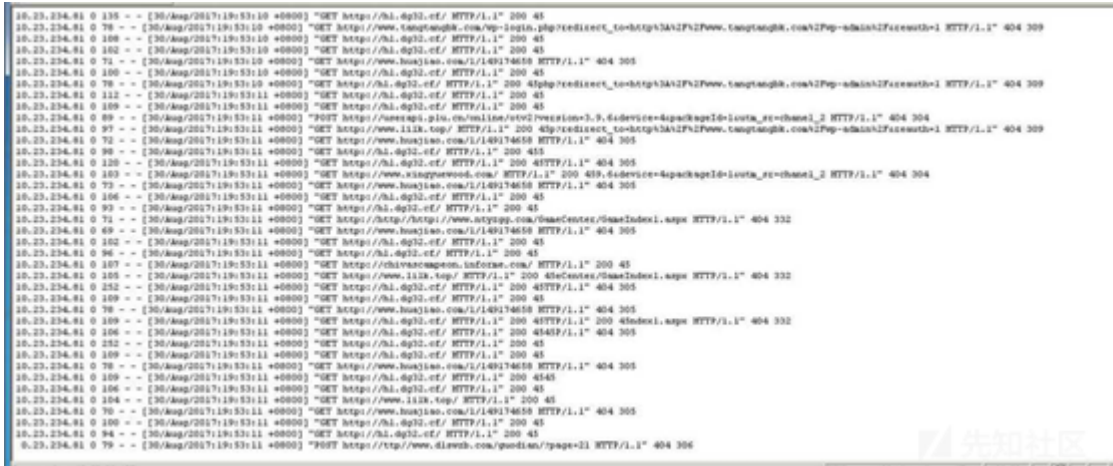


## 案例分享

之前遇到一个十分精彩的应急响应案例，当时溯源和解决问题都很顺利。但是后续在复现这个现象以及渗透测试中攻击利用这个威胁点上出现了一些问题，断断续续研究了好久。

先说一下这个应急响应，事情发生在当天下午，然而我是晚上约9点半的时候才接到电话。然后惯例建群，拉上项目组 and 各大领导们在群里说明事件严重性，要求全力支持配合。

- 1、Apache服务器的access.log日志出现很多奇怪日志；
- 2、该服务器性能下降，内存占用高。

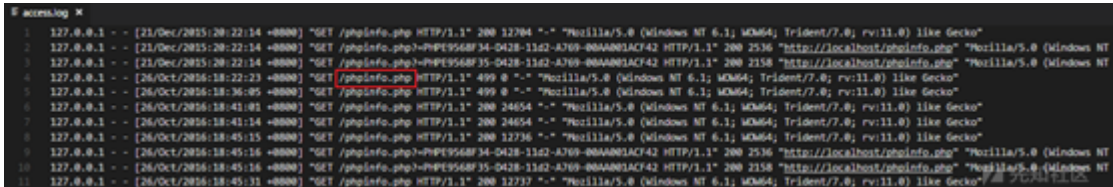


拿到截图后，可以从这个access.log中看到很多奇怪的第三方域名，没有一个域名是属于本地合法域名集的，而且响应码几乎全部是200，少数的几个404确实也是对应的网站。

由于本人才疏学浅，没见过什么大世面，属于部门一等的划水运动员。刚开始以为这就是一个简单的url跳转漏洞导致的奇怪的日志记录。对这些第三方域名进行访问测试。

从我错误的分析来看，前后是矛盾的。因为目前可以确定的是该攻击者是在进行黑产交易，但是从以往对黑产类型的攻击事件的应急响应可以得到一个比较显著且具有实际意义。

接着我比对了正常的access.log发现正常的access日志的Request请求并不会包含http协议头和域名字段，而是只记录url路径。这时候我才发现事情并不简单。



经过再一次地分析后发现这是一个利用中间件服务器的代理功能进行恶意代理攻击的事件，也就是说这台Apache服务器被人作为中间代理去访问其他站点。致使其在日志上。

基本上原因是找到了，但是还遗留了一个问题，就是那个内网源IP是怎么回事？我想了一会，判断这可能是一台做反向代理的机器，而且很大程度上是可能是一台和Apache。

这里对WAF的4种部署方式做个简单的概念叙述。

WAF一共有4种部署方式：

- 1、串联部署

最为常见的部署方式，增加WAF负载，容易出现单点故障；

- 2、旁路部署

需要进行二层或三层流量牵引，部署复杂，但是可以降低WAF负载，提高利用效率，只转发Web相关流量；

- 3、反向代理

客户端与服务器端通讯不透明，对外IP即为WAF的公网IP，WAF接收到请求后将更换请求源地址为自身IP，向内网发起请求转发；

- 4、镜像监听

只能检测服务是否被攻击，不能对客业务进行防护；

也就是说该案例的实际网络架构和攻击者的攻击路径分析如下图所示：



```
</VirtualHost>
# vim: syntax=apache ts=4 sw=4 sts=4 sr noet
```

禁用不安全的HTTP方法，基本原则是只允许开启GET和POST方法。特别地，要禁用CONNECT方法。

重启服务器，策略生效

当然这样解决的操作基本上只能作为临时修复办法，因为通过这种方式是通过设置403 Forbidden来禁止转发第三方域名的请求，只能解决恶意代理攻击，但是无法有效解决服务器负载过高的问题，因为请求最终还是会发送到服务器上，服务器还是要对其做处理。

所以终极解决办法就是：










你有WAF，你为什么不用WAF呢？或者F5也行啊。

## 攻击测试

经过全方面的测试，有如下5种测试，现在对这5种方法做简单分析。其中主要是对nmap的nse脚本和msf的rb脚本做了代码分析，其实都是简单的socket代码，主要是想模拟各种攻击。

1、nmap的脚本测试（分析以Windows环境为例）；

我们先分析一下http代理测试，该测试使用的脚本为http-open-proxy，该脚本位于nmap的scripts目录下，名称为http-open-proxy.nse

名称	修改日期	类型	大小
 http-mcmp.nse	2017/8/1 6:33	NSE 文件	4 KB
 http-methods.nse	2017/8/1 6:33	NSE 文件	8 KB
 http-method-tamper.nse	2017/8/1 6:33	NSE 文件	7 KB
 http-mobileversion-checker.nse	2017/8/1 6:33	NSE 文件	3 KB
 http-ntlm-info.nse	2017/8/1 6:33	NSE 文件	5 KB
 http-open-proxy.nse	2017/8/1 6:33	NSE 文件	9 KB
 http-open-redirect.nse	2017/8/1 6:33	NSE 文件	5 KB
 http-passwd.nse	2017/8/1 6:33	NSE 文件	7 KB
 http-phpmyadmin-dir-traversal.nse	2017/8/1 6:33	NSE 文件	先36 KB

我们打开这个nse文件看一下代码里这个nmap脚本到底是怎么使用以及怎么实现测试的。

从简单的description中我们可以看到这个脚本就是用来测试HTTP代理是否打开，但是注意去看详细描述会发现，该脚本是以测试通过http代理是否能够访问google，响应时间。

但是nmap考虑到了这一点，在@usage中，我们看到他贴心地为我们加上了除了基本的nse使用方法，也就是一script http-open-proxy以外，在2009-05-14的版本更新中Joao Correa为我们添加了自定义proxy.url的参数方案。

```
8  description=[[
9  Checks if an HTTP proxy is open.
10
11  The script attempts to connect to www.google.com through the proxy and
12  checks for a valid HTTP response code. Valid HTTP response codes are
13  200, 301, and 302. If the target is an open proxy, this script causes
14  the target to retrieve a web page from www.google.com.
15  ]]
16
17  ---
18  -- @usage
19  -- nmap --script http-open-proxy.nse \
20  --      --script-args proxy.url=<url>,proxy.pattern=<pattern>
21  -- @output
22  -- Interesting ports on scanme.nmap.org (64.13.134.52):
23  -- PORT      STATE SERVICE
24  -- 8080/tcp  open  http-proxy
25  -- | proxy-open-http: Potentially OPEN proxy.
26  -- |_ Methods successfully tested: GET HEAD CONNECT
```

所以在全局代码中我们看到了两个基本的function，一个是custom\_test()，一个是default\_test()。从参数获取我们可以看到custom\_test()要求提供host，port，test\_url，

由于基本测试形式一样，唯一不同点就是default\_test()方法中固定了url和pattern参数为默认值。所以我们这里仅仅分析一下custom\_test()的方法实现即可。

首先定义两个最后需要用来反馈给用户的return值，lstatus和response，初始化lstatus为false，最后只要存在http proxy打开，赋值为true。接着就是初始化各个入口参数，hostname，port，test\_url之类的，这里还对test\_url传入的用户输入值做了处理，如果用户输入没有加上<http://>

```

52 function custom_test(host, port, test_url, pattern)
53     local lstatus = false
54     local response = {}
55     -- if pattern is not used, result for test is code check result.
56     -- otherwise it is pattern check result.
57
58     -- strip hostname
59     if not string.match(test_url, "^http://.*") then
60         test_url = "http://" .. test_url
61         stdnse.debug1("URL missing scheme. URL concatenated to http://")
62     end
63     local url_table = url.parse(test_url)
64     local hostname = url_table.host
65
66     local get_status = proxy.test_get(host, port, "http", test_url, hostname, pattern)
67     local head_status = proxy.test_head(host, port, "http", test_url, hostname, pattern)
68     local conn_status = proxy.test_connect(host, port, "http", hostname)
69     if get_status then
70         lstatus = true
71         response[#response+1] = "GET"
72     end
73     if head_status then
74         lstatus = true
75         response[#response+1] = "HEAD"
76     end
77     if conn_status then
78         lstatus = true
79         response[#response+1] = "CONNECTION"
80     end
81     if lstatus then response = "Methods supported: " .. table.concat(response, ", ") end
82     return lstatus, response
83 end

```

由于三种方法的区别仅仅在于使用的HTTP方法不同，这里就仅以test\_get()方法为例进行分析。我们追溯一下test\_get()方法，由于其是属于proxy的属性方法，进入proxy

```

92 --- Builds the GET request and calls test
93 -- @param host The host table
94 -- @param port The port table
95 -- @param proxyType The proxy type to be tested, might be 'socks4', 'socks5' or 'http'
96 -- @param test_url The url to send the request
97 -- @param hostname The hostname of the server to send the request
98 -- @param pattern The pattern to check for valid result
99 -- @return the result of the function test (status and the request result)
100 function test_get(host, port, proxyType, test_url, hostname, pattern)
101     local status, socket = connectProxy(host, port, proxyType, hostname)
102     if not status then
103         return false, socket
104     end
105     local req = "GET " .. test_url .. " HTTP/1.0\r\nHost: " .. hostname .. "\r\n\r\n"
106     stdnse.debug1("GET Request: " .. req)
107     return test(socket, req, pattern)
108 end

```

这边首先通过connectProxy()方法进行连接，我们通过定位到connectProxy()方法，发现其实就是简单的socket连接，并且这里的socket连接是通过proxyType的指定来进

```

183 --- Creates a socket, performs proxy handshake if necessary
184 --- and returns it
185 -- @param host The host table
186 -- @param port The port table
187 -- @param proxyType A string with the proxy type. Might be "http", "socks4" or "socks5"
188 -- @param hostname The proxy destination hostname
189 -- @return status True if handshake succeeded, false otherwise
190 -- @return socket A socket with the handshake already done, or an error if
191 function connectProxy(host, port, proxyType, hostname)
192     local socket = nmap.new_socket()
193     socket:set_timeout(10000)
194     local status, err = socket:connect(host, port)
195     if not status then
196         socket:close()
197         return false, err
198     end
199     if proxyType == "http" then return true, socket end
200     if proxyType == "socks4" then return socksHandshake(socket, 4, hostname) end
201     if proxyType == "socks5" then return socksHandshake(socket, 5, hostname) end
202     socket:close()
203     return false, "Invalid proxyType"
204 end

```



可以看到与它的名字相符，这只是一个http代理测试的脚本，同样既然这边定义了socks代理连接，那肯定有socks代理的测试脚本，没错，就是socks-open-proxy。

socks-open-proxy的基本方法也是和http-open-proxy一样，同样也是custom\_test()方法和default\_test()方法，所以也是可以自定义测试url的。跟之前分析的一样，唯一

```
65 -- make requests
66 status4, get_r4, cstatus4 = proxy.test_get(host, port, "socks4", test_url, hostname, pattern)
67 status5, get_r5, cstatus5 = proxy.test_get(host, port, "socks5", test_url, hostname, pattern)
68
69 fstatus = status4 or status5
70 if(cstatus4) then response[#response+1]="socks4" end
71 if(cstatus5) then response[#response+1]="socks5" end
72 if(fstatus) then return fstatus, response end
```

所以使用nmap做测试的总结方法就是如下一条命令：

```
nmap 192.168.1.1 --script http-open-proxy socks-open-proxy --script-args proxy.url=www.baidu.com
```

## 2、msfconsole的auxiliary/scanner/http/open\_proxy模块

先看一下msf接口的简单description：HTTP Open Proxy Detection，基本与nmap的脚本功能出发点是一样的。

先调用一下这个模块看一下有哪些options是属于用户输入范畴的。

```
msf auxiliary(open_proxy) > show options
Module options (auxiliary/scanner/http/open_proxy):
```

Name	Current Setting	Required	Description
CHECKURL	http://www.google.com	yes	The web site to test via alleged web proxy
MULTIPORTS	false	no	Multiple ports will be used: 80, 443, 1080, 3128, 8000, 8080, 8123
Proxies		no	A proxy chain of format type:host:port[,type:host:port][...]
RHOSTS		yes	The target address range or CIDR identifier
RPORT	8080	yes	The target port (TCP)
SSL	false	no	Negotiate SSL/TLS for outgoing connections
THREADS	1	yes	The number of concurrent threads
VALIDCODES	200, 302	yes	Valid HTTP code for a successfully request
VALIDPATTERN	<TITLE>302 Moved</TITLE>	yes	Valid pattern match (case-sensitive into the headers and HTML body) for a successfully request
VERIFYCONNECT	false	no	Enable CONNECT HTTP method check
VHOST		no	HTTP server virtual host

可以看到msf的模块也支持自定义checkurl，并且提供自定义测试标记，即响应码和匹配模式，并且还支持是否验证CONNECT通道代理方法。所以可以发现从用户自定义角

然后对模块的方法实现进行分析，首先是一个初始化方法，其实就是默认值的赋值，和刚刚在options选项中看到的基本一样。

```
30 register_options(
31 [
32   Opt::RPORT(8080),
33   OptBool.new("MULTIPORTS", [ false, "Multiple ports will be used: 80, 443, 1080, 3128, 8000, 8080, 8123", false ]),
34   OptBool.new("VERIFYCONNECT", [ false, "Enable CONNECT HTTP method check", false ]),
35   OptString.new("CHECKURL", [ true, "The web site to test via alleged web proxy", "http://www.google.com" ]),
36   OptString.new("VALIDCODES", [ true, "Valid HTTP code for a successfully request", "200,302" ]),
37   OptString.new("VALIDPATTERN", [ true, "Valid pattern match (case-sensitive into the headers and HTML body) for a successfully request",
38 ])
39
40 register_wmap_options({
41   'OrderID' => 1,
42   'Require' => {},
43 })
44 end
```

第二个方法是run\_host()，这个方法就是自定义用户输入的内容。直接看一下执行主体的verify\_target()方法，脚本里写的代码很简单。就是调用了send\_request\_cgi()方法

其实重点在于这个send\_request\_cgi()方法，定位一下这个方法地址，先看一下全局include，可以确定为include

Msf::Exploit::Remote::HttpClient，找到这个方法的实现脚本地址为/opt/metasploit-framework/embedded/framework/lib/msf/core/exploit/http/client.rb，其实如身

```
bc@msf0ocus:/opt/metasploit-framework$ grep -r "def send_request_cgi" /opt/metasploit-framework/
/opt/metasploit-framework/embedded/framework/lib/msf/core/exploit/http/client.rb:  def send_request_cgi(opts={}, timeout = 20)
/opt/metasploit-framework/embedded/framework/lib/msf/core/exploit/http/client.rb:  def send_request_cgi!(opts={}, timeout = 20, redirect_depth = 1)
/opt/metasploit-framework/embedded/framework/modules/exploits/linux/misc/jenkins_java_deserialize.rb:  def send_request_cgi(opts={}, timeout = 20)
bc@msf0ocus:/opt/metasploit-framework$
```

理一下send\_request\_cgi()方法主体，执行方法主要是下面这一段，所以接下来要定位一下connect()方法，来弄清楚request\_cgi的具体执行参数的表达含义。

```
347 c = connect(opts)
348 r = c.request_cgi(opts)
349
350 if datastore["HttpTrace"]
351   print_line('#' * 20)
352   print_line('# Request:')
353   print_line('#' * 20)
354   print_line(r.to_s)
355 end
356
357 res = c.send_recv(r, actual_timeout)
358
359 if datastore["HttpTrace"]
360   print_line('#' * 20)
361   print_line('# Response:')
362   print_line('#' * 20)
363   print_line(res.to_terminal_output)
364 end
365 disconnect(c)
366 res
```

connect()方法首先会判断一下ssl协议，然后开始连接http server，可以看到这边的参数opt其实就是制定远程主机rhost的值，实际上就是check\_url传入的值。

```
139   nclient = Rex::Proto::Http::Client.new(  
140     opts['rhost'] || rhost,  
141     (opts['rport'] || rport).to_i,  
142     {  
143       'Msf'      => framework,  
144       'MsfExploit' => self,  
145     },  
146     dossl,  
147     ssl_version,  
148     proxies,  
149     client_username,  
150     client_password  
151   )
```

追踪到/opt/metasploit-framework/embedded/framework/lib/rex/proto/http/client.rb的request\_cgi()方法，其实也就是基本的定义参数，然后进行client客户端去连

```
147   def request_cgi(opts={})  
148     opts = self.config.merge(opts)  
149  
150     opts['ctype']      ||= 'application/x-www-form-urlencoded'  
151     opts['ssl']         = self.ssl  
152     opts['cgi']         = true  
153     opts['port']        = self.port  
154  
155     req = ClientRequest.new(opts)  
156     req  
157   end
```

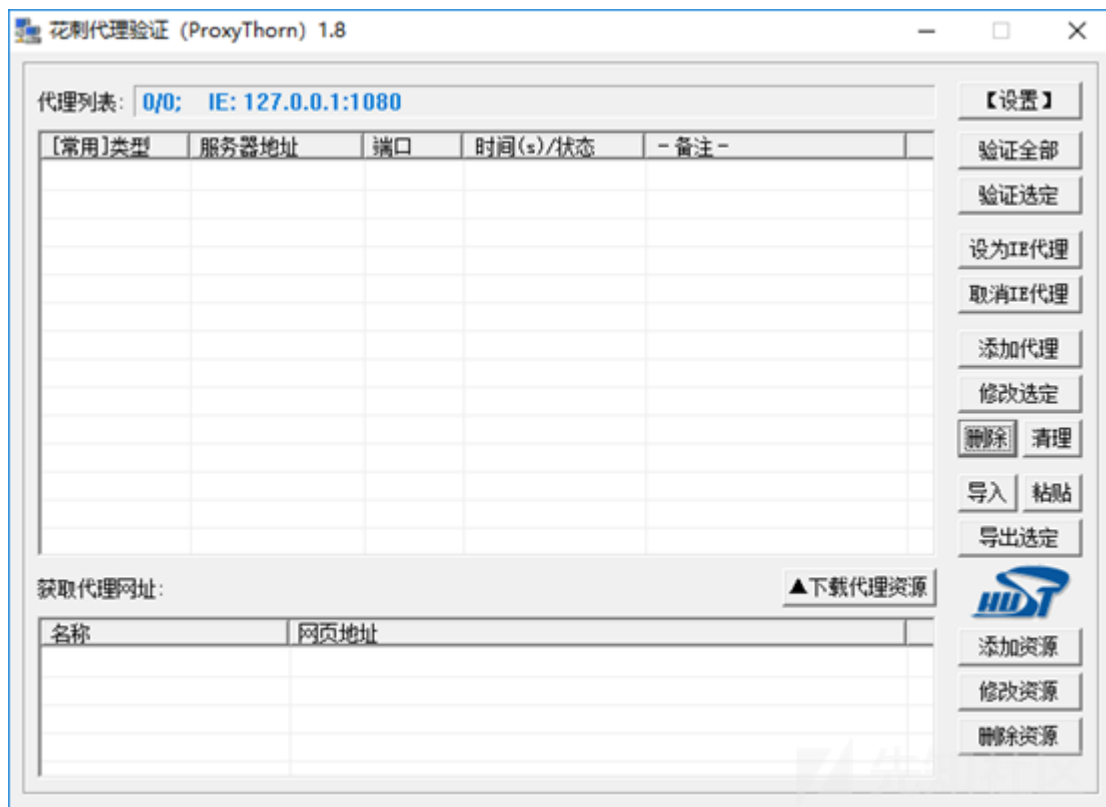
这个client.rb的整体的代码结构和之前的有点类似，只不过更加细化，在initialize()方法中可以看到读取了connect()方法中传入的8个参数值。这边其实分析到底也是一层so

```
27   def initialize(host, port = 80, context = {}, ssl = nil, ssl_version = nil, proxies = nil, username = '', password = '')  
28     self.hostname = host  
29     self.port     = port.to_i  
30     self.context  = context  
31     self.ssl      = ssl  
32     self.ssl_version = ssl_version  
33     self.proxies  = proxies  
34     self.username = username  
35     self.password = password
```

```
165   def connect(t = -1)  
166     # If we already have a connection and we aren't pipelining, close it.  
167     if (self.conn)  
168       if !pipelining?  
169         close  
170       else  
171         return self.conn  
172       end  
173     end  
174  
175     timeout = (t.nil? or t == -1) ? 0 : t  
176  
177     self.conn = Rex::Socket::Tcp.create(  
178       'PeerHost'   => self.hostname,  
179       'PeerPort'   => self.port.to_i,  
180       'LocalHost'  => self.local_host,  
181       'LocalPort'  => self.local_port,  
182       'Context'    => self.context,  
183       'SSL'        => self.ssl,  
184       'SSLVersion' => self.ssl_version,  
185       'Proxies'    => self.proxies,  
186       'Timeout'    => timeout  
187     )  
188   end
```

### 3、花刺代理工具测试；

工具是傻瓜工具，一看就知道怎么用，不再赘述，值得说的是该工具需使用管理员身份打开。



4、在线代理有效性测试；  
经过一些测试，该测试站点测试效果很好。  
<http://web.chacuo.net/netproxycheck>



5、上线前白盒分析（基线审核）[httpd.conf](#)相关安全配置文件  
点击收藏 | 0 关注 | 1  
[上一篇：2018web安全测试秋季省赛Wr...](#) [下一篇：PowerShell Remoti...](#)

1. 0 条回复
- 动动手指，沙发就是你的了！

[登录](#) 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

[RSS](#) [关于社区](#) [友情链接](#) [社区小黑板](#)