

DDoS

虫潮降临：Zergeca僵尸网络分析报告

**Alex.Turing, Acey9**

2024年6月19日 • 17 min read



背景

样本&C2检测捕获样本小更新（我们文章发布之后）84.54.51.82的画像ScannerMirai Downloader&C2Zergeca C2使用的漏洞DDoS攻击统计

逆向分析

0x00: 字串解密0x01: Persistence Modulei. 小实验A

0x02: Silivaccine Module

i. 小实验B

0x03: Zombie Module

i. 通信协议

i. 小实验C

总结

IOC

Sample

Domain

IP

Appendix

IdaPython Script

背景

2024年5月20日，当大家都在愉快地庆祝节日时，不知疲倦的XLab大网威胁感知系统于14点左右捕获了一个可疑的ELF文件，路径为/usr/bin/geomi。该文件使用变形的UPX加壳，幻数为0x30219101，从俄罗斯上传到VirusTotal，未被任何杀软引擎检测出恶意行为。当晚22点，另一个使用相同UPX幻数的geomi文件从德国上传到VT。可疑的文件路径的，变形的UPX壳，以及多国上传的情况引起了我们的关注。经过分析，我们确认这是一个使用Golang实现的僵尸网络。由于其C2使用了“ootheca”字符串，让人联想到星际争霸中的铺天盖地的虫族，我们将其命名为Zergeca。

从功能上来说，Zergeca不仅是一个典型的DDoS僵尸网络，除了支持六种不同的攻击方法外，还具备代理、扫描、自升级、持久化、文件传输、反向shell和收集设备敏感信息等功能。从网络通信来看，Zergeca也具有以下独特之处：

1. 支持多种DNS解析方式，优先使用DOH进行C2解析
2. 使用不常见的Smux库实现C2通信协议，并通过xor进行加密

在对Zergeca的基础设施进行关联分析时，我们发现其C2对应的IP地址

84.54.51.82自2023年9月以来，至少为两个Mirai僵尸网络提供了服务。我们推测，Zergeca背后的作者正是在运营Mirai僵尸网络期间积累了一定的经验，随后推陈出新，创造了Zergeca。

6月10日，XLab指令跟踪系统捕获了一条当前样本不支持的7号DDoS指令，说明Zergeca的作者仍在积极开发和更新，新样本尚未被社区发现。目前，Zergeca样本&C2的检测率都很低，再考虑到Zergeca在DDoS方面的潜在危害，我们决定编写本文，向社区分享我们的发现。

样本&C2检测

从样本层面来看，我们一共捕获了4个Zergeca样本，它们的功能几乎是一样的，高度相似的样本，检测率却相差如此之多，该怎么解释这个异常呢？大部分杀软厂商对23ca4ab1518ff76f5037ea12f367a469的研判结果是Generic Malware，我们推测杀软对于Zergeca检测基于hash的特征，因此只要hash变化了，检测的效果就不理想了。

MD5	DETECTION	FIRST SEEN	TELEMETRY
23ca4ab1518ff76f5037ea12f367a469	28/64	2024.05.20	Russian
9d96646d4fa35b6f7c19a3b5d3846777	0/67	2024.05.20	Germany
d78d1c57fb6e818eb1b52417e262ce59	1/67	2024.05.22	China
604397198f291fa5eb2c363f7c93c9bf	1/66	2024.06.11	France

为了验证猜想，我们在23ca4ab1518ff76f5037ea12f367a469文件的尾部加入了4字节“Xlab”，重新上传VT后，检测率变成了9/67，部分证明了我们的推测。

而且当前的检测是基于加壳样本；脱壳后，检测为0。

The screenshot shows the VirusTotal analysis interface for a modified file. On the left, there's a green circular progress bar with '0 / 67' and a 'Community Score' slider. The main panel displays the following information:

- Status:** No security vendors and no sandboxes flagged this file as malicious.
- File Details:** c51cf3173da4fbf4ba432efb2ae2479969694022d77816c17a6... (SHA-256 hash), Size: 8.42 MB, Last Modification Date: a moment ago. File type: ELF, 64bits.
- Analysis Options:** Follow, Reanalyze, Download, Similar, More.
- Community:** A small icon showing a person with a gear and the text 'ELF'.

At the bottom, there are tabs for DETECTION, DETAILS, RELATIONS, BEHAVIOR (selected), CONTENT, TELEMETRY, and COMMUNITY.

从域名层面来看，四个样本共用两个在同一天创建的C2域名。样本优先使用DOH（DNS over HTTPS）方式进行C2解析，这在一定程度上掩盖了样本和C2域名之间的关系。正是因为这个缘故，VT甚至没能将C2域名和样本关联在一起，检测率自然就很低。

DOMAIN	DETECTION	CREATE DATE
ootheca.pw	1/93	2024.04.28
ootheca.top	1/93	2024.04.28

捕获样本小更新（我们文章发布之后）

2024年6月19日，捕获了支持7号DDoS的样本，

60f23acebf0ddb51a3176d0750055cf8，7号DDoS指令对应的方法为
psh0vhFlood。

84.54.51.82的画像

Zergeca的2个C2指向同一个IP地址 84.54.51.82，从我们的数据来看，它于2023年9月投入使用，角色非常丰富，其间充当过Scanner，Downloader，Mirai僵尸网络C2，Zergeca僵尸网络C2。

Scanner

2023.09.18开始扫描活动，主要扫描Telnet、HTTP、socks4等协议，扫描端口主要是 23 8080 3128 80 8888 等

Mirai Downloader&C2

2023年09、10月以2024年4月 84.54.51.82 主要被当作Mirai僵尸网络的Loader IP和Downloader IP被使用

- 2023年09、10月作为Loader和Downloader IP用来植入以下相关样本。

```
# Downloader

http://84.54[.51.82/jaws
http://84.54[.51.82/bin
http://84.54[.51.82/596a96cc7bf9108cd896f33c44aedc8a/db0fa4b8db0333367e9bda3ab68b80
# CC
mirai://bot.hamsterrace.space:59666
```

- 2024.04作为Loader IP用来植入以下相关样本

```
# Downloader
http://145.239[.108.150/Fantazy.sh
http://145.239[.108.150/Fantazy/Fantazy.arm5
http://145.239[.108.150/Fantazy/Fantazy.arm6
http://145.239[.108.150/Fantazy/Fantazy.mpsl
http://145.239[.108.150/Fantazy/Fantazy.sh4
http://145.239[.108.150/Please-Subscribe-To-My-YT-Channel-VegaSec/1isequal9.x86
http://145.239[.108.150/cache

# CC

mirai://145.239.108.150:63645
```

Zergeca C2

2024年4月29号起开始作为Zergeca C2被使用。相关CC域名和解析记录入下图所示：

使用的漏洞

我们的视野中 84.54.51.82 传播样本的手段主要是Telnet弱口令和部分已知漏洞，相关漏洞编号如下：

```
Telnet Weak Password  
CVE-2022-35733  
CVE-2018-10562  
CVE-2018-10561  
CVE-2017-17215  
CVE-2016-20016
```

DDoS攻击统计

2024年6月初到中旬Zergeca僵尸网络主要攻击目标集中在加拿大、美国、德国等地区，攻击类型主要为 ackFlood (atk_4)，受害者分布在多个国家和不同的ASN中。

逆向分析

我们视野中的4个Zergeca样本都是x86-64CPU架构，针对Linux平台。样本中有“android, darwin, windows”等字串，再加上golang天然的跨平台能力，我们认为作者或许会在有一天实现全平台支持。本文为最早捕获的样本为主要分析对象，它使用UPX加壳，幻数为 0x30219101。对于此类变形UPX壳，只需将幻数修改回标准的“UPX!”，就可以使用upx -d进行脱壳。

```
MD5:23ca4ab1518ff76f5037ea12f367a469  
Mgaic:ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically linked, corru  
Packer: UPX  
Version:0.0.01c
```

脱壳后可以看出Zergeca是一个Go语言实现的僵尸网络，符号没有混淆，因此逆向分析比较容易。

上图为main_main函数的代码片段，从功能上来说，可以拆解成4个不同的模块。其中persistence与proxy不用多说，前者是持久化，后者则是代理；silivaccine用于清除竞争对手，达到独占设备的目的；最重要是zombie，它实现了完整的僵尸网络功能，将被侵入设备的敏感信息上报给C2之后，等待执行C2下发的指令，支持6种DDoS攻击方法，扫描，反弹SHELL等功能。

OXOO: 字串解密

Zergeca大量敏感字符串都使用xor加密，通过IDA查看**xor秘钥**的交叉引用，数量高达240次，存在于各种功能函数之中。实际上在一次解密过程中会用到2次xor的秘钥，一次是初始化，一次是解密，所以真正需要解密的次数为120。

当需要使用某个字符串时，使用以下类似的代码片段解密，基中**xor_key**原始值为
`EC 22 2B A9 F3 DD DF 1C CD 46 AC 1E`，实际只使用前6字节，即
`EC 22 2B A9 F3 DD`。

为了方便分析，我们需要将所有的加密字串还原，手动解密120次，肯定不现实。由于解密过程不是在一个特定的函数中，因此我们没有办法通过函数参数直接获得密文。经过CFG分析，我们发现大部分与解密相关的代码块存在一个特定模式：

1. xor block的前驱&后继块数均为1
2. xor block的前驱块p1，第一条指令为mov，它的第1个operand为地址，指向**xor_key**的原始长度

3. xor block的后继块s1，第一条指令为cmp，它的第1个operand为数字，指定了密文的长度
4. p1的前驱块p2，第一条指令为lea，它的第1个operand为地址，指向了密文的起始地址

基于上述模式，我们实现了附录的idaPython解密代码，效果如下，最终统计111处成功解密，9处不符合模式。

9处不符合模式的代码分布在以下6个函数中，其中packets__Cursor的Read/WriteString两个函数用于加解网络报文，可以直接忽略。

```
geomi_bot_zombie__Zombie_Connect  
geomi_common_utils_init_0_func1,  
geomi_bot_discovery_Run,  
geomi_common_packets__Cursor_WriteString,  
geomi_common_packets__Cursor_ReadString,  
geomi_common_utils_RandomUserAgent
```

分析其余4个函数失败的原因，我们发现密文在这些函数中不再是单一密文，而是一个数组，因此模式匹配失败。以RandomUserAgent为例，user_agent_list包含了1000个加密的UA。

对于这种形式，我们可以用manual_decode函数进行解密，第1个参数为密文数组起始地址，第2个参数为数组元素个数。

```
key=b"\xEC\x22\x2B\xA9\xF3\xDD"  
  
def manual_decode(base,cnt):  
    for i in range(cnt):  
        start=idc.get_qword(base)  
        addr=idc.get_qword(start+i*16)  
        size=idc.get_qword(start+8+i*16)  
        buff=idc.get_bytes(addr,size)  
        out=bytearray()  
        for k,v in enumerate(buff):  
            out.append(v ^ key[k%6])  
        print(out.decode())  
  
manual_decode(0x00000000C56FA0,1000) #user agent
```

```
manual_decode(0x0000000000C56F80,0xc) #opennic dns  
manual_decode(0x000000000C56C40,2) # c2
```

部分UA, OpenNIC DNS, C2解密效果如下:

至此所有字符串都成功的解密，我们可以开始逆向分析Zergeca的各种功能了。

0x01: Persistence Module

Zergeca在被侵入的设备中增加系统服务geomi.service实现持久化，当设备重启或进程被结束时，自动生成geomi新进程。

```
[Unit]  
Description=  
Requires=network.target  
After=network.target  
[Service]  
PIDFile=/run/geomi.pid  
ExecStartPre=/bin/rm -f /run/geomi.pid  
ExecStart=/usr/bin/geomi  
Restart=always  
[Install]  
WantedBy=multi-user.target
```

小实验A

在虚拟机中运行Zergeca样本并重启。设备重启后，geomi.service自动运行Zergeca样本，生成的进程名为geomi，pid为897。当使用kill -9 897结束进程后，马上生成了新的geomi进程8460。

当网络管理员发现设备中有geomi进程且有可疑的流量时，可以尝试以下步骤进行清理。

1. 删除/etc/systemd/system/geomi.service
2. 删除ExecStart参数指向样本文件

3. 结束geomi进程

oxo2: Silivaccine Module

Zergeca为了独占设备，内置了一份竞争对手名单，涵盖了挖矿，后门木马，僵尸网络等各种威胁。名单中部分字串如下所示，有大家熟悉的mozi, kinsing, 以及各种矿池等。Zergeca会对系统实时监控，当系统中的 `进程名, 运行时参数` 命中了名单中的字串，则直接结束进程，并删除对应的二进制文件。

MOZI.A	COM.UFO.MINER	KINSING	KTHREADDI
kaiten	srv00	meminitrv	.javae
solr.sh	monerohash	minexmr	c3pool
crypto-pool.fr	f2pool.com	xmrpool.eu

小实验B

我们将系统程序/bin/sleep，重命名为Mozi.a并运行，可以看出Mozi.a进程被kill，且相应的二进制文件也被删除。

oxo3: Zombie Module

Zergeca通过 `geomi_common_utils_Resolve` 函数解析C2的IP地址。`Resolve` 函数支持四种不同的解析器：Public DNS、Local DNS、DOH（DNS over HTTPS），以及OpenNIC。

Zergeca的策略是优先使用以下2个DOH Resolver，因此在DNS流量看不到C2域名的解析。

```
https://cloudflare-dns.com/dns-query  
https://dns.google/resolve
```

得到C2的IP后，Bot向其上报设备的敏感信息，Zergeca定义了一个名为DeviceInfo的结构体用于存储设备的敏感信息，涵盖"国家，公网IP，操作系统，用户组，运行目录，是否可达"等。

```
struct DeviceInfo
{
    Country string
    PlucAddress byte[]
    MAC string
    OS string
    ARCH string
    Name string
    MachineId string
    Numcpu uint32
    CPUMODEL string
    username string
    uid string
    gid string
    Users []string
    Uptime time.Duration
    PID      uint32
    Path string
    checksum []uint8
    version string
    Reachable bool
}
```

随后等待执行C2下发的指令，通过不同的Handle处理不同的指令。

指令号与功能的对应如下表所示：

ID	TASK
0x01	Proxy
0x02	Reverse Shell
0x03	FileTransfer
0x05	Self-update
0xa0	DDoS
0xb0	Stop Discovery
0xb1	Start Discovery

其中DDoS支持6种不同的攻击向量

SUB-ID	ATTACK VECTOR
1	minecraft
2	httpPPS
3	synFlood
4	ackFlood
5	pushFlood
6	rstFlood

通信协议

Zergeca使用smux实现了Bot与C2的通信。Smux (Simple MULTipleXing) 是Golang的多路复用库，它依赖于底层连接来提供可靠性和排序，例如TCP或KCP，并提供面向流的复用。Smux的特性中有一条为 `Minimized header(8Bytes), maximized payload`，它的意思是凡是Smux生成的网络报文都会附带8字节的header，格式为 `VERSION(1B) | CMD(1B) | LENGTH(2B) | STREAMID(4B) | DATA(LENGTH)`。

事实上，从分析的角度来说，我们只关心LENGTH与DATA这两个字段。以下图实际捕获的流量为例，它涵盖了上线，上报设备信息，`0xb0`指令，心跳等多种报文。

- 上线报文长度为0x04字节，内容为硬编码的 `13 3a 12 79`
- 上线设备信息报文长度为0xd5字节（视设备而定），除IP外都使用xor加密，key为 `EC 22 2B A9 F3 DD`，以下为解密后的DeviceInfo

```
pos: 0x4 len: 0x2 <----> b'JP'
pos 0x7 len: 4 <----> 45.14.XX.XX
pos: 0xc len: 0x11 <----> b'72:ba:29:e9:b8:08'
pos: 0x1f len: 0x5 <----> b'linux'
pos: 0x26 len: 0x5 <----> b'amd64'
pos: 0x2d len: 0x6 <----> b's22262'
pos: 0x35 len: 0x20 <----> b'b19642a3c672d4f20cbdb5b1569bf98f'
pos: 0x5b len: 0x29 <----> b'Intel(R) Xeon(R) CPU E5-2678 v3 @ 2.50GHz'
pos: 0x86 len: 0x4 <----> b'root'
```

```
pos: 0x86 len: 0x4 <----> b'root'
pos: 0xa2 len: 0x2 <----> b'\x92\xf1'
pos: 0xa6 len: 0xe <----> b'/usr/bin/geomi'
pos: 0xb6 len: 0x14 <----> b'r\xbd>\xcfY\x15[\xd9]\xa4\xe7m\x86\x9f\xbf\x89'
pos: 0xcc len: 0x7 <----> b'0.0.01c'
```

- 0xb0指令报文长度为0x08字节，功能为停止扫描
- 心跳报文长度为0x03字节，内容为 ff 00 00

我们再看一下DDoS相关的报文，它的格式为 cmd(1byte) + length(2 bytes) + sub_cmd(1 byte) + target_info (length-1)，其中cmd为0xa0，表示这是一个DDoS相关的指令，sub_cmd为0x4，表示ackflood。target_info只需关心前4字节，它是被攻击的IP，1f 06 10 21 即为 31.6.16.33。

当Bot接收到上述指令后，产生的攻击流量如下所示，和我们分析是能一一对应的。

小实验C

我们根据网络协议分析的结果，实现了一个fake c2用于控制Bot，观察Bot接收到不同指令时的行为。在这个实验中，我们向Bot下发了**0xb1**指令，即"开始扫描"。

Bot收到此指令后，马上向随机生成的IP地址的16个端口发起扫描。

总结

经过逆向分析，我们对Zergecar的作者有了初步的认识：内置的竞争对手名单表明其作者对Linux生态下流行的威胁非常熟悉；使用变形UPX加壳、敏感字符串的xor加密、以及通过DOH隐藏C2解析等技术，显示了他们的免杀意识；基于Smux实现网络协议则展示了其开发能力。面对这样一个既会运营、又懂对抗、还能开发的对手，我们在未来如果再看到他的新作品也不会感到意外，只想说：“Give it your all and wow us!”

这是我们目前掌握的关于Zergeca的基本情报，欢迎具有独特视角的同行企业提供新的线索，比如Init Access，支持7号DDoS指令的新样本等。对我们研究感兴趣的读者也可通过[Twitter](#)联系我们获取更多详细信息。

IOC

Sample

```
23ca4ab1518ff76f5037ea12f367a469  
9d96646d4fa35b6f7c19a3b5d3846777  
d78d1c57fb6e818eb1b52417e262ce59  
604397198f291fa5eb2c363f7c93c9bf
```

```
f68139904e127b95249ffd40dfeedd21  
d7b5d45628aa22726fd09d452a9e5717  
6ac8958d3f542274596bd5206ae8fa96
```

```
pathced with "xlab" at the end of file  
980cad4be8bf20fea5c34c5195013200
```

```
sample captured on 2024.06.19, support ddos vector 7  
60f23acebf0ddb51a3176d0750055cf8
```

Domain

```
ootheca.pw  
ootheca.top  
bot.hamsterrace.space
```

IP

84.54.51.82

The Netherlands | None | None

AS202685 | Aggros Operations Ltd.

Appendix

IdaPython Script

```
# Test script, only for 23ca4ab1518ff76f5037ea12f367a469
# Modidy keyaddr,sizeaddr in your case

def decode(buf):
    key=b"\xE0\x22\x2B\xA9\xF3\xDD"
    out=bytearray()
    for i in range(len(buf)):
        out.append(buf[i]^key[i%6])
    return out

count=0
notcount=0
failedfunc=[]
successedfunc=[]

keyaddr=0x0000000000C56FC0
sizeaddr=0x0000000000C56FC8

refs=XrefsTo(keyaddr, flags=0)
for ref in refs:
    f_blocks = idaapi.FlowChart(idaapi.get_func(ref.frm), flags=idc.FC_PREDS)
    for blk in f_blocks:
        if blk.start_ea!=ref.frm:
            continue
        if len(list(blk.preds()))!=1 and len(list(blk.succs()))!=1:
            continue
        predblk=list(blk.preds())[0]
        succsblk=list(blk.succs())[0]

        if idc.get_operand_value(predblk.start_ea,1)!=sizeaddr:
            continue
        if idc.get_operand_type(succsblk.start_ea,1)!=0x5:
            print(idc.get_func_name(ref.frm),hex(ref.frm),"not matched")
            notcount+=1
            failedfunc.append(idc.get_func_name(ref.frm))
            continue
        ppredblk=list(predblk.preds())
        if len(ppredblk)!=1:
            continue
        addr=idc.get_operand_value(ppredblk[0].start_ea,1)
        size=idc.get_operand_value(succsblk.start_ea,1)
```

```
buf=idc.get_bytes(addr,size)
out=decode(buf)
count+=1
print(idc.get_func_name(ref.frm),hex(ppredblk[0].start_ea),"matched, cipher")
successedfunc.append(idc.get_func_name(ref.frm))

print("\n-----Statistic-----")
print(f'Success:{count},Failed:{notcount}\n')
print("-----Success Function-----")
print(set(successedfunc),'\n')
print("-----Failed Function-----")
print(set(failedfunc),'\n')
```