漏洞类型 (Vulnerability Type)	核心原理	典型的系统调用特征 (Typical Syscall Signatures)	分析要点与实例
命令/代码注入 (Command/Code Injection)	应用程序将不受信任的外部输入（如用户请求）当作命令或代码来执行。	1. execve() 的意外调用: 一个本不该创建新进程的进程（如Web服务器的工作进程、数据库进程）突然调用了 execve。2. 可疑的子进程: fork() 或 clone() 之后紧跟着一个 execve，并且执行的是 /bin/sh, /bin/bash 等Shell解释器。3. 参数异常: execve 的参数中包含来自网络请求的痕迹，例如 wget, curl, nc 等命令。	这是最明显、最经典的特征。• 实例: Shellshock漏洞。Apache的CGI进程在处理恶意环境变量后，会直接调用 execve 来执行攻击者注入的命令，这个调用在正常的业务流程中完全不存在。
内存损坏(Memory Corruption)	通过缓冲区溢出、越界写、UAF（Use-After-Free）等手段破坏程序的内存结构，通常目标是劫持程序的控制流。	1. 程序崩溃 (SIGSEGV): 在 read(), recv(), recvfrom() 等接收输入的系统调用之后，程序很快因为访问无效内存而崩溃。2. 劫持控制流后的 execve: 攻击成功后，如果payload是启动一个shell，那么最终特征会和“命令注入”一样，出现一个意外的 execve。3. mmap() /mprotect(): 攻击者为了执行shellcode，可能会调用 mmap 分配一块新的内存，并用 mprotect 将其标记为可执行（PROT_EXEC）。	内存损坏本身发生在用户态，系统调用层面不直接可见。我们观察到的是其后果。• 要点: 关注输入型syscall (read, recv*) 和程序控制流改变（execve）或崩溃之间的关系。一个Web服务器进程调用 mprotect 并设置 PROT_EXEC 标志是极度可疑的。

权限与访问控制错误(Privilege & Access Control)	程序错误地授予了攻击者本不该拥有的权限，或允许其访问受限资源。	1. 权限提升: 一个非root进程成功调用了 setuid(0), setgid(0), setreuid(0, 0) 等函数，将其权限提升为root。2. 路径遍历(Path Traversal): open(), stat(), access() 等文件操作的syscall，其路径参数包含大量的 ../，试图访问预期目录之外的文件（如 open("/var/www/html/../../../etc/passwd", ...)）。3. 越权文件访问: 进程成功 open(), unlink(), chmod() 了一个它在正常逻辑下无权访问的敏感文件（如配置文件、密码文件、其他用户的文件）。	这类特征直接反映在系统调用的参数和返回值上。• 实例: Dirty COW漏洞利用成功后，最后一步就是修改 /etc/passwd 等文件，会产生对这些文件的 write() 调用。而提权成功后，最典型的行为就是调用 setuid(0)。
信息泄露(Information Leakage)	应用程序无意中将敏感信息（如内存数据、密钥、配置）泄露给攻击者。	1. 异常的 write()/sendto(): 在接收到一个简短的请求（通过 read/recvfrom）后，程序通过 write 或 sendto 向网络套接字（socket）发送了异常大的数据块。2. 数据内容不匹配: 发送的数据内容与应用层协议严重不符，可能包含文本、二进制代码、内存地址等混合的垃圾数据。3. 读取敏感文件: 进程 open() 并 read() 了它本不该访问的配置文件、密钥文件或设备文件（如 /dev/mem），随后将数据通过网络发送出去。	特征在于输出型系统调用的数据量和内容异常。• 实例: Heartbleed漏洞。服务器在收到一个小的心跳包后，会通过 write()/sendto() 返回一个巨大的（可达64KB）内存块，这就是最关键的syscall特征。
竞争条件(Race Condition)	程序的输出依赖于多个线程或进程之间不可预测的事件发生顺序。	1. 高频交错的系统调用: 多个线程/进程在极短时间内，对同一个资源（如文件、内存区域）进行反复、交错的操作。例如，open()/access() 和 write()/rename() 交替进行（TOCTOU攻击）。2. 特定序列的并发调用: 某些漏洞需要特定的系统调用序列并发执行才能触发。	竞争条件的特征非常难以捕捉，因为它不依赖于单个异常的调用，而依赖于特定调用序列的时间关系。需要分析并发线程的trace。• 实例: Dirty COW。一个线程循环调用 madvise(MADV_DONTNEED)，另一线程循环 write() 到 /proc/self/mem，这种高频并发的特定操作组合就是它的syscall指纹。

后门与Rootkit(Backdoor & Rootkit)	恶意代码修改操作系统行为以隐藏自身、提供持续性访问。	1. 系统调用被劫持 (Hooking): 使用 strace 等工具观察到的系统调用结果与实际不符。例如 ls 看不到某个文件，但 cat 却能打开它。这说明 getdents64 可能被劫持了。(注意：strace 本身可能被欺骗)。2. 异常的网络监听: 一个不该监听网络的进程(如 systemd 的某个子服务)调用了 socket(), bind(), listen()。3. 反向连接: 任何进程(尤其是系统服务进程)突然调用 socket() 并 connect() 到一个外部的、可疑的IP地址。	Rootkit的分析通常需要更底层的工具(如 bpftrace, SystemTap)来检查内核的系统调用表(sys_call_table)是否被修改。其特征是系统行为的根本性改变。• 实例: 一个钩住了 accept() 的后门，在接收到包含特定“魔法密码”的连接后，会直接调用 execve 启动一个shell。