Client Command & Server Configuration

Admin

SSH Tunneling: Examples, Command, Server Config

What Is SSH Port Forwarding, aka SSH Tunneling?

SSH port forwarding is a mechanism in SSH for tunneling application ports from the client machine to the server machine, or vice versa. It can be used for *adding encryption to legacy applications*, *going through firewalls*, and some system administrators and IT professionals use it for *opening backdoors* into the internal network from their home machines. It can also be abused by hackers and malware to open access from the Internet to the internal network. See the SSH tunneling page for a broader overview.

Local Forwarding

Local forwarding is used to forward a port from the client machine to the server machine. Basically, the SSH client listens for connections on a configured port, and when it receives a connection, it tunnels the connection to an SSH server. The server connects to a configurated destination port, possibly on a different machine than the SSH server.

Typical uses for local port forwarding include:

- Tunneling sessions and file transfers through jump servers
- Connecting to a service on an internal network from the outside
- Connecting to a remote file share over the Internet

Quite a few organizations for all incoming SSH access through a single jump server. The server may be a standard Linux/Unix box, usually with some extra hardening, intrusion detection, and/or logging, or it may be a commercial jump server solution.

Many jump servers allow incoming port forwarding, once the connection has been authenticated. Such port forwarding is convenient, because it allows tech-savvy users to use internal resources quite transparently. For example, they may forward a port on their local machine to the corporate intranet web server, to an internal mail server's IMAP port, to a local file server's 445 and 139 ports, to a printer, to a version control repository, or to almost any other system on the internal network. Frequently, the port is tunneled to an SSH port on an internal machine.

In OpenSSH, local port forwarding is configured using the `-L` option:

```
ssh -L 80:intra.example.com:80 gw.example.com
```

This example opens a connection to the `gw.example.com` jump server, and forwards any connection to port 80 on the local machine to port 80 on `intra.example.com`.

By default, anyone (even on different machines) can connect to the specified port on the SSH client machine. However, this can be restricted to programs on the same host by supplying a *bind address*:

```
ssh -L 127.0.0.1:80:intra.example.com:80 gw.example.com
```

The `LocalForward` option in the OpenSSH client configuration file can be used to configure forwarding without having to specify it on command line.

Remote Forwarding

In OpenSSH, remote SSH port forwardings are specified using the `-R` option. For example:

```
ssh -R 8080:localhost:80 public.example.com
```

This allows anyone on the remote server to connect to TCP port 8080 on the remote server. The connection will then be tunneled back to the client host, and the client then makes a TCP connection to port 80 on `localhost`. Any other host name or IP address could be used instead of `localhost` to specify the host to connect to.

This particular example would be useful for giving someone on the outside access to an internal web server. Or exposing an internal web application to the public Internet. This could be done by an employee working from home, or by an attacker.

By default, OpenSSH only allows connecting to remote forwarded ports from the server host. However, the `GatewayPorts` option in the server configuration file sshd_config can be used to control this. The following alternatives are possible:

```
GatewayPorts no
```

This prevents connecting to forwarded ports from outside the server computer.

```
GatewayPorts yes
```

This allows anyone to connect to the forwarded ports. If the server is on the public Internet, anyone on the Internet can connect to the port.

```
GatewayPorts clientspecified
```

This means that the client can specify an IP address from which connections to the port are allowed. The syntax for this is:

```
ssh -R 52.194.1.73:8080:localhost:80 host147.aws.example.com
```

In this example, only connections from the IP address `52.194.1.73` to port 8080 are allowed.

OpenSSH also allows the forwarded remote port to specified as 0. In this case, the server will dynamically allocate a port and report it to the client. When used with the `-O forward` option, the client will print the allocated port number to standard output.

Opening Backdoors into the Enterprise

Remote SSH port forwarding is commonly used by employees to open backdoors into the enterprise. For example, the employee may set get a free-tier server from Amazon AWS, and log in from the office to that server, specifying remote forwarding from a port on the server to some server or application on the internal enterprise network. Multiple remote forwards may be specified to open access to more than one application.

The employee would also set `GatewayPorts yes` on the server (most employees do not have fixed IP addresses at home, so they cannot restrict the IP address).

For example, the following command opens access to an internal Postgres database at port 5432 and an internal SSH port at port 2222.

```
    ssh -R 2222:d76767.nyc.example.com:22 -R 5432:postgres3.nyc.example.com:5432
aws4.mydomain.net
```

Server-Side Configuration

The `AllowTcpForwarding` option in the OpenSSH server configuration file must be enabled on the server to allow port forwarding. By default, forwarding is allowed. Possible values for this option are `yes` or `all` to allow all TCP forwarding, `no` to prevent all TCP forwarding, `local` to allow local forwardings, and `remote` to allow remote forwardings.

Another option of interest is `AllowStreamLocalForwarding`, which can be used to forward Unix domain sockets. It allows the same values as `AllowTcpForwarding`. The default is `yes`.

For example:

```
    AllowTcpForwarding remote     AllowStreamLocalForwarding no
```

The `GatewayPorts` configuration option as described above also affects remote port forwardings. Possible values were `no` (only local connections from server host allowed; default), `yes` (anyone on the Internet can connect to remote forwarded ports), and `clientspecified` (client can specify an IP address that can connect, anyone can if not specified).

How to Prevent SSH Port Forwarding from Circumventing Firewalls

We recommend that port forwarding be expressly disabled when not needed. Leaving port forwarding enabled can expose the organization to security risks and backdoors. For example, if a server intended to only provide SFTP file transfers allows port forwardings, those forwardings might be used to gain unintended access into the internal network from the Intranet.

The problem is that port forwarding can in practice only be prevented by a server or firewall. An enterprise cannot control all servers on the Internet. Firewall-based control can also be tricky, as most organizations have servers in Amazon AWS and other cloud services, and those servers are usually accessed using SSH.

SSH's solution

SSH's Tectia SSH Client/Server is a commercial solution that can provide secure application tunneling along with SFTP and secure remote access for enterprises.

Further Information

> More information on SSH tunneling

We at SSH secure communications between systems, automated applications, and people. We strive to build future-proof and safe communications for businesses and organizations to grow safely in the digital world.

Stay on top of the latest in cybersecurity

Be the first to know about SSH’s new solutions, product updates, new features, and other SSH news!