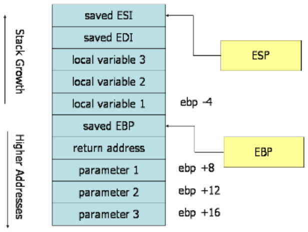
ECE568 Aid Sheet / Jason Pham / 1003078144 / MY-150

Security Fundamentals

Confidentiality: the protection of data or resources from exposure. Important to conceal the contents of data/resource, and its existence
Integrity: The trustworthiness of the data or resource. Important to ensure correctness of its contents and origin
Availability: ability to access or use the data or resource as desired. Harder to ensure availability (temporarily unavailable due to system crashes, inadequate resources, etc.)
Threat: any method to potentially breach security and cause harm.
Vulnerability: a flaw that weakens the security of a system.
Compromises: When an attacker matches a threat with a vulnerability
Risk = P(threat) * P(vulnerability) * Cost(compromise)
Trust: Defines how much exposure a system has to a particular interface (more trust => greater threat)
Reflections on Trust: Ken Thompson postulated the ultimate virus that could never be eradicated is to compromise the compiler. If a virus can be hidden in a compiler, it can infect the OS and all other programs, and use them to hide its existence. Infecting highest level of trust.

Buffer Overflows

Stack smashing: Requires a string, buffer on the stack, and a bug where the input string being copied into the buffer is unchecked.
Optimizing Shellcode:
- Create array: [“/bin/sh”, NULL]
- Load %ebx with “/bin/sh”, %ecx with address of the array, %edx NULL
- Trap into the kernel to call exec: Put 0xb into %eax, execute int 0x80

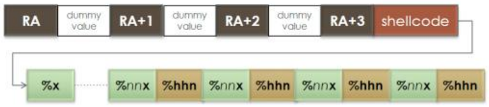


Some additional notes:
- The process stack grows downwards on Intel
- ASLR, NX-pages, and stack canaries will stop most attacks
- execve is one of the variants of the exec system call in libc
- libc is, by default, dynamically linked, to any C program
- If buffer is not large enough to hold shellcode: put the sc in another buffer somewhere else (sometimes you can put sc after the buffer)
- If program forms buffer from several strings: Provide sc in pieces.
- You can defeat ASLR if the attacker can read beyond the end of the buffer until they see an address (return/frame pointer address) and can guess where the stack/code segment is located.
- Use return oriented programming to defeat NX-pages by constructing gadgets linking already compiled snippets of instructions to redirect execution of the program to launch libc system calls.

Format String

`sprintf(buf, “%s”, attacker_string);` - Can exploit buffer overflow similar to strcpy
`snprintf(buf, len, attacker_string, ...);` - No buffer overflow risk, but attacker gets to specify the format string
- Arguments are pushed to the stack in reverse order
- Copies data from the format string until it reaches “%”. The next argument on the stack is then fetched
- Extra “%” will continue to read off the stack (usually starts with buffer)
- Printing far enough can grab frame and stack pointers

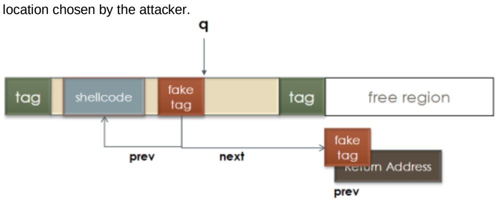
Information Leakage: Valuable information further up the stack
%n: Assumes argument is a pointer, writes number of characters written so far. You can take control of a program if %n points to the saved return address on the stack.
Exploiting format strings:
1) At the front of the format string, put address where you think the return address is stored on the stack
2) Put the shellcode in the format string
3) Put enough “%” arguments so that the argument pointer points to the front of the format string
4) Put a %n at the end and overwrite the return address to point at the shellcode in the buffer
How do you get %n to be a high value? Supply a width arg ex: %24d
Stack address is really large issue: Write one byte at a time. Use lowest-order byte stored by %hhn, it is incremented with modulo-256.
(Note: `%n: int *; %hn: short *; %hhn: char *;`)



Additional Notes:
- Limiting length of snprintf doesn’t stop the attack because it interprets the entire format string regardless of size limit

Double Free Attacks

malloc: maintains a doubly-linked list of free/allocated memory regions
Each region is maintained in a **chunk tag** that is stored just before the region. Each chunk maintains: free bit + link to next/prev chunk tag.
Free region also has a tag associated with it
free: Sets the free bit and consolidates adjacent free regions
Double-free vulnerability: Program calls free on region that contains data by the attacker (free(q) where q is the address in allocated space)
That attacker can set the values in their fake chunk tag, free will overwrite a memory location chosen by the attacker.



Other vulnerabilities

Return to libc: Change return address to point to start of system func. Injects a stack frame on the stack, just before return, sp points to &system. System expects its arguments at the top of the stack.
Function pointers: Overwrite function pointer with buf overflow to redirect execution of program
Dynamic Linking: OS links libc funcs at runtime to arbitrary locations. Records these in the *Procedure Linkage Table* and *Global Offset Table*
GOT: Table of pointers to func: contains abs. memory loc. of each func
PLT: Table of code entries (similar to switch statement). Each code entry invokes the corresponding function pointer in GOT
Additional Notes:
- First time func is called: runtime linker loads the library, updates GOT entry based on where the library is loaded.
- PLT/GOT <u>always</u> appear at known locations when analyzing objdump

Return-oriented Programming: Use carefully-selected sequences of instructions, located at the end of existing functions to form gadgets
Deserialization Vulnerabilities: Trick common libraries that have vulnerabilities to execute code when de-serializing objects (via MitM)
Bad Bounds Check: Allows reading arbitrary memory locations
Augment Overwrite: Overwriting argument passed into a sensitive function (like exec) using some buffer overflow attack.

Defenses

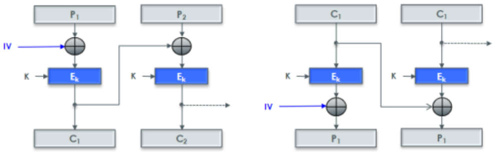
Stackshield: Put return addresses in separate stack w/ no data buffers
Stackguard: On function call, a random canary value is placed before the return address to ensure it was not overwritten (doesn’t stop format string attacks however!)
Libsafe: Dynamically loaded library (overrides libc.so), intercepts calls to dangerous functions such as strcpy (validates sufficient space)
Address-Space Layout Randomization (ASLR): OS maps the stack of each process at a randomly selected location with each invocation. Guards well against return-to-libc attacks.
Non-executable (NX) Pages: Stack is made non-executable

Cryptography

Desirable goals: Want confusion and diffusion
Confusion: Obscuring relationship between cipher and plain text. Making sure statistical analysis is hard. E(M1+M2) != E(M1) + E(M2)
Diffusion: Spread the influence of individual plaintext characters over much of the ciphertext. Each output bit is affected by many input bits. Repetitive patterns in plaintext should be hidden.
Four-properties: Confidentiality (secrecy of data i.e. ciphers), Integrity (trustworthiness i.e. hashes), Authentication (principal proves identity/origin of data - signature/MAC), Non-Repudiation (prevents principal from denying they performed an action - trusted third party)
Cipher: Obfuscates information to seem random to anyone that doesn’t possess a key
Trapdoor one-way: One-way function that’s easy to compute, hard to inverse. With a special key, the inverse is easy to compute.
Kerckhoff’s Principle: Security of an encryption system depends on the secrecy of the key **K**, not the encryption system itself
Cryptanalysis: Ciphertext-only, Known-Plaintext (knowing cipher-plaintext pairs), Chosen-Plaintext / Chosen-Cipher text
Substitution Cipher: Map different letters to one another (confusion)
Permutation Cipher: Transposes the plaintext characters (diffusion)
Polyalphabetic Cipher: Change mapping with every character
One-time Pad (Vernam) Cipher: Random substitution with every char.
Key is same length as message, compute xor on message. Theoretically unbreakable except 100% overhead, key must only be used once (easier to reverse with more), and malleable/tamperable
Symmetric Key Cipher: Uses same key to encrypt and decrypt data
Stream Cipher: Key is used to generate a pseudo-random sequence of bits and xor’d with plaintext. Useful for streaming bits one at a time, synchronization problem: if bits are lost/messed with, plaintext is corrupt
Block Cipher: Plaintext is divided into blocks and encrypted (padded)
Stream vs. Block: Stream ciphers are simpler and fast but were mostly proprietary/patented so block ciphers became more common

Block Ciphers

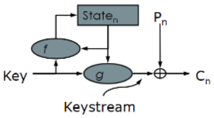
DES: Uses 56 bit key, block length of 64 bits.Each round (total 16): input is split in L,R. Two halves are switched, some computation modifies half the bits – the result is xor’d with the other half. Each round includes computation with portion of the key (subkey Kn). Output of the round becomes the input for the next.
DES Computation: $f(R_{n-1}, K_n)$: Expand R, XOR with subkey, S-box compresses key, permutation box
3DES: Longer key length and chain the DES algorithm multiple times, split key into three keys plaintext -> encrypt -> decrypt -> encrypt -> ciphertext. Backwards compatible with legacy DES.
AES: Number of rounds is based on key length/block size. Each round: byte substitution, shift rows, mix columns, addition of round keys. Can use 128-,192-,256-bit keys, and 128-,192-,256-bit blocks.
Measuring: Security, Performance, Error Propagation, Error Recovery
Electronic Codebook (ECB): Messages is broken into block size chunks, each chunk is encrypted separately with same key. High performance, low security. Since plaintext blocks always encrypt to the same cipher text blocks, cipher can reveal macro-structure of the plaintext. Error only affects block - easy to recover, resend/skip block.
Cipher Block Chaining (CBC): Make every blocks’ input dependant on the cipher text of the pervious block. For first block, use initial value (IV).
IV should <u>never</u> be reused.



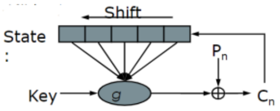
<u>Security</u>: good security – any change in plaintext affects later blocks. Modification of cipher block affects at most 2 blocks during decryption.
<u>Performance</u>: No parallelism, must be run sequentially.
<u>Error Propagation</u>: Transmission error only affects current block
<u>Error Recovery</u>: Receiver can drop/resend block and continue decrypt
Additional Notes:
- Cipher Feedback (CFB) and Output Feedback (OFB) allow encryption and decryption in units of less than a full block at a time (i.e. they convert block ciphers into stream ciphers)
- CFB/OFB have security, error and recovery prop. like stream ciphers
- CFB: Pipelining is possible; OFB: key stream is indep. of plaintext, allows performing cipher operation in advance, easy error correction
- Reusing IV on CBC leaks whether 2 messages start with the same sequence block – open to Chosen Plaintext Attack
- IV Reuse on CFB/OFB would be similar to a “Two-time Pad”

Stream Ciphers

Keystream: Similar to the pad in OTP except pseudo-random
Synchronous Stream Ciphers: Keystream is independent of message text. State is modified by the function **f** and the key. Each step uses feedback in which **f** takes the current state to produce the new state. Encryption XORs the keystream with plaintext; Decryption uses key to produce same keystream, and XORs the keystream with ciphertext to recover plaintext. Initial state is often referred to as **IV**



Self-synchronizing Stream Ciphers: Keystream depends on plaintext, the state consists of a *shift register*. Every ciphertext bit created is shifted into the shift register and fed back as input into **g**. Ciphertext has effect on the next **n** bits (**n** = length of shift register)



Properties:
<u>Security:</u> - Stream ciphers have similar prop to OTP, dangerous to use same keystream to encrypt 2 messages. Sync: key or IV must be changed for new message. Self-sync: insert random data at beginning.
- Using random data for sync won’t work because
- Malleable (ciphertext can be changed to generate related plaintext)
- With self-sync, attacker can replay previous-sent ciphertext into stream, and the cipher will resync

<u>Performance</u>: - Stream ciphers have better performance than block
- Sync stream can pre-compute key-stream before message arrives

<u>Error prop</u>: - *sync*: transmission error only affects corresponding p.t. bits
- *self sync*: error will affect next **n** bits

<u>Error recovery:</u> - *sync*: Recovery is impossible if cipher and keystream are out of sync, unless we know exactly how much ciphertext was lost
- *self sync*: stream ciphers will recover after **n** bits have passed

Key Exchange

Definition: Establishing shared secret across an insecure channel
Trusted Third-party: A central key server delegates keys to everyone: server **T**, client A **Ka**, client B **Kb**. A wants to communicate with B

1. A ➔ T : { A, B }
2. T ➔ A : $\{K_{AB}\}_{KA}$, $\{K_{AB}\}_{KB}$
3. A ➔ B : $\{K_{AB}\}_{KB}$

<u>Problems</u>: B doesn’t know with whom he’s communicating, third party attack could capture $(K_{ab})_{Kb}$ message and subsequent message from A to B and replay them later in order to make B repeat previous action. B can’t tell if the message actually came from A

Needham-Schroeder:
1. A → T : $\{A, B, N_A\}$ // A picks a nonce N_A
2. T → A : $\{N_A, K_{AB}, B, \{K_{AB}, A\}_{Kb}\}_{Ka}$
    // N_A in reply insures message isn’t a replay
    // includes B’s name: confirms who this is intended for
    // Note that session key for B is encrypted with A’s key
3. A → B : $\{K_{AB}, A\}_{Kb}$ // The message from T includes A’s name
4. B → A : $\{N_B\}_{Kab}$ // B uses nonce to verify they’re speaking with A
5. A → B : $\{(N_B - 1)\}_{Kab}$
    // Receiving the decrement tells B that A has the key and is
    // responding to the new message (not a replay)

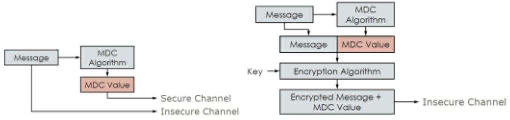
Problems with Trusted Server T: T can be compromised giving attacker every session and user key. Attacker can try to crash/overload server, making secure communication impossible
Diffie-Hellman: A select **n** (large prime modulus), g s.t. 0 < **g** < **n**, and a random integer **x** and computes $\mathbf{P = g^x}$ mod **n**. A sends **P**, **g**, and **n**.
B select random integer **y** and computes $\mathbf{Q = g^y}$ mod **n**. B sends **Q** back
A computes $\mathbf{Q^x}$ mod $\mathbf{n = g^{xy}}$ mod **n**. B computes $\mathbf{P^y}$ mod $\mathbf{n = g^{xy}}$ mod **n**
Both now know $\mathbf{g^{xy}}$ mod **n**. Susceptible to man-in-the-middle attack due to lack of authentication of the remote party.
Public-Key Key exchange: Asymmetric cyptosystem using private/public key pairs. A randomly selects **x**, encrypts w/ B’s pub key
Public Key Authentication: Message is encrypted with sender’s private key, any recipient can decrypt this. Only sender could’ve encrypted this thus providing authentication and non-repudiation.
RSA: n = pq (**p,q** are large prime numbers), size of **n** defines key size.
φ = (p-1)(q-1). Public key is coprime of φ. Private key **d**: **ed** = 1 mod φ φ, **p**, **q** must be secret. $C = M^e$ mod n (M must be > n). $M = C^d$ mod n
<u>Additional Notes:</u>
- RSA has poor resistance to spoofing because encryption uses exponentiation. If someone signs messages adversary gives them, then the attacker can trick them into signing message they have never seen.
- Suppose victim will not sign **M**, but adversary can pick **K** and get victim to sign **KM** and **K**, then a signature of **M** can be recovered.
- Still susceptible to man-in-the-middle without Public Key Infrastructure

Public Key Infrastructure (PKI): A trusted third party vouches for the identity of a key (i.e. the key belongs to a principle)
Pretty Good Privacy (PGP): Instead of central trusted party, PGP uses web of trust. Every user is capable of signing certs. A can verify public key belongs to B by signing a cert with A’s private key; if C can verify A’s public key, then C can sign it with C’s private key. Trust is transitive, if you trust C, then you can trust A and B. If you only trust A, you can trust B but not C.
Certificate Revocation: important aspect of certificate scheme is the ability to revoke certificates. You can use revocation cert (created at the time of the public key signed by a PKI) stored securely/safely.

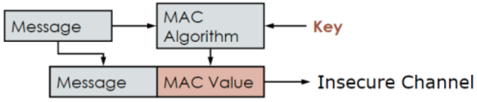
Hashes, MACs, Digital Signatures

Hashes: converts large input into a smaller output **H(m) = h**
m: pre-image, **h:** hash-value, **H():** lossy compression function
- Modification Detection Codes (MDC) provide integrity
- Message Authentication Codes (MAC): provide integrity and auth.
- Digital Signatures provide integrity, auth, and non-repudiation

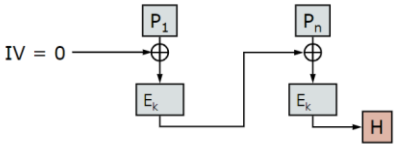
Ideal Hash: Preimage resistance (hard to reverse), 2nd preimage resistance (given m, hard to find m’), collision resistance (h2f m & m’)
MDC: Send message and its hash allows receiver to detect modification



If confidentiality is required, send encrypted message instead. MDC with encryption provides confidentiality, integrity, and authentication.
Common MDC: MD5 (broken), SHA1 (weak 160 bit), SHA256 (256 bit)
SHA1: Hashes 512 bits blocks at a time, each block is passed through 4 rounds of ops., each round use 20 ops. to update 160 bit state, after a block is processed SHA1 output is used as input for next block
MD5: Similar to SHA1, 4 rounds, 16 operations
MAC: h = H(k, M), **k** is secret key. Receiver knows sender knows key.



CBC-MAC: Similar to CBC encrypt except single hash value produced



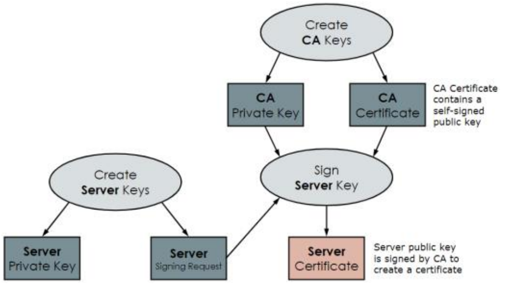
Keyed-hash MAC/HMAC: concat key with message and hash

$$HMAC = H[(K \oplus opad) \,||\, H((K \oplus ipad) \,||\, M)]$$

Assume hash block size is **n** bits, **K** is padded with zeros to **n** length, **opad** = 0x3636… repeated to **n** bits, **ipad** = 0x5c5c… to **n** bits
HMAC = H(key1 || H(key2 || message)) applies hash twice for security.
Attacks against MAC:
- Single hash allows an attacker to add arbitrary information to the end of message and compute a new MAC, without knowing key **K**
- <u>Existential Forgery</u>: An attacker can create a valid text-MAC pair without control over the text
- <u>Selective Forgery</u>: Create valid text-MAC pair with text of their choice
- <u>Key Recovery</u>: An attacker can recover the key

Hash/Merkle Tree: Provides data integrity and easy updates
Blockchain: Allows for journal of events, with both integrity and auth.
Block = timestamp + hash of prev block + hash tree of transactions + transactions + Nonce + signature
Certificates: A message signed by a trusted entity (certificate authority) Recipient decrypts cert using CA public key to obtain sender’s private key. Recipient decrypts signature of signed message using sender’s public key, generating a value to be compared with hash of message.

X.509 Cert: Contains:
- Issuer: info about CA; - Expiry & validity dates; - Version Number;
- Subject public key
- Cert signature: digital signature of first part of the cert, signed by Issue’s private key
- Subject: info about bearer (most important part being *common name (CN)* i.e. name of host);



Secure Communication Protocols

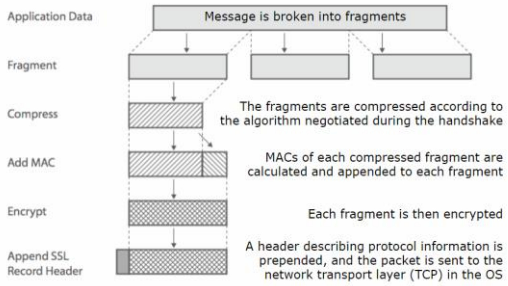
Attacker goals: Key recovery, plaintext recovery, message forgery

Attack model: Passive attack (listen/record messages), Active attack (spoofing, replay, DOS), Adaptive (learn something with each modified message, use that to create the next modified message)
- MDC prevents spoofing (hard to tamper message with MDC value)
- Prevent replay by appending one time nonce to message and cipher
  h = H(M || nonce); C = cipher text || nonce || h
- Use incremental sequence number to detect reordering attack
  h = H(M || sequence number), C = cipher text || sequence number || h

SSL: Requires application support on both ends, not network support
Two phases: 1) Key exchange/handshake 2) Communication

SSL Handshake: 3 steps: 1) Establishes the suite of ciphers each side supports, and what eversion of the protocol is being used; 2) Securely establishes a shared secret that can be used as a session key 3) Auth each others’ identities, via certifications.
- Only authenticates machines, not users making requests
- User auth is not done by SSL
- Machine authentication is optional, not done for every SSL interaction
- **Downgrade attack** is possible - filter/alter support of machine to use significantly weaker encryption schemes or shorter encryption keys
- Protects against spoofing (MAC at end of handshake), reordering/deletion, replay, MitM (certs are used to auth pub key)

SSL Communication:


Application Data
Message is broken into fragments
Fragment
Compress
The fragments are compressed according to the algorithm negotiated during the handshake
Add MAC
MACs of each compressed fragment are calculated and appended to each fragment
Encrypt
Each fragment is then encrypted
Append SSL Record Header
A header describing protocol information is prepended, and the packet is sent to the network transport layer (TCP) in the OS

- Protect against spoofing (encrypted fragments have MAC), reordering/deletion (sequence number), replay

Performance Issues: Data is encrypted using symmetric block ciphers (fast). Server uses asymmetric decryption during handshake (1000x slower). Hardware acceleration is available to perform public key ops.

Web authentication and security

Cookie authentication: Authentication token for web service.
Browser sends user/pass, server returns cookie, cookie sent in request for further accesses to the same site. Only option for stateful HTTP
- Cookie not passed in URL, do not reveal password
- Cookies have expiry time
- Good cookie: MAC(username, expiry time, …)

Cookie Caveats:
- Should not be easy to guess a valid cookie
- Cookies should not be used for auth without SSL
- Cookies should not be made persistent
- Third-party cookies allow tracking users across different sites

Same origin policy: Dictates that scripts from one origin cannot access or set the properties of a document from another origin. Two URLs are treated as same origin iff they use the same protocol, hostname, and protocol. Can be circumvented through third-party JS (ex. advertising, blog post sites host arbitrary JS)

Cross-Site Scripting (XSS): Inject malicious script code into web pages viewed by other users. Type 1: Reflected; Type 2: Persistent

Reflected XSS: Attacker crafts URL that targets vulnerable site, user needs to click on URL for successful attack, website is not modified.
- Send HTML <script> in GET/POST header, browser runs script circumventing same origin policy.
- Allows attacker to perform arbitrary actions on HTML page
- Webserver should check if input doesn’t contain script value
- XSS results from poor input sanitation

Persistent XSS: Attacker posts arbitrary code on a vulnerable site, user must visit victim site for successful attack, website is modified.
- Ex. MySpace custom pages (poor input validation)
- Prevention: Convert all special characters before sending to user
  `<a href=’’>` becomes `&lt;a href=&#039; &#039;&gt;`
- Whitelist small set of characters instead of blacklist
- HTTP_only cookies (prevents to JS from interacting with cookies)

SQL Injection: Attacker is the person browsing the webpage
Example attack string: `‘ or 1 == 1 --`

HTTP Response Splitting: Split HTTP response header into spoofed header and body. Header and body are separated by a carriage return & line feed. Suppose header contains data from user, attacker can split header into a smaller header and a valid body of attackers choosing. Similar to XSS vulnerability.

Cross-site request forgery (CSRF): Allows unauthorized commands from user to the website. Attacker tricks user into visiting a site that contains a link the user might have visited. If the user’s browser contains a valid auth cookie, the attacker issues an auth request on behalf of the user. Bypasses same origin policy. Exploits the trust that a website has for a user’s browser.

Prevention of CSRF: No easy way of preventing
- Limit lifetime of sessions/auth cookie
- Check HTTP referrer header (possible to fake referrer header)
- Require secret token (possible to load up page in JS and stealing secret token)
- Require authentication information in GET/POST parameters, not just in cookies only (Possible to inject information in GET/POST requests)

Access Control

One-time passwords: Use challenge-response authentication

Federated Identity: Designate a central Identity Provider, describes how authentications is achieved (Only FI: Don’t trust external providers)

Single Sign-on (SSO): Describes what users have to do (Only SSO: no outsource of identity is provided to external providers)

User Authentication Flow: 1. User authenticates to the identity of service provider 2. Service provider send unforgeable “token” with identity 3. User presents token to service, which accepts in lieu of auth the user 4. Service now has a certified identity of the user - Tokens must be unforgeable and must not be leaked

Authentication: Service wants to determine identity of untrusted user

Authorization: User wants to permit an untrusted service access to sensitive resources (e.g. the user’s identity and data)

OAuth Authorization Code Grant: 3 parties: 1. Resource owner (User) 2. Authorization Server (Google) 3. Client (service provider)
1) Client will provide 1 or more URIs where tokens should be sent. 2) Receive a unique Client ID identifies the client to authorization server.
- Revocation is fine for authorization, but tricky for authentication

Security Policies: Govern how a system handles information to ensure that a system maintains some security property.

Confidentiality Policy: Confidentiality defines who is authorized to access information or resources to help protect info from leaking.

Integrity Policy: trustworthy/reliability of information;Prevent corruption

Bell-La Padula (BLP): Multi-level security has two types of elements:
- Subjects: active participants in the system - Objects: data/resource needed to be protected; various levels of security (top secret, classified etc.); number of categories in the system (FBI, NSA, POTUS, etc.). Information flows upwards

Biba Integrity: Downward information flow, each subject and object has associated integrity level. There are 3 rules:
1) Simple Integrity Property: **s** is permitted to read **o** iff **i(s) ≤ i(o)**
2) *-Integrity Property: **s** is permitted to write **o** iff **i(o) ≤ i(s)**
3) **s1** can execute **s2** iff **i(s2) ≤ i(s1)**

Side Channel: Allows unintentional information flow out of your system
(Ex. R/F leakage, power usage, electromagnetic (stray signal from monitor), I/O device voltage analysis, Power analysis)

Covert Channel: An attacker creates intentionally to allow implicit information flow, outside of specified policy (info leaked intentionally)
Hard to detect. Use non-interference to analyze covert channels iff any sequence of inputs to a process produces the same output.

Non-interference caveats: Very strict property & difficult to satisfy. Identifying every covert channel isn’t necessarily usable. Sender side must be able to control what is transmitted into channel, and receiver must be able to interpret information sent BUT channel must have sufficient capacity to be useful.

Miscellaneous

Find saved RIP: in gdb, `break func_name`; `info frame`

Modular Arithmetic: every element **x** has a additive inverse **x + x’ = 0**. Every element, excluding 0, has a multiplicative inverse s.t. **x ● x’ = 1**.
Ex. Additive inverse of 4: (4 + 3) mod 7 = 0; multiplicative inverse of 4: (4 ● 2) mod 7 = 1; multiplicative inverse of 5: (5 ● 3) mod 7 = 1

RSA Addition: encrypt(K ● M) = $(K \bullet M)^d$ // d is secret
= $(K^d \bullet M^d)$ = encrypt(K) ● encrypt(M)

Resistance of Ideal Hash: Hash length of **n** bits: 2nd preimage resistance expected num guesses 2^{n-1}; collision resistance $2^{n/2}$

ZKP: 1) Completeness (if true, honest verifier will be convinced of this fact by the honest prover) 2) Soundness (if false, no cheating prover can convince honest verifier that it is true) 3) Zero-knowledge (if true, verifier doesn’t learn anything other than prover’s identity)

Network Security

Address Resolution Protocol (ARP): IP layer uses IP addresses to route packets, maps IP addresses to MAC addresses so packets can be sent to link layer. ARP broadcast sends request to all hosts with IP, only host with said IP responds with its MAC. All subsequent packets are sent using the given MAC address. ARP broadcasts are never forwarded outside a subnet, attacker must control machine on subnet

ICMP “Smurf” Attack: Floods lots of traffic to victim machine with ICMP packets from many different machines. Attacker sends ping stream to a lot of machines on the network with src=victim IP
Defenses: At host: Disable response to ping broadcast; At router/switch: disable broadcast forwarding

TCP Connection Spoof: TCP Handshake uses sequence number as weak auth. Attacker can forge packet with source set to client’s address by guessing the sequence number, thereby connecting as the victim.

TCP Reset Attack: Attacker spoofs sender’s connection and sends RST packet.
Defense: Ignore bogus RST packets, e.g., multiple packets

TCP SYN Flood: Send many connections with spoofed IP addresses. Victim allocs resource for each request until timeout until half-open connections are exhausted, then no more requests are accepted.
Defenses: - Reduce half-open connection timeout | - Drop half-open connections randomly - Send SYN-ACK cookies **:** Client sends SYN, server responds with SYN-ACK ISNs = H(src addr, src port, dest addr, dest port, rand). Honest client responds with ACK(ISNs). Server regenerates ISNs and checks that client response matches ISNs. Rand is derived from 32bit time counter. (Vuln to connection spoofing)

Border Gateway Protocol (BGP): Autonomous system (AS) that are independently managed and connect to each other via gateways. BGP updates routing info in a peer-to-peer manner

BGP Attack: Similar to ARP attack, attacker spoofs “good” routes

Domain Name System (DNS): Hierarchical name system for resources on the Internet. Maps symbolic names to IP addresses

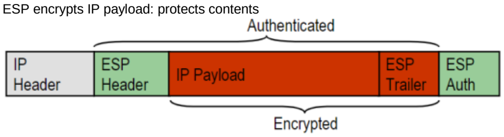
DNS Name Server: Each domain has an NS for DNS mapping for its domain, and in turn can assign other authoritative NS for sub-domains. Hierarchy makes it distributed and helps avoid single-central registers.

DNS Cache Poisoning: 1. Exploit vulnerability of DNS software (buffer overflow) 2. Spoof DNS response for single host/entire domain
Single Host: Send DNS request for domain to victim NS. Attacker floods victim NS with several DNS responses with spoofed IP and query ID guesses. If query ID guess is correct and arrives before legit, poisoned.
Domain: Generate uniquely random subdomain and make DNS requests. Flood victim NS with forged DNS responses with spoofed NS (not domain) with various QID guesses, if correct NS is cached, it will be evicted. If attacker succeeds, it controls all of victimdomain.com

DNS Rebinding Attack: Poisoning replaces victim domain -> attacker IP. Rebinding replaces attacker domain -> victim IP. Bypasses same origin policy - attacker and victim IP appear to belong to same domain
Defenses: Browsers don’t consider same IP addresses because sites might switch IP for load balancing. Browser can pin DNS/IP mapping to value of first DNS response. Block resolution of external names into local IP addresses at a local nameserver.

Cryptographic Protocols: Application Layer: SSL, SSH | Transport Layer: Use ISNs for TCP/IP | Network Layer: IPSec

IPSec: Composed of 2 layers: Authenticated Headers (AH) and Encapsulating Security Payload (ESP) – AH protects IP packets using MAC stored in AH Header. ESP encrypts IP payload: protects contents



Transport Mode: Both endpoints support IPSec, but intermediary routers don’t. Encrypts/auths the packet payload similar to SSL

Tunnel Mode: Endpoints don’t support IPSec, but endpoint gateways do. Encrypts/auths the packet header and payload, and encapsulates it in another regular IP packet (similar to SSH tunneling/VPN software)

IPSec vs SSL: If specific service is required and is supported by SSL, use SSL. If access to entire network is required, VPN/device using IPSec is a good choice. - In most cases, SSL is better for security. If one connection is compromised, others are not. - SSL can provide better access control since with IPSec allows access to entire network.
- SSL does not require special software/config (already in browser)
- SSL is more compatible with firewalls, unless IPSec and firewalls are on the same device - IPSec often doesn’t interpolate well, so both sides are required to have the same vendor’s device - IPSec supports pre-shared keys so PKI is not needed - IPSec has lower overhead since every user can use same secure channel while SSL requires connection establishments for each channel

Firewalls: Machine whose function is to control access to internal network. Access policy usually determined by port number.

Firewall Deployment: Firewalls are normally placed at entrypoints between external and internal network. Sometimes machines need access to both internal/external requiring internal firewall and demilitarized zone (DMZ) firewall

Malware

Malware: Software that is hostile, intrusive, or annoying, and is designed to infiltrate or damage a computer system

Backdoor: A method that allows bypassing authentication procedure

Spyware: Installed surreptitiously to intercept user’s action with comp

Virus vs. Worm: Viruses spread secretly, makes a lot of effort to avoid detection, needs a host program to infect and is not a stand alone program, slow spreading (requires human help) | Worms goal is to spread as quickly as possible, stand alone program, does not infect other programs, spreads automatically without human help

Virus: Work by inserting instructions into existing programs, on execution the virus may propagate to other programs. At the beginning: Virus will overwrite start of program and insert some fixup code to replicate the code it overwrote. Virus length is limited. At the end of the program, virus overwrites first instruction to jump to virus instructions, and then jumps back to beginning of program, virus only has 1 instruction and is not limited by length

Virus Detection: Scanner looks for signatures, which are strings of bit corresponding to instructions found in known viruses. Signatures should not be too short to avoid false positives; too long misses variants

Polymorphic Virus: Vary virus payload by using small decryption engine to decrypt the virus body (ex. XOR). Encryption key should change when virus propagates, decryption routine should be small to make it hard to build a signature.

Defeating Polymorphic Virus: Run program/file in emulator and wait for the decryption routine to run, signature scanner should pick up

Metamorphic Virus: Change their code on every infection by rewriting themselves: change register allocation, use equivalent instruction sequences, change ordering of blocks of code, insert code into different portions of code (not just beginning) but slows down infection rate

Detection: Run emulator and look for sequences of executed instructions but not very reliable, some viruses leave markers on other files so they know not to infect them again, possible to look for markers

Word improvements: Improve speed by hit list scanning (pre-seed worms with hosts that are potentially vulnerable and with high bandwidth so initially worm has a lot of scanning bandwidth) | Local scanning vs random scanning (worm can try infecting to local host firsts since connecting to them is faster) | Use UDP instead of TCP

Defenses: Regularly patch, disable unnecessary services, services that don’t need external visibility should be firewalled. After worm is released: shut down vulnerable service, create signature for detected worm and filter them at network layer

Rootkit: Subverts the mechanism that report on processes, files, registry entries, etc. May be memory-based (destroyed on reboot) or persistent by config file and runs on each boot
User mode: Intercepts lib/sys calls at user level; modifies return result
Kernel mode: intercepts sys calls in kernel, changes kernel code or data structures, and modifies returned results
Defenses: similar to virus/worm defenses, use host/network based signature detection, file integrity checks that bypass sys calls

Physical Security

Multifactor: Something you have/are/can do/know

Side-channel attack: Unintended consequence of a security system that leads to being less secure (ex. vault on a ship would unlock itself)

Privilege Escalation: Vulnerability in many systems makes it easy for someone of lower privilege key to create higher privilege (ex masterkey)
For keys: Start with copy of lower privileged key, file down the first pin to identify cuts of the master, repeat for all pins.

Layered Security: Combining multiple layers of security (electronic + physical) – ability to expire lost/stolen keys, easy to retrofit into existing systems, much harder to dissect and audit electronic portion

Content-type Attacks

Right-to-left Override: Unicode character (U+202E) is defined as RLO, switches the direction of text displayed: ex. www.payp[ROL]moc.la becomes www.paypal.com – hides from casual inspection

Cloud Computing

Confidentiality: Customer software/data, usage statistics/patterns Threats: observing access patterns of storage and VMs, recovering data from previously-running VM

Integrity: Customer software/data. Threats: Race conditions: exploits of weaknesses in data caching, data consistency; manipulating block/object storage; integrity of VM

Availability: Uptime of hypervisor/VM, durability of client data, geolocation of data.
Threats: Attacks on hypervisors and storage layer

Hypervisor: Low-level software component that allows commodity compute hardware to be virtualized and partition into VMs – trusted to isolate VMs from one another, from security/performance standpoint

Trusted Platform Module (TPM): Secure co-processor on motherboard of host running hypervisor. Signs hash of software running at boot (attestation), to verify integrity of the code that’s running

Firewalls: Customer-controlled firewalls allow customers to restrict traffic to/from their VMs.

Cryptography: 2 mechanisms for protecting customer data: in transit, while at rest

In-transit: Data protected by SSL, CSP servers have certs signed by known CAs

At-rest: Different methods depending on vendor

Networking: Since users use IP addresses belonging to CSP, malicious user can ban/blacklist IP address. Defense: Monitor spoofed packets blocking some outbound services (eg. spam)

Information Leakage: shared cache, storage/covert channels

Cache Timing: Attacker running code on same processor as victim can use shared cache as timing channel to infer info about data being used in computation by victim. Attack alternates b/w prime+probe phases
Prime: Attacker fills shared cache with data, evicting all of victim’s cache, when victim executes code, loads by victim will cause attacker’s data to be evicted from cache
Probe: Attacker reads data from cache and times how long each read takes inferring what data was evicted. Ex: allows for AES key recovery
Defense: Alloc memory so there’s no overlap in cache lines between different customers. Allocing memory for sensitive data should not be evicted from the cache, thus not affect timing of attacker mem access

Covert Channel: Ex: Attacker compromises VM and tries to covertly exfiltrate info w/o victim knowing.

Data Security: Goal – prove with probability that CSP has maintained integrity, availability, and durability of customer data | Solution – probabilistic algo where customers make specially-constructed queries on their data: If queries are answered correctly by the CSP, it proves with high prob. that the I,A,D of customer data has been maintained.

Proof of Retrieval (POR): Customer encrypts file and randomly embeds set of randomly-valued check blocks call *sentinels* (bonus: encrypted sentinels look like other file blocks). Customer later challenges the CSP by asking for random collection of sentinel blocks – if CSP changes large portion of file, high prob it also suppressed number of sentinels. Checksums are used to detect small changes.

Provable Data Possession (PDP): Client pre-computes tags for each block of a file, stores file with a server. Tags are computed using homomorphic encryption (tags computed for multiple files combined to single value). Client verifies that server possess the file by generating challenge against randomly selected file blocks, server calculates result for requested blocks, and sends it back as proof of possession. Client is convinced of data possession with retrieving file blocks.