Sécurisation des protocoles

SSH : Secure SHell Protocol

Benoît Morgan

benoit.morgan@enseeiht.fr

Carlos Aguilar

carlos.aguilar-melchor@isae-supaero.fr

IRIT

Plan

Plan

Sources

Lecture intensive de :

[1] https://tools.ietf.org/html/rfc4250
The Secure Shell (SSH) Protocol Assigned Numbers

[2] https://tools.ietf.org/html/rfc4251
The Secure Shell (SSH) Protocol Architecture

[3] https://tools.ietf.org/html/rfc4252
The Secure Shell (SSH) Authentication Protocol

[4] https://tools.ietf.org/html/rfc4253
The Secure Shell (SSH) Transport Layer Protocol

[5] https://tools.ietf.org/html/rfc4254
The Secure Shell (SSH) Connection Protocol

[6] https://tools.ietf.org/html/rfc4255
Using DNS to Securely Publish Secure Shell (SSH) Key Fingerprints

[7] https://tools.ietf.org/html/rfc4256
Generic Message Exchange Authentication for the Secure Shell Protocol (SSH)

Sources

Et de :

[8] https://tools.ietf.org/html/rfc4419
Diffie-Hellman Group Exchange for the Secure Shell (SSH) Transport Layer Protocol

SSH in a nutshell

Définition

The Secure Shell (SSH) Protocol is a protocol for secure remote login and other secure network services over an insecure network. [1]

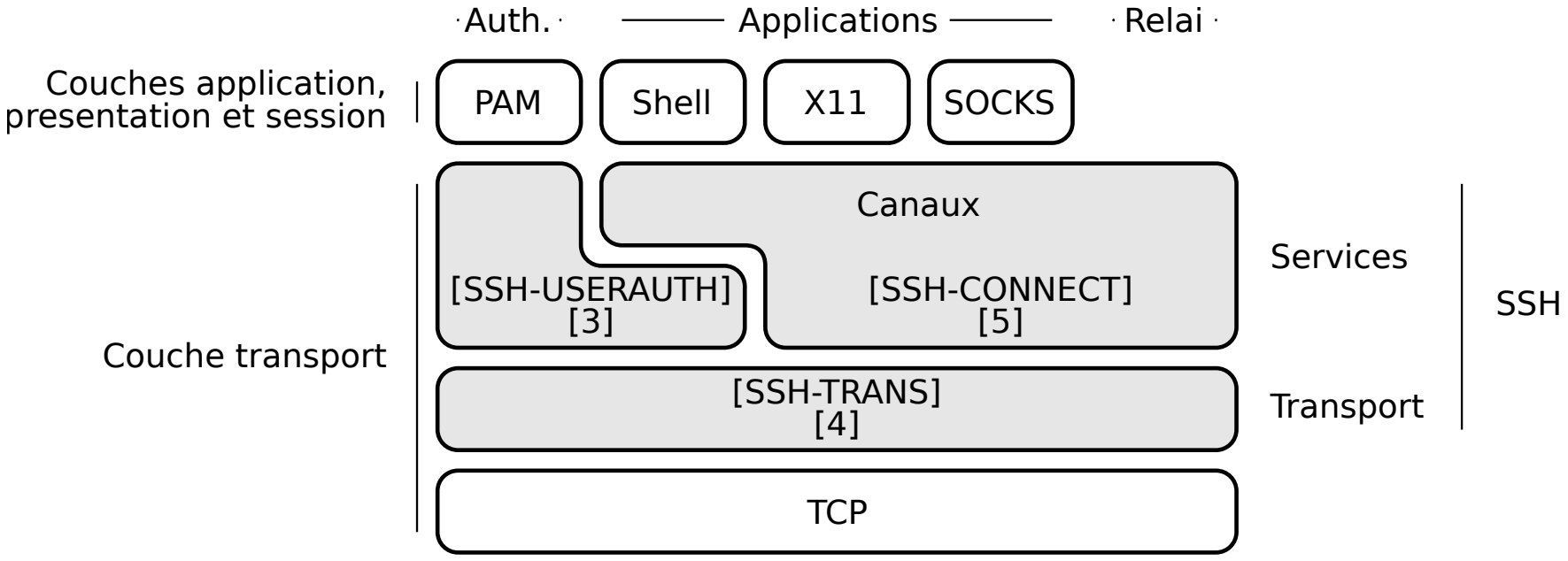
Services applicatifs construits à l'aide d'SSH

- Terminal distant (interactif ou non)
- Transfert de fichiers
- Proxy SOCKS5
- Encapsulation / redirection de flux

Sécurité

Chaque paquet et chiffré et envoyé avec un tag d'intégrité (MAC)

Pile protocolaire SSH



Apports (un peu plus en détail)

Une nouvelle couche de transport

- Confidentialité des messages
- Integrité (Authenticité) des messages
- Authentification du serveur

Un système d'authentification des utilisateurs

- Authentification de l'utilisateur
- Autorisation de l'utilisateur

Une couche connexion

- Canaux de communication
- Qualité de service

Plan

La couche transport 1/4

Objectifs

- Sécurisation de services réseaux
- Chiffrement fort
- Authentification du serveur
- Protection de l'intégrité des messages (crypto)
- Compression des messages si nécessaire

Services

- [SSH-USERAUTH] : authentification et autorisation d'utilisateurs
- [SSH-CONNECT] : accès à des services sécurisés

Dépendances

- SSH : couche de sécurisation d'un transport fiable

→ Transport transparent en mode flux, TCP/22

La couche transport 2/4

Flexibilité : négociation des algorithmes et protocoles

- Échange de clés
- Authentification à clé publique des hôtes (serveur, ou hôte client)
- Chiffrement symétrique client $\leftrightarrow$ serveur
- Tag d'intégrité client $\leftrightarrow$ serveur
- Compression client $\leftrightarrow$ serveur

Mécanisme d'extension des algorithmes et des services

- Nombre fini d'algorithmes standards
- ex : `hmac-sha1`, `diffie-hellman-group1-sha1`
- Extensions désignées par l'ajout de `@dns-domain`
- ex : `chacha20-poly1305@openssh.com`, `aes256-gcm@openssh.com`

La couche transport 3/4

Format des paquets

- 32 kilo octets de taille maximum
- 16 octets de taille minimum

```
(1) uint32    packet_length
(2) byte      padding_length
(3) byte[n1]  payload          n1 = packet_length - padding_length - 1
(4) byte[n2]  random_padding   n2 = padding_length
(5) byte[m]   MAC              m  = MAC length, e.g. 160 bits (HMAC-SHA-1)
```

- La taille d'un paquet est contrainte de la manière suivante : $\text{taille}((1)||(2)||(3)||(4)) \equiv 0\ [\max(\text{taille d'un bloc chiffré}, 8)]$
- Doit permettre de déchiffrer au plus tôt la taille du message
- (5) est présent après négociation des protocoles

La couche transport 4/4

Protocole

0. Identification des versions
1. Phase d'échange des clés
   1. Négociation des algorithmes
   2. Échange des clés
   3. Authentification du serveur

→ Début des transmissions chiffrées

2. Demande d'accès au service

∞ Échange de messages de services

→ Rééchange des clés : refaire l'étape 1

Performances

Les étapes 1 et 2 doivent s'effectuer en 2 à 3 allers / retours

Phase d'échange des clés

Négociation des algorithmes de la session

- Échange des listes ordonnées d'algorithmes supportés (1 = préféré)
- Pour les deux sens par les deux pairs en même temps, on devine

⇒ Évite une négociation si accord immédiat

- Sinon négociation en fonction des listes d'algorithmes

Message

```
byte          SSH_MSG_KEXINIT
byte[16]      cookie (random bytes)                      Anti rejeu
name-list     kex_algorithms                             Variations de DH
name-list     server_host_key_algorithms                 Ex: RSA-sha1
name-list     encryption_algorithms_client_to_server     Chiffres symétriques
name-list     encryption_algorithms_server_to_client
name-list     mac_algorithms_client_to_server            Tag d'intégrité
name-list     mac_algorithms_server_to_client
name-list     compression_algorithms_client_to_server    Compression
name-list     compression_algorithms_server_to_client
[..]
```

Rappels informels chiffrement asymétrique

Signature RSA : intuition

- $\exists e, d, n, m \in \mathbb{N}, m < n | (m^e \mod n)^d \mod n = m$, "faciles à générer"
- Chiffre : $m, (e, n) \rightarrow m^e \mod n | m =$ clair, $K_{priv} = (e, n)$
- Dechiffre : $c, (d, n) \rightarrow c^d \mod n | c =$ chiffré, $K_{pub} = (d, n)$
- Sign : $c, K_{priv} \rightarrow$ Chiffre$(\mathcal{H}(c), K_{priv}) | \mathcal{H} =$ fonction de hashage

Échange de clé Diffie Hellman : intuition

- Alice et Bob choisissent un groupe fini et un générateur de ce groupe
- Alice choisi a au hasard et calcule $A = g^a$ et l'envoie à Bob
- Bob choisi b au hasard et calcule $B = g^b$ et l'envoie à Alice
- Alice et Bob calculent $MSK = A^b = B^a = (g^a)^b = (g^b)^a = g^{ab}$
- Exemple : le corps $\mathbb{Z}/p\mathbb{Z}$ et p premier et calcul modulo p

Échange de clé et authentification du serveur (1/3)

Pré-requis

Chaque serveur dispose d'une clé publique l'identifiant : e.g. `/etc/ssh/ssh_host_rsa_key.pub` (générée lors de l'inst. du serveur)
Cette clé publique correspond à une paire de clés notée $(verif_s, sign_s)$

Protocole

- Client génère une clé DH éphémère $(pub_c = g^{priv_c}, priv_c)$ et envoie pub_c
- Serveur génère une clé DH éphémère $(pub_s = g^{priv_s}, priv_s)$ et calcule $PMK = pub_c^{priv_s}$
- Serveur calcule $H = \mathcal{H}(\text{Context}||verif_s||pub_c||pub_s||PMK)$, $\mathcal{H}$ fonction de hashage
- Serveur calcule $\sigma = \text{Sign}(sign_s, H)$
- Serveur envoie $(verif_s, pub_s, \sigma)$
- Client calcule PMK, H et vérifie σ en utilisant $verif_s$

PMK : *Primary Master Key* $\Rightarrow$ dérivation des clés de session
Context = (les deux *MSG ID strings*||les deux *MSG negociations*)
Comment sait le client que $verif_s$ correspond au serveur ?

Échange de clé et authentification du serveur (2/3)

Vérification de la clé du serveur

Le client utilise un principe de TOFU en enregistrant les clés dans `~/.ssh/knwons_hosts` :

- Hôte pas dans le fichier → demande de confirmation
- Hôte présent mais clé différente → refus
- Hôte présent et clé inchangée → ok

Si on réinstalle il faut penser à sauvegarder les clés de sshd !

Échange de clé et authentification du serveur (3/3)

Message d'initialisation Diffie Hellman client vers serveur

byte	SSH_MSG_KEXDH_INIT	
mpint	e	pubc

Message de réponse Diffie Hellman réponse serveur vers client

byte	SSH_MSG_KEXDH_REPLY	
string	server public host key and certificates (K_S)	ex: clé RSA
mpint	f	pubs
string	signature of H	

Message de fin de l'échange et début du chiffrement

byte	SSH_MSG_NEWKEYS

Démo : `ssh -c aes256-ctr -oKexAlgorithms=diffie-hellman-group-exchange-sha256 m0rgan.net`

Dérivation des clés de session

Que faire de la clé maître primaire (PMK) et de l'empreinte H ?

Objectif

- Générer localement les clés de chiffrement symétrique
- Générer localement les clés de tag d'intégrité

Protocole

- Dérivation de clés à l'aide de H, MSK et de la fonction de hashage $\mathcal{H}$
- $ID = H$: H est l'identifiant unique de cette session, et jamais modifié
- $IV_{sc} = \mathcal{H}(MSK \| H \| \text{'A'} \| ID)$: IV serveur vers client
- $IV_{cs} = \mathcal{H}(MSK \| H \| \text{'B'} \| ID)$: IV client vers serveur
- $KE_{sc} = \mathcal{H}(MSK \| H \| \text{'C'} \| ID)$: clé de chiffre serveur vers client
- $KE_{cs} = \mathcal{H}(MSK \| H \| \text{'D'} \| ID)$: clé de chiffre serveur vers client
- $KI_{sc} = \mathcal{H}(MSK \| H \| \text{'E'} \| ID)$: clé de tag d'intégrité serveur vers client
- $KI_{cs} = \mathcal{H}(MSK \| H \| \text{'F'} \| ID)$: clé de tag d'intégrité serveur vers client

Chiffrement

Objectifs

- Protection de la confidentialité des messages
- Clé des chiffrement différents selon de sens de communication
- Chiffrement de la totalité des paquets
- Les données forment un unique chiffré par sens de communication

Protocole

- Algorithmes de chiffrement forts (clé $>$ 128 bits, blocs $>$ 128 bits)
- Déchiffrement de la taille lors de la réception du premier bloc d'un paquet
- L'*IV* du paquet N est conditionné déterminé par le paquet $N - 1$

Algorithmes spécifiés (2018)

- `aes128-ctr`, `aes256-ctr`
- *AEAD* : `chacha20-poly1305`, `aes128-gcm`

Tag d'intégrité (MAC)

Objectifs

- Protection de l'intégrité de chaque paquet échangé
- Intégrité $=$ authenticité $+$ validité $+$ précision $+$ intégralité

Protocole

- Ajout d'un tag d'intégrité à chaque paquet échangé
- Avant l'échange de clé le *mac* n'est pas présent (taille $= 0$)
- $mac = \text{MAC}(KI, \text{numéro de séquence} \| \text{clair})$
- Numéro de séquence est implicite est non transmis
- Numéro de séquence $= 0$ en début de session
- Incrémenté jusqu'au dépassement d'entier sur 32 bits

Algorithmes spécifiés (2018)

- HMAC : HMAC-SHA-1, HMAC-SHA-256, tags AEAD
- HMAC : $K, C \rightarrow \mathcal{H}((ipad \oplus K) \| \mathcal{H}((opad \oplus K) \| C))$

Dernière étape : demande de service

Services

- Couche protocolaire s'exécutant au dessus de la couche transport
- `"ssh-userauth"` : authentification des utilisateurs
- `"ssh-connect"` : service operationnel (sessions interactives, etc.)

Protocole

- Demande d'accès

```
byte      SSH_MSG_SERVICE_REQUEST
string    service name                "ssh-userauth" ou "ssh-connect"
```

- Réponse

```
byte      SSH_MSG_SERVICE_ACCEPT
string    service name                "ssh-userauth" ou "ssh-connect"
```

- Si service non accessible, le serveur doit se déconnecter

Plan

Service d'authentification des utilisateurs 1/3

Objectifs

- Authentifier le client selon plusieurs méthodes robustes
- Le mode d'authentification peut être négocié
- Le serveur doit pouvoir demander plusieurs modes d'authentification

Dépendances

- S'appuie sur la couche transport [SSH-TRANS]
- L'échange de clé doit être terminé et le service demandé
- L'identifiant de session courante

Méthodes d'authentification

- Par clé publique : `"publickey"`
- Par mot de passe : `"password"`
- Basé sur l'hôte du client : `"hostbased"`

Service d'authentification des utilisateurs 2/3

Protocole

0. Le client liste les méthodes supportées (optionnel)
1. Le client exécute une des méthodes d'authentification
2. Le serveur accepte l'authentification du client

? Si d'autres méthodes nécessaires ou échec $\rightarrow$ (1)

Méthodes multiples

- Le serveur retourne un message d'échec avec réussite partielle
- Il liste les méthodes encore nécessaire à utiliser pour un succès

Lister les méthodes supportées par le serveur

- Envoi d'un requête avec méthode `"none"`
- Le serveur retourne un échec avec la liste des méthodes
- Si le client ne doit pas être authentifié $\rightarrow$ succès

Service d'authentification des utilisateurs 3/3

Message de requête d'authentification

```
byte      SSH_MSG_USERAUTH_REQUEST
string    user name in ISO-10646 UTF-8 encoding [RFC3629]
string    service name in US-ASCII
string    method name in US-ASCII
[..]      method specific fields
```

Message de réponse positive

```
byte      SSH_MSG_USERAUTH_SUCCESS
```

Message de réponse négative

```
byte          SSH_MSG_USERAUTH_FAILURE
name-list     authentications that can continue
boolean       partial success
```

Si `partial success` est vrai, recommencer jusqu'au succès (1)

Authentification des utilisateurs par clé publique 1/2

Génération

`ssh-keygen` → paire de clés `id_rsa.pub`, `id_rsa` dans `~/.ssh`
On génère tout aussi souvent des clés DSA

Transfert

L'utilisateur ajoute le contenu de `id_rsa.pub` dans `~/.ssh/authorized_keys` au niveau du serveur

L'authentification

À chaque connexion le client ssh propose d'utiliser une authentification par clé publique avant d'en proposer une par mot de passe.

1. Le client cherche `id_rsa.pub` dans `~/.ssh` et la propose au serveur
2. Le serveur vérifie que la clé est dans `authorized_keys`
3. Si c'est le cas il demande au client de signer un message unique et accepte l'authentification si la signature est correcte

Authentification des utilisateurs par clé publique 2/2

1 - Requête d'authentification par clé publique

```
(1) byte      SSH_MSG_USERAUTH_REQUEST
(2) string    user name in ISO-10646 UTF-8 encoding  ex: bmorgan
(3) string    service name in US-ASCII               ex: "ssh-connect"
(4) string    "publickey"
(5) boolean   If FALSE key proposal or execution
(6) string    public key algorithm name              ex: rsa-sha2-256
(7) string    public key blob
(8) string    signature                              si execution
```

2 - Réponse de clé acceptée par le serveur

```
    byte      SSH_MSG_USERAUTH_PK_OK
    string    public key algorithm name from the request
    string    public key blob from the request
```

3 - Signature de requête - renvoi de 1 avec signature

signature $= \mathrm{Sign}(ID \| (1) \| (2) \| (3) \| (4) \| (5) \| (6) \| (7))$

Autres méthodes d'authentification

Par mot de passe : `"password"`

- Mot de passe envoyé en clair dans le tunnel chiffré
- Support du changement de mot de passe après expiration
- Interface avec le système de mot de passe du serveur (PAM sous GNU)

Basée sur l'hôte : `"hostbased"`

- Méthode moins sécurisée que les deux premières
- mais meilleure que les méthodes des protocoles en `r` (ex : `rsh`)
- Basé sur la signature d'une requête envoyée au serveur
- Utilisation de l'algorithme de signature des hôtes négocié au début

Après l'authentification

Prochain service

- Le service auquel le client veut accéder est dans la requête
- Une fois l'authentification réussie, le serveur lance le service

Échec de l'authentification

- Le serveur se déconnecte sur échec de l'authentification

Plan

La couche connexion 1/2

Objectifs

- Multiplexer des canaux de communication
- Accès à des sessions interactives
- Relayage de trafic réseau
- Qualité de service par canaux

Services

- Type de canaux par type de service rendu
  - Sessions interactives (shell, programmes, copie de fichiers)
  - Relayage TCP/IP client $\leftrightarrow$ serveur
  - Relayage de protocole X11
- Protocole de gestion de canaux : création / fermeture
- Gestion de fenêtres de transmissions

La couche connexion 2/2

Protocole

- Requêtes globales qui affectent tous les canaux et l'hôte
  - Gestion du relayage réseau
- Messages spécifiques à un canal
  - Ouverture / fermeture de canaux
  - Gestion de fenêtre de transmission
  - Requêtes de canal spécifique, ex : création de pseudo terminal

Requête globale : exemple relayage TCP/IP

```
byte      SSH_MSG_GLOBAL_REQUEST
string    request name in US-ASCII only          ici: "tcpip-forward"
boolean   want reply
....      request-specific data follows
string    address to bind                         ex: "0.0.0.0"
uint32    port number to bind                     ex: 1234
```

Canaux de communication 1/3

Concepts

- Toutes les données utiles sont échangées par des canaux
- Une session interactive ou un relayage par canal
- Associé à un identifiant par sens de communication
- Gestion individuelle des fenêtres de transmission

Protocole : cycle de vie d'un canal

1. Ouverture d'un canal d'un type donné
2. Requêtes spécifiques de configuration du canal
- ∞ Échange des données et gestion des fenêtres
- N Fermeture du canal

Canaux de communication 2/3

Ouverture de canal

- Utilisation du message d'ouverture d'un canal avec un type donné
- Sélection d'un identifiant local unique
- Sélection de la taille max d'un message et de la fenêtre de transmission

Message

```
byte      SSH_MSG_CHANNEL_OPEN
string    channel type in US-ASCII only       ex: "session"
uint32    sender channel                      id côté client
uint32    initial window size                 window côté client
uint32    maximum packet size                 taille max côté client
....      channel type specific data follows
```

```
byte      SSH_MSG_CHANNEL_OPEN_CONFIRMATION
uint32    recipient channel                   id côté serveur
uint32    sender channel                      id côté client
uint32    initial window size                 window côté serveur
uint32    maximum packet size                 taille max côté serveur
....      channel type specific data follows
```

Canaux de communication 3/3

Transfert de données

- Utilisation du message de transfer de données

```
byte      SSH_MSG_CHANNEL_DATA
uint32    recipient channel
string    data
```

```
byte      SSH_MSG_CHANNEL_EXTENDED_DATA
uint32    recipient channel
uint32    data_type_code                    ex: sortie d'erreur
string    data
```

- Consomme taille de `data` sur la fenêtre de transmission. 0 = stop.

Ajustement de la fenêtre de transmission

- Chaque pair peut ajuster les fenêtre de transmission avec ce message

```
byte      SSH_MSG_CHANNEL_WINDOW_ADJUST
uint32    recipient channel
uint32    bytes to add
```

Canaux de session interactive 1/2

Configuration du canal avec des requêtes spécifiques

- Type de canal : `"session"`
- Allocation de pseudo terminal (`/dev/ptsx`) pour shell)

```
byte      SSH_MSG_CHANNEL_REQUEST
uint32    recipient channel
string    "pty-req"
boolean   want_reply
string    TERM environment variable value (e.g., vt100)
uint32    terminal width, characters (e.g., 80)
uint32    terminal height, rows (e.g., 24)
uint32    terminal width, pixels (e.g., 640)
uint32    terminal height, pixels (e.g., 480)
string    encoded terminal modes
```

- Demande de relayage de requêtes X11. Support des cookies X11
- Doit être complété par l'ouverture d'un canal X11 pour le transfert des données X11 relayées.

Canaux de session interactive 2/2

Exécution de programmes finaux

- Transfert de variables d'environnement

```
byte      SSH_MSG_CHANNEL_REQUEST
uint32    recipient channel
string    "env"
boolean   want reply
string    variable name
string    variable value
```

Exécution de programmes finaux

- Requête d'exécution de shell
- Requête d'exécution de programme
- Requête d'exécution de système spécifique (ex : transfert de fichier)

```
byte      SSH_MSG_CHANNEL_REQUEST
uint32    recipient channel
string    "exec"
boolean   want reply
string    command
```

Canaux de relayage de trafic TCP/IP 1/2

Configuration d'un canal de relayage pair distant → pair local

- Type de canal : `"forwarded-tcpip"`

1. Requête globale `"tcpip-forward"`, écoute du distant sur un port
2. Si traffic arrive, pair distant ouvre un canal `"forwarded-tcpip"`

```
byte      SSH_MSG_CHANNEL_OPEN
string    "forwarded-tcpip"
uint32    sender channel
uint32    initial window size
uint32    maximum packet size
string    address that was connected
uint32    port that was connected
string    originator IP address
uint32    originator port
```

- Exemple : redirection inverse de trafic ciblant `serveur:2222` vers la machine du client

Canaux de relayage de trafic TCP/IP 2/2

Configuration d'un canal de relayage pair local → pair distant

- Type de canal : `"direct-tcpip"`
- Pair local distant ouvre un canal `"direct-tcpip"`

```
byte      SSH_MSG_CHANNEL_OPEN
string    "direct-tcpip"
uint32    sender channel
uint32    initial window size
uint32    maximum packet size
string    host to connect
uint32    port to connect
string    originator IP address
uint32    originator port
```

- Exemple : relayage de trafic local `localhost:1234` vers le serveur distant `battle.net:4000` pour jouer à Diablo II à l'ENSEEIHT.

Plan

Utilisations

Shell

```
ssh monlogin@serveur
ssh monlogin@serveur commande
```

Transfert de fichiers : scp

```
scp [-r] [log1@serv1:]fic1 [log2@serv2:]fic2
```

Exemple : `scp -r caguilar@ssh.laas.fr:presentations/ .`
Auto-complétion si authentification par clé publique !

Transfert de fichiers : rsync

```
rsync -avz -delete [log1@serv1:]fic1 [log2@serv2:]fic2
```

Ne transfère que les différences
Exemple : `rsync -avz -delete boulot/ ssh.laas.fr:boulot/`

Plan

Redirection client → serveur

Principe

Un port TCP du client est redirigé vers un port TCP d'une autre machine *via* le serveur SSH

Commande

```
ssh [-g] -L localPort:destHost:destPort user@sshRelay
```

Fonctionnement

1. Une connexion SSH est créée entre le client et `sshRelay`
2. Les paquets arrivant au port `localPort` du client sont envoyés à `sshRelay` par la connexion SSH (chiffrement+MAC)
3. `sshRelay` envoie ces paquets vers `destHost:destPort` (en clair)

Si -g n'est pas précisé écoute sur localPort en loopback seulement

Redirection client → serveur : utilisation (1/2)

Sécurisation de services non fiables

Mettre en place dans un hôte un serveur ssh et un service non sécurisé accessible seulement en local.

Exemple pour un service web :

```
ssh -L 8080:localhost:80 user@securedServer
http://localhost:8080
```

⇒ restriction d'accès aux personnes avec un compte ssh !

Routage à la source sécurisé

Le certificat de votre banque/serveur ssh/etc à changé !

```
ssh -L 8080:serveurDouteux:80 user@sshServer
http://localhost:8080
```

Le certificat va être demandé depuis votre labo/entreprise/farm

Redirection client → serveur : utilisation (2/2)

Sécuriser notre trafic

Si on a un AP non sécurisé

```
ssh -L 5223:talk.google.com:5223 login@sshServer
```

Traverser (légitimement ?) un firewall (en sortie)

Situation idéale :
Serveur ssh dans une DMZ
Client dans un réseau fermé en sortie (pas en 22)

```
ssh -L 1234:serveurQcq:portDest monCompte@sshServer
```

Sortie seulement en 80 ? Ok `/etc/ssh/sshd.conf Port = 80`

Traverser (légitimement ?) un firewall (en entrée)

Proxy ssh ou réseau ouvert en entrée 22 (ou un autre port)

```
ssh -L 1234:serveurSMTP:portSMTP monCompte@sshServer
```

Redirection serveur → client

Principe

Un port TCP du serveur SSH est redirigé vers un port TCP d'une autre machine *via* le client

Commande

```
ssh -R sshServerPort:destHost:destPort user@sshServer
```

Fonctionnement

1. Une connexion SSH est créée entre le client et `sshServer`
2. Les paquets arrivant au port `sshServerPort` du serveur ssh sont envoyés au client par la connexion SSH (chiffrement+MAC)
3. Le client envoie ces paquets vers `destHost:destPort` (en clair)

Serveur écoute en loopback seuelement sauf si `GatewayPorts=true`

Redirection serveur → client : utilisation (1/2)

Avoir un point fixe d'accès

Créer un retro-tunnel sur un serveur ssh d'où qu'on soit pour toujours pouvoir accéder à notre machine)

```
ssh -R 2222:localhost:22 login@sshServer
```

Traverser illégitimement un firewall en entrée

Réseau complètement fermé en entrée, sortie sur 22 (ou autre)

Accéder à un poste par ssh

Depuis `sshIntServer` :

```
ssh -R 2222:localhost:22 loginExt@sshExtServer
```

Depuis l'extérieur :

```
ssh loginExt@sshExtServer
ssh -p 2222 loginInt@localhost
```

Si `GatewayPorts=true` sur `sshExtServer` on peut directement faire

```
ssh -p 2222 loginInt@sshExtServer
```

Redirection serveur → client : utilisation (2/2)

Traverser illégitimement un firewall en entrée (suite)

Réseau complètement fermé en entrée, sortie sur 22 (ou autre)
Par exemple un réseau univ. (ou le home network d'un ami !)

Accéder à un service quelconque
Depuis `sshIntClient` :
`ssh -R 2222:smtpIntServer:25 loginExt@sshExtServer`
Depuis l'extérieur :
Si votre ordi est `sshExServer` configurer smtp avec `localhost:2222`
Si c'est un autre vous pouvez

- Activer `GatewayPorts` sur `sshExtServer` et configurer smtp avec `sshExtServer:2222` (pas bon du tout !)
- Forwarder dans votre ordi un port local (par ex. 3333) vers `sshExtServer:2222` par ssh
  `ssh -L 3333:localhost:2222 loginExt@sshExtServer`
  et configurer smtp sur notre ordi avec `localhost:3333`

On peut faire du netcat aussi mais ça ne sera pas chiffré !

Redirection par SOCKS (1/2)

Limitations du port forwarding statique

Je peux ...
créer un tunnel de `8080` vers `intran7:80`
créer un tunnel de `8081` vers `grr:80`
... Je veux naviguer sur le web en faisant croire que je suis à l'N7
Une redirection statique pour chaque serveur web que je veux contacter ?

Le protocole SOCKS

- Protocole sécurisé permettant de faire un proxy TCP/UDP
- Fait pour avoir un point de sortie authentifié pour les firewalls
- Un utilisateur (authentifié) demande de créer une connexion vers un serveur donné sur un port donné et le serveur initialise la connexion et relaie les informations entre client et serveur
- Autant de connexions en parallèle qu'on veut

Il faut parler le SOCKS au niveau client (navigateurs, gest. courrier le font)

Redirection par SOCKS (2/2)

Utilisation de ssh

```
ssh [-g] -D 8080 login@sshServer
```

- Crée un proxy SOCKSv5 sur sshServer
- Forwarde votre port local 8080 sur le proxy

Configuration Firefox :
Édition→Préférences→Avancé→Réseau→Paramètres

À chaque fois que vous demandez une page sur un nouveau serveur :

- Firefox demande au proxy SOCKS de créer une connexion TCP sur le port 80 du serveur
- Un canal SOCKS est crée entre client et proxy pour cette connexion
- Firefox envoie dans ce canal toutes les requêtes pour le serveur web
- Le serveur redirige ces requêtes en remplaçant les en-têtes Eth/IP/TCP

Redirection sécurisé de flux X sur ssh (1/3)

Comment faire ?

```
monClient:~$ ssh -X login@sshServer
sshServer:~$ firefox
```

Comment ça marche ? (idée)

- Crée un deuxième cookie xauth (temporaire) au niveau du client
- L'envoie sur le serveur par ssh et le défini comme `~/.Xauthority`
- Redirige le port 6000+n du serveur vers le port 6000 du client
- Définit sur le shell distant `DISPLAY=localhost:n`
- Fait un swap des cookies en live lors des connexions (compl. à l'oral)

Ça marche pas ...

Vérifier qu'il y a pas l'option `-nolisten tcp` dans `xserverrc`

Redirection sécurisé de flux X sur ssh (2/3)

Problèmes de sécurité

Aucune attaque réseau connue
Si un attaquant peut accéder à `~/.Xauthority` ...
Il peut se connecter sur le port 6000+n avec le bon cookie !

Exemples de méfaits

- Screenshot du bureau du client :
  `xwd -display localhost:n -root -out test.xwd`
- Voir les clients X : `xlsclients` (pas que sur le serveur)
- Surveiller les événements d'E/S : `xev`
- Faire un keylogger : `DISPLAY=localhost:10 xmacrorec2 -k 0`

Redirection sécurisé de flux X sur ssh (3/3)

L'option -Y : ForwardX11Trusted

Normalement les attaques précédentes n'auraient pas dû marcher avec -X
-Y est l'option permettant aux clients distants d'avoir un accès total
-X devrait placer les clients distants dans un groupe "untrusted"

Mais l'extension X Security dont dépendait ce système a été abandonnée ...
XACE est la nouvelle extension de sécurité qui pourrait permettre de résoudre ces problèmes !
Mais elle ne sera pas opérationnelle pendant un moment donc attention !

Plan

Fin !