PSI-CAD-Project Report SoSe 2019

Alexander Böhner (1937944)
mailto:alexander.boehner@stud.uni-bamberg.de

Melanie Vogel (1738258)
mailto:melanie-margot.vogel@stud.uni-bamberg.de

September 30, 2019

Contents

List of Figures

1 Instructor Guides

1.1 Instructors Guide: MessageCenter

This scenario concerns types of Cross-Site Scripting (XSS) attacks and provides two tasks to perform such attacks. The first scenario implements a persistent XSS attack where the second implements a reflective XSS attack.

Attackers inject JavaScript code into a website via input fields or URL parameters and the website sends this malicious code to the browser of the victim. The script can do the same things and access the same data that the original scripts of the website can do because it comes from the same origin. XSS refers to all attacks where JS/HTML is injected into a site. Malicious XSS scripts can–read cookies, read all information on the visited site, change what the victim sees or interact with the site like the victim would.

1.1.1 Related Work

As a first source the Open Web Application Security Project (OWASP) provides an overview on the different types of Cross-Site Scripting[1]. The organization also provides a cheat sheet on how to protect from XSS attacks [2].

Further the concept of same-origin policy [3] and in this context also CORS[4] should be explained to gain an understanding of the attack. Since XSS in in the top 10 of OWASP most critical security risks[5] for years, many online sources on how to perform such attacks exist [6]. [Gro+07] describes very detailed a variety of XSS attacks, especially from a very technical point of view and provide also an overview on frameworks to perform such attacks. Others, such as [BV08] describe the procedure of XSS attacks as well as the automated procedure to perform such attacks.

In general, a lot of literature on XSS attacks is about how to prevent oneself from such attacks. For example, tools were developed and can be used to protect oneself against such attacks[Kir+06][BV08]. Other tools such as Pixy provide support for detecting vulnerabilites in web applications [JKK06].

[1] https://www.owasp.org/index.php/Types_of_Cross-Site_Scripting
[2] HTTPs://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.md
[3] https://security.stackexchange.com/questions/8264/why-is-the-same-origin-policy-so-important
[4] https://www.codecademy.com/articles/what-is-cors
[5] https://www.cloudflare.com/learning/security/threats/owasp-top-10/
[6] https://brutelogic.com.br/blog/

1.1.2 Task: AdminDiss

You are really upset about MATESHOP's reckless security behaviour and especially the expensive prices on the website for Mate and Chunk beverages - the websites admin blamed your low Mate consumption! Be persistent! Insult the administrator as a *dumb donkey* in an alarming way.

The goal of the task is that every time the admin calls the forum's site he or she will be insulted. When the attack was successful the hacker receives a flag that can be pasted into the corresponding field on the insekta page.

1.1.2.1 Special Notes

- The admin implemented the sanitization filter for scripts by himself. Traditional <script> tags are escaped but not images or svgs for example where JS code can be hidden.

1.1.2.2 Guide

The task can be guided as follows including additional hints to achieve the goal and understand the basics of the attack:

1. **What does persistent XSS actually mean?**

   The first thing to do for this task is to understand that a persistent XSS is required. This means the injected code shall be not only temporary but shall be persistent on the back-end server and thus be delivered to future requests.

   Learning objective: Understand persistent XSS attacks.

2. **Where within the webshop is newly added data stored for more than one request?**

   Discover where on the webshop the input is stored and is viewed by the admin from time to time. The forum should be discovered since the admin visits the forum from time to time (every 5 seconds).

3. **How can malicious code be injected? Follow up: How can the website admin block such contents?**

   Try script-tags and find out they are escaped because they aren't visible.

   Learning objective: Escaping of specific potentially malicious code snippets via black or whitelist.

4. **How else can JS code be injected?**

   Discover different ways to include a script into a HTML Element other than with the help of script-tags. Try to post an images or svgs

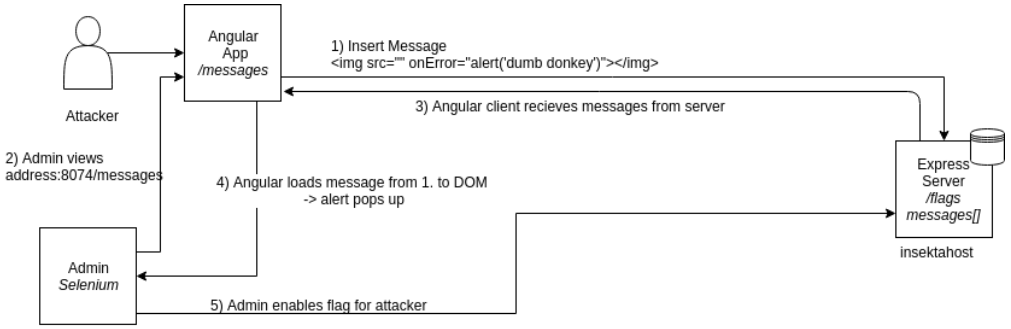

Figure 1.1: Illustration of the persistent XSS attack.

Learning objective: Predefined javascript functions for actions can be used to insert and execute code.

1.1.3 Task: BugReport

After insulting the admin it is time to expose the admin even further. The admin is eager to fix the security issue from the insult and will be happy about every kind of help he or she can get. Find out what kind of private stuff the admin hides from Google by stealing his or her identity! Perform a reflective XSS attack.

The goal is to steal the cookie value of the admin and log in as the admin. After that the flag is hidden in a file called private.php.

1.1.3.1 Special Notes

- Students must set up a web server by themselves to redirect the cookie value.
- The injected link must be clickable i.e. href and a tags must be used.

1.1.3.2 Guide

The task can be guided as follows including additional hints to achieve the goal and understand the basics of the attack:

1. **What does reflective XSS mean?**

   In contrast to persistent XSS, reflected XSS is not persistent on the server. When e.g. a link with malicious code is clicked, the server may generate HTML with the malicious script in it and deliver it to the link-clicker.
   Learning objective: Understand reflectivee XSS and how data can be delivered to the attacker and the victim as well.

2. **What does identity mean in this context? How does the webshop technically know its the admin who is logged in?**

   Find out what *identity* means in this context. For example, each logged-in user will receive a session cookie to be identified during the visit in the webshop. Cookies are basically text files that are locally stored on your machine and can be read by the browser in order to maintain a state in a stateless HTTP world. For instance, when you throw articles in your shopping basket in a webshop or when you authenticated yourself, such information may be associated with a cookie. With cookies you therefore do not have to do these kind of actions with each request.

   Learning objective: Understand sessions and cookies and their purpose.

3. **How is data transferred from the client to the server?**

   First, find out how to help the admin. How is the admin being informed about security issues? Watch out for a reporting mechanism (bug report). Then, think about how client and server communicate via HTTP requests. Recognize that only the short description field will be transferred to the server. Therefore the network communication must be observed via the developer tools. The client transfers only the short description field via HTTP POST requests to the server where the data in form of a JSON object is stored for further processing i.e. in this case the server sends the malicious link to the admin via email and the admin clicks the link. Imagine the admin gets an e-mail with the link to a helpful page in the header. Provide a clicklable URL (href tag necessary)!

   Learning objective: Understand how the communication between client and server works. Learn to use the developer tools in your browser, especially the observation of the network traffic.

4. **What is the structure of the malicious link?**

   Basically, one shall inject some malicious javascript code as a query string into the URL that the admin should click on. An example link could look like this: `<a href="private.php/?q=<script>..</script>"`. The *?* indicates the beginning of the query, *q* is the name or key and everything after the = is the payload of the query in form of concatenated parameters each separated with the *&* sign. The admin has a private file that he or she wants to hide from search engines such as Google i.e. wants to hide it from a web-crawler (robots.txt). The robots.txt contains useful information: It refers to the query parameter with the name *q* and a file called private.php. Visiting this page results in an 401 HTTP error which means you are not authorized to view the content of this page, therefore one has to have the cookie value of the admin to have permission to read it. HTTP status or error codes are basically standardized status responses that you get from a webserver and based on the interpretation of these codes you can derive what happened during the processing of a request. From a programmatical point of view, these status codes can be used in conditional checks to provide useful information to the users or exception handling.

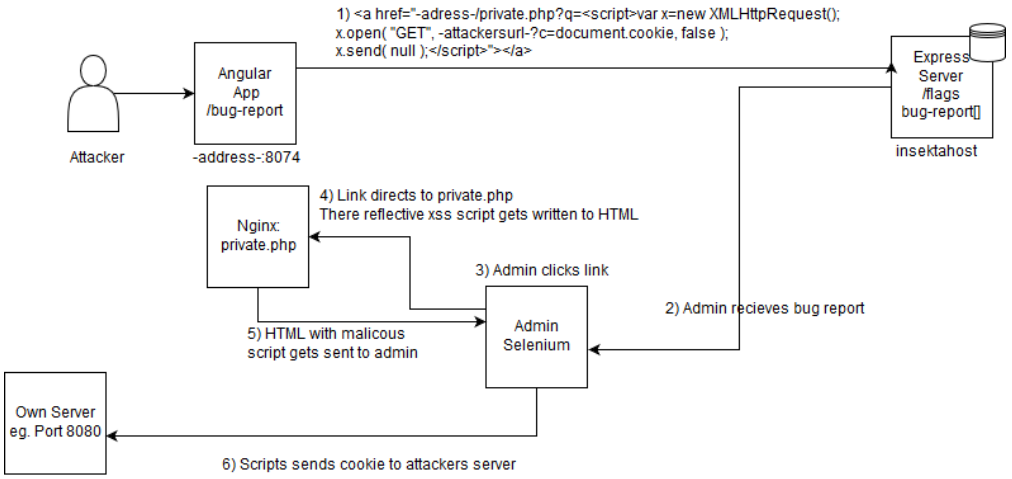

Figure 1.2: Illustration of the reflected XSS attack.

Learning objectives: Understand the structure of an URL with a query and additional parameters, understand the use of a robots.txt and understand the meaning of HTTP status codes.

5. **How can you send data from the admin to yourself?**

   When you open the browser developer tools you will find the cookie information of your own machine. JS provides also functions to read the cookies, you can try that out directly within the developer tool. Now, think about how to get another browsers cookie, for example with an XMLHttpRequest or document.cookie embedded in an link. It is necessary to transport the via JS stolen cookie on you own machine. Webservers are the communication medium to request and receive data that can be used for this. So setting up a webserver that the injected script will contact is part of the task.

   Learning objective: Understand how to read cookies and the purpose of a webserver.

Bibliography

[BV08] Prithvi Bisht and VN Venkatakrishnan. "XSS-GUARD: precise dynamic prevention of cross-site scripting attacks". In: *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer. 2008, pp. 23–43.

[Gro+07] Jeremiah Grossman et al. *XSS attacks: cross site scripting exploits and defense*. Syngress, 2007.

[JKK06] Nenad Jovanovic, Christopher Kruegel, and Engin Kirda. "Pixy: A static analysis tool for detecting web application vulnerabilities". In: *2006 IEEE Symposium on Security and Privacy (S&P'06)*. IEEE. 2006, 6–pp.

[Kir+06] Engin Kirda et al. "Noxes: a client-side solution for mitigating cross-site scripting attacks". In: *Proceedings of the 2006 ACM symposium on Applied computing*. ACM. 2006, pp. 330–337.