DV2546 Software Security
Lab 03- Web Security

Kaavya Rekanar
9405217184
kare15@bth.se

Siva Venkata Prasad Patta
9312217137
sipa15@bth.se

Introduction

Web Security is a branch of Information Security that specifically deals with security of websites, web applications and web services.
The Open Web Application Security Project (OWASP) is a non-profitable charitable organization focused on improving the security of software worldwide [4]. The web applications prone to vulnerabilities are called Broken Web Applications. OWASP Broken Web Application Project (OWASPBWAP) is an anthology of vulnerable web applications that is distributed on a Virtual Machine designed to practice penetration testing in a safe manner [5].
This project comprises of three types of applications: training applications, realistic intentionally vulnerable applications and old version of real vulnerable applications.
All these applications have been explored and a clear report has been given in this documentation. The identified vulnerabilities have been exploited and the processes have been described in detail. The vulnerability repository played an important role in giving an introduction about SQL Injection and the threats it poses in the field of web security. The knowledge gained, number of hours spent, and the level of difficulty felt in performing the tasks have been reported in the Reflexion section.

Task 1: Vulnerability Repository

The repository we chose is National Vulnerability Database.
Vulnerability:
In National Vulnerability Database, the vulnerability we have picked is CVE-2015-7835- Uncontrolled creation of large page mappings by PV guests
This vulnerability is of the type CWE-20: Input Validation.
The Xen Security Team has announced a critical security bug (XSA 148) in the hypervisor code handling memory virtualization for the PV VMs.
It says that the code to validate level 2 page entries is bypassed when certain conditions are satisfied, which would mean that a PV guest can create writeable mappings using super page mappings. Such mappings hold the risk of violating Xen intended invariants for pages where Xen is supposed to keep read only.
This statement clearly explains the fact that the bug is a very critical one. Probably, the worst that the Xen hypervisor has been affected with.

1.1. Description of the bug

The above stated bug is in the mod_12_entry () function of the source code, which precisely handles the PV guests to update their page table mappings.

```
35      /* Update the L2 entry at pl2e to new value nl2e. pl2e is within frame pfn. */
36      static int mod_l2_entry(l2_pgentry_t *pl2e,
37                              l2_pgentry_t nl2e,
38                              unsigned long pfn,
39                              int preserve_ad,
40                              struct vcpu *vcpu)
41      {
42          l2_pgentry_t ol2e;
43          struct domain *d = vcpu->domain;
44          struct page_info *l2pg = mfn_to_page(pfn);
45          unsigned long type = l2pg->u.inuse.type_info;
46          int rc = 0;
47
48          if ( unlikely(!is_guest_l2_slot(d, type, pgentry_ptr_to_slot(pl2e))) )
49          {
50              //...
51          }
52
53          if ( unlikely(__copy_from_user(&ol2e, pl2e, sizeof(ol2e)) != 0) )
54              return -EFAULT;
55
56          if ( l2e_get_flags(nl2e) & _PAGE_PRESENT )
57          {
58              if ( unlikely(l2e_get_flags(nl2e) & L2_DISALLOW_MASK) )
59              {
60                  //...
61              }
62
63              /* Fast path for identical mapping and presence. */
64              if ( !l2e_has_changed(ol2e, nl2e, _PAGE_PRESENT) )
65              {
66                  adjust_guest_l2e(nl2e, d);
67                  if ( UPDATE_ENTRY(l2, pl2e, ol2e, nl2e, pfn, vcpu, preserve_ad) )
68                      return 0;
69                  return -EBUSY;
70              }
```

Figure 1: The mod_12_entry() function

Though, the code looks perfectly fine here, it becomes buggy if we also consider the definition of the L2_DISALLOW_MASK macro:

```
#define L2_DISALLOW_MASK (base_disallow_mask & ~_PAGE_PSE)
```

Figure 2: definition of L2_DISALLOW_MASK

and also, the initialization of base_disallow_mask:

```
79      void __init arch_init_memory(void)
80      {
81          (...)
82          /* Basic guest-accessible flags: PRESENT, R/W, USER, A/D, AVAIL[0,1,2] */
83          base_disallow_mask = ~(_PAGE_PRESENT|_PAGE_RW|_PAGE_USER|
84                                 _PAGE_ACCESSED|_PAGE_DIRTY|_PAGE_AVAIL);
85
```

Figure 3: initialization of L2_DISALLOW_MASK

The exploitation can be done when the attacker might request the setting of the (PSE | RW) bits in L2 PDE, which is possible because the L2_DISALLOW_MASK does not exclude the PSE bit; something which has been added to support the superpage mappings for PV guests, thus making the whole L2 table available to the attacker(guest) with R/W rights (memory-

2MB), and modify one or more of the PTEs present to point to an arbitrary MFN the attacker would like to access.
This exploitation would not really be fatal, if the attacker had no way of tricking Xen into treating this superpage as a valid table of PTEs. But, unfortunately, there is no way to stop this, which ended up with Xen treating the attacker filled memory as a set of valid PTEs for the PV guest. The guest can now access whatever MFNs the attacker decided to wrote into the PTEs by referencing the addresses mapped by these pages, which in other words means that the guest can access all the system's memory quite reliably.
The attack works setting aside the fact that opt_allow_superpage is true or not.
This bug has been introduced when the support for superpages has been added.

1.2. Risks involved

Malicious PV guest administrators can escalate privilege so that they can control the whole system
Admittedly, this bug is a subtle one, because the vulnerability is not really present in the code but has to injected by the attacker; but the fact that this bug has been existing comfortably for the past 7 years is very disturbing. Xen is used by many of the websites mostly because of the security factor it provides. Being a type-1 hypervisor for so long, this simply implies that the development process is not actually prioritizing security as much as we think it is. The fact that there have been many bugs listed in XSA which have been more specifically found on Xen itself. There has been an addition of many new features over the years lately, but development in security of the system has always been lacking.

Vulnerable Systems:
Xen 3.4 and the versions after that are vulnerable.
Only x86 systems are vulnerable; ARM systems are safe.
Both 32-bit and 64-bit PV guests can exploit the vulnerability.

1.3. Mitigation and Counter Measures

Running only HVM guests can help in avoiding this vulnerability.
The defect can be efficiently solved if para-virtualized(PV) VMs are completely removed from usage and HVMs are used. The better option would be to use PVH VMs for better isolation. But this seems to be a valid counter measure only if the processor also supports SLAT. Otherwise, the complexity of the hypervisor code needed to implement Shadow Paging offsets any potential benefits of using CPU-assisted virtualization.
Majority of the modern laptops support SLAT, so this might be a possibly successful counter measure.

Patching:
Given the significance of the bug, there were patches made in Xen. Packages have been patched directly to the repository to avoid further complications. The packages are:

```
* xen packages, version 4.4.3-8 (R3.0)
* xen packages, version 4.1.6.1-23 (R2)
```

The patches made are:

```
xsa148.patch                xen-unstable, Xen 4.6.x
xsa148-4.5.patch            Xen 4.5.x
xsa148-4.4.patch            Xen 4.4.x, Xen 4.3.x

$ sha256sum xsa148*.patch
f320d105a4832124910f46c50acd4803fe289bd7c4702ec15f97fb611b70944d  xsa148.patch
7f78efd001f041a0e5502546664d28011cb881d72c94ea564585efb3ca01ddfe  xsa148-4.4.patch
272a729048471cea851d4a881f3f2c32c7be101e2a452d2b2ceb9d66908ee4a3  xsa148-4.5.patch
$
```

Task 2: OWASP Broken Web Applications- WebGoat

WebGoat is a deliberately insecure J2EE web application designed to teach web application security concepts, maintained by OWASP. While running a WebGoat application, a computer becomes very vulnerable. And hence, it is suggested that the computer is connected directly to the local host and the internet connection to the working computer is switched off.

To complete this task, we have downloaded and installed the WebGoat application, Burp suite application and the Hacker Firefox browser.

Burp suite is a Java application that can be used to secure or penetrate web applications. It comprises of different tools like the proxy server, a web spider, intruder and a repeater.

When Burp Suite is used as a proxy server, it allows the user to manipulate the traffic that passes through it, i.e., between the web browser (client and the web server). This is a typical instance of Man-in-the-middle (MITM) type attack architecture. Burp uses tables to make changes to web traffic, in order to manipulate data before it is sent to the web server. Using this functionality, exception situations can be reproduced, allowing any bugs and vulnerabilities on the webserver to be accurately pinpointed.

The Hacker Firefox is a portable Firefox with web hacking add-ons useful for instant web security assessment [7]. Video tutorials in the WebGoat application were helpful for us to play around and execute the selected tasks. The three task vulnerabilities exploited in Web goat as discussed below.

Sub-task 1: Buffer Overflows- Off-by-one overflows

Vulnerability and Method

Initially, we start the Web Goat application and parallely open the Hacker Firefox and Burp Suite applications.

Clicking on Start WebGoat in Hacker Firefox will start the application. Choosing our vulnerability from the left side of the browser window, we proceed into the task.

In this, we are accessing the Internet provided in OWASP Hotel using Buffer overflow technique, without actually making a payment for it. For the execution, we followed this method:

The proxy address is changed to **127.0.0.1: 8080**, after selecting the vulnerability in the Hacker Firefox browser, and **Apply** the changes made. Next, we switch the intercept to OFF mode by clicking on **Proxy-> Intercept->Intercept is On** button. The **Unhide hidden form fields** is checked in the Response Modification section. Now, we switch on the **Intercept is OFF** button.

We entered *random* credentials for the first step: First name- Harry, Last name- Potter and Room number- 1202, in the Hacker Firefox browser and **Submit** the form.

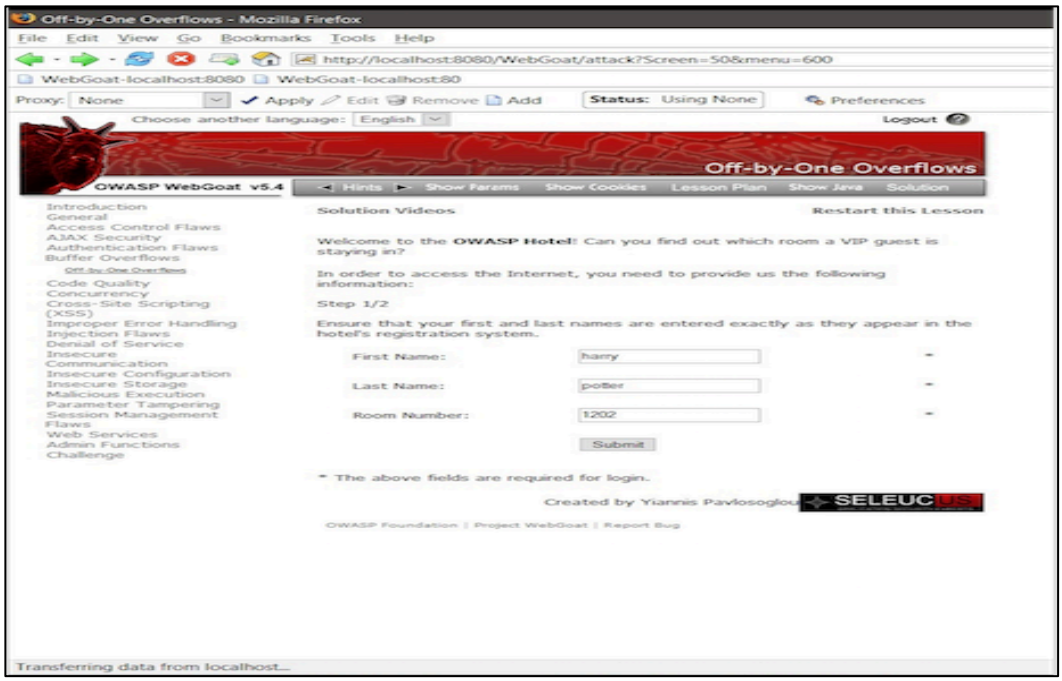

Figure 4: Entering Random credentials

Noticing the Burp Suite browser, we can see our randomly entered credentials, and this is the place where we use our buffer overflow technique to exploit the vulnerability in the source code- which is that the buffer accepts any character in the room number field.

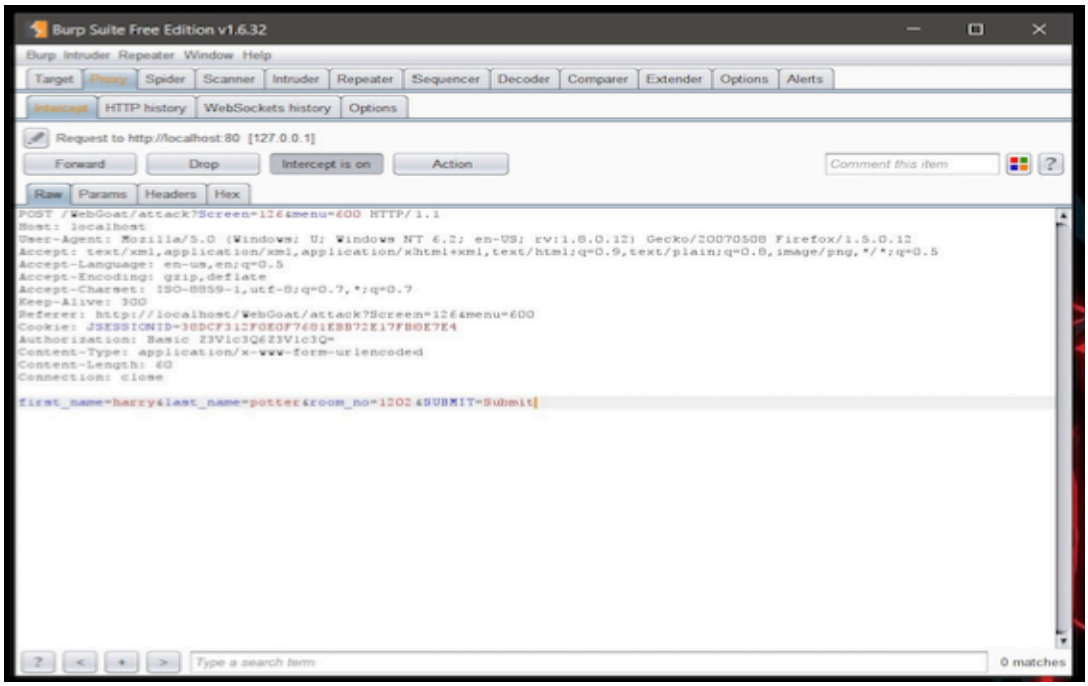

Figure 5: Burp Suite showing the credentials entered

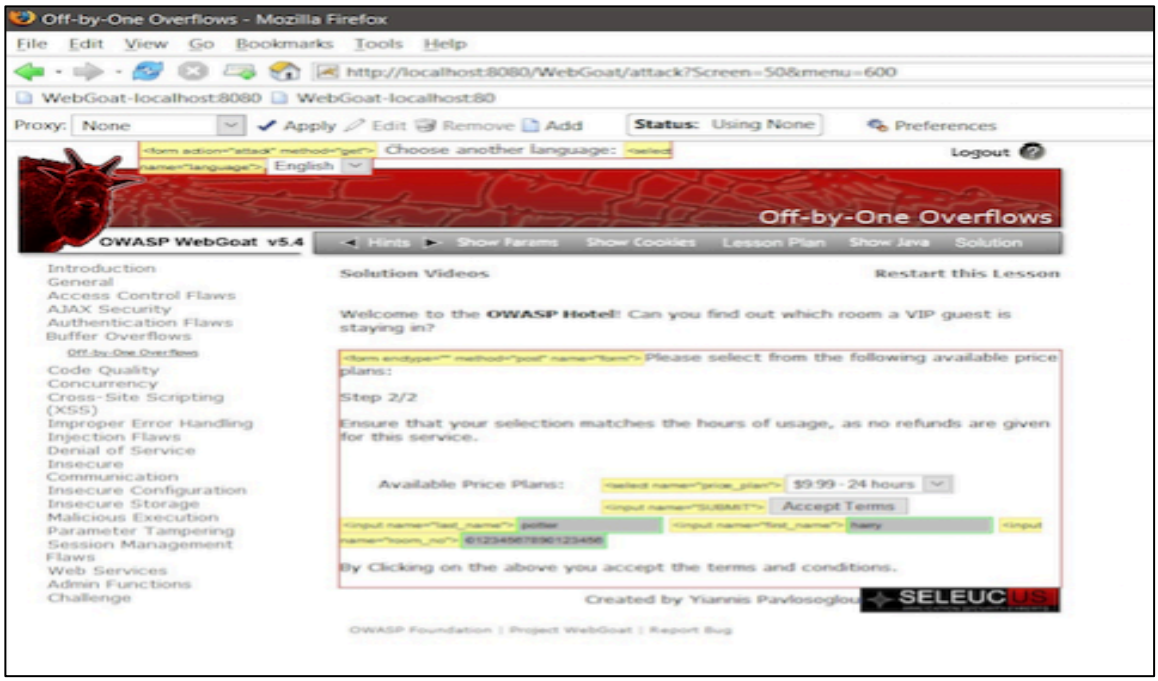

Figure 6: Source code of the form

We created a separate file of **4097** characters to overflow, as the buffer is holding 4096 characters in total for correct functionality (This was found using trial and error method). The text in this file is now substituted in the space provided for room number, and we click **Forward** button in the Burp Suite application.

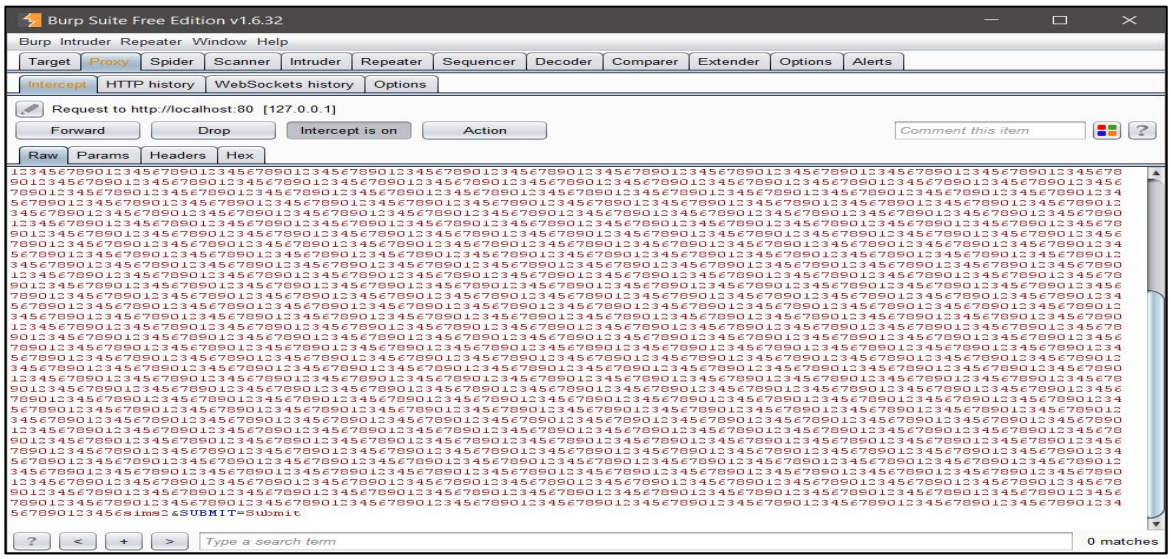

Figure 7: Buffer overflow of 4097 characters

Now, the same random credentials are filled in the Hacker Firefox browser and we **Accept Terms**. Switching to the Burp Suite application, the **Forward** button is clicked. This will show us the credentials of all the VIPs residing in the hotel. We turn **OFF** the Intercept in the Burp Suite application.

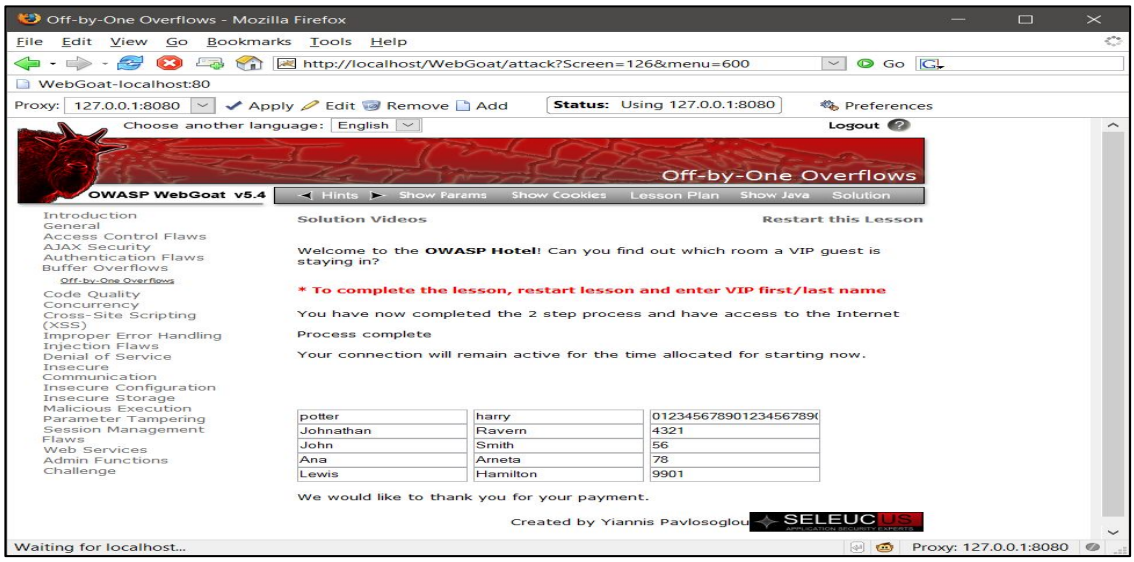

Figure 8: Details of all the VIPs staying at OWASP hotel

Using these credentials, we can now successfully login using anyone's id and have an access to the Internet. For this, the WebGoat is restarted in the Hacker Firefox browser and using the credentials of VIP's acquired, we *access the Internet* on their name. The VIP will not even be aware of his lost credentials.

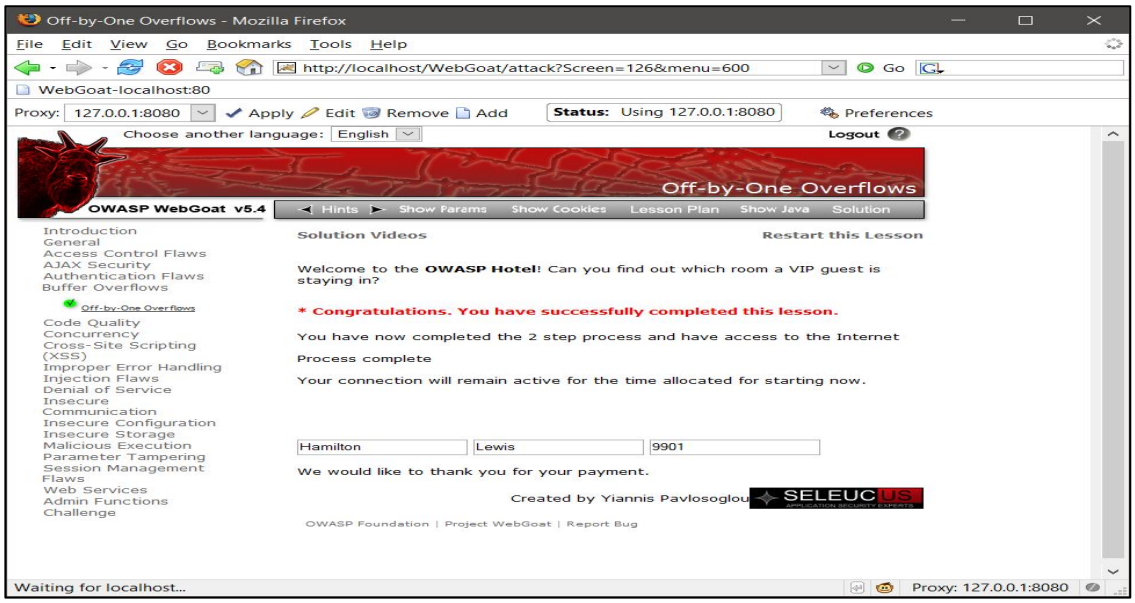

Figure 9: Accessing Internet using a VIP's credentials

Counter Measure

The memory for storing the room number is not properly defined. Any restriction on the size of buffer would have prevented this exploitation. This can act as a restriction for the overflow.
Using safe functions like fgets(), strncpy(), strncat() etc in coding could also have prevented this. This could have prevented the attacker from gaining access to confidential data.

Sub Task 2: Denial of Service- DoS from multiple logins

Vulnerability and Method

Denial of Service attacks have become a major issues in web applications. If the end user cannot conduct business or perform the service offered by the web applications, then both time and money are wasted in the process. Business loss may count up to millions of dollars. If an e-commerce site can generate 1 million per hour, then that hour of DoS attack can create a loss of 1 million for that business. The vulnerability we have exploited here is denying service by using multiple logins. The method we followed is described below.

The web application has a two connection for the database. We can create an overflow if this is to be exploited, for which we need a minimum of three valid user login credentials to create a DoS attack.

We start by finding out the valid users in the system, for which we use a *SQL injection* method. On entering the username and password as **test' or '1'= '1'**, we get a list of valid users.

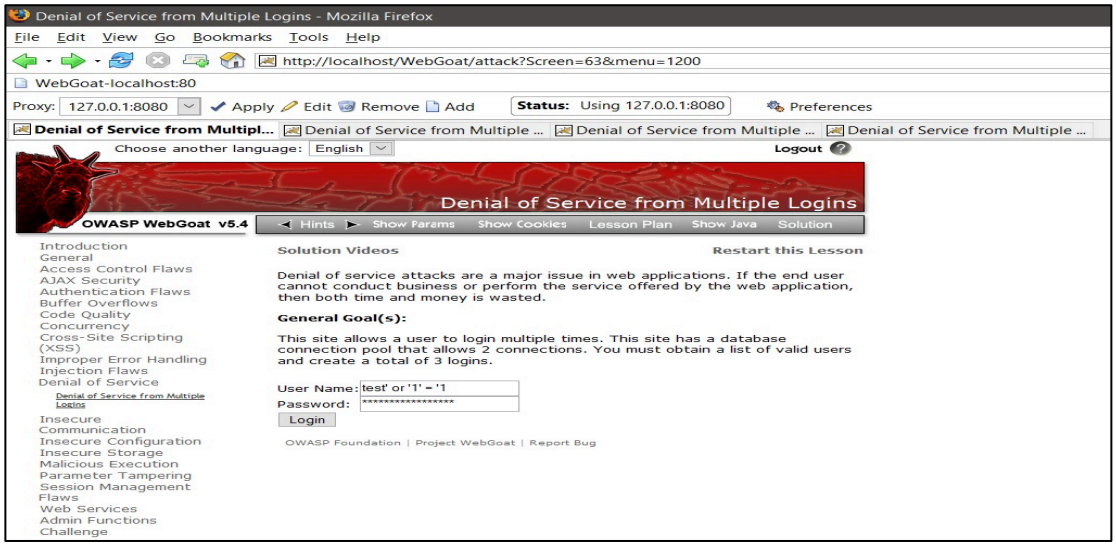

Figure 10: Logging in using SQL Injection

We copy the details separately in a notepad file so that it will be useful for logging in.

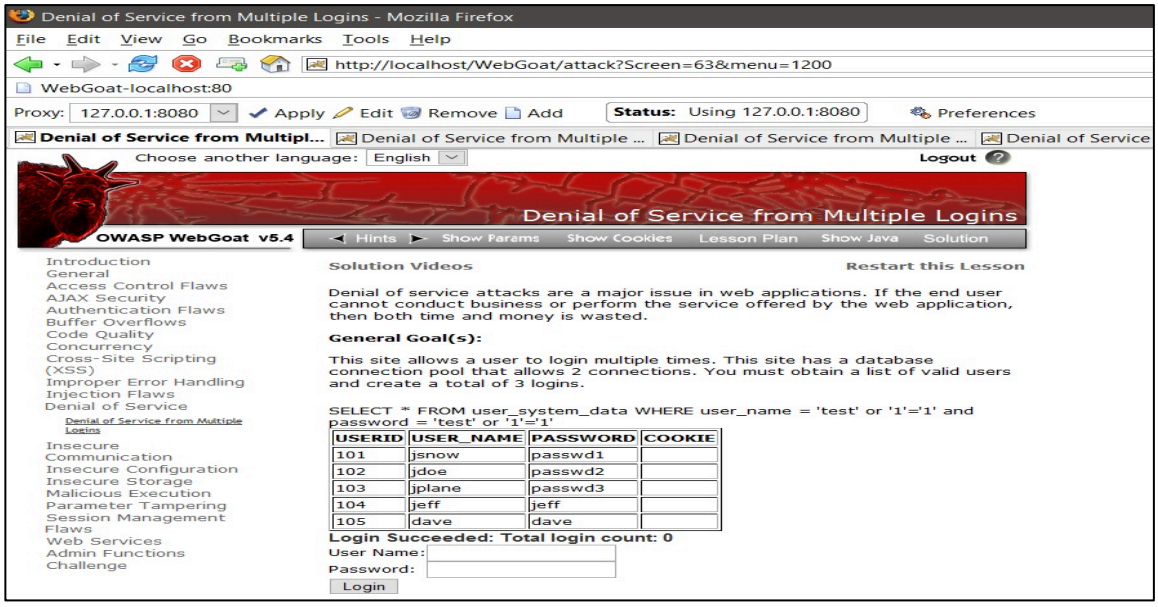

Figure 11: Details of valid users

We use those details and log in using three different credentials, which completes the task.

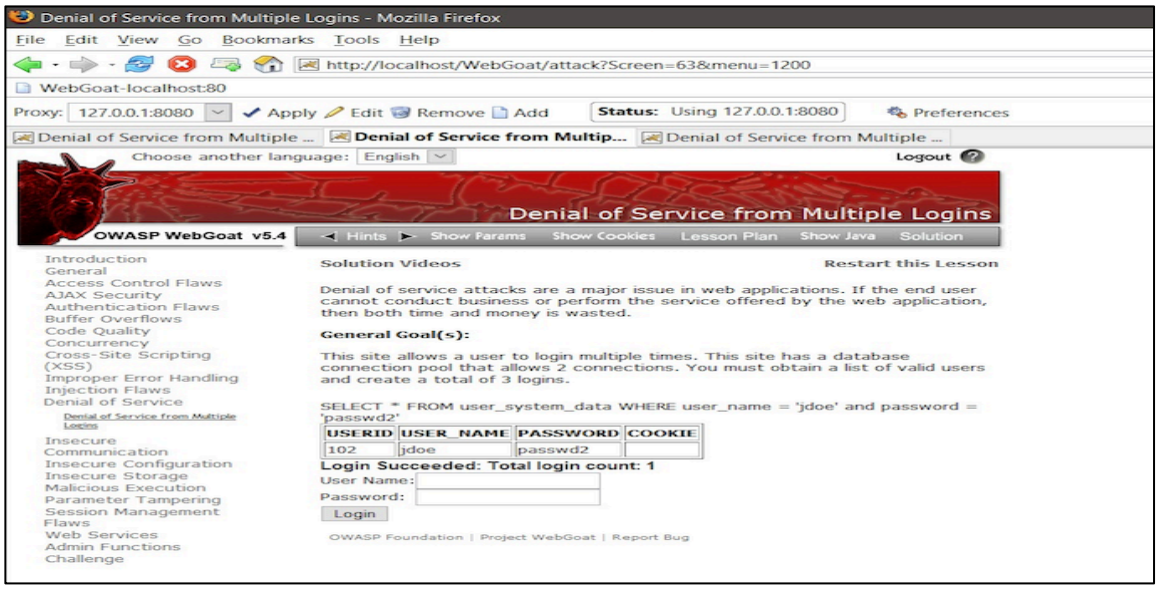

Figure 12: First login attempt

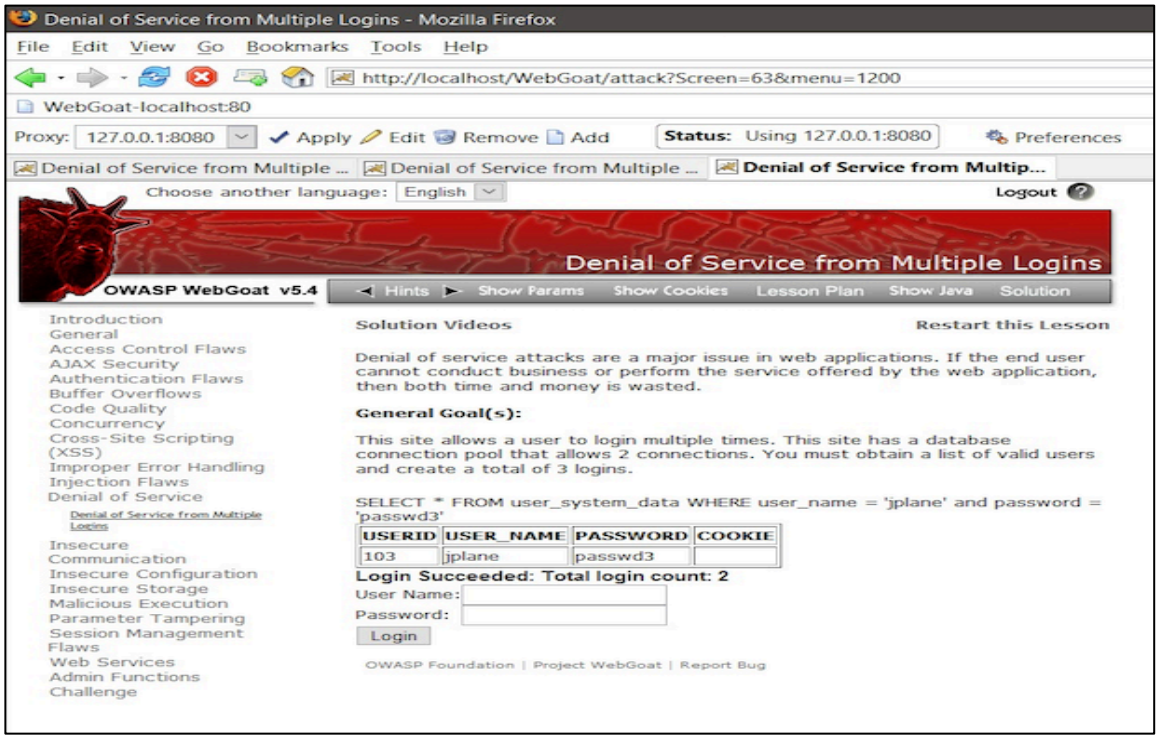

Figure 13: Second login attempt

We are now able to exploit the vulnerability, in a website where it only accepts two connections.

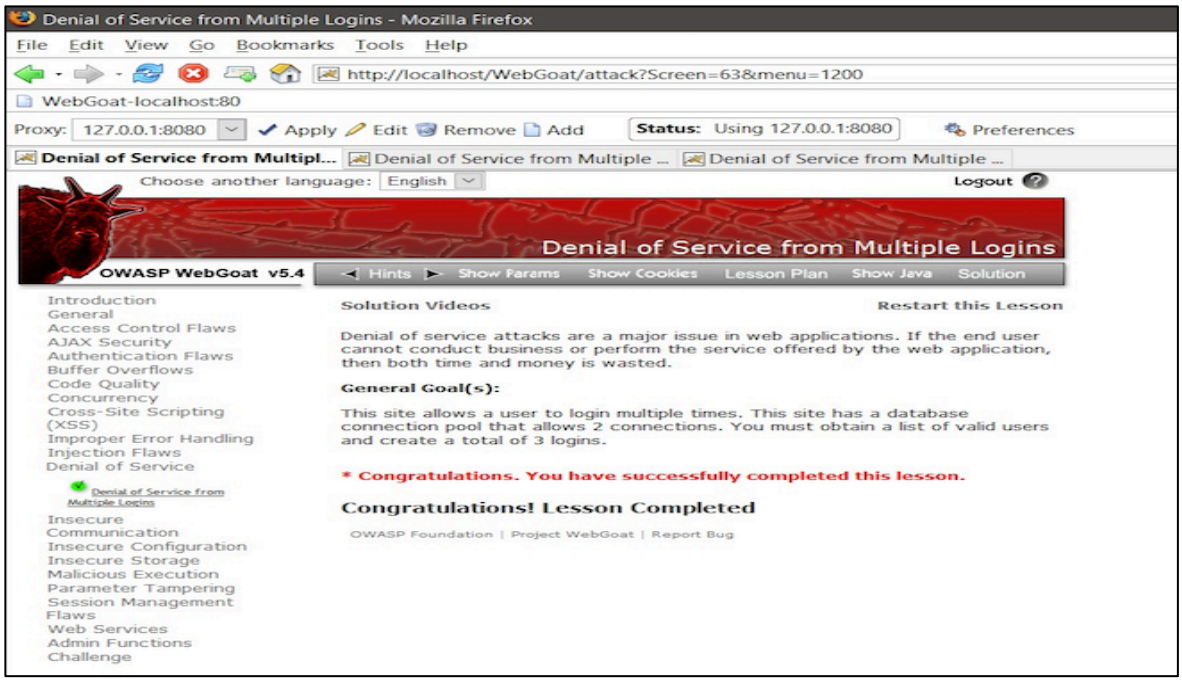

Figure 14: Successful third login

Counter Measure

Care must be taken not to store too much data in a user session object. Storing a lot of information in a session is the reason for Denial of Service attacks.
The authentication system of the target application also has to be modified. A possible defense mechanism is to stop Brute- Force technique of exploitation, which stops any trial for possible passwords after three to five attempts of login. But, this could mean that even a legitimate user will not be able to access their account if some hacker has attacked it unless attacked. This defense mechanism could possibly be turned into a DoS attack against an application if there is a way to predict valid user login accounts.
There is a business versus security balance that must be reached based on the specific circumstances surrounding an application. There can be pros and cons to locking accounts, but each enterprise would have to balance their risks and benefits.

Sub Task 3: Injection flaws- Numeric SQL

Vulnerability and Method

SQL Injection is a code injection technique, used to attack data-driven applications in which malicious SQL statements are inserted into an entry field for execution, for instance, dumps the database contents to the attacker.
Attacks against numeric parameters are the simplest way to achieve an SQL injection. This kind of vulnerability is also widely spread since developers often consider that numeric parameters are safe when in most of the cases, they are not.
The goal of this attack is to fool the web application to list data it should not. For that, we will need to formulate a response which tricks the application in displaying all weather stations and their confidential data.
To accomplish this task, we need “Tamper Data” plugin. For that, we need to go to Web Developer in Tools and then choose Get more tools. We can install Tamper Data by searching it and then downloading the plug in. Tamper data is to enterprise with the command, similar to Man-in-the-middle.
Go to Tools->Tamper Data->Start Tamper.

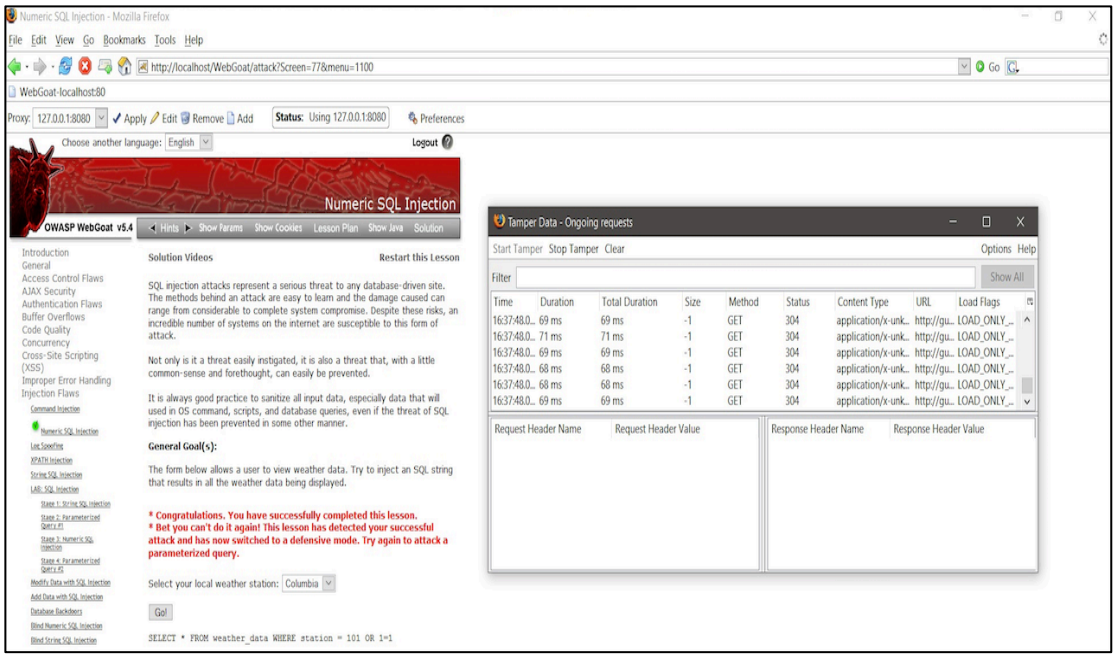

Figure 15: The window after we start Tamper

Now, press Go in the WebGoat server, where we are working on Numeric SQL Injection.
In the Tamper Data window, press Tamper when it asks if we would like to continue tampering with the requested host.

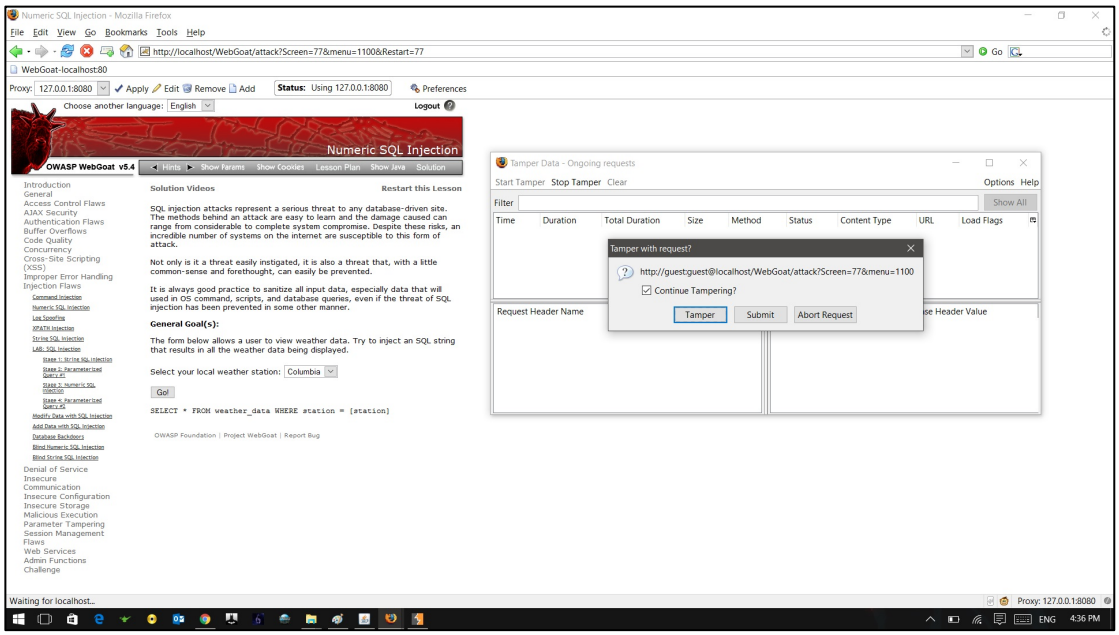

Figure 16: Continuing Tamper

A Tamper Popup window appears, where we change the station details to **101 or 1=1**. "1=1" is the SQL string used to get information about weather data. Press OK.

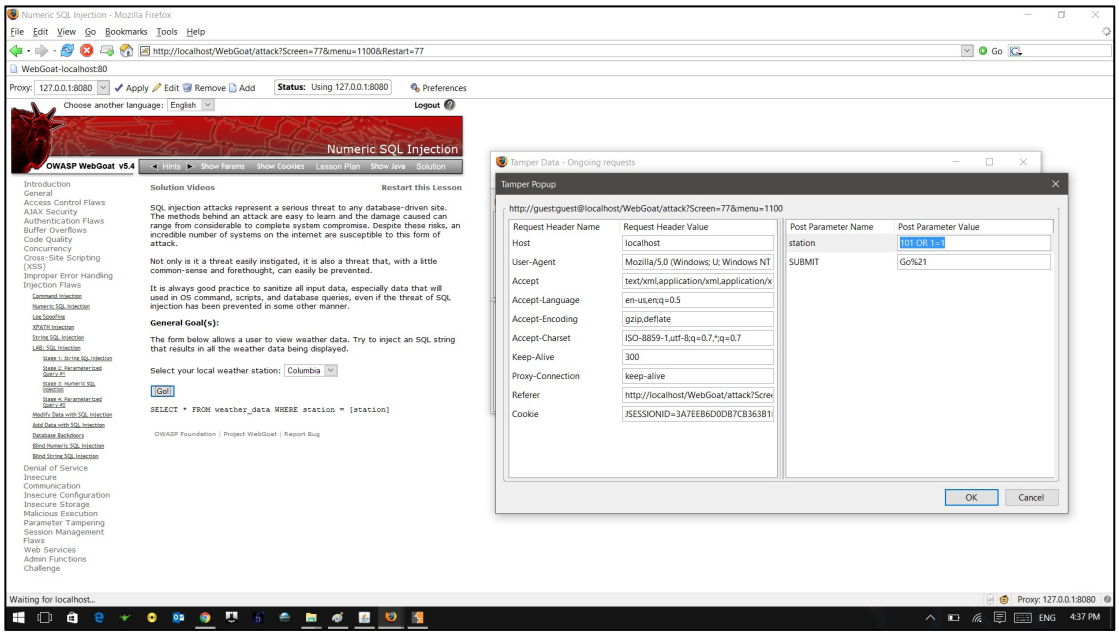

Figure 17: Changing details in station

This completes the task as we will be able to view the weather data with Station, Name, State they are in, the minimum temperature and the maximum temperature of the area.

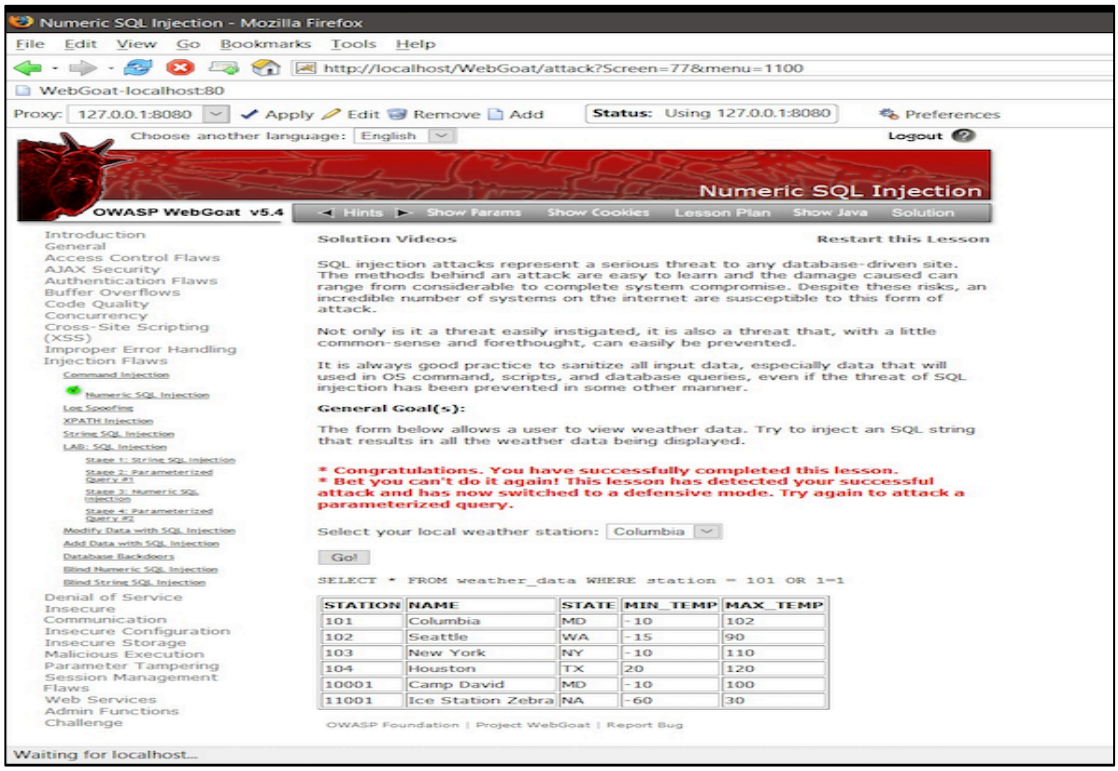

Figure 18: Details of all the weather data at different stations

Counter Measure

Injecting text into numeric parameter is definitely counter intuitive. Nevertheless, it is possible because some weakly typed languages do not force variables to keep their initial data type. As a result, it is possible to insert a crafted SQL statement in parameters that were supposed to contain numeric values. Hence, care must be taken that numeric data types are not vulnerable to such attacks.

Task 3: OWASP Broken Web Applications- Mutillidae

Mutillidae is a free, open source, deliberately vulnerable web application providing a target for web-security enthusiast [10]. Being the third task of our lab assignment we were required to do 10 forms in order to get acquainted with the vulnerabilities in Web Security.
We download the xampp software and paste it in the mutillidae folder in htdocs. We run the Apache, MySQL and Tomcat servers in xampp and in the browser, we open the mutillidae web page by typing the IP address in the space for url. The vulnerabilities are listed to the left side of the page from which we were asked to select 10 forms.

Form 1: A1-SQL Injection-Extract Data- User Info (SQL)

Vulnerability and Method

The first vulnerability we have selected from SQL Injection is to extract user info. This task requires an attacker to login through admin credentials and extract data. We need to perform SQL Injection as we do not have the admin credentials. We give the username as **admin** and the password is given as **admin' or '1'='1.**

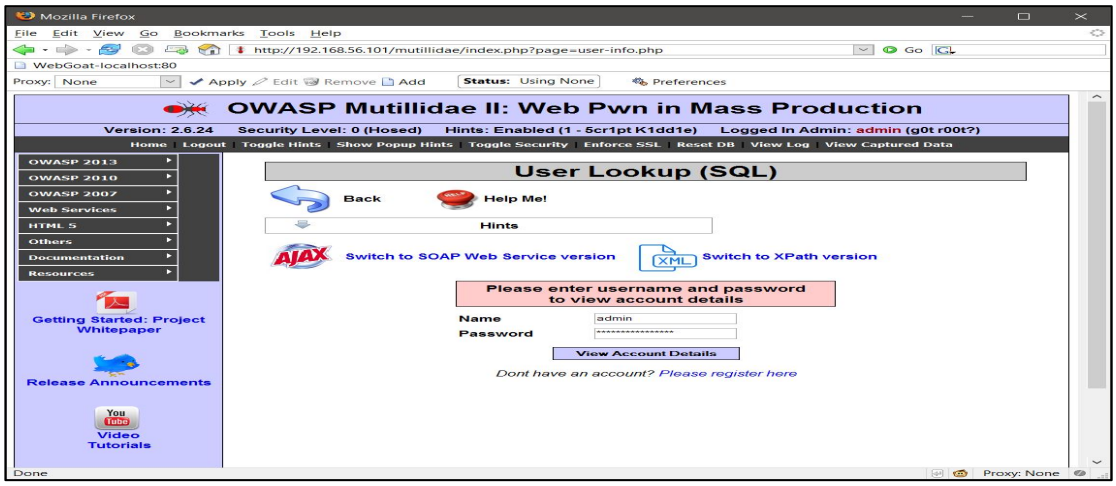

Figure 19: Entering the username and password

On clicking the **Login** button, a list of usernames, passwords and signatures of all the registered users is displayed. Thus, the data can be extracted using SQL Injection.

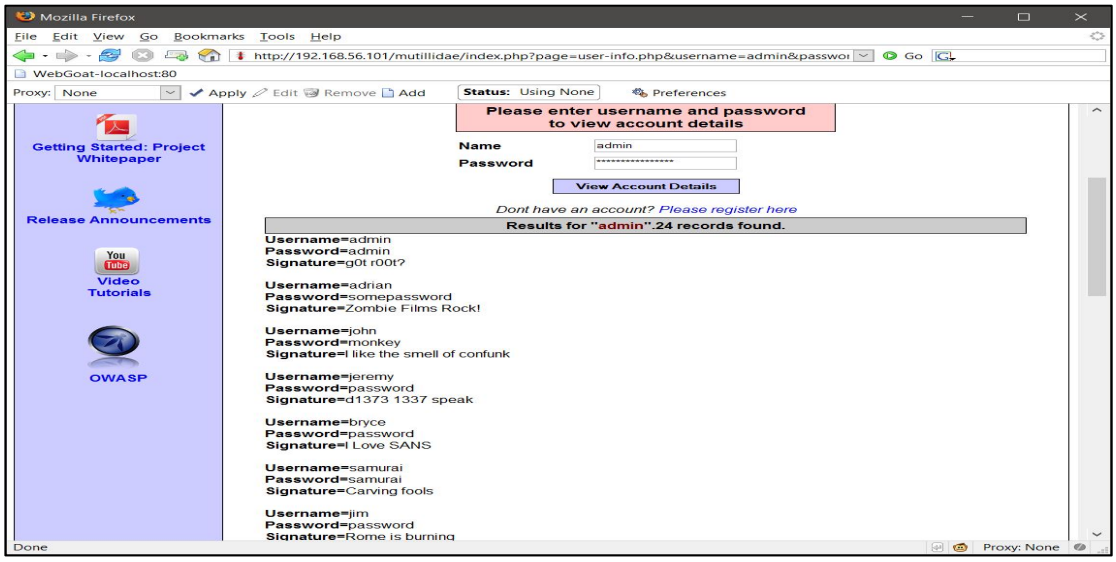

Figure 20: Credentials of registered users displayed

Counter Measure

The improper use of validation controls for passwords led to this exploitation. If this can be checked, then the vulnerability is removed. The validation controls should be able to restrict any kind of SQL Injection attack.

Form 2: A1- SQL Injection- Bypass Authentication- Login

Vulnerability and Method

The second vulnerability we have selected from SQL Injection is *Bypass Authentication-Login* vulnerability. The steps followed to exploit this vulnerability are as follows.
After selecting the vulnerability, an authentication page opens up. We need the admin credentials in order to login; but as we do not have them, we have to perform an SQL Injection. The username is given as **admin** and password is **admin' or '1'='1**. We have used the same method for Denial of Service attack in our subtask 2 for task 1.

Figure 21: Credentials entered

The admin page can be retrieved on clicking the Login button. Thus, the SQL Injection vulnerability has been exploited, giving us access to the admin page.

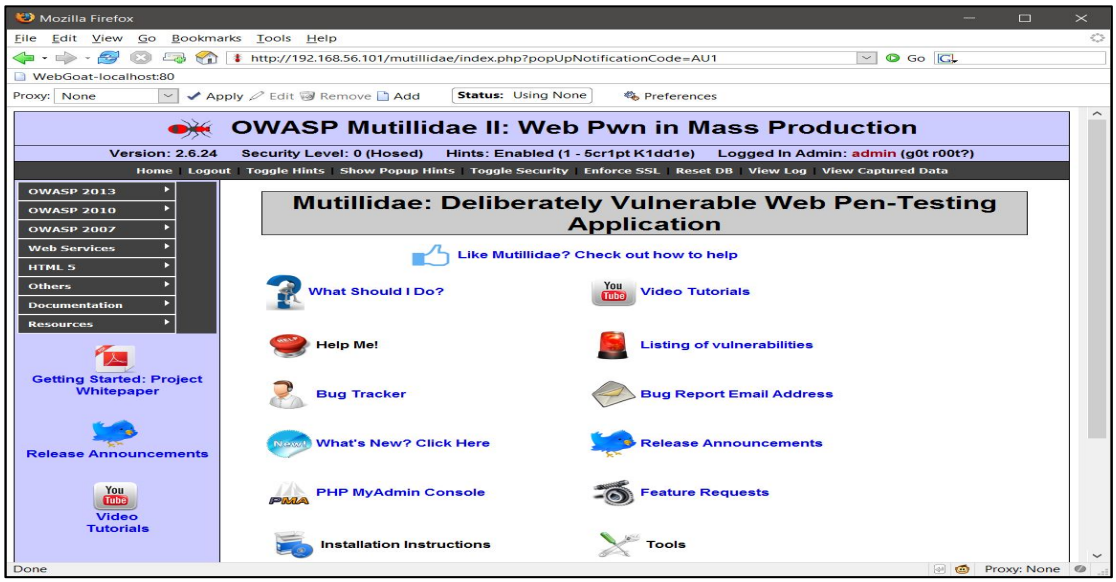

Figure 22: admin page displayed

Counter Measure

The improper use of validation controls for passwords led to this exploitation. If this can be checked, then the vulnerability is removed.

Form 3: A2- Broken Authentication and Session Management – Authentication Bypass-via SQL Injection

Vulnerability and Method

The first vulnerability we have selected from Broken Authentication and Session Management is *Authentication Bypass Authentication-via SQL Injection* vulnerability.
We gain authorization in this task using SQL Injection, through Tampering data (similar to sub task 3 in task 1).
The steps followed to exploit this vulnerability are as follows.
Select **Tools** -> **Tamper Data** ->**Start Tamper**.

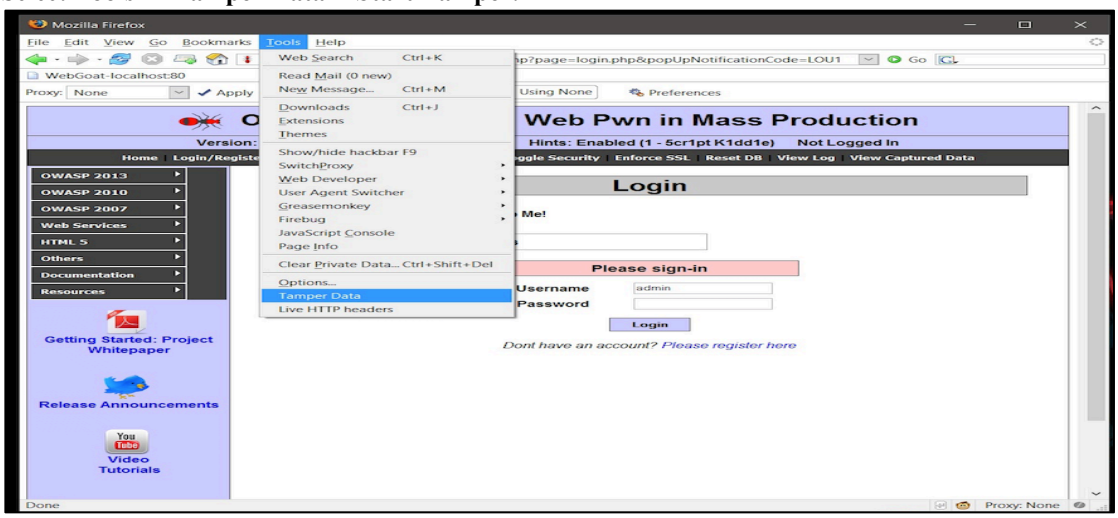

Figure 23: Tamper data option in Tools

In the authentication page, enter password as pass and click **Login**.
A tamper Request window opens up, where we have to choose **Tamper**. In the tamper popup, change the password element to **='or'1'='1**. Click **OK** after changing the password.

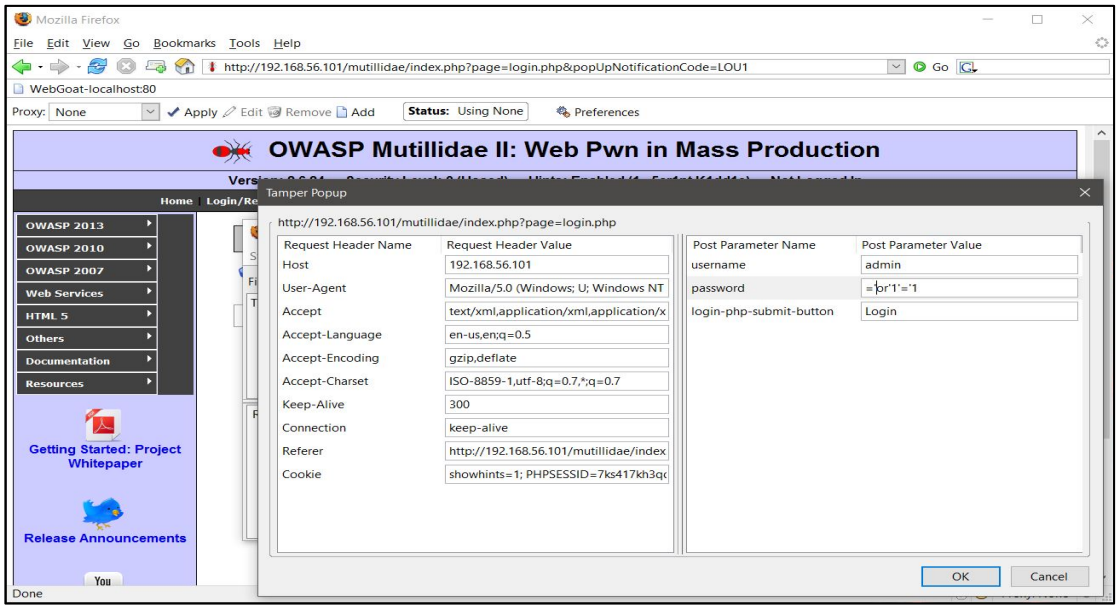

Figure 24: Tamper popup window

Click **Submit** after unchecking the **Continue tampering** option. We can view the homepage after this; thus, exploiting the vulnerability.

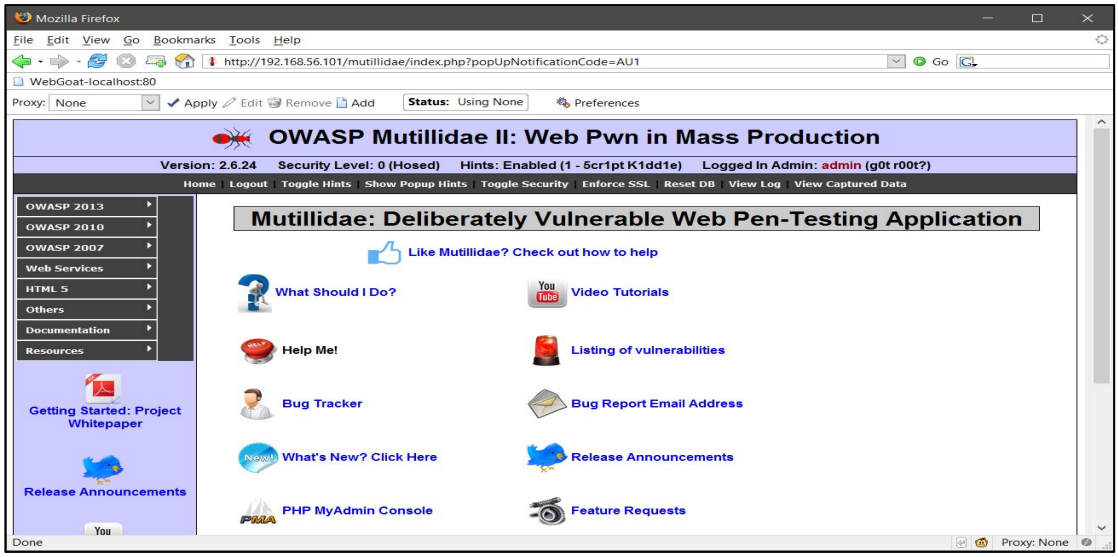

Figure 25: Vulnerability exploited

Counter Measure

If the validation mechanism for credentials was strong enough, this exploitation could have been avoided.

Form 4: A2- Broken Authentication and Session Management – Privilege Escalation-via cookies

Vulnerability and Method

The vulnerability we chose is self explanatory; an attack can be made by exploiting through cookies. The steps followed to do so are as follows.
Initially, we login as a basic registered user.
Select **Cookies** from **Web Developer** option in the **Tools** menu. Choose **Cookie Information** option.
Click on **Edit Cookie** from **uid cookie**.
In the Edit Cookie pop up window, change the value to **1,** which indicates admin and *check the Session cookie* option.

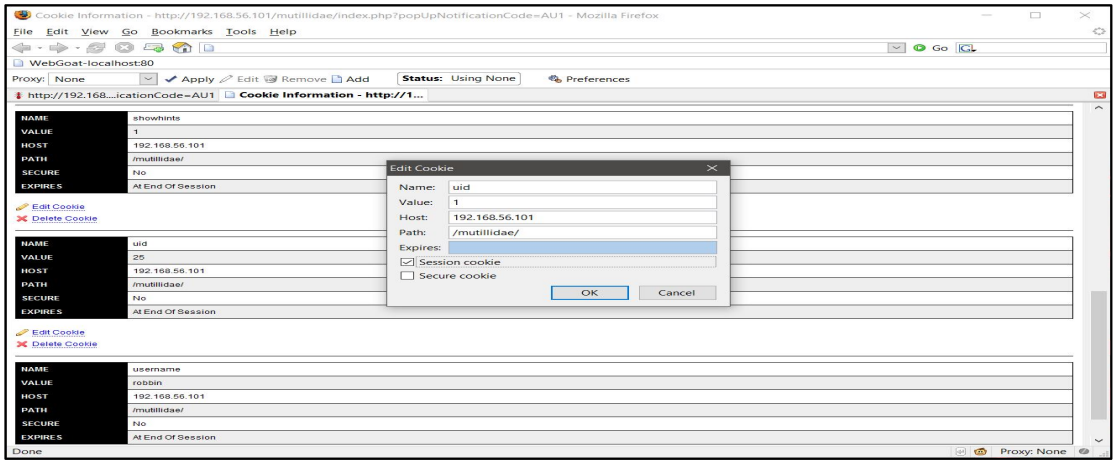

Figure 26: Edit Cookie pop up window

The attacker can log in as admin after clicking on the **OK** button. Thus, the Broken Authentication vulnerability is exploited via cookies.

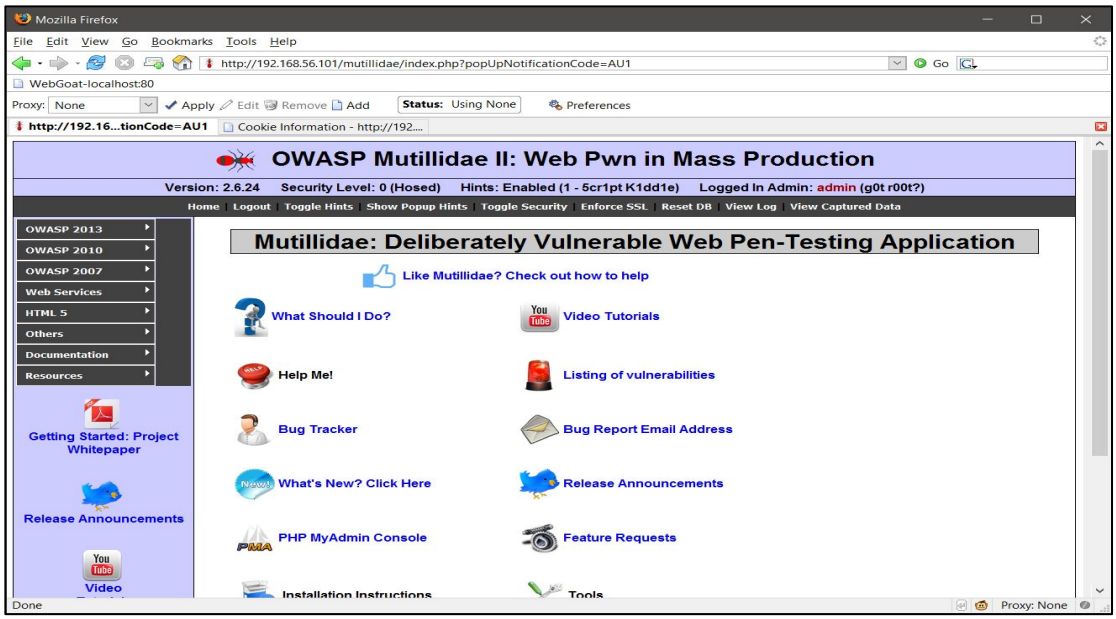

Figure 27:admin page accessed

Counter Measure

Users should not be provided with the cookie information. By preventing access to cookie, the access to the admin page can be prevented. Session ID protection is also a good way to avoid the exploitation.

Form 5: A3- Cross Site Scripting (XSS)- Reflected (First Order) – Set Background Color

Vulnerability and Method

From the third set, we chose Set Background color vulnerability. The script written can exploit this vulnerability. In order to complete this task, the script is typed in the space provided for background color.

<script> alert ("Cookies which do not have the HTTPOnly attribute set"=document cookie)</script>

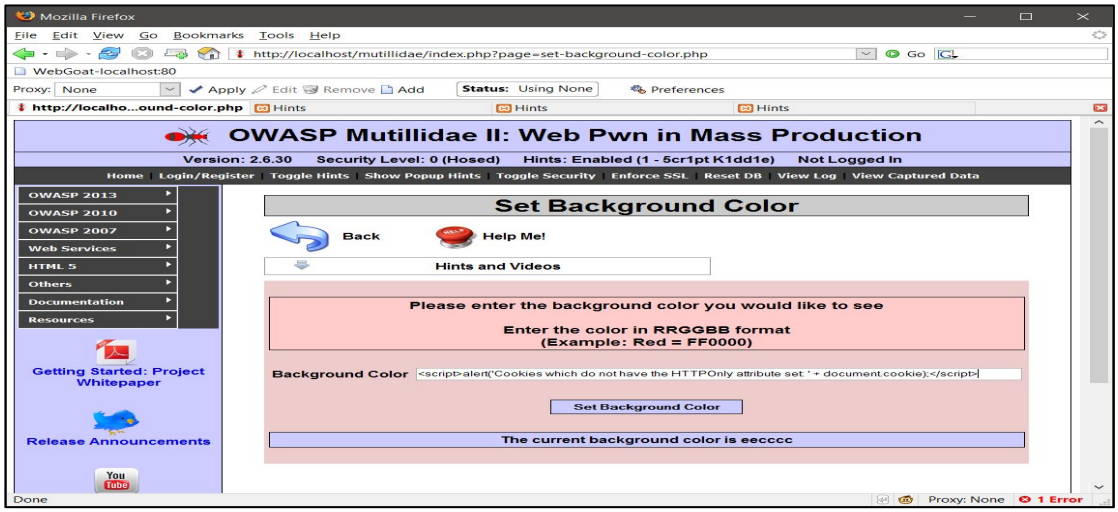

Figure 28: Entering the script

This piece of html script will do the job. It derives the cookies and redirect them to the attackers account.

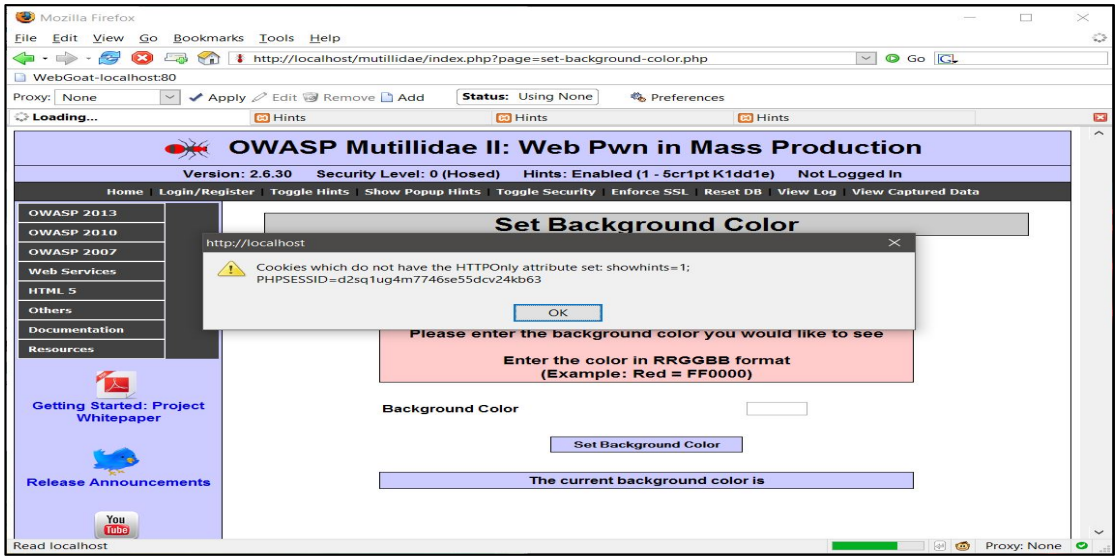

Figure 29: The source code

Counter Measure

Strict rules to avoid unsafe entry of data should be set up for input data. This can act as a counter measure to prevent Cross Site Scripting vulnerability.

Form 6: A3- Cross Site Scripting (XSS)- Persistent (Second Order) – Add to your blog

Vulnerability and Method

The second task selected in set 3 is Add to your blog vulnerability.
This vulnerability is exploited by redirecting the data of the valid user to the attacker's blog. We cause the attack by introducing the script in valid user's page.

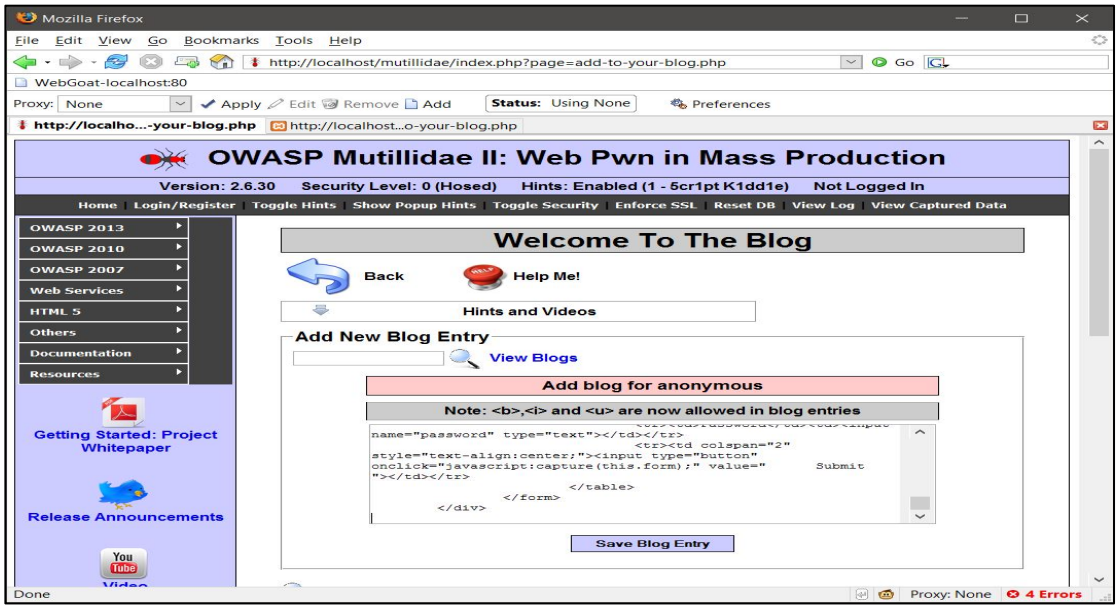

Figure 30: Malicious code entered by the attacker

An error message is displayed stating that wrong credentials have been entered, when a user tries to login. This entitles a legitimate user to enter their details once again, which will now be redirected to the attacker.
As soon as the user clicks on the **Login** button, the attacker's page opens up showing all the credentials of the user.

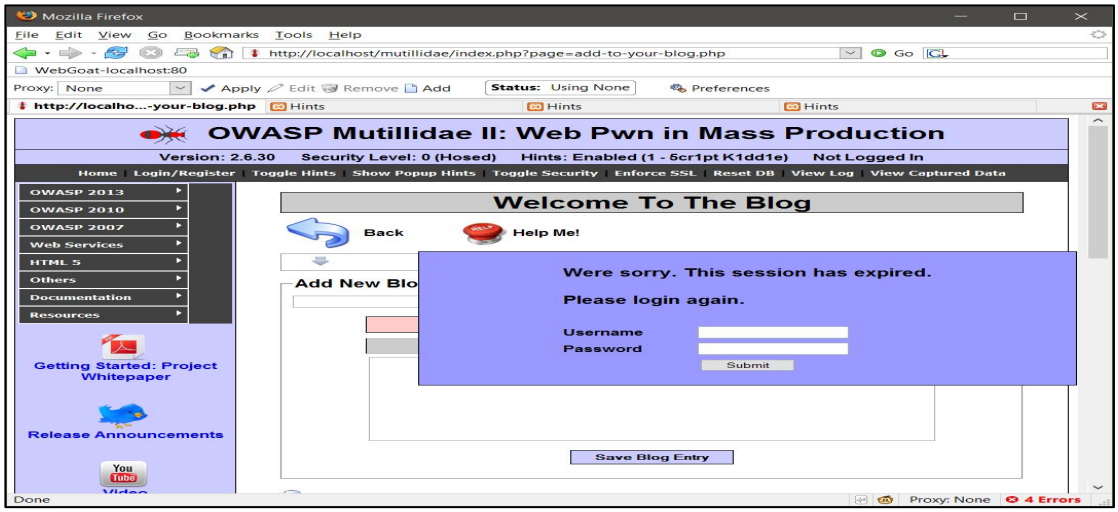

Figure 31: Error message displayed

Thus, the vulnerability has been exploited through *add-to-your-blog* parameter.

Counter Measure

Strict rules to avoid unsafe entry of data should be set up for input data. This can act as a counter measure to prevent Cross Site Scripting vulnerability.

Form 7: A4-Insecure Direct Object References- Source Viewer

Vulnerability and Method

The next vulnerability we chose is *Source Viewer* in Insecure Direct Object References.
As the name suggests, we are going to take authentication by viewing the source code of the credentials stored.
The steps to exploit this vulnerability are as follows. We type the name of the file in the viewer field after selecting the vulnerability.
The source code of the file can be viewed in **add-to-your-blog.php.**
Select **Live HTTP Headers** option from **Tools** menu. A list of all the HTTP headers present will pop up in a window. When we press the **Replay** button, a *live HTTP Replay* window will pop up, giving us the name of the file which can be edited.
We can give the name of the file which has to be viewed here; to view the password file, we have to replace the **add-to-your-blog.php** file to **C:/xampp/passwords.txt**.

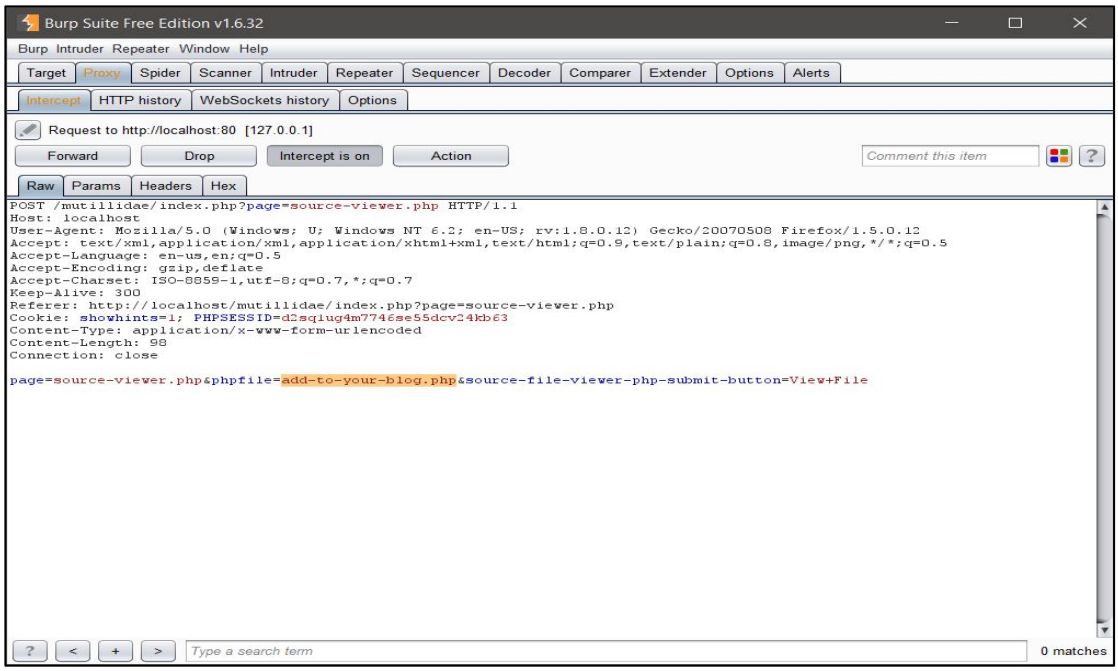

Figure 32: Location of changing path in BurpSuite

Click **Replay** button now. On doing so, the source code of the passwords.txt file is displayed, using which we can exploit the vulnerability.

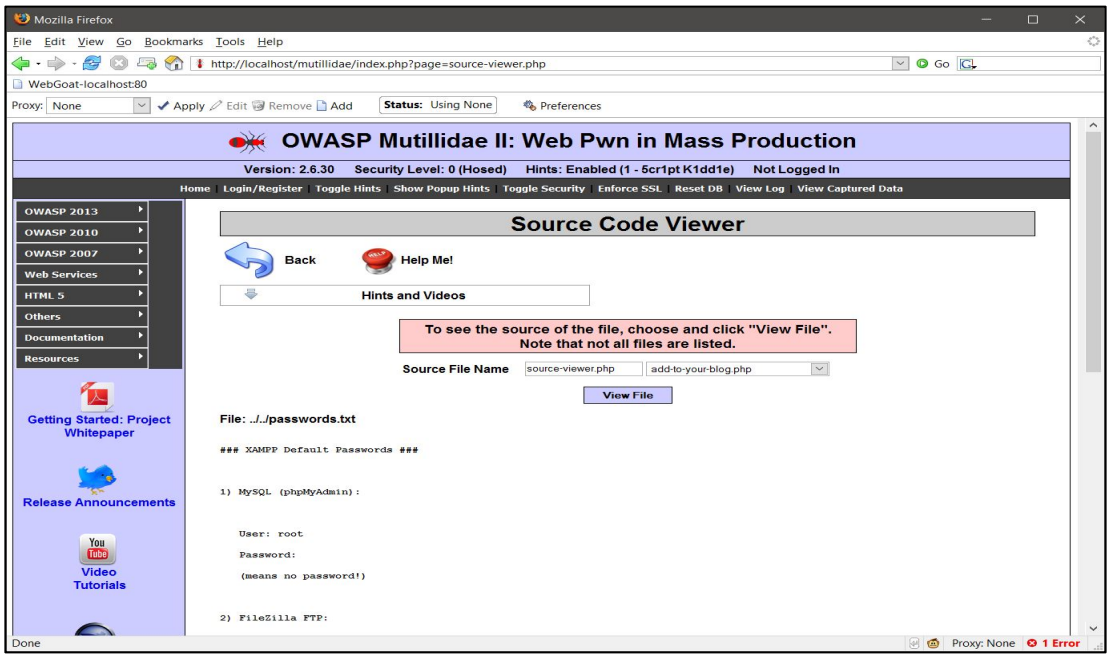

Figure 33: displayed passwords.txt

Counter Measure

An access check control for every use of direct object reference from an untrusted source can prevent this exploitation. This can act as a counter measure from Insecure Direct Object Reference Attack.

Form 8: A5- Security Misconfiguration- "Secret" Administrative pages

Vulnerability and Method

We have chosen " Secret" Administrative pages from Security Misconfiguration as our eighth form.
In this attack, we make use of the configuration settings to exploit the vulnerability. The blog of the user is displayed along with the name of the file after selecting the vulnerability.
Switching to the BurpSuite Application, we switch **ON** the intercept and the source code of the concerned file appears. Now, send the code to **Intruder** and add a **$** to the name of the file; this is for better distinguishing.

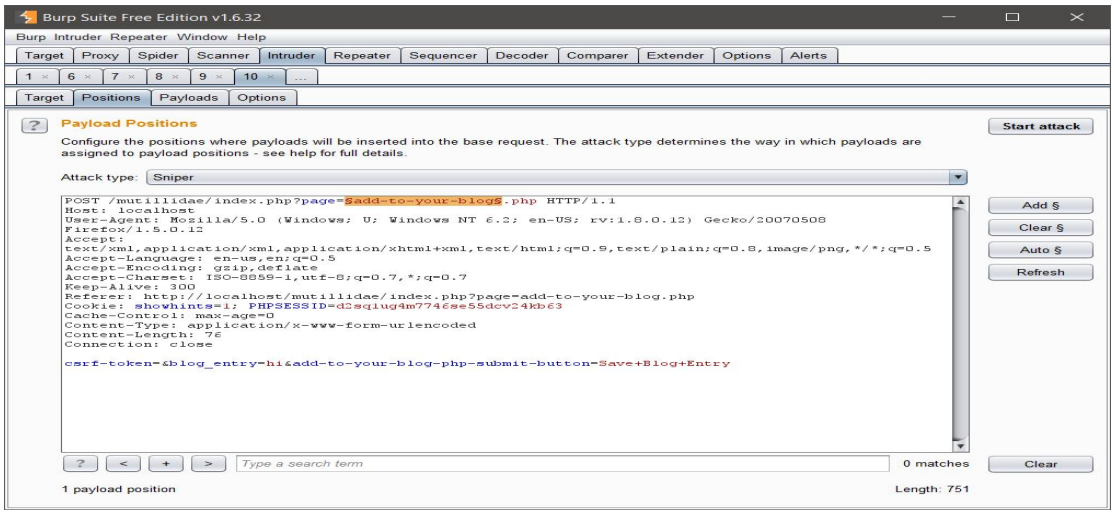

Figure 34: addition of $ to the file name

Select **Payload Options** from **Payloads** in **Intruder**, and add some random files to Payload Options. Click **Start Attack.**

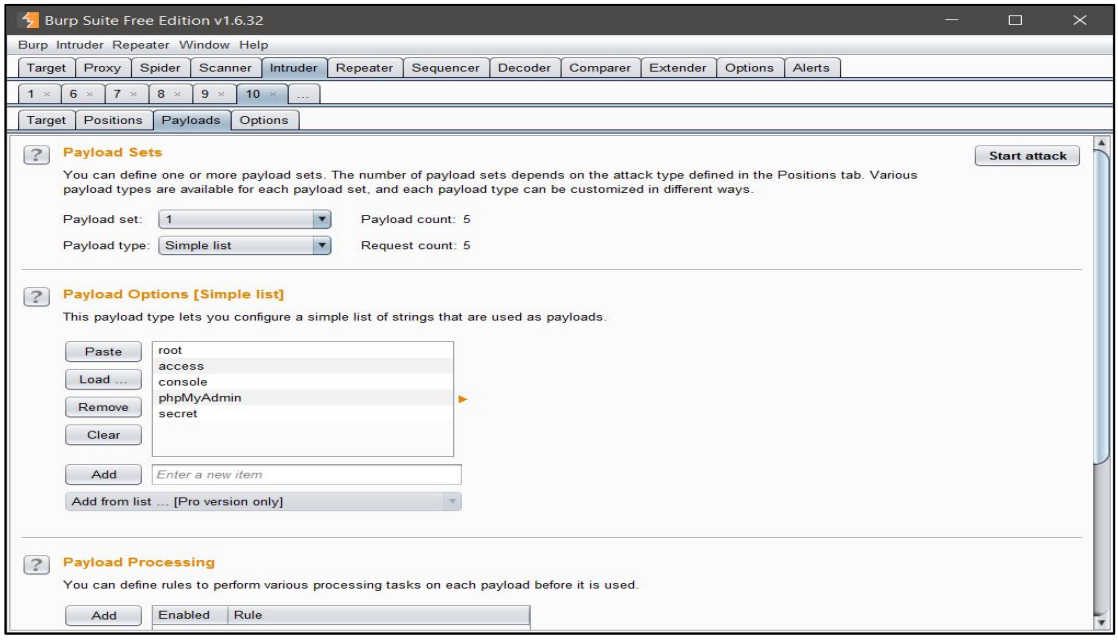

Figure 35: BurpSuite displaying filename in the intruder

An *Intruder attack 10* pop up window appears showing the result of our attack. Selecting root from those files, we choose **In current session** from **Request in browser** option.

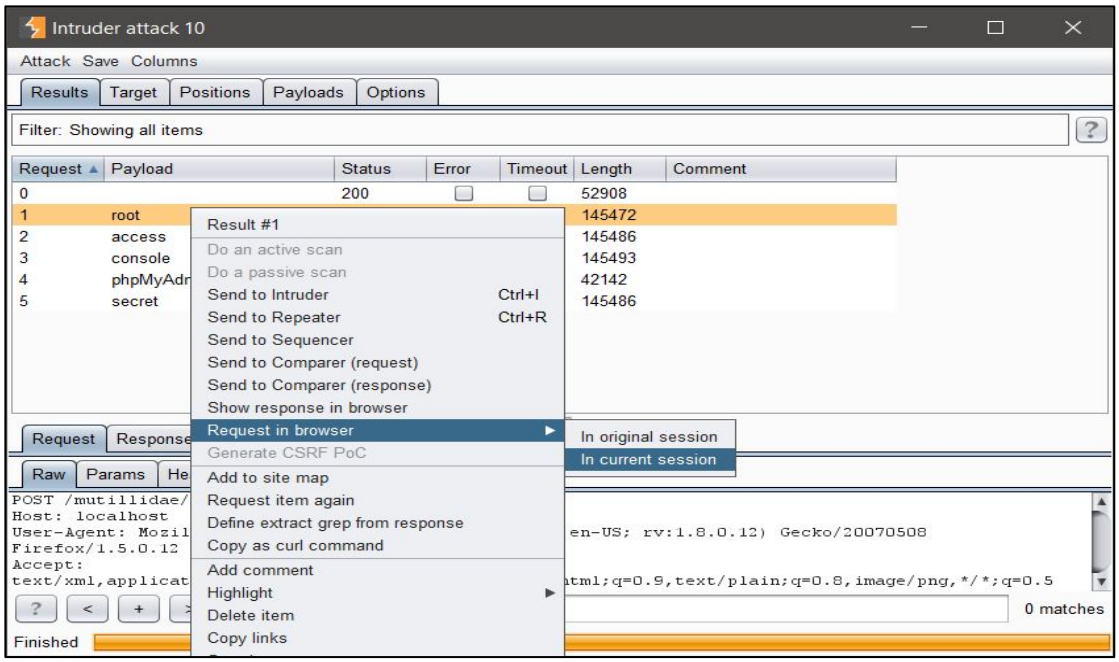

Figure 36: Options selected

We now run the required file that has to be attacked and the *source code* and *.php version* are displayed as the output. Hence, the exploitation of Security Misconfiguration vulnerability has been done.

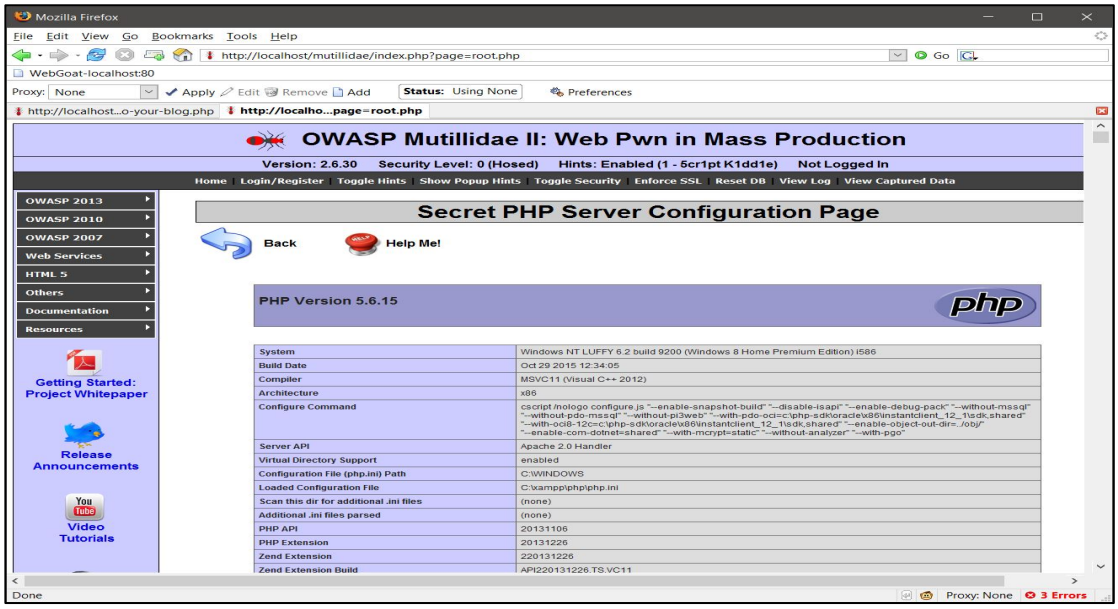

Figure 37: the php version displayed

Counter Measure

Such security flaws can be avoided if the software is updated at regular intervals. A strong application architecture should be designed to secure the components, which could act as a counter measure to prevent such exploitations.

Form 9: A5- Security Misconfiguration- Directory Browsing

Vulnerability and Method

The ninth form discussed is again from Security Misconfiguration and the vulnerability exploited here is Directory Browsing. This is similar to the previous form discussed, i.e., the vulnerability in configuration settings is exploited here.

The **directory-browsing.php** page is opened after selecting the vulnerability to the left.

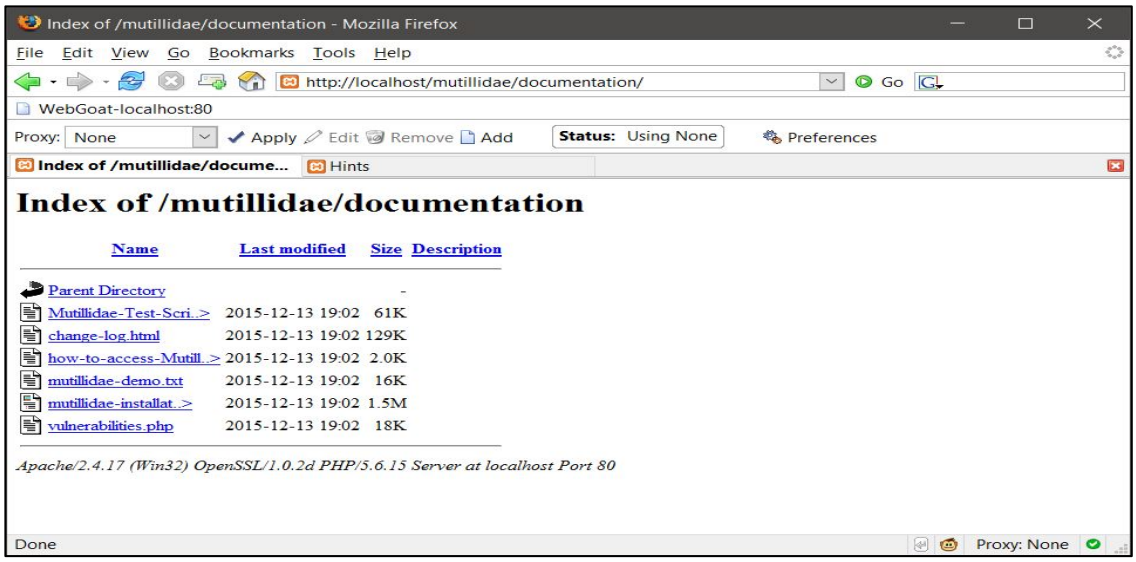

Figure 38: directories displayed

We **change** the filename in the url to something confidential like passwords folder. This will open up the passwords folder and all the files present in that folder; thus exploiting the security misconfiguration vulnerability.

Counter Measure

A simple changing of the url resulted in exploiting the vulnerability. This could be prevented if the architecture was strongly built to prevent access to such confidential information. The passwords could have also been kept in an encrypted format to prevent such easy access.

Form 10: A8- Cross Site Request Forgery- Add to your blog

Vulnerability and Method

The last form is from the Cross Site Request Forgery. We chose Add-to-your-blog vulnerability.
This attack is somewhat similar to Cross Site Scripting attack where the attacker gets a hold of any current user's credentials who is logged on. Similarly, a logged on user's information is leaked and redirected to the attacker in this form. But here, we do it using **HTTP** requests. The steps for this exploitation are as follows.
The attacker **logs in** initially and **adds** an unsafe script to his blog. When some user tries to view the attacker's blog, this unsafe script is **copied** onto the user's blog without his knowledge.

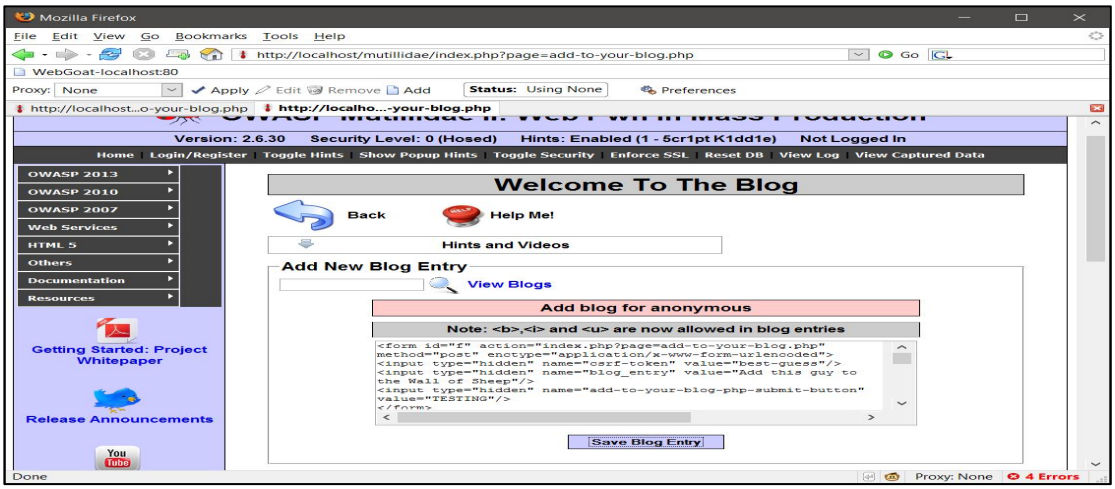

Figure 39: Unsafe script by the attacker

The user will view a blog by selecting the author and the clicking on the **View Blog Entries** button. The user will be able to view the blog after clicking on the button. This process is going to be never ending, which will result in the attacker successfully acquiring the credentials of anyone by inserting the unsafe script into his blog.

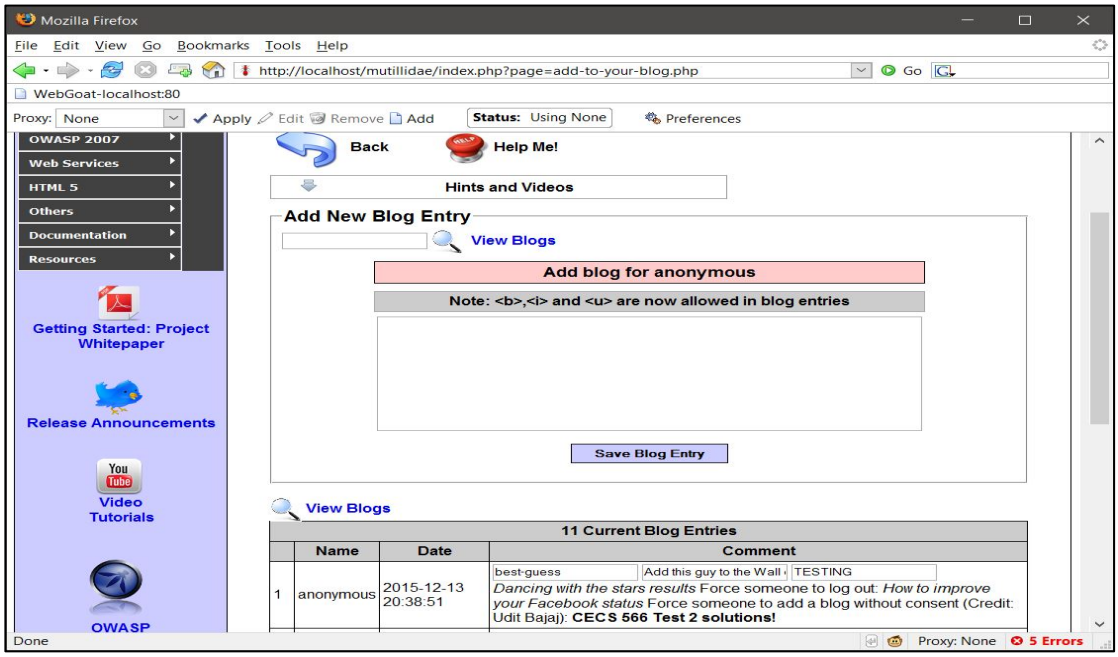

Figure 40: the credentials redirected to the attacker

Counter Measure

This vulnerability has been exploited because an unsafe function *gets()* is used in the source code. A safe function *post()* could replace the same functionality in a safer manner thus mitigating the Cross site request forgery attack.

Reflexion

This assignment deals with various vulnerabilities present in Broken Web Applications. We were supposed to learn it from the OWASP website through penetration testing. It helped us a great deal in gaining knowledge about vulnerabilities and their side effects.

Task 1

We were supposed to read from a vulnerability repository similar to task 1 in our first lab assignment; except that this had to be done on SQL Injection. This vulnerability has to be no greater than 90 days. It was a little difficult for us to choose the topic, because of different reasons, most of them were above 90 days and some of them were not really interesting to work on. After a search of 2 hours, we settled on a topic about online shopping which both of us found interesting. It was quite surprising to see that a small injection could actually help us have lots of stuff for free delivered at our place. It was an easy task and took about 3 hours more for browsing and documenting our point of view.

Task 2

WebGoat certainly gave us a lot to choose from. The vulnerabilities and lessons were fun to play around. We found it an interesting way to actually gain knowledge on penetration testing. It gave us an experience of a real hacker, which seemed very cool, for some time; until we completed the lesson, of course. That encouraged us to see many other vulnerabilities. Overall, we spent more than 10 hours on this task. The documentation took a lot of time as we were not exactly sure about the screen shots to include in our report. Everything felt important at a point of time, that we were considering about insertion of videos on how we performed the task.

Task 3

The lengthiest task of all; we again had to choose what to do from the long list that the website provided. This application had a lot to teach us, which resulted in us learning a lot. Considering it as a practice for our viva, we even tried some exercises more than twice; just to be sure we were getting it right. We spent about 20 hours on this task.

The assignment felt easy to complete; that might be due to the fact that we were starting to enjoy this subject much more than we expected. Looking back, we can confidently say that the assignments had actually taught us so much about Security than we could have learnt if we read a textbook.

References

1. Open Source Vulnerability Database(OSVDB) Website: http://osvdb.org/
2. MyBB forum software website: http://www.mybb.com/
3. Counter measures for SQL Injection: http://msdn.microsoft.com/en-us/magazine/cc163917.aspx
4. Open Web Application Security Project (OWASP) Website: https://www.owasp.org/
5. Open Web Application Security Project Broken Web Application (OWASPBWA) Website: https://www.owasp.org/index.php/OWASP_Broken_Web_Applications_Project
6. WebGoat Home page in OSWAP website: https://www.owasp.org/index.php/Category:OWASP_WebGoat_Project
7. Source Firefox website for Hacker Firefox: http://sourceforge.net/projects/hackfox/
8. WebGoat video tutorial website: http://webappsecmovies.sourceforge.net/webgoat/
9. Counter measures against buffer overflow: http://www.emc.com/emc-plus/rsa-labs/historical/countermeasures-against-buffer-overflow-attacks.html
10. Mutillidae home page from OWASP website: https://www.owasp.org/index.php/Category:OWASP_Mutillidae