译者：興趣使然的小胃

来源：https://www.anquanke.com/post/id/189507

0x00 前言

CPDoS的全称为**C**ache-**P**oisoned **D**enial-**o**f-**S**ervice（缓存污染拒绝服务），是一种新的Web缓存污染攻击，目标是导致目标站点及web资源无法正常提供服务。

0x01 工作原理

基本的攻击流程如下所示：

1、攻击者发送简单的HTTP请求，请求中包含针对目标web资源的恶意头部。该请求由中间缓存端处理，恶意头部字段尽量保持隐蔽；

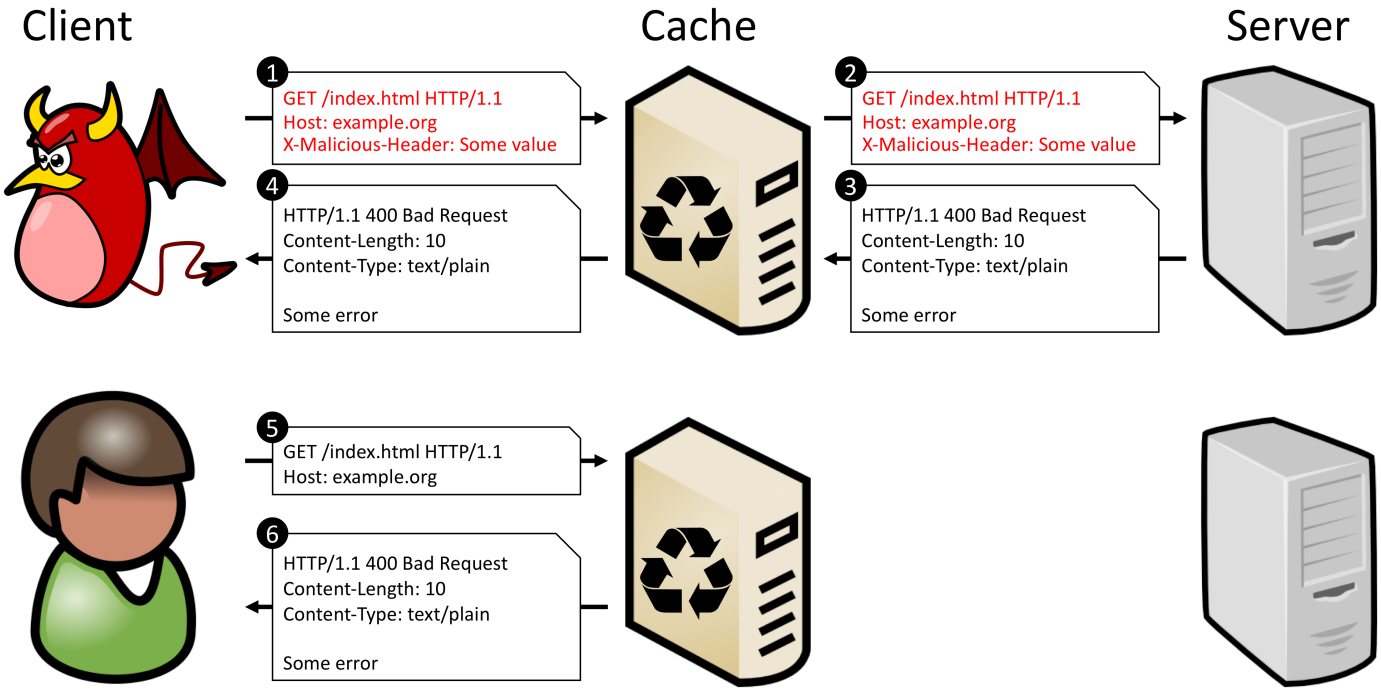
2、由于缓存端并没有存储目标资源的最新副本，因此会将请求转发至原始服务器。在原始服务器端，由于请求中包含恶意头部，因此会出现错误；

3、因此，原始服务器会返回一个错误页面，该页面（而不是所请求的资源）会被缓存端存储；

4、当攻击者在响应中看到错误页面，就知道攻击已成功完成；

5、合法用户在后续请求中尝试获取目标资源；

6、合法用户无法拿到原始内容，会得到已被缓存的错误页面。



利用CPDoS攻击方式，恶意客户端可以阻止用户访问通过CDN（如Cloudfront或者Clloudflare）发布或者在代理缓存端（如Varnish或Squid）托管的任何web资源（因此攻击者可以禁用各种资

源，如安全更新、JavaScript文件甚至整个站点）。需要注意的是，攻击者只需要精心构造一次请求，就能限制正常用户访问目标内容的后续请求。

0x02 CPDoS类别

我们检测到3类不同的CPDoS：

- HHO（HTTP Header Oversize，HTTP头部过长）
- HMC（HTTP Meta Character，HTTP元字符）
- HMO（HTTP Method Override，HTTP方法覆盖）

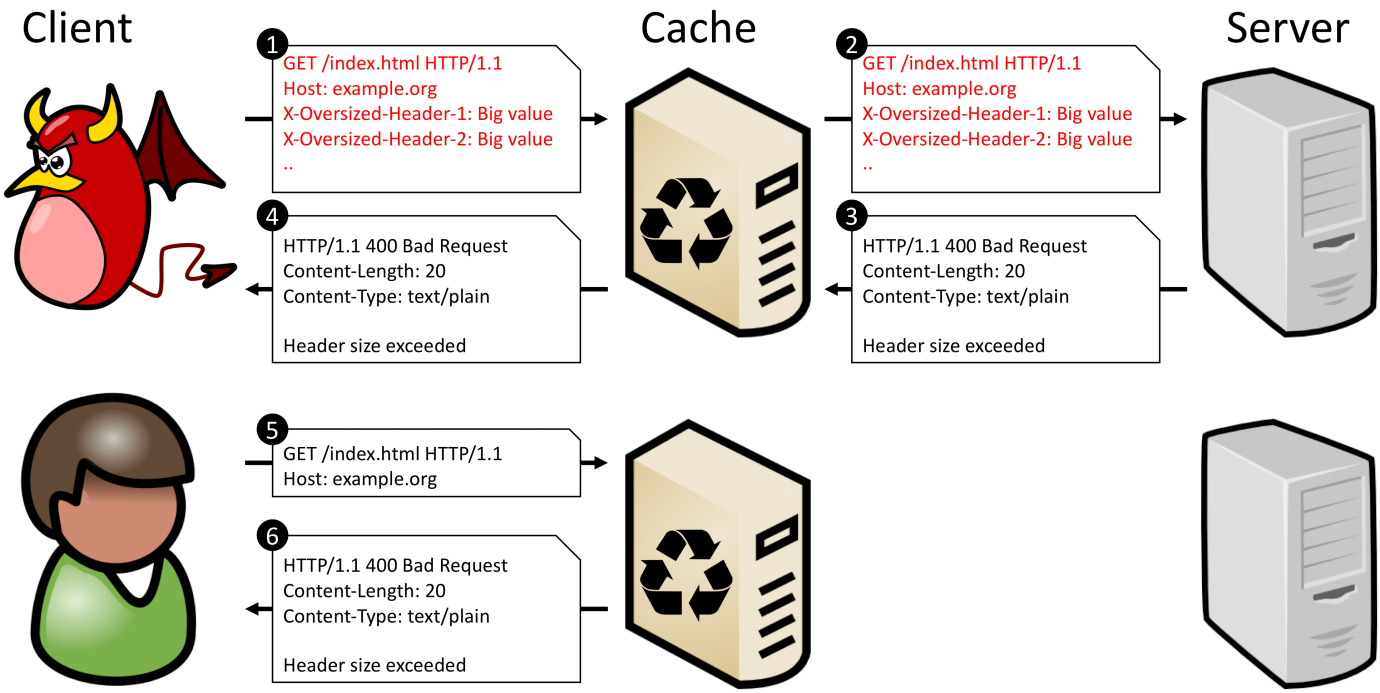
HHO

HTTP请求头部中包含对中间系统及web服务器非常重要的一些信息，其中包括与缓存相关的头部字段或客户端支持的媒体类型、语言及编码对应的元数据。HTTP标准中并没有定义HTTP请求头部的大小，因此，中间系统、web服务器以及web框架在大小限制上都有自己的规定。大多数web服务器以及代理（Apache HTTPD）所限制的请求头的大小约为8192字节，以缓解请求头缓冲区溢出或ReDoS攻击。然而，其他有些中间系统的限制条件会超过这个阈值。比如，Amazon Cloudfront CDN允许的大小为20480字节。攻击者可以利用这一点，通过请求头大小限制来发起缓存污染攻击，最终达到拒绝服务效果。

当web应用使用的缓存端允许的头部大小比原始服务端允许的还大时，这种场景下攻击者就可以发起HHO CPDoS攻击。为了攻击这种web应用，恶意客户端可以发送一个HTTP GET请求，其中头部大小比原始服务端支持的最大值大，但小于缓存端支持的最大值。为了完成该操作，攻击者可以有两种选择。攻击者可以构造一个请求，请求头中带有许多恶意头部字段（参考如下Ruby代码片段）；攻击者还可以选择使用单个头部，其中包含一个过长的键值。

```
require 'net/http'
uri = URI("https://example.org/index.html")
req = Net::HTTP::Get.new(uri)

num = 200
i = 0

# Setting malicious and irrelevant headers fields for creating an oversized
header
until i > num  do
    req["X-Oversized-Header-#{i}"] = "Big-Value-
000000000000000000000000000000000000"
    i +=1;
end

res = Net::HTTP.start(uri.hostname, uri.port, :use_ssl => uri.scheme == 'https')
{|http|
    http.request(req)
}
```

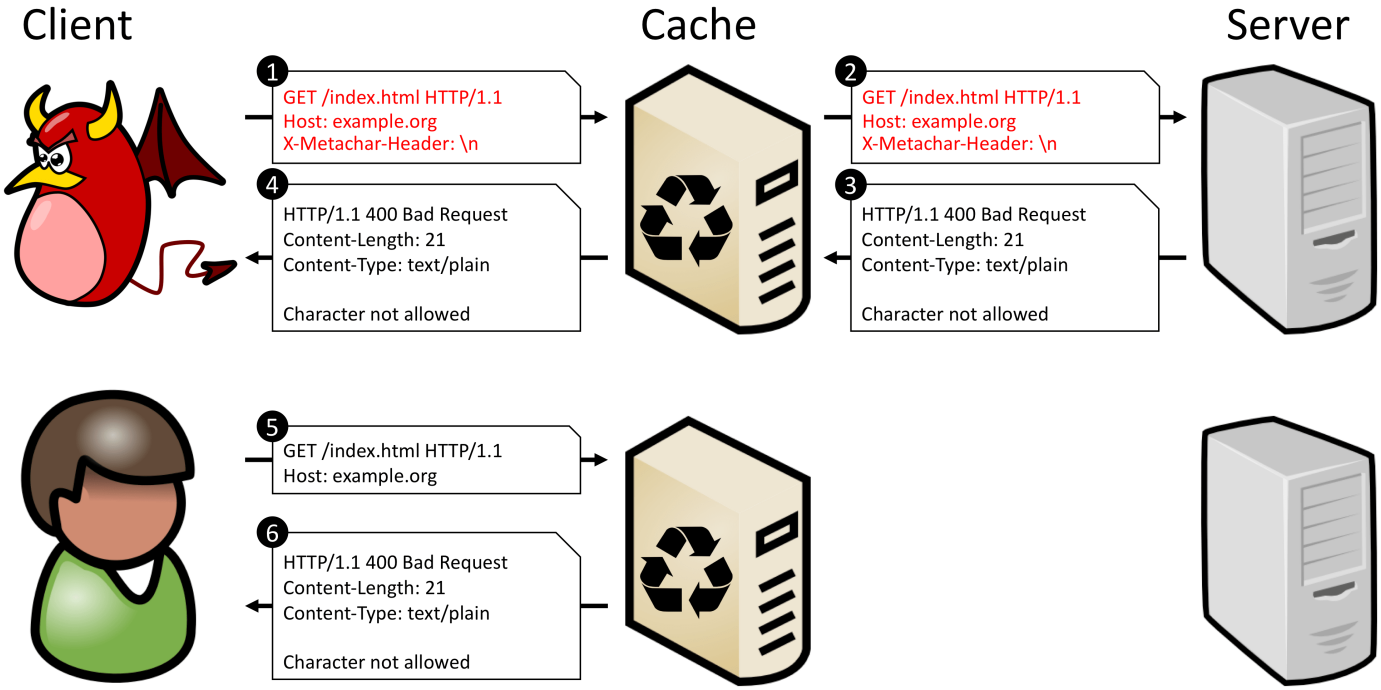
下图就是典型HHO CPDoS攻击场景，其中恶意客户端发送由上述代码片段构造的一个请求。缓存端会将该请求（包括所有头部字段）发往目标端点（因为头部大小低于20480字节）。然而，web服务端会阻止该请求，返回一个错误页面（因为请求头超过了头部大小限制）。带有`404 Bad Request`状态码的错误页面现在会被缓存端缓存。针对目标资源的后续所有请求都会返回错误页面，而不会返回正常内容。



大家可以观看该视频了解针对托管在Cloudfront上示例web应用的HHO CPDoS攻击。在此次攻击中，攻击者有选择性地将web资源替换成错误页面，最终导致整个页面无法提供服务。

HMC

HMC CPDoS攻击的原理与HHO CPDoS攻击类似。这里攻击者并没有发送过长的头部，而是尝试使用包含有害元字符的请求头来绕过缓存端。元字符可以为控制字符，比如换行符/回车符（`\n`）、换行符（`\r`）或者响铃符（`\a`）。



缓存端可能不会阻止该请求或者过滤其中的元字符，直接将这种请求发送到原始服务端。由于请求中包含有害的元字符，因此原始服务端可能会将该请求划分到恶意类别，返回错误消息，该消息会被缓存端缓存并复用。

HMO

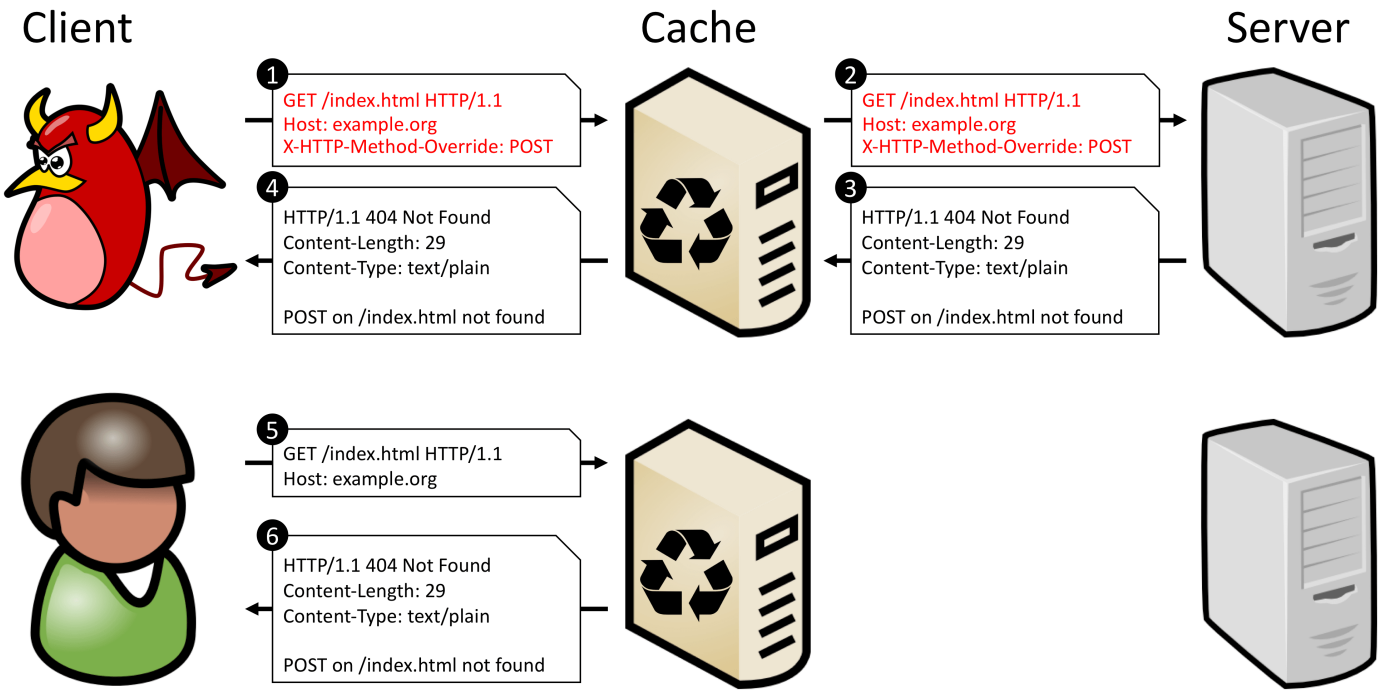
HTTP标准提供了多个HTTP方法，以便web服务端以及客户端执行各种web事务。GET、POST、DELETE及PUT是web应用及基于REST的web服务最常使用的几种HTTP方法。然而，许多中间系统（如代理、负载均衡器、缓存以及防火墙）只支持GET及POST方法。这意味着使用DELETE及PUT的HTTP请求会被阻止。为了绕过该限制，许多基于REST的API或web框架（如Play Framework 1）提供了一些头部字段，如`X-HTTP-Method-Override`、`X-HTTP-Method`或者`X-Method-Override`，以处理被阻止的HTTP方法。一旦请求到达服务端，这些头部字段会引导web应用使用对应的值来覆盖请求行中的HTTP方法。

```
POST /items/1 HTTP/1.1
Host: example.org
X-HTTP-Method-Override: DELETE

HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 62

Resource has been successfully removed with the DELETE method.
```

如上所示，该请求可以绕过安全策略，通过`X-HTTP-Method-Override`头来使用`DELETE`请求。在服务端，这个POST请求会被解析为DELETE请求。

当中间系统会阻止不同的HTTP方法时，这些头部字段就非常有用。然而，如果web应用支持这种头部，并且使用了web缓存系统（如反向代理缓存或CDN）来优化性能，那么恶意客户端可以利用

这种架构来发起CPDoS攻击。下图演示了典型的HMO CPDoS攻击场景，其中使用了`X-HTTP-Method-Override`头。



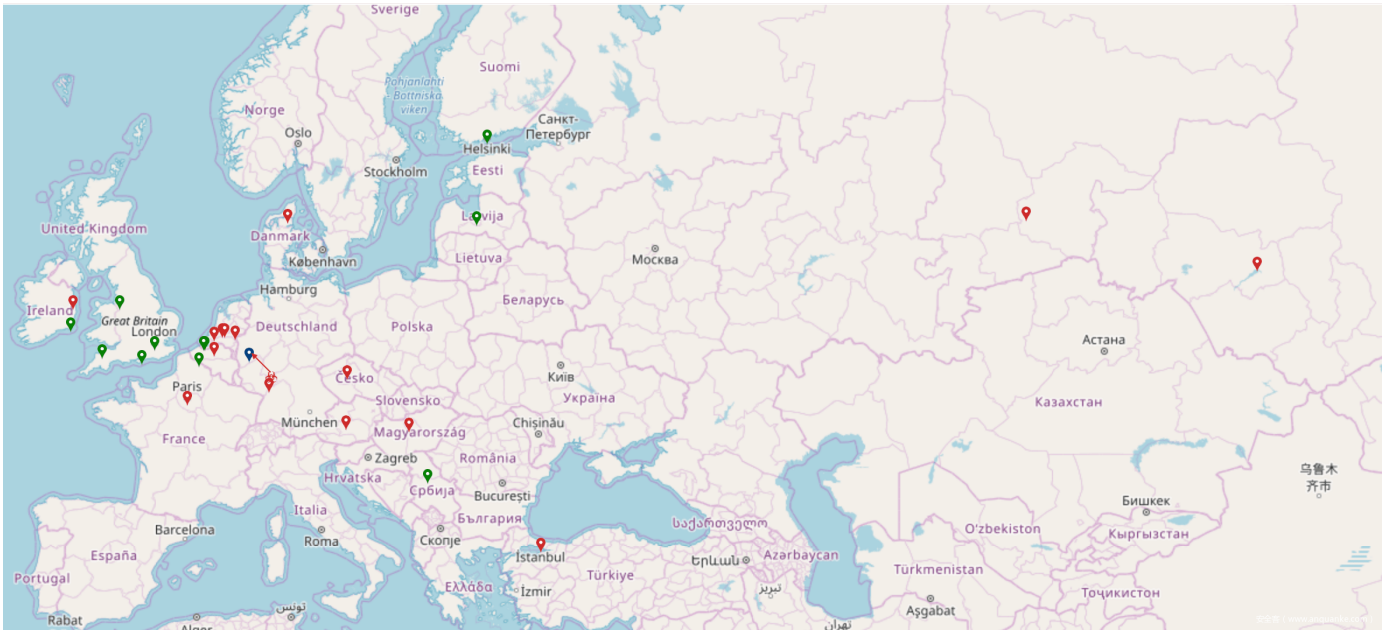
在上图中，攻击者在发送的`GET`请求中，`X-HTTP-Method-Override`头部字段的值为`POST`。存在漏洞的缓存端会将该请求解析为发往`https://example.org/index.html`的正常`GET`请求。然而，web应用会将该请求解析为`POST`请求。因此，web应用会基于`POST`请求来返回响应。这里我们假设目标web应用并没有在`/index.html`上实现适用于`POST`请求的任何逻辑。在这种情况下，web框架（如Play Framework 1）会返回错误消息，状态码为`404 Not Found`。缓存端会认为返回的响应以及错误码对应的是发往`https://example.org/index.html`的`GET`请求。根据RFC 7231，`404 Not Found`是可以被缓存的一种状态码，因此缓存端会存储并为后续的重复请求复用这个错误代码。后面每个正常的客户端如果向`https://example.org/index.html`发送`GET`请求，都会收到被缓存的错误消息以及`404 Not Found`状态码，不会看到真正的web应用起始页面。

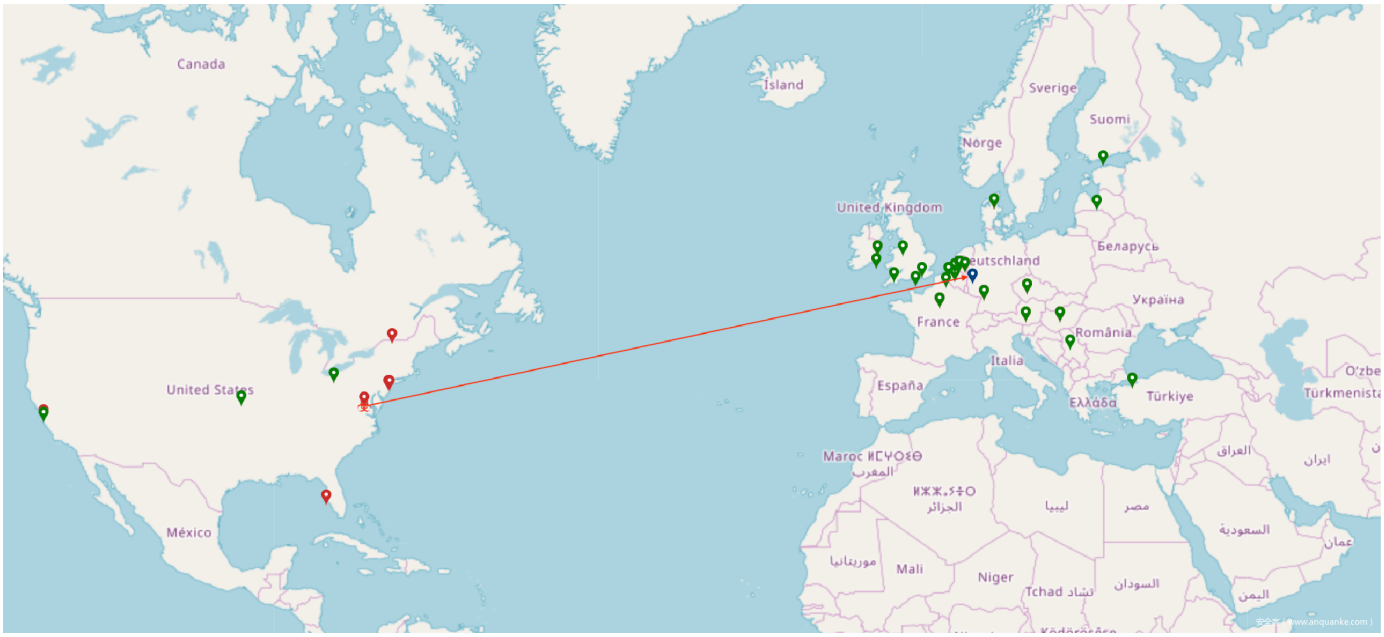
针对web应用的HMO攻击可参考此处视频，其中攻击者使用Postman工具阻止用户访问起始页面。

0x03 影响范围

下图显示了CPDoS攻击对CDN的影响。注入错误页面后，CDN会将其发布到世界各地的边缘缓存服务器。下图演示了这个错误页面在CDN架构中的辐射范围，其中，红色坐标代表会显示错误页面的位置。幸运的是，并不是所有边缘服务器都会受这种攻击影响，这类边缘服务器用绿色坐标标识，该坐标代表客户端可以接收到正常页面。蓝色坐标为原始服务端的位置，剩余一个坐标为攻击者的位置。

根据第一张图，当攻击者从法兰克向德国科隆的某个源服务器发起CPDoS攻击时，欧洲及亚洲部分位置会受到影响。根据第二张图，当攻击者从美国北弗吉尼亚对德国科隆的同一个源服务器发起CPDoS攻击时，美国部分区域会受到影响。





我们通过TurboBytes Pulse及KeyCDN的测速工具来分析攻击结果。这两个服务都能提供测试环境，覆盖遍布全球的许多测试代理。

0x04 CPDoS漏洞概览

下图整体描述了哪种web缓存系统与HTTP实现搭配时会受CPDoS攻击影响，如果大家想了解更多细节，可访问此处下载详细报告。

Cache / HTTP Implementation	Apache HTTPD	Apache TS	Nginx	Squid	Varnish	Akamai	Azure	CDN77	CDNSun	Cloudflare	CloudFront	Fastly	G-Core Labs	KeyCDN	StackPath
Apache HTTPD + (ModSecurity)	o	o	o	o	o	o	o	o	o	o	HHO, HMC	o	o	o	o
Apache TS	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Nginx + (ModSecurity)	o	o	o	o	o	o	o	o	o	o	HHO	o	o	o	o
IIS	o	o	o	o	(HHO)	(HHO)	o	(HHO)	o	(HHO)	HHO, HMC	(HHO)	o	o	o
Tomcat	o	o	o	o	o	o	o	o	o	o	HHO	o	o	o	o
Squid	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Varnish	o	o	o	o	o	o	o	o	o	o	HHO, HMC	o	o	o	o
Amazon S3	o	o	o	o	o	o	o	o	o	o	HHO	o	o	o	o
Google Cloud Storage	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Github Pages	o	o	o	o	o	o	o	o	o	o	HHO, HMC	o	o	o	o
Gitlab Pages	o	o	o	o	o	o	o	o	o	o	HMC	o	o	o	o
Heroku	o	o	o	o	o	o	o	o	o	o	HHO	o	o	o	o
ASP.NET	o	o	o	o	(HHO)	(HHO)	o	(HHO)	o	(HHO)	(HHO), (HMC)	(HHO)	o	o	o
BeeGo	o	o	o	o	o	o	o	o	o	o	HMC	o	o	o	o
Django	o	o	o	o	o	o	o	o	o	o	(HHO), (HMC)	o	o	o	o
Express.js	o	o	o	o	o	o	o	o	o	o	HMC	o	o	o	o
Flask	o	o	o	o	o	(HMO)	o	o	o	o	HMO, (HHO), (HMC)	o	o	o	o
Gin	o	o	o	o	o	o	o	o	o	o	HMC	o	o	o	o
Laravel	o	o	o	o	o	o	o	o	o	o	(HHO), (HMC)	o	o	o	o
Meteor.js	o	o	o	o	o	o	o	o	o	o	HMC	o	o	o	o
Play 1	o	o	o	o	HMO	HMO	o	HMO	o	HMO	HHO, HMO	HMO	o	o	o
Play 2	o	o	o	o	o	o	o	o	o	o	HHO, HMC	o	o	o	o
Rails	o	o	o	o	o	o	o	o	o	o	(HHO), (HMC)	o	o	o	o
Spring Boot	o	o	o	o	o	o	o	o	o	o	HHO	o	o	o	o
Symfony	o	o	o	o	o	o	o	o	o	o	(HHO), (HMC)	o	o	o	o

0x05 缓解措施

HHO及HMC CPDoS之所以能攻击成功，原因在于存在漏洞的缓存端在默认情况下会存储包含错误代码的响应（入如400 Bad Request）。根据HTTP标准，这是不被允许的一种行为。Web缓存标准只允许缓存404 Not Found、405 Method Not Allowed、410 Gone以及501 Not Implemented`错误代码。因此，我们需要根据HTTP标准策略来缓存错误页面，这是缓解CPDoS攻击的第一个步骤。

内容提供方还必须为对应的错误情况来使用响应的状态码。比如，许多HTTP实现中会为过长的头部返回`400 Bad Request`代码，这并不是合适的状态码。当特定头部过长时，IIS甚至会使用`404 Not Found`。真正适用于过长请求头的错误代码为`431 Request Header Fields Too Large`。根据我们的分析，这个错误信息并不会被任何web缓存系统所缓存。

针对HHO及HMC CPDoS攻击的另一种有效策略就是从缓存中排除错误页面。一种方法就是在每个错误页面中添加`Cache-Control: no-store`头。另一个选项就是在缓存配置中禁用错误页面缓存功能。类似CloudFront或者Akamai之类的CDN会提供类似配置，便于用户进行操作。

此外，我们也可以部署WAF（web应用防火墙）来缓解CPDoS攻击。然而，WAF必须部署在缓存端前方，以便在恶意内容到达原始服务器前采取阻止措施。如果WAF部署在原始服务器前方，也会被攻击者利用，成功缓存错误页面。

如果大家想了解关于缓解措施方面的更多细节，请参考我们的研究论文。