Sécurité - CVE 2018–10933

Cyril Dussert - Emilien Mottet

`https://www.cvedetails.com/cve/CVE-2018-10933/`

28 novembre 2018

1 Service compromis

La faille de sécurité etudiée (CVE 2018–10933) concerne la librairie `libssh` côté serveur dans les versions inférieures à 0.7.6 et 0.8.4. Néanmoins, des systèmes Linux embarquent la librairie dans leurs dépôts. Ces systèmes deviennent ainsi aussi vulnérables. Il s'agit d'Ubuntu pour ses versions LTS (14.04, 16.04 et 18.04), ainsi que Debian Stretch (9.0). Le risque est un contournement de la politique de sécurité.

Le type de cette faille est principalement `Authentication Issues`.

Cette faille de sécurité affecte les applications qui utilisent `libssh` pour implémenter un serveur SSH. Le client SSH n'est pas concerné. Cette faille ne concerne pas `libssh2` ni `openssh`.

2 Description de la vulnérabilité

Cette faille permet une authentification non autorisée sur un serveur. Au cours de l'échange lors de l'authentification d'un client aurpès d'un serveur, la simple présentation d'un message `SSH2_MSG_USERAUTH_SUCCESS` de la part du client, à la place d'un `SSH2_MSG_USERAUTH_REQUEST` (normalement attendu). permet à l'attaquant de s'authentifier correctement sans aucun besoin d'identifiants.

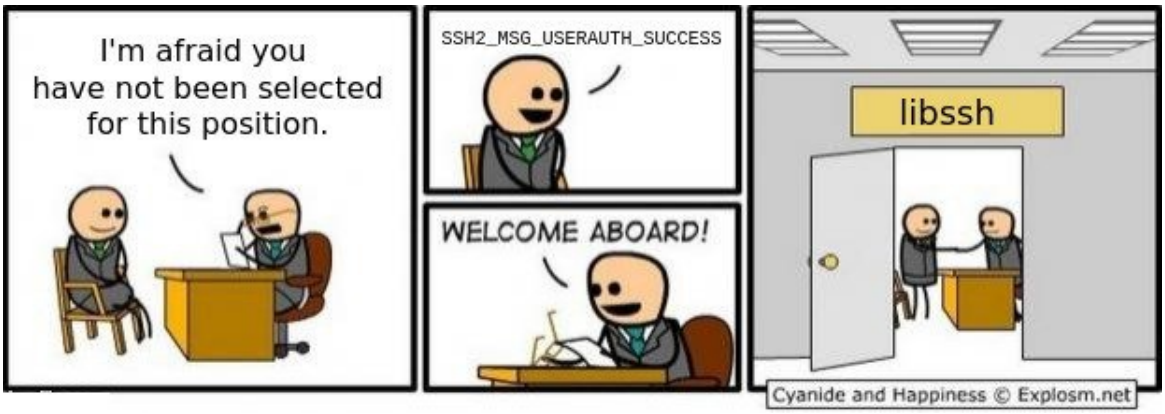

Figure 1 – CVE-2018-10933 in real world

3 Cibles de la faille

Comme décrit ci-dessus, les cibles concernées par cette faille sont les serveurs utilisant la librairie `libssh` comme serveur SSH.

3.1 Versions concernées

— Versions supérieures à 0.6
— Version inférieures à 0.7.6
— Version inférieures à 0.8.4

3.2 Trouver des victimes

Grâce à shodan (google du hacker) nous pouvons trouver une liste de serveur utilisant une librairie potentiellement. Screenshot disponible dans le README.md
`https://www.shodan.io/search?query=product%3A%22libssh%22+version%3A%220.6.3%22`

4 Exemple d'architecture

La dangerosité de cette faille est qu'il n'y a pas besoin d'une architecture spécifique pour l'exploiter. Il suffit de voir les distributions vulnérables debian et ubuntu surtout qu'il existe un grand nombre de distributions basées sur ces distributions. Il suffit d'un serveur, accessible sur un réseau (par exemple, internet), utilisant `libssh` pour son serveur SSH et d'un client malicieux.

Une architecture faillible peut être un serveur avec un service sshd sur un port exposé. Et que son serveur ssh utilise une librairie libssh vulnérable.

5 Préconisations de sécurité

Etant donné qu'il n'y à pas de workaround directement accessible pour palier cette faille, la meilleure solution pour s'en protéger est de mettre à jour **régulièrement** les librairies présentes sur le serveur. Peu de temps après la découverte de la faille, un patch correctif (versions 0.7.6 et 0.8.4) était disponible.

Un bon moyen de suivre les différents problèmes de sécurité importants est de regarder régulièrement les bugs les plus critiques déclarés par les différents systèmes d'exploitation (par exemple, Ubuntu security notices : `https://usn.ubuntu.com/`). Pour le cas de debian (`https://www.debian.org/security/`), il est une très bonne pratique d'ajouter le dépot debian-security. Avec cette ligne à ajouter dans votre `/etc/apt/sources.list`

```
deb http://security.debian.org/debian-security stretch/updates main contrib non-free
```

Il va de soi que le mieux est d'avoir une infrastructure homogène (mêmes bases logicielles sur différents serveurs), afin de pouvoir réagir efficacement (un seul patch à appliquer).

Dans le cas des systèmes RedHat Enterprise Linux, les notifications par mail des nouvelles failles sont très efficaces. De plus, il ne faut pas hésiter à faire appel au support en cas de doute sur n'importe quel point en rapport avec le système d'exploitation.

5.1 Bonnes pratiques à mettre en oeuvre

Voici une liste de bonnes pratiques qui pourraient vous sauver la vie dans le cadre de cette faille :

— Mettre à jour le système régulièrement
— Ne pas exposer son serveur SSH directement sur internet (bien configurer son iptables)
— Changer le port d'écoute par défaut de SSH et éviter les ports communs (6291 par exemple)
— Interdire à root de se connecter au serveur via SSH (`PermitRootLogin no` dans `/etc/ssh/sshd_config`) et avoir une bonne gestion des sudoers
— Utiliser un framework de prévention d'intrusions (ex : Fail2ban)
— Avoir un système de log. Sur ubuntu /var/log/auth.log. Nous n'avons pas réussi à vérifier ceci, car nous avons testé avec l'exemple fourni par libssh et qui est limité pas de variable d'environnement initialisé à la connection et pas de log.

6 Améliorer le développement

Nous n'avons pas trouvé le commit créant la faille de sécurité. Mais une telle faille est surprenante. Les tests ne doivent pas être négligé dans le génie logiciel. Des tests en boites noires, comme par exemple des tests fonctionnels essayant de reproduire des comportements malicieux aurait pu éviter peut-être une telle faille.

7 PSSI

Nous imaginons que nous sommes dans une grosse industrie (type Orange/Michelin, au hasard).

7.1 Criticité de la faille et potentiel impact

La facilité d'exploitation de la faille est effrayante et peut permettre à un attaquant de nuire à un SI assez facilement. Mais pour un groupe, plusieurs garde-fous sont déjà présent et efficaces. À notre connaissance notre industrie n'a pas d'accès ssh ouvert directement sur internet. Aucun port ssh trouvé sur shodan pour un nom de domaine Michelin.

Par contre, une personne avec de mauvaises intentions, ayant un accès sur l'intranet pourrait nous faire très mal. Il est tellement fréquent d'avoir les ports ssh ouverts sur les serveurs que je pense même que c'est peut-être par défaut. Donc tous les serveurs sont potentiellement faillibles. Évidemment l'utilisateur root ne peut se connecter directement aux machines. Mais les comptes pouvant se connecter sont présents dans les sudoers. Ils détiennent en général de nombreux privilèges, comme se connecter en root. Donc cette faille, combinée avec une élévation de privilèges est rendue facilement exécutable par des sudoers laxistes. Par expérience le logging des connexions n'est pas très bien réalisé.

En résumé, depuis internet presque aucun danger, mais depuis l'intranet, il exsite beaucoup de dangers.

7.2 Action préventive et curative

Ne pas donner d'accès à l'intranet depuis internet. Comme le SI de l'Ensimag, nous ne pouvons pas nous connecter directement, nous devons faire un rebond par depots.ensimag.fr. Ceci ne laisse pas entrer l'attaquant directement sur l'intranet.

Revoir la gestions des sudoers.

Ne pas avoir de serveur ssh démarré par défaut sur les machines. Avoir un système de logs performant, quitte à forcer les utilisateurs à utiliser des logiciels côté client pour se connecter (exemple : enregistrement des sessions vidéos de tout ce qui est fait sur la machine).

Automatiser les déploiements logiciels en n'utilisant pas SSH. Il est possible de n'utiliser que https. Destruction des machines virtuelles après une connection ssh sur la machine (exemple de la présentation Uber).

7.3 Formation des utilisateurs à envisager

Avoir des sysadmins à la pointe, lecteurs des maillings lists des distributions installés sur les serveurs. Automatiser au maximum les opérations de déploiement et gestion de la vie du logiciel. Les actions manuelles en SSH sur une machine doivent être limitées, voire prohibées. Utiliser des distributions respectables (debian,ubuntu,redhat,gentoo), avec des sources logicielles fiables. Faire la guerre au shadow IT, c'est le mal des SI et aussi un danger pour la sécurité de l'entreprise. Si la DSI ne maitrise pas son SI, il est très délicat d'avoir une bonne sécurité.

La gestion des serveurs doit ainsi suivre une politique d'entreprise stricte (datacenters propres, ou IaaS de prédilection...), pour n'importe quel projet.

8 Expérimentation

8.1 Le serveur

Afin de rendre facile cette expérimentation, nous vous avons préparé une image Docker contenant un seveur SSH utilisant `libssh 0.7.4` Tout cela se trouve dans le dossier `libssh-7.4-ssh_server_fork` et un simple `docker build -t unsecure-libssh .` au sein du dossier permettra de construire l'image.

Ensuite, il vous suffit de démarrer le serveur sur le port de votre choix (ici 2222) en utilisant `docker run -it -p 2222:22 unsecure-libssh`.

8.2 Le client malicieux

Côté client, un script pour l'exploitation de cette faille est disponible dans le dossier `script`. Vous avez juste besoin d'une version récente de Python (par exemple 3.6) pour l'exécuter. N'oubliez pas d'installer les librairies tierces requises en exécutant `pip install -r requirements.txt` au sein du dossier.

Ensuite, le script fonctionne comme suit : `python exploit.py <host> <port> <command>`. Exécutez-le de manière à contacter le conteneur Docker préalablement lancé, par exemple : `python exploit.py localhost 2222 id` exécutera la commande `id` au sein de l'image Docker et affichera la sortie sur le terminal. Vous devriez normalement voir une sortie du genre :

```
(venv) $ python script/exploit.py localhost 2222 id
INFO:paramiko.transport:Connected (version 2.0, client libssh_0.7.4)
uid=0(root) gid=0(root) groups=0(root)
```

On remarque que le client n'a eu besoin d'aucune authentification et a pu exécuter une commande en tant que root...