Web Application Security
XSS and CSRF injections and mitigations

Ananth Jillepalli & Risab Manandhar

July 17, 2023
Version 2.4

University of Idaho

CS 539: Applied Security Concepts

Summary

Cross-Site Scripting(XSS) is a script technique to send malicious code in the form of browser side scripts. XSS occurs when input to a web-page from the input fields are not handled properly, causing the web-page to execute a malicious script.

Cross-Site Request Forgery (CSRF) on the other hand, is an attack that forces an authenticated end user to execute unwanted actions on a web application. In this tutorial, we will be learning how to exploit the vulnerabilities in web applications using XSS and CSRF as the best ethical hackers do, and then work on the mitigation techniques to mitigate such attacks.

Contents

1 Objectives of this Tutorial >

The goal of this tutorial is to secure web applications in general. Specifically, we focus on:

1. Understand what is a XSS-vulnerability and how it is exploited by attackers.
2. Gain some insight on steps to mitigate XSS attacks.
3. Understand what is a CSRF vulnerability and how it is exploited by attackers.
4. Gain some insight on steps to mitigate CSRF attacks.

This tutorial is not a complete user's guide to Web Application Security. In contrast, this tutorial focuses just on the XSS and CSRF attacks and their respective mitigations. In the ensuing tutorial, we will briefly explain, through either activities or challenges, the following specific things about XSS and CSRF attacks:

1. Understand what causes a Cross-Site Scripting (XSS) vulnerability and observe how the vulnerability is turned into an exploit by attackers.
2. Gain some insight on measures/steps to mitigate the observed XSS vulnerability exploits. However, this tutorial does not claim to make the reader an expert in XSS vulnerability mitigation. This tutorial is intended to be a beginner's guide to enhancing web application security.
3. Understand what causes a Cross Site Request Forgery (CSRF) vulnerability and observe how the vulnerability is turned into an exploit by attackers.
4. Gain some insight on measures/steps to mitigate the observed CSRF vulnerability exploits. Same as before, this tutorial does not claim to make the reader an expert in CSRF vulnerability mitigation, but instead, this tutorial will provide reader with a decent understanding on how to avoid common pitfalls that plague many software developers internationally.

2 Required Background <>

We assume that the reader of this tutorial has an extent of background knowledge in the following areas:

1. Working experience on usage of computers and software applications, like web browsers, and virtualization apps.
2. Basic overall idea of computer networks and Internet.
3. Fundamental knowledge on topics like networking mechanisms, web application functioning, etc.,
4. An overall idea on general issues like website security, data privacy, computer/network security, etc.,

Due to restrictions on time and manpower resources, we are not able to make the ensuing tutorial to be completely self-contained from the perspective of a user. As such, the tutorial is best used when the user already has certain background skills and knowledge. The following are some areas where we expect the users of this tutorial to have some previous skills/knowledge:

1. Practical experience on using computers, installing and using common software applications (particularly web browsers, and virtualization software platforms). The tutorial does not explain how to navigate within the operating system's graphical user interface (GUI). Similarly, the tutorial also does not explain how to browse the Internet, how to install software applications, and how to use/navigate software applications.
2. An overall idea on the working os computer networks and Internet. The tutorial expects a user to understand common computer networking terms and their technical meanings. For example, "websites", "scripting", "reference forgery", and etc.,
3. Fundamental or very basic idea on networking mechanisms and web application functioning. The tutorial expects a user to understand technical concepts like OSI model of networks, setting up a web application, navigating a web application database, and querying the database for information, and knowledge of basic network attacks like man-in-the-middle attack etc., are helpful.
4. Also bit of exposure to logic notations and elementary programming skills would be very helpful in understanding the PHP code involved in most of the activities and chal-

lenges. An overall brief idea on general computer-related issues like “website security”, “data privacy”, “computer security”, “network security”, “application permissions”, “network access privileges” and etc., will help a lot.

3 Hardware and Software Requirements <>

We recommend having at least the following hardware and software specifications for smooth execution of this tutorial's activities and challenges:

1. A computer which can at least boot 1 virtual machines (VM) smoothly, with no noticeable lag and delay
2. A functional virtualization software platform. For example: VMWare, VirtualBox, or UTM.
3. One vanilla Ubuntu VM (preferably 20.04 LTS) and as an addition, LAMP stack, and `bWAPP` web application(*****).

Since the tutorial deals with a lot of buggy code and code manipulation, it is highly recommended to NOT perform the activities and challenges of this tutorial on your own operating system. Please use a virtualization software, and build a virtual machine which can be used for the experiments of the tutorial.

(*****) Since this tutorial deals with web application security, in order to understand common web application vulnerabilities and exploits, we require a buggy web app. Fortunately, we found just what we needed in the form of `bWAPP`, a deliberately and insecurely developed web app, which has 100 bugs/vulnerabilities in it.

Ubuntu VMs setup advise: Though this tutorial can be executed with any Ubuntu operating system compatible with `bWAPP`, we have tested the code and functionality on Ubuntu 20.04 LTS only and cannot guarantee if the tutorial's code works on other versions.

Once the vanilla installation of Ubuntu 20.04 LTS finishes, the user of this tutorial is strongly encouraged to install `bWAPP` right away. LAMP is a pre-requisite for `bWAPP` to function, so that is required to be installed beforehand.

Note: Please go to part **2** of the **Appendix** section for a guide on how to install LAMP stack on Ubuntu 20.04 and also for resources and website links to access some of the required software packages for the tutorial.

4 Problem Statement <>

1. XSS attack can be performed on any web application, where injection of any malicious script can lead to either: access of cookies, or session tokens, or other sensitive information, which is retained by the web browser.
2. CSRF attacks can be performed with a little help of social engineering (such as sending a malicious web-link via email or chat), specifically targeting state-changing requests, not theft of data, where an attacker may trick the users of a web application into executing actions of the attacker's choosing.

1. XSS attack can be performed in any real-world web application, where the end users' web browser has no means to identify that the script should not be trusted, and thus, script will be executed. This causes the malicious script to access cookies, session tokens, or other sensitive information retained by the web browser. Once this information is made accessible to the attacker, it is not long before the attacker is able to exploit user's identity and data. [1].
2. Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application, in which the end user is currently authenticated. CSRF attacks specifically target state-changing requests, and usually do not involve theft of data, since the attacker has no way to see the response to the forged request.

   With a little help of social engineering (such as sending a malicious link via email or chat), an attacker may trick the users of a web application into executing actions of the attacker's choosing. If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email address, and so forth. If the victim is an administrative account, CSRF can compromise the entire web application. [2].

5 Outline of the Tutorial <>

1. Proposed solutions to the problems discussed in the earlier slide.
2. Background information regarding XSS and CSRF.
3. Activities involving observation of attack execution and mitigation.
4. Challenges enhancing understanding of attacks, followed by mitigations.
5. Conclusion and Appendix.

6 Proposed Solutions <>

1. Escaping special symbols before inserting untrusted data into HTML element content (to counter XSS).
2. Use HTTPOnly cookie flag (to counter XSS).
3. Validate user-authenticated response multiple times, with either: a CAPTCHA, or use re-authentication, or using a one-time valid token (to counter CSRF)

1. PHP uses **htmlspecialchars** function so as to escape or translate special characters. The translations performed are [7]:
   - '&' (ampersand) becomes '&amp;'
   - ' " ' (double quote) becomes '&quot;' when ENT_NOQUOTES is not set.
   - " ' " (single quote) becomes '&#039;' (or &apos;) when ENT_QUOTES is set.
   - '<' (less than) becomes '&lt;' —— '>' (greater than) becomes '&gt;'
2. If the HttpOnly flag (optional) is included in the HTTP response header, the cookie cannot be accessed through client side script (again if the browser supports this flag). As a result, even if a cross-site scripting (XSS) flaw exists, and a user accidentally accesses a link that exploits this flaw, the browser (primarily Internet Explorer) will not reveal the cookie to a third party.

   If a browser does not support HttpOnly and a website attempts to set an HttpOnly cookie, the HttpOnly flag will be ignored by the browser, thus creating a traditional, script accessible cookie. As a result, the cookie (typically your session cookie) becomes vulnerable to theft of modification by malicious script

   OWASP Provides list of XSS (Cross Site Scripting) Prevention Cheat Sheet that can be found in following url:
   `https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html`

7 Related News: Recent Exploits <>

1. eBay XSS bug left users vulnerable to an (almost) undetectable phishing attack (13 JAN 2016) [4]
2. XSS bug in Yahoo Mail could have let attackers take over email accounts(21 JAN 2016) [5]
3. Using a new CSRF Vulnerability, attacker can login to LinkedIn through victim's credentials remotely(24 FEB 2016) [6]

1. The eBay sign-in page includes a URL parameter in its address and the contents of that parameter are written in to the page before it's shown to the user. Unfortunately the page didn't check what was in the URL parameter before including it in the page so a researcher by the name of "MLT" was able to use it and include his own code alongside eBay's [4].

2. A Finnish researcher, Jouko Pynnönen, of the security firm "Klikki Oy", discovered last month, a Cross-Site Scripting (XSS) vulnerability in Yahoo's web-mail that would have allowed attackers to fully compromise email accounts just by sending a malicious email.

   To have their account taken over, a victim would have only needed to open and view the email. Pynnönen also sent himself another rigged email with a hidden script that covertly sent the receiver's in-box data to an external website.Because the malicious code is in the message's body, the code is executed every time a user opens an email [5].

3. According to Rohit Dua, a security researcher based in India, the issue existed in a portal on LinkedIn's Help Center site. To exploit the issue, a user would've had to sign into LinkedIn, gone to the site's Help forum and started a discussion. By entering in a few lines of code, Dua claims an attacker could've executed script [6].

8 Background: XSS <>

Cross-Site Scripting (XSS) attacks occur when:

1. Data enters a Web application through an untrusted source, most frequently a web request.
2. The data is included in dynamic content that is sent to a web user without being validated for malicious content.

The severity of these attacks range from a minor annoyance to complete account compromise and/or sabotage.

1. The malicious content sent to the web browser often takes the form of a segment of JavaScript, but other forms may also include HTML, Flash, or any other type of code that the browser may execute. The variety of attacks based on XSS is almost limitless, but they commonly include transmitting private data, like cookies, or other session information, to the attacker; redirecting the victim to web content controlled by the attacker, or performing other malicious operations on the user's machine using the web site as a scapegoat all the while [1].

2. The most severe XSS attacks involve disclosure of the users' session cookie, allowing an attacker to hijack the users' session and take over the account. Other damaging attacks include the disclosure of end user files, installation of Trojan horse programs, redirecting the users to some other page or site, or modify the way a content is presented [1].

9 Background: CSRF <>

Cross-Site Request Forgery (CSRF) attacks occur when:

1. The user is currently authenticated to the site and is tricked to submit a malicious request.

2. Web application do not use preventive measures that can easily implemented.

Such attacks silently submits a “post” form without letting user know about it, using victim’s authenticated session information.

1. CSRF is an attack that tricks the victim into submitting a malicious request. It inherits the identity and privileges of the victim to perform an undesired function on the victim’s behalf. Web browser requests automatically include any credentials associated with the site, such as the user’s session cookie, IP address, Windows domain credentials (very uncommon), and so forth. Therefore, if the user is currently authenticated to the website, the site will have no way to distinguish between the forged requests sent by the attacker in the guise of victim and a legitimate request sent by the victim.

2. CSRF attacks target functionality that causes a state change on the server, such as changing the victim’s email address or password, or purchasing something. Forcing the victim to retrieve data doesn’t benefit an attacker because the attacker doesn’t receive the response, the victim does. As such, CSRF attacks target state-changing requests.

   It’s sometimes possible to store the CSRF attack on the vulnerable website itself. Such vulnerabilities are called “stored CSRF flaws”. This can be accomplished by simply storing an IMG or IFRAME tag in a field that accepts HTML, or by a more complex cross-site scripting attack. If the attack can store a CSRF attack in the site, the severity of the attack is amplified. In particular, the likelihood of attack execution is increased because the victim is more likely to view the page containing the attack than some random page on the Internet. The likelihood is also increased because the victim is sure to be authenticated to the website already [2]

10 Mitigating XSS Attacks <>

Using escaping technique:
Consider a form that takes in 2 values such that ***$firstname* = *first name*** and ***$lastname* = *last name***

For proper sanitization of the input values obtained from the input field, the interpretation can be done as:
$data = htmlspecialchars($data, ENT_QUOTES, "UTF-8")

Escaping technique is a mechanism of code sanitizaton wherein, a code request from users will be sanitized (or escaped); which means they are not copied as they are for interpretation, but they are transformed into something relevant, but not dangerous.

For example, PHP uses **htmlspecialchars** function so as to escape or translate special characters. The translations performed are [7]:

- '&' (ampersand) becomes '&amp;'
- ' " ' (double quote) becomes '&quot;' when ENT_NOQUOTES is not set.
- " ' " (single quote) becomes '&#039;' (or &apos;) only when ENT_QUOTES is set.
- '<' (less than) becomes '&lt;'
- '>' (greater than) becomes '&gt;'

11 Questions: XSS <>

Q1. What is the most basic indication that a cross-site scripting attack might be possible on a web site?

Q2. What is one example of a language that can be leveraged in an XSS attack?

Q3. If the languages can be leveraged in an XSS attacks, what shall we do then? Never use such languages?

Answer to Q1: The existence of a user-input form indicates the possibility of script injection if the user input is echoed back on a web page.

Answer to Q2: Some common scripting languages used in XSS attacks are JavaScript and ActiveX.

Answer to Q3: Use mitigation techniques that we discussed earlier.

12 Activity I: XSS Attack <>

1. Go to `http://www.bwapp.com/login.php`
2. Login: bee Password: bug
3. From "Choose your bug:" drop-down list, select "Cross-Site Scripting - Reflected (GET)"
4. Set security level to **low** and click **Hack**
5. Provide different combinations of first name and last name and look for the potential vulnerability.

NOTE: From this point onwards, the activity part begins. To see the VM setup directions, please visit **Slide 3**

This activity demonstrates an XSS-attack in its' most simple form; which is to inject a script into the input field, which will then be used to exploit the XSS-vulnerability. The following combinations of input data can be used to see the difference between a normal login and a an attack attempt at injecting malicious script.

Combination 1
First Name: Joe
Last Name: Vandal
Output Result: "Welcome Joe Vandal"

Combination 2
First Name: Joe
Last Name: Vandal <script>alert('H@CKT'); </script>
Output Result: "Welcome Joe Vandal"; along with an alert (pop-up box) containing the text 'H@CKT'.

13 Challenge I: Modified XSS Attack <>

As a continuation to previous activity, try to inject some input parameters such that

1. Inputs values are returned back to the user itself with the welcome message as follows:
   Welcome Joe Vandal ***Congratulations!!! Click here to see your prizes!!!***
2. When user performs mouse-over action on ***Congratulations!!! Click here to see your prizes!!!***, it displays cookie information for the user as an alert.

Hint 1:
Append the string ***Congratulations!!! Click here to see your prizes!!!*** after the Last Name
Hint 2:
Cookie information can be obtained from **document.cookie**, use **onmouseover** action to **alert** the cookie information

14 Challenge II: Mitigate Challenge I Attack <>

Try sanitizing inputs such that the scripts you executed in the challenge 1 acts just as a normal script

For this challenge, the equivalent php file is located at: $/var/www/html/bWAPP/public\_html/xss\_get.php$

In this challenge, we have to mitigate the attack developed in Challenge 1.

Hint:

1. Consider **case 0** to be the one with low security level
2. Remember the **Mitigation** slide which talks about input sanitization
3. Use **htmlspecialchars** function to sanitize user inputs

15 Activity II: CSRF Attack Part I <>

1. From "Choose your bug:" drop-down list, select "Cross-Site Request Forgery(Change Password)"
2. Set security level to **low** and click **Hack**
3. Change the password from "bug" to something else. You should be able to do it.
4. Right Click on the browser and view the source of the web-page.
5. Copy Web content as shown below

This activity involves demonstration of a common CSRF attack. The goal of this activity is to modify passwords using a cross site request forgery.

```
<h1>CSRF (Change Password)</h1>

<p>Change your password.</p>

<form action="/bWAPP/csrf_1.php" method="GET">

        <p><label for="password_new">New password:</label><br />
        <input type="password" id="password_new" name="password_new"></
            p>

        <p><label for="password_conf">Re-type new password:</label><br
            />
        <input type="password" id="password_conf" name="password_conf"
            ></p>

        <button type="submit" name="action" value="change">Change</
            button>

</form>
```

16 Activity II: CSRF Attack Part II <>

1. Modify the copied content as shown in the notes section below. Note the following changes:
   (a) action field in the form tag
   (b) addition of value attributes in the input tags
2. Save it as html file in desktop
3. Open the html file and press **Change**

Attackers tend to execute such process in the background silently where your password is being compromised secretly.

```
<h1>CSRF (Change Password)</h1>

        <p>Change your password.</p>

        <form action="/bWAPP/csrf_1.php?" method="GET">

                <p><label for="password_new">New password:</label><br
                   />
                <input type="password" id="password_new" name="
                   password_new" value = "bug" ></p>

                <p><label for="password_conf">Re-type new password:</
                   label><br />
                <input type="password" id="password_conf" name="
                   password_conf" value = "bug" ></p>

                <button type="submit" name="action" value="change">
                   Change</button>

        </form>
```

17 Challenge III: Modified CSRF Attack <>

1. From **Choose your bug**, select **Cross-Site Request Forgery(Change Secret)**
2. Set security level to **low** and then press **Hack**
3. As we did in previous tutorial, create a HTML page that can be injected using CSRF, and execute that html page such that it changes the secret statement to something else.

Hint: This challenge involves changing the secret statement of a user, by employing cross site request forgery mechanism. You'll have to create a new HTML page, based on the following code:

```
<h1>CSRF (Change Secret)</h1>

        <p>Change your secret.</p>

        <form action="/bWAPP/csrf_3.php" method="POST">

                <p><label for="secret">New secret:</label><br />
                <input type="text" id="secret" name="secret"></p>

                <input type="hidden" name="login" value="bee">

                <button type="submit" name="action" value="change">
                   Change</button>

        </form>
```

18 Questions: CSRF <>

What was the problem with in the following pages?

1. Page where the password was changed
2. Page where the secret was changed

Can you think about the ways to mitigate the problems in the above pages?

Answer:

1. For the page where we changed the password, there is no re-authentication process in place when the password was changed. It can be changed such that there is some re-authentication process implemented while trying to change the password
2. For the page where we changed the secret, it is annoying if we were to re-authenticate then as well. So we will be using one-time tokens so as to mitigate this attack

19 Mitigating Attack from Activity II <>

We will be adding current password input field to this form

1. Go to the **csrf_1.php** page located in */var/www/html/www.bWAPP.com/public_html/*
   *
2. Search for the location in code, where the new passwords are retrieved and added to the database.
3. Uncomment the code that enables current password checking†.
4. Go the html construct down below and uncomment the component that is supposed to display "Current Password" field†.

To mitigate this attack of exploiting new password allocation, we will be including the current password field which will be asking for the current password and verify if the current password is valid or not(attacker should have the knowledge of current password as well so as to attack in the same way as it was working before)

Thanks to Tyler Cleveland (Fall 2021 CptS 427 at Washington State University) for the following:

- * This location could instead be */var/www/html/bWAPP/bWAPP/*
- †Once the corresponding code lines are identified, one may find that they are already uncommented.
- The Security Level setting needs to be changed to a value that's medium or high (i.e. not **low**) for the mitigation to work.

20 Challenge IV: Mitigate Challenge III Attack<>

Use one-time token implementation such that the system uses tokens that change as per the session and thus, grabbing tokens from the web-application of the target victim doesn't work

- The page associated with this challenge is **csrf_3.php** located in */var/www/html/www.bwapp.com/public_html/* *

Thanks to Tyler Cleveland (Fall 2021 CptS 427 at Washington State University) for the following:

- * This location could instead be */var/www/html/bWAPP/bWAPP/*
- The solution for this challenge has been updated as of July 17, 2023.

21 Conclusions <>

- Both XSS and CSRF are still a major concern for web application security.
- Solutions to the problem are: proper input validation, special character encoding, and proper authentication of the user and session.
- Despite solutions, why are XSS and CSRF still a concern?
    - Lack of expertise implementing the secured web application and security not being a high priority in comparison to product delivery.

22 Appendix: Solutions and Change-log <

1. Solutions to the challenges
   (a) Challenge I
   (b) Challenge II
   (c) Challenge III
   (d) Challenge IV
2. Tutorial-Related Resources
3. Change-Log

1. **Solutions to the Challenges:**

   (a) **Challenge I:**
   First Name: ABC
   Last Name: def <b onmouseover=alert(document.cookie)>Congratulations!!! Click here to see your prizes!!! </b>

   (b) **Challenge II:**
   In xss_get.php file, update case 0 such that
   Original implementation:
   $data = no_check($data);
   Correct Implementation:
   $data = htmlspecialchars($data, ENT_QUOTES, "UTF-8");

   (c) **Challenge III:**

```
<div id="main">
        <h1>CSRF (Change Secret)</h1>
        <p>Change your secret.</p>
        <form action="http://www.bwapp.com/csrf_3.php" method="
            POST">
                <p><label for="secret">New secret:</label><br
                    />
                <input type="text" id="secret" name="secret"
                    value="Your Secret has been changed"></p>
                <input type="hidden" name="login" value="bee">
                <button type="submit" name="action" value="
                    change">Change</button>
        </form>
        </br>
        <font color="green">The secret has been changed!</font>
</div>
```

(d) **Challenge IV:**

Change the security from low to medium or high and repeat the activity. Observe the following changes in the code workflow of the **csrf.php** file:

1. Instead of

```
if(isset($_REQUEST["login"]) && $_REQUEST["login"])
```

we see the following code gets executed:

```
if(isset($_REQUEST["token"]) and isset($_SESSION["token"]) and
    $_REQUEST["token"] == $_SESSION["token"])
```

2. Instead of

```
<input type = "hidden" name="login" value="<?php echo $login;?>
    ">
```

we see the following code gets executed:

```
<input type = "hidden" id="token" name="token" value="<?php
    echo $_SESSION['token']?>">
```

2. **Tutorial-Related Resources:**

   A free virtualization software platform - VirtualBox - Download

   A free virtualization software platform for M1 processors - UTM - Download[1]

   Ubuntu 20.04 LTS - Canonical Ltd. - Download

   Guide to Installing LAMP on Ubuntu 20.04 LTS - DigitalOcean - Read

   Guide to Configuring MySQL on Ubuntu 20.04 LTS - DigitalOcean Read[2]

   `bWAPP` by Malik Mesellem - SourceForge - Download[3]

   Docker image `bWAPP` - DockerHub Image

   Docker compose `bWAPP` - GitHub - Compose

   Live hosting for `bWAPP` - HakHub - live app

[1]Thanks to Funso Oje (Fall 2021 CptS 427 at Washington State University) for testing the tutorial with 20.04 and for the M1 processor guide link.

[2]Use this tutorial to setup a root password for your MySQL installation. You will encounter 'connection failed' error if you don't have a root password setup for MySQL.

[3]If you do not configure a server IP address for Apache, the default address would be 'localhost', which can be accessed through 'http://localhost'.

3. **Change-Log:**

Web Application Security Change-log			
Ver.	**Date**	**Authors**	**Changes**
v1	Apr. 23rd 2016	Risab Manandhar	First draft of tutorial.
v2	July 18th 2016	Ananth Jillepalli	Major content additions and remodeled the structure.
v2.1	July 26th 2016	Ananth Jillepalli	Added the ‘Appendix’ section and other minor enhancements.
v 2.2	July 31st 2017	Ananth Jillepalli	Changed the licensing from CC BY-NC-ND 4.0 to CC BY-NC-SA 4.0
v 2.3	Nov. 05th 2021	Ananth Jillepalli	Updates and solved inconsistencies
v 2.4	Nov. 26th 2022	Ananth Jillepalli	Added docker and live app options for `bWAPP` hosting

References

[1] "Cross-site Scripting (XSS)", last accessed 11th Apr. 2016, `https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)`

[2] "Cross-Site Request Forgery (CSRF)", last accessed 11th Apr. 2016, `https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)`

[3] "bWAPP an extremely buggy web app !", last accessed 10th Apr. 2016, `http://www.itsecgames.com/`

[4] "eBay XSS bug left users vulnerable to (almost) undetectable phishing attacks", last accessed 11th Apr. 2016, `https://nakedsecurity.sophos.com/2016/01/13/ebay-xss-bug-left-users-vulnerable-to-almost-undetectable-phi\shing-attacks/`

[5] Lisa Vaas, "XSS bug in Yahoo Mail could have let attackers take over email accounts", last accessed 11th Apr. 2016, `https://nakedsecurity.sophos.com/2016/01/21/xss-bug-in-yahoo-mail-could-have-let-attackers-take-over-emai\l-accounts/`

[6] Eric Tjossem, "Hacker News CSRF Vulnerability", last accessed 11th Apr. 2016, `http://www.tjosse.me/articles/csrf-on-hn/`

[7] "htmlspecialchars", last accessed 11th Apr. 2016, `http://php.net/manual/en/function.htmlspecialchars.php`

[8] Anup C., Joe F., "Cross-Site Scripting Tutorial", last accessed 11th Apr. 2016