CS 136 Notes

Lecture Notes

1/8 - Week 1

- Purpose of the class is to perform secure software development.
- Computer security is necessary because people have malicious intents, there is a lot of money handled by computers, lots of private information on computers (and those computers are connected to the Internet), and our society is dependent on the correct operation of computers.
- Security transitioned from something that was never a problem to something that was a problem but people didn't really care, but now it is a problem and people care.
- Large scale security problems
    - Malicious code attacks - People who write code to purposely do something bad.
        - Ex) Viruses, trojan horses, malware, cryptojacking of servers, web cameras (basically using the unauthorized use of cycles on those devices to mine crypto), etc
        - Stuxnet worm is also an example
        - Attacks on IOT devices
    - Distributed denial of service attacks
        - Use large number of compromised machines to attack one target
        - Network attack where you attack a computer on a network by sending traffic to it. Internet is great at delivering packets and it doesn't really care if the packet is good or bad. If you have 20,000 machines sending crappy packets to that one site, then the computer on the receiving end has to deal with all these packets. Attacker overwhelms that computer.
        - Very hard problem to solve
    - Vulnerabilities in commonly used systems
        - In software
            - Android, Windows, iOS, Wordpress, Adobe Flash,
        - In hardware
            - Intel/AMD processors, etc
- Electronic Commerce Attacks
    - Identity thefts - People taking out loans in your name, using your credit card, etc
    - Loss of data from laptop theft
    - Extortion via DDoS attacks (DDoS some betting site and bribe them into paying you in exchange for stopping the attack)
    - All the above result in you having some financial loss
- Security attacks are sometimes not found until months later.
- Governments also come into the business of cyber attacks. Every nation state with tech capabilities have done these attacks.

    - Stuxnet was an example of real cyberwarfare done by the US.
    - Other examples are attacks on government power grids, cyberspying by many nations, Russian election hacking in 2016.
- Some attempts to deal with cybersecurity could restrict what people are allowed to do, which could damage liberty.
    - Does data mining for terrorists pose a threat to the liberty of normal people?
- Lots of concerns about what companies are doing with the information they gather about you.
- Why aren't computer systems secure?
    - Hard technical problems
    - Low cost/benefit and only pays off when there is actually trouble. You don't really know when spending the money on creating a secure system actually stopped a potential attack or if nobody was going to attack you anyway or your current system was good enough.
    - Ignorance because a number of computer users are not technical, computers are made to be easy to use, and we don't have the right security recommendations.
    - A lot of the core internet design, popular programming languages, commercial OS were created before security was really a concern, so we have a lot of legacy systems create constraints.
        - Retrofitting these systems works poorly. Can't really change fundamental aspects about how these systems work.
- There are lots of issues with people trying to patch systems when security flaws are found, since the patches deal with the immediate problem (rather the underlying cause), done under pressure, hard to get everyone the patch, introduce new problems, etc.
- Cyber attacks are also being done increasingly quickly, under a minute for an attack to succeed.
- Security Terms
    - Security is a policy - statement of things you want to happen and things you don't want to happen. Don't say how to achieve the goal. The protection specifies the mechanism and they implement the security policies.
    - Vulnerability is a weakness that allows attackers to cause problems, but most vulnerability are never exploited.
    - Exploit is the actual incident of taking advantage of a vulnerability. Results in an actual bad effect.
    - Trust - You do things for people you trust, and don't do things for those you don't trust.
        - How do you express trust? Who do you trust? When does trust change? If A trusts B, an someone else is claiming to be B, how do you know they are actually B? Lots of identity issues.
        - Most of the vulnerabilities are when people put themselves in a position of trust when it fact they shouldn't have had that privilege in the first place. An attacker seems to be an entity that you trust and is able to perform their malicious action.

            - Phishing is an example since you click on an email link telling you to change your password, when really it was a malicious actor.
        - There is transitive trust in the form of chains of certificates, peer applications, etc. In the case of user -> web server -> database, they trust each other but does DB really trust the user? Programs that call other programs are cases of transitive trust.
- What are our security goals?
    - Confidentiality - If something is a secret, others shouldn't know it
    - Integrity - Don't let someone change something if they shouldn't.
    - Availability - Don't let someone stop others from using services.
- The threats to these goals are theft, privacy, destruction, etc.
- Passive threats are eavesdropping and no modification of information. NSA is an example, they don't want to change the information, they just want to listen in. Passive threats are mostly related to information and secrecy of that information.
- Active threats are more aggressive and commit certain actions.
- The best computer security practices can be undermined by bad human practices. Social engineering attacks look to get people to do things that they shouldn't do.
    - Phishing is the most common example. Attackers send plausible email requesting you to visit a website and get some information from you. The attacker themselves controls that site and they can obtain the information that you enter on that website. The success rate of this depends on whether the attacker is able to the convince the victim that they are real.
- Security is hard because it's expensive (computing resources + people's time/attention), we are working in an environment where there are genuine adversaries, there is limited knowledge about security among computer scientists, we have to get everything right/bug-free, and we want to share secrets and information in a controlled way, which is tough for human relations in general.
    - Also, even if you get the computer to be secure, people could attack the users, the programmers, the hardware people, etc.
- The Principle of Easiest Penetration says that an intruder is expected to attack you where you're weak instead of where you're strong.
- The Principle of Adequate Protection says that computer items must be protected only until they lose their value and must be protected to a degree consistent with their value.
    - Worthless things don't need protection.
    - Things that expire with time only need protection for that time period.

1/10 - Week 1

- Security is policy specifying what you want to happen. So what type of policies do we want?
- Design principles for secure systems
    - Economy: Since security costs cash, we have to take that into account. They have to be cheap in development cost, shouldn't add overhead when you use the

system (as minimal CPU cycles, data storage, etc), should be easy to verify the security measure, should do only what needs to be done (which keeps overhead down) - basically keeping small and simple, and only performing the security measures that you want to see.
    - Complete mediation: Need to apply security on every access to an object that we want to protect. Objects are like files, access to database, etc. At the moment a user wants to access some object, then you need to completely check if they have access.
        - Complete mediation doesn't play well with economy of use since every time you access something, you have to do some check with adds to the overhead.
    - Open design: Shouldn't rely on secrecy of design. If the opponent knows the details of the system, then they still shouldn't be able to break in. It's hella hard to keep secrets, so if the security is based on the secret, then security automatically disappears if it leaks. But like, doesn't mean that you should release it to everyone, just saying that you assume that those design details will leak, and the system will still be secure.
    - Separation of Privileges: Provide some mechanism that separates privileges used for one purpose from those used for another. This allows for flexibility in access to certain objects, which files/devices can be accessed by people, etc. That flexibility gives you choices in determining what different users should be able to do. Want to say yes to some things (show a webpage) and no to others (be a superuser on my computer).
    - Least privilege: Give bare minimum access rights to complete a task. This is dependent on good separation of privileges. We should be able to set a fine granularity. It's challenging because it's not really dependent on the mechanics, but rather you have to think about what people should and shouldn't do.
    - Least common mechanism: Avoid sharing parts of the system's mechanism since the coupling leads to possible security breaches. Virtualization of computing resources when you have different users is an example of this.
    - Acceptability: The mechanism must be simple to use. Would be useful to have encrypted email, but it's not really practical to have to decrypt Macy's emails. People need to be able to use the system without thinking about it. Also never say no when you should have said yes - basically never prevent permissible access.
    - Fail safe Designs: If any failure occurs, no security should be lost and the default should be that that person shouldn't be able to do what they wanted to do.
- A lot of the above principles are in conflict with each other at times.
- You need to have a clear security policy (aka how the system should ideally behave) and if you don't have a clear policy, then you don't have a secure system.
    - Need to ask yourself if the mechanisms B achieve the goals outlined in policy A.
- Sometimes security policies are informal and it can be hard to determine if a system meets the goals. Hard to verify and code these policies up.

    - “Only authorized users should be able to log in”
- If you want to make the policies more formal, they are typically expressed mathematically and tend toward a lot of precision, and the policies are often matched to a policy model, and it is hard to express informally.
- Bell-La Padula is the best known formal security policy. The goal was to make sure that only certain people with certain clearances were able to access certain information.
    - Combines mandatory and discretionary access control.
    - There are two parts: Clearances and Classifications.
        - Clearances: Subjects/People have these. The clearance describes how trusted the subject is (Somebody having top secret clearance gives them high level of trust).
        - Classifications: Objects have these. The classification describes how sensitive the object is. Same sort of categories as clearances.
    - Formally, this policy prevents any subject from ever getting read access to data at higher classification levels than the subject’s clearance.
    - Seems easy to code up because you need to just do a check if the classification level of the object and the clearance of the subject and if that checks out, then you let the subject read the object.
        - However, a subject with top secret clearance can look at a top secret object and then write that to a confidential object, and then the confidential users can see that information. Basically, we aren’t cool with this because we care about the information inside of an object.
    - In order to counter the above point, then we have the next constraint that S can write O iff l_s <= l_o. This prevents writing down.
    - Note: A subject with lower privilege can *write to but not read* from objects with higher classifications. A possible problem is where people can write straight garbage to important documents. They can't throw the doc away or read it, but they can still mess with it.
    - When using this stuff in practice, there have to be mechanisms for reclassification of information. (Technically, a general can't write down, but if a general wants to, then has to be a reclassification).
- Biba is an integrity policy. The subjects and objects have integrity levels. Subjects with high levels are less likely to screw up data. Data with high levels are less likely to be screwed up (probably because it’s important and needs not to be screwed up).
    - 3 Integrity Rules
        - S can write to o iff i(o) <= i(s)
            - Let the experienced person be able to change a lot of objects, but not the new intern.
        - S1 can execute s2 iff i(s2) <= i(s1)
            - The user needs to be more trusted if the program is less trusted.
        - S can read o iff i(s) <= i(o)
            - The interns are allowed to read a lot of stuff.

- Hybrid models are handling the issues when you want things to be separated. You don't want an employee who handles accounts for 2 competing businesses. Information of the two companies shouldn't be leaked to each other.
- The problems with security policies are that they are hard to define properly, how to create the mechanisms, hard to see the implications of the policy especially if the system is getting to be complex, people want "security" but aren't quite sure what specifically it means, etc.
- Tools for security
    - Physical security: Prevent bad people from physically getting to your computer. The problem is that networking and connection to the Internet pokes a hole through this. Also, there are challenges if you try to work on something on a mobile device.
    - Access Controls: Having mechanisms for only letting authorized parties to access the system. Once the data is outside your system, cannot really control it. Needs to be differentiation between people and the information they can access.
    - Encryption: Algorithms to hide the content of the data or the communication. Really important in a networked environment.
    - Authentication: Methods of ensuring that someone is who they say they are. This is important for access control.
    - Encapsulation: Methods of allowing outsiders limited access to your resources. This is what cloud computing is about. Different from access control since this refers to things that you can't even see. You cannot even ask for it.
    - Intrusion Detection: When some security measure fails, then we need to be able to notice that something is wrong and take steps to correct the problem.
    - Common Sense: Need to combat social engineering attacks by just being safe.
- Access control in particular deals with subjects, objects, and access.
- Mandatory access control is dictated by the underlying system and discretionary is under the command of the user, and the system enforces what they choose.
    - Discretionary is more common than mandatory.
    - The reality is that users never really change the defaults.
- The mechanisms for access control are
    - Access control lists (ACL): For each protected resource, maintain a list that specifies who can access the object, and do a check when people request something.
        - The problems with the list are that you need to make sure the requestor is the person they say they are, you need to make sure nobody improperly modifies the ACL, etc.
    - Capabilities: Say that every subject has a list of data items that they have access to. It's like having a bunch of keys for different places. The possession of the capability for an object implies that access is allowed. The capability has to be unforgeable.

        - In a network where the subjects are different machines, the capabilities list is stored on that machine and there is no real way for the capability checking system to figure out if the subject forged the capability.
        - If you want to revoke capabilities, then you can try to destroy it (if you can find it), or revoke on use (which means storing additional information), or generation numbers.
        - The pros of capabilities are that it is easy to determine what subject can access and easy model for transfer of privileges.
        - The cons are that it's hard to determine who can access an object, revocation is hard, and crypto required to stop forgery.
    - Role Based Access Control
        - Enhancement to ACLs. Each user has certain roles that they can take while using the system. At any point, the user has a certain role and the system will only give access to the things required for that role.
        - The changing of roles has to be nontrivial and requires some secure authentication.
        - The practical limitations are the number of roles per user, disjoint role privileges, sys admin overheads, etc.
- If you have a case of distributed access control, ACL's work if you have a global namespace for subjects. However, capabilities are tougher because they rely on unforgeability which can only be provided by cryptographic methods.
- Reference monitor is the code that checks if a given attempt to reference an object should be allowed. We want the reference monitors to be correct, properly placed, efficient, simple, and flexible.

1/15 - Week 2

- You can think of computer security as being about keeping secrets.
    - One method is making the secret stored in a secret place and having it be hard to access.
    - Another is to make the secret hard for others to read, but not for us or the authorized parties to read.
- Cryptography is about having a set of people that can decipher the code, while everyone else can't.
- Encryption is the process of hiding information in plain sight by transforming the secret data into something else so that even if the attackers see that info, they can't understand the underlying secret. They can't go from the transformed data into the original secret.
- So, in effect, encryption is one of the techniques or methods that are used in cryptography.
- Encryption, specifically, involves one byte pattern being transformed into another byte pattern and having that process be reversible.
- In encryption, we have a sender S, receiver R, and attacker O where we want to send a message from S to R without O being able to understand the information being sent.

Plaintext P is the original form of the message. Ciphertext C is the transformed form of the message.
    - Encryption is process of making the message unreadable by O.
    - Decryption is the process of making the encrypted message readable by R
    - A system performing these transformations is a cryptosystem
    - The rules or algorithm for transformation sometimes called a cipher
- In order to perform the transformations back and forth, both S and R need to know a Key K. That key is a secret. Now an issue is being able to send K from S to R secretly because if O is able to get it, then they are able to do decryption on the transformed message.
    - Successful use of keys is extremely important in determining if you have a secure system.
- Encryption Algo
    - C = E(K,P)
- Decryption Algo
    - D could be the same as C. D definitely has a key and plaintext as arguments.
- Symmetric systems use the same keys for C and D while asymmetric use different keys.
- In these systems, if you change only the key, a plaintext encrypts or decrypts to a different ciphertext. Also, decryption should be very hard without knowing the key.
- Cryptanalysis is the process of trying to break a cryptosystem. Basically, the attacker is trying to find the meaning of an encrypted message without being given the key. You can:
    - Analyze the encrypted message and deduce contents.
    - Analyze 1 or more encrypted messages to find a common key.
    - Analyze cryptosystem to find a flaw.
- Since most cryptosystems are breakable in theory, job of the designer is to make the cost of breaking the system so high that it is infeasible.
- Attacks (What does the attacker have available to them)
    - Ciphertext Only: Don't know anything about the plaintext or the key or the algo to create the ciphertext. Gotta work with probability distributions, patterns of common characters. This is a really hard attack.
    - Known Plaintext: If you have the matching sample of plaintext and ciphertext for one message, then can I use that to extract more information about the other ciphertext messages.
        - Example is if you know a particular part of an encrypted message is a date or a time, then can you try to figure out the rest of the message.
    - Chosen Plaintext: If you can submit plaintext to the system and back comes the encrypted version. By examining what happens when you encrypt certain plaintexts, then you can get a lot of details especially if you have varying levels of plaintexts.
        - Differential cryptanalysis iteratively uses these varying plaintexts to break the system. From the system point of view, you need to make sure that your cipher is resistant to this.

    - Algorithm and Ciphertext: Know the rules of the algorithm and you have the encrypted text. However, you don't have the key. Since you have the algorithm you can do runs of the algorithm with all the possible keys until the plaintext is determined. This is more of a brute force attack.
    - Timing Attacks: Know the algorithm and can see the progress of the algorithm. Then you can watch the timing of the operations being used and observing power use by hardware to try to deduce keys.
- Substitution Ciphers - Substitute one or more characters with different characters using some set of rules or mappings. Decryption will reverse the substitutions.
    - Caesar Ciphers are a type of substitution cipher where you translate each letter a fixed number of positions in the alphabet. Pros are that it is simple, but con is that it fails to conceal a lot and you only have 25 effective keys.
        - In order to attack this type of cipher, you could look at letter frequencies. In most languages, some letters occur more frequently and since this cipher translates all occurrences of one letter into the same ciphertext letter, all you need to figure out is the offset.
        - First, need to guess the type of data, count frequencies of the encrypted symbol, match to observed frequencies of unencrypted symbols in plaintext, figure out what possible keys/offsets might be, and check.
    - You can also have more complex character mappings instead of an offset. It's better because finding the mapping for one character doesn't necessarily give the mappings for all the characters to the attacker.
    - Polyalphabetic ciphers say that a given plaintext character doesn't always translate to the same ciphertext character. You could have multiple mappings depending on the situation.
        - Ex) Move character by +1 offset when they are in even positions and move -1 offset when they are in odd positions.
        - Attackers can guess this by treating all the even characters as one set and the odd characters as another set.
        - Kasiski method tries to find repetitions of the encryption pattern. Uses index of coincidence to predict the number of alphabets.
- Cryptanalyst, the attacker, knows they'll have success when you get non-garbage after translating with their predicted key. In most cases, if your key is wrong, then you'll still get garbage from it. Basically, if the decrypted text makes sense, then you've got the key.
- There is a theoretically perfect substitution cipher that is unbreakable without the key. One time pads.
    - There is a different substitution alphabets for every character. Those alphabets constitute the key. This is probably unbreakable without the key.
    - The secureness of this is that any key is equally likely and any thus any plaintext could produce the ciphertext since there are so many possible keys.
    - If the key is truly random, provable that it cannot be broken without the key. Problems are that you need one bit of key per character in the message. Key

distribution is painful since if the message is 1 TB, then S and R have to share a TB key. You need to be able to securely send that terabyte of data, and if you could, why not just send the message? Also it's required to have a truly random number generator.
    - But people don't really have true one time pad systems because pads are sometimes distributed with some other crypto mechanism, they are generated with non-random process, or they are reused.
- Permutation Ciphers - Instead of substituting different characters, scramble up the existing characters. And use an algorithm based on the key to control how that scrambling takes place. Decryption uses a key to unscramble the ciphertext.
    - One feature is that the characters don't really change, but rather their positions in the text.
    - Columnar transformation is a way of converting the message into a kind of grid and then sending the message.
        - To attack these, try to figure out how many columns are used and take note of digrams, trigrams, etc to see what letters frequently appear together. Ask what transformations would need to occur to make those digrams and trigrams appear.
    - Double transpositions is basically columnar transformation except you do it twice using a different number of columns. Generalized transpositions as well.
- Strong modern ciphers tend to use both permutations and substitutions. Permutations scramble text patterns while substitutions hide the underlying text characters.
- Quantum cryptography

1/17 - Week 2

- If we want to have a good cipher (rules for mapping the plaintext to ciphertext), we need it to be well matched to the requirements of the application (amount of secrecy should match the labor to make that security) and need simplicity for the algorithm as well as for choosing the key. We also want simplicity of implementation and the errors should not propagate. Also, size of plaintext and ciphertext should be the same, encryption should maximize confusion (the relation between plain and ciphertext should be complex) and diffusion (plaintext info distributed through ciphertext)
- Stream ciphers convert one symbol of plaintext into a symbol of ciphertext. Block ciphers wait until you have a block of data, and then encrypts that block.
    - Stream: Take a plaintext character, encrypt it as a function of the cipher and the key, kick it out, repeat until you have no more plaintext.
        - The pros are the speed of encryption and decryption. The symbols are encrypted as soon as it is available. You also have low error propagation depending on the cryptographic mode. This is because if you make a mistake for one character or one part of the message, it doesn't really have an effect on any other component.

        - The cons are that you have low diffusion (symbols are separately encrypted and each symbol only contains info about one plaintext symbol), susceptible to insertions and modifications.
    - Block: Take X number of characters from the plaintext, perform one operation that encrypts them, and out comes from the ciphertext.
        - The pros are that you have good diffusion since we can shuffle around characters within the block, and you can detect if any insertions happen since you expect that the encrypted text will arrive in known lengths.
        - The cons are that you need to wait for a whole block of data and worse error propagation since errors can affect entire blocks.
- RC4 is an example of a stream cipher where the key is changing as encryption is happening. The cipher isn't changing but the key is.
    - Create a 0-255 array and that is your key stream. You do some operation (like an XOR) between the current element of the key stream and the current element of the plaintext that you want to encrypt (And on the decryption side, you just do that same operation in the case of XOR, or a similar operation to decrypt it).
    - In terms of that key stream, start with filling a 256 byte array, choose a key, fill a second array with the key, and then swap around bytes in the first array, which is your key stream.
    - The pro is that it is very fast, but the weakness is that everyone knows the initial key stream of 0-255 (counter to that is that you shuffle N number of times or you drop the first few hundred of the keys).
- Cryptographic Modes
    - Let's say you have a bunch of data to encrypt. Block ciphers will have limited block size and stream ciphers will just keep going. A problem can occur if each block of data is independently encrypted with the same key. The effect is that the same plaintext, if it is repeated in different blocks, gets converted to the same ciphertext.
        - Electronic Code Book is the mode that represents this method of just using static mappings.
    - A cryptographic mode is a way of applying a particular cipher. It is a combination of key, cipher, and some sort of feedback that will affect the mapping from plaintext to ciphertext.
    - Cipher Block Chaining ties together a group of related encrypted blocks and hides that two blocks are identical. The encrypted version of the previous block is used to encrypt this block. Every block's encryption depends on previous blocks' contents. Good for diffusion but bad for error propagation.
        - When you're the sender and you are encrypting, you XOR the plaintext with the ciphertext of the previous block (use IV instead if you're on the first block), and then encrypt that result instead of just encrypting the original plaintext.
    - Other modes are CFB, OFB, etc

- Initialization vectors are used with CBC. They ensure that encryption results are always unique, even for the first block that gets sent.
    - For the first block, XOR the first block with a random string (which is the IV). Then encrypt the message and send the encrypted message + IV in plaintext. Then, for the next block of data to send, you'll use the first block for the XOR.
- I guess with IVs and CBC, you are modifying the plaintext that you have before applying the cipher, and so therefore even if you send the same plaintext at two different times, you get different ciphertexts they map to.
    - Sender POV: Apply XOR between current plaintext and previous ciphertext and then encrypt
    - Receiver POV: decrypt the message, apply XOR with previous ciphertext block (or IV if first block) to get the plaintext.
- Cryptography can be used for secrecy by making sure that only those who know the proper keys are able to decrypt messages.
- Can also use cryptography for authentication where I want to prove to you that I created a piece of data. You can try to do it by creating a key that only you and I know.
- Also with cryptography, changing 1 bit of an encrypted message completely changes it. However, you can do a checksum on the data, and encrypt that as well.
    - If you don't need secrecy, then you don't need to encrypt the message, but rather just encrypt the checksum.
- Recall that symmetric cryptosystems are where the sender and receiver have the same key, where asymmetric system where the sender and receiver have different keys.
    - A problem with symmetric cryptography is that you need a separate key for each pair of people trying to communicate with each other, and this makes key distribution difficult.
    - Pros of symmetric are that the encryption pretty much gives you authentication since if you know the sender has the same key as you, then you know it's them. These approaches are typically faster too.
- Example symmetric key ciphers
    - Data Encryption Standard is a symmetric cipher that uses block encryption, substitutions, permutations, and multiple rounds of these repeated operations.
        - Problem is that you can cheap brute force attacks on DES since there are short keys.
    - AES is the better solution since it uses 128 or 256 bit keys, there aren't any complete breaks. However, attacks can work on versions of AES with fewer rounds and attacks on crypto only get better over time.
- Public key encryption system is an asymmetric system
    - Most popular public key algo is RSA.
    - Keys are created in pairs, where one is kept secret by the owner, and the other is made public to the world. You encrypt the message with the receiver's public key, you send the message, and the other person decrypts the message with their private key.

    - There is a possible problem with trying to verify that someone's public key belongs to them rather than somebody posing to be them.
        - Key distribution infrastructure gives you assurance that a given key belongs to a particular person.
- Authentication with shared keys requires authentication servers and authentication with public keys requires you to encrypt the message with your private key, and then since others know my public key, they can check my claim directly.
- Security of RSA depends on factoring large numbers. The keys are functions of a pair of really large prime numbers. Figuring out the relationship between public and private key pairs is equivalent to factoring the product of those two prime numbers. There is a tradeoff between increasing the key length in order to make factoring harder and making the encryption and decryption more expensive. Instead of brute force approach to attacking PK systems, attack the mathematical relationship between public and private key.
    - Difference between AES and RSA is that AES uses simple operations, but RSA uses exponentiation and the key selection is a lot more expensive.
- Common to use both symmetric and asymmetric at the same time. You use RSA to authenticate and establish a session key, and then you can use AES.
    - PK is used to set up the session and then once you have both parties knowing the same key, then you can go on to use symmetric key communication.
- Digital signatures allow you make sure that the person sending the doc is really them.
    - It has to be unforgeable, verifiable, non-repudiable, cheap to compute/verify, can't be reusable (can't use same signature for 2 different documents), no reliance on trusted authority, and signed document is unchangeable.
    - Creating these signatures involves cryptography. You can know the key K and encrypt the document but how do others know?
- Shared key cryptography: Signatures with shared key encryption requires a trusted 3rd party. The signer encrypts the doc with their private key that they share with the 3rd party, and the receiver checks the validity of the doc by talking with that same trusted 3rd party.
- Public key cryptography: Here, the signer encrypts the doc with their private key, the receiver checks validity by decrypting with the signer's public key. However, the main caveat is that the receiver must be certain he has the right public key.

1/22 - Week 3

- Doesn't matter how strong your crpyto algo/cipher is. If your attacker has the keys, then the security is automatically gone.
    - The ciphers don't end up getting cracked often, but keys get leaked all the time.
        - RSA, AES, Blowfish all haven't been cracked yet.
- Keys are evaluated based on:

- Length: If cipher is perfect and can't be broken in any way, then the strength depends on the key length since the only attack is through brute force and therefore, the longer the key, the more brute force required.
    - More bits are generally more secure (one time pad is one one end of the security/length), but encryption is done in hardware and more bits in hardware cost more. Software encryption slows down a bit too. Also, some ciphers have defined key lengths. Also, if you don't have a perfect cipher, then it can be broken regardless of key length.
- Randomness: Don't let the attacker know how you're generating the key. Need to make that choice as close to true random as possible, but does not require true randomness.
    - One method is to use cryptography itself. Start with a random number, run a cryptographic hash (like SHA 256), and thus the new number looks pretty random and extract the number of bits you want for your key. Then, if you want more keys, you can run hashes on those old numbers.
    - However, you can make the case that hashing does not produce random numbers because the hash is deterministic. Also, is the part that you extract random? Also, if you use the process of hashing the n-1 key to get the n key, then if an attacker gets one key k, then all the keys k+1 onward are known to the attacker.
    - Perfect forward secrecy means that the compromise of one key will not compromise any others. The above approach doesn't have it since it derives one key from another using a known and repeatable algorithm (SHA hashing).
    - Getting random noise: Observing physical processes (like cosmic rays, disk drive delays, etc), record them, and assign bit values to the outcomes. This is described as gathering entropy.
        - Keys derived with the proper use of randomness have good perfect forward secrecy.
        - In order to get entropy, don't do it yourself, use a package.
    - Overall, just make sure you get a new key every time, it's non-reproducible, and no obvious patterns.
- Lifetime: How long should we keep a key that we're using? Because the longer we keep a key, the more likely it is to be found, the more data is exposed if it's compromised and the more resources can be devoted to breaking it.
    - Therefore, if something that is a secret (like a key) you should be able to change it. If not, it's a vulnerability.
    - The cons of changing are that changing keys is inconvenient (decrypt and encrypt with the new key), keys for particular types of sessions (VoIP) should be changed, some keys must be stored permanently (for encrypted disk drives).

        - In terms of storage of keys, you can try to permanently store on the machine or have people remember passwords (and then create hash keys from that) or create hardware security enclaves.
            - Symmetric session keys should be removed once the session ends. Destroying keys is hard since that information could be in caches, virtual memory pages, etc
            - Long term symmetric keys for disk encryption and therefore you need to have safe storage,
            - Private asymmetric keys also require long term storage.
    - Secrecy: Whether other people can find out your key. The same private keys are often shared on multiple machines for multiple users. Also, some keys are backed up on tapes in plaintext form. Private keys have to be private, never shared, and never left lying in insecure places. The whole security of PK systems depends on the secrecy of the private keys.
    - Generation: With two people who want to talk and if they want symmetric keys, then generating one is easy but making sure both have the key and that nobody else knows about the key is a bit harder.
- For the aforementioned issue of trouble with exchanging keys, you can use key exchange protocols.
    - Diffie/Hellman Key Exchange allows two people to share a key without previously sharing any secrets and communicating across an insecure channel. Basically, the two people agree on a large prime number n and a number g (which aren't secrets). Both then choose large random numbers (x and y), do computation, and sends them (X and Y) in plaintext over the network. Using those 4 values + their random numbers, they can compute k and k' which are equivalent. Nobody else can know this because even they know most of the numbers involved (n, g, X, and Y), they don't have the random numbers that the two people used.
        - Alice secretly chooses a large random integer x and sends Bob $X = g^x \bmod n$
        - Bob secretly chooses a random large integer y and sends Alice $Y = g^y \bmod n$
        - Alice computes $k = Y^x \bmod n$
        - Bob computes $k' = X^y \bmod n$
        - k and k' are both equal to $g^{xy} \bmod n$
        -
        - One of the problems is that it does not make a guarantee about who the two parties involved in the communication are. There isn't a way to know if Bob got Y from Alice and if Alice got X from Bob. If there is a person in the middle that intercepts Y, and sends Y' to Bob instead. There's no way to authenticate the people who are talking. Therefore, you need to use something else to authenticate the parties involved either during the key exchange or after you have the encrypted session.
- Authentication for key distribution is important for key distribution. Otherwise, you have no way of making sure that you're sending the key to the right person. So, if we have a

key on one machine and we want to get it to another machine, how do we make sure we're sending to the right person.
    - Key servers: Job of the machine is to distribute keys to other machines. Clients and key servers need to authenticate themselves to each other. Therefore clients don't authenticate with other clients directly.
        - There is kind of a transitive trust issue in that if person A authenticates to Server S and vice versa and person B authenticates to Server S and vice versa, then A is automatically kinda trusting B.
    - Certificates: Authentication without servers. Certificate is an electronic doc that proves that you are who you claim to be. Public key certificate is a copy of your public key signed by a trusted authority. Presentation of that certificate is the authentication of your key.
        - Basically, people give their public key to trusted authority, and the authority returns a certificate that contains that person's public key signed by the authority's private key (so nobody can fake being the authority), which you can use to give to others and that will authenticate you. If somebody wants to talk with you, they give you their certificate, and then you decrypt the certificate using the authority's public key (which every person keeps a copy of), and then you now have an authenticated public key for the user.
        - Hierarchies are where you have authorities signing other authority certificates. Basically, it solves the situation where you need multiple authorities for authenticating 1-2 billion users. Single authority at the top produces certificates for the next layer down.
            - One of the side effects is that you may get a signed certificate and you may not know that authority that signed it, but it could be the case that you know the authority that signed the authority's certificate.
            - In reality, we don't use one single hierarchy, but rather a large number of independent certifying authorities
        - There are issues with how the authorities really determine that the person should be given this certificate. What if people pretend to be different people and thus they get certificates for those other people? What do the authorities do to make sure that the people who are asking for the certificates are in fact the right people? Do I really trust that authorities criteria for validation that the person is the right person? Do I need to go to a different authority with stricter rules?
- Important note is that authentication mechanisms will prove somebody's identity but not if they can be trusted.
- Revocation is situation where we used to trust an entity, but something happened, and we don't want to trust anymore.
- Operating systems come with a set of pre-trusted certificate authorities. The system automatically trusts the certificates signed by those entities.

- There can be problems that come up if authorities are compromised. Attackers could generate a lot of certificates signed by that authority.
    - The compromise happened because of out of date antivirus software, weak passwords, etc.
- Certificates have expiration dates, which is helpful for making sure entities that are long gone aren't trusted.
- One of the problems with this whole certificate thing is that any website can obtain a certificate, but users aren't really sure which of the certificate authorities that signed a particular certificate are actually trustworthy. Therefore, we have to kinda trust all the certificates, regardless of who they are signed by.

1/24 - Week 3

- Any security protocol is a set of rules that allow two parties to communicate. They work to achieve certain security goals. We assume that the encryption and the strength of the cipher is strong. Given that, we shouldn't screw anything else up (such as getting keys from one person to another).
    - Sequence in a protocol is important.
    - These protocols should have very specific goals and only be suitable for those goals. Also, minimalism is important in the form of fewest messages, least possible data, and least possible encryption.
        - "If you only do this much, can you achieve the goal?"
- Different security protocols assume different levels of trust between participants.
- There are some types of security protocols.
    - Arbitrated protocols: Involves using a trusted 3rd party.
    - Adjudicated protocols: Trusts third party, after the fact. The party acts as the examiner that gets involved only after something happens.
    - Self-enforcing protocols: No trusted third party.
- In the former two, you have a trusted arbitrator who does not care about the communicating parties. They are disinterested and are trusted by all the legitimate participants.
    - They often simplify protocols, but add overhead in communication and protocol may be limited if you don't have that 3rd party.
- In general, security protocols should be intended for a particular goal and possibly only suitable for that purpose. An important second goal is minimalism in the form of fewest possible messages, least possible data and encryption.
- For key-exchange protocols, we need to determine how to get keys to participants securely, quickly, and even in cases where they haven't communicated before.
- Key-Exchange with Symmetric Encryption and Arbitrator
    - Alice and Bob each share a key with Trent. They share different keys to Trent. How do Alice and Bob share a symmetric key?
    - Alice sends a request for a session key to Trent. She wants to talk with Bob. (This is sent in plaintext and can be seen by others). Trent creates a new

symmetric key. We have to get it to Alice and Bob without others knowing. Trent will encrypt that key with the key that Alice originally shared and he also sends an encrypted key with the key that Bob originally shared. Basically the message has two parts. Alice can extract the session key from the first ciphertext. Then, Alice sends the 2nd part of the message to Bob, and he can then decrypt it.

- General class of security problems is a man-in-the-middle attack where an active attacker interposes themselves in a protocol, and affects the protocol.
    - In the previous problem Mallory will intercept the message from Alice to Trent which asks if she can talk to Bob. Mallory can change the message to say that Alice wants to talk to Mallory. Trent will be cool with that and he will send the shared key encrypted with K_a and with K_m. Then, once Alice gets it, she thinks the 2nd part of the message is encrypted with Bob's key. She tries forwarding it over, but Mallory intercepts and is able to decrypt the symmetric key since she has K_m. Now, Mallory and Alice share a key and Alice thinks she's talking to Bob.
- Problem with these attacks is that the third party doesn't really know what to do, and Alice didn't verify whether Trent did the right thing.
    - One solution is encrypting the first request instead of having that be plaintext. In that case, the people in the middle can't read the request and thus they can't forge or alter that request to send to Trent. Trent can also include the identity of the other participant so that Alice knows who the shared key is for.
- There is another problem with replay attack where the attacker eavesdrops on a bunch of past messages and sends those again at a later point.
    - Basically, if an attacker has stolen a symmetric key from a previous session, they they can start replaying messages after somebody tries to start with a connection with that other person again. And so instead of getting a new key from the trusted 3rd party, Alice will get replayed messages and will think that key is the one to be used.
- Key-Exchange with Public Key Cryptography
    - No trusted 3rd party. Both parties send each other their public key. Alice generates a session key and sends it to Bob encrypted with his public key, signed with her private key.
        - $\underline{\mathrm{E}_{KE_A}(\mathrm{E}_{KD_B}(K_S))}$
    - Bob verifies that the message came from Alice and then decrypts Alice's message with his private key. Then, just encrypt session with shared session key.
- When trying to do key distribution, you almost always require authentication to make sure you know who you're sharing the key with.
- Needham-Schroeder Key Exchange
    - Uses symmetric cryptography and requires a trusted authority.

    - Alice generates a random number, and sends that + person she wants to talk to + her name. That number is not the key, it's just sent in plaintext. The purpose is to help defend against replay attacks. The kind of random number is called a nonce, where it's basically evidence that this run of the program is a new run and not a replay of a previous run. Trent sends back an encrypted version (with Alice's key) of the random number + person that Alice wants to talk to + symmetric key + encrypted version (by Bob's key) of the symmetric key + person Bob is talking to, which will be forwarded to Bob. Bob also generates a random number and encrypts that message with the symmetric key and sends that to Alice. Using the same symmetric key, Alice sends back an encrypted version of that number - 1 and Bob gets it and double checks it is equivalent to his number - 1.
        - The random numbers help because they assure that the replies are fresh. They prevent the case where Alice requests a session with Bob from Trent. Mallory sends back a replay of a previous message sent by Trent (which contains all the right info - Talking to Bob, plausible key, etc) and then Alice uses it to try to talk with Bob and both of them think that things are cool, but they don't realize that Mallory can intercept messages and he can also replay old conversations by basically resending E_Ks(message).
- Timestamps can also be used to fight some of these MITM attacks. It can limit the time during which an exposed key is dangerous.
    - Problems with time stamps is that they require a globally synchronized set of clocks.
    - There is a suppress replay attack if the sender's clock is ahead of the receiver's and so an attacker can intercept it and replay it later when it is still within the time threshold.
- Handling clock problems will require you to rely on clocks that are synchronized and hard to tamper with. Also, you can make sure that all comparisons are made against the same clock, so no two clocks will need to be synchronized.

1/29 - Week 4

- Good security is based on good access control, which only works if you have good authentication. This means that you have a good system for determining the identity of some entity (process, machine, human user).
    - Need to have some notion of identity (process ID, email, etc) and some degree of proof of identity.
- Authentication is determining who you are (not question of good/bad, whether you should/shouldn't be able to do something).
    - Used in the physical world a bunch. You look at someone's face and you can identify them. For identifying people we don't know, we use things like ID cards, drivers license (ID by credential), introduction by someone (ID by

recommendation), asking things only they know (ID by knowledge) - passwords are examples -, looking at where they are located (ID by location) - person behind the counter at the DMV.
    - All the above have cyber analogs, but the difference is that the two entities involved (thing being ID'd, and the thing doing the authentication) are not human, no physical preference is required, and there is often no later recheck of identity.
- Authorization is determining what someone is allowed to do. So, you can't really authorize without authentication. The whole purpose of authentication is to make authorization decisions.
- The problems relating to authentication of computers and programs are that they have no physical characteristics, it's generally easy to duplicate programs, they don't have the ability to recover from a misidentification (once something has been authenticated, that authentication will be accepted as correct regardless of time passed), etc.
- 4 Main authentication mechanisms
    - Something you know - Passwords. Basically, this is a way of authenticating a user by requiring them to produce a secret.
        - The problems with these passwords is that they have to be unguessable yet easy for people to remember. However, because of this, brute force attacks often work on them. Also, if networks connect remote devices to computers, it is susceptible to password sniffers. Aka relying on the encryption of your Wifi network doesn't work.
            - To avoid these problems, make sure the passwords are sufficiently long, unguessable, never written down, never be shared, etc. It's hard to achieve this simultaneously.
        - Many systems will ask for the password once, but used on its own, complete mediation (all accesses to objects be checked to ensure they are allowed) is not achieved. We're trading security for convenience here. You can have a situation where someone can pose to be you if others can use the authenticated machine.
        - In terms of the OS handling passwords, OS has to check passwords when users want to log in. The OS stores an encrypted version of the password, the OS encrypts the offered password (one way function that doesn't let you go from output to input), and the two are compared. We have to remember the key in order to encrypt the passwords.
        - Also, you can have a dictionary attack where if they have an encrypted password file and they know the hash method used, then they are able to check all the words in a dictionary that they create. Instead of using an actual dictionary, you just create a dictionary based on the probability of words used as passwords, partly set up procedures, common names, etc.
        - Another problem is that if two people have the same password, then the hash would map to the same thing and thus if someone finds the encrypted version then the attackers knows info about both. Salted passwords are a solution where you add random number to the password

thus making the encrypted version different. The caveat is that the salt number has to be stored somewhere. Can be in plaintext because the purpose of the salt isn't secrecy, but rather it's uniqueness.
        - From the system POV, you can limit the amount of login attempts (which counters dictionary attacks), encrypt all the passwords that users give you (try to encrypt as fast as possible from a code POV and don't leave the plaintext password inside temp files or anything like that), protect the encrypted version of the password file, use encrypted network transport for passwords (so that people can eavesdrop), generate new passwords when old ones are forgotten (so that you don't have to store a plaintext version of the password), but transporting them is an issue so we use email of the temporary password
        - Something that can be used in addition to passwords is a challenge response system that asks for different information every time, and if the user is correct, then they are authenticated.
    - Something you have - Smart cards or tokens. Basically these are hardware devices readable by a computer. Generally this requires you to connect the authentication device to a computer.
        - Problems are that it is subject to theft and spoofing. The person who stole it can authenticate yourself.
            - In order to counter this stuff, you can use two factor authentication.
        - Plug smart card into computer, computer sends message to server, server sends a challenge, card encrypts the challenge using some hardware, and sends the message back, and the server compares the two. If they are the same, then the server knows the user should be authenticated because that smart card is the only thing that is able to properly encrypt the challenge.
            - The cryptography is performed on the smartcard hardware itself.
    - Something you are - Biometrics. Refers to things like fingerprints, voice patterns, retinal patterns, etc.
        - To authenticate, you allow the system to measure these physical characteristics and that data is converted to binary and compared to stored values, and some level of match is required.
        - The problem is that it requires very special hardware, isn't as foolproof, many physical characteristics vary too much, and when it is cracked, there aren't a whole lot of other options (only have 2 retinas).
        - This approach works well when you get really clean readings from legit users, the biometric readers themselves are secure, the attacks are rare or difficult, and this approach is used in conjunction with other authentication.
        - This approach doesn't work well because it's really hard to find needles in haystacks (lots of false positives and also really hard to spot the lone true

positive), hard to work with low quality readings, when the biometric reader is easy to spoof, and when the biometric is noisy (can cause a lot of false negatives).
    - Somewhere you are - Usually identifying a role. Requires sufficient proof of physical location.
        - Possible problem is whether the message in question is coming from a machine that's nearby,

1/31 - Week 4

- OS gives us the lowest layer of software visible to users. They have complete hardware access.
- The OS controls access to application memory, it controls scheduling, ensures processes get resources. If the OS isn't doing this stuff securely, then anything can go wrong.
    - Thus almost all other security systems must assume a secure OS at the bottom.
- Most of today's computers support a single user. However, they often run multiple processes through downloaded code from webpages.
    - There are also a number of embedded machines where there is no human user and there is no web browsing. There's no way of really detecting that something is going wrong.
- How can we make sure OS is secure? Need to make sure you're running the right OS and that it is unaltered (Attackers can try to make you go to different versions of OS's or try to make you boot a certain other OS). This is called trusted computing. Specifically, we are saying that we trust what we're running.
    - In order achieve this trusted computing, we need hardware we can trust, ensure the boot program behaves, ensures that runs the right OS, make sure the OS protects the application level, and so on up the stack.
        - Bootstrap loader is stored in a special piece of hardware, and it has persistent memory.
- We want something to check the bootstrap loader and make sure it is the right one. We created the Trusted Platform Module, which is a hardware solution that is in most computers, and it proves that the OS was booted with the right bootstrap loader by using hardware and cryptographic techniques.
    - TI;dr is that it takes a digital signature of the loader that is currently there, and then compares it to a signature that is expected, which is in memory.
    - The TPM also provides key storage and crypto support which can be used in other software.
    - Main goal of TPM is to make sure the OS is correct.
    - OS can request the TPM to verify the applications it runs.
- Transitive trust in TPM: Trust bootstrap because someone claims it's cool -> trust OS b/c bootloader -> trust app b/c OS says to trust it.
- TPM does not guarantee security, but rather just verifies trust.

- SecureBoot does relatively the same thing as TPM. It's designed by Microsoft. It's just going to make sure that you boot the right OS, while TPM does a bunch of other checks. It only boots systems with prearranged digital signatures (they are issues on who chooses what signatures are okay - Linux, Windows, etc).
- TPM and SecureBoot uses the general theme of hardware security enclaves where you build elements of computer security into hardware.
    - A hardware security enclave refers to some hardware that is typically tamper-resistant. The OS will rely on this hardware (login security).
    - It is a great place to store private keys.
- iPhone security enclave stuff: There was a key inside of the enclave, HW only provides the key on login attempt.
- OS needs to authenticate users. Humans can log in and processes will then run on their behalf.
    - In person authentication: From the OS POV, they need to verify that the person you is physically using the device is the person. Passwords and biometrics can be used.
    - Remote user authentication: Passwords also used, sometimes through public key cryptography.
    - Applications and web servers/browsers sometimes have to do authentication instead of the OS.
        - Web server security has to be good because when you visit multiple sites, they send content back, and that content normally will have executables and scripts. For each of the websites, a new thread gets forked. There must be security between the threads even though they're in the same process. A solution would be to spawn new processes where data is separate but you'll get a performance hit.
- A successful login creates a process under the ID of the user who logged in.
    - Processes can fork off more processes and child processes get the same ID as parent.
    - Only system calls can change a process' ID.
- In terms of protecting memory, we have to protect page tables and virtual memory. Gotta protect this stuff because we have executable code, process data, and confidential data stored in this memory.
- We're able to protect this memory by having a logical separation of processes that run concurrently.
- Memory is divided into page frames, the processes have address spaces divided into logical pages. For a process to use a page, it has to be in a page frame. There is a protection of pages since processes are given page tables that translate logical to physical pages. Thus, the process can only access its own pages.
    - The virtual addresses that a process has are all mapped to physical addresses that don't overlap.

- Reusing pages refers to one process having access to a physical page, not needing it, that page being deallocated, and then another process needed a page and then having that deallocated page map to it.
    - Strategies for cleaning the content of the pages include zeroing on deallocation, zeroing on reallocation, zeroing on use, etc.
    - Another strategy is to clean pages in the background.
- There are also some systems with special interfaces into memory, which need to be protected. These interfaces can possibly bypass page table protections and just go directly to the physical memory. This is something attackers can exploit.
- In buffer overflow, the content that overflows the buffer will go onto the stack, right after the buffer, which overwrites the current function and instruction pointer, which lets the attacker return to an arbitrary address allowing them to run different code than expected.
    - The effects are that an unprivileged local user runs a program with greater privileges because if the overflow is in a root program, it gets all the privileges. Another effect is that data and memory can be overwritten.
    - The reason these buffer overflows happen is that the C calls copy data into a buffer without checking if the length copied is greater than the size of the buffer.
- Stack overflows are the most common buffer overflow. This is intended to alter the contents of the stack with the intention of jumping to exploit code.
- Heap overflows are less common. Heap is normally used to store dynamically allocated memory. These heap overflow attacks can alter variable values, modify where function pointers point to, but they cannot really jump to arbitrary code.
- Fixing buffer overflows involves
    - Writing code with input validation
    - Using languages that prevent them with bounds checking (modern languages instead of C, C++)
    - Add OS controls that prevent overwriting to a stack
    - Don’t allow execution from certain places (certain pages in memory can be writable but not executable or vice versa) - also called data execution prevention DEP.
    - Address Space Layout Randomization (ASLR) which means to randomly move around where things are stored (base address, libraries, heaps, etc). Problem is that you gotta keep track of where everything is located. However, it does make it harder for the attacker to know.
    - Modify the location of the return pointer.
- OS also needs to protect the communication between different processes (they can communicate through messages, semaphores, shared memory, sockets, etc). If process A wants to steal info from process B, then they can
    - Try to convince OS that the process is really B
    - Break into B’s memory
    - Forge a message from another process C that is allowed to get the secret.
    - Eavesdrop on a process like C that is getting info.

- Covert channel is a situation the requires cooperation of sender and receiver. This is normally process to process.
    - From OS POV, easy to deal with if you deal the details about how the channel is used since you can add randomness/noise into the channel to wash out the message.
- Files are the most common example of a typically shared resource. In order to make these safe, we need to use encrypted file systems. The issues come when you think about when the cryptography needs to occur, where the key will come from, granularity of the cryptography, etc.
    - When cryptography needs to occur: Possibly when the user opens the file or by a user command?
    - Where key comes from: Provided by humans, stored in file system, on another computer?
    - Granularity: Encrypt an entire disk, per file, per block?
- Often times, with crypto file systems, you can get the same protection through OS access control (if you're worried about unauthorized access), or network encryption (if you're worried about sending data over network).
- Full disk encryption is the act of making all data on the disk encrypted, and the data is encrypted/decrypted when it enters and leaves the disk. This prevents you access to a disk if it has been stolen.
    - Hardware disk encryption is faster and transparent, while it is more expensive and might not fit into certain machines.
    - Software is much more common.

2/5 - Week 5

- There is a lot of security problems with networks.
    - Wiretapping since the path for messages is really complicated and thus some people have the ability to listen in.
    - Impersonation: Did the message come from the source we think it comes from? Is it a good packet or not? To determine the identity, we look at the source address (which can be spoofed) or can use cryptographic techniques (in order to sign a message that can only be signed by one party).
    - Attacks on message: Modification of the message (refers to general concept of message integrity) is where the attacker can alter messages and this normally requires access to part of the path that the message takes. Also the messages can be read at intermediate gateways and routers. Attackers can also get useful info just by looking at traffic analysis.
    - Denial of service attacks: Prevent legitimate users from doing their work by flooding the network/routers or corrupting routing tables. Attackers do this by injecting some form of traffic to some IP address.
- One DDoS attack is the Syn flood which is where the attacker uses the initial session and prevent new real TCP sessions.

    - The attacker will send a SYN message, and the other person will send a SYN-ACK after it get it (You also add another entry to your table of open TCP connections).
    - Solution is to limit how many entries from one source get an entry on the table of TCP connections.
    - Server cookies is also another solution. The machine sends back a SYN-ACK (with a cookie) after it gets a SYN. there is not change to the table,
- General DDoS attacks can see success with the mere volume of the traffic. The attacks are made to annoy, for extortion, to prevent adversary from doing something important.
- DDoS attack methods can involve pure flooding of the network connection or overwhelming of other resources (like CPU or memory resources).
- The distributed part of DDoS refers to the characteristic that lots of machines are being used to send messages to overwhelm a server.
- Defending against DDoS attacks involves not just stopping the flood of messages since you have to ensure service to legit clients. If you choose to deliver only a limited number of garbage, you likely wouldn't be responding to the legit packets. You also can't drop all the packets that you get.
- The tough parts about this problem is that you have a bunch of bot machines that can send you messages, the Internet is designed to deliver traffic (regardless of whether it is good or bad), and IP spoofing makes it easy to hide where you're coming from.
- The main defense approaches are:
    - Overprovisioning: Buying a bunch of servers and machines to respond to requests.
    - Dynamic increases in provisioning: If you are under a DDoS, then have the ability to add more capacity. This is effective if you can afford it.
    - Hiding: If people don't know where you are, then they can't send messages to you.
    - Tracking attackers: If you know all the attackers, you still can't do anything about it.
    - Legal approaches
    - Reducing the volume of the attack: Try to get rid of the garbage traffic and just respond to the good packets.
- Reflector attacks are where the attacker sends packets to a 3rd party instead of the target. That 3rd party sends info to the actual target. This is good for the attackers since the packets arrive at the target from a bunch of source IP addresses. Also, the reflector's response can be a lot larger than the first message (and thus this leads to signal amplification).
    - Common types of reflectors are DNS servers that can turn small requests into large results, NTP which is a protocol flaw that allows amplification of 200x.
- In terms of defense, there are also traffic control mechanisms
    - Filtering: Filter some packets because of their source address values (usually because you think their source address is spoofed).

        - You can also drop outgoing packets with source addresses not in a particular range because network shouldn't be creating packets with this source address.
        - Also, when packets are leaving the internet and entering the router and then get routed to a particular machine, then you look for particular cases where the src and dest are your own addresses. This is an indication the packet is spoofed because that message should have been been sent locally.
        - You can also filter based on other things besides source addresses. There are worm signatures, protocol identifiers, unallocated IP addresses in IPv4 space, and some source addresses are for local use only.
        - In reality, these limits are hard to implement because the packets are handled so fast and we don't want more overhead.
    - Rate limits are also used in that routers can have limits on the traffic they send to a destination. This is helpful for limiting the amount of info, but not good enough to differentiate between good and bad traffic.
    - Padding: In order to stop attackers from knowing traffic patterns, patterns add extra traffic to hide the real stuff.
    - Can also use routing control to conceal the traffic in the network. Specifically, onion routing is used to hide who is sending traffic to whom.
    - Firewalls: Controlling what traffic comes into the network. It sits in between a LAN/WAN and the internet.
        - It is a form of security that is called perimeter defense, which defends the outside strongly.
        - The firewall needs to look at every packet, and make decision to either deliver and drive.
        - The problem is that breach of the perimeter compromises all the security.
    - Defense in Depth is the follow up solution to the above problem where you don't rely on a single defense mechanism (shown through a firewall) and instead have multiple layers.
- Different types of firewalls
    - Filtering Gateways is a stateless firewall that primarily look at packet header info (IPs, port numbers, etc), and accept/deny packets. They can also be configured to filter based on packets that are going to particular ports.
        - This type is fast, cheap, and transparent. However, it is dependent on header authentication and relies on router security.
    - Application Level Gateways is a type of stateful firewall that understands the application level details of the traffic and that traffic is accepted or rejected based on the probable results of accepting it.
        - Packets are treated as flows instead of individually. These gateways are designed to handle common types of traffic like TCP and HTTP. Based on the type, packets can be handled differently.

        - The pros are that it is flexible and offers content based filtering, but it is slower and more complex and expensive.
    - Reverse Firewalls are ones that keep stuff from the inside from getting outside. Can be used when we want to conceal details of the network from other people and prevent compromised machines from being able to send confidential information out.
- Deep packet inspection refers to looking into packets beyond just their headers.
- There are also firewall proxies that are capable of understanding types of traffic.

2/7 - Week 5

- Stateless Firewalls - Doesn't remember anything about past packets. The decision about whether to let it through is just based on the individual packet you get. The pro is that it is cheaper (don't have to save state) but the problem is that you can't understand the implications of delivery if you don't know the context or the packets around it.
- Stateful Firewalls - Since a lot of network traffic is connection oriented, the firewall will remember the state of each connection (source address, port, etc). The plus side is that proper handling of connections is possible, but handling the information about the connections is complex.
- If firewall is stateless and there is an incoming packet that looks like the 2nd part of an HTTP request, then you cannot know if it's a good packet unless you know about the first part.
- Firewalls should be invisible. Users inside should be able to communicate outside and external users should be able to invoke internal services transparently.
- A lot of systems give special privileges to specific sites or users. Firewalls can support this as long as you have authentication capabilities.
- Firewalls don't provide confidentiality unless the data is encrypted. But the firewall needs to be able to decrypt it to do its inspection and figure out if the firewall should send it through or not. However, now the firewall needs to have access to the key, which means it's one more party that has the super important key. Also, the firewall has to do the computational work in dealing with the decryption of the packet.
- Firewall can be a point of attack and this it needs to be secure.
- Need to make sure that the firewall is in between you and the attacker. This is hard because you have so many different machines with different functionalities. Solution is to figure out what is exposed to the outside world and what is only internally accessible. Typically, we'll use less secure public network, more secure internal network, and separate firewalls.
    - Demilitarized zone (DMZ) refers to way to configure multiple firewalls for a single organization.
        - One example is having a strict firewall between your LAN and the internet, and having a loose firewall between web server and internet. The DMZ is the location in between the two firewalls.

        - The advantages are that you can customize the firewalls + traffic analysis for different purposes, keeps less safe traffic away from critical resources.
        - The disadvantages are that things inside the DMZ itself is not that well protected, and people can poke around in between the firewalls.
        - But you should use DMZ because it makes it harder to get to the easy target and limits damage when one is compromised.
- Firewall hardening is the decision to devote a special machine to only firewall duties. We alter the OS operations on that machine so that you only allow firewall activities and you only allow strict access to the machine.
    - Problem is that you have to update files and software stuff and go to the actual machine and do it. System has to be up to date since security is so important. New vulnerabilities are discovered all the time, and thus updating the firewall is crucial. Sometimes you need to open a port on the firewall so that these updates can be downloaded and applied. However, you need to make sure to close those opened doors.
- In firewalls, we need to close the back doors. Firewall security is based on the assumption that all traffic goes through the firewall. So must be careful with wireless connections, portable computers, modems, etc. Therefore, need to put a firewall at every entry point to the network or make sure to direct all the traffic to the one main firewall.
    - In public Wifi networks, you can go from your machine to the access point and then go to one of the other machines connected to that same access point. Then let's say that person who get infected goes back to office which is protected by a firewall. However, it will infect those that are connected.
    - The solution is to try to quarantine the computer until you know it is safe.
- Single machine firewalls are where instead of having a separate machine protecting the network, you have a single machine that puts software between the outside world and the machine. Therefore, in the previous case, we would never have to prep for the scenario where you can access other computer connected to the same access point.
    - The pros of this approach are that you can customize the firewall, you are the owner, can do deeper inspection since you have access to specific parts of the OS. It also provides defense in depth.
    - The negative sides is that this firewall only protects one machine and it is less likely to be properly configured.
- Encryption and Networks Security: This is basically using the same types of encryption algos just at different places in the network stack.
- Link level encryption refers to encryption across a set of links (physical connection between a sender and receiver) between a source and a destination. Each link decrypts, and re-encrypts and sends to the next link.
- End to end encryption is where the message gets encrypted at the beginning link and decrypted at the last link. The messages sent between the inner nodes are encrypted. However, in this case, the con is that the headers have to be in plaintext (since each link in the middle cannot decrypt that message that it is seeing). The plus side is that you have less computation that you have to do at each link.

    - Question to think of here is the definition of what the endpoints are.
- Another option instead of link level encryption, we can do cryptography at the network layer of the IP stack. IPsec is the standard for doing this. This allows options for encrypting and authenticating packets. With IPsec, we get message integrity, authentication, and confidentiality. IPsec is designed for end to end encryption and has sub-protocols like ESP.
    - Encapsulating Security Payload (ESP) protocol is the act of placing encrypted data within the ESP, which has normal IP headers.
        - Encrypting just the payload of the packet is transport mode. The original IP header is not encrypted, the packet payload is encrypted, and the ESP header is authenticated.
        - Encrypting an entire IP packet is tunnel mode. The original IP and the packet payload are encrypted, and the ESP header is authenticated. Tunnel mode is used when there are security gateways between the sender and the receiver.
    - There is also a term called a Security Association which is a simplex connection that affords security services to the traffic on it. Basically it is a secure one way channel. An SPI (Security Parameters Index) is something that uniquely identifies an SA.
- SSL and TLS do end to end encryption at other different layers, basically they secure network applications. TLS is an improved version of SSL, but they basically do the same thing. SSL/TLS is the core crypto for web traffic and are what we use to protect web browsing.
    - SSL is a client server operation where client contacts server, authentication and key exchange take place, and then the packets are encrypted with that key and cipher at the application level.
    - The authentication is done by the server communicating to the client using a certificate, client then gives the server stuff to derive a session key, and once they both have knowledge of the key, then they can start sending encrypted packets.
- Difference between IPsec and TLS is that IPsec is in between network and transport layers and secures packets but not connections. It is usable with any transport layer. TLS, though, is above the transport layer and it only works with TCP.
- Virtual Private Network (VPN) is something that allows secure cooperation between offices that can be far apart. We use encryption to convert a shared line to a private line, and set up a firewall at each installation's network. The messages sent in between are encrypted with keys that the firewalls at each installation has knowledge of.
    - Security of the VPN depends on the key secrecy.

2/14 - Week 6

- Wireless networks are like a lot of other networks, except they are short range, almost always broadcast (anyone who's listening in the vicinity will read the message), support

mobility, may see a machine you haven't seen before, and are very open (anyone can use free wifi).

- Types of Wireless Networks
    - 802.11 networks which are the equivalent of Ethernet. Basically small range of connectivity.
    - Bluetooth networks: Very short range and exclusive 1:1 connection.
    - Cellular phone networks
    - Line of sight networks: Good for high rise buildings which allow two groups to network between each other.
    - Satellite networks: Very very long distances
- Wireless networks use encryption for security, specifically link encryption which is where you encrypt the traffic just as it cross the network. Encryption starts at your phone, and then the decryption will happen at the wireless access point (thus it can see the plaintext of your packets).
    - HTTP has end to end encryption because you encrypt/decrypt at the browser and the web server.
- 802.11 protocols were developed a while ago, and they didn't really include security.
    - Wifi became really popular, but we figured out the security issues pretty late. There are now a huge number of hardware units in the field, and thus it was hard to change this hardware. We couldn't change the protocols.
- The first solution was Wired Equivalency Protocol (WEP). Claimed to be just as good as a wired connection. It provided encryption but didn't change the 802.11 protocol. WEP used a stream cipher with 104 bit keys (key doesn't change a lot since then you would have to notify everyone connected). The problem was that creating the session key from the permanent key + IV was not very secure. WEP was cracked in 2001 in a minute.
- The next solution was WPA and WPA2. The modes are TKIP (there are vulnerabilities) and AES (not cracked yet). Tl;dr use WPA2.
- To make wireless networks more secure, you can combine link and end-to-end encryption. Wi-Fi networks encrypt packets crossing the link and network interactions encrypt at the source and decrypt at the destination.
    - So basically the wireless access point and 802.11 only provide link level encryption while HTTPS uses end to end encryption. So really when you're using the internet, you have double encryption.
- Honeypot is a machine set up to attract attackers, and you want to learn about the attackers and what they are trying to do. It needs a lot of software that is watching what is happening to it. Put the machine outside your firewall so that it is accessible. However, it is not clear as to how much we learn from the logs of the attacker's activity.
    - A honeynet is a collection of honeypots. It could be a bunch of machines or a bunch of VMs. No other machines will be on the network. All the traffic that is incoming is probably attack traffic since this isn't actually a useful network that people can use. These are useful for learning about attacker activity on the network level. It is useful for analyzing the behavior of botnets. Useful for looking at how they communicate and what type of information changes. Honeynets also

give evidence of DDoS attacks through backscatter because the network will send the packet backed to a probably spoofed address. If that spoofed address is yours, then you know something is up.

- If you catch a bot, you analyze it to look for similarities with other bots, etc
- Honeypots are not as useful to a local area network as a firewall is. It's only helpful if you want to take a look at the honeypot logs, which isn't as high on the priority list.
- In general, protecting against intruders involves access control, firewalls, authentication, etc. However, they all don't work at all times. Therefore, it will be useful to detect improper behavior to know when someone has broken through the system. These are intrusion detection systems.
- Detecting bad behavior is different from preventing them since the latter is expensive and may involve things that we didn't foresee. However, can we detect the effects of the attacks we didn't see.
- These intrusion detection systems will look for activity like setting uid on root executables as being suspicious. In this situation, the real problem was that someone got root access.
    - Also, you can have a whole lot of false positives in intrusion detection.
- Intrusions are hard to stop since it involves any attempts to compromise the integrity, confidentiality, and availability of a resource.
    - External intrusions are where you have someone not allowed to use your machine trying to influence/use your machine through some vulnerability.
    - Internal intrusions are where you are already an authorized user on the system but that you're trying to gain privileges beyond what you have (can't de-enroll other students). This isn't the majority of problems now (~15%), but it is more dangerous since insiders have a lot more info.
- Intrusion detection involves watching and identifying behavior that characterizes intruders and avoid improper detection of legit access. If there are false positives, you waste time investigating things that aren't problems.
    - Detecting behaviour involves looking at packet, the processes run, etc. You run algos on the data that you acquire, and this uses network bandwidth and compute power and memory to do that intrusion detection. Most of the time, we spend the cost of looking of legit behavior 99.999% of the time.
    - Detecting behaviour involves examining logs that keep track of what's going on on the machine. The logs grow and grow for a long time, and thus the intrusion detection can also reduce the length of the log by getting rid of the information that isn't helpful anymore.
- The time when you do this intrusion detection is important because it takes up compute power, network bandwidth, etc.
    - Off-line intrusion is when you do the detection program at some set time, like 3 AM. This allows you to run more complicated algos, but can find intrusions way too late.

    - On-line intrusion is when you run the detection programs at all times, which could be computationally expensive and can affect the performance of other tasks, but you'll catch the intruders asap.
- Failures in intrusion detection involve FP, FN, and subversion errors where there are exploitable errors in the intrusion detection system themselves. Attacker couldn't get into anything else on the system but they could get into the detection program.
- Intrusion detection needs to be continuously running, resistant to crashes (fault tolerant), resistant to attackers (subversion), minimal overhead, observe abnormal behavior, adaptable to different OS/systems, and is difficult to fool.
- The intrusion detection system can be run on the host computer, and the program will examine the logs of that computer in particular.
    - The pros are that you have lots of info to work with, you only have to look at this computer, and the info is in readily understandable form.
- Another option is to do network intrusion detection where all the activity on a LAN is examined. This is done by looking at network traffic and using distributed system techniques.
    - The pros are that you don't need to use any resources on the other user machines. Instead of having software running on N number of machines, you only have to run the software on one machine which is thus easy to configure.
    - However, one issue is that there's a lot of info that pass on the network, and thus you need to figure out what are the right things to process. Sensors are the components that grab only the relevant data. They examine network traffic and record the relevant stuff to forward to the intrusion detection system. This reduces the data volume that the intrusion detection system has to account for. The sensors also help to determine which packets are worth doing deep packet inspection (where you examine more than just the header) on.
- A wireless intrusion detection system observes the behavior of a 802.11 wireless network. You can look for problems specific to the environment. The issue is that you only understand the attacks on link layer protocols, rather than the higher network protocol layers.
- You can also have application specific IDS and you can run on host or the network. The advantage is that you can use these for specialized machines (web servers, database servers) and you will likely have lower overheads than general IDS systems since you're looking for very specific behavior.
- The 3 approaches to IDS
    - Misuse intrusion detection: Detect things known to be bad. Figure out the undesirable actions, look for those, and signal when they happen.
        - Ex) You could look for a bit pattern that can come up in some piece of malware.
        - You could look for specific attacks (like SYN floods) that you already know about. However, it's also more effective to look for known suspicious behavior since there could be a million types of attacks that you don't know about.

        - In order to look for this misuse, you can look at logs, system activities, state of the system (location of certain files), sniffing the network, etc.
        - This approach has few false positives since you know exactly what you're looking for, it's simply, and it's hard to fool.
        - However, the main issue that you can only detect known problems, and in security, a lot of problems are unknown.
        - Most commercial intrusion detection systems detect misuse.
    - Anomaly intrusion detection: Detect deviations from normal behavior. While misuse only finds known problems, this approach will hopefully find new attacks based on examining the behavior of a normal system, and then detecting times when there is some anomaly.
        - In order to do this, we can use statistical models to determine what normal user/program behavior is.
        - The best pro side is that this could detect previously unknown attacks. These are also more flexible in that the attack could change in small ways and get past the misuse system, but it won't get past this one since the effect is still the same.
        - However, it's hard to diagnose the nature of an attack since you just kinda look at events, notice that they aren't normal, but then you have to figure out what to do.
        - Most research is in this area but not a lot of current systems use it since we haven't really gotten it to work perfectly.
    - Specification intrusion detection: Define good states of the system and detect when the system is not in those states. Different from misuse since that says that certain things are bad (which we don't, specification kind of says the opposite). Different from anomaly since that focuses on differences from normal behavior (when specification defines what's good and calls the rest bad).
        - One of the big questions is determining what the important state is for you to look at and how to define whether that state is good.
        - Pros are that it is very formal way of you saying what good states are and wanting the system to be in those states. This means you can detect unknown attacks since those will likely move the state to something "not good".
        - However, it's hard to specify the correct state and there is a possibility that attackers can do what they want without changing from a "good" state
- Intrusion detection systems need to be right for you and they cannot be static, they have to change along with your requirements as an organization.
- Systems need to evolve
    - Manually - Humans are involved thus having restrictions on how often you can change things.
    - Automatically - Attackers can try to evolve the system in a certain way to accept future attacks.

- Intrusion detection systems need to be pretty quick. However, it also can't be too sensitive since there would be too many false positives. Same issue with false negatives and not being sensitive enough.
- Once the IDS system goes off, there could be an automated response which could be turning off a particular service because you're afraid it's being attacked which doesn't let you continue normal operation if you need that service. If you have a manual response instead, then you get a bit more flexibility in the response (ignore or investigate) but it's manual which means it'll take more time and energy.
- False negative rate and false positive rate are at odds with each other inherently.
- Intrusion prevention system is basically an IDS that takes automatic action when an intrusion is detected.
- There are different types of IDSs. Some that are host and some are network and they give you different capabilities.
- There are 3 main ones: Snort, Bro, and RealSecure ISS.
    - Snort: Network intrusion detection system which has rule based description of good and bad traffic.
    - Bro: Similar to Snort, but more sophisticated non-signature methods.
    - RealSecure ISS: Distributed client/server architecture that incorporates network and host components
- IDSs have a lot of issues and they aren't that sophisticated but there aren't a whole lot of alternatives. And so we use IDS since it will fight most of the simple and well known attacks.
    - Good to use multiple IDSs since each will detect different things, and it's good to have multiple types (host, network, etc)

2/19 - Week 7

- Malware is the creation of software to do malicious tasks. The advantages are the speed, mutability, and anonymity (software is attacking you, not a particular person).
    - Malware shows up on your machine because you've unknowingly imported it into your system through mail, executables, shrink wrapped software, etc.
    - However, sometimes it's an insider who puts it into the system or there is a system vulnerability.
- Virus is one type of malware and it is a self replicating program that can infect other programs by modifying their environment. It is always attached to some other existing program (virus code gets attached at a binary code level) and when that program runs, the virus becomes active and infects other programs.
    - When a program is run, it is run with privileges which can write privileges for other programs. The virus can copy itself onto other uninfected programs.
    - Preferably you want the virus to get attached to a program that is being run by a super user so that you can write to any other file.
- A lot of viruses come in the form of pieces of code that attach to data files - macros and email attachments. These files contain executables that could be dangerous.

    - What looks like a data file can actually turn out to be data that is accompanied by a program.
    - We shouldn't really want arbitrary execution of code.
- There are virus toolkits that make it easy to create viruses for anyone and there is a low barrier of entry. This allows other hackers to build viruses simply. However, the viruses created by these are pretty easy to detect since we can analyze the toolkit and figure out a defense.
- Finding viruses involves
    - Basic precautions: Don't import untrusted programs, download from the commercial source, scanning programs for viruses, limit the targets viruses can reach through access control or VMs, and monitor updates to executables which aren't that frequent.
    - Containment: Running suspicious programs in an encapsulated environment so that it limits virus spread. But then, you have to be 100% sure that the code running in that VM can't escape.
    - Looking for changes in file sizes (since viruses are pretty much additional code onto programs): If the code is added naively, then the size of the executable will increase. However, it doesn't work for files whose size normally changes.
        - There are also cavity viruses where code can get inserted into locations where there were originally just no-ops or zeros.
    - Signature scanning: If there is a virus in an executable, we can search for specific signs of it. The traces are characteristics code patterns. We find the virus by looking for the signature.
        - In order to do this, we create a DB of virus signatures, reading a bunch of files in the system (also sometimes have to scan boot sectors and other interesting places), looking for matches. The problem with this is that it is slow since the operation of reading data off of disk is always slowing than the CPU.
        - The weaknesses of this approach include situations where the virus changes the signature, you can only scan for known signatures, and there are viruses that mess up the OS that prevent you from finding the signature.
    - Checksum comparisons, intrusion detection methods (looking for specific virus behavior instead of what the virus code looks like), looking for attack variants similar to signatures, and identifying clusters of similar malware.
- There are polymorphic viruses that replicate in slightly varying ways in order to slightly change the signature. There's also a manual approach to do polymorphism by knowing which signatures are known by the defenders and changing the virus accordingly.
- Stealth viruses are ones that try to hide signs of its presence. Fighting these involve a careful reboot from a clean source. However, there could be peripheral memory (that doesn't get affected with a reboot) that is infected with malware.

- Preventing virus infections involves running virus detection programs, keeping virus signature DB up to date, disable programs that run executables without asking the user, and making sure users are careful about what they run (aka educating users!).
- When malware infections happen, you can reboot from a clean, write protected medium, replace infected files with clean backup copies.
- Trojan Horses are programs that are seemingly useful but contains code that contains harmful things.
    - Remote Access Trojans are ones that allow the creator to remotely access a machine. There isn't necessarily a malicious action being done, but rather the goal is getting a foothold on a machine or network.
- Trapdoors are secret entry points into legitimate programs. Found in login systems and system utilities.
    - Malware can take over a machine and can insert trapdoors, which allow attackers to get back in even if the original point of entry is closed.
- Logic bombs are in legit programs and refers to code that explodes (do very malicious behavior) under certain conditions.
    - People always get caught and go to jail though.
- Ransomware and extortionware refers to situation where attacker breaks in and demands money to undo it.
- Worms are programs that seek to move from system to system. Similar to viruses. Once they get onto computers, then they can install trapdoors, perform DDoS attacks, etc.
    - The famous worms are the Internet Worm (1988), Code Red and Code Red II (2001), and Stuxnet.
- Difference between all these: Trojan horses are seemingly good program with bad code, viruses infect other programs, and worms try to go from machine to machine.
- Botnets refer to collection of compromised machines that a single person controls.
    - They are a form of malicious software (distributed system software) that allows the attacker to organize a group of compromised computers to perform a more powerful attack.
    - The machines in the botnet have to communicate with each other. They use tech similar to file sharing systems.
    - There are some reliability issues if people realize that their machine is being compromised and thus from the attacker POV they have to have some mechanism of dealing with a computer going offline.
    - There is also a lot of security that goes into making sure that nobody else can take control of the botnet.
    - You want to spread the botnet as much as possible, and you do this through phishing and trojan horses.
    - Botnets are characterized by the number of nodes or machines that were compromised and used for malicious intent.
    - It is difficult to defend against botnets because of the number of machines that are being used, the fact that the attacker's actual machine isn't in the botnet (and

thus no way to track them down), legal and international issues, and then the handling of huge numbers of infected nodes.
        - Some approaches are that you want to clean up the nodes (but we cannot force people to look into it and fix it), actually interfere with the botnet operations, or to not communicate with any machines you know are bot nodes (but some activity on these computers are legitimate).
- Spyware is another type of malicious software and used to gather information on the activities of the computer's owner through a software that is installed on the computer. It can go through files and report the info back to the owner of the spyware.
    - Designed to be very hard to remove and hard to detect. Make it look like this is not a compromised machine.
    - Gathering of sensitive data like passwords and credit card numbers is the main goal of spyware.
    - Sometimes there isn't any malicious intent, though. For example, observations of user activities to allow for targeted advertising.
    - Spyware comes from the user installing something without knowledge of the impact. It can also be part of the payload of worms.
- Malware has two generic components to allow it to be easier for other people to create those components and thus their own malware.
    - Dropper - Piece of code that runs on the victim's machine and the purpose is to fetch a more complex piece of malware from somewhere else.
    - Rootkit - Software designed to maintain the illicit access to the computer. Installed after you get root access on the system and the purpose is to continue that continued privilege, and make it extremely difficult to remove.
        - Often means replacing system components with compromised versions

2/21 - Week 7

- Writing secure software involves figuring out what your security goals are, and using them to be part of the whole software development process.
    - One of the problems is that people say that they will add security later. Retrofits are very hard and insecure designs offer too many attack opportunities to fix. Tl;dr designing security from the beginning works better.
- Some security goals could be to think about if you need limited access for certain users (that are connecting to a webserver for example), if you have privacy issues (if you're collecting data and storing on the machine, then you need to protect it), and if availability is important.
- Also, when building software, you need to actually achieve the original goal of the software. You care about security, but more about the original goal. Security is a secondary goal. You have to think about the degree of security as it relates to how much risk you're willing to take on. This also relates to the amount of work you want to put in.

    - You need to look at the risk the software can tolerate, but this often comes with compromises on the other goals that conflict with security, and you need to make trade-offs.
- Spiral model of SW development involves determining objectives, identifying/resolving risks (which is where you include security risks along with the other risks), development and testing, and then planning the next iteration of the cycle.
    - Determining the risk involves thinking about possible issues
- Good security principles.
    - Secure the weakest link because attackers will look for the attack that is the easiest. Basically, concentrate on the most vulnerable elements, and think about the easy attacks that attackers can make (they're more likely to try a buffer overflow rather than breaking cryptography).
    - Do defense in depth where you don't let one security breach bring everything down.
    - Fail securely so that even if something goes wrong, the attacker isn't able to do anything.
    - Use Least privilege, aka give minimum access necessary to users for the minimum amount of time needed.
    - Compartmentalize aka divide the programs into piece and don't let compromise of one piece compromise the others. Set up a limited number of interfaces between pieces.
        - Bad compartmentalization example is where getting root privilege gives away the whole system and so if a mail server (which had to be run as root) got compromised then the system is lost. The solution is to allow root programs to run under other identities.
    - Value simplicity because complexity is the enemy of security. It's harder to understand all the behaviors of the complex system and thus there are more opportunities to screw it up.
    - Promote privacy because losing customers private data is really bad. Don't ask for data you don't need and avoid storing the data permanently since the more time you have it, the more of a chance it gets stolen.
- Some other tips: Re-use your secure program components, use one implementation of encryption, build only the code you need, choose simple over complex algos.
- When creating systems, need to remember that users won't read documentation, users are lazy, etc.
- When you choose different technologies (OS's, languages, libraries, etc), they have different security properties.
    - In reality, you don't choose the OS (SE Linux is the most trusted though), but try to exercise choice in other areas.
    - If you have the choice of programming language
        - You should not choose C/C++ because they are susceptible to buffer overflows (but they're efficient af since for example they're not checking

the bounds on your arrays so it's a tradeoff between performance and security).
        - Java is a bit better with better error handling and security features. Has its own issues with exception handling and inheritance issues.
        - Python: You won't have buffer overflows and has a lot of extensibility (YAML, pickles, import paths) which means you can add more code to the program which could allow for attackers to get malicious code into your program.
        - Scripting languages: There is a lot of variability. Perl is good, but Javascript and CGIbin are bad. The general issue for all of them are that they are interpreted languages and thus the interpreters could have security flaws. The scripts are also visible and are very high level and understandable to everyone (while in compiled language, you just have the compiled machine language code)
- Open source vs closed source in terms of security: Some say open source better because of the many eyes which means more people will look at it, find bugs, and increase the security. However, it's not really true because not a whole lot of people are looking at it. Also, attackers can look at the source code and try to find flaws.

2/26 - Week 8

- Attackers can try to attack error handling code (provide input that causes the program to get an error), since that code gets run very rarely and thus could be untested. The problems could be not cleaning everything up, variables not getting reset, program continuing with old values, and could cause security mistakes.
- Good practice is to check the return codes of all the functions that you call. It's dangerous to go ahead if it turns out your call didn't work properly.
- There are certain programming areas that are more prone to security problems.
    - Buffer overflows
    - Verification of input: Need to make sure that user input is safe. There are buffer overread problems which is basically where another user sends you a message ("hello") and a buffer length of 512. You'll get this message and return the hello plus the next 507 characters in memory. The main problem is that someone sent you a packet with purposely incorrect fields.
        - Solution is to update the software to check the buffer length in the heartbeat messages.
        - The impact of this was that it could be used to steal private keys associated with certificates.
    - Checking return codes: Want to do this because it is dangerous to continue on with your plan if some function you called before didn't work.
    - Privilege escalation is where some program parts can get expanded privileges when they shouldn't have them. Basically it refers to attackers using program flaws to obtain greater access privileges they shouldn't get.

        - An example program is where you have a program that calls setuid and then forks a process that has the same root privileges.
        - Solution is to not run as many programs with those high privileges, switch back to the real caller as soon as you can, and compartmentalize.
        - Another solution could be to run some in a VM and only give access to the safe stuff, but it's hard to determine what is safe, and VMs might not have perfect isolation (attackers can figure out that they are in a VM).
    - Race conditions usually involve multithreaded programs and they are caused by different order in which the threads are executing.
        - Types of race conditions are file races, permission races, ownership races, and directory races.
        - Another type of race condition is dirty COW vulnerability and basically allows unprivileged users to write read only memory. Happened because two people are sharing a piece of memory. Instead of creating copies, OS tries to just keep one copy and wait until somebody changes it, and then gives copies. There's some problem where one person will get privilege escalation.
        - Preventing race conditions involves minimizing the time between time of check (to see if someone is allowed to do something) and time of action (based on that authentication), being careful with files that users can change, and use locking.
            - The big problem with race conditions is that people will try to slip in execution in between other actions.
    - Randomness and Determinism: A lot of things require randomness (generating keys) and so how do we get that randomness?
        - Common solution is pseudorandom number generators which refers to mathematical methods that produce a string of random like numbers, but it's actually deterministic. Problem because if the attacker can get hold of one random number, they can generate the rest of the numbers.
        - The attacks on these PRNGs are cryptographic ones where you observe a stream of numbers and try to deduce the function. There are also state attacks where the attackers gain knowledge of the internal state of the PRNG (some sort of key/input that the algorithm is using to create the sequence of numbers). Instead of gaining the knowledge, attackers can try to influence you to use certain state.
            - An example of the gaining knowledge one was with a card shuffling algorithm and it was seeded with a clock value that could easily be obtained.
            - A lot of problems stem from the key/seed value being easy to obtain (from the browser or some user info).
        - If you want to go away from PRNGs, you can try to use hardware randomness.

        - Other tips for PRNGs are to not use seed values obtainable outside the program (don’t seed it based on user input).
    - Proper use of cryptography
        - Basically, be careful about your choice of keys, the distribution of those keys, key management, and application of cryptographic operations (make sure the cryptography actually occurs instead of there being an error that you don't end up dealing with, because in those cases, the cryptography doesn’t happen and messages get sent in plaintext).
    - Variable synchronization: Basically you have variables where changes to one should reflect changes to another (pointer to a buffer and the length of the buffer) so we need to make sure the right changes are reflected.
    - Variable initialization: Some languages let you declare variables without having to init them to some value. This allows values from one function to leak into values for those variables in other functions. Programs will often reuse a buffer and so old data in that area may be reused when it shouldn’t be. There are also use-after-free bugs. Can be used to execute code or see data that we shouldn't see.
    - Remote Code Execution Vulnerabilities: Programs allowing plug-ins, extensions, and dynamic code are vulnerable to execute code that has been provided from outside.
        - Avoiding the problem involves trying not to add extensibility, don’t make system calls that run arbitrary code (don’t have eval() calls), and have input validation.

2/28 - Week 8

- Security on the web important because a lot of financial and private information flows through web applications.
- In the web, users interact with servers and there is no central authority.
- A big problem is that we want to protect clients from each other (if they are both communicating with the same server)
- Also want to make sure that a client’s interaction with a server doesn’t leak to another server that the client is communicating with.
- We are trying to protect the client’s private data, the server’s private data, the integrity of their transactions, the client/servers’ actual machines, and the server availability.
- Browser security refers to protecting interactions from one site from those with another. The browser is pretty much an OS since it shares resources among different processes, and so there should be access restrictions and separation. However, the browser doesn’t have control of the hardware or anything like that. Browsers also have arbitrary extensibility and support multiple mutually untrusting pieces of code.
- The lock on browser says that you know that the web site is who it claims to be.
- Browsers have a feature where pages from a single origin can access each other’s stuff, but pages from a different origin can't (YouTube tab can't access Wells Fargo tab).

    - This is called the same origin policy.
    - Related to web cookies. Basically they are data that a website asks your browser to store, and then you send the state back again when you make some action.
    - Wells Fargo won't get the cookies associated with my interaction with YouTube.
- SQL Injection attacks are attacks on a database that is running SQL queries where the content is based on user input.
    - Basically, this is a problem with unvalidated input
    - Solution is to look at the web input and determine if it is a SQL fragment or not, but the problem is that some SQL control characters are widely used in real data.
    - You can use parameterized variables which means that you don't interpret the parameters as SQL. (Basically 'and 1=1' won't be actually interpreted as SQL code but rather as the actual name of the user).
- Drive by downloads are where just visiting a page immediately downloads a script.
- Some solutions are to
    - Disable (all or certain) scripts in browser but the problem is that it takes away too much of the good web functionality.
    - Use secure scripting languages, but they aren't popular and not foolproof and you can't force others to use these languages.
    - Use isolation mechanisms to limit the scope of the potential harm. Run all the scripts in an isolated VM, but scripts may need to do something legitimate outside the VM or those scripts could also just escape the VM.
    - Find bad scripts, get signatures, and then check the list and allow/disallow based on it. However, the script can be easily changed and the signature won't match. Also, identification and placing the script on the blacklist is a manual and tough one.
- Cross site scripting (XSS) is where users can upload information which get stored and displayed. If somebody uploads a script, then that could get stored, and other users could see it and download it. The scripts runs on that user's machines and gets run with the user's full privileges.
    - Non persistent XSS is where a user puts a small script in a URL web link that points to a legit webpage. When someone clicks the link, the link causes the script back to be echoed back to the user's browser and then it gets executed as a script on the user's machine, and never gets stored on the server of the web link that you're going to.
    - Persistent XSS is the case we explained 2 bullet points above and is where you upload to a website, gets stored in a DB, and then the DB displays to others.
- The biggest goals of these attacks is to steal personal info which is stored in the cookies at the client side. You can only get the information associated with that site.
- Solutions could involve not letting users upload, not letting users upload scripts, or protect the user's browser (allow users to specify where the content can be loaded from).
- Cross Site Request Forgery (CSRF) is where someone poses as a legitimate user and it is where they fool the server into doing something.

    - The way that it works is that an attacker puts a link to a bank on his page. When you click it, info about the attacker (his authentication info) goes with the request and where it came from (referral header) and bank will transfer money from your account to his.
- There are also data transport issues since the web is a network application. Web traffic can be unencrypted since it's expensive to encrypt/decrypt. This can allow sensitive data to pass in the clear.
    - HTTPS is a solution which is basically cryptography layered on top of HTTP. The crypto is based on TLS/SSL. HTTPS performs authentication and two way encryption of traffic.
    - Web servers can choose to run HSTS (a security policy mechanism) which requires browsers to use HTTPS.

3/5 - Week 9

- Determining whether a given system is secure.
- You can have secure system standards and check to see whether a system meets the standards at certain levels. The Common Criteria was to set the standards by which OS, database, firewall, etc. security can be evaluated.
- When someone needs a secure system, you specify the security you need by using the CC methodology, and then you look for the products that meet that profile.
- Problems with these certifications is that they are expensive to use, it is slow to get certification, the certification may not really mean that much, and the certification is honestly more for paperwork than actual software security.
- For evaluating a custom system, review is based on design and architecture.
- You want to do security reviews at the different stages in a product's lifecycle (during design, completion of coding, production time). Different security issues can arise at different stages.
    - Design reviews are more for discovering fundamental flaws at a high level. Threat modeling is used, basically it is looking for threats against what the system is trying to do. We can look into the software's attack surface which is comprised of the different entry points, the ways an attacker can interact with the software, etc. The surface doesn't indicate actual flaws but just places where they could occur. Smaller surface = better. After finding attack surface,
        - Do threat modeling by collecting info on the design (assets, access points, components, use scenarios, etc)
        - Application modeling (data flow diagrams, describe OO classes/interactions)
        - Threat IDing (find security threats to assets). Create attack trees to show the different attacks and the ways those attacks can happen (which are the branches that extend down).

                - Ex) The attack is where the attacker gets users' personal info, and the ways it can happen are gaining access to DB, hijacking user session, login as target user, etc.

            Then, look at each element and consider each STRIDE threat.
            - Documenting findings where you summarize the threads found, and prioritize which to solve first. You can prioritize DREAD methodology.
            - Prioritize implementation review which is a review of the actual code/implementation.
        - After design and implementation review, you then do an application review which is basically looking at a complete application. Hard to start if there's no design review. The process involves
            - Preassessment which is getting a high level view of the system
            - Application/design/live testing review
            - Documentation and analysis
            - Remediation support which is you helping them fix the problems.
- Design review can be
    - Top down: Start with higher level knowledge and gradually go deeper.
    - Bottom up: Look at the code details first and then build an overall understanding of what the code does.
- Code auditing strategies
    - Code comprehension (CC) strategies is looking at source code to find vulnerabilities.
        - One strategy is tracing malicious input and look at the paths of data/control. Follow the input data through the program. Does it lead to a problem?
        - Another is analyzing a particular module and then figure out if there are any issues with that module.
    - Candidate point (CP) strategies are where you create a list of potential issues and look for them in code. Basically look for known calls/functions that are questionable.
        - Simple lexical candidate points: Look for text patterns and specific function calls (strcpy())
    - Design generalization (DG) strategies are where you build a model of design to look for high and medium level flaws. Not really looking for small bugs, but looking at how the system actually works for an implementation POV.
- The guidelines for the above strategies are to perform flow analysis, re-read code, and check important algorithms, and use test cases (buffer overflows, weird input, etc) to make sure the algos work.
- Useful auditing tools involve source code navigators, debuggers, and fuzz testing (make sure program works with wide range of important values) tools.
- Analyzing running systems

    - Logging: Keeping track of important system information for later examination. OS and applications record messages about activities and events that may be important.
        - Access logs in particular are ones that list which users have accessed which objects, failed login attempts, changes in permissions, port scan (to figure out if you have web servers running) as well as the failures that may have occurred.
        - Big issue with logging is that you have to deal with a large amount of data and there will only be a few important lines. There is also a performance overhead in you writing to a file.
        - Another issue is that attackers will try to edit the log to remove traces of their behaviour. Solutions would be to make the log append only or log to a remote machine.
        - Another is that things that get logged must be considered for privacy.
            - Aka don't log the actual password the user entered on a failed password attempt.
    - Auditing: Process of checking whether the working system is doing what you're supposed to be doing. In order for someone to do an audit, they have to have security knowledge, need to know the security policy, independence, and trustworthiness (since they get privileged access to your system).
        - Should be done periodically and also after major system changes and when problems arise.
        - Audits should review control structures, logs, examine how aware users are of security, and physical security (who actually has access to it).

3/7 - Week 9

- Privacy is different from security in that privacy is the ability to keep certain information secret. It's not only info that you have (passwords, SSNs, etc) but also the information about what you're doing (GPS signals)
- Threats to your privacy include cleartext transmission of your data which can allow eavesdroppers to listen in, sites that save information on us, location privacy, etc.
- If companies have personal data on users, what can they do to protect that data?
    - Store encrypted data
        - Main issue is with the key and who has access to it.
    - Full logging and careful auditing
    - Limited access to the database
- All messages on the Internet are tagged with the sender's IP address and there is a mapping between an IP and a machine.
- Some privacy solutions
    - Data encryption: Store the private data in encrypted form so even if the attacker gets access they can't read it.

    - Anonymizers: Anonymous email/messages so that sites make requests on behalf of fake identities.
- Onion routing is used to handle the issue of people knowing who you're talking to. You want to conceal the sources and destinations by sending packets between a lot of places.
    - Those packets will get mixed up between all the routers, and now it's hard to tell which packets went into the routers and which came out.
    - A group of nodes have to be the onion routers and users get the public keys of that node. The users will then send the packets through those routers.
    - Sending the packet involves encrypting the packet with the destination key and wrap with another packet and encrypt that with the key to one of the onion routers. You can do the wrapping and encrypting a bunch of times for a bunch of routers.
        - Side effect is the packet gets bigger and bigger with the increase in the number of headers that are included.
    - The issues with onion routing are the proper use of keys and the overheads with multiple hops and encryptions for the packet to get from source to destination.
    - Tor is the most popular onion routing system.
        - Government doesn't like Tor because the Tor routers must know each other's identities and the traffic behavior is very noticable (if you look at a TOR router, it is taking a bunch of packets from weird places and then sending out to weird places).
- When dealing with aggregates of private user info, you can use:
    - Perturbation - Adding noise to the sensitive values
    - Blocking - Don't let aggregate query see the sensitive data.
    - Sampling - Randomly sample part of the data.
        - Every time you query, you get different responses and people don't know which stats correspond to what

3/12 - Week 10

- The 20 Critical Security Controls was a list of security things created by the US government. The Controls were general things to be careful about. It was based on:
    - Offense informs defense (don't defend against something the opponent isn't doing)
    - Prioritization
    - Metrics
    - Continuous diagnostics and mitigation
    - Automation

1. Take inventory of the devices on your system. Need to know what you have so that you can protect it. Any device can be a point of entry. The new, unauthorized, experimental and temporary devices are the problems. There are automated tools that will look for

new devices on the network. Active tools will query to see what's around and passive ones will analyze the network traffic.

2. Take inventory of the software on your machines. Most of the attack will come through the software and we need to know how to remove unnecessary programs. Look for authorized and unauthorized software on all the machines. Create a list of the approved software and what those machines need to have running.
3. Secure configurations for hardware and software. Some default configurations are not secure so you need to make sure they are up to date and secure. Solution is to create standard secure configurations that you want all your machines/software to have.
4. Continuous Vulnerability Assessment and Remediation: Have someone on the lookout for new vulnerabilities and security concerns. Check the news, figure out if your system is at risk, run vulnerability scanning tools.
5. Controlled Use of Admin Privileges: Limit the number of people who have that privilege (limits the attack surface) since admin privilege gives attackers a huge amount of control. Use automated tools to log who has admin privileges.
6. Monitoring of Security Logs: They are important sources of info on attacks. Make sure that the logs are writing the right info, leave enough disk space for logs, and actually take time to analyze those created logs.
7. Email and Web Browser Protection: Social engineering attacks like phishing email. Most attacks will come through email and browser. Set standard browsers and email clients. Limit use of scripting languages. Use spam filtering and scan all attachments.
8. Malware Defenses: Run malware detection on everything (and actually look at the outputs of those systems), limit the use of external devices, find malicious domains.
9. Limitations and Control of Ports, Protocols, Services: Take note of what software has been installed and which are really necessary, and drop the ports and protocols for everything else (let's say you don't support ICMP email, then you drop those packets). Also use firewalls with default deny rules. Also take note of the services that are visible from the outside.
10. Data Recovery Capability: Even if attackers can alter or steal your data, having a backup will give you an advantage. Make sure those backup disks are physically and cryptographically secure.
11. Secure Configurations for Network Devices: Make sure configurations of these devices are secure.
12. Boundary Defense: Firewalls are necessary but not enough. Have your boundary be sophisticated in that you have different zones that each have different permissions and access to different components.
13. Data Protection: Encrypt your data so that if attackers steal it, then the loss is minimized since they can't get in. Encrypt data always when you're sending it over the network.
14. Controlled Access Based on Need To Know: Make sure that only certain people can access critical data.
15. Wireless Access Control: Know what devices are connected in your network and make sure they run your secure configuration.

16. Account Monitoring and Control: Remove inactive accounts since attackers can use them.
17. Security Skills Assessment and Training
18. Application Software Security: Look at both commodity and in-house applications.
19. Incident Response Capability: Do something when you're attacked. Be prepared to respond.
20. Penetration Testing and Red Team Exercises: Try to get into the system from outside and inside your system boundaries so that you can see what's possible and what's not.

Reading Notes

Week 1 Readings

- Security products are useful for what they don't allow people to do, which is different from all the other types of computer applications.
- No amount of functional testing will ever be able to catch all the security flaws.
- Security evaluations between different versions of products is also something that can change.
- The main idea is that because of the difficulties associated with finding security flaws, the lack of security reviews, and the complexity of computer programs, security will be an ongoing issue.

- Social engineering is generally a hacker's clever manipulation of the natural human tendency to trust.
- Goals of social engineering are similar to the goals of creating malicious code attacks. It's to obtain information and access to systems for malicious intent.
- Social engineering can be physical or psychological.
    - The physical attacks are like walking into a room and finding passwords written on a pad, or calling someone and asking for password through some convoluted story
    - For psychological, the main objective is to convince the person disclosing the information that the social engineer is in fact a person that they can trust with that sensitive information.

Lab 1 - Permissions and Firewalls Notes

- Purpose of the lab is to teach principle of least privilege through POSIX permissions and iptables firewalls.
- POSIX is a standards family and is used to formalize the API for variants of Unix OS's.
    - Basically it defines the APIs for functions like permissions, threads, networking, etc.
- Linux is substantially compliant with POSIX but not officially able to have the distinction.

- One of the main parts about the POSIX standards is the traditional Unix file system permissions, which are evaluated to give access to system resources.
    - Each file has read, write and execute permissions for 3 groups of users (user/owner, group, and other)
        - Users have their own unique IDs, groups are collections of users and they have group IDs as well, and other is everyone that is not the user (owner) of the file and or a member of the file's group class.
        - All system users go into these 3 non-overlapping sets.
    - A bitstring is split up into the type of the entity (file, directory, etc), then the user permissions, then group, and then other. There's also more content with the setuid, setgid, and sticky bits.
    - Permissions set on a directory do not always apply to the contents of the directory.
- Other systems have more than 3 access groups, ability to change permissions easier, more features, etc that traditional UNIX permissions don't have.
- Sudo is an application that enhances the expressiveness of the permissions without actually changing the system.
- Lots of commands you can use to add users to a system, add groups, change ownership/mode of a file, etc.
- First firewalls were used to look at the headers of individual packets (basically checking things like the source/dest addresses, port numbers, etc) and then the firewall would make decisions on which packets to deny and which to let through. These firewalls are stateless because they kinda work independently regardless of the things that came before or other factors about the system. They just look at individual packets.
- Stateful firewalls keep tables of network connections in memory and use that additional info to help determine whether a packet should be let through or not.
    - You can configure these to only allow packets to the port if a legitimate TCP connection has been established.
- Application layer firewall is also an extension and it is able to not only look at the header of the packet, but also the payload in the context of the application the packet is being sent to.
- Firewalls deal not only with keeping bad stuff out, but also with keeping information in.
    - Default allow policy is associated with the goal of keeping things out, and basically says that we should allow anything not considered a threat.
    - Default deny is one where you only allow things that are strictly required.
- Iptables, nmap, ifconfig, telnet, netcat are all helpful firewall related commands.

Assignment Info

- Want to create an answer key for a hiring exam that asks questions about permissions.

Lab 2 - SQL Injection

- Buffer overflows is when a buffer (bounded region of memory) is assigned more data than it can hold. The buffer contents overflow into the next available memory space and thus overwrites the data contained there.
    - In the strictest sense, a "buffer overflow" is when a buffer of size b is assigned data of size c where c > b.
- To fight buffer overflows, we need to do input validation. Basically, it's making sure that all accepted input fits within the logical constraints of the application.
- SQL is a database querying language. It allows programmers to ignore the particulars of the database internals. Instead, they can use a (mostly) standardized query language to access the data they need, remaining largely independent of the database application, database memory management, and physical storage.
- A common class of attacks are called "SQL injection attacks" which -- like directory traversal and buffer overflow vulnerabilities -- are the result of trusting non-validated input and implicitly granting applications privilege they do not require.

Lab 3 - Computer Forensics

- In this lab, we are going to analyze disk images of compromised computers in order to figure out what is wrong with them.
- Forensics is mainly about answering questions in a court of law, and gaining evidence from a partially unknown sequence of events. Computer Forensics is the application of that science to answer questions in the field of computer systems.
    - In CF, we have to be able to have the answers for computer science equivalents of “Who was here”, “What did they leave behind”, etc.
- In order to positively establish that a suspect was involved or wasn’t involved in something, we can try to use information about usernames (person who controls the user account is generally responsible), network addresses of the machines from where packets are being sent, CPU Serial Numbers (unique numbers for each processor), etc
- The tools used to be able to figure out this type of information are:
    - Disk imaging: Obtains a sector by sector copy of the entire hard disk
    - Editors/Viewers: Used to open files and search for things within them.
    - System Logs and Process Accounting: Shows aspects of systems’ operation, like kernel operations, system operations, etc. There are also files that contain the last series of commands the user has entered.
    - Network Scanning and Monitoring: Keep track of the traffic it creates and receives. You can map open ports, monitor connections, and filter/view packets.
- If you’re looking for data that isn’t shown in plain sight, you’ll need to look at deleted files, temporary files, hidden files, caches, spools, and RAM.
- Tools for ensuring data integrity involve chain of custody (all evidence accesses and transferral of custody be logged and verified), cryptographic hashes to make sure that data has not been changed.
    - More info on hashes: Hashes are used to create manageable digests of collected data immediately upon collection. Ideally, the investigator would create two disk

images from the system being investigated and verify that the hash digest of both images match.

- The initial state of the system should be documented when it comes into the investigator's control.
- Need to also pay attention to whether the attacker will have traps that will damage evidence (erasing the hard disk on a proper shutdown)