Plateforme de fingerprinting pour SSH

Mohamed MZIOU

Département RST (Réseaux et Services de Télécommunications) Télécom SudParis

Tuteur Télécom SudParis : Olivier LEVILLAIN
Tuteur ENSTA Paris : Françoise LEVY-DIT-VEHEL

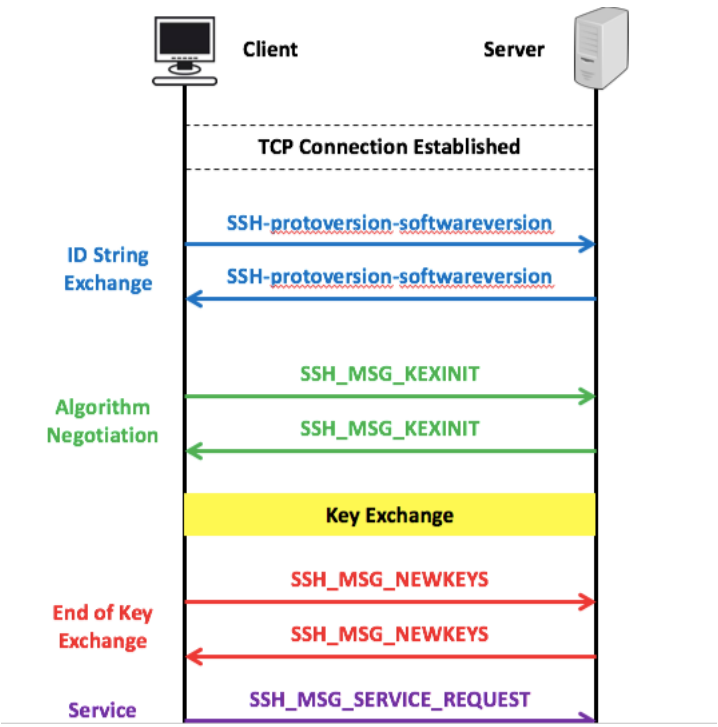
25 Août 2025

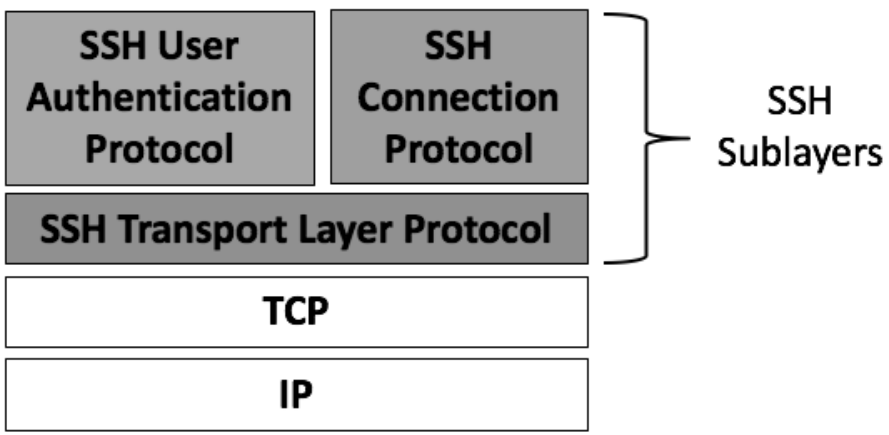
Plan

Plan

Le protocole SSH

Secure Shell (SSH) est un protocole de communication sécurisé. Il impose un **échange de clés** de chiffrement en début de connexion. Par la suite, tous les échanges sont **authentifiés** et **chiffrés**.





Le *Fingerprinting*, c'est quoi ?

Une **empreinte numérique** est une donnée de petite taille qui permet d'identifier de manière unique une donnée ou un système plus large.

- **Couche des informations publiées** : principalement la bannière, version du protocole...
- **Couche de configuration** : capacités cryptographiques, algorithmes supportés, fonctionnalités...
- **Couche comportementale** : différences observables dans le comportement des machines d'état selon les piles logicielles.

Objectifs

- Étudier les **déploiements SSH** dans des environnements réels
  - Développer un **banc de tests** pour les principales implémentations SSH **open source**.
  - Observer et analyser leur déploiement **réel** et **opérationnel**.
  - Identifier les **spécificités** et **particularités** de chaque implémentation.
- Étudier les méthodes de **fingerprinting** existantes
  - Déterminer ce qui peut être déduit à partir des **informations publiées**.
  - **Recenser** et **tester** les outils disponibles en ligne.
  - Identifier les **limites** et **points faibles** de chaque méthode.
- Développer une nouvelle méthode basée sur le **comportement**
  - Réaliser un **prototype**.
  - Effectuer des **tests préliminaires**.

Plan

L'environnement Docker

- Docker crée et exécute des applications dans des conteneurs **isolés**.
- Les conteneurs sont **légers**, **portables** et **autonomes**.
- Dans notre contexte, il permet de **déployer** rapidement plusieurs piles SSH isolées.
- Facilite des expériences **reproductibles** sans toucher au système hôte.

SSH Test Bed en quelques chiffres

Impl.	**Versions**	**Dates**	**#Stacks**
OpenSSH	4.4-10.0	2006-2025	63
Dropbear	v0.44–v2025.88	2005–2025	48
WolfSSH	1.4.13–1.4.20	2023–2025	8
libssh	0.8.3–0.11.2	2018–2025	25
AsyncSSH	2.0–2.21	2019–2025	34

Plan

L'*Identification String*

Lorsque la connexion est établie, les deux parties DOIVENT envoyer une **chaîne d'identification**. Cette chaîne DOIT être de la forme :

`SSH-protoversion-softwareversion <SP> commentaires <CR> <LF>`

Exemple :

`SSH-2.0-OpenSSH_9.6p1<SP>Ubuntu-3ubuntu13.11<CR><LF>`

Elle garantit la **compatibilité du protocole**, participe à l'**échange de clés Diffie-Hellman** et marque le début de l'**échange cryptographique**

Identification ≠ Empreinte fiable

- L'*Identification String* indique ce que la pile **prétend être**, plutôt que ce qu'**elle est réellement**.
- C'est un simple message **non chiffré**, envoyé en **texte clair**.
- Il est trivial pour une implémentation donnée de se faire passer pour une autre.
- Encore plus trivial pour un tiers inconnu, il suffit d'envoyer un message à travers un socket brut pour qu'un client SSH le reconnaisse comme valide.

Plan

Recensement des serveurs SSH via scan réseau

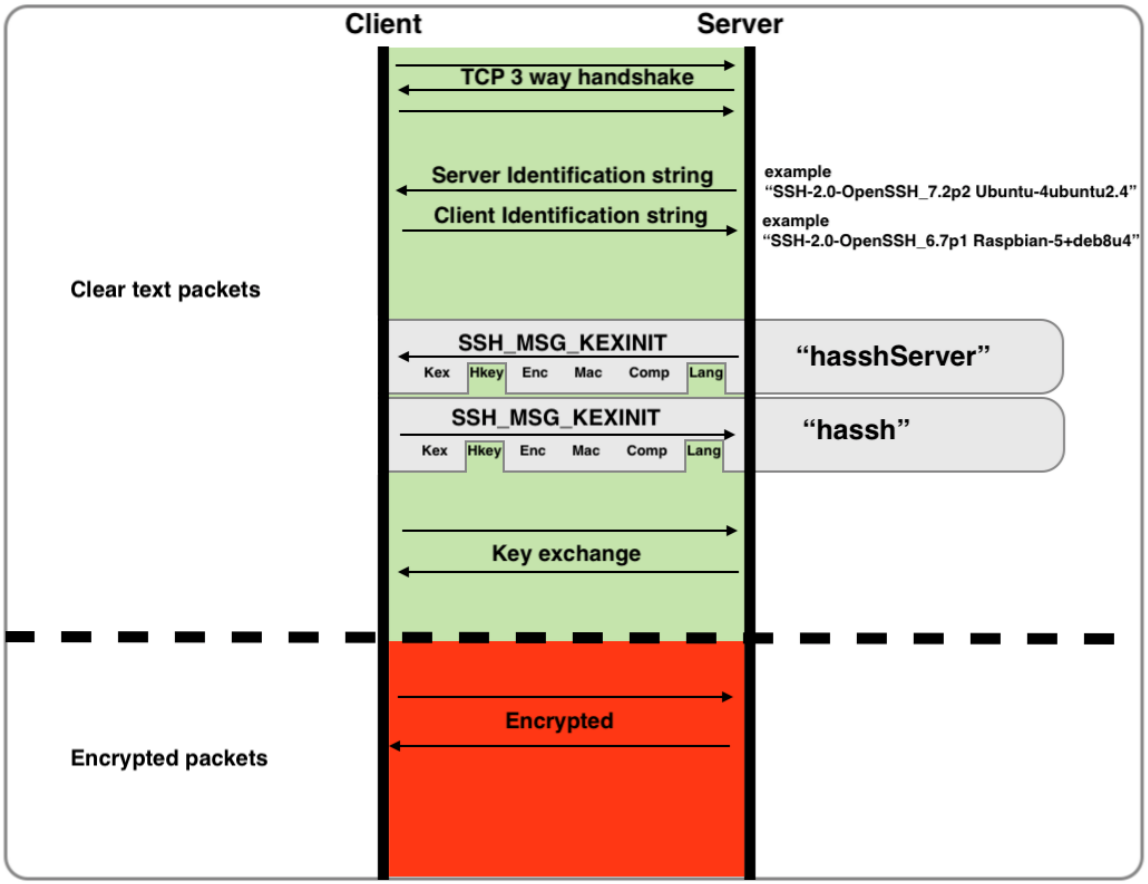
- Une approche simple : énumération des configurations grâce à des outils comme Nmap.
- Nmap (*Network Mapper*) : est un outil open-source de découverte réseau et d'audit de sécurité, pas un outil de *fingerprinting*.
- Le flag `-sV` permet de "détecter" la pile, mais ne fait que relayer l'*Identification String*.
- Possibilité d'utiliser les scripts Nmap pour extraire des informations détaillées.

```
sinda@sinda-pc:~$ nmap -sV --script ssh2-enum-algos.nse -p 2222 localhost
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-08-17 03:14 CEST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000071s latency).

PORT     STATE SERVICE VERSION
2222/tcp open  ssh     OpenSSH 10.0 (protocol 2.0)
| ssh2-enum-algos:
|   kex_algorithms: (10)
|       mlkem768x25519-sha256
|       sntrup761x25519-sha512
|       sntrup761x25519-sha512@openssh.com
|       curve25519-sha256
|       curve25519-sha256@libssh.org
|       ecdh-sha2-nistp256
|       ecdh-sha2-nistp384
|       ecdh-sha2-nistp521
|       ext-info-s
|       kex-strict-s-v00@openssh.com
|   server_host_key_algorithms: (4)
|       rsa-sha2-512
|       rsa-sha2-256
|       ecdsa-sha2-nistp256
|       ssh-ed25519
|   encryption_algorithms: (6)
|       chacha20-poly1305@openssh.com
|       aes128-gcm@openssh.com
```

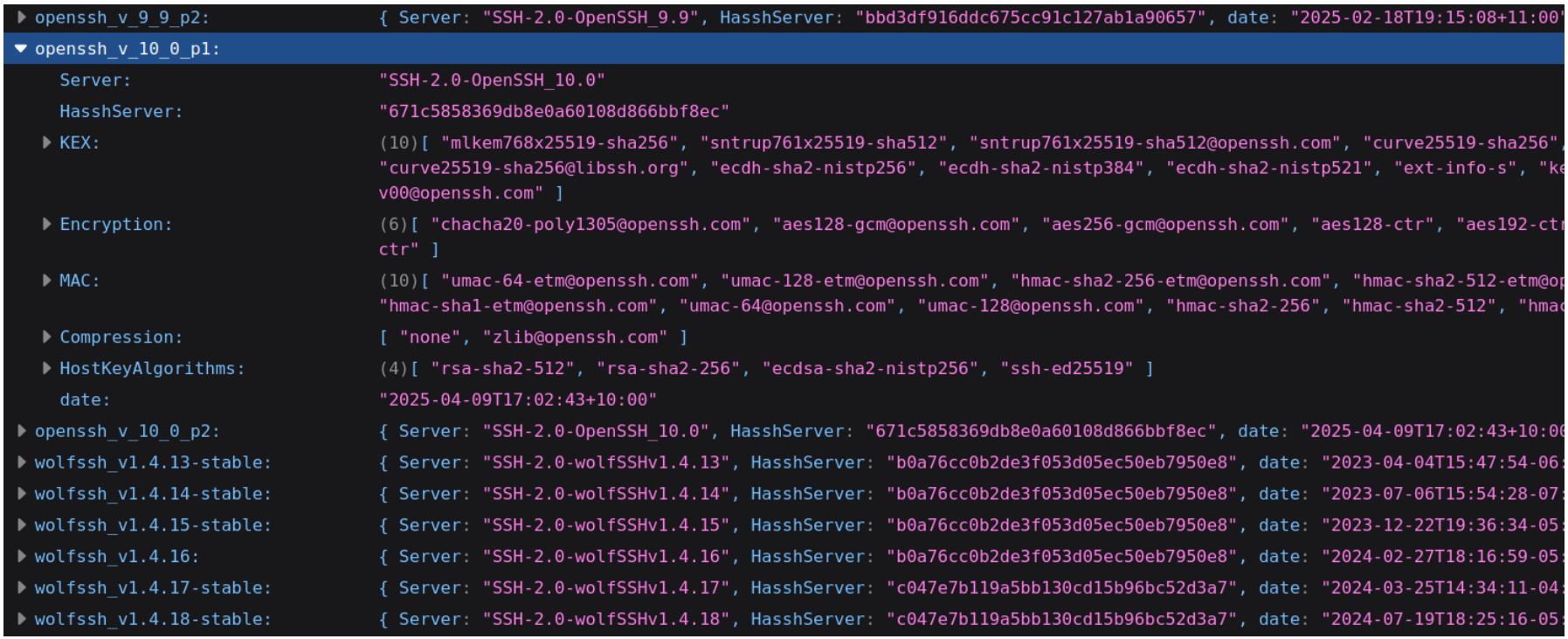
HASSH

- HASSH est une méthode open-source de fingerprinting Développée par Salesforce pour identifier les clients et serveurs SSH.
- Utilise un hash MD5 des algorithmes dans `SSH_MSG_KEXINIT`.



Résultats de HASSH sur SSH Test Bed

Sur les configurations par défaut de chaque implémentation, HASSH fonctionne globalement bien pour identifier nos piles serveur.



Limitation de HASSH

- Montrer que **deux implémentations SSH complètement différentes** peuvent avoir **la même empreinte HASSH**.
- **Étape intermédiaire** : énumérer tous les algorithmes supportés par nos implémentations, comparer avec la config par défaut.
- **Résultat** : expérience réalisable entre une version d'OpenSSH et une version de libssh.

Algorithmes Partagés `OpenSSH7.6p1` et `libssh0.8.3`

Type	Algorithmes
Kex	`diffie-hellman-group14-sha256`, ...
Clé Hôte	`ssh-rsa`, `rsa-sha2-512`, ...
Chiffrement	`aes128-ctr`, `aes256-gcm@openssh.com`, ...
MAC	`hmac-sha2-256`, `hmac-sha2-512`, ...

- **Exercice bonus** : un tiers inconnu peut envoyer des messages via un **socket brut**, se faire reconnaître comme une pile SSH valide et même partager la **même empreinte HASSH**.

Plan

Historique

- OS fingerprinting actif : envoyer des paquets et observer les réponses pour détecter des différences TCP/IP.
- Plus pertinent pour notre projet : inférence de machines à états pour étudier les piles TLS face à des séquences inhabituelles.
- Principe : traiter la pile comme une **boîte noire** et observer son comportement pour distinguer les implémentations.

Application dans SSH

- Les implémentations SSH ont de subtiles différences de comportement.
- Envoyer un **message inattendu** peut provoquer des réponses intéressantes.
  - Ex. : CVE-2018-10933, certaines versions de libssh acceptent `USERAUTH_SUCCESS` à la place de `USERAUTH_REQUEST`.
- Envoyer des **paquets classiques** dans un ordre inattendu peut révéler des comportements différents.

Proof of Concept

- Création de messages SSH classiques :
  - **Authentification et clés** : *kexinit*, *newkeys*, *userauth_request_none*, *userauth_request_password*, *userauth_request_publickey*
  - **Messages système** : *disconnect*, *unimplemented*
  - **Transitions de sous-protocoles** : *service_request_ssh-userauth*, *service_request_ssh-connection*, *service_request_autre*
  - **Spécifique au serveur** : *userauth_success*, *userauth_failure*, *userauth_banner*
- Construction d'un **graphe partiel de la machine à états** pour visualiser les transitions et réactions de la pile SSH.
- Génération d'une **empreinte unique** à partir de l'encodage des transitions, obtenue avec un **hash SHA-256**.

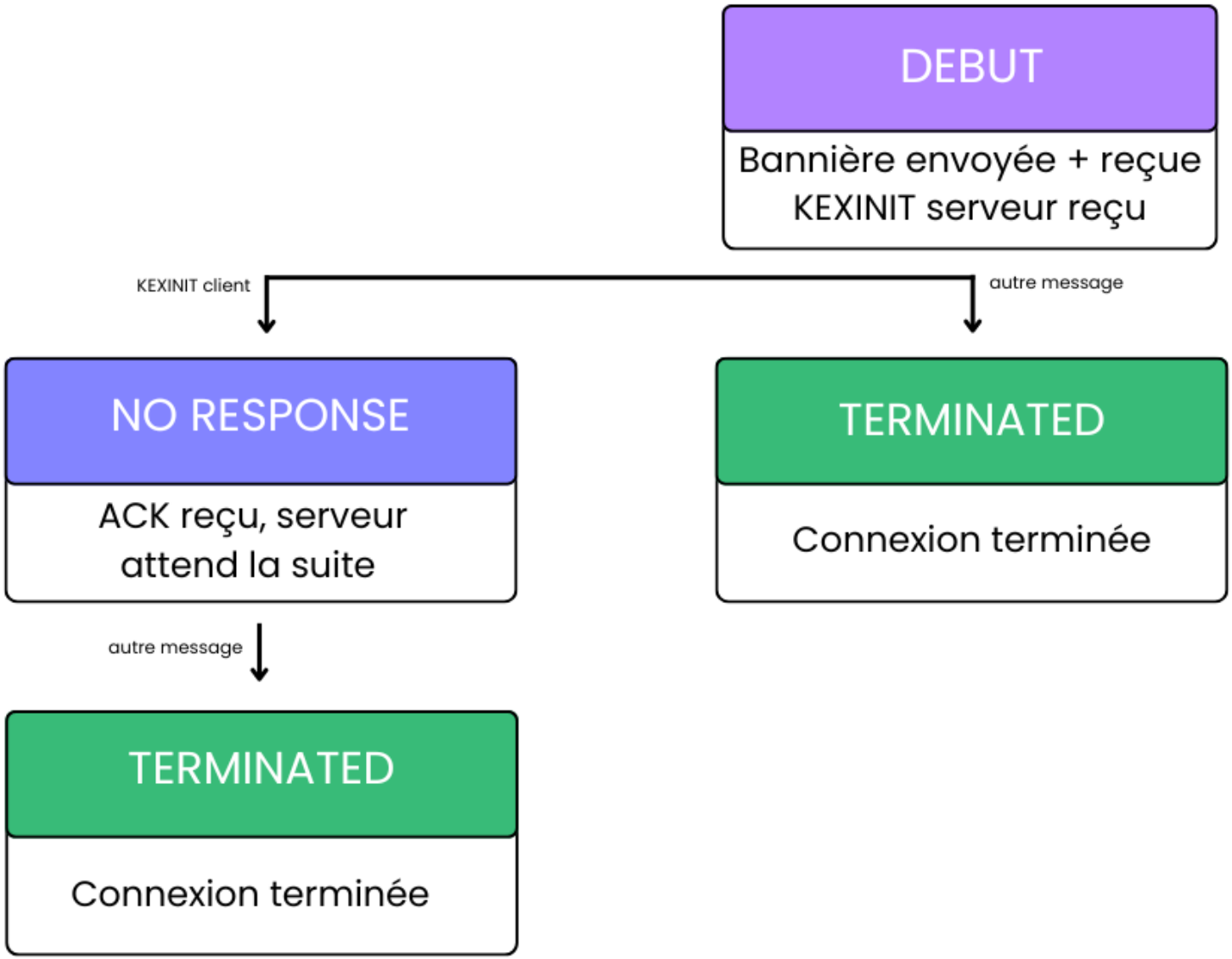
Résultats préliminaires

- Chaque implémentation de pile SSH possède une empreinte distincte.

Library	Fingerprint
WolfSSH	v1.4.13–16 : bceabb... v1.4.14 : 8b6149...
Dropbear	0.47–0.53.1 : 442c11... 2014.65–2025.88 : c38067...
OpenSSH	6.8–8.9 : 026033... 9.6–10.0 : 93f679...

- Les versions proches d'une même implémentation présentent peu de variations, **ce qui est logique**.
- Malgré cela, la méthode reste plus robuste que les approches existantes.

Exemple pour Dropbear 0.47–0.53.1



Développements futurs

- Améliorer la lisibilité et le traitement des graphes générés.
- Optimiser le traitement de multiples séquences simultanément pour gagner du temps (avec le design actuel, une pile prend environ 5min30).
- Ajouter un **apprenant automatique** pour inférer le comportement des piles, similaire au travail fait sur TLS avec l'algorithme L*.
- Évaluer la robustesse de cette méthode et déterminer comment un acteur malveillant pourrait falsifier l'empreinte d'une implémentation SSH donnée.

Bibliographie

MedMzi. *ssh-test-bed*. GitHub. `https://github.com/MedMzi/ssh-test-bed`, original date : 2025-06-12.

corelight. *hassh*. GitHub. `https://github.com/corelight/hassh`, April 2025, original date : 2024-01-29.

Ben Reardon. Open Sourcing HASSH, September 2018.

LIBSSH Auth Bypass (CVE-2018-10933).

Chris M. Lonvick and Tatu Ylonen. The Secure Shell (SSH) Protocol Architecture. RFC 4251, January 2006.

Chris M. Lonvick and Tatu Ylonen. The Secure Shell (SSH) Transport Layer Protocol. RFC 4253, January 2006.

Chris M. Lonvick and Tatu Ylonen. The Secure Shell (SSH) Authentication Protocol. RFC 4252, January 2006.

Chris M. Lonvick and Tatu Ylonen. The Secure Shell (SSH) Connection Protocol. RFC 4254, January 2006.

What is Docker ?, September 2024.

Aina Toky Rasoamanana, Olivier Levillain, Hervé Debar. Towards a systematic and automatic use of state machine inference to uncover security flaws and fingerprint TLS stacks. ESORICS 2022, LNCS 13556, pages 637–657, Copenhagen, September 2022. Springer Nature Switzerland.

Mrinal Prakash. Enumerating SSH with Nmap, April 2023.

Secure Shell (SSH) Protocol Parameters.
`https://www.iana.org/assignments/ssh-parameters/ssh-parameters.xhtml`.

Expect Script SSH Example Tutorial — DigitalOcean.

Merci pour votre attention !

Contribution :
- banc de tests pour les implémentations SSH
- analyse des méthodes de fingerprinting
- développement d'une approche comportementale