Security and Protection

Operating Systems
Wenbo Shen

Some slides are from IPADS@SJTU, Prof. Qingni Shen@PKU, and CS 587 by Jon A. Solworth@ UIUC

Disclaimer

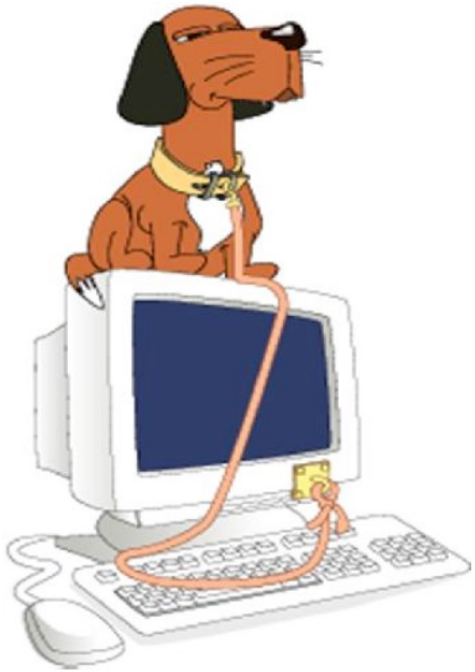
- Many images used in this lecture are collected from the Internet, and they are used only for the educational purpose. The copyright belong to the original owners, respectively.

Outline

- Security evaluation criteria
  - TCSEC, ITSEC, CC, GB17859
- Common concepts
  - Trusted computing base
  - Attack surface
  - Defense in-depth
- Protection - Access Control
  - Authentication
  - Authorization
  - Auditing
  - Reference monitor
- Security
  - Attacks and defenses

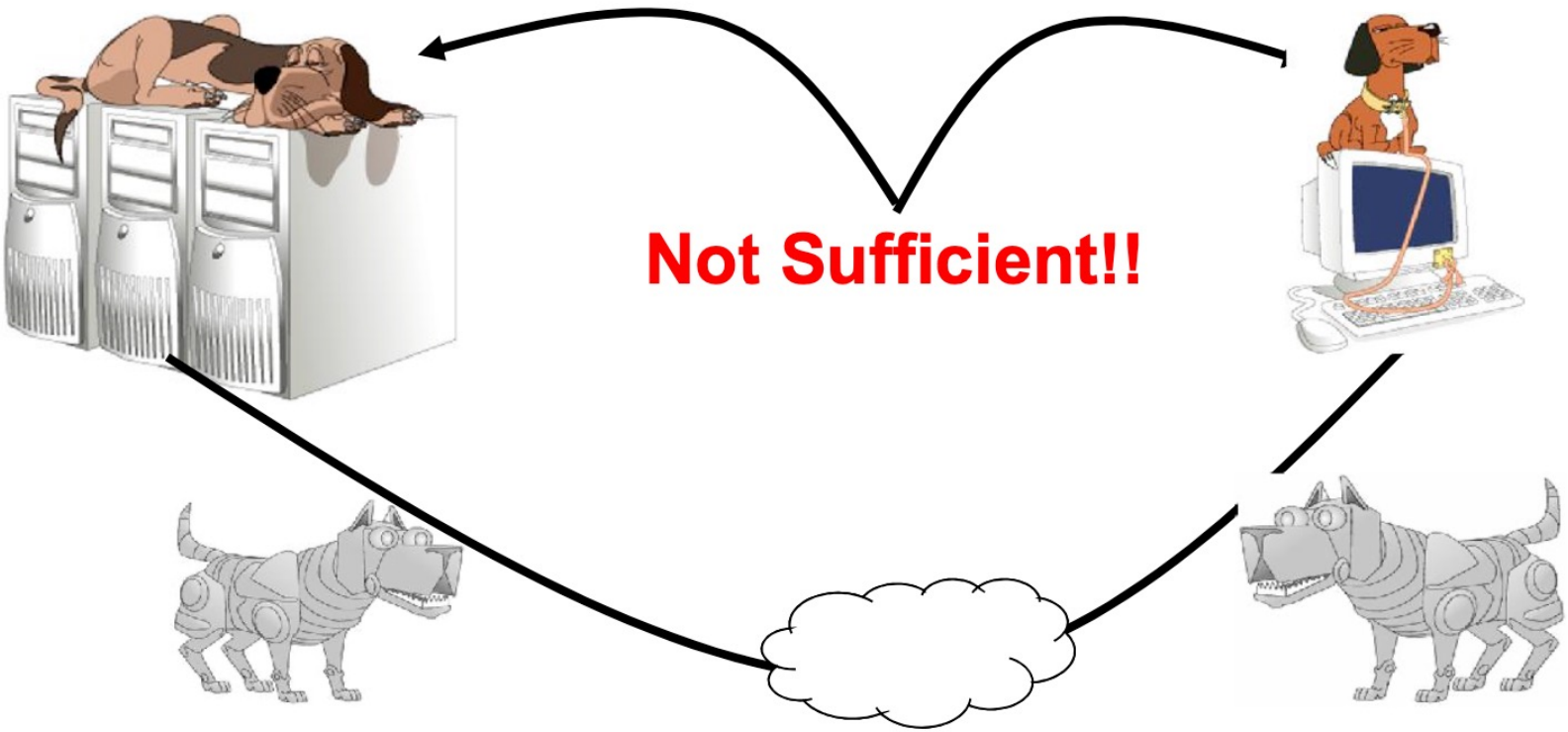
Computer Security

- Computer security is about provisions and policies adopted to protect information and property from theft, corruption, or natural disaster.
  - while allowing the information and property to remain accessible and productive to its intended users.
- Security of computers against intruders (e.g., hackers) and malicious software (e.g., viruses).



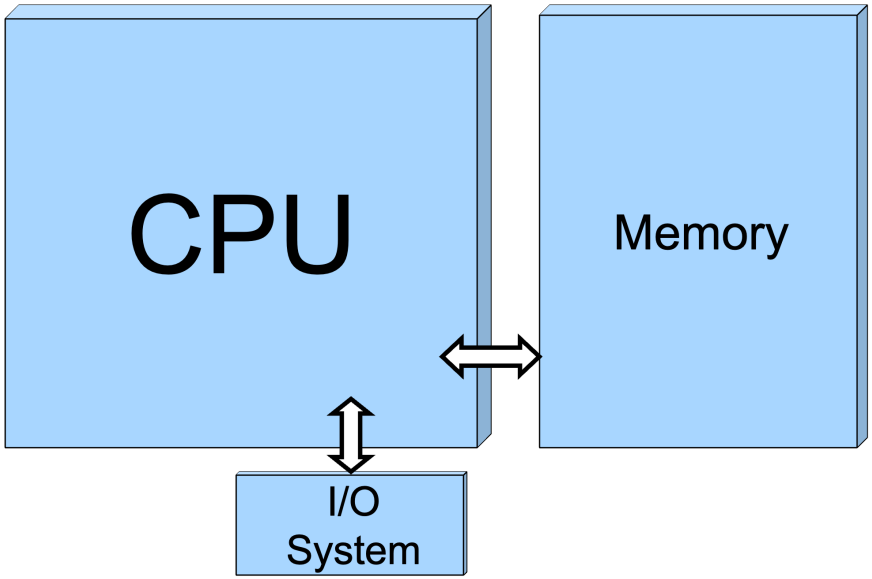
Network Security

- Network security on the other hand deals with provisions and policies adopted to prevent and monitor unauthorized access, misuse, modification, or denial of the computer network and network-accessible resources.



Security is deeply rooted in OS

- OS concept
  - Resource abstractor and allocator
  - Allocation needs control, which is security
- CPU introduced privileged mode
  - Kernel mode vs user mode
- Memory introduced partition and paging for memory isolation
  - Partition – Base and limit
  - Paging – Page table
- IO introduced access control list
  - Permission control



Why need security at the OS level?

- 系统中有许多需要保护的数据
  - 如账号密码、信用卡号、照片视频等
  - 操作系统需要允许这些数据被合法访问，但不允许被非法访问
- 系统中可能存在许多恶意应用
  - 操作系统需要与这些恶意应用作斗争，保护自己，限制对方
- 操作系统不可避免的存在漏洞
  - 操作系统需要考虑自己被完全攻破的情况下依然提供一定的保护

These jeopardize operating systems' fundamental functionality, i.e., resource allocation.

Security in Operating System

- Security refers to providing a protection system to computer system resources such as:
  - CPU, memory, disk
  - software programs and
  - most importantly data/information stored in the computer system
- So a computer system must be protected against
  - unauthorized access by users and
  - malicious access to system including viruses, worms etc...

Who are the attackers?

- Vandals (Hackers, crackers) driven by intellectual challenge.
- Insiders: employees or customers seeking revenge or gain informal benefits
- Natural disasters: flooding, fire, storms, earthquake...
- Criminals seeking financial gain.
- Organized crime seeking gain or hiding criminal activities.
- Terrorist groups or nation states trying to influence national policy.
- Foreign agents seeking information (spying) for economic, political, or military purposes.
- Tactical countermeasures intended to disrupt military capability.
- Large organized terrorist groups
- Cyber attacks

What are the vulnerabilities?

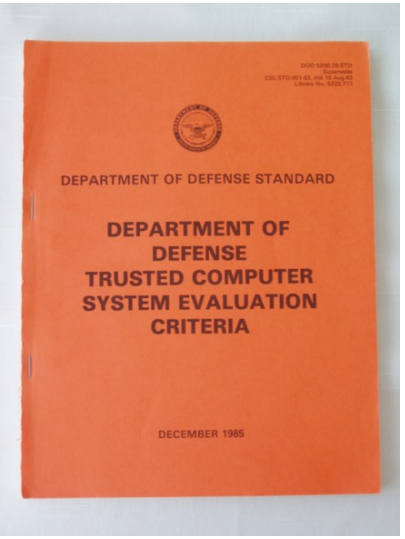
- Physical vulnerabilities (Eg. Computer can be stolen)
- Natural vulnerabilities (Eg. Earthquake)
- Hardware and Software vulnerabilities (Eg. Failures)
- Media vulnerabilities (Eg. Hard disks can be stolen)
- Communication vulnerabilities (Ex. Wires can be tapped)
- Human vulnerabilities (Eg. Insiders)
- Poorly chosen passwords
- Software bugs (non reliability of software) – buffer overflow attacks

System security evaluation criteria

- TCSEC
  - Trusted Computer System Evaluation Criteria
- ITSEC
  - Information Technology Security Evaluation Criteria
- CC
  - Common Criteria
- GB17859

Trusted Computer System Evaluation Criteria

- Trusted Computer System Evaluation Criteria (TCSEC)
  - By DoD in 1983
  - The Orange Book
  - Set the system security evaluation criteria
  - Four division: D, C, B, A
    - D – Minimal protection
    - C – Discretionary protection
    - B – Mandatory protection
    - A – Verified protection



- Rainbow Series
  - https://en.wikipedia.org/wiki/Rainbow_Series

Trusted Computer System Evaluation Criteria

级别	系统的安全可信性
A1	形式化认证，安全级别最高
B3	全面的访问控制、可信恢复
B2	良好的结构化设计、形式化安全模型
B1	强制存取控制（MAC）
C2	较完善的自主存取控制（DAC）、审计
C1	自主存取控制
D	没有安全防护

操作系统	安全级别
SCO OpenServer	C2
Linux	C2
Windows Server 2008/2003	C2
Saloris	C2
DOS	D
Multics (Failed)	B2

Information Technology Security Evaluation Criteria

- 1990年5月，法国，德国，荷兰和英联邦发布标准ITSEC(Information Technology Security Evaluation Criteria).
- 1991年6月，由欧共体发布1.2版本，使评估和认证机制具有可操作性。
- ITSEC 是一套评估产品和系统安全的标准，每次评估通过全面而广泛的功能性测试和渗透测试的方式对IT安全属性进行详细的检查。
- 首次提出了机密性、完整性和可用性的概念
- 首次提出ST (Security Target)的概念
- 将安全性要求分为“功能”和“保证”两个部分：
  - 功能要求：为满足安全需求而采取的一系列技术安全措施，如访问控制、审计、鉴别等
  - 保证要求：确保功能正确实现及有效性的安全措施，如渗透测试，安全脆弱性分析等。
- 分别衡量安全功能与安全保证要求
  - 安全功能等级 F1-F10
  - 安全保证等级 E0-E6

CC - Common Criteria

- CC通用评估标准的发展
  - 1991年，在欧共体的赞助下，英、德、法、荷制定的共同标准《信息技术安全评定标准》ITSEC
  - 1996年，发布了通用安全测评标准CC (Common Criteria for IT Security Evaluation)1.0版，以美国TCSEC, FC、加拿大CTCPEC及欧共体ITSEC为基础,六国七方(美国NSA和国家技术标准研究所NIST、加、英、法、德、荷)共同提出.
  - 1997年，颁布CC标准的2.0版
  - 1999年，由ISO确立为国际标准，即ISO/IEC 15408。
- 针对在安全性评估过程中信息技术产品和系统的安全功能及相应的保证措施，**CC 提供了一组通用要求，使各种独立的安全评估结果具有可比性。**
- 评估过程为满足这些要求的**产品和系统的安全功能**以及相应的保证措施**确定一个可信级别**。
- 评估结果可以帮助用户确定信息技术产品和系统对他们的应用而言**是否足够安全**，以及在使用中**隐藏的安全风险是否可以容忍**。

我国相关标准的发布

- 中国国家质量技术监督局，中华人民共和国国家标准：计算机信息系统安全保护系统等级划分准则，**GB17859**-1999,1999.
- 中国国家质量技术监督局，中华人民共和国国家标准：信息技术信息安全技术 信息技术安全性评估准则，GB/T 18336.1~3-2001，2001.
- 中国国家质量技术监督局，中华人民共和国国家标准：信息安全技术 操作系统安全技术要求。GB/T 20272—2006，2006。
- 中华人民共和国公安部，计算机信息系统安全等级保护操作系统技术要求， GA/T 388-2002, 2002
- 中华人民共和国公安部，计算机信息系统安全等级保护通用技术要求， GA/T 390-2002, 2002

CC/GB17859/TCSEC/ITSEC对比

CC	中国 GB17859	美国 TCSEC	英国 FC	加拿大 CTCPEC	欧洲 ITSEC
EAL1		D	-	-	-
EAL2	第一级：用户自主保护级	C1	-	-	E1
EAL3	第二级：系统审计保护级	C2	T-1	T-1	E2
EAL4	第三级：安全标记保护级	B1	T-2	T-2	E3
		-	T-3	T-3	-
		-	T-4	-	-
EAL5	第四级：结构化保护级	B2	T-5	T-4	E4
EAL6	第五级：访问验证保护级	B3	T-6	T-5	E5
EAL7		A1	T-7	T-6	E6
-		-	-	T-7	-

望安科技

Concepts in system security

- 可信基（Trusted Computing Base）
  - 为实现计算机系统安全保护的所有安全保护机制的集合
  - 包括软件、硬件和固件（硬件上的软件）
- 攻击面（Attacking Surface）
  - 一个组件被其他组件攻击的所有方法的集合
  - 可能来自上层、同层和底层
- 防御纵深（Defense in-depth）
  - 为系统设置多道防线，为防御增加冗余，以进一步提高攻击难度

Trusted computing base (TCB)

- Think carefully about what you are trusting with your information
  - if you type your password on a keyboard, you are trusting:
    - the keyboard manufacturer
    - your computer manufacturer
    - your operating system
    - the password library
    - the application that is checking the password
- TCB = set of components (hardware, software) that you trust your secrets with

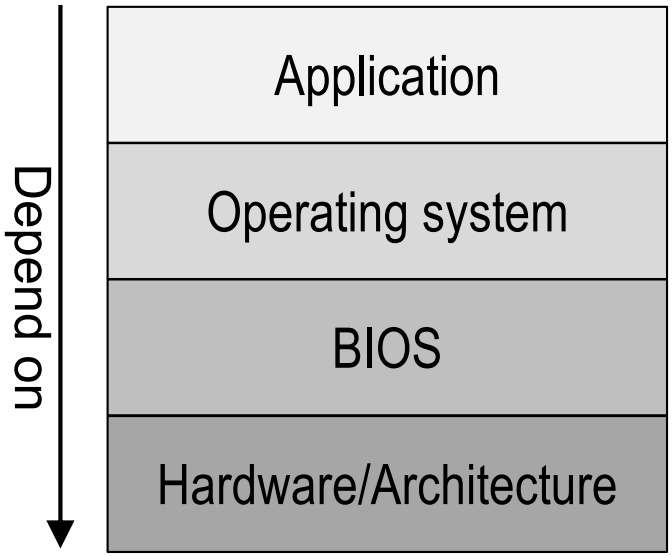
Trusted computing base (TCB)

- Trusted computing base
  - The trusted computing base (TCB) is that part of the computer system which you trust (if it fails can impact security)

- TCB issues
  - The larger the TCB is, the more difficult it is to make it secure.
  - The TCB should be as simple as possible
    - Minimize misuse
    - Enable better verification

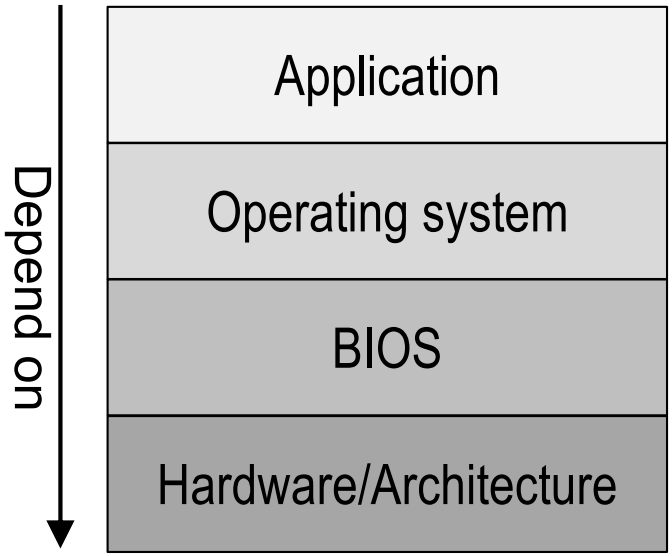
TCB in layered systems

- Systems built in layers
- Higher levels depend on lower levels, but lower levels do not depend on higher levels.
  - Hence, if component C depends upon Cj for its security and
  - Cj is insecure, then C cannot be secure.
- What is the TCB for Application, OS and BIOS?
- Since a component almost always depends upon its lower levels for security, the TCB usually includes all lower levels.
  - The lower the level that protections can be added the smaller the TCB.
  - The smaller the TCB, the easier it is to validate.



Attacking Surface

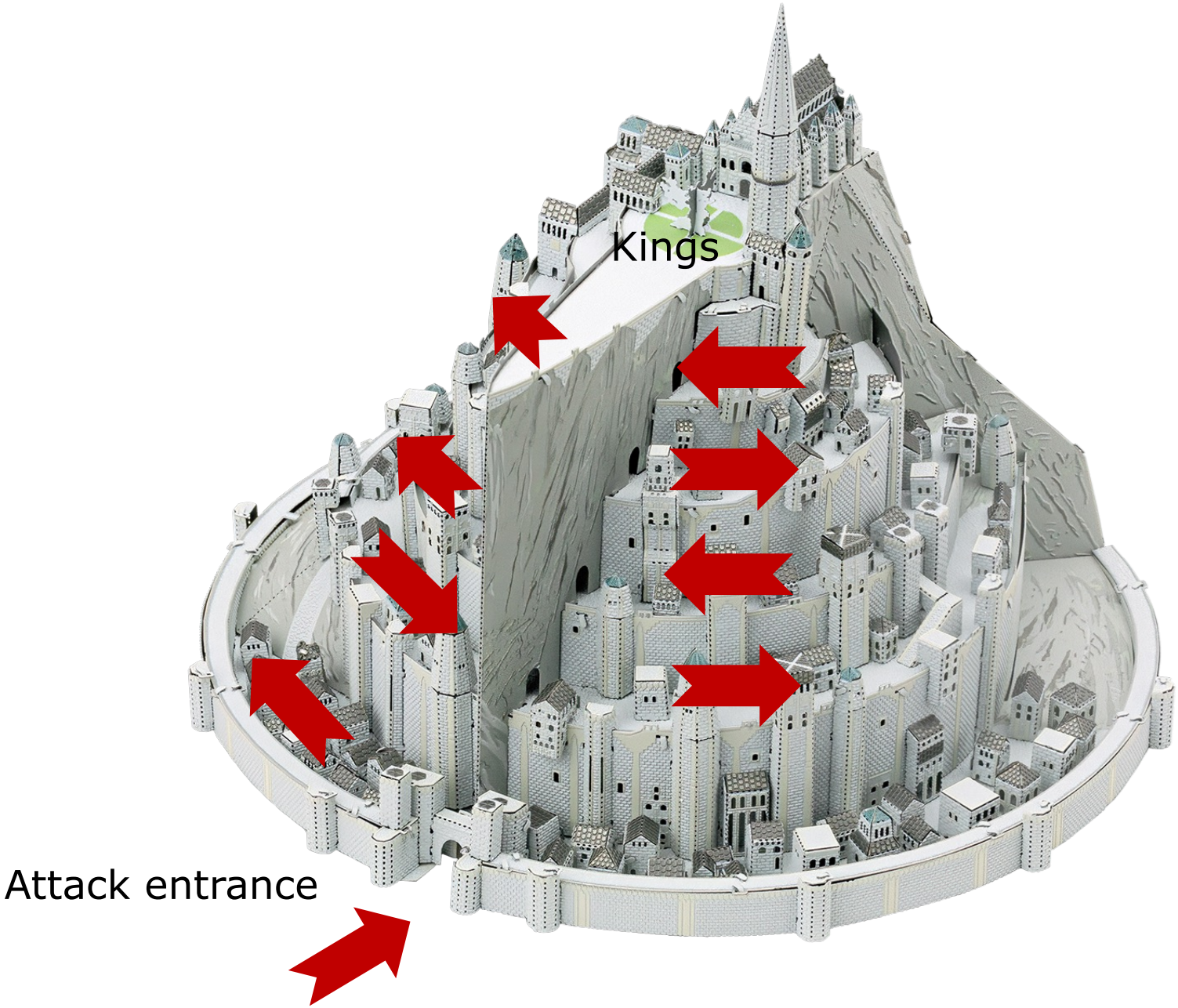
- The attack surface is the number of all possible points, or attack vectors, where an unauthorized user can access a system and extract data.
- The smaller the attack surface, the easier it is to protect.
- What is the attack surface of OS?



Defense in-depth

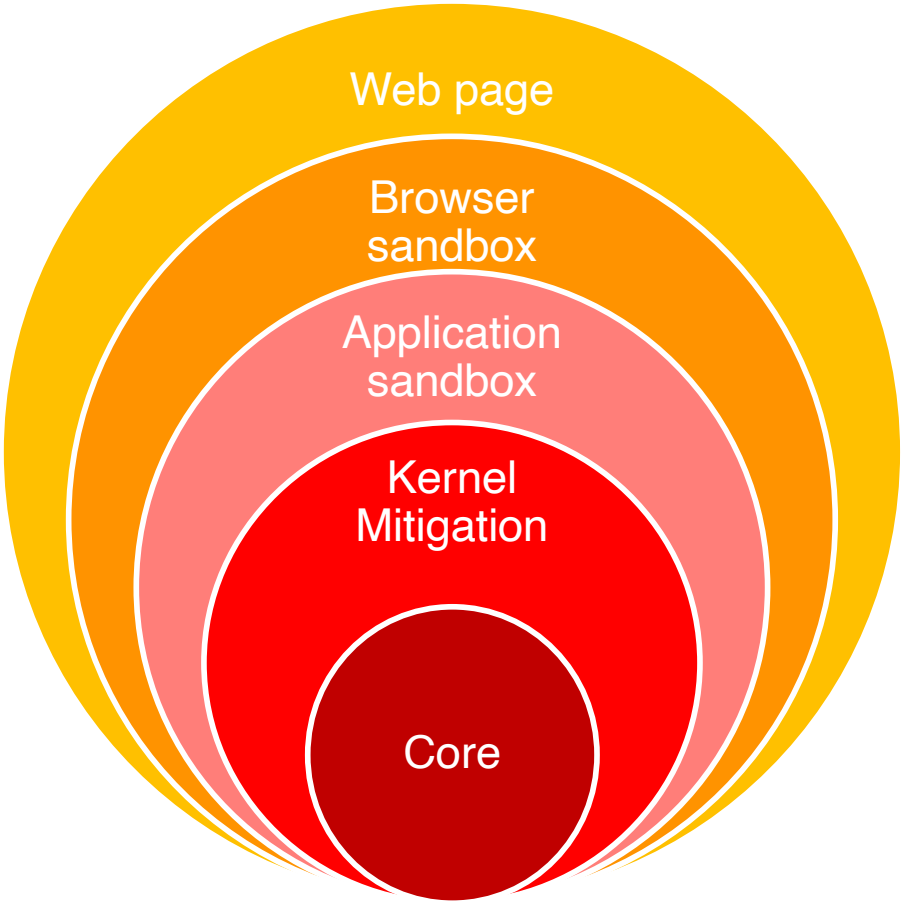
- What can you think of?
- Minas Tirith
  - The capital of Gondor at the end of the Third Age, lay at the eastern end of the White Mountains, built around a shoulder of Mount Mindolluin.

Defense in-depth



Defense in-depth

- Defense in depth is a concept used in information security in which multiple layers of security controls (defense) are placed throughout an information technology (IT) system.
- Its intent is to provide redundancy in the event a security control fails or a vulnerability is exploited.
- Get root from web page
  - Very hard
  - http://zerodium.com/program.html



Outline

- Security evaluation criteria
  - TCSEC, ITSEC, CC, GB17859
- Common concepts
  - Trusted computing base
  - Attack surface
  - Defense in-depth
- Protection - Access Control
  - Authentication
  - Authorization
  - Auditing
  - Reference monitor
- Security
  - Attacks and defenses

Access control

- Access control (访问控制)
  - Access control is an essential element of security that determines who is allowed to access certain data, apps, and resources—and in what circumstances.
  - 按照访问实体的身份来限制其访问对象的一种机制
  - 为了实现对不同应用访问不同数据的权限控制

Access control

- Authentication (认证)
  - identifying users and programs
  - 验证某个发起访问请求的主体的身份
- Authorization (授权)
  - determining what access users and programs have to things
  - 授予某个身份一定的权限以访问特定的对象
- Auditing – record what users and programs are doing for later analysis

Authentication 认证

- 知道什么 （Something you know）
  - 例如密码/口令、手势密码、某个问题的答案等
- 有什么 （Something you have）
  - 例如 USB-key、密码器等实物
- 是什么 （Something you are）
  - 如指纹、虹膜、步态、键盘输入习惯等属于人的一部分

- What does the computer do with this information?
  - Assign you an identifier
    - Unix: 32 bit number stored in cred structure
    - Windows NT: 27 byte number, stored in an *access token* in kernel

Authentication 认证

- 进程与用户之间如何绑定?
  - 每个进程的PCB/cred中均包含了uid字段
  - 每个进程都来自于父进程，继承了父进程的uid
  - 用户在登录后运行的第一个进程（shell），初始化uid字段
  - 在Windows下，窗口管理器会扮演类似shell的角色

```
struct cred {
        atomic_t        usage;
#ifdef CONFIG_DEBUG_CREDENTIALS
        atomic_t        subscribers;    /* number of processes subscribed */
        void            *put_addr;
        unsigned        magic;
#define CRED_MAGIC      0x43736564
#define CRED_MAGIC_DEAD 0x44656144
#endif
        kuid_t          uid;            /* real UID of the task */
        kgid_t          gid;            /* real GID of the task */
        kuid_t          suid;           /* saved UID of the task */
        kgid_t          sgid;           /* saved GID of the task */
        kuid_t          euid;           /* effective UID of the task */
        kgid_t          egid;           /* effective GID of the task */
        kuid_t          fsuid;          /* UID for VFS ops */
        kgid_t          fsgid;          /* GID for VFS ops */
```

Authorization 授权

- How does the system know what one user is allowed to do?
  - Authorization matrix:
    - Objects = things that can be accessed
    - Subjects = things that can do the accessing (users or programs)

	Alice	Bob	Carl
/etc	Read	Read	Read Write
/homes	Read Write	Read Write	Read Write
/usr	None	None	Read

Access Control Lists 访问控制列表

- 权限矩阵
  - 对象与实体的关系
- 矩阵有多大?
  - 假如系统中有 100 个用户，每种权限用 1 个 bit 来表示，那么每个文件都至少需要 300 个 bit 来表示 100 个用户 的 3 种权限
  - 假设这些 bit 都保存在 inode 中，通常 inode 的大小为 128-Byte 或 256-Byte，300 个 bit 相当于一个 inode 的 15% 至 30%
  - 每当新建一个用户的时候，都必须要更新所有 inode 中的权限 bit，不现实

	对象-1	对象-1	对象-3
实体-1	读/写	读/执行	读
实体-2		读/执行	读/写
实体-3	读		读/写

基于角色的访问控制（RBAC）

- RBAC 1996：将用户（人）与角色解耦的访问控制方法
  - Role-Based Access Control
  - 提出了角色的概念，与权限直接相关
  - 用户通过拥有一个或多个角色，间接地拥有权限
  - "用户-角色"，以及"角色-权限"，一般都是多对多的关系
- RBAC的优势
  - 设定角色与权限之间的关系比设定用户与权限之间的关系更直观
  - 可一次性地更新所有拥有该角色用户的权限，提高了权限更新的效率
  - 角色与权限之间的关系比较稳定，而用户和角色之间的关系变化相对频繁
    - 设计者负责设定权限与角色的关系（机制）
    - 管理者只需要配置用户属于哪些角色（策略）

授权机制：POSIX的文件权限

- 将用户分为三类
  - 文件拥有者、文件拥有组、其他用户组
  - 每个文件只需要用9个bit即可：3种权限（读-写-执行） x 3 类用户
- 何时检查用户权限？
  - 每次打开文件时，进行权限检查和授权
    - open()包含可读/可写的参数，OS根据用户组进行检查
    - 引入fd，记录本次打开权限（授权），作为后续操作的参数
  - 每次操作文件时，根据fd信息进行检查

```
drwx------+  8 wenbo  staff    256  4  6 14:54 Pictures
drwxr-xr-x+  5 wenbo  staff    160  3 10 08:56 Public
drwxr-xr-x   7 wenbo  staff    224  4  6 08:54 Workspace
```

最小特权级原则：setuid 机制

- 问题：passwd 命令如何工作?
  - 用户有权限使用 passwd 命令修改自己的密码
  - 用户的密码保存在 /etc/shadow 中，用户无权访问
  - 本质上是以文件为单位的权限管理粒度过粗——怎么解决?

- 解决方法：运行 passwd 时使用 root 身份（RBAC的思想）
  - 如何保证用户提权为root后只能运行passwd?
    - 在passwd的inode中增加一个SUID位，使得用户仅在执行该程序时才会被提权，执行完后恢复，从而将进程提权的时间降至最小
  - passwd程序本身的逻辑会保证某一个用户只能修改其自身的密码

```
-rwxr-xr-x   76 root   wheel    167136  4  2 00:46 strip
-rwsr-xr-x    1 root   wheel    137424  4  2 00:46 su
-r-s--x--x    1 root   wheel   1246528  4  2 00:46 sudo
-rwxr-xr-x    2 root   wheel    150864  4  2 00:46 sum
```

setuid的安全风险

- setuid 不同于RBAC
  - setuid在Linux下通常用于以root身份运行，拥有的权限，远超过必要
  - 必要权限：用户能够读写 /etc/passwd 文件中的某一行
  - 实际权限：
    - 用户能够访问整个 /etc/passwd 文件
    - 用户（短暂地）拥有root用户的权限
- setuid的安全隐患
  - 一旦 passwd 程序存在漏洞，如 buffer-overflow 导致的返回地址修改，则攻击者很容易以root身份通过ROP运行 execv("/bin/sh")

权限控制的另一种思路：Capability

- 提供细粒度控制进程的权限
  - 初衷：解决root用户权限过高的问题
  - Root用户可以bypass所有内核check
  - All or nothing is not good
  - System needs a **fine-grained** access control
- 基本的思想是把root的能力拆分，分为几十个小的能力，称为capability
  - Linux kernel v.2.2 introduced **Capabilities** (2000)
  - Up to now, 41 capabilities (0-40)

```
#define CAP_CHOWN            0

#define CAP_DAC_OVERRIDE     1

#define CAP_DAC_READ_SEARCH  2
```

```
#define CAP_BPF                 39


/* Allow checkpoint/restore related oper
/* Allow PID selection during clone3() *
/* Allow writing to ns_last_pid */

#define CAP_CHECKPOINT_RESTORE  40
```

权限控制的另一种思路：Capability

- 基本的思想是把root的能力拆分，分为几十个小的能力，称为capability
  - 语义都是预先由内核定义，而不允许用户进程自定义
  - 不允许传递，而是在创建进程的时候，与该进程相绑定
  - 没有提供 Capability ID，无法通过 ID 索引内核资源进行操作

- Capability表示一种能力
  - 例如：读取/foo文件，写入/foo文件，等等
  - 有点像钥匙，能打开某一把锁的话就能进行某个操作
  - 每个进程可以拥有一组能力

Linux的Capability

- Capability怎么实现?
  - 仅仅是一串bit，但必须保存在内核中，否则进程就可以任意伪造
    - 通常保存在进程的cred中，在进程进行某个操作的时候内核检查
  - 可以把不同Capability的组合对应为ACL中的不同组
  - 因此使用Capability的控制粒度可以很细，而且不需要建立大量的组

```
struct cred {
        ...
        kuid_t          uid;            /* real UID of the task */
        kgid_t          gid;            /* real GID of the task */
        ...
        kernel_cap_t    cap_inheritable; /* caps our children can inherit */
        kernel_cap_t    cap_permitted;  /* caps we're permitted */
        kernel_cap_t    cap_effective;  /* caps we can actually use */
        kernel_cap_t    cap_bset;       /* capability bounding set */
        kernel_cap_t    cap_ambient;    /* Ambient capability set */
```

Linux的Capability

- Capability怎么实现?
  - 在内核操作中进行权限检查

```
244
245  /*
246   * Check if the given command is allowed.
247   *
248   * Only a subset of commands are allowed for unprivileged users. Commands used
249   * to format the media, update the firmware, etc. are not permitted.
250   */
251  bool scsi_cmd_allowed(unsigned char *cmd, fmode_t mode)
252  {
253          /* root can do any command. */
254          if (capable(CAP_SYS_RAWIO))
255                  return true;
256
257          /* Anybody who can open the device can do a read-safe command */
258          switch (cmd[0]) {
259          /* Basic read-only commands */
260          case TEST_UNIT_READY:
261          case REQUEST_SENSE:
262          case READ_6:
```

fd与Capability的类似之处

- 文件描述符 fd 可以看做是一类 Capability
  - 用户不能伪造 fd，必须通过内核打开文件（回顾 file_table/fd_table）
  - fd 只是一个指向保存在内核中数据结构的"指针"
  - 拥有 fd 就拥有了访问对应文件的权限
  - 一个文件可以对应不同 fd，相应的权限可以不同
- fd 也可以在不同进程之间传递
  - 父进程可以传递给子进程（回顾pipe）
  - 非父子进程之间可以通过 sendmsg 传递 fd

Linux Capability问题

- 基本的思想是把root的能力拆分，分为几十个小的能力，称为capability
  - Linux kernel v.2.2 introduced **Capabilities** (2000)
  - Up to now, 41 capabilities (0-40)

- 然而，理想和现实往往时候差距的

Linux Capability问题

- 基本的思想是把root的能力拆分，分为几十个小的能力，称为capability
  - Linux kernel v.2.2 introduced **Capabilities** (2000)
  - Up to now, 41 capabilities (0-40)

- 理想状态：41个capabilities均分 root能力
- 现实：**CAP_SYS_ADMIN** 占据了1/3的 permission checks
  - Becomes the new root
  - https://lwn.net/Articles/486306/

- More problems

Linux Capability问题

- 使用混乱

```c
int scsi_ioctl(struct scsi_device *sdev, int cmd,
↪  void __user *arg)
{
  ...
  case SCSI_IOCTL_SEND_COMMAND:
    if (!capable(CAP_SYS_ADMIN) ||
    ↪  !capable(CAP_SYS_RAWIO))
      return -EACCES;
    return sg_scsi_ioctl(sdev->request_queue, NULL,
    ↪  0, arg);
  ...
}
```

```c
int scsi_cmd_ioctl(struct request_queue *q, ...,
↪  void __user *arg)
{
  ...
  case SCSI_IOCTL_SEND_COMMAND:
    ...
    if (!arg)
      break;
    err = sg_scsi_ioctl(q, bd_disk, mode, arg);
    break;
  ...
  return err;
}
```

```c
int sg_scsi_ioctl(struct request_queue *q,
↪  gendisk *disk, fmode_t mode, struct
↪  scsi_ioctl_command __user *sic)
{
  ...
  err = blk_verify_command(req->cmd, mode);
  ...
  return err;
}

int blk_verify_command(unsigned char *cmd,
↪  mode)
{
  ...
  if (capable(CAP_SYS_RAWIO))
    return 0;
  ...
  return -EPERM;
}
```

Which one is better

- ACLs:
  - Can have large numbers of objects
  - Easy to grant access to many objects at once
  - Require expensive operation on every access
- Capabilities
  - Hard to manage huge number of capabilities
  - They have to come from somewhere
  - They are fast to use (just pointer dereferences)
- Most systems use both
  - ACLs for opening an object (e.g. fopen())
  - Capabilities for performing operations (e.g. read())

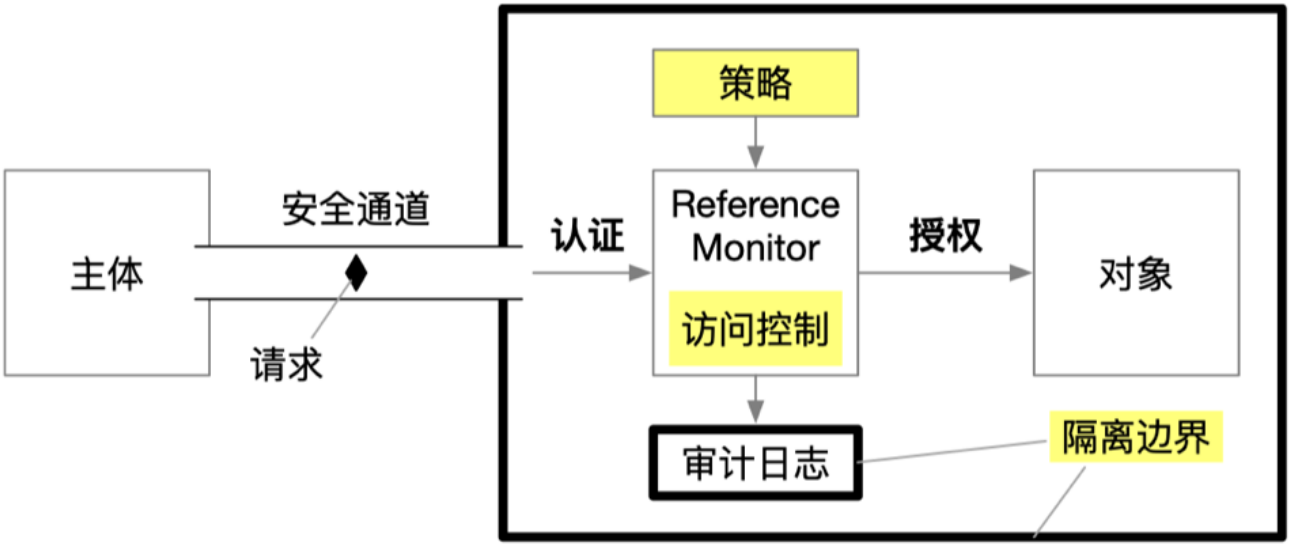
Auditing

- Record what users and programs are doing for later analysis.
- Providing an audit framework is an important **requirement** for a CC-CAPP/EAL (Common Criteria-Controlled Access Protection Profiles/Evaluation Assurance Level) certification.
- The Linux audit framework provides a CAPP-compliant (Controlled Access Protection Profile) auditing system that reliably collects information about any security-relevant (or non-security-relevant) event on a system.

```
wenbo@Mac linux % vim kernel/audit.c
audit.c            audit_fsnotify.c  audit_watch.c     auditsc.c
audit.h            audit_tree.c      auditfilter.c
```

Reference monitor

- 引用监视器（Reference Monitor）
  - 是实现访问控制的一种方式
  - 主体必须通过引用（reference）的方式间接访问对象
  - Reference monitor 位于主体和对象之间，进行检查



Reference Monitor 负责两件事：

1. Authentication：确定发起请求实体的身份，**认证**
2. Authorization：确定实体确实拥有访问资源的权限，**授权**

Reference monitor

- 引用监视器（Reference Monitor）
  - 引用监视器机制必须保证其不可被绕过(Non-bypassable)。
  - 设计者必须 充分考虑应用访问对象的所有可能路径，并保证所有路径都必须通过引用才能进行。
  - 例如，应用必须通过文件描述符来访问文件，而无法直接访问磁盘上的数据或通过 inode 号来访问文件数据。文件系统此时就是引用监视器，文件描述符就是引用。

DAC与MAC

- 自主访问控制（DAC: Discretionary Access Control）
  - 指一个对象的拥有者有权限决定该对象是否可以被其他人访问
    - 例如，文件系统就是一类典型的 DAC，因为文件的拥有者可以设置文件如何被其他用户访问
    - 换句话说， DAC 允许一个普通用户配置其所拥有的对象的访问权限。
  - 但是对部分场景（如军队）来说，DAC过于灵活
    - 例如，文件拥有者是否真的有权可随意设置文件权限?
- 强制访问控制（MAC: Mandatory Access Control）
  - 由"系统"增加一些强制的、不可改变的规则
    - 例如，在军队中，如果某个文件设置为机密，那么就算是指挥官也不能把这个文件给没有权限的人看——这个规则是由军法（系统）规定的
  - MAC与DAC可以结合，此时MAC的优先级更高

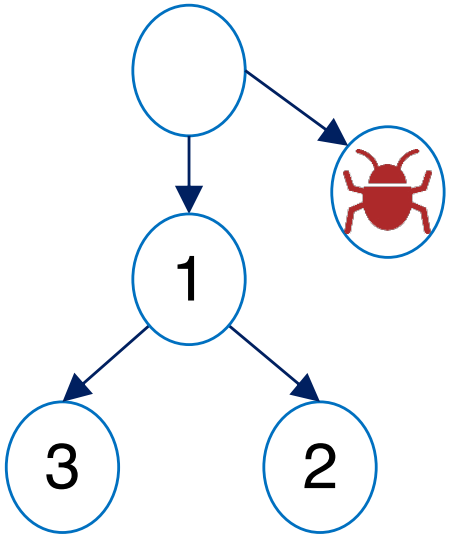
Outline

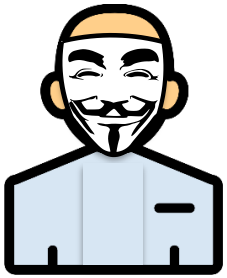
- Security evaluation criteria
  - TCSEC, ITSEC, CC, GB17859
- Common concepts
  - Trusted computing base
  - Attack surface
  - Defense in-depth
- Protection - Access Control
  - Authentication
  - Authorization
  - Auditing
  - Reference monitor
- Security
  - Attacks and defenses

Evolution of Attacks and Defenses

1. Code injection attack
2. Code reuse attack
3. Non-control-data attack

```
bool admin;
……
① if (admin)
     root_access();
  else
     normal_user_access();
……

② root_access(){……}
③ normal_user_access(){……}
```



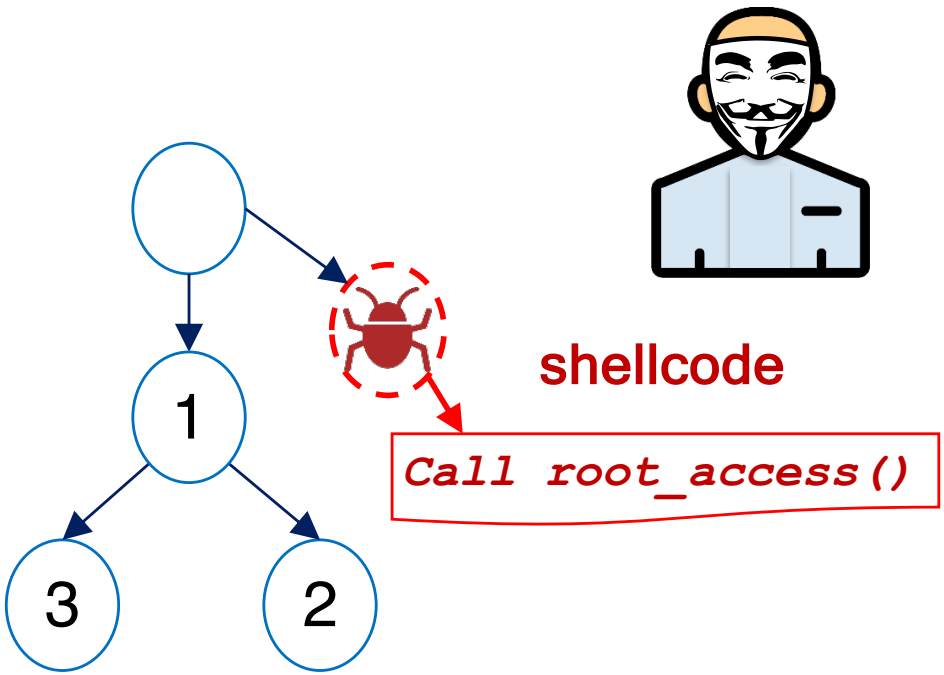


https://asokan.org/asokan/research/HARP-Current.pdf

Evolution of Attacks and Defenses

1. Code injection attack
2. Code reuse attack
3. Non-control-data attack

```
bool admin;
……
① if (admin)
     root_access();
  else
     normal_user_access();
……

② root_access() {……}
③ normal_user_access() {……}
```

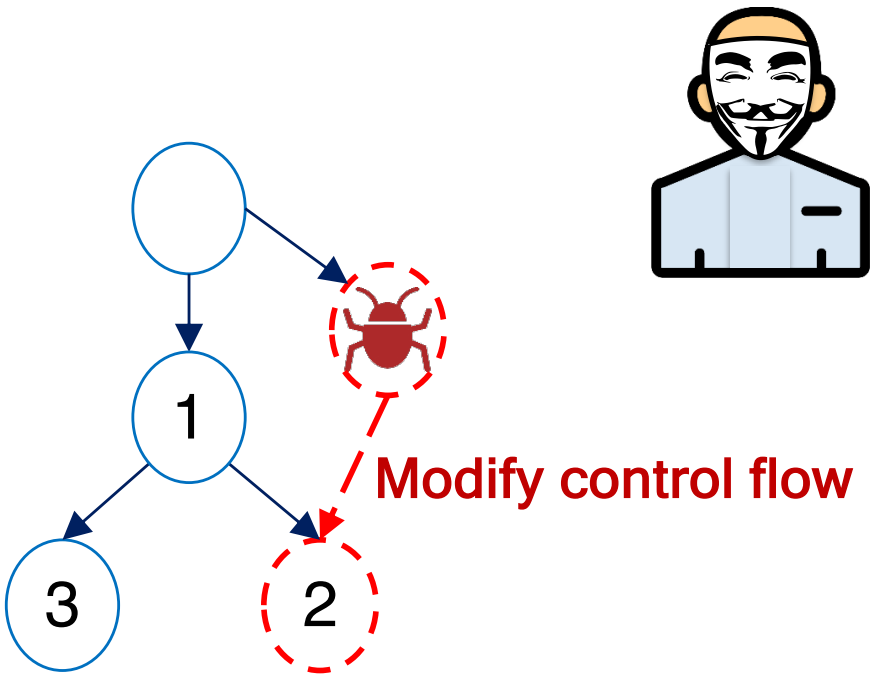


https://asokan.org/asokan/research/HARP-Current.pdf

Evolution of Attacks and Defenses

1. Code injection attack
2. Code reuse attack
3. Non-control-data attack

```
bool admin;
……
① if (admin)
     root_access();
  else
     normal_user_access();
……

② root_access() {……}
③ normal_user_access() {……}
```

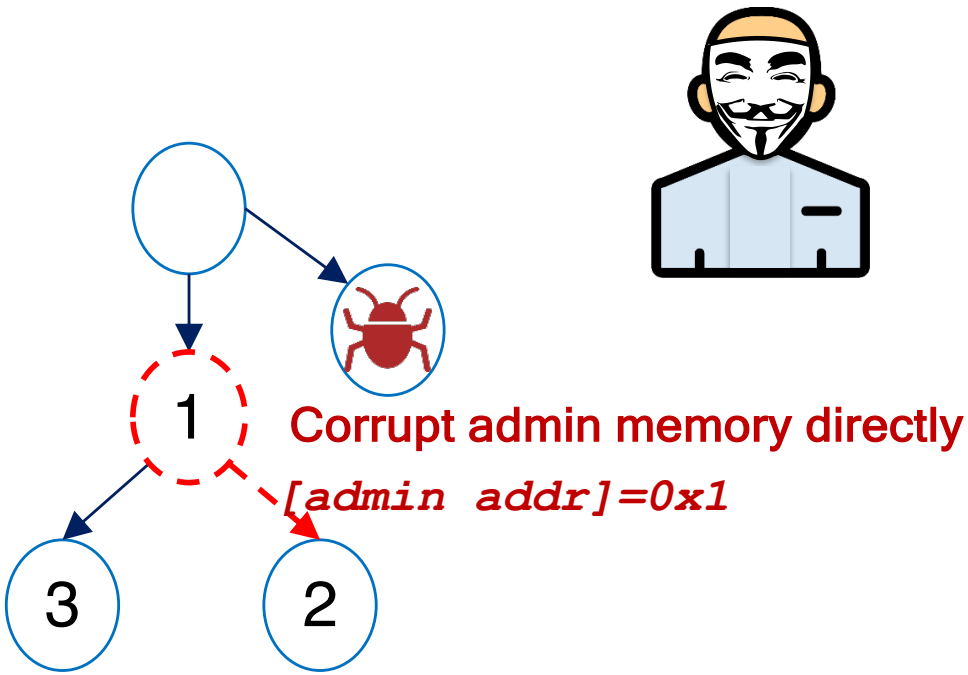


https://asokan.org/asokan/research/HARP-Current.pdf

Evolution of Attacks and Defenses

1. Code injection attack
2. Code reuse attack
3. Non-control-data attack

```
bool admin;
……
① if (admin)
     root_access();
  else
     normal_user_access();
……

② root_access(){……}
③ normal_user_access(){……}
```



https://asokan.org/asokan/research/HARP-Current.pdf

Evolution of Attacks and Defenses

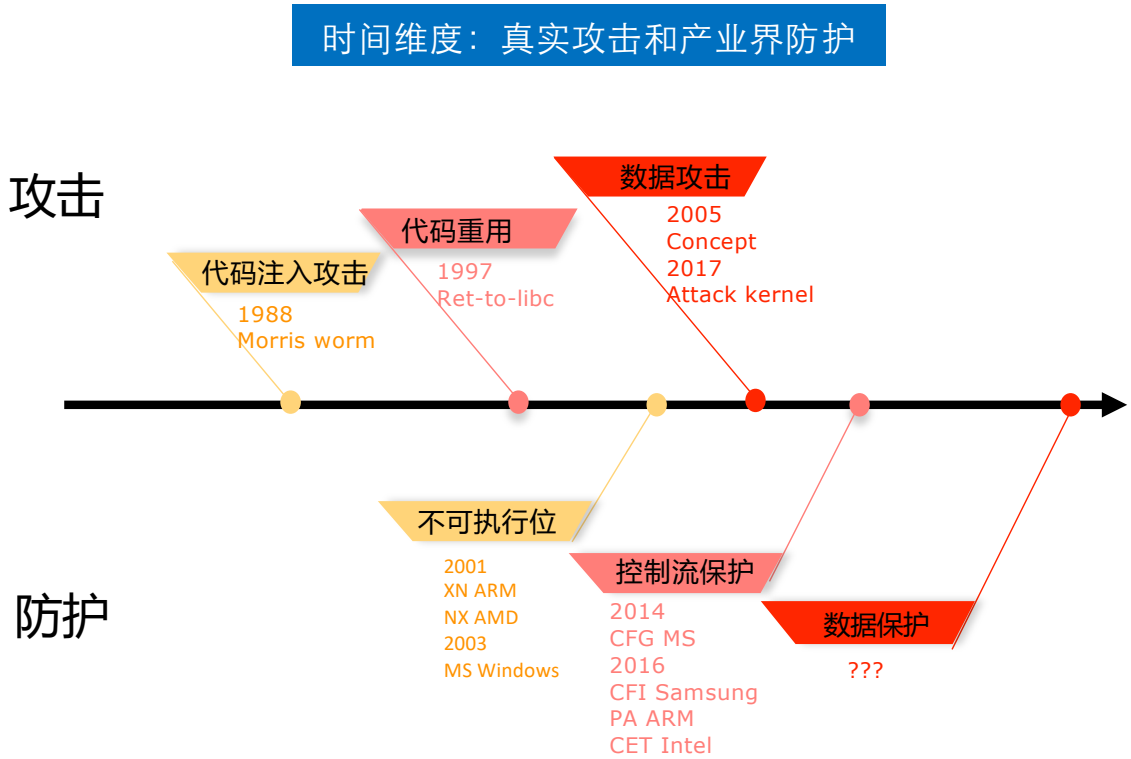
- Summary from grsecurity

Attacks	Defenses
Execute new (injected) code (shellcode)	Non-executable pages, runtime code generation control, ASLR
Execute existing code out-of-(intended)-order (return-to-libc, ROP/JOP)	Control flow integrity, RAP, ASLR
Execute existing code in-(intended)-order (data-only attacks)	Open question (RANDSTRUCT, KERNSEAL, etc)

https://pax.grsecurity.net/docs/PaXTeam-H2HC15-RAP-RIP-ROP.pdf

Evolution of Attacks and Defenses

- Attack evolution
  - 复杂性指数级增加
    - 攻击难度指数级增加
    - 隐蔽性在增加
  - 控制能力缩小
    - 数据攻击依然能root内核
- Defense evolution
  - 软件到硬件
  - 学术界原型到产业界实用方案
  - 有滞后性



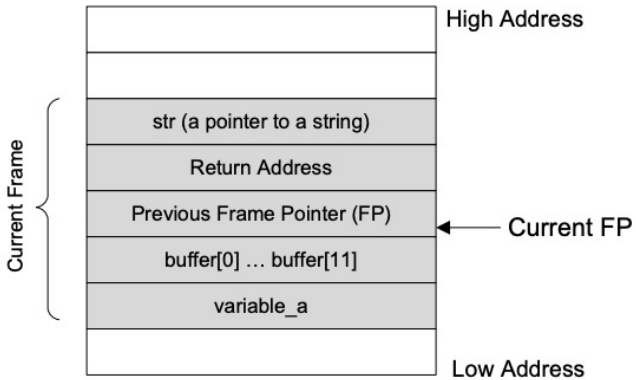
内核代码注入攻防

内核代码注入 | 内核代码重用 | 内核数据攻击

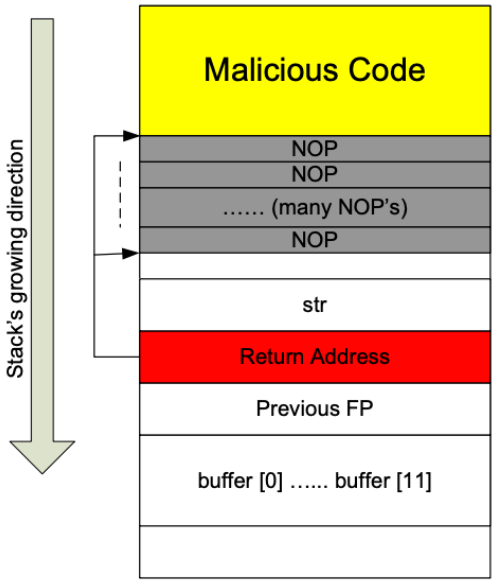
■ 代码注入攻击 Code injection attack

```
void func (char *str) {
   char buffer[12];
   int  variable_a;
   strcpy (buffer, str);
}

Int main() {
   char *str = "I am greater than 12 bytes";
   func (str);
}
```

(a) A code example



(b) Active Stack Frame in func()



Source: http://www.cis.syr.edu/~wedu/seed/Labs_16.04/Software/Buffer_Overflow/

内核代码注入攻防

内核代码注入 → 内核代码重用 → 内核数据攻击

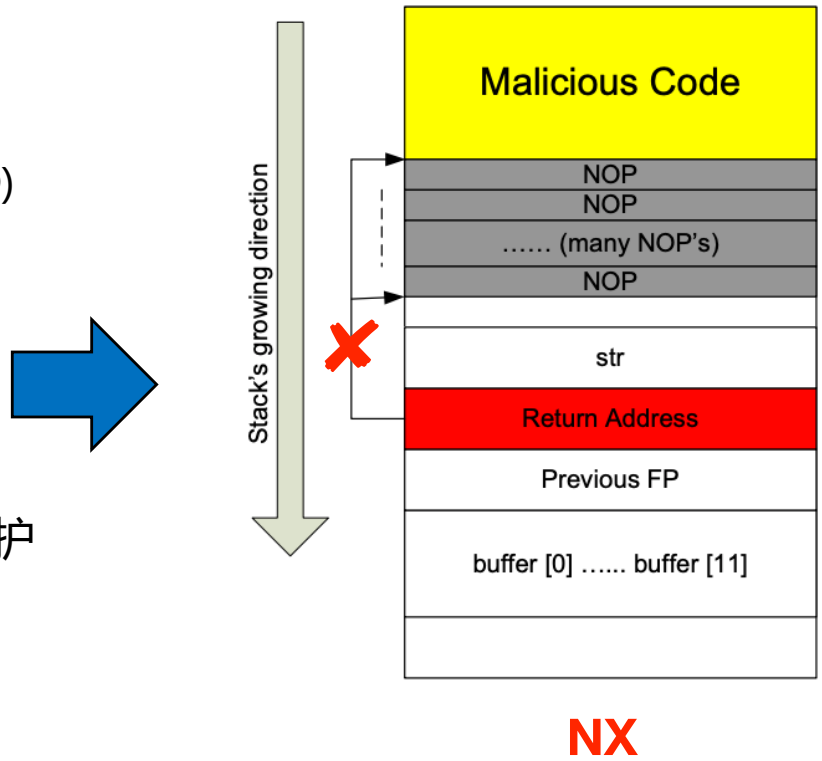
- 通过内核漏洞
  - 篡改已有代码
  - 注入新的代码
  - 或者跳到用户代码，比如jump-to-user
- 系统控制能力大
  - 可执行新代码
  - 危害大
- 多见于早期Linux
  - Kernel Text RWX (Android 2013)

- 内核代码注入防护
  - 保护已有代码W^X
  - 硬件支持，杜绝注入
    - 数据不可执行(2001 XN ARM; NX AMD)
    - 特权不可执行(2011 SMEP Intel; PXN ARM)
- 通过内核页表来实现
  - 在内核页表设置相应的保护位，实现保护
  - 多数Android设备, 包含Google Pixel

内核代码注入攻防

内核代码注入 | 内核代码重用 | 内核数据攻击

- 内核代码注入防护
  - 保护已有代码W^X
  - 硬件支持，杜绝注入
    - 数据不可执行(2001 XN ARM; NX AMD)
    - 特权不可执行(2011 SMEP Intel; PXN ARM)
- 通过内核页表来实现
  - 在内核页表设置相应的保护位，实现保护
  - 多数Android设备, 包含Google Pixel

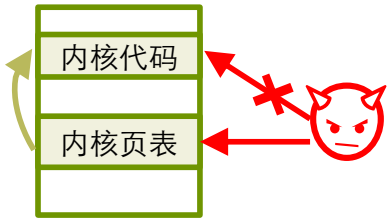


Source: http://www.cis.syr.edu/~wedu/seed/Labs_16.04/Software/Buffer_Overflow/

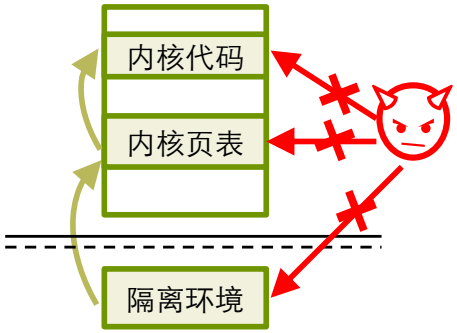
内核代码注入攻防

内核代码注入 → 内核代码重用 → 内核数据攻击

- 通过内核页表来实现
  - 对内核页表没有保护
  - 攻击者可篡改页表，去掉保护
  - 进而篡改代码



- 通过隔离环境保护内核页表
  - 通过隔离环境，避免内核漏洞影响
  - 硬件支持可信执行环境
    - Intel VT, SGX; ARM TrustZone, Hypervisor, SKEE; RISCV PMP
  - 实现了纵深防御defense-in-depth



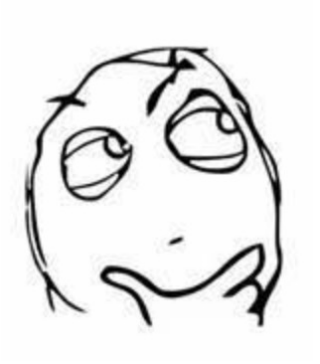
内核代码重用攻防

内核代码注入 | **内核代码重用** | 内核数据攻击

- Cannot inject new code or change existing code
- How about reuse code?

Program

I'm the most lovely and benign program you can imagine. I embody functionality like document analysis of different types. I'm not harmful in any way.



https://www.usenix.org/sites/default/files/conference/protected-files/sec14_slides_backes.pdf

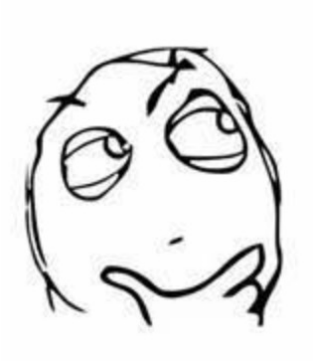
内核代码重用攻防

内核代码注入 | **内核代码重用** | 内核数据攻击

- Cannot inject new code or change exist code
- How about reuse code?

Program

I'm the most lovely and benign program you can imagine. I embody functionality like document analysis of different types. I'm not harmful in any way.



https://www.usenix.org/sites/default/files/conference/protected-files/sec14_slides_backes.pdf

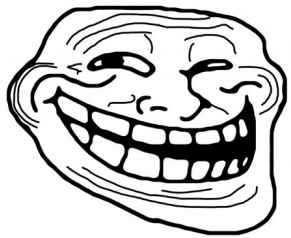
内核代码重用攻防

内核代码注入 | **内核代码重用** | 内核数据攻击

- Cannot inject new code or change exist code
- How about reuse code?

Program

I'm the most lovely and benign program you can imagine. I embody functionality like document analysis of different types. I'm not harmful in any way.



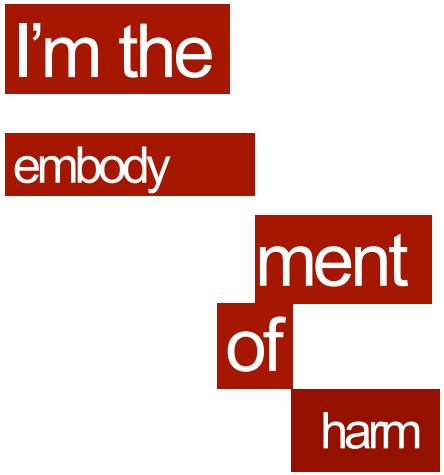
Malware

I'm the embodyment of harm

https://www.usenix.org/sites/default/files/conference/protected-files/sec14_slides_backes.pdf

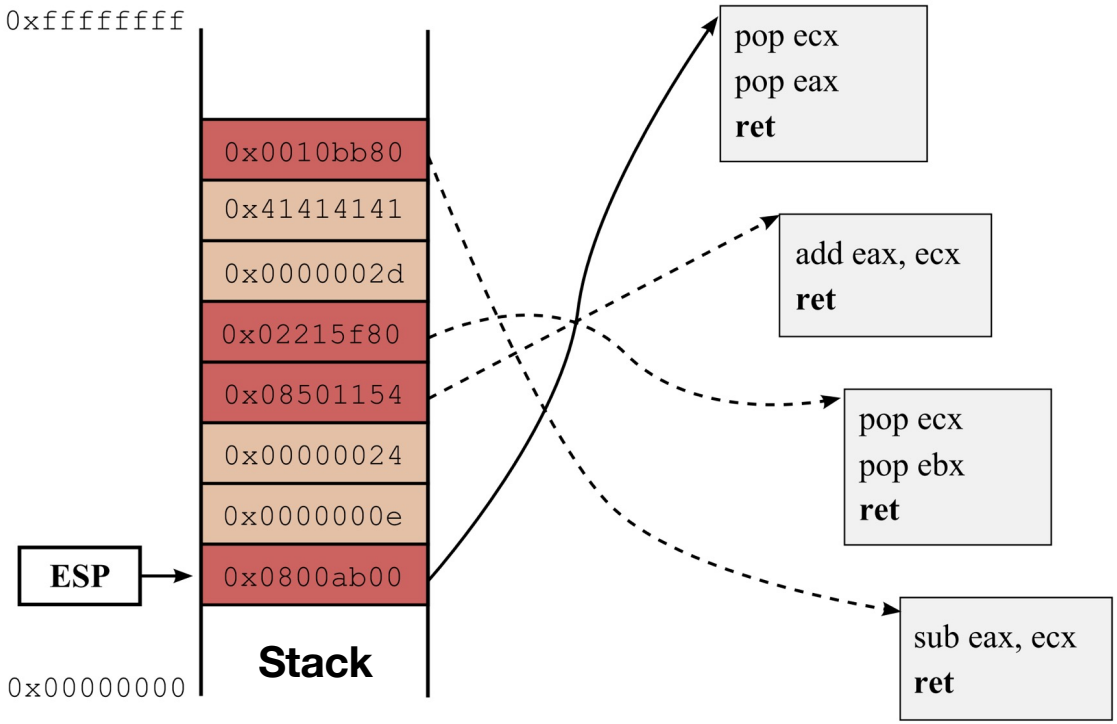
Code reuse attack

- Existing code snippet, called *gadget*
- By changing control data to chain them together
  - Changing return address, termed ROP attack
  - Changing function pointer, termed as JOP attack



Code reuse - ROP attack

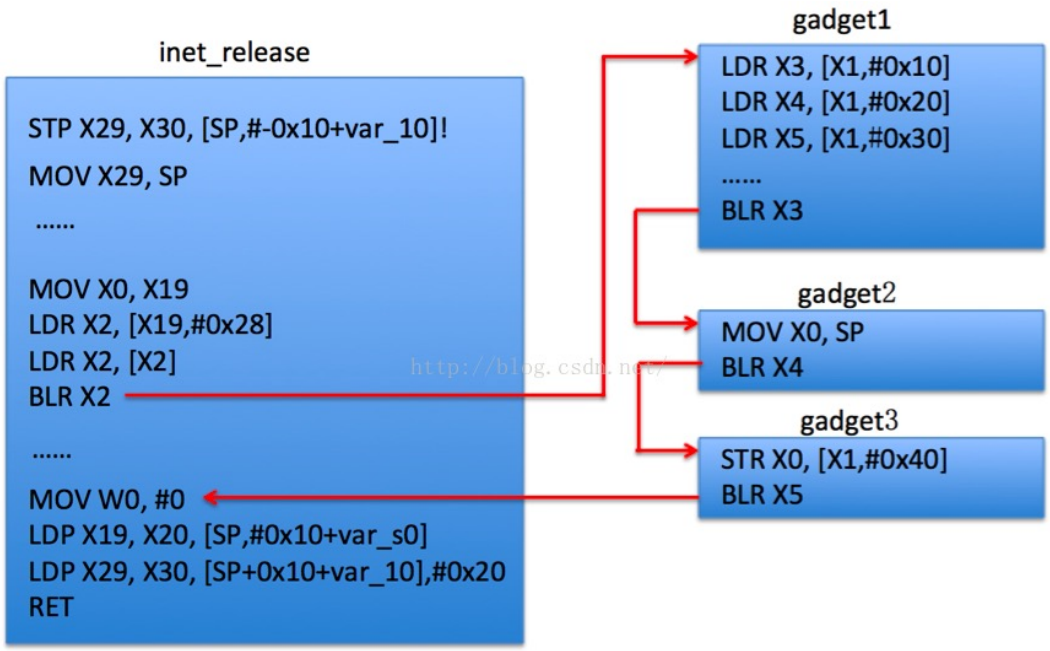
- Return-oriented programming
  - Attacker has full control of return address on stack
- A very simple ROP chain that calculates 0xe+ 0x24−0x2d.
  - Result is in the eax register



Source: Size Does Matter Why Using Gadget-Chain Length to Prevent Code-Reuse Attacks is Hard

Code reuse - JOP attack

- Jump-oriented programming
  - Corrupt function pointer
  - Following example will leak stack pointer



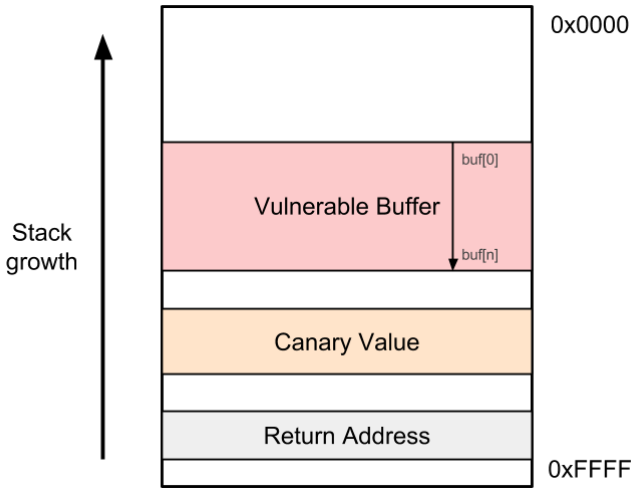
ROP泄露SP的大致思路

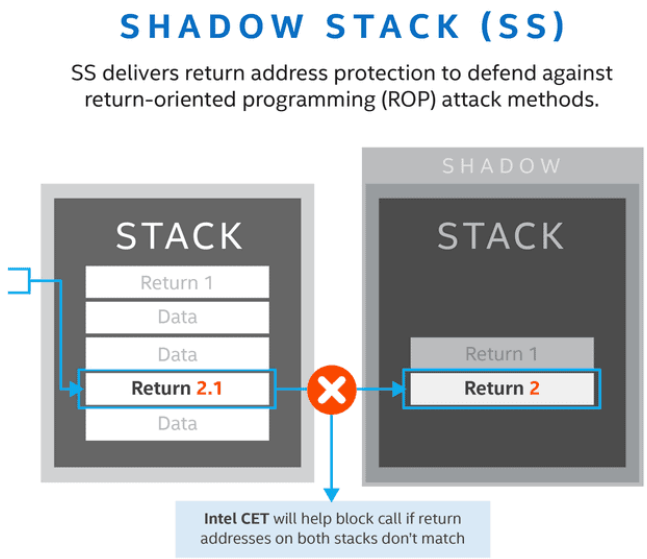
Source: Size Does Matter Why Using Gadget-Chain Length to Prevent Code-Reuse Attacks is Hard

内核代码重用攻防

内核代码注入 | **内核代码重用** | 内核数据攻击

- 保护返回地址
  - Stack canary
  - Randomization
  - Shadow stack
- 硬件支持
  - Intel CET 2016

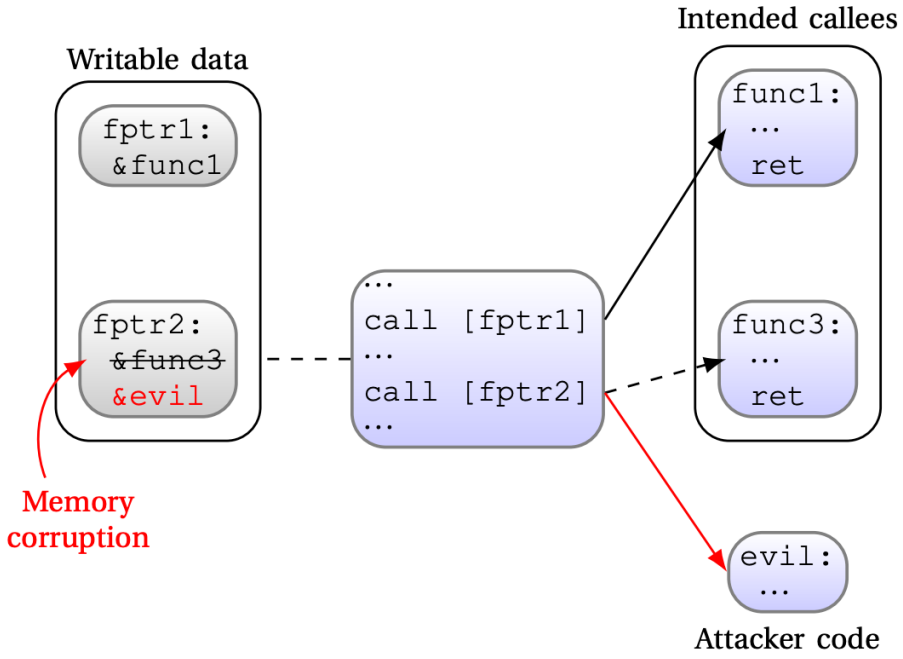




内核代码重用攻防

内核代码注入 | 内核代码重用 | 内核数据攻击

- 保护函数指针
  - 跳转检查
    - MS CFG, LLVM CFI
  - 指针保护
    - CPI, PA based
- 硬件支持
  - Intel CET 2016
  - ARM PA 2016



(a) Without CFI.

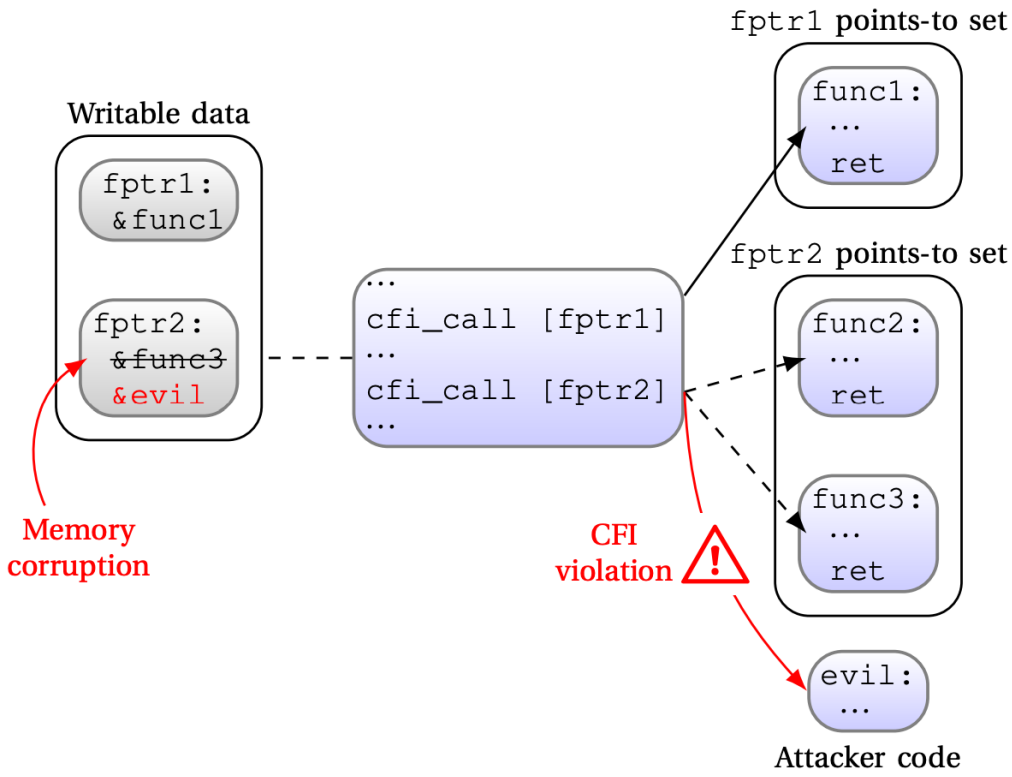
Source: Back To The Epilogue: Evading Control Flow Guard via Unaligned Targets

Source: https://eyalitkin.wordpress.com/2017/08/18/bypassing-return-flow-guard-rfg/

内核代码重用攻防

内核代码注入 | 内核代码重用 | 内核数据攻击

- 保护函数指针
  - 跳转检查
    - MS CFG, LLVM CFI
  - 指针保护
    - CPI, PA based
- 硬件支持
  - Intel CET 2016
  - ARM PA 2016



(b) With CFI.

Source: Back To The Epilogue: Evading Control Flow Guard via Unaligned Targets

Source: https://eyalitkin.wordpress.com/2017/08/18/bypassing-return-flow-guard-rfg/

内核代码重用攻防

内核代码注入 | 内核代码重用 | 内核数据攻击

- ASLR 与 KASLR （内核地址布局随机化）
  - ASLR 通过随机化地址空间布局来提高系统攻击难度
  - KASLR是对内核启用地址随机化
- KASLR 可缓解 代码重用 攻击
  - 攻击者需要知道用户态恶意代码在内核中直接映射区域的地址
  - KASLR 通过将内核的虚拟地址布局进行随机化，使攻击者准确定位内核地址的难度大大提升

内核数据攻击

内核代码注入 → 内核代码重用 → 内核数据攻击

- 控制数据被保护后，攻击者提出非控制数据攻击
  - 返回地址和函数指针以外的数据
  - 影响关键的安全特性
  - 仅利用非控制数据攻击做到内核提权-2017
- 非控制数据防护
  - 种类繁杂，难以实行统一有效保护
  - 主流操作系统均缺乏对数据攻击的有效防护

SELinux绕过

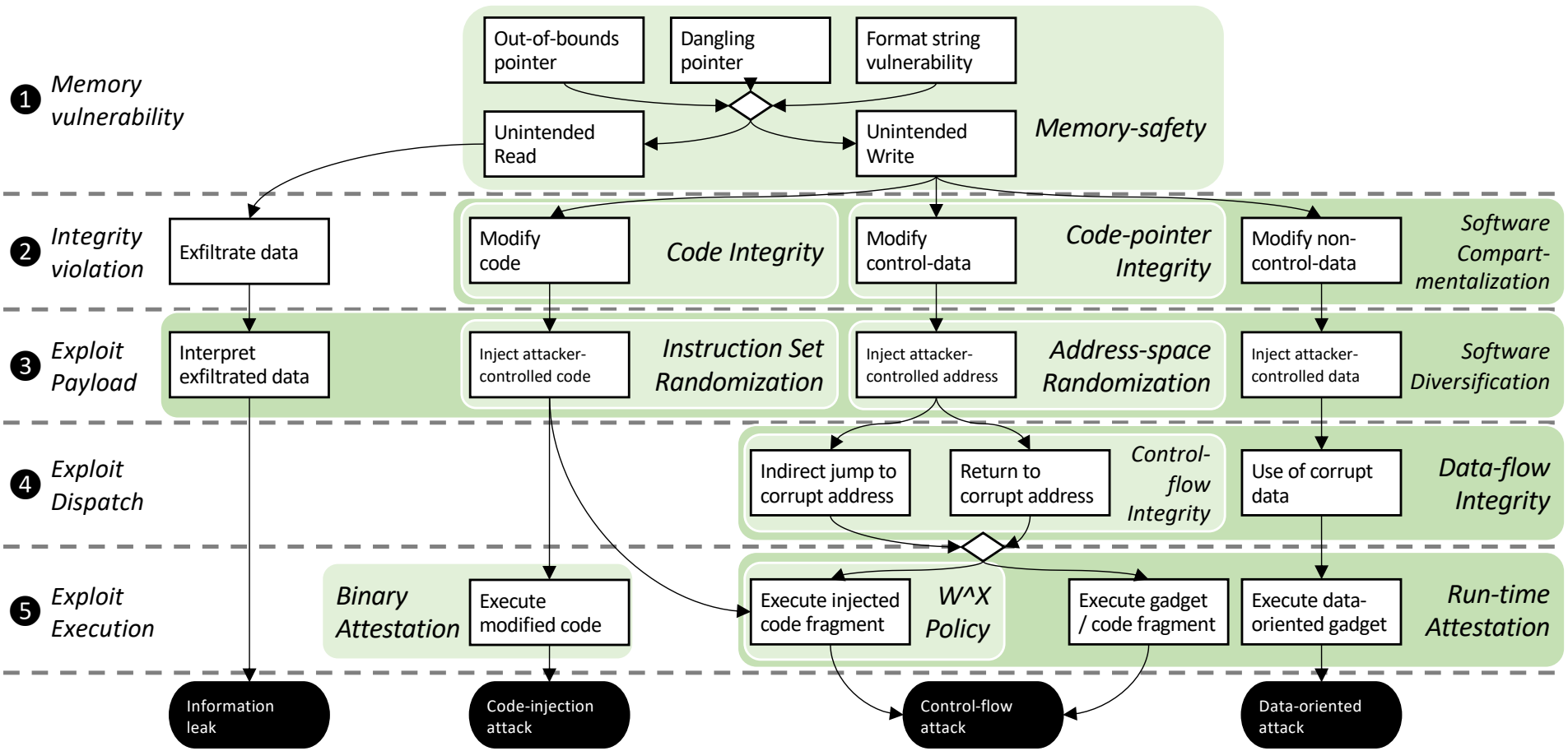
```
int ss_initialized;

void security_compute_av(u32 ssid,
                         u32 tsid,
                         u16 orig_tclass,
                         struct av_decision *avd,
                         struct extended_perms *xperms)
{
        ......
        if (!ss_initialized)
                goto allow;
        .....
}
```

Selected Research & Vulnerabilities

1988-99	**ret2libc** *Solar Designer (Phrack)*	**Morris Worm:** RCE in `fingerd`	**CVE-1999-1416:** DoS & RCE in Solaris Answerbook2		**Format string vulnerabilities** *Anders (Bugtraq. 1999)*
2001	**Advanced ret2libc** *Nergal (Phrack)*				
2005	**x86-64 borrowed code chunks exploitation** *Krahmer*				**Non-control-data attacks** *Chen et al (SSYM. '05)*
2007	**ROP on x86** *Shacham (CCS'07)*				
2008	**ROP on ATMEL AVR** *Francillon et al (CCS'08)*	**ROP on SPARC** *Buchanan et al (CCS'08)*			
2009	**ROP Rootkits** *Hund et al (USENIX Sec. '09)*	**ROP on PowerPC** *FX Lindner (BlackHat USA)*	**ROP on ARM / iOS** *Miller et al (BlackHat Europe)*		
2010	**ROP w/o Returns** *Checkoway et al (CCS'10)*	**CVE-2010-3765**: *Nobel Peace Price website 0day*	**CVE-2010-2883:** *RCE in Adobe Reader and Acrobat*		
2011-12		**CVE-2011-1938:** *RCE in PHP*	**CVE-2012-0003:** *RCE in WMP MIDI library*		**String-Oriented Programming** *Payer (28C3. '11)*
2013	**JIT-ROP** *Snow et al (IEEE S&P'13)*	**CVE-2013-3893:** *RCE in Internet Explorer*	**CVE-2014-9222:** *Misfortune cookie in RomPager*		
2014	**Blind ROP** *Bittau et al (IEEE S&P'14)*	**Stitching Gadgets** *Davi et al (USENIX'14)*	**CVE-2014-0160:** *Heartbleed vuln. in OpenSSL*		**Write Once, Pwn Anywhere** *Yu (BlackHat USA'14)*
2015	**Out-of-Control** *Göktas et al (IEEE S&P'14)*	***Gadget size Matters*** *Göktas et al (USENIX'14)*	**ROP is Still Dangerous** *Carlini et al (USENIX'14)*		**Data-Oriented Exploits** *Hu et al (USENIX Sec.'15)*
2016	**SROP** *Bosman et al (IEEE S&P'14)*	**Control-flow Bending** **Carlini et al (USENIX Sec.'16)**	**CVE-2016-0034:** *Angler RCE in Silverlight*		**DOP** *Hu et al (IEEE S&P '16)*

https://asokan.org/asokan/research/HARP-Current.pdf

Taxonomy of Run time Attacks and Defenses



Adapted from Szekeres et al., SoK: Eternal War in Memory, IEEE SP (2013)

https://asokan.org/asokan/research/HARP-Current.pdf

Takeaway

- Security evaluation criteria
  - TCSEC, ITSEC, CC, GB17859
- Common concepts
  - Trusted computing base
  - Attack surface
  - Defense in-depth
- Access Control
- Attacks and defenses