EN

Breaking Network Boundaries with SSH Services

rootkiter
2024年5月31日 • 6 min read

Background

Recently, XLab received a project that requires the deployment of a self-developed system on a restricted intranet. Due to objective constraints, we are unable to directly access their intranet. Initially, we planned to use "Sunlogin(Remote Control Tool)" as a solution, but the network bandwidth was insufficient to support desktop operations, which poses certain difficulties for the project. Finally, we consider using SSH to establish a tunnel and ensure the

smooth progress of the project. After the project is completed, we will consider organizing relevant information about SSH tunnels, which can also serve as a reference for navigation and bypassing intranet security technologies.

Scenario Considerations

Since we are preparing a systematic review, we cannot just address a single need, but must consider various use cases. Therefore, I have outlined the following scenarios for readers to match as needed:

> 1. *Two friends in different network domains need a unified network channel to collaboratively set up systems, develop programs, conduct penetration testing, and more.*
> 2. *In an intranet penetration scenario, where the boundary host is an intranet host, the goal is to use the boundary host as a springboard to access deeper layers of the network.*
> 3. *After entering a restricted network, I hope to use my local network to indirectly access a restricted service*
> 4. *At home, setting up a one-click XXX service, and wanting to show it off to friends while away.*

To achieve these scenarios, we can break down SSH tunneling into several key concepts, which are as follows:

> 1. *Port Forwarding*
> 2. *Reverse Port Forwarding*
> 3. *SSH's ProxyJump*
> 4. *Forward SOCKS proxy service*
> 5. *Reverse socks proxy service*
> 6. *Secure Mutual Access of Hosts within a Dual NAT Network*
> 7. *Supplement: SSH Login Based on the SOCKS Protocol*

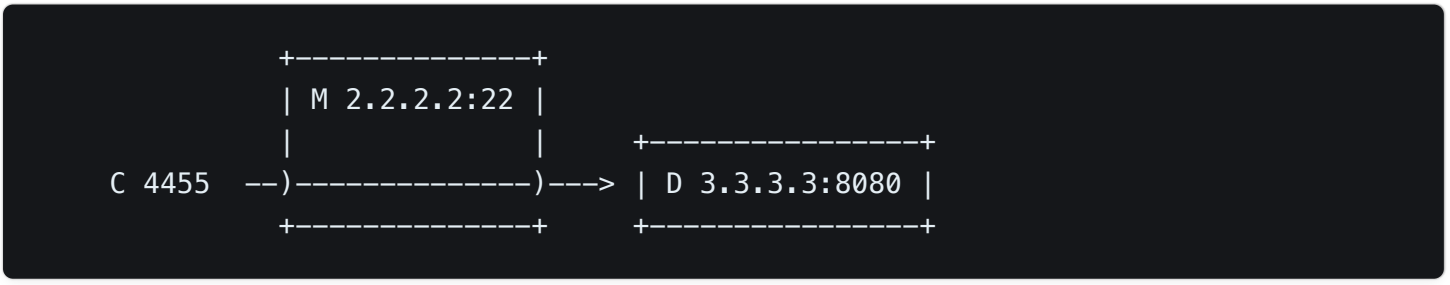
Main

Before starting,
some readers might be concerned about using Windows as their daily office environment without SSH. For the server part, you can solve this by choosing a cheap VPS from a cloud provider. For the client part, consider using the WSL subsystem. At the very least, an SSH client would suffice. The examples in this article are mainly command line-based, but there are similar features available in some GUI tools which readers can explore on their own. For instance, I used to use similar features on Xshell. However, ever since the supply chain attack on Xshell was exposed, I stopped using it. I hope everyone ensures security when using GUI tools for work and doesn't unwittingly help others.

Once these concerns are addressed, we can proceed to prepare the learning environment. You'll need to set up a remote SSH server (M) and your current "local machine (C)." We'll use different parameter options to log into M from C to unfold the study in steps 1-5.

(In the diagrams in the chapters, arrows indicate the direction of data flow.)

1. Port Forwarding



Assumption: We have a public SSH machine M (2.2.2.2:22) and a service D (3.3.3.3:8080) that needs to be accessed. Our local office machine (local machine) C wants to access D through M. We can log in to M from machine C using the following command.

```
[client]$ ssh -L 4455:3.3.3.3:8080 ubuntu@2.2.2.2
```

After successful authentication, machine C will listen on port 127.0.0.1:4455, and the service will be provided by D (3.3.3.3:8080).

PS: Some people might know that the tor service is by default listening on the local loopback address, but the tor service cannot be used directly in China. So, what have you clever folks thought of?

2. Reverse Port Forwarding

Assumption: We have a public SSH machine M (2.2.2.2:22), a local machine C, and a service E (4.4.4.4:9090) that we want to access. We want to access the service E from M through C. This requires reverse port forwarding. You can log in to M from the local machine C using the following command.

```
[client]$ ssh -R 5544:4.4.4.4:9090 ubuntu@2.2.2.2
```

After authentication, machine M will listen on port 127.0.0.1:5544. The service provided by E (4.4.4.4:9090) can be accessed through port 5544 on M, with the communication packets being relayed via the local machine C.

PS: If we replace the service address of E with the loopback address 127.0.0.1 in the command, we can access the services provided by the local machine C from M.

3. SSH's ProxyJump

Login to D(3.3.3.3:22) via host M(2.2.2.2:22).

```
[client]$ ssh -J ubuntu@2.2.2.2:22 admin@3.3.3.3
```

PS: Key authentication is preferable to password authentication.

PPS: When using a regular account as a jump host, it is possible to indirectly bypass the security rule that prohibits remote login as the root account.

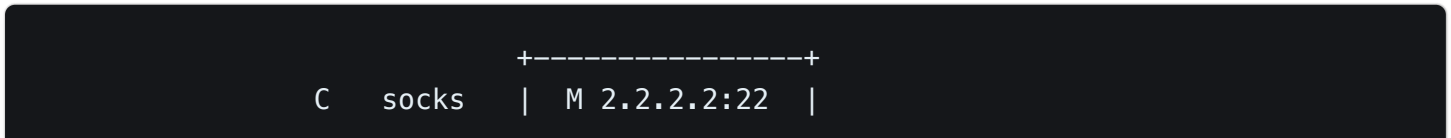
4. Forward SOCKS proxy service

In China, bypassing the firewall to access the internet is a common skill, and we can try to access restricted websites in this way.

```
[client]$ ssh -D 1080 admin@2.2.2.2
```

Then, you can configure "socks5://127.0.0.1:1080" in your browser and enjoy the internet.

5.1 Reverse socks proxy service



```
              /---------(---- 1088 <-|   |
              |         +----------------+
          +----+-----+
          v    v     v
         SSH  RDP  HTTP
```

In contrast to 4, if the remote server is on a relatively restricted network and requires access to certain network resources through the local machine, then it may be appropriate to consider enabling reverse proxy.

```
[client]$ ssh -R 1088 admin@2.2.2.2
```

Afterwards, the M host can access client network resources through port 1088.

5.2 Another Reverse socks proxy service

```
                          +----------------+
                          |  M 2.2.2.2:22  |
              C           |                |
                 /---------(---- 1088 <--   |
                 |        +----------------+
      socks 127.0.0.1:1080
            +----+-----+
            v    v     v
           SSH  RDP  HTTP
```

In many older versions of SSH servers, reverse SOCKS proxy services are not supported. In such cases, consider using `2` + `4` to assemble a reverse proxy service.

First, obtain a forward SOCKS proxy service on the client laptop.

```
[client]$ ssh -D 1080 admin@127.0.0.1
```

Forward it to the remote server next.

```
[client]$ ssh -R 1088:127.0.0.1:1080 admin@2.2.2.2
```

Now you've got it.

6. Secure Mutual Access of Hosts within a Dual NAT Network

Up to this point, some of the more astute readers may have already thought of a method for mutual access between two hosts within the same internal network. It simply requires the target host to reverse map its service to a public VPS, and then use C to access this mapping.

Depending on individual needs, combining the capabilities of 1-5, numerous mapping solutions can be assembled.

Here, we provide a relatively simple solution for reference:

```
                +----------------+         +------------------+
                |  M 2.2.2.2:22  |         |      NAT-S       |
      NAT-C  ---)--------> 2222 -)------>  | ssh:127.0.0.1:22 |
                +----------------+         +------------------+
```

First, reverse forward the SSH service of the target host to the M host.

```
[S]$ ssh -R 2222:127.0.0.1:22 ubuntu@2.2.2.2
```

Then access this service on C:

```
[client]$ ssh -J ubuntu@2.2.2.2 -p 2222 admin@127.0.0.1
```

Based on your scenario, you can forward other types of services, such as RDP, VNC, SMB, etc.

7. Supplement: SSH Login Based on the SOCKS Protocol

In some scenarios, we need to login the SSH server through the socks proxy.

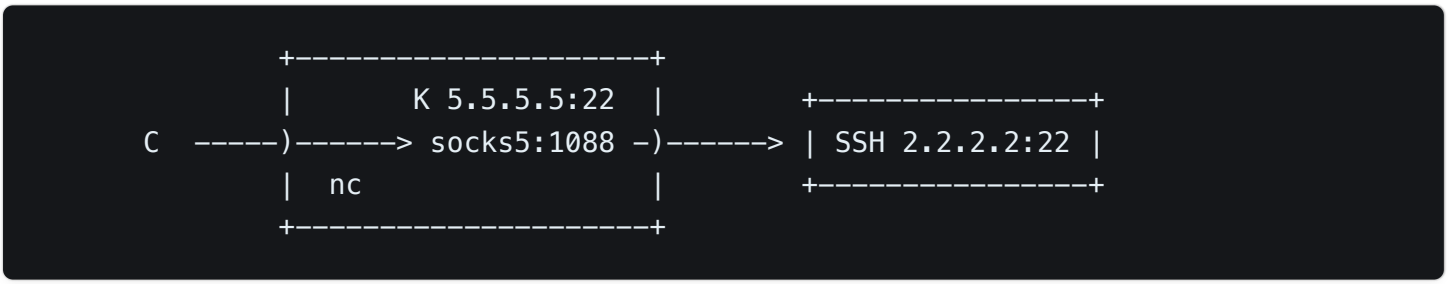
PS: Since this part requires the proxy connection feature of `nc` (`nc -X`), and considering that this is a relatively new option in nc, we will discuss this in two situations:

use local nc

Example of Commands:

```
[client]$ ssh -o 'ProxyCommand nc -X 5 -x 127.0.0.1:1080 %h %p' admin@2.2.2.2
```

Using the server-side nc.



This option is used to login the host and execute `nc -X` commands.

```
[client]$ ssh -o 'ProxyCommand ssh root@5.5.5.5 nc -X 5 -x 127.0.0.1:1088 %h %p' ro
```

Here you need to pass the authentication of K first, then you can log in to the M host.

PS:
Actually, the interesting aspect of this additional functionality lies in the ability to stack forward and reverse tunnels on top of an SSH proxy connection. In general, by mapping a certain SSH service in the target network, we can access other targets within this network. It's like giving me an SSH service, and I can give you

access to an internal network. The downside is that building multiple layers of tunnels using command-line parameters can be cumbersome. Readers who have a need for this functionality may want to first understand the relevant points of the `.ssh/config` file.

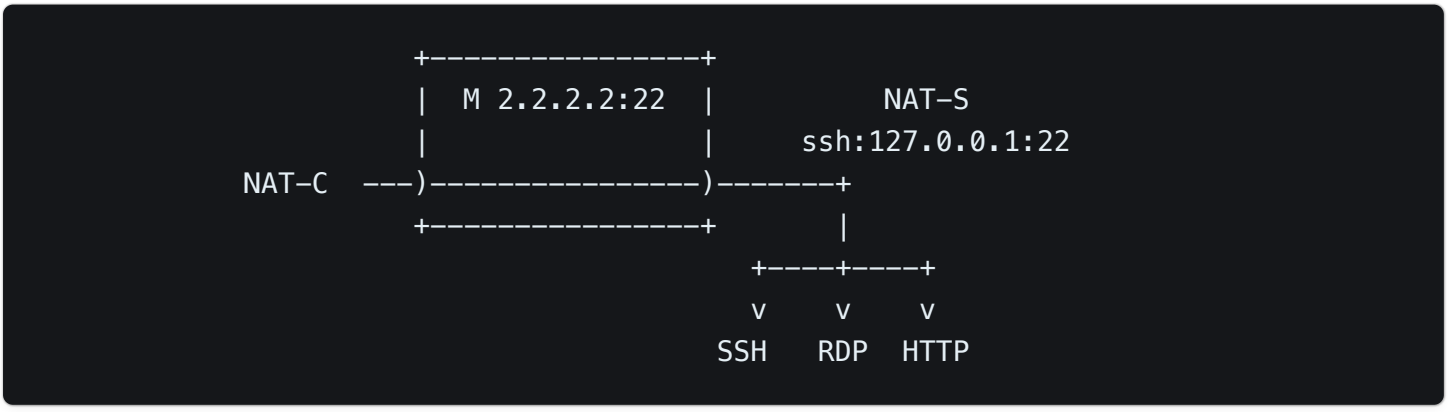
Review

Through the review in this article, we know that SSH service is a very powerful foundational service. Besides remote management of Linux/Unix hosts, it also provides various capabilities for establishing network tunnels, enabling the construction of complex and efficient network tunnels to meet daily work demands.

Exercises

Finally, we have prepared some post-lesson exercises. Interested friends can check their learning progress on their own and exchange answers in the Slack group.

Details



Assumption: Suppose there is a public host M (2.2.2.2:22), and two hosts NAT-C and NAT-S on different internal networks, with an SSH service running on NAT-S.

Question 1: Now, if we want to access network resources reachable by NAT-S from NAT-C using SOCKS proxy, how should we organize the commands?
Question 2: If I want to access host D (3.3.3.3:22) based on NAT-S as a "jump host", how should I organize the commands?

Slack Link :https://join.slack.com/t/xlab-tunnel/shared_invite/zt-2jpzgp3kl-CrrHDT705_nACkgMfptuBg