Herbsttagung 2022 des ZKI-Arbeitskreises Supercomputing

Zugang zu kritischen Infrastrukturen mit Jump-Hosts und SSH-Zertifikaten

Bernd Wiebelt, Universität Freiburg

www.nemo.uni-freiburg.de | www.hpc.uni-freiburg.de | www.bwhpc.de/wiki | www.bwhpc.de

Motivation

- Dilemma:
  - Protect (critical) IT infrastructures from cyberattacks
  - Provide (convenient) access for authorized users
  - In particular: Provide access for administrative users
- Goal:
  - Reduce attack vectors
  - Monitor and log suspicious activities
  - Threat analysis and mitigation plan prior to incident: “What if...”

Reducing attack vectors I

- Cut the cable and require physical access
  - Viable for a few cases
  - Most systems need to be reachable via network, if only for management
- Use private IP addresses
  - Problem: How big is the private subnet? (Uni FR uses 10.0.0.0/8)
  - Some gateway hosts still need to be reachable via public IP addresses
- Firewall
  - Rules get more complex with increasing number of functional subnets
  - Strategy: Per host, central or hybrid? Reviews and audits in place?

Reducing attack vectors II

- Allow only access from trusted IP addresses
  - Probably most successful counter measure after HPC attack in 05/2020
  - Makes VPN mandatory when working from home or on the road
- VPN
  - Works quite OK most of the time
  - Annoying client software, not only but in particular for Linux
  - Often configured to give access to the whole private network (e.g. 10.0.0.0/8), i.e. using a hammer to drill a hole
  - Routing everything through the tunnel is not optimal, having different VPN routing options is a support nightmare

Reducing attack vectors III

- Avoid using passwords over network, they are evil most of the time
  - Functionally, they constitute a SHARED secret between two partners
  - Passwords are received as plain text on the authenticating side, even if encrypted on transport
  - You have to assume that the other side has not been corrupted
  - (By the way: If the corruption is on your end, the game is over no matter what)
- Alternative 1: Use a 2FA
  - Second factor needs to be verified by a third party (i.e. not the same server you login to)
- Alternative 2: Use private/public key authentication
  - Well proven, but may come come with subtle problems and scaling issues
  - SSH Certificates may come in handy

Jumphost usage (classic)

- Gateway for external access to internal services
- Function Jump (SSH)
  - Allows to access hosts behind a firewall and/or on a private network
  - Jump host is allowed access via routing or firewall rules
- Function Transfer (SCP, SFTP)
  - Allows to access and copy files
  - Jump host has access to a central file server
- Function Interactive (Login)
  - Do other work, e.g. ping machines for aliveness

Jumphost threat analysis

- Number of users
  - Risk of stolen identities and hacking attempts scales with number of users
- Password access (...without 2FA)
  - Successful attacker can log passwords and steal identities
- File transfer
  - Interactive login not needed. Unfortunately, vanilla SSH offers both for historical reasons
- Interactive logins pose a high risk. Don’t do any of those two things EVER:
  - localhost$ ssh user@jumphost
    jumphost$ ssh admin@server
    Private key on jump host. Compromised SSH daemon will steal it once you type your passphrase
  - localhost$ ssh -A user@jumphost
    jumphost$ ssh admin@server
    Private key on localhost. Agent forwarding will protect your private key, but a compromised SSH daemon can log your input stream and can hijack sessions

UNI FREIBURG

Jumphost recommendations

- General
  - No password access, public key only
  - Public keys managed externally, i.e. no direct user access to .ssh/authorized_keys
- Dedicated Transfer Hosts
  - SFTP only
  - No interactive login or port forwarding allowed
- Dedicated ProxyJump Hosts
  - No interactive sessions, no valid shell (e.g. /sbin/nologin)
  - No writeable home directory
  - Only allow ProxyJump to other hosts

SSH ProxyJump (“Jump hosts and Do hosts“)

- localhost$ ssh -J user@jump_host.public admin@do_host.private
- Or use .ssh/config:
  Host do_host.private
  ProxyJump user@jump_host.public
- Advantages:
  - Session is encrypted end-to-end between localhost and do_host
  - Jump host does not need to be trusted. Even if jump host is compromised
    - Attacker cannot gain any secret information
    - Attacker cannot hijack active session
  - This holds even if the SSH daemon itself on the jump host is compromised
- Disadvantages:
  - do_hosts are needed, since there is no interactive login on jump hosts any more
  - do_hosts need to be reachable from jump hosts (e.g. firewall rules), so compromised jump hosts can still be potential hops for further attacks
- Compromising a jump host is no longer a jackpot.

UNI FREIBURG

SSH Certificates (motivation)

- SSH private/public keys are easy to start with
  ssh-keygen
  ssh-copy-id user@remote_host
- Individual management becomes complex quickly, if done properly
  - A distinct keypair for every service? (Hint: Use .ssh/config)
  - One passphrase for all? One for each key?
  - Key lifecycle: Should I renew keys? If so, how often?
  - Book keeping: Where did I deploy my public keys again?
- Public key administration for critical infrastructures does not scale well
  - Key lifecycle: How often should keys be renewed?
  - How do you manage deployment? Ansible? Puppet? Rsync? Git?
  - In particular: What happens if a new administrator arrives?
  - In particular: What happens if an administrator retires?

UNI FREIBURG

SSH Certificates (principal idea, user view)

- Before:
  - Create private/public key pair, e.g. id_rsa and id_rsa.pub
  - Deploy public key (id_rsa.pub) to remote machine, e.g. via ssh-copy-id
  - Problem: Public keys become static files scattered on various remote machines
- After:
  - Create private/public key pair, e.g. id_rsa and id_rsa.pub
  - Transmit id_rsa.pub for certification, receive id_rsa-cert.pub
  - Keep id_rsa, ida_rsa.pub and id_rsa-cert.pub in your local .ssh directory
  - Login to all remote machines that trust the certificate

SSH Certificates (principal idea, administrative view)

- Before:
  - Bad practice: Deploy initial public key for new admins on servers, self-service after that.
  - Slightly better practice: Develop a deployment mechanism for user/admin public keys on remote servers. Takes time and effort, no standard implementation available
- After:
  - ONCE: Create a SSH CA (Certificate Authority). This is just a single command creating two files. Keep the private file in a secure location.
  - ONCE (per remote machine): Establish trust relationship between remote machine and CA. This is just a single SSH option line and a file copy
  - ONCE (per user): Sign the user's public key and return the certificate. The certificate is useless without the private key, so this can even be transmitted on non-trusted channels (e.g. E-Mail)

UNI FREIBURG

SSH Certificates (in practice)

- Create Certificate Authority
  - Command: ssh-keygen -f CA
  - Creates two files: CA (private part) and CA.pub (public part)
- Establish trust relationship between a machine and CA
  - Copy CA.pub to server:/etc/ssh/CA.pub
  - Add the following line to /etc/ssh/sshd_config
    TrustedUserCAKeys /etc/ssh/CA.pub
- Sign a user's public key
  - ssh-keygen -s CA -I ID -n USERNAME -V +52w id_rsa.pub
  - This creates id_rsa-cert.pub, which needs to be transmitted to the user
  - The user can login as the "principal" USERNAME on any machine that trusts the CA
  - Using "root" as the principal is not a good idea
  - ID can be used to identify the certificate (e.g. for revocation)
  - Option -V +52w limits the validity of the certificate to 52 weeks

SSH Certificates Advantages

- Restrictions are in-built (i.e. time validity) and can be verified by the users themselves. (To some extent this is also possible with .ssh/authorized_keys, but more cumbersome.)
- Management of static public key files on remote machines is no longer needed
- Users can opt to utilize a single private/public key with multiple services and certificates via .ssh/config:

```
Host alice
        IdentityFile .ssh/id_rsa.pub
        CertificateFile id_rsa-alice-cert.pub
Host bob
        IdentityFile .ssh/id_rsa.pub
        CertificateFile id_rsa-bob-cert.pub
```

- SSH certificates can also be used for signing host keys, eliminating the need to keep and distribute a central ssh_known_hosts file

UNI FREIBURG

SSH Certificates: Principals vs Usernames

- Recap: Signing a public key involves a “principal”:
  ssh-keygen -s CA -I ID -n USERNAME -V +52w id_rsa.pub
- The default behaviour is to associate principals to usernames
- This works well and as expected for individual unprivileged users
- Note that you can login as the respective user on any machine that trusts the CA
- In particular, using “root” as principal is probably too powerful
- Instead, consider using the AuthorizedPrincipalsFile directive in /etc/ssh/sshd_config:
  AuthorizedPrincipalsFile /etc/ssh/AuthorizedPrincipals/%u
- etc/ssh/AuthorizedPrincipals/root would then contain a list of all principals that are allowed to login as root on this machine

UNI FREIBURG

SSH Certificates: Outlook

- Certificates that last for a longer period (i.e. a few months) are not yet optimal and only a step in the right direction
  - You forget about renewal
  - You need certification revocation mechanisms
  - You still need some management scripts
- Ideally, certificates would only be valid for a short time (e.g. work day)
- Administrators should use strong authentication (e.g. 2FA or FIDO2) to get a certificate and add it to their ssh-agent
- Content of AuthorizedPrincipalsFile should be managed in a central location to allow for granular and flexible access rights (there is also AuthorizedPrincipalsCommand)
- This is actually what big internet companies already seem to have in place

UNI FREIBURG