




# 2021-i春秋-春季赛逆向WP

原创

syj-re  于 2021-06-25 20:57:15 发布  137  收藏

分类专栏: [逆向](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: [https://blog.csdn.net/m0\\_51713041/article/details/118228523](https://blog.csdn.net/m0_51713041/article/details/118228523)

版权



[逆向](#) 专栏收录该内容

28 篇文章 0 订阅

订阅专栏

## 2021-i春秋-春季赛

### 一: backdoor

参考了一些Go语言的Socket编程

<https://www.cnblogs.com/yinzhengjie/p/7261584.html>

<https://studygolang.com/articles/14344>

<https://blog.csdn.net/natpan/article/details/82866619>

<https://blog.csdn.net/RA681t58CJxsgCkKJ31/article/details/103258253>

顾名思义, 就叫后门

发现是用GO语言写的, 使用IDA7.6打开进行分析

```

void __cdecl main_main()
{
    __int64 v0; // [rsp+10h] [rbp-78h]
    __int64 v1; // [rsp+18h] [rbp-70h]
    __int64 v2; // [rsp+20h] [rbp-68h]
    __int64 v3; // [rsp+30h] [rbp-58h]
    __int64 v4; // [rsp+38h] [rbp-50h]
    __int64 v5; // [rsp+48h] [rbp-40h]
    __int64 val; // [rsp+50h] [rbp-38h]
    const __int64 *v7; // [rsp+60h] [rbp-28h] BYREF
    char **v8; // [rsp+68h] [rbp-20h]
    __int128 v9; // [rsp+70h] [rbp-18h]
    __int64 v10; // [rsp+80h] [rbp-8h] BYREF

    while ( (unsigned __int64)v10 <= *(_QWORD *)*(_QWORD *)NtCurrentTeb()->NtTib.ArbitraryUserPointer + 16LL )
        runtime_morestack_noctxt();
    v2 = net_listen((__int64)"tcp", 3LL, (__int64)"127.0.0.1:8721", 14LL);
    if ( v3 )
    {
        v9 = 0LL;
        v7 = &qword_2A7260;
        v8 = &aaErrorConnectin;
        *(_QWORD *)&v9 = *(_QWORD *)(v3 + 8);
        *(_QWORD *)&v9 + 1 = v4;
        v1 = log_fatal((__int64)&v7, 2LL, 2LL);
    }
    while ( 1 )
    {
        val = *(__int64 (__golang **)))(v2 + 24)();
        v5 = v0;
        if ( v1 )
        {
            v9 = 0LL;
            v7 = &qword_2A7260;
            v8 = &aaAcceptingConne;
            *(_QWORD *)&v9 = *(_QWORD *)(v1 + 8);
            *(_QWORD *)&v9 + 1 = v2;
            log_println((__int64)&v7, 2LL, 2LL);
        }
        HIDWORD(v0) = HIDWORD(val);
        v1 = v5;
        runtime_newproc(16u, (char)&off_2D2A90, val);
    }
}

```

使用net\_Listen监听了本机的8721端口，然后下面检测是否监听正确，没有就log\_fatal\*\*，\*\*再下方是读取从监听的端口发送过来的信息，最后调用的off\_2D2A90地址处的main\_handleConnection进行处理

main\_handleConnection函数：

```

void __golang __noreturn main_handleConnection(__int64 a1, __int64 a2)
{
    runtime__type_0 *v2; // rdi
    void *v3; // rsi
    __int64 v4; // rax
    _QWORD *v5; // [rsp+8h] [rbp-78h]
    __int64 v6; // [rsp+8h] [rbp-78h]
    char v7; // [rsp+18h] [rbp-68h]
    __int64 v8; // [rsp+20h] [rbp-60h]
    __int64 v9; // [rsp+28h] [rbp-58h]
    __int64 v10; // [rsp+30h] [rbp-50h]
    unsigned __int64 v11; // [rsp+50h] [rbp-30h]
    __int64 addr; // [rsp+58h] [rbp-28h]
    __int64 cmp_data; // [rsp+60h] [rbp-20h]
    void *retaddr; // [rsp+80h] [rbp+0h] BYREF

    while ( (unsigned __int64)&retaddr <= *(_QWORD *)*(_QWORD *)NtCurrentTeb()->NtTib.ArbitraryUserPointer + 16LL
) )
    {
        runtime_morestack_noctxt();
        runtime_newobject(v2, v3);
        cmp_data = (__int64)v5;
        *v5 = 0xCF6E7633149F46F5LL;
        v5[1] = 0xD33674C27C6FE28ALL;
        v5[2] = 0xF592D63BE79440D9LL;
        v5[3] = 0xD33CF4C0B83E001BLL;
        v5[4] = 0x8B615DC202F30A50LL;
        v5[5] = 0x181C7380D6FF6BBLL;
        v4 = runtime_makeslice((__int64)&dword_2A73E0, 1024LL, 1024LL);
        for ( addr = v4; ; v4 = addr )
        {
            v6 = v4;
            (*(void (**)(void))(a1 + 40))();
            if ( !v9 )
            {
                if ( (unsigned __int64)(v8 - 1) > 0x400 )
                    runtime_panicSliceAcap(a2, v6);
                v11 = v8 - 1;
                v10 = encoding_base32__ptr_Encoding__EncodeToString(encoding_base32_StdEncoding, addr, v8 - 1, 1024LL, v8
, 0LL);
                if ( v9 == 24 ) //验证加密之后的字符串长度是否是24
                {
                    runtime_memequal(); //比较base32加密之后的字符串
                    if ( v7 )
                    {
                        v9 = 1024LL;
                        main_Decrypt(cmp_data, 48uLL, 48LL, addr, v11);
                        if ( v10 )
                            v8 = (*(__int64 (__golang **)(__int64, __int64))(a1 + 80))(a2, v10);
                    }
                }
            }
        }
    }
}

```

```

.text:000000000297B7C      nop     dword ptr [rax+00h]
.text:000000000297B80      cmp     rcx, 400h
.text:000000000297B87      ja     loc_297C79
.text:000000000297B8D      mov     [rsp+80h+var_30], rcx
.text:000000000297B92      mov     rax, cs:encoding_base32_StdEncoding
.text:000000000297B99      mov     [rsp+80h+var_80], rax
.text:000000000297B9D      mov     rax, [rsp+80h+addr]
.text:000000000297BA2      mov     [rsp+80h+var_78], rax
.text:000000000297BA7      mov     [rsp+80h+var_70], rcx
.text:000000000297BAC      mov     [rsp+80h+var_68], 400h
.text:000000000297BB5      call   encoding_base32_ptr_Encoding_EncodeToString
.text:000000000297BBF      mov     rax, [rsp+80h+var_60]
.text:000000000297BC1      mov     rcx, [rsp+80h+var_58]
.text:000000000297BC4      cmp     rcx, 24
.text:000000000297BC8      jnz    loc_297B35
.text:000000000297BCE      mov     [rsp+80h+var_80], rax
.text:000000000297BD2      lea    rax, aM4ydcylom5bgcy ; "M4YDCYLOM5BGCY3LMQYDA4Q="
.text:000000000297BD9      mov     [rsp+80h+var_78], rax
.text:000000000297BDE      mov     [rsp+80h+var_70], rcx
.text:000000000297BE3      call   runtime_memequal
.text:000000000297BE8      cmp     byte ptr [rsp+80h+var_68], 0
.text:000000000297BED      jz     loc_297B35
.text:000000000297BF3      mov     rax, [rsp+80h+cmp_data]
.text:000000000297BF8      mov     [rsp+80h+var_80], rax
.text:000000000297BFC      mov     [rsp+80h+var_78], 30h ; '0'
.text:000000000297C05      mov     [rsp+80h+var_70], 30h ; '0'

```

1 我们从端口发送过去的信息

2 加密之后的

3 比较两个字符串

[https://blog.csdn.net/m0\\_51713041](https://blog.csdn.net/m0_51713041)

如果我们从端口发送过去的信息在base32标准加密之后和"M4YDCYLOM5BGCY3LMQYDA4Q="相等，就进行解密，这里就有两种思路。可以将M4YDCYLOM5BGCY3LMQYDA4Q=进行base32解码然后得到我们发送的数据，为"01angBackd00r"，但是我用nc发的时候，卡了很久，最后看见在这个地方减一

```

if ( !v9 )
{
    if ( (unsigned __int64)(v8 - 1) > 0x400 )
        runtime_panicSliceAcap(a2, v6);
    v11 = v8 - 1;
    v10 = encoding_base32_ptr_Encoding_EncodeToString(encoding_base32_StdEncoding, addr, v8 - 1, 1024LL, v8, 0LL);
    if ( v9 == 24 )
    {
        runtime_memequal();
        if ( v7 )
        {
            v9 = 1024LL;

```

[https://blog.csdn.net/m0\\_51713041](https://blog.csdn.net/m0_51713041)

然后多输入了一个字符，使用nc发过去之后，base32加密后才能和比较数据相等。

发送过程：

先创建一个文件



然后运行backdoor.exe

之后使用nc直接重定向入端口，会反弹回一个flag

```
D:\CTF\COMPETITIONS\2021ichunqiu\backdoor>nc 127.0.0.1 8721 < a.txt
flag {93b867f2-62b6-760a-7acb-355c80281e12}
```

另外一种思路就是静态解，加密后的数据我们知道

```
cmp_data = [0x149F46F5, 0xCF6E7633, 0x7C6FE28A, 0xD33674C2, 0xE79440D9, 0xF592D63B, 0xB83E001B, 0xD33CF4C0, 0x02F30A50, 0x8B615DC2, 0x0D6FF6BB, 0x0181C738]
```

跟进去看是什么解密

进去之后再跟进

```

64 v33 = v10;
65 input_ = runtime_makeslice((__int64)&byte_2A7360, v10, v10);
66 for ( j = 0; j < (unsigned int)a5; j = v13 + 1 )
67 {
68     v13 = j;
69     v14 = j >> 2;
70     if ( v14 >= v33 )
71         runtime_panicIndex(v23, v27);
72     if ( v13 >= a5 )
73         runtime_panicIndex(v23, v27);
74     *(_DWORD *)(input_ + 4 * v14) |= 8 * (v13 & 3) < 0x20 ? *(unsigned __int8 *)(addr + (unsi
75 }
76 main_decrypt(cmp, n, n, input_, v33);
77 v15 = (unsigned int)(v31 - 1);
78 if ( v31 <= v15 )
79     runtime_panicIndex(v24, v28);
80 v16 = *(unsigned int *)(v30 + 4 * v15);
81 if ( (unsigned int)v16 < 4 * (int)v31 - 7 || (unsigned int)v16 > 4 * (int)v31 - 4 )
82     return 0LL;
83 v32 = v16;
84 result = runtime_makeslice((__int64)&dword_2A73E0, v16, v16);
85 for ( k = 0; v32 > k; k = p + 1 )

```

000D7671 main.Decrypt:76 (298071) | [https://blog.csdn.net/m0\\_51713041](https://blog.csdn.net/m0_51713041)

可以很明显的看见是一个xtea

```

v9 = a2 - 1;
for ( i = 0x9E3779B9 * (0x34 / (unsigned int)a2) - 0x4AB325AA; i; i += 1640531527 )
{
    v20 = (i >> 2) & 3;
    v21 = v9;
    while ( v9 )
    {
        v11 = v9 - 1;
        if ( a2 <= v11 )
            runtime_panicIndex((__int64)v23, v24);
        v12 = (((v7 >> 3) ^ (16 * *(_DWORD *)(cmp + 4 * v11))) + ((4 * v7) ^ (*(_DWORD *)(cmp + 4 * v11) >> 5)));
        v13 = i ^ v7;
        v14 = v9;
        key_index = v20 ^ v9 & 3;
        if ( key_index >= v8 )
            runtime_panicIndex((__int64)v23, v24);
        v16 = v12 ^ ((*((_DWORD *)key + key_index) ^ *(_DWORD *)(cmp + 4 * v11)) + v13);
        if ( a2 <= v14 )
            runtime_panicIndex((__int64)v23, v24);
        v7 = *(_DWORD *)(cmp + 4 * v14) - v16;
        *(_DWORD *)(cmp + 4 * v14) = v7;
        v9 = v11;
    }
}

```

[https://blog.csdn.net/m0\\_51713041](https://blog.csdn.net/m0_51713041)

```

runtime_panicIndex((__int64)v23, v24);
(*((_DWORD *)key + key_index)
14 )

```

然后所谓的key，就是把通过端口传过去的字符串四个字节四个字节的取了当key

可以得到：

```
unsigned int key[] = {0x61313067,0x6142676E,0x30646B63,0x7230};
```

写出解题脚本：

```

#include <stdio.h>
#include <stdlib.h>
#define DELTA 0x9e3779b9//0x61C88647
int main()
{
    unsigned int v[] = {0x149F46F5, 0xCF6E7633, 0x7C6FE28A, 0xD33674C2, 0xE79440D9, 0xF592D63B, 0xB83E001B, 0xD3
3CF4C0, 0x02F30A50, 0x8B615DC2, 0x0D6FF6BB, 0x0181C738};
    unsigned int key[] = {0x61313067,0x6142676E,0x30646B63,0x7230};
    unsigned int sum = 0;
    unsigned int y,z,p,rounds,e;
    int n = 12; //传入的是12
    int i = 0;
    rounds = 6 + 52/n;
    y = v[0];
    sum = (rounds*DELTA)&0xffffffff;
    do
    {
        e = sum >> 2 & 3;
        for(p=n-1;p>0;p--)
        {
            z = v[p-1];
            v[p] = (v[p] - (((z>>5)^(y<<2))+(y>>3)^(z<<4))) ^ ((key[(p^e)&3]^z)+(y ^ sum))) & 0xffffffff;
            y = v[p];
        }
        z = v[n-1];
        v[0] = (v[0] - (((key[(p^e)&3]^z)+(y ^ sum)) ^ ((y<<2)^(z>>5))+((z<<4)^(y>>3)))) & 0xffffffff;
        y = v[0];
        sum = (sum-DELTA)&0xffffffff;
    }while(--rounds);
    for(i=0;i<n;i++)
    {
        printf("%c%c%c%c",*((char*)&v[i]+0),*((char*)&v[i]+1),*((char*)&v[i]+2),*((char*)&v[i]+3));
    }
    //flag{93b867f2-62b6-760a-7acb-355c80281e12}
    return 0;
}

```

## 二: hardchal

题给提示:

```

I designed a very HARD challenge for you, please enjoy~
How-to-run:
1. Install Icarus Verilog
2. > vvp ./chal

```

查看Icarus Verilog的文档，从中看一些我们分析时用的到的

<https://github.com/steveicarus/iverilog/blob/master/wp/README.txt#L852>

比如: 参数, 函数, 变量等的规范

根据最开始的一些值, 初步判断是TEA一类

```
> CTF > COMPETITIONS > 2021ichunqiu > chal
80 S_00000201bac04ae0 .scope module, "main" "main" 2 1603;
81 .timescale 0 0;
82 _v0 .var "_v0", 31 0;
83 _v1 .var "_v1", 31 0;
84 char_idx .var/i "char_idx", 31 0;
85 v00000201bba697c0_0 .var "clk", 0 0;
86 delta .var "delta", 31 0;
87 v00000201bba69720_0 .net "done", 0 0, L_00000201bbc6b2d0; 1 drivers
88 enc_v0 .net "enc_v0", 31 0, L_00000201bb80a1d0; 1 drivers
89 enc_v1 .net "enc_v1", 31 0, L_00000201bb80b510; 1 drivers
90 v00000201bba68b40_0 .var/i "file", 31 0;
91 i .var/i "i", 31 0;
92 input_char .var/s "input_char", 31 0;
93 k0 .var " ", 31 0;
94 k1 .var "k1", 31 0;
95 k2 .var "k2", 31 0;
96 k3 .var "k3", 31 0;
97 ok .var/i "ok", 31 0;
98 reset .var "reset", 0 0;
99 round .var/i "round", 31 0;
100 v00000201bba69e00_0 .var/s "value1", 31 0;
101 v00000201bba694a0_0 .var/s "value10", 31 0;
```

[https://blog.csdn.net/m0\\_51713041](https://blog.csdn.net/m0_51713041)

然后在下方代码的开始处找到了比较数据:

```
File Edit Selection View Go Run Terminal Help • chal - Visual Studio Code
chal
D: > CTF > COMPETITIONS > 2021ichunqiu > chal
15238 .scope S_00000201bac04ae0;
15239 T_3; //比较第一部分
15240 %wait E_00000201bb7ee420;
15241 %load/vec4 round;
15242 %cmpi/e 0, 0, 32;
15243 %jmp/0xz T_3.0, 4;
15244 %load/vec4 enc_v0;
15245 %cmpi/ne 3208527578, 0, 32;
15246 %flag_mov 8, 4;
15247 %load/vec4 enc_v1;
15248 %cmpi/ne 423585179, 0, 32;
15249 %flag_or 4, 8;
15250 %jmp/0xz T_3.2, 4;
15251 %pushi/vec4 0, 0, 32;
15252 %store/vec4 ok, 0, 32;
15253 T_3.2 ;
15254 %load/vec4 v00000201bba68fa0_0;
15255 %store/vec4 _v0, 0, 32;
15256 %load/vec4 v00000201bba688c0_0;
15257 %store/vec4 _v1, 0, 32;
15258 T_3.0 ; //比较第二部分
15259 %load/vec4 round;
15260 %cmpi/e 1, 0, 32;
15261 %jmp/0xz T_3.4, 4;
15262 %load/vec4 enc_v0;
15263 %cmpi/ne 699878777, 0, 32;
15264 %flag_mov 8, 4;
15265 %load/vec4 enc_v1;
15266 %cmpi/ne 1677098023, 0, 32;
15267 %flag_or 4, 8;
```

[https://blog.csdn.net/m0\\_51713041](https://blog.csdn.net/m0_51713041)

像这样的总共:



```
0xbf3e3eda,0x193f659b,0x29b74d79,0x63f67c27,0x6330fb62,0xce7d5340,0xd29371e9,0xae2f0140,0x87ab6341,0x2069c5d,0xe10392fe,0xbb0e1442
```

然后紧接着根据比较的结果判断结果

```
339      %Tflag_or 4, 0;
340      %jmp/0xz right_label, 4;
341      %pushi/vec4 0, 0, 32;
342      %store/vec4 ok, 0, 32;
343  v right_label ;                               //Congratulations
344      %load/vec4 ok;
345      %cmpi/ne 0, 0, 32;
346      %jmp/0xz fault_label, 4;
347      %vpi_call 2 1661 "$write", "Congratulations~\012" {0 0 0};
348      %jmp finish_label;
349  v fault_label ;                               //Invalid flag
350      %vpi_call 2 1663 "$write", "Invalid flag\012" {0 0 0};
351  v finish_label ;
352      %vpi_call 2 1664 "$finish" {0 0 0}; https://blog.csdn.net/m0\_51713041
```

然后继续往下:

```
T_5 ;
%pushi/vec4 0, 0, 1;
%store/vec4 v00000201bba697c0_0, 0, 1;
%pushi/vec4 0, 0, 32;
%store/vec4 char_idx, 0, 32; //char_idx = 0
%pushi/vec4 1, 0, 1;
%store/vec4 reset, 0, 1; //reset = 1
%pushi/vec4 1, 0, 32;
%store/vec4 ok, 0, 32; // ok = 1
%pushi/vec4 2654435769, 0, 32;
%store/vec4 delta, 0, 32; //delta = 2654435769
%pushi/vec4 3735928559, 0, 32;
%store/vec4 k0, 0, 32; // k0 = 3735928559
%pushi/vec4 3405691582, 0, 32;
%store/vec4 k1, 0, 32; // k1 = 3405691582
%pushi/vec4 269488144, 0, 32;
%store/vec4 k2, 0, 32; // k2 = 269488144
%pushi/vec4 16843009, 0, 32;
%store/vec4 k3, 0, 32; // k3 = 16843009
%vpi_call 2 1771 "$write", "Checking, please wait...\012" {0 0 0};
%vpi_func 2 1772 "$" 32, "flag" {0 0 0};
%store/vec4 file, 0, 32;
%load/vec4 file;
%cmpi/e 0, 0, 32;
%jmp/0xz T_5.0, 4;
%vpi_call 2 1774 "$write", "Invalid flag\012" {30 0 0};
%vpi_call 2 1775 "$finish" {0 0 0};
```

这里得到了delta和key，分别是：

```
0x9e3779b9
0xdeadbeef,0xcafebabe,0x10101010,0x10101010
```

然后根据delta，以及逻辑分析，可以确定就是TEA，且并没有其它什么怪异的操作

直接写出C脚本得到flag：

```
#include <stdio.h>
#include <stdint.h>

//加密函数
void encrypt (uint32_t* v, uint32_t* k) {
    uint32_t v0=v[0], v1=v[1], sum=0, i; /* set up */
    uint32_t delta=0x9e3779b9; /* a key schedule constant */
    uint32_t k0=k[0], k1=k[1], k2=k[2], k3=k[3]; /* cache key */
    for (i=0; i < 32; i++) { /* basic cycle start */
        sum += delta;
        v0 += ((v1<<4) + k0) ^ (v1 + sum) ^ ((v1>>5) + k1);
        v1 += ((v0<<4) + k2) ^ (v0 + sum) ^ ((v0>>5) + k3);
    } /* end cycle */
    v[0]=v0; v[1]=v1;
}

//解密函数
void decrypt (uint32_t* v, uint32_t* k) {
    uint32_t v0=v[0], v1=v[1], sum=0xC6EF3720, i; /* set up */
    uint32_t delta=0x9e3779b9; /* a key schedule constant */
    uint32_t k0=k[0], k1=k[1], k2=k[2], k3=k[3]; /* cache key */
    for (i=0; i<32; i++) { /* basic cycle start */
        v1 -= ((v0<<4) + k2) ^ (v0 + sum) ^ ((v0>>5) + k3);
        v0 -= ((v1<<4) + k0) ^ (v1 + sum) ^ ((v1>>5) + k1);
        sum -= delta;
    } /* end cycle */
    v[0]=v0; v[1]=v1;
}

int main()
{
    uint32_t array[] = {0xbf3e3eda,0x193f659b,0x29b74d79,0x63f67c27,0x6330fb62,0xce7d5340,0xd29371e9,0xae2f0140,
0x87ab6341,0x2069c5d,0xe10392fe,0xbb0e1442};
    uint32_t key[4] = {0xdeadbeef,0xcafebabe,0x10101010,0x10101010};
    int i = 0;
    for(i=0;i<12;i+=2)
    {
        uint32_t temp[2];
        temp[0] = array[i];
        temp[1] = array[i+1];
        decrypt(temp, key);
        printf("%c%c%c%c%c%c%c",*((char*)&temp[0]+3),*((char*)&temp[0]+2),*((char*)&temp[0]+1),*((char*)&temp[
0]+0),*((char*)&temp[1]+3),*((char*)&temp[1]+2),*((char*)&temp[1]+1),*((char*)&temp[1]+0));
    }
    //flag{d88ca56a-b934-11eb-8529-0242ac130003}
    return 0;
}
```



[创作打卡挑战赛](#) >

[赢取流量/现金/CSDN周边激励大奖](#)