

CC攻击的变异品种——慢速攻击

武汉研发中心 彭元

DDoS 攻击凭借其严重的后果以及简单的操作，一直都是攻防中重要的攻击方式。未知攻焉知防，随着 DDoS 攻击的演变，信息安全的防御也在同步升级。发展到今天，DDoS 攻击已经多种多样。本文简要分析了慢速攻击以及防护策略。

CC 攻击的本名叫做“Challenge Collapsar”，是一种专门针对于 Web 应用层的 FLOOD 攻击，攻击者操纵网络上的肉鸡，对目标 Web 服务器进行海量 Http Request 攻击，直到服务器带宽被打满，造成了拒绝服务。

由于伪造的 HTTP 请求和客户正常请求没有区别，对于没有流量清洗设备的用户来说，这无疑就是噩梦。而我们今天谈的这种攻击方式，就是 CC 攻击的一个变异品种——慢速攻击。

什么是慢速攻击

一说起慢速攻击，就要谈谈它的成名历史了。HTTP Post 慢速 DoS 攻击第一次在技术社区被正式披露是 2012 年的 OWASP 大会上，由 Wong Onn Chee 和 Tom Brennan 共同演示了使用这一技

术攻击的威力。

这个攻击的基本原理如下：对任何一个开放了 HTTP 访问的服务器 HTTP 服务器，先建立了一个连接，指定一个比较大的 Content-Length，然后以非常低的速度发包，比如 1-10s 发一个字节，然后维持住这个连接不断开。如果客户端持续建立这样的连接，那么服务器上可用的连接将一点一点被占满，从而导致拒绝服务。

和 CC 攻击一样，只要 Web 服务器开放了 Web 服务，那么它就可以是一个靶子，HTTP 协议在接收到 Request 之前是不对请求内容作校验的，所以即使你的 Web 应用没有可用的 form 表单，这个攻击一样有效。

在客户端以单线程方式建立较大数量的无用连接，并保持持续

发包的代价非常的低廉。实际试验中一台普通 PC 可以建立的连接在 3000 个以上。这对一台普通的 Web server，将是致命的打击。更不用说结合肉鸡群做分布式 DoS 了。

鉴于此攻击简单的利用程度、拒绝服务的后果、带有逃逸特性的攻击方式，这类攻击一炮而红，成为众多攻击者的研究和利用对象。

慢速攻击的分类

发展到今天，慢速攻击也多种多样，其种类可分为以下几种：

•Slow headers：Web 应用在处理 HTTP 请求之前都要先接收完所有的 HTTP 头部，因为 HTTP 头部中包含了一些 Web 应用可能用到的重要信息。攻击者利用这点，发起一个 HTTP 请求，一直不停的发送 HTTP 头部，消耗服务器的连接和内存资源。

抓包数据可见，攻击客户端与服务器建立 TCP 连接后，每 30 秒才向服务器发送一个 HTTP 头部，而 Web 服务器再没接收到两个连续的 \r\n 时，会认为客户端没有发送完头部，而持续的等待客户端发送数据。

9	0.002048	10.67.3.220	10.67.3.215	TCP	74	55673 > xfer [SYN] Seq=0 win=29200 Len=0 MSS=
10	0.002058	10.67.3.215	10.67.3.220	TCP	78	xfer > 55673 [SYN, ACK] Seq=0 Ack=1 win=16384
11	0.003012	10.67.3.220	10.67.3.215	TCP	66	55673 > xfer [ACK] Seq=1 Ack=1 win=29312 Len=
12	0.003027	10.67.3.220	10.67.3.215	TCP	294	55673 > xfer [PSH, ACK] Seq=1 Ack=1 win=29312
2337	0.157268	10.67.3.215	10.67.3.220	TCP	66	xfer > 55673 [ACK] Seq=1 Ack=229 win=65307 Len=
2418	0.187146	10.67.3.220	10.67.3.215	TCP	74	55673 > xfer [PSH, ACK] Seq=229 Ack=1 win=29312
3049	0.360263	10.67.3.215	10.67.3.220	TCP	66	xfer > 55673 [ACK] Seq=1 Ack=237 win=65299 Len=
3219	0.045284	10.67.3.220	10.67.3.215	TCP	74	55673 > xfer [PSH, ACK] Seq=237 Ack=1 win=29312
4628	0.220488	10.67.3.215	10.67.3.220	TCP	66	xfer > 55673 [ACK] Seq=1 Ack=245 win=65291 Len=
4785	60.046920	10.67.3.220	10.67.3.215	TCP	74	55673 > xfer [PSH, ACK] Seq=245 Ack=1 win=29312
6079	60.188855	10.67.3.215	10.67.3.220	TCP	66	xfer > 55673 [ACK] Seq=1 Ack=253 win=65283 Len=
6369	90.048112	10.67.3.220	10.67.3.215	TCP	74	55673 > xfer [PSH, ACK] Seq=253 Ack=1 win=29312
7778	90.268052	10.67.3.215	10.67.3.220	TCP	66	xfer > 55673 [ACK] Seq=1 Ack=261 win=65275 Len=

```
Stream Content
GET / HTTP/1.1
Host: 10.67.3.215
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; windows NT 5.1; Trident/4.0; .NET CLR 1.1.4322; .NET CLR 2.0.50313; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729; MSoffice 12)
Content-Length: 42
X-a: b
X-a: b
X-a: b
X-a: b
```

•Slow body：攻击者发送一个 HTTP POST 请求，该请求的 Content-Length 头部值很大，使得 Web 服务器或代理认为客户端要发送很大的数据。服务器会保持连接准备接收数据，但攻击客户端每次只发送很少量的数据，使该连接一直保持存活，消耗服务器的连接和内存资源。

Time	Source	Destination	Protocol	Length	Info
24	174.765851	10.67.3.220	TCP	74	57517 > xfer [SYN] Seq=
25	174.765873	10.67.3.215	TCP	78	xfer > 57517 [SYN, ACK]
27	174.766844	10.67.3.220	TCP	66	57517 > xfer [ACK] Seq=
30	174.786000	10.67.3.215	TCP	401	57517 > xfer [PSH, ACK]
69	174.955818	10.67.3.215	TCP	66	xfer > 57517 [ACK] Seq=
292	184.665462	10.67.3.220	TCP	76	57517 > xfer [PSH, ACK]
350	184.799597	10.67.3.215	TCP	66	xfer > 57517 [ACK] Seq=
412	194.665660	10.67.3.220	TCP	107	57517 > xfer [PSH, ACK]
470	194.862163	10.67.3.215	TCP	66	xfer > 57517 [ACK] Seq=
532	204.665869	10.67.3.220	TCP	115	57517 > xfer [PSH, ACK]
590	204.815263	10.67.3.215	TCP	66	xfer > 57517 [ACK] Seq=
652	214.666034	10.67.3.220	TCP	119	57517 > xfer [PSH, ACK]
710	214.877693	10.67.3.215	TCP	66	xfer > 57517 [ACK] Seq=

```
Stream Content
POST /py HTTP/1.1
Host: 10.67.3.215:82
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; windows NT 6.1; Trident/4.0; SLCC2)
Referer: http://code.google.com/p/slowhttptest/
Content-Length: 4096
Connection: close
Content-Type: application/x-www-form-urlencoded
Accept: text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5

foo=bar&FUH5WH1=P&Gn4cGUXp4Q24=15Gkek7HN6yIHPN1G6rryHXI4BA&0UPaRgrwzmrSHZOuom=kxY1xzRVYjIL
Z65owKJE68o1qua2En&JLHAhOk153YnBuQu1LwLa7xewXkFpi=yRnwodxx0JAP2P9rau2yN5et7pFAejN=SL00u
zp6Qzhvt8j8qud1n7ikmJ
```

抓包数据可见，攻击客户端与服务器建立 TCP 连接后，发送了完整的 HTTP 头部，POST 方法带有较大的 Content-Length，然后每 10s 发送一次随机的参数。服务器因为没有接收到相应 Content-Length 的 body，而持续的等待客户端发送数据。

•Slow read：客户端与服务器建立连接并发送了一个 HTTP 请求，客户端发送完整的请求给服务器端，然后一直保持这个连接，以很低的速度读取 Response，比如很长一段时间客户端不读取任何数据，通过发送 Zero Window 到服务器，让服务器误以为客户端很忙，直到连接快超时前才读取一个字节，以消耗服务器的连接和内存资源。

抓包数据可见，客户端把数据发给服务器后，服务器发送响应时，收到了客户端的 ZeroWindow 提示（表示自己没有缓冲区用于接收数据），服务器不得不持续的向客户端发出 ZeroWindowProbe 包，询问客户端是否可以接收数据。

228	0.725321	10.67.3.220	10.67.3.215	TCP	74	57705 > xfer [SYN, Seq=0 Win=1152 Len=0 MSS=1460 SACK_PERM=1 TSval=229.0.725318
229	0.725318	10.67.3.215	10.67.3.220	TCP	78	xfer > 57705 [SYN, ACK] Seq=0 Ack=1 Win=16384 Len=0 MSS=1460 WS=1
231	0.726318	10.67.3.220	10.67.3.215	TCP	66	57705 > xfer [ACK] Seq=1 Ack=1 Win=1152 Len=0 TSval=1165968476 TSV=236.0.745526
236	0.745526	10.67.3.220	10.67.3.215	TCP	255	57705 > xfer [PSH, ACK] Seq=1 Ack=1 Win=1152 Len=189 TSval=1165968476 TSV=238.0.749634
238	0.749634	10.67.3.215	10.67.3.220	TCP	1218	[TCP window full] xfer > 57705 [ACK] Seq=1 Ack=190 Win=65346 Len=0
239	0.750881	10.67.3.220	10.67.3.215	TCP	66	[TCP zeroWindow] 57705 > xfer [ACK] Seq=190 Ack=1153 Win=0 Len=0
383	1.130870	10.67.3.215	10.67.3.220	TCP	67	[TCP zeroWindowProbe] xfer > 57705 [ACK] Seq=1153 Ack=190 Win=65346
391	1.131685	10.67.3.220	10.67.3.215	TCP	66	[TCP zeroWindowProbeAck] [TCP zeroWindow] 57705 > xfer [ACK] Seq=499.7.943211
499	7.943211	10.67.3.215	10.67.3.220	TCP	67	[TCP zeroWindowProbe] xfer > 57705 [ACK] Seq=1153 Ack=190 Win=65346
503	7.945054	10.67.3.220	10.67.3.215	TCP	66	[TCP zeroWindowProbeAck] [TCP zeroWindow] 57705 > xfer [ACK] Seq=622.17.568187
622	17.568187	10.67.3.215	10.67.3.220	TCP	67	[TCP zeroWindowProbe] xfer > 57705 [ACK] Seq=1153 Ack=190 Win=65346
629	17.569260	10.67.3.220	10.67.3.215	TCP	66	[TCP zeroWindowProbeAck] [TCP zeroWindow] 57705 > xfer [ACK] Seq=766.36.709882
766	36.709882	10.67.3.215	10.67.3.220	TCP	67	[TCP zeroWindowProbe] xfer > 57705 [ACK] Seq=1153 Ack=190 Win=65346
777	36.709882	10.67.3.220	10.67.3.215	TCP	66	[TCP zeroWindowProbeAck] [TCP zeroWindow] 57705 > xfer [ACK] Seq=969.75.208822
969	75.208822	10.67.3.215	10.67.3.220	TCP	67	[TCP zeroWindowProbe] xfer > 57705 [ACK] Seq=1153 Ack=190 Win=65346
984	75.209769	10.67.3.220	10.67.3.215	TCP	66	[TCP zeroWindowProbeAck] [TCP zeroWindow] 57705 > xfer [ACK] Seq=

```
Stream Content
GET /py/ HTTP/1.1
Host: 10.67.3.215:82
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.7; rv:5.0) Firefox/5.0.1
Referer: http://code.google.com/p/slowhttpptest/

HTTP/1.1 200 OK
Date: Wed, 22 Apr 2015 16:30:50 GMT
Server: Apache/2.4.4 (win32) OpenSSL/0.9.8y PHP/5.4.16
X-Powered-By: PHP/5.4.16
Set-Cookie: PHPSESSID123=it9c2lk0ifouds8ic2kuau50n5; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Set-Cookie: PHPSESSID123=it9c2lk0ifouds8ic2kuau50n5; expires=Wed, 19 Nov 1981 08:52:00 GMT; path=/; httponly
Set-Cookie: NumVisitsPhp=1
```

使用较多的慢速攻击工具有：Slowhttpptest 和 Slowloris。

哪些服务器易被慢速攻击

慢速攻击主要利用的是 thread-based 架构的服务器的特性，这种服务器会为每个新连接打开一个线程，它会等待接收完整整个 HTTP 头部才会释放连接。比如 Apache 会有一个超时时间来等待这种不完全连接（默认是 300s），但是一旦接收到客户端发来的数据，这个超时时间会被重置。正是因为这样，攻击者可以很容易保持住一个连接，因为攻击者只需要在即将超时之前发送一个字符，便可以延长超时时间。而客户端只需要很少的资源，便可以打开多个连接，进而占用服务器很多的资源。

经验证，Apache、httpd 采用 thread-based 架构，很容易遭受慢速攻击。而另外一种 event-based 架构的服务器，比如 nginx 和 lighttpd 则不容易遭受慢速攻击。

如何防护慢速攻击

Apache 服务器现在使用较多的有三种简单防护方式。

•mod_reqtimeout : Apache2.2.15 后, 该模块已经被默认包含, 用户可配置从一个客户端接收 HTTP 头部和 HTTPbody 的超时时间和最小速率。如果一个客户端不能在配置时间内发送完头部或 body 数据, 服务器会返回一个 408REQUEST TIME OUT 错误。配置文件如下:

```
<IfModule mod_reqtimeout.c>
RequestReadTimeout header=20-
40,MinRate=500 body=20,MinRate=500
</IfModule>
```

•mod_qos : Apache 的一个服务质量控制模块, 用户可配置各种不同粒度的 HTTP 请求阈值, 配置文件如下:

```
<IfModule mod_qos.c>
# handle connections from up to
100000 different IPs
QS_ClientEntries 100000
# allow only 50 connections per IP
QS_SrvMaxConnPerIP 50
# limit maximum number of active
TCP connections limited to 256
MaxClients 256
```

```
# disables keep-alive when 180 (70%)
TCP connections are occupied
QS_SrvMaxConnClose 180
# minimum request/response speed
(deny slow clients blocking the server,
keeping connections open without
requesting anything
QS_SrvMinDataRate 150 1200
</IfModule>
```

•mod_security : 一个开源的 WAF 模块, 有专门针对慢速攻击防护的规则, 配置如下:

```
SecRule RESPONSE_STATUS "@streq 408" "phase:5,t:none,nolog,pass,
setvar:ip.slow_dos_counter=+1,
expirevar:ip.slow_dos_counter=60,
id:'1234123456'"
SecRule IP:SLOW_DOS_COUNTER
"@gt 5" "phase:1,t:none,log,drop,
msg:'Client Connection Dropped
due to high number of slow DoS alerts',
id:'1234123457'"
```

传统的流量清洗设备针对 CC 攻击, 主

要通过阈值的方式来进行防护, 某一个客户在一定的周期内, 请求访问量过大, 超过了阈值, 清洗设备通过返回验证码或者 JS 代码的方式防护。这种防护方式的依据是, 攻击者们使用肉鸡上的 DDoS 工具模拟大量 Http Request, 这种工具一般不会解析服务端返回数据, 更不会解析 JS 之类的代码。因此当清洗设备截获到 HTTP 请求时, 返回一段特殊 JavaScript 代码, 正常用户的浏览器会处理并正常跳转不影响使用, 而攻击程序会攻击到空处。

而对于慢速攻击来说, 通过返回验证码或者 JS 代码的方式能达到部分效果。但是根据慢速攻击的特征, 可以辅助以下几种防护方式: 1、周期内统计报文数量, 一个 TCP 连接, HTTP 请求的报文中, 报文过多或者报文过少都是有问题的, 如果一个周期内报文数量非常少, 那么它就可能是慢速攻击; 如果一个周期内报文数量非常多, 那么它就可能是一个 CC 攻击。2、限制 HTTP 请求头的最大许可时间, 超过最大许可时间, 如果数据还没有传输完成, 那么它就有可能是一个慢速攻击。