Google IT Support Specialization

Course 5: IT security

03 aaa security not roadside assistance

Table des matières

[1 Understanding IT security threat 2](#_Toc14376424)

[2 Pelcgbybtl cryptology 2](#_Toc14376425)

[3 AAA security not roadside assistance 2](#_Toc14376426)

[3.1 Authentication 2](#_Toc14376427)

[3.1.1 Authentication best practices 2](#_Toc14376428)

[3.1.2 Multifactor authentication 3](#_Toc14376429)

[3.1.3 Certificates 5](#_Toc14376430)

[3.1.4 Ldap 6](#_Toc14376431)

[3.1.5 Radius 6](#_Toc14376432)

[3.1.6 Kerberos 7](#_Toc14376433)

[3.1.7 Tacacs 8](#_Toc14376434)

[3.1.8 Single sign on 8](#_Toc14376435)

[3.1.9 Rob path to it 9](#_Toc14376436)

[3.2 Authorization 9](#_Toc14376437)

[3.2.1 Authorization and access control methods 9](#_Toc14376438)

[3.2.2 Access control 10](#_Toc14376439)

[3.2.3 Access control list 11](#_Toc14376440)

[3.3 Accounting 11](#_Toc14376441)

[3.3.1 Tracking usage and access 11](#_Toc14376442)

[3.3.2 Rob important skills in security 12](#_Toc14376443)

# Understanding IT security threat

# Pelcgbybtl cryptology

# AAA security not roadside assistance

## Authentication

### Authentication best practices

In this module, we'll cover the three A's of security, which are authentication, authorization, and accounting. We'll cover exactly what they are, how they relate to each other, and their common implementations and protocols. Let's kick things off with authentication, and by extension identification. You should be familiar with authentication in the form of username and password problems when accessing things like your email. So let's take that as an example to show the differences between identification and authentication. Identification is the idea of describing an entity uniquely. For example, your email address is your identity when logging into your email. But how do you go about proving you are who you claim to be? That's the process we call authentication. When accessing your email, you're claiming to be your email address, and you'd supply a password associated with the identity to prove it's you, or at least you know the password associated with the e mail account. Pretty straightforward, right? This is distinctly different from authorization, which pertains to the resources and identity has access to. These two concepts are usually distinguished from each other in the security world, with the terms authn for authentication and authz for authorization. In our e mail account login example, by authenticating using your email address and password, your identity is authorized to access your email inbox, but you're not authorized to access anyone else's inbox. We really don't want anyone else getting access to our inbox, right? So what can we do to ensure that only we are able to identify and authenticate as our e mail account? We could start by ensuring that we're using a strong password. But what exactly constitutes a strong password? Well, what do you think of the password ponies? Would you categorize that as a strong password? I hope not. That password is super short, and only six characters, and all of those characters are lowercase letters. This is a short and simple password, but that could be easily broken through brute force or dictionary based attacks. Ponies would almost definitely be in a dictionary file, and six characters doesn't provide a large pool of possibilities for an attacker to try. We can ensure a password is strong by making it longer and more complex, adding numbers, uppercase letters, and special characters like punctuation. What do you think of the strength of this password? That seems way more secure, doesn't it? It adds complexity, which increases the pool of possible passwords, and is longer at 10 characters. But which of these two passwords do you think you would be able to remember tomorrow? Probably not the strong one, right? This highlights a super important concept in security. There's often a trade off between security and usability. With our password example, the more usable password that's easy to memorize is less secure, while the more secure password is much more challenging to remember. This concept applies to many other security subjects, not just passwords. You can think of security as risk mitigation, and when it comes to risk mitigation, it's impossible to completely eliminate the risk. The best you can do is understand the risks your systems face, take measures to reduce those risks, and monitor them. Think about it like this, the most secure computer system is one that's disconnected from everything, including networking, and even power and is locked in a concrete bunker hundreds of feet underground that no one has access to. While this is incredibly secure machine, almost impossible to compromise, it's basically useless since it's powered off and no one can access it. This is an extreme example of the security versus usability trade off, but you get the point. Coming back to our password example, we obviously need to find some sort of happy medium, but where we have a reasonably secure password that's also somewhat easy to memorize. How about something like this. We start with a short phrase, Ilikeponies, then replace some letters with numbers that resemble the letters to help with memorization. We also swapped the S with the Z since they're similar sounding and tacked on some numbers as a suffix. At first glance, this seems like a very complex password and will be hard to memorize, but it's easier than a random password example. Problem solved, right? Well, you should actually be wary of this number substitution process, since it's well known by attackers and password cracking tools. As an I.T. support specialist, ensuring that your organization uses strong passwords and practices, good password hygiene are super important. They're literally the keys to the kingdom. So what should we do? Incorporating good password policies into an organization is key to ensuring that employees are securing their accounts with strong passwords. A good password policy system would enforce length requirements, character complexity, and check for the presence of dictionary words, which would undermine the strength of passwords. Passwords should never be written down or recorded in plain text, reused across accounts or be shared with anyone. Password reuse is a risk because in the event the password for one account is compromised, other accounts using the same password would also be at risk. Sharing passwords should also be a no go, since this undermines the identity of an account because someone else now has the ability to log in as that user. Along with requiring the use of strong passwords, a password rotation policy is also recommended, since it safeguards against potential undetected compromised passwords. But it's important that a password rotation period isn't too short. Why? The inconvenience of having to change passwords so often may actually encourage poor security behavior by users. So let's say you required your organization to create highly complex passwords and to change them every three months. It's very likely that a significant percentage of users would write down their passwords on post it notes or on their phones. A big no no. Despite the policy being designed to increase security, it actually has the opposite effect because of the inconvenience it causes your users.

### Multifactor authentication

In the last video, we learned about basic authentication in the form of username, password, sometimes referred to as single factor authentication. But there are other more complex and secure authentication mechanisms. Keep in mind the security versus usability tradeoff, as we work through the different types of multifactor authentication. Multifactor authentication is a system where users are authenticated by presenting multiple pieces of information or objects. The many factors that comprise a multifactor authentication system can be categorized into three types. Something you know, something you have, and something you are. Ideally, a multifactor system will incorporate at least two of these factors. Something you know would be something like a password, or a pin for your bank or ATM card. Something you have would be a physical token, like your ATM or bank card. Something you are would be a piece of biometric data, like a fingerprint or iris scan. The premise behind multifactor authentication is that an attacker would find it much more difficult to steal or clone multiple factors of authentication, assuming different types are used. If multiple passwords are used, security isn't enhanced by that much. This is because passwords, however many, are still susceptible to phishing or keylogging attacks. By using a password in conjunction with a security token is a game changer. Even if the password is compromised by a phishing attack, the attacker would also need to steal or clone the physical token to be able to access the account. And that's much less likely to happen. We won't cover passwords again here since we talked about them in detail in the last section. But here's the quick rundown. Physical tokens can take a few different forms. Common ones include a USB device with a secret token on it, a standalone device which generates a token, or even a simple key used with a traditional lock. A physical token that's commonly used generates a short lived token. Typically a number that's entered along with a username and password. This number is commonly called a One Time Password or OTP since it's short lived and constantly changing value. An example of this is the RSA SecurID token. It's a small, battery powered device with an LCD display, that shows a One Time Password that's rotated periodically. This is a time based token sometimes called a TOTP, and operates by having a secret seed or randomly generated value on the token that's registered with the authentication server. The seed value is used in conjunction with the current time to generate a One Time Password. Now, as long as the user has possession of their token, or can view the display of the token, they are able to log in. I should also call out that the scheme requires the time between the authenticator token, and the authentication server to be relatively synchronized. This is usually achieved by using the Network Time Protocol or NTP. An attacker would need to either steal the physical token or clone the token if they're able to steal the secret seed value. Since a time based token is synchronized with the server using time, which is not a secret, that would be sufficient for an attacker to clone a token. There are also counter based tokens, which use a secret seed value along with the secret counter value that's incremented every time a one time password is generated on the device. The value is then incremented on the server upon successful authentication. This is more secure than the time based tokens for two reasons. First, the attacker would need to recover the seed value and the counter value. Second, the counter value is also incrementing when it's being used. So, a cloned token would only be useful for a short period of time before the counter value changes too much and the clone token becomes un synchronized from the real token and the server. These token generators can either be physical, dedicated devices, or they can be an app installed on a smartphone that performs the same functionality. Another very common method for handling multifactor today, is that the delivery of one time password tokens using SMS. But this has been subject to some criticism, because of the observed attacks through this channel. The problem with relying on SMS to transmit an additional authentication factor is that you're dependent on the security processes of the mobile carrier. SMS isn't encrypted, nor is it private. And it's possible for SMS to be intercepted by a well funded attacker. Even worse, there have been accounts of SMS based multifactor codes being stolen by calling the mobile provider. The attacker impersonates the owner of the line of service to redirect phone calls and SMS to a phone the attacker controls. If the attacker has already compromised the password and can get SMS redirected to them, they now get full access to the account. Of course, there's a convenience tradeoff when you use a physical token. You have to carry around another device in order to authenticate. If the device is lost or damaged, the user won't be able to authenticate until the device is replaced. This also requires support overhead, since devices will fail, be lost, run off batteries, and get out of sync with the server. Using an app on a smartphone addresses some of these issues, but still, require some additional support and inconvenience. When prompted to log in, the user must retrieve a device or phone from their pocket and manually transcribe the numbers into the authentication page. These generated one time passwords are also susceptible to man in the middle style phishing attacks. A user can be tricked into going to a fake authentication page by sending a phishing email. Something on the lines of, "your account has been compromised, please log in and change your password immediately." When the victim enters their credentials in the fake page, including the one time password, the attacker has all the information needed to take over the account. The other category of multifactor authentication is biometrics, which has gained in popularity in recent years, especially in mobile devices. Biometric authentication is the process of using unique physiological characteristics of an individual to identify them. By confirming the biometric signature, the individual is authenticated. A very common use of this in mobile devices is fingerprint scanners to unlock phones. This works by registering your fingerprints first, using an optical sensor that captures images of the unique pattern of your fingerprint. Much like how passwords should never be stored in plain text, biometric data used for authentication, so, it also never be stored directly. This is even more important for handling biometric data. Unlike passwords, biometrics are an inherent part of who someone is. So, there are privacy implications to theft or leaks of biometric data. Biometric characteristics can also be super difficult to change in the event that they are compromised unlike passwords. So, instead of storing the fingerprint data directly, the data is run through a hashing algorithm and the resulting unique hash is stored. One advantage of biometric authentication over knowledge or token based systems, is that it's more reliable to identify an individual for authentication, since biometric features aren't usually shareable. For example, you can't give your friend your fingerprints so that they can log in as you. Well, you'd hope not anyway. But as schools start to introduce fingerprint based attendance recording systems, students are finding ways to trick the system. They're creating fake fingerprints using things like glue, allowing friends to marking each other as present if they're late or if they skip school. This is harder to achieve than sharing a password, but it's sort of ingenious of these kids to think up. They really go the extra mile to skip school these days. Not that I'm condoning this behavior, but you can read more about it just after this video. Other biometric systems use features like iris scans, facial recognition, gate detection and even voice. Microsoft developed the biometric authentication system for Windows 10, called Windows Hello, which supports fingerprint identification, iris identification and facial recognition. It uses two cameras, one for color and one for infrared, which allows for depth detection. This way, it's not possible to trick the system using a printout of an authorized user's face. An evolution of physical tokens is the U2F or Universal Second Factor. It's a standard developed jointly by Google, Yubico and NXP Semiconductors. The finalized standard for U2F are being hosted by the FIDO alliance. U2F incorporates a challenge response mechanism, along with public key cryptography to implement a more secure and more convenient second factor authentication solution. U2F tokens are referred to as security keys and are available from a range of manufacturers. Support for U2F authentication is built into the Chrome browser and the Opera browser, with native Firefox support coming soon. Security keys are essentially small embedded cryptoprocessors, that have secure storage of asymmetric keys and additional slots to run embedded code. Let's do a quick rundown on how exactly security keys work, and how their improvement over an OTP solution. The first step is registration, since the security key must be registered with a site or service. At registration time, the security key generates a private public key pair unique to that site, and submits the public key to the site for registration. It also binds the identity of the site with the key pair. The reason for unique key pairs for each site is for privacy reasons. If a site is compromised, this prevents cross referencing registered public keys, and discovering commonalities between sites based on registration data. Once registered with the site, the next time you're prompted to authenticate, you'll be prompted for your username and password as usual. But afterwards, you'll be prompted to tap your security key. When you physically tap the security key, it's a small check for user presence to ensure malware cant authenticate on your behalf, without your knowledge. This tap will unlock the private keys stored in the security key, which is used to authenticate. The authentication happens as a challenge response process, which protects against replay attacks. This is because the authentication session can't be used again later by an eavesdropper, because the challenge and resulting response will be different with every authentication session. What happens is the site generates a challenge, essentially, some randomized data and sends this to the client that's attempting to authenticate. The client will then select the private key matching the site, and use this key to sign the challenge and send the signed data back. The site can now verify the signature using the public key that was registered earlier. If the signature checks out, the user is authenticated. From a security perspective, this is a much more secure design than OTPs. This is because, the authentication flow is protected from phishing attacks, given the interactive nature of the process. While U2F doesn't directly protect against man in the middle attacks, the authentication should take place over a secure TLS connection, which would provide protection from this type of attack. Security keys are also resistant to cloning or forgery, because they have unique, embedded secrets on them and are protected from tampering. From the convenience perspective, this is a much nicer authentication flow compared to OTPs since the user doesn't have to manually transcribe a string of numbers into the authentication dialog. All they have to do is tap their security key. Nice and easy. As an IT support specialist, you may come across multifactor authentication setups, that you'll be responsible for supporting. You might even be tasked with helping to implement one. So, it's important to understand how they provide enhanced account protection, along with the options that are available.

### Certificates

In the last section, we mentioned client certificates as a form of authentication. As we learned earlier, certificates are public keys that are signed by a certificate authority or CA as a sign of trust. We covered TLS server certificates, but there can also be client certificates. These operate very similarly to server certificates but are presented by clients and allow servers to authenticate and verify clients. As an IT support specialist, it's important for you to understand client certificates and certificate based authentication, since you might encounter these in the course of your career. It's not uncommon for VPN systems or enterprise Wi Fi setups to use client certificates for authentication. So understanding how they work will help you troubleshoot issues. In order to issue client certificates, an organization must set up and maintain CA infrastructure to issue and sign certificates. Part of certificate authentication also involves the client authenticating the server, giving us mutual authentication. This is a positive since the client can verify that it's talking to the real authentication server and not an impersonator. In this case, all clients that are using certificate authentication would also need to have the CA certificate in their certificate trust store. This establishes trust with the CA and allows the client to verify it's talking to the real server when trying to authenticate. Certificate authentication is like presenting identification at the airport. You show your ID or your certificate to prove who you are. The ID is checked to see if it was issued by an authority that is trusted by the verifier. Was it issued by a government entity or is it a novelty license from a gift shop? Obviously, one of those IDs would be accepted at the airport, similar to a certificate being signed by a trusted CA. When you're at the airport, the expiration date on your ID will also be checked to ensure it's still valid. The same thing applies to certificate authentication, although the certificates have two dates that need to be verified. Not valid before, and not valid after. Not valid before is checking to see if the certificate is valid yet since it's possible to have certificates issued for future use. Not valid after is a straightforward expiration date, after which the certificate is no longer valid. Airport authorities also have a list of specific IDs that are flagged. If your ID is on that list, then you'll be rejected for air travel. Similarly, the certificate will be checked against a revocation list or a CRL. This is a signed list published by the CA which defines certificates that have been explicitly revoked. One last step that's performed as part of the authentication server verification process is to prove possession of the corresponding private key, since the certificate is a signed public key. If we don't prove possession, there's nothing stopping an attacker from copying the certificate, since it's not considered secret, and pretending to be the owner. To avoid this, possession of the private key is verified through a challenge response mechanism. This is where the server requests a randomized bit of data to be signed using the private key corresponding to the public key presented for authentication. This is similar to how the airport checks the photo on your ID to make sure you look like the person in the photo and aren't impersonating them.

### Ldap

LDAP, or Lightweight Directory Access Protocol, is an open industry standard protocol for accessing and maintaining directory services. When we say directory services, we're referring to something similar to a phone or email directory. It's most commonly used as a backend for authentication of accounts. The LDAP specification describes the data structure of the directory itself. And defines functions for interacting with the service, like performing look ups and modifying data. You can think of a directory like a database, but with more details or attributes, describing entities within the database. The structure of an LDAP directory is a sort of tree layout and is optimized for retrieval of data more so than writing. Think of it as being similar to a phone book being used for looking up data far more often than making modifications to that data. Directories can be hosted across lots of different LDAP servers to facilitate more rapid look ups, and are kept in sync through replication of the directory. So what kind of data gets stored in directory entry, exactly? Similar to an address book, an entry for a particular user will contain information pertaining to that user account, like their first and last name, phone number, desk location, email address, login shell, and other such data. Along with object attributes the location of an entry within the overall data structure will represent information pertaining to the objects as relationships between objects. Because LDAP uses a tree structure called a Data Information Tree, objects will have one parent and can have one or more children that belong to the parent object. You can also think about this like a file system with a root file system in folders under that. The folder an object belongs to will provide information about that object because of its relationship to the parent object. In LDAP language, we call these folders Organizational Units, or OUs. They let us group related objects into units like people or groups to distinguish between individual user accounts and groups that accounts can belong to. This tree structure also allows for inheritance and nesting of objects, where attributes or properties of a parent object can be inherited by children further down the tree. Now, since it is possible for entries in the directory to share attributes, there must be a unique identifier for each entry. We call this, Distinguished Name, or DN. Coming back to our file system analogy, you can think of a DN as a full path to a file as opposed to a file name. This is because you can have multiple files with the same file name across a file system. But the fully qualified path to the file would describe one unique file. Some of the more common operations that can be called by a client to interact with an LDAP server are bind, which is how clients authenticate to the server. StartTLS, which permits a client to communicate using LDAP v3 over TLS. Search, for performing look ups and retrieval of records. Add/delete/modify which are various operations to write data to the directory, and Unbind, which closes the connection to the LDAP server. Now, there are many implementations of LDAP servers, like Active Directory from Microsoft and OpenLDAP for open source implementations.

### Radius

RADIUS or Remote Authentication Dial In User Service, is a protocol that provides AAA services for users on a network. It's a very common protocol used to manage access to internal networks, WiFi networks, email services and VPN services. Originally designed to transport authentication information for remote dial up users, It's evolved to carry a wide variety of standard authentication protocols like EAP or Extensible Authentication Protocol. While it's unlikely that you'd be responsible for configuring RADIUS server as an IT support specialist, you might be supporting clients that authenticate against a RADIUS back end server. In those cases, it's good to understand what role the RADIUS server plays in this authentication scenario so you're better prepared to troubleshoot issues that may come up. Clients who want to authenticate to a RADIUS server don't directly interact with it. Instead, when a client wants to access a resource that's protected, the client will present authentication credentials to a NAS or Network Access Server which will relay the credentials to the RADIUS server. The RADIUS server will then verify the credentials using a configured authentication scheme. RADIUS servers can verify user authentication information stored in a flat file or can plug into external sources like SQL databases, LDAP, Kerberos or Active Directory. Once the RADIUS server has evaluated the user authentication request, it replies with one of three messages access reject, access challenge or access accept.

### Kerberos

Kerberos is a network authentication protocol that uses tickets to allow entities to prove their identity over potentially insecure channels to provide mutual authentication. It also uses symmetric encryption to protect protocol messages from eavesdropping and replay attacks. The name Kerberos is taken from the Greek mythical character of the same name. A three headed guard dog protecting the gates to Hades, the underworld. Seems like an appropriate choice for an authentication protocol, don't you think? Kerberos was originally developed at the Massachusetts Institute of Technology in the US, and was published in the 1980s as version four. Years later, in 1993, version 5 was published. Today Kerberos supports AES encryption, and implements checksums to ensure data integrity and confidentiality. When joined to a Windows domain, Windows 2000 and newer versions will use Kerberos as the default authentication protocol. Microsoft also implemented their own Kerberos service with some modifications to the open protocol like the addition of the RC 4 Stream Cipher. We mentioned tickets earlier which is a sort of token that proves your identity. They can be used for authenticating to services protected using Kerberos or in other words or within the Kerberos realm. The authentication tickets let users authenticate to services without requiring username and password authentication for every service individually. A ticket will expire after some time, but it has provisions for automatic transparent renewal of the ticket. Let's run down the details of how the Kerberos protocol operates. First, a user that wants to authenticate enters their username and password on their client machine. Their Kerberos client software, will then take the password and generate a symmetric encryption key from it. Next, the client sends a plain text message to the Kerberos, AS or Authentication Server which includes the user ID of the authenticating user. The password or secret key derive from the password aren't transmitted. The AS uses the user ID to check if there is an account in the authentication database, like an active directory server. If so the AS will generate the secret key using the hashed password stored in the key distribution center server. The AS will then use the secret key to encrypt and send a message containing the client TGS session key. This is a secret key use for encrypting communications with the Ticket Granting Service or TGS, which is already known by the Authentication Server. The AS also sends a second message with a Ticket Granting Ticket or TGT, which is encrypted using the TGS secret key. The Ticket Granting Ticket has information like the client ID, ticket validity period, and the client taking granting service session key. So the first message can be decrypted using the shared secret key derived from the user password. It then provides the secret key that can decrypt the second message giving the client a valid Ticket Granting Ticket. Now, the client has enough information to authenticate with the Ticket Granting Server. Since the client has authenticated and received a valid Ticket Granting Ticket, it can use the Ticket Granting Ticket to request access to services from within the Kerberos realm. This is done by sending a message to the Ticket Granting Service with the encrypted Ticket Granting Ticket received from the AS earlier along with the service name or ID the client is requesting access to. The client also sends a message containing in authenticator which has the client ID and a time stamp that's encrypted with the client Ticket Granting Ticket session key from the AS. The Ticket Granting Service decrypts the Ticket Granting Ticket using the Ticket Granting Service secret key, which provides the Ticket Granting Service with the client Ticket Granting Service session key. It then uses the key to decrypt the authenticator message. Next, it checks the client ID of these two messages to ensure they match. If they do, it sends two messages back to the client. The first one, contains the client to server ticket which is comprised of the client ID, client address, validity period, and the client server session key encrypted using the service's secret key. The second message, contains the client server session key itself, and is encrypted using the client Ticket Granted Service session key. Finally, the client has enough information to authenticate itself to the service server or SS. The client sends two messages to the SS. The first message is the encrypted client to server ticket received from the Ticket Granting Service. The second is a new authenticator with the client ID and time stamp encrypted using the client server session key. The SS decrypts the first message using its secret key which provides it with the client server session key. The key is then used to decrypt the second message, and it compares the client ID in the authenticator to the one included in the client to server ticket. If these IDs match, then the SS sends a message containing the time stamp from the client supplied authenticator encrypted using the client server session key. The client, then decrypts this message, and checks that the time stamp is correct authenticating the server. If this all succeeds, then the server grants access to the requested service on the client. Wow, okay, are you with me? I know that was a lot. Kerberos has received some criticism because it's a single monolithic service. This creates a single point of failure danger. If the Kerberos service goes down, new users won't be able to authenticate and log in. Aside from the availability issues, if the central Kerberos server is compromised, the attacker would be able to impersonate any user by generating valid Kerberos tickets for their user account. Kerberos enforces strict time requirements requiring the client and server clocks to be relatively closely synchronized, otherwise, authentication will fail. This is usually accomplished by using NTP to keep both parties synchronized using an NTP server. The trust model of Kerberos is also problematic, since it requires clients and services to have an established trust in the Kerberos server in order to authenticate using Kerberos. This means, it's not possible for users to authenticate using Kerberos from unknown or untrusted clients. So things like BYOD or Bring Your Own Device, and cloud computing are incompatible, or at least very challenging to implement securely with Kerberos authentication. Now as an IT support specialist, you're likely to encounter Kerberos authentication especially, in an environments running Microsoft Active Directory. Understanding how the underlying protocol functions will help when troubleshooting issues that may come with it.

### Tacacs

TACACS+ pronounced TACACS plus. It stands for Terminal Access controller Access Control System Plus. It's a Cisco developed AAA protocol that was released as an open standard in 1993. It replaced the older TACACS protocol developed in 1984 for MILNET. The unclassified network for DARPA, which later evolved into NIPRNet. TACACS+ also took the place of XTACACS or Extended TACACS which was a Cisco proprietary extension on top of TACACS. TACACS+ is primarily used for device administration, authentication, authorization, and accounting, as opposed to RADIUS, which is mostly used for network access AAA. It's important to call out these differences in the characteristics of what these services provide. Though the differences are primarily related to the authorization, and accounting portions, more so than authentication. While you might not encounter a TACACS+ implementation in the course of your support career, it's something that you should be aware of. TACACS+ is mainly used as an authentication system for network infrastructure devices, which tend to be high value targets for attackers. This may be something to consider implementing as your organization grows.

### Single sign on

Single Sign On or SSO is an authentication concept that allows users to authenticate once to be granted access to a lot of different services and applications. Since reauthentication for each service isn't needed, users don't need multiple sets of usernames and passwords across a mix of applications and services. SSO is accomplished by authenticating to a central authentication server, like an LDAP server. This then provides a cookie, or token that can be used to get access to applications configured to use SSO. Kerberos is actually a good example of an SSO authentication service. The user would authenticate against the Kerberos service once, which would then grant them a ticket granting ticket. This can then be presented to the ticket granting service in place of traditional credentials. So, the user can enter credentials once and gain access to a variety of services. SSO is really convenient. It allows users to have one set of credentials that grant access to lots of services, making it less likely that passwords will be written down or stored insecurely. This should also reduce the overhead for password assistance support and removes time spent re authenticating throughout the workday. So, what's the downside? Well, an attacker that manages to compromise an account has a lot more access under an SSO scheme. User credentials will grant access to all applications and services that that account is permitted to access. So, a big plug here for using multifactor authentication in conjunction with an SSO scheme. But this opens a new channel of attack, theft of SSO session cookies or tokens. Instead of targeting credentials directly, attackers can try to steal the SSO tokens directly which will permit wide access if even for a short amount of time. Stealing these tokens, also lets an attacker dodge multifactor authentication protections since the session token permits access without requiring full authentication until the token expires. An example of an SSO system is the openID, the centralized authentication system. This is an open standard that allows participating sites known as Relying Parties to allow authentication of users utilizing a third party authentication service. This allows sites to permit authentication without requiring the site itself to have authentication infrastructure, which can be tricky to implement and maintain. It also lets users access the site without requiring them to create a new account, simplifying access management across a wide variety of sites. Instead, a user just needs to already have an account with an identity provider. To ask for authentication, first a relying party looks up the openID provider, then establishes a shared secret with the provider if one doesn't already exist. The shared secret will be used to validate the openID provider messages. Then, the user will be redirected or asked to authenticate in a new window through the identities provider's log and flow. Once authenticated, the user will be prompted to confirm if they trust the relying party or not. Once confirmed, credentials are relayed to the relying party, typically in the form of a token not actual user credentials which indicates the user is now authenticated to the service. Now it's time for practice quiz on authentication.

### Rob path to it

[MUSIC] When I was a kid, my parents ended up having, I think it was a Commodore 64, and I loved playing video games on that. Then they bought an NES, a Nintendo Entertainment System, and I ended up playing that a whole bunch. When I got a little bit older, we got another computer and I started playing a game called Morrowind, but my computer couldn't really handle it very well. That end up, is what got me interested in computers. I had this need for playing video games, and that need necessitated me, learning how to fix the computer and do things to it. I just kept on going down that path. I started playing a lot of security type stuff back when I was in the late high school early college, Syberia. You could go and do something called r poisoning, you could make yourself, pretend like you're the router for a system, and then everyone is setting a package is for you, and you could see what's going on in the network. No one really cared back in the day about security, and it was just really open and free, and it was so many bad things that could happen. I just felt like this had to change. With that state of the world, you need people to be able to fix it, and I felt like this is a good way to help the world, basically. If I'm out there, patching up these holes and making the world a little bit better, then, I just felt that I'll be a little self fulfilling, maybe feel good, I guess. [MUSIC]

## Authorization

### Authorization and access control methods

Earlier, we covered authentication, the first component of the three A's of security. Next up, we'll cover authorization, which is usually tightly coupled with authentication. Now, while authentication is related to verifying the identity a user, authorization pertains to describing what the user account has access to or doesn't have access to. These are separate and distinct components of AAA that have different purposes. A user may successfully authenticate to a system by presenting valid credentials. But if the username they authenticated as isn't also authorized to access the system in question, they'll be denied access. When we talked about Kerberos earlier, the user authenticated and received a ticket granting ticket. This can then be used to request access to a specific service by sending a request to the ticket granting service. This is when authorization comes into play, since the ticket granting service will decide whether or not the user in question is permitted to access the service being requested. If they're not permitted or authorized to access the service, the request will be denied by the ticket granting service. If the user is authorized, the ticket granting service would return a ticket, which authorized the user to access the service. One very popular open standard for authorization and access delegation is OAuth, used by companies like Google, Facebook, and Microsoft. We'll go in to OAuth in the next video.

### Access control

OAuth is an open standard that allows users to grant third party websites and applications access to their information without sharing account credentials. This can be thought of as a form of access delegation because access to the user's account is being delegated to the third party. This is accomplished by prompting the user to confirm that they agree to permit the third party access to certain information about their account. Typically, this prop will specifically list which pieces of information or access are being requested. Once confirmed, the identity provider will supply the third party with a token that gives them access to the user's information. This token can then be used by the third party to access data or services offered by the identity provider directly on behalf of the user. OAuth is commonly used to grant access to third party applications, to APIs offered by large internet companies like Google, Microsoft, and Facebook. Let's say you want to use a third party meme creation website. This website lets you create memes using templates and gives you the option to save your creations and email them to your friends. Instead of the site sending the emails directly, which would appear to be coming from an address your friends wouldn't recognize, the site uses OAuth to get permission to send the memes using your email account directly. This is done by making an OAuth request to your email provider. Once you approve this request, the email provider issues an access token to the site, which grants the site access to your email account. The access token would have a scope, which says that it can only be used to access email, not other services associated with the account. So it can access email but not your cloud storage files or calendar, for example. It's important that users pay attention to what third party is requesting access and what exactly they're granting access to. OAuth permissions can be used in phishing style attacks to gain access to accounts without requiring credentials to be compromised. This works by sending phishing emails to potential victims that look like legitimate OAuth authorization requests, which ask the user to grant access to some aspects of their account through OAuth. Once the user grants access, the attacker has access to the account through the OAuth authorization token. This was used in an OAuth based worm attack in early 2017. There was a rash of phishing emails that appeared to be from a friend or colleague who wanted to share a google doc. When the sharing link was followed, the victim was prompted to log in and authorize access to email documents and contacts for some third party service, which only identified itself as the name Google Apps. But it was actually a malicious service that would then email contacts from their email account perpetuating the attack. In case you like to read more about it, I've included a link in the next reading. It's important to distinguish between OAuth and OpenID. OAuth is specifically an authorization system and OpenID is an authentication system. Though they're usually used together, OpenID Connect is an authentication layer built on top of OAuth 2.0 designed to improve upon OpenID and build better integration with OAuth authorizations. Since TACACS plus is a full AAA system, it also handles authorization along with authentication. This is done once a user is authenticated by allowing or disallowing access for the user account to run certain commands or access certain devices. This lets you not only allow admin access for users that administer devices while still allowing less privileged access to other users when necessary. Here's an example, since your networking teams are responsible for configuring and maintaining your network switches, routers, and other infrastructure. You'd give them admin access to your network and equipment. Meanwhile, you can have limited read only access to your support team since they don't need to be able to make changes to switch configurations in their jobs. Read only access is enough for them to troubleshoot problems. The rest of the user accounts would have no access at all and wouldn't be permitted to connect to the networking infrastructure. So more sophisticated or configurable AAA systems may even allow further refinement of authorization down to the command level. This gives you much more flexibility in how your access is granted to specific users or groups in your organization. RADIUS also allows you to authorize network access. For example, you may want to permit some users to have Wi Fi and VPN access while others may not need this. When they authenticate to the RADIUS server, if the authentication succeeds, the RADIUS server returns configuration information to the network access server. This includes authorizations which specifies what network services the user is permitted to access.

### Access control list

An access control list or ACL, is a way of defining permissions or authorizations for objects. The most common case you may encounter deals with file system permissions. A file system would have an ACL, which is a table or database with a list of entries specifying access rights for individuals or groups for various objects on the file system like folders, files or programs. These individual access permissions per object are called Access Control Entries and they make up the ACL. Individual entries can define permissions controlling whether or not a user or group can read, write or execute objects. ACLs are also used extensively in network security, applying access controls to routers switches and firewalls. Network ACLs are used for restricting and controlling access to hoster services running on hosts within your network. Network ACLs can be defined for incoming and outgoing traffic. They can also be used to restrict external access to systems and limit outgoing traffic to enforce policies or to prevent unauthorized outbound data transfers. We'll deep dive more into network ACLs in future lessons when we cover network security in more detail.

## Accounting

### Tracking usage and access

Last but not least, the final A of the Triple A's of security is Accounting. This means, keeping records of what resources and services your users access or what they did when they were using your systems. A critical component of this is auditing, which involves reviewing these records to ensure that nothing is out of the ordinary. If we're watching and recording usage of our systems, but never actually checking the usage data, that's not super useful. So what exactly do counting systems keep track of? Well, that depends on the purpose and intent of the system. For example, a TACACS+ server would be more concerned with keeping track of user authentication, what systems they authenticated to, and what commands they ran during their session. This is because TACACS+ plus is a device access AAA system that manages who has access to your network devices and what they do on them. CISCO's AAA system supports accounting of individual commands executed, connection to and from network devices, commands executed in privileged mode, and network services and system details like configuration reloads or reboots. RADIUS will track details like session duration, client location and bandwidth, or other resources used during the session. This is because radius is a network access AAA system. So it tracks details about network access and usage. RADIUS accounting kicks off with the Network Access Server sending an accounting request packet to the accounting server that contains an event record to be logged. This starts the accounting session on the server. The server replies with an accounting response indicating that the message was received. The NASS will continue sending periodic accounting messages with statistics of the session until an accounting stop packet is received. RADIUS accounting can be used for billing purposes by ISPs because it records the length of a session and the amount of data sent and received by the user. This data can also be used to enforce data or time quotas, limiting the duration of sessions or restricting the amount of data that can be sent or received. But, this accounting information isn't detailed and won't contain specifics of what exactly the user did during the session. Information like websites visited or what protocols were used aren't recorded. Other logging utilities that will cover later meet that use case. Wow. Okay. We covered a lot in this module. Great work. Take the time to go over anything that may have been a little dense. When you're ready, we've got a project for you that will test you on authentication, authorization, and accounting system concepts. Once that's done, you can head to the next video, where we'll dive into network security, monitoring, and logging.

### Rob important skills in security

So, I think the thing that I enjoy the most is just making other people happy, may have some type of issue, their computer is broken in some way and you fix it. And then there's a giant smile on their face and they're so happy that they can get back to their work and get back to what they're doing. And that's just really nice to be able to help someone like that and have them so grateful. So, IT support has helped me and my career as a security engineer because when you're a security engineer, you have to actually be able to tell people, "No. You shouldn't do this. You shouldn't do that. It's not secure." But as an IT support person, you're supposed to say, "Yes. Yes, I will fix that. Yes, I will do this. Yes, I will do that." So it's kind of bridged the gap for me as I say, "No, but here's a better way to do it," or, "Hey, you can do it this way and you'll be able to get what you want." When I first joined running the code that creates all of our firewall policy, that took about 40 minutes to get through all the different policies. And I just thought about this and I said, "This is extremely long. We're running this multiple times a day. Everyone on the team is doing that." So, I took a look at it and then I got it down to about six minutes from start to finish for it to run. And I actually did the calculation at the end and it saved enough money that we could have hired another person at the end of that. I know plenty of people in the field who don't have a degree at all. They have their high school education and that's it. To be successful, you need to be able to problem solve. You need to be able to interface with people and communicate. Those are the two big things. If you can figure out how to solve someone's problem and you can tell them how that problem needs to be fixed, you got at least 90 percent of what needs to be done.