

火绒软件测试初学者,火绒初步评测-软件逆向-看雪论坛-安全社区|安全招聘|bbs.pediy.com...

转载

方萌-CFT 于 2021-07-26 06:37:17 发布 519 收藏

文章标签: [火绒软件测试初学者](#)

求大神们轻喷.....

注: 本文测试主要依据3.x版本的病毒库, 但是4.x环境依然有效

0x0

调试了好几个晚上, 在茫茫汇编代码里, 在无穷无尽的各种结构体里, 太容易绕晕了, 一个结构体套一个结构体, 一个指针一个函数都容易乱指地址, 看着看着就晕乎了.....

写本文有些忐忑, 但是处于对火绒的一个评价的态度, 还是希望火绒今后会更加强大一些, 同时, 这里的测试不够全面, 仅仅测试了几个小的方面。

0x1简介

引擎

火绒自研反病毒引擎, 是火绒强大的保障

火绒反病毒引擎是一款由10余年经验资深工程师主导研发并拥有完全自主知识产权的反病毒引擎, 是国内少有的自主研发并保持每周活跃更新的新一代反病毒引擎。

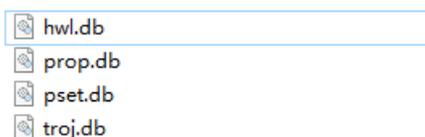
- 多项前沿技术
- 轻巧高效强悍
- 引擎动态增强

前几天发现火绒已经升级到4.x版本了。于是乎, 好奇心比较重, 然后用x64dbg调试之。这里说一下, x64dbg已经可以完全代替OD了, 谁用谁知道啊.....

0x2病毒库

就目前技术而言, 无论什么杀毒软件, 依然绕不开病毒库的。火绒的病毒库有4个文件。Ps: 但是我只发现prop和pset两个库有关联, (其他的库, 我真的没用到.....)

火绒的病毒库比较坑, 不是直接解压了就可以用的。而是各种xor解密, 还有一个函数专门负责算key的函数。



在这4个病毒库中:

Pset.db主要是病毒名称。

Prop.db包含了特征码和特征字符串等

Hwl、troj两个库, 还真没用到.....

但是, 不要以为病毒库这样就结束了, 在文件libvxf开头的dll文件中, 依然发现了病毒匹配判断的相关函数。

下图：病毒库header部分的结构体如下，如果有未知的，就当看不到看不到吧.....

56 41 52 44 59 48 46 53	00 00 01 00	C0 F7 4B 57
00 00 00 00	F1 E4 06 C0	13 3D 00 00
29 00 00 00	04 B3 12 00	31 E4 06 00
21 2E A8 31	24 31 38 9A	
13 AC 2E BF 9D 1B 44 0E	80 0A 5B DE	96 87 CD A0
B6 B3 35 41 3E B5 19 19	97 04 08 9A	11 58 4A 9C
29 A2 B9 DA BB 52 34 BE	A6 88 16 22	1F 0B EE 0D
C7 17 96 DE 06 C8 F4 81	95 00 55 95	BA DC 3F B6
5F 20 D0 A5 9D 2F 3C 89	75 56 F5 2C	9A 24 A0 89
3C D1 83 29 62 4F 05 2D	91 61 4E B4	49 70 4B 6A
12 97 81 F2 F0 58 98 A8	AA AD 13 D8	BB 47 F9 44
B8 63 1C 01 71 04 E7 B4	44 42 3A 3A	50 53 45 54
00 00 00 00	00 00 00 00	00 00 00 00

```

struct st_virdb_header
{
    DWORD flag1_DRAV; //DRAV; 56
    DWORD flag2_SFHY; //SFHY; 60
    DWORD gudingzhi_0; // 00 00 01 00
    DWORD unknown_0; // ----->未知;
    DWORD gudingzhi_1; //恒为0;
    DWORD pack_file_len; //F0 08 0C C0 整个文件长度; 比如值为: c00807a6; , 其整个文件长度为: 807a6
    DWORD virus_num; // 9A EB 00 00 建立索引的循环次数;
    DWORD sha1_len; // 29 00 00 00 // 0x29; 固定值。
    DWORD unpack_len; // 解压后的长度;
    DWORD file_neirong_len; // 除去文件头部后的长度。
    byte ch_decode_key[128]; //计算key的。
    DWORD tail_flag[2]; //结尾标志;
    DWORD tail_zero[4]; //结尾的16个0;
};
    
```

经过一长串循环计算，病毒库xor完毕，解压就是利用zlib啦，解压后，病毒库文件建立索引。(ps: 这里的索引，是我这么称呼的-具体是什么，得火绒的大神来解释了。还有一点比较有意思的地方，我猜测这个写索引的作者：是198707xx出生的(xx是我故意给隐藏的)，因为代码里用了198707xx来做计算.....)，这里我直接dump出来了整个添加了索引的病毒库。

32 66 38 65 38 35	38 38 34 37 34 37 62 37 35 34	2 f 8 e 0 3 8 8 4 7 4 7 b 7 5 4
6 36 64 33 33 33 32 30 38 62 38		c b 6 d f f 6 d 3 3 3 2 0 8 b 8
6 63 64 00 00 00 00 00 00 00		0 a b d 0 6 c d
1 34 03 10 10 00 54 45 53 54 2f		. 9 4 TEST! /
4 65 73 74 46 69 6C 65 21 48 75		AV Eng Test File! Hu
0 02 02 04 00 12 64 40 29 27 54		o r o n g d e) ' T
0 00 00 00 00 00 4D 00 00 20 03 1C		8 2 M
2 6F 6A 61 6E 2F 46 61 6B 65 49		C . 0 . Trojan / Fake I
1 00 05 00 00 02 02 04 00 6A		M E j
1 D7 02 02 04 00 E1 76 1C 6F 31	 f ! x v . o 1
4 00 88 B6 56 9E 92 8B 3B B2 00		á ÷ - V Ž ' < ; ' 2
0 00 46 00 20 02 26 43 10 20	 F & C .
1 6E 53 70 79 2F 47 61 6D 65 53		. Trojan Spy / Game S
8 00 01 00 05 00 00 02 02 04		p y . a
4 C0 24 12 02 02 04 00 32 1D 71		. % 1 g 9 = Å \$ 2 . q
0 00 00 00 00 00 00 00 8E 00 00		. x ð Ž
0 00 54 72 6F 6A 61 6E 53 70 79		. & C . Trojan Spy
4 00 25 31 67 39 A4 C0 24 12 02	 % 1 g 9 = Å \$
4 00 25 31 67 39 A4 C0 24 12 02	 % 1 g 9 = Å \$
B3 B1 DE 86 89 01 02 04 00 39		. . . x " f ³ ± P † %

下图所示建立索引前后的对比：

00 00 00 00 00 00	68 00 00 20 05 28	EÁ.....k...()
50 00 56 69 72 75	73 2F 58 6F 72 65 72 2E	@.P.Virus/Xorer.
75 65 69 60 70 6C	00 00 00 01 00 01 00 00	x@uniml.....
26 57 40 11 64 5B	39 08 60 00 00 20 05 28	EÁwM.d[90k...()
50 00 56 69 72 75	73 2F 58 6F 72 65 72 2E	@.P.Virus/Xorer.
75 6E 69 60 70 6C	00 00 00 01 00 01 00 00	x@uniml.....

Ps:有趣的一点是，在火绒3.x年代，有处代码写的相对不严谨，虽然在4.x时代已经修改了。此处代码如下：malloc后直接去使用了，而没有验证申请的内存是否成功.....

```

push    eax                ; size t
call    _malloc            ; 没有去验证是否申请的内存成功。

mov     esi, eax
add     esp, 0Ch           ; Add
mov     [esp+1E4h+v53_unpack_buf], esi
cmp     esi, edi           ; Compare Two Operands

```

0x3静态扫描

关于静态扫描，我只测试了几个较为简单的。所以不能以偏概全。

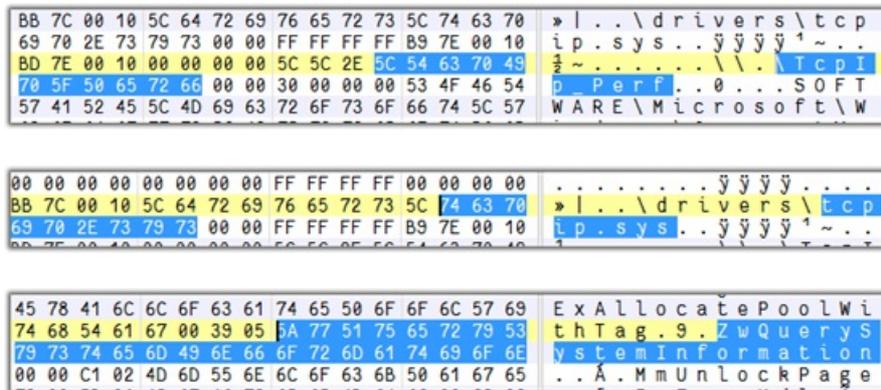
测试1: conficker病毒无壳版

Conficker病毒作为典型的僵尸网络病毒，依然活跃在互联网上。此次用它测试也正是来看看火绒的静态扫描能力。

果不其然，火绒发现了病毒.....



经过调试，我发现病毒库中命中conficker的特征有4条，如下图：



其中第一条匹配数据库prop.db内容：prop是病毒特征库。

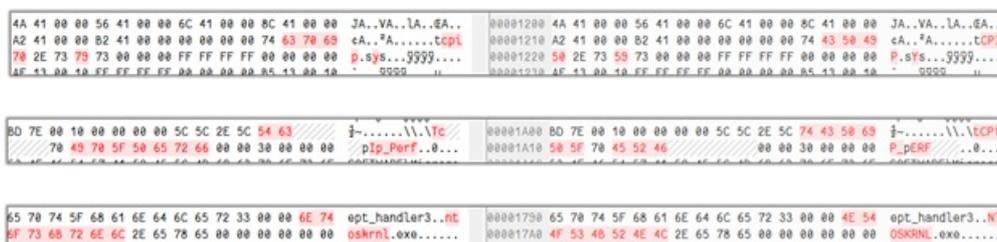
```
命中1:
B7 74 02 FB 56 C3 9B 2A //索引。
5F 00 00 20 //总长度 and 0x3fff fff
00 FC FF FF //固定。
02 04 00 00 //未知
00 16 00 20 // 匹配长度 //and 0x3fff fff ; shr 0x8;
16 00 20 00 // 未知
5C 44 6F 73 44 65 76
69 63 65 73 5C 54 63 70 49 70 5F 50 65 72 66 2C
00 20 00 5C 00 44 00 6F 00 73 00 44 00 65 00 76
00 69 00 63 00 65 00 73 00 5C 00 54 00 63 00 70
00 49 00 70 00 5F 00 50 00 65 00 72 00 66 00
```

然后我们再来看看在pset中命中了多少条索引呢？如下图。可见在pset中命中4条索引，然后修改

```
BA 59 5E 27 6C C3 A2 6A -> pset.db 索引
5C 00 00 20 // 块总大小。
04 22 // 0x2204 shr 0x9 //病毒名称的长度;
43 10 40 00 //索引数量
52 6F 6F 74 6B 69 74 2F 43 6F 6E 66 69 63 6B 65 72 00 //病毒名称:
08 00 //判断条件
01 00 //判断条件
08 00 00 00 //判断条件
02 02 04 00
B7 74 02 FB 56 C3 9B 2A -->index 1;
02 02 04 00
58 26 73 2C DA 9D 82 5A -->index 2;
02 02 04 00
D9 60 FF 17 3B 4A 29 81 -->index 3;
02 02 04 00
BD 70 96 94 B9 BC CA 6E -->index 4;
```

通过上图，我们发现，火绒在判断是否是病毒的时候一种方法是，采用了多段内容间字符串匹配，来预防误报。如果pset中的索引能匹配到3条，则可以确定是病毒，并且报毒，然后通过pset进行确认病毒名称。

如此，火绒在某些病毒匹配过程中，会依靠特定的字符串进行匹配。既然如此，我们修改一下被检测到的字符串，再来测试，如下图：(右侧是修改后的)



修改后，没有再报毒了.....



再测试一个Linux病毒。

<input checked="" type="checkbox"/> 文件路径	病毒名称	处理方式
<input checked="" type="checkbox"/> C:\www\j686_viru	HackTool/Linux.Flooder	建议删除  

这个是命中4条特征：

特征1：67 65 74 52 61 6E 64 6F 6D 49 50

特征2：63 6F 6D 6D 53 65 72 76 65 72

特征3：73 65 6E 64 4A 55 4E 4B

特征4：73 63 61 6E 50 69 64

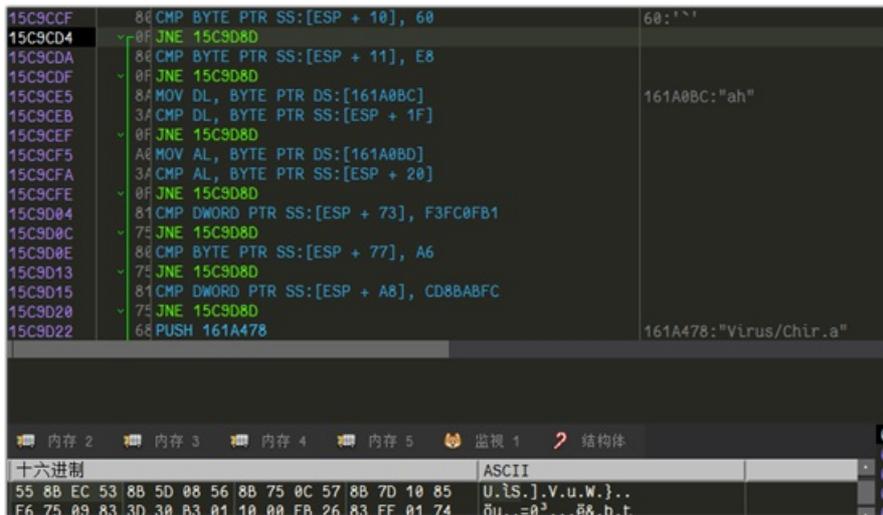
通过上述两个病毒匹配，发现火绒杀毒在匹配特征码的过程中，采用了匹配API名称和独有字符串的方法来检测的。在字符串匹配模式下，只要稍微修改一丢丢，即可绕过检测，匹配API过程中，需要在源码中修改API的名称来绕过检测。在这方面匹配的过程中，火绒还是考虑不够严谨。

测试2：特征码匹配

特征码匹配作为传统的匹配方式，在各种杀毒软件中屡见不鲜。火绒也不会避过这些特征。下面，我们通过几个例子代码，可见一斑。(不要误解这个词语的意思啊，语文一定要学好.....)

我们在此选择libvfx自带的程序进行匹配。

例子1：

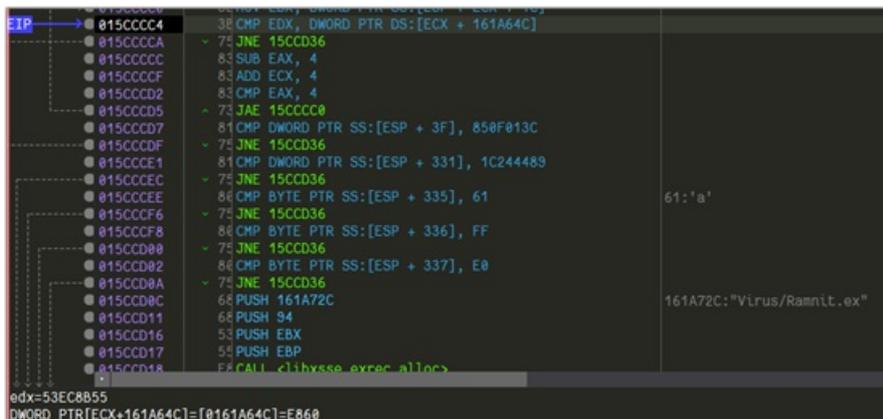


```

15C9CCF 80 CMP BYTE PTR SS:[ESP + 10], 60
15C9CD4 0F JNE 15C9D8D
15C9CDA 80 CMP BYTE PTR SS:[ESP + 11], E8
15C9CDF 0F JNE 15C9D8D
15C9CE5 8A MOV DL, BYTE PTR DS:[161A0BC]
15C9CEB 3A CMP DL, BYTE PTR SS:[ESP + 1F]
15C9CEF 0F JNE 15C9D8D
15C9CF5 A6 MOV AL, BYTE PTR DS:[161A0BD]
15C9CFA 3A CMP AL, BYTE PTR SS:[ESP + 20]
15C9CFE 0F JNE 15C9D8D
15C9D04 81 CMP DWORD PTR SS:[ESP + 73], F3F0FB1
15C9D0C 75 JNE 15C9D8D
15C9D0E 80 CMP BYTE PTR SS:[ESP + 77], A6
15C9D13 75 JNE 15C9D8D
15C9D15 81 CMP DWORD PTR SS:[ESP + A8], CD8BABFC
15C9D20 75 JNE 15C9D8D
15C9D22 6A PUSH 161A478
161A478: "Virus/Chir.a"

```

例子2：



```

EIP → 015CCC4 30 CMP EDX, DWORD PTR DS:[EAX + 161A64C]
015CCCCA 75 JNE 15CCD36
015CCCCC 83 SUB EAX, 4
015CCCCE 83 ADD ECX, 4
015CCCD2 83 CMP EAX, 4
015CCCD5 73 JAE 15CCCC0
015CCCD7 81 CMP DWORD PTR SS:[ESP + 3F], 850F013C
015CCDDF 75 JNE 15CCD36
015CCCE1 81 CMP DWORD PTR SS:[ESP + 331], 1C244489
015CCCEC 75 JNE 15CCD36
015CCCEE 80 CMP BYTE PTR SS:[ESP + 335], 61
015CCCF6 75 JNE 15CCD36
015CCCF8 80 CMP BYTE PTR SS:[ESP + 336], FF
015CCD00 75 JNE 15CCD36
015CCD02 80 CMP BYTE PTR SS:[ESP + 337], E0
015CCD0A 75 JNE 15CCD36
015CCD0C 6A PUSH 161A72C
015CCD11 6A PUSH 94
015CCD16 53 PUSH EBX
015CCD17 55 PUSH EBP
015CCD18 EB CALL c:\hyssse_exec_alloc>
161A72C: "Virus/Ramnit.exe"

```

其实从这些例子中，我们不难发现，特征码匹配的方式相对更容易绕过，这完全是依靠病毒库大量的数据进行安全防护.....

当然，这里的测试不够全面，还有一些沙盒测试，其他匹配没有测试到。不可能方方面面全面的总结火绒。

0x4总结

火绒作为新兴的杀毒软件，默默耕耘着自己的一片天地，从火绒3.x版本到现在4.x，火绒的反调试技术成熟了许多。但是由于其容量轻便的原因，导致其病毒库容量较小，杀毒的范围和能力有限。