Lecture 5 – Web Security

Networks and System Security

Trojan Horses

- Useful-seeming programs containing hidden malicious code
- Three models: continue original function + malicious activity, modify original function, or completely replace function

Malware Payloads

System Corruption

- Data destruction, unwanted displays, ransomware (encrypt data and demand payment)
- BIOS attacks, logic bombs (code that "explodes" when conditions are met)

Attack Agent

- Bots/zombies/drones: secretly take over computers to launch attacks
- Botnets: coordinated collections of bots
- Uses: DDoS attacks, spamming, keylogging, malware spreading, manipulating polls

Remote Control

- Distinguishes bots from worms (worms self-propagate, bots are centrally controlled)
- IRC servers, HTTP protocols, or peer-to-peer for distributed control

Information Theft

- Keyloggers: capture keystrokes for credentials
- Spyware: monitor wide range of system activity
- Phishing: masquerade as trusted sources
- Spear-phishing: targeted, researched attacks on specific recipients

Stealthing

- Backdoors/trapdoors: secret entry points bypassing security

Goldsmiths

UNIVERSITY OF LONDON

- Rootkits: maintain covert admin access while hiding presence

Rootkit Types

- Persistent (activates on boot)
- Memory-based (doesn't survive reboot)
- User mode (intercepts API calls)
- Kernel mode (intercepts native APIs)
- Virtual machine based (runs OS in VM)
- External mode (in BIOS, directly accesses hardware)

Countermeasures

Prevention Elements

1. Keep systems patched and current
2. Set appropriate access controls
3. User awareness and training

Detection Approaches

- Four generations of antivirus: simple scanners, heuristic scanners, activity traps, full-feature protection
- Host-based behavior-blocking: monitors program behavior in real-time
- Perimeter scanning: ingress monitors (at network border), egress monitors (catch malware sources)

Worm Defense Classes

- Class A: Signature-based filtering
- Class B: Filter-based containment
- Class C: Payload classification
- Class D: Threshold random walk scan detection
- Class E: Rate limiting
- Class F: Rate halting

Lecture 5 – Web Security

Web Security Fundamentals

Core Principles

- Authentication & Authorization: verify identity, control access
- Input Validation: prevent malicious data entry
- Secure Communication: HTTPS and encryption
- Session Management: protect sessions from hijacking

Web Security Model Components

- **Client**: Browser (fetches and renders content)
- **Server**: Web/Application/Database servers (process requests, serve content)
- **Protocol**: HTTP/HTTPS (stateless, root of many security challenges)

Key Concepts

- **Sessions & Cookies**: Fix statelessness using cookies as session identifiers (user's "ticket")
- **Same-Origin Policy (SOP)**: Browser security rule preventing documents from one origin interacting with resources from another origin

OWASP (Open Web Application Security Project)

Key Principles

- Security by Design: integrate from development start
- Security by Default: most secure default settings
- Defense in Depth: multiple security layers
- Least Privilege: grant only necessary access
- Fail Safe Defaults: deny unless explicitly allowed

- Complete Mediation: check permissions on every access
- Psychological Acceptability: user-friendly security

OWASP Top 10 Critical Risks

1. **Broken Access Control**: inadequate user permission enforcement
2. **Cryptographic Failures**: weak/misused encryption
3. **Injection**: malicious input altering app behavior
4. **Insecure Design**: flawed architecture
5. **Security Misconfiguration**: defaults, unpatched systems
6. **Vulnerable & Outdated Components**: libraries with known vulnerabilities
7. **Identification & Authentication Failures**: weak login, session issues
8. **Software & Data Integrity Failures**: unverified updates
9. **Security Logging & Monitoring Failures**: insufficient detection
10. **Server-Side Request Forgery (SSRF)**: unauthorized server requests

---

Attacking the Server-Side

Zero Trust Input Principle

"Never, ever trust user input"

- All user data (form fields, URL parameters, cookies, headers, files) is untrusted and potentially malicious
- Attackers control client-side and can bypass frontend checks
- Input validation is first line of defense

Injection Flaws (SQL Injection)

Problem: Sending malicious code in fields expecting data

Example: User enters ' OR '1'='1 in password field

Vulnerable Code:

Goldsmiths

UNIVERSITY OF LONDON

Lecture 5 – Web Security

```
SELECT * FROM users WHERE user = '[username]' AND pass = '[password]'
```

Primary Defense: Parameterized Queries (Prepared Statements)

- Database compiles query first, then inserts user data into placeholders
- Malicious data never executed as code

Broken Access Control (IDOR)

Problem: Application fails to verify authorization after authentication

Example: User at /view_profile.php?user_id=500 changes to user_id=501 and accesses another user's profile

Primary Defense: Server-side authorization checks on every request

- Server must verify: "Does logged-in user have permission to access this resource?"

Attacking the Client-Side

Cross-Site Scripting (XSS)

Concept: Injecting malicious JavaScript into pages viewed by victims

Types:

1. **Stored XSS** (most dangerous): Script saved on server (e.g., in comments), attacks everyone viewing the page
2. **Reflected XSS**: Script in URL/link, attacks users clicking malicious links

Primary Defense: Context-Aware Output Encoding

- Neutralize user data before displaying in HTML
- < becomes &lt;, > becomes &gt;
- Browser displays text but doesn't execute as code

Encoding by Context:

- HTML Body: encode <, >, &, ", '
- HTML Attribute: encode quotes and escape characters

- JavaScript Context: escape quotes, backslashes, control characters
- URL Context: percent-encoding (e.g., %20 for space)

Best Practices:

- Use libraries like OWASP Java Encoder or ESAPI
- Never mix encoding with input validation
- Combine with Content Security Policy (CSP)

Cross-Site Request Forgery (CSRF)

Problem: Tricking authenticated user's browser into sending unintended requests

Example Attack:

1. Victim logged into mybank.com (using session cookies)
2. Victim visits evil.com
3. evil.com contains: <img src="http://mybank.com/transfer?to=attacker&amount=1000">
4. Browser automatically attaches mybank.com session cookie
5. Bank thinks victim authorized the transfer

Primary Defense: Anti-CSRF Tokens

- Server embeds unique, secret token in every form
- Server validates token on submission
- Attacker on evil.com cannot guess the token, so forged requests fail

---

Key Takeaways

1. **Never trust user input** - validate and sanitize everything
2. **Use parameterized queries** to prevent SQL injection
3. **Check authorization server-side** on every request
4. **Encode output** contextually to prevent XSS

Lecture 5 – Web Security

5. **Use anti-CSRF tokens** to prevent request forgery
6. **Apply defense in depth** - multiple security layers
7. **Keep systems patched** and use secure defaults