Engineer

SSH

un shell chiffré sur une autre machine

Ce que c’est

SSH, abréviation de Secure Shell, est un protocole réseau qui permet de se connecter à un autre ordinateur et d’y exécuter des commandes à travers un réseau auquel personne ne fait confiance, avec un trafic chiffré et une vérification préalable de l’identité des deux extrémités. Le nom recouvre trois choses : le protocole, publié par l’IETF en janvier 2006 sous la forme d’un ensemble de RFC articulé autour de la RFC 4251 ; OpenSSH, l’implémentation maintenue par le projet OpenBSD et utilisée dans les exemples de cet article ; et la commande `ssh`, qui communique avec un démon serveur nommé `sshd`.

La RFC 4251 décrit trois couches sur une seule connexion, généralement au port TCP 22, le numéro attribué par l’IANA. La couche transport réalise l’échange de clés, prouve l’identité du serveur au moyen de sa clé d’hôte, puis chiffre et vérifie l’intégrité de tout ce qui transite. La couche d’authentification de l’utilisateur fonctionne dans ce tunnel et établit l’identité du client : par clé publique, méthode que chaque implémentation doit prendre en charge, par mot de passe, méthode facultative, ou par confiance fondée sur l’hôte. La couche connexion divise le tunnel en canaux : une seule connexion authentifiée peut ainsi transporter à la fois un shell interactif, une commande isolée, un transfert de fichiers SFTP et plusieurs redirections de ports.

SSH a remplacé telnet et rlogin, qui envoyaient les sessions et les mots de passe en clair sur le réseau. La version du protocole utilisée aujourd’hui est la version 2.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce que SSH ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. Des comptes, mots de passe et droits attribués selon les besoins exacts de chaque poste, puis retirés le jour du départ.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. 1 000 règles RBAC pour sécuriser Linux — 1,000 règles d'accès écrites pour combler une faille de sécurité

1 000 règles RBAC pour sécuriser Linux Contactez-nous →

Pourquoi c’est important

Sur un serveur, SSH est la porte d’entrée. Une machine dans un centre de données ou dans le cloud a rarement un clavier à portée de main : administration, déploiements, copies de fichiers et configuration automatisée passent donc par SSH. Quiconque s’y connecte avec assez de privilèges contrôle la machine.

L’authentification par clé publique rend SSH à la fois pratique et robuste. La clé privée reste sur la machine de l’utilisateur, et la clé publique correspondante est placée sur le serveur, par défaut dans `~/.ssh/authorized_keys` . À la connexion, le client signe des données liées à cette session : la clé elle-même ne traverse donc jamais le réseau. Un mot de passe volé peut être réutilisé de n’importe où, et toute personne pouvant atteindre le port d’un serveur qui accepte les mots de passe peut tenter de les deviner. C’est précisément ce que font les outils automatisés. OpenSSH accepte les mots de passe dès son installation : les désactiver relève donc d’une décision propre à chaque serveur.

La confiance fonctionne aussi dans l’autre sens : le client doit savoir qu’il a atteint le bon serveur. Chaque serveur possède une clé d’hôte, et la RFC 4251 propose deux manières de lui faire confiance : une liste locale sur le client associant chaque nom d’hôte à sa clé, ou une autorité de certification qui signe les clés d’hôte, le client ne conservant alors que la clé de cette autorité. La première méthode est la plus courante, dans sa forme la moins stricte, que la RFC accepte comme concession à la facilité d’utilisation : accepter la clé sans vérification au premier contact, l’enregistrer dans `~/.ssh/known_hosts` , puis la comparer à chaque connexion ultérieure. C’est ce qu’on appelle généralement la confiance au premier usage. La sûreté de cette première connexion dépend entièrement du réseau traversé : voilà pourquoi OpenSSH demande confirmation avant d’enregistrer une nouvelle clé et refuse catégoriquement une clé différente présentée par un hôte connu.

SSH est aussi un transport sur lequel s’appuient d’autres outils. Les envois et récupérations Git passent par lui, `scp` et `sftp` l’utilisent pour copier des fichiers, Ansible atteint ses hôtes par ce canal et la redirection de ports permet à un portable d’accéder à une base de données qui n’écoute que sur l’interface de boucle locale d’un serveur, sans l’ouvrir à Internet.

À quoi cela ressemble en pratique

L'hôte de l'entreprise accepte les connexions interactives uniquement par SSH, et son durcissement est contrôlé en trois étapes à chaque exécution du rôle d'automatisation qui gère la configuration. Ces contrôles datent de septembre 2026 : un rapport d'accès avait comparé les paramètres effectivement retenus par le démon à ceux écrits par le rôle, et deux différaient. Les valeurs par défaut de la distribution figuraient dans des lignes actives lues avant le fichier complémentaire du rôle, et sshd gardait la première valeur. Renommer ce fichier pour qu'il soit classé avant celui du fournisseur a comblé l'écart. Désormais, une configuration SSH doit franchir trois contrôles : la configuration candidate doit être syntaxiquement valide avant sa mise en place, l'ensemble des fichiers assemblés est vérifié après installation et les paramètres effectivement retenus par le démon sont comparés à 24 directives attendues. Une valeur supplantée par une ligne antérieure arrête donc l'exécution. Deux directives sont délibérément exclues : le démon actuel ne les prend plus en charge.

Un mois plus tôt, une revue menée en août 2026 avait découvert trois protections contre la force brute qui n'avaient jamais fonctionné depuis la création de l'hôte. La jail de bannissement surveillait le port 22 alors que le démon écoutait sur 1986. La limitation de débit du pare-feu venait après une règle plus large autorisant le même trafic : la première règle correspondante l'emportant, la limite ne voyait jamais de paquet, bien que la sortie d'état continue de l'afficher. L'action de bannissement dépendait d'un outil de filtrage de paquets que le gestionnaire de paquets ne faisait que recommander : la jail repérait donc les attaquants, puis produisait une erreur à chaque tentative de blocage. La revue a laissé deux règles durables : chaque mesure de sécurité est accompagnée d'un contrôle prouvant qu'elle agit, et une règle de pare-feu est vérifiée par sa position autant que par sa présence.

La forge Git de l'entreprise utilise SSH comme transport, pas comme moyen d'ouvrir une session. Elle a été choisie parce qu'elle se pilote par SSH et n'a aucun panneau d'administration web à défendre. Personne n'y accède anonymement ni sans clé, et chaque dépôt est privé. Son point d'écoute SSH n'accepte des connexions que sur la boucle locale, au port 23231. Les personnes extérieures y accèdent en passant par un hôte de rebond : le pare-feu n'ouvre donc aucun port supplémentaire pour elle.

Où les problèmes surviennent

Une directive de durcissement figure dans le fichier et le serveur l'ignore. Pour la plupart des mots-clés, `sshd` conserve la première valeur lue, et une directive `Include` lit les fichiers qu'elle inclut dans l'ordre lexicographique. Si le fichier principal commence par inclure un dossier de fichiers complémentaires, un fichier du fournisseur classé avant celui ajouté localement prend le dessus, alors que les deux paraissent corrects. Une recherche dans la configuration prouve seulement qu'une ligne a été écrite. Interroger plutôt le démon : `sshd -t` vérifie la syntaxe et `sshd -T` affiche les paramètres qu'il utilisera réellement.

```
/etc/ssh/sshd_config.d/10-vendor.conf:     PasswordAuthentication yes
/etc/ssh/sshd_config.d/50-hardening.conf:  PasswordAuthentication no

$ sudo sshd -T | grep -i '^passwordauthentication'
passwordauthentication yes
```

Une clé d'hôte modifiée est supprimée puis acceptée. Lorsqu'un serveur connu présente une autre clé d'hôte, `ssh` affiche un avertissement et refuse la connexion. Le réflexe consiste à lancer `ssh-keygen -R` sur le nom d'hôte, à se reconnecter et à répondre yes : c'est tout ce qu'il faut à un intermédiaire malveillant. Confirmer d'abord la nouvelle empreinte par un autre canal, par exemple avec `ssh-keygen -lf` sur la clé publique d'hôte du serveur depuis sa console. Dans les scripts, `StrictHostKeyChecking no` est pire qu'il n'y paraît : outre l'enregistrement silencieux des nouvelles clés, ce réglage autorise, dans certaines limites, les connexions aux hôtes dont la clé a changé. Le réglage `accept-new` enregistre les hôtes inconnus tout en refusant les clés modifiées.

Les connexions par mot de passe restent ouvertes à tout Internet. `PasswordAuthentication` vaut yes par défaut, tout comme l'authentification interactive au clavier, qui peut demander le même mot de passe par une autre voie. Dès que chaque utilisateur possède une clé, régler les deux sur no. `PermitRootLogin` vaut déjà `prohibit-password` par défaut : les mots de passe sont refusés pour root, mais une clé de root reste admise. Déplacer le démon hors du port 22 change son point d'écoute, pas les personnes qui peuvent tenter de deviner les accès. Chaque protection qui nomme le port doit le suivre, comme le montre la jail ci-dessus.

La redirection d'agent prête les clés au serveur. Avec `ForwardAgent` activé, ou `ssh -A`, toute personne capable de contourner les permissions de fichiers sur l'hôte distant, root compris, peut utiliser l'agent pour s'authentifier ailleurs sous l'identité du propriétaire de l'agent pendant que la connexion reste ouverte. Elle ne peut pas copier la clé, mais peut signer avec elle. Pour atteindre une machine derrière un bastion, utiliser plutôt un hôte de rebond : `ssh -J bastion.example.com app.internal` redirige une connexion TCP à travers le bastion et négocie la session de bout en bout avec la cible. Le bastion relaie ainsi des octets qu'il ne peut pas lire, et la clé reste sur le portable.

Pour aller plus loin

- RFC 4251 : The Secure Shell (SSH) Protocol Architecture — les trois couches, les deux modèles de confiance dans les clés d'hôte et les considérations de sécurité.
- RFC 4253 : The Secure Shell (SSH) Transport Layer Protocol — l'échange de clés, la vérification de la clé d'hôte et la chaîne de version que chaque extrémité envoie d'abord.
- RFC 4252 : The Secure Shell (SSH) Authentication Protocol — le fonctionnement sur le réseau des authentifications par clé publique, mot de passe et confiance fondée sur l'hôte.
- sshd_config(5) — chaque mot-clé du serveur avec sa valeur par défaut, et la règle qui retient la première valeur obtenue.
- ssh_config(5) — le côté client, notamment StrictHostKeyChecking, ProxyJump et ForwardAgent.
- sshd(8) — les modes de test `-t` et `-T`, ainsi que les formats authorized_keys et known_hosts.

Glossaire

Choisissez selon vos contraintes réelles

Présentez le parcours de déploiement, les risques opérationnels et les personnes d'astreinte. Nous pouvons définir un changement avec des vérifications, une procédure de reprise et une responsabilité claire.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu'est-ce que SSH ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

welcome@engineer.company +45 50 10 61 84 Copenhague, Danemark

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/ssh/