Access Control and Authentication

Linux Systems Security

Access Control

- Authenticating users is the act of trying to verify that a user is who they claim to be
- We generally rely on the concept that if a user has information that should only be known by the user themselves, then they are very likely to be authentic
- More sophisticated systems including biometrics and multifactor authentication are based on the same reliance that if a user has the physical access items, they are authentic
- Limiting or controlling access can happen in stages, first at login or connection, then at application launch, then runtime access, etc.
- This unit focuses on user access to the system via login and authentication of hosts and individuals

Login Access Control

- Logins typically happen on physical connections such as terminals, and on network connections using one of several possible remote access protocols
- /etc/security/access.conf specifies rules about where users can login from - see access.conf(5) or review the comments in the file
- /etc/securetty provides a simple mechanism to limit where root can log in, default file allows root to log in pretty much anywhere - see file for documentation

Passwords

- Passwords are most commonly required for a user to authenticate
- Typing them can be avoided by substituting ssh keys which is essentially a long unguessable saved password (the private key) which is validated using a public key instead of comparison with a stored hash of the password - protects against brute forcing hashes and weak passwords
- Access to services/running applications does not necessarily require a UNIX login, each service may have its own authentication mechanism and policy enforcement options, e.g. mail, web applications, database access, etc.
- Various access control programs use pluggable authentication modules (PAM) to provide rules and mechanisms for user authentication
- http://www.linux-pam.org/Linux-PAM-html/Linux-PAM_SAG.html is an extensive guide to using Pluggable Authentication Modules (PAM) to provide, enhance, and enforce access-related security features of Linux, see https://thelinuxcode.com/linux_pam_tutorial/ for a simplified overview of how PAM works

Password Management in Linux

- /etc/shadow file holds encrypted passwords for local unix accounts and has several fields which can be used to enforce a password policy for ordinary users - see shadow(5) and passwd(1) man pages
- Interesting password options for /etc/pam.d/login could include minlen, remember, obscure - see pam_unix(8) and pam.conf(5) man pages
- /etc/login.defs provides some additional control options to customize login access restrictions (retries, timeouts, logging, etc.) - see login.defs(5)
- libpam-pwquality provides a PAM module which significantly enhances the minimum allowed password quality, other modules can provide additional functionalities
- Consider using a robust password cracking program to identify when users are using poor passwords, and set their password to expire when a poor one is found (e.g. passwd -w 3 -x 1 username), this is a good task for a script

Legacy Remote Access

- telnet is a very old protocol which allows creation of connections to ports on remote hosts using tcp
- telnet uses no compression, no encryption, no awareness of connection types, it simply connects your terminal and keyboard to a remote socket
- telnet is still a useful tool for testing and diagnostic purposes, but the telnetd daemon is extremely uncommon now and strongly discouraged

Legacy Remote Access

- rsh/rcp are tools using a login-specific protocol to establish user access to their unix accounts on remote hosts
- rsh/rcp connect to a rlogind daemon on a remote site and check for environment files which might allow a user to connect to their login shell without entering a password for purposes of interactive access or file transfer
- rsh/rcp do not support compression or encryption and are not useful for other purposes, enabling the rlogind service is strongly discouraged

SSH Remote Access

- ssh/scp/sftp is the current preferred remote access toolset and supports much more than unix account access
- ssh tools support encryption and compression
- OpenSSH has both client and server side tools and configurations
- /etc/ssh holds the configuration files and is world accessible because client configuration files (ssh_config mainly) are kept there
- With your public key installed under your home directory on a server, any user who has your private key can log into the server without manual password entry - Protect your private key with a passphrase! - can also configure MFA
- See https://linux-audit.com/using-ssh-keys-instead-of-passwords/

SSH Server

- /etc/ssh/sshd_config is for the server daemon process
- options to consider include port number, host key files, logging config, logingracetime, permitrootlogin, pubkey authentication, allowusers, denyusers, maxauthtries, maxstartups, passwordauthentication
- /etc/ssh has host key files provided to clients during session establishment for host identity confirmation

https://help.ubuntu.com/lts/serverguide/openssh-server.html

Authentication with Certificates

- SSL/TLS certificates identify an entity (typically a host or individual) and are signed, usually by a trusted authority
- Your trusted authority list is included with your OS and you can modify it (i.e. ca-certificates package)
- Certificates are used with public/private key pairs (the public key is in the certificate) to enable encryption
- SSH does not use certificates and keeps keys in different formats from SSL/TLS
- Certificates can be used to identify clients by applications that establish two-way authenticated TLS/SSL connections

Public/Private Key Pairs

- Private key decrypts things encrypted with public key
- Public key decrypts things encrypted with private key
- Key generation tools are part of ssh and openssl packages (ssh-keygen and openssl genpkey)
- easy-rsa package is a software package which is more user friendly for in-house simple CA administration, typically installed in /etc/openvpn/easy-rsa

Certificates

- Root certificates are self-signed and are part of the public key infrastructure (PKI)
- Certificates must be signed by a trusted authority, or cannot be trusted for authentication
- Untrusted certificates can still be used for encryption, even if they cannot be trusted for authentication, most commonly used for things like SMTPS, IMAP4S, POP3S, intranet servers, reducing them to being a file format for public key storage
- Creating a certificate involves creating a signing request and having it signed to produce the actual certificate

Requesting a Certificate

- Begin by generating a private key if you do not already have one
  ```
  openssl genpkey -algorithm RSA -out /etc/ssl/private/mykey.pem -pkeyopt rsa_keygen_bits:2048
  ```
- Generate a request with embedded public key using your private key
  ```
  openssl req -new -key /etc/ssl/private/mykey.pem -out mycsr.csr
  ```
- Send the request to the Certificate Authority along with your authentication documentation and payment, and they will send back your certificate file
- Install your certificate file in `/etc/ssl/certs` and you can use it in your configuration files for services wanting to use SSL/TLS, identifying as the server named in the certificate

Certificate Authority

- A certificate authority (CA) is an organization that validates signing requests and then signs them with the CA's private key, adding the CA info to the certificate
- Validation normally requires documentation of identity and authority from requester
- CAs charge for this service, and certificates expire so this is an ongoing business
- CAs can revoke certificates
- You can create your own CA trivially although no-one will trust your signed certificates unless they install your CA certificate as a trusted root certificate on their system - useful for intranet certificates
- This is the certificate creation method most commonly used when using certificates for client authentication

Private CA Example with CA.pl

- You can create a demo CA in /etc/ssl (e.g. cd /etc/ssl;/usr/lib/ssl/misc/CA.pl -newca)
- For each site you want certificates for, generate a key and csr (e.g. cd /etc/ssl;/usr/lib/ssl/misc/CA.pl -newreq)
- Sign the csr with your CA certificate and key to produce their certificate (e.g. cd /etc/ssl;/usr/lib/ssl/misc/CA.pl -sign)
- Send newcert.pem, newkey.pem to the requesting host to be installed and configured into whatever service will use the certificate (e.g. apache2)
- Make sure to install your CA certificate on clients that will be accessing the sites which use certificates you have signed

Deploying a Private-CA Signed Certificate

- Most browsers have their own trusted certificates list but users get warned about certificates which are not signed by a trusted root, this would defeat the purpose of using a better certificate than self-signed
- Various operating systems have distinct methods of installing trusted root certificates
- Installing a new CA certificate on Ubuntu starts with saving the CA certificate file to /usr/share/ca-certificates/somefilename.crt and then dpkg-reconfigure ca-certificates
- Installing a new CA certificate on a Mac looks like this
  security add-trusted-cert -d -r trustRoot -k "/Library/Keychains/System.keychain" "/private/tmp/certs/certname.cer"
- Installing a new CA certificate on a Windows host takes different forms for the different OS versions as well as for different browsers, IIS-generated CSRs would also have a different command to sign them with the demoCA (e.g. cd /etc/ssl;ca -policy policy_anything -notext -in clients.server.com.req -days 365 -out clients.server.com.crt)
- For a service provider, trusting client's certificates requires trusting the certificate signer, so the client certificate CA must be trusted by the server