**Omar Dayya – 201611504**

Upload your project proposals , project groups may include two students.

You have a week to submit a proposal (due date 11.05.2020) and you'll have two weeks to deliver ( due date 24. 05.2020).  Any work submitted at any time between 24 hours and up **to** 48 hours **late will** receive a deduction of **20**% of the marks available, and so on, at the rate of an additional 10% of available marks deducted per 24 hours, until the [assignment](https://webonline.cankaya.edu.tr/mod/assign/view.php?id=77945) is submitted or no marks remain.

Some project ideas:

**Research Topics:**

Security of Smart Grid

Security of VANETS

Software Vulnerabilities

Industrial Control System Security

**Potential**[**Projects**](https://webonline.cankaya.edu.tr/mod/assign/view.php?id=84722)**with Demo: (This type of**[**projects**](https://webonline.cankaya.edu.tr/mod/assign/view.php?id=84722)**will be evaluated with 20% bonus)**

Honeypot- Honeynet

Sand-Boxing

Digital Forensics

DDoS

Session Hijacking

Cross Site Scripting

**Possible Layout of proposals:**

**Title of project**

**Aim of the project (at least 5-10 sentences)**

**Group members and responsibilities**

**References ( short literature reviews - at least  15-20 sentences)**

Cross Site Scripting (XSS) Attack

It is type of code injection attack where the attacker sends a code to the browser to execute in order to send the attacker all the credentials and details of a certain user. The attacker tries to steal the client’s session cookie, so the attacker can impersonate you in order to get into the credentials.

Ingredients:

HTMLl tags

Js functions

Js numbers

Js strings

The attacker is going to send a POST/GET request which includes a script, normally Javascript code, which is sent to the web application’s database and then the attacker waits for the script to execute, inside the script normally is a link that is sent to the user where when opened by the user gets executed by the browser and the attacker successfully receives the cookies of that client to empersonate him.

So, there are normally 3 types of XSS attacks:

* Stored xss attacks /persistent
* Reflected xss attack / non-persistent
* DOM based attacks

**Reflective XSS**

For example, the attacker could send the victim a misleading email with a link containing malicious JavaScript. If the victim clicks on the link, the HTTP request get initiated from the victim's browser and sent to the vulnerable web application. The malicious JavaScript is then reflected back to the victim's browser, where it is executed in the context of the victim user's session.

**Stored / Persistent XSS**

Lets say we have a web application that allows users to enter a username that is displayed on each user’s profile page. The application stores each username in a local database. A malicious user notices that the web application fails to sanitize the username field and inputs malicious JavaScript code as part of their username. When other users view the attacker’s profile page, the malicious code automatically executes in the context of their session.

**DOM Based XSS (Document Object Model)**

DOM-based XSS vulnerabilities usually arise when JavaScript takes data from an attacker-controllable source, such as the URL, and passes it to a sink that supports dynamic code execution, such as eval() or innerHTML. This enables attackers to execute malicious JavaScript, which typically allows them to hijack other users' accounts.

* It's not even safe in HTML! strip\_tags() is not enough to protect values in attributes, e.g., <input value="$foo"> might be exploited with $foo = " onfocus="evil() (no <,> needed!)

So the correct solution is to escape data according to requirements of language you're generating. When you have plain text and you're generating HTML, you should convert text to HTML with htmlspecialchars() or such. When you're generating e-mail, you should convert text to quoted-printable format, and so on.

Defend on depth

* Escape user input. Escaping means to convert the key characters in the data that a web page receives to prevent the data from being interpreted in any malicious way. It doesn't allow the special characters to be rendered.
* Validate user input. Treat anything that originates data from outside the system as untrusted. Validate all the input data. Use a whitelist of known, acceptable, good input.
* Sanitize data. Examine and remove unwanted data, such as HTML tags that are deemed to be unsafe. Keep the safe data and remove any unsafe characters from the data.

So first thing, show this 🡪 ?username=omar97

**Warning**: file\_put\_contents(log.txt): failed to open stream: Permission denied in **/opt/lampp/htdocs/xss/attack/ok.php** on line **5**

Access denied by folder 🡪 drwxr-xr-x 2 bitnami bitnami 4096 May 18 18:27 attack

ls -l

sudo chmod -R 777 \*target folder\*

Thus, session cookies are sensitive information which, if compromised, may allow an attacker to impersonate the legitimate user and gain access to his existing web session. This attack is called **session hijacking**.

<b onmouseover=alert(‘XSS testing!‘)></b>

<script>destroyWebsite();</script>

$c = $\_POST['comment\_area']; //TEXT AREA WITHOUT STRIP\_TAGS

$comment = e($c);

// $comment = strip\_tags($\_POST[comment\_area]);

**Cross-Site Scripting (also known as XSS)** is one of the most common application-layer web attacks. XSS vulnerabilities target scripts embedded in a page that are executed on the client-side (in the user’s web browser) rather than on the server-side. XSS in itself is a threat that is brought about by the internet security weaknesses of client-side scripting languages, such as HTML and JavaScript.

Key Concepts of XSS

* XSS is a web-based attack performed on vulnerable web applications.
* In XSS attacks, the victim is the user and not the application.
* In XSS attacks, malicious content is delivered to users using JavaScript.