[原创]分享一个对抗JUMPOUT的小技巧

angelToms 4 2

大牛

2020-4-22 14:52

本人小菜一枚，虽然技术不咋地，但是对一些事物比较有耐心并且善于观察，偶尔会因为自己的细心获得一些额外的小收获；今天我就来分享一个小技巧，不是什么高大上的技巧，但相信很多人在逆向过程中都遇见过；该技巧用来对抗JUMPOUT，它对于我这样的小白来说还是挺有用的，如果你和我一样小白可以以这种方法试试，大佬嘛， 你就一笑而过吧，哈哈！

前几天看见有人发sig3算法的帖子，帖子是好帖子写的不错；算法中对于so的保护用到了和阿里差不多的技术，技术细节这里不在赘述了；但看见有人用临时账户留言说是四年前的技术，于是勾起了我的好奇心；回想过去曾经尝试逆向sgmain三个不同版本6.3.80、6.4.36、6.4.176，我内心的感觉是老版本破解难度要远远强于新版本，老版本应用了很多技术，就混淆这块就有cfg对抗(我也不知道专业人士怎么叫)、动态跳转、跳转表、动态参数、fla、很多混淆花指令等，而新版本好像只用了fla、 cfg对抗技术。于是我下载了一个比较新的版本6.4.1229来看，发现和6.4.176没啥变化。

那既然变化不大，看来四年前的技术不是指sgmain那一套，可能指的新技术指的是avmp吧，可是我怎么都感觉avmp和livteVM纠缠在一起，本人实在才疏学浅，就不钻牛角尖了。

进入正题，现在我就来分享这个对抗技巧，关于它是啥原理我这里就不再描述了，想得到专业描述你最好google去；先看一下对抗前的效果图（以最新的 6.4.1229 为例）：

```
void JNI_OnLoad()
{
  int v0; // ST30_4@1

  v0 = *(_DWORD *)off_AF58C;
  JUMPOUT(&unk_F5D0);
}
```

```
; Attributes: bp-based frame

EXPORT JNI_OnLoad
JNI_OnLoad

var_24= -0x24

PUSH            {R4-R7,LR}
ADD             R7, SP, #0xC
SUB             SP, SP, #0x34
MOV             R6, SP
ADDS            R2, R6, #7
ADDS            R2, #0x21 ; '!'
STR             R0, [R6,#0x40+var_24]
LDR             R0, =(off_AF58C - 0xF5A4)
ADD             R0, PC ; off_AF58C
LDR             R0, [R0]
LDR             R0, [R0]
STR             R0, [R2,#8]
MOVS            R0, #0x9D ; '
STR             R0, [R2]
ADDS            R0, R6, #7
ADDS            R0, #0x21 ; '!'
ADR             R5, 0xF5B4
NOP
LDR             R4, =0xFFFFFF80
LDR             R1, [R0]
ADDS            R4, R4, R1
ADDS            R5, R5, R4
MOVS            R0, #0x16
BX              R5
; End of function JNI_OnLoad
```

从上面两幅图，我们可以清楚的看到ida无法给我们分析出完成的cfg了，但是别急ida要远比我们想象的聪明，你看第一幅图

```
  v0 = *(_DWORD *)off_AF58C;
  JUMPOUT(&unk_F5D0);
}
```

ida给我们算出了一个JMMPOUT的地址，这里是F5D0，这个F5D0是不是值得思考一下，在看汇编代码

```
STR             R0, [R2]
ADDS            R0, R6, #7
ADDS            R0, #0x21 ; '!'
ADR             R5, 0xF5B4
NOP
LDR             R4, =0xFFFFFF80
LDR             R1, [R0]
ADDS            R4, R4, R1
ADDS            R5, R5, R4
MOVS            R0, #0x16
BX              R5
; End of function JNI_OnLoad
```

不巧，这个BX R5的最终结果就是这个 F5D0，哦，哦，哦，扫噶，看来ida还是挺聪明的，那我们把它patch成跳转到F5D0后会怎么样呢？我先给你们附上代码：

```
def put_unconditional_branch(source, destination):
    offset = (destination - source - 4) >> 1
    if offset > 2097151 or offset < -2097152:
        raise RuntimeError("Invalid offset")
    if offset > 1023 or offset < -1024:
        instruction1 = 0xf000 | ((offset >> 11) & 0x7ff)
        instruction2 = 0xb800 | (offset & 0x7ff)
        PatchWord(source, instruction1)
        PatchWord(source + 2, instruction2)
    else:
        instruction = 0xe000 | (offset & 0x7ff)
        PatchWord(source, instruction)



put_unconditional_branch(here(), 0xF5D0)
```

没错就修改代码这一行 `put_unconditional_branch(here(), 0xF5D0)`

修改后我们f5发现，ida没变化，是ida喝多了嘛？不不不，我们需要继续帮助ida分析；我们把鼠标放在JNI_OnLoad块上，点ida的edit functions->delete function, 然后在 JNI_OnLoad起始处按p（创建函数），在f5，哦，不错哦，效果如下（屏幕太大截取一部分）：

```
{
  v5 = v3;
LABEL_11:
  while ( 1 )
  {
    v3 = v5;
    if ( v5 == 170 )
      break;
    v27 = (signed int *)v2;
    v24 = v5;
    sub_8B4D0(v28, v1);
    v6 = sub_8B580(0);
    if ( !v6 )
    {
      v25 = 55;
      v31 = 16;
      JUMPOUT(__CS__, (char *)&loc_F634 + (v30 ^ 0x16));
    }
    v7 = sub_82B48();
    sub_83494(v7);
    sub_83180(v6);
    sub_83A74((char *)&word_A86BE + 1, &v21 - 14, 49);
    v8 = (*(int (__fastcall **)(int, int *))(*(_DWORD *)v6 + 24))(v6, &v21 - 14);
    v2 = 184;
    if ( !v8 )
      goto LABEL_27;
    v29 = v8;
    sub_E950();
    sub_8199C(v6);
    v9 = sub_15FF8(v6);
    v2 = 184;
    v5 = 170;
    if ( v9 >= 0 )
    {
      v10 = sub_7D198(v6);
      v2 = 184;
      v1 = 170;
      if ( v10 < 0 )
        goto LABEL_6;
      v11 = v29;
      v12 = sub_2F420(v6, v29);
      v2 = 184;
      if ( v12 < 0 )
      {
LABEL_27:
        v3 = v24;
        break;
      }
      v27 = dword_64 + 84;
      if ( sub_158E0(v6, v11) < 0 )
      {
        v3 = v24;
        v2 = (signed int)v27;
        break;
      }
      v13 = sub_FD70(v6, v11);
      v2 = 184;
      v5 = 170;
      if ( v13 >= 0 )
      {
        v14 = v29;
        v15 = sub_721B0(v6, v29);
        v2 = 184;
        v1 = 170;
        if ( v15 < 0 )
          goto LABEL_6;
```

我们发现ida帮我们识别了很大一部分code，**这里有个细节需要说明一下，你f5后可能没有达到我这种效果，可能是这样的：**

```
unsigned int __fastcall JNI_OnLoad(int a1)
{
  unsigned int result; // r0@1
  void **v20; // r1@1
  int v24; // [sp+10h] [bp-30h]@8
  int v27; // [sp+1Ch] [bp-24h]@1
  signed int v29; // [sp+28h] [bp-18h]@1
  int v31; // [sp+30h] [bp-10h]@1

  v27 = a1;
  v31 = *(_DWORD *)off_AF58C;
  v29 = 157;
  result = (unsigned int)sub_10004 | (sub_8CA94(62929) >> 31);
  v20 = &off_AF58C;
  if ( *(_DWORD *)*v20 != *(_DWORD *)(v24 + 8) )
    _stack_chk_fail(result);
  return result;
}
```

为什么呢？原因在于ida在重新建立函数时遇到其他函数头分析就结束了，我们需要帮助它，先删除那些被ida错误分析的函数块，在通过ALT+P编辑函数修改JNI_OnLoad的函数尾；这里我就传授给你们一下ALT+P的使用大法，一般人你发现不了，哈哈

// ida alt+p 修改函数开始和结束地址；如修改结束地址，这个地址必须不再另外一个函数内，如果在需要先删除那个函数

// 在修改，才能成功

f5和我图中一样效果后，我们发现还是有一些JUMPOUT，而且这次的JUMPOUT没有给出要跳转的具体偏移；看来ida只能计算直接赋值的情况，对于间接赋值的，它无能为力，没关系，我们帮助它，

看这个JUMPOUT

```
v30 = 16;
JUMPOUT(__CS__, (char *)&loc_F634 + (v29 ^ 0x16));
```

它给我们算出了一部分F636，但是后面的加值（v29）它不知道，我们跳转到 F636看看汇编代码

```
MOVS            R0, #0x37 ; '7'
LDR             R1, [R6,#0x10]
STR             R0, [R1]
ADDS            R0, R6, #7
ADDS            R0, #0x21 ; '!'
ADDS            R3, R6, #7
ADDS            R3, #0x25 ; '%'
ADR             R2, 0xF634


LDR             R4, =0xCA
SUBS            R4, #0xB4 ; '
LDR             R5, [R0]
EORS            R4, R5
ADDS            R2, R2, R4
MOVS            R1, #0x10
STR             R1, [R3]
BX              R2
```

r4 = ca - b4 = 0x16，eors r4， r5，看来它不知道r5的值（这个r5就对应伪代码中的v29），r5来自r0，追溯到函数顶部r0 = 0x9d， 即0x9d ^ 0x16 = 8b, 8b + F634 = F6BF(具体追溯过程我就不讲了，都很好找)，继续patch，继续删函数继续创建函数(主要thumb还是arm，注意偏移)，哟，效果不错哦，只剩下一个JUMPOUT了：

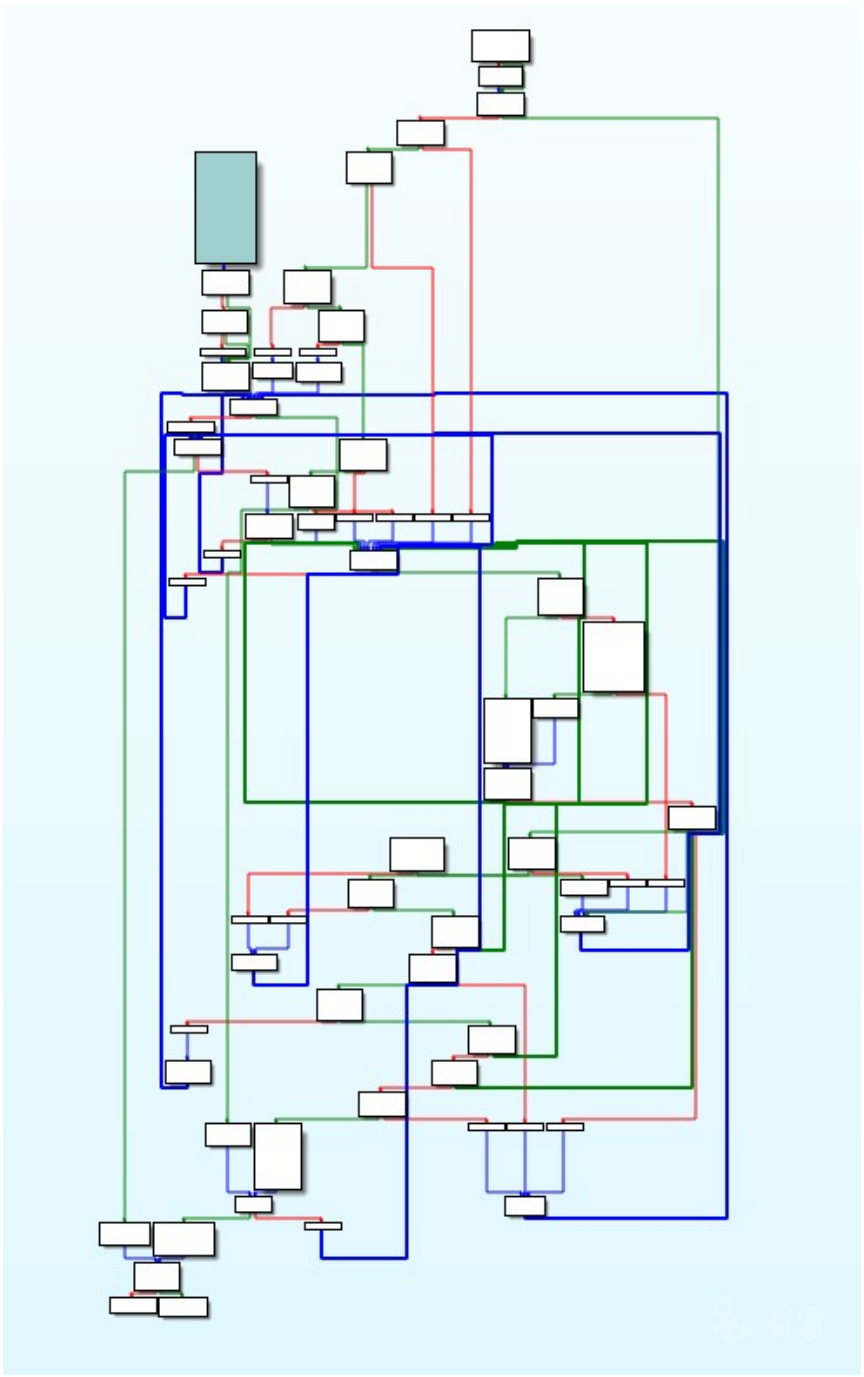
```
v21 = v7;
v31 = 117;
JUMPOUT(__CS__, v30 + 63391);
}
```

```
MOVS            R0, #6
LDR             R1, [R6,#0x10]
STR             R0, [R1]
ADDS            R0, R6, #7
ADDS            R0, #0x21 ; '!'
STR             R0, [R6,#0x18]
ADDS            R1, R6, #7
ADDS            R1, #0x25 ; '%'
STR             R5, [R6]
LDR             R5, [R6,#0x18]
ADR             R2, 0xF780
NOP
LDR             R3, =0x1F
LDR             R4, [R5]
ADDS            R3, R3, R4
ADDS            R2, R2, R3
MOVS            R0, #0x75 ; 'u'
STR             R0, [R1]
BX              R2
```

继续步骤二操作，63391对应16进制数为F79F，很明显 ida算出了0xF780 + 0x1f = F79F，但它不知道r4的值，我们需要帮助它确定（也就是伪代码中v30的值）

```
MOVS            R0, #6
LDR             R1, [R6,#0x10]
STR             R0, [R1]
ADDS            R0, R6, #7
ADDS            R0, #0x21 ; '!'
STR             R0, [R6,#0x18]
ADDS            R1, R6, #7
ADDS            R1, #0x25 ; '%'
STR             R5, [R6]
LDR             R5, [R6,#0x18]
ADR             R2, 0xF780
NOP
```

很明显r5还是0x9d，所以F79F + 0x9d = F83C， 继续patch，继续删函数继续创建函数(主要thumb还是arm，注意偏移)，哟，效果不错哦，好像没有JUMPOUT了哦：



我说这是fla，你还有什么话说，哈哈， 确实这个也是4年前的技术，哈哈，伪代码太多，这里就不截图了，但我随便截一个内部的函数伪代码的图吧：

```
dword_D226C = sub_8CD04(a1, "java/lang/Integer");
dword_D2270 = sub_8CD04(v1, "java/lang/Float");
dword_D2274 = sub_8CD04(v1, "java/lang/String");
dword_D2278 = sub_8CD04(v1, &dword_8CC34);
dword_D227C = sub_8CD04(v1, "java/util/HashMap");
dword_D2280 = sub_8CD04(v1, "java/util/Set");
if ( dword_D2280 && dword_D226C && dword_D2270 && dword_D2274 && dword_D2278 && dword_D227C )
{
  dword_D2284 = ((int (__fastcall *)(_JNIEnv *))v1->functions->GetMethodID)(v1);
  dword_D2288 = ((int (__fastcall *)(_JNIEnv *, int, int *, const char *))v1->functions->GetMethodID)(
                  v1,
                  dword_D227C,
                  &dword_8CC8C,
                  "(Ljava/lang/Object;)Ljava/lang/Object;");
  dword_D228C = ((int (__fastcall *)(_JNIEnv *, int, const char *, const char *))v1->functions->GetMethodID)(
                  v1,
                  dword_D227C,
                  "<init>",
                  "(I)V");
  dword_D2290 = ((int (__fastcall *)(_JNIEnv *, int, int *, const char *))v1->functions->GetMethodID)(
                  v1,
                  dword_D227C,
                  &dword_8CCAC,
                  "(Ljava/lang/Object;Ljava/lang/Object;)Ljava/lang/Object;");
  v2 = ((int (__fastcall *)(_JNIEnv *, int, const char *, const char *))v1->functions->GetMethodID)(
         v1,
         dword_D2280,
         "toArray",
         "()[Ljava/lang/Object;");
  v3 = v2;
  dword_D2294 = v2;
```

至此完美收工，技术要点：

1、遇见JUMPOUT就patch

2、ida会帮我们解析出一些跳转的偏移，它解析不了的，我们需要帮助它

3、记得删除函数块和新建函数块，同时有些时候你需要帮助ida确定函数尾

4、我的代码不能patch所有地方，每遇到一个JUMPOUT，你需要修改偏移，然后手动patch（因为它们的JUMPOUT特征不固定，想完美patch，需要完美的patch代码，哈哈，不过它一个函数JUMPOUT的地方还不是很多，你按我的方法做也不累）

5、至于fla嘛，你随意吧，网络上对抗fla的帖子还是挺多的，另外想动态调试可以忽略它

最后于 2020-4-22 14:54 被angelToms编辑，原因:

最新回复 (23)

上海刘一刀 2 2020-4-22 15:34 2楼

学习了 数字壳也很多 jumpout

大牛

FraMeQ 2020-4-22 15:41 3楼

mark

大侠

wx_0xC05StackOver 2020-4-22 15:45 4楼

。。。

极客

最后于 2020-4-22 15:52 被wx_0xC05StackOve

hasking 2020-4-22 15:47 5楼

学到了 谢谢。 ida把寄存器调用翻译成 jmpout 是算优化了吧

极客

又见飞刀z 2020-4-22 15:53 6楼

几个月前的问题终于得到了解决，感谢大佬

极客

壹久玖 2020-4-22 16:12 7楼

mark

极客

最新回复 (23)

Editor 2020-4-22 16:29 8 楼

版主

感谢分享哦～

万抽抽 2 2020-4-22 20:59 9 楼

大牛

看好楼主！因为某些原因没能跟楼主共事，但技术能力是杠杠的，有需要逆向安全人才的大佬们，赶快联系吧。

angelToms 2 2020-4-22 21:09 10 楼

大牛

> 万抽抽 看好楼主！因为某些原因没能跟楼主共事，但技术能力是杠杠的，有需要逆向安全人才的大佬们，赶快联系吧。

哈哈，感谢大佬抬爱

winkar 2020-4-23 10:21 11 楼

极客

想请教一下楼主，liteVm是什么？
之前在sgmain里看到过相关的代码，但没找到相关的调用，所以没深究。后来搜了搜也没找到相关的资料。

Fireeye 2020-4-23 11:08 12 楼

极客

mark

0x指纹 3 2020-4-23 15:27 13 楼

大牛

感谢分享!

感谢你曾来过 2020-4-23 16:12 14 楼

临时

求教 啥事fla和cfg 好多技术名词看不懂

winkar 2020-4-23 16:18 15 楼

极客

> 感谢你曾来过 求教 啥事fla和cfg 好多技术名词看不懂

fla是ollvm的一个混淆选项，控制流平坦化。
cfg是控制流图。IDA会把代码划分成基本块，再加上单向边组成图。这个图就是CFG。

最新回复 (23)

大牛

angelToms 2 2020-4-23 19:35 16 楼

> winkar 想请教一下楼主，liteVm是什么？ 之前在sgmain里看到过相关的代码，但没找到相关的调用，所以没深究。后来搜了搜也没找到相关的资料。

35

6

¥

liteVM是什么，首先大佬莫笑，我也不知道，可以猜想一下：

1、有没有调用，可以试着在sdk中删除相关java代码，跑一跑

2、sdk源码暴露给我们的是它和InvocationHandler、Proxy有关，用Class.forName显示的加载了一下，没有也没关系

3、在库中你可以找到它的创建code，可能需要还关注一下bb2i34u32clsb

4、另外有些结构比较有意思，它注册了很多，例如：

```
// DCD aJnigetversion      ; "JNIGetVersion"
// DCD ndiu34h834f+1
// DCD 1
// DCD 1
// DCD 1
// DCD aJnifindclass       ; "JNIFindClass"
// DCD ndi3uq4fh024+1
// DCD 1
// DCD 1
// DCD 1
// DCD aFopen_0            ; "fopen"
// DCD __imp_fopen
// DCD 2
// DCD 1
// DCD 0
// DCD aFreopen_0          ; "freopen"
// DCD __imp_freopen
// DCD 3
// DCD 1
// DCD 0
// DCD a_litevm_printf     ; "_litevm_printf"
// DCD sub_9C19C+1
// DCD 3
// DCD 1
// DCD 1
// DCD a_litevm_fprint     ; "_litevm_fprintf"
// DCD sub_9C240+1
// DCD 4
// DCD 1
// DCD 0
```

有没有跑偏我不知道，其他就留给想象吧.....

最后于 2020-4-27 09:44 被angelToms

极客

沧桑的小白鼠 2020-4-23 20:50 17 楼

太厉害了，我要努力学习，跟上大神的步伐。

极客

winkar 3 2020-4-24 09:56 18 楼

> angelToms liteVM是什么，首先大佬莫笑，我也不知道，可以猜想一下： 1、有没有调用，可以试着在sdk中删除相关java代码，跑一跑 2、sdk源码暴露给我
>
> 和InvocationHandler、P ...

删除Java代码去找native调用这个思路好，学到了

最新回复 (23)

极客

东京不热 4 2020-4-25 13:19 19 楼

```
def put_unconditional_branch(source, destination):
    offset = (destination - source - 4) >> 1
    if offset > 2097151 or offset < -2097152:
        raise RuntimeError("Invalid offset")
    if offset > 1023 or offset < -1024:
        instruction1 = 0xf000 | ((offset >> 11) & 0x7ff)
        instruction2 = 0xb800 | (offset & 0x7ff)
        PatchWord(source, instruction1)
        PatchWord(source + 2, instruction2)
    else:
        instruction = 0xe000 | (offset & 0x7ff)
        PatchWord(source, instruction)
put_unconditional_branch(here(), 0xF5D0)
```

楼主这个是python？ 请问idc能实现同样的效果吗?你的代码里面好像没有发现while 是肉眼发现一个jumpuot修复一个? 而不是整个so修复？

最后于 2020-4-25 13:20 被东京不热

大侠

koflfy 3 1 2020-4-25 15:24 20 楼

mark

极客

ZwCopyAll 4 2020-4-26 09:57 21 楼

学习

极客

_air 5 2020-4-26 11:27 22 楼

> 东京不热 def put_unconditional_branch(source, destination):   ...

https://www.anquanke.com/post/id/179080#h2-3楼主抄的这篇文章的代码也没有说一声，具体分析可以看这个文章

大牛

angelToms 4 2 2020-4-26 12:22 23 楼

> _air https://www.anquanke.com/post/id/179080#h2-3楼主抄的这篇文章的代码也没有说一声，具体分析可以看这个文章

我虽然是菜鸡，但你真高抬你自己了，我就是抄了，怎么滴吧，你写的？轮子谁不会造，拿来主义有什么不好？再说我干嘛要说，那是老毛子写的，写的，还具体分析看那篇文章， 咋能看出花呀？我随便改一下，改成idc，你就不逼逼了？有意思嘛？好像就你会搜索一样，大家连搜索都不会！

最后于 2020-4-26 12:38 被angelToms

极客

_air 5 2020-4-26 12:47 24 楼

> angelToms _air https://www.anquanke.com/post/id/179080#h2-3楼主抄的这篇文章的代码也没有说一声，具体分析可以看这个文章 ...

 火药味真重

wx_nu无情

内容

回帖 表情