Hardening OpenSSH

The OpenSSH server offers remote Secure Shell services towards your users. This benchmark focuses on the hardening of OpenSSH within a Gentoo Hardened environment.

Applicable platforms

- cpe:/o:gentoo:linux

Version: 1

Revision history

- **draft** (as of 2012-07-14)

Table of Contents

1. Introduction

The OpenSSH service is one of the most used SSH providing services.

Table of Contents

1.1. Using this guide

The guide you are currently reading is the guide generated from this SCAP content (more specifically, the XCCDF document) using **openscap**, a free software implementation for handling SCAP content. Within Gentoo, the package `app-forensics/openscap` provides the tools, and the following command is used to generate the HTML output:

```
### Command to generate this guide ###
# oscap xccdf generate guide openssh-xccdf.xml > guide-openssh-xccdf.html
```

Secondly, together with this XCCDF XML, you will also find an OVAL XML file. The two files combined allow you to automatically validate various settings as documented in the benchmark.

You can test the benchmark against your configuration.

```
### Testing the rules mentioned in the XCCDF document ###
# oscap xccdf eval --cpe gentoo-cpe.xml --profile xccdf_org.gentoo.dev.swift_profile_default opens
```

To generate a full report in HTML as well, you can use the next command:

```
### Testing the rules and generating an HTML report ###
# oscap xccdf eval --cpe gentoo-cpe.xml --profile xccdf_org.gentoo.dev.swift_profile_default --res
```

The benchmark is also available as data stream. In this case, you do not need to provide the various files - all you need is the benchmark file. For instance:

```
### Testing the rules based on the data stream
# oscap xccdf eval --profile xccdf_org.gentoo.dev.swift_profile_default openssh-ds.xml
```

Finally, this benchmark will suggest some settings which you do not want to enable. That is perfectly fine - even more, some settings might even raise eyebrows left and right. We'll try to document the reasoning behind the settings but you are free to deviate from them. If that is the case, you might want to create your own profile which only contains the rules you want checked. You can then use that profile instead of the Default one.

1.2. Available XCCDF Profiles

As mentioned earlier, the XCCDF document supports multiple profiles. For the time being, one profile is defined:

- Default contains all mentioned tests

Substitute the profile information in the commands above with the profile you want to test on.

2. Configuration Settings

In this section, we look at the configuration settings of an OpenSSH service

Table of Contents

- 2.2.1.a. file /etc/ssh/sshd_config must have a line that matches ^PermitRootLogin no
- 2.2.2. Use public key authentication
  - 2.2.2.a. file /etc/ssh/sshd_config must have a line that matches ^PasswordAuthentication no
- 2.2.3. Disable ChallengeResponseAuthentication
  - 2.2.3.a. file /etc/ssh/sshd_config must have a line that matches ^ChallengeResponseAuthentication no

2.3. Access control related settings

- 2.3.1. Only allow specific group(s) access
  - 2.3.1.a. file /etc/ssh/sshd_config must have a line that matches ^AllowGroup
- 2.3.2. Only allow specific host(s) access
  - 2.3.2.a. file /etc/hosts.allow must have a line that matches ^sshd:
  - 2.3.2.b. file /etc/hosts.deny must have a line that matches ^sshd: ALL
- 2.3.3. Only listen on proper interfaces
  - 2.3.3.a. file /etc/ssh/sshd_config must have a line that matches ^ListenAddress
  - 2.3.3.b. file /etc/ssh/sshd_config may not have a line that matches ^ListenAddress.*0.0.0.0
  - 2.3.3.c. file /etc/ssh/sshd_config may not have a line that matches ^ListenAddress[ ]*::$

2.4. Disable unused settings

- 2.4.1. Disable TCP forwarding
  - 2.4.1.a. file /etc/ssh/sshd_config must have a line that matches ^AllowTcpForwarding.*no

2.1. Default OpenSSH settings

OpenSSH comes with some sane defaults to start with. These should not be touched.

2.1.1. Ignore Rhosts

Historically, users could define a `.rhosts` or `.shosts` file in which they mention the systems from which they log on to the system (the client hosts). When the user then logs on from one of these remote locations, the shell service would not ask for password authentication and just automatically log in the user.

The shell service treats `.shosts` mentioned hosts a bit different: it first checks that hosts identity using some public key authentication scheme (in which case the host keys of the clients are placed in `/etc/ssh/ssh_known_hosts` or `~/.ssh/known_hosts`).

This is however a very insecure setup and can be easily circumvented. It only performs host-based authentication, not user authentication, and in case of the `.rhosts` file this host-based authentication is only based on the hostname/IP matching.

For this reason, support for the `.rhosts` and `.shosts` files is by default disabled.

```
### /etc/ssh/sshd_config : IgnoreRhosts
# If set, IgnoreRhosts must be set to yes (which is the default)
IgnoreRhosts yes
```

2.1.1.a. file /etc/ssh/sshd_config may not have a line that matches ^IgnoreRhosts.*no

file /etc/ssh/sshd_config may not have a line that matches ^IgnoreRhosts.*no

2.1.2. Do not allow RSA Host Authentication

As part of the Rhosts implementation, OpenSSH supports using RSA authentication for remote hosts. With RSA authentication enabled, hosts mentioned in the `.rhosts` (or `/etc/hosts.equiv`) files need to be authenticated based on their RSA key. This applies to the SSH protocol version 1 only.

As Rhosts is found insecure, this option does not make rhosts more feasible to use. For this reason, this option is by default disabled.

```
### /etc/ssh/sshd_config : RhostsRSAAuthentication
# If set, RhostsRSAAuthentication must be set to "no" (which is the default).
RhostsRSAAuthentication no
```

2.1.2.a. file /etc/ssh/sshd_config may not have a line that matches ^RhostsRSAAuthentication.*yes

file /etc/ssh/sshd_config may not have a line that matches ^RhostsRSAAuthentication.*yes

2.1.3. Do not allow Host-based Authentication

As part of the Rhosts implementation, Ope SSH supports using public key authenticatoin for remote hosts. With this enabled, hosts mentioned in the `.rhosts` (or `/etc/hosts.equiv`) files need to be authenticated based on their public key. This applies to the SSH protocol version 2 only.

As Rhosts is found insecure, this option does not make rhosts more feasible to use. For this reason, this option is by default disabled.

```
### /etc/ssh/sshd_config : HostbasedAuthentication
# If set, HostbasedAuthentication must be set to "no" (which is the default)
```

```
HostbasedAuthentication no
```

2.1.3.a. file /etc/ssh/sshd_config may not have a line that matches ^HostbasedAuthentication.*yes

file /etc/ssh/sshd_config may not have a line that matches ^HostbasedAuthentication.*yes

2.1.4. Do not Permit Empty Passwords

If password-based authentication is used, it is wise not to allow empty passwords.

Allowing empty passwords within your network makes the services *very* vulnerable to exploit, even when the software is fully up-to-date.

```
### /etc/ssh/sshd_config : PermitEmptyPasswords
# If set, PermitEmptyPasswords must be set to "no" (which is the default).
PermitEmptyPasswords no
```

2.1.4.a. file /etc/ssh/sshd_config may not have a line that matches ^PermitEmptyPasswords.*yes

file /etc/ssh/sshd_config may not have a line that matches ^PermitEmptyPasswords.*yes

2.1.5. Use PAM

PAM (Pluggable Authentication Modules) is a powerful framework for managing authentication of users and services in a flexible manner. By default, OpenSSH uses PAM for the authentication of users.

One of the many advantages of PAM is that you can add in additional rules you want to enforce during the authentication. You can limit access based on login count (or number of failures), use centralized authentication repositories (like OpenLDAP), allow access only during specific time windows, etc.

It is strongly advised to use PAM for SSH authentication too (but do manage the PAM configuration properly!) Be aware though that the authentication services themselves (is the user who he sais he is) of PAM are not used if public key authentication is used. The other services, which include the access controls mentioned earlier, are still consulted though.

```
### /etc/ssh/sshd_config : UsePAM
# If set, UsePAM must be set to "yes" (which is the default)
UsePAM yes
```

2.1.5.a. file /etc/ssh/sshd_config may not have a line that matches ^UsePAM.*no

file /etc/ssh/sshd_config may not have a line that matches ^UsePAM.*no

2.1.6. Only use version 2 of the SSH protocol

The first version of the SSH protocol has been found insecure: TODO.

For this reason, it is strongly advised to use version 2 of the protocol only. This is also the default for OpenSSH.

```
### /etc/ssh/sshd_config : Protocol
# If set, Protocol must be set to 2 only (which is the default)
Protocol 2
```

2.1.6.a. file /etc/ssh/sshd_config may not have a line that matches ^Protocol.*1

file /etc/ssh/sshd_config may not have a line that matches ^Protocol.*1

2.1.7. Use privilege separation

With privilege separation enabled, the SSH daemon has a tiny footprint running as root, whereas the rest of the application runs as an unprivileged process to deal with the incoming network traffic. This can be tuned with `UsePrivilegeSeparation yes` which is the default for OpenSSH.

```
### /etc/ssh/sshd_config : UsePrivilegeSeparation
# If set, UsePrivilegeSeparation must be set to yes (which is the default)
UsePrivilegeSeparation yes
```

2.1.7.a. file /etc/ssh/sshd_config may not have a line that matches ^UsePrivilegeSeparation.*no

file /etc/ssh/sshd_config may not have a line that matches ^UsePrivilegeSeparation.*no

2.1.8. Disable X11 forwarding

SSH supports forwarding X11 packets, so X11 applications started on the remote system have their display shown on the client. This behavior is by default disabled.

```
### /etc/ssh/sshd_config : X11Forwarding
# If set, X11Forwarding must be set to no (which is the default)
X11Forwarding no
```

2.1.8.a. file /etc/ssh/sshd_config may not have a line that matches ^X11Forwarding.*yes

file /etc/ssh/sshd_config may not have a line that matches ^X11Forwarding.*yes

2.1.9. Enable strict mode

When `StrictModes yes` is enabled, the SSH daemon will only allow a remote user to log on when some of the important files in that users' home directory have the proper privileges and ownership. This behavior is by default enabled.

```
### /etc/ssh/sshd_config : StrictModes
# If set, StrictModes must be set to yes (which is the default)
StrictModes yes
```

2.1.9.a. file /etc/ssh/sshd_config may not have a line that matches ^StrictMode.*no

file /etc/ssh/sshd_config may not have a line that matches ^StrictMode.*no

2.2. Authentication-related settings

Being a remote shell service, authentication is one of the main features that OpenSSH provides. A few settings help us in hardening the SSH server even further.

2.2.1. Disable root logins

As root is one of the most powerful accounts, direct access to root should be limited. It is advised that, if a process needs root privileges, it uses a functional account which has the right to call one or a few commands as root, but nothing else.

With OpenSSH, it is possible to prohibit direct root access towards the system if feasible within your architecture. This

can be accomplished using the `PermitRootLogin no` directive. If you need root logins, consider only allowing specified command access (forced-commands-only).

```
### /etc/ssh/sshd_config : PermitRootLogin
# Set this to "no" or, if needed, "forced-commands-only"
PermitRootLogin no
```

2.2.1.a. file /etc/ssh/sshd_config must have a line that matches ^PermitRootLogin no

file /etc/ssh/sshd_config must have a line that matches ^PermitRootLogin no

2.2.2. Use public key authentication

By default, OpenSSH uses interactive, keyboard-based password logins. One intrinsic problem with passwords is that they can be weak, but also that hacked passwords can be used from other locations.

A safer approach for remote shell invocation is to use a keypair: the key is much stronger than most passwords, making brute-force improbably and dictionary-attacks useless. The private key is only known by you (on your system) and optionally (but preferably) protected by a (strong) passphraze so that adversaries that force access to your system can still not use your private key.

Such a keypair an be generated by the users using **ssh-keygen -t dsa** after which the private and public keys are stored in `~/.ssh`

On the OpenSSH server level, you can force the use of public key authentication (and thus deny keyboard-interactive password logins) using `PasswordAuthentication no`.

```
### /etc/ssh/sshd_config : PasswordAuthentication
# Set this to "no"
```

```
PasswordAuthentication no
```

2.2.2.a. file /etc/ssh/sshd_config must have a line that matches ^PasswordAuthentication no

file /etc/ssh/sshd_config must have a line that matches ^PasswordAuthentication no

2.2.3. Disable ChallengeResponseAuthentication

In OpenSSH, a (confusing) parameter called `ChallengeResponseAuthentication` is available (and by default enabled). Many users might believe that this implements a more secure authentication method (based on a challenge and a token that need to be verified - i.e. multi-factor authentication). However, in case of this parameter, this isn't true.

The `ChallengeResponseAuthentication` setting enables *TIS Challenge/Response* in SSH protocol version 1, and keyboard-interactive in SSH protocol v2. Hence, in our case, it is best set disabled as we do not want regular password authentication to be enabled (and don't use protocol version 1).

```
### /etc/ssh/sshd_config : ChallengeResponseAuthentication
# Set this to "no"
ChallengeResponseAuthentication no
```

2.2.3.a. file /etc/ssh/sshd_config must have a line that matches ^ChallengeResponseAuthentication no

file /etc/ssh/sshd_config must have a line that matches ^ChallengeResponseAuthentication no

2.3. Access control related settings

By default, OpenSSH allows access from any location and by any user who gets authenticated properly. However, it is safer if you can restrict access from hosts that are allowed to access the SSH service (and not other hosts) as well as users that are known to access the system remotely.

2.3.1. Only allow specific group(s) access

Not every user on your system needs to be able to remotely log on to the system. Many users on your system are local-only, either because they are services accounts, or because the users are only meant to log on directly (or through another service).

With OpenSSH, you can limit SSH access to users defined in a limited set of (Unix) groups. It is recommended to define a Unix group (like `ssh` if that isn't used by the service daemon itself) in which those users are defined, and then only allow SSH access for this group.

```
### /etc/ssh/sshd_config : AllowGroup
# Set this to the unix group whose members are allowed access
AllowGroup ssh
```

2.3.1.a. file /etc/ssh/sshd_config must have a line that matches ^AllowGroup

file /etc/ssh/sshd_config must have a line that matches ^AllowGroup

2.3.2. Only allow specific host(s) access

Not every host on your network (or beyond) needs access to your system. On the contrary, most hosts probably shouldn't have SSH access to your system.

With a service called *tcpwrappers* OpenSSH allows administrators to define the hosts allowed access (or explicitly not allowed access) in the `/etc/hosts.allow` and `/etc/hosts.deny`.

For a good secure setting, it is recommended to disallow access from any host, and then explicitly grant access from a select set of hosts (or subnetworks).

```
### /etc/hosts.allow
# Give the list of allowed hosts or networks
sshd: 192.168.1.0/24
```

```
### /etc/hosts.deny
# Deny access by default from everywhere
sshd: ALL
```

2.3.2.a. file /etc/hosts.allow must have a line that matches ^sshd:

file /etc/hosts.allow must have a line that matches ^sshd:

2.3.2.b. file /etc/hosts.deny must have a line that matches ^sshd: ALL

file /etc/hosts.deny must have a line that matches ^sshd: ALL

2.3.3. Only listen on proper interfaces

By default, OpenSSH listens on all available interfaces. In many cases though, this isn't necessary.

Multihomed systems (i.e. systems with multiple network interfaces) usually only use a single interface for the administrative access, whereas the other interface is to connect to the Internet or disclose the "business applications".

On dual stack systems (i.e. systems with an IPv4 and IPv6 stack) the IPv6 (or IPv4) address might not be in use, or not for the administrative access (like through OpenSSH). In these cases, it is wise not to have OpenSSH listen on these addresses either.

```
## /etc/ssh/sshd_config : ListenAddress
# Define a ListenAddress, but do not set it to "any address"
# (which is 0.0.0.0 in IPv4 and :: in IPv6)
ListenAddress 192.168.100.121
```

2.3.3.a. file /etc/ssh/sshd_config must have a line that matches ^ListenAddress

file /etc/ssh/sshd_config must have a line that matches ^ListenAddress

2.3.3.b. file /etc/ssh/sshd_config may not have a line that matches ^ListenAddress.*0.0.0.0

file /etc/ssh/sshd_config may not have a line that matches ^ListenAddress.*0.0.0.0

2.3.3.c. file /etc/ssh/sshd_config may not have a line that matches ^ListenAddress[ ]*::$

file /etc/ssh/sshd_config may not have a line that matches ^ListenAddress[ ]*::$

2.4. Disable unused settings

OpenSSH has a few more options that it supports. If you, however, have no need for these options, it is safer to have them disabled. Potential vulnerabilities that might be discovered later on these options then have no effect on your system.

2.4.1. Disable TCP forwarding

SSH supports "tunneling", where packets are forwarded over a (partially) secure channel towards another location. If you do not need this, disable TCP forwarding through `AllowTcpForwarding no`

```
### /etc/ssh/sshd_config : AllowTcpForwarding
# If not needed, disable TCP forwarding
AllowTcpForwarding no
```

2.4.1.a. file /etc/ssh/sshd_config must have a line that matches ^AllowTcpForwarding.*no

file /etc/ssh/sshd_config must have a line that matches ^AllowTcpForwarding.*no