client: Browser (Chrome, Firefox) - fetches and renders content
- Server: Web server (Apache, Nginx) + Application/Database server - processes requests
- Protocol: HTTP/HTTPS - stateless (root of many security challenges)

Key Concepts:

- Sessions & Cookies: Solve statelessness by storing session identifiers
- Same-Origin Policy (SOP): Browser security rule preventing documents from one origin from interacting with resources from another origin

3. OWASP (Open Web Application Security Project)

What is OWASP?

Global non-profit focused on improving software security through free, open-source tools, documentation, and community resources.

Key Security Principles

- Security by Design: Integrate security from development start
- Security by Default: Most secure default settings
- Defence in Depth: Multiple security control layers
- Least Privilege: Grant only necessary access
- Fail Safe Defaults: Deny access unless explicitly allowed
- Complete Mediation: Check permissions on every access
- Psychological Acceptability: User-friendly security

Popular OWASP Resources

- OWASP Top 10 (critical web app security risks)
- OWASP ZAP (vulnerability scanner)
- ASVS (Application Security Verification Standard)
- Cheat Sheet Series (secure coding best practices)

OWASP Top 10 Risks

- Broken Access Control: Inadequate user permission enforcement
- Cryptographic Failures: Weak/misused encryption
- Injection: Malicious input altering app behaviour
- Insecure Design: Flawed architecture
- Security Misconfiguration: Default settings, unpatched systems
- Vulnerable & Outdated Components: Libraries with known vulnerabilities
- Identification & Authentication Failures: Weak login mechanisms
- Software & Data Integrity Failures: Unverified updates
- Security Logging & Monitoring Failures: Insufficient detection
- Server-Side Request Forgery (SSRF): Unauthorized server requests

4. Attacking the Server-Side (Backend)

Zero Trust Input Principle

- "Never, ever trust user input"
- All data from users is untrusted and potentially malicious
- Sources: form fields, URL parameters, cookies, HTTP headers, uploaded files
- Attackers control client-side and can bypass frontend checks
- Input validation is the first line of defence

Example: For a "last name" field, limit character types (hyphens, quotes, ASCII vs Unicode) and length

Injection Flaws

- Concept: Sending malicious code where only data is expected

- SQL Injection (SQLi) Example:
- User enters: ' OR '1'='1 in password field
- Vulnerable query: SELECT * FROM users WHERE user = '[username]' AND pass = '[password]';
- This becomes: SELECT * FROM users WHERE user = '' OR '1'='1' AND pass = '' OR '1'='1';
- Primary Defence: Parameterized Queries (Prepared Statements)
- Database compiles command first
- User data inserted into specific placeholders
- Malicious data never executed as code

Broken Access Control

- Concept: Failure to verify authorization after authentication
- Insecure Direct Object Reference (IDOR) Example:
- User sees own profile: .../view_profile.php?user_id=500
- Attacker changes URL to: .../view_profile.php?user_id=501
- Views someone else's profile without permission
- Primary Defence: Server-side authorization checks on every request
- Verify logged-in user (from session) has permission for requested resource

5. Attacking the Client-Side (Browser)

Theme

- "The server may be secure, but can it be tricked into attacking its own users?"
- Cross-Site Scripting (XSS)
- Concept: Injecting malicious client-side script (JavaScript) into pages viewed by victims

Types:

- Stored XSS (Most Dangerous):
- Malicious script saved on server (e.g., in comments, user profiles)
- Attacks everyone who views that page
- Reflected XSS:
- Malicious script in URL (e.g., search query)
- "Reflected" back to the user
- Attacks the user tricked into clicking link
- Primary Defence: Context-Aware Output Encoding
- Neutralize user-supplied data before displaying
- Example: > becomes >, < becomes <
- Browser displays text but doesn't execute as code

Context-Specific Encoding:

- HTML Body: Encode <, >, &, ", '
- HTML Attribute: Encode quotes and escape characters
- JavaScript Context: Escape quotes, backslashes, control characters
- URL Context: Use percent-encoding (e.g., %20 for space)

Best Practices:

- Use libraries (OWASP Java Encoder, ESAPI)
- Never mix encoding with input validation
- Combine with CSP (Content Security Policy)
- Cross-Site Request Forgery (CSRF)
- Concept: Tricking authenticated user's browser into sending unintended requests

How It Works:

- Victim logged into mybank.com (using session cookies)
- Victim visits evil.com
- evil.com contains: <img src="http://mybank.com/transfer?to=attacker&amount=1000">
- Browser automatically attaches mybank.com session cookie
- Bank thinks the victim authorized the transfer
- Primary Defence: Anti-CSRF Tokens
- Server embeds a unique, secret, hidden token in every form
- When form is submitted, server validates the token
- Attacker cannot guess the token, so forged request fails

Key Takeaways

- Never trust user input - validate and sanitize everything
- Use parameterized queries to prevent SQL injection
- Implement server-side authorization checks on every request
- Encode all output based on context to prevent XSS
- Use anti-CSRF tokens for state-changing operations
- Apply defence in depth - multiple security layers
- Follow OWASP guidelines and use their resources
- Keep systems patched and updated
- Implement proper logging and monitoring
- Security must be built in from the start (Security by Design)