

2019-2020学年春季学期

计算机体系结构安全

Computer Architecture Security

授课团队：侯锐、朱子元、宋威
助 教：李沛南

计算机体系结构安全

Computer Architecture Security

[第3/4次课] 体系结构安全概述

授课教师：宋威

授课时间：3月17/24

○宋威

○中国科学院信息工程研究所，信息安全国家重点实验室

○副研究员，博士生导师

○计算机体系结构安全

○缓存侧信道防护

○控制流完整性防护

○课程：

○体系结构安全概述

○代码复用攻击和防御

○缓存侧信道攻击和防御

邮箱：songwei@iie.ac.cn

内容概要

- 系统安全基础知识
- 安全漏洞
- 安全攻击
- 安全防御
- 体系结构的安全问题
- 体系结构的安全防御
- 总结与讨论

内容概要

- **系统安全基础知识**
- 安全漏洞
- 安全攻击
- 安全防御
- 体系结构的安全问题
- 体系结构的安全防御
- 总结与讨论

什么是安全

○ Safety

- 自然属性的安全
- 抵御自然灾害
- 非人为攻击，不确定性

737-Max

防火、防盗

○ Security

- 人为属性的安全
- 抵御故意攻击
- 人为攻击，强确定性

金融：系统性风险、人为操纵

汽车：ABS、雷达、电子锁

○ 体系结构安全是指 Security

什么是计算机 systems 安全

○ 物理安全

- 保护计算机设备、设施(含网络) 免遭破坏。

○ 运行安全

- 保障系统功能的正常运行。

○ 信息安全

- 防止信息被故意的或偶然的泄露、更改、破坏

服务器机房：

过载保护、水冷、负载均衡、冗余服务器、磁盘阵列、防火墙、杀毒软件。

信息安全 (CIA)

- 概要机密性(Confidentiality)

- 信息仅被合法的实体访问，不泄漏给未授权的实体。

- 完整性(Integrity)

- 信息只能由授权实体修改，不被偶然或蓄意地篡改、伪造、丢失等。

- 可用性(Availability)

- 信息能够随时被授权实体访问并使用。

信用卡交易

- 通过对计算机体系结构做调整、改变、加固来维护信息安全各个特性。
 - 机密性、完整性、可用性
- 其他安全特性
 - 不可否认性
 - 可存活性
 - 可控性
 - 可认证性
 - 可审查性

计算机体系结构的范畴

○ 狭义定义

- 专指指令集(ISA: **Instruction Set Architecture**)
- 硬件和软件的接口

○ 广义定义

- **ISA: 指令集**
- **微结构(Microarchitecture): 处理器实现**
- **系统**
 - 硬件系统: 缓存、内存、外设、片上总线/网络
 - 软件系统: 操作系统、虚拟化、固件、系统加载

从安全角度思考体系结构安全

- **机密性：看不见不该看见的**
 - 优先级、用户
 - 安全/非安全
 - 硬件/软件

- **完整性：改不了不该改的**
 - 控制流完整性
 - 数据完整性

- **可用性：停不了不该停的**
 - 资源竞争：CPU、内存/网络带宽、I/O外设

体系结构安全的基本思路

- 限制 (Compartment)
 - 隔离 (Isolation)
 - 最小权限法则 (Least privilege)
- 纵深防御
 - 多种防御措施同时工作
 - 保护最薄弱环节
 - 冗余
 - 变化
- 简单化 (KISS: keep it simple, stupid.)

- 安全的定义
- 计算机体系结构安全

内容概要

- 系统安全基础知识
- **安全漏洞**
- 安全攻击
- 安全防御
- 体系结构的安全问题
- 体系结构的安全防御
- 总结与讨论

计算机系统的安全问题

- 计算机系统的安全问题源于系统中广泛存在的**安全漏洞**。
 - 既有软件的也有硬件的
- 安全漏洞是指计算机系统**中的缺陷**，实际上就是系统安全问题产生的根源所在。
- 安全**攻击**利用目标系统中存在的安全漏洞来破坏目标系统的安全特性。

例子

- 机密性

- 破解软件的算号器

- 完整性

- 玩游戏用作弊器获得无限生命无限金钱

- 可用性

- 木马软件不停地在浏览器里面插广告

勒索软件加密磁盘、远程操控摄像头、网页挖矿

漏洞产生的原因

- 安全漏洞是计算机系统中存在的**缺陷/错误/后门**。
 - 缺陷：设计失误
 - 错误：实现错误
 - 后门：（恶意）功能

- 计算机系统的安全漏洞是**不可避免的**。
 - 系统复杂性
 - 安全和效率间的矛盾
 - 安全和功能间的矛盾

系统复杂性

○ 系统过于复杂
导致错误不可避免。

○ 软件设计不当。

○ 软件复杂性

○ 软件实现错误

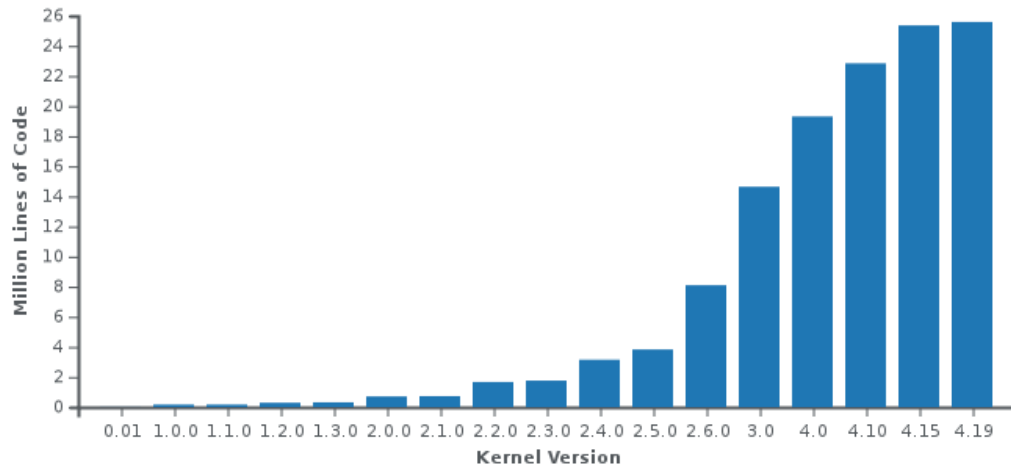
○ Linux kernel 2018
3500行/日

○ 检查、审计基本靠人

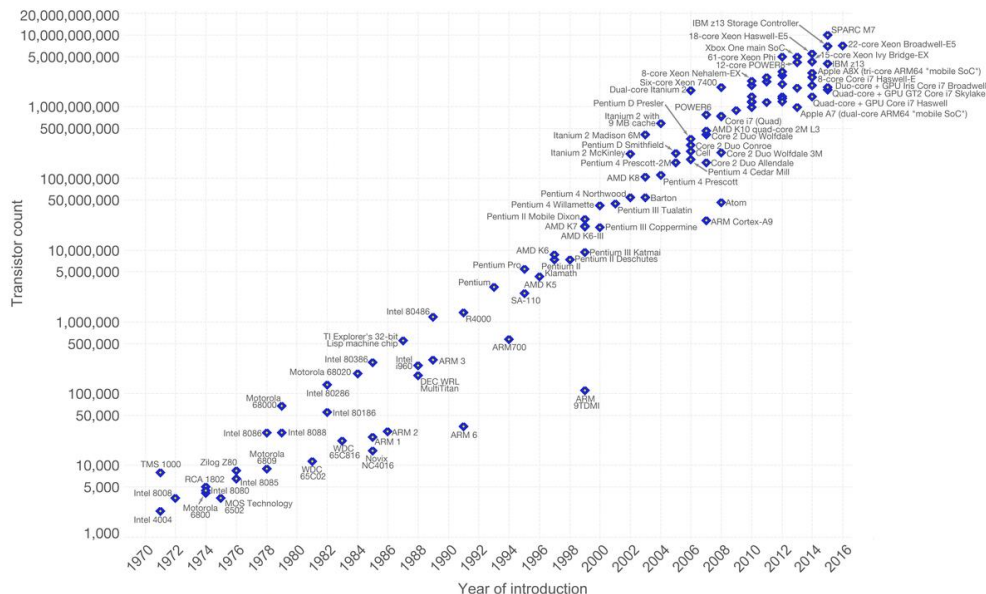
○ 硬件复杂性

○ 硬件设计、实现错误

○ 摩尔定律：
每两年晶体管数量增倍



Moore's Law – The number of transistors on integrated circuit chips (1971-2016) OurWorld in Data
 Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are strongly linked to Moore's law.



Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)
 The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic. Licensed under CC-BY-SA by the author Max Roser.

安全和效率间的矛盾

○缓冲区溢出

- 边界检查 (执行)
- 边界信息 (存储)

○控制流完整性

- 跳转检查 (执行)
- 合法跳转目标 (存储)
- 指针类型 (存储)

○幽灵 (Spectre)

- 推测执行时的安全检查 (执行)

在高级语言到机器码的编译过程中，**语义的不断丢失**是（软件）安全漏洞的主要来源之一。



安全和功能间的矛盾

○检测/监测后门

- 远程调试、返厂错误检测
- 政府监管、监视、网络战争

○安全/隐私与易用性

- 复杂安全配置与傻瓜使用：Redhat SELinux
- 隐私数据搜集：手机App、AI音响

○安全与可获得性

- 互联网，连还是不连？
- 优盘，用还是不用？
- 文件，加密还是不加密？

安全漏洞的分类

○ 纯软件漏洞

- 由于软件设计失误或实现错误造成的漏洞。
- 一般直接在软件修复 (bugfix) 。

○ 硬件相关的软件漏洞

- 硬件 (系统) 为了效率而忽视安全, 导致软件可能的异常行为。
- 传统上使用软件防御, 现在考虑在体系结构层面加固。
 - 补全丢失的安全特征交由硬件检查。

○ 纯硬件漏洞

- 硬件设计不当导致的缺陷或错误、甚至是故意留下的后门。
- 应该由硬件修复, 软件配合。

纯软件漏洞

○ 网络协议漏洞

- 中间人攻击
- DNS劫持
- http劫持



○ 加密解密协议

- 无加密明文传输
- 弱密钥
- 弱验证



硬件相关的软件漏洞

任意地址读写、信息泄露

- 越界访问
- 释放后使用UAF
- 缓存侧信道
- Meltdown/Spectre (推测执行漏洞)



任意代码执行、提权

- 直接代码注入
- 控制流劫持
- 伪造虚拟函数表
- 不可信代码注入 (信任根问题)

```
007: 20 63 65 69 6c 28 20 28 - 2d 73 20 24 66 69 6c 65 [ cell( (-s $file)
008: 29 20 2f 20 31 36 29 3b - 0a 0a 20 20 20 20 6d 79 ) / 16).. my]
009: 20 24 66 6f 72 6d 20 3d - 20 65 65 6e 67 74 68 20 [ $form = length]
00a: 73 70 72 69 6e 74 66 20 - 27 25 78 27 2c 20 24 73 [ sprintf "%x", $s]
00b: 69 7a 65 2d 31 3b 0a 20 - 29 20 20 24 66 6f 72 6d [ i:=1;.. $form]
00c: 20 7c 7c 3d 20 31 3b 0a - 0a 20 20 20 20 70 72 69 [ ]=1;.. pri]
00d: 6e 74 20 22 46 69 6c 65 - 3a 20 24 66 69 6c 65 5c [ nt "file: $file"]
00e: 6e 22 3b 0a 20 20 20 20 - 6f 70 65 6e 20 6d 79 20 [n];.. open my]
00f: 24 66 68 2c 20 27 3c 27 - 2c 20 24 66 69 6c 65 20 [ $fh, '<', $file]
010: 6f 72 20 64 69 65 20 24 - 21 3b 0a 0a 20 20 20 20 [ or die $!;.. ]
011: 6d 79 20 24 69 20 3d 20 - 30 3b 0a 20 20 20 77 [ my $i = 0;.. w]
012: 68 69 6c 65 20 28 20 6d - 79 20 24 72 62 20 3d 20 [ hile ( my $rb = ]
013: 72 65 61 64 20 24 66 68 - 2c 20 6d 79 20 24 62 75 [ read $fh, my $bu]
014: 66 2c 20 31 36 20 29 20 - 7b 0a 0a 20 20 20 20 [ f, 16 ) {.. ]
015: 20 6d 79 20 40 78 20 3d - 20 75 6e 70 61 63 6b 20 [ my @x = unpack]
016: 27 28 48 32 29 2a 27 2c - 20 24 62 75 66 3b 0a 20 [ "(H2)", $buf;.. ]
017: 20 20 20 20 20 70 75 73 - 68 20 40 78 2c 20 27 20 [ push @x, ]
018: 20 27 20 75 6e 74 69 6c - 20 40 78 20 3d 3d 20 31 [ ' until @x == 1]
019: 36 3b 0a 0a 20 20 20 20 - 20 20 24 62 75 66 20 3d [ $;.. $buf += ]
01a: 7e 20 74 72 2f 5c 78 32 - 30 2d 5c 78 37 45 2f 2e [ w.tr/\x20-\x7E/.]
01b: 2f 63 3b 0a 0a 20 20 20 - 20 20 20 24 62 75 66 20 [ /c;.. $buf]
01c: 2e 3d 20 27 20 27 20 78 - 20 28 31 36 20 2d 20 6c [ .= ' x (16 - 1]
01d: 65 6e 67 74 68 28 24 62 - 75 66 29 29 3b 0a 0a 20 [ length($buf);.. ]
01e: 20 20 20 20 20 70 72 69 - 6e 74 66 20 22 25 30 2a [ printf "%0*]
01f: 78 3a 20 25 73 20 5b 25 - 73 5d 5c 6e 22 2c 0a 20 [ x: %s [%s]\n".. ]
020: 20 20 20 20 20 20 20 20 - 20 24 66 6f 72 6d 2c 0a [ $form,.. ]
021: 20 20 20 20 20 20 20 20 - 20 20 24 69 2b 2b 2c 0a [ $i++;.. ]
022: 20 20 20 20 20 20 20 20 - 20 20 73 70 77 69 6e 74 [ sprintf
```



纯硬件漏洞

○ 硬件后门

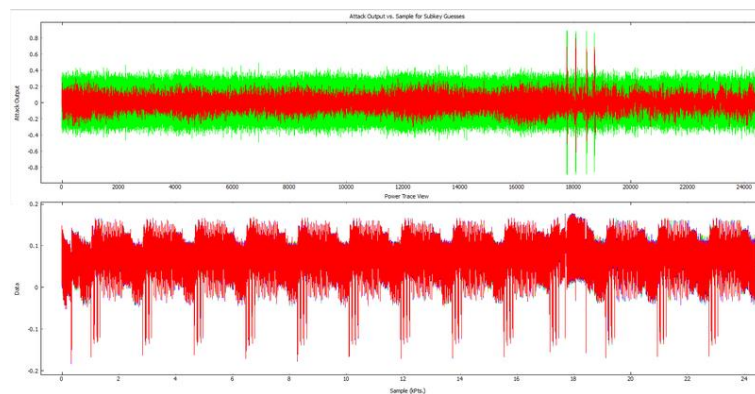
- 测试后门
- 硬件木马

○ 物理侧信道

- 电流
- 功耗
- 电磁辐射

○ 物理攻击

- 电磁干扰
- 功耗过载



安全漏洞分类：按类型

○纯软件漏洞

- DDoS (ARP轰炸、拒绝服务)
- 通讯协议漏洞 (DNS劫持、Http劫持)
- 算法漏洞 (人工智能图像识别错误)
- 加解密漏洞 (弱密钥、中间人)

○硬件相关的软件漏洞

- 缓冲区溢出 (越界访问, 读取密钥/关键数据)
- 控制流劫持 (修改代码指针, 提升权限, 病毒复制)
- 释放后重用UAF (指针失效, 任意内存读写)
- 物理地址暴露 (解析地址随机化, 破坏硬软件隔离)
- 软件侧信道 (微体系结构信息暴露)
- 瞬态执行错误 (熔断和幽灵, 预测执行检查忽略)
- 不可信代码注入 (可信链条错误)

体系结构相关

○纯硬件漏洞

- 硬件后门 (硬件木马、测试后门)
- 物理侧信道 (功耗分析、电磁分析)
- 传感器错误 (传感器失效、误读)
- 物理防护 (散热失效、电磁干扰、电磁辐射)

以上分类并不完全, 不能覆盖所有的安全漏洞, 同时存在交叉。复杂安全漏洞依赖于多种漏洞, 并不一定单独存在。

安全漏洞分类：按时间

○未知漏洞

- 未被发现的漏洞（挖洞）

- 0day漏洞（已被发现，但没有修复/上报）

○已知漏洞

- 1day漏洞（已上报/发现，但未修复/完全部署）

○已修复漏洞

- 旧系统，未更新补丁

○漏洞数据库：

- CVE (1999) : <https://cve.mitre.org/>

- 国家信息安全漏洞库 (2009) : <http://www.cnnvd.org.cn/>

- 漏洞出现的原因
- 计算机体系结构相关的漏洞

内容概要

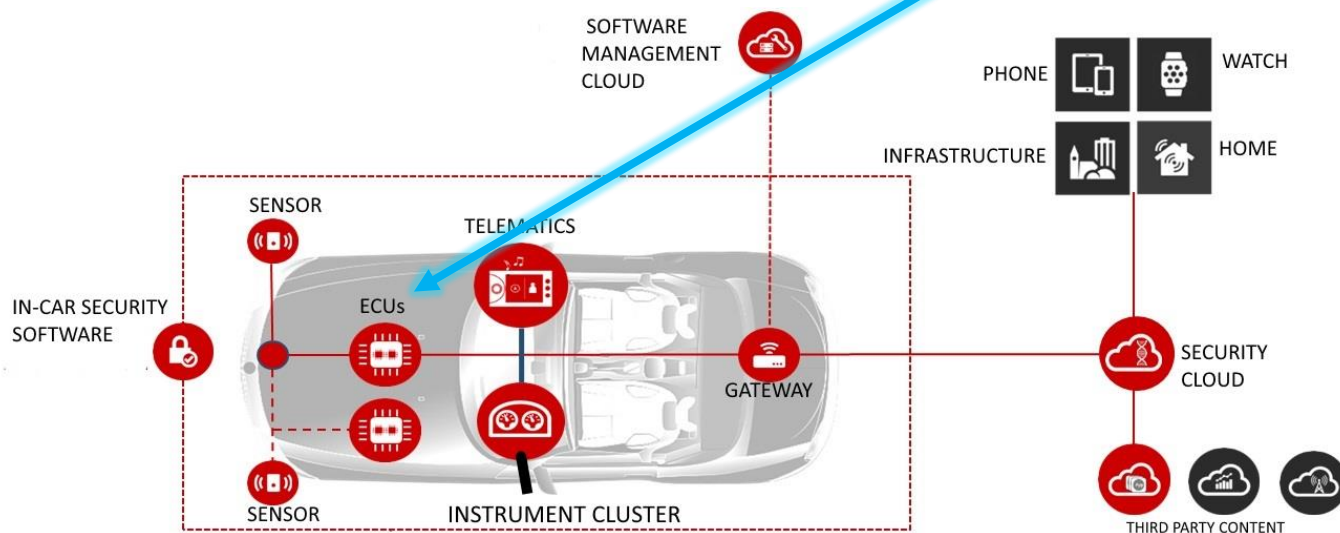
- 系统安全基础知识
- 安全漏洞
- **安全攻击**
- 安全防御
- 体系结构的安全问题
- 体系结构的安全防御
- 总结与讨论

安全攻击和漏洞利用

- “**漏洞利用**”通过使用安全漏洞来造成不期望发生的软硬件行为，经常被用来劫持计算机系统、恶意提权或者造成服务暂停（DDoS）等攻击。
- 安全攻击的范畴更大，包含所有对计算机系统、信息系统、网络系统的恶意操作。包括使用**社会工程学**的攻击。
- 本课程只涉及**计算机体系结构相关**的漏洞利用和防御。

攻击案例：操控特斯拉轿车

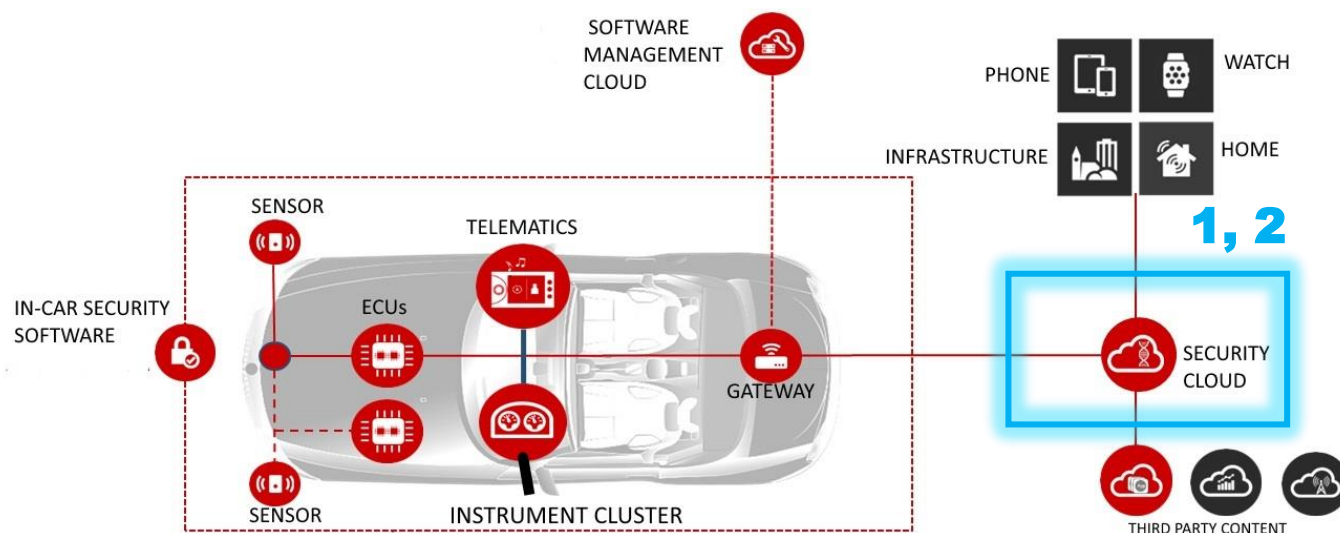
2016年9月，科恩实验室（腾讯），远程控制Tesla model S，车灯变彩灯。



攻击案例：操控特斯拉轿车-1

○ 第一步：远程连接

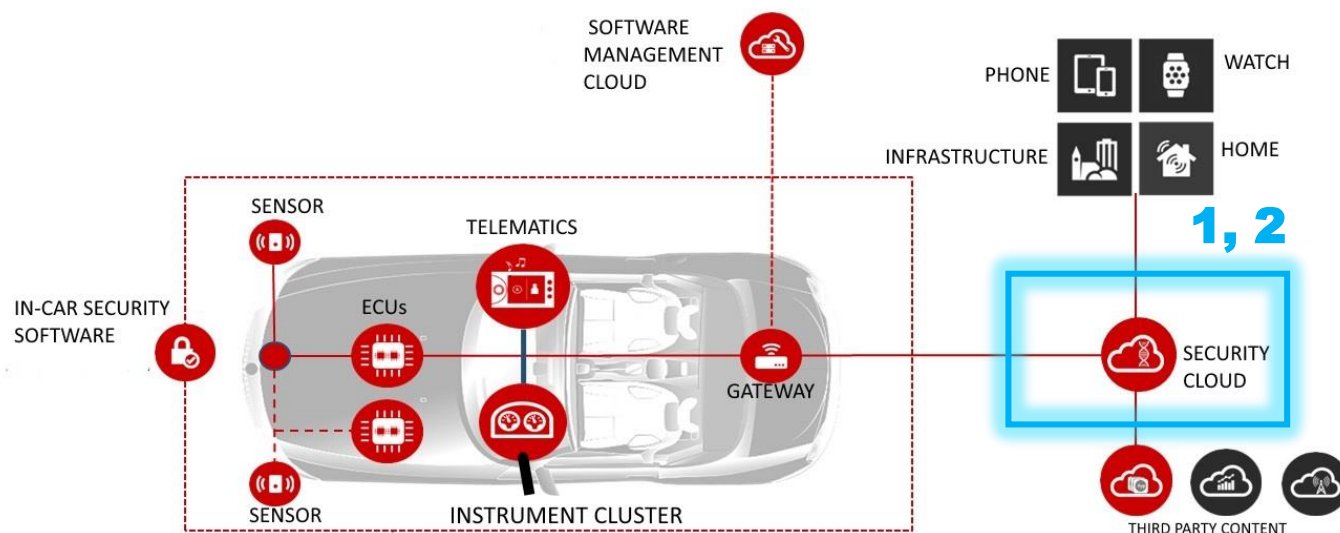
- Tesla会自动使用默认密码连接服务wifi并访问初始页面。
- 伪造wifi连接点，在初始网页嵌入恶意Javascript代码。利用WebKit漏洞和CVE-2011-3928，获得任意代码执行权限（信息泄露、释放后使用、越界访问、代码注入、突破沙盒）。
- 只获得浏览器权限（类似于用户权限）。



攻击案例：操控特斯拉轿车-2

○ 第二步：提权变成root

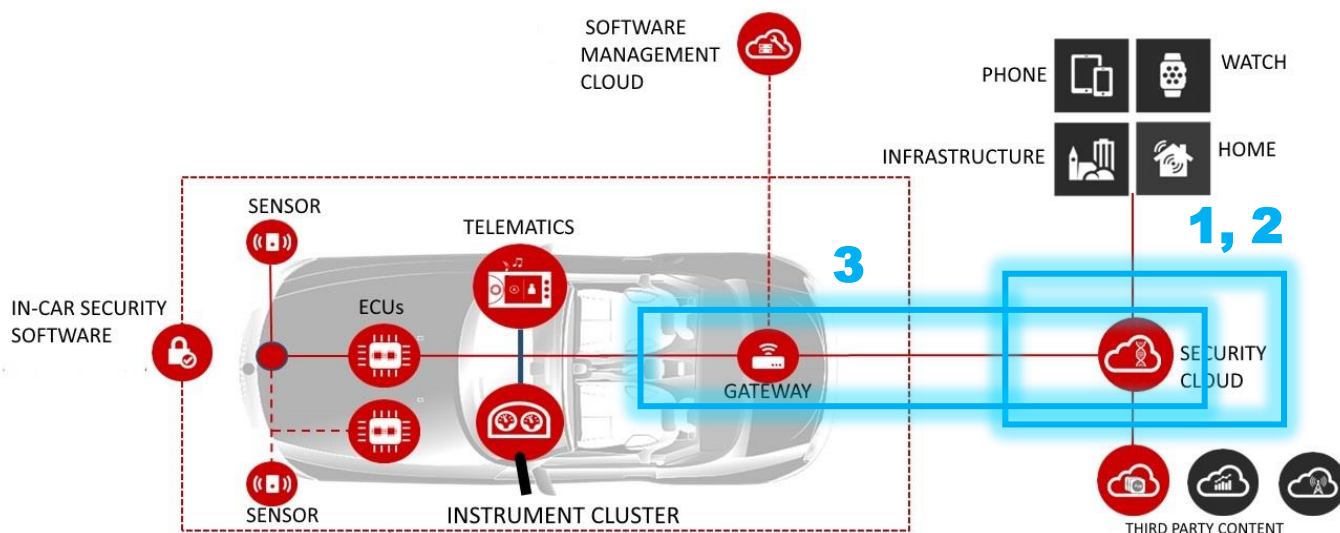
- 旧版本的Linux + AppArmor保护。
- 利用CVE-2013-6282，替换syscall程序，卸载AppArmor，修改root id，变成root（内核syscall缺陷）。
- 只能控制多媒体（由以太网连接）系统的权限，不能控制汽车实际控制系统（由网关物理隔离）。



攻击案例：操控特斯拉轿车-3

第三步：控制网关

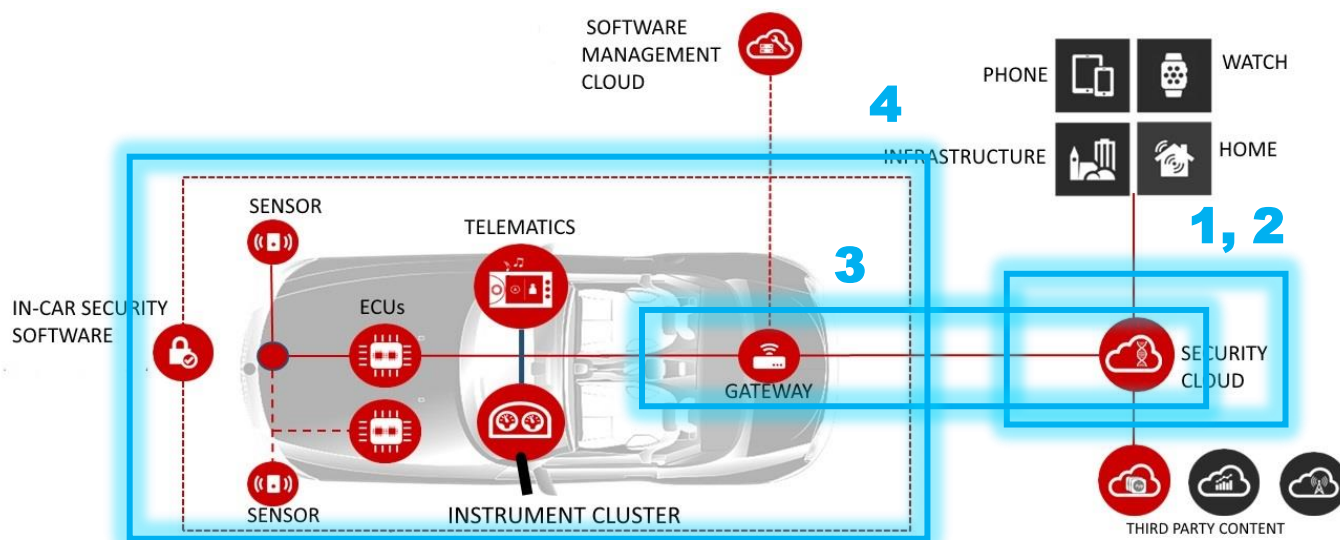
- Tesla的网关可以由多媒体系统触发执行固件更新操作。更新过程中的log多媒体系统可见，固件验证使用简单CRC32哈希（信息泄露、弱加密算法、弱认证保护）。
- 利用固件更新log，伪造更新固件和相应验证哈希，利用多媒体子系统触发固件更新，完全控制网关（代码注入）。
- 网关通过CAN网络控制汽车的各个ECU部件，CAN通讯协议专有不可知。



攻击案例：操控特斯拉轿车-4

○第四步：逆向部分CAN通讯协议

- 并没有什么好办法，通过大量实验逆向工程。
- 成功解析网关和车灯控系统之间的通讯，完成对车灯的任意开启控制。
- 车灯被远程控制，变成彩灯！



<https://www.blackhat.com/docs/us-17/thursday/us-17-Nie-Free-Fall-Hacking-Tesla-From-Wireless-To-CAN-Bus.pdf>

Tesla攻击总结

○攻击路径复杂

- 假wifi、网页、多媒体系统、网关、CAN网络、车灯ECU。

○攻击方法多样

- 网页JS注入、越界访问、信息泄露、释放后使用、代码注入、内核API缺陷、弱加密、弱认证、可信链条失败、逆向工程等等。

○攻击所需知识众多

- WebKit/Javascript、Linux 内核、固件设计、CAN总线、汽车控制系统等等。

真实世界的攻击是利用众多安全漏洞的复杂多步骤攻击工程。

攻击分类

○物理攻击

- 电磁干扰、辐射、功耗分析、电磁分析、等等。

○底层硬件攻击

- USB电流冲击、PCIe逆向、等等。

○软件简单攻击

- 缓冲区溢出、越界数据读写、释放后复用、代码指针改写、数据改写、shellcode注入、恶意程序注入、等等。

○软件组合攻击

- 利用缓存测信道的地址随机化绕过、利用代码/数据指针改写的代码复用、利用预测执行的内核信息泄露、利用API设计错误的内核提权、SGX信息泄露、等等。

○纯软件攻击

- 弱加密、弱认证、网络中间人、DNS劫持、ARP轰炸、密码撞库、钓鱼邮件、等等。

体系结构相关，利用软件来攻击硬件/系统。

以上分类并不完全，不能覆盖所有的攻击种类。

- 现实攻击的复杂性
- 计算机体系结构相关的攻击

内容概要

- 系统安全基础知识
- 安全漏洞
- 安全攻击
- **安全防御**
- 体系结构的安全问题
- 体系结构的安全防御
- 总结与讨论

攻击和防御的较量

○攻击

- 攻击是多步骤的
- 每一个步骤只要找到几个能用的漏洞就可以了
- 而现存的漏洞是很多的
- 攻击不怕尝试，100次失败，1次成功就可以
- 攻击需要创造性思维

○防御

- 防御往往是局部的
- 只要能完全防御住一个步骤即可抵御攻击
- 而完全防死一个步骤也很难
- 防御怕不全面，哪怕放过一个漏洞也是洞
- 不过，不完全的防御如果能极度加大攻击难度也是好防御
- 防御需要系统性思维

安全防御的基本原则

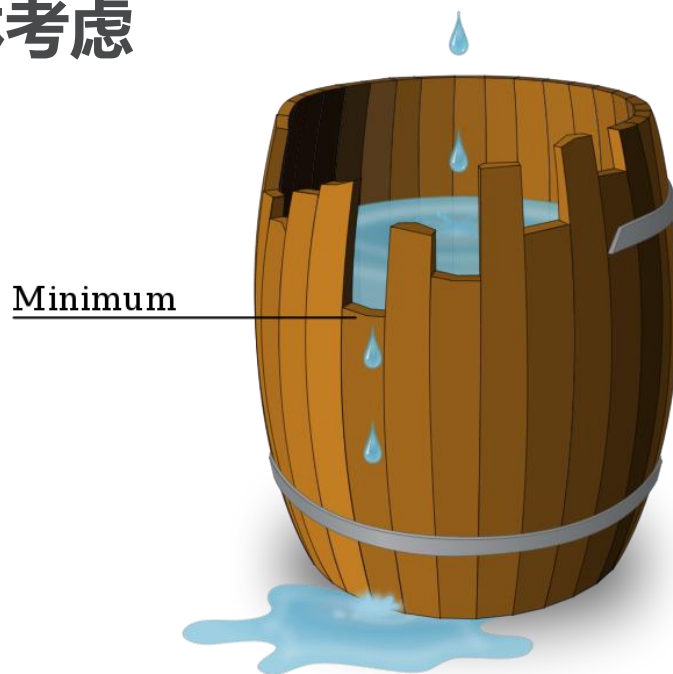
- 整体性原则
- 分层原则
- 最小特权原则
- 简单性原则

整体性原则

- 一个系统的安全由其最不安全的特性决定。
 - 攻击者往往首先攻击最容易被利用的漏洞
 - 最不安全的特性（某种程度上）决定了系统攻击难度，即安全性

- 安全系统的构建应从系统的整体考虑
 - 尽量达到先天/本征安全
 - 而非后天打补丁

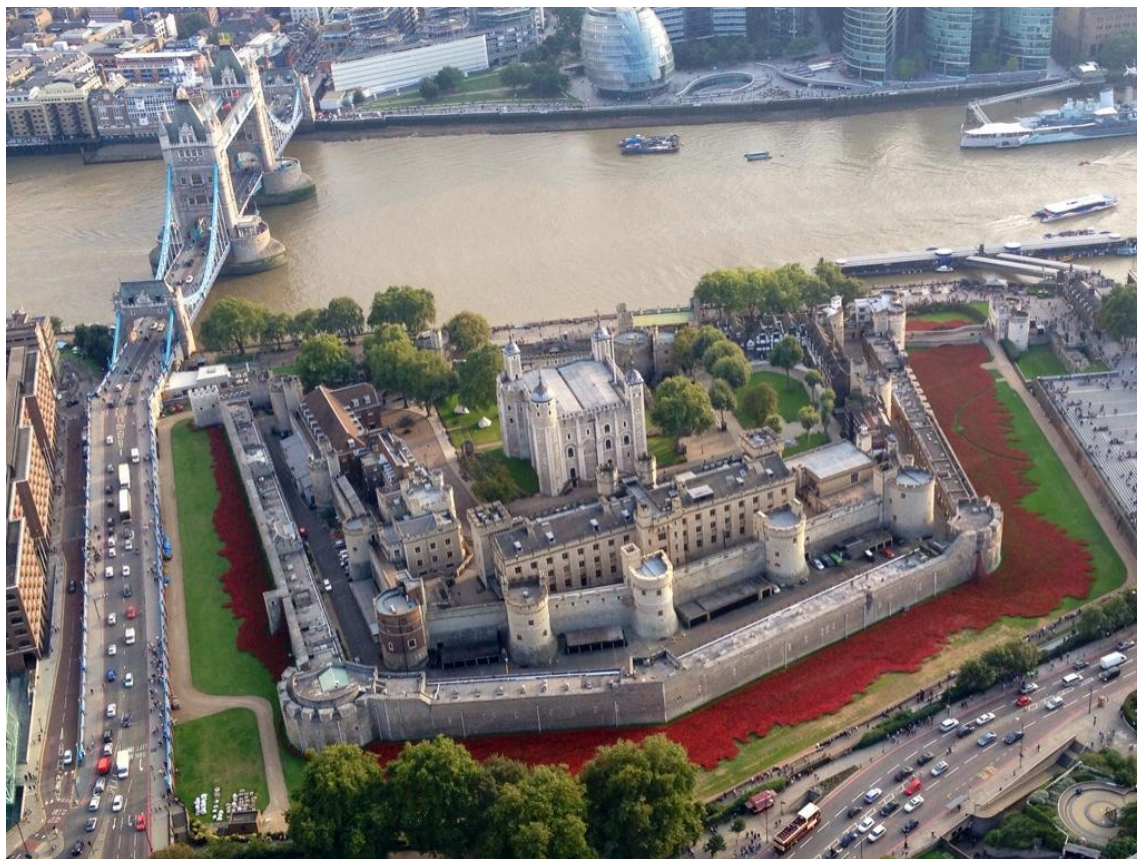
- 应整体考虑模块/层次之间的协议/通讯/协同



分层原则

- 系统就像一个城堡，要一层一层的防御。
 - 一层防御失效并不代表系统沦陷
 - 防御之间互相照应

伦敦塔：Tower of London



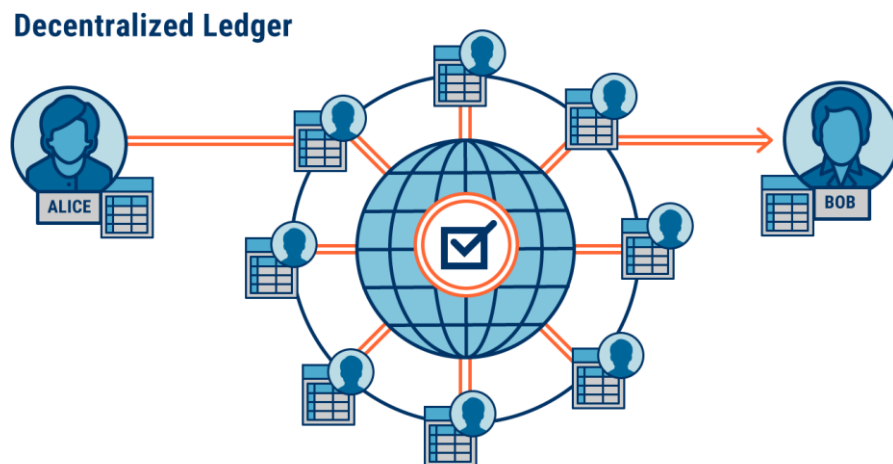
最小特权原则

- 权力分散，只给予所必需的最小特权，确保可能的事故、漏洞等原因造成的损失最小。

○ 银行和比特币

- 银行：知晓一切，单一失败节点（single point of failure）。

- blockchain：
分散记账
多个账本记录
不知记录具体内容



CBINSIGHTS

简单性原则

- 保持系统的简洁性，避免不必要的冗余
 - 容易理解则容易维护
 - 容易分析则容易防护全面
- KISS: Keep it simple, stupid!



OR



安全防御的基本方法

- 修补漏洞
- 隔离
- 行为分析和检测
- 验证
- 随机化
- 冗余

- 直接修补漏洞 (patch)

- 优点:

- 简单有效

- 缺点:

- 治标不治本

- 缺乏系统性

- 缺乏通用性

- 引入新漏洞

- 按优先级、用户、安全性分类，并按类分割资源
- 物理隔离：
 - 断网、禁止接入外部媒体
- 系统隔离：
 - 系统态与用户态
 - 多用户支持
 - 虚拟机（沙盒）
 - SGX
- 数据隔离：
 - 用户数据访问控制
 - 加密文件/磁盘
- 缺点：不能保护来自于同类的攻击。

- 假设：攻击行为和正常行为存在差异
- 运行前：
 - 基于特征的病毒/木马扫描
 - 加壳（加密存储，运行时解密）
- 运行时：
 - 基于规则的恶意行为检测、异常分析
 - 恶意行为隐藏，持久攻击（APT：advanced persistent threat）
- 运行后：
 - 报告分析
 - 海量数据、特征挖掘

○可信计算：

- 在运行前检验程序的数据完整性、可信来源等等
- 只运行可信的程序
- 缺点：
 - 并非所有的程序/数据都来自可信来源

○形式化验证：

- 验证硬件/软件不存在隐藏漏洞/后门
- 缺点：
 - 现实设计的复杂度过大，难以完整验证
 - 验证还不能处理所有攻击类别

○通过随机化来隐藏数据，防止信息泄露

- 地址随机化：阻挠程序静态分析
- 指令随机化：阻挠动态指令分析
- 内存数据随机化：阻挠动态数据分析
- 缓存随机化：阻挠缓存侧信道攻击

○缺点：

- 熵值较低
- 通过持续攻击解随机化

- 软硬件冗余

- 攻击者需要同时攻击多个软/硬件（异质）实例

- 拟态

- 模拟攻击目标来迷惑/捕获攻击

- 缺点：

- 冗余的实现代价较大

- 安全防御的基本原则
- 安全防御的几种方法

- 白帽和黑帽的区别?
- 为什么安全被忽视?
- 为什么安全那么难做?
- 为什么要讨论体系结构安全?

内容概要

- 系统安全基础知识
- 安全漏洞
- 安全攻击
- 安全防御
- **体系结构的安全问题**
- 体系结构的安全防御
- 总结与讨论

计算机体系结构的范畴

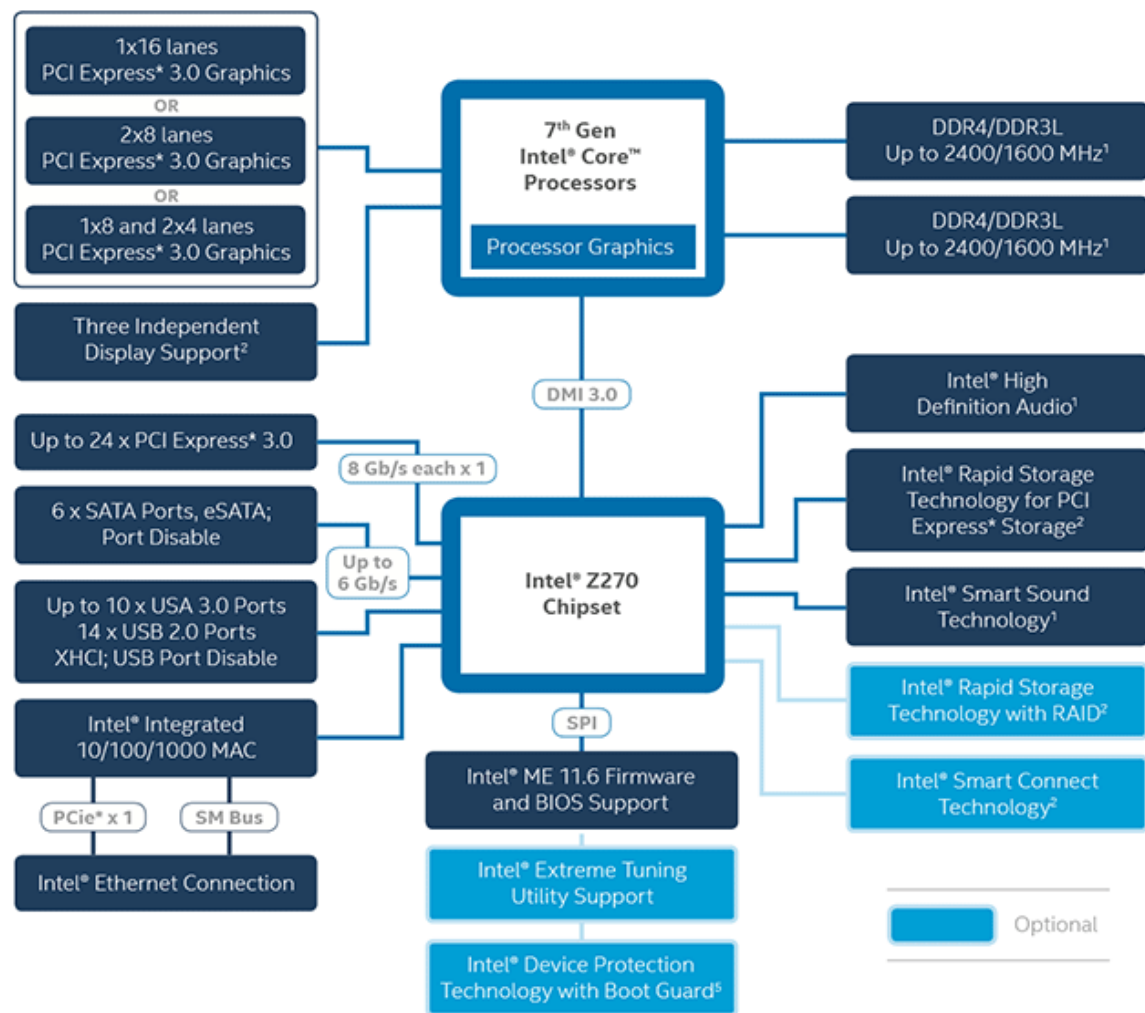
○ 狭义定义

- 专指指令集(ISA: **Instruction Set Architecture**)
- 硬件和软件的接口

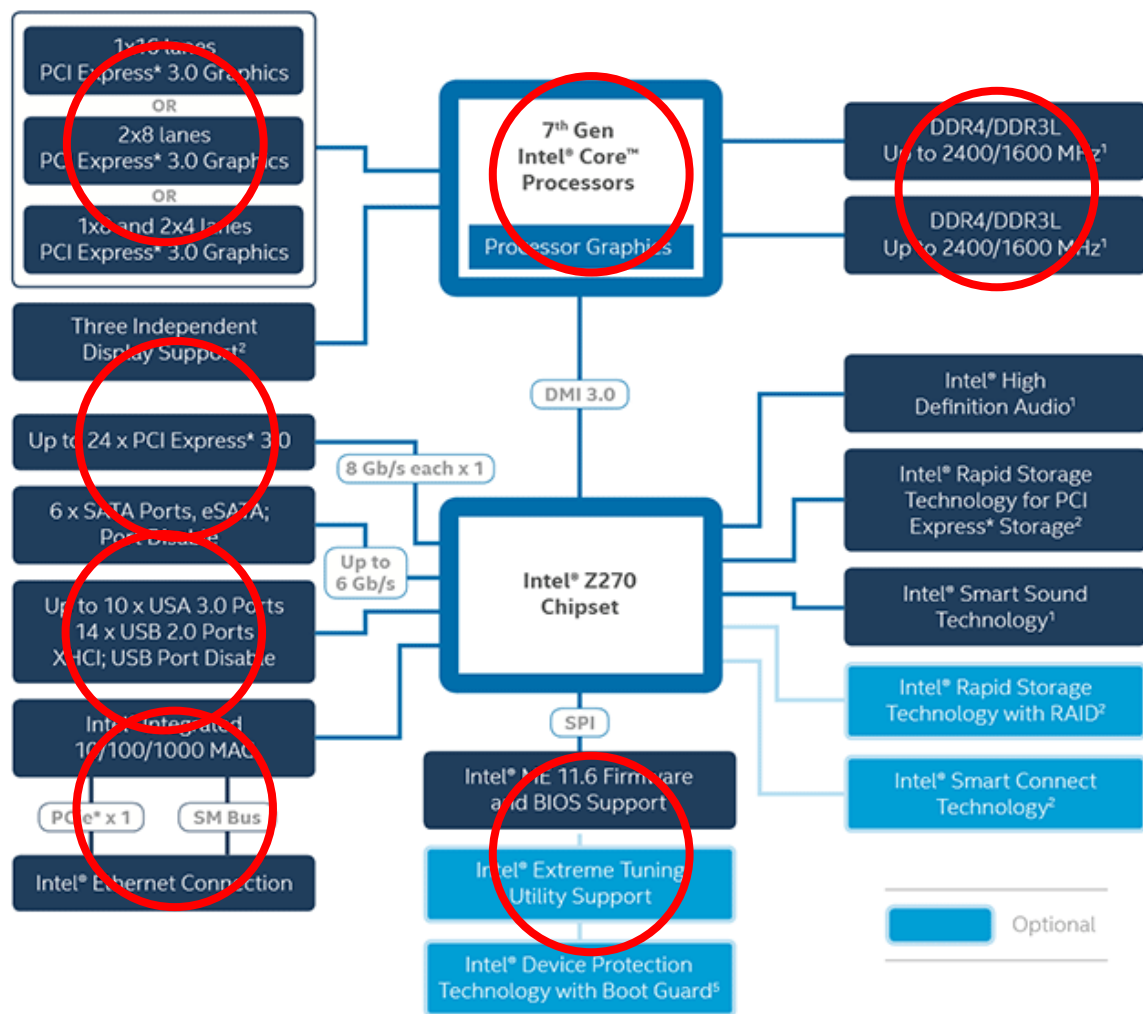
○ 广义定义

- **ISA: 指令集**
- **微结构(Microarchitecture): 处理器实现**
- **系统**
 - 硬件系统: 缓存、内存、外设、片上总线/网络
 - 软件系统: 操作系统、虚拟化、固件、系统加载

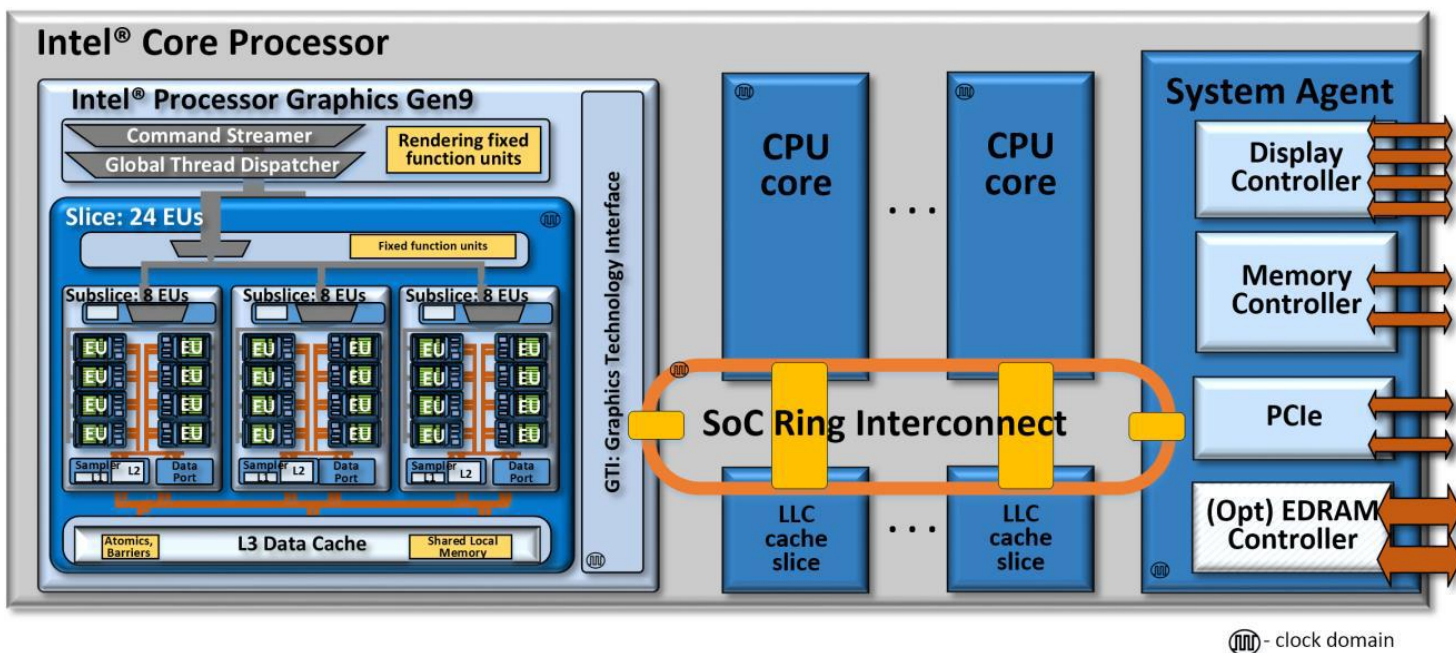
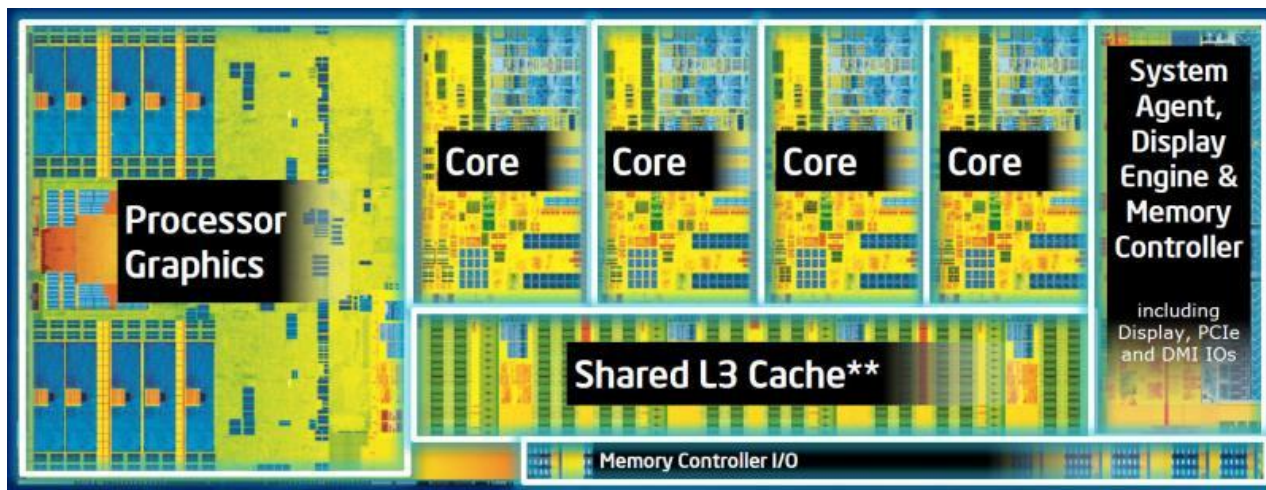
现代计算机结构-母版



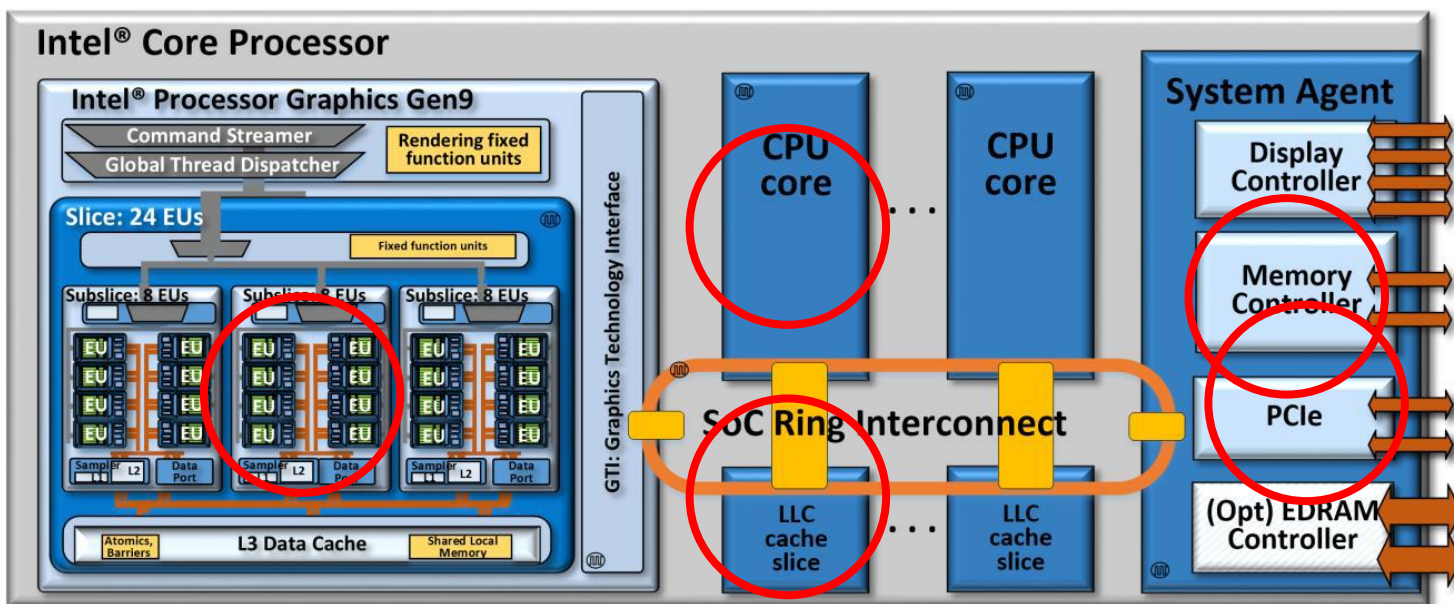
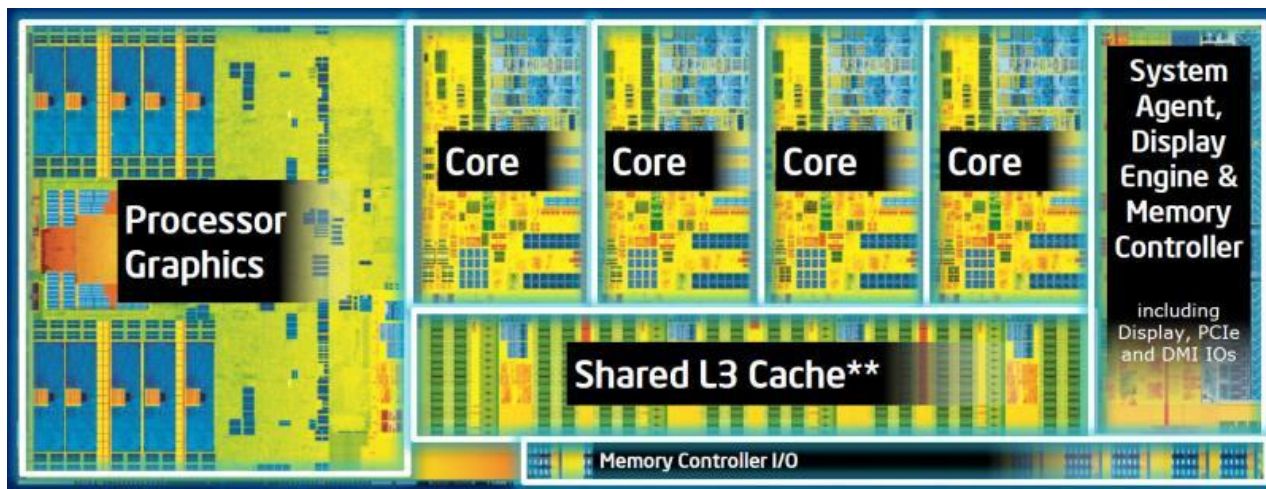
现代计算机结构-母版



现代计算机结构-处理器

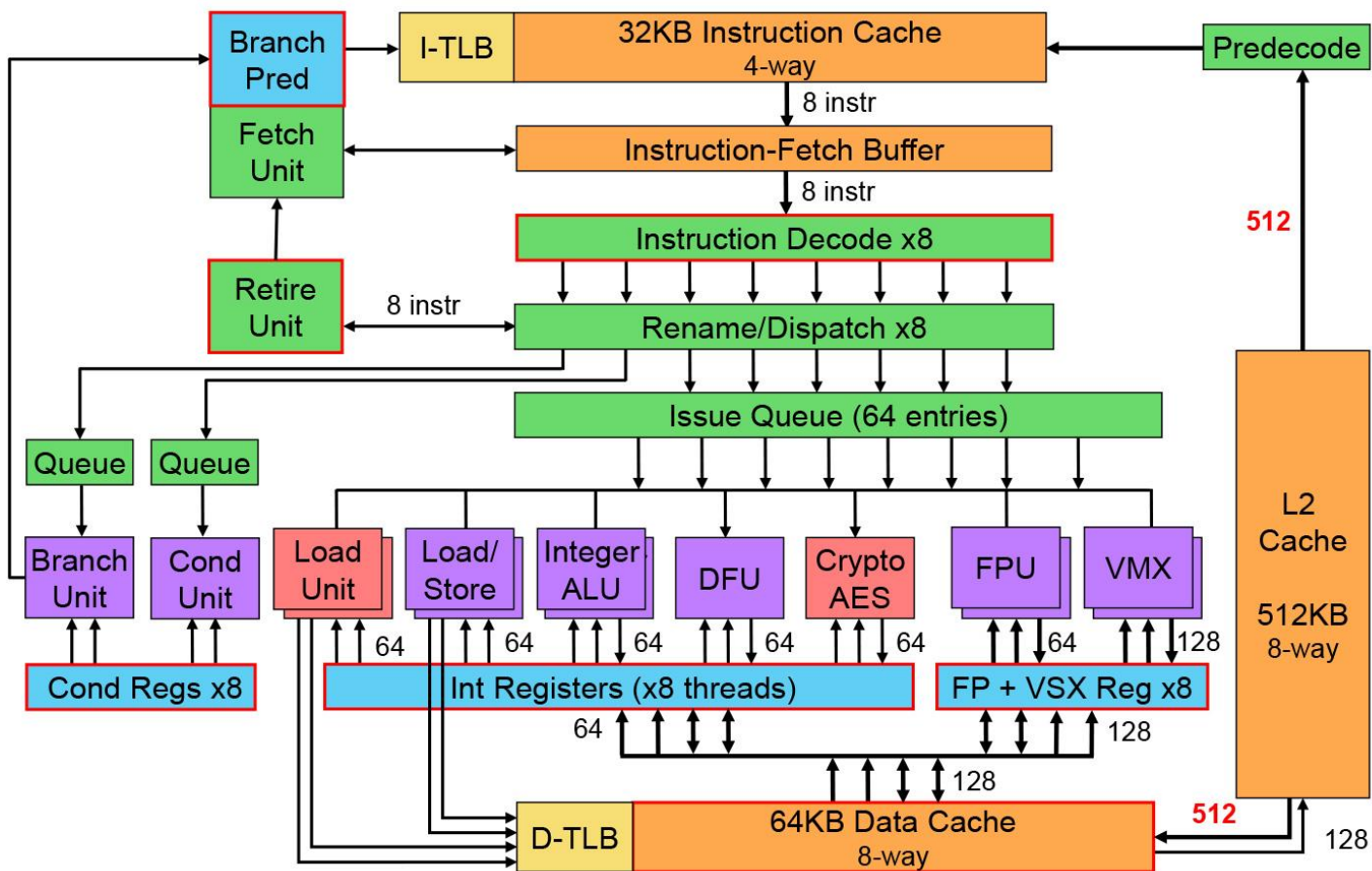


现代计算机结构-处理器

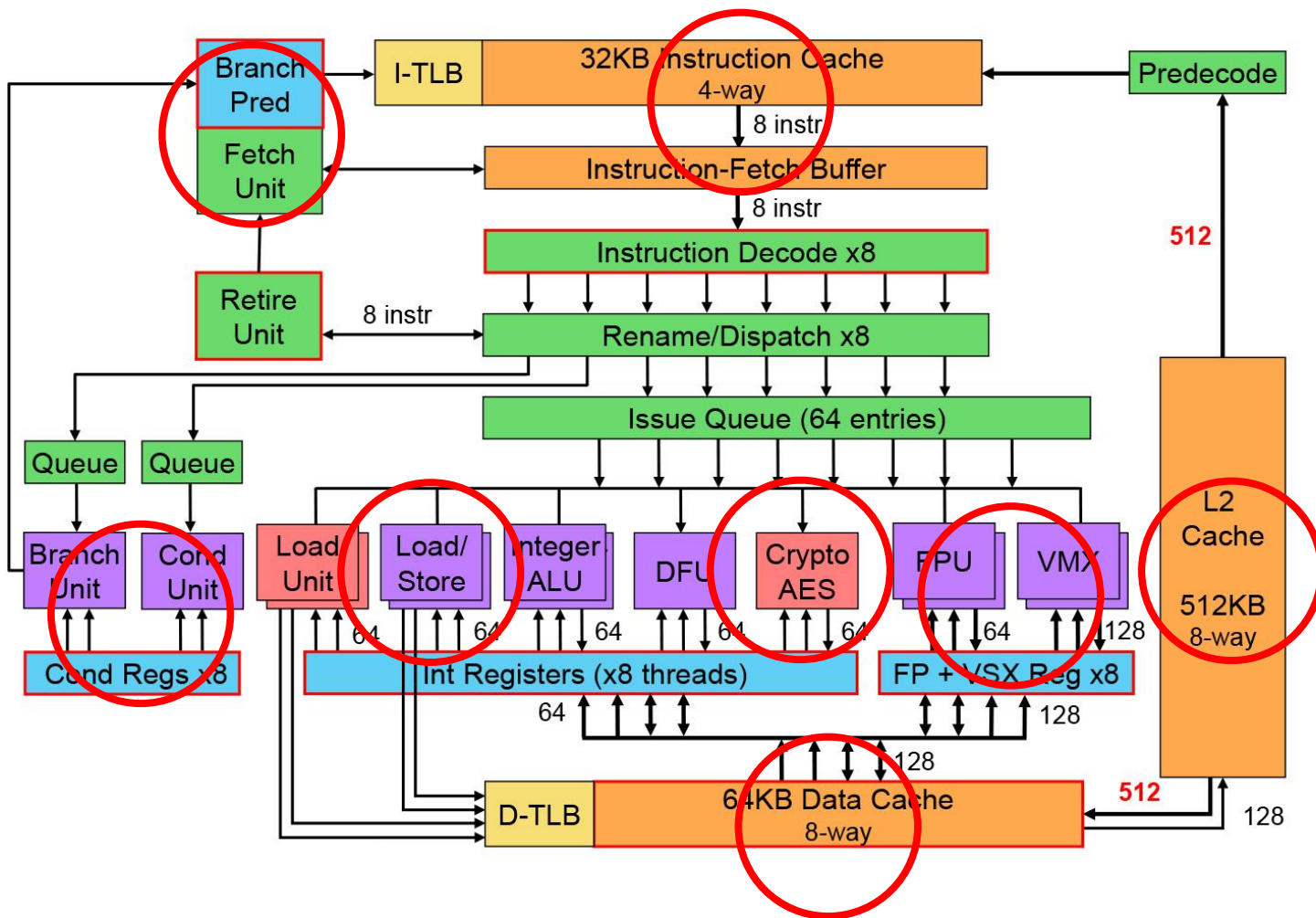


- clock domain

现代计算机结构-处理器核心



现代计算机结构-处理器核心



○体系结构中存在的信息泄漏问题

处理器核心存在的安全问题-寄存器

○ Intel's Lazy FPU flaw

- FPU寄存器是浮点运算单元的特殊寄存器

- FPU寄存器可能长达128比特，长于普通的64比特寄存器

- FPU寄存器有时被安全程序用来保存密钥

- 上下文切换时为了效率，FPU寄存器保留原值

- 恶意程序通过读取FPU寄存器获得密钥信息

- <https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00145.html>

- <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-3665>

○ 分析

- 直接信息泄露

- 为了执行效率而忽略安全

- 设计失误

处理器核心存在的安全问题-运行时间差

- 基于运算时间的timing attack
 - 单个指令的执行时间不同（普通指令和除法指令）
 - 不同输入导致的运行指令数量不同
 - 恶意程序通过匹配程序输入和程序运行时间的关联推断信息
 - 辅助使用性能计数器、通过推测控制流获得更多信息
 - <https://tlseminar.github.io/docs/ssl-timing.pdf>
 - <https://www.paulkocher.com/TimingAttacks.pdf>
- 分析
 - 间接信息泄露
 - 任何攻击者可控并可测的运行状态差别都可能造成侧信道

内存系统存在的安全问题-任意内存读取

○危险的指针

```
void *p = 用户输入;  
printf("%d", (unsigned int)p);
```

p指向代码、内核、其他线程、其他进程?

○按地址直接读取内存

- 忽略数据的来源
- 忽略数据的访问权限
- 高级语言到低级语言的语义丢失
- 低级语言对安全的忽视

○分析

- 直接信息泄露
- 现有的优先级、虚拟内存机制、进程隔离机制已防范大部分的攻击。
- 缺乏细粒度的（同一进程和优先级内）数据来源检查和访问控制。

内存系统存在的安全问题-缓存侧信道-flush

○基于flush的缓存侧信道攻击

- 缓存是缩小内存访问延时的重要模块
- 缓存与否可造成显著的内存访问延迟差异(20周期对200周期)
- x86-64系统允许用户程序使用flush指令去除特定缓存数据
- 攻击者通过flush指令扫描缓存
 - <https://www.usenix.org/node/184416>

○分析

- 间接信息泄露
- 微体系结构信息暴露
- flush指令权限管理缺失
- 缺乏细粒度的内存权限管理
- 任何攻击者可控并可测的运行状态差别都可能造成侧信道

内存系统存在的安全问题-缓存侧信道-prime

○基于prime的缓存侧信道攻击

- 缓存是缩小内存访问延时的重要模块
- 缓存与否可造成显著的内存访问延迟差异(20周期对200周期)
- 构造特殊内存访问序列(eviction set) 替换特定缓存块
- 攻击者通过eviction set扫描缓存
 - <http://css.csail.mit.edu/6.858/2014/readings/ht-cache.pdf>

○分析

- 间接信息泄露
- 微体系结构信息暴露
- 任何攻击者可控并可测的运行状态差别都可能造成侧信道

处理器核心存在的安全问题-瞬时攻击

○推测执行

```
raise_exception();  
access(probe_array[data * 4096]);
```

○瞬时攻击 (Transient Attack)

- 推测执行的指令仍然会在缓存等微体系结构中留下痕迹
- 侧信道攻击可以将微体系结构中的信息暴露出来
- 攻击者通过构造特殊的推测执行来泄露信息
 - <https://meltdownattack.com/meltdown.pdf>

○分析

- 微体系结构的改变被忽视
- 推测执行被劫持
- 推测执行时的权限检查丢失

I/O系统存在的安全问题-DMA攻击

○片上总线上的DMA攻击

- DMA: Direct Memory Access, 无需处理器参与的直接内存访问
- 高速外设(USB、PCIe、FireWire)可通过DMA访问内存
- 恶意设备可伪装成USB、PCIe、FireWire设备直接访问内存
 - https://en.wikipedia.org/wiki/DMA_attack

○分析

- 直接信息泄露
- 为了执行效率而忽视了安全
- 内存访问管理漏洞
- 体系结构设计失误

○体系结构中存在的恶意信息修改问题

内存系统存在的安全问题-恶意内存修改-溢出

○危险的指针

```
char p[8];  
char *str = "用户输入";  
while(str != 0) *p++ = *str++;
```

p指向代码、内核、其他线程、其他进程?

○写溢出

- 未检查数据的边界，越界写不报错
- 高级语言到低级语言的语义丢失
- 低级语言对安全的忽视

○分析

- 直接数据修改
- 缺乏细粒度的数据访问控制
- 缺乏数据边界信息

○强制类型转换

```
byte b = 0; byte *bp = &b;  
double *f = (double *)bp;  
*f = 0.01;
```

○数据写边界扩大

- 低级语言不检查强制类型转换
- 指针指向了更大的（错误的）数据范围
- 转换后的写行为导致错误的数据被覆盖
- 高级语言到低级语言的语义丢失

○分析

- 直接数据修改
- 缺乏细粒度的数据访问控制
- 缺乏数据类型信息

内存系统存在的安全问题-恶意内存修改-UAF

○ Use after free (UAF)

```
int *a = malloc(sizeof(int));  
free(a);  
int *b = malloc(sizeof(int));  
*a = 5; // actually write to *b
```

○ 野指针被使用

- 低级语言不检查指针的有效性
- 指针指向了已经被释放的内存
- 被释放的内存又被分配
- 高级语言到低级语言的语义丢失

○ 分析

- 直接数据修改
- 缺乏指针的有效性检查
- 缺乏动态内存的安全防护

内存系统存在的安全问题-直接代码注入

○代码注入

- 攻击者通过内存漏洞直接将代码写入可写内存
- 溢出、UAF
- 再利用内存漏洞（比如更改一个函数指针）执行写入的代码
- 恶意代码被执行

○分析

- 内存漏洞导致代码被写入可写内存
- 内存漏洞导致控制流被修改
- 代码执行是未检查代码来源

- 思考：即时编译（Just-in-time JIT）怎么办？

- 防御：可写不可执行，数据流分析

内存系统存在的安全问题-代码复用

○代码复用

- 可写内存不可执行保护
- 静态代码分析
- 利用已有代码完成单个指令功能
- 利用栈（可写函数局部存储）构造间接跳转序列
- 利用已有代码完成攻击

○分析

- 可写区间的数据影响了函数跳转（间接跳转）流
- 间接跳转无检查，函数指针无保护

- 防御：内存随机化（ASLR），阻止静态分析

内存系统存在的安全问题-运行时代码复用

○运行时代码复用

- 动态只读内存扫描
- 动态代码分析
- 动态攻击构造和跳转序列生成
- 利用栈（可写函数局部存储）存储间接跳转序列
- 利用已有代码完成攻击
 - <https://ieeexplore.ieee.org/document/6547134>

○分析

- 可写区间的数据影响了函数跳转（间接跳转）流
- 间接跳转无检查，函数指针无保护
- 阻止静态分析治标不治本

- 防御：代码指针完整性、控制流完整性

内存系统存在的安全问题-非控制数据攻击

○非控制数据攻击

- 既然代码指针被保护
- 修改保存代码指针的数据指针 (VTable)
- 修改跳转判断相关的数据
- 仍然可以改变程序的行为

○分析

- 所有数据的完整性都需要保护
- 防御：数据完整性 (spatial and temporal)

针对体系结构的硬软件攻击

○泄露

○直接

- 寄存器、控制寄存器
- 内存
- 总线

○间接（侧信道）

- 流水线侧信道、瞬时攻击
- 缓存侧信道

○修改

○直接

- 内存数据任意写
- 直接代码注入

○间接

- 代码复用
- 非控制数据攻击

○其他

虚拟化、可信计算（SGX、TrustZone）、启动（Intel ME）、DDoS、功能计数器、调试、后门。

内容概要

- 系统安全基础知识
- 安全漏洞
- 安全攻击
- 安全防御
- 体系结构的安全问题
- **体系结构的安全防御**
- 总结与讨论

操作系统中的防御—优先级权限管理

○基于优先级的权限管理

○用户态程序看不见系统态信息

- 页表、性能计数器、缓存控制指令

○系统看不见虚拟机信息

- 同驻OS，物理内存分配

○系统看不见底层机器信息

- 中间件、固件、安全启动

○分析

○不同优先级用户拥有不同权限

○恶意操作系统/虚拟机？（云安全）

- 隔离运行（SGX）

○操作系统操作用户空间？（用户态注入内核恶意代码）

- SMEP/SMAP

○基于用户的权限管理

○Linux的用户/组管理

○文件访问控制

○虚拟内存/PID进程号

○内存中的数据隔离

○分析

○不同用户间的资源隔离

○共享动态链接库

○基于动态库的代码复用攻击、侧信道攻击

○共享内存

○基于共享内存的侧信道攻击

○共享处理器线程

○基于流水线的侧信道攻击

○共享末级缓存 (LLC)

○基于LLC的侧信道攻击

操作系统中的防御—沙盒

- **在用户态中提供隔离环境**
 - **虚拟系统**
 - 防止恶意操作系统
 - **即时编译**
 - 浏览器沙盒、防止恶意网站、多媒体文件
- **分析**
 - **为不可信/不安全的执行提供资源隔离**
 - **沙盒和被隔离程序仍然共享大量资源**
 - 提权和越狱攻击

操作系统中的防御—不精确的时钟

- 降低用户态/沙盒内程序可获得时钟的精度
 - 大量的测信道攻击为利用时间测信道
 - 攻击者和被攻击者的时间同步
- 分析
 - 攻击者可以自己构造高精度时钟
 - 大量用户程序依赖于高精度时钟
 - 方法简单，在沙盒中大量使用

操作系统中的防御—清除上下文

- 在进程/优先级切换时的上下文清除
 - 一般寄存器/FPU寄存器 (vector, etc.)
 - 缓存清除
 - 一般是L1(TLB), 防止基于L1的侧信道
- 分析
 - 寄存器清除基本已是标准操作(FPU/GPU不一定)
 - 缓存清除会带来性能代价
 - 在高可靠性时采用

操作系统中的防御—可写不可执行

- 严格区分代码和数据
 - 代码不可写，数据不可执行
 - 防止直接代码注入
 - 大量部署
 - 可执行不可读（区分代码和只读数据）
- 分析
 - 基本防止的简单的代码注入攻击
 - 直接触发了代码复用攻击

操作系统中的防御—地址随机化 (ASLR)

○利用地址随机化防止静态代码分析

- 程序加载时随机化页地址
- 代码复用攻击依赖于大量的事前分析
- 静态分析不能获得动态代码/数据地址
- 用户和内核态地址随机化
- 大量部署

○分析

- 为代码复用攻击加大了难度
- 所引入的熵值很低
- 随机化在程序运行期间不变 (内核地址随机化)
- 内核地址随机化已死!
 - “KASLR is Dead: Long Live KASLR” 2017
https://link.springer.com/chapter/10.1007%2F978-3-319-62105-0_11

操作系统中的防御—页表隔离

○用户和系统不共享页表

- 以前：系统也表是用户页表的一部分，有利于syscall
- 用户态程序将彻底不能解析系统页表
- 只有syscall所在页被同时放于两个页表
- 减少用户对系统内存空间的攻击

○分析

- 很有效的防御机制
- 需要处理器硬件支持
- 即将被大量部署（meltdown的主要防御手段）

操作系统中的防御—页着色

- 利用虚拟内存分配阻止缓存侧信道
 - 为不同进程/用户/优先级分配页着色
 - 着色不同的虚拟页不共享cache set
 - 不同着色的程序间不能发起缓存侧信道攻击
- 分析
 - 软件缓存隔离
 - 运行代价较低
 - 着色数收缓存大小限制
 - 对同着色内的攻击失效

编译器中的防御—栈保护

- 在栈的边界上添加canary
 - 当canary被改写时说明栈溢出
 - 大量部署
- 分析
 - 为栈溢出攻击添加了一定难度
 - Canary值可以被猜出
 - Canary位置已知，可以绕过
 - 不一定需要栈溢出
 - 实用但是防御薄弱

- 检查代码中指针的误用
 - 从源头上减少内存错误
 - 代码复用攻击依赖于代码中已有的bug
 - 大量部署
- 分析
 - 有事被忽略
 - 检查误报
 - 特殊使用被误报
 - 攻击不依赖于简单错误
 - 实用，应该变为软件从业基本检查

编译器中的防御—静态CFI

- 编译器添加的CFI检查
 - 编译器静态分析程序控制流
 - 将控制流写入代码中
 - 在函数跳转时检查控制流是否匹配
- 分析
 - 能防范大部分的CFI攻击
 - 静态控制流分析不准确
 - 对动态链接库支持不好
 - 程序运行代价大
 - 已部署LLVM，还未大量推广使用

处理器中的防御—加解密指令集

- 专门用于加解密运算的指令和运算单元
 - 加速加解密的速度
 - 加解密时间不依赖于数据
 - 减少侧信道的途径
- 分析
 - 有效的加速的加解密运算
 - 明显降低的时间攻击的可能性
 - 已是业界的标准做法
 - 缓存侧信道仍然存在
 - 物理测信道仍然存在

处理器中的防御—Intel MPX

- Intel MPX (Memory Protection Extensions)
 - ISA指令集扩展，用于支持数据完整性（溢出、空间完整性）
 - GCC 5 / Linux 4.18 支持
 - 为数据添加metadata，记录数据的边界
- 分析
 - 平均代价50%，最差>200%
 - 对后兼容性导致漏洞
 - 不能完全保证完整性（时间完整性）
 - 还不如编译器内存泄露检查
 - GCC/Linux已有提意建议撤出支持

处理器中的防御—ARM PA

- ARM Pointer Authentication (PA)
 - ISA扩展，使用加密技术支持CPI
 - 为指针添加加密支持，保存指针对应的MAC
 - 使用指针时检查MAC
 - 指针和MAC不匹配则失败
- 分析
 - 可以有效抵御大部分的指针完整性攻击
 - 不能完全阻止重演攻击 (replay attack)
 - 性能代价

处理器中的防御—Intel SGX

- Intel Software Guard Extensions (SGX)
 - ISA指令集扩展，用于支持硬件隔离的安全执行环境
 - 隔离的数据和代码空间
 - 加密的内存存储
 - 基于可信的加密认证
 - 操作系统协助页表分配
 - 类似：ARM TrustZone
- 分析
 - 解决不可信云平台执行私有超算的问题
 - 基本有效
 - 操作系统协助页表分配导致了漏洞
 - 共享缓存仍然存在侧信道
 - 被隔离的安全程序可能成为触及不到攻击源

处理器中的防御—Intel CET

- Intel Control-flow Enforcement Technology (CET)
 - ISA指令集扩展，用于支持CFI
 - 硬件影子栈支持完整的反向CFI
 - 粗粒度正向CFI
 - 操作系统协助影子栈分配
- 分析
 - 能有效防御大部分的CFI攻击
 - 粗粒度前向CFI
 - 操作系统协助影子栈分配存在漏洞
 - **还没有处理器公布!**

处理器中的防御—Intel CAT

- Intel Cache Allocation Technology (CAT)
 - 扩展CPUID，硬件LLC分割 (partitioning)
 - 不同partition的数据互相不共享LLC
 - 阻止不同partition之间的缓存侧信道
- 分析
 - 能有效防御partition之间的缓存侧信道攻击
 - 对partition内的攻击无效
 - 对L1的攻击无效
 - 影响性能 (降低了LLC的效率)

处理器中的防御—ARM Tagged Memory

- 使用硬件标签标注内存区域
 - 类似于内存着色，利用标签标注页的分配
 - 用更细粒度的页分配来防止越界访问
 - 页粒度的资源隔离
- 分析
 - 也许有用
 - 页粒度较大
 - 标签大小限制了着色数量
- 最近添加了细粒度的标签，搭配PA实现CFI
- 准备进入LLVM

研究领域提出的防御

- 指令随机化
- 细粒度的动态内存空间随机化
- 随机内存
- 随机缓存
- 基于硬件的ROP/JOP检测
- 基于标签的CPI/CFI检查
- 缓存/总线监视器
- 安全协处理器

体系结构的硬软件防御

○泄露

○直接

- 寄存器、控制寄存器 (清除上下文)
- 内存 (内存访问控制、随机化)
- 总线 (IOMMU、IO防火墙)

○间接 (测信道)

- 流水线测信道、瞬时攻击 (改进处理器/ISA设计)
- 缓存测信道 (缓存隔离、随机化、加密)

○修改

○直接

- 内存数据任意写 (访问控制、数据完整性、编译器优化)
- 直接代码注入 (可写不可执行)

○间接

- 代码复用 (CFI、CPI、隔离、编译器优化)
- 非控制数据攻击 (数据完整性、编译器优化、DFI)

○其他

虚拟化、可信计算 (SGX、TrustZone)、启动 (Intel ME)、DDoS、功能计数器、调试、后门。

- 为什么软件不能解决所有安全问题?
- 硬件是否能解决所有问题?
- 为什么要提出体系结构安全的概念?
- 测信道的根源?
- 数据完整性的含义?

Q&A