CS 161: Computer Security Notes

Enzo Massyle

September 4, 2024

1 Introduction To Cryptography

Cryptography is the study of how to send data in the presence of an attacker. It is essential to use cryptography in todays world to ensure the all user data and private information is secure. First, we will go over some definitions.

1.1 Alice, Bob, Eve, Mallory

Each person had the following role:

- Alice and Bob wish to communicate securely
- Eve is a eavesdropper who listens to Alice and bobs conversation but does not tamper with the data. We call this person an honest-but-curious hacker.
- Mallory is a malicious attacker who wishes to tamper with the data that Alice and Bob are sending.

The goal when sending data across a network is that Eve is not able to listen in on the data (**Confidentiality**) and Mallory can never tamper with the data.

1.2 The three main goals of cryptography

- Confidentially: No one is able to listen in on communication
- Integrity: If the data is tampered with, we should be able to detect it
- Authenticity: We should be able to correctly identify and prove who sent the data

1.3 Keys

Keys are one of the main components of cryptography. It is what allows us to recover the original message after it was sent through a cryptography algorithm. There are two main types of key methods:

- **Symmetric key model**: The sender and receiver of the message both have the same secret key.
- **Asymmetric key model**: There are two different keys. The secret key which only the user of the data has access to, and the public key, which anyone who wishes to communicate with the user has access to.

When we design cryptography algorithms, we should make the assumption that the attacker knows what algorithm we are using and even how it works. Even if it is something that we came up with ourselves and is completely original.

1.4 Symmetric Key Encryption

The first type of encryption algorithms we will talk about is symmetric key algorithms. That is, both the receiver and the sender have access to the same private key. Here is the general structure of how symmetric key-encryption works.

$$\text{Let } K = \text{Secret Key}, M = \text{Message}, C = \text{Cipher Text}$$

$$\text{Generate Key: } KeyGen() = K$$

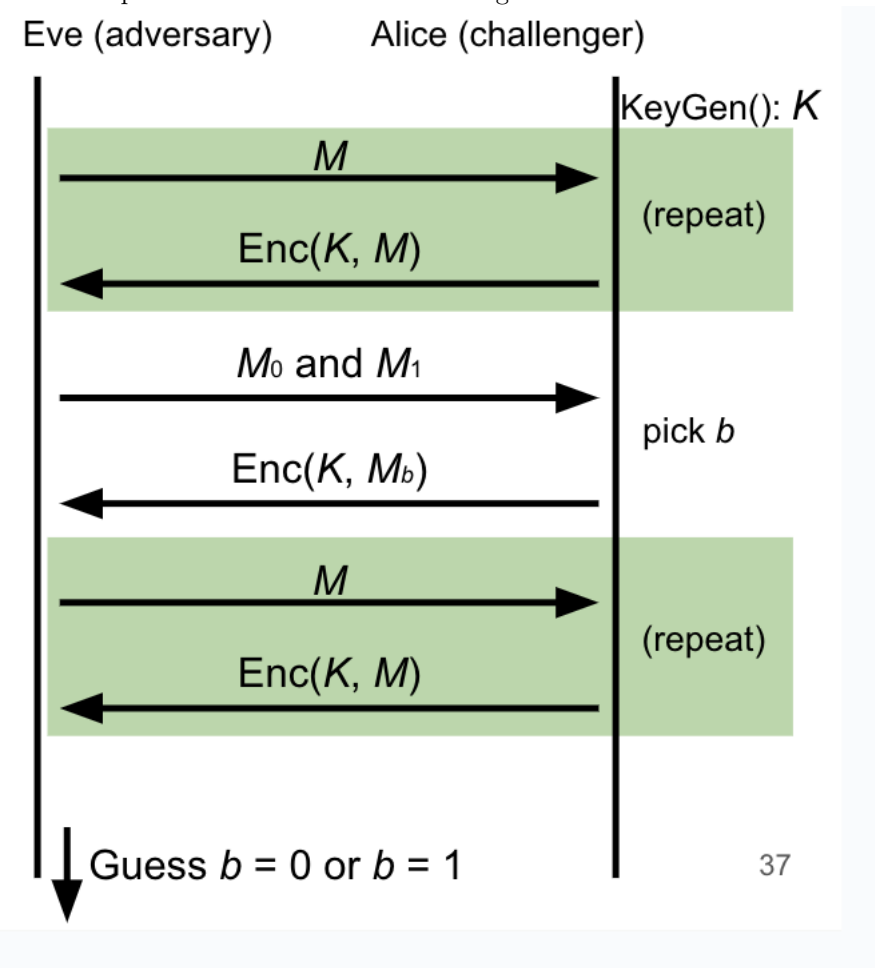
$$\text{Encrypt Message: } Encrypt(K, M) = C$$

$$\text{Decipher Message: } Decrypt(K, C) = M$$

One extra note for correctness: Decryption is the inverse of encryption. That is $Decrypt(K, Encrypt(K, M)) = M$

1.5 Defining Confidentiality

Since confidentialy is so important when sending data, we need a proven way to show that a given algorithm is confidental. This is where the IND-CPA comes in. IND-CPA stands for indistinguishability under chosen plaintext attack. Here is how it goes:



- Eve will send a plain-text message to Alice, the Cipher text will be sent back
- Two messages M_0, M_1 will be sent to Alice, Alice will then choose which message to encrypt and send the message back. It is important to note that Eve does not know which message Alice decided to encrypt.
- Eve can then repeat step 1 as many times as they want.

- Eventually from step 2, Eve will guess what message was the cipher text generated from. Either M_0 or M_1

From this method. If the encryption algorithm is good, Eve will only be able to correctly guess what message Alice sent back with $\approx \frac{1}{2} + x$ accuracy. Here x stands for some negligible extra chance that eve has to guess it from information that is leaked from the encryption algorithm. Usually something that is leaked in encryption algorithms is the length of the message. Since the amount of wasted space that would be needed to hide the length of all messages that are sent would simply be too expensive. So for the purpose of the IND-CPA game, both messages must be of the same length.

1.6 Brief History of Cryptography

Since this section will not be tested on, I will keep it brief. We will introduce a few of the first ever known encryption algorithms to send messages. These were before computers existed so these are pen and paper algorithms. As you can probably guess, they were shit.

1.6.1 Caesar Cipher

In the Caesar Cipher, There is a shift of all character by some offset. So in this case the key K is the offset. To encrypt the message every character gets converted to the character that is K letters later in the alphabet. To decrypt this message, we simply would go K letters back in the alphabet to get the original letter. This obviously fails the IND-CPA test. Eve can send a message and get back the cipher text and immediately know what the key is.

1.6.2 Substitution Cipher

In this encryption algorithm, each character has a unique mapping to another character in the alphabet. This again is very easy to break and breaks the IND-CPA test. All the Eve needs to do it send a test message including the whole alphabet, The cipher text returned will show you the mapping for each character. This is the key K. At this point eve has the key and can decipher any message that comes their way.

2 One Time Pads and Block Ciphers

So far we have covered some of the first encryption algorithms and we saw the immediate flaws in them and how they do fot the IND-CPA standard. Now we will go over some symmetric key encryption algorithms.

2.1 One Time Pads

2.1.1 A Review on XOR

XOR is a bitwise operation that takes two bits and returns true iff one of them are true. Common saying is one or the other but not both.

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

Table 1: XOR Truth Table

Here are alos some useful properties about XOR:

- $X \oplus X = 0$
- $X \oplus 0 = X$
- $X \oplus Y = Y \oplus X$

2.1.2 One time Pads

The first real encryption algorithm we will use is a one time pad. In a one time pad, we will generate some bit-string and perform an XOR on our plain text message (also as a bitstring). This will make our cipher text pretty much look like giberish with no real way for the attacker to find out what our message is. Here is how the process goes

$$\text{Generate key (bitstring to XOR): } KeyGen() = K$$

$$\text{Generate cipher text: } Encrypt(K, M) = K \oplus M = C$$

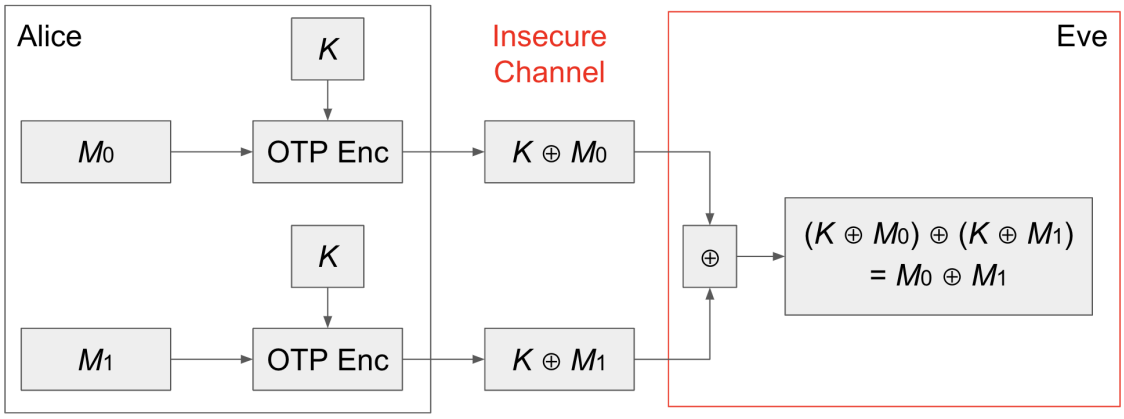
$$\text{Decript cipher text: } Decrypt(K, C) = K \oplus C$$

We see the Decryption works because $K \oplus C = K \oplus K \oplus M = 0 \oplus M = M$

One time pads work great because. If we remember the IND-CPA test, the attacker can send two messages and get back a cipher text. But remember that the cipher text sent back to the attacker could be from either M_0 or M_1, the attacker doesn't know. So there is no way of finding out the key and recovering the message.

2.1.3 Two Time Pads?

The one time pad worked so well, could we use the same key again to encrypt another message? Well, that would not work so well due to the following.



In this picture, the same key reused to encrypt another message, hence a two time pad. The problem with this is due to the property of XOR. that is $(K \oplus M_0) \oplus (K \oplus M_1) = K \oplus K \oplus M_0 \oplus M_1 = M_0 \oplus M_1$. Form $M_0 \oplus M_1$ The attacker can find out information such that which buts of M_0 and M_1 are the same which is revealing partial information about the messages sent.

2.1.4 The Impracticalities of One Time Pads

One time pads are not very practical to use in the real world for a couple of reasons.

- **Key generation**: For every message we send, we have to generate a new key. This takes time since the keys must be randomly generated for every message.
- **Key Distribution**: Even when the key is generated, we need to find some way to distribute the key securely, since if the attacker has the key, they can simply decrypt the message. But if there was already a way to securly send the key over a channel, we would just send our message over this channel

It is due to these reasons that One time pads are not very useful in the real word.

2.2 Block Ciphers

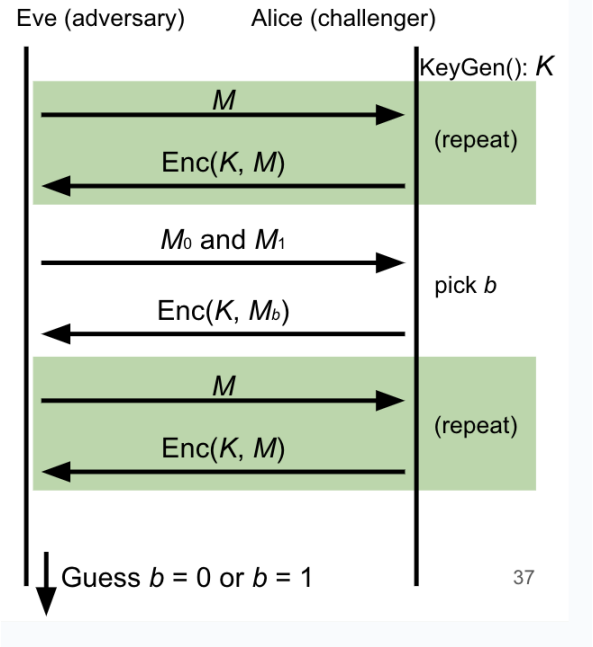
Block Ciphers are the next cryptographic scheme we will be discussing. Block Ciphers behave like the following. A block cipher takes fixed sized blokc of bits and encodes them to a mapping of different bits. It goes like the following:

$$\text{Generate Mappings (Key): } KeyGen() = K$$

$$\text{Generate cipher text through mappings: } Encrypt(K, M) = C$$

$$\text{Recover Message through the inverse mappings: } Decrypt(K, C) = M$$

There are two rules for the mappings. The first rule is that the mapping must be random. This makes sense since the if the mapping was predictable, then the attacker could recover information about the message send. Second is the mapping must be bijective. If a mapping is not bijective, this means the inverse mapping does not exist, which means we would not be able to decrypt our cipher text!. So if this IND-CPA secure? We lets look at the following



Recall our IND-CPA picture, so how would an attacker break the block cipher. We it is as simple as the following:

- Attacker sends 2 messages M_0 and M_1, gets back either C_0 or C_1
- Attacker then sends M_0 again, if the encryption matches what they got when the 2 intial messages where sent, they know the original message was M_0 if the message is not the same, they know the message is M_1
- Thus, the attacker can correctly guess the message sent with probability 1.

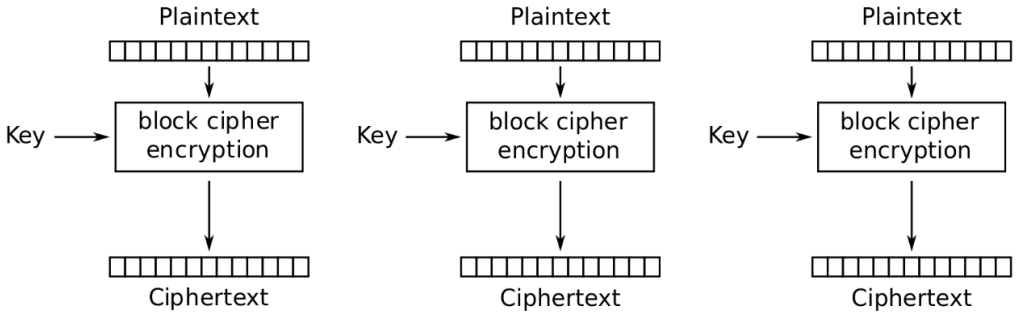
You see, the issue with regular block ciphers are that they are deterministic. That is, they do not change. So next we will introduce things that make this encryption scheme non-deterministic. Another issue, is that we can only encrypt messages of a fixed size. If the message is bigger than we are able to encrypt, we simply just cannot do it.

2.2.1 Block Ciphers With Modes

We will explore the modes Electronic Code Book (EBC) mode, Cipher Block Chaining (CBC) mode, and CTR mode.

2.2.2 EBC Mode

Like we mentioned, if the size of our plain text is more than n, then we run into an issue where we cannot encrypt. In this scenario, we can simply break up our message into blocks of length n, having these blocks chained together. It would look something like this.

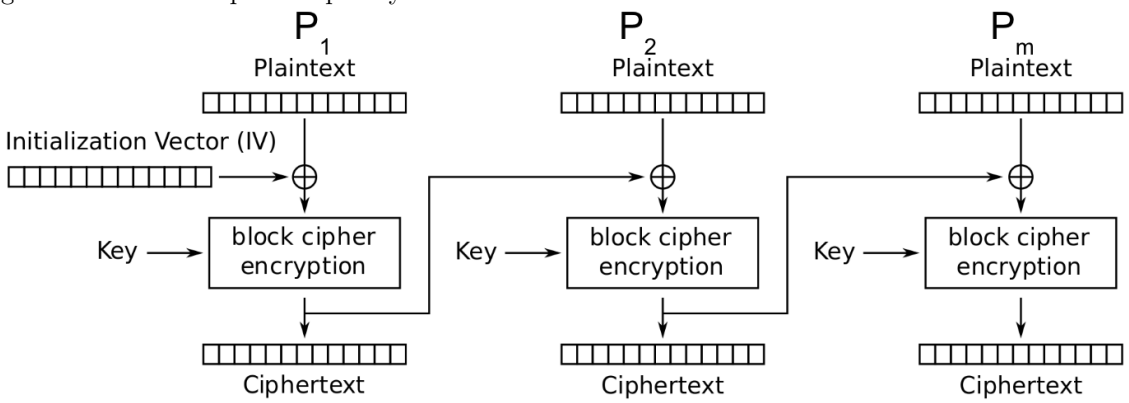


As we see in this cryptography scheme, the plaintext message gets broken up into sections and then ran through the block cipher encryption.

While this seems to do the trick, there arises an issue with this scheme, it is that plaintexts that are exactly the same will produce the same exact cipher text, this leaks information about the message, since we do not want the attacker to know what parts of the message are the same.

2.2.3 CBC Mode

Here we will solve the issue that arose when doing EBC encryption. We will visit the concept of XOR again to make our input completely non-deterministic.



Cipher Block Chaining (CBC) mode encryption

Lets explain what is going on here. We do a couple things extra with CBC compared to EBC.

- Choose an initialization vector (IV) to XOR our first block of plain text.

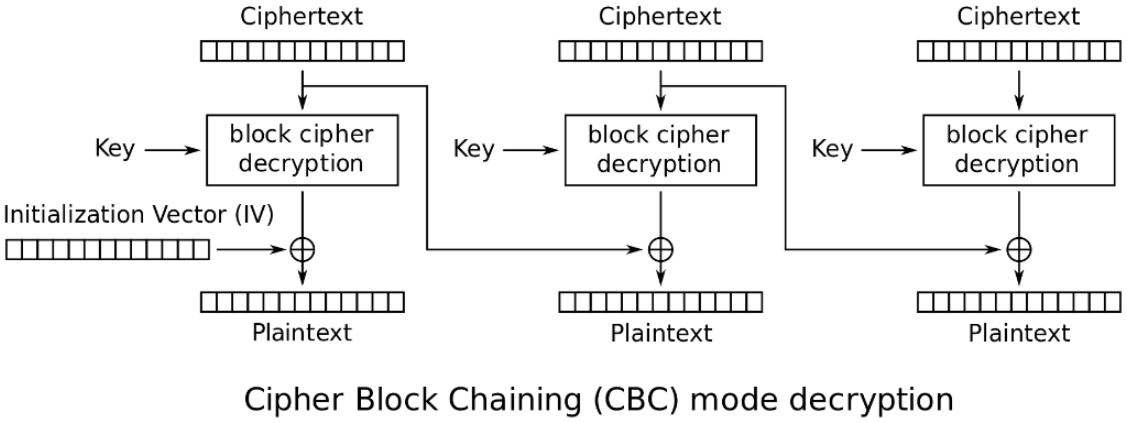
$$C_1 = E(K, P_1 \oplus IV)$$

- For the rest of the plain text blocks we will use the previous cipher text to XOR it with the current plain text to introduce more randomness.

$$C_i = E(K, P_i \oplus C_{i-1})$$

One thing to note in this scheme, is that we must do the encryption in series, since the output of the current cipher text depends on the previous cipher text, so we must compute the previous one first.

Okay, so now that we know how to encrypt using CBC, how do we decrypt? Take a look at the following diagram.



Cipher Block Chaining (CBC) mode decryption

]

From this picture, we can see how to decrypt, we do the following:

- Take each cipher text and run it through the block cipher decryption, remember that since our mapping was bijective, there exists a way to get back to our original input.
- XOR the output from the block cipher decription with the previous cipher text, except for the very first plain text, which we will XOR with the IV.

$$\text{Definition of Encryption: } C_i = E_K(P_i \oplus C_{i-1})$$

$$D_K(C_i) = D_K(E_K(P_i \oplus C_{i-1}))$$

$$\text{Remember our Decryption is the inverse of encryption: } D_K(C_i) = P_i \oplus C_{i-1}$$

$$\text{XOR with previous block: } D_K(C_i) \oplus C_{i-1} = P_i \oplus C_{i-1} \oplus C_{i-1}$$

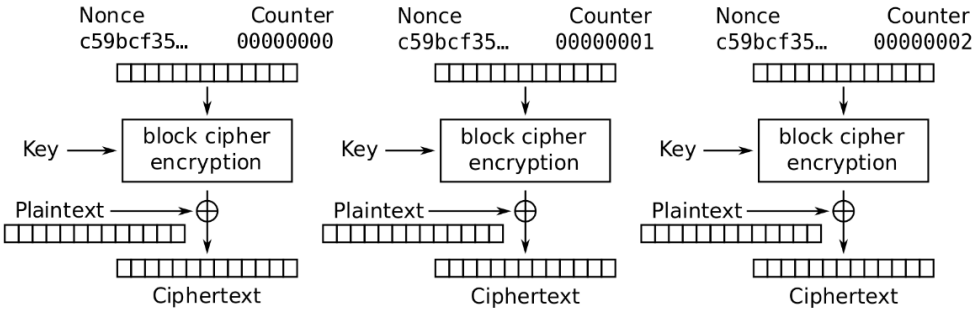
$$\text{Result: } D_K(C_i) \oplus C_{i-1} = P_i$$

Unlike the encryption, the decryption for CBC is completely parallelizable. This is because we have already computed the cipher text so we have all the information needed to calculate each block of the plain text at the same time.

With this encryption scheme we need to make sure that the initialization vector is completely randomized. If this is not randomized, then it is possible for the attacker to leak information about the message.

2.2.4 CTR Mode:

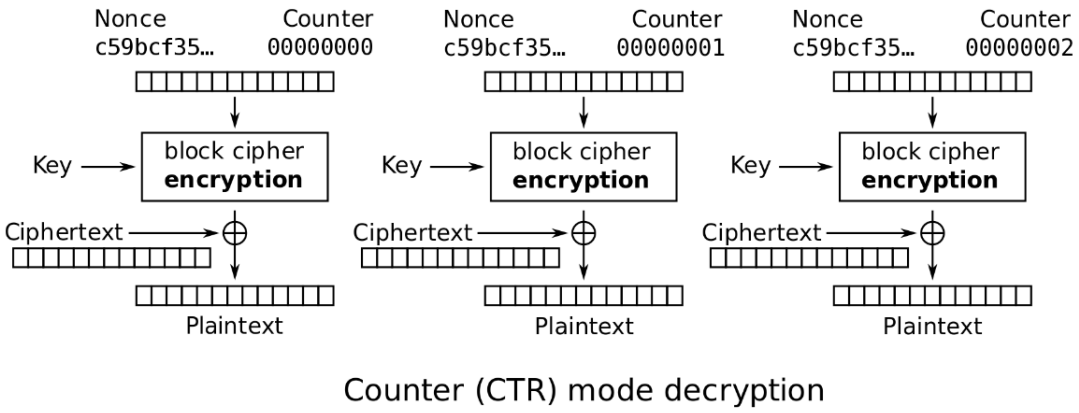
Currently, the only issue with CBC mode is that the encryption is not parallelizable. We want to speed this up. So this is where CTR mode comes in, another encryption scheme that is IND-CPA.



We see the idea of CTR mode is similar to CBC mode. However it is a little different. Here is how it goes:

- Generate a number only used once (a nonce)
- Formulate a block with the nonce and the counter. The counter starts at 0 and for each block the counter increments. Even though we only change the next block by 1 bit, we know that still completely changes the mapping. We then XOR this output from the encryption block with the plain text to generate the cipher text.

This method is completely parallelizable, because for each block of encryption, all we need is the plain text and the nonce/counter block which are already known and do not need anything before
The decryption is also parallelizable, it goes as the following:



Counter (CTR) mode decryption

We see the decryption is very similar to the encryption, the only thing is that the cipher text and the plain text is switched.

2.3 Lack of Integrity and Authenticity

So far, when using these encryption schemes, The thing we have been lacking is integrity and authenticity. Remember the definitions of integrity: if our data is tampered with, we should be able to detect it.And authenticity: we should be able to confirm the user that sent a given message. This is where we are going to make use of hash functions and MACS.

2.3.1 Hashing

The definition of hashing goes as follows. A hash function is a function that takes in an input of arbitrary length produces an output of fixed length n. Some properties of a hash function are the following:

- Deterministic: If i give an identical input to a hash function H n times, I should get the same output n times.
- Collision resistant. If $x \neq y \rightarrow H(x) = H(y)$, then a collision happened. Our hash function should not do this.
- Unpredictable. We should not be able to make an educated guess of what a hash function would output. So even if we change one bit in the input, we should get a completely different output.
- One-way: we should not be able to invert the hash
- They should be efficient to compute.

Hashing is a step in the right direction in providing integrity to sending messages across channels. However, there are still some issues with this. First off, Collisions are bound to happen. Think about it, if we had a message that was $k > n$ bits, meaning there are 2^k different possible bit-strings that make unique messages, buit when we put our message into a hash function, there are anly 2^n different bit-strings. Since the number of bits is smaller, there are therefore less combinations which means collisions are bound to happen.

Another scenario that could happen is Alice is trying to send a message to Bob over an insecure channel, Alices sends the message with a cryptographich hash. But Mallory intercepts the message and tampers with it. They themselves can change the message and the cryptographic hash since no key is required for computing the hash. Thus when Bob gets the message, he will check the hash and see that is matches the one sent from Alice. The problem is that has was not Alice's hash, it was Mallory's.

Thus we see that hashing alone does not provide much authenticity, this is where we will use message authentication codes (MACs)

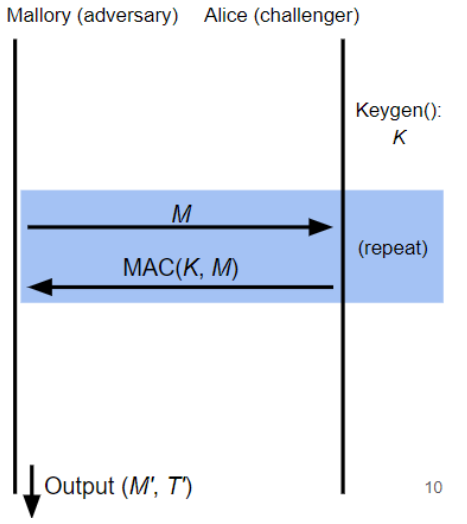
2.3.2 MACs

Remember, the whole goal with integrity is that if someone tampered with a message. We should be able to recognize that a given message was tampered with so we can take according action, maybe trashing the data could be an example. In message authentication codes, with sending the message, we will send a tag along with the message, we will compute the tag with $T = MAC(K, M)$ and send this along with the message. Once the receiver on the other side gets the message, they will recompute the tag with $T = M(K, M)$ and if the tags match, we can ensure that the message sent came from a safe user. To format this a littler nicer, here is how sending messages with tags work:

- Generate key: $K = \text{KeyGen}()$
- Generate tag: $t = \text{MAC}(K, M)$ Takes in a key K and a message M of arbitrary length, and outputs a tag of fixed length.

Why this is a great way to send a message over an insecure channel, is that we cannot compute that tag of a message without the key. That is, if a malicious attacker tries to alter our message, the tag cannot be changed, since the attacker does not have the key. So that when the receiver gets the message and computes they tag, if it doesn't match the original tag in the message sent, we know the message must have been tampered with. Without the key, there is no way for an attacker to generate the tag associated with a message M. This is a formally defined security game called EU-CPA which says that the probability of forming a new message M and its associated tag T is negligible. Here is a picture of the EU-CPA security game

1. Mallory may send messages to Alice and receive their tags
2. Eventually, Mallory creates a message-tag pair (*M′*, *T′*)
   - *M′* cannot be a message that Mallory requested earlier
   - If *T′* is a valid tag for *M′*, then Mallory wins. Otherwise, she loses.
3. A scheme is EU-CPA secure if for *all* polynomial time adversaries, the probability of winning is 0 or negligible



We see in this image of the game, Mallory, the attacker can send as many messages as they please to get the tag of the related message. The point of this game is no matter how many times Mallory does

this, the probability that they can forge a tag for a message they have not given to alice is negligible. Note it has to be a new message, since the MAC is deterministic, once we konw the tag T for a message M we know it forever.

There are a few different MAC schemes that we will go over. Mainly the two NMAC and HMAC.

2.3.3 NMAC

One though to make the MAC is to append the MAC to the message and then hash the whole thing. We would do it like the following:

- Generate two n-bit keys using KeyGen()
- Compute the NMAC: $H(K_1||H(K_2||M))$. Doing it this way prevents what is called a length extension attack.

While this does provide a secure way to send a message. THe problem with NMAC is we need to generate two keys for every message sent. This is the idea of the HMAC and how we need only 1 key to accomplish the same task.

2.3.4 HMAC

For HMAC, the formulation will be very similar.

$$H((K^{'} \oplus opad)||H(K^{'} \oplus ipad)||M)$$

The the keys will be different due to ipad and opad. It is not too important what the ipad and opad are, as long as they differ by at least 1 bit, we will get completely different hashes.

2.4 Summary of MACs

To wrap up MACs, they are a great way to provide integrity to sending messages over insecure channels. Since MACs are unforgeable, if an attacker tries to tamper with the message, the MAC will change and the attacker cannot generate the MAC that is associated with the new message that they created. Also in some cases, MACs do provide authenticity, but not always. It does it in the sense of If i get a message that has a valid MAC, I know it came from someone with the correct MAC. That does not mean I can identify who it came from. If the secret key is only shared between two people, then MACs do provide authenticity. Lastly, MACs are not confidential at all, as we discussed MACs are deterministic so they fail the IND-CPA test.

2.5 Authenticated Encryption

Now that we have encryption schemes to make the message confidential (i.e. no leaking anything about the message). And ways to provide integrity and authenticity through MACS, we now want to find a way to have authenticated encryption. A scheme that has all three properties we want in cryptography: Confidentiality, integrity and authenticity.

2.5.1 Combining Schemes

The first idea that probably comes to mind is when sending a message, we will send the message along with the MAC. For example something like this.

$$E(K_1, M), MAC(K_2, M)$$

While this seems like a good idea, this still does not provide confidentiality since the MAC is deterministic and not IND-CPA secure. So this will not work.

Another idea that comes to mind is to encrypt the message like normal, but compute the MAC on the cipher text instead of the original plain text message.

$$E(K_1, M), MAC(K_2, C)$$

The message is fully encrypted, and the MAC will leak information about the cipher text but that is okay so this is a confidential encryption scheme.

Another way we would also do it is to encrypt the mas as well. Something like this:

$$E(K_1, M||MAC(K_2, M))$$

In this scheme, we also get full security, everything is encrypted.

2.6 MAC-then-Encrypt vs Encrypt-then-MAC

There are two main ways that we can go about combining out encryption and MACs.

The first one is encypt-then-MAC. in this method, we will first encrypt our plain text message, then compute the MAC on the cipher text. and then send this across our network. the second one is MAC-then-encrypt. In this process, we will be computing the MAC on the plaintext message, and then encrypting the message, then sending both across the network.

So which one is better? well, we always prefer to do encrypt-then-MAC since in the MAC-then-encrypt scheme, we cannot detect tampering until after we decrypt. In the encrypt-then-MAC scheme, since we did computed the MAC on the cipher text, we do not need to decrypt to find out if the message was tampered with.

3 PRNGs

So far what we have been talking about in encryption schemes, is there is always a factor of randomness. The problem is, generating true randomness is quite expensive. There is a little circuit inside of you computer that is unpredicatble and will provide true randomness.
Entropy: The measure how truly unpredictable outcomes are.
So if something has high entropy, it is very unpredictable. This is how the true randomness inside our computers are generated, through circuits with high entropy. The problem is that these are expensive and quite slow. This is where the Pseudo Random Number Generators comes in: PRNG. We use a little bit of randomness from our computers as input to the PRNG and the PRNG has a set algorithm that will produce outputs that look random but are actually just a set of numbers that look random. This means that PRNGs are deterministic!

Every PRNG has two main functions:

- **Seed**: initlialize the state of the PRNG using true entropy from the machine
- **Generate**: outputs the next pseudo random number in a stream.

There are three main properties of PRNGs:

- Deterministic
- Efficient
- Provide great security. Cannot distinguish from random.

4 Diffie-Hellman Key Exchange

So far we have been talking about symmetric encryption schemes. The problem with these schemes, is we need to make sure the sender and the receiver both have the same key. But how can we do this if we are communicating over an insecure channel? This Diffie-Hellman key exchange is an idea of how we can do this.

The concept of one way functions are like the following take a function $f(x) = y$ if we are given

x it is pretty easy to compute $f(x)$, but if we were given y it is really hard to find the x that got us there. Think of like a cow and a hamburger. It is relatively easy to turn a cow into a hamburger (kinda cruel and sad) but it is impossible to turn a hamburger into a cow. This is the idea behind the Diffie-Hellman Key exchange. Here is how it will go when Alice and bob want to share something.

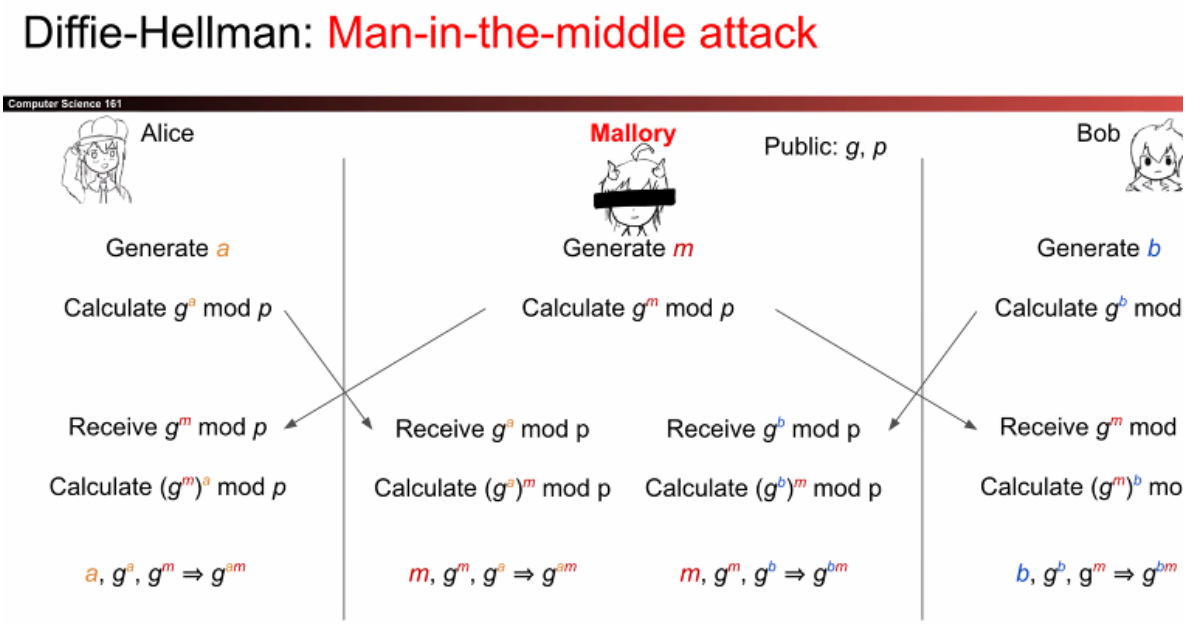
- Two constants p and g are picked. where $1 < g < p - 1$
- Alice picks secret value a and Bob picks a secret value b
- Values $A = g^a \mod p$ and $B = g^b \mod p$ are then announced publicly.
- Alice using the knowledge of the Bob's public value B can calculate $B^a = (g^b)^a = g^{ab}$
- Bob using the knowledge of Alice's public value A can calculate $A^b = (g^a)^b = g^{ab}$
- We see Alice and Bob get the same message at the end. And this g^{ab} is indistinguishable from a random number.

Thus we can use this scheme to share a key across an insecure network, then proceed to use a symmetric encryption scheme that we have gone over in class. These keys that are generated can only be used ephemerally (for a short period) this is because the longer we use the key the longer the time window exposure an attacker has to that key. Another one is convenience. We do not wanna carry this key around going from website to website, just generate a new one.

Another benefit of Diffie-Hellman is Forward secrecy. This means aftera, b and K are discarded, none of it is saved. So if Eve was able to get Alice and Bobs secrets, that is $g^a \mod p$ and $g^b \mod p$. Even still can't decrypt these messages because they still don't know what a, b, K are and they have been discarded.

4.1 Man in the middle attack

The man in the middle attack is a method that is used by attackers to steal the private key for symmetric encryption. This goes as the following.

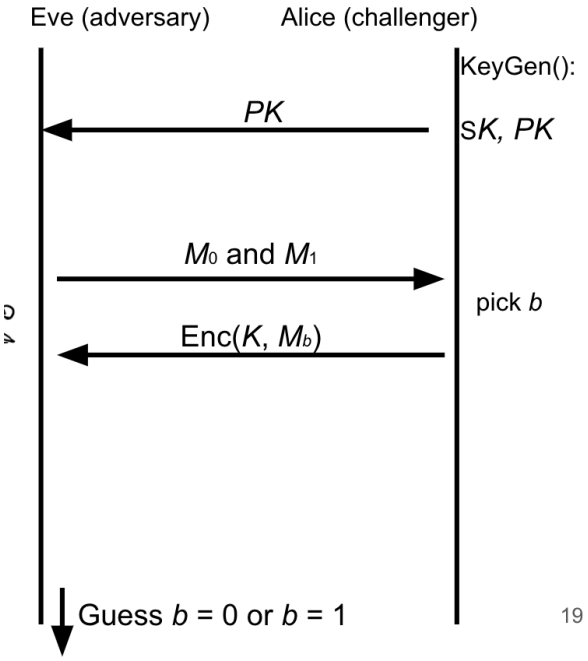


As we can see in this picture, essentialy mallory is impersontaing as bob to alice and as alice to bob. In this they agree on g and p and then send messages to eachother. By doing this impersonation, Mallors is able to gain the agreed symmetric key to send messages, this means Mallory can send and receive whatever they please.

5 Public Key Encryption

The main idea behind public key encryption is that everyone can send messages to some recipient, but only the recipient can decrypt these messages. There are three parts to public key encryption:

- Generate a public and private key pair
- when sending a message to the recipient, encrypt it using the public key to produce the cipher text
- Only decrypt cipher texts using the private key.



You might notice that this is even simpler than the IND-CPA symmetric key encryption. This is because of the public key. Since the attacker has access to the public key, they can encrypt as many messages as they want and see the cipher text, this removes the initial part we had for the symmetric-key encryption. Again, the attacker should be able to correctly guess whether M_0 or M_1 was encrypted with the probability of $\frac{1}{2} + \epsilon$ where ϵ is negligible.

5.1 EIGama Encryption

EIGama encryption is a public key encryption scheme that goes like the following:

- First off, we will establish system parameters p and g where p is a large prime number and $1 < g < p - 1$.
- Say Bob wants to be able to receive messages over an insecure network. Bob will choose a random value b. Note this is private (only known to Bob)
- Bob will then compute the public key, which will be $B = g^b \mod p$
- When Alice wants to send a message to Bob, they will take this public key to encrypt the message. We do this by computing the cipher text as a pair of numbers. For the first number Alice will choose a random value r to compute $R = g^r \mod p$. Then they will use the public key and the message they want to send to compute $S = M * B^r \mod p$. We will send the cipher text as (R, S)

- Now the next step is for Bob to decrypt Alice's message, we will do this by computing $R^{-b} * S \mod p$

We can work out the math to see that this works:

$R^{-b} * S \mod p = (g^r)^{-b} \mod p * M * B^r \mod p = (g^r)^{-b} \mod p * M * (g^b)^r \mod p = g^{-rb} * g^{rb} * M \mod p = M \mod p$

We see that we get $M \mod p$. Note that this means that we can only encrypt messages up to length p, or else we will lose critical information about the message.

With this being our first attempt at public key encryption, there are some issues with it. First that if we send a message that is 0, we see that C_2 will be 0, meaning that we will automatically know that the message is 0. Which is not confidential since we leak information about the message. To combat this, some libraries that use this encryption will include padding to every message to ensure the message will never be 0.

5.2 RSA Encryption

Here is the process:

- Again, we will pick some system parameters, this time is will be two large primes p and q
- Compute $N = pq$
- Choose e some properties about e is that is it coprime to $(p-1)(q-1)$ and $2 < e < (p-1)(q-1)$
- Note the public key is N and e
- Compute the private key $d = e^{-1} \mod (p-1)(q-1)$
- To encrypt a message M we will use the public keys e and N: $\text{Enc}(e, N, M) = M^e \mod N$
- To decrypt a message C we will use the prime key d: $\text{Dec}(d, C) = C^d = (M^e)^d \mod N$
- Through some CS 70 knowledge, mainly Euler's theorem and Fermat's little theorem, this simplifies to $M \mod N$

The current problem with this RSA scheme is that is it deterministic, meaning it is not IND-CPA secure. To combat this, there are schemes that use RSA and padding to strengthen its security.

5.3 Digital Signatures

Like MACs for symmetric encryption, Digital signatures are the way to provide integrity and authenticity to asymmetric encryption schemes (like RSA). Essentialy when the person with the private key wants to send a message to someone, They will send a signature along with the message. the user when getting the message will verify the signature. Here is the general scheme on how it works.

- Generate Key: $\text{KeyGen}() = PK, SK$ PK is the public key and SK is the private key
- Sign the message $\text{sign}(SK, M) = sig$ generate the signature with the private key SK and the message
- The receiver will verify the message sent: $\text{Verify}(PK, M, sig) = TrueorFalse$

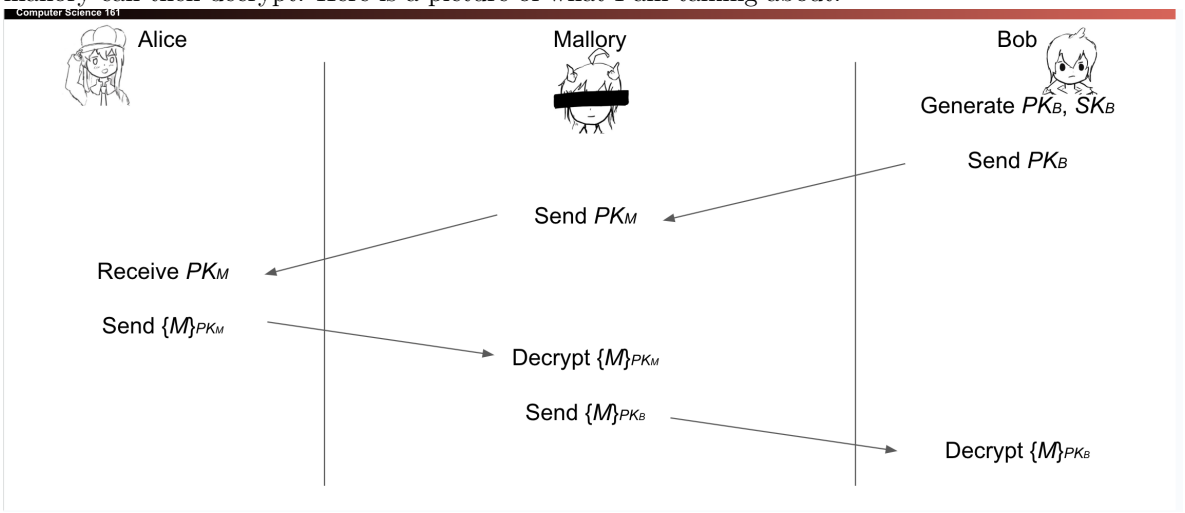
How we would generally use Digital Signatures on messages is that we will first hash the message and then compute the signature on the hashed message, this allows us to provide signatures for arbitrarily long messages.

5.3.1 RSA Signatures

Here we will go over how signatures work for the RSA encryption scheme. Like we mentioned before we establish public keys N and e and we have the private key d. So for signatures we will have the signing key be d and the verification key be N and e. So this is very similar to regular rsa encryption, but we will sign the message, and once again this is done with using the private key.

5.4 Distributing Public Keys

While asymetric encryption seems great, there is still the issue of distributing the public key. For example if alice wants to send a message to bob, we need bob's public key to encrypt the message. But while bob is sending the public key, mallory can intercept this message and replace bob's public key with mallory's public key. Alice will then use mallory's public key to encrypt the message, which mallory can then decrypt! Here is a picture of what I am talking about:



An Idea to solve this is to sign the bob's public key before it is sent, then ideally alice can verify the signature. However, the issue with this idea is we need the public key to verify the signature in the first place. So we are trying to verify the public key with the public key. So you can kind of see that this is a circular problem. To combat this we will use something called a **Trust Anchor**

5.4.1 Trust-on-First-Use

The idea with Trust on First Use is that we will have some sort of trust anchor. This is something that we will implicity always trust to not be malicious. This is because we need something to start off with that we always trust. This would be something like a trusted directory service where we can store users and their public keys to that we can use public key encryption to send messages. However the question arises, how is this any better than just having the user send their public key. Can't the attacker still do a man in the middle attack? Well, the main difference is that we should already know the public key for this trusted source, so we can encrypt the messages we send to there. We can get this public key from it being hardcoded on a device or something similar to that. We should not have to receive it over a network.

There are many limitations to doing something like this to provide integrity some include the following:

- Trust: Every single device will have to go through this anchor, meaning we have to put our full faith in this one source
- Bottleneck: Since every device needs to go through this to get public keys, there are a ton of requests, and this anchor can only handle so many at a time.
- Security: All an attacker has to do is successfully invade this trusted anchor and then we have lost all security

So what should we use instead? We will introduce what is called digital certificates.

5.4.2 Digital Certificates

Digital certificates are a way for us to provide integrity without having to use a trusted anchor. It works like the following:

- We encrypt a public key with some other person's public key
- We then sign this encryption with the encryptors private signing key
- The person who then receives this public key can verify it came from said encryptor

So if Alice wanted to send a message to Bob and Jerry sent alice Bob's public key with their private signature, if Alice knows that Jerry is trustworthy, she can safely verify that the key came from Jerry and use it to communicate with bob. It is important to note that alice did not have to go to a dedicated trusted anchor to get this digital certificate, we can get it an any way we please since we are able to verify who it came from and then decide for ourselves if this meets our integrity standards.

Like the trusted anchor, the same downfall occurs with the trusing part, we must trust who the certificate of authority is, in this case it was Jerry, So we act as if we put our full faith in them. If they act maliciously, we are done for. However, the other factors that made the trusted anchor bad, like scalability, and bottlenecking are no longer an issue, since we can have many certificates of authority. In fact many browsers have almost 100 CA's

5.4.3 Certificate Chaining

If Jerry needs to sign all certificates, you can imagine that this would take a very long time. Usually in digital certificates, there is chaining involved. So we can usually provide digital certificates for a certain subset of users and then those users can provide more digital certificates to others to split up the work. So this method of certificate chaining will form a tree.

5.4.4 Revocation

Currently in the scheme we have talked about so far, there is no way to revoke a digital certificate, once it has been accepted, this can lead to serious issues if we accept a DS that is malicous, pretty much giving the attacker all the means to do whatever they want. So we want a way to have our system detect certificates that are indicated to be revoked. There are a couple of ways to do this as mentioned in the textbook

- Certificates can have an expiration date. So all certificates will expire after some time (maybe a day) this will limit the range of attacking that can be done with a faulty certificate. However this causes efficiency issues, since after our certificate expires, we will need to request a new one every time. Many queries to these CA's can be strenuous.
- Revoked list: Each CA can have a list of revoked certificates, which would be all the certificates that they have deemed bad and have revoked in the past, Then each user can download this list and check if any of their certificates are in this list. If so, revoke them. This can also have efficiency issues, depending on how often the CA wants users to download this revocation list and how long this list is

5.4.5 Web of trust method

Another way is pretty much using a chain of users to communicate. So if Alice wants to contact Doug but Alice does not know Doug's public key, but suppose that Bob does and Alice knows Bob's public key, We can then use bob to optain a copy of Doug's public key. However there are some issues with this. First the level of trust becomes weaker and weaker the more connections we have to go through to be able to communicate with someone. Alice might trust Bob but that does not mean that Alice trusts Doug, who could have given Bob and sloppy or malicous signed certificate. It is still unknown if this web of trust will able to be implemented in the real world.

5.4.6 The leap of faith mehthod

The last method we will touch on is the leap of faith method of trusting public keys. This is used heavily in SSH. So when a user connects to a client for the first time (no previous history) and the client issues their public key to the user, the user will accept this public key blindly and then use save it to use later with further requests to this client. This means that we only have to request this public key once and if the first time goes okay, then the consequent times will also be okay. Although this leaves us open to MITM attacks during the very first connection, every other connection is secure. Also this is very easy for users since they do not need to know anything about security, the SSH client will handle it all for them.

5.5 Passwords

We will first go over some of the security risks with passwords.

- Guessing: and attacker can make educated guesses on what a user password is
- Phishing: an attacker can try to get the user to give them their password
- Eavesdropping: If a user and a client are talking over an insecure network and messages are sent as plain text (like http). An eavesdropper can use usernames and passwords
- Server is compromised: if the server that holds the passwords are compromised then all security is lost
- Malware: some malware can get user passwords

So we need to figure out how to mitigate these

5.5.1 Eavesdroppping mitigations

The easiest way to mitigate eavesdropping is to use an encrypted channel to give usernames and passwords. This is done using secure socket layer (SSL)

5.5.2 Mitigations for client-side malware

It is very hard to mitigate this, mainly because once an attacker has malware on your computer, they have lots of control. The main defense that is used nowadays to defend against this is two-factor authentication. Where you combine you login password with some type of authentication on another device, like your phone.

5.5.3 Mitigations for guessing

Since computers can run scripts to brute force guess passwords, we really do need mitigations for this. Some mitigations are the following:

- Rate-limiting: we limit how many attempts a user can use to try to login
- CAPTCHAs: We introduce an extra factor to login that theoretically only a human can solve, to prevent scripted attacks
- Passoword nudging: encouraging users to provide stronger passwords. Often things like a password strength meter are used.

The first two mitigations are useful for targeted attacks, meaning an attacker is trying to get access to a specific user's account. However these mitigations are not as useful for untargeted attacks, where we are just trying to get access to someone's account, it doesn't really matter who. Passoword nudging is good for all cases since stronger passwords will just be harder to guess.

5.5.4 Mitigation for Server being compromised

The most prevalent issue with passwords being stored clear on a database is that if the database is broken into, the attacker will have access to all the passwords. Usually it is common practice to not store passwords in the clear, so even if our database gets breached, an attacker cannot simply see all of the user information

This is usually accomplished through password hashing, so before we store a user's passwords in the database, we will hash it first to a 256-byte hash. Due to the properties of how hash functions work, the main two being that they are deterministic and are collision resistant, they make really good ways to store user passwords. However, there are some shortcomings to this, the main thing that makes hashfunctions great for this is that they are deterministic, but that makes it bad since an attacker can brute force guess passwords and put them through this hash function, An attacker can do an untargeted attack and since the hash is the same for everyone, this doesn't really provide much more security.

So we will need to use some modifications when hashing. We will introduce what is called a salt, which is pretty much just an extra parameter which can make a hash with the same source still different, as long as we have two different salts.

The last thing that we need to do it limit the ability to brute force these attacks. One nice things about hash functions is that they are fast, but since they are so fast, an attacker can brute force these very easily. So what is done to mitigate this is to make a slow hash function, which is usually done by a functions that iterates over hash functions for many times.

6 The Web

6.1 Intro to the Web

6.1.1 The basics of a URL

A URL (Uniform Resource Locator) is what is used to maneuver around in the web. It is formatted in a specific way:

```
http://evanbot@www.cs161.org:161/whoami?k1=v1&k2=v2#anchor
```

```
http://www.example.com/index.html
```

- Protocol: This is the very first part of the URL that is in blue. It describes how we will retrieve the resources describe throughout the rest of the URL. The ones we need to know are HTTP and HTTPS
- Location: This is after the URL. This is usually the domain name of where to look on the web for the resources. Optionally, you can provide a user (user@domainName)
- Path: After you get to the proper domain of where you want to go, you have to specify what resource you want, this can be done through specifying a path to look for those resources. It is much like a file path
- **Note these first three parts of a URL are mandatory for each URL. The parameters below are optional**
- Arugments: we can provide arguments to our path. The arguments will start with a specified ?, then key=value, with an & separating each.
- Anchor: Lastly we can provide an anchor which is usually for scrolling to a certain part of a page, this anchor is not sent to the server but is available on the web browser.

6.1.2 HTTP Basics

HTTP is the most common protocol for sharing and getting resources over the web. Here is a picture of what a HTTP request might look like:

```
POST /login HTTP/1.1
Host: squigler.com
Content-Length: 40
Content-Type: application/x-url-formencoded
Dnt: 1

username=alice@foo.com&password=12345678
```

The first line contains the method of the request and the HTTP version we are using. In this class we will be using the following two methods:

- GET: when we want to retrieve resources from the client
- POST: when we want to give resources to the client

The next header lines will be the following:

- Host: Who we are requesting to do the given HTTP method
- Content-length: How long the message body is
- Content-type: the type of content in the response
- DNT: denotes whether this HTTP request should be tracked.

6.1.3 Elements of a webpage

There are three main elements to a web page:

- HTML: HTML's purpose is to display information through text, we can do things such as plain text, links, and much more
- CSS: this is used to style our HTML output to make it look pretty
- JavaScript: This provides much of the functionality of a website, including buttons, interactivity and much more.

It is important to note that CSS and Java script are very powerful if exploited.

6.1.4 Same Origin Policy

when you have multiple web pages open at the same time, it poses a risk that if you have a malicous website open, it can look at and tamper with other websites that you also have open. To combat this, each website is isolated from each other. The only exception to this is if two websites have the same origin, two websites have the same origin if the following is true:

- Same protocol
- Same domain
- Same port

Note that all three must be the same for the websites to share information with each other. Take this following example of these two websites:

- http://wikipedia.org
- https://www.wikipedia.org:82

We will compare the protocol, domain and port of these two URLs.

- protocols do not match. The first one uses http while the other one uses https
- The domains do not match. The difference is one of them uses www. which makes it a different domain
- The ports do not match. The first link uses no port while the second one does

Note that you only need one of these constraints to fail for the same origin policy to fail. We just decided to use an example that failed all three to see how the comparison works.

There are a few special exceptions to the same origin policy. We will go over them right now:

- **Scripts:** If a page loads a javascript, its origin will be the page that loaded
  - if we loaded `<scriptsrc="http://google.com/tracking.js"></script>` on `http://cs161.org`, the origin of this script is `http://cs161.org`
  - Another note about javascript is that is can communicate with webpages from different origins with the function **postMessage**. The functionality however is very limited.
- **Images** If a page loads an image, its origin will be wherever the image comes from
  - if we loaded `<imgsrc="http://google.com/logo.jpg">` on `http://cs161.org`, the origin of this script is `http://google.com`
- **Frames:** Frames will have the origin of wherever the frame was retrieved from
  - if we had `<iframesrc="http://google.com"></iframe>` on `http://cs161.org`, the origin of this script is `http://google.com`

6.2 Cookies

6.2.1 Cookies Intro

Cookies are another very important component of the web. Each cookie is just a name-value pair to save some state about the session with the browser and the server. These cookies are stored on your browser. So for example, If I enable dark mode on a website, If i moved around on this website, I would still want the be in dark mode in subsequent requests. So the browser will store something such as **DarkMode=True**.

6.2.2 Cookie Attributes

Cookies have 5 attributes:

- **Domain & Path**: Tell the browser which URL to send the cookies to
- **Secure**: If set to true, we should only send the cookies over https for security
- **HttpOnly**: Javascript cannot access and modify the cookies
- **expires**: Indicates how long the cookie is valid for

Two very important cookie attributes are the **Domain** and **Path** attributes. These are the attributes that will determine which cookies are sent to a given URL. This is important because we do not want to send all of our cookies to each URL we access. If the following criteria is met, we will send a given cookie to a URL:

- If the **Domain** attribute of the cookie is a suffix of the URL's domain
- If the **Path** attribute of the cookie is ap prefix of the URL's path

We can think of it like this, if these don't match, then the cookie would have no real use in this location on the web. So only the relevant cookies are sent to each URL.

For example a cookie with **Domain with google.com** and a **Path /Enzo/Stuff** would be included in a URL such as http://foo.google.com/Enzo/Stuff/index.html since the Domain attribute is a suffix of the URLs domain and the Path attribute is a prefix of the URLs path.

6.2.3 Setting Cookie Domain and Path

Since we only want to send cookies to URLs that match the criteria we mentioned above, when are these attributes set? Well, When a cookie is set, the Domain and Path attributes must be relevant with the current servers's URL, So for example google.com can cannot set a cookie with the Domain attribute of amazon.com, since amazon.com is not present in the current server's URL. The same is true for the Path attribute, it must be present at a prefix in the current server URL path. This prevents people from making setting cookies that can go to different URLs, because this can lead to malicous behavior. For example if evil.com was able to make cookies that would be sent to bankofamerica.com, that would be an issue.

6.2.4 Session Tokens: A Special Type of Cookie

Imagine you are logged in on a website. You would want to stay logged in for many request to this server. This aspect is so important that it gets a special name: a session token. All a session token is is a cookie, but this cookie will hold a value that will automatically verify the user. So when a user succesfully logs in to a server, the server will create a session token and send it to the browser. For subsequent request, the browser will send this cookie along with the request to verify the user automatically, making it very user-friendly.

Session tokens should be both random and unpredictable so attackers cannot guess them. Usually, with these tokens especially, the Secure and HttpOnly flags will be set for top of the line security, since this is a very important and sensitive cookie.

6.2.5 Cross-Site Request Forgery (CRSF) Attack

An attack involving cookies is a CRSF attack. In this type of attack, we pretty much fool a user into clicking a link that does something that is not intended. For example we could fool the user to click a link such as `http://example.com/logout`. If the user clicked on this. Their browser will attach the session token, the server will accept it and will log the user out.

We can do much worse things with this, take this example: `https://bank.com/transfer?amount=100&recipient=mallory`. If a user clicked a link that did this, it we would send $100 to mallory.

Common ways to do this are to have the user click on links that perform an HTTP POST request, since post requests can change the server state, in this post request, a common way to perform this attack is to use an HTML form to make the malicious request and then send this to the server. So even if the user does not want to do this, if they click a link that does this, the server will deem it okay since the session token matches the user.

6.2.6 CRSF Token

CRSF tokens are a way to provide integrity to web requests. We mentioned in the previous section that the attacker can form an HTTP post request that, if the user has a valid session token with bank.com, it would transfer $100 to mallory. This worked previously because before we had no way of identifying who actually sent the request, the user or the attacker.

The solution to this is the CRSF token. Basically, when we are on bank.com, the server will generate a CSRF token to attach to the user's session. This token will then be verified on the server end when they get requests. Since only the server knows how to generate the Token, the attacker can no longer send requests, since they do not know the token, even if they tried to bypass it by providing a token, there is a very very very high change that it will be incorrect, since the CRSF token is meant to be random and unpredictable so we cannot generate it. If the token does not match what the server has, the request will be rejected.

6.2.7 Referer Validation

Another method of defense for CRSF attacks is using the Refer header in the HTTP request. This header pretty much just says where the request came from. It could be something like google.com. So the server can check this field and determine if the referer is suspicuous or not and decide whether or not to reject the request. This is good extra-defense along with CSRF tokens but this should not be used on its own. The reason for this is that it is too much up to interpretation whether a referer is valid or not. Many browsers will simply omit the refere header, which means it would be blank when a server receives a request, it is up to the server to say whether a blank referer is valid or not. So just to be safe, this should just be used on top of CSRF tokens.

6.3 Cross Site-Scripting (XSS) Attacks

Another form of attacking on the web is an XSS attack. In this type of attack, we attempt to inject malicous javascript code onto a website of another domain. Usually we should not be able to due this because of the same-origin policy we defined earlier. That is, we cannot look at or modify anything that is not the same origin as us. However, this says nothing about storing scripts on servers in a "valid" way or sending HTTP requests that have user input. There are two main types of XSS attacks that we will go over: Stored XSS and Reflexted XSS

6.3.1 Stored XSS

In a stored XSS attack, an attacker finds some way to store malicous javascript code onto a server. An example which is also in the textbook when a Facebook post is made. When this happens, Facebooks server will store information about the post so when the user clicks on it, they will be able to see what their friend's posted. If an attacker makes a post, they can inject a malicious script into there, so when other users click on this post, it will run that attacking script.

6.3.2 Reflected XSS

The other type of XSS attack is reflected XSS. This attack relies on an HTTP request allowing user input and that same request displaying what the user input was. So for example in search engines you need to provide user input, and some will say what you searched for. So if I provided a malicous script as my user input, and the request dispalyed my input, it would then run the malicious script.

6.3.3 XSS Defense: Sanitization

One thing you might think of to defend against XSS attacks is to sanitize HTTP requests. You could possibly search through the HTML body for keywords such as ¡script¿ and remove anything inside these tags. However attackers can get very clever to avoid this, Take the following examples:

- You could never use the `<script>` tage: `<imgsrc=1href=1onerror="JavaScript:alert("XSSattack!")"/>`
- You can plan for a sanitation of the `<script>` tag: `<scr<script>ipt>alert("XSSattack!")</scr<script>ipt>`
- You can replace the <and >characters with &lt and &gt. These will be read at such in the HTML body but will be interpreted as the brackets when it is processed

These are things to definitely consider when sanitizing a script. Luckily there is a set standard for sanitation which is pretty solid. We will not discuss it in these notes as it is not in the textbook.

6.3.4 XSS Defense: Content Security Policy

This defense mechanism pretty much says that we can only run scripts that are from certain domains. So for example the example.com domain could have a policy such that is will only run scripts from *example.com. That is any domain that ends with a suffix of example.com would be considered a safe domain to run scripts from. This completely gets ride of being able to run scripts encoded in html code. So we have to run all of our scripts externally, completely removing the opportunity for

an attacker to do this type of attack. How the server will determine what domains are safe are usually through an extra HTTP header called Content-Security-Policy, which will define our safe domains.

To iterate it once more, we can only now run scripts through using the `<script>` tag and an external reference to the script. We cannot insert it directly in the HTML body.

6.4 UI Attacks

UI attacks are the way that we execute many other attacks we have discussed so far like XSS and CSRF.

6.4.1 Attack: Clickjacking

Clickjacking is a UI attack in which the attacker attempts to steal a click from the user. This means they try to get the user to click on something that gives them control of their browser, being able to use their cookies and being able to do attacks that we have mentioned previously. Here are some ways clickjacking is done:

- **Fake Download button:** Stealing a user click
- **Manipulative HTML form:** A user could think a form sends $5 but it actually sends $100
- **Fake cursor:** Get the user to think their mouse is somewhere else than where it acutally is
- **Draw a browser page, with malicious links behind it.**

These attack heavily rely on a naive user. In which there are many out in the world, who will click on links without thinking twice about it. So we need ways to get the user to think about their decision more when interacting with UI

6.4.2 Defense: Clickjacking

- **Pop-ups:** When the user clicks on something, have a pop up to ask again if they are sure they want to continue
- **UI Randomization:** Randomize where UI elements are on a webpage it is harder for an attacker to insert malicious buttons
- **Direct attention to user click:** We can do stuff such as freezing the rest of the page or highlighting the user's curso so they are aware of what they are clicking
- **Delay click:** Have it so a user has to hover over the UI element for enough time before they click, so they don't misclick on accident.

With these defenses, it is important to take in account human factors in terms of what they would like for a website. We don't want to become so defensive that the web page becomes a hassle to use.

6.5 SQL Injection

6.5.1 Attack

Almost all servers nowadays store their data in an SQL database, and attackers have found ways to use SQL malicously to steal informaton from these servers. Take a look at the following picture from the

For example, consider a website that stores a SQL table of course evaluations named `evals`:

id	course	rating
1	cs61a	4.5
2	cs61b	4.4
3	cs161	5.0

A user can make an HTTP GET request for a course rating through a URL:

```
http://www.berkeley.edu/evals?course=cs61a
```

To process this request, the server performs a SQL query to look up the rating corresponding to the course the user requested:

```
SELECT rating FROM evals WHERE course = 'cs61a'
```

Just like the code injection example, if the server does not properly check user input, an attacker could create a special input that allows arbitrary SQL code to be run. Consider the following malicious input:

```
garbage'; SELECT password FROM passwords WHERE username = 'admin
```

textbook

We we see that http://www.berkeley.edu/evals?course=cs61a Will perform a SQL query, utilizing what we passed in as input. So it is very easy for an attacker to get sensitive information out of this. This is how we formulated the injection attack.

- Provide **garbage** for the original intended input. We do not care about this at all
- Add a quote (') This is to end the string associated with **garbage** and start a new SQL query.
- Add a semicolon to end the current query so we can start another one.
- Next is to make the injection, whatever query we think will get us the information we are trying to steal. In this case we get the admin's password!

Now lets look at another example from the textbook: This case is attempting to log in, usually when we try to login to a server, it will attempt to access the SQL database with the given username and password. If any rows are returned, then we deem it a successful login. Here is an example:

```
SELECT username FROM users WHERE username = 'alice' AND password = 'password123'
```

So if there are any rows where the username is 'alice' and the password is 'password123', the number of rows will be greater than 0 and we will deem it a successful login. What if we did something like this?

```
SELECT username FROM users WHERE username = 'alice' OR 1=1;
SELECT username FROM users WHERE username = 'alice' AND password = '_____'
```

Here we use 2 SQL queries, the one we really care about is the first one. We make the first query to return rows where the username is 'alice' or 1=1. As we know this logical statement will always be true, hence we will just get all of the rows for the users. As long as at least one user exists in the database, the server will deem it a successful login. So we were able to log in even though we did not know any user's password. Note that we have another query here, but this is simply so the entire query still works, we do not care about what the second query outputs. Something clever we could do is to use SQL's comment syntax (–) to comment out the other query so we just get this:

```
SELECT username FROM users WHERE username = 'alice' OR 1=1--' AND password = 'garbage'
```

6.5.2 Defense

Escaping Inputs:

The defense of escaping inputs has the do with the following: We will basically prepend an escape character to any character passed in that can be used as an attack. for example in the previous attacks, the single quote (') was used to end the string early, so the attacker could write another SQL query. Converting the single quote to \' will treat it as a literal quote and will not end the string. Similarly we can do this for other dangerous characters such as the comment syntax (–).

This is a good start to defending SQL injections but is easily dodgeable, for example, the user could expect this and pass in \' in which our escaper might produce \\' which out escape the escape character but not the quote, which will produce the attack like we mentioned before. So generally it is never advised to build your own escaper, there are ones that exist already that you should use instead.

Parameterized SQL/Prepared Statements.

The next type of defense is called Parameterized SQL/Prepared Statements. In this defense, all SQL queries are compiled before inserting user parameters, this way nothing that the user puts in can be interpreted as a SQL query. So if they try to do an SQL query, it will likely that they will just get nothing back because everything they put will be not be interpreted as a query. Hence this is really the best way to defend against injections, because the attacker cannot inject any queries at all!

The biggest setback with this defense is its compatibility, Since SQL is so general there are many different versions of it. So if your SQL library does not support parameterized SQL, just switch to one that does.

6.6 CAPTCHAs

CAPTCHAs are used mainly as a defense from brute force attacks, such as brute forcing a user's password or DoS attacks. The primary goal of CAPTCHAs are to provide problems that are easily solvable by humans but very hard to solve by computers. This makes so the web is very-human friendly but not bot friendly. Typical CAPTCHAs you might have seen is googles CAPTCHAs where it asks you to identify all the pictures with, example a bus. Other common ones are images with funky letters and numbers and it asks you to type in what those characters are and so on.

The current issue with CAPTCHAs is that computers are getting faster by the day and can solve algorithms quicker, just because CAPTCHAs are hard to solve by computers, it does not mean it is unsolvable. So as computers get better, this security gets weaker. Also CAPTCHA solving services exist, in which you pay a small fee, possible $0.10 per CAPTCHA and an outsourced human will solve it for you. This is very reasonable for an attacker if what they are trying to access is worth more than a couple dollars to them.

7 Networking

7.1 Intro

Before we discuss network security, we must know how networks are generally set up. This will serve as this intro.

7.2 MACs IPs and Ports

7.2.1 MACs

MACs are a unqiue identifier of a particular machine. One exists on every device such as you phone, laptop, tablet and so on. It is a unique identifier of a particular machine. The structure is 6 pairs of hex numbers, each separated by a semi-colon. An example computer MAC can be the following: **ca:fe:f0:0d:be:ef**. There is a special MAC address which broadcasts a message to everyone on a local network, it is: **ff:ff:ff:ff:ff:ff**

7.2.2 IP Addresses

IP Addresses are a unique identifier that is location specific. So we talked about with MAC addresses that these are baked into your computer, they cannot be changed. This is not the case with IP addresses. Depending on where you are on the globe, your IP address can change. This is mainly because when someone wants to send a message to someone, we have to find out where to send the message to. Providing a location specific address will allow our message to get to where it needs to go. This usually involves sending the message over many routers, which know where to direct the message to eventually get to the recipient.

There are two versions of IP Addresses:

- IPv4: uses a 32-bit addresses, where we use 4 integers between 0 and 255, so each number will be a byte long. an example is the following: **128.32.131.10**. So there are $2^{32} \approx 4$ billion unique addresses
- IPv6: uses a 128-bit address, where we have 8 2-byte hex combos. It would look something like this: **cafe:f00d:d00d:1401:2414:1248:1281:8712**. So there are 2^{128} unique addresses, which will be more than plenty probably forever.

7.2.3 Ports

When a connection is made, the port allows us to establish a place on the machine we are connecting to to send and receive messages. So when we connect to a port, the receiver knows to look at that port for messages from some sender.

7.3 The 7 layers of the Internet

7.3.1 Layer 1: Moving Data

The very first thing we should be concerned with when trying to communicate with other devices is how we are going to move data from one machine to another. There are a multitude of ways we can do this, we can use humans, wires, pigeons, it really doesn't matter as long as we accomplish the job. Typically two most common ways of transmitting bits across devices are wires and WIFI.

7.3.2 Layer 2: Local Area Network (LAN)

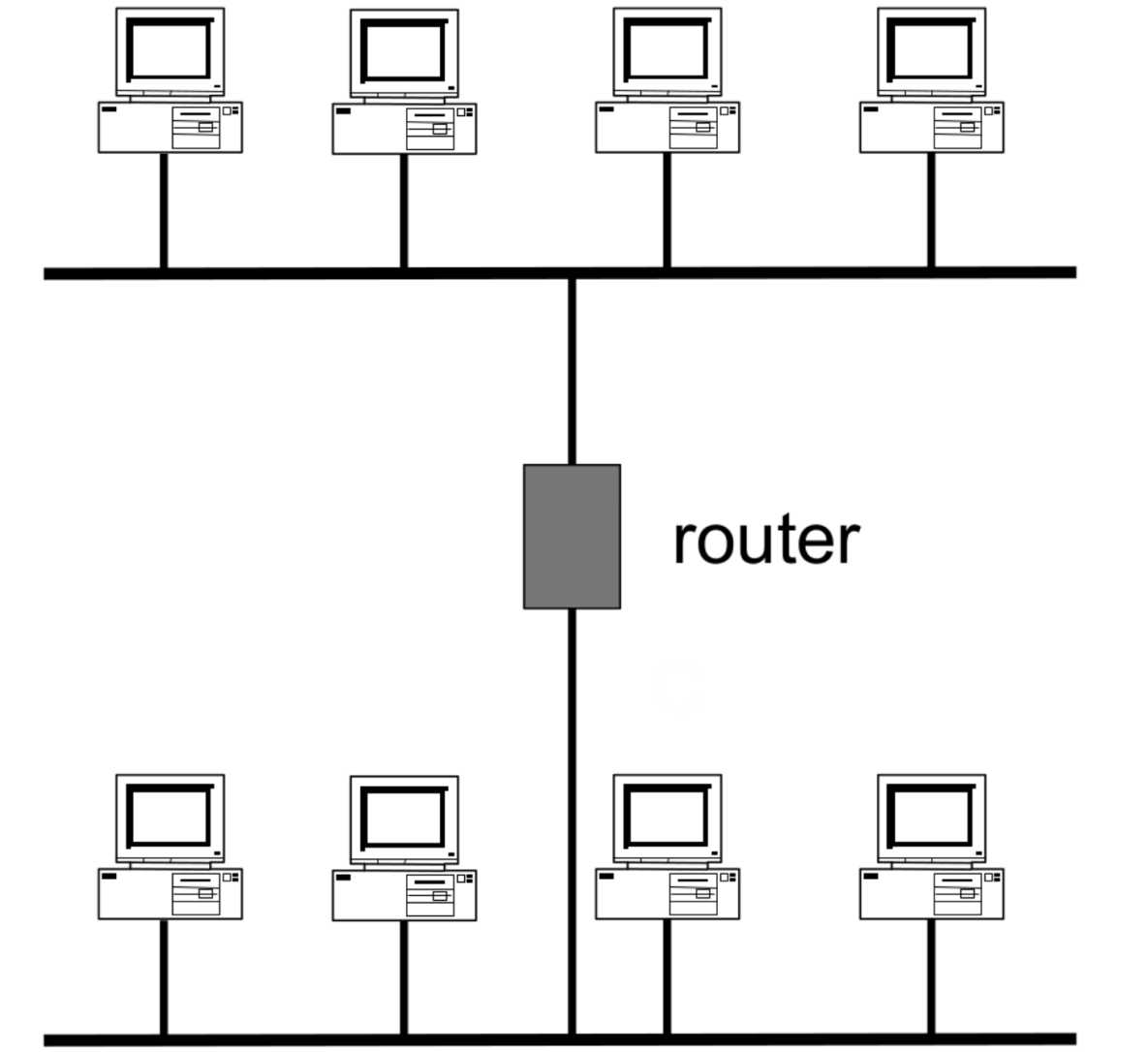
After we have finished layer 1, we can use this building block to connect a bunch of computers within the same local area. This is called a LAN. We connected devices together to form a LAN. This layer relies upon computers already being able to reliably send bits to each other via some medium. We use this method to communicate with a bunch of local computers.

7.3.3 Layer 3: Internetwork

Once we have a bunch of LANs all over the place, it would sure be nice to have a way to connect all of these LANs together. Doing this allows any computer to talk to any other computer. If we are able to do this, we are very close to building what we know as the internet.

Previously we talked IP addresses. This would be which layer it would fall into. The point of this layer is to be able to send packets from one devices to any other device. Using IP addresses will allow

us to be able to get a message from one end of the world to another. Thankfully, we use routers to connect two LANs together in a very organized fashion. Here is a picture. Doing this will allow us to send messages either to the computers connected in our LAN if the recipient is in the LAN, or send it to the router to send to some other LAN if the recipient is not in our LAN



7.3.4 Transport

We will not get into transport too much right now, but the main point of the transport layer is to be able to provide the service of sending variable length packets from one device to another device. Some protocols that exist on this already are **TCP** and **UDP**.

7.3.5 Application

This is the highest level of abstraction for a network, this is where many things that we use on the daily. Things such as the internet, online video games, and online services are all examples of the application layer.

8 Headers

Headers are essential to networking. The main point of headers is to be able to provide information that each layers needs to know to do their job. Take this example. Alice is sending a letter to Bob,

what really matters here is the message that Alice is sending to Bob, but we also need to do some things such as provide where bob lives. That way we can give our message to a post office, which can get the message to bob's address, At the point where Bob receives the letter, he does not care about the address anymore and can disregard it, to see the message that alice send (who knows maybe is a love confession letter)

This is general Idea with headers, When we want to send some packets across a network, we need to go down the layers of abstraction to add the appropriate information, that the respective layers of the recipient can unpack and do things such as verification, seeing who the message is from and so on. Once the recipient layer is done with a given header, we can throw is away, it will no longer be useful for the above layers. So if you forget everything about headers, remember these three things

- When the sender is sending a message, we go down the abstraction layer and append headers
- when we are receiving a message, we go up the abstration layers and toss out headers, after we read them
- Each header will only be useful for a specific abstraction layer, it will be seen as garbage for every other layer.

8.0.1 The different types of network attackers

Here is a table of the types of network attackers we will be talking about

	Can modify or delete packets	Can read packets
Man-in-the-middle attacker	✓	✓
On-path attacker		✓
Off-path attacker		

8.1 ARP

8.1.1 ARP Protocol

ARP is a networking protocol that has to do with finding the IP address of some receiver. So if alice wanted to send a message to bob, alice needs to somehow find bob's IP address so the message can correctly get there. The protocol is pretty simple, it goes like the following:

- Suppose we know the IP address we want to send our message to.
- We will broadcast (ask everyone) to our LAN if they know who's MAC address corresponds to the IP address
- If the receiver is inside the LAN, they will respond with their MAC address.
- Else the router will respond with their MAC address
- The sender will cache this MAC address associated with this IP and send the message

In this protocol, we should only expect one response either from the correct person in the LAN or the router. So it makes sense for the first response we receive, we will cache it and end the request. This is the key part of ARP spoofing.

8.1.2 ARP Spoofing

Since in the protocol we described above, you will wait for your first response and then exit the algorithm, an attacker could lie and say, "yes this is my IP address and my MAC address is (blah)" The sender would then accept this, cache the attackers MAC address and then send the message meant for bob to the attacker. This is an example of a race condition. Essentially, it is a race to see who will send alice their information first, bob, or mallory.

8.1.3 ARP Spoofing Defense: Switches

Previously, every single person on the network had a cache of IP to MAC addresses. The idea here is to have a much larger cache, called a switch that everyone in the LAN goes to when trying to send a message. Since with ARP spoofing, there is no real way of being able to tell if a message is coming from someone malicous or not, The best compromise that we have is to not broadcast as much. Since we are holding many more mappings in a much larger cache, this will allow us to not have to broadcast as much, hence being less vulnerable to ARP spoofing.

8.2 Dynamic Host Configuration Protocol (DHCP)

DHCP is a protocol for when we need to be conifgured in a network. We can think of getting configured like being initialized. For a user to connect to a network, They need a few things:

- Their own IP Address
- IP Address of the DNS server
- IP Address of the router

These are the steps for the DHCP protocol

- When a new client wants to connect, they broadcast this request.
- Usually, the DHCP server will respond with the information needed to for a client to connect, Note it is possible for there to be multiple DHCP servers, so they will all send the things to the client that they need to connnect
- The client will then broadcast to the network which DHCP offer that they chose. For the server they chose, they will take note of the IP address they gave for the client and mark it as being used. For all the other offers that did not get accepted, it will be able to use that IP address for another connection

8.2.1 DHCP Attacks

Similar to ARP spoofing, an attacker can fake being a DHCP server and send a user the stuff needed. The key part of the offer is that the user needs the IP address of the router. An attacker can send their own IP address, so whenever a user needs to send a message outside of the LAN, it will go to the attacker. And again, there is no real way of confirming that an offer made is coming from a real DHCP server. So this is another race condition between the attacker and the DHCP servers.

8.3 Promotion of a successful attack

In either ARP of DHCP attacker, if an attacker was successful, they get promoted to being a man in the middle (MITM) attacker. Since they can intercept messages, where they can read them, change them, or delete them.

8.4 WiFi Backgroung

WiFi belongs in the second layer of the networking protocol, as a means to transmitting data from one device to another. In many ways for this class, it is similar to ethernet, The differences it has in respect to how it works is out of scope for this class. Here are the basics of how WiFi works for what we need to know.

- A WiFi network will have an Access Point (AP), which will continuously be broadcasting, that it is present and will say its name. This is called the Service Net Identifier **(SSID)**
- If a WiFi network has no password, your computer can automatically join to it, in this scenario, All data is transmitted the network without encryption. Meaning anyone else on the network is able to see the packets you are transmitting and possible inject malicious data.
- If a WiFi network has a password, there is a protocol that exists to secure the communications, This protocol is called WiFi Protected Access: Pre-Shared Key **(WPA2-PSK)**

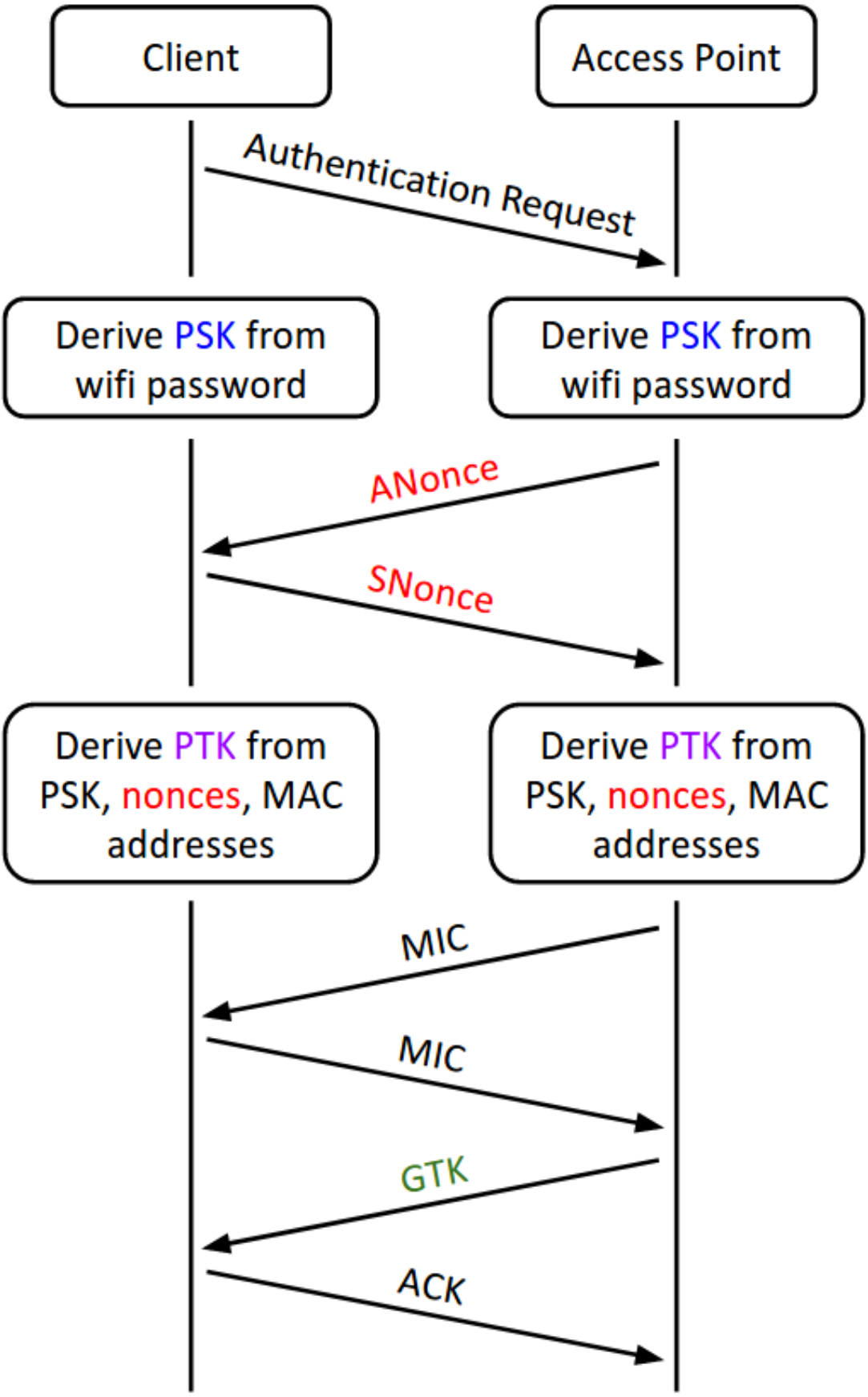
8.4.1 WPA2-PSK Protocol

This protocol allows for allow users that connect to the WiFi network to have secure communications by cryptography.

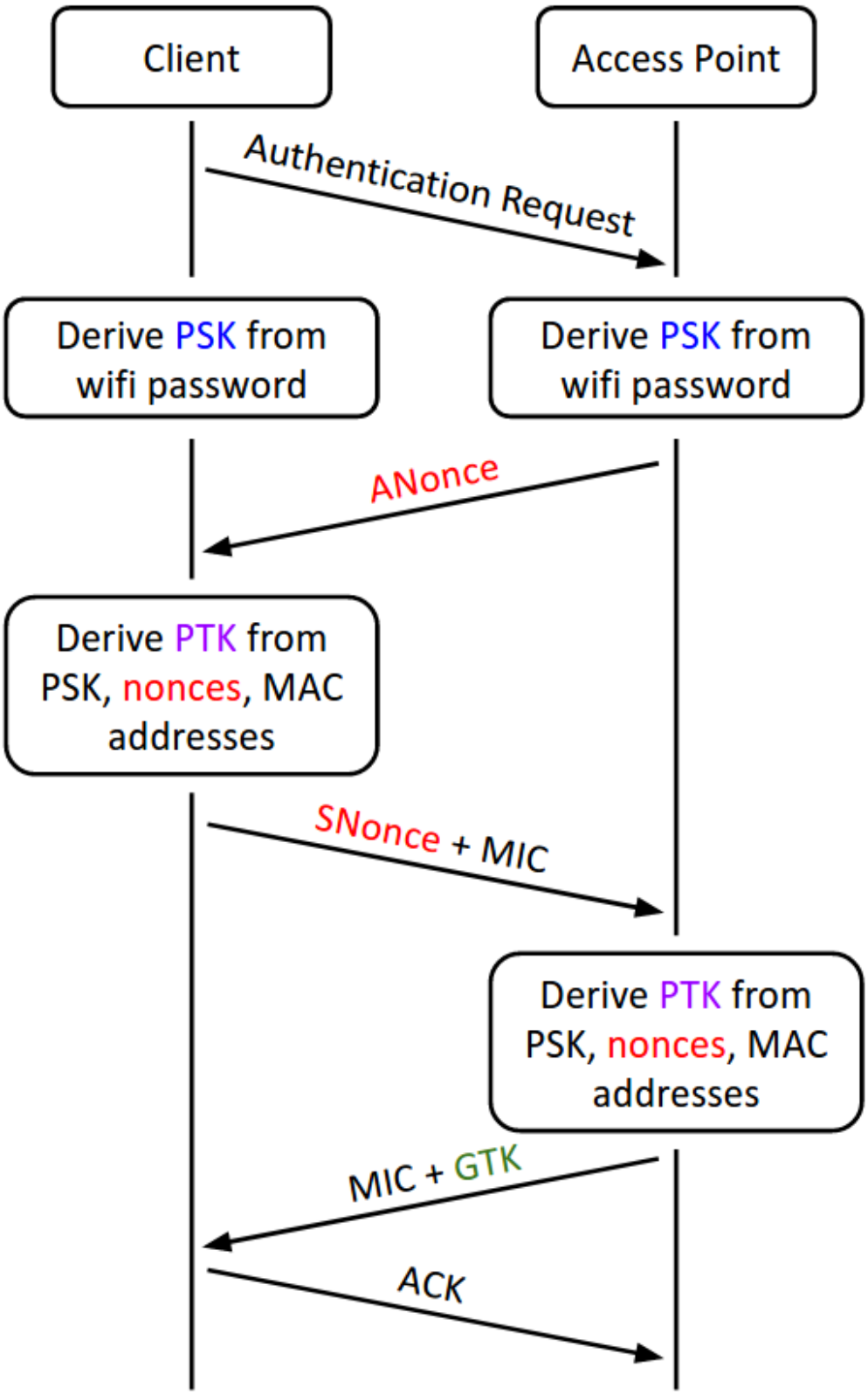
Some basics about the protocol, is that it has a password that all users must now to be able to log into the network. This is the password you enter when you want to log on. The protocol is as follows:

- Client requests access to the WiFi's access point.
- Both the access point and the client will derive the PSK from the WiFi password to connect to the network
- Bot the access poitn and the client will generate a nonce. Let the client nonce be called **SNonce** and the access point nonce be **ANonce**
- At this point, both the client and the access point have genreated the PSK and have both nonces
- With the resources from the previous bullet and the MAC addresses from both the client and the access point. Both the client and the access point will generate the **Pairwise Transport Key**
- Next, the access point and the client will exchange MACs (Note these are the cryptography MACs, not the MAC addresses). The point of this is the check that no tampering was involved in the protocol, specifically with the nonces.
- If all goes well, the access point will encrypt the Group Temporal Key **(GTK)** and send it to the client
- Lastly, the client will send an acknowledgement message to indicate that is successfully received the GTK

The purpose of the GTK is that it is used to broadcast messages across the whole network. Remember, this means it sends the message to the MAC address of all f's to do this. Since the GTK is the same for everyone, this provides a neat and simple way to encrypt messages that you want to send to the whole network. Below is a graphic describing this protocol


Client
Access Point
Authentication Request
Derive PSK from wifi password
Derive PSK from wifi password
ANonce
SNonce
Derive PTK from PSK, nonces, MAC addresses
Derive PTK from PSK, nonces, MAC addresses
MIC
MIC
GTK
ACK

You see that this protocol involves a 6-way handshake, we can optimize this down to a 4-way handshake by combining some steps. If you undertsand the 6-way handshake, you should be able to understand why the 4-way handshake still works. A diagram is provided below.



8.4.2 WPA2-PSK Attacks

One thing about this protocol is that everything except the GTK is send unencrypted, meaning if there is an on-path attacker and they can eavedrop on this protocol, they can learn the nonces and MACs necessary to generate the PTK. Using this, the attacker can eavesdrop and decrypt all messages that are on the network. Even if the attacker is not on the network, it is possible for them to try and brute-force the password, which could allow use the PSK to generate the PTK. This is definitely possible to brute force, depending on the strength of the WiFi password.

8.4.3 WPA2-PSK Defense

The main issue with the previous protocol is that all that was required to gain access to the WiFi network is the WiFi password. After that, everything else falls into place. To combat this, it would be nice to add another layer of security, perhaps providing some uniqueness to each user. the WPA-Enterprise scheme does this. It gives each user and unique username and password. This allows the user to prove their identity to the WiFi network, once this is verified, the client is given a Pairwise Master Key **(PMK)** instead of the PSK from the previous protocol. Note that the PMK is generated randomly. Since this is the case, there is no feasible way to brute force this, making it much more secure. Note this protocol is still note secure if for whatever reason, an attacker already has access to the server.

8.5 IP Routing

8.5.1 Intro

In this section, we will talk about IP routing. More specifically, when a sender want to send a packet to someone. How do we know where to send it?

8.5.2 Subnets

Subnets are a group of IP addresses that have the same prefix. This is a very common way to identify a group of computers in a local network. The format is as folows:

$$128.32/16$$

Any IP address that had this prefix, would be in the same subnet. For example, in this one, there are 2^{16} IPs in this subnet. We know this because the prefix contains 16 bits, meaning the suffix contains $32 - 16$ bits. So for example, all of your devices in a house are probably in the same subnet.

8.5.3 Sending Packets

So when I want to send a packet, how do I know where to send it to? There are 2 possible scenarios.

- **The address is in your local network:** You can figure out this by checking if you are in the same subnet as who you are trying to send the message two. You then can use **ARP** as discussed before.
- **The address is no in you local network:** If your address is not in your local network, you will have to use something called Border Gateway Protocol**(BGP)** to figure out where to send it.

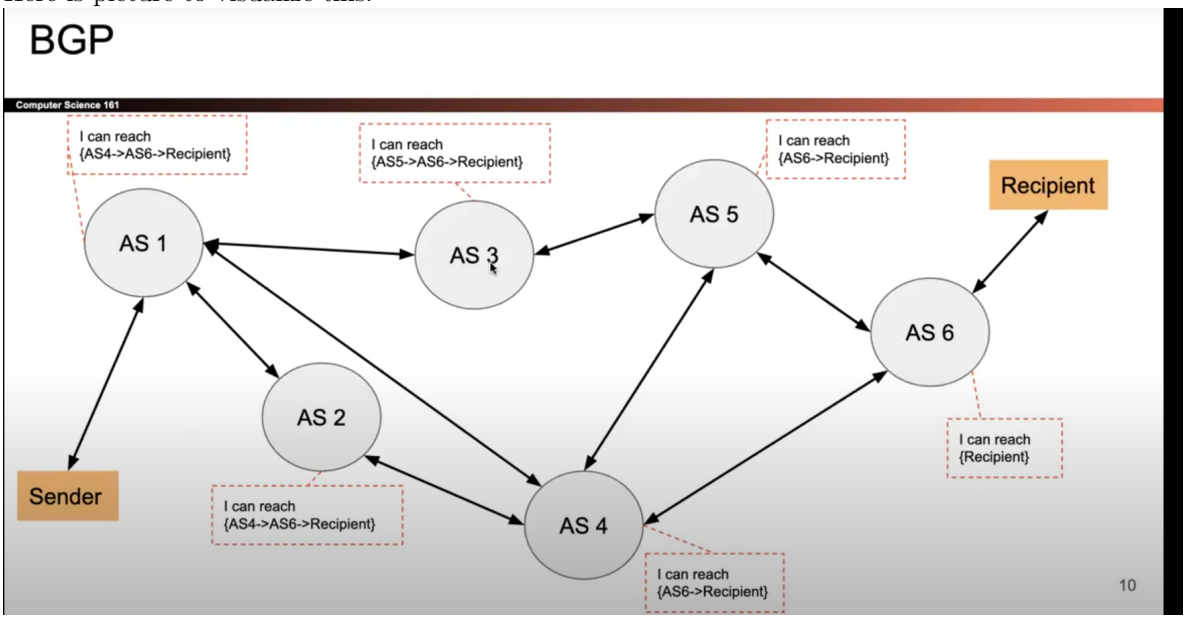
8.5.4 BGP

BGP will tell provide a way to transmit data (via packets) in the right direction. The will be a very high level overview, just so that we know enough to talk about the security aspect. It goes as the following:

- The gateway closest to the destination address will announce it.
- Its neighboring gateways (the edges) will announce their path to the destination

- when the gateway wants to send a packet out of its local network. It will check all the gateways its connected to, and find the shortest path to the destination.

Here is picture to visualize this:



You see that we send through things called Autonomous Servers **(AS)** these are pretty much just a fancy local network. They are usually businesses, corportations, universities, and other things along that line. They usually have many local networks.

8.5.5 BGP Attacks

The threat that BGP is exposed to is that any attacker can simply lie about their path to the destination. For example, mallory can claim they have the shortest path to the destination (even thought they don't), the packets will be sent the them instead. At this point, mallory is a man in the middle attacker.

8.5.6 Transmission Control Protocol (TCP)

TCP is a layer 4 protocol, involving the transmission of bits. It relies on the IP protocols of layer 3. All that the IP protocols will do is try their best to send the packets where they need to go, but there is no real guarantee to if the packets will actually get there and that they are not tampered with. This is where TCP will come into play. Here are some of the roles that TCP plays.

- **Handling the order that packets come in:** Since there is no guarantee that the order of the packets when they arrive will be the same as when they were sent, TCP provides a way to put them back in order. It does this by marking each packet with a number, that way when they are received, the receiver can simply rearrange them based on those numbers.
- **Handling when packets get dropped:** TCP will also send an additional packet back to the sender to indicate that the packet has been received. If the sender never gets this message, we can assume the packet got dropped and attempt to send it again. If the acknowledgement message gets dropped, this protocol still works, since the sender will simply just send the message again.

This does a couple things:

- Provide a byte stream abstraction: Lower levels no longer have to worry about what order the bytes get sent
- Lower level layers do not have to worry about if the packet got sent due to the acknowledgement
- Allow multiple services to use the same IP address using ports

8.6 Ports

Ports are a way to distinguish multiple devices that have the same IP address. This allows a user to have multiple connections in a computer happening at the same time. So if I have two connections to google going at the same time, These would be coming from different ports, so google knows which person to send packets to. The port numbers themselves are not really important for private computers, however for publicly known computers (like servers), the port is what allows the client to connect, since they need to know where to connect.

8.6.1 TCP: The Protocol

TCP is constructed in the following way:

- A client and a server make a connection
- client says they are going to start sending packets at sequence number x , and acknowledgement of y and the sequence is of length A.
- the server will then send its response, that will start at the clients acknowledgement number y and have length B. And it will also send an acknowledgement saying that they received the clients request. The acknowledgement will be the number $x + A$.
- if the client wants to send another message, they will start their next sequence at the last acknowledgement number, in this case $x + A$ and will send an acknowledgement, they it has received up to $y + B$ so far.

So the way we can think about this more high-level is that the client will first attempt to make the connection with the server, by sending a request to connect, and if they connect, they will also send a sequence number of which they will start sending their packets at, call it x. If the client accepts the connection, they will also send a sequence number, say y and acknowledge that the next byte they send should start at $x + 1$. Then the client will send their first message, starting at the the client's acknowledgement, $x + 1$, acknowledge the client, noting that their next message should start at $y + 1$ and that their message is of length A. The client will then send their message, starting at the clients acknowledgement $y + 1$, saying their message is of length B, and lastly acknowledge that they have received the clients message, noting that their next message should start $x + 1 + A$. This pattern will go on until the connection is terminated.

8.6.2 Terminating a connection

When we want to stop the TCP protocol, there are two ways we can do this.

- **The nice way: FIN** When one side (either the client or the server), sends this signal, this indicates that the side who sent this will no longer be sending messages, however they can still receive messages from the other side. Once both sides, sent this signal, the protocol will terminate. In a way, you can think of this being a two sided way to end a connection
- **The rude way: RST** When either side sends the RST signal, this says they they are no longer sending messages, and also they will no longer be receiving messages, When this signal gets send, the protocol immediately ends. Usually this would happen if your computer encountered an error that it couldn't recover from or something like that.

8.6.3 TCP Flags

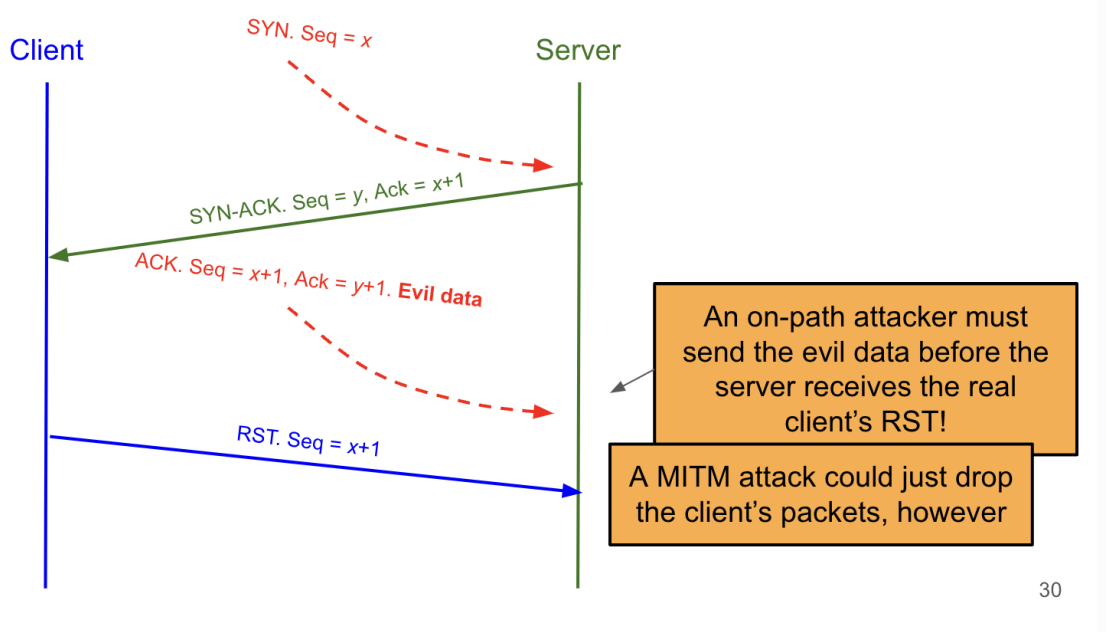
There are 4 main flags in the TCP protocol,

- **ACK**: Acknowledge that message was received
- **SYN**: Indicate that we want to start a new connection
- **FIN**: Indicate that sender will no longer be sending messages, can still receive
- **RST**: Inidcate that the sender will no longer be sending or receiveing messages. Pretty much ends connection immediately.

8.6.4 TCP Attacks

now that we know about TCP, we will talk about the attacks that can be done through this protocol, If you have notice so far, when talking about TCP, no cryptography has been mentioned, so no security, no integrity and no authenticity. Meaning the following attacks are very possible:

- **Data injection**.
  - If we have a man in the middle attacker, they can simply receive the packets, change their data and send them along their way.
  - If they are an on path attacker, it is a little harder. They would need to see the packet trying to be send, replicate its header information, and send it to the recipient, hoping it gets there before the original packet.
  - If they are an off path attacker, they would need to be able to guess a couple things about the header, such as the sequence number, it needs to be in the right range of numbers, because if it was off my too much, it would likely not be recognized as a valid packet. If the headers are guessed correctly, they also need to get the packet there before the original one.
- **RST injection**
  - If an attacker wants to end a connection going on between a client and server, they can spoof a packet that has the RST flag set, which would terminate the whole connection. This is common in denial of service attacks (DoS)
- **TCP Spoofing**: With TCP spoofing, an attacker starts a TCP faking to be a client. Here is a diagram of how it works.



Essentialy the attacker will start a connection faking to be the client, the server will then responds how it usually would, since the client never started a connection, it will try to end it with the RST flag, but before that flag gets sent, the attacker has the opportunity to inject as much malicious data as possible to the server.

- For a MITM attacker, this is very easy, since we can see every message being send and change then in any way we want
- For an on-path attacker, we cannot drop the RST message that the client will send, so we have a race condition and have to inject our malicious data before the server recieves the RST flag

- For an off-path attacker, we deal with the race condition like the on-path attacker, but we also need to guess things like the sequence number and the ack number for the packet to be believable. This is why it is so important to use random generators to starts your sequence numbers, because if they were deterministic, an off-path attacker would have the same capabilities of an on-path attacker since it can deterministly guess the starting sequence.

8.7 User Datagram Protocol (UDP)

UDP is another protocol on the transport layer like TCP. so when deciding on a protocol to use on the transport layer, you really have two options, TCP or UDP, but you cannot choose both. We can think of UDP as TCP but with no reliability. So you for example, if I am sending packets and one of them gets dropped, I do not try and send that packet again, I just move on to sending the next packet. So you might be thinkin why we would ever use this over TCP, since TCP provided many nice properties to ensure we receive all the packets and can have the message be in order for us. Well, the main reason we might prefer UDP and TCP is when occasional errors are okay and we really need speed. Recall in TCP, we need to do a 3-way handshake before we even send our first message. With UDP we just send the messages right away. Also in TCP a packet could keep getting dropped, in which we would keep resending the same packet over and over again. Think about watching a livestream or playing an online video game, you would probably rather have some lag for a couple seconds rather than have your whole thing freeze until it receives the correct packets. These are cases in which we prefer speed over reliability, hence use UDP.

The UDP attacks are almost exactly the same as the TCP attacks, except there is no handshake, so it is even easier for the attacker to spoof request.

9 Transport Layer Security (TLS)

9.0.1 Introduction to TLS

TLS is a protocol right above the transport layer of abstraction. That is, it is built on top of TCP or UDP. In this layer, we will be dealing with all the security issues that we previously showed in TCP.

Remember that when doing security we want to provide confidentiality (no one can see our messages), integrity (no tampering) and authentication (verify sender). We will first go over the TLS protocol and how it provides this

9.0.2 TLS protocol

The TLS protocol is pretty long and definitely has many steps, here is the protocol.

- The client and the server both say Hello to establish the connection. In this initialization, they will exchange random numbers
- The server will next send a certificate, if you recall a certificate is no more than a public key with a signature from a vetted authority. The client will receive this certificate and verify this signature. Note that this does not tell anything about the who is actually sending the certificate. An attacker can very well send the public key with a valid signature.
- Since we cannot guarantee the server with the certificate, we will send a premaster secret (PS), which can be a random number that is encrypted with the public key given from the certificate. The server should be able to decrypt it if the public key sent was actually theirs.
- Next, the random numbers sent from earlier and the PS are all put together and hashed to generate four keys:
  - Client to server sym key: C_B
  - Server to client sym key: C_S
  - Client to server MAC: I_B

  - Server to client MAC: I_S

- Both the client and server will send MACs on the information from the handshake so far, each side will evaluate the MACs to ensure no tampering.
- If all goes well til this point, the client and server can start sending messages , along with their MACS and encrypting them in the process. The diagram for this protocol is pretty long so I would recommend looking at the lecture slides to get a visual understanding.
- **An Extra Note:** To prevent replay attacks that are within the connection, timestamps are put on the messages. So if we got a message with a timestamp that does not make sense (ex: timestamp was an hour ago) we can identify this as a replay attack and handle it. Replay attacks on different connections are not a problem since we generate random numbers every connection.

9.0.3 Forward Secrecy

Recall that something has forward secrecy if an attacker takes note of an exchange, and then in the future learns the private key, but still cannot learn the exchange they noted earlier. The question we want to answer is if TLS has forward secrecy. The answer is it depends on the protocol you use to share the PS. You can either use RSA or Diffie Hellman. Since RSA uses the private key to share the PS, TLS with RSA has no forward secrecy. If you use Diffie-Hellman, the private key is not used at all to share the PS, just values g a and b which are thrown away after each connection (they don't stay around) so learning a and b from one connection will not give you any useful information for another connection.

9.0.4 TLS Efficiency:

Here is the efficiency information for TLS:

- Public-key cryptography has a minor cost in terms of efficency since algorithms like RSA and diffie-helman are exponential in time, but they are only done once
- Sym-key cryptography is pretty much free, since this is just XORs and block cipher operations. Modern computer chips are also specifically designed to make these AES encryption extremely fast.
- The first message will take some time due to the handshake, but after this, it is very efficient

Overall from looking at the little slowdown TLS will make, the security benefits well outweigh it and make it very worth to use.

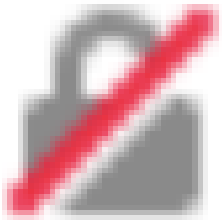
9.0.5 Other properties of TLS

Now we will talk about other security properties we care about and if TLS provides it:

- **End-to-End security**: It does not who is in betweena connection, off-path, on-path, or MITM, everything is secure, meaning no information will be leaked about the messages being send across the connection
- **Anonymity:** Anonymity is the property is if we are talking to a server, we don't want to leak that we are actually talking to the server. When the certificate is sent in the TLS handshake, this usually has the server's name so TLS does not provide anonymity.
- Availability: If a man in the middle attacker is present, they can drop all messages that are send over the connection, so in that way that can stop the TLS connection.

9.0.6 TLS In Practice:

A cool thing to know is the difference between HTTP and HTTPS is HTTPS adds the extra TLS layer onto the networking protocol, while HTTP does not. Many web browsers highly encourage servers to use HTTPS, by implementing discouraging icons such as the one's below that could sway users from going onto insecure websites.



9.1 Domain Name System (DNS)

DNS is used when you query for something on the internet. It is the protocol that gets you where you want to go on the internet. Here is a general structure.

- If a name server does not have the ability to serve your query, they will direct you to someone who might know. It is kind of like saying "hey I don't know, but my friend might, here is their contact."
- The name servers are organized in a tree hiearchy. With the root name server being on the top, then maybe the level below will be all the top level domains like .edu, .com and more. Then below that would be more sub-domains and so on.

9.1.1 Protocol

Here is the general protocol on how it works.

- When we make a query, we might as the root name server if they know how to serve our query
- If the root server does not know, they look at our query, and direct us to another name server that will have a better idea on how to server our query
- We will then ask this name server, if they don't know, they will direct us to another name server.
- This pattern continues until someone can server our query or we error out, possibly if we tried to go to a website that does not exist.

9.1.2 DNS Packet using UDP

DNS will use UDP since we need this to be fast, the downsides UDP usually has such as packets arriving out of order or not being sure if all packets arrived is not an issue here. This is because we can fit all the information we need for DNS in one packet. So we just send one packet, and will get a response if the packet is received, if we don't get a response, we simply send the packet again. Here is what is inside a DNS packet:

- **ID Number:** The ID number serves the purpose of being an identifier for this specific DNS request. When we get a response, the ID number will also be attached. So for example, If I made two different DNS queries, and I get two responses back, the ID number will tell me which response was for which request.
- **Counts:** Keeps counts of the variuous records which we will get into next
- **Records:** The place that holds the actual data of the DNS request. Types of records are questions, anwswers, authority and additional.

9.1.3 DNS Records

DNS records are all name value pairs. We can declare a few different types of name value pairs.

- **Answer type (A):** Maps a domain name to a IPv4 address
- **Name server type (NS):** Maps a zone to a domain. A zone is a marker telling which part of the internet this DNS server is an expert in. This could be .edu or .com

Lets go into more detail about each section:

- **Question Section:** This section will have what is being asked. So for example I can be asking to get to www.berkeley.com. Usually these are A type records with the IPv4 address section being blank since we do not yet know the answer to our the question we are trying to query
- **Answer Section:** This section will usually have the answer to what was asked in the question section.
- **Authority and additional section:** These sections will be used to help guide you where to go if the DNS server you asked cannot serve your request. The authority section will have information along the lines of who might be able to help serve your request and the additional section could have some information on how to get there. Authority sections are usually of the NS type, since they will tell you the domain name of the DNS servers that might know how to handle your request. Additional sections are of the A type since they will map the domain name from the authority section to an IP address.

9.1.4 DNS Caching

The reason DNS is designed this way with records is to make it easily cachable. Being able to cache these records allows the clients to not have to make as many DNS requests, which makes their searches fast. It also eases the workload on the server side, with less DNS requests coming in. It is a win win for everyone.

9.2 DNS Attacks

9.2.1 Cache Poisoning

The main vulnerability that DNS has is what is called cache poisoning. What this means is if an attacker is able to send a DNS packet to an attacker that gets accepted, it will be cached for a decently long time, this could be 1 day or sometimes even up to a week.

The main thing that is tweaked to make a DNS packet malicious is the IP address of the server of the servers suggested to go from DNS. the Authority section which has NS records will tell you what domain name to ask, and the additional section will have the mappings from those domain names, to their IP addresses. The attacker wants to make these IP addresses malicious.

9.2.2 Defense: Baliwick Checking

One defense in DNS is Baliwick checking, in this, we essentially check what mappings the server gave for domain names and check if they are qualified to give them. For example, if we are asking a zone that specialized in .edu domains, and they give us a IP address for a .com domain, Baliwick checking would say they this server is not qualified to give information about .com domains and throw it away.

This attacks is easier for different thread models than others. We will go through the main three we have been talking about. MITM, On-path and off-path:

- **MITM:** This attack is very simple for the MITM threat model. When the honest server sends tha DNS packet to the client, The attacker will can change the true IP address to a malicious one, which will then get cached by the client.

- **On-path:** Similary to other attacks for the on-path attacker thread model, The on-path attacker can see the honest DNS packet fromthe DNS server, this allows them to spoof their own packet with all the correct information such as the ID number and other records, but with malicious changes to IP addresses. At this point the honest packet and the spoofed packet are both sent to the client, and it is a race condition to see which packet will get there first.
- **Off-Path:** Like the on-path attacker, the off-path attacker must spoof their own packet, but this time they will have to guess the contents of the DNS packet. They will need to guess things such as the type of records to have, and the most difficult part being guessing the correct ID number, if the ID number does not match the one sent by the client, the client will reject it.

9.2.3 Brute Force attacking:

With off-path attackers, They are forced to guess the ID number of the DNS query, the ID number gets 16 bits in the DNS packet, so there are a total of 2^{16} possible ID numbers. In any good DNS protocol, this ID number will be randomized so an attacker cannot make an inferred guess on what the next ID number will be. $2^{16} \approx 65000$ is not that many numbers to choose from, and one would think this is easily brute forcible. For a long time this was deemed good enough from the caching principle, Once a client gets a response, their result will get cached, meaning they will not have to ask again until the cache entry expires. So for a long time many people thought Spoofing DNS packets for an off-path attacker were not brute forcible

That was until the **Kaminsky Attack**. In this, we essentially give the client a bunch of fake domain names, that are unique, so the client is forced to make a DNS request for each of them, Say there are n fake domain names, then the attacker gets n chances of spoofing the packet pretty much instantaneously. Here is a picture of the different scenarios with a Kaminsky attack:

	Attacker **correctly** guesses the ID number	Attacker does **not** correctly guess the ID number
Attacker **beats** the race condition against the legitimate NS	The recursive resolver caches a mapping from the legitimate domain name to the attacker's IP	The recursive resolver ignores the response because the ID does not match the request sent earlier
Attacker does **not** beat the race condition against the legitimate NS	The recursive resolver caches something saying "This domain does not exist"	The recursive resolver caches something saying "This domain does not exist"

9.3 DNS Over TLS

Since we just talked about TLS as a way to establish a secure end to end connection, one would think it makes sense to attach TLS to DNS to make it more secure such that no one can be sniffing our packets over our connection without us knowing about it. While TLS does provide security to DNS, there are a few downsides to implementing DNS with this protocol,

- It is slow, We need to do the whole handshake to perform the DNS query
- It provides security for the connection, but does nothing if the person on the other side of the connection is malicious.
- It does nothing to ensure cache entries will not get changed in the client's cache

With these drawbacks, there has been another way to secure DNS which is called DNSSEC:

9.4 DNSSEC:

DNSSEC attempts to address the faults that occured when trying to put TLS on top of DNS. With DNS, we only really need to ensure integrity, since all the things we are sending over the network is public information, no real need to encrypt. Here are the important features about DNSSEC:

- Every DNS server that sends a DNS packet will also send a their signature, which if you recall is when the sender sign the message with their private key and then the user verifies it with the sender's public key.
- The DNS server's public key is also sent, to allow the receiver (us) to verify the signature
- Along with the signature, a certificate will be also be sent to signify that domain that the sender is directing you to is trustworthy. This works like a tree structure so the root of trust is the root DNS server
- DNSSEC does not require any confidentiality, since the data being sent back and forth is public information, this is why TLS is not really necessary

9.4.1 Non-existent domains with DNSSEC

So you might be wondering, how does this work if we try to ask for a domain that does not exist on the internet. Well there a couple of things we can try:

- **Don't authenticate it**, meaning that in the answer section, it will be blank. But this will not work, This is mainly because a MITM attacker could delete this record in the answer section, fooling you to believe that a valid domain name does not exists on the web.
- **We can keep the private key on the name server** and sign records in real time, so if a domain name did not exist, it would be verified with a signature. However, a caviate to this is that signing records in real time is very slow, especially since DNS is a protocol that is used so often. Another reason we should not do this is because if the name server was hacked, they now have access to the private key and can spoof records.

With these limitations, we have some solutions to this:

- **Offline signature generation:** compute all signatures head of time on a system that is not connected to the internet. After all the signatures have been generated, we put them on the name server. So the name server itself does not have the private key to generate the signatures, but just the signatures themselves.
- **Knowing when a domain does not exist:** What is done here is that we put the domains in alphabetical order. When doing it this way, it gives us a finite number of things we need to provide signatures for. So for example, if I had the valid domain names help.cs162.com and killmepls.cs162.com, if I tried to search for ihateos.162.com, we would return the record that said "no domain exists between help.cs162.com and killmepls.cs162.com"

9.4.2 Attack: Domain Enumeration

With what is mentioned above for proving the nonexistence of a domain, An attack that attackers can perform from this is what is called domain enumeration. Through this, an attacker can try a bunch of different domain names and find out all of the valid domain names for a given entity. So for example, if google.com had a secret domain called secrets.google.com, the attacker could perform domain enumeration and find out that this domain exists. It is up to personal opinion whether this is an issue or not, Some argue that this allows the attacker to find out information that they should not be able to. Others say that it is okay to have all of your domain names known. If there is confidential information, they should not be on a domain server, or they should be protected somehow, either by a password or whitelist people that have access.

9.4.3 Domain enumeration Defense: Hashed Denial of Existence

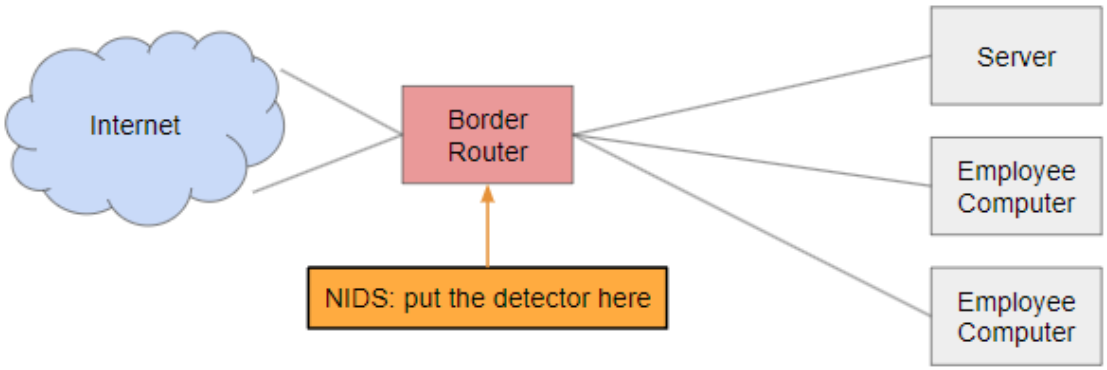
A way to avoid having all of these domain names exposed is hashing all the valid domain names. So it is the same process as before, but instead all of the domain names are hashed so an attacker cannot find out information about the domain names. However, domain enumeration is still possible through brute force, since most domain names are pretty short.

9.4.4 Implementation errors in DNSSEC

The most common error in DNSSEC implementation wise is the client not verifying what is returned from the DNSSEC protocol. Since the recrusive resolver which executes DNSSEC could be malicious, it is vital that the client verifies what they get back to make sure it came from a verified source.

10 Intrusion Detection

One of the security principles that we mentioned in the beginning of the semester is detect if you can't prevent. So this section will be talking about ways and tools we can use to detech attacks. Since there is not a defense for every single attack, many attacks are caught by detection systems. Network Intrusion Detection System (NIDS) A network detection system is a detector that is installed on a your router that connects your machines to the internet, here is a picture:



Here is a little bit about how a NIDS works. Inside the NIDS, there is a table of all the active connections and the state of each connection, within each connection , there can be statistics regarding each connection that you can analyze. And one of the most important things of course is detecting attacks. Here are some of the pros about NIDS:

- **We only need a single detector** to cover a whole LAN since we connect it to the router. So anything that goes through the router to any of the computers on the LAN will have the packets analyzed analyzed and checked for attacks.
- **It is easy to scale**, If we add more computers to the LAN, just add more computing power to the NIDS
- **Easy implementation**, You do not have to anything to the existing systems on the LAN, just putting the NIDS on the router will provide this security with leaving the systems themselves untouched.

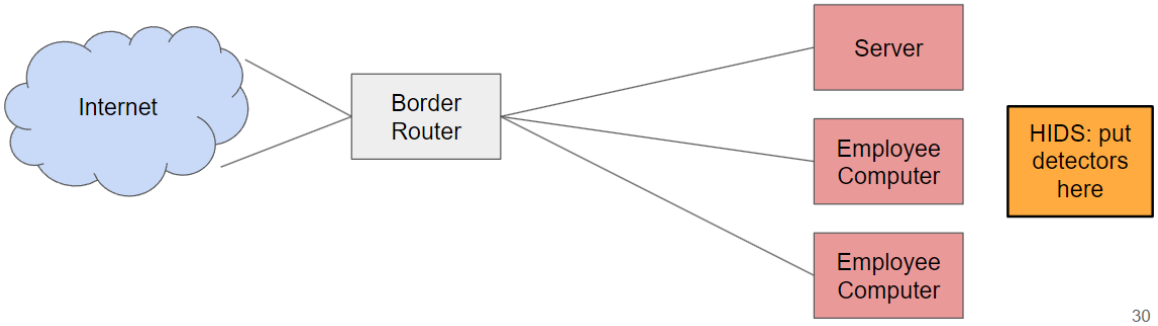
The biggest drawback with NIDS is that the NIDS does not really now how to interpret the packets that are coming through the network. For example, if an attacker tries to do a path traversal attack, the NIDS might see a packet that looks like one, but how does the NIDS know what this path accesses on the computer it is being sent to? it doesn't which makes it hard for the NIDS to interpret if this would be an attack on the system or not. Using this knowledge, an attacker could possibly exploit this by providing malicious inputs that are not detected by the NIDS. A defense to this is to make sure the host and NIDS have all of the information needed to make sure they are using the same interpretations. Another defense is having the NIDS flag everything they find suspicious so they can

be further analyzed later.

Another drawback with NIDS is that, since most communications over the internet uses TLS, the NIDS cannot read any traffic across the conenction, since it is encrypted. So solve this, the NIDS would need access to the private keys, however, that can violate some TLS principles since now the Users will have to share they private key with the NIDS.

10.1 Host based Intrusion detections system

This is another type of detection system that differs from NIDS. The main difference is that we put detection tools on each system instead of putting one on the router. It looks like this:



Some advantages with HIDS are the following:

- Less room for interpretation, since the HIDS is on the end host machine that gets the packets, so they know exactly how to interpret them.
- It works easier over a secure connection like TLS
- More layers of security. Previously if the NIDS got hacked, everything is toast. Now if a HIDS gets hacked, it affects only one machine.

The main drawback with this is that it is definitely more expensive to implement, since we need to install a detector for every machine.

10.2 More detection systems

10.2.1 Logging

Logging is when we keep a running log of the activity on a machine, and then analyze the logs later. An example is each night, we might run a script on the logs and analyze them for attacks. The most prevalent benefit to this is that it is very cheap to implement. You do not need an extra fancy hardware. Another benefit is there is no inconsistency with the interpretations since we are checking the logs on our own machine.

The main downside is that we cannot check the attacks in real time, like we could with the NIDS or HIDS. It is possible for an attacker to go into your logs and erase attacks that happened since you are not checking in real time.

10.3 Detection accuracy

Within detectors there are a couple of terms that are useful to know.

- **False positive** Detector alerts when there is no attack
- **False negative** Detector does not alert when there is an attack.

From these we can derive probabilities such as the false negative rate and the false positive rate.

Detectors are bound to produce both false negative and false positives. You can tweak you detector so it has less false positives or less false negatives. But usually a decrease in one rate will come with an increase in the other rate. So it is up to the user to perform some cost analysis and determine whether having a low false positive rate or a low false negative rate is better. Usually having a detector with a low false negative rate is better, since we want to detect as many attacks as possible.

10.3.1 Combining Detectors

You can combine detectors to have accomplish changing a certain rate. here are two main examples: itemize

Parallel composition: Alert if either detector alerts. This reducese the false negative rate, but increases the false positive rate

Series composition: Alert only when both detectors alert. This reduces the false positive rate but increases the false negative rate.

10.4 Styles of Detection

We will go over four main styles of detection that are used. itemize

Signature Based: In this style, we keep a list of patterns (or signatures) that are common styles of attacks. If we see something come through that is in this list, we alert that there is an attack. This is really good at detecting known attacks, but does nothing in detecting brand new attacks

Specification Based: We make specific rules on what is allowed and flag anything that is not considered allowed behavior. This allows us to detect brand new attacks. The main drawback is that something will need to sit down and make all the specifications on what behavior is allowed and what is not.

Anomaly Detection: Have a model that knows what normal activity looks like, and alert on activity that is an outlier from normal activity. This is able to detect attacks we have not seen before but is not used too much in practice since it is very theoretical.

Behavioral Detection: Looking at what actions are triggered by the specific input. This is great since it has a low false positive rate and can detect unseen attacks. But it will only detect the attacks after they have happened. it might also fail to detect defective attacks. So if a hacker sucks and fails to perform an attack correctly, since no out of the ordinary actions were triggered on our system, we will not alert.

Vulnerability Scanning: Perform a set of attacks on our own system before launching it. This is great since we can see if there are any vulnerabilities in our system. The main downside is that it takes time to formulate all these attacks.

Honeypots: This is a sacrificial tool meant to lure in attackers and identify them as intruders.

10.5 IDS vulnerabilities

Since IDS is a device that are added on to existing systems, they have limited resources, and are vulnerable to Denail of Service attacks, since we can exhause all of the IDS memory

11 Denial of Service

A denail of service (DoS) attack is an attack that makes a service that was previously available to the public no longer available. An example of this is taking down a server like cs162.com (hehe not a bad idea). These are very common and there are a couple ways of executing a DoS attack

- Exploiting a program flaw: If you can cause the program to crash, then you have successfully did a DoS attack. Common attacks for this are buffer overflow and SQL injection

- Use all the resources: Everything has limited resources so if you do something to consume all of it, like spamming a bunch of requests to a website, it does not have any other resources available for their other users and will crash.

11.1 Application Level DoS attacks

Application Level attacks are attacks you can perform on the application itself to cause a DoS. A couple are the following:

- **Resource Consumption:** We can do things such as fill up disk or consume all the RAM of an application to cause it to crash
- **Algorithmic Complexity attack:** If we know a certain service uses an algorithm that runs very slow on certain inputs, we can exploit that. For example if a service uses quicksort, there are certain arrays where quicksort performs very poorly. It can be the difference between something taking a couple of seconds and a couple of years.

11.2 Applicate Level DoS Defenses

Some ways to defens against application level DoS attacks are the following:

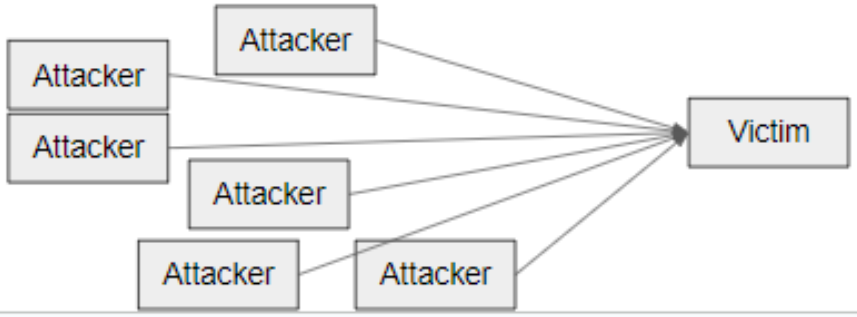
- **First you must identify who is trying to perform a DoS attack** so you can try to remove their access to your service. This requirs some method to authenticate users and identify them
- **Allocate a certain number of resources** for each user. So if a user consumes all of their resources, they are not consuming other user's resources.
- **Proof-of-work:** Make the user do some extra human work to issue a request, an example of this could be a CAPTCHA
- **Allocate more resources overall**, this is obvious but not always possible since this is expensive. There are things called content delivery networks that you can pay for which will allocate a huge amount of resources for you so you do not need to get them all yourself. It is like a pool of resources that many people can use at once and they are dynamically allocated depending on what services need at what time.

11.3 Network DoS attacks

In network DoS attacks, attacks are performed to overwhelm network protocols. This is another common way to overwhelm a system, if we simply are sending more packets than the network can process, many packets will then become corrupted and dropped, causing tons of issues which will probably cause the system to crash.

11.3.1 Distributed Denial of Service (DDoS)

A DDoS attack is something you have probably heard before, it is a type of DoS attack, the main difference is that there are many difference attackers attacking a system at the same time. Making it much more possible to overflow the server. Since in most cases a server will have more processing power than one machine, but if we have hundreds or thousands of machines attacking a server, it is easy to overwhelm it. Here is a picture:



11.3.2 Amplified Denial of Service

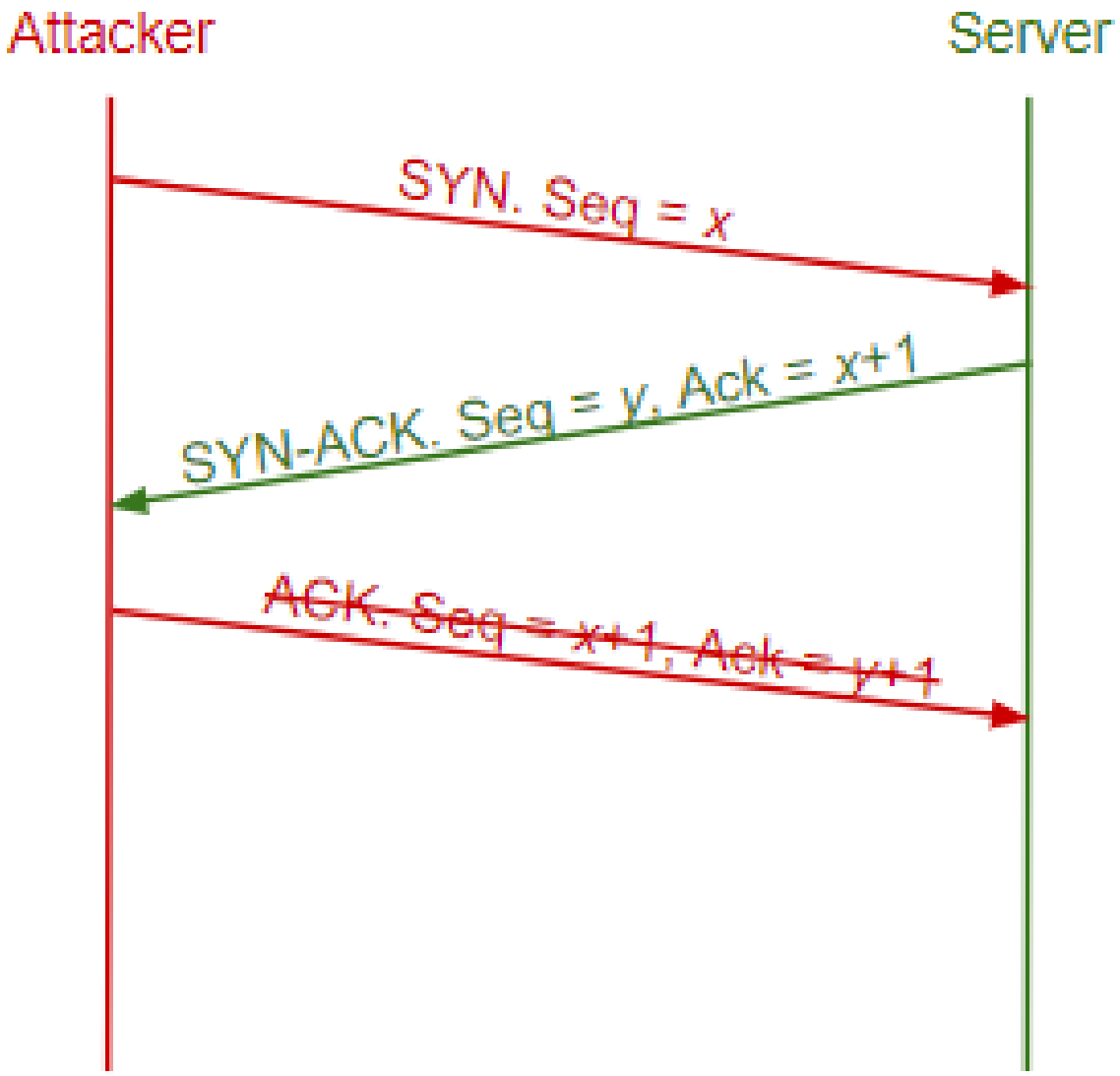
Another mehthod to overflow your target is to use public tools to make your requests consume more resources. A common example is using DNS, since we send one record and DNS will send something that is like 50x bigger, so if we send a DNS request and tell it to respond to a specific user, we can overwhelm this user more effectively.

11.4 Network Level DoS Defenses:

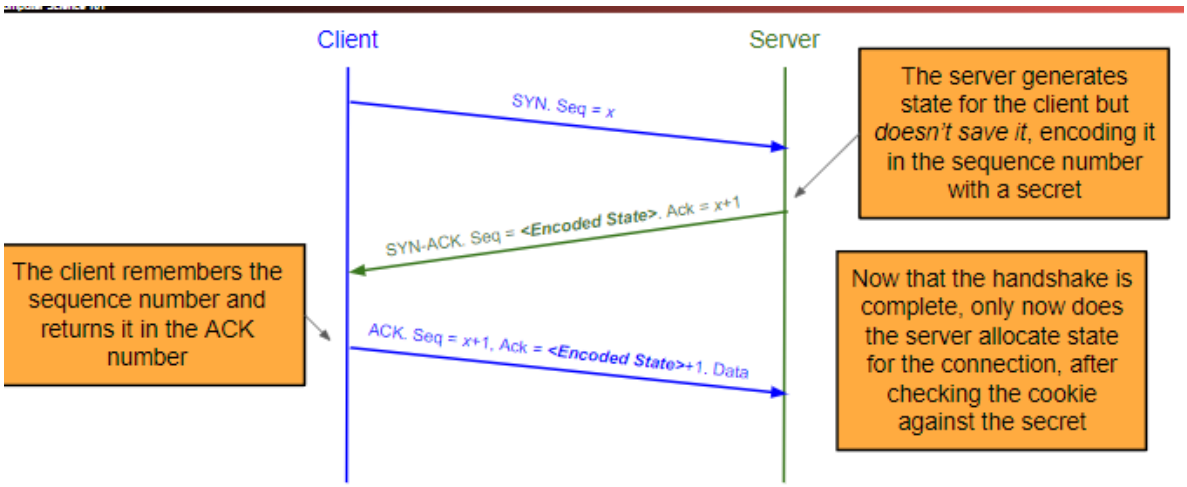
- Filter out packets that are deemed to be part of a DoS attack, we can simply drop them so they never get processed and use the system's resources. The difficult part of this is being able to identify which packets are malicious
- Buy more resources so that you can increase your networking bandwidth, making it harder to become overwhelmed.

11.5 Network DoS Example SYN Cookies

We know in TCP we must first send a request to connect called the SYN request, here we send the sequence number, For many servers when they get this request, they will allocate some memory to handle the connection. But what if we never continue sending messages to the server? Then this connection will stay open, with the server waiting to receive more connections. We can send a bunch of these and overflow the server's memory. Here is a picture:

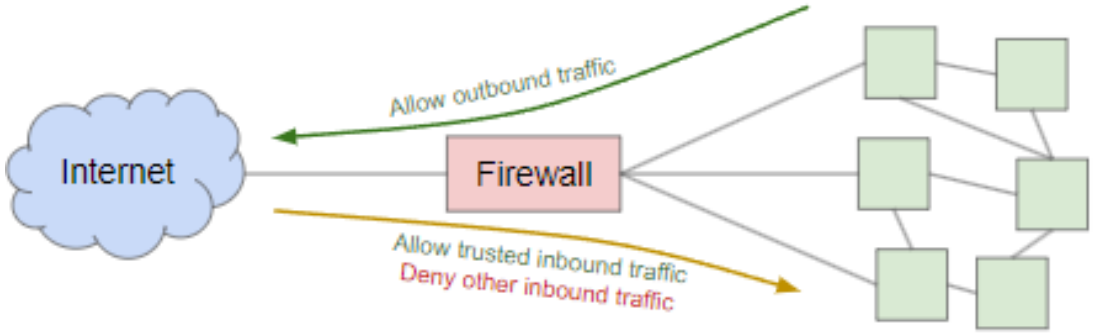


The main way that this is solved is not allocated space to store the state right away, but only after the TCP handshake has been completed. We would send the state in the response, which would force the client to back the state, allowing us to retrieve everything we need to set up the connection. At this point, the handshake has been completed and the server can allocate some memory to handle the connection.



12 Firewalls

Firewalls are a tool that are used to provide protection from a local network to the outside world. Everything that a local network wants to send to the internet will go through the firewall and everything coming into the local network will be going through the firewall as well. Here is a simple picture to illustrate this.



So how do we know what to allow and what to know allow when packets are passing through the firewall, there are a couple of different policies we can use.

- Default allow policy: We allow all traffic except traffic from those specified on a denylist. And as we find out more issues, we can add things to the denylist. With this we are more prone to attacks that can take down our whole system, but it is more flexible
- Default deny policy: this is the opposite of the default allow policy, where we deny all traffic except things specified on allow list. So as needs arise and users need packets that the firewall dropped, they will get added to the allow-list. This is safer but can be pretty annoying since lost of things will get dropped that shouldn't

12.0.1 Packet filtering

There are two main types of packet filters that we can implement:

- Stateless packet filters: memoryless filters, meaning they have no memory of the packets that came before it. A possible packet filter for this is allow all outgoing traffic from the LAN, and only allow inbound traffic that is a response to an outgoing connection, and deny everything else.
- Stateful packet filters: These filters will keep track of current inbound and outbound connections, where we can specify what actions are allowed and what actions are denied from this. For example we can allow a tcp connection that ame from tcp connection 4.5.5.4:* -¿ 3.1.1.2:80

12.0.2 Subverting Packet Filters

There are a couple strategies we can use to get around packet filters. Some strategies are the following:

- Send packets out of order. Some packet filters may check for certain patterns in the sequence of packets, if we send them out of order, we can get around this filter, since they will be reconstructed in order on the other side through TCP
- Exploiting a packet's time to live, we can send garbage along with our attack, the garbage packets will make it to the firewall, which will conceal our attack, however they will not make it to the final destination, since we will set the packets time to live (TTL) to expire before it make it to the final destination.

To defend these attacks there are different types of firewalls:

- Proxy firewall: Many of the previous problems mentioned was because we were operating with the packet mindset, so an idea is to have it at the connection mindset. We can put a firewall in the middle of a TCP connection with a client and a server, to the firewall is a MITM. everything will go through the firewall, which will allow to to verify everything and getting the nice TCP properties.
- Application proxy firewalls: There are certain protocols at the application layer that we can use proxys for like HTTP.

12.1 Getting around a Firewall with a VPN

We can use a virtual private network (VPN) to get around a firewall, essentially we can connect to a local network inside a firewall by going through a VPN server. Since we can make request to this VPN server and the firewall allows it, we will have to provide some authenticaion like a username and password. But this allows us to get do things that would usually get denied by the firewall.

13 Malware

So far when we have mentioned a malicious attacker getting access to your computer, what do they actually do when they have access? Well, usually they will put some kind of malware on your computer. Malware is defined as malicious software.

There is a notion that malware will self-replicate its own code, so it won't only affect the computer the malware is on, but it will also go to affect other computers. There are two main types of malware:

- Virus: Malware that requires user action to activate. You can think of this like the user has to click on a file or run a certain application
- Worm: Malware that does not require the user to do anything to activate it, You can think of the attacker planting something in the OS or something that always runs in the background and does not require any user action to run.

13.1 Viruses

13.1.1 Spreading the virus

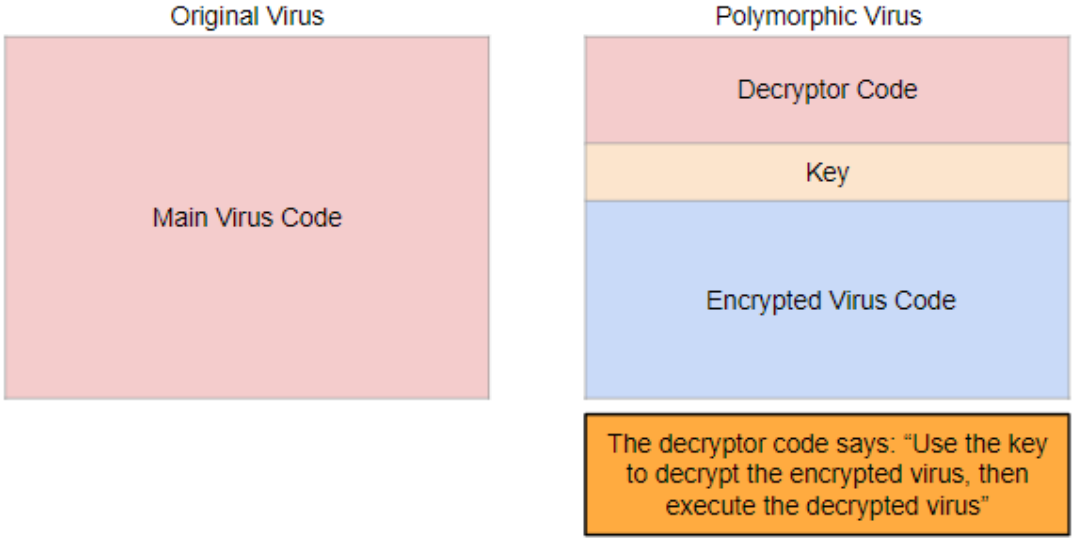
When the virus is executed, it will immediately look for opportunities to spread to other systems. Some common things it will do it send emails to others with this code attacked or copy their code onto any flash drives that might possible be used to plug into other computers.

13.1.2 Detecting the virus

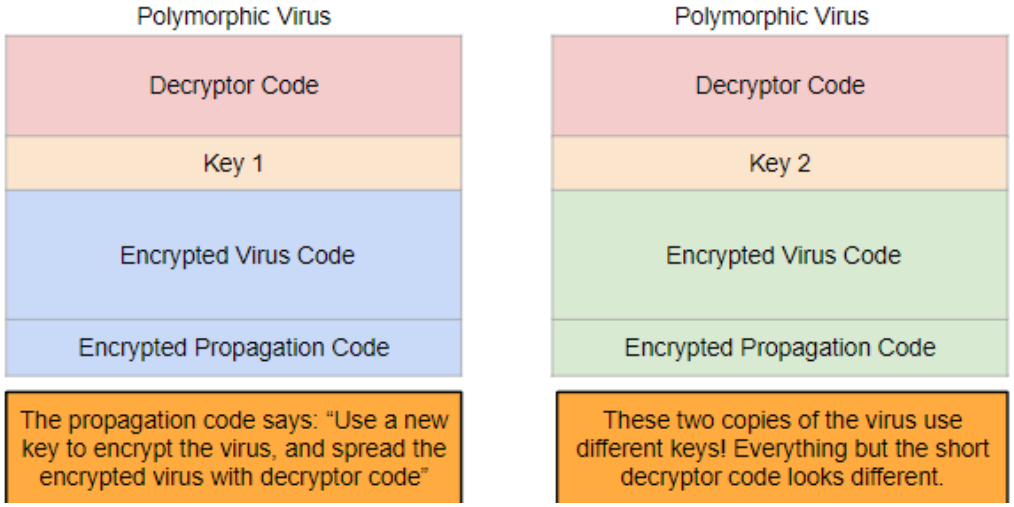
The most common way to detect viruses that Anti-virus software uses is signature based detection. This is when we compile a huge list of malware code that has been seen in the past and if we do a scan and see any of these code patterns anywhere on the system, we can detect a virus. This of course is vulnerable to new attacks that have never been used before since they will not be on anyones list of seen malware.

13.1.3 Evading detection

Some other evasion strategies that attacers use to avoid anti-virus detection is to change the appearance of known attacks to make it harder to detect. One affective way to do this is for an attacker to encrypt their program and send the malware, decryption and key used for decryption all in one. This way the malware looks different every time and once the user activates it, it will be decrypted using the key sent with the virus and then executed. Here the attacker does not care about keeping the key secret, since they do not really care about keeping the malware secret from anyone, but it is used to avoid detection software. Here is a picture of what this might look like.



A even more creative thing an attacker can do is the malware will then generate a new key and enrcypt its own code with that new key so it looks even different from the previously encrypted version, and then send that off to propogate. It would look like this.



Detecting this can be much more difficult since the only things we can really go off of is the decryptor code. This would result in many more false positives since decryption is used in many non-malicious programs.

Another thing that can be done is have a program that changes the semantics of a malicious program. Essentially, it would change aroung parts of your program, but it would overall have the same functionality.

13.1.4 Defenses for Evasion

Probably one of the most effective ways to detect malware given these evasion strategies is to evaluate the behavior of these malicious programs instead of just looking at them. Of course, there is one main caveat to this, being we have to run the program to see its behavior. In doing this, we can already lose the battle. So it is important that we execute potential malware in a sandboxed environment where it cannot spread.

Another thing we can do is flag unfamiliar code that we have never seen before. Again this will lead to more false positives but it will let us be more careful.

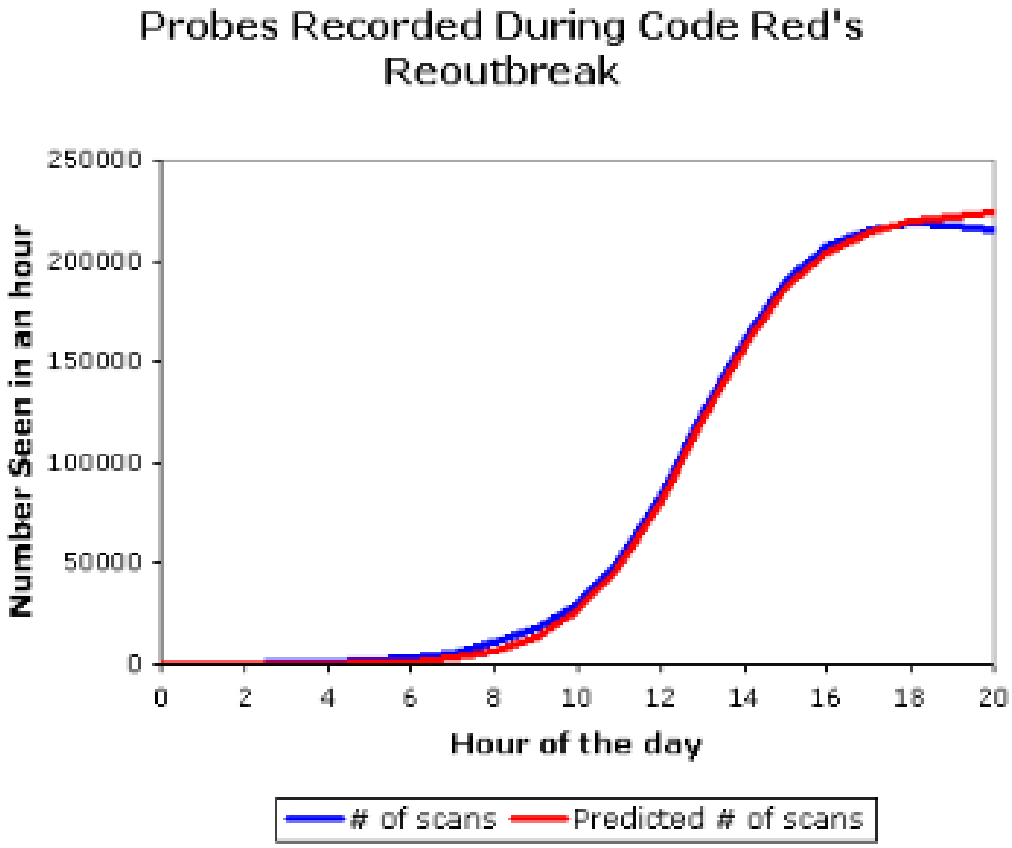
13.2 Worms

The main difference from worms and viruses is worms do not need user action to run. This can be done in a couple ways.

- Use a buffer overflow to inject its code
- Use SQL injection
- Exploit XSS vulnerability

The way worms spread really depends on the worm, some don't care who they infect, so they will just try and spread to as many people as possible. One strategy is just to select random IP addresses and send the worm to them. Some have a hit-list of IP addresses, maybe if they are trying to target everyone inside an organization.

Worms can have exponential growth theoretically if everyone they infect goes on to infect others, but practically, there will get to a point where they cannot really infect any more people because either something breaks and they are not able to spread anymore (like the internet breaking) or there is no one else to infect. Here is picture visualizing this, we see if amount of people getting infected starts to flatten out.



3

13.3 Recovering from a virus

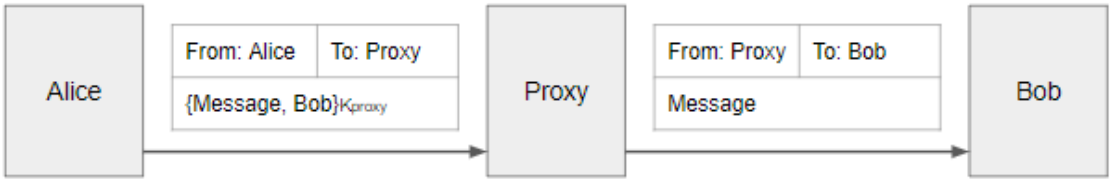
There is software that will attempt to recover your system if it has experienced a virus, however many times it is very very difficult since you don't know where this virus could have spread. It could have compromised your whole disk, or even worse compromise your operating system. At this point you might has well just throw away your system and get a new one.

13.4 Anonymity and Tor

One thing that has been mentioned a couple of times is the concept of Anonymity, or in simple terms, concealing your identity on the internet. It turns out this this is actually pretty difficult, since most protocols for connection like TCP and UDP involve two people with known IP addresses communicating, and packets will contain the source and destination IP address.

13.4.1 Proxies and VPNs

One thought is the use a proxy to communicate with another person on the internet here is a diagram



Here all the information will go through the proxy so Alice can send a message to the proxy instead of Bob himself. So when bob gets the messgage, he would not know it came from Alice, just that it came from the proxy.

There comes an issue with this, what if the proxy is malicious? Since the proxy is pretty much a MITM, they will now know who you are and who you can to send messages to. It becomes the same issues that we have talked about many times for MITM attackers.

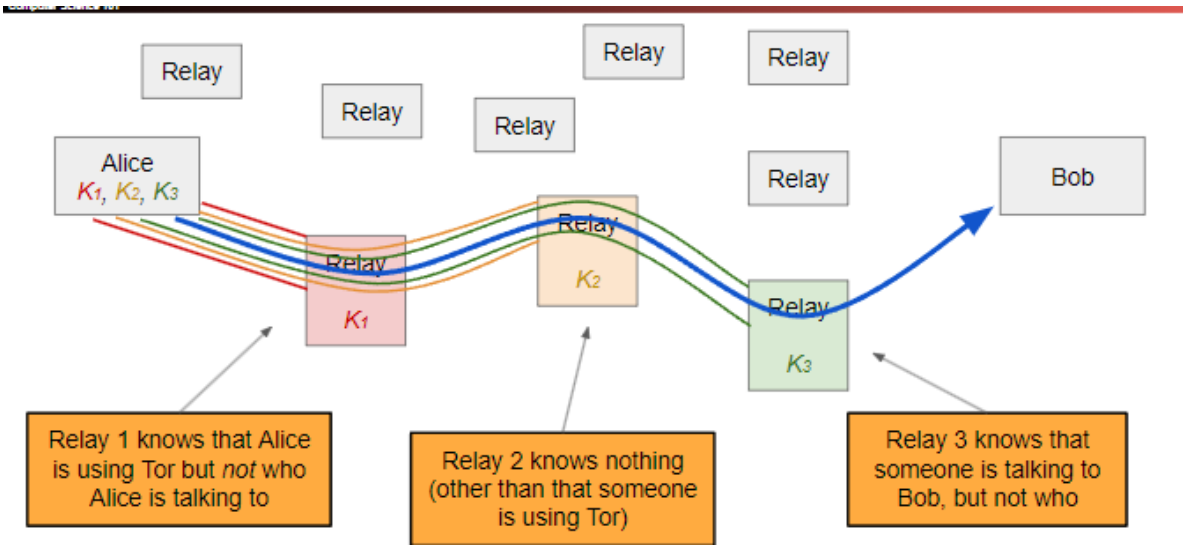
VPNs can also be used as a middle man, one key differnce is that VPNs work at the network layer so Alice is not sendthing the message to the proxy, but instead sending from the VPN directly, but since it is a VPN, Alice's identity is concealed.

13.4.2 Drawbacks

Some drawbacks with anonymity, specifically with what is mentioned above is that it is slower to send messages, since there it more hops across the network. It can be more expensive since many VPNs cost many. Lastly, there is a possibility of the proxy being malicious and it has MITM power.

13.5 Tor

A service that exists on the internet that tries to provide anonymity is Tor. Tor is essentially a network of a ton of proxies which are called relays that will be used to send messages. So instead of just alice sending a message to a proxy, which then sends it to bob, the proxy might send it to another proxy, which it might send to another proxy, and eventually get to bob. This provides more anononymity since most of the relays don't know who the final recipient of the message is, they only know which relay to send the message to. Here is a picture of sending a message through Tor



Here relay with K_3 is something called the exit node. The exit node is able to see who the final recipient is and what the final message says. So again there is potential for the exit node to cause problems since they would be a MITM and could be malicious. So it is important we use something like TLS on top of our connection protocol so they cannot see or tamper with traffic.

In tor there is strength in numbers, the more relays, usually the better, since the relays can communicate with eachother, they can learn information about the message and the sender and recipient if they are all willing to share information. However, all that is needed to stop this is have one honest relay who will not snitch (1 rule, never snitch).

There is a notion of a guard node in Tor, and these are nodes that have existed for a long time in the Tor system, and these will more likely be used as exit nodes since they are more trustworthy.

Tor can be even used to conceal a server's identity. This is a little more complicated to do, but this is the basics of how the dark web exists. It is able to hide the server's identity.

It is important to note that Tor really only works if many people are using Tor, since Tor does not hide the fact that you are using Tor. So if you are the only one using Tor on a network, people can still find out who you are.

Tor can be used for many illegal surfaces on the dark web and in practice, Tor is actually used for these things.

14 C Format Specifiers for exploits

- %c - character
- %u - unsigned integer
- % s - string
- %d - decimal
- %n - stored number of bytes printed so far into argument provided
- %hn - stores number of bytes printed to far into argument provided up to 2 bytes