

DDoS

币安智能合约正在被Smargaft僵尸网络滥用

**Alex.Turing, Acey9**

2024年2月2日 · 22 min read



背景

什么是币安智能链 (BSC)、合约Smargaft合约分析Q: 什么是eth_call?DDoS活动统计样本传播分析逆向分析Downloader分析下载执行Bot样本结束竞争对手

Propagation Task

Common Tasks

- i. 0x00: 初始化
- i. 0x01: GetC2
- i. 0x2: DDoS Task
- /.
- /.
- /.
- i. 0x3: Ip-Forward Task
- /.
- /.
- i. 0x4: Persistence Task
- i. 0x5: Killer Task

总结

联系我们

IOC

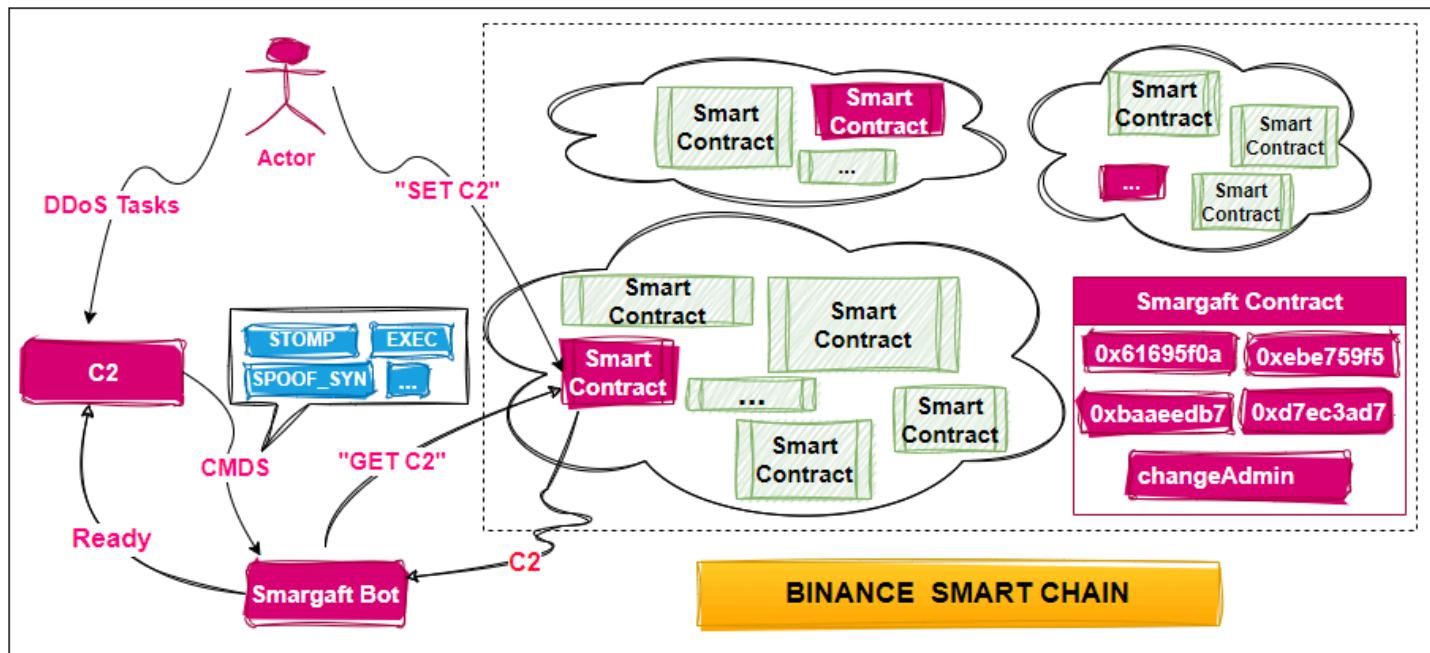
sample md5

C2(port:81).

Downloader(port:82).

背景

在XLab的日常工作中，我们的僵尸网络监控系统每天都能检测到大量基于Mirai, Gafgyt代码魔改而来的变体的僵尸网络。这些变体已经司空见惯，无法引起我们的兴趣。然而，今天的主角是一个异类，虽然它复用了一些Gafgyt的攻击向量，但很明显程序的逻辑结构是重新设计的。除此之外其作者还有一些让人眼前一亮的创新，比如利用币安智能链（Binance Smart Chain）的合约托管命令和控制中心（C2），比如通过病毒式感染Shell脚本实现持久化等。我们在进行逆向分析时，发现一些杀毒软件将这个僵尸网络的ARM样本标记为Mirai，这是不准确的。基于这个僵尸网络使用智能合约以及Gafgyt的攻击向量，我们将它命名为Smargaft，它的主要功能是DDoS攻击，执行系统命令，提供socks5代理服务等。



Smargaft最大的亮点是使用智能合约托管C2，这种技术于2023年10月被首次披露，业内称之为**EtherHiding**，它充分利用区块链的公开性和不可篡改性，“链上”的C2无法被移除，是一种相当高级且少见的Bullet-Proof Hosting技法。这是我们在僵尸网络领域首次见到此类技术的应用。智能合约这种云端配置C2的另一个好处就是灵活性，病毒作者甚至可以通过精心设计监控代码和智能合约进行互动，实现满足特定条件时自动更新C2，或者根据环境变化自动调整攻击策略。鉴于合约一旦被滥用即可能迅速成为网络犯罪的有力工具，从而大幅增加了监测和管理的难度，我们决定撰写本文与社区分享我们的最新发现，希望能够帮助大家更有效地识别和预防这类新型网络威胁。

什么是币安智能链（BSC），合约

BSC (Binance Smart Chain)，即币安智能链，是由币安（Binance）开发和维护的一个区块链平台。它于2020年推出，旨在为去中心化应用程序（DApps）和智能合约提供支持，类似于以太坊（Ethereum）。

智能合约是一种在区块链上执行的自动化协议或程序。它们是预先编写好的代码，旨在根据特定的条件自动执行操作或合同条款。智能合约允许在没有中介的情况下进行可信的交易和交互。当特定条件得到满足时，智能合约可以触发各种操作，例如转移数字资产、分配奖励、创建令牌等。

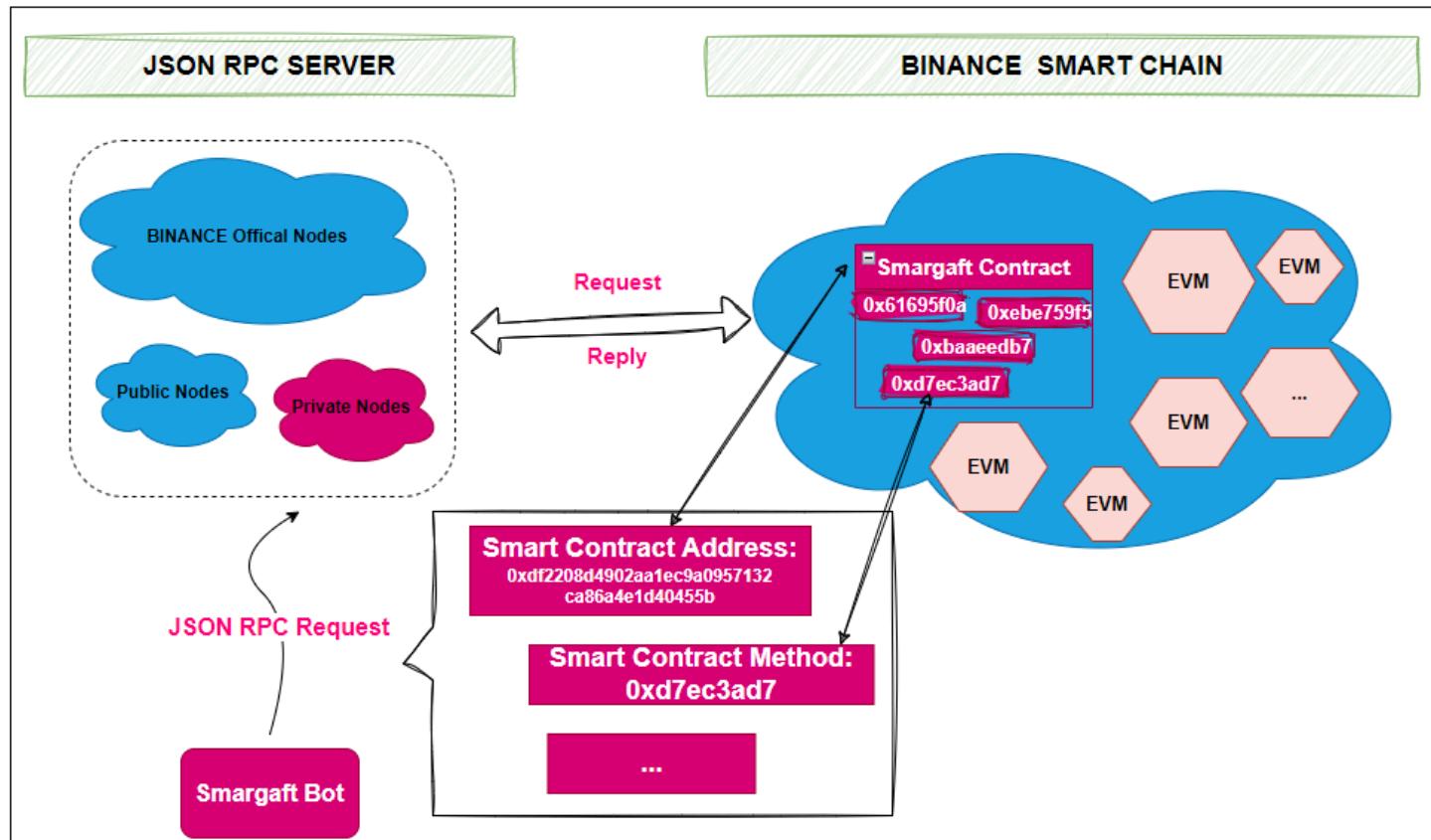
XLab按：

智能合约就像一个特殊的盒子，放在区块链上。这个盒子可以保存数据，做一些计算，并且允许人们查看和使用这些数据。

区块链的一个特点是信息永远不会被删除，所以这个盒子会一直存在；区块链的另一个特点是所有交易都是有记录的，所以利用这个盒子做的操作，都是有迹可循，无法篡改。

在“Smargaft利用智能合约进行C2管理”的场景中，实际上意味着Smargaft的开发者通过智能合约来配置和管理C2，而C2的相关信息最终被存储在区块链上。一旦C2配置完成，Smargaft的恶意软件（Bot样本）便通过JSON RPC与区块链网络通信来获取C2信息。

具体来说，Bot样本会向一个RPC服务器发送请求，这个请求包含了Smargaft智能合约的地址和要调用的合约函数。RPC服务器接收到这个请求后，会将它转发给区块链网络。随后，区块链节点接收到请求，并利用EVM（以太坊虚拟机）来加载和执行智能合约中指定的函数。执行完毕后，结果会沿着原路返回给Bot样本。



综合考虑区块链查询数据的方法，以及区块链本身的特点，“智能合约托管C2”这一技术的优势主要体现在三个方面的“不可阻断”上：

1. 访问通道不可阻断：由于RPC节点的多样性，包括官方节点、公共节点以及私人搭建的节点，很难通过黑名单覆盖所有可能的访问路径。以Smargaft为例，即使币安官方节点在中国被封锁，仍有多个公共节点可供使用。Smargaft的Bot样本内置了14个不同的RPC服务器地址，确保至少有一部分节点可用。
2. 配置C2的机制不可阻断：在区块链上部署的恶意智能合约难以被直接封禁或删除。以Smargaft为例，其智能合约地址（如"0xdf2208d4902aa1ec9a0957132ca86a4e1d40455b"）不会受到币安或其他区块链运营方的直接监管或阻断。
3. C2信息不可阻断：C2信息一旦通过智能合约存储在区块链上，就无法被删除或篡改。在Smargaft案例中，C2信息作为一个记录被永久保存在币安智能链的特定区块（如34731229号区块）上，确保了其持久性和不可篡改性。

Smargaft合约分析

对于智能合约，可以使用币安提供的平台[BscScan](#)进行浏览和分析。Smargaft样本中使用的合约地址为**0xdf2208d4902aa1ec9a0957132ca86a4e1d40455b**，我们整理出了时间线，可以看出Smargaft的作者概率是俄罗斯人，于2023.09.15开始尝试智能合约托管C2这种技术，9月20日基本完成了测试，12月27日正式投入使用。

```
+ 2023-09-15:  
  + Transfer  
  + Src Wallet: 0x7Be249AA69c631c7aa5De4F3aDbFb8A9db8DfD09  
  + Dst Wallet: 0x16Cc46219d062257F384D85F84c7AbC7D9e34444  
  + Amount: 0.06094994 BNB ($17.75)  
+ 2023-09-15 :  
  + Create the first contract (unstripped)  
  + Wallet: 0x16Cc46219d062257F384D85F84c7AbC7D9e34444  
  + Contract: 0xe77c6a0E10F2A469fb2afa667C99180E186233a8
```

```

+ Init:
    + Addr: 45.95.146.93
+ The comment is Russian
+ 2023-09-15:
    + Set address: 1.1.1.1
    + Wallet: 0x16Cc46219d062257F384D85F84c7AbC7D9e34444
    + Contract: 0xe77c6a0E10F2A469fb2afa667C99180E186233a8
+ 2023-09-15:
    + Create the second contract (same as the first one, but stripped)
    + Wallet: 0x16Cc46219d062257F384D85F84c7AbC7D9e34444
    + Contract: 0x862fbef2456499a37e9146f1ef6eb5a57c2fb97a
    + InitAddr:
+ 2023-09-15:
    + Change the IP addresses around
        + 45.95.146.93 -> 10.202.30.40 -> 45.95.146.93 -> 1.22.33.44 -> 45.
    + Wallet: 0x16Cc46219d062257F384D85F84c7AbC7D9e34444
    + Contract: 0x862fbef2456499a37e9146f1ef6eb5a57c2fb97a
+ 2023-09-20:
    + Create the third contract (add one variable and its set/get function, sti
    + Wallet: 0x16Cc46219d062257F384D85F84c7AbC7D9e34444
    + Contract: 0xdf2208d4902aa1ec9a0957132ca86a4e1d40455b
    + Init:
        + Addr: 45.95.146.93
        + New Parameter: hello
+ 2023-12-27
    + Set address: 45.95.146.93;94.103.188.167;185.132.125.193
    + Wallet: 0x16Cc46219d062257F384D85F84c7AbC7D9e34444
    + Contract: 0xdf2208d4902aa1ec9a0957132ca86a4e1d40455b

```

最初的合约 `0xe77c6a0E10F2A469fb2afa667C99180E186233a8` 是能直接看到源代码，其中有大量的俄语注释，它的主要功能是通过**getServerAddr**, **setServerAddr**从链上读取&写入ServerAddr。

```

pragma solidity ^0.8.0;

contract ControlEbanataContract {
    string private serverAddr;
    address public admin;

    // Событие для уведомления о смене адреса сервера
    event ServerAddrChanged(string newAddr);

    constructor(string memory initialServerAddr) {
        serverAddr = initialServerAddr;
        admin = msg.sender;
    }

    // Изменить адрес сервера (только администратор)
    function setServerAddr(string memory newAddr) public {

```

```

        require(msg.sender == admin, "Only the admin can change the server address");
        serverAddr = newAddr;
        emit ServerAddrChanged(newAddr);
    }

    // Получить текущий адрес сервера
    function getServerAddr() public view returns (string memory) {
        return serverAddr;
    }

    // Изменить администратора (только текущий администратор)
    function changeAdmin(address newAdmin) public {
        require(msg.sender == admin, "Only the admin can change the admin");
        admin = newAdmin;
    }
}

```

中间经过几次测试，最终来到Smargaft的合约

0xdf2208d4902aa1ec9a0957132ca86a4e1d40455b，此时的合约是编译后的字节码。

虽然无法直接看到Smargaft合约的源代码，但可以通过点击"Decompile Bytecode"进行反编译。

```

def storage:
    stor0 is array of struct at storage 0
    stor1 is array of struct at storage 1
    adminAddress is addr at storage 2

def unknown61695f0a(array _param1) payable:
{...}
    if _param1.length:
        stor1[].field_0 = Array(len=_param1.length, data=_param1[all])
{...}

def unknownd7ec3ad7() payable:
{...}
    return Array(len=stor1.length % 128, data=mem[128 len ceil32(stor1.length.field_1
{...}

def unknownbe759f5(array _param1) payable:
{...}
    if _param1.length:
        stor0[].field_0 = Array(len=_param1.length, data=_param1[all])
{...}
def unknownbaaeedb7() payable:

```

```
{...}  
    return Array(len=stor0.length % 128, data=mem[128 : len].ceil32(stor0.length.field_1  
{...}
```

这是一个简单的合约应用，使用**storage**储存变量stor0, stor1, adminAddress。方法0x61695f0a的功能是把输入 `_param1` 逐字节保存到stor1中，0xd7ec3ad7则是读取stor1中的数据并以string形式返回；方法0xebe759f5, 0xbbaeedb7的功能类似，只不过它们操作的是变量stor0。有编程经验的读者应该一眼就明白这对种类似**set/get**的方法，通过这种**set/get**的方式与合约互动，可以在链上写入或更新数据。

以Smargaft的实际操作为例，其作者于 Dec-27-2023 09:55:26 PM +UTC 通过 **0x61695f0a**方法，将C2数据写入到链上。

而在Smargaft的bot样本，则通过**eth_call**调智能合约的方法**0xd7ec3ad7**获得C2。

Q: 什么是**eth_call**？

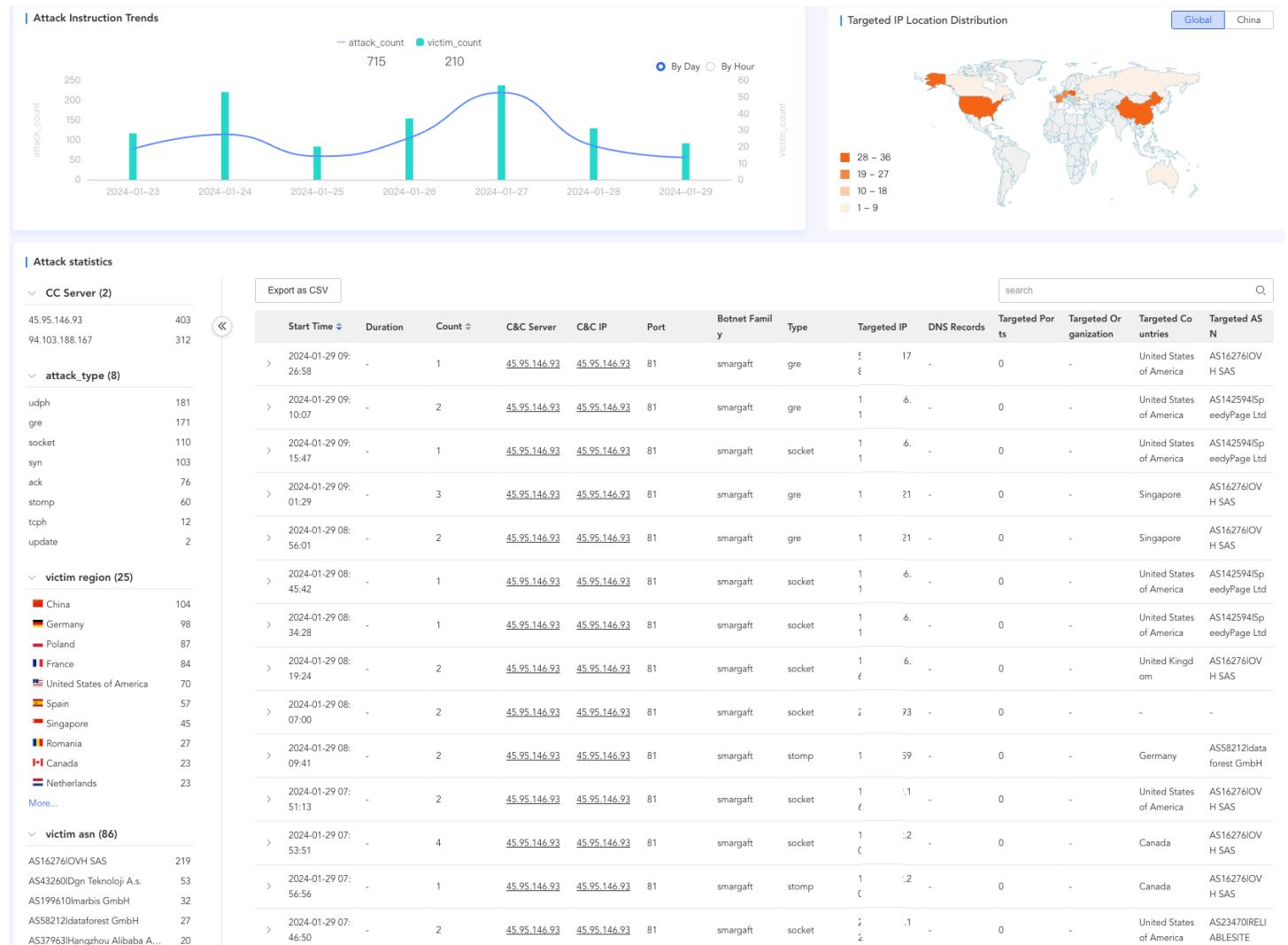
细心的读者肯定在"SET C2"的图中发现有一个**Transaction Fee**，即调用0x61695f0a这个方法是用花钱的，事实上智能合约的函数调用基本都是要花钱的。这在"Set C2"的场景是没有问题，毕竟使用的次数少；但到了"Get C2"的场景，就有一个问题："如果Bot的数量极大，那岂不是每一次获取C2，都得花一大笔钱？"，这显然是不可接受的。幸好Binance的SDK中提供了一个方法 `eth_call`，它允许用户读取智能合约的数据而不需要产生任何区块链交易，不会改变区块链的状态，自然也就不需要花钱了。

XLab按：

`eth_call` 最初设计用于模拟合约执行以读取数据或测试，而不会产生任何实际影响，它甚至不会被记录在区块链上。所以，你可以免费、不留痕迹且稳健地获取你的数据（恶意负载），而不会留下任何痕迹。

DDoS活动统计

从被攻击的目标地理位置看，Smargaft僵尸网络的攻击目标遍布全球，并没有针对性，攻击目标主要分布在中国、波兰、美国、德国、法国等地区。具体统计如下图所示：



一个有意思的现象是，Smargaft的攻击目标同时会被几个不同的僵尸网络攻击，我们推测有一个DDoS平台汇聚了诸多不同的僵尸网络。

样本传播分析

从我们的数据看，Smargaft使用已知漏洞传播Downloader到目标设备，Downloader植入成功后再下载Bot样本的方式传播，Smargaft利用的漏洞如下：

VULNERABILITY	AFFECTED
CVE-2013-5948	ASUS RT-AC68U other RT series routers
CVE-2020-8515	DrayTek Vigor Router
LILIN DVR RCE	LILIN DVR
TVT API RCE	TVT DVR

逆向分析

Downloader分析

我们以mips架构的的Downloader为主要分析对象，它的基本信息如下，样本使用标准的UPX壳。

```
MD5: 2cf03e7425da7244be659d53a972708c
Magic: ELF 32-bit LSB executable, MIPS, MIPS-I version 1 (SYSV), statically linked,
Packer: UPX
```

Downloader的核心功能较为直接，主要包括两个方面：首先，它用于下载并执行下一阶段的Bot样本；其次，它旨在消除竞争对手。

下载执行Bot样本

下载器（Downloader）通过82端口与下载服务器（Downloader Server）建立连接，并发送一个请求，包含想要下载的Bot样本的CPU架构信息。服务器接收到这个请求后，首先回复一个10字节的字符串，用于指示Bot样本的文件大小；之后就开始发送Bot样本。

实际交互产生的流量如下图所示，可以看出Downloader请求下载mips样本，样本大小为915732字节。

结束竞争对手

Downloader通过/proc文件系统监控系统的进程，强制结束特定的进程以求达到独占设备的目的。

1. 4个文件传输进程

```
curl  
wget  
tftp  
ftpget
```

2. 112个系统进程

```
init  
[kthreadd]  
[ksoftirqd/0]  
[khelper]  
...  
total 112 system process name
```

通过以下代码用于判断系统进程的真伪，依据两个关键指标：一是判断进程对应的可执行文件是否可以被正常访问；二是检查该文件的最后修改时间。这样，可以有效识别并区分真正的系统进程与可能的竞争对手（或假冒）进程。

Bot样本分析

我们捕获了ARM, MIPS ,X86 3个不同CPU架构的Smargaf bot样本，隶属于2个不同的版本。它们的主要区别在于是否支持蠕虫式传播，本文以老版本的X64样本为主要分析对象，它的基本信息如下，样本使用标准的UPX加壳。

```
MD5: 7f741495f14c828c20db4de6251673fd  
Magic: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically linked, corr  
Packer: UPX  
Version: V0
```

Smargaft的功能相对简单。在设备被侵入后运行时，首先它会检查当前用户，如果是root，则会额外的开启扫描传播任务。随后，它通过绑定本地端口来确保单一实例运行，并操纵watchdog以防止设备重启。接着，它初始化五个任务，任务包括通过智能合约获取C2，DDoS攻击，持久化等。最终，Smargaft会在一个无限循环中运行，根据预设的时间间隔轮询这些任务。下文将围绕这些任务，剖析Smargaft的功能实现。

1. Propagation Task

2. GetG2 Task

3. DDoS Task

4. Forward Task

5. Persistence Task

6. Killer Task

Propagation Task

Smargaft通过以下代码进行感染的任务，代码逻辑比较简单：通过port_probe函数构造syn报文，对10.0.0.0/8这个网段IP的目标端口进行嗅探，样本中一共硬编编码了22个端口。变量v4记录了嗅探IP的次数，当**v4=100**时，对开放了目标端口的ip，尝试使用CVE_2021_41653进行感染传播；当**v4=500**时，随机产生公网IP进行嗅探。

细心的读者肯定会发现，上图中的这段代码是有bug的，v4的值在**v4==100**的代码块中，被重置为0，如此一来v4==500的代码块永远不会执行，这导致Smargaft散失了公网蠕虫式扫描传播的能力。我们在Smargaft的新版本中V1中，并没有看到作者对此进行修复，只是简单粗暴的删除了这个功能。

以下为smargaft针对CVE_2021_45653的构建的payload

Smarfagt在构造syn报文时，使用的源端口为55555，以下为实际的嗅探流量，可以明显看出**1 IP: 22 Port**这个模式，和上文的分析能对应上。

Common Tasks

0x00: 初始化

Smargaft使用以下代码片段初始化一个task，并插入到任务链中。

经过分析，我们确定了Task的结构，它包含了任务方法，上次任务开始时间，任务间隔，任务类型，下一个任务等信息。

```
struct Task
{
    _QWORD *task_proc;
    _QWORD last_time;
    _DWORD interval_time;
    _DWORD task_type;
    Task *task_next;
};
```

将IDA中相关的变量重新定义为**Task***类型之后，效果如下

Smargaft一共支持5个不同的任务，以下为各个任务的详细属性。

TYPE	TASK	INTERVAL(SECOND)
1	GetC2	3000
2	DDoS	8
3	Ip-Forward	360000
4	Persistence	360000
5	Killer	1

当所有任务初始化完成后，通过一个永真循环进行对任务进行轮询。

0x01: GetC2

此任务每3000秒执行一次，首先通过以下代码构建一个的JSON对象

实际生成的JSON如下所示，

```
{  
  jsonrpc: "2.0",  
  method: "eth_call",  
  id: 1,  
  params: [  
    {  
      to: "0xdf2208d4902aa1ec9a0957132ca86a4e1d40455b",  
      data: "0xd7ec3ad7"  
    },  
    "latest"  
  ]  
}
```

这个JSON对象是一个用于币安智能区块链的RPC（远程过程调用）请求，其各个字段含义如下：

1. `jsonrpc` : 表示使用的JSON RPC协议版本，这里是"2.0"。JSON RPC是一种轻量级的远程过程调用协议，允许发送包含特定命令的请求到以太坊节点。
2. `method` : 指定调用的方法，这里是"eth_call"。`eth_call` 方法用于执行智能合约的函数调用，但不会产生任何区块链上的状态变化。
3. `id` : 请求的唯一标识符，这里是1。这个ID用于区分不同的请求和响应，通常为一个数字或字符串。
4. `params` : 这是一个数组，包含了方法所需的参数。
 - 第一个参数是一个对象，包含两个字段：
 - a. `to` : 指定调用的智能合约地址，这里是"0xdf2208d4902aa1ec9a0957132ca86a4e1d40455b"。
 - b. `data` : 调用合约函数的编码数据，这里是"0xd7ec3ad7"，通常是一个特定函数签名的哈希值。
 - 第二个参数指定了区块的状态，这里使用"latest"表示使用最新区块的状态来执行这个调用。

XLab按：

"latest"总是取区块中最新的状态，这个语法糖给自动更新了C2提供了可能性。假设一个场景，C2在IOC库中，都被阻断了。此时作者只需要使用合约更新C2，而Bot无需任何更新，依托latest的性质，重新发送RPC请求，就能拿到新的C2。

生成JSON后，Smargaft会在14个硬编码的RPC节点中随机选择一个，发起请求并解析返回的JSON对象中的'result'值。

等效的脚本如下所示

```
curl -X POST URL -d '{ "jsonrpc": "2.0", "method": "eth_call", "id": 1, "params": [
```

以 `https://rpc.ankr.com/bsc` 为例，将上文脚本中的URL替换为 `https://rpc.ankr.com/bsc`，运行后就能直接得到以下result值。

接着从result中读出C2，处理逻辑为从result的第122字节开始读，直到第一个非0的数字；然后以hexstring的方式解释非0位置之后的数据，第一个字节为C2长度，随后的数据为C2列表，以 ; 分隔。

00000000 2b 34 35 2e 39 35 2e 31 34 36 2e 39 33 3b 39 34 |+45.95.146.93;94|
00000010 2e 31 30 33 2e 31 38 38 2e 31 36 37 3b 31 38 35 |.103.188.167;185|
00000020 2e 31 33 32 2e 31 32 35 2e 31 39 33 |.132.125.193|

最后Bot和C2的81端口建立连接，发送6个字节的上线包 `ready\x00`，并通过回包长度是否大于等于0验证C2存活状态，首个存活的C2将被用于下一步的DDoS任务。

0x2: DDoS Task

此任务每8秒执行一次，通过以下代码和C2的81端口建立连接，发送5字节的上线包 ready，接收C2下发的指令，支持执行系统命令，DDoS攻击，socks5代理等功能。

通过v205这个变量的交叉引用可以看出通信协议为文本类型，一共支持15个不同的指令。

以下为各个指令与其对应的功能：

CMD	FUNCTION
ack	DDoS Vector
syn	DDoS Vector
gre	DDoS Vector
tcph	DDoS Vector
udpg	DDoS Vector
udph	DDoS Vector
httph	DDoS Vector
stomp	DDoS Vector
spoof_vse	DDoS Vector
spoof_syn	DDoS Vector
socket	DDoS Vector
socks	socks5 proxy
kill	kill self
exec	exec system cmds
update	bot update

以实际捕获的攻击为例：

当Bot接收到上面的指令，就会使用udph攻击向量对43.249.192.173的17481端口发起DDoS攻击。

0x3: Ip-Forward Task

此任务每100小时执行一次，首先通过将 `/proc/sys/net/ipv4/ip_forward` 置为1，开启据包转发的功能。

接着Bot会向C2的8083端口发送随机生成的256字节的UDP报文；如果Bot在root权限下运行，还会构造一个源地址为1.1.1.1的的UDP报文，发送到C2的8083端口。

0x4: Persistence Task

此任务每100小时执行一次，主要目的有俩个。

1. 通过 `Interfere_proc` 函数，将 `tmpfs, devpts, minix, sysfs` 中的任意一个挂载到bot进行的`/proc/pid`目录上，使得依赖于`/proc`文件系统的工具都无法正常获取bot进程的信息。
以netstat为例，实际效果如下，可以看出已经无法正常显示进程的的PID，Program Name等信息了。
2. 通过 `Infect_sh` 函数实现病毒式感染指定目录中的后缀为sh的文件，感染方式是在该类文件的尾部加上 `\n\n\n bot absolute path &\n`，如此一来每次该类脚本运行时，bot也会得到启动的机会。该函数的第一个参数为目录，第二个参数为感染的最大层数。

为了演示感染效果，我们在`/home`目录下，创建了`goat`目录。

```
./goat/
├── 1.sh
└── 3layer
    ├── 2.sh
    └── 4layer
        └── 3.sh
```

实际中的对goat目录和/etc目录的感染情况如下所示，可以看出大量.sh文件尾部被写入了执行bot的语句 `/home/kali/sample/main.x86.unp &`，而 `/home/goat/3layer/4layer/3.sh` 之所以没有被感染，是因为它所在的目录深度已超过3。

0x5: Killer Task

此任务每1秒执行一次，处理逻辑为通过`/proc/[PID]/cmdline` 获取进行的命令行，如果带有`-sh`，或`tftp`，则直接结束进程。

实际效果如下，可以看到tftp进程被结束。

总结

Smargaft使用的智能合约托管技术，EtherHiding，相比传统托管方法，展现出了明显的优势。它充分利用了区块链的核心特性，包括去中心化、透明性和不可篡改性。在这种模式下，托管过程中无法在区块链层面实施任何干预或阻断，使其成为了一种更高级别的“防弹”托管技术。智能合约技术本身的学习门槛并不高，相信未来更多恶意软件开发者会将这一技术做为常规武器。Stay Vigilant, Stay Safe.

联系我们

对我们的研究感兴趣的读者，可以通过[Twitter](#)与我们联系。同时我们也有一个不情之请，在DDoS统计中，明显能看出数个僵尸网络协同攻击的情况，但是我们对DDoS黑产的生态缺乏深入的了解，欢迎了解内幕读者与我们分享。

IOC

sample md5

```
version:v0
```

```
ab794804d7ff5b60327c5051281af80d  
e1967081d11debe9757b96a5bb350fa6  
b9a9ce54a3b695adcafa9afa556ffac9  
38f441481e8765112fb83e556fb33076  
7b8c1aa92861e24c81376c6ccb620da5  
78bfeff2b2303b9fa51b087c4d762687  
7f741495f14c828c20db4de6251673fd
```

```
version: v1
```

```
8171c8238bd465772b70b943305279cd  
323658f0151ec7b7009523e2c368963b  
ae64e92bb05ab7ece41f45c79d8e3e6e  
88996cbc2658d1294e55863a9fefafaf  
dc9cdae709ce4fdf1ceb8b70dd108d36
```

C2(port:81)

```
45.95.146.93 The Netherlands|Noord-Holland|Amsterdam AS49870|Alsycon B.V.  
94.103.188.167 Moldova|None|None AS200019|ALEXHOST SRL  
185.132.125.193 China|Hongkong|Hongkong AS9009|M247 Europe SRL
```

Downloader(port:82)

```
45.95.146.93 The Netherlands|Noord-Holland|Amsterdam AS49870|Alsycon B.V.  
94.103.188.167 Moldova|None|None AS200019|ALEXHOST SRL  
185.132.125.193 China|Hongkong|Hongkong AS9009|M247 Europe SRL
```

What do you think?

5 Responses



Upvote



Funny



Love



Surprised



Angry



Sad

0 Comments

1 Login ▾

G

Start the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS

Name

1

Share

Best [Newest](#) [Oldest](#)