Insecure Mate Web Shop
– Guides –

October 2021

This document contains background information to the three scenarios / types of attacks, to be able to explain the approach to the students and give helpful and supportive hints during the challenge.

1 Cross-Site-Scripting (XSS)

This scenario concerns types of Cross-Site Scripting (XSS) attacks and provides two tasks to perform such attacks. The first task 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 Basics

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]. [1] describes very detailed a variety of

[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/

XSS attacks, especially from a very technical point of view and provide also an overview on frameworks to perform such attacks. Others, such as [2] 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[3][2]. Other tools such as Pixy provide support for detecting vulnerabilites in web applications[4].

1.2 Task: AdminDiss

1.2.1 Tipp

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.

1.2.2 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

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

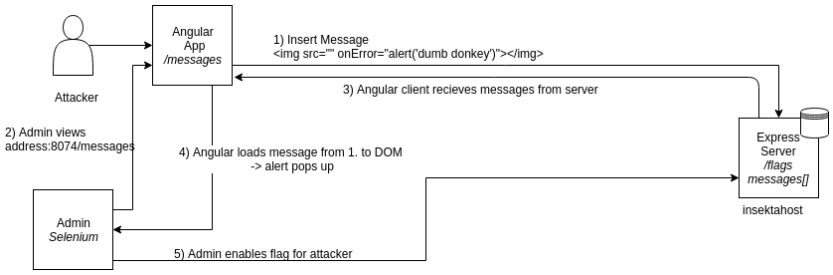

Figure 1: Illustration of the persistent XSS attack.

1.3 Task: BugReport

1.3.1 Tipps

- 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.

1.3.2 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.

   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

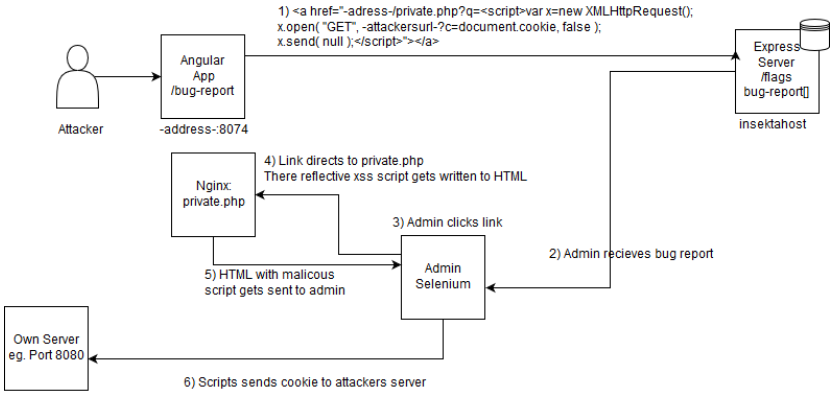

Figure 2: Illustration of the reflected XSS attack.

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.

2 SessionHijack

The Session Hijacking attack consists of the exploitation of the web session control mechanism, which is normally managed for a session token. Because http communication uses many different TCP connections, the web server needs a method to recognize every user's connections. The most useful method depends on a token that the Web Server sends to the client browser after a successful client authentication. A session token is normally composed of a string of variable width and it could be used in different ways, like in the URL, in the header of the http requisition as a cookie, in other parts of the header of the http request, or yet in the body of the http requisition. The Session Hijacking attack compromises the session token by stealing or predicting a valid session token to gain unauthorized access to the Web Server[7].

2.1 Basics

For this topic OWASP provides a short introduction online[8]. It is a good source for initial research in this area.
The type of session hijacking in this task is based on a predictable session ID,

[7] `https://www.owasp.org/index.php/Session_hijacking_attack`
[8] `https://www.owasp.org/index.php/Session_hijacking_attack`

which can be brute forced. For example [5] shows how such attacks may work. The brute forcing is possible because the session ids here are poorly constructed. Following the guidelines of [6] helps in understanding and preventing such mistakes.
In other common session hijack variants, XSS plays an important role. For solving this scenario an insight to sessions and cookies[9] might be helpful.

2.2 Task: Session Hijack

2.2.1 Tipps

- It is required to write a script in any programming language to produce a number of sessionIds, since this is kind of a brute force attack.
- The sessionId consists of two values that must be used as a combination to hack the session of Donald.

2.2.2 Guide

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

1. **What is hidden in the server configs?.**

   In general, existing vulnerable mechanisms of the server configuration should be exploited. At first you should observe how your own session ID looks like and how the requests of you and Donald to the server look like. Every server has a logging page for the requests send from the client. In this case it can be accessed by everyone via its filename. Find out what webserver is used and what the file name could be. The log file contains the own sessionId and another one which is from Donald. Compare the log file with the sessionId in the web developer tools in the browser. The clue is that the sessionId consists of two parts named *id* and *session* where *id* is basically the cookie and the *session* is a generated string. The attacker should notice that the logged requests consist only of the first part, the *id*, whereas a successful hack of the profile requires both *id* and *session* value.

   Learning objective: Understand mechanisms to monitor server communication traffic and understand how sessionIds are structured.

2. **Hands-on: Collect sessionIds!**

   View how sessionIds change with each request. They are newly generated by each request. The attacker should now know that new *session* cookies were generated with each request to `/session?id=...` at the backend-server (i.e. expresserver at port 3000). Requesting those sessionIds for an already existing *id* cookie, the server responds with a specific error

[9] `https://www.ietf.org/rfc/rfc2109.txt`

message. After querying a few of those sessionIds, the attacker should at some point find out, that these sessionId's are restricted to specific values. The goal hereby is to collect as many *session*'s as possible in order to get information about the space of possible session ids.

Learning objective: Learn how sessions are generated.

3. **Whats the right combination?**

   In order for the attacker to come up with own session cookies, he or she should notice that the *session* value is an text encoded as a hexadecimal ascii value. After decoding the values, it should be obvious that the sessionId's are made up of a bavarian adjective and corresponding noun. Several combinations of adjectives and nouns must be combined and sent via an request. With the help of these values the attacker can construct own sessionId's by combining all the retrieved values and perform a brute force attack with the first part of the cookie from the access.log and the unknown second part via the request url: `http://192.168.178.32:3000/profile?id=<id>&session=<session>`.

   Learning objective: Learn to perform a brute force attack.

3 SQL Injection (SQLi) - Password Hack

This scenario covers the topic of Blind SQL Injections and is separated into two tasks where the first task shall prepare the attacker for the *real* attack in the second task.

SQL injection refers to all attacks where queries are injected via data input form into a client application. The injection then is executed by predefined SQL statements i.e. dynamically constructs the query when the input is stored in a variable. Whenever this is possible, the attacker can manipulate (INSERT, UPDATE, DELETE) data in the database such as passwords etc. or other administrative tasks. The prerequisite to perform such kind of attacks is that the SQL language is used. However, the interacting platform or database i.e. whether MySQL, sqllite or PostGreSQL does not matter.

3.1 Basics

To gain an understanding on how such attacks work, a complete sql injection attack can be viewed in several tutorials on the web[10][11].

An overview on SQLi attack types and example applications can be found in [7], including essential concepts that are useful for this scenario. These essential concepts include the use of tautologies and UNION queries and they are both explained in an understandable way. This paper also includes an evaluation

[10] `https://www.youtube.com/watch?v=WFFQw01EYHM`
[11] `https://portswigger.net/web-security/sql-injection/union-attacks`

of detection-focused and prevention-focuses techniques/tools from research and the techniques are described according to the attack types.

For example a highly cited detection-focused tool is the monitoring tool AMNESIA [8]. Prevention-focus mechanisms such as [9] present architectures that do not even make it possible to perform SQLi attacks.

From the web research, several open-source SQL injection tools to perform such attacks exist[12]. For example SQLMap - Automatic SQL Injection And Database Takeover Tool is a commonly known and used tool.

In this scenario we present error-based SQLi and blind SQLi where the first one is often referred to as the *standard* SQL injection since it was used in the early age of SQL injections. However, since the attack is one of the most common attacks, website providers learned from the error-based attacks and try to remove the basis for this attacks i.e. the error messages that provide useful information for the attacker. Therefore blind SQL attacks should be investigated and how they can be applied[10].

3.2 Task: Warm-Up

This task should help to gain some understanding of general SQL attacks and the difference between blind SQL and error-based SQL injection. Find out which type of SQL injection attack we use here.

The secret is basically the answer whether it is an error-based or blind sql injection attack. The goal of the task is to sensitize attackers for the difference between the attacks.

3.2.1 Tipps

- Wild cards are used in the db query [13].

3.2.2 Guide

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

1. **Which input fields are vulnerable? Start with a basic SQL attack**

   The basic SQL attack starts with trying to provoke an error message. At first, the attacker must detect the vulnerable input field. Some fields use a prepared statement, some not. However, the search field is the only field that is possible. A prepared statement will make a check before executing the query to the database, where a plain hand-over from the input field can modify the query. In order to produce an error, the attacker should insert some statements that evaluate to true or false (via an tautology) in every possible input field.

[12] `https://kalilinuxtutorials.com/sql-injection/`
[13] `https://www.w3schools.com/sql/sql_wildcards.asp`

2. **What can be observed by the server response?**

   The attacker should notice that no error message is generated but the response from the server is different when inserted in the search field.

   The response from the server is displayed as the list of items or the JSON object that can be viewed in the developer tools. Since no error message is displayed but the content changes the attacker knows it is a blind sql attack.

   Learning objective: Understand the difference between a error-based and blind SQLi attack and understand the syntax for inserting malicious SQL in this scenario.

3.3 Task: Password Hack

After the warm-up task, this task aims to implement an effective SQL injection to read from the database. The goal of this task is to access the admins password which is stored as a not-salted MD5 hash in the database.

3.3.1 Tipps

- The task can also be solved with a binary search instead of the UNION operator. Therefore the password must be guessed character by character.
- When using the UNION operator: Each select statement within must have the same number of columns and each column must have similar data types. Further the columns must be in the same order.

3.3.2 Guide

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

1. **How is the database organized?**

   In order to retrieve information from a database you must be aware of the underlying database schema i.e. the tables, attributes and relations. Basically in such attacks the required table and attribute names must be guessed by the attacker. However, it is often the case that the admin is stored as the first user in the database and therefore have attributes such as the username *admin* and id 1. First, try to insert a query to find users with the username admin and observe the number of attributes (number of columns) that are stored in one row.

   Learning objectives: Find entry point to the database for SQLi attacks.

2. **How to get the required information?**

   Basically the scenario can be solved with a binary search by collecting the password character by character or either with the UNION selector. The

UNION keyword lets you execute one or more additional SELECT queries to the *original* SELECT query of the search string i.e. the original search query is in form of *SELECT inputstring FROM table*. This additional SELECT statement appends the results to the original query i.e. the query that takes the search string and performs a SELECT operation on the database. Two conditions must be met: both tables must have the same number of columns as well as the same datatypes. Therefore you have to observe the previously mentioned JSON object.

Learning objectives: Understand how the query for the displayed items looks like and abuse of the UNION operator.

3. **What security mechanisms can be used to store passwords?**

   Passwords are usually not stored in plain text in the database for security reasons. Even if an attacker is able to hack the database and get the passwords from users, the attacker will not be able to use the hashed password for a login, etc. Traditional encryption algorithms include the md5 algorithm (which is not recommended because of security gaps.) The usage of md5 can be guessed by the byte length of 128 bytes. The hashed password can be decoded in an online decoder because it is unsalted i.e. no additional string is appended to increase the entropy of the password value.

References

[1] Jeremiah Grossman, Seth Fogie, Robert Hansen, Anton Rager, and Petko D Petkov. *XSS attacks: cross site scripting exploits and defense*. Syngress, 2007.

[2] 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*, pages 23–43. Springer, 2008.

[3] Engin Kirda, Christopher Kruegel, Giovanni Vigna, and Nenad Jovanovic. Noxes: a client-side solution for mitigating cross-site scripting attacks. In *Proceedings of the 2006 ACM symposium on Applied computing*, pages 330–337. ACM, 2006.

[4] 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)*, pages 6–pp. IEEE, 2006.

[5] David Endler. Brute-force exploitation of web application session ids. 12 2001.

[6] Kevin Fu, Emil Sit, Kendra Smith, and Nick Feamster. The dos and don'ts of client authentication on the web. In *USENIX Security Symposium*, pages 251–268, 2001.

[7] William G Halfond, Jeremy Viegas, Alessandro Orso, et al. A classification of sql-injection attacks and countermeasures. In *Proceedings of the IEEE International Symposium on Secure Software Engineering*, volume 1, pages 13–15. IEEE, 2006.

[8] William GJ Halfond and Alessandro Orso. Amnesia: analysis and monitoring for neutralizing sql-injection attacks. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, pages 174–183. ACM, 2005.

[9] Stephen W Boyd and Angelos D Keromytis. Sqlrand: Preventing sql injection attacks. In *International Conference on Applied Cryptography and Network Security*, pages 292–302. Springer, 2004.

[10] Cameron Hotchkies. Blind sql injection automation techniques. *Black Hat Briefings*, 2004.