

使用 SSH 服务打破网络边界



rootkiter

2024年5月31日 • 20 min read

缘起

场景考量

正式开始

1. 正向端口转发(从本地访问服务端可达服务).
2. 反向端口转发(从服务端访问本地可达服务).
3. 利用 SSH 服务的主机作为跳板, 访问其他主机的SSH(SSH 跳板登陆).
4. 正向 socks 代理建立(服务端开放socks代理, 供本地使用).
- 5.1 反向 socks 代理建立(本地开放 socks 代理, 供服务端使用).
- i. 5.2 反向 socks 代理建立(远程ssh 版本过低, 不支持 socks 反向代理).
6. 利用公网 SSH 服务实现双 NAT 网络内主机的安全互访
7. 补充: 基于 socks 协议的 ssh 登陆

安全风险分析

总结

课后练习

缘起

部门近期接到一个项目, 需要在受限内网部署一套自研的系统, 受一些客观条件限制, 我们的同事又无法肉身直连, 本来大家是考虑用“向日葵”就可以了, 但是通过界面敲命令行属实有些卡顿, 这给项目推进带来了不小的困难, 如果能搭建一组网络隧道, 直连目标机器, 那工作起来一定丝滑不少. 考虑到过往经历, 面对这种撞枪口的需

求, 索性就打了一套 SSH 组合拳, 搭了几条解决不同需求的管道, 结果惊呆了小伙伴们, 既然如此, 不如索性系统性的梳理一下, 广而告之, 遂有了这篇blog.

场景考量

既然准备系统性梳理, 那就不能只解决某一个需求, 而是要考虑到更多的使用场景, 所以我大概考虑了以下几个场景, 供读者自行匹配:

1. 两个身处不同网域的小伙伴, 需要一个统一的网络管道, 共同开展系统搭建/程序开发/渗透测试等工作;
2. 在内网渗透场景下, 边界主机为内网主机, 希望以边界主机为跳板向更深层网络发起访问;
3. 进入受限网络后, 无法访问某个关键服务, 希望反向利用本地网络访问这个服务;
4. 在家里内网搭了个一键 XXX 服务, 出门在外想和小伙伴显摆一下;

一般面对这类场景使用 "EarthWorm/ngrok/frp/向日葵远控"等第三方工具肯定是第一个映入脑海的, 但转念一想这些工具在一些公司内, 是禁止使用的, 有些公司甚至直接将相应的网络特征列入了黑名单, 一旦使用就可能触碰公司红线, 那肯定就不能随使用啊. 如果此时能利用 SSH 服务解决上述问题, 那就很妙了, 因为 SSH 是日常工作中最常用到的基础服务, 可以说每一台服务器都有 SSH 也不为过, 甚至一些家用路由器都有这东西, 相对来说安全性也比上面提到的第三方工具高出好几个量级. 虽然前几天 xz 后门事件闹的人人自危, 但如果换成某个第三方工具出现类似问题, 想在第一时间知道问题所在并修复, 那肯定是极为困难的.

所以本文将相对系统的梳理出关于 SSH 隧道的更多知识点, 供读者解决上述场景的需求, 为读者拓宽 SSH 在网络隧道领域中更多的高级用法, 这些知识点包括:

1. 正向端口转发(从本地访问服务端可达服务)
2. 反向端口转发(从服务端访问本地可达服务)

3. 利用 SSH 服务的主机作为跳板, 访问其他主机的SSH(SSH 跳板登陆)
4. 正向 socks 代理建立(服务端开放socks代理, 供本地使用)
5. 反向 socks 代理建立(本地开放 socks 代理, 供服务端使用)
6. 利用公网 SSH 服务实现双 NAT 网络内主机的安全互访(EW/向日葵远控等工具的平替)
7. 补充: 基于 socks 协议的 ssh 登陆

正式开始

在开始之前,

可能有读者会有所顾虑, 说他日常办公环境是 Windows, 没 SSH 怎么办, 这里服务部分,

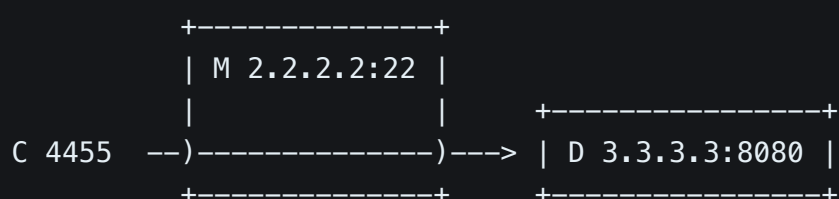
可以在云厂商挑个便宜的 VPS 解决, 客户端部分可以考虑用 WSL 子系统. 最不济有个

ssh 客户端也够用了. 本文的样例当然是以命令行为主, 相对的一些界面类工具也有类似的功能, 读者可以自行挖掘. 比如, 笔者曾经在 Xshell 上就使用过相似的功能, 不过自打那年 Xshell 被曝供应链攻击之后, 就不再使用了, 希望大家在使用界面类工具办公时一定要注意安全, 别为他人做嫁衣.

解除心理顾虑后, 那面就是准备学习环境, 需要准备一台远程 SSH 服务器 M, 再加上你正在使用的“本地机器 C”. 下面我们利用从 C 以不同参数选项登陆 M 的方式, 展开 1-5 的学习.

(章节中字符示意图的箭头, 均为数据流向)

1. 正向端口转发(从本地访问服务端可达服务)



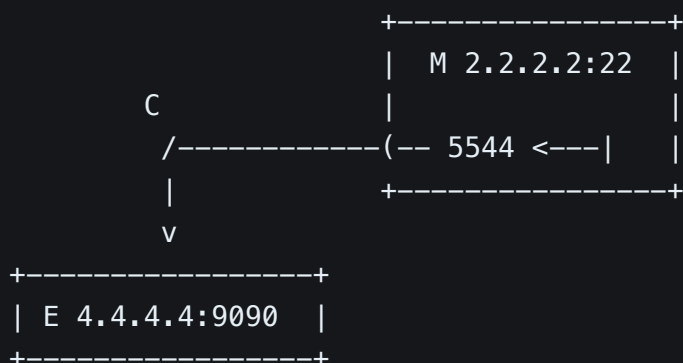
假设: 我们有一台公网 SSH 机器 M (2.2.2.2:22), 和一个待访问服务 D (3.3.3.3:8080), 我们的本地办公机(本机) C 希望通过 M 访问 D, 我们便可以在本机 C 用如下命令登陆 M.

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

登陆认证通过后, 本机 C 将监听 127.0.0.1:4455 端口, 服务则由 D(3.3.3.3:8080) 提供.

PS: 可能有些人知道, tor 服务默认就是监听在本地回环地址上的, 然而 tor 服务在国内又是无法直接使用的, 那位机智的小伙伴, 你想到什么了 [狗头].

2. 反向端口转发(从服务端访问本地可达服务)



假设: 我们有一台公网 SSH 机器 M (2.2.2.2:22), 和一个(本机) C, 以及一个待访问服务

E(4.4.4.4:9090), 我们现在想在 M 上通过 C 访问 E 的服务, 这就是一个反向转发的需求,

可以在本机 C 上用如下命令登陆 M.

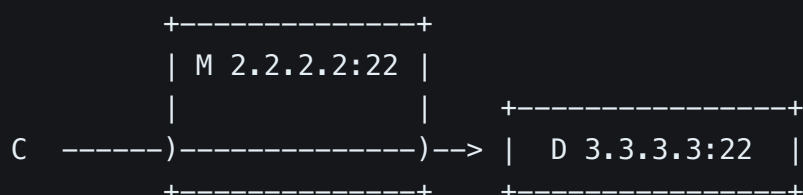
```
[client]$ ssh -R 5544:4.4.4.4:9090 ubuntu@2.2.2.2
```

登陆认证通过后, M 机器上将监听 127.0.0.1:5544 端口, 服务由 E(4.4.4.4:9090) 提供,

访问 5544 端口的服务时, 交互报文通过本机 C 实现中转.

PS: 如果我们将命令中 服务E 的部分换成 127.0.0.1 的回环服务地址, 就可以在 M 上访问 本机 C 提供的服务了.

3. 利用 SSH 服务的主机作为跳板, 访问其他主机的 SSH(SSH 跳板登陆)



通过 “1. 正向端口转发” 的了解, 我们掌握了正向端口转发的建立方法, 当待访问服务 D

同样也为 SSH 服务(3.3.3.3:22)时, 可以触发一个 SSH 客户端的特制选项(中继登陆/跳板登陆)

通过下面的命令, 我们就可以以 M 为跳板, 登陆 D 的 SSH 服务. 当然, 你需要先后通过 M 和 D 的登陆认证.

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

PS: 本节需要先后输入两个主机的密码认证, 如果你设置的密码略长, 就很容易输错抓狂,

所以我强烈建议, 日常使用公私钥认证的方式进行主机登陆, 可以免去很多烦恼. 当然

你的私钥要绝对的保密, 它的安全性取决于你的安全意识, 和科技发展的进度.

PPS: 熟悉 SSH 的小伙伴应该知道, 很多服务器会限制 root 账号远程登录, 仅接受本地回环地址登录, 但其实以服务自身的普通账号为跳板, 是可以登录 root 账号的, 感兴趣的可以尝试一下, 里面有个小坑坑, 看你能不能绕过去, 反正我当初可是想了半天才想清楚. 但这个技巧请不要随意用在自动化运维上, 小心出现事故哦.

4. 正向 socks 代理建立(服务端开放socks代理, 供本地使用)



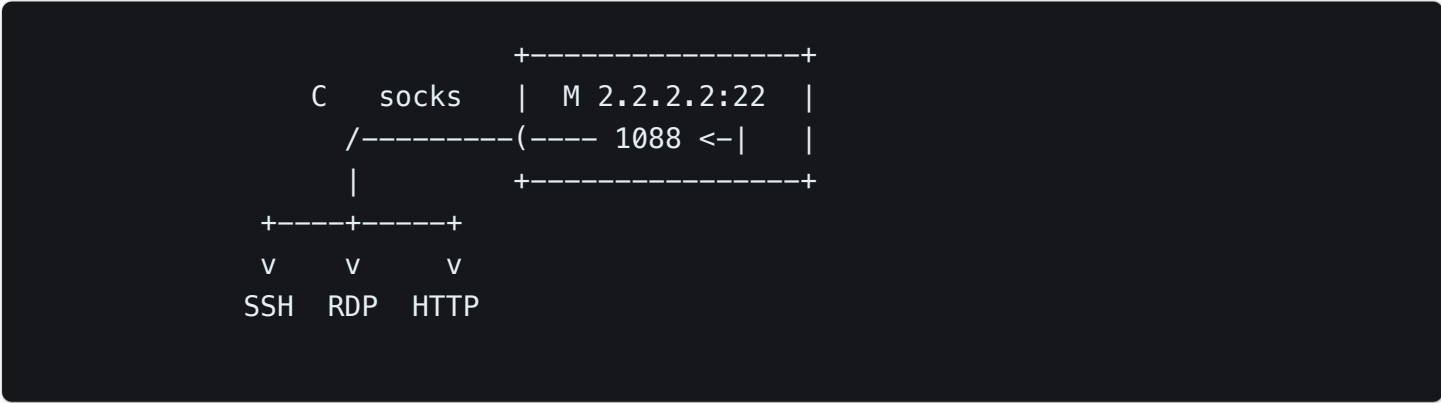
这部分内容, 和笔者之前写过的 [正向socks代理](#) 讲的是一个事, 由于 socks 协议是比较完善的代理协议, 所以常在各种规避流量审计的场景中出现, 很多爱好者会在浏览器上配置相关代理插件, 完成限制性服务的访问.

在 M (2.2.2.2:22) 上开启 socks 代理服务, 供本机 C 使用的登陆命令如下:

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

执行后, 会在本机 C 监听 127.0.0.1:1080 端口, 对应的 socks 服务由 M 主机提供.

5.1 反向 socks 代理建立(本地开放 socks 代理, 供服务端使用)



与 4 的条件相反, 如果远程服务器是一个相对受限的网络, 需要通过本机才能访问某些网络资源, 这时就可以考虑开启反向代理.

假设: 主机M (2.2.2.2:22) 是一台受限的 SSH 服务器, 本机 C 是一台网络资源充沛的客户端, 就可以用下面的命令登陆 M.

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

认证通过后, 远程主机 M 会监听 127.0.0.1:1088 端口, 对应的 socks 服务, 由本机 C 提供.

PS: 命令中参数选项 -R 同“2. 反向端口转发”是一样的, 区别在于参数值不同, 当指定待访服务时, 含义就是反向端口转发, 如果不指定待访服务, 就是反向 socks 代理.

5.2 反向 socks 代理建立(远程ssh 版本过低, 不支持 socks 反向代理)



ssh 的 socks反向代理选项是近年才增加的功能, 如果 C 主机上 ssh 版本过低, 很可能
会不支持这个选项, 此时可以考虑利用 2 + 4 组装一个反向代理 socks .

首先在 C 主机登陆自身, 利用 4 完成一个对自身正向代理的开放,
(此处要求, C 自身有一个可以登陆的SSH服务器),

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

随后将这个代理服务, 反向映射到 M 主机

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

这样 M 的 1088 端口上就是 C 开放的代理服务了.

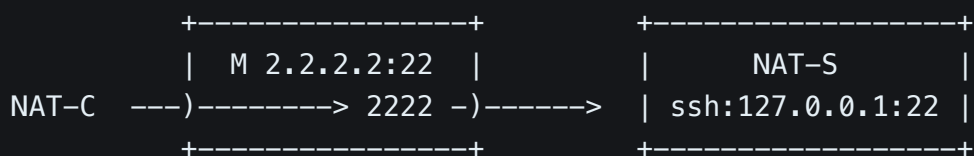
6. 利用公网 SSH 服务实现双 NAT 网络内主机的安全互访

梳理到这一步, 可能部分智慧读者, 已经想到双内网主机互访的方法了, 没错只需要待访

目标把服务反向映射到公网 VPS, 然后再用 C 访问这个映射, 就齐活了.

可以说这一步是根据各自不同需求, 融合 1-5 的能力, 可以组装出非常多的映射方案.

这里提供一个相对简单的方案供参考:



假设: 家里有一台主机 S, 公网有一台 SSH 服务器 M (2.2.2.2:22), 我手头现在有一台本机 C. 那么需要先将家里主机的 SSH 服务映射到 M 主机 127.0.0.1:2222 上, 命令如下:

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

然后在本机 C 利用跳板机登陆的方式登陆 M 的 127.0.0.1:2222 端口即可, 命令如下:

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

这里给出的是 C 访问 S 的 SSH 服务的一种简化的版本, 在一些场景下可能不适用, 这时你就要利用 1-5 拼接一个符合的隧道方案了, 在不考虑网速/带宽的情况下, 你可以考虑把 RDP-3389 / VNC-5900 等类型的服务进行映射, 这里就不一一展开了.

如果读者想要反过来从 S 访问 C 的 SSH 服务, 那就角色互换, 就成了, 将 C 某个服务也映射到 M 主机上即可.

7. 补充: 基于 socks 协议的 ssh 登陆

在一些场景下, 我们期待以一个固定的 S 为 socks 跳板, 跳到不同主机的 SSH, 实现再次配置, 这时, 我们就需要通过 socks 代理登陆 SSH 服务器.

PS: 由于这一部分需要借助 nc 的 代理连接功能(nc -X), 考虑到这是 nc 中一个比较新的选项, 所以此处分两种情况进行讨论:

本地 C 就拥有可用 nc

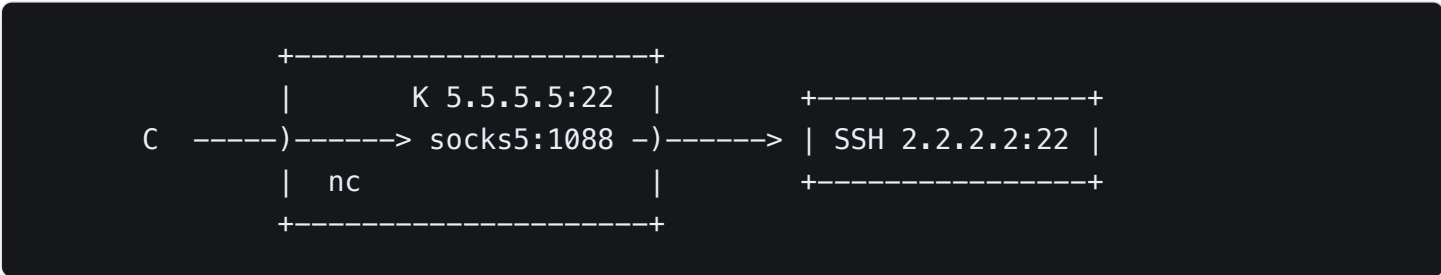


假设我们在 127.0.0.1:1080 上有一个 socks 服务, 我们要利用它登陆 M(2.2.2.2:22) 那么应对这种场景的命令如下:

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

通过认证后, 就达成了通过 socks5 代理, 登陆 2.2.2.2 的目标.

本地 C 没有 nc, 但居中主机 K 有可用 nc



此刻, 假设已经将 S 的 socks 服务映射到了 K(ssh:5.5.5.5:22) 的 1088 端口上, 那么我们就可以通过下述命令, 使用 S 的 socks 代理连接 M(2.2.2.2:22):

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

这里需要先通过 K 的认证, 然后就可以登陆 M 主机了.

PS: 其实这部分补充功能比较有意思的点在于, 在代理连接 ssh 的基础上, 可以再继续叠加正反向隧道, 也就是一般情况下, 我们只需要将目标网络中的某个 ssh 服务映射出来, 就可以对这个网络内的其他目标进行访问了, 有点给我一个 ssh 服务, 就可以还你一个内网的意思. 弊端就是用命令行参数建多层管道比较繁琐, 对这部分有需求的读者可以先了解一下 `.ssh/config` 文件的相关知识点, 有了这个配置文件的加持, 就可以实现无限代理层叠加, 将数据流向变得极为复杂难以溯源.

安全风险分析

至此, 基于SSH 搭建网络隧道的全部内容, 就全部梳理完了, 笔者作为安全从业者, 自然要评估一下, 相比以往双内网互访的解决方案来说, 此种隧道方案可能引入哪些风险, 便于我们使用时, 做到心中有数.

前提假设

从攻击面分析, “6. 利用 SSH 服务达成双内网主机互访” 的场景中引入的关键点就在于

“居中 SSH 服务器”这个角色, 它在场景中起到绝对核心的作用, 而两端主机, 各自处在自己的内网保护下, 默认是安全的, 如果它已经被攻陷, 那也和本章内容无关.

所以, 我们直接讨论 “SSH服务器完全被黑客攻破, 且换了一个被定制过的 SSH服务器,

攻击者在上面为所欲为” 的情况下, 是否会引入新的威胁.

认证风险

先看 SSH认证部分, 发起端 C 和服务提供端 S 是各自主动连接到 SSH 服务的, 如果各自通过密码完成SSH登陆认证, 那么这个 SSH 服务的密码会被攻击者掌握,

然而此时 SSH服务 早就陷落了, 这个密码的泄露是过去式, 不是新的威胁. 如果双方是
通过公私钥完成无密认证, 那么攻击者只能拿到公钥, 脱离私钥的公钥, 则没什么大用.

监听风险

再看隧道中传递的报文内容是否会被监听, 以 “C 通过隧道登陆S机器上的SSH服务” 为例,

在隧道建立过程中, 由 S 将自己的服务映射到失陷主机的某一端口, 供 C 登陆访问, 此时 C 利用隧道登陆的是 S 提供的SSH服务, 失陷主机看到报文则是 C/S 协商后的 SSH 报文, 居中的失陷SSH服务器起到的是媒人的角色, 它负责把S的服务端口介绍给C,

而隧道内的通讯安全则由 S 提供的服务进行维护. 如果 S 提供的不是 SSH 而是非加密

信道的服务, 则失陷主机可以看到双方的明文报文, 所以, 在实际使用中, 我们需要选择

一个提供了加密信道的服务进行隧道映射. 事实上我们日常生活工作中, 就算直接内网使用

也应该使用加密信道.

中间人风险

监听问题后, 还会衍生出一个中间人的问题, 如果失陷主机分别伪造 ssh 客户端和 ssh 服务器居中编瞎话, 客户端是否有手段识别, 我们还是回到 “C 登陆 S 的SSH服务” 这个场景, S 把 SSH 服务映射到M 后, 从 C 的视角看, M 映射服务指纹 和 S 的

服务器指纹是一样的, 而这个指纹在我们最初配置网络隧道的过程中, 是一定会被 C 记住的, 如果在某一刻链路中间出现了中间人, 这个指纹就会对不上, 已经记住的连接过程就会失败, 需要手动处理服务指纹, 才能重新登陆, 此刻 SSH 服务

指纹, 就可以起到一个示警的作用. 而如果你使用的居中 SSH 服务器从一开始就不安全, 就需要手动校验指纹是否匹配情况, 这一步也是可以人为确认的.

一般 Linux 上默认会把登陆过的 SSH 服务器指纹保存在 `~/.ssh/known_hosts` 文件内,

供各位行人肉确认.

第三方暴露风险

最后, 我们再看一下内网服务对外部的披露程度, 我假定操作过程中, 所有命令都是从本文提供的样例命令贴上去的, 那么此时, 内网的服务只会被映射到居中 SSH 主机的 127.0.0.1:xxxx 端口, 也就是回环地址监听的服务, 访问者必须从居中 SSH 主机内部才能发起访问, 当居中主机失陷后, 能够访问它的人除了系统管理员(也就是你), 以外, 再加一个攻击者, 对除此外的其他人还是处于封闭状态. 此刻服务披露程度增加了攻击者, 这就要求我们应该将经历过安全考验的服务(如 SSH 服务)映射出去.

SSH服务自身风险

这部分, 我们能做的就很少了, 尽量保证服务器的系统版本一直是最新版, 也就足够了, 好在 SSH 是个非常重要的服务, 如果它被披露漏洞, 会瞬间传遍大街小巷, 你一定会第一时间知道需要更新它了(参考近期的 xz 后门事件).

最终建议

基于以上分析, 我们可以发现, 即使是在 居中SSH服务 完全失陷, 这种极端环境下, C/S之间的通讯还仍然具有一定程度的防泄密能力, 如果能做到以下三点工作, 我们还是能够有足够的信心来保证两端的主机安全性的:

1. 居中 SSH 服务的登陆密码要独立于你的常用密码体系, 以免居中密码泄漏, 直接用这个口令进入你的两端主机, 采用公私钥认证最佳.
2. 被映射服务, 应当具备加密通讯的能力, 以防止服务器被攻陷后有人旁听.
3. 被映射服务, 最好是个能够抵御中间人攻击的服务, 比如 SSH 服务, 它的指纹系统就有告警中间人攻击的能力. 且第一次连接时, 要人为确认指纹信息的准确

性, 以免居中

服务器自始至终都是攻陷状态.

总结

通过本文的梳理, 我们知道 SSH 服务是个非常强大的基础服务, 它除了可以远程管理

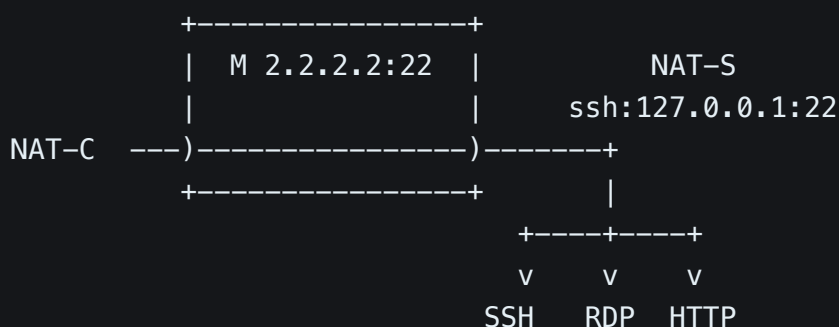
Linux/Unix 主机外, 还提供了多种架设网络隧道的能力, 借助它可以搭建复杂高效的网络隧道. 以满足日常的工作需求.

至此, “使用 SSH 服务打破网络边界-助力内网渗透” 的全部内容, 就结束了.

课后练习

blog 的最后, 笔者准备了一道课后练习题, 小伙伴们可以自行检查下学习效果, 我们还准备了一个 slack 群, 交流答案. 后续这个群会转为隧道知识交流群, 供隧道类的知识和技巧交流.

练习题




假设: 现在有一台公网主机 M(2.2.2.2:22), 和两台不同内网的主机 NAT-C 和 NAT-S ,
其中 NAT-S 上有一个 SSH 服务;


问1: 现在我们想从 NAT-C 以 socks 代理的方式访问 NAT-S 可达的网络资源, 应该如何组织命令.


问2: 如果我想基于 NAT-S 以 "跳板登陆" 的方式, 访问 主机 D(3.3.3.3:22), 又该如何组织命令.


交流群 :https://join.slack.com/t/xlab-tunnel/shared_invite/zt-2jpszgp3kl-CrrHDT705_nACkgMfptuBg


What do you think?
7 Responses



Upvote


Funny


Love


Surprised


Angry


Sad

0 Comments

 Login ▼

G

Start the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS 

Name

 Share

Best Newest Oldest