第 6 章

高级系统运维

数据是企业或者组织的核心资产，一些新崛起的互联网科技公司，例如，抖音、阿里巴巴、美团拥有大量的用户数据，对这些数据的分析与挖掘的价值甚至已经超过了各自公司的主营业务价值。在享受大数据分析便利和效果的同时，如何注意对安全的管控，如何保护核心资产的保密性、完整性、可用性也逐渐成为越来越重要的工作事项。同时越来越多的企业运维人员开始把系统服务的稳定性放在运维工作的首要位置，而大数据系统在这些系统中所表现的稳定需求也尤为突出。如何保持一个系统可以连续性地对外提供服务，减少系统的故障情况，从而增加实际使用过程中的用户体验度、运维的高可用（high availability），已经成为系统从设计到实施乃至后期维护中都要着重考虑的内容。

本章将从系统运维的视角出发，展开描述高级系统运维中的安全管理、系统优化，以及系统的高可用架构等相关概念，并从实践的角度扩充介绍相关的技术实践方案和优化方案。

6.1 安全管理

安全管理的主要目标是保障系统的安全和稳定运行，以及资产的保密性、完整性和可用性。

（1）保密性是指对数据的访问控制，只有被授权的用户才能允许使用。

（2）完整性指的是保证数据没有在未经授权的方式下改变。

（3）可用性是指在系统服务时间内，确保服务的连续可用。

在 ISO 中，信息安全的定义是在技术上和管理上为数据处理系统建立的安全保护，保护计算机硬件、软件和数据不因偶然和恶意的原因而遭到破坏、更改和泄漏。

从互联网诞生以来，有关信息安全的问题一直呈现上升态势，而随着近年来相关

软件和硬件技术的进一步发展、相关安全管理制度和技术的逐渐完善，代码扫描和漏洞检测工具的日趋成熟，系统安全已经逐步步入一个新的高度；然而安全的风险和威胁并没有随着体系的日益成熟而消除，创建和维护一个相对安全的系统，仍然是每一个行业从业者不得不考虑的问题。如图 6-1 所示为目前各漏洞类型的占比情况。

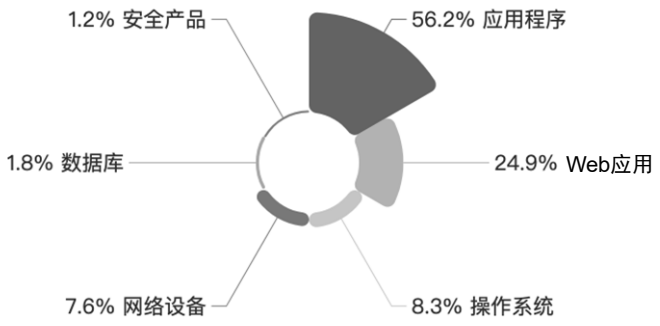

图 6-1　信息安全漏洞占比

6.1.1　资产安全

1．环境设施安全

环境的区分可以分为服务器机房环境和日常终端办公环境，然而无论是其中何种环境，都需要遵循以下的规则去进行权限细化。细化的过程首先是对环境最小颗粒度的拆分，按照功能把每个环境拆分成多个很小的功能区域，每个区域设置对应的门禁措施，并为具备相关权限的开发者或管理人员配置进出权限。

当前应用比较广泛的门禁系统主要分为卡片式、密码式、生物特征式和混合式。卡片式的门禁系统，进入人员需要持卡刷卡进出权限环境；密码式门禁系统，人员凭借提前配置好的密码进出权限环境；生物特征式的门禁系统，就如字面意思，人员可以通过生物相关的唯一性特征进出权限环境，这类生物相关特征目前比较多的有指纹、虹膜、面部识别等；混合式的门禁系统可能会采取以上所列举的一种或多种认证方式进行组合认证进入场所。对于非组织内部的相关人员，则要求制度规范拥有一套对应的登记机制流程，确保外来人员可以在组织监控下进出相关关键场所。

为了保护重要的电子设备和数据资源，机房防火系统一般都会安装延误探测器，在起火产生明火前发现火警，并且在火势进一步扩大前进行电源截断，使用灭火设备手动灭火。一般在数据中心或者机房场景中，电子元器件会在遇水后发生故障，因此会采用气体灭火系统。该系统可以将具有灭火能力的气态化合物通过自动或者手动的方式释放到火灾发生区域。其主要使用的气态化合物有二氧化碳、七氟丙烷、三氟甲烷、烟烙烬等。另外需要注意的是，数据中心还应该安装适当的防火墙，这样可以使火源控制在局部范围内而不会进一步扩大，从而把火灾的损失降到最低。除了防火系

统外，防水、防雷、防鼠患等措施也需要被考虑到环境防护因素中。

视频监控也是一个常用的安全管控手段，在关键的通道、入口处安装相应监控设备，通过实时监控的方式获取对应环境的实际情况，并根据存储容量，及时归档监控视频的内容，方便后期随时查询。监控的内容除了图像内容之外，还应包括环境的温度、湿度、电力工作情况等。

2．设备安全

常见的设备管理措施首先是对所有设备的统计登记和编号；然后是在设备发生变化时及时去进行设备信息的维护，变化场景主要有设备的购入、报修、报废、迁移、升级调整等；另外每年需要重新对所有设备信息进行定期的审计复核，确保数据的准确。目前，已经有二维码或者 RFID 内置的标签，可以粘贴在各种设备的物理表面，也有自动化的物理机架可以直接配合对应的设备管理系统对物理硬件设备进行信息化管理。

6.1.2 应用安全

1．技术安全

1）安全漏洞

随着软件技术的发展，对系统、网络、物理方面应用层的入侵手段逐步增多，而入侵的门槛也同步变低，同时应用由于自身需求的不断变化而快速迭代，来自应用层的攻击问题凸显出来。OWASP（开放式 Web 应用程序安全项目）根据攻击向量、技术影响、漏洞可检测性、漏洞普遍性几个维度的评估，列出了十大 Web 应用漏洞，如表 6-1 所示。

表 6-1 OWASP 十大 Web 安全漏洞

漏　洞	概　述
注入	注入攻击漏洞，例如，SQL、OS 以及 LDAP（轻型目录访问协议）注入。这些攻击发生在当不可信的数据作为命令或者查询语句的一部分，被发送给解释器时。攻击者发送的恶意数据可以欺骗解释器，以执行计划外的命令或者在未被恰当授权时访问数据
失效的身份认证和会话管理	与身份认证和会话管理相关的应用程序功能往往得不到正确的实现，这就导致了攻击者破坏密码、密钥、会话令牌，或攻击其他的漏洞去冒充其他用户的身份（暂时或永久的）
跨站脚本（XSS）	当应用程序收到含有不可信的数据，在没有进行适当的验证和转义的情况下，就将它发送给一个网页浏览器，或者使用可以创建 JavaScript 脚本的浏览器 API，利用用户提供的数据更新现有网页，这就会产生跨站脚本攻击。XSS 允许攻击者在受害者的浏览器上执行脚本，从而劫持用户会话、危害网站或者将用户重定向到恶意网站

续表

漏　洞	概　述
失效的访问控制	对于通过认证的用户所能够执行的操作，缺乏有效的限制。攻击者会利用这些缺陷来访问未经授权的功能和/或数据，例如，访问其他用户的账户、查看敏感文件、修改其他用户的数据、更改访问权限等
安全配置错误	由于许多设置的默认值并不是安全的，因此，必须定义、实施和维护这些设置。此外，所有的软件应该保持及时更新
敏感信息泄露	许多 Web 应用程序和 API 没有正确保护敏感数据，例如，财务、医疗保健和 PII。攻击者可能会窃取或篡改此类弱保护的数据，进行信用卡欺骗、身份窃取或其他犯罪行为。敏感数据应该具有额外的保护，例如，在存放或在传输过程中的加密，以及与浏览器交换时进行特殊的预防措施
攻击检测与防护不足	大多数应用和 API 缺乏检测、预防和响应手动或自动化攻击的能力。攻击保护措施不限于基本输入验证，还应具备自动检测、记录和响应，甚至阻止攻击的能力。应用所有者还应能够快速部署安全补丁以防御攻击
跨站请求伪造（CSRF）	一个跨站请求伪造攻击迫使登录用户的浏览器将伪造的 HTTP 请求，包括受害者的会话 cookie 和所有其他自动填充的身份认证信息，发送到一个存在漏洞的 Web 应用程序
使用含有已知漏洞的组件	组件，例如，库文件、框架和其他软件模块，具有与应用程序相同的权限。如果一个带有漏洞的组件被利用，这种攻击可以促成严重的数据丢失或服务器接管。应用程序和 API 使用带有已知漏洞的组件可能会破坏应用程序的防御系统，并使一系列可能的攻击和影响成为可能
未受有效保护的 API	现代应用程序通常涉及丰富的客户端应用程序和 API，例如，浏览器和移动 App 中的 JavaScript，其与某类 API（SOAP/ XML、REST/JSON、RPC、GWT 等）连接。这些 API 通常是不受保护的，并且包含许多漏洞

2）安全开发

从应用自身角度出发，如果是应用代码本身产生的漏洞，那么在代码层加固或者编码时就解决漏洞无疑是最根本有效的方法。这就对系统的设计阶段提出了要求。

（1）设计完整的认证和授权。在设计和开发应用程序时，常常会首先定义认证和授权模块，使用认证和授权技术对使用的用户进行身份认证和权限授权。认证是对用户身份的甄别，通过对用户的登录账号与密码进行验证匹配，判定用户是否有权限进入或者获取系统相关的服务功能。为了遵循安全性和便捷性的要求，也可以通过生物标识、统一认证、客户端证书认证、动态口令复核的方式进行用户身份的认证。认证的完成其实只是授权认证的基础，只是标识用户具备了访问和登录的权限，接下来是对用户权限的查询与赋权。

由于 Web 应用中的用户众多，大部分 Web 应用权限系统的设计都会采用 RBAC（role-bases access contrl）模型，是一种基于角色进行应用环境访问的权限控制策略。

系统会在预定义配置中划分出几类用户角色的赋权，这里的角色可以理解为具备同类行为和责任范围的一组权限共同组。只要把角色赋予用户，就可以使用户具备与角色等同的授权内容，而不用特定关心是具体哪一个用户。当然用户也可以同时包含

多个角色，从而获得一个可配置的复杂角色身份。一个用户可以拥有多个角色，而一个角色也可以囊括多个使用用户。角色访问控制的优点显而易见：便于授权管理和赋权；便于按照工作和业务进行权限分级，责任独立可控；便于文件的分级管理且适合大规模实现。角色访问是一种有效而灵活的安全措施，系统管理模式明确，且可以节约管理开销。

在具体的系统设计和实现中，还有两个重点的问题：权限信息的存储和权限的校验。在权限控制模块中，需要用到和管理的信息有：系统的所有角色、系统的所有用户、系统所有的功能、系统所有的资源、用户跟角色之间的关系、角色跟功能之间的关系、角色跟资源之间的关系或者用户跟资源之间的关系。而对这些数据的缓存手段也有很多种，比较通用的就是数据库的存储，当然用 LDAP 服务器、XML 文件来存储权限也很常见。有了对权限信息的存储，用户对权限的获取就变得可行。

针对这些权限下的资源的校验，主要包含功能校验权限和数据校验权限两个方面。功能校验权限是指用户是否可以执行或者使用该项功能或者服务，而数据校验则是判定用户是否能访问某块数据区域，这两者在用户使用系统时可以说缺一不可。在进行权限的校验时也要注意对登录用户的数据缓存，减少在服务使用时频繁地进行权限查询和用户查询，这会导致服务本身之外的系统开销，影响系统的性能。要能够保证在权限校验覆盖没有问题的情况下选择更简单有效的校验方式，从而从设计层面解决类似权限请求占用系统服务开销的问题。

除了 RBAC 模型外，还有一些其他的权限控制方式去控制用户权限。例如，当数据的访问权限非常复杂时，会使用 ACL 的方式；而在一些系统中，用户的权限是随着用户的状态和上下文变化的，这时就要使用基于用户属性的权限控制方式，通过逻辑计算用户的属性来得到最后的权限信息。

（2）数据过滤。因为暴力危险输入造成的漏洞是危害性最大、影响面最广的。健壮的输入和输出过滤可以大大减少应用受攻击的风险。而 XSS 跨站攻击和 SQL 注入这两个高危风险都是由于没有完善有效的数据过滤或者数据过滤不当引起的。

数据过滤的原则覆盖了输入过滤与输出过滤，输入过滤的不严谨会导致不被期望的代码在服务器端被恶意执行从而导致系统的异常甚至是底层数据的爆破删改，而输出过滤不当则有可能在客户端被植入恶意的 HTML 或者 JavaScript 代码。

对输入的过滤可以分为两种：第一种是黑名单限制，第二种是白名单放行。顾名思义，第一种是对错误输入格式的约束，不在约束范围内的即为正确输入。第二种则是对正确输入的囊括，对不在囊括范围内的统一进行拦截和拒绝，从过滤的方式上可以很明显地看出，在理论上白名单方式要比黑名单方式更加安全，因为前者对输入的范围进行了控制。

（3）敏感信息加密。对于黑客来说，有价值的数据只有读出来才有价值，而保护有价值信息最好的技术之一就是加密。加密是将信息的编码进行杂凑，使不知道密码的人无法获知数据的意义。对于 Web 应用来说，信息的传输和存储都需要加密。在传

输层面上，可以使用 HTTPS 加密传输有密码、账户等敏感信息的 HTTP 请求或者回复；在服务器端，使用 Base64 加密算法对保存在配置文件、数据库的用户密码进行加密存储，防止密码外泄。

（4）保留审计记录。对用户访问应用中的关键操作应该予以记录，以便日后进行审计。审计记录的内容至少应包括事件日期、时间、发起者信息、类型、描述和结果等。审计的关键操作就是日志的记录。一种流行的日志 API 是 log4j 系列，而且它已经被移植到了 C、C++、C#、Perl、Python、Ruby 和 Eiffel 语言上。

3）安全测试

自动化测试工具可以自动生成输入参数，并根据反馈结果来判断系统是否存在安全漏洞，自动化测试速度快，测试用例可复用以及持久化，能够针对性地排查一些特定安全漏洞。例如，代码越界、页面注入、远程执行等。但是自动化测试工具也有其局限性，它是根据请求参数 request 所得到的返回结果 response 来提取一些特征，从而发现和识别漏洞。另外也可以尝试从代码扫描的角度出发，通过代码扫描工具去核查代码中的漏洞问题。例如，Sonar 可以直接在服务器上对代码仓库的代码进行定时扫描，也可以在开发工具上直接继承 Sonar 插件实时检查开发过程中的代码。

自动测试工具即使对部分漏洞来说也存在误报、漏报的情况，而且其漏洞案例的实时性更新依赖于人工的添加。但是由于其速度快，再结合人工检查确认的方式，可以相对客观地评估应用的安全情况。

4）运维加固

虽然通过后期手段和测试很难完全避免所有的安全漏洞，但是通过测试扫描后剩余的漏洞数量会大比例减少，而且安全漏洞本身也依赖于输入以及调用的触发。在架构环境中部署防火墙、在前端及后端定义输入规则、对恶意输入及非法输入进行屏蔽限制，也可以达到对安全漏洞的屏蔽和对恶意输入过滤的目的。

另外，对于整个架构环境中的操作系统、数据库、网络系统等，也要进行定期的扫描和漏洞库更新，及时更新或者升级相关补丁内容。

2．数据安全

1）存储安全

在 Hadoop 集群中，应用层实现了数据的多客户端存储和备份，每个实例数据都存在 3 个副本存储，任何一个副本出现问题都不会导致数据的完全丢失。如果不具备应用层的数据多点备份能力，那么就要考虑硬件层面的 RAID（磁盘阵列）。

RAID 即“独立磁盘构成的具有冗余能力的阵列”之意。磁盘阵列指由多个磁盘构建成一个巨大容量的磁盘组，利用个别磁盘提供数据所产生的加成效果提升整个磁盘的系统效能。通过这项技术，将数据切割为多个区段，分别存放在各个硬盘上。磁盘阵列还能利用同位检查（parity check）的概念，在数组中任意一个硬盘发生故障时，仍可读出数据，在数据重构时，将数据经计算后重新置入新硬盘中。RAID 技术主要

包含 RAID 0～RAID 50 等数个规范，它们的侧重点各不相同。

大数据的本身一般并不是数据的生产方，而是通过对数据的收集和分析，获得分析结果的一种手段。大数据系统的主要功能之一就是对源数据的备份，但是数据的存储终究是需要成本的，越高的存储速度意味着越高的硬件价格。目前主流大数据框架 Hadoop 的相关技术就使用了相对主流标准的硬件设备，例如 PC 服务器，从而减少了昂贵的存储硬件支出。然而需要注意的是，即使使用了较为标准和性价比的设备，存储数据的规模也不能一味扩大。在构建大数据系统的过程中，准确定位数据的规模并且使用相关方法保证数据存储不会持续扩展也是一项必要指标。例如，通过划分存储的时限使定义好的历史数据，在分析使用量不大的情况下及时归档迁移到磁带系统中。

2）传输安全

如果数据的传输经过了不安全的网络，那么使用加密和安全的协议就是必要的措施。

超文本传输协议（HTTP）是目前被用于在 Web 浏览器和网站服务器之间传递信息的主要手段之一。HTTP 以明文方式发送内容，如果攻击者截取了 Web 浏览器和网站服务器之间的传输报文，就可以直接读懂获取其中的信息内容，因此 HTTP 不适合传输一些涉敏信息。为了解决这一缺陷，另一种传输协议应运而生：安全套接字层超文本传输协议（HTTPS）。

HTTPS 在原有 HTTP 的基础上加入了 SSL 协议，为浏览器和服务器之间的通信内容进行加密，依靠 SSL 证书对服务器身份进行验证。采用 HTTPS 的服务器必须从证书授权中心（certificate authority，CA）申请一个用于证明服务器用途类型的证书。客户端通过信任该证书，从而信任该主机。

在另外一些场景中，会对数据本身进行脱敏处理，对数据中的敏感信息进行数据屏蔽或者修改，实现对敏感隐私数据的可靠保护。例如，在系统导出客户类似身份证号、手机号、卡号、客户号等个人信息时都需要进行数据脱敏操作。

3）访问安全

如前文所述，应用系统本身要建立健壮的认证和访问控制机制，从而防范数据的越权访问。但是近些年来，屡次发生的数据泄露问题，基本都是由内部人员的泄露造成的。针对这个问题，信息的追溯也变得重要起来。一方面，通过审计手段记录员工对数据的详细访问操作；另一方面，可以在数据层面加上水印，这样通过泄露的信息可以很容易确定涉事人员。

数字水印技术即通过在原始数据中嵌入秘密信息水印（watermark）来验证该数据的所有权。这种被嵌入的水印可以是一段文字、标识、序列号等，通常这种水印是不可见或不可擦的，它与源数据紧密结合在一起并隐藏其中，且可以在不破坏源数据使用场景的情况下保存下来，其原理如图 6-2、图 6-3 所示。

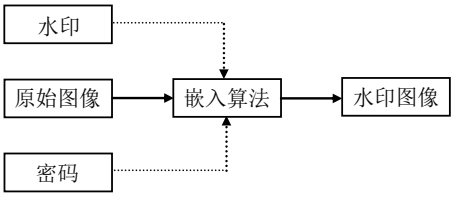

图 6-2　水印信号的嵌入

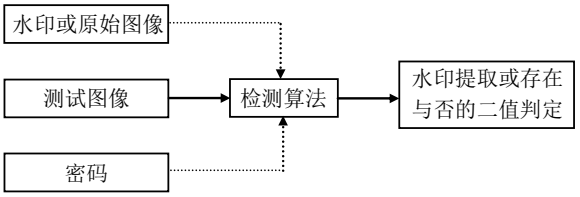

图 6-3　水印信号的验证

通过水印的设置方法，每个员工访问到的数据界面上都有一层肉眼无法看到的信息，一旦该界面被泄露出去，通过还原算法就可以从泄露书中获取到泄露人员的相关信息。

6.1.3　安全威胁

1．人为失误

人为失误（human error）是指在人的实际操作过程中，由于人本身的不稳定性所导致的错误。从人性的角度来说，只要是人的操作，就有可能存在失误。在各行各业都存在人为差错所造成的严重后果。同样，在系统运维领域，人为失误也可能造成系统服务停止、业务中断等不良影响。

大事故很少是由一个原因引起的，多是由诸多问题串联在一起同时发生所造成的。海恩法则表明在一起重大事故下有 29 起事故征候，而且在其下面还有 300 起事故征候苗头（严重差错）。虽然人为差错主要是由人自身造成的，但是论其起因，可以从人、环境、工具、流程 4 个方面进行总结，如表 6-2 所示。

表 6-2　人为失误的原因

分　类	详 细 内 容
人自身原因	（1）厌倦与疏忽。 操作人员对工作感到无聊，没有成就感，心理存在抵触情绪；操作人员对工作重要性意识不足。 （2）疲劳或者疾病。 操作人员身体处在不良状态，注意力无法正常集中，身体反应较一般情况变慢。 （3）知识或技能缺乏。 操作人员不知道，不熟悉，忘记正确的操作方法；按照自己的习惯或者设想的操作方法去操作；无法预见操作后果。 （4）过于自信。 操作人员对自己的知识能力过于自信，可能做违反流程的操作，为了快点干完省略了一些必要的步骤。例如，驾驶事故高发于有一定驾龄的司机。 （5）心理压力。 过度担心后果造成心理压力过大，精神处于亢奋紧张状态

续表

分　类	详 细 内 容
环境原因	（1）非常规事件。 突发事件，操作人员未能及时调整状态，精神处于紧张亢奋状态；对突发事件的处理可能违反常规流程，造成操作风险。 （2）外界刺激。 来自于环境的刺激较多或者更换了新环境，使操作人员无法集中注意力
工具原因	（1）人机设计不合理。 不方便操作人员使用，难以掌握；工具的一些操作本身容易混淆，无法明显区分。 （2）违反标准，或者无统一标准。 例如，一般的汽车都是刹车在左，油门在右，如果违反了这个标准，或者这个标准没有统一，则很有可能形成操作风险。 （3）工具反常。 例如工具平时的响应只需要 1 s，但是在某些情况下变成了 5 s，等待的时间间隔可能打乱了操作人员的节奏感，进而形成操作风险
流程原因	（1）流程烦琐。 操作流程步骤繁多，实施时可能产生遗漏或者错误。 （2）存在交叉作业。 流程上需要操作人员在不同工具、不同对象间切换操作。由于人思维存在惯性，或因形成的条件反射造成失误

2．外部攻击

1）恶意程序

恶意程序是未经授权运行的、怀有恶意目的、具有攻击意图或者实现恶意功能的所有软件的统称，其表现形式有很多：僵尸程序、蠕虫、黑客工具、计算机病毒、特洛伊木马程序、逻辑炸弹、漏洞利用程序、间谍软件等。大多数恶意程序具有一定程度的隐蔽性、破坏性和传播性，难以被用户发现，会造成信息系统运行不畅、用户隐私泄露等后果，严重时甚至导致重大安全事故和巨额财产损失等。

2019 年第一季度，信息安全厂商卡巴斯基公司公开透露，共阻止了全球 203 个国家中在线发生的 843 096 461 次攻击。20 年前每天可能只检测到 50 个新病毒，10 年前大概有 14 500 个，现在每天能收集几十万甚至上百万个，并且数量还在增加。

2）网络入侵

网络入侵是指根据系统所存在的漏洞和安全缺陷，通过外部对系统的硬件、软件及数据进行攻击的行为。网络攻击的手段有多种类型，通常从攻击目标出发，可以分为主机、协议、应用和信息等的攻击。

2020 年 12 月，SolarWinds 公司的基础设施遭到黑客网络攻击，该公司名为 Orion 的网络和应用监控平台的更新包被黑客植入后门，并将其命名为 SUNBURST，同时向该软件的用户发布木马化的更新，其中包括美国财富 500 强中的 425 家公司、美国前十大电信公司、美国前五大会计师事务所、美国军方所有分支机构、五角大楼，以

及全球数百所大学和学院。此次黑客攻击很可能影响到了 1.8 万名 SolarWinds 软件用户，数百名工程师受到影响。

3）拒绝服务攻击

拒绝服务攻击（DoS）即攻击者通过攻击使目标机器停止提供服务。常见的手段有通过大批量请求耗光网络带宽，使合法用户无法访问服务器资源。分布式的拒绝服务攻击手段（DDoS）是在传统的 DoS 攻击基础之上产生的一类攻击方式。

单一的 DoS 攻击一般是采用一对一方式的，当单机资源过小，CPU 速度、内存以及网络带宽等各项性能指标不高时，攻击会尤为有效。分布式的拒绝服务攻击手段（DDoS）则是通过更多的分布式主机发起对单一服务的攻击，用更大规模的攻击使主机不能正常工作。

2020 年 8 月 31 日，新西兰证券交易所网站在周一的市场交易开盘不久再次崩溃。这已是自 2020 年 8 月 25 日以来，新西兰证券交易所连续第 5 天“宕机”。2020 年 8 月 25 日，新西兰证券交易所收到分布式拒绝服务（DDoS）攻击，袭击迫使交易所暂停其现金市场交易 1 小时，严重扰乱了其债务市场。

4）社会工程

为获取信息，利用社会科学，尤其是心理学、语言学、欺诈学将其进行综合，利用人性的弱点，并以最终获得信息为最终目的的学科称为社会工程学（social engineering）。

社会工程学中比较知名的案例是网络钓鱼，通过大量来自各种知名机构的诱惑性垃圾邮件，意图引导受攻击者提供自身敏感信息的一种攻击方式。最典型的网络钓鱼攻击是将收信人引诱到一个通过精心设计与目标组织的网站非常相似的钓鱼网站上，诱使并获取收信人在此网站上输入个人敏感信息，通常这个攻击过程不会让受害者警觉。网络钓鱼网站被仿冒的大都是电子商务网站、金融机构网站、第三方在线支付站点、社区交友网站等。

5）外部攻击实例

（1）XSS 跨站脚本攻击。XSS（cross-site script，跨站脚本攻击）是一种网站应用程序的安全漏洞攻击，是代码注入的一种。它允许恶意用户将代码注入到网页中，其他用户在观看网页时就会受到影响。这类攻击通常包含 HTML 以及用户端脚本语言。

它可以分为两类：反射型和持久型。

反射型 XSS 攻击场景：用户单击嵌入恶意脚本的链接，攻击者可以获取用户的 cookie 信息或密码等重要信息进行恶性操作。

解决方法：开启 cookie 的 HttpOnly 属性，禁止 JavaScript 脚本读取 cookie 信息。

持久型 XSS 攻击场景：攻击者提交含有恶意脚本的请求（通常使用<script>标签），此脚本被保存在数据库中。用户再次浏览页面，包含恶意脚本的页面会自动执行脚本，从而达到攻击效果。这种攻击常见于论坛、博客等应用中。

解决方法：前端提交请求时，转义<为&lt，转义>为&gt；或者后台存储数据时进

行特殊字符转义。建议后台处理，因为攻击者可以绕过前端页面，直接模拟请求，提交恶意的请求数据。可以考虑在后台加入对应的数据校验，也可以考虑统一对后台的数据进行特殊字符的转义。

另外，所有的过滤、检测、限制等策略建议在服务端一侧完成，而不是使用客户端的 JavaScript 去做简单的校验。因为真正的攻击者可以绕过页面直接通过模拟页面的请求进行数据非法录入。

例如，在表单中填写类似脚本的语句，如图 6-4 所示。

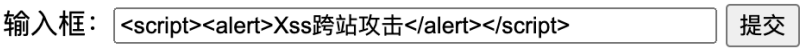

图 6-4　脚本语句录入

单击“提交”按钮后页面回显会解析 JavaScript 脚本并弹出对话框，如图 6-5 所示。

图 6-5　XSS 攻击弹出框

解决思路如下。

在页面端增加转义字符过滤，清洗输入框录入的数据，并增加页面的校验规则。

增加录入内容的正则校验。

```
var inputValue=this.value;
var regl= /^[A-Za-z]+$/;
if(regl.test(inputValue)){
      alert("输入格式正确");
      return;
}else{
  alert("输入格式不正确")
}
```

在输出数据时，能将 HTML 标记转成常用字符串形式（专门去解析 HTML 元素其实是为了防止 XSS 攻击）。

```
var replaceSpecial = function(v){
    return _.template("<\%- m \%>", { variable: "m" })(v);
};
```

后台增加对应的过滤器，用来过滤前台传递的参数数据。

```
public class XssFilter implements Filter {

      @Override
```

```
    public void destroy() {
    }
    /**
     * 过滤器用来过滤的方法
     */
    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
throws IOException, ServletException {
        //包装 request
        XssHttpServletRequestWrapper xssRequest = new
XssHttpServletRequestWrapper((HttpServletRequest) request);
        chain.doFilter(xssRequest, response);
    }
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
    }
}

public class XssHttpServletRequestWrapper extends HttpServletRequestWrapper {
    HttpServletRequest orgRequest = null;

    public XssHttpServletRequestWrapper(HttpServletRequest request) {
        super(request);
    }
    /**
     * 覆盖 getParameter()方法，将参数名和参数值都做 XSS 过滤
     * 如果需要获得原始的值，则通过 super.getParameterValues(name)来获取
     * getParameterNames,getParameterValues 和 getParameterMap 也可能需要覆盖
     */
    @Override
    public String getParameter(String name) {
        String value = super.getParameter(xssEncode(name));
        if (value != null) {
            value = xssEncode(value);
        }
        return value;
    }
    @Override
    public String[] getParameterValues(String name) {
        String[] value = super.getParameterValues(name);
        if(value != null){
            for (int i = 0; i < value.length; i++) {
                value[i] = xssEncode(value[i]);
            }
        }
        return value;
```

```
    }
    @Override
    public Map getParameterMap() {
        return super.getParameterMap();
    }
    /**
     * 将容易引起 XSS 漏洞的半角字符直接替换成全角字符，在保证不删除数据的情况下保存
     * @return 过滤后的值
     */
    private static String xssEncode(String value) {
        if (value == null || value.isEmpty()) {
            return value;
        }
        value = value.replaceAll("eval\\((.*)\\)", "");
        value = value.replaceAll("[\\\"\\\'][\\s]*javascript:(.*)[\\\"\\\']", "\"\"");
        value = value.replaceAll("(?i)<script.*?>.*?<script.*?>", "");
        value = value.replaceAll("(?i)<script.*?>.*?</script.*?>", "");
        value = value.replaceAll("(?i)<.*?javascript:.*?>.*?</.*?>", "");
        value = value.replaceAll("(?i)<.*?\\s+on.*?>.*?</.*?>", "");
        return value;
    }
}
```

在 web.xml 中增加对应的过滤器拦截。

```
<filter>
  <filter-name>XssFilter</filter-name>
  <filter-class>XXXXX(该类的路径).XssFilter</filter-class>
</filter>
<filter-mapping>
  <filter-name>XssFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

（2）SQL 注入。攻击者在 HTTP 请求中注入恶意 SQL 命令，例如 drop table users，服务器用请求参数构造数据库 SQL 命令时，恶意 SQL 被执行。

解决思路：后台处理。例如，使用预编译语句 PreparedStatement 进行预处理，如果有使用 mybatis 等持久层框架的话，建议控制相关 SQL 参数的填写格式。

（3）DDoS 攻击。DDoS 攻击又叫流量攻击，是最基本的 Web 攻击；攻击的方式有很多种，其中最常用的就是静态文件攻击。通过劫持大量的 IP 然后同时访问同一个静态文件，使服务器一直处于通信堵塞状态，网络带宽被全部占满，使网站崩溃处于不可访问状态，SSH 连接服务器处于卡顿状态，无法执行相应的命令。

解决思路如下。

查看内存是否已被占满。

```
free -m
```

如果 used 项的值接近 total 的值，说明内存快不够了，正常情况下至少保持 1 GB 左右的大小。查看进程情况。

```
top -c
```

然后按 Shift + M 组合键对进程进行排序，如果这个时候出现卡顿或者执行不连贯，应先关闭当前 SSH 连接，重新打开新的 SSH 连接窗口。

进入 Nginx 日志目录（默认安装的情况下在 /etc/Nginx/logs 路径下）。

```
cd /etc/Nginx/logs
```

查看实时日志（根据 Nginx 配置时所配置的日志文件）。

```
tail -f access.log
```

可以很清晰地看出黑客攻击的路径或点，这时就可以根据具体情况来配置了。

①停掉 Nginx 进程。

②查看 Nginx 的进程 id。

```
ps -ef | grep Nginx*
```

③用 kill -9 [pid]强制性把 Nginx 杀掉。

```
kill -9 23423
```

④修改 Nginx 配置文件（根据自己配置的所在目录）。

```
vi /etc/Nginx/conf.d/Web.conf
```

⑤找到刚刚黑客攻击的地方，代码如下。

```
location /cstor / {
  …
}
```

⑥增加防盗链，修改如下。

```
location / cstor / {
   valid_referers xxx.com;
   if ($invalid_referer) {
                return 403;
   }
   …
}
```

注：xxx.com 为用户自己所需要放开的地址，例如，用户公司的网站域名等。

3．信息泄露

信息泄露是信息安全的重大威胁，国内外都发生过大规模的信息泄露事件。

2020 年 2 月，体育连锁巨头迪卡侬（Decathlon）发生大范围数据泄露事件，起因是 1.23 亿条记录被保存在一个并不安全的数据库中。这是由 vpnMentor 安全研究人员发现的，并在 2020 年 2 月 24 日公布。该数据库属于迪卡侬西班牙和迪卡侬英国公司。泄露的数据涉及员工系统用户名、未加密的密码、API 日志、API 用户名、个人身份信息等。对于迪卡侬员工来说，涉及的信息包括姓名、地址、电话号码、生日、学历和合同明细；而对于客户来说，涉及的信息包括未加密的电子邮件、登录信息和 IP 地址。

2020 年 3 月 31 日，万豪国际集团发布公告称，正在调查一起涉及客户个人信息泄露事件，约 520 万名客户的资料可能被泄露。在不到两年的时间里，万豪发生了第二次大规模数据泄露事件；最终，因未能确保客户个人数据安全，万豪国际被罚 1840 万英镑。与此同时，接连两次的股价大跌，直接导致万豪数十亿元市值蒸发。

除了外部攻击的信息泄露外，企业的内部员工利用访问权限获取用户的相关数据，并非法在黑市上贩卖牟利。这些被贩卖的数据，会被黑客或者其他不法分子利用，借助社会工程学，对受害者进行诈骗。

4．灾害

灾害发生的概率非常小，但是后果是巨大的，可能会造成整个数据中心停止运行。

1）洪灾

由于恶劣天气和排水不畅，可能会造成水倒灌进数据中心，发生设备短路等故障。2009 年 9 月 9 日，土耳其伊斯坦布尔遭遇暴雨并引发洪水。疯狂肆虐的洪水淹没了该市 Ikitelli 区的大部分地段，也淹没了位于该区的 Vodafone 数据中心。

2）火灾

2008 年 3 月 19 日，美国威斯康星数据中心被火烧得一塌糊涂。根据事后统计，这次大火烧掉了 75 台服务器、路由器和交换机，当地大量的站点都发生了瘫痪。

3）地震

2011 年 3 月 11 日，日本遭受了 9 级大地震。在此次地震中，日本东京的 IBM 数据中心也受损严重。

4）人为因素

2015 年 5 月 27 日下午 5 点左右，由于光纤被挖断，部分用户无法使用支付宝。随后支付宝工程师紧急将用户请求切换至其他机房，受影响的用户才逐步恢复使用。

6.1.4 安全措施

1．安全制度规范

政府、企业以及其他组织一般会制订内部的信息安全相关制度，用以规范和约束管理体系中的各项工作内容，从而确保环境的安全稳定运行，一般包括以下内容。

1）人员组织

人员组织用以明确细分各级人员对于信息安全的责任和义务，明确信息安全的管理机构和组织形式。

2）行为安全

行为安全用以明确细分每个人在组织内部允许和禁止的行为。

3）机房安全

机房安全制度明确细分出入机房、上架设备所必须遵守的流程规范。

4）网络安全

网络安全制度明确组织内部的网络区域划分，以及不同环境网络功能和隔离措施。

5）开发过程安全

开发过程安全制度明确软件的开发设计和测试遵守相关规范，开发体系和运维体系分离，源代码和文档应落地保存。

6）终端安全

终端安全制度明确终端设备的使用范围，禁止私自修改终端设备，应设置终端口令，及时锁屏，及时更新操作系统补丁等。

7）数据安全

数据安全制度不对外传播敏感数据，生产数据的使用需要在监督和授权下执行。如果需要对外提供相关敏感数据，应对数据进行脱敏处理。

8）口令安全

明确口令的复杂程度、定时修改周期等。

9）临时人员的管理

明确非内部员工的行为列表、外包人员的行为规范，防范非法入侵。

2．安全防范措施

在各个层次都有成熟的安全产品，可以供选择来构建组织内部的防御体系，如表 6-3 所示。

表 6-3　安全产品层次

分　类	安 全 产 品
机房	门禁系统、消防系统、摄像系统
服务器	防病毒软件、漏洞扫描工具、配置核查系统
网络	防火墙、入侵监测系统、入侵防御系统
终端	防病毒软件、行为控制和审计软件、堡垒机
应用程序	漏洞扫描工具、源代码扫描软件、证书管理系统、统一认证系统、身份管理系统
数据备份	数据备份软件
流程管理	运维管理平台、安全管理平台、审计平台

定期对系统进行大规模摸底扫描，并组织相关内外部资源对资源进行渗透性测试，发现并且解决系统中的安全风险点。

在组织团队和新员工入职时，就对所有的开发人员进行有针对性的安全培训，严格遵守对应的编码规范，强化安全编码和信息安全的意识。有不少人认为安全的技术产品就可以完全规避所有的安全问题，但事实并非如此。例如，如图 6-6 所示就展示了一种针对 SSL 的中间人攻击，利用该攻击模式，可以破解或者修改传输内容，也可以让客户端做的输入过滤失效。

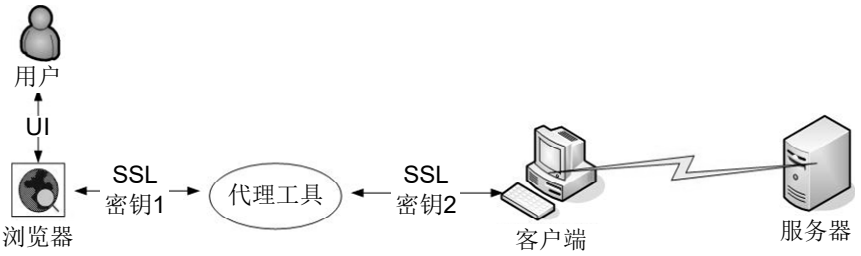

图 6-6　SSL 中间人攻击

制订的安全制度规范需要严格执行，制度中禁止的行为绝对不能因为技术因素或者人为因素而忽略执行，从而产生严重的后果。

6.2　系统优化

6.2.1　Hadoop 配置优化

1．Hadoop 硬件配置规划优化

硬件配置的优化主要基于以下几个方面。

（1）机架：节点平均分布在机架之间，可以提高读操作性能，并提高数据可用性；节点副本存储在同一机架，可提高写操作性能。Hadoop 默认存储 3 份副本，其中两份存储在同一机架上，另一份存储在另一机架上。

（2）主机：Master 机器配置高于 Slave 机器配置。

（3）磁盘：存放数据做计算的磁盘可以做 RAID 0，或考虑冗余保护需要做 RAID 0+1，提高磁盘 I/O 并行度。

由于磁盘 I/O 的速度是比较慢的，如果一个进程的内存空间不足，它会将内存中的部分数据暂时写到磁盘，当需要时，再把磁盘上的数据写到内存。因此可以设置合理的预读缓冲区大小来提高 Hadoop 里面大文件顺序读的性能，以此来提高 I/O 性能。

（4）网卡：多网卡绑定，做负载均衡或者主备冗余保护。

2．操作系统规划优化

以下合理规划对文件系统的性能提升会有较大帮助：cache mode、I/O scheduler、调度参数、文件块大小、inode 大小、日志功能、文件时间戳方式、同步或异步 I/O、writeback 模式等规划。

3．Hadoop 集群配置规划优化

1）集群节点内存分配

例如，一个数据节点，假如 task 并行度为 p，单个任务内存开销为 m GB，则节点内存配置如下。

m×4 (DataNode)+m×2 (NodeManager)+ m×4 (ZooKeeper)+ m×p

示例：并行度为 8，单任务内存开销为 1 GB，则节点内存可配置为 18 GB。

2）集群节点规模

假如每天产生的大数据容量为 d TB，需保存 t 个月，每个节点硬盘容量为 h TB，Hadoop 数据副本数为 k（通常为 3），硬盘最佳利用率为 R（常取 70%），则配置的节点数 n 可计算如下。

$n=d\times k\times t\times 30/h/R$

示例：如果每天产生的大数据容量为 1 TB，需保存 1 个月，每个节点硬盘容量为 2 TB，Hadoop 数据副本数 k 为 3，硬盘最佳利用率 70%，则节点数 n 计算如下。

$n=1\times 3\times 1\times 30/2/70\%$，$n$ 约为 65。

6.2.2　Hadoop 性能优化

1．内存优化

1）NameNode、DataNode 内存调整

在$HADOOP_HOME/etc/hadoop/hadoop-env.sh 配置文件中，设置 NameNode、DataNode 的守护进程内存分配可参照如下方案。

HADOOP_NAMENODE_OPTS：Hadoop 对应的命名空间节点设置参数。

```
export
HADOOP_NAMENODE_OPTS="-Xmx512m-Xms512m -Dhadoop.security.logger=${HADOOP_SECURITY_LOGGER:-INFO,RFAS} -Dhdfs.audit.logger=${HDFS_AUDIT_LOGGER:-INFO,NullAppender} $HADOOP_NAMENODE_OPTS"
```

即将内存分配设置成 512 MB。

HADOOP_DATANODE_OPTS：Hadoop 对应的数据节点设置参数。

```
DataNode：
export HADOOP_DATANODE_OPTS="-Xmx256m -Xms256m -Dhadoop.security.logger=ERROR,RFAS $HADOOP_DATANODE_OPTS"
```

即将内存分配设置成 256 MB。

注意：-Xmx、-Xms 这两个参数保持相等可以防止 JVM 在每次垃圾回收完成后重新分配内存。

2）ResourceManager、NodeManager 内存调整

在$HADOOP_HOME/etc/hadoop/yarn-env.sh 配置文件中，设置内存分配如下，可

以修改其中内存设置值。

YARN_RESOURCEMANAGER_HEAPSIZE：YARN 资源管理堆空间大小。

YARN_RESOURCEMANAGER_OPTS：YARN 资源管理设置参数。

```
ResourceManager：
export YARN_RESOURCEMANAGER_HEAPSIZE=1000   export YARN_RESOURCEMANAGER_
OPTS=""
```

即将内存分配设置成 1000 MB。

YARN_RESOURCEMANAGER_HEAPSIZE：YARN 资源命名空间节点堆大小。

YARN_RESOURCEMANAGER_OPTS：YARN 资源管理命名空间节点设置参数。

```
NodeManager：
export YARN_NODEMANAGER_HEAPSIZE=1000
export YARN_NODEMANAGER_OPTS=""
```

即将内存分配设置成 1000 MB。

3）Task、Job 内存调整

在$HADOOP_HOME/etc/hadoop/yarn-site.xml 文件中配置。

```
yarn.scheduler.maximum-allocation-mb
```

其中设置了单个可申请的最小/最大内存量，默认值为 1024 MB/8192 MB。

```
yarn.nodemanager.resource.memory-mb
```

总的可用物理内存量，默认值为 8096 MB。

对于 MapReduce 而言，每个作业的内存量可通过以下参数设置。

```
mapreduce.map.memory.mb：
```

设置物理内存量，默认值为 1024 MB。

2．配置多个 MapReduce 工作目录，提高 I/O 性能

在以下配置文件中设置相关参数，达到分散 I/O、提高 I/O 性能的目的。

```
$HADOOP_HOME/etc/hadoop/yarn-site.xml #对应文件及目录
```

yarn.nodemanager.local-dirs：存放中间结果。

yarn.nodemanager.log-dirs：存放日志。

```
$HADOOP_HOME/etc/hadoop/mapred-site.xml #对应文件及目录
```

mapreduce.cluster.local.dir：MapReduce 的缓存数据存储在文件系统中的位置。

$HADOOP_HOME/etc/hadoop/hdfs-site.xml：提供多个备份以提高可用性。

dfs.namenode.name.dir：HDFS 格式化 namenode 时生成的 nametable 元文件的存储

目录。

dfs.namenode.edits.dir：HDFS 格式化 namenode 时生成的 edits 元文件的存储目录。

dfs.datanode.data.dir：存放数据块（dateblock）的目录。

多个目录之间以“,”分开，如下所示。

```
/data1/dfs/name,/data2/dfs/name, /data3/dfs/name #对应文件及目录
```

3．压缩 MapReduce 中间结果，提高 I/O 性能

由于 HDFS 存储了多个副本，为避免大量硬盘 I/O 或网络传输的开销，可以压缩 MapReduce 中间结果，提高性能。

配置$HADOOP_HOME/etc/hadoop/mapred-site.xml 文件。

```
<property>#设置 MapReduce 输出结果是否压缩
 <name>mapreduce.map.output.compress</name>
 <value>true</value>
</property>
<property> #设置 MapReduce 压缩机制
<name>mapreduce.map.output.compress.codec</name>
<value>org.apache.hadoop.io.compress.SnappyCodec</value>
</property>
```

其他 MapReduce 参数调优描述如下。

（1）mapred.reduce.tasks(mapreduce.job.reduces)。

默认值：1。

说明：默认启动的 reduce 数。通过该参数可以手动修改 reduce 的个数。

（2）mapreduce.task.io.sort.factor。

默认值：10。

说明：Reduce Task 中合并小文件时，一次合并的文件数据，每次合并的时候选择最小的前 10 进行合并。

（3）mapreduce.task.io.sort.mb。

默认值：100。

说明：Map Task 缓冲区所占内存大小。

（4）mapred.child.java.opts。

默认值：-Xmx200m。

说明：jvm 启动的子线程可以使用的最大内存。建议值-XX:-UseGCOverheadLimit -Xms512m -Xmx2048m -verbose:gc -Xloggc:/tmp/@taskid@.gc。

（5）mapreduce.jobtracker.handler.count。

默认值：10。