Software and System Security 2 - S8 FS25

Raphael Nambiar

Version: 16. Juni 2025

SECURING INFORMATION SYSTEMS

Information System

Definition: Structured set of components to collect, process, store, communicate information

- Applications, services, IT assets
- Software, hardware
- Data, methods, procedures
- People (users, operators)

Information Security Management System (ISMS)

Definition: Structured approach to manage information security

- Risk management framework
- Includes: people, processes, technology
- Goal: keep risks at acceptable levels
- Implemented by management (typically CISO)
- Checklist-style, high abstraction
- Not a technical solution

Security Controls

Definition: Countermeasures to reduce, detect, respond to risks

- **Types:**
  - Preventive - stop incidents (e.g., firewalls, auth)
  - Detective - identify incidents (e.g., IDS)
  - Corrective - limit damage (e.g., backups)
- **Attributes:**
  - Security Property: CIA
  - Function: Identify, Protect, Detect, Respond, Recover
  - Category: People, Physical, Technology, Organizational

ISO 27000 Series

ISO 27001: Lists high-level controls (e.g., disposal, network security)
ISO 27002: Implementation guidance for ISO 27001 controls

- Abstract, generic - industry-independent
- Checklist-like reference
- Example: Malware protection - anti-virus, user training

CIS Controls

Best-practice guidelines whose development started in 2008
Definition: Practical, prioritized controls from real-world attacks

- **Groups:**
  - IG1 - Basic hygiene (SMEs)
  - IG2 - Mid-level, enterprise-grade
  - IG3 - Advanced protection, targeted threats
- **Examples:**
  - CSC 1 - Inventory of devices (active + passive)
  - CSC 2 - Inventory of software (whitelisting)
  - CSC 7 - Continuous vuln. management (scanners, patching)

Measuring Security

Challenge: Measuring security = hard / approximate

- **Methods:**
  - Audits (compliance vs. standards)
  - Penetration testing
  - Risk = Likelihood × Impact
- **Metrics:**
  - % vulnerabilities patched in time (NIST SP 800-55)
  - Ratio blocked/successful malware (ISO 27004)
- **Purpose:**
  - Assess control effectiveness
  - Demonstrate compliance
  - Guide security decisions

Key Takeaways

- Securing systems = people + process + tech
- ISMS / CIS = frameworks, not full solutions
- Controls must be context-specific + prioritized
- Measuring helps track + improve security posture

Threat Landscape

Definition

Definition: Collection of threats in a domain/context

- Focus: Threat types, agents, vectors (not mitigations)
- Supports risk evaluation:
  - Risk = Threat × Vulnerability × Consequence
  - Risk = Likelihood × Impact

Threat Agents

Attributes: Motivation, Resources, Skill, Role

- **Cyber Criminals:** money,secrets, medium-high skill/resources, *-as-a-Service
- **Online Social Hackers:** High social, low-medium tech skill, psychology-based attacks
- **Cyber Spies:** State/corp, espionage, very high skill/resources
- **Employees:** Insider threat, low-medium skill, intentional/unintentional
- **Script Kiddies:** Low skill, use public tools, motive: fun/fame
- **Others:**
  - Hacktivists - political/societal goals
  - Cyber Fighters - nationalists (non-state)
  - Cyber Terrorists - fear/political damage

Cyber Kill Chain

7 Steps of an Attack:

1. Reconnaissance - gather info
2. Weaponization - create exploit + payload
3. Delivery - transmit payload (email, USB...)
4. Exploitation - trigger vuln.
5. Installation - install malware
6. Command & Control - remote channel
7. Actions on Objectives - data theft, damage

Defenders can break the chain at any step.

Security Controls & SIEM

Fundamental Control Principles

- **Least Privilege** – minimum necessary access
- **Fail-Safe Defaults** – deny by default
- **Complete Mediation** – every access checked
- **Separation of Privilege** – multiple conditions for access
- **Least Common Mechanism** – minimize shared components
- **Open Design** – transparency over obscurity
- **Psychological Acceptability** – usability of security
- **Goal:** reduce attack surface, enforce secure defaults

SIEM Overview

Definition: SIEM = Security Information & Event Management

- Collects, normalizes, stores, correlates, and analyzes security data
- Central component of SOC (Security Operations Center)
- Supports detection, alerting, forensic analysis
- Dashboards, queries, incident timelines

SIEM Components

- **Sensors:** Sources that generate security-relevant data for the SIEM
  - **NIDS (Network Intrusion Detection System):** Monitors network traffic for anomalies (e.g., Snort, Suricata)
  - **HIDS (Host Intrusion Detection System):** Monitors system-level activity like file access, login attempts (e.g., OSSEC)
- **Log Collection & Normalization:**

- Collect logs from various sources (firewalls, servers, applications)
- Normalize into a common structured format (fields: timestamp, source IP, event type, etc.)
- Enables correlation and efficient querying

- **Asset Inventory:**
  - List of known systems, owners, IPs, roles, and criticality
  - Provides essential context for alerts and triage
  - Supports prioritization of incidents and reduces false positives
- **Vulnerability Scanner:**
  - Scans systems for known weaknesses (CVEs – Common Vulnerabilities and Exposures)
  - Tools: Nessus, OpenVAS
  - Results feed into SIEM to help prioritize alerts
- **Correlation Engine:**
  - Central logic unit that links related events to detect complex attacks
  - *Simple rule:* 5 failed logins → brute force detection
  - *Complex rule:* new login location + privilege change + file access = suspicious behavior
  - Enables detection of attacker TTPs (Tactics, Techniques, Procedures)

Pyramid of Pain

- Defense model: higher levels = harder for attacker to adapt
- Indicators (low to high): Hashes, IPs, Domains, TTPs
- Goal: detect & disrupt attacker TTPs, not just IOCs

SIEM Lab Summary

Will not be tested in the exam.

Security Testing (Part 1)

Security Testing Methods

Purpose: Identify, assess, and improve security posture
Methods:

- **Vulnerability Scanning** – Automated tools for known vulns (e.g., OpenVAS, LGTM)
- **Penetration Testing** – Manual & tool-assisted attack simulation to find & prove risks
- **Red Teaming** – Simulate real attackers to test detection/response across all layers
- **Purple Teaming** – Red & Blue collaboration to improve detection & response
- **Breach & Attack Simulation (BAS)** – Automated, scripted attack scenarios (e.g., MITRE ATT&CK)
- **Bug Bounty** – Crowdsourced testing (public/private), pay-per-find

Comparison:

- **Scanning:** Known vulns in 3rd-party apps/infrastructure
- **Pentesting:** Custom/web apps, focused scope
- **Red Team:** Test defenses (SOC), full attack paths
- **Purple/BAS:** Improve detection, develop new rules
- **Bug Bounty:** Live targets, continuous findings, public feedback

Penetration Testing

Definition:

- Simulated attack to discover exploitable vulnerabilities and evaluate risk
- NIST: Mimic real-world attacks to bypass security mechanisms

Motivations:

- Uncover weaknesses missed by automated tools
- Validate defense mechanisms & configurations
- Raise awareness, justify security budgets
- Fulfill compliance (e.g., PCI-DSS, HIPAA)

Scope Targets:

- IT Assets – Web apps, networks, infrastructure
- Data – Customer info, credentials
- Physical – Building entry
- Social – Phishing, manipulation

Success Factors: Skills, creativity, tools, lateral thinking

Penetration Testing Methodologies

- **OSSTMM** – Full-spectrum testing, formalized scoring model
- **OWASP Testing Guide** – Web app testing procedures & tools
- **NIST SP 800-115** – General framework, tools, validation
- **PTES** – Practical industry guide (incomplete/outdated)

Other resources: SANS checklists, training materials

Pentest Phases

1. **Pre-engagement** – Define scope, methods, rules, contacts
2. **Intelligence Gathering** – Collect public/recon info
3. **Threat Modeling** – Map potential attack paths
4. **Vulnerability Analysis** – Identify exploitable issues
5. **Exploitation** – Gain access or demonstrate impact
6. **Post-Exploitation** – Lateral movement, persistence
7. **Reporting** – Document findings, risk, mitigation

Pre-Engagement Phase

Scope:

- What systems, techniques, and depth of testing
- Channels: physical, human, network, wireless, telecom
- Define inclusions/exclusions (e.g., äll except billing module")

Rules of Engagement:

- Test windows (e.g., 20:00–06:00), backup constraints
- Use of stealth/evasion (depends on method: black/gray/white box)
- Evidence handling – encrypted, need-to-know access
- **Permission to Test Document** – mandatory, defines scope, 3rd party authorization, liability

Communication:

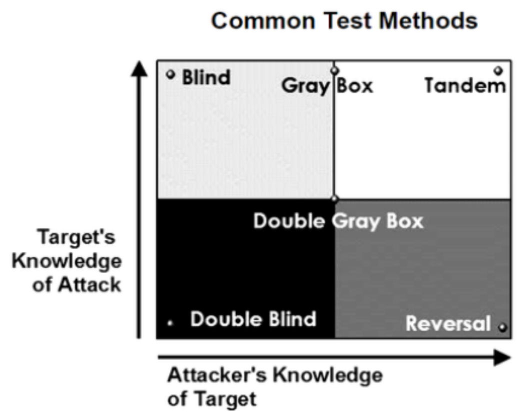
- Define secure channels (e.g., file sharing, IM, phone)
- Emergency contacts for incident handling
- Frequency of status reporting

Pitfalls:

- Clients unclear on purpose (real risk vs checkbox)
- Scope Creep – informal extensions must be managed properly

Common Test Models (OSSTMM)

- **Blind** – Testers get no info (like attackers)
- **Double Blind** – Even defenders don't know
- **Gray Box** – Limited internal info shared
- **White Box** – Full internal info shared
- **Crystal Box / Tandem** – Collaboration with client
- **Reversal / Red Teaming** – Realistic adversary simulation



Evidence Handling

- Avoid storing PII/PHI unless necessary
- Prove access via: screenshots, permission lists, flags
- All data must be encrypted and access-limited

Testing Method Comparison

0.0.1 Vulnerability Scanning

- **What:** Automated scanning for known vulnerabilities using signatures
- **Purpose:** Identify known vulnerabilities early
- **Compliance:** GDPR, HIPAA, PCI-DSS
- **Assets:** Source code, applications, infrastructure
- **Result:** List of potential vulns + risk rating
- **Method:** Tools like OpenVAS, LGTM; fully automated
- **Requirement:** Vulnerability mgmt. capability (triage + patching)
- **Frequency:** Continuous (due to changing signatures and assets)

0.0.2 Classical Penetration Testing

- **What:** Ethical hacking to discover and verify vulnerabilities
- **Purpose:** Find easy-to-moderate vulns + remediation advice
- **Assets:** Limited scope (app, service, system)
- **Result:** Verified vulns, risk scores, how-to-fix
- **Method:** Manual + tools (OWASP Testing Guide)
- **Requirement:** Test environment + vuln mgmt.
- **Frequency:** 1–4×/year or per release cycle

0.0.3 Red Team Testing

- **What:** Realistic attack simulation to test detection/response
- **Purpose:** Measure SOC effectiveness and incident handling
- **Assets:** Broad – physical, human, cyber layers
- **Goal:** Achieve mission (e.g., steal data) without detection
- **Result:** Goal outcome + detailed attack path
- **Method:** Custom attack scenarios (may include social engineering)
- **Requirement:** Mature security org (IR, SOC, controls)
- **Frequency:** Periodic (e.g., annually)

0.0.4 Purple Team Testing

- **What:** Red + Blue collaboration for better detection/prevention
- **Purpose:** Improve SOC rules, detection logic, tuning
- **Assets:** Selected systems, employee targets
- **Result:** Improved detection rules, hardening plans
- **Method:** Controlled attack simulation + feedback loop
- **Requirement:** Cross-team collaboration
- **Frequency:** Periodic (e.g., quarterly)

0.0.5 Breach & Attack Simulation (BAS)

- **What:** Continuous, automated kill chain simulation
- **Purpose:** Evaluate SOC resilience using known attack paths
- **Assets:** Based on selected attack scripts (e.g., MITRE ATT&CK)
- **Result:** Summary of detection/resistance to scripted attacks
- **Method:** Automated platforms (SaaS)
- **Requirement:** Like Purple Team, but with budget for automation
- **Frequency:** Continuous

0.0.6 Bug Bounty Programs

- **What:** Crowdsourced vulnerability testing
- **Purpose:** Discover real-world vulnerabilities
- **Type:** Public (anyone) or Private (invite-only)
- **Assets:** Mostly apps/services with clear rules
- **Result:** Vulnerability reports with PoCs
- **Method:** According to platform rules (HackerOne, Bugcrowd, etc.)
- **Requirement:** Legal setup + risk acceptance
- **Frequency:** Continuous

Penetration Testing (Part II)

Intelligence Gathering

Goal: Collect relevant information from public sources to aid attacks
Types:

- Physical – maps, building layout
- Logical – org charts, partners
- Infrastructure – IPs, domains, hosts
- Documents – metadata, open data leaks
- HUMINT – staff info, social profiles

Levels:

- L1: Automated (compliance-focused)
- L2: Tools + manual (best-practice)
- L3: Manual, stealthy, social-focused (APT-style)

Techniques:

- Passive – undetectable (e.g., Shodan, WHOIS)
- Semi-passive – DNS queries, public info
- Active – detectable (e.g., scanning)

Recon Techniques & Tools

Website Analysis: Org data, staff, emails
Google Dorking:

- Operators: `inurl:`, `intitle:`, `ext:`
- Tools: GHDB, ExploitDB

Domain/IP Discovery:

- WHOIS, SAN certs, Robtex, FindSubdomains
- DNS Tools: `dig`, `nslookup`
- RIR lookup, BGP Toolkit

Passive Tools: Shodan, Censys, Maltego

Scanning

Purpose: Map attack surface – find hosts, ports, services
Nmap:

- `-sS`: SYN scan (stealth)
- `-sT`: TCP connect
- `-sU`: UDP scan
- `-sV`, `-O`, `-A`, `-p-`
- NSE scripts: `--script=banner`, etc.

Network Tools: `traceroute`, `hping3`, `telnet`, `nc`, `openssl`

Footprinting Defenses & HUMINT

Identify: Firewalls, WAFs, IDS

- Tools: Nmap scripts, banner fingerprinting
- Techniques: Packet crafting, evasion, SE

Human Intelligence:

- Social media analysis, username lookup (Knowem, etc.)
- Pretexting, phishing, physical visits

Penetration Testing (Part III)

Threat Modeling

Purpose: Identify vulnerabilities by analyzing system designs and attacker goals.

- **Attacker-Centric:** Map how attackers move from entry points to target assets.

- **Defender-Centric:** Map organizational defenses and simulate attack paths avoiding them.
- **Techniques:** STRIDE, Attack Trees
- **Assets:**
  - Primary: Within test scope (e.g., CRM frontend)
  - Secondary: Outside scope but shared (e.g., employee DB on same server)
- **Threat Relevance:** Secondary assets may alter attacker models (e.g., insiders become relevant).

Attack Patterns and Frameworks

- **CAPEC:** Focused on application-level attacks and training
- **MITRE ATT&CK:** Real-world adversarial behavior, red-team and defense-oriented
- CAPEC and ATT&CK are complementary and cross-referenced

Vulnerability Analysis

Goal: Discover and confirm security issues that can be exploited.
Techniques:

- Scanners: Nmap, Nessus, GVM, sqlmap, XSStrike
- Source code scanners, manual analysis (e.g., CIS Benchmarks)
- Web scanners: Crawl and test input points
- Active fuzzing: E.g., American Fuzzy Lop
- Track findings with attack trees to avoid redundant work

Challenges:

- **False Positives:** Patched systems not reflected in version info
- **False Negatives:** Backported fixes not updating version number
- **Environment Dependent:** Network position, authentication, etc.

Exploitation

Goal: Gain access by leveraging vulnerabilities.
Methods:

- Exploits: SQL injection, buffer overflows, MitM, USB, social engineering
- Select vector based on success/detection probability
- Consider mitigation bypass: DEP, ASLR, AV, WAF

Expertise Levels:

- **Basic:** Use public exploits
- **Advanced:** Modify/tune exploits and payloads
- **Expert:** Discover new vulnerabilities (zero-days), reverse engineering

Post Exploitation

Goal: Assess value of access and maintain control (e.g., lateral movement).
Activities:

- Pivoting, island hopping
- Follow rules of engagement to prevent real harm

Metasploit Framework (MSF)

Purpose: Exploit development and execution platform.
Modules:

- **Exploits:** Execute payloads
- **Payloads:** Single (self-contained), stagers/stages (modular)
- **Meterpreter:** Advanced in-memory post-exploitation agent
- **Auxiliary:** Scanning, info gathering, DoS
- **Post:** System interaction, enumeration, credential dumping

Architecture:

- Ruby-based, modular structure
- msfconsole: Primary CLI interface
- Can integrate with external tools (Nmap, Nessus)

Lab: Exploitation and Metasploit

Goal: Learn practical exploitation using the Metasploit Framework (MSF).
Target Environment:

- Vulnerable Linux machine in virtual lab setup
- Services exposed: SSH, Samba, HTTP

Key Commands:

- `nmap -sS -sV -O -A <IP>` – scan target for open ports and services
- `msfconsole` – launch Metasploit CLI
- `search <keyword>` – find exploits or modules
- `use <module>` – load exploit/module
- `set RHOST <IP>` – set remote host
- `set PAYLOAD <payload>` – select appropriate payload
- `exploit` – execute attack
- `sessions -i <id>` – interact with session
- `getuid`, `sysinfo`, `ps`, `hashdump`, `shell` – post-exploitation

Exploitation Process:

- Scan for vulnerable services (e.g., VSFTPD)
- Search and select matching exploit in Metasploit
- Configure exploit parameters (RHOST, RPORT, payload)
- Launch exploit and gain reverse shell via Meterpreter

Metasploit Modules Used:

- **Exploit:** `exploit/unix/ftp/vsftpd_234_backdoor`
- **Payload:** `linux/x86/meterpreter/reverse_tcp`
- **Auxiliary:** `scanner/portscan/tcp`, `scanner/ftp/ftp_version`

Post-Exploitation Tasks:

- Enumerate users/processes
- Dump password hashes (`hashdump`)
- Launch interactive shell or pivot to further targets

Key Learnings:

- How to map vulnerabilities to working exploits
- Effective use of Meterpreter for post-exploitation
- Importance of version info and accurate scanning

Exploitation

Goals:

- Understand the concept of Return-Oriented Programming (ROP)
- Learn to craft a ROP chain to achieve a specific goal
- Explain how ROP circumvents NX/DEP protection
- Understand conditions to bypass NX/DEP, ASLR, and stack canaries

Protection Mechanisms (Revisited)

ASLR (Address Space Layout Randomization):

- Randomizes base addresses of stack, heap, and libraries at runtime
- Makes it harder to predict memory layout for reliable exploitation

NX/DEP (No-eXecute / Data Execution Prevention):

- Marks stack or heap memory regions as non-executable
- Prevents execution of injected shellcode
- Enforced by hardware and OS support

Stack Canaries:

- Random value placed before return address
- Checked before function return to detect overwrites
- Abort execution if changed, thus preventing basic buffer overflows

Exploits – Concepts and Classification

Definition: An exploit is software/data/command sequence abusing a vulnerability to cause unintended behavior.
Types:

- Local – exploit system where attacker already has access
- Remote – exploit over the network
- Client-side – requires user interaction (e.g., opening a file)
- Server-side – no user interaction needed
- 0-day – exploits unknown/unpatched vulnerabilities

Examples:

- Ping of Death (oversized packet)

- JavaScript browser exploit
- Netgear CVE-2017-5521 (redirect and token reuse)

Memory Corruption Vulnerabilities

Types:

- Buffer overflows (no/incorrect bounds checking)
- Indexing errors
- Arbitrary memory writes
- Use-after-free
- Type confusion

Protection Mechanisms (Revisited)

- **ASLR (Address Space Layout Randomization):** Randomizes memory locations
- **NX/DEP (No-eXecute/Data Execution Prevention):** Marks memory as non-executable
- **Stack Canaries:** Detect stack corruption before function return

Return-Oriented Programming (ROP)

Concept:

- Reuses existing code (gadgets) to perform operations
- Gadgets end in `ret` instructions to chain control flow
- Bypasses NX/DEP as no new code is injected

Steps to Exploit with ROP:

1. Find target function address
2. Determine offset to return address
3. Overwrite return address with function address
4. If parameters are needed, add them to stack + a gadget (e.g., `pop; pop; ret;`)
5. Chain multiple calls using gadgets

Challenges and Countermeasures

Stack Canaries: Prevent direct ret address overwrite; workaround:

- Overwrite function pointer instead
- Leak and reuse canary value
- Use jump-over techniques

ASLR:

- Makes gadget address guessing hard
- Mitigated via info leaks or brute force (easier on 32-bit)

Control Flow Integrity (CFI):

- Detects invalid indirect calls
- Requires programs to be compiled with special flags (e.g., `/guard:cf`)

Conclusion

- ROP is powerful but challenged by modern protections
- Still useful where protections are weak or missing (e.g., IoT, legacy systems)
- New attack trends focus on memory read/write primitives, logic flaws, and side-channels

Lab: Return-Oriented Programming (ROP)

Goal: Exploit a buffer overflow using ROP to bypass NX and partially mitigate ASLR.
Target Setup:

- C binary with buffer overflow
- Protections: NX enabled, ASLR (may be disabled), no stack canaries
- Architecture: x86_64

Tools Used:

- `gdb` – debugging and memory inspection
- `pwntools` – Python scripting for exploit automation
- `ROPgadget` – find usable gadgets in binaries
- `objdump -d <binary>` – disassemble to find function addresses
- `readelf -s <binary>` – find symbols like `system`, `/bin/sh`
- `cyclic`, `cyclic -l <value>` (from pwntools) – determine buffer overflow offset
- `setarch ‘uname -m‘ -R <binary>` – run binary with ASLR disabled

Exploitation Steps:

1. Find overflow offset using `cyclic` pattern
2. Locate `system` and `/bin/sh` address
3. Find gadget to control `RDI` (e.g., `pop rdi; ret;`)
4. Build payload:
   - Padding to offset
   - Gadget to set argument
   - Call to target function
5. Test with `gdb` and launch exploit

Key Concepts Practiced:

- Overwriting return address with controlled data
- Chaining existing instructions (gadgets) to invoke desired code
- Understanding calling convention (x86_64 → first arg in RDI)

Malware (Part I)

Overview and Goals

Definition: Malware (malicious software) is code that compromises CIA (confidentiality, integrity, availability) or behaves without admin/user consent.
Goals:

- Understand common malware types: worms, Trojans, ransomware, rootkits, bootkits
- Understand malware communication strategies
- Understand why malware defense is hard

Malware History (Milestones)

- **1949 – Von Neumann:** Self-replicating programs (theoretical)
- **1982 – Elk Cloner:** First virus in the wild (Apple II, boot sector)
- **1988 – Morris Worm:** First internet worm, infected 2000 Unix systems
- **2001 – Win32.S-0-1:** First social network worm via MSN

Malware Classification

Types of Classification:

- **By Type:** virus, worm, Trojan, bot, etc.
- **By Behavior:** e.g., info stealer, downloader
- **By Family/Lineage:** code origin or evolution

Note: Categories are not mutually exclusive.

Key Malware Types

- **Trojan Horse:** Disguised as legitimate software
- **Backdoor/RAT:** Allows attacker remote control
- **Downloader:** Downloads more malicious tools
- **Dropper:** Installs malware locally from embedded data
- **Bot/Botnet:** Controlled fleet for DDoS, spam, credential theft
- **Spyware/Monitor:** Logs keystrokes, screen, audio, etc.
- **Information Stealer:** Auto-extracts specific data (e.g., cookies, documents)
- **Scareware/Adware:** Manipulates user with fake alerts or annoying ads
- **Ransomware:** Encrypts files and demands ransom (often using public-key crypto)
- **Virus:** Infects files and propagates with user assistance
- **Worm:** Self-replicating, spreads autonomously via vulnerabilities

Advanced Malware Concepts

Living Off the Land: Abuses legitimate tools (e.g., PowerShell)
Fileless Malware:

- Only resides in memory
- Injected via exploits or via legitimate software

Cryptominer: Uses resources to mine cryptocurrency
Spambot/Mailer: Sends email from compromised accounts

Rootkits

Goal: Stealth and persistence
Types:

- **User-Mode:** API hooking, runs with user privileges
- **Kernel-Mode:** SSDT hooking, device drivers
- **Bootkits:** Infect bootloader, early execution
- **Hypervisor (Ring -1):** Hides OS in VM (e.g., Blue Pill)

- **Firmware Rootkits:** BIOS, NIC, HDD, routers

Detection:

- Look for altered data structures (e.g., SSDT)
- Timing analysis
- Use of external time sources for hypervisor detection

Malware Communication

Goals:

- Ensure resilience to take-downs
- Remain undetected

Architectures:

- **Client-Server:** Direct communication with C2
- **Peer-to-Peer (P2P):** Resilient, harder to disrupt

Evasion Techniques:

- Fast Flux – rotating IPs rapidly via DNS
- DGA – generate new domains dynamically
- Domain Fronting – mask C2 as legitimate service
- Use of legit apps (Dropbox, Evernote, IRC)

Covert Channels

Smart Communication:

- Mimics "normal" network behavior
- Protocols: HTTP(S), DNS, SSH, etc.

Covert Channels:

- Delay-based exfiltration (e.g., WLAN inter-packet delays)
- DNS Covert Channels (data in DNS queries)

Example:

- `cl1020-getcmd-lastwasok.adversary.com` encodes commands
- Response can be IP-encoded instructions (e.g., `100.105.114.32`)

Malware Part 2

Overview

Goals:

- Understand why malware defenses are still weak
- Learn how Anti-Virus (AV) works (signatures, fuzzy hashes, behavior, etc.)
- Recognize evasion techniques and AV limitations

Detection Techniques

AV Systems Use:

- **Static Analysis:** Without execution (file metadata, binary/code)
- **Dynamic Analysis:** With execution (memory, syscalls, network)

Detection Engines:

- **Signature-based:** Exact/fuzzy match to known byte sequences
- **Heuristic-based:** Rules from domain experts (structure, imports)
- **Behavior-based:** Detects what malware *does*
- **Reputation-based:** Based on file origin, age, prevalence

Anti-Virus Architecture

- Host + Network based components
- **Cloud AV:** Submits file metadata (fuzzy hashes, origin, behavior)
- Unknown files quarantined and uploaded
- Signature updates allow instant response post-"patient zero"

Signatures

Traditional:

- Byte sequences, hash matches (e.g., MD5, SHA-1)

Fuzzy Hashes (CTPH):

- **ssdeep:** Compares pieces using rolling hash and edit distance
- **sdhash:** Bloom filters from rare byte sequences
- **TLSH:** N-gram frequency distribution

YARA Rules:

- Rule-based matching (conditions, strings)
- Used in malware classification, Office analysis, pcap, etc.

Heuristic and Behavioral Detection

Heuristic:

- Static: File structure and metadata anomalies
- Dynamic: Simulated execution to observe rules

Behavioral:

- Observes runtime actions: file/registry access, networking
- Can be performed in sandbox (e.g., Cuckoo)

Reputation and ML

Reputation-based Detection:

- Based on age, prevalence, and origin

Machine Learning:

- Static + dynamic features used to train classifiers
- Can learn new variants without manual rule updates

Evasion Techniques

- **File Format Tricks:** Rename, embed in obscure types
- **Compression:** Zip bombs, password-protected archives
- **Polymorphism:** Self-mutating payload
- **Metamorphism:** Full code mutation (not just payload)
- **Sandbox Detection:** Check for mouse/keyboard input, clock, registry, VMs
- **Timing Tricks:** Sleep until analysis period is over

Effectiveness of AV

- AV is effective but imperfect
- No tool guarantees full protection
- Test results vary by setup, are often vendor-sponsored
- Retrospective testing hard due to update mechanisms

Mobile Platform Security

Mobile Device Security Characteristics

- Similar hardware/software to standard computers but used differently
- Frequently lost or stolen, contain personal/sensitive data
- Used across various networks including untrusted Wi-Fi
- Store credentials, easy app installation, low user awareness
- High-value attack targets
- Vendors implement protection beyond standard computers:
  - Prevent malicious app installation
  - Prevent data access if device is stolen
  - Restrict admin rights for users

iOS Security Model

Principles:

- Secure Boot Process
- File Encryption and Data Protection
- Passcode, Face ID / Touch ID
- Keychain
- Signed Apps only (App Store)
- Sandboxing and Permissions

Secure Boot Chain:

- Boot ROM (immutable, contains Apple Root CA key) → iBoot → Kernel
- Ensures only trusted iOS kernel is executed
- Vulnerabilities in Boot ROM can lead to jailbreaks (e.g., Checkm8)

File Encryption:

- AES-256 hardware encryption between memory and flash
- Unique UID (per device) and GID (per processor class)
- Encryption uses per-file keys wrapped with class keys, further encrypted with UID/passcode
- Metadata encrypted with a file system key (stored in effaceable storage)
- Secure Enclave manages sensitive key operations

Passcodes and Protection Classes:

- 4/6-digit or alphanumeric codes used as extra key material
- Protection classes: Complete, Until First Unlock, No Protection
- Secure Enclave enforces delays to slow down brute-force attacks

Keychain:

- Secure password/credentials storage
- Tied to app ID and protection classes
- Variants: Device-only, Passcode-protected, Encrypted backups

App Code Signing and App Store:

- Only Apple-signed or Apple-certified apps run
- App Store reviews apps for private APIs or suspicious behavior

Runtime Security:

- Mandatory Access Control (TrustedBSD MAC)
- Sandboxing enforced per-app via kernel policies
- ASLR, NX-bit, WX memory, signed code enforcement

Inter-App Communication:

- Limited to URL schemes
- Receiving app must register scheme, can validate sender

Jailbreaking:

- Escalate privileges via vulnerabilities (BootROM, iBoot, userland)
- Allows root rights, unsigned code, disables sandboxing
- Tools: Cydia, OpenSSH
- Decreases security; not recommended for production devices

Android Security Model

Principles:

- Runtime security features
- Permissions model
- App code signing
- Limited inter-app communication
- Full Disk and File-Based Encryption

Runtime and Sandboxing:

- Based on modified Linux kernel
- Apps run in ART VM, each app has a dedicated Linux user
- Sandboxing enforced via Linux DAC
- Uses APKs/AABs with dedicated data directories

Security Features:

- ASLR, NX-bit for native code
- Java code safer, but native C/C++ code still used

Permissions:

- Pre-6.0: Accept all at install
- Post-6.0: Runtime permissions; dangerous vs. normal
- Dangerous permissions grouped; one grants all in group

Code Signing and Distribution:

- Self-signed certificates (not by Google)
- Signing ensures app updates only by same developer
- Apps can share user IDs if signed with same key
- Apps can be installed from anywhere (Play Store, side-load)

App Store Security:

- Play Store reviews apps but not foolproof
- Repackaged APKs, hidden behavior, runtime code loading
- Malware can be injected post-approval

Inter-App Communication:

- Uses Intents for messaging
- Implicit (via URL schemes) and explicit (targeting component)
- Allows bidirectional communication

File Encryption:

- Android 5–9: Full Disk Encryption (FDE) via dm-crypt
- Android 10+: File-Based Encryption (FBE) using fscrypt
- FDE key stretch via scrypt; stored in TEE if available
- Weak passwords = weak protection

Rooting:

- Grants user root access, disables restrictions
- Done via bootloader unlock or privilege escalation
- Often uses su binary + SuperSU manager

Summary

- Mobile devices face higher risks than desktops (loss, theft, sensitive data)
- iOS: Strong, restrictive security model
- Android: Flexible, but weaker model due to diversity
- Jailbreaking and rooting remove built-in protections

Mobile Apps Security

Overview

- Covers 6 key categories of mobile app security.
- Derived from OWASP Mobile Top 10, but reorganized and adapted.
- Categories: Data Storage and Leakage, Secure Communication, Authentication and Authorization, Inter-App Communication, Client-Side Injection, Reverse Engineering.

Data Storage and Data Leakage

- Sensitive data can be exposed via file system, backups, copy/paste buffer, logs, screenshots.
- **Insecure storage:**
  - `shared_prefs` (Android), `Documents` (iOS) – accessible via root/malware, included in backups.
  - Shared storage (e.g., `/sdcard`) is insecure.
- **Preferred:**
  - Use Android Keystore or iOS Keychain/SecureEnclave.
  - Encrypt with user-provided secrets for extra protection.
- **Best Practices:**
  - Store credentials only when justified.
  - Avoid logs/screenshots/copy-paste with sensitive data.
  - Use platform APIs to disable screenshots and copy/paste.

Secure Communication

- Use TLS (HTTPS) for all network communication.
- Enforce HTTPS on Android 9+ and via Info.plist on iOS.
- **Best Practices:**
  - Disable insecure protocols (SSL, TLS 1.0) and weak ciphers.
  - Do not bypass certificate validation.
  - Use CA-signed certificates.
- **Certificate Pinning Options:**
  - Pin CA certificate (stable, fewer app updates).
  - Pin server certificate (requires updates on renewal).
  - Use self-signed cert and custom checks (more control).

Authentication and Authorization

- Use strong authentication even on mobile.
- Support 2FA (e.g., mTAN, shared secrets, auth apps).
- Avoid offline authentication: app logic and secrets can be reverse-engineered.
- **Best Practices:**
  - Use access tokens for API calls (e.g., OAuth-style).
  - Avoid storing critical credentials locally.
  - Prefer server-side access control.

Inter-App Communication

- Android: communication via Intents and exported components.
- iOS: communication via URL schemes.
- **Risks:**
  - Unintended exposure of Activities/Services.
  - Malicious apps hijacking communication or injecting data.
- **Best Practices:**
  - Set `android:exported="false"` unless needed.
  - Validate all incoming data.
  - Authenticate the calling app if needed.

Client-Side Injection

- Client-side version of injection attacks (SQLi, LFI, etc.).

- Risks include bypassing local auth, accessing local files, or abusing IPC.
- **Best Practices:**
  - Use prepared statements.
  - Whitelist input values (e.g., filenames, usernames).
  - Validate URL schemes for file access.

Reverse Engineering

- Attackers analyze apps to extract secrets, patch logic, or clone functionality.
- Tools: `strings`, disassemblers (e.g., IDA, Hopper), debuggers (`gdb`, `Frida`).
- **Best Practices:**
  - Obfuscate code (e.g., ProGuard, DexGuard).
  - Use anti-debugging checks.
  - Detect rooted/jailbroken devices; limit functionality.
  - Never rely solely on local checks or hardcoded secrets.