ISC 2019 7th
第七届互联网安全大会

国产操作系统安全的现状与发展

深度科技

何谓国产操作系统

限定范围

- 通用操作系统，主要指桌面与服务器操作系统
- 不包含：IoT操作系统、嵌入式操作系统与超算系统

主要特征

- 除了x86架构外，还支持龙芯、申威、飞腾等国产处理器
- 现在均以Linux为基础系统，但各个处理器平台现状有差异

Linux安全子系统

- 访问控制/AC：主体（用户、进程）能访问哪些客体（进程、文件）
- 用户验证：现实实体成为系统内实体的准入管理，PAM
- 资源隔离：chroot、namespace
- 配额限制：ulimit、/etc/security/limits.conf、cgroups
- 系统沙箱：seccomp
- 可信计算：TPM/TCM/TPCM、IMA/EVM
- 安全审计：auditd
- 磁盘加密：基于dev-mapper的LUKS等
- 网络防护：iptables/nftables

访问控制

- 自主访问控制/DAC：用户控制自己所拥有的资源如何被访问
  - 基于inode与内核任务(task_struct)的用户凭证（cred）
  - ugo/rwx：chmod
  - 细粒度的访问控制管理：{s,g}etfacl，基于扩展属性
- 能力模型（POSIX capabilities）：细粒度的特权划分，基于扩展属性
- 强制访问控制/MAC：系统（管理员）设置哪类主体能访问哪类客体
  - 基于内核关键路径上函数埋点的LSM框架
  - 安全模型：BIBA/BLP
  - 主要实现：SELinux、AppArmor……

访问控制的现状

- 自主访问控制/DAC：设置繁琐，用户没有坚实的安全知识与全局观
- 能力模型（POSIX capabilities）：只能局部实施，全局实施会有大量的兼容性问题
- 强制访问控制/MAC：需要为每个软件配规则，跟踪软件版本升级，不适合桌面系统，系统管理员负担重，扩展性差，缺乏链式支持
- 各国产处理器平台均能支持

用户验证及其现状

- PAM：可插拔认证模块系统
  - 基于auth、account、session与password的四类认证库模块的组合
  - 不仅是认证：资源配额、会话超时、口令复杂度……
- 主要问题：
  - 传统PAM基于TUI，但桌面系统需要PAM支持GUI
  - 新技术的集成：Windows Hello、人脸、指纹、Ukey、虹膜、语音……
- 各国产平台支持良好

可信计算及其现状

- 主流内核已支持TPM（drivers/char/tpm）
- TCM/TPCM尚未进入主流内核，硬件芯片、主板与TSS均需改造与适配
- 主要问题：
  - 部分应用可用于保护敏感数据（如密钥）
  - 全面应用将导致计算机系统的维护相当繁琐
  - 从BIOS→bootloader→内核→进程的全链应用将明显降低系统性能
- 各国产处理器平台性能较低，全面支持性能开销更大

其它安全子系统

- 资源隔离：chroot → namespace
- 配额限制：进程与会话级别的ulimit/limits.conf → cgroups v1/v2
- 系统沙箱：基于bpf的seccomp，应可以外置管理
- 安全审计：涉及系统调用，与架构相关
- 磁盘加密：基于dev-mapper
- 网络防护：iptables → nftables，内核会更偏向eBPF（如XDP）
- 各国产平台支持良好，功能与x86平台基本一致（除了eBPF-jit）

主要问题

- 事后防护，需要考虑极多兼容性问题
- 配置繁琐易错，维护工作量大，如MAC
- 性能开销大，在国产处理器平台上更明显，如可信计算
- 整体性差，扩展性差，如链式LSM，PAM对GUI与新技术的支持
- 在国产平台上涉及到ISA部分有差异，如eBPF-jit

新的思路

- 微内核，如Fuchsia、seL4、以及鸿蒙
- 形式化验证，如seL4，以及鸿蒙
- 安全的系统开发语言，如rust
- 新的访问控制模型，如Capability
- 软硬结合，如TEE、SGX、NXE、MPX、CET等
- 新的分析/检验方法，如基于边界的控制流/数据流分析，Fuzzer测试，基于DL的漏洞分析
- 新的应用执行环境与全生命周期管理

微内核

- 内核代码基极小，极便于验证与加固
- 避免驱动许可证的问题（类似HAL/UIO）
- 上下文切换开销大，在国产平台上可能导致更严重的性能损耗
  - Fuchsia：开发中，性能开销大
  - seL4：仅用于嵌入式系统
  - 鸿蒙：开发中

形式化验证

- 显然不可能存在无安全隐患的系统，参考停机问题
- 从数学上证明某类bug的不存在性，如内存访问越界、空指针、死锁等
- 主要问题：
  - 工作量巨大，需要开发十倍以上的形式化验证代码
  - 仍需假定底层的安全性：处理器、内存、工具链等
  - 较大的性能开销，如禁用DMA、以及多核并发的处理
  - 仅能用在较小的代码基上（每天实际提交代码不超过1000行？）

安全的系统开发语言

- C语言贴近RAM模型的开发方式使得安全难以保障
  - 大部分安全问题源于内存访问安全、类型安全与并发安全
- rust语言
  - 无GC，细粒度的内存控制，适用于系统编程（如内核与VMM）
  - 通过ownership与lifetime在编译期保障内存、类型与并发安全
  - 通过安全甚至提供了更好的性能
  - 百度的mesalock/mesatee，微软MSRC，ChromeOS的crosvm

新的访问控制模型（Object-based Capability）

- 访问资源时由系统创建描述符，应用没有通用的资源路径
- 避免了应用程序以用户的名义（权限）进行不安全（误？）的操作
- 操作系统：seL4、Fuchsia、Midori
- 其它系统：编程语言、库、应用软件
- 可能的问题：
  - 新的访问控制方式，需要重写软件
  - 配置过于繁琐

应用环境与生命周期管理

- 安全问题主要在用户态
  - 应用软件总代码量更大，总开发人员更多
  - 应用软件处于更外部，更易受到攻击
  - 应用软件数据价值更高
- 动态链接与软件间共用文件导致了牵一发而动全身的依赖关系
- 软件没有完整的生命周期控制，来源不明导致安全问题
- 参考：
  - iOS与Android的应用与系统，应用与应用之间相互隔离
  - iOS的应用生命周期有完整的验证

ISC 2019 第七届互联网安全大会

Thank You

2019.8.20