

## Linux Remote Management Protocols

In the world of Linux distributions, there are many ways to manage the servers remotely. For example, let us imagine that we are in one of many locations and one of our employees who just went to a customer in another city needs our help because of an error that he cannot solve. Efficient troubleshooting will look difficult over a phone call in most cases, so it is beneficial if we know how to log onto the remote system to manage it.

These applications and services can be found on almost every server in the public network. It is time-saving since we do not have to be physically present at the server, and the working environment still looks the same. These protocols and applications for remote systems management are an exciting target for these reasons. If the configuration is incorrect, we, as penetration testers, can even quickly gain access to the remote system. Therefore, we should familiarize ourselves with the most important protocols, servers, and applications for this purpose.

### SSH

Secure Shell (SSH) enables two computers to establish an encrypted and direct connection within a possibly insecure network on the standard port [TCP 22](#). This is necessary to prevent third parties from intercepting the data stream and thus intercepting sensitive data. The SSH server can also be configured to only allow connections from specific clients. An advantage of SSH is that the protocol runs on all common operating systems. Since it is originally a Unix application, it is also implemented natively on all Linux distributions and MacOS. SSH can also be used on Windows, provided we install an appropriate program. The well-known OpenBSD SSH ([openSSH](#)) server on Linux distributions is an open-source fork of the original and commercial [SSH](#) server from SSH Communication Security. Accordingly, there are two competing protocols: [SSH-1](#) and [SSH-2](#).

We can imagine that we want to manage a remote host. This can be done via the command line or GUI. Besides, we can also use the SSH protocol to send commands to the desired system, transfer files, or do port forwarding. Therefore, we need to connect to it using the SSH protocol and authenticate ourselves to it. In total, OpenSSH has six different authentication methods:

1. Password authentication
2. Public-key authentication
3. Host-based authentication
4. Keyboard authentication
5. Challenge-response authentication
6. GSSAPI authentication

We will take a closer look at and discuss one of the most commonly used authentication methods. In addition, we can learn more about the other authentication methods [here](#) among others.

#### Public Key Authentication

In a first step, the SSH server and client authenticate themselves to each other. The server sends a certificate to the client to verify that it is the correct server. Only when contact is first established is there a risk of a third party intercepting itself between the two participants and thus intercepting the connection. Since the certificate itself is also encrypted, it cannot be imitated. Once the client knows the correct certificate, no one else can pretend to make contact via the corresponding server.

After server authentication, however, the client must also prove to the server that it has access authorization. However, the SSH server is already in possession of the encrypted hash value of the password set for the desired user. As a result, users have to enter the password every time they log on to another server during the same session. For this reason, an alternative option for client-side authentication is the use of a public key and private key pair.

The private key is created individually for the user's own computer and secured with a passphrase that should be longer than a typical password. The private key is stored exclusively on our own computer and always remains secret. If we want to establish an SSH connection, we first enter the passphrase and thus open access to the private key.

Public keys are also stored on the server. The server creates a cryptographic problem with the client's public key and sends it to the client. The client, in turn, decrypts the problem with its own private key, sends back the solution, and thus informs the server that it may establish a legitimate connection. During a session, users only need to enter the passphrase once to connect to any number of servers. At the end of the session, users log out of their local machines, ensuring that no third party who gains physical access to the local machine can connect to the server.

#### Default Configuration

The [sshd\\_config](#) file, responsible for the OpenSSH server, has only a few of the settings configured by default. However, the default configuration includes X11 forwarding, which contained a command injection vulnerability in version 7.2p1 of OpenSSH in 2016. Nevertheless, we do not need a GUI to manage our servers.

#### Default Configuration

```
Default Configuration
Govardhan Gujji22@htb:[/htb]$ cat /etc/ssh/sshd_config | grep -v "#" | sed -r '/^#\s*/d'
Include /etc/ssh/sshd_config.d/*
ChallengeResponseAuthentication no
UsePAM yes
X11Forwarding yes
PrintMotd no
AcceptEnv LANG LC_*
Subsystem    sftp    /usr/lib/openssh/sftp-server
```

Most settings in this configuration file are commented out and require manual configuration.

#### Dangerous Settings

Despite the SSH protocol being one of the most secure protocols available today, some misconfigurations can still make the SSH server vulnerable to easy-to-execute attacks. Let us take a look at the following settings:

| Setting                    | Description                                 |
|----------------------------|---|
| PasswordAuthentication yes | Allows password-based authentication.       |
| PermitEmptyPasswords yes   | Allows the use of empty passwords.          |
| PermitRootLogin yes        | Allows to log in as the root user.          |
| Protocol 1                 | Uses an outdated version of encryption.     |
| X11Forwarding yes          | Allows X11 forwarding for GUI applications. |
| AllowTcpForwarding yes     | Allows forwarding of TCP ports.             |
| PermitTunnel               | Allows tunneling.                           |
| DebianBanner yes           | Displays a specific banner when logging in. |

Allowing [password authentication](#) allows us to brute-force a known username for possible passwords. Many different methods can be used to guess the passwords of users. For this purpose, specific patterns are usually used to mutate the most commonly used passwords and, frighteningly, correct them. This is because we humans are lazy and do not want to remember complex and complicated passwords. Therefore, we create passwords that we can easily remember, and this leads to the fact that, for example, numbers or characters are added only at the end of the password. Believing that the password is secure, the mentioned patterns are used to guess precisely such "adjustments" of these passwords. However, some instructions and [hardening guides](#) can be used to harden our SSH servers.

#### Footprinting the Service

One of the tools we can use to fingerprint the SSH server is [ssh-audit](#). It checks the client-side and server-side configuration and shows some general information and which encryption algorithms are still used by the client and server. Of course, this could be exploited by attacking the server or client at the cryptic level later.

#### SSH-Audit

```
SSH-Audit
Govardhan Gujji22@htb:[/htb]$ git clone https://github.com/jtesta/ssh-audit.git && cd ssh-audit
```

```
Govardhan Gujji22@htb:[/htb]$ ./ssh-audit.py 10.129.14.132
```

```
# general
(gen) banner: SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.3
(gen) software: OpenSSH 8.2p1
(gen) compatibility: OpenSSH 7.4+, Dropbear SSH 2018.76+
(gen) compression: enabled (zlib@openssh.com)
```

```
# key exchange algorithms
(kex) curve25519-sha256          -- [info] available since OpenSSH 7.4, Dropbear SSH 2018.76
(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 (2048-bit) -- [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
```

```
# host-key algorithms
(key) rsa-sha2-512 (3072-bit)     -- [info] available since OpenSSH 7.2
(key) rsa-sha2-256 (3072-bit)     -- [info] available since OpenSSH 7.2
(key) ssh-rsa (3072-bit)          -- [fail] using weak hashing algorithm
                                -- [info] available since OpenSSH 2.5.0, Dropbear SSH 0.28
                                -- [warn] a future deprecation notice has been issued in OpenSSH 7.3
                                -- [fail] using weak elliptic curves
                                -- [info] available since OpenSSH 5.7, Dropbear SSH 2013.62
(key) ecdsa-sha2-nistp256        -- [fail] using weak elliptic curves
                                -- [info] available since OpenSSH 5.7, Dropbear SSH 2013.62
(key) ssh-ed25519                -- [info] available since OpenSSH 5.7, Dropbear SSH 2013.62
...SNIP...
```

```
< ----- >
```

The first thing we can see in the first few lines of the output is the banner that reveals the version of the OpenSSH server. The previous versions had some vulnerabilities, such as [CVE-2020-14145](#), which allowed the attacker the capability to Man-In-The-Middle and attack the initial connection attempt. The detailed output of the connection setup with the OpenSSH server can also often provide important information, such as which authentication methods the server can use.

#### Change Authentication Method

```
Change Authentication Method
Govardhan Gujji22@htb:[/htb]$ ssh -v cry0l1t3@10.129.14.132
```

```
OpenSSH_8.2p1 Ubuntu-4ubuntu0.3, OpenSSL 1.1.1f 31 Mar 2020
debug1: Reading configuration data /etc/ssh/sshd_config
...SNIP...
debug1: Authentications that can continue: publickey,password,keyboard-interactive
```

```
Cry0l1t3@10.129.14.132's password:
```

```
< ----- >
```

For potential brute-force attacks, we can specify the authentication method with the SSH client option [PreferredAuthentications](#).

```
Change Authentication Method
Govardhan Gujji22@htb:[/htb]$ ssh -v cry0l1t3@10.129.14.132 -o PreferredAuthentications=password
```

```
OpenSSH_8.2p1 Ubuntu-4ubuntu0.3, OpenSSL 1.1.1f 31 Mar 2020
debug1: Reading configuration data /etc/ssh/sshd_config
...SNIP...
debug1: Authentications that can continue: publickey,password,keyboard-interactive
debug1: Next authentication method: password
cry0l1t3@10.129.14.132's password:
```

```
< ----- >
```

Even with this obvious and secure service, we recommend setting up our own OpenSSH server on our VM, experimenting with it, and familiarizing ourselves with the different settings and options.