Security and Privacy

Analyzing security

When analyzing an attack, it is important to lay out the scenario:

- The **system** is the entity providing some computer-related functionality that is being attacked
  - e.g. a firewall, the application gateway, the encryption mechanism
- The **asset** is the object of value that will be threatened by the attack
  - e.g. data, privacy, trust in the organization
- The **vulnerability** of the system is the system flaw that allows the attack to happen
  - e.g. contains an unpatched buffer overflow, uses MD5 hashes, limited bandwidth
- The **exploit** is the method by which the attack is conducted, in order to achieve **attacker objectives**
  - e.g. sending a malformed HTTP request, sending a phishing email
  - Attacks usually involve a chain of exploits
- The **countermeasures** are techniques specifically designed to defeat the exploit, but which may be circumvented by the attacker
  - e.g. rate-limiting connections to prevent DDoS

Analyzing security

An attack threatens one of the **CIA Principles** of the system:

- **Confidentiality**: Information is secret; there is no leakage of information
  - e.g. Authentication cookies, secret keys, messages, passwords
- **Integrity**: Information is correct, or the system is behaving correctly
  - e.g. successfully generating a fake transaction compromises integrity
  - e.g. turning a victim PC into a bot compromises integrity
- **Availability**: System is available for its normal usage
  - e.g. DDoS compromises availability
  - Turning off/air-gapping a system that should be available does not constitute security

  In cryptography, the final principle is replaced with **Authenticity**: communicating parties know who they are talking to

Analyzing security

A large governmental surveillance project has installed a backdoor in a ciphersuite. Specifically, all messages encrypted by this system could be decrypted by this government agency.

- Which CIA Principle is compromised?
- What is the system?
  - The ciphersuite
- What is the asset?
  - The message content of everyone using this ciphersuite
- What is the vulnerability?
- What is the exploit?
- What are the countermeasures?
  - Most encryption algorithms are not designed with an inexplicable number
  - Cryptographers strictly follow the principle of open design, allowing them to analyze and determine which algorithm has a backdoor

Malicious code

Attacker objective: to run **malicious code** on a target system. Generally two approaches:

- Trick the owner of the target system into running their code
  - Trojans, spear-phishing, or supply side attacks
- Take over a benign but vulnerable program on the target system
  - The vulnerable program needs to be taking some form of input, such as packets
- Bypassing authentication to run the code directly

Remark

The above are often combined; for example, taking over a vulnerable program to edit the password file, then transferring and running malicious code directly

Coding flaws

- Programs running on victim systems can have **coding flaws** due to human error, bad design, or intentional choices

Example (Programming flaws)

Dangling pointers, failed bounds checks, race conditions

- The most serious flaws allow control flow takeover, i.e. the program executes the attacker's code/commands
  - If the program is setuid zero (running as root), then the attacker can run any commands on the target system
  - Exploits are most dangerous if the program runs as a background daemon taking network input, which usually allows zero-click attacks

Example (Zero-click exploit)

Sending an image in a message that takes over the target system without opening it because of a flaw in the messaging app.

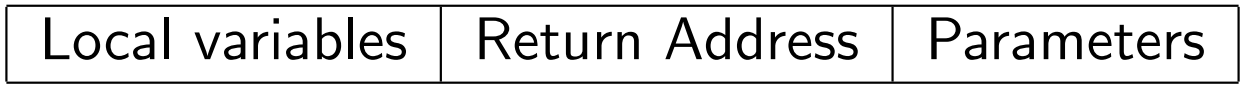
Flaws

Control flow takeover flaws include:

- Buffer overflow: Overflowing the stack/heap allows control flow takeover. Mainly an issue in C/C++
  - Stack overflow: Overwrites return address
  - Heap overflow: Overwrites virtual table pointers
  - While most other languages are much less vulnerable, libraries in other languages are often written in C...
- Heap exploits: Use After Free, Double Free
- Format string vulnerabilities
- Parsing vulnerabilities

Stack overflow

Every function has a function stack which looks like:



- The return address points to the next line of code to execute after this function returns
- If an array is defined of fixed size, it becomes a local variable in the defining function
- C arrays are pointers; an attempt to write too much data into an array will not be stopped by the compiler/processor
- Overwriting the return address allows the attacker to take over control flow

Format string vulnerabilities

- In C/C++, variadic functions can take a variable number of inputs
- During runtime, if too few inputs are presented, it will not halt; it will treat the following memory space as its parameters
- `printf` is a variadic function. `printf("%d %d");` prints two integers from stack memory after the `printf` function's parameters
- `printf(username)` can leak memory (or even overwrite memory) if the username contains a format string

Control Flow Integrity

CFI defenses can reduce attack surface for control flow takeover:

- Stack canaries: Random strings in the stack that are checked regularly to detect overflow
- NX (No-Execute): OS marks heap/stack as NX, preventing code injection into these areas
  - Does not prevent code reuse attacks which take advantage of existing code, including libraries
- Address Space Layout Randomization: Reducing consistency of calling injected or library code by randomizing their address on startup
- Various cryptographic mechanisms for memory protection and kernel code protection

Static analysis

Static analysis examines code for vulnerabilities without running it.

- **Taint analysis**: If variables marked as tainted reach a sensitive function, the program should halt
  - Helpful against format string vulnerabilities, SQL injection, information leakage
  - Tainted variables should include user input, packets, sensitive data
- **Data flow analysis**: Determine the range of values variables could take
  - Follows code logic between assignment and use
  - Useful for detecting failed bounds checks, invalid values, divide by zero
- **Control Flow Graph**: Determine the set of all valid paths between blocks of code
  - Crucial for data-flow analysis
  - Also detects programming errors such as faulty loops and conditions that are always false

Static analysis tools were also proposed to check if code is malicious, but obfuscation defeats them

Anomaly detection

For **polymorphic** code, we need heuristic detection strategies with dynamic analysis:

- Communicating with a server in a suspicious manner (indicative of Command and Control)
- Calling kernel functions in a suspicious manner (too many writes, opening password file...)
- Too much entropy (such as random filenames, indicative of obfuscation)
- Self-replication

Available sources of data for anomaly detection: kernel calls, filesystem, memory, network usage

Advanced Persistent Threats

APTs are usually government-sponsored, high-resource entities, aiming to achieve political objectives, which changes their pattern of attack:

- Stealth is a high priority
  - Obfuscation, Living Off the Land, Lateral Movement, etc.
- Vulnerabilities are zero-days
- Common objectives include surveillance, data exfiltration, destruction
- Active attack monitoring with human controllers

Remark

Which defenses work (relatively) well against APTs and which defenses work poorly?

Cryptography

Cryptography achieves CIA principles with three sets of tools:

- **Confidentiality** - solved with encryption. Public key encryption is used for bootstrapping symmetric key encryption, the latter of which is more efficient.
- **Integrity** - solved with message authentication codes. MACs are keyed hashes that can only be generated with possession of the secret key, and changes to the message will invalidate the MAC.
- **Authenticity** - solved with signatures in a public key infrastructure. Signatures can only be generated with possession of the private signature key, and can be validated by anyone with the verification key; but the verification key itself needs a source of trust.

Threat model is Man in the Middle (MITM), who can intercept, observe and replace messages in the communication channel

Encryption - Symmetric Key Encryption

Allows data to be sent across an observable channel with confidentiality.
Also called secret key encryption

- For secret key K and plaintext M, we have ciphertext:

$$C = Enc_K(M), M = Dec_K(C)$$

- Both parties need the same secret key K, which cannot be sent in the same channel
- Encryption and decryption algorithms are public
  - Kerckhoffs's Principle; cryptography should not rely on secret algorithms
- Uses block ciphers, almost always AES; stream ciphers (e.g. RC4, AES CTR) are rarely used nowadays
  - Block ciphers need a valid mode of operation to avoid duplicate ciphertexts

Encryption - Public Key Encryption

Also achieves confidentiality, but without requiring a shared secret key

- For encryption key E, decryption key D, and plaintext M, we have:

$$C = Enc_E(M), M = Dec_D(C)$$

- The encryption key is public (anyone can encrypt), decryption key is private (only receiver can decrypt)
- In Diffie-Hellman Key Exchange, both parties directly create a shared secret key while exchanging key generation information
- Relies on the computational hardness of a problem
  - e.g. Diffie-Hellman: Discrete logarithm problem
- Much less efficient than SKE
- Common schemes: RSA, ECC

Hashes

A cryptographic hash h can be applied to a message M that achieves the following properties:

- Given $h(M)$, it is difficult to find M
- It is difficult to find any M, M' such that $h(M) = h(M')$

The latter also implies that small/predictable changes to M should never result in the same hash.
Many applications as the "workhorses of cryptography":

- Verify integrity (MACs)
- Password storage
- Reduce message sizes for signatures

Common schemes: SHA (SHA-2/SHA-256), MD5 (insecure)

Message Authentication Codes

A common MAC is a keyed hash of the form $h(k||M)$

- Because of h, it is difficult to obtain M, or to change M while keeping the same hash
- Defeats the threat of malicious changing the message; it is impossible for an attacker to generate a hash without k, or to obtain k from such a hash
- Provides authenticity if you can guarantee who has k

Nowadays, usually "merged" with encryption by using an AEAD mode of operation in AES

Signatures

If we flip our PKE key management, i.e. keeping the "encryption" key private while publicizing the "decryption" key, we have a signature scheme:

- "Encryption" becomes "signing" – only one person can generate the signature
- "Decryption" becomes "validation" – anyone can validate the signature
- Hash the message before signing because signatures are slow
- Usually only the key negotiation stage contains a signature, used to deliver public key for PKE or key material for DHKE

Public Key Infrastructure

Problem: How do you deliver verification keys?

- If sent during key negotiation, the attacker can replace it with their own verification key and subsequently replace all parts of key negotiation

Solutions vary by technology, examples:

- In TLS, verification keys of trusted Certificate Authorities come with the browser
- In DNSSEC, verification keys of trusted Root Servers are publicly announced
- In PGP, verification key trust is bootstrapped with signatures forming a web of trust
- Hardware can have built-in verification keys

Attacks on cryptography

What capabilities should we assume for an adversary?

- Current consensus is that the attacker should be able to consult a decryption/encryption oracle

Definition

A decryption oracle decrypts any ciphertext submitted by the attacker; an encryption oracle encrypts. They use the same key as the channel the attacker is trying to break.

- Security property defined as adversarial game: after any (polynomial) number of oracle consultations, the attacker attempts to guess if a ciphertext comes from one of two plaintexts and wins if they do so with non-negligibly greater than chance probability
  - Proof by security reduction: if the attacker can win such a game, the attacker can use their strategy to solve a problem thought to be difficult

Attacks on cryptography

- Direct attacks ("cryptographic breaks"): shortcuts in computation that allow an adversary to do better than random guessing
  - Due to factorization algorithms, 2048-bit RSA is considered to have roughly the same security as 112-bit AES
- Breaking encryption modes with weaker cryptographic properties: Padding Oracle, BEAST
- Side channel attacks: CRIME, BREACH
  - Utilizing size of compressed packet to deduce information about the packet

TLS

The most commonly used ciphersuite, used to access the Internet (currently TLS 1.2+1.3):

- Encrypts the Application Layer
- Uses AES with AEAD, Elliptic Curve Diffie Hellman, and Certificate Authorities
- Certificate Authorities publicly log all certificates they generate
- Encrypted SNI can also encrypt the domain name

Networks

Network security objectives include:

- Prevent malicious packets from reaching potentially vulnerable systems
- Preventing communication with a command & control system
- Preventing data exfiltration
- Preventing availability attacks, e.g. DDoS

All of these can involve machine learning

Tools

- Port mirroring: A L2 Switch can "copy" all of its traffic into another device for monitoring and analysis (Switch Port Analyzer in Cisco)
- Gateway: A router that separates internal organizational traffic from the outside Internet; commonly used to implement ingress/egress firewall rules
- HTTP Proxy: A second machine that occupies the registered IP of the HTTP server but only proxies HTTP to the real HTTP server; implementing firewall rules and security checks such as SQL scanning

Intrusion detection

IDS detect intrusions on two types of data:

- Host-based: Decision made on data visible from a computer system, such as CPU usage, files, registry, and kernel calls
- Network-based: Decision made on network packets, including TCP/IP headers, TLS headers, DNS, and possibly packet content

Detection is computed with:

- Signature-based IDS: Compared with signatures of known traffic/files; usually without ML
- Anomaly-based IDS: Uses ML to determine if data is unexpected compared to known benign data, or previous data. Generally uses unsupervised learning

In modern frameworks, IDS is often centralized (e.g. EDR systems); data from multiple machines are taken together to show a more complete picture of potential attacks

TLS

With TLS, several things remain accessible to a MITM attacker:

- TCP/IP layer (addresses, ports)
- Domain names (encrypted SNI is unpopular/banned)
- Packet sizes, direction, timing
- Other channels such as DNS

In particular, it remains easy to identify someone's destination

Internet Privacy

Privacy can be understood from many perspectives:

- Legal perspective: can be considered an extension of freedom from unreasonable search and seizure
- Ethics perspective: framed as the "right to be left alone", or the "right to self-expression"
- Data perspective: the right to control your own data
- Security perspective: often considered to be confidentiality
- Anonymity perspective: it is the dissociation between identifiers and behavior information

Internet Privacy

Achieving Internet Privacy is difficult:

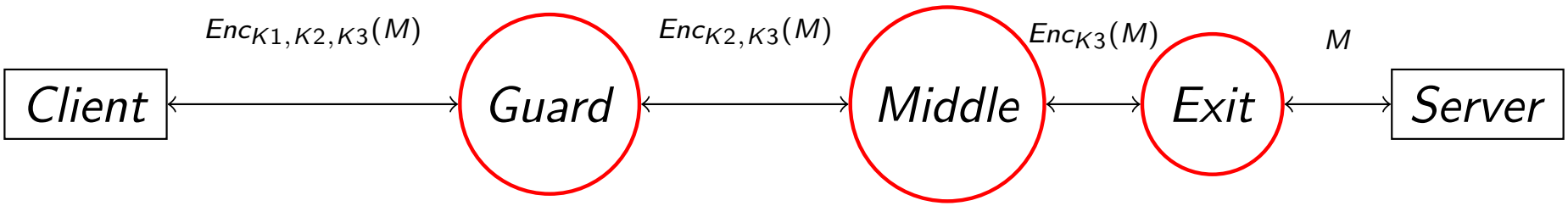
- TLS cannot encrypt source and destination IP
- VPNs protect against ISPs and eavesdroppers on the wire, but they replace ISPs as the "trusted third-party"
- Proxies are easily blocked; cannot maintain a paid service
- Highly motivated and resourceful attacks; large corporations unwilling to risk crackdown
- Conflict between network security needs and privacy needs

Tor

Uses onion routing to achieve (some) internet privacy goals

- Client connects to web server through three relay nodes. Client must run Tor but server does not need to
- Client is aware of server, but server cannot see the true identity of the client
- Nodes are volunteers. Tor's design minimizes the amount of information that nodes can see
- Any eavesdropper (and any node) can only see at most one of "client IP" and "destination IP"
  - Attacker objective is therefore to recover one given the other

Tor



- Only Client and Guard know $K1$, only Client and Middle know $K2$, only Client and Exit know $K3$
  - Key negotiation takes place through nodes; Middle and Exit do not know who the client is
- One Tor connection is a circuit. A circuit can hold multiple TCP connections (multiplexing) and lasts for 10 minutes
- A single TCP connection is a stream inside of a circuit
- Nodes are randomly selected but attempts to avoid same AS; if attacker controls both guard and exit for a client (and knows that), the attacker wins

Tor

Some attacks:

- **Circuit confirmation attacks**: A type of flow correlation. Attacker believes it might control the same guard and exit for a client; by observing flow, it can confirm this
- **DoS attacks against other nodes**: Maximizing the chance that the client picks your nodes
- **Blocking nodes**: Tor nodes are public and easily blocked. Resolved partly through bridges.
- **Browser Fingerprinting**: Server creates a fixed identifier for the client based on client browser/OS information
- **Website Fingerprinting**: Guard/pre-Guard eavesdropper attempts to guess the web server based on traffic flow characteristics

Data Privacy

Data can be processed while protecting privacy

- **k-anonymity**: Releasing data in a sanitized manner that does not indirectly reveal private information
- **Differential privacy**: allowing query on data without revealing private information
- **Secure multi-party computation**: allows two parties holding data to jointly compute a function without sharing any data
- **Private information retrieval**: allows fetching information without revealing the query
- **Zero-knowledge proofs**: allows proof of knowledge of something without revealing any information about it (such as a password)

k-anonymity

- In data release, many types of information can be considered **quasi-identifiers**: phone numbers, ZIP codes, age, etc.
- Each data point can also contain sensitive attributes, such as medical condition
- We want to release this data for research, but we also need to anonymize quasi-identifiers: but by how much?
- In k-anonymity, a data point is considered safe for release if the QIDs are exactly the same as $k-1$ other data points
- This is not sufficient to break the link between sensitive attributes and quasi-identifiers

Differential privacy

- Answering query Q on database D directly often leads to privacy leakage

Example

Query the mean salary before and after a person joins

- A query is said to be ϵ-differentially private if for any neighboring databases D, D', and for any potential query output x,

$$\frac{P(Q(D) = x)}{P(Q(D') = x)} \leq e^{\epsilon}$$

- Two databases are neighboring if they differ by one entry (one person's data)
- Differential privacy ensures that the query answer x gives limited information about the data D

Model security

- Data poisoning attacks: Creating data points that would lead the classifier to train incorrectly
  - Most effective in federated learning
  - White-box attacks allow access to gradients
- Adversarial perturbations: Creating data points that would be classified falsely
  - Usually assumes query access to model
- Hardening models against such attacks is ongoing work

Model privacy

- It is possible to recover the data that a model was trained on
- Membership inference attacks: Determine if a given data point was in the training set or not (usually by examining model confidence)
- Model inversion attacks: Reconstructing data points such as images given gradients/model information

Biometrics

- Used broadly for authentication: fingerprint, face, voice, etc.
- Often uses machine learning to compare samples
- Attacks: mimicry, stealing biometric information; causes false negatives or false positives