第6章

访问控制

微软本可以将有效的安全措施纳入标准,但理智占了上风。安全系统有适得其反的坏习惯,毫无疑问它们会造成巨大的问题。

– 里克·梅伯里

优化包括采用一些有用的东西,然后用几乎可以用但更便宜的东西代替它。

罗杰·李约瑟

6.1 简介

我最初是在一台 IBM 大型机上学习编程的,它的输入是穿孔卡片,输出是打印机。您拿着一副纸牌排队,运行作业,然后带着打印输出离开。所有的安全都是有形的。随后出现了可以同时运行多个程序的机器,以及防止一个程序干扰另一个程序的保护问题。您不希望病毒从您的浏览器窃取密码,或修补银行应用程序以窃取您的钱。许多可靠性问题源于应用程序相互误解或相互争斗。但是当客户希望应用程序共享数据时,将应用程序分开是很棘手的。

如果您的电子邮件客户端和浏览器在不同的机器上运行,这将使网络钓鱼变得更加困难,因此您无法仅单击电子邮件中的 URL,但这会让生活变得太艰难。

从 20 世纪 70 年代开始,访问控制成为计算机安全的重心。这是安全工程与计算机科学相遇的地方。它的功能是控制哪些主体(人、进程、机器……)可以访问系统中的哪些资源 他们可以读取哪些文件,可以执行哪些程序,如何与其他主体共享数据,等等.它变得极其复杂。如果您首先翻阅 700[illegible]

在 O/S 级别的 Windows,您的第一反应可能是“我希望我能学音乐！”在本章中,我试图帮助您理解这一切。

访问控制在许多不同的级别上工作,至少包括：

1. 应用层的访问控制可以表达一个非常丰富的、特定领域的安全策略。在您回答几个安全问题之前,银行的呼叫中心工作人员通常不允许查看您的帐户详细信息;这不仅可以阻止外人冒充您,还可以阻止银行工作人员查找名人或他们的邻居的账户。有些交易可能还需要主管的批准。与现代社交网站上的访问控制的复杂性相比,这不算什么,现代社交网站将有一大堆关于谁可以查看、复制和搜索来自谁的数据的规则,以及用户可以设置以修改这些规则的隐私选项.

2. 应用程序可以写在中间件之上,例如网络浏览器、银行的簿记系统或社交网络的数据库管理系统。这些强制实施了许多保护属性。例如,簿记系统确保借记一个账户的交易必须记入另一个账户,借方和贷方保持平衡,这样钱就不会被创造或销毁;他们还必须允许系统的状态在以后重建。

3. 由于操作系统从较低级别的组件构建文件和通信端口等资源,它必须提供控制对它们的访问的方法。您的 Android 手机将不同公司编写的应用程序视为不同的用户,并保护他们的数据免受彼此的影响。

   当共享服务器将属于不同用户的虚拟机、容器或其他资源分开时,也会发生同样的情况。

4. 最后,操作系统依赖于处理器及其相关内存管理硬件提供的硬件保护,这些硬件控制给定进程或线程可以访问哪些内存地址。

当我们从硬件开始,通过操作系统和中间件再到应用程序层时,控制变得越来越复杂,可靠性也越来越低。我们发现在多个层上实现了相同的访问控制功能。例如,Android 提供的不同手机应用程序之间的分离反映在您的浏览器中,浏览器根据其来源的域名分离网页材料（尽管这种分离通常不太彻底）。而在应用层或中间件层构建的访问控制可能会在很大程度上复制底层操作系统或硬件中的访问控制。它可能会变得非常混乱,要理解它,我们需要了解基本原则、通用架构以及它们是如何演变的。

我将从讨论支持多进程隔离的操作系统保护机制开始 o 。这些在历史上最早出现 与 1960 年代的第一个分时系统一起被发明 并且它们仍然是构建许多更高层机制的基础,并在更高层激发类似的机制。他们经常被描述为

自主访问控制 (DAC) 机制,将保护留给机器操作员,或强制访问控制 (MAC) 机制,通常由供应商控制,保护操作系统本身不被恶意软件修改。我将介绍软件攻击和抵御它们的技术 MAC、ASLR、沙盒、虚拟化以及硬件可以做什么。现代硬件不仅为虚拟化和功能提供 CPU 支持,还提供硬件支持,例如用于可信启动的 TPM 芯片,以阻止恶意软件的持续存在。这些帮助我们解决旧的单用户 PC 操作系统（如 DOS 和 Win95/98)的有毒遗留问题,这些操作系统允许任何进程修改任何数据,并限制许多不会运行的应用程序,除非你欺骗它们认为它们是以管理员权限运行。

6.2 操作系统访问控制

操作系统提供的访问控制通常使用某种机制验证委托人,例如电话中的密码或指纹,或服务器中的密码或安全协议,然后授权对文件、通信端口和其他系统资源的访问。

访问控制通常可以建模为访问权限矩阵,其中文件的列和用户的行。我们将写 r 表示读取权限,w 表示写入权限,x 表示执行程序的权限,以及 - 表示根本没有访问权限,如图 6.1 所示。

	经营账目会计审计 系统程序		数据	踪迹
山姆	rwx	rwx	写字	r
爱丽丝	x	x	写字	-
鲍勃	收据	r	r	r

图 6.1 – 朴素的访问控制矩阵

在这个简化的示例中,Sam 是系统管理员并且具有通用访问权限（审计跟踪除外,即使他也只能读取）。

经理 Alice 需要执行操作系统和应用程序,但只能通过批准的接口 她不能篡改它们。她还需要读写数据。审计员鲍勃可以阅读所有内容。

这通常就足够了,但在簿记系统的特定情况下,这并不是我们所需要的。我们要确保交易的格式正确 每笔借方都由别处的贷方平衡 所以我们不希望爱丽丝对账户文件拥有不受限制的写入权限。我们还希望 Sam 没有此访问权限。所以我们更希望只有通过会计程序才能对会计数据文件进行写访问。访问权限现在可能如图 6.2 所示：

用户	经营账目会计审计 系统程序		数据	踪迹
山姆	rwx	rwx	r	r
爱丽丝	收据	X	-	-
会计程序	收据	收据	写字	w
鲍勃	收据	r	r	r

图 6.2 – 用于簿记的访问控制矩阵

表达这种类型策略的另一种方式是使用（用户、程序、文件）的访问三元组。在一般情况下，我们关心的不是一个程序，而是一个保护域，它是一组共享对相同资源的访问的进程或线程。

访问控制矩阵（无论是二维还是三维）可用于实施保护机制以及对其进行建模。但它们无法很好地扩展：一家拥有 50,000 名员工和 300 个应用程序的银行将拥有一个包含 15,000,000 个条目的矩阵，这不仅会增加性能开销，而且容易受到管理员错误的影响。我们将需要一种更好的方法来存储和管理这些信息，两个主要选项是压缩用户和压缩权限。对于第一个，我们可以使用组或角色同时管理大量用户，而对于第二个，我们可以按列（访问控制列表）或行（功能，也称为协议的“票”）存储访问控制矩阵工程师和手机上的“权限”）[1639，2020]。

6.2.1 组和角色

当我们审视大型组织时，我们通常会发现大多数员工都属于少数类别之一。一家银行可能有 40 或 50 名员工：出纳员、呼叫中心接线员、信贷员等。只有几十个人（安全经理、首席外汇交易商……）需要个人定制的访问权限。

因此，我们需要设计一组组或功能角色，可以将 sta 分配给这些组。一些供应商（例如 Microsoft）几乎可以互换使用组和角色这两个词，但更仔细的定义是，组是主体列表，而角色是一个或多个主体可能承担的一组固定访问权限一段的时间。角色的典型例子是船上的手表主管。任何时候都只有一名值班员，并且有一个正式的程序，当换班时，一名值班员换另一名值班员。在大多数政府和企业应用程序中，重要的是角色而不是个人。

组和角色可以组合。当前在海上的所有船只的watch ocers是一组角色。在银行业务中，剑桥分行的经理可以通过集团经理的成员资格和担任剑桥分行代理经理的角色来表达他们的特权。集团经理可能会表示在组织中的级别（甚至可能是薪水范围），而角色代理经理可能包括一名助理会计师，而经理、副经理和分公司会计师都生病了。

我们是否需要注意这种区别是应用程序的问题。在军舰上,如果所有高级人员都被杀,即使是普通海员也可能会站岗。在一家银行,我们可能有一项政策,“超过 1000 万美元的转账必须由两名员工批准,一名至少是经理,另一名至少是助理会计师”。如果分公司经理生病了,那么作为经理的助理会计师可能不得不让区域总部在大额转账上提供第二个签名。

6.2.2 访问控制列表

简化访问权限管理的传统方法是每次将访问控制矩阵存储为一列,以及该列所指的资源。这称为访问控制列表或 ACL(发音为“ackle”)。

在我们上面的第一个示例中,文件 3(帐户文件)的 ACL 可能如图 6.3 所示。

用户计费	数据
山姆	写字
爱丽丝	写字
鲍勃	r

图 6.3 – 访问控制列表 (ACL)

作为管理安全状态的一种方式,ACL 有许多优点和缺点。在用户管理自己的文件安全的环境中,它们是自然的选择,并从 1970 年代开始在 Unix 系统中广泛使用。它们是基于 Unix 的系统(例如 Linux 和 Apple 的 macOS)以及衍生产品(例如 Android 和 iOS)中的基本访问控制机制。 Windows 中的访问控制也基于 ACL,但随着时间的推移变得更加复杂。在集中设置访问控制策略的情况下,ACL 适用于保护面向数据的环境;它们不太适合用户人数众多且不断变化的情况,或者用户希望能够将他们的权限委托给另一个用户在一段时间内运行特定程序的情况。 ACL 易于实现,但对于运行时的安全检查并不有效,因为典型的操作系统知道哪个用户正在运行特定程序,而不是自调用以来它已被授权访问哪些文件。操作系统必须在每次访问文件时检查 ACL,或者以其他方式跟踪活动访问权限。

最后,将访问规则分布到 ACL 中会使查找用户有权访问的所有文件变得乏味。验证没有文件是全局可读甚至是全局可写的可能涉及检查数百万用户文件的 ACL;对于大型复杂公司来说,这是一个真正的问题。尽管您可以编写脚本来检查服务器上的任何文件是否具有违反安全策略的 ACL,但您可能会被技术变化绊倒;由于管理员也忘记检查容器的 ACL,转向容器导致许多公司数据泄露。 (容器本身通常很糟糕,因为它是一项被数十家毫无头绪的初创公司出售的新技术。)并撤销员工的访问权限

刚刚被解雇的人通常必须通过取消他们的密码或身份验证令牌来完成。

让我们看一下 ACL 的一个重要示例 它们在 Unix（及其衍生版本 Android、MacOS 和 iOS)中的实现。

6.2.3 Unix操作系统安全

在传统的 Unix 系统中,文件不允许有任意的访问控制列表,而只是允许文件被读取、写入和执行的 rwx 属性。正常显示的访问控制列表有一个标志来显示文件是否是一个目录,然后标志 r,w 和 x 分别代表所有者,组和世界;然后它具有所有者的名称和组名称。设置了所有标志的目录将具有 ACL：

drwxrwxrwx 爱丽丝账户

在图 6.1 中的第一个示例中,文件 3 的 ACL 将是：

-rw-r----- 爱丽丝账户

这记录文件只是一个文件而不是一个目录;文件所有者可以读写它;小组成员（包括 Bob)可以读但不能写;非组成员根本没有访问权限;文件所有者是爱丽丝;该组是 Accounts。

在机器启动时获得控制权的程序（操作系统内核)作为管理员运行,并且可以不受限制地访问整个机器。所有其他程序都以用户身份运行,并由主管调解其访问权限。访问决策是根据与程序关联的用户标识做出的。但是,如果这是零（根）,则访问控制决定为“是”。所以 root 可以做它喜欢做的事 访问任何文件,成为任何用户,或其他任何事情。

更重要的是,有些事情只有 root 可以做,比如启动某些通信进程。在具有自主访问控制的系统中,root 用户标识通常可供系统管理员使用。

这意味着系统管理员可以做任何事情,所以我们很难将审计跟踪作为他们无法修改的文件来实现。在我们的例子中,山姆可以修改账户,如果他被错误地指控修改了账户,他将很难为自己辩护;更重要的是,一个成功成为管理员的黑客可以删除他入侵的所有证据。

保护日志免受 root 危害的传统且仍然是最常见的方法是将它们分开。在过去,这意味着将系统日志发送到上锁房间中的打印机;如今,这意味着将其发送到另一台机器,甚至是第三方服务。正如我们稍后讨论的那样,它也可能越来越多地涉及强制访问控制。

其次,ACL 只包含用户名,不包含程序名;所以没有直接的方法来实现（用户、程序、文件)的访问三元组。

相反,Unix 提供了一种间接方法:set-user-id (suid) 文件属性。
程序的所有者可以将代表该程序的文件标记为 suid,这使它能够以其所有者的特权而不是调用它的用户的特权运行。因此,为了实现上面第二个示例所需的功能,我们可以创建一个用户“account-package”来拥有

文件 2（帐户包），使文件 suid 并将其放在 Alice 可以访问的目录中。然后可以为这个特殊用户提供帐户程序所需的访问权限。

但是，当您采用具有三个维度（用户、程序、数据）的访问控制问题并使用二维机制实现它时，结果远不如三元组直观，而且人们容易犯错。程序员通常很懒惰或面临紧迫的期限；所以他们只是让应用程序 suid root，所以它可以做任何事情。这种做法会导致一些令人震惊的安全漏洞。做出访问控制决策的责任从操作系统环境转移到应用程序，大多数程序员没有足够的经验来检查他们应该检查的每一件事。 （很难知道要检查什么，因为调用 suid root 程序的人控制着它的环境，并可能以意想不到的方式操纵它。）

第三，ACL 不擅长表达可变状态。假设我们希望一项交易在执行之前得到经理和会计师的授权；我们可以在应用程序级别执行此操作（例如，通过等待第二个签名的事务队列）或通过使用 suid 做一些花哨的事情。管理状态访问规则很困难；它们会使刚刚被解雇的用户的撤销变得复杂，因为很难追踪他们打开的文件，而且 stu 可能会卡住。

第四，Unix ACL 只命名一个用户。如果一个资源将被多个资源使用，并且你想在操作系统级别进行访问控制，你有几个选择。对于旧系统，您必须使用组；较新的系统实现了扩展 ACL 的 Posix 系统，它可能包含任意数量的命名用户和命名组实体。理论上，ACL和suid机制往往可以达到预期的效果。在实践中，程序员常常太急于弄清楚如何做到这一点，而安全接口通常也太繁琐而无法使用。因此人们将他们的代码设计为要求比它严格应该拥有的更多的特权，因为这似乎是完成工作的唯一方法。

6.2.4 能力

管理访问控制矩阵的下一种方法是按行存储它。这些称为能力，在上面图 6.1 的示例中，Bob 的能力将如图 6.4 所示：

用户运营	账户会计审计系统程序		数据	踪迹
鲍勃	收据	r	r	r

图 6.4 – 能力

能力的优势和劣势与 ACL 大致相反。运行时安全检查更有效，我们可以毫不费力地委派一项权利：Bob 可以创建一个证书，上面写着 ‘这是我的能力，我特此委派给 David 从上午 9 点到下午 1 点读取文件 4 的权利，

签署鲍勃 。另一方面,更改文件的状态变得更加棘手,因为很难找出哪些用户可以访问。当我们必须调查事件或准备证据时,这可能会很烦人。事实上,可扩展系统最终会在内部使用事实上的功能,因为即时的系统范围内的撤销太昂贵了;在 Unix 中,文件描述符是真正的功能,即使在 ACL 权限甚至文件所有者更改后,文件描述符也会继续授予访问权限一段时间。在分布式 Unix 中,访问可能会在 Kerberos 票证的生命周期内持续存在。

那么我们可以完全取消 ACL 吗?人们在 1970 年代建造了实验机器,这些机器在整个 [2020] 期间都使用了各种功能;第一个商业产品是 Plessey System 250,一个电话开关控制器 [1575]。 IBM AS/400 系列系统于 1988 年为主流计算市场带来了基于能力的保护,并取得了一定的商业成功。密码学中使用的公钥证书是有效的功能,并从 20 世纪 90 年代中期开始成为主流。功能已经开始补充操作系统中的 ACL,包括更新版本的 Windows、FreeBSD 和 iOS,我将在后面描述。

在某些应用程序中,它们可能是表达安全策略的自然方式。
例如,医院可能有访问规则,如“护士应有权访问他或她病房的所有患者,或过去 90 天内去过那里的所有患者”。在基于传统 ACL 的早期系统中,每个访问控制决策都需要参考管理系统,以找出哪些护士和哪些患者在哪个病房,何时 - 但这使得 HR 系统和患者管理系统的安全性至关重要,这锤炼的可靠性。通过向护士提供带有证书的 ID 卡来解决问题,这些证书使他们有权访问与多个病房或医院部门相关的文件 [535、536]。如果您可以使系统中的信任关系反映出您正在尝试自动化的世界部分中的信任关系,那么您应该这样做。使用 grain 可以在堆栈的各个层面带来优势,使事情更有用,支持更安全的默认设置,减少错误,减少工程工作量并节省资金。

6.2.5 DAC 和 MAC

在过去,任何可以物理访问计算机的人都可以控制所有这一切:你可以加载任何你喜欢的软件,检查内存或磁盘上的所有内容,并更改任何你想更改的内容。这是自主访问控制 (DAC) 背后的模型:您以管理员模式启动计算机,然后作为管理员,您可以让权限较低的帐户可用于不太受信任的任务 例如运行由您不熟悉的公司编写的应用程序完全信任,或将远程登录权限授予他人。但这会使事情难以大规模管理,在 1970 年代,美国军方启动了一项庞大的计算机安全研究计划,其目标是保护机密信息:确保标记为“绝密”的文件永远不会被公开只有“秘密”许可的用户,而不管任何普通用户甚至主管的行为。在这样的多级安全 (MLS) 系统中,系统管理员不再是老板:最终控制权掌握在制定安全策略的远程政府机构手中。这些机制开始被描述为强制性的

访问控制（MAC）。主管,或者根访问权限,如果你愿意的话,是在远程控制之下的。这推动了强制访问控制技术的发展　这是一个引人入胜的故事,我在本书的第 2 部分中讲述了它。

从 20 世纪 80 年代开始,安全工程师也开始研究安全完整性等级的概念;粗略地说,一个更可靠的系统一定不能依赖一个不太可靠的系统。他们开始意识到他们需要类似于多级安全性的东西,但为了安全。军事系统的人也开始意识到保护机制本身的防篡改是至关重要的。在 1990 年代,随着计算机和网络的速度变得足以处理音频和视频,创意产业开始游说数字版权管理 (DRM),希望防止人们通过共享音乐和视频破坏他们的商业模式。这也是强制访问控制的一种形式　阻止订阅者与非订阅者共享歌曲在很多方面就像阻止绝密用户与秘密共享情报报告一样

用户。

在 2000 年代初期,随着许多操作系统供应商开始将 MAC 研究计划中的想法和机制整合到他们的产品中,这些想法汇集在一起。催化剂是微软和英特尔的一项举措,旨在将加密技术引入 PC 平台以支持 DRM。

英特尔认为 PC 的商业市场已经饱和,因此增长将来自家庭销售,他们认为家庭销售将需要 DRM。

微软从 DRM 开始,然后意识到为文档提供权限管理也可能是将客户紧紧锁定在 Windows 和 Oce 中的一种方式。他们建立了一个行业联盟,现在称为可信计算组,将密码学和 MAC 机制引入 PC 平台。

为此,操作系统必须具有防篡改功能,这是通过单独的处理器、可信平台模块 (TPM) 实现的,TPM 基本上是安装在 PC 主板上的智能卡芯片,用于支持可信引导和硬盘加密。 TPM 监控启动过程,在每个阶段都需要对目前已加载的所有内容进行散列,以检索解密下一阶段所需的密钥。系统的真正监督者现在不再是机器所有者,而是操作系统供应商。

MAC 基于 TPM 和可信启动,自 2006 年起在 Windows 6 (Vista) 中使用,作为对持久性恶意软件的防御1。 TPM 标准和架构被其他操作系统供应商和设备 OEM 所采用,现在甚至还有一个基于 Google 产品的开源 TPM 芯片 OpenTitan 项目。然而,这种设计的主要目的,无论其本身的设计是开放的还是封闭的,都是将硬件设备锁定为使用特定软件。

1微软有更雄心勃勃的计划;它的项目 Palladium 将为权限管理应用程序提供一个新的、更值得信赖的世界,以及为遗留软件提供的正常世界。他们在 2003 年推出了信息权限管理　文档的 DRM,但企业并不买账,将其视为锁定游戏。事实证明,两个世界的实现对于 Vista 来说太复杂了,经过两次单独的开发努力后,它被放弃了;但这一愿景从 2004 年开始一直存在于 Arm 的 TrustZone 中,我将在下面讨论。

6.2.6 苹果的 macOS

Apple 的 macOS 操作系统（以前称为 OS/X 或 Mac OS X)基于运行在 Mach 内核之上的 Unix 的 FreeBSD 版本。 BSD层提供内存保护;除非以高级权限运行,否则应用程序无法访问系统内存（或彼此的内存）。这意味着,例如,您可以使用“强制退出”命令终止楔入的应用程序,而无需重新启动系统。在这个 Unix 核心之上是许多图形组件,包括 OpenGL、Quartz、Quicktime 和 Carbon,而在表面上,Aqua 用户界面为用户提供了优雅和连贯的视图。

在文件系统层面,macOS 几乎就是一个标准的 Unix。默认安装禁用了 root 帐户,但是可以管理系统的用户在允许他们 su 到 root 的组“wheel”中。如果你是这样的用户,你可以安装程序（当你这样做时会要求你输入 root 密码）。自版本 10.5 (Leopard) 以来,它一直基于 TrustedBSD,这是一种包含强制访问控制机制的 BSD 变体,用于保护核心系统组件免受恶意软件篡改。

6.2.7 iOS

自 2008 年以来,Apple 凭借 iPhone 引领了智能手机革命,iPhone（以及 iPad 等其他设备)使用 iOS 操作系统 现在（到 2020 年)第二受欢迎。 iOS基于Unix； Apple 从 CMU 中获取 Mach 内核并将其与 FreeBSD 版本的 Unix 融合,在性能和稳健性方面进行了大量更改。例如,在 vanilla Unix 中,一个文件名可以有多个路径名,这些路径名导致一个 inode 代表一个文件对象,这是操作系统所看到的;在 iOS 中,这已得到简化,因此文件具有唯一的路径名,而这又是文件级访问控制的主题。同样,有一个 MAC 组件,其中来自域和类型执行 (DTE) 的机制用于防止篡改核心系统组件（我们将在第 9 章中更详细地讨论 DTE）。

Apple 推出此功能是因为他们担心应用程序会导致 iPhone 变砖,从而导致保修索赔。

应用程序也有权限,这是能力;他们请求访问设备服务的能力,例如移动网络、电话、短信、相机,以及应用程序首次尝试使用此类服务的时间。如果用户同意,则授予此权限2。许多设备服务开启了可能的侧信道攻击;例如,拒绝访问键盘的应用程序可以使用加速度计和陀螺仪推断按键。我们将在第 2 部分有关该主题的章节中讨论边信道。

苹果生态系统是封闭的,因为 iPhone 只能运行应用程序

2首次使用信任模型可以追溯到 1990 年代由 Symbian 推广的 Java 标准 J2ME,以及几乎同时出现的 Resurrecting Duckling 模型。 J2ME 还支持安装时信任等等。当 Apple 和 Android 出现时,他们最初做出了不同的选择。在每种情况下,拥有应用程序商店都是一项关键创新;诺基亚没有意识到这对推动双向市场的发展很重要。应用商店通过决定哪些应用可以运行来进行一些访问控制。苹果是硬实力,安卓是软实力;我们将在有关电话的章节中对此进行讨论。

Apple 已签署 3。这使公司能够提取应用程序收入的一部分,并筛选应用程序中的恶意软件或其他不良行为,例如利用侧通道来破坏访问控制。

iPhone 5S 引入了指纹生物识别和支付功能,为 A7 处理器添加了安全区域 (SE) 以提供单独保护。

Apple 决定既不信任 iOS 也不信任 TrustZone 处理此类敏感数据,因为在修补漏洞之前,漏洞会提供临时访问权限。它的工程师还担心在 ROM 中可能会发现无法修补的漏洞(Checkm8 最终发生了这种情况)。虽然 iOS 可以访问系统分区,但用户的个人数据是加密的,密钥由 SE 管理。密钥管理由一个独特的 256 位 AES 密钥引导,该密钥被烧录到片上系统上的熔断链路中。当设备启动时,用户有十次尝试输入密码的机会;只有这样,文件密钥才能从主密钥派生并可用 4。当设备被锁定时,一些键仍然可用,以便 iOS 可以计算出是谁发送了传入消息并通知您;这种便利的代价是取证设备可以获得对用户数据的一些访问权限。 SE 还管理升级并防止回滚。此类公开信息可以在 iOS 安全白皮书 [128] 中找到。

移动设备的安全性是一个相当复杂的问题,不仅涉及访问控制和防篡改,还涉及整个生态系统 从提供 SIM 卡到应用商店的运营,再到人们如何使用设备的文化,企业如何尝试操纵它们以及政府机构如何监视它们。我将在第 2 部分有关电话的章节中详细讨论这一点。

6.2.8 安卓

Android 是世界上使用最广泛的操作系统,根据谷歌的数据,2019 年 5 月有 25 亿台活跃的 Android 设备。 Android基于Linux;来自不同供应商的应用程序在不同的用户 ID 下运行。 Linux 机制在文件级别控制访问,防止一个应用程序读取另一个应用程序的数据并耗尽内存和 CPU 等共享资源。与在 iOS 中一样,应用程序具有权限,这实际上是功能:它们授予对 SMS、相机和地址簿等设备服务的访问权限。

应用程序以签名包的形式出现,作为 .apk 文件,虽然 iOS 应用程序由 Apple 签名,但 Android 的验证密钥以自签名证书的形式出现,并作为开发者的名称。这支持更新的完整性,同时维护开放的生态系统。每个包都包含一个需要一组权限的清单,用户必须批准“危险”的 大致上,那些可能花钱或泄露个人数据的。在 Android 的早期版本中,用户必须在安装时批准批次或不运行该应用程序。但经验表明,大多数用户只需点击任何内容即可完成安装过程,而且您会发现甚至手电筒应用程序也需要访问您的地址簿,因为他们可以将其出售换取金钱。所以Android 6搬家了

3 有一些例外:企业可以获得内部应用程序的签名密钥,但这些
如果滥用可能会被列入黑名单。

4我将在防篡改章节中讨论熔断链接,以及 iPhone PIN 重试失败
在关于监视和隐私的章节中。

对首次使用的 Apple 信任模型;为早期版本编译的应用程序仍然需要安装功能。

从Android 5开始,SELinux被用于通过强制访问控制加固操作系统,不仅可以保护核心系统功能免受攻击,还可以实现进程强隔离和日志违规。 SELinux 由 NSA 开发,用于支持政府系统中的 MAC;我们将在第 9 章进一步讨论它。其理念是行动需要三方同意:用户、开发人员和平台。

与 iOS（实际上是 Windows)一样,Android 的安全性是整个生态系统的问题,而不仅仅是访问控制机制的问题。新的电话生态系统与旧的 PC 生态系统截然不同,但继承了旧有线电话系统的足够多的特征,因此值得在第二部分的电话一章中单独讨论。我们将在有关侧信道和监视的章节中考虑其他方面。

6.2.9 视窗

当前版本的 Windows (Windows 10) 似乎是第三大最受欢迎的操作系统,在 2020 年 3 月的月活跃设备数量达到了 10 亿（直到 2016 年,Windows 一直处于领先地位）。Windows 有一个复杂得吓人的访问控制系统,快速浏览一下它的发展过程可能会更容易理解正在发生的事情。

早期版本的 Windows 没有访问控制。 Win dows 4 (NT) 出现了突破,它非常像 Unix,并受其启发,但有一些扩展。首先,不仅仅是读取、写入和执行,还有用于获取所有权、更改权限和删除的单独属性,以支持更灵活的委派。这些属性适用于组和用户 ,组权限允许您实现与 Unix 中的 suid 程序几乎相同的效果。属性不像在 Unix 中那样只是 on 或 o ,而是有多个值:您可以设置 AccessDenied、AccessAllowed 或 SystemAudit。这些按以下顺序解析:如果在相关用户或组的 ACL 中遇到 AccessDenied,则无论是否存在任何冲突的 AccessAllowed 标志,都不允许访问。更丰富的语法让您可以安排事项,这样日常配置任务（例如安装打印机)就不必需要完全的管理员权限。

其次,可以将用户和资源划分到具有不同管理员的域中,并且可以在域之间沿一个方向或两个方向继承信任。在典型的大公司中,您可能会将所有用户放在由 HR 管理的人员域中,而服务器和打印机等资产可能在部门控制下的资源域中;个别工作站甚至可以由其用户管理。可以安排一些事情,使部门资源域信任用户域,但反之则不行 因此被黑客攻击或粗心的部门管理员不能造成太大的外部损害。各个工作站反过来会信任该部门（但反之则不然）,以便用户可以执行需要本地权限的任务（例如安装软件包）。限制被黑客入侵的管理员可能造成的损害仍然需要仔细组织。用来管理这一切的数据结构,并隐藏

来自用户界面的 ACL 详细信息称为注册表。它的核心曾经是管理远程身份验证的 Active Directory 使用 Kerberos 变体或 TLS,封装在安全支持提供程序接口 (SSPI) 后面,使管理员能够插入其他身份验证服务。 Active Directory 本质上是一个数据库,它在分层命名空间的域中组织用户、组、机器和组织单位。

它隐藏在 Exchange 后面,但随着微软成为一家基于云的公司并将其用户转移到 Oce365,它现在正被逐步淘汰。

Windows 以两种方式添加了功能,可以覆盖或补充 ACL。首先,可以通过配置文件允许或拒绝用户或组访问。安全策略是由组而不是整个系统设置的;组策略覆盖个人配置文件,并且可以与站点、域或组织单位相关联,因此它可以开始解决复杂的问题。可以使用标准工具或自定义编码来创建策略。

功能进入 Windows 的第二种方式是,在许多应用程序中,人们使用 TLS 进行身份验证,而 TLS 证书在 Active Directory 的权限之外提供了另一个面向功能的访问控制层。

我已经提到 Windows Vista 引入了可信启动以使操作系统本身具有防篡改功能,因为它始终启动到已知状态,从而限制了恶意软件的持续存在。它添加了三个进一步的保护机制,以摆脱以前所有软件都以 root 身份运行的默认设置。首先,内核对开发人员关闭;其次,图形子系统和大部分驱动程序已从内核中删除;第三,用户帐户控制 (UAC) 将默认管理员权限替换为用户默认权限。以前,如此多的日常任务需要管理权限,以至于许多企业将所有用户都设置为管理员,这使得恶意软件难以遏制;许多开发人员编写他们的软件时都假设它可以访问所有内容（见[？]）。

根据微软工程师的说法,这是 Windows 缺乏健壮性的一个主要原因:应用程序以不兼容的方式使用系统资源。
因此他们添加了一个应用程序信息服务来启动需要提升权限的应用程序并使用虚拟化来包含它们:例如,如果他们修改注册表,他们不会修改“真实”注册表,而只是修改他们可以看到的版本.

自 Vista 以来,桌面充当后续用户进程的父进程,因此即使管理员以普通用户身份浏览网页,他们下载的恶意软件也无法覆盖系统文件,除非获得后续授权。当任务需要管理员权限时,用户会收到提升提示,要求他们输入管理员密码。 （Apple 的 macOS 类似,尽管幕后细节有所不同。）由于管理员用户经常被诱骗安装恶意软件,Vista 以文件完整性级别的形式添加了强制访问控制。

基本思想是,在缺少某些受信任的进程（例如通过微软在有问题的代码上）。

2012 年,Windows 8 添加了动态访问控制,让您可以控制

用户根据上下文访问,例如他们的工作 PC 与他们的家庭 PC 和他们的电话;这是通过 Active Directory 中的帐户属性完成的,这些属性显示为关于用户的声明,或者在 Kerberos 票证中显示为关于域的声明。在 2016 年,Windows 8.1 添加了一个更清晰的抽象,主体可以是在安全上下文中运行的用户、计算机、进程或线程,也可以是此类主体所属的组,以及代表此类主体的安全标识符 (SID)。当用户登录时,他们将获得带有他们所属 SID 的票证。 Windows 8.1 还通过添加 Microsoft 帐户（以前称为 LiveID)为迁移到云计算做好了准备,用户可以借此登录 Microsoft 云服务而不是本地服务器。在本地存储凭据的地方,它使用虚拟化来保护它们。最后,Windows 10 添加了许多功能来支持向具有多种客户端设备的云计算迁移,从证书固定（我们将在网络安全一章中讨论)到旧的安全注意序列 ctrl 的废除alt-del（这在触摸屏设备上很难做到,而且用户也不理解）。

总而言之,Windows 的发展提供了比以前在大众市场上销售的任何系统都更丰富、更灵活的访问控制工具集。它是由企业客户驱动的,这些客户需要管理数以万计的员工 ,他们在数百个不同的站点执行数百种不同的工作角色,提供内部控制以限制少数不诚实员工可能造成的损害 或受感染的机器。 （这些控件的实际设计方式将是我们在银行业务和簿记一章中的主题。)这一发展的驱动力是微软一半以上的收入来自获得超过 25,000 个席位许可的公司;但企业客户要求的灵活性的代价是复杂性。为大型 Windows 商店设置访问控制是一项高技能工作。

6.2.10 中间件

在计算的早期,在文件和程序级别进行访问控制很好,因为这些是重要的资源。自 1980 年代以来,规模和复杂性的增长导致访问控制在其他级别而不是（或同时)在操作系统级别完成。例如,簿记系统通常在 Oracle 等数据库产品之上运行,Oracle 将操作系统视为一个大文件。所以大部分的访问控制都得在数据库中完成;所有操作系统供应品都可以是每个登录用户的经过身份验证的 ID。而且从 1990 年代开始,客户端的很多工作都由 Web 浏览器完成。

6.2.10.1 数据库访问控制

在人们开始使用网站购物之前,数据库安全主要是幕后关注的问题。但企业现在拥有处理库存、调度和电子商务的关键数据库,前端是将交易直接传递给数据库的网络服务器。这些数据库现在包含了很多与我们生活息息相关的数据 银行账户、车辆登记和就业记录 有时出现故障会将它们暴露给随机的在线用户。

数据库产品,如 Oracle、DB2 和 MySQL,有自己的访问控制机制,这些机制以操作系统机制为模型,具有通常对用户和对象都可用的特权（因此这些机制是访问控制列表和功能的混合体） .然而,典型的数据库访问控制架构在复杂性上与 Windows 相当;现代数据库本质上是复杂的,它们所支持的事物也是如此 通常是业务流程涉及比文件或域更高级别的抽象。

可能存在旨在防止任何用户对太多客户了解太多的访问控制;这些往往是有状态的,并且可能处理可能的统计推断而不是简单的是-否访问规则。我在第 2 部分中用了整整一章来探讨推理控制这一主题。

易于管理通常是一个瓶颈。在我咨询过的公司中,操作系统和数据库访问控制由不同的部门管理,这些部门之间互不交谈;并且 IT 部门通常不得不进行粗略的黑客攻击,以使各种访问控制系统看起来像一个整体一样工作,但这会带来严重的漏洞。

一些产品允许开发人员绕过操作系统控制。例如,Oracle 既有操作系统帐户（其用户必须由平台在外部进行身份验证） ,也有数据库帐户（其用户直接由 Oracle 软件进行身份验证）。使用后者通常很方便,可以省去与其他部门正在做的事情同步的努力。在许多安装中,可以直接从外部访问数据库;甚至在它被 Web 服务前端屏蔽的地方,这通常包含让 SQL 代码插入数据库的漏洞。

因此,数据库安全故障会直接导致问题。 2003 年的 Slammer 蠕虫使用针对 Microsoft SQL Server 2000 的堆栈溢出漏洞进行自我传播,并在受感染的机器向随机 IP 地址发送大量攻击数据包时产生大量流量。

正如 Windows 很难安全地配置一样,因为它非常复杂,典型的数据库系统也是如此。如果你不得不锁定一个 - 或者甚至只是了解发生了什么 - 你最好阅读一本专业教科书,例如 [1174],或者请一位专家。

6.2.10.2 浏览器

Web 浏览器是我们依赖于访问控制的另一个中间件平台,它的复杂性常常让我们失望。主要的访问控制规则是同源策略,即网页上的 JavaScript 或其他活动内容只允许与其最初来自的 IP 地址进行通信;此类代码在沙箱中运行以防止它更改主机系统,正如我将在下一节中描述的那样。但是很多事情都可能出错。

在本书的前几版中,我们将 Web 安全视为服务器配置方式的问题,以及这是否会导致跨站点漏洞。例如,恶意网站可能包含旨在产生特定副作用 ect 的链接或表单按钮：

https://mybank.com/transfer.cgi?amount=10000USD&recipient=thief

这个想法是,如果用户点击登录到 mybank.com 的用户,则可能存在执行交易的风险,因为存在有效的会话 cookie。因此,支付网站部署了一些对策,例如使用短期会话和反 CSRF 令牌（会话 cookie 的不可见 MAC)以及检查 Referer: 标头。 Web 身份验证机制也存在问题;我在 4.7.4 节中简要描述了 OAuth。如果你以设计网页为生,你最好更详细地了解所有这些的机制（参见示例 [119]）;但许多开发人员并没有给予足够的关注。

例如,正如我在 2020 年所写的那样,Amazon Alexa 刚刚被证明在跨源资源共享方面配置错误,这意味着任何破坏另一个 Amazon 子域的人都可以用恶意技能替换目标 Alexa 上的技能 [1481] .

到现在为止,我们可能一直都应该将浏览器视为访问控制设备。毕竟,浏览器是您膝上型电脑上的地方,您运行的代码是由您不想信任的人编写的,而且他们偶尔会怀有恶意;正如我们之前讨论的那样,手机操作系统为不同的用户运行不同的应用程序,以提供更强大的保护。

即使没有恶意,如果浏览器因为其中一个选项卡中的脚本而挂起,您也不希望重新启动浏览器。 （Chrome 试图通过在单独的操作系统进程中运行每个选项卡来确保这一点。）

浏览器中的漏洞被路过式下载攻击所利用,访问攻击网页可能会感染您的机器,即使没有这个,现代网络环境也极难控制。许多网页充满了由多个广告网络和数据代理提供的跟踪器和其他不良内容,这嘲笑了同源政策背后的意图。

恶意行为者甚至可以使用 Web 服务来洗白来源:例如,攻击者将目标站点加上他自己的一些恶意脚本混搭,然后让受害者通过 Google Translate [1854] 等代理查看它].

谨慎的人会通过直接输入 URL 或使用书签来访问他们的银行网站;不幸的是,营销行业训练每个人点击电子邮件中的链接。

6.2.11 沙盒

1990 年代后期出现了另一种类型的访问控制:软件沙箱,由 Sun 以其 Java 编程语言引入。该模型是用户想要运行她作为小程序下载的一些代码,但担心小程序可能会做一些令人讨厌的事情,例如窃取她的地址簿并将其邮寄给营销公司,或者只是霸占CPU 和电池电量耗尽。

Java 的设计者通过提供一个“沙盒”来解决这个问题 一个受限的环境,在这个环境中,代码无法访问本地硬盘（或者至多只能临时访问一个受限目录）,并且只允许与它来自的主机（同源策略）。这是通过让解释器（Java 虚拟机（JVM））执行代码来强制执行的,只有有限的访问权限 [783]。这个想法适用于网页中使用的主要脚本语言 JavaScript,尽管它实际上是一种不同的语言;和其他活动内容。还使用了 Java 版本

在智能卡上,以便它们可以支持不同公司编写的小程序。

6.2.12 虚拟化

虚拟化是云计算的动力;它使一台机器能够独立地模拟多台机器,这样你就可以在数据中心租用一台虚拟机(VM),每月只需几十美元,而不必为整个服务器支付可能一百美元的费用。 IBM [496] 在 1960 年代发明了虚拟化;可以使用 VM/370 将单个机器划分为多个虚拟机。最初,这是关于使新的大型机能够运行来自多个旧机器架构的遗留应用程序;对于一家购买了两台计算机的公司来说,将一台用于生产环境,另一台作为一系列逻辑上独立的机器用于开发、测试和次要应用程序,这很快就成为一种常态。在主机操作系统之上运行虚拟机监视器 (VMM),然后在其之上运行其他操作系统是不够的;您必须处理显示处理器状态的敏感指令,例如绝对地址和处理器时钟。 2003 年出现了适用于带有 VMware ESX Server 的英特尔平台的工作 VMM,并且(尤其是)

2003 年的 Xen,它充分考虑了资源使用情况,足以促成 AWS 和云计算革命。借助处理器支持,事情可以做得更干净,英特尔自 2006 年以来通过 VT-x 提供了这种支持,我将在下面讨论其详细信息。 VM 安全声明在某种程度上基于这样的论点,即 VMM 管理程序的代码可以比操作系统小得多,因此更容易进行代码审查和保护;是否实际上存在更少的漏洞当然是一个经验问题 [1575]。

在客户端,虚拟化允许人们在主机之上运行来宾操作系统(例如,在 macOS 之上运行 Windows),这不仅提供了灵活性,还提供了更好的包容性前景。例如,一名员工可能在他们的笔记本电脑上运行两个 Windows 副本 一个用于 oce 环境的锁定版本,另一个在家里使用。三星提供 Knox,它在手机上创建一个虚拟机,雇主可以锁定并远程管理,而用户也可以在同一设备上享受普通的 Android。

但是使用虚拟化来分隔客户端的安全域比看起来要难。人们需要在多个虚拟机之间共享数据,如果他们使用临时机制,例如 U 盘和网络邮件帐户,这会破坏分离。安全的数据共享绝非易事。例如,Bromium5提供为企业 PC 上的特定应用程序量身定制的 VM,因此您有一个用于 Oce 的 VM,一个用于 Acrobat Reader,一个用于您的浏览器等等。这使公司能够合理安全地使用旧的、不受支持的软件。那么如何下载一个Oce文档呢?好吧,浏览器将文件从其 VM 导出到主机硬盘,将其标记为“不受信任”,因此当用户尝试打开它时,他们将获得一个新的 VM,其中包含该文档以及 Oce,仅此而已。

当他们随后通过电子邮件发送这个不受信任的文档时,有一个 Outlook 插件可以阻止它在“已发送邮件”窗格中呈现。将网络服务集成到应用程序中,事情变得更加混乱;关于哪些站点可以访问哪些 cookie 的规则很复杂,并且很难处理单点登录和工作流程

5现在归惠普所有

跨多个域。剪贴板也需要更多的规则来控制它。许多规则会不时更改,并且是启发式的,而不是硬性的、可验证的访问逻辑。简而言之,如果要对用户透明,则在客户端使用 VM 进行分离需要与操作系统和应用程序进行深度集成,并且在安全性和可用性之间需要进行大量权衡。

实际上,您是在将虚拟化改造到不是为其构建的现有操作系统和应用程序。

容器一直是 2010 年代后期的热门新话题。它们发展成为云计算中虚拟化的轻量级替代方案,并且经常与它混淆,尤其是营销人员。我的定义是,虽然 VM 有一个完整的操作系统,通过管理程序与硬件隔离,但容器是一个与其他容器共享内核的隔离的来宾进程。容器实现通过虚拟化操作系统机制的一个子集来分离进程组,包括进程标识符、进程间通信和名称空间;他们还使用沙盒和系统调用过滤等技术。商业动机是尽量减少客人的规模、他们的交互复杂性和管理他们的成本,所以他们与编排工具一起部署。与任何其他新技术一样,许多初创公司的热情多于经验。 Jerry Gamblin 于 2019 年进行的一项调查显示,在 Docker Hub 上可供开发人员使用的前 1000 个容器中,有 194 个设置了空白根密码 [743]。如果你打算使用云系统,你需要认真注意你的工具选择,还要学习另一套访问控制机制 那些由服务提供商提供的 ,比如亚马逊 AWS Identity and Access管理（IAM）。这增加了另一层复杂性,人们可能会弄错。例如,2019 年,一家为银行和警方提供生物特征识别服务的安全公司使其整个数据库都没有受到保护;两名研究人员使用 Elasticsearch 发现了它,并在数据库中发现了数百万人的照片、指纹、密码和安全许可级别,他们不仅可以读取,还可以写入 [1864]。

但是,即使你正确地绑定了一个云系统,分离机制所能达到的效果也存在硬件限制。 2018 年,发布了两类强大的侧信道攻击:Meltdown 和 Spectre,我将在下一节中进行讨论,并在有关侧信道的章节中进行更详细的讨论。那些使用容器来部署支付处理的银行至少隐含地依赖于他们的容器很难在亚马逊或谷歌这样规模的云中定位。有关虚拟化和容器发展的全面调查,请参阅 Randal [1575]。

6.3 硬件保护

大多数访问控制系统不仅着手控制用户可以做什么,而且还限制程序可以做什么。在很多系统中,用户既可以编写程序,也可以下载安装程序,而这些程序可能存在bug甚至是恶意的。

防止一个进程干扰另一个进程是保护问题。限制问题是阻止程序通信的问题

非通过授权渠道向外传播。每种都有好几种口味。目标可能是防止主动干扰,例如内存覆盖写入,或阻止一个进程直接读取另一个进程的内存。这就是商业操作系统打算做的事情。军事系统也可能会尝试保护元数据关于其他数据、主题或进程的数据 这样一来,例如,用户无法发现其他用户登录到系统的内容或他们正在运行的进程。

除非使用沙盒技术（这对一般编程环境来说限制太多）,否则解决单个处理器上的保护问题至少意味着要有一种机制来阻止一个程序覆盖另一个程序的代码或数据。可能存在共享的内存区域以允许进程间通信;但是必须保护程序免受意外或故意修改,并且必须能够访问受到类似保护的内存。

这通常意味着硬件访问控制必须与处理器的内存管理功能集成在一起。一个经典的机制是段寻址。内存由两个寄存器寻址,一个段寄存器指向一段内存,一个地址寄存器指向该段内的一个位置。段寄存器由操作系统控制,通常由称为引用监视器的操作系统组件控制,该组件将访问控制机制与硬件联系起来。

处理器本身的实现变得更加复杂。早期的 IBM 大型机有一个双态 CPU:机器要么处于授权状态,要么不处于授权状态。在后一种情况下,程序被限制在操作系统分配的内存段中;在前者中,它可以随意写入段寄存器。授权程序是从授权库加载的程序。

给定合适的授权库,可以在此之上实施任何所需的访问控制策略,但这并不总是有效的;系统安全取决于将不良代码（无论是恶意的还是错误的)排除在授权库之外。所以后来的处理器提供了更复杂的硬件机制。

Multics 是 1960 年代在麻省理工学院开发的一种操作系统,它启发了 Unix,它引入了表示不同级别特权的保护环:环 0 程序可以完全访问磁盘,主管状态在环 2 中运行,用户代码处于不同的级别特权级别较低[1684]。它的许多功能已被更新的处理器采用。

接口硬件和软件安全机制存在许多普遍问题。例如,经常会发生诸如应用程序代码之类特权较低的进程需要调用特权较高的进程（例如设备驱动程序)的情况。执行此操作的机制需要谨慎设计,否则可能会出现安全漏洞。此外,性能可能非常依赖于不同特权级别的例程是通过引用还是通过值调用 [1684]。

6.3.1 英特尔处理器

早期PC使用的Intel 8088/8086处理器没有系统模式和用户模式之分,任何运行的程序都控制着整机[6]。 80286 添加了受保护的段寻址和环,因此 PC 第一次可以运行正确的操作系统。 80386 具有内置虚拟内存和足够大的内存段 (4 Gb),可以忽略它们并将机器视为 32 位平面地址机器。 486 和 Pentium 系列芯片增加了更多性能（缓存、乱序执行和附加指令,如 MMX）。

保护环由多种机制支持。当前特权级别只能由环 0（内核)中的进程更改。过程不能直接访问较低级别环中的对象,但有一些门允许在不同的特权级别执行代码并管理支持基础设施,例如多个堆栈段。

从 2006 年开始,英特尔增加了对 x86 虚拟化的硬件支持,称为英特尔 VT,这有助于推动云计算的采用。一些处理器架构如 S/370 和 PowerPC 很容易虚拟化,Gerald Popek 和 Robert Goldberg [1532] 于 1974 年建立了这方面的理论要求;它们包括所有暴露原始处理器状态的敏感指令都是特权指令。然而,原生 Intel 指令集具有敏感的用户模式指令,需要杂乱的解决方法,例如重写应用程序代码和为托管操作系统打补丁。在硬件中添加 VMM 支持意味着您可以按照设计在 ring 0 中运行操作系统； VMM 在下面有自己的内存架构副本。您仍然需要捕获敏感的操作码,但系统调用不会自动要求 VMM 干预,您可以运行未修改的操作系统,运行速度更快,系统通常更健壮。现代 Intel CPU 现在有九个环:环 0-3 用于普通代码,在环 0-3 下面是一组用于虚拟机管理程序的更进一步的环 0-3 VMM 根模式,底部是用于 BIOS 的系统管理模式 (SMM)。实际上,使用的四个级别是 SMM,VMX root 模式的 ring 0,操作系统的正常 ring 0,以及高于应用程序的 ring 3。

2015 年,英特尔发布了 Software Guard eXtensions (SGX),它让受信任的代码在飞地（内存的加密部分)中运行,而其余代码照常执行。该公司在 Trusted Computing 计划的早期就致力于此类架构,但直到它需要一个 enclave 架构来与 TrustZone 竞争时才顺其自然,我将在下一节中讨论。加密由内存加密引擎 (MEE) 执行,而 SGX 还引入了新指令和内存访问检查,以确保非 enclave 进程无法访问 enclave 内存（甚至根进程也不行）。 SGX 已针对 DRM 和保护云 VM 进行推广,特别是那些包含加密密钥、凭证或敏感个人信息的;这是受到 Spectre 和类似攻击的威胁,我在边信道一章中详细讨论了这一点。由于 SGX 的安全边界是 CPU,其软件在主内存中加密,这会带来真正的惩罚

[6]它们是根据一项速成计划开发的,目的是在出现后挽救市场份额 RISC 处理器和 iAPX432 的市场失败。

在时间和空间上。过去的另一个缺点是 SGX 代码必须由英特尔签名。公司现在已经委托签名（这样坏人就可以得到代码签名），并且从 SGXv2 开始，将向其他人开放信任根。因此，人们正在试验 SGX 恶意软件，这种恶意软件仍然无法被防病毒软件检测到。由于 SGX 应用程序无法发出系统调用，人们曾希望 enclave 恶意软件不会造成太大伤害，但 Michael Schwarz、Samuel Weiser 和 Daniel Gruss 现在已经研究出如何从 enclave 上发起隐秘的面向返回编程 (ROP) 攻击主机应用程序；他们争辩说，问题在于飞地应该做什么缺乏明确性，任何合理的威胁模型都必须包括不受信任的飞地 [1688]。这个简单的观点可能会迫使人们重新思考 enclave 架构；英特尔表示，“未来，英特尔的控制流执行技术 (CET) 应该有助于解决 SGX 7 内部的这一威胁。至于接下来会发生什么，AMD 在 2016 年发布了完整的系统内存加密，而英特尔则宣布了竞争对手。这旨在应对冷启动和 DMA 攻击，并保护代码免受不受信任的管理程序的侵害；它还可能会提升下一代飞地的空间和性能限制。然而，Jan Werner 及其同事发现，在虚拟环境中使用 AMD 产品时，会受到多种推理和数据注入攻击。 [2010].显然还有一些路要走。

除了访问控制漏洞，还有加密问题，这
我将在高级密码工程一章中讨论。

6.3.2 ARM 处理器

Arm 是手机、平板电脑和物联网设备中最常用的处理器内核；仅手机就使用了数十亿，高端设备的芯片组中有几十个不同大小的 Arm 内核。最初的 Arm（代表 Acorn Risc Machine）是第一个商业 RISC 设计；它于 1985 年发布，就在 MIPS 之前。 1991 年，Arm 成为一家独立的公司，与英特尔不同，它不拥有或经营任何晶圆厂：它授权一系列处理器内核，芯片设计师将这些内核包含在他们的产品中。早期的核心有一个 32 位数据路径，包含 15 个寄存器，其中 7 个被系统进程的银行寄存器覆盖，以减少中断时切换上下文的成本。有多种管理模式，处理快速和正常中断，复位时进入的系统模式，以及各种异常处理。内核最初不包含内存管理，因此基于 Arm 的设计可以对其硬件保护进行广泛定制；现在有带有内存保护单元 (MPU) 的变体，还有其他带有处理虚拟内存的内存管理单元 (MMU) 的变体。

2011年，Arm推出版本8，支持64位处理，可以虚拟化多个32位操作系统。管理程序支持添加了另一种管理模式。这些内核有各种尺寸，从具有超过十几级深度流水线的大型 64 位超标量处理器，到用于廉价嵌入式设备的微型处理器。

TrustZone 是支持“两个世界”模型的安全扩展

7 2019 年针对 ROP 攻击的最佳防御措施似乎是 Apple 在 iPhone X3 及更高版本中使用保存在寄存器中的密钥对指针进行签名的机制；这会阻止 ROP 攻击，因为攻击者无法猜测签名。

上面提到;它于 2004 年提供给手机制造商 [44]。手机是 enclaves 的“杀手级应用”,因为运营商想要锁定补贴手机,而监管机构想要使控制 RF 功能的基带软件具有防篡改功能 [1239]。 TrustZone 支持普通操作系统和通用应用程序的开放世界,以及用于处理敏感操作的封闭飞地,例如加密和关键 I/O(在手机中,这可能包括 SIM 卡和指纹读取器)。处理器是处于安全状态还是非安全状态与其处于用户模式还是管理模式是正交的(尽管它必须在安全模式和管理程序模式之间进行选择)。封闭的世界拥有一个单一的可信执行环境 (TEE),具有独立的堆栈、一个简化的操作系统,并且通常只运行由 OEM 签名的可信代码 尽管三星的 Knox 旨在提供“家庭”和“工作”环境您的手机,允许常规丰富的应用程序在安全环境中执行。

尽管 TrustZone 于 2004 年发布,但一直关闭到 2015 年;原始设备制造商使用它来保护自己的利益,并且不向应用程序开发人员开放它,除非偶尔在 NDA 下。与英特尔 SGX 一样,似乎还没有办法处理恶意 enclave 应用程序,这些应用程序可能作为 DRM 与游戏应用程序捆绑在一起,或者由专制国家强制执行;而且,与英特尔 SGX 一样,使用 TrustZone 创建的 enclave 应用程序可能会引发透明度和控制问题,这可能会蔓延到可审计性、隐私等方面。再次,公司内部人士咕哝着“等等看”;毫无疑问,我们会的。

Arm 的最新产品是 CHERI8 ,它为 Arm CPU 添加了细粒度的功能支持。目前,Chrome 等浏览器将标签放在不同的进程中,这样即使一个网页的脚本运行缓慢,也不会拖慢其他标签的速度。如果每个网页中的每个对象都可以单独进行沙盒处理,那就太好了,但这是不可能的,因为每个进程间上下文切换的 CPU 周期成本很高。 CHERI 使生成子线程的进程能够为其分配对特定内存范围的读写访问权限,以便多个沙箱可以在同一进程中运行。这是在 2018 年作为产品发布的,我们预计将在 2021 年看到第一块硅。这项技术的长期承诺是,如果它在 Windows、Android 和 iOS 等操作系统中得到充分使用,它本来可以防止近年来的大多数零日攻击。大规模采用新的保护技术需要花费真金白银,就像从 32 位 CPU 切换到 64 位 CPU 一样,但它可以节省大量补丁的成本。

6.4 出了什么问题

Android、Linux 和 Windows 等流行的操作系统非常庞大和复杂,每天都有数十亿用户在非常不同的情况下测试它们的功能。发现了许多错误,其中一些错误会产生具有典型生命周期的漏洞能力。发现错误后,将错误报告给 CERT 或供应商;补丁已发货;补丁被逆向工程,可能会产生漏洞;以及没有及时打补丁的人

8完全披露:这是由我在剑桥和其他地方的同事团队开发的由罗伯特·沃森 (Robert Watson) 领导。

可能会发现他们的机器已受到威胁。在少数情况下,漏洞会立即被利用而不是被报告 称为零日利用,因为攻击从漏洞已知存在的第零天开始发生。

漏洞生命周期的经济学和生态学是安全经济学家深入研究的主题;我将在第三部分讨论这个问题。

攻击者的传统目标是在系统上获得一个普通帐户,然后成为系统管理员,这样他们就可以完全接管系统。第一步可能涉及猜测或社会工程密码,然后使用操作系统错误从用户升级到根 [1129]。

用户/root 的区别在 21 世纪变得不那么重要了,原因有二。首先,Windows PC 是最常见的在线设备(直到 2017 年 Android 超越它们),因此它们是最常见的攻击目标;由于他们以管理员身份运行许多应用程序,因此任何可能受到威胁的应用程序都会授予管理员访问权限。其次,攻击者有两种基本类型:有针对性的攻击者,他们想要监视特定的个人,其目标通常是获得对该人帐户的访问权限;规模化攻击者,他们的目标通常是破坏大量 PC,他们可以将这些 PC 组织成僵尸网络以赚钱。这也不需要管理员访问权限。即使您的邮件客户端不以管理员身份运行,它对于控制的垃圾邮件发送者仍然有用。

然而,僵尸网络牧民确实更喜欢安装 rootkit,顾名思义,rootkit 以 root 身份运行;它们也称为远程访问木马或 RAT。
用户/root 的区别在商业环境中仍然很重要,在这种情况下,您不希望敌对情报机构、企业间谍公司或犯罪团伙将此类工具包安装为高级持续威胁大骗局

一个单独的区别是漏洞利用是否可蠕虫 - 它是否可以用于在没有人为干预的情况下将恶意软件从一台机器快速在线传播到另一台机器。 Morris 蠕虫是第一个大规模案例,此后出现了很多。我在第 2 章中提到了 Wannacry 和 NotPetya;这些利用了美国国家安全局开发的漏洞,然后泄露给其他国家行为者。操作系统供应商对蠕虫攻击反应迅速,通常会发布乱序补丁,因为它们可能造成的破坏规模很大。在撰写本文时,最麻烦的蠕虫攻击是 Mirai 的变体,这是一种蠕虫,用于接管使用已知根密码的物联网设备。

它出现于 2016 年 10 月,目的是利用闭路电视摄像机,此后已经产生了数百个版本,用于接管不同的易受攻击的设备并将它们招募到僵尸网络中。可蠕虫攻击通常使用 root 访问权限,但不是必须的;该漏洞能够自动向前传输就足够了9。

无论如何,技术攻击的基本类型在一代人中并没有发生太大变化,现在我将简要介绍一下。

9 在极少数情况下,即使是人为传播也能使恶意软件迅速传播:ILoveYou 蠕虫就是一个例子,它在 2000 年通过带有该主题行的电子邮件进行自我传播,导致足够多的人打开它,运行一个脚本将其发送给所有人在新受害者的地址簿中。

6.4.1 粉碎堆栈

经典的软件利用是内存覆盖攻击,俗称“粉碎堆栈”,如 1988 年的 Morris 蠕虫所使用的;这感染了如此多的 Unix 机器,以至于它扰乱了互联网,并使恶意软件力量完全引起了大众媒体的注意 [1806]。在 1990 年代末和 2000 年代初 [487],涉及违反内存安全的攻击占操作系统攻击的一半以上,但此后这一比例一直在缓慢下降。

程序员通常不注意检查参数的大小,因此向程序传递长参数的攻击者可能会发现其中一些参数被视为代码而不是数据。 Morris 蠕虫中使用的经典示例是 Unix finger 命令中的一个漏洞。一个常见的实现是接受任何长度的参数,尽管程序只为这个参数分配了 256 个字节。当攻击者使用带有较长参数的命令时,参数的尾部字节最终会覆盖堆栈并由系统执行。

通常的利用技术是为参数的尾部字节安排一个着陆点 一长段无操作 (NOP) 命令,或其他不改变控制流的寄存器命令,其任务是如果处理器执行了其中任何一个,则捕获处理器。着陆台将处理器交付给攻击代码,攻击代码将执行一些操作,例如直接创建具有管理权限的 shell（参见图 6.5）。

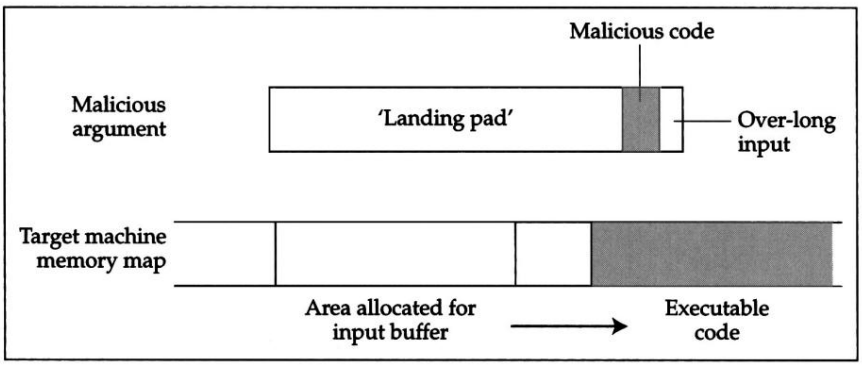

图 6.5：- 堆栈粉碎攻击

堆栈覆盖攻击早在 1988 年之前就存在了。大多数 1960 年代早期的分时系统都受到这个漏洞的影响,并修复了它 [804]。
70 年代初期的渗透测试表明,最常用的攻击策略之一仍然是“意外参数”[1165]。 Intel 的 80286 处理器在 1982 年引入了显式参数检查指令 验证读取、验证写入和验证长度,但大多数软件设计人员都避免使用它们以防止体系结构依赖性。已经发现针对各种可编程设备的堆栈覆盖攻击 - 甚至针对智能卡和硬件安全模块之类的东西,其设计者真的应该更了解这些东西。

6.4.2 其他技术攻击

许多漏洞是同一主题的变体,因为它们发生在语法 A 中的数据被解释为语法 B 中的代码时。堆栈溢出是当数据被接受为输入（例如 URL)并最终作为机器执行时代码。这些是类型安全的失败。事实上,堆栈溢出可以被视为内存安全故障或无法清理用户输入,但每种类型都有更纯粹的示例。

use after free 类型的安全失效是现在远程执行漏洞最常见的原因,近年来提供了大量针对浏览器的攻击。当一块内存被释放然后仍在使用时,可能会发生这种情况,这可能是因为混淆了程序的哪一部分负责释放它。如果现在分配了一个恶意块,它最终可能会在堆上占据一席之地,而当一个旧的无害函数被调用为一个新的时,恶意函数可能会被调用。内存安全主题还有许多其他变体;缓冲区溢出可能由不正确的字符串终止、将大小不合适的缓冲区传递给路径操作函数以及许多其他细微错误引起。有关分类法,请参阅 Gary McGraw 的书“软件安全性 [1266]”。

SQL 注入攻击是最常见的基于无法清理输入的攻击,当粗心的 Web 开发人员将用户输入传递到后端数据库而不检查它是否包含 SQL 代码时,就会出现这种攻击。游戏通常会因错误消息而泄露,有能力且有动机的用户可能会从中推断出足够的信息来发起攻击。 Web 开发人员使用的其他语言（例如 PHP)也存在类似的命令注入问题。通常的补救措施是将所有用户输入视为可疑并对其进行验证。但这可能比看起来更难,因为很难预测所有可能的攻击,并且为一个 shell 编写的过滤器可能无法意识到另一个 shell 中存在的扩展。在可能的情况下,应该通过设计此类攻击来仅在安全的上下文中对用户输入采取行动;如果有必要将特定漏洞列入黑名单,则需要适当维护该机制。

一旦处理了此类类型安全和输入清理攻击,下一个可能就是竞态条件。这些发生在交易分两个或多个阶段进行时,其中在第一阶段验证访问权限,在第二阶段完成一些敏感的事情。如果有人可以改变两个阶段之间的状态,这可能会导致攻击。一个典型的例子出现在早期的 Unix 版本中,创建目录的命令“mkdir”过去分两步工作:分配存储空间,然后将所有权转移给用户。由于这些步骤是分开的,用户可以在后台启动“mkdir”,如果这在暂停之前只完成了第一步,则可以使用第二个过程用密码文件的链接替换新创建的目录。然后原始进程将恢复,并将密码文件的所有权更改为用户。

一个更现代的例子是容器中使用的包装器来拦截应用程序对操作系统的系统调用,解析它们,并在需要时修改它们。这些包装器在内核的地址空间中执行,检查所有系统调用的进入和退出状态,并且仅封装安全逻辑。他们通常假设系统调用是原子的,但是现代的

操作系统内核是高度并发的。系统调用彼此之间不是原子的;两个系统调用可能会相互竞争以访问共享内存,这会导致检查时间到使用时间 (TOCTTOU) 攻击。一个早期（2007 年)的示例调用了一个路径,该路径的名称超出页面边界一个字节,导致内核在获取页面时进入休眠状态;然后它会替换内存中的路径 [1992]。从那以后还有其他人,随着时间的推移,每个 CPU 芯片中的处理器越来越多,容器成为部署应用程序的越来越普遍的方式,这种攻击可能会成为越来越大的问题。一些操作系统具有专门处理并发攻击的功能,但该领域仍在不断变化。

一种不同类型的定时攻击可能来自备份和恢复系统。如果您可以让用户恢复他们自己的文件,而不是必须调用系统管理员,那将很方便 但是您如何保护信息资产免受时间旅行者的侵害?人们可以重新获得被撤销的访问权限,并玩出更微妙的把戏。

最近吸引了大量研究工作的一种攻击是面向返回的编程（ROP)[1708]。许多现代系统试图通过数据执行预防来防止类型安全攻击 将内存标记为代码或数据,这种措施可以追溯到 Burroughs 5000;如果所有代码都经过签名,您肯定会认为无法执行未经授权的代码吗?错误的!

攻击者可以寻找小工具 具有某些有用效果的指令序列,以返回结束。通过收集足够的小工具,有可能组装一台图灵强大的机器,并将我们的攻击代码实现为 ROP 小工具链。然后所要做的就是夺取调用堆栈的控制权。这是从 return-to-libc 攻击演变而来的,它使用公共共享库 libc 来提供易于理解的小工具;此后开发了许多变体,包括使 SGX 飞地中的恶意软件能够对主机应用程序发起隐蔽攻击的攻击 [1688]。最新的攻击变体,面向块的编程（BOP),通常可以从程序模糊测试发现的崩溃中自动生成攻击,从而破坏当前的控制流完整性控制 [964]。这种攻击和防御的共同进化无疑将继续下去。

最后还有侧通道。攻击技术的最新重大创新针对 CPU 流水线行为。 2018 年初,两种改变游戏规则的攻击开创了这一类型:Meltdown,它利用英特尔处理器上的乱序执行创建的侧通道 [1172],以及 Spectre,它利用英特尔、AMD 和 Arm 处理器上的推测执行 [1068]。基本思想是大型现代 CPU 的流水线又长又复杂,以至于它们会向前看并预测接下来的十几个指令,即使这些指令是不允许当前进程执行的（假设访问检查是两条指令)将来它会禁止的读取操作是在那之后的两条指令）。未采用的路径仍然可以将信息加载到缓存中,从而以延迟的形式泄漏信息。通过一些技巧,一个进程可以安排一些事情来读取另一个进程的内存。我将在本书第二部分的边信道章节中更详细地讨论 Spectre 和 Meltdown。尽管缓解措施已经发布,但同类攻击仍在继续被发现,可能需要数年时间和新一代处理器才能完全控制它们。

这一切让我想起了本章开头 Roger Needham 的话。
优化包括用几乎可以用但更便宜的东西替换可以用的东西;现代 CPU 的优化程度如此之高,以至于我们一定会看到更多关于 Spectre 主题的变体。此类攻击不仅限制了容器和虚拟机可以提供的保护,还限制了 TrustZone 和 SGX 等飞地机制提供的保护。特别是,它们可能会阻止谨慎的公司将高价值的加密密钥委托给飞地,并延长老式硬件加密的使用寿命。

6.4.3 用户界面故障

攻击堡垒的一种常见方法是诱使守卫帮助你,操作系统也不例外。最早的攻击之一是特洛伊木马,这是一个邀请管理员运行的程序,但其中包含一个令人讨厌的惊喜。人们会编写游戏来检查玩家是否是系统管理员,如果是,则会创建另一个具有已知密码的管理员帐户。一种变体是编写一个与通用系统实用程序同名的程序,例如列出 Unix 目录中所有文件的 ls 命令,并将其设计为在调用真正的实用程序之前滥用管理员权限(如果有的话)。然后您向管理员抱怨该目录有问题。当他们进入目录并键入 ls 查看那里有什么时,损坏就完成了。这是混淆代理问题的一个示例:如果 A 代表 B 执行某些任务,并且其权限来自 A 和 B,并且 A 的权限超过 B,则事情可能会出错。这种特殊情况下的修复很简单:管理员的“PATH”变量(调用命令时要搜索的目录列表,以查找适当命名的程序)不应包含“.”。 (当前目录的符号)。现代 Unix 版本默认带有这个。但这仍然是一个示例,说明您必须如何正确获取大量小细节才能使访问控制变得健壮,而这些细节并不总是事先很明显。

就历史上受到攻击的系统数量而言,用户界面故障最严重的例子可能包括两个事实:首先,Windows 总是弹出确认对话框,这会训练人们点击框以完成工作;其次,直到 2006 年,用户需要成为管理员才能安装任何东西。这个想法是,将软件安装限制为管理员可以让微软的大企业客户(例如银行和政府部门)锁定他们的系统,这样员工就无法运行游戏或其他未经授权的软件。但在大多数环境下,普通人需要安装软件才能完成工作。因此,数以亿计的人拥有本不应需要的管理员权限,并在网站弹出一个框告诉他们做某事时安装恶意代码。许多应用程序开发人员坚持让他们的代码以 root 身份运行,这使情况变得更加复杂,他们要么是出于懒惰,要么是因为他们想要收集他们不应该拥有的数据。 Windows Vista 开始摆脱这一点,但恶意软件生态系统现已在 PC 世界中建立起来,并且开始在 Android 生态系统中扎根,因为企业迫使人们安装应用程序而不是使用网站,并且应用程序需要访问权限他们实际上不应该拥有的各种数据和服务。出色地

稍后在有关电话的章节中讨论这个问题。

6.4.4 补救措施

软件安全并非一帆风顺。在 2000 年代,情况明显好转。在世纪之交,90% 的漏洞都是缓冲区溢出;到 2008 年本书第二版问世时,还不到一半,现在更少了。几件事造成了不同 erence。

1. 第一个由特定防御组成。 Stack canary 是编译器在堆栈返回地址旁边插入的随机数。

   如果堆栈被覆盖,那么金丝雀很可能会发生变化 [487]。数据执行保护(DEP)将所有内存标记为数据或代码,并阻止前者被执行;它出现在 2003 年的 Windows XP 中。地址空间布局随机化(ASLR)同时到来;通过在系统的每个实例中使内存布局不同,攻击者更难预测目标地址。这一点特别重要,因为现在有工具包可以绕过 DEP 进行 ROP 攻击。控制流完整性机制包括在编译时分析可能的控制流图,并在运行时通过验证间接控制流传输来强制执行;这出现在 2005 年,并在接下来的十年中被纳入各种产品中 [348]。然而,分析并不精确,面向块的编程攻击是为利用漏洞而发展起来的技巧之一 [964]。

2. 第二个包含更好的通用工具。 Coverity 等静态分析程序可以发现大量潜在的软件错误,并突出显示代码偏离最佳实践的方式;如果从项目一开始就使用它们,它们会产生很大的不同 erence。 (如果稍后添加,它们可能会抛出数以千计的警报,处理起来很痛苦。)根本的解决方案是使用更好的语言;我的同事越来越多地使用 Rust 而不是 C 或C++10 编写系统代码。

3、三是加强培训。 2002 年,Microsoft 宣布了一项安全计划,要求每位程序员都接受如何编写安全代码的培训。 (他们为此制作的书《编写安全代码》[927] 仍然值得一读。)其他公司纷纷效仿。

4. 最新的方法是 DevSecOps,我将在第 3 部分讨论它。扩展敏捷开发方法以允许非常快速地部署补丁和响应事件;它可以使投入设计、编码和测试的努力能够针对最紧迫的问题。

建筑很重要;拥有以受控方式发展的干净界面,在对产品安全有长期利益的经验丰富的人的密切关注下,可以产生巨大的差异 。程序应该只有

10Rust 于 2010 年从 Mozilla 研究中脱颖而出,并已被用于重新开发 Firefox;它是在 2016 年至 2019 年的 Stack Overflow 年度调查中被评为最受欢迎的语言。

他们需要多少特权:最小特权原则[1639]。软件的设计也应该确保默认配置和通常最简单的做某事的方式应该是安全的。健全的架构对于实现安全默认设置和使用最小权限至关重要。然而,许多系统都带有危险的默认值和混乱的代码,使各种接口暴露于 SQL 注入等本不应该发生的攻击。这些包括个人和企业激励措施的失败,以及教育不足和安全工具的可用性差。

6.4.5 环境蠕变

当环境变化破坏安全模型时,会导致许多安全故障。在初始环境中充分发挥作用的机制通常在更广泛的环境中会失败。

访问控制机制也不例外。例如,Unix 最初被设计为“单用户 Multics”(因此得名)。然后,它成为一个操作系统,供实验室中共享一台机器的许多技术娴熟且值得信赖的人员使用。在这种环境中,安全机制的功能主要是包含错误;以防止一个用户的输入错误或程序崩溃删除或覆盖另一个用户的文件。最初的安全机制足以满足此目的。

但 Unix 安全成为典型的“成功灾难”。自 Ken Thomson 于 1969 年在贝尔实验室开始研究 Unix 以来的 50 年里,Unix 被反复扩展而没有适当考虑如何扩展保护机制。 Berkeley 的版本假设从一台机器扩展到一个机器网络,这些机器都在一个 LAN 上并且都在一个管理之下。 Internet 机制(telnet、ftp、DNS、SMTP)最初是为安全网络上的大型机编写的。

大型机是自治的,网络在安全协议之外,并且没有授权转移。因此,伯克利模型真正需要的远程身份验证根本不受支持。 Sun 的扩展(例如 NFS)加入了聚会,假设一家公司拥有多个受信任的 LAN。我们不得不改造 Kerberos、TLS 和 SSH 等协议,使其成为将世界维系在一起的胶带。数十亿部手机的出现,有时通过 wifi 有时通过移动网络进行通信,并运行来自数百万作者(其中大多数是自私的,其中一些是恶意的)的应用程序,已经让安全工程师以更快的速度追赶.

将许多不同的计算模型混合在一起是造成当前混乱的一个因素。他们的一些初始假设仍然部分适用,但不再适用于全球。互联网现在有数十亿部电话、数十亿物联网设备,也许还有十亿台个人电脑,以及数百万个组织,这些组织的管理者不仅无法合作,而且可能存在冲突。有竞争的公司;相互鄙视的政治团体,以及相互交战的民族国家。用户不是值得信赖但偶尔不称职,现在基本上是不熟练的 但有些人既有能力又充满敌意。

代码过去只是漏洞百出 但现在有很多恶意代码。对通信的攻击曾经是情报机构的职权范围 现在他们可以由从互联网上下载攻击工具的年轻人来完成

net 并启动它们,但对它们的工作原理一无所知。

6.5 总结

访问控制机制在系统中的多个级别运行,从硬件到操作系统和中间件（如浏览器)再到应用程序。更高级别的机制可以更具表现力,但由于从内在复杂性到实施者技能等各种原因,更容易受到攻击。

访问控制的主要功能是限制特定组、用户和程序可能通过错误或恶意造成的损害。应用最广泛的例子是客户端的 Android 和 Windows,以及服务器端的 Linux;他们有共同的血统和许多建筑相似之处。基本机制（及其问题)无处不在。大多数攻击都涉及对错误的机会主义利用;复杂、广泛使用或两者兼而有之的产品特别容易发现漏洞并将其转化为漏洞。已经开发了许多技术来减少实施错误的数量,以降低由此产生的错误引起漏洞的可能性,并且更难将漏洞转化为漏洞;但大型软件系统的整体可靠性提高缓慢。

研究问题

访问控制中的大多数问题都是在 1960 年代或 1970 年代初期发现的,并在 Multics [1684] 和 CAP [2020] 等实验系统上得到解决。从那时起,门禁系统的大部分研究都涉及在新环境（例如手机)中重新设计基本主题。

最近的研究主题包括 enclaves 和用于添加更细粒度访问控制的 CHERI 机制。另一个问题是:开发人员将如何有效地使用这些工具?

在第二版中,我预测“未来几年的一个有用的研究主题将是如何设计出不仅健壮而且可用的访问控制机制 对程序员和最终用户都是如此。” Yasemin Acar 和其他人最近的工作拾取了这一点,并将其发展成为安全研究中发展最快的领域之一 [11]。许多（如果不是大多数)技术安全故障至少部分是由于开发人员预期使用的保护机制的可用性差。我在密码学一章中已经提到,密码 API 如何经常诱使人们使用真正不安全的默认值,例如使用 ECB 模式加密长消息;访问控制同样糟糕,任何对 Windows 系统或 Intel 或 Arm CPU 中的访问控制机制不感冒的人都会发现。

作为预告片,这是一个新问题。我们能否将我们在技术层面（无论是硬件、操作系统还是应用程序)的访问控制知识扩展到组织层面?在 20 世纪,提出了许多安全政策,从 Bell-LaPadula 到 Clark-Wilson,我们在更广泛的讨论

第 2 部分的长度。现在我们有有趣的技术类比,是时候重新审视深度外包和虚拟组织的世界了吗?

进一步阅读

Allison Randal 在 [1575] 有虚拟化和容器的历史; Robert Watson 在 [1993] 中讨论了强制访问控制如何适用于 OS X 和 iOS 等操作系统;以及其架构师 Li Gong 编写的 Java 安全性参考书 [783]。云原生安全基金会正试图推动人们围绕容器和其他技术转向更好的开源实践,以部署和管理云原生软件。回顾一下,关于 Unix 安全的经典描述是 Fred Grampp 和 Robert Morris 在 1984 年 [805] 以及 Simson Garfinkel 和 Eugene Spa ord 在 1996 年 [753],而互联网安全的经典描述是 Bill Cheswick 和 Steve Bellovin [221] 给出了许多针对 Unix 系统的网络攻击的例子。

Carl Landwehr 对 1960 年代到 1980 年代操作系统中发现的许多缺陷提供了有用的参考 [1129]。Willis Ware 在 1970 年 [1986] 发表了关于该主题(实际上是关于一般计算机安全性)的最早报告之一; Butler Lampson 关于限制问题的开创性论文发表于 1970 年代 [1125],三年后,另一篇有影响力的早期论文由 Jerry Saltzer 和 Mike Schroeder [1639] 撰写。我们让学生阅读的关于访问控制问题的教科书是 Dieter Gollmann 的“计算机安全”[779]。关于 Intel 的 SGX 及其 CPU 安全架构的标准参考由 Victor Costan 和 Srini Devadas [479] 提供。

软件安全领域发展迅速;攻击从一年到下一年发生了显着变化(至少在细节上)。经典的起点是 Gary McGraw 2006 年的书 [1266]。从那时起,我们就遇到了 ROP 攻击、Spectre 等等;一个简短但有用的更新是 Matthias Payer 的软件安全 [1504]。但是要真正跟上,仅仅阅读教科书是不够的;您需要关注 Usenix 和 CCS 等安全会议以及 Bruce Schneier、Brian Krebs 和我们自己的 lightbluetouchpaper.org 等安全博客。当前攻击的最详细信息可能在 Google 的零项目博客中;例如,请参阅他们对在野外发现的 iPhone 攻击的分析,以深入了解使用强制访问控制组件 [204] 攻击现代操作系统所涉及的内容。