MIECT: Security 2017-18

Practical Exercise: XSS: Cross-Site Scripting

October 4, 2017 Due date: no date

Changelog

- v1.0 - Initial Version.

Introduction

Cross-Site Scripting (XSS) attacks are a kind of attacks within Web interactions where an attacker performs indirect attacks against Web clients through a vulnerable Web application. The primary result is that some external code is injected into the victim web browser is executed. All existing context, including valid cookies, as well as computational resources of the victim become available to the attacker.

The attack can be conducted based on data stored in the server, such as a forum message or a blog post, and this is named a Stored XSS Attack.

The attack data can also be encoded in a `URL` sent by the attacker directly to the victim. Taking in consideration where the untrusted data is used, the attack can be considered a Server Side Attack, or a Client Side Attack. And all four combinations are possible.

The problem itself is always due to improper, or insufficient validation of data external to the system.

1 Environment setup

For this project you should use the virtual machine provided for these classes. **We will be using software that is purposely vulnerable. Do not use your own laptop!**.

Please obtain the compressed file present at `http://atnog.av.it.pt/~jpbarraca/classes/security/xss.tar.bz2` and uncompress it. You will find further instructions in the `README.MD` file that is present in this file.

The application contains one user, identified by the username `Administrator` and password `top-secret`.

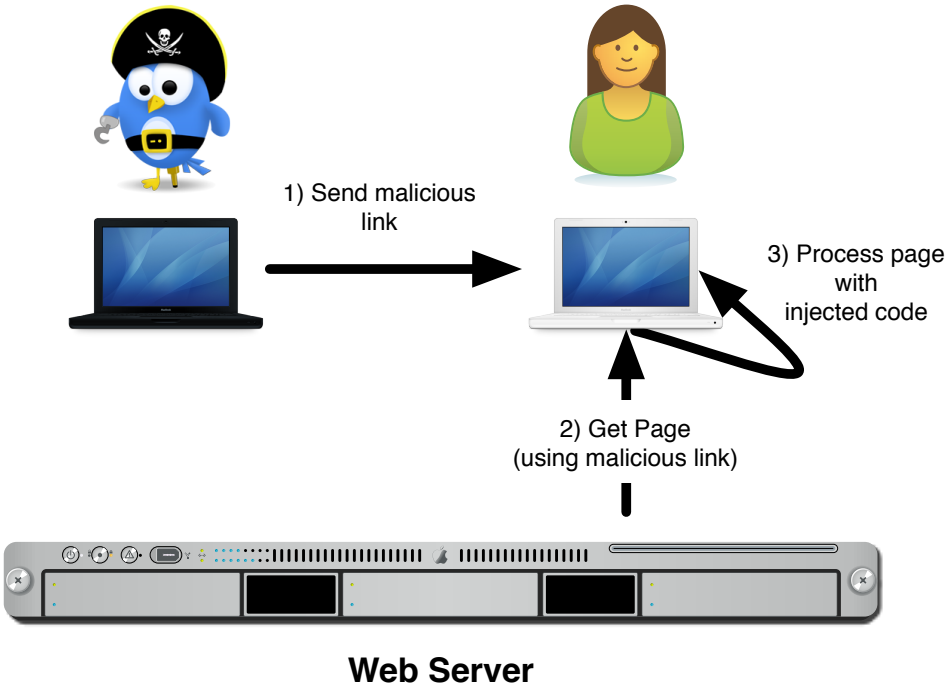

Figure 1: Reflected XSS Attack

2 Cross-Site Scripting

2.1 Reflected XSS Attack

A Reflected XSS attack is similar to the previous, but it is assumed that the attack is non-persistent. With this attack it becomes possible to manipulate the browser DOM for a single user, or for multiple users which access a page through the same specially crafted URL. Figure 1 depicts a typical attack scenario.

The application we are using is vulnerable to this attack. Can you find it? Search for an action that changes the URL. That is, an action that will redirect to the same page but with added variables and content in the URL. If the page behaves differently based on the URL variables, it is possible that a Reflected XSS Attack is present.

Can you identify where is the vulnerable code (Client vs Server)?

Can you fix it?

2.2 Stored XSS Attack

The Stored XSS Attack (or persistent) allows an attacker to place a malicious script (usually Javascript) into a webpage. Victims accessing the webpage will render all scripts, including the one injected by the attacker. This attack is very common in place where information is shared between users through web technologies (e.g., forums and blogs). In this case, an attacker must compose an specially crafted message, and hides some script in its source code and put it in some place which is accessed by a victim. All users accessing that place would execute the exploit. Figure 2 depicts this

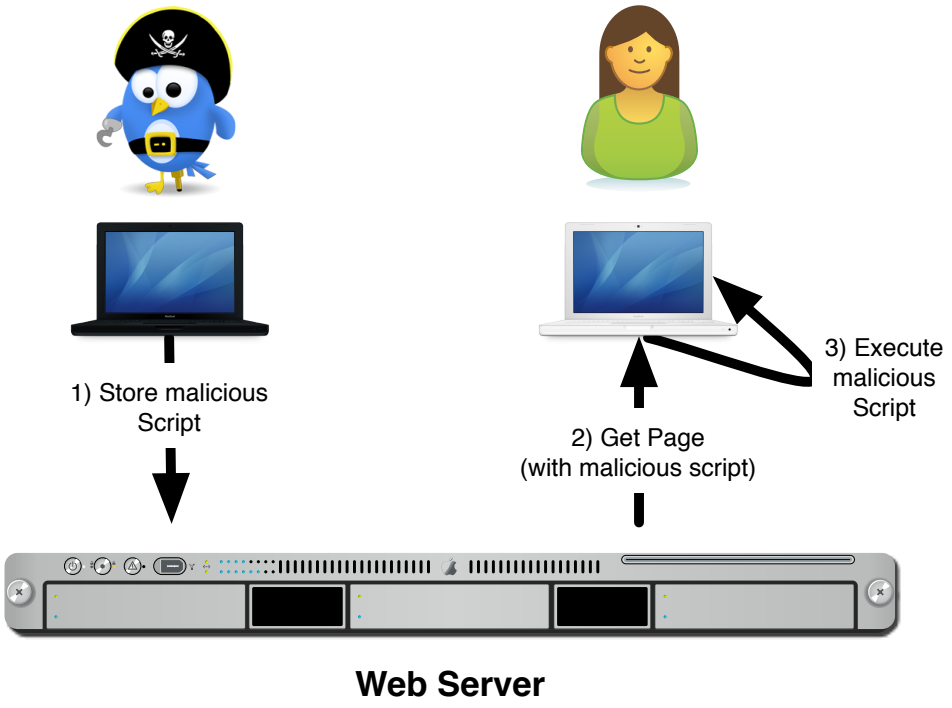

Figure 2: Stored XSS Attack

attack.

The application we are using is vulnerable to Stored XSS Attacks, and there are vulnerabilities both in the server and client sides. Can you find the vulnerabilities?

For the Server Side Stored XSS Attack, look for an action that stores a message into the server. For it to be a Server Side Attack, the payload must be included in the web page when the page is built by the server.

For the Client Side Stored XSS Attack, look for code that loads dynamic content into the webpage using Javascript. Use the Web Inspector built in the browser and see if you can find it. Can you trigger a successful attack? Take in consideration that sometimes, `<script>` tags are not evaluated directly, but Javascript can be included in objects event handlers (e.g., `onload`, `onclick`...)

2.3 CSRF Attack

The Cross-Site Request Forgery (CSRF) attack consists in injecting code that, using the credentials and capabilities of the browser viewing a given object, may attack another system. This attack can be used for simple DoS, tracking users, or invoke requests on systems with the identity of the victim. Figure 3 depicts a typical attack scenario.

This exploits the fact that, for usability, functionality and performance purposes, system cache authentication credentials in small tokens named cookies. When a user accesses a service, such as a social sharing application, or a Online Banking solution, a session is initialized, and will be kept valid for a long period. Even if the user abandons the webpage. However, if the user visits another page which has a CSRF exploit targeting the first page, it is possible to invoke services using the user identity, without his knowledge. This attack is frequently done using the `<img>` tag, however, other tags can be used.

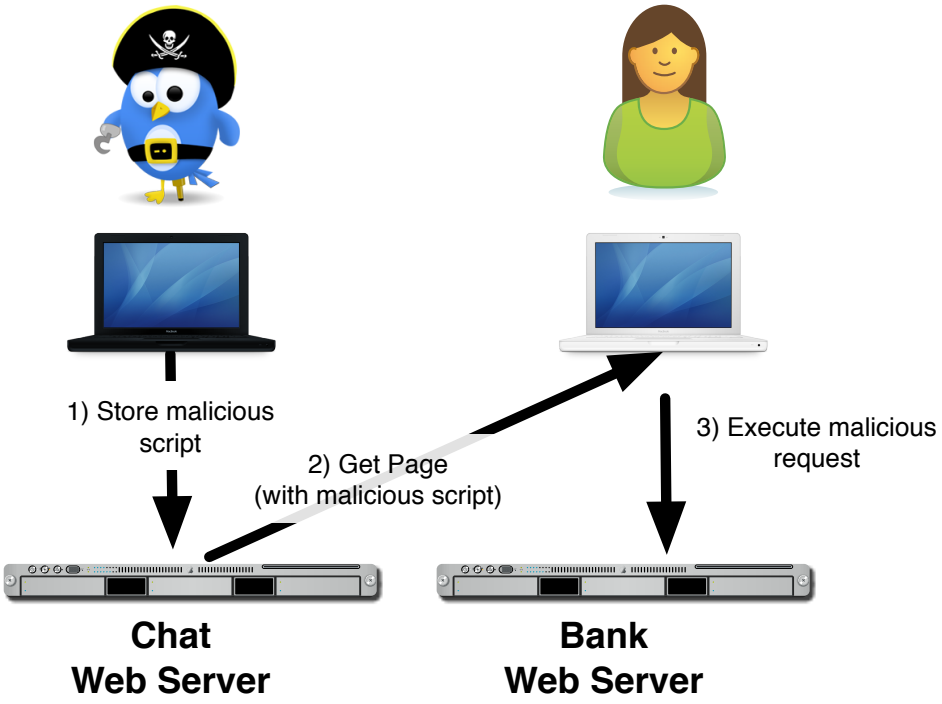

Figure 3: CSRF Attack

As an example, consider that a forum post contains the following content:

```
LOL. That was a good one Op. :)
<img src='http://vulnerable-bank.com/transfer.jsp?amount=1000
&to_nib=12345300033233'></img>
```

When the browser tries to load the image, it will invoke an action to an external server. In this hypothetical case, it would transfer funds from the victims bank account to the attacker's bank account.

Sometimes a more complex interaction is required, and the attack will actually inject Javascript code. Can you build a working attack?

In the scripts directory of the package you downloaded, there is script named `hacker_server.py`, which will dump to `stdout` all data that is posted to it (using `HTTP POST`). Run the script directly and do a `POST` to `http://localhost:8000`.

Can you send the cookie to that server?

2.4 Cross Origin Resource Sharing

Consider that `CodeUA` has an API, through which users are able to execute actions without the web page interface. There is a public mock project name `security-vulnerable-test` which can be used for this purpose.

You can create a wiki page by issuing the following action:

```
PUT /projects/security-vulnerable-test/wiki/attack.xml
<?xml version="1.0"?>
<wiki_page>
  <text>I am vulnerable</text>
  <comments>comments</comments>
</wiki_page>
```

The same page can be deleted by issuing:

```
DELETE /projects/security-vulnerable-test/wiki/attack.xml
```

The important thing is that these actions will be executed using the victim session. This is a reason why it is important to invalidate sessions after a short time, or to logout from web sites.

Can you create a successful attack? Check the browser console for the message that is presented. In particular, check for errors.

In order to analyze the result, use `wget` and get the `CodeUA` webpage and print the headers sent by the server: `wget -S --spider https://code.ua.pt`. In this particular case, the server is not allowing cross origin requests as this must be explicitly authorized when using modern browsers.

If a similar request is made to the `hacker_server.py`, using `wget -S --spider --post-data="foo" http://localhost:8000`, the header `Access-Control-Allow-Origin: *` is present. This specific header allows browsers to reach this page from any origin, allowing the reception of data through `CSRF`. If the header is omitted, current browsers will forbid the request.

For more information regarding Cross Origin Resource Sharing, check the Mozilla documentation at `https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS`.

Further Reading

1. `https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet`
2. `http://html5sec.org`