第27章

安全系统发展

我自己的经验是,如果开发人员拥有一套干净、富有表现力的特定安全要求,就可以构建出非常紧凑的机器。他们不必是安全专家,但他们必须了解他们正在尝试构建什么以及它应该如何工作。

– 里克·史密斯

当谈到成为时尚的奴隶时,美国经理们让青春期的女孩看起来像粗犷的个人主义者。

– Geo Nunberg

狐狸知道很多事情;刺猬一件大事。

– 阿奇洛科斯

27.1 简介

到目前为止,我们已经讨论了各种各样的安全应用程序、技术和问题。如果您是一名在职工程师、经理或顾问,付费构建或维护具有某些安全保证要求的系统,那么您现在正在寻找一种系统化的方法来解决这个问题。这将我们带到了风险分析、系统工程方法论等主题,最后是秘诀:如何管理团队以编写安全代码。

秘密在于实际上没有秘密,无论是酱汁还是其他任何东西。从 1980 年代的橙皮书到现在的敏捷开发,很多人声称有一个,并且对当下的热情产生了宗教狂热。

但对此提出的第一个建议是正确的。在 1960 年代,Fred Brooks 带领团队进行了世界上第一个真正大型的软件项目,即 IBM S/360 大型机的操作系统。在他的经典著作《人月神话》中,他描述了他们所遇到的所有问题,他的结论是“没有灵丹妙药”[328]。没有神奇的公式可以使

本质上很难的工作很容易。本章开头还有 Archilochus 的名言:狐狸知道很多事情,而刺猬知道一件大事。管理安全开发是狐狸知识而不是刺猬知识。一个经验丰富的安全工程经理必须知道成千上万的小事;这就是为什么这本书这么肥!随着软件无处不在并开始与安全交互,安全工程经理的工作越来越难。

2017 年,我改变了在剑桥教授本科生的方式。到那时为止,我们将安全课程与软件工程分开教授,后者侧重于安全。但大多数现实世界的系统都需要两者,并且它们以复杂的方式纠缠在一起。安全和保障都是紧急属性,确实必须从一开始就融入其中。两者都涉及对可能出错的系统性思考,无论是意外还是恶意。事故会使系统受到攻击,而攻击会使系统降级,从而变得危险。该课程由我的同事 Alastair Beresford 在我 2019 年休假期间进一步开发,由于大流行 [89],2020 年软件和安全工程课程现在以十个视频讲座的形式在线上线。该课程旨在为我们一年级的本科生打下坚实的基础,为以后在安全、密码学和软件工程方面的工作打下坚实的基础。与本书一样,它介绍了基础知识,从定义到协议和加密的基础知识,然后是人和组织问题以及技术问题的重要性,并通过案例历史进行了说明。它讨论了您如何设定安全和安保目标,如何随着系统的发展管理它们,以及如何将合适的思维方式和工作方式灌输到您的团队中。成功与态度、工作实践和技能有关。

您必须问的两个问题是:“我们是否在构建正确的系统?”和“我们建造得对吗?”在本章的其余部分,我将从我们如何评估和管理安全风险开始;然后继续讨论我们如何构建系统,一旦我们有一个规范来工作。然后,我将讨论由于组织行为而出现的一些危险 一种真实但经常被忽视的内部威胁。

27.2 风险管理

安全工程和安保工程的核心在于确定优先级:花多少钱来保护免受什么影响。风险管理必须在管理企业乃至国家所有风险的更广泛框架内进行。这通常做得很糟糕。新冠病毒危机本应让所有人清楚地看到,尽管流行病在包括英国在内的许多国家/地区的风险登记册中名列前茅,但大多数政府将更多的弹性预算用于恐怖主义,这在名单中排名靠后。最近经历过 SARS 或 MERS 的国家,如台湾和韩国,做得更好:他们准备对居民进行大规模检测和追踪接触者,并迅速做出反应。英国浪费了两个月的时间才意识到这种疾病很严重,以数万人的生命为代价。

那么什么是风险登记册呢?一种通用的方法,由

我所在大学的管理机构拟定一份可能出错的清单,为它们的严重性和发生概率打分 1 到 5,然后将这些相乘得到 1 到 25 之间的数字。

例如,如果一所大学的 20% 的收入来自该来源,那么大学可能会将“由于经济衰退导致的研究合同收入损失”的严重性评为 5/5,并将“可能性”评为 4/5,因为经济衰退经常发生但不会每年,给出 20 的原始产品。然后你写下你为减轻这些风险中的每一个而采取的措施,并在风险委员会中就每个风险的减轻程度进行辩论。例如,您通过制定一项规则来控制可变研究合同收入的风险,该规则只能用于雇用合同人员 ,而不是终身教员;然后您可能会同意此规则将风险从 20 降低到 16。然后您按顺序排列所有风险并指定一名高级官员作为每个风险的所有者。

国家风险评估有些相似:您根据可能造成的死亡人数(数百万?数千?数十?)对每个可能的不良事件(流行病、地震、森林火灾、恐怖袭击等)进行评级,然后根据概率对其进行评级每个世纪你期望多少。例如,英国国家风险登记册将大流行性流感排在首位,严重程度为 5(可能导致多达 750,000 人死亡),可能性为 4,并在 2017 年表示:“预计将出现一个或多个主要危害在英国每五年一次。最严重的是大流行性流感、全国停电和严重的洪水”[361]。然后,您可以通过缓解措施找出合理可行的方法,无论是针对大流行病的检疫计划和 PPE 储备,还是限制洪水和地震造成的破坏的建筑规范和分区。您进行成本效益分析并将优先事项转化为政策。您可能会以各种方式出错。英国在很大程度上忽略了流行病,因为国家安全委员会已被安全和情报机构占领;他们将恐怖主义列为优先事项,而卫生部长并不是固定出席者 [1848 年]。我已经在第 26.3 节中讨论了恐怖主义;在这里我只想补充一点,失败的另一个方面是政策超调。当 9/11 告诉世界恐怖袭击可以杀死数千人而不是几十人时,并且这些机构获得了更多的弹性预算,这让他们变得贪婪:他们开始谈论恐怖分子获得核武器的风险,所以他们会在登记册上有一个更可怕的威胁来证明他们的预算是合理的。

在商业中,你也会发现政治行为和组织行为都会妨碍理性的风险管理。但是你通常有关于更常见损失的真实数据,所以你可以尝试一种更量化的方法。标准方法是计算每种可能的损失情况的年度预期损失 (ALE),即预期损失乘以平均每年预期发生的事件数。银行 IT 系统的典型 ALE 分析可能有数百个条目,包括我们在图 27.1 中看到的项目。

请注意,虽然对于常见损失(例如“出纳员拿走现金”)可能有准确的数字,但低概率高风险损失(例如大型汇款欺诈)的发生率主要是猜测 尽管您有时可以得到通过询问保险报价来进行粗略的完整性检查。

ALE 长期以来一直被 NIST 标准化为美国政府采购中使用的技术。英国政府使用一种称为 CRAMM 的工具来

损失类型	金额 发生率 ALE $250,000 $100,000 $10,000
SWIFT欺诈	50,000,000 美元 .005 $648,000
ATM诈骗（大）	250,000 美元 .2
ATM诈骗（小）	20,000 美元 .5
出纳员取现金	3,240 美元 200

图 27.1：- 年化损失预期（ALE)项目

信息安全风险的系统分析,现代审计文化正在到处传播这样的工具。但是为低概率威胁生成这样一个表的过程往往只是反复猜测。顾问列出他们能想到的所有威胁,附加概念概率,计算出 ALE,将它们相加,发现银行的 ALE 超过了它的收入。然后,他们将总数调整到能够证明其客户 CISO 所说的政治上可能的最大安全预算的任何数额。如果这听起来有点愤世嫉俗,我很抱歉;但这似乎经常发生。关键是,ALE 可能具有一定的价值,但您需要了解哪些部分基于数据,哪些部分基于猜测,哪些部分基于政治。

产品风险不同不同。不同的行业因其发展方式和监管历史而以不同方式行事。每个部门的规则,无论是汽车、飞机、医疗设备还是铁路信号,都是根据事故和行业游说而制定的。欧盟正日益成为世界安全监管机构,因为它是最大的市场,因为华盛顿对安全的关心不如布鲁塞尔,而且对于原始设备制造商来说,按照欧盟安全规范进行设计比拥有多种产品更容易。

我将在下一章更详细地讨论安全和保障认证。
就目前而言,汽车、飞机和医疗设备的软件必须根据批准的程序进行开发,接受我们稍后将讨论的分析,并以特定方式进行测试。

保险可以帮助管理较大但不太可能发生的风险。但保险业务也不完全科学。您的保险费过去常常会给出您的业务运行风险的一些信号,尤其是当您为八位数或以上的损失购买保险时。但保险业是一个周期性行业,自 2017 年左右以来,许多新公司开始提供针对网络犯罪的保险,将利润挤出市场。因此,客户将不再忍受侵入性问卷调查,更不用说评估员的实地考察了。因此,大多数保险公司评估风险的能力现在是有限的;我将在第 28.2.9 节中进一步讨论它们的工作机制。他们还对同时引发许多索赔的相关风险保持警惕,因为这将迫使他们持有更多的储备;由于一些网络风险是相关的,政策往往要么将它们排除在外,要么相对昂贵 [275]。 （冠状病毒危机让公司了解相关风险,因为一些保险公司拒绝支付业务中断风险保单 即使是那些明确提到员工因流行病而无法上班的风险的保单;企业正在询问保险公司反过来保险的意义是什么。）

除了精算风险,大公司购买保险的一个非常重要的原因 以及许多其他公司行为 是为了保护

高管,而不是股东。正在应对的风险表面上看起来是可操作的,但实际上是法律、监管和公关风险。

董事要求购买责任保险,根据英国和美国法律,专业疏忽是指专业人士未能按照其专业中合理胜任的人所要求的水平履行职责。因此,疏忽索赔是根据行业或专业的现行标准进行评估的,从而强烈刺激了从众行为。这就是管理如此受时尚驱动的业务的原因之一(根据本章开头的引述)。

这溢出到用于证明安全预算合理性的话语中。在 80 年代中期,每个人都在谈论黑客(即使他们的人数很少)。
从 80 年代后期开始,病毒占据了企业的想象力,人们通过销售防病毒软件致富。 20世纪90年代中期,防火墙成为明星产品。 1990 年代后期见证了 PKI 的狂热。到 2017 年是区块链。

在所有这些喧嚣中,安全专业人员必须保持冷静的头脑,努力了解真正的威胁是什么。

我们将在后面的部分回到组织行为。首先,让我们看看我们可以从安全工程中学到什么。

27.3 安全关键系统的教训

关键计算机系统是指尽可能避免某一类故障的计算机系统。根据故障类别,它们可能是安全关键型、业务关键型、安全关键型或环境关键型。

安全关键类型的明显例子包括飞行控制和自动制动系统。有大量关于这个主题的文献,并且已经开发了许多方法来帮助智能地管理风险。

27.3.1 安全工程方法

安全工程方法与经典安全工程一样,倾向于系统地进行从安全分析到规范再到产品的工作,并假设您从一开始就构建安全而不是尝试对其进行改造。通常的程序是识别危害和评估风险;决定应对它们的策略(回避、约束、冗余……);追踪对硬件和软件组件的危害,从而将其识别为关键;确定同样重要的操作员程序,并研究各种应用心理学和运筹学问题;列出指定安全机制必须做什么的安全功能要求,以及指定安全功能令人满意地执行的可能性的安全完整性要求;最后决定一个测试计划。测试的结果不仅是您有信心实时运行的系统,而且是证明运行它的安全案例的集成部分。基本框架在 ISO 61508 等标准中有所规定,这是一个用于相对简单的可编程电子设备(例如化工厂的控制系统)的基本安全框架。这已经通过针对特定行业的更专业的标准得到了扩展,例如用于道路车辆的 ISO 26262。

如果确实出现问题,此安全案例将提供证据,即

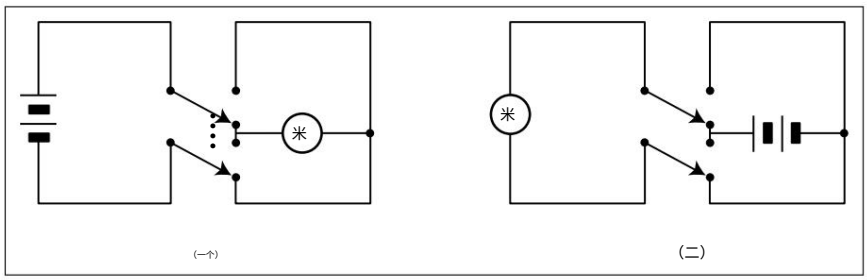

图 27.2：- 电机换向电路中的危险消除

你尽了应有的注意。它通常包括危害分析、安全功能和完整性要求，以及表明已达到所需故障率的测试结果（组件级和系统级）。测试可能必须由认可的第三方进行；对于机动车公司来说，安全案例是由同一公司的不同部门完成的，具有独立的管理。

由于其供应链，车辆的情况更为复杂。顶部是品牌，您可以在汽车前部看到其徽章。然后是原始设备制造商 (OEM)，就汽车而言通常是同一家公司，但并非总是如此；在其他行业中，品牌和 OEM 是完全分开的。一辆现代汽车将拥有来自数十家制造商的零部件，其中直接与品牌打交道的一级供应商进行大部分研发工作，但轮流从其他公司获得零部件。

在汽车行业，品牌对汽车进行型式认可并承担主要责任，但如果出现问题，则要求零部件供应商进行赔偿（大多数国家的法律不允许您对死亡和伤害承担责任）。该品牌在安全功能和完整性的重要部分以及安全案例中都依赖供应链。还有紧张局势：正如我们已经指出的那样，安全认证可能会阻止及时应用安全补丁。现在让我们看看常见的安全工程方法以及它们可以教给我们什么。

27.3.2 危害分析

在理想情况下，我们或许能够完全从系统中设计出危险。
例如，考虑图 27.2 中的电机反转电路。在左侧的设计中，双刀双掷开关使从电池流经电机的电流反向。但是，这有一个潜在的问题：如果开关的两个极中只有一个移动，电池就会短路，并可能引起火灾。解决方法是将电池和电机调换，如右图修改后的电路。在这里，开关故障只会使电机短路，不会使电池短路。安全工程不仅涉及正确操作，还涉及纠正故障。

危险消除在安全工程中也很有用。我们在 12.3.2 节中看到了 SWIFT 早期设计的一个例子：在那里，用于验证一家银行和另一家银行之间交易的密钥在

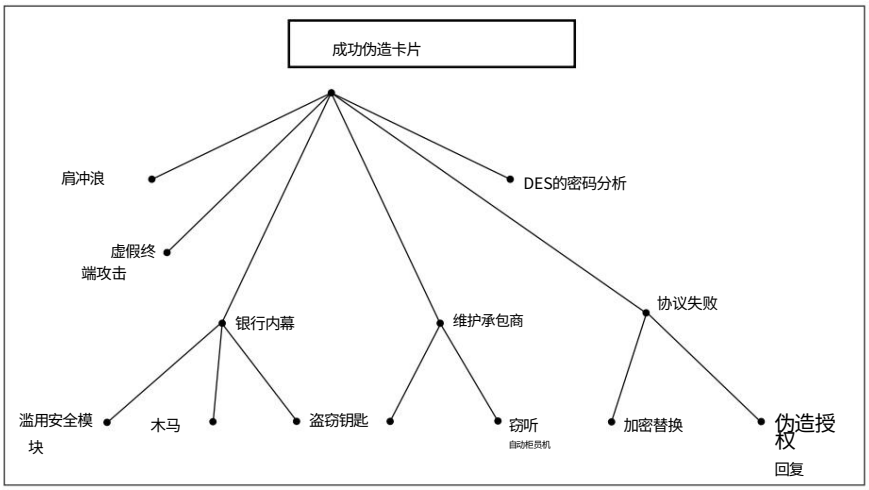

图 27.3：- 威胁树

银行,因此 SWIFT 没有伪造有效交易的手段,其员工和系统的信任度也降低了。通常,最小化可信计算基数是一种消除危险的练习。这同样适用于隐私工程。例如,如果您正在设计一个接触者追踪应用程序来监控流行病中谁可能感染了谁,一种方法是拥有一个包含每个人手机位置历史记录的中央数据库。然而,这有明显的隐私危害,可以通过在每个人的手机上保存蓝牙联系历史,并上传任何请病假的人的联系历史来减少这种危害。然后,您可以在更好的隐私和更好的追踪之间做出政策决定。

27.3.3 故障树和威胁树

尽可能多地消除危险后,下一步就是识别可能导致事故的故障。识别可能出错的事情的一种常见的自上而下的方法是故障树分析,其中构建的树的根是不良行为,其连续节点是其可能的原因。这种自上而下的方法很自然,因为你有一个复杂的系统,其中有少量你必须避免的众所周知的不良结果。它以自然的方式延续到安全工程。图 27.3 显示了自动柜员机欺诈的故障树（或威胁树,因为它在安全工程中经常被称为）示例。

美国国防部使用威胁树。你从每一个不希望的结果开始,然后通过写下每一个可能的直接原因来回溯。然后,您通过添加每个先决条件来递归。通过在树叶周围工作,您应该能够看到可能破坏您的安全策略的技术攻击、操作失误、物理渗透等的每种组合。您从中得到的另一个好处是攻击路径之间的共性可视化,这使得更容易推理如何以最少的努力破坏最多的攻击 ort。在一些

变体,攻击分支有反制子分支,可能有反反制攻击分支等等,用不同的不同颜色来强调。威胁树可以相当于系统的攻击手册,因此它可能是高度机密的,但这是国防部的要求 如果系统评估人员或认证人员能够发现重要的额外攻击,他们可能会使产品失败。

27.3.4 失效模式和影响分析

回到安全至上的世界,另一种进行危险分析的方法是故障模式和影响分析 (FMEA),由 NASA 首创,它是自下而上而不是自上而下1。这涉及跟踪每个系统组件故障的后果,一直到对任务的影响。这是具有少量众所周知的关键组件或子系统的系统(例如飞机)的自然方法。例如,如果您要驾驶飞机飞越海洋或山脉,在发动机故障的情况下您无法滑行到机场,那么发动机功率就至关重要。因此,您研究了动力装置发生故障的平均时间及其故障模式,从连杆断裂到燃料耗尽。您坚持单引擎飞机使用可靠的引擎,并规范维修计划;飞机有不止一个油箱。当运载大量乘客时,您坚持使用多引擎飞机并训练机组人员处理引擎故障。

人们错过了后来证明是关键的故障模式的一个航空航天例子是 1986 年挑战者号航天飞机的失事。NASA 项目经理知道助推火箭中的 O 形环存在风险,并且在之前的飞行中发现了损坏;同时,承包商知道低温会增加风险;但这些担忧并没有集中起来,也没有传达给 NASA 的最高管理层。一个因寒冷而变脆的 O 形环失效 导致航天飞机和七名机组人员损失。在由此产生的调查委员会中,物理学家理查德·费曼在电视上展示了这一著名的现象,他将 O 形环样本放入夹具中,在冰水中冷冻,然后显示当他松开它时,它仍然凹陷并且没有弹回[1612]。这说明失败通常不仅仅是技术上的,还涉及人们在组织中的行为方式:当保护机制跨越制度边界时,例如汽车,您需要考虑法律和经济学以及工程。随着我们转向依赖各种第三方服务和基础设施的自动驾驶汽车,此类问题将变得更加复杂。

27.3.5 威胁建模

故障树和 FMEA 都取决于分析师对系统的真正了解;它们很难自动化,不能完全重复,并且可以通过对子系统的细微更改来完成。因此,对故障模式的彻底分析通常会将自上而下和自下而上的方法与一些特定的方法结合起来

1FMEA 在技术意义上是自下而上的,即分析是从单个组件开始的,但它的实际管理通常具有自上而下的风格,因为一旦有了概要设计就开始处理安全案例,并随着设计的进行逐步完善演变为产品。

人们随着时间的推移学会了应用程序。许多行业现在不得不重新考虑其传统的安全分析方法以纳入安全性。

在汽车安全方面,复杂的供应链意味着我们必须对车辆及其子系统进行多重连锁分析。传统的子系统分析可能会分析前照灯的故障模式,因为在夜间驾驶时失去前照灯会导致事故。除了通过使用两个或更多灯来降低灯故障的风险外,您还担心开关故障,当开关变为电子开关时,您可以构建可能的硬件和软件故障的故障树。当我们将其从安全扩展到安全时,我们会考虑攻击者是否可能接管汽车的娱乐系统,并在汽车以足够快的速度行驶时使用它在 CAN 总线上发送恶意的“lamp o ”消息这很危险。此分析可能会导致在机舱 CAN 总线和动力总成 CAN 总线之间设置防火墙的设计决策。 （这是道路车辆网络安全新草案 ISO 21434 标准中的工作示例 [962]。）

更一般地说,从安全到安全的转变意味着必须系统地考虑内部人员。正如复式簿记旨在抵御单个不诚实的职员,并经过重新设计以抵御职员在其 PC 上安装恶意软件的类似威胁,因此现代大型系统通常旨在限制损害,如果承诺单个组件。那么如何将恶意内部人员纳入威胁模型呢?如果你使用 FMEA,你可以在不同的位置添加一个对手,就像我们的恶意“lamp o ”消息一样。至于更复杂的系统,Frank Swiderski 和 Window Snyder [1851] 描述了 Microsoft 在 2003 年大力推动使 Windows 和 Oce 更安全之后采用的方法。

这不是纯粹的自上而下或自下而上,而是一种中间相遇的方法。基本思想是,你不仅要列出你试图保护的资产（进行交易的能力、访问机密数据等）,还要列出攻击者可用的资产（可能是订阅你的系统的能力,或者操纵你提供给他的智能卡的输入,或者在你的呼叫中心找到一份工作）。然后,您可以跟踪整个系统的攻击路径,从一个模块到另一个模块。您试图弄清楚信任级别可能是多少;障碍在哪里;以及可以使用哪些技术（例如欺骗、篡改、否认、信息泄露、服务拒绝和特权提升）来克服特定障碍。威胁模型可用于安全开发生命周期中不同点的各种目的,从架构审查到目标代码审查和渗透测试。

有多种方法可以管理由此产生的海量数据。一种基本方法是针对安全机制构建危害矩阵,如果安全策略是每个严重危害必须至少受到两个独立机制的约束,那么您可以在每个相关列中检查两个条目。因此,您可以用图形方式证明,在存在相关危险的情况下,至少需要两次故障才能导致事故。另一种方法,系统理论过程分析 (STPA),从危险开始,然后在自上而下的过程中设计控制,从而导致系统的架构设计;这有助于区分交互控制循环 [1150]。这种方法论涉及安全工程 [1556]。一种或另一种方式,为了使复杂性

可管理,您可能必须组织安全和安保目标的层次结构。
本书第 II 部分中讨论的安全策略可能会为您提供我们在那里讨论的应用程序的答案的开始,并为其他人提供一些灵感。然后,此层次结构可以根据您所在行业使用的术语来驱动风险矩阵或风险处理计划。

27.3.6 量化风险

安全关键系统社区有许多处理故障和错误率的技术。组件故障率可以统计测量;软件中的错误数量可以通过我将在下一章讨论的技术进行跟踪;并且在不同类型的活动中有很多关于操作员失误概率的经验。安全关键系统中人为因素工程的圣经是 James Reason 的书“人为错误”;我在第 3 章中讨论了 2010 年代安全可用性研究的兴起,因为来自安全领域的经验教训已经开始渗透到我们的领域。

一项任务的错误率取决于它的熟悉程度和复杂程度、压力大小和成功线索的数量。如果一项任务简单、经常执行并且有强烈的成功提示,则错误率可能为十万分之一。然而,当在需要逻辑思维且操作员处于压力之下的混乱环境中首次执行任务时,成功完成的可能性就会降低。三哩岛和切尔诺贝利告诉核工程师,无论你做了多少次设计走查,当红灯真正亮起时,最严重的错误就会出现。一次又一次的空难调查得出了同样的教训。当数十个警报同时响起时,很有可能有人会按错按钮。一个指导原则是默认为安全状态:抑制核反应,使飞机恢复直线水平飞行,或让自动驾驶汽车停在路边。没有任何原则是万无一失的,安全状态可能难以衡量。如果有草地边缘,车辆很难分辨路边在哪里;在波音 737Max 坠毁事件中(我在第 28.2.4 节中详细描述),飞行控制计算机试图保持飞机水平,但被一个有故障的迎角传感器弄糊涂了,而是将飞机俯冲到地面。

紧急情况下安全可用性的另一个原则是保持提供给操作员的信息,以及可供他们使用的控件,既简单又直观。在过去,每个饲料都进入一个单一的仪表或刻度盘,它们的空间有限。现在的诱惑是把一切都给运营商,因为你可以。在过去,设计师知道紧急情况会给飞行员带来狭隘的视野,所以他们将这 6 个仪器放在他们真正需要的仪器中。现在可能有五十个警报而不是两个,飞行员很难弄清楚要查看电子飞行信息系统的哪个屏幕上的哪个菜单。它比航空广泛得多。海军的一个例子是 2017 年麦凯恩号航空母舰在新加坡海峡发生的碰撞,其中 UI 混乱是一个主要因素。转向控制被转移到错误的掌舵站,发动机没有及时节流,导致在繁忙的航道上未经命令转向港口,

与一艘化学品油轮的撞击,以及十名水手的死亡 [1929]。

因此,非完全自主的系统必须保持可控,为此需要了解可能出现的人为错误。关于使特定类型的错误更常见的认知偏差和其他心理因素,我们已经了解了很多;我们在第 3 章讨论过它们,谨慎的工程师会研究它们在各自领域的工作原理。在操作员已经具备一定技能的频繁执行的任务中,错误很少发生,而在操作员感到压力和惊讶时更容易出错。这开始让我们走出风险领域,那里的赔率是已知的,进入不确定的领域,在那里他们不是。

在安全系统中,也可能会在重要但很少执行的任务中出现最严重的错误。安全可用性不仅仅是向最终用户呈现一个漂亮的直观界面。它应该以符合威胁和保护的常见心理模型的方式呈现风险,并且用户对压力的可能反应应该导致安全结果。

对执行每项关键任务的人员的技能水平和任何已知的错误可能性估计要切合实际,这一点很重要。

飞机设计师可以依赖任何持有商业飞行员执照的人的可预测技能水平,造船商知道商船中一名船员的长处和短处。汽车可以并且确实由年老体弱、年轻且缺乏经验、被乘客分心或受酒精影响的司机驾驶。在专业方面,可用性测试可以与员工培训结合起来获利:当飞行员在模拟器中进行进修课程时,教员会抛出各种设备故障、恶劣天气、机舱危机和空中交通管制的组合对他们感到困惑。他们观察了哪些压力组合会导致致命事故,以及这些压力在不同驾驶舱类型中有何不同。这些数据是对驾驶舱设计师的宝贵反馈。在航空领域,安全运营的激励机制足够强大且协调一致,规模也足够大,足以支持学习系统。即便如此,还是有代价高昂的灾难,例如波音 737Max 飞行控制软件 这不仅有至少一个严重的错误,而且逃脱了适当的故障模式和影响分析,因为负责的工程师 – 在他们的经理的压力下完成按时完成项目 错误地认为飞行员能够应对任何失败 [89]。因此,该软件依赖于单个攻角传感器,而不是使用飞机上安装的两个传感器,传感器故障导致致命事故 2。

在测试冗余系统的可用性时,需要注意故障屏蔽:如果输出由三个处理器之间的多数表决决定,其中一个发生故障,那么系统将继续正常工作 但其安全裕度将有所下降被侵蚀,也许以操作员无法正确理解的方式。驾驶舱系统之一失灵的客机飞行导致了几起空难;尽管飞行员可能在理智上意识到驾驶舱显示器的其中一个数据馈送是不可靠的,但他们可能会在压力下通过反射而不是与其他仪器进行检查来依赖它。因此,您必须认真考虑如何保留故障

2 DO178 和 DO254 等航空安全标准除了冗余以减轻共模故障外,通常还要求在测量类型、物理、处理特性方面具有多样性。

即使它们的直接影响被减轻,也是可见和可测试的。

安全关键系统的另一个教训是,虽然安全要求规范和测试标准将需要作为律师和监管机构的安全案例的一部分,但将它们与主流产品文档集成是一种很好的做法。如果安全案例是单独的,那么很容易在批准后将其搁置一旁,并且无法妥善维护。 (这是导致波音 737Max 灾难的一个因素,因为飞行控制软件安全案例背后的可用性假设没有从之前的 737 型号更新。)从基于项目的软件管理转向敏捷方法,并通过 DevOps对于 DevSecOps,最终开始将安全管理嵌入到产品发展的方式中。我们将在下一节讨论这个问题。

最后,安全就像安全一样,因为它确实必须在系统开发时内置,而不是改装。两者之间的主要区别在于故障模型。安全处理随机故障的影响,而在安全方面,我们假设有一个敌对的对手可以在最不方便的时间以最具破坏性的方式导致我们系统的某些组件发生故障。在对手面前,人们自然会更加规避风险;我将在 28.4 节中讨论这个问题。安全工程师将对 MTBF 为 109小时的关键飞行控制系统进行认证;安全工程师不得不担心对手是否可以强行为十亿分之一的故障设置先决条件并按需使飞机坠毁。

实际上,我们的任务是对一台计算机进行编程,使其在可能最不方便的时刻给出微妙而恶意的错误答案。我将此描述为“为撒旦的计算机编程”,以将其与更常见的墨菲 [113] 编程问题区分开来。这是安全工程困难的原因之一:撒旦的计算机更难测试 [1668]。

27.4 优先保护目标

如果您有一个创建全新产品或从根本上改变现有产品的项目,最好花一些时间根据第一原则思考保护优先级。仔细的安全分析或威胁建模练习可以提供一些数字来说明这一点。在详细制定安全案例或安全策略时,了解上下文至关重要,本书的大部分内容都是关于与广泛应用程序相关的威胁模型。您还应该尝试从环境或上下文中改进对风险的数值估计。

在业务系统的情况下,分析将取决于风险和回报之间的权衡 。安全人员往往过分关注前者。如果您的公司营业额为 1000 万美元,毛利润为 100 万美元,盗窃损失为 150,000 美元,您可能会就“如何通过阻止盗窃将利润提高 15%”进行减损宣传;但如果你能将营业额翻一番,达到 2000 万美元,那么即使亏损翻三倍,达到 450,000 美元,股东也会更愿意这样做。现在的利润为 155 万美元,增长了 85%,而不是 15%。在线欺诈引擎的经验证实了这一点。在与

在众多零售商中,我注意到获得最佳结果的公司是那些欺诈管理团队向销售部门而非财务部门报告的公司。英国一家典型的实体店零售商可能会拒绝大约 4% 的购物篮,因为欺诈引擎会对商品、送货地址和付款细节的组合发出警报。因此,如果您可以改进欺诈引擎并仅拒绝 3%,那就是增加 1% 的销售额 – 一个让您的首席营销官大开眼界的前景。但如果反欺诈团队转而向首席财务官报告,他们很可能会被视为成本而非机会。

同样,在线服务的站点可靠性工程师也学会了不要让系统过于可靠。如果本地 Internet 可用性只有 99%,那么 99.9% 的服务就可以了;如果您的用户都不会注意到差异 erence,那么再花费数百万美元来达到 99.99% 就毫无意义。

你最好 o 故意设置一个 0.1% 的错误预算,你可以有效地使用它,例如通过造成偶尔的故意失败来锻炼你的弹性机制 [236]。这让我想到了安全管理中的一场公开辩论:人们是否应该以不在其所依赖的任何软件中打开 CVE 为目标?勾选框方法是说“当然不能有开放的 CVE”,但这可能会带来相当高的合规成本。

如果你是谷歌,并且编写了你自己的所有基础设施,也许你可以瞄准那个目标;许多公司不能也必须优先考虑。我将在后面的第 27.5.7.1 节中更详细地讨论 CVE。

所以不要相信那些只会谈论“加强安全”的人。通常它已经太紧了,你真正需要做的只是稍微改变一下 。在本书的第一版中,我介绍了一个超市自助结账的案例研究。二十年前,多家超市开始引入自助结账通道。有些人开始沉迷于损失,并通过在产品重量方面积极挑战客户来让安全性妨碍可用性。获得优势的商店之一开始采用更宽容的方法,他们根据经验逐渐调整。

最终,该行业想出了如何运营自助结账通道,但实施质量仍然存在很大差异。到 2020 年初,先驱是像瑞典 Lifvs 这样没有员工的小型便利店 ;你用一个应用程序打开商店的门,扫描你的购买并在线支付。

亚马逊也在试验完全自助的食品店。我们看到未来 20 年的创新都挤进了 2020 年冠状病毒封锁的几个月;到 6 月,其他超市一直在敦促我们下载他们的扫描应用程序,在我们挑选商品时扫描我们购买的商品,将它们记入卡中,然后就可以走了。

许多现代商业模式曾被认为风险太大,从自助超市本身开始,当时杂货商将所有商品都放在柜台后面。理查德·西尔斯 (Richard Sears) 在 1880 年代采用“保证满意或退款”的口号时,每个人都认为他会破产,但他发明了现代邮购业务。在商业中,利润是对风险的回报。但成功的企业家可能必须迅速提高安全性。

最近的一个例子是视频会议平台 Zoom 到 2020 年 3 月,该平台的用户从 2000 万增加到 2 亿,并在此过程中从企业平台转变为更像是公共事业的平台 迫使他们

进入一项重大的安全工程工作 [1763]。

安全方面的取舍更难。从逻辑上讲,在发达国家,一条人命的价值可能是几百万美元,这是一个普通人一生的收入。然而,我们对安全行为所揭示的人命的实际估值从改善道路交叉口的大约 50,000 美元到火车保护系统的 5 亿美元以上不等 这只是在交通政策的背景下。卫生政策的差异更大,每挽救一条生命的成本从流感疫苗和某些癌症筛查的几百美元到最不有效的干预措施的数十亿美元 [1869];在其他安全环境中,家用烟雾报警器每挽救一条生命要花费数百美元,而“反恐战争”的数量则为数十亿 [1350]。这种不合理性的原因是相当容易理解的 我在 3.2.5 节中讨论了心理学,在 26.3.3 节中讨论了政策方面。安全偏好会因敌对行动的威胁而急剧改变;人们可能会完全忽略万分之一被设计不当的医疗设备杀死的风险,直到设备有可能被黑客入侵,此时甚至万分之一的风险都会变得可怕。我在 28.4 节讨论了这个现象。

27.5 方法论

软件项目通常花费的时间比计划的时间长,成本比预算的高,并且错误比预期的多3。到 1960 年代,这已被称为软件危机,尽管“危机”一词可能不适用于现在已经持续了两代人的状态,例如计算机不安全。

不管怎样,软件工程这个术语是由 Brian Randall 在 1968 年提出的,并定义为:

> 软件工程是建立和使用合理的工程原理,以经济地获得可靠且在真实机器上高效工作的软件。

先驱们希望以基础科学的基础和设计规则的框架 [1420],以我们建造船舶和飞机的相同方式解决这个问题。从那时起,取得了很大进展,但结果出乎意料。回到 1960 年代后期,人们希望我们能够将失败的大型软件项目的数量从当时观察到的 30% 左右减少。但我们仍然看到大约 30% 的大型项目失败了 不同之处在于失败的规模要大得多。现代工具让我们在跌倒之前越过复杂的山峰 ,但失败率取决于公司经理对风险的偏好。我们将在本章结尾的组织行为部分进一步讨论这个问题。

软件工程是关于管理复杂性,其中有两种。使用不适当的工具进行编程会带来附带的复杂性,例如一些早期机器只支持的汇编语言;用这种语言编写具有图形用户界面的现代应用程序将是极其乏味和容易出错的。那里

3 这有时被称为“胡夫定律”,以纪念大金字塔的建造者。

也是处理大而复杂问题的内在复杂性。
例如,一家银行的核心系统可能涉及数千万行代码,这些代码实现了通过多个不同交付渠道销售的数百种不同产品,而且任何人都无法理解。

附带的复杂性主要使用技术工具来处理。最重要的是高级语言,它们隐藏了处理机器特定细节的大部分苦差事,并使程序员能够在适当的抽象级别开发代码。他们带来了自己的费用;许多漏洞是 C 语言特性的结果,如果我们重新运行历史,我们肯定会使用 Rust 之类的东西来代替。还有一些正式的方法,例如静态分析工具,可以检查特别容易出错的设计和编程任务。

内在复杂性需要一些微妙不同的东西:帮助我们将问题划分为可管理的子问题并限制这些子问题可以相互作用的程度的方法。这些反过来又得到了他们自己的工具集的支持。基本上有两种方法 自上而下和迭代。

27.5.1 自上而下的设计

系统开发的经典模型是 Win Royce 在 1960 年代为美国空军 [1628] 形式化的瀑布模型。这个想法是您从系统要求的简明陈述开始;将其详细说明为规范;实施和测试系统的组件;然后将它们集成在一起并作为一个系统进行测试;然后上线系统上线运行。从 1970 年代到 2000 年代中期,这就是美国国防部所有系统的开发方式,全世界许多政府都效仿他们的做法,不仅包括国防领域,还包括行政和医疗保健领域。当我在 1980 年代在银行业工作时,那里也是经过批准的流程,由 IBM、政府和大型会计师事务所大力推广。

这个想法是用用户语言编写需求,用技术语言编写规范,单元测试根据规范检查单元,系统测试检查是否满足要求。在此链中的前两个步骤中,有关于我们是否正在构建正确的系统(验证)的反馈,在接下来的两个步骤中有关于我们是否正在构建正确的系统(验证)的反馈。可能有四个以上的步骤:一个常见的细化是随着需求被发展成更详细的规范而进行一系列细化步骤。但顺便说一下。

瀑布模型的定义特征是开发从第一个需求陈述到系统在现场的部署不可避免地向下流动。尽管每个阶段都有对其前身的反馈,但(比如)系统测试对需求没有系统级反馈。

有一个用于安全关键系统开发的版本称为 V 模型,在该模型中,系统向下流动到实施,然后爬回另一侧的验证和确认山,在那里它根据实施、规范和要求。这是一个

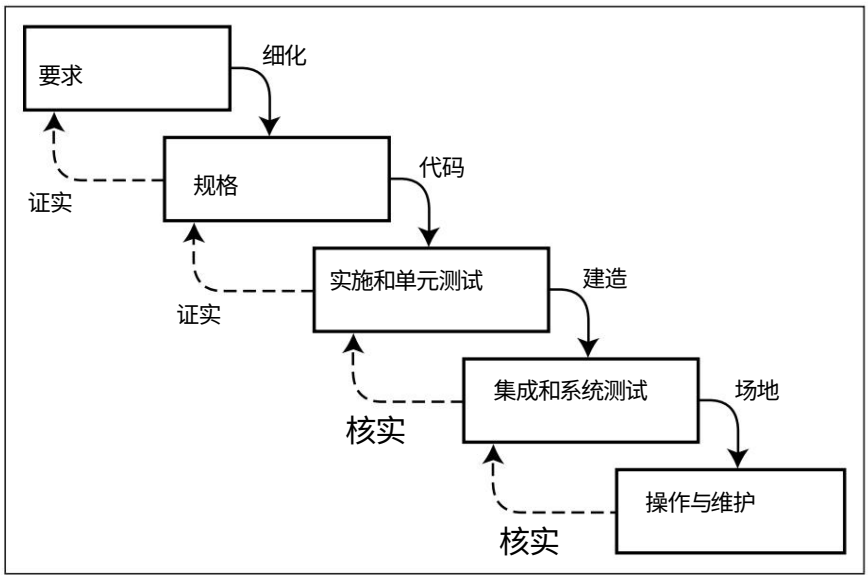

图 27.4：- 瀑布模型

德国政府标准,也广泛用于世界航空航天工业;它出现在汽车软件安全的 ISO 26262 标准中。但是,尽管它是从左到右而不是自上而下编写的,但它仍然是一个单向过程,需求驱动系统,验收测试确保满足需求,而不是一种根据需求演化需求的机制的经验。与其说是不同的动物,不如说是不同的图表。

瀑布模型是德国格哈德·帕尔 (Gerhard Pahl) 和沃尔夫冈·拜茨 (Wolfgang Beitz) 在第二次世界大战后为设计和建造机床等机械设备而开发的方法学的先驱 [1490];显然,Pahl 的一名学生后来回忆说,它最初的设计目的是让工程专业的学生入门,而不是作为对经验丰富的设计师实际工作的准确描述。 Win Royce 还把他的模型看作是开始从混乱中获得秩序的一种手段,而不是它发展成的规范系统。

瀑布模型的优势在于它要求尽早阐明系统目标、体系结构和接口;通过提供明确的里程碑目标,它使项目经理的任务更容易;它可以通过对每个步骤和任何延迟的规格更改单独收费来提高成本透明度;并且它与多种工具兼容。

在可以使其发挥作用的地方,它通常是最好的方法。关键问题是在任何开发或原型制作工作之前是否详细了解需求。有时是这种情况,例如在编写编译器或（在安全领域)设计密码处理器以实现已知事务集并通过特定级别的评估时。有时由于外部原因需要采用自上而下的方法,例如星际空间探测器,您只能对其进行一次射击。

但通常情况下,详细的需求事先并不知道,因此需要采用迭代方法。技术可能正在改变;环境可能正在改变;或者项目的关键部分可能是人机界面的设计,这可能涉及测试多个原型。设计师最重要的任务往往是帮助客户决定他们想要什么,虽然这有时可以通过讨论来完成,但通常需要一些原型设计。

有时一个正式的项目太慢了。雷金纳德·琼斯将英国在第二次世界大战中电子战的相对成功归因于英国科学家迅速将 stu 组合在一起,而德国人使用严格的自上而下的开发方法,获得了设计精美的设备,但总是晚了六个月[990]。

但使用迭代开发的最常见原因是我们从我们想要改进的现有产品开始。即使在计算的早期,大多数程序员的努力也总是花在维护和增强现有程序上,而不是开发新程序上;调查表明,成功的 IT 产品的总拥有成本的 70-80% 是在首次投入使用后产生的,即使使用瀑布方法也是如此 [2060]。如今,随着软件成为嵌入式代码、应用程序和云服务的问题,它们都变得越来越复杂,许多公司的现实是“维护就是产品”。

即使在 20 世纪 90 年代后期,当最复杂的人工制品是 Microsoft Oce 等软件包时,编写这样的东西的唯一方法就是从现有版本开始并增强它。这并没有使瀑布模型过时;相反,它通常用于管理项目以开发主要的新功能,或重构现有代码。但是,现在一个主要产品的整体管理很可能是基于迭代的。

27.5.2 迭代设计:从螺旋式到敏捷式

迭代开发有不同的形式,从快速原型设计到巩固新产品规格,再到修复或增强现有系统的管理过程。

在第一种情况下,一种方法是螺旋模型,其中开发通过预先商定的迭代次数进行,在迭代中构建和测试原型,管理人员能够评估每个阶段的风险,以便他们决定是否继续与下一次迭代或减少他们的损失。由 Barry Boehm 设计,它被称为螺旋模型,因为该过程通常如图 27.5 所示。在许多应用中,初始原型是关键的第一步;从一家旨在制作演示以向投资者展示的初创公司,到一家公司构建新产品模型以展示焦点小组,再到旨在证明某些拟议技术并非完全不可能的 DARPA 种子项目。安全工程师的原型应用范围从安全可用性测试台到概念验证攻击代码。关键是解决你面临的最糟糕的问题,从而尽可能地降低项目风险。

第二种情况我们现在描述为敏捷开发,可以概括为

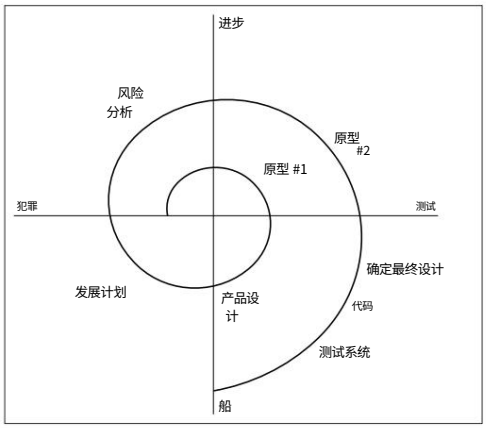

图 27.5: – 螺旋模型

标语：“解决你最糟糕的问题。重复”。

哈伦·米尔斯 (Harlan Mills) 是进化方法的早期倡导者，他教导说，您应该构建最小的可用系统，在真实用户身上试用，然后以小增量添加功能。到 1990 年代，打包软件行业就是这样学会工作的：随着 PC 的功能越来越强大，软件产品变得如此复杂，以至于无法从头开始经济地开发（或重新开发）它们。事实上，微软不止一次尝试重写 Word，但每次都放弃了。 1994 年微软的史蒂夫·马奎尔 (Steve Maguire) 撰写的《调试开发过程》是一部具有里程碑意义的关于演进式开发的早期著作 [1209]。在这种世界观中，产品不是项目的结果，而是涉及不断修改先前版本的过程的结果。微软将其方法与当时最大的 IT 公司 IBM 的方法进行了对比；在 IBM 生态系统中，瀑布方法占主导地位。 （IBM 人则谴责微软是一群没有纪律的黑客，他们生产有缺陷的、不可靠的代码；但在微软窃取了他们的主要业务市场后，IBM 濒临死亡的经历被归因于 IBM 开发方法的僵化 [390]。）从那时起，专业实践在四分之一个世纪里得到了发展，进化发展现在被称为“敏捷”，但它是同一头野兽。

关于进化发展的一个关键见解是，正如生物物种的每一代都必须生存才能使物种继续存在一样，每一代进化的软件产品也必须是可行的。核心技术是回归测试。每隔一段时间 通常是一天一次 所有致力于产品不同功能的团队都会检查他们的代码，这些代码被编译成一个构建，然后针对大量输入自动进行测试。

回归测试检查过去有效的东西是否仍然有效，以及旧错误是否没有找到回来的路。某人的代码总是有可能破坏构建，因此我们认为当前的“一代”是最后一个有效的构建。当系统必须协同工作时，事情会稍微复杂一些，例如当应用程序必须与云服务对话时，或者当车辆中的多个电子组件必须协同工作时，或者必须定制单个车辆组件才能工作时在几种不同的车辆中。你可以结束

构建和测试制度的层次结构。但不管怎样,我们总是有可行的代码,我们可以将这些代码用于 Beta 测试,或者我们流程的下一阶段可能是什么。

测试技术可能是 1990 年代和 2000 年代初期软件工程中最大的实际改进。在自动回归测试被广泛使用之前,IBM 工程师曾经估计 15% 的错误修复要么引入了新错误,要么重新引入了旧错误 [18]。向进化发展的转变与许多其他变化有关。例如,IBM 将系统分析师、程序员和测试人员的角色分开;分析师与客户交谈并生成设计,程序员对其进行编码,然后测试人员查找代码中的错误。激励措施并不完全正确,因为程序员可能会抛出大量有缺陷的代码,并希望其他人会修复它。这很慢并且导致代码臃肿。

微软取消了分析师、程序员和测试人员之间的区别;它只有开发人员,他们与客户交谈并负责修复自己的错误。这阻碍了编写大量错误的糟糕程序员,因此更多的代码是由更熟练和更细心的开发人员生成的。根据 Steve Maguire 的说法,这就是使 Microsoft 赢得了统治 32 位操作系统世界的战斗的原因;他们更好的开发方法让他们从 IBM [1209] 手中夺取了价值 1000 亿美元的商业软件市场。

27.5.3 安全开发生命周期

到 2000 年代初,微软已经超越 IBM 成为领先的科技公司,但它面临着越来越多的批评,因为 Windows 和 Oce 中的安全漏洞导致越来越多的恶意软件出现。服务器转向 Linux,个人用户开始购买 Mac。最终在 2002 年 1 月,比尔盖茨向所有员工发送了一份“可信计算”备忘录,命令他们将安全置于功能之上,并在工程师接受安全培训时停止所有开发。他们的内部培训材料变成了书籍和论文,帮助推动更广泛的生态系统发生变化。我已经在第 27.3.5 节中讨论了他们的威胁建模;他们对安全开发的首次尝试出现在 2002 年 Michael Howard 和 David LeBlanc 的“编写安全代码”[927] 中,其中阐述了 Microsoft 管理安全生命周期的早期方法,我在本书的第二版中对此进行了讨论。随着时间的推移出现了更多,它们的安全开发生命周期 (SDL) 出现在 2008 年,被 Windows 开发人员广泛采用。

广泛使用的 2010 年 SDL“简化实施”本质上是一个瀑布过程 [1308]。它“旨在减少软件漏洞的数量和严重程度”,并“在开发过程的所有阶段引入安全和隐私”。“pre-SDL”部分是安全培训;假定所有开发人员都获得了基础课程,其内容将取决于他们是否正在构建操作系统、Web 服务或其他内容。

然后有五个 SDL 组件。

1.要求:这涉及风险评估和质量门或“错误栏”的建立,如果代码包含某些类型的缺陷,它将阻止代码进入下一阶段。要求本身

定期审查;在微软,评审的时间间隔绝不会超过六个月。

2.设计:这个阶段需要威胁建模和攻击面的建立,以反馈到产品的详细设计中。

3. 实施:在这里,开发人员必须使用经过批准的工具,避免或弃用不安全的功能,并对代码进行静态分析以检查是否已完成。

4. 验证:此步骤涉及动态分析、模糊测试和攻击面审查。

5.发布:这是基于事件响应计划和最终安全性的前提
审查。

除了为所有开发人员提供一些基本的安全培训外,还有一些更深入的组织方面。首先,安全需要来自开发团队外部的主题专家 (SME),以及团队内部的安全或隐私拥护者来检查是否一切都已完成。

其次,有一个成熟度模型。从 1989 年开始,Watts Humphrey 在卡内基梅隆大学 (CMU) 的软件工程研究所开发了能力成熟度模型 (CMM),其基础是能力是团队的功能,而不仅仅是个人开发人员的功能。乐队不仅仅是将六位有能力的音乐家聚集在一起,软件也是如此。开发人员从不同的不同的编码风格、不同的注释和格式化代码约定、不同的 API 管理方式,甚至不同的工作流程节奏开始。 CMU 的研究表明,新成立的团队往往会低估项目的工作量,而且他们花费的时间也有很大差异;根据平均开发时间,合作得最好的团队能够更好地预测他们需要多长时间,但也减少了方差 [1937]。

这需要自律以牺牲资源分配的一些效率,以便为个别工程师提供连续性并保持团队的集体专业知识。 Microsoft 对此进行了调整,并为开发人员团队定义了四个级别的安全成熟度。

27.5.4 门控开发

这说明推动进化发展的最大公司恢复了瀑布式安全方法。当时的许多安全工程方法都与瀑布假设联系在一起,并且出于多种原因,自动测试本身对安全工程师的用处不大。安全属性既是突发的又是多样的,我们安全工程师的人数较少,而且在工具方面的投资也没有那么多。特定的攻击类型通常需要特定的补救措施,并且许多安全漏洞跨越系统的抽象级别,例如当规范错误与用户界面功能交互时 - 这种问题很难设计自动化测试。但是,虽然回归测试还不够,但它是必要的,因为它可以找到受到变化影响的功能。尤其重要的时候

开发冲刺添加了许多可以相互交互的功能。出于这个原因,Windows 的安全补丁是门控开发的一个例子:每隔一段时间,产品的预发布版本就会通过一系列额外的测试和审查,并为发布做好准备。这在具有安全或安保要求的系统中相当普遍。准备工作可能涉及使用各种外围设备和应用程序对 Windows 进行测试,或者对受管制产品的软件进行重新认证。

许多人忽视的一个问题是安全要求不断发展,并且还必须进行维护和升级。它们可能受到不断变化的环境、不断演变的威胁、对新旧平台的新依赖以及一系列其他因素的驱动。有些变化是隐含的;例如,当您升级静态分析工具时,您可能会在现有代码库中发现数百个“新”错误,您必须对这些错误进行分类。微软再次成为这方面的先驱。

当在 Windows 中发现漏洞时,仅仅修补它是不够的;编写它的人可能已经编写了十几个类似的,现在分散在整个代码库中,一旦你发布了一个补丁,坏人就会研究并理解它。因此,您不仅可以修复一个错误,还可以更新您的工具链,以便找到并消除产品中所有类似的错误。为了管理 Microsoft 及其客户的成本,该公司于 2003 年开始将补丁捆绑到每月更新中,即现在著名的“补丁星期二”。从那时到 2015 年,所有客户 - 从企业到用户家用 PC 和平板电脑 在每个月的第二个星期二更新软件。这样的修补会产生更多的依赖性。现代质量工具可以帮助您检查没有代码有 CVE 打开,因此如果您的所有客户都依赖这些工具,他们也应该打补丁。但很多人没有:手机和台式机上多达 70% 的应用程序在它们使用的开源库中存在漏洞,通常可以通过简单的更新来修复[1695]。自 2015 年以来,Windows 家庭用户不断收到更新4。

许多相同的考虑适用于安全关键系统,这些系统在许多方面与安全系统相似。安全性与安全性一样,是整个系统的涌现属性,它不会组合。在大多数应用中,安全性过去依赖于广泛的上市前测试。但是,没有安全保障的联网设备很难有安全保障,而且现在汽车等设备已连接到互联网,它们也正在获得补丁周期。然而,确保最新版本的安全关键系统满足安全案例可能需要进行广泛且昂贵的测试。例如,一辆汽车可能包含来自不同组件供应商的数十个电子控制单元 (ECU),除了测试各个 ECU 之外,您还必须测试它们如何协同工作。

汽车行业的公司相互怀疑,不会彼此共享源代码,即使在保密协议下也是如此,因此测试可能很复杂。主要测试装置可能是一辆“实验室汽车”,其中包含特定型号汽车的所有电子设备,以及可让您模拟各种操作甚至事故的额外测试系统。这些需要花费真金白银,而且您还需要保留真车

4这也打破了一些东西:我们曾经正要向电视工作人员展示一个使用身体动作捕捉套装的实验,当我们用来驱动它的 Windows 笔记本电脑自动更新时,捕捉软件突然不能再工作了。随后,我们接到了荷兰软件开发商的疯狂电话,幸运的是,我们在几个小时后得到了他们的更新,正好赶上演出。

道路测试。当欧盟决定要求汽车公司在最后一辆车离开展厅后十年内修补汽车软件时,维持实验室车队和真实测试车的成本是汽车公司拖延的原因之一。

这是特斯拉具有显着优势的一个方面;作为一家以软件为核心业务的科技公司,特斯拉可以在数周内测试和交付变更,而传统汽车公司则需要数年时间,因为他们将大部分软件开发留给了零部件供应商 [404]。传统上,汽车软件合同涉及十年的支持;现在您需要支持产品开发三年,陈列室七年,再过十年。我将在下一章讨论这方面的可持续性方面。

与此同时,特斯拉正在迫使传统行业提高竞争力,大众宣布他们已经斥资 80 亿美元创建了一个合适的软件部门,而他们的特斯拉竞争对手项目已经延期 [1686]。

27.5.5 软件即服务

自 2010 年代初以来,越来越多的软件托管在中央服务器上,由瘦客户端访问并按订阅付费,而不是出售和分发给用户。典型的客户在许可费用之外还要承担许多运行软件的成本,不仅包括服务器和运营商的成本,还包括部署、定期升级和管理软件的成本。如果供应商可以从他们所有的客户那里接管这些任务,许多重复的成本就会被消除,并且他们可以因为他们的专业知识而更好地管理事情。可以对软件进行检测,以便开发人员可以在仪表板上监控其性能的所有方面。

软件即服务 (SaaS) 背后的关键技术创新是持续集成和持续部署。供应商可以将一些客户迁移到新版本进行测试,然后再迁移其余客户,而不是让成千上万的客户管理数十个不同版本的软件。升级变得更加可控,因为它们可以根据真实客户数据的快照在空运行中进行测试,称为暂存环境。

一些公司现在每天部署几次,因为他们的经验是,与更大的部署(例如 Microsoft 的星期二补丁)相比,频繁的小更改更安全并且破坏某些东西的风险更小。

部署本身是暂时的。 SaaS 公司通常会在负载均衡器后面的 VM 上运行的许多服务实例上运行其软件,负载均衡器为管理正在运行的服务提供了一个间接点。单独的实例还提供单独的故障域以提高稳健性。为了进行滚动部署,我们配置了一个负载均衡器,将 1% 的流量发送到一个新版本的实例,通常被称为“金丝雀”,以矿工用来检测一氧化碳泄漏的笼中鸟命名。如果金丝雀幸存,部署可以逐步前滚到新的服务实例。如果日志系统检测到任何问题,开发人员会收到警报。需要注意的是,如果用户在交易期间在设计的新旧版本之间摇摆不定,事情就不会出错。如果您进行了破坏向后兼容性的更改,您通常会构建一个既适用于旧系统又适用于新系统的中间阶段(无论如何,我们在 1980 年代的银行大型机世界中就是这样做的)。

通过分阶段发布和滚动部署来管理风险的能力改变了测试的经济性。您可以极快地修复错误这一事实意味着您可以通过更少的测试达到目标质量水平。

您还可以看到用户所做的一切,因此您第一次可以真正从安全性、安全性和收入的角度了解可用性是如何失败的。当然,通常是收入推动了对它的利用。

分析收集器将所有行为事件写入日志,该日志被馈送到数据管道中用于指标、分析和查询。这反过来又支持可以对可能的功能进行广泛的 A/B 测试的实验框架。广告驱动的服务可以通过参与指标进行优化,例如活跃用户、每个用户会话的时间和特定功能的使用。受控实验也用于提高安全性;例如,谷歌通过测量数百万用户对过期证书的不同警告的反应来调整其浏览器警告。

这些改进本身通常很小,因此您确实需要对照实验来衡量它们;但是当你做很多的时候,它们就会加起来。将此类框架构建到分阶段部署机制中的投资会增加规模回报;您拥有的用户越多,您获得统计显着性的速度就越快。因此,大公司可以比规模较小的竞争对手更快地优化产品;与许多其他数字技术一样,SaaS 不仅可以在短期内削减成本,而且可以增加长期锁定。每次访问大型 SaaS 公司的服务时,您可能都在不知不觉中参与了数十甚至数百次实验。关于运行多个并发实验以及部署系统增强功能,有很多繁琐的细节。

当您将多个微服务组合在一起时,事情会变得更加复杂。这将我们带到了基础架构即代码的世界,也称为云原生开发或 DevOps,其中一切都在容器、VM 等中开发,因此所有基础架构都基于代码并且可以快速复制。您还可以使用容器来简化事情,将尽可能多的安全依赖项与代码打包在一起。新代码可以快速部署到测试基础设施并进行实际测试。如果您想手动管理滚动部署,您可以这样做,但这不可扩展且容易出错。解决方案是编写部署代码,作为应用程序开发过程的一部分,使用云平台 API 允许应用程序自行部署和相关基础设施,并连接到监控机制。在过去的几年中,一些工具包已经可用,允许工程师以更具声明性的方式执行此操作。

我所知道的最好的指南是谷歌 2013 年的书“站点可靠性工程”； SRE 是 DevOps 的术语 [236]。谷歌在用大量低成本 PC 构建大型可靠系统的艺术方面处于行业领先地位,为负载平衡、复制、分片和冗余构建必要的工程。在 2000 年代和 2010 年代初期,由于他们的运营规模比其他任何人都大,他们不得不将更多任务自动化并变得擅长。 SRE 的目标是可用性、延迟、性能、效率、变更管理、监控、应急响应和容量规划。核心策略是应用软件工程技术来自动化系统管理任务,以便在快速创新与可用性之间取得平衡。

正如我们已经指出的,如果满足以下条件,则争取 99.9999% 的可用性毫无意义

ISP 只允许用户在 99% 或 99.9% 的时间内访问您的服务器。如果您设置一个现实的错误预算,比如 0.1% 或 0.01% 的不可用性,您可以使用它来实现许多事情。首先,大多数中断是由于实时系统更改造成的,因此您可以很好地监控延迟、流量、错误和饱和度,并在出现任何问题时迅速回滚。您使用剩余的错误预算来支持您的实验框架,并进行受控中断以清除依赖项。 （这是由 Netflix 开创的,它的“混沌猴子”偶尔会关闭路由器、服务器、负载均衡器和其他组件,以检查弹性机制是否按预期工作;这种“消防演习”现在已成为行业标准,涉及关闭整个数据中心。）

在 12.2.6.2 节中,我们提到了技术债。这个概念,由于沃德坎宁安,概括了发展捷径就像债务的观察。每当我们吝啬文档、快速解决问题、不彻底地测试修复、未能建立安全控制或未能解决错误的后果时,我们都在储存可能的问题必须在未来连本带息偿还[41]。技术债务对于初创公司或接近生命周期尽头的系统可能有意义,但它通常是管理不善或激励机制不一致的产物。随着时间的推移,系统可能会深陷债务之中,以至于变得难以维护或使用;它们必须被重构或替换。对于一家银行来说,必须更换其核心银行系统是非常昂贵和具有破坏性的。所以管理技术债务真的很重要;这是自本书第二版以来系统管理思想的变化之一。 DevOps 哲学的一个重要方面是无债务运行。

27.5.6 从 DevOps 到 DevSecOps

正如我所写的那样,2020 年的前沿是将敏捷思想和方法不仅应用于开发和运营,还应用于安全。从理论上讲,这可能意味着“一切皆代码”的策略;在实践中,这意味着不仅要维持现有的安全等级（以及相关的安全案例）,还要应对新的威胁、环境变化和令人惊讶的漏洞。将两者结合起来涉及实际工作,有时需要重新发明。例如,我在第 12.2.2 节中提到,DevOps 破坏了银行多年来依赖的开发和生产之间的分离;在需要职责分离的地方,我们必须重新构想它。

我们在与我们合作的公司中看到了几种不同的方法。下面我将举两个例子,我们可以粗略地称之为微软世界和谷歌世界。当然还有很多其他的。

27.5.6.1 Azure 生态系统

在过去的 25 年中,从银行和保险公司到零售业再到航运和采矿业,世界上大多数最大的商业公司都在 Windows 上构建了他们的企业系统,现在正在将它们迁移到 Azure,通常使用系统集成和设施管理公司来完成实际工作.典型的客户混合使用本地系统和云系统,新的开发大多从前者迁移到后者。这里的政策主要由

四大审计师除了他们的标准内部控制功能集外,还跟随 Microsoft 要求安全的开发生命周期。用于威胁建模、静态分析、动态分析、模糊测试、应用程序和网络监控、安全编排和事件响应的数十种工具造成了巨大的开销,数十人将数据从一种工具复制到另一种工具。 DevSecOps 的任务是通过自动化这些管理任务来逐步集成这些工具。

为了支持这个生态系统,微软通过进一步的步骤扩展了它的 SDL:定义指标和合规性报告;威胁建模;密码标准;管理第三方组件的安全风险;渗透测试;以及标准化的事件响应。该公司现在声称其 10% 的工程投资用于网络安全。有能力的系统集成和设施管理公司已经想出了将这些步骤构建到他们的工作流程中的方法;许多实际工作涉及集成他们或他们的客户购买的第三方安全产品。

适当的自动化对于安全团队继续改进他们的游戏、扩大他们的范围和提高效率至关重要;没有它,他们就会越来越落后,最终会筋疲力尽 [1846]。

在这样的公司中,DevSecOps 的组织原则将是“左移”,这可以涵盖很多方面:统一的主题是将安全性(如软件和基础设施)转移到代码库中。一种策略是让事情“快速失败”,包括在开发过程中尽早让安全专家参与进来,以避免以后的延迟:对每个开发人员的代码进行预提交静态分析,以最大限度地减少失败的构建;购买或构建专业工具来检测错误,例如不正确的身份验证、使用加密功能的错误以及注入机会;新版本的自动和手动安全测试;配置和部署的自动化测试,包括扫描暂存网络和检查凭据、加密密钥等。虽然早在 2010 年,微软就认为操作安全与软件安全是分开的,但现代 Azure 商店将通过持续监控、手动渗透测试和最终发现错误的第三方的漏洞奖励来跟进部署,从而关闭循环。稍后我们将更详细地讨论这些。

27.5.6.2 谷歌生态系统

第二种观点来自从事基础设施工作的工程师,据我所知,最好的参考资料是六位谷歌工程师于 2020 年出版的一本书,“构建安全可靠的系统”[23]。 DevSecOps 策略与亚马逊有点相似,但针对他们的产品进行了优化;他们的 CTO Werner Vogels 在 [1966] 中对此进行了描述。但是,对 Google 体验的描述要详细得多。本节借鉴了他们的书,以及最近在主要服务公司工作的同事。

在构建数亿人依赖的基础设施系统时,快速实现支持功能的自动化,并拥有真正强大的威胁识别、事件响应、损害限制和服务恢复流程至关重要。因此,虽然设施管理公司可能致力于整合支持功能以节省资金并减少错误,但重点

在主要服务公司,可靠性是重中之重。我已经提到了谷歌网站可靠性工程的方法:设定一个现实的目标,比如99.9% 的可用性,然后通过在故障恢复、升级和实验之间分配 0.1% 的停机时间来使用剩余错误预算。

这反过来又推动了进一步的原则,例如可恢复性设计、可理解性设计以及尽可能阻止人类接触生产系统的愿望。对新二进制文件的增量部署进行自动化是不够的;您还希望停止系统管理员必须在路由器中输入复杂的命令行来配置网络;正如我们在 21.2.1 节中提到的,这是大多数网络中断的来源。

您可以通过构建合适的工具代理来管理此类风险。这可能涉及大量工作来调整二进制文件和配置文件的更新,并确定如何在 SRE 和安全工程团队之间分配支持和恢复工作。安全测试会带来更多的复杂性。您如何构建测试基础架构以行使最小特权?您如何测试包含大量个人信息的系统?您如何测试使 SRE 团队能够紧急访问实时系统的玻璃碎片机制?其中大部分是我们在 1980 年代的大型机世界中已经不得不处理的问题,但它们只是偶尔出现,并且是通过人类的聪明才智和信任一些关键人员来处理的。将一切从数千名用户扩展到数十亿意味着必须自动化更多。

仍然存在紧张局势。在站点可靠性工程中,告警应尽可能简单、可预测和可靠;但在安全方面,一些随机化通常是个好主意。

在应用程序级别,系统越来越多地划分为具有可防御的安全边界和防篡改安全上下文的微服务组件,因此如果 Alice 破坏了购物系统的目录,她仍然不能像 Bob 一样花钱,因为支付服务是独立的。每个组件通常将实现为多个并行副本或分片,从而提供更小的故障域。这些域使您能够限制任何妥协的爆炸半径;理想情况下,您希望能够在不使整个系统离线的情况下处理入侵。分区系统也可以设计为具有弹性,但这并不简单。当 domain 失败时,你什么时候启动一个新的,什么时候做一些不同的事情?依赖关系是什么?哪些组件应该故障打开,哪些应该故障安全?在拥塞或受到攻击的情况下,什么样的性能下降是可以接受的?减载和节流的作用是什么?您可以合理地给用户和商业模式带来什么样的痛苦?您是否放弃了一些广告,需要额外的验证码来登录,还是两者兼而有之?您如何测试和验证所有这些弹性机制?

大公司投入大量工程时间来为此类服务构建应用程序框架。网页也有标准框架,不仅要在第一时间防止SQL注入和跨站脚本攻击,还要提供对几十种不同语言的支持。

有一个单一的前端来终止所有 http(s) 和 TLS trac 意味着如果你必须更新你的证书管理机制或密码套件你只需要做一次,而不是在你所有的不同服务中。单个前端还可以为负载平衡和 DDoS 保护提供单个位置,以及

至于许多其他功能,例如支持数十种不同的语言。

使用类型封装来强制执行 URL、SQL 等的属性可以减少您需要验证的代码量。如果您拥有也易于理解的构造安全 API,那是最好的。谷歌有一个名为 Tink 的加密 API,它强制更正确地使用。它需要使用密钥管理服务,无论是在谷歌云、AWS 还是 Android 密钥库中。这适合用于管理加密终止、代码来源、完整性验证和工作负载隔离的整体框架,称为 BeyondProd [998]。

27.5.6.3 创建学习系统

无论您是采用 Microsoft 的方法、Google 的方法还是您自己的方法,要调整这样一个过程,您都需要指标,合适的指标包括向开发团队开放的安全票证数量、安全失败构建的数量以及所需的时间用于新应用程序以符合相关法规（无论是 SOX、GDPR 还是 HIPAA）。随着 Dev、Sec 和 Ops 的融合,指标和管理流程与第 21.4 节中讨论的网络防御机制融合在一起,从网络监控到安全事件和事件管理。但所有这些都需要智能管理。

一家经营良好的公司可以通过冲刺让所有开发/运维人员更清楚地看到安全流程 ,你可以通过冲刺来进行隐私影响评估、改进访问控制、扩展日志记录或其他任何事情。一家经营不善的公司将设法达到指标,这会造成紧张:他们的安全人员最终可能会达成相互矛盾的目标,即把坏人拒之门外,以及通过达到所有用于证明其合理性的指标来“喂养野兽”团队自己的存在[1846]。重要的是要了解作为组织管理结构功能的冲突自然会在何处出现,并以某种方式使冲突保持建设性。

不过,在这两种情况下,重要的驱动因素之一将是漏洞生命周期。错误变成漏洞利用然后攻击的过程,注意到这些攻击导致漏洞报告,使用防火墙等设备进行临时防御,然后不仅针对直接用户而且沿着复杂的供应链推出最终补丁的过程越来越多安全管理的核心。

27.5.7 漏洞周期

早在 20 世纪 70 年代和 80 年代,人们有时将发现系统中的安全漏洞然后轻蔑地修复它们的进化过程描述为渗透和修补。希望限制攻击面的架构和形式化方法的应用的某种组合能够让我们逃脱。正如我们所见,这并没有真正奏效,除了一些边缘情况,例如加密设备。到 2000 年代初,我们得出的结论是,我们必须更好地管理补丁周期,而安全漏洞披露法、CERT 和负责任的披露等现代方法在此期间得到巩固。

漏洞周期由某人（研究人员)发现供应商维护的系统中的漏洞的过程组成。

研究人员可能是客户、学者、国家情报机构的承包商，甚至是罪犯。他们可能会在市场上出售。 Jean Camp 和 Catherine Wolfram 在 2000 年首次提出了脆弱性市场的想法 [371]；公司成立是为了购买漏洞，随着时间的推移，出现了几个市场。大多数大型软件和服务公司现在都提供漏洞赏金，金额从数千美元到数十万美元不等；另一个极端是运营商购买漏洞并出售给网络武器制造商（卖给军事和情报机构）和法医公司（卖给执法部门）等剥削者。这些运营商现在提供数百万美元用于持续远程攻击 Android 和 iOS。

研究人员也可以直接向供应商披露漏洞 现在许多供应商都有漏洞赏金计划，为试图匹配市场价格的披露漏洞支付奖励，至少在数量级上。

随着针对流行平台的零日攻击的市场价格已涨至六位数甚至七位数，漏洞赏金也是如此。例如，Apple 悬赏 100 万美元奖励任何无需用户点击即可破解 iOS 内核的人。 2019 年，至少有六名黑客仅通过漏洞赏金平台 HackerOne [2030] 就赚取了超过 100 万美元。大型错误赏金的一个缺点是，虽然错误过去是自然发生的，但我们现在看到它们是故意引入的，例如代码最终出现在重要平台上的开源项目的贡献者。这种供应链攻击曾经是民族国家的专利；现在他们正在开放 [890]。

如果在供应商发布补丁之前在野外使用漏洞，则称为零日漏洞，通常用于有针对性的攻击。如果使用得足够多，那么最终会有人注意到；攻击被报告，然后供应商发布一个补丁，然后可以对其进行逆向工程，以便许多其他参与者现在拥有漏洞利用代码。未能修补系统的客户现在很容易受到犯罪团伙可以大规模部署的多种攻击。

获得正确的补丁周期是信息安全经济学中的一个问题，就像其他任何问题一样，因为不同利益相关者的利益可能会发生根本性的分歧。

1. 供应商宁愿根本找不到错误，以节省打补丁的费用。他们会在必要时进行修补，但希望将成本降至最低，如果他们的代码出现在许多产品版本中，这可能包括大量测试。事实上，如果他们的代码用于现在需要修补的客户设备（如汽车），他们可能必须支付赔偿金以支付客户的成本；因此，在这些行业中，拖延和拒绝的动机更为强烈。

2. 普通客户可能更希望没有发现错误，以避免打补丁的麻烦。懒惰的客户可能无法打补丁，并因此被感染。 （如果受感染的机器所做的只是发送一些垃圾邮件，它们的所有者可能不会注意到或关心。）

3. 典型的安全研究人员希望为他们的发现获得一些奖励，无论是名誉、金钱还是修复他们所依赖的系统。

4. 情报机构希望快速了解漏洞，以便在发布补丁之前将它们用于零日攻击。

5. 安全软件公司受益于未修补的漏洞,因为他们的防火墙和 AV 软件可以寻找妥协指标来阻止利用它们的攻击。

6. 大公司不喜欢补丁,政府部门也不喜欢,因为针对企业的关键系统测试新补丁并将其推出的过程非常昂贵。更好的公司已经建立了自动化来处理像微软星期二补丁这样的常规事件,但是更新或风险评估他们办公室和工厂中的数以万计的物联网设备将在未来几年内令人头疼。大多数公司只是没有足够好的资产库存系统来应对。

在 1990 年代,争论的起因是那些对软件供应商让产品几个月甚至几年都没有打补丁感到沮丧的人。bugtraq 邮件列表的建立是为了让人们可以匿名披露错误;但这意味着产品可能会在一两个月内完全易受攻击,直到编写、测试和发布补丁,直到客户公司测试并安装它。这引发了关于“负责任的披露”的辩论,提出了关于研究人员应该给供应商多长时间的喘息空间的各种建议 [1572]。

出现的共识是,研究人员应向计算机应急响应小组 (CERT)5 披露漏洞,全球 CERT 网络将通知供应商,并在漏洞发布之前延迟发布补丁。最终披露的威胁让供应商大吃一惊;延迟给了他们足够的时间在发布之前正确测试修复;研究人员因投递简历而获得荣誉;客户在收到错误报告的同时得到了错误修复;大公司定期组织更新日期,供企业客户计划。哦,这些机构有一条热线连接到他们当地的 CERT,所以他们提前了解了自然发生的漏洞利用并可以利用它们。这是 26.2.7.3 节中描述的交易的一部分,该交易于 2000 年结束了 Crypto War 1。

27.5.7.1 CVE 系统

工业方面是 1999 年推出的常见漏洞和披露 (CVE) 系统,它为公开发布的软件包中报告的漏洞分配编号。这是由 Mitre 维护的,但它将 CVE 的分配委托给了大型供应商。 CVE ID 通常包含在安全公告中,使您能够搜索报告日期、受影响产品、可用补救措施和其他相关信息的详细信息。有一个通用漏洞评分系统 (CVSS),它提供漏洞严重性的数字表示。随着时间的推移,计算这个的方法变得越来越复杂,现在取决于攻击是否需要本地访问、它的复杂性、所需的努力、它的效果、利用代码和补丁的可用性、目标的数量和损坏的可能性。

5欧盟正在重新命名这些 CSIRT 计算机安全事件响应团队。

NIST 的国家漏洞数据库 (NVD) 被描述为“综合性网络安全漏洞数据库,集成了所有公开可用的美国政府漏洞资源并提供对行业资源的参考”,它基于 CVE 列表。这些资源对于自动跟踪漏洞和更新至关重要。现在报告了成千上万个漏洞,发布了数百个补丁,自动化是必不可少的。

随着系统逐渐成熟,它成为安全经济学家的研究课题。传统主义者认为,由于漏洞数量众多且互不相关,而且大多数漏洞利用都使用从现有补丁逆向工程得到的漏洞,因此应该尽量减少披露。实用主义者认为,从理论和实证的角度来看,需要公开威胁才能让供应商打补丁。我在 8.6.2 节中讨论了这个论点。从那时起,我们看到了针对大众市场用户的自动升级的引入,在漏洞中做市的公司的建立,以及对漏洞相关程度的实证研究。模一些调整,当前计算机行业的做事方式已经稳定了十多年。

27.5.7.2 协调披露

然而,一些行业远远落后。在第 4.3.1 节中,我描述了大众如何起诉伯明翰和奈梅亨大学的学者,因为他们发现并负责任地披露了大众远程钥匙输入系统中的漏洞,这些漏洞已经被在线提供的汽车盗窃工具所利用。这是一个错误,因为它引起了人们对漏洞的关注,大众汽车在法庭上正式败诉。像微软和谷歌这样的公司用了 20 年的时间才知道,运行漏洞赏金计划和每月修补比威胁起诉人更有效,但许多传统行业的公司仍然没有解决这个问题,尽管他们的产品包含更多和更多更多软件。

大众汽车案例中的一个问题是,研究人员最初向其关键输入系统的供应商披露了该漏洞,而后者只是在最后一刻才告知大众汽车。由于此类供应链问题,负责任的披露已让位于协调披露。很少有公司再构建所有自己的工具,甚至一个孩子的玩具也可能有多个软件依赖项。如果它进行语音和手势识别,它可能包含一个运行某种 Linux 或 FreeBSD 版本的 Arm 芯片,与运行另一种 Linux 版本的云服务通信,并且可以由运行在 Android 或 iOS 上的应用程序控制。玩具的安全将取决于安全通信;例如,2019 年 2 月发现 Enox 的“Safe-KID-One”玩具手表与其后端服务器之间的通信未加密,因此黑客理论上可以追踪和呼叫孩子。回应是立即在欧盟范围内进行安全召回 [654]。弄错这类事情可能会让你的产品和你的公司猝死。

现在,当有人在用于数十种嵌入式产品的平台中发现可利用的错误时会发生什么?这可能会造成创伤,例如 Linux 中的 Shellshock 错误和 OpenSSL 中的 Heartbleed 错误(也影响了 Linux)。如果 Linux 获得紧急补丁,协调披露是一个

噩梦:Linux 维护者可能会私下与主要的 Linux 发行版一起工作,并与开发人员保持密切联系的 Android 等衍生产品一起工作。但是有数以千计的产品结合了 Linux,从闹钟到电视,从儿童玩具到地雷。您可能会突然发现您楼宇安全系统中的闭路电视摄像机都变得可以破解,而供应商无法快速或根本无法修复它们。在平台上协调披露是一个非常困难的问题。

没有灵丹妙药,但您仍然可以做很多事情,从记录您的上游和下游依赖关系,通过积极测试您所依赖的软件,以便您练习和理解错误报告机制,到成为其开发人员的一部分社区。

处理此类冲击只是一个过程的一个方面,该过程在 2010 年代后期成为其自身的专长,即安全事件和事件管理。

27.5.7.3 安全事件和事件管理

当您了解到漏洞或攻击时,您需要一个事件响应计划来确定您将要做什么。在过去,供应商可能需要几个月的时间才能对产品的新版本作出回应,而且通常什么都不做,只会发出警告(甚至拒绝)。如今,美国和欧洲的违规通知法都要求公司披露个人隐私可能受到损害的攻击,人们希望问题能迅速得到解决。您的计划需要四个组成部分:监控、维修、分发和保证。

首先,确保您尽快了解漏洞 - 最好不要晚于坏人(或媒体)。这意味着建立一个威胁情报团队。在某些应用程序中,您只能从专业公司获取威胁情报数据,而如果您是 IoT 供应商,则操作自己的蜜罐可能是谨慎的做法,这样您就可以立即收到有人攻击您的产品的警告。倾听客户的意见很重要:您需要一种有效的方式让他们报告错误。提供一些奖励可能是个好主意,例如为他们的下一次升级提供积分、彩票甚至现金。您绝对需要参与第 27.5.7 节中描述的错误赏金、漏洞市场、CERT 和 CVE 的更大技术生态系统。

其次,您需要能够修复问题。二十年前,这意味着每个产品团队都有一名成员带着传呼机“随叫随到”,以防凌晨三点需要修理。如今,这意味着准备对从漏洞报告到重大漏洞的任何事情做出精心策划的响应。这将从我们在第 21.4.2.3 节和威胁情报团队中讨论的入侵检测和网络监控功能扩展到确定负责的开发团队并通知上游供应商和下游客户。响应团队可能还需要其他通信方式。你有没有想过你是否需要卫星电话?

第三,你需要能够快速部署补丁:如果所有软件都在你自己的服务器上运行,那么这可能很容易,但如果它涉及为数百万消费设备打补丁代码,那么就需要提前计划。让您的客户每天访问您的网站一次并检查是否有升级似乎很容易,

但是,如果他们自己的系统依赖于您的设备并且他们需要测试任何依赖性,就会出现紧张关系 [195]:快速应用补丁的先驱可以发现破坏他们系统的问题,而花时间测试的人将更容易受到攻击.供应链越长,利益冲突就越难管理。操作非常重要:未经测试的紧急补丁程序可能弊大于利,经验告诉我们,在紧急情况下,您只需尽快运行正常的补丁程序 [23]。

最后,您需要提前教育您的 CEO 和主要董事会董事,让他们了解快速和诚实地处理安全漏洞的必要性,以保持信心并限制损失,方法是向他们提供令人信服的公司表现良好和其他表现不佳的例子.您需要有一种机制来联系您的 CEO 并立即向他们介绍情况,以便他们可以表明一切都在控制之中并让您的主要客户放心。因此,您需要知道可能急需帮助的每个人的手机号码和家庭电话号码。你需要一个应对媒体的计划。你最不需要的就是数十名记者打电话给你的公关人员甚至你的总机接线员,因为你疯狂地努力修复这个错误。为不同严重程度的事件准备好一组新闻稿,这样您的 CEO 只需选择正确的新闻稿并填写详细信息。一旦第一个(或第二个)记者打来电话,它就可以发送。

提醒您的 CEO,美国和欧洲都有违反安全漏洞的披露法,因此如果您的系统遭到黑客攻击并且数百万客户卡号遭到泄露,您必须通知所有当前和以前的客户,这会花费真金白银。您可能会被起诉。如果您有 1000 万客户的个人数据遭到泄露,这可能意味着 1000 万封信件每封 5 美元和 300 万张重新发行的信用卡每张 10 美元,即使您没有收到银行对这些账户的实际欺诈损失的索赔。 (这很可能会发生;您可能预计在 300 万个账户中,无论如何每年都会有数万个账户遭受欺诈,银行会就所有这些问题起诉您。)重大违规造成的经济损失可以轻松打到九位数。如果这种情况不止一次发生在您身上,您可能会失去客户:在一次通知违规后客户流失率可能仅为 2%,但在两次后客户流失率可能仅为 30%,而在三次后客户流失率甚至更高 [2037]。由于一些 CEO 在重大漏洞后被解雇,信息安全已成为 CEO 的问题。

27.5.8 组织风险管理不善

组织问题不仅是系统失败的促成因素,如组织记忆的丧失和监控不断变化的环境的机制的缺乏。它们通常是主要原因。关于人们在组织中的行为方式的文献很多,我在第 8.6.7 节中谈到了这一点,并且我在不同的章节中给出了许多进一步的例子。

然而,随着项目变得越来越大,组织因素的重要性也会增加。贝佐斯定律说,你不能运行一个开发项目,其人数不能超过两个比萨饼所能养活的人数。一个 8 人的团队几乎是可以管理的,但是同时拥有 6 个这样的团队并不能使速度提高 6 倍。如果一个项目涉及多个团队,成员之间不能随意交谈,或者你

变得混乱;由于没有带宽,他们无法通过最低的公共管理器路由所有通信。随着规模的扩大,协调将开始涉及中层管理人员、员工部门和委员会的激增。一个干净的军事指挥链的通信复杂性,对于没有横向交互的 N 个人来说,是 log N;每个人都必须咨询其他人的地方,就是N2;并且任何子集都可以组成一个委员会来考虑这个问题,它可以走向2N 。商学院的人对此写了大量文章,他们的方法通常基于案例研究。

许多大型开发项目已经崩溃和燃烧。无论灾难是安全问题、安全问题还是软件根本无法运行,问题似乎都大同小异;因此,安全人员可以从研究一般工程文献中记录的项目失败中学到很多东西。

Bill Curtis、Herb Krasner 和 Neil Iscoe [504] 撰写了大型软件项目灾难的经典研究。他们发现,未能理解需求是主要原因:应用领域知识的分散通常会导致波动和冲突的需求,进而导致沟通中断。我在本科生讲座中举的例子是伦敦救护车服务的一个新调度系统崩溃,这个项目过于雄心勃勃,规范不充分,没有真正的测试,导致该市一天没有救护车。

这样的例子太多了;我使用伦敦救护车服务案例是因为随后的调查很好地记录了原因 [1805]。

那天我也正好在伦敦,所以我记得。如果你还没有读过调查报告,我建议你读一读。 （事实上我强烈建议你阅读大量项目失败的案例研究。）

千年虫提供了另一个有用的数据点。如果人们接受许多大型商业和政府系统需要大量的维修工作才能将两位数的日期更改为四位数的日期以为 2000 年做准备,并且传统经验认为很大一部分大型开发项目延迟交付或从未交付总之,许多人自然而然地认为大量系统将在 1999 年底失效,并预测会出现大范围的混乱。但这并没有发生。当然,中小企业所用系统的风险被夸大了;我们对大学的所有系统进行了彻底检查,发现没有什么不能很容易修复的[69]。尽管如此,一些运营对经济至关重要的大公司（例如银行和公用事业公司)的系统确实需要大量维修。然而,没有关于严重后果失败的报告。这似乎支持 Curtis、Krasner 和 Iscoe 的论点。 Y2K bug 修复的要求是众所周知的:“我希望这个系统继续工作,就像现在一样,一直到 2000 年及以后”。

这就是我选择引用 Rick Smith 的话来作为本章开头的原因之一:“我自己的经验是,拥有一套简洁、表达明确的特定安全要求的开发人员可以构建非常紧凑的机器。他们不必是安全专家,但他们必须了解他们正在尝试构建什么以及它应该如何工作。”

组织很难处理不确定性,因为它妨碍了

设定目标并计划实现这些目标。因此,有能力的团队首先解决难题,以减少不确定性;这是 DARPA 的使命,也是螺旋模型的核心。有大量关于如何管理项目中的不确定性的商学院文献 [1179]。但即使在定义相当明确的项目中,也很容易出错。面对一个难题,人们通常会疯狂地攻击一个相关但更容易的问题;我们已经看到了很多例子,比如在 26.2.8 节中。

在问题开放式的安全领域,风险管理可能更糟。我们真的不知道下一场狗屎风暴会从哪里来。在 20 世纪 90 年代后期,我们认为我们已经有了安全的智能卡;然后是差分功率分析。在 2010 年代中期,我们认为我们有足够的 CPU 供竞争对手公司在亚马逊数据中心的同一台机器上运行他们的工作负载;然后是Spectre。我们还曾经认为 Apple 产品不会感染恶意软件,而且人脸识别技术永远不足以成为真正的隐私威胁。尽管摩尔定律正在放缓,但还会有更多的惊喜。

中层管理人员更喜欢他们可以通过在清单上打勾来实施的方法,但是要处理不确定性和开放式风险,您需要一个开放式学习的过程,让人们注意警报、欺诈或安全事故或客户投诉 任何你可以从中学到的东西。但是清单需要较少的管理注意力和精力 ,质量官僚喜欢它们。我在第 9.6.6 节中指出,经过认证的流程有很强的取代批判性思维的趋势;设计人员无需不断审查系统的保护要求,只需找到他们的清单即可。结果往往适得其反。如果不首先解决困难的问题,您就会隐藏不确定性,之后情况会更糟 6。此外,人们很快就学会了如何玩弄清单。我们的安全专家告诉公司付钱给聪明人以预测可能出现的问题,与董事会告诉管理人员使用更少、更便宜的工程师更快地交付产品之间存在着永恒的紧张关系。

当威胁模型具有政治敏感性时,事情就会变得更加复杂。
典型的问题是攻击是来自内部还是外部。内部人员往往是最大的安全风险,无论是因为他们中的一些人是恶意的还是因为他们中的大多数人粗心大意。但是你不能只训练你所有的员工,让他们对彼此和对客户都毫无帮助,除非你可能是政府部门或其他垄断企业。你必须找到控制的最佳点,这通常意味着要弄清楚如何将其嵌入到文化中。例如,银行经理知道双控安全锁可以降低家人被扣为人质的风险,大额交易需要两人签名意味着在出现问题时需要额外的肩膀来承担重担。

在组织中建立正确的风险生态系统需要微妙和坚持。控制和其他保护措施的文化嵌入是艰苦的工作;如果您接触多家公司,那么观察他们如何管理从代码审计(技术专业人士坚持)到尾随(半导体公司竭力防止)的所有规则以及人们是否期望一只手放在一个

6我将在下一章讨论 ISO 27001。目前的执行摘要是,几乎每家遭受大数据泄露的公司都获得了 ISO 27001 认证,但它失败了,因为他们的审计员说有些事情是可以的,但事实并非如此。

他们上下楼梯时的栏杆（能源公司的最爱）。
这些风险文化从何而来，它们是如何被推广的，为什么它们会按部门聚集？正如我们在第 12.2.6.3 节中讨论的那样，他们的交易内部控制结构可能会受到审计师的严重影响，但更广泛的安全文化差异很大 而且很重要。

另一个因素是，优秀的 CISO 几乎和母鸡的牙齿一样稀有。顶级科技和金融科技公司中有一些明星，但担任 CISO 可能是一份吃力不讨好的工作。优秀的工程师通常不想要它，或者不具备应对的人际交往能力，而雄心勃勃的经理往往会逃避这份工作。在许多组织中，晋升取决于资历和人脉；因此，如果你想成为首席执行官，你将不得不花费 20 年的时间在不冒犯太多人的情况下爬上等级。成为 CISO 意味着总是对人们说不，而没有技术背景的通才无论如何也无法破解它。这份工作也会带来很大的压力，以及倦怠的风险； CISO 的平均任期约为两年 [430]。无论如何，围绕风险和安全嵌入适当的文化是 CEO 和董事会的事。如果他们认为这不重要，那么 CISO 就没有机会。但现在违规行为已导致足够多的 CEO 被解雇，或损失数百万美元的股票，该部落的其他成员开始关注。

风险生态系统可能被扭曲的一种方式是，如果一家公司设法安排事情，以便将其运营的系统的一些风险转嫁给第三方。这消除了照顾的动机，从而产生了道德风险。我们在第 12.5.2 节中讨论了银行试图将支付系统中的欺诈责任转移给持卡人、商户或两者的情况。

如果员工知道客户的投诉将被忽略，他们可能会变得懒惰甚至不诚实 。另一个例子是亨利·福特，他认为如果你被他的车撞伤了，你应该起诉司机，而不是他；法院和立法者花了几十年时间才确定产品责任。

公司也可能从冒险者转变为过于规避风险，然后又变回来。关键管理人员的个性确实很重要。我所在的大学在聘请工程师担任校长时表现得非常积极，在聘请律师时表现得胆怯，在聘请医生时表现得居中。

问题的另一个来源是系统设计决策是由不太可能对它们负责的人做出的。发生这种情况的原因有很多。 IT 人员流动率可能很高，很大程度上依赖于合同工 ；对裁员的恐惧会把忠诚的员工变成偷偷摸摸的求职者。这在大型公共部门 IT 项目中可能是一个特殊的问题：参与其中的部长或公务员都不希望在七年后项目交付时在场。因此，在处理大型系统项目时，不要忘记环顾四周，问问自己，当出现问题时，谁来承担责任。

另一个是，在聘请安全或安全顾问来帮助进行产品设计时，公司有动力选择“足够好”但不会要求太高的公司；来自四大公司的温和审查将比来自专家的详细审查有用得多，专家可能会建议更昂贵的设计更改。事实上，如果一家公司决心获得完全安全的产品，那么他们应该聘请多名专家。我们描述了

在第 14.2.3 节中,这如何帮助设计预付费电表,后来对学生进行的实验证实,你考虑拟议系统设计的人越多,他们发现的潜在危险和漏洞就越多 [68] .当然,这种情况很少发生。

27.6 管理团队

要开发安全可靠的代码,您需要建立一支具有正确文化、正确技能组合和正确激励的团队。

许多现代系统已经非常复杂,以至于很少有开发人员能够处理它们的所有方面。那么如何建立具有互补技能的强大开发团队呢?五十多年来,这一直是一个激烈争论的话题,不同的作者反映了他们的个人风格或公司文化。长期以来,它一直与多样性等文化问题纠缠在一起,尽管这些问题自 2010 年代中期以来才受到重视。

27.6.1 精英工程师

回到 1960 年代,弗雷德·布鲁克斯 (Fred Brooks) 的名著《人月神话》描述了从开发世界上第一个大型软件产品 IBM S/360 大型机操作系统 [328] 中吸取的经验教训。

他描述了“首席程序员团队”,这是他的同事 Harlan Mills 提出的一个概念,其中一名首席程序员 开发主管,用今天的语言来说 由一名工具匠、一名测试员和一名语言律师提供支持。当时的想法是,有些程序员比其他程序员更有生产力,因此与其将他们提拔到管理层并“失去”他们,不如为他们创建职位,并获得高级经理的薪水和尊重。在 1960 年代到 1980 年代的其他科技公司,甚至在我在 1980 年代后期工作的银行 IT 部门,也发现了同样的方法。

微软、谷歌和 Facebook 等更现代的公司所持的观点是,您首先只想雇用超高生产力的工程师 尤其是如果您每年收到 100 万份简历,但计划只雇用 20,000 名新工程师。一种方法是雇用人员作为承包商几个月,看看他们的表现如何;但这对刚毕业的学生来说更难,因为即使是来自精英学校的聪明学生也可能需要几个月的时间才能在商业团队中变得富有成效。生产力也是一个文化问题。在一家公司茁壮成长的工程师在另一家公司的表现可能要差得多。一个相关的问题是,如果你让每位候选人都接受多名工程师的面试,这不仅会浪费工程师的时间,还会延续一种不太欢迎女性工程师的文化。精英大学的情况和理工科差不多,每个名额都有几十个申请者;多年来,我们已经学会了建立机制来监控招聘和录取的多样性。

这两种方法并不冲突。现代科技公司聘请了多位科技巨星,从国际知名设计师到获得图灵奖的计算机科学家。其中一家公司的观点是,如果你没有程序员的职业结构,你就不能指望编写出好的软件。人谁

想要终生编写软件,并且擅长于此,必须得到尊重,但是您的组织发出了这样的信号 无论是薪水、奖金、股票还是诸如进入行政餐厅之类的装饰品。大学明白这一点;我们教授经营这个地方。科技公司也明白这一点,一两家银行已经开始这样做了。但政府通常令人震惊。在英国公务员制度中,座右铭是“科学家应该随叫随到,但不要高高在上”。我所知道的不止一家汽车公司在招聘和留住优秀的软件工程师方面存在实际问题。其中一个,软件工程师有望在五年后成为经理,或者保持初级工资等级,而在另一个,所有工程师都应该穿着西装上班(而且薪水仍然很低)。我将在 27.6.6 节中回到这个问题。

27.6.2 多样性

在计算机出现之初,有很多女性程序员 直到 1960 年代后期,她们仍占多数,其中包括 Grace Hopper 和 Dame Stephanie Shirley(她以“Steve Shirley”的身份经营她的公司多年)等先驱。当我在 1970 年代初开始工作时,性别平衡仍然比今天好得多。也有少数民族。水星、双子座和阿波罗任务的轨道计算是由一位非裔美国女性凯瑟琳·约翰逊领导的。但在美国和英国,事情已经变成了男性主导。自从我在 1990 年代成为一名学者以来,尽管为招收更多女学生做出了巨大努力,但大约六分之一的本地计算机科学专业学生是女性。然而,在东欧的前共产主义国家,这一比例约为三分之一。 (我们通过招收大量来自南欧和东欧的学生改善了性别平衡。)在印度,性别接近平衡。所以这是一个文化问题,关于它是如何产生的有很多争论。是缺乏榜样,还是学校职业顾问的错,还是许多 IT 商店只是女性不愉快的工作环境?这当然是一个问题:我在第 2.5.1 节中讨论的 Gamergate 丑闻暴露了一些游戏社区中深深的厌女症,而#MeToo 运动凸显了硅谷的许多性别歧视案例。

即使在计算机科学中,我们也看到很多亚文化差异。上次我去参加硬件会议 一个 Arm 开发者活动我看到了大约 500 名男性,但只有三名女性(都是印度人)。在安全领域,在 1990 年代,当重点是密码学和操作系统内部时,我们绝大多数是男性,但现在我们已经接受了设计、可用性和心理学的重要性,因此更加平衡。榜样和历史确实很重要。有女性教员的研究小组从有能力的女性那里得到了更多的申请7。

更多样化的团队更有效,真正的改变不是你雇用的第一位女性,而是当你有足够多的人来改变团队文化时。这可能意味着三个或更多。这也意味着获得更多开明的管理者。显然,雇用厌恶女性的恶霸是个坏主意,尽管很难提前发现他们。更巧妙的是,如果你想吸引更多的女性

7从 1960 年代开始,我们的自然语言处理小组就实现了性别平衡由已故的凯伦斯帕克琼斯。

并留住他们,管理人员而不是工作可能是一个想法。
你必须保护你的员工 ,给他们空间去做他们擅长的事情。欺凌者也常常是小兵;除了指挥下属,他们还奉承上级。很多时候,这些人不了解技术上发生的事情,因此他们不知道谁是富有成效的,并且不得不通过计时或他们对自己的讨好程度来判断人。如果这种管理风格在整个组织中传播,我的建议是去其他地方。

27.6.3 培养技能和态度

现代开发在保持团队在一起的愿望之间存在紧张关系,以便他们变得更有效率和可预测性,并调动人们来发展他们的技能,阻止他们变得陈旧,并确保不止一个人能够维护一切重要的事情.

您还需要多种技能。例如,如果您正在编写一个应用程序,您可能需要几个人编写 Android 代码,一个人编写 Apple 代码,另一个人编写服务器代码。根据任务的不同,可能会有一位用户倡导者领导可用性测试;提倡安全或安保;建筑师,其工作是保持整体设计简洁高效;一个担心 API 的语言律师,一个运行回归测试机器的测试工程师和一个维护静态和动态分析工具的工具匠。

如果您正在进行持续集成,您将拥有一名专门从事 A/B 测试的工程师,而如果您采用门控方法,则测试重点可能是与第三方产品或安全认证的兼容性。您需要考虑一下您尝试在每个开发人员中获得多少这些技能,以及有多少是跨团队工作或作为顾问进来的主题专家。由于您不能让一个项目的人多于两个比萨饼所能养活的人数,因此您希望您的一些人拥有两种或更多这些技能。优秀的科技公司在公司内部缓慢轮换工程师,以获得一系列技能,使他们对公司的价值最大化(即使这也使他们对其他人的价值最大化,并使他们更容易离开)[1209]。

但只有技能是不够的:你需要让人们一起工作。多年来,这里的工作实践也发生了变化。到 2010 年左右,敏捷开发人员采用了“scrum”,整个开发团队每天召开五分钟的站立会议,唯一允许发言的人是开发人员。他们描述了他们已经做了什么,他们将要做什么以及存在什么问题。一些公司已将团队转移到 Jira 等协作工具。在我们的团队中,我们将每日午餐与每周一次的正式进度会议结合起来。 (自冠状病毒封锁以来,正式会议变得更加重要,我们已努力通过其他在线活动来补充它。)

如果发现错误(甚至是他们自己编写的错误)的人只是悄悄地修复它们,这是一种不好的做法;由于错误是相关的,因此可能会有更多错误。
错误跟踪很重要,能够保存良好统计数据的票务系统是提高质量的重要工具。作为良好实践的一个例子,在空中交通管制中,预计出现错误的管制员不仅应该修复它,而且应该通过公开抗议立即宣布错误:“我有 Speedbird 123

误在航站楼控制区八零高度,奉命下降至六零。”这样任何其他具有潜在冲突 trac 的控制器都可以注意到、喊出并协调。软件没有那么引人注目,但也没什么不同:您需要让您的开发人员乐于分享他们的经验,包括他们的错误。

团队建设的另一个因素是采用标准风格。管理不善的团队的一个信号是代码库是各种风格的混乱混合体,每个人都在做自己的事情。当程序员签出一些代码来处理它时,他们可能会花半个小时对其进行格式化并将其调整为自己的风格。除了浪费时间之外,重新格式化的代码还会使您的分析工具出错。您还需要在代码中添加注释,因为人们阅读代码的时间通常多于编写代码的时间。您想知道编写漏洞的程序员认为他们在做什么:是设计错误还是编码失误?但是团队很容易就“正确”的评论数量和风格争论不休。因此,当您开始一个项目时,让每个人都坐下来,让他们花一个下午的时间来敲定您的房屋风格。只要足以供以后阅读代码和理解错误,那么风格是什么并不重要:但重要的是人们接受并符合目的的一致风格。创造这种风格是一种比下午玩彩弹射击或任何最新的企业团队建设时尚更好的团队建设活动。

27.6.4 涌现性质

一个争论是你是让每个人都负责保护他们自己的代码,还是有一个每个人都依赖的安全专家。同样的问题也适用于航空电子设备等领域的安全。正如领先公司所发现的那样,答案是“两者兼而有之”。我们已经注意到,微软发现让开发人员负责改进他们自己的设计和修复他们自己的错误更有效,而不是像 IBM 在上个世纪所做的那样,将这些职能分配给分析师、程序员和测试人员。微软和谷歌现在都让新手工程师参加安全“新兵训练营”,这样每个人都了解基础知识,并且在多个级别都有主题专家。这些人的范围从拥有硕士学位或同等内部资格的在职安全顾问,到拥有密码学或虚拟化复杂细节的博士学位的人。

诀窍在于管理团队中的专业化程度,以及专家(例如安全架构师和测试专家)与其他开发人员互动的方式。

27.6.5 改进您的工作流程

您还需要仔细考虑您将使用的工具。专业的开发团队通过使用适当的工具避免了本书中描述的大量问题。您可以通过使用现代语言(例如 Rust)来避免缓冲区溢出,或者如果您必须使用 C 或 C++,则制定严格的编码约定并使用静态分析工具(例如 SonarQube 和 Coverity)强制执行它们。

通过使用维护良好的库,您可以避免加密问题,例如定时攻击和弱随机数生成器。但是您需要了解您的工具的局限性。例如,就 Coverity 而言,它的作者解释说,虽然从项目开始就使用它很好,但在中途采用它会带来实际成本,因为你突然有 20,000 多个错误报告需要分类,并且你的发布日期滑了几个月 [235]。静态分析工具的改进,比如为了应对新型攻击,也会在现有代码库中引发大量警报。对于加密库,我们在第 6 章中讨论了它们如何倾向于提供弱操作模式,例如 ECB 作为默认值,因此您需要确保您的团队改用 GCM。 （加密货币是您需要与主题专家交谈的领域之一。）

您将不断添加新工具,无论是为了在更新您的网站时避免跨站点脚本漏洞和 SQL 注入,还是为了确保您的客户数据不会在 S3 存储桶中保持全球可读。如果您不关注安全新闻,您可能不知道最新的漏洞利用和攻击,因此您可能没有意识到什么时候您必须增长自己的专业知识或购买它。但是您不能只是购买所有东西;安全行业有许多不道德的运营商,他们利用无知的客户。您需要了解您需要购买什么以及为什么,然后您需要将其与您现有的工具集成,否则您的安全操作人员将花费更多的时间将 IP 地址从一种工具复制到另一种工具。执行一些您自己的自动化操作有助于增强您的员工 并节省时间。

您的工具和库必须支持您的架构。这里的一件关键事情是您需要能够安全地发展 API。一个系统的架构比其他任何东西都更多地是由它的接口定义的,它会因一千个小切口而衰减:程序员需要一个文件处理例程,该例程使用比现有参数多两个参数,因此他编写了一个新例程 这可能本身是危险的,或者可能只会增加复杂性,从而间接导致最终失败。在理想情况下,您会依靠编程语言使用类型安全机制来防止 API 问题。

但是依赖项的跨系统扇出对安全 API 是一个真正的危害。
我们在 20.5 节中看到了加密硬件安全模块的 API 如何扩展以支持数百家银行的遗留 ATM 系统,直到我们突然意识到由此产生的功能交互使它们完全不安全。许多其他应用领域也存在类似的紧张局势,从一百多种不同型号手机中使用的手机基带软件,到一百多种不同汽车中使用的车辆组件。必须有更好的方法来管理它;我预计具有高扇出的应用程序将朝着微服务架构的方向发展,该架构具有用于不同调用应用程序的通用核心和可插拔代理。

27.6.6 最后……

您还需要了解如何管理人员,而人力资源部门无法为您做到这一点8。技术管理不能由多面手来完成

8HR 的主要工作是损害限制 阻止离职者起诉你。

他们不太可能赢得 sta 9 的信任。过于内向而无法吸引和激励他人的工程师也无法很好地完成这项工作。太多的经理去这份工作并不是因为他们认为自己可能擅长这份工作,而是因为这是获得体面薪水的唯一途径。成功的技术经理必须热爱并理解技术;他们还必须爱人和理解人。

对于您的明星工程师,您需要创造其他领导角色。他们可能是在研发实验室中最高产的创新者。他们可能是您机构记忆的守护者:了解您产品背后三十年历史的老前辈,可以阻止人们重蹈覆辙。他们可以为您的工程人员提供道德领导 ,并为您的客户提供保证。他们可以帮助吸引想要与他们一起工作的聪明的年轻新员工。但我觉得关键在于:你的公司拥有一项或多项工程专业。它们的形状是什么?谁领导他们?

他们与您的竞争对手相比如何?你如何成长和发展它们?如果你突然意识到你必须统一公司的安全工程和安全工程专业,谁来做这件事,以及如何做?

27.7 总结

管理项目以构建或增强必须满足安全性、安全性或两者的关键要求的系统是一个难题。随着越来越多的设备获得 CPU 和通信,我们需要构建能够真正工作的东西,同时防止任何可能使它们成为攻击目标的漏洞。换句话说,您需要软件安全性 – 以及其他功能和其他紧急属性,例如安全性和实时性能。

如果您正在构建全新的东西,或者对现有系统进行主要功能增强,那么了解需求通常是该过程中最困难的部分。更温和的系统演进可能涉及对需求的更微妙的更改。可以从外部强制进行较大的更改;成功并流行的系统可能会受到攻击。

由于这种动态上下文,编写安全代码很困难:第一个问题是弄清楚您要做什么。然而,即使给出了严格的规范,或者来自黑客攻击您产品的人的持续反馈,您也不会感到无聊。在雇用合适的人、为他们提供合适的工具、帮助他们开发正确的工作方式、以正确的方式为他们提供专业知识支持以及最重要的是创造一个让他们工作的环境来改善他们的工作方面,存在许多挑战。安全能力。

9作为一个数学怪人,我总是倾向于看到 MBA 类型和其他企业政客,就像罗切斯特伯爵看到国王查理二世一样:他从不说愚话,也不说智慧话。”

研究问题

本章讨论的问题是我们领域中最难和最重要的问题之一。然而,由于它们处于软件工程、应用心理学、经济学和管理学的边界,因此很少受到关注。这些接口中的每一个都可能是一个富有成效的研究领域。

安全经济学和安全心理学在过去几年取得了长足的进步,我们现在知道我们需要做更多的工作来让开发人员更容易使用安全工具。一个合乎逻辑的下一步是将我们所知道的与安全经济学和安全可用性相结合。

然而,许多失败是由于组织行为造成的。每一位经验丰富的开发人员或安全顾问都有关于公司的不当激励、有毒文化、高员工流动率、无能管理以及我们在呆伯特卡通片中看到的所有其他事情的恐怖故事。如果有人要收集一个由组织中不令人满意的激励措施引起的安全失败案例库,例如 [876, ?],这可能会很有用。如果有了良好的经验基础,接下来会发生什么?

已故的 Jack Hirshleifer 认为,我们应该尝试设计一种组织,让管理者被迫从错误中吸取教训:我们如何才能做到这一点?您如何建立制度结构来监控威胁环境的变化,并将它们不仅用于系统开发,还用于内部控制等支持活动?也许我们需要像代码管理这样的东西?在员工以适当的谨慎水平行事的意义上,您如何设计一个“安全激励相容”的组织?组织的文化人类学会说些什么?我们在上一章中看到,政府对基地组织构成的明显新威胁的反应在许多方面是如何适应不良的:我们太多的社会复原力预算花在了反恐行动上,而牺牲了其他方面的准备工作流行病等社会风险。同样,合规性、安全战区和安全战区占用了太多典型公司的弹性预算。因此,太多的安全开发工作都是为了合规性,而不是适当地管理安全和安全风险。我们如何设计反馈机制,使我们能够将适量的精力投入到正确的地方?或者我们是否需要更广泛的结构性变革,例如拆分四大会计师事务所?

进一步阅读

管理信息系统的开发有大量的、分散的和多学科的文献。每个人都应该阅读一些经典著作,例如 Fred Brooks 的“Mythical Man Month”[328] 和 Nancy Leveson 的“Safeware”[1149]。

Brooks 和 Barry Boehm [272] 讨论了软件生命周期的经济学。截至 2020 年,每个人都应该阅读的现代书籍可能是关于 SRE [236] 和“构建安全可靠的系统”[23] 的 Google 书籍。 Microsoft 的安全开发生命周期方法有许多在线资源； Frank Swiderski 和 Window Snyder [1851] 讨论了他们关于威胁建模的学说;和他们的安全副总裁 Mike Nash de

在 [1385] 描述了大安全推动和采用安全开发生命周期的背景。关于安全功能和完整性要求以及相关工程流程的最通用标准集是 IEC 61508;还有更多的行业特定标准。例如,有用于加工厂控制系统的 IEC 61511、用于机械安全的 IEC 62061 以及用于铁路的 EN 5012x 系列。在航空领域,它是用于电子硬件的 RTCA DO-254 和用于软件的 RTCA DO-178C,而在汽车行业,它是用于安全的 ISO 26262 和用于安全的 ISO 21434 尽管在撰写本文时,这仍然只是一个草案。物联网标准也在进行中,目前的草案是 ETSI EN 303 645 V2.1。

我们可以从其他工程学科中学到很多东西。 Henry Petroski 讨论了桥梁建造的历史、桥梁为什么会倒塌,以及土木工程师如何从倒塌中吸取教训:往往会发生的是,一个既定的设计范例被拉伸和拉伸,直到它突然因为一些不可预见的原因而失效 [1518]。 IT项目失败是另一个必要的研究课题; Christoph Loch、Arnoud DeMeyer 和 Michael Pich 有一本关于如何管理项目不确定性的案例集 [1179]。对于安全故障,重要的是关注领先的安全博客,例如 Schneier on Security、Krebs on Security 和 SANS,以及商业媒体。

商学院的文献对组织方面进行了详细讨论,但这可能会让局外人感到困惑。许多商业学者称赞商业,这对销售机场书籍来说很好,但我们需要的是对组织如何失败的更批判性的理解。如果你只想读一本书,那就是 Lewis Pinault 的“咨询恶魔” 一位前内部人士关于大型咨询公司如何掠夺他们的客户的忏悔 [1527]。查尔斯·汉迪 (Charles Handy) 等组织理论家谈到公司的文化基于权力、角色、任务或人员,或某些组合。这不仅仅是谁可以接触到谁,而是谁准备好听谁的,谁会忽略谁的命令。也许这样的见解可以帮助我们设计更有效的工具和工作流程,以支持人们实际工作的最佳方式。