Tool for Teaching Web Application Exploits and Defences

Krystof Bezdek

4th Year Project Report
Computer Science
School of Informatics
University of Edinburgh

2024

Abstract

In the current digital age, creating an educational tool that focuses on both the identification of web security vulnerabilities and the learning of effective defences is crucial, particularly for those vulnerabilities as pervasive and dangerous as Cross-Site Scripting (XSS).

This report introduces an innovative tool designed to demystify XSS attacks and defences for inexperienced students, providing them with a practical, hands-on learning experience. My tool, named WSED, is a web application that simulates a blog, allowing users to interactively learn and understand the mechanics of XSS vulnerabilities, including both reflected and stored types. Through a guided tutorial and interactive challenges, WSED aims to bridge the gap between theoretical and applied knowledge, enabling students to comprehend the complexities of web security in a controlled, risk-free environment.

This report details the design principles behind WSED, its educational methodology, and the feedback from evaluations conducted with Informatics students at The University of Edinburgh. According to my results, tools like WSED can significantly enhance understanding of web security concepts, making them indispensable resources in the cybersecurity education landscape.

Research Ethics Approval

This project obtained approval from the Informatics Research Ethics committee.
Ethics application number: 988079
Date when approval was obtained: 2023-12-11
The participants' information sheet is included in the Appendix H. It serves as the consent form as well, and it was given to the participants as part of the questionnaires.

Declaration

# Acknowledgements

First I would like to thank Myrto D. Arapinis, my supervisor. Without her support, guidance, and willingness to patiently answer all my questions, I would not have been able to complete this project.

I would like to thank all the students who answered my student questionnaire, and especially all the six students who sacrificed their time to participate in my Think Aloud sessions.

I would also like to thank my dear friends, Laura Tomsova for always patiently proof-reading anything important I write, and Zoe Lai for teaching me how to use Canva.

Last but not least, I would like to thank my family who were always supportive of me and made my educational journey possible.

Table of Contents

Chapter 1

Introduction

In today's digital era, web applications have become ubiquitous, serving as the backbone of daily activities for millions of users worldwide. From e-commerce and social networking to online banking, these platforms are integral to the seamless execution of a multitude of tasks. However, the extensive reliance on these applications has escalated the risk and impact of web security vulnerabilities, with Cross-Site Scripting (XSS) being identified as the second most prevalent Common Weakness Enumeration (CWE) vulnerability in the Open Web Application Security Project (OWASP)'s Top 25 Most Dangerous Software Weaknesses list [2]. This highlights a critical challenge in safeguarding digital interactions against malicious exploits.

Concurrently, the demand for skilled computer security specialists is surging [39], driven by the increasing sophistication of cyber threats [19]. Despite this growing need, there remains a significant gap in the education and training of these professionals, particularly in providing hands-on, applied experience in computer security [43]. This discrepancy underscores the urgency to evolve educational approaches to better prepare the next generation of cybersecurity experts, equipped not only with theoretical knowledge but also with practical skills to navigate and mitigate the complexities of web security threats.

1.1 Project Aim

This project discusses the entire development lifecycle of a web application that serves as a tool for teaching web security exploits and defences. It aims to teach users about Cross-Site Scripting (XSS), the most common web application vulnerability, and how to exploit it. The target audience for this tool is students who took the Computer Security course at The University of Edinburgh and have little to no applied web security experience. However, the students are expected to have programming experience from other classes and basic knowledge of front-end technologies such as HTML and JavaScript. After using the tool, students should be well equipped to tackle more challenging exploratory tools, such as Google's Gruyere, or participate in Capture the Flag competitions. The tool can be used as well by the teaching staff for demonstrating XSS exploits and introducing the students to the applied side of web security.

The WSED web application presents theoretical background information about XSS, followed by two exercises where the users get to exploit XSS vulnerabilities by conducting two of the most common XSS attacks - reflected and stored. Finally, the tool presents the users with detailed descriptions of various defence methods used against XSS attacks. Hints, solutions, step-by-step guidance and explanations are included to cater to the target audience.

The methodologies used in this project include explicit requirements gathering from the student questionnaire and implicit requirements gathering from conducting cognitive walkthrough evaluation of Google's Gruyere and uncovering opportunities for usability improvements. The web application is then designed and implemented in accordance with the requirements, and finally tested with the target audience to evaluate if the learning goals and requirements are met. While the tool meets its targets, further improvements could be implemented and additional testing is recommended. The learning goals for users of this tool are stated as follows:

1. The user should be able to understand how XSS attacks work and the differences between the two most common types.

2. The user should be able to perform both basic reflected and stored XSS attacks.

3. The user should know the common and most effective XSS defence techniques.

1.2 Report Structure

The remaining chapters are structured as follows:

Chapter 2 - Web Attacks and Defences provides an overview of the web vulnerabilities taught in the Computer Security course, and establishes the most prevalent one as the focus of the tool.

Chapter 3 - Related Work establishes the context of Capture the Flag, reviews existing software used as a source of inspiration, and showcases the benefits of gamification.

Chapter 4 - Requirements Gathering describes the process of requirements gathering used and defines them.

Chapter 5 - Design and Implementation outlines the design philosophy, presents the technologies used, and provides details of the development methodology.

Chapter 6 - Evaluation Methodology describes the methodologies used in order to evaluate the tool.

Chapter 7 - Sprint One presents the development details and the evaluation results achieved during Sprint One.

Chapter 8 - Sprint Two presents the development details and the evaluation results achieved during Sprint Two.

Chapter 9 - Conclusions summarises what has the project achieved and discusses potential limitations and future work.

Chapter 2

Web Attacks and Defences

In this chapter, we will briefly outline some of the most common web application vulnerabilities taught in the Computer Security course, such as Cross-Site Request Forgery (CSRF), SQL Injections (SQLi) and Command Injections, followed by a detailed description of how to exploit and defend against Cross-Site Scripting (XSS). All of the following web application vulnerabilities are placed every year in the CWE Top 25 Most Dangerous Software Weaknesses [2], with XSS (CWE-79) placing as the 2nd most common dangerous software weakness overall, followed by SQLi (CWE-89) in 3rd place, and finally CSRF (CWE-352) in 9th place.

2.1 Common Web Application Vulnerabilities

2.1.1 Cross-Site Request Forgery

Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated [26]. This is done by building an exploit URL and tricking the victim into clicking it. CSRF attacks try to change the state of the website on a server, for example, making the victim send money to the attacker's bank account. Retrieval of information is not the goal for CSRF attacks, but change of state is, because the response from the server is sent to the victim and the attacker cannot see it. CSRF vulnerabilities occur when web applications fail to validate the source of incoming requests.

To defend against CSRF attacks, three techniques are commonly employed. First, each user session-associated request is validated using a secret token that is difficult for attackers to guess. Second, the HTTP Referrer header, which provides information about the origin of the request, is validated for each request. Third, custom headers attached to XMLHttpRequests, such as the X-Requested-With header, are used to differentiate legitimate AJAX requests from CSRF attacks. However, it is important to note that employing these three techniques alone does not guarantee complete protection against all CSRF attacks [5]. Therefore, a defence-in-depth strategy should also be implemented to further enhance security.

2.1.2 Injections

Injection attacks are malicious techniques used to exploit vulnerabilities in software applications by injecting unauthorised code or commands into input fields. The injected code can manipulate databases or execute commands. Two such common types of injection attacks are SQL and Command injections [45].

SQL injection attacks make use of unchecked input fields to execute SQL queries in the vulnerable server's database. A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete) or execute administration operations on the database, such as shutdown of the database management system [25].

Similar to SQL injection attacks, Command injections make use of improper input validation to execute commands on the host operating system. These attacks are executed with the privileges of the vulnerable application in the system shell [46]. A successful command injection attack can use the *whois* command to query information about Domain names, IP addresses, and ASN of the operating system.

To mitigate the risk of injection attacks, two key strategies are utilised. First, parameterised statements are employed, enabling databases to distinguish between code and data regardless of user input. Second, user input is escaped and validated against predefined criteria for data format, length, and character sets, with non-conforming input being rejected. Additionally, sanitisation processes remove or encode harmful characters and escape special characters prior to their use in database queries or commands. Emphasising a multi-layered defence approach is essential to lessen the risk associated with these vulnerabilities [32].

2.2 Cross-Site Scripting

Cross-Site Scripting (XSS) is a type of security vulnerability typically found in web applications [27]. This flaw allows attackers to inject malicious scripts into web pages viewed by other users. XSS attacks occur when an application includes untrusted data in a web page without proper validation or escaping, thereby enabling attackers to execute scripts in the victim's browser. Such scripts can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site, leading to malicious activities such as key logging [17]. XSS vulnerabilities can be exploited by crafting malicious links, form inputs, or URL parameters, and they are mitigated through proper data sanitisation, validation, and the use of secure coding practices.

There are two most common XSS attacks: Reflected and Stored. Both of these attacks target the users of the web application, exploiting their trust in the web application. The attackers try to inject the web application with a malicious script, be it JavaScript, HTML Flash, PHP, or other type of code that can execute in a victim's browser, and use it to steal cookies, session tokens, sensitive information the browser stores, or deface the web page and even redirect the user to malicious sites. The main difference between these attacks is how the injection occurs.

2.2.1 Reflected Attack

In Reflected XSS attack, the attacker injects a malicious script into a web page through the user's request (for example in a search form or URL parameter) and includes it directly in its response without proper validation or encoding, which is then immediately reflected back to the victim and executed by their web browser [17] [27]. The malicious script, embedded in a crafted request (often a URL), is reflected back to the victim's browser, where it executes in the context of the victim's session. Unlike stored XSS, which persists in the web application, reflected XSS is non-persistent and requires a user to actively trigger the attack, often through clicking on a link or a misleading email. While it requires more direct interaction with the target victim, reflected XSS is still a dangerous and common form of web application vulnerability due to its potential for immediate impact and its relative ease of execution.

2.2.2 Stored Attack

In Stored XSS attack, the attacker injects a malicious script into a web application's persistent storage [17] [27]. This typically happens through input forms or any other data submission points that accept user-generated content and store it. The malicious script, once stored, is then served as part of the web application's content to other users. When these users access the compromised page, the malicious script executes in their browsers. Unlike other forms of XSS, stored XSS does not require a victim to click on a link; it automatically executes when the compromised page is loaded. Due to its persistent nature and potential for widespread impact on users, stored XSS is often considered more severe than other types of XSS attacks.

2.2.3 Defences

Defending against any type of attack requires robust and deep defence - implementing just a single method is not enough, using the right combination of defensive techniques is necessary to prevent XSS. This section outlines some of the common methods of protecting web applications from XSS attacks [34].

2.2.3.1 Output Encoding

Output Encoding [34] ensures that any user-generated input displayed on the page is escaped correctly, meaning that characters that could be interpreted as HTML or JavaScript are converted to safe representations. Below are common problematic characters that should be encoded. Convert ("|" is a separator):

& to **&amp;** | < to **&lt;** | > to **&gt;** | " to **&quot;** | ' to **&amp;x27;**

2.2.3.2 Input Validation and Sanitisation

Escape dangerous characters supplied in user input and accept only known good input by whitelisting headers, cookies, query strings, form fields, hidden fields, and other data [35]. While input sanitisation is overall a very important aspect of computer security, output encoding is much more important for XSS prevention, because the script execution happens during the rendering of the output, not during the injection of the data.

2.2.3.3 Content Security Policy

Content Security Policy (CSP) [33] HTTP headers restrict the sources from which scripts can be loaded. CSP can effectively prevent the execution of unauthorised scripts, even if an attacker manages to inject malicious code into a web page. Below is an example of such a header.

Content-Security-Policy: default-src 'self'; script-src 'self'; object-src 'none'; base-uri 'self';

default-src 'self' only allows content (scripts, styles, images, etc.) from the site's own domain.

script-src 'self' only allows JavaScript to be executed if it comes from the same domain. This excludes inline scripts and scripts loaded from other domains.

object-src 'none' prevents loading plugins like Flash.

base-uri 'self' restricts the URLs which can be used in a document's <base> element.

2.2.3.4 HttpOnly

HttpOnly [38] flag is used to prevent access to cookie data via JavaScript. This can help mitigate the damage of certain XSS attacks by not allowing stolen cookies to be used in session hijacking. The HttpOnly attribute is specified in the Set-Cookie HTTP response header from the server. Below is an example of the HttpOnly attribute being set.

Set-Cookie: SessionId=xyz123; Path=/; Expires=Wed, 09 Jun 2024 10:18:14 GMT; HttpOnly

SessionId=xyz123 is the cookie being set.

Path=/ indicates that the cookie is available for pages under the specified path (in this case, the root directory).

Expires=Wed, 09 Jun 2024 10:18:14 GMT specifies when the cookie will expire and be removed.

HttpOnly tells the browser that this cookie should not be accessible via JavaScript through the document.cookie function or other client-side mechanisms.

2.3 Summary

We explored common web application vulnerabilities, how can they be exploited, and what are some effective defence strategies. These vulnerabilities, including CSRF, XSS, and injection attacks, pose significant threats to web application security. Learning about these vulnerabilities is essential for keeping web applications safe and should be known by every computer scientist or software engineer.

Chapter 3

Related Work

In this chapter, we explain the concept of Capture the Flag systems and discuss how are they utilised in the cybersecurity world. We also outline various existing tools for teaching web security which act as an inspiration for this new tool, and finally, we delve into gamification and its benefits in the world of higher education.

3.1 Capture the Flag

Capture the Flag (CTF) is an outdoor team-based game that combines strategy, stealth, and physical activity. Two or more teams compete to capture the opposing team's flag and bring it back to their own base without being tagged by opponents. Each team hides their flag anywhere in their designated home territory. In order to win, players have to employ tactics like guarding their flag, launching surprise attacks, and forming defensive and offensive strategies [8].

The main idea of CTF games in cybersecurity is very similar: teams or single players compete to find flags in specific territories - territories being vulnerable software, such as web applications, mobile applications, or networks, and flags being solutions to the given problems, usually in the form of strings or other elements designed by the creators of the specific CTF [28]. The most common kind of CTFs is Jeopardy where a team or player have to complete a set of challenges, which are usually about exploiting a vulnerability and finding the flag [1].

Capture the Flag games and competitions have been a part of the cybersecurity world for decades and have been proven to be an effective way for participants to learn about new vulnerabilities, gain practical experience and deepen their understanding of the underlying cybersecurity concepts [44]. However, these challenges are usually of an exploratory nature and are designed for people who already have solid foundations in cybersecurity and are confident with their skills. This makes it very difficult for casual learners to start playing them and be able to solve the problems, making them inadequate for use as teaching tools [29] [11].

3.2 Tools for Teaching Web Security

These are some noteworthy platforms for learning about cybersecurity and practising exploits and defences that serve as an inspiration for this project.

3.2.1 Google's Gruyere

Google's Gruyere [16] is a unique educational tool designed by Google, crafted to mimic a blog website filled with intentional security vulnerabilities, akin to CTF competitions. It aims to educate users about common web vulnerabilities through detailed descriptions, challenges, and hints for exploiting and defending against these flaws. The platform covers a wide range of security issues, such as Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), and SQL Injection, providing an invaluable learning resource for those keen on understanding and improving in web security.

However, Gruyere's non-linear, sandbox-style learning environment, while offering a realistic exploration of vulnerabilities, may pose difficulties for beginners. This format demands a degree of self-initiative and persistence from users, which can be challenging for those new to web security or preferring more structured guidance. Despite this, Gruyere excels as an engaging tool for practical cybersecurity education, encouraging a hands-on approach to learning and problem-solving in the field of web security.

3.2.2 Portswigger

Portswigger is a company known for its software application called Burp Site used for penetration testing of web applications. Portswigger provides a Web Security Academy where users can learn about various topics in cybersecurity [37]. Each topic has labs associated with it, so users can test their skills and exploit vulnerabilities in contained environments. Their approach and interface are similar to Gruyere's, however, each lab has its own environment only with a specific vulnerability, unlike Gruyere, where you have a large single sandbox to play in. This challenge modularisation makes it more friendly towards beginners, but it can still be overwhelming to beginners with the sheer amount of information and options presented to them.

3.2.3 Hack the Box

Hack the Box is a cybersecurity platform that offers gamified environments for individuals and organisations to test their penetration testing skills [18]. It provides a hands-on learning experience for cybersecurity enthusiasts, offering an extensive collection of hacking challenges, Capture The Flag competitions, and a supportive community forum where members can collaborate, share knowledge, and sharpen their skills. Unlike Gruyere and Portswigger, Hack the Box does not provide materials on theory in security concepts and their challenges are more complex, often requiring users to inspect source code, write their own scripts, or use known cybersecurity tools. The gamification aspect makes the challenges more fun, as users may find themselves helping a human resistance group hack a human resources database of aliens in order to extract information from them. However, due to the difficulty of the challenges and lack of learning material, Hack the Box is not an ideal option for beginners.

3.2.4 Other Online Learning Platforms

Online learning platforms like freeCodeCamp and Codecademy have grown in popularity in recent years, particularly in the field of technology and programming [40]. freeCodeCamp offers a wide range of coding courses, focusing on interactive learning through coding exercises and projects, enabling users to build a portfolio as they progress. Codecademy emphasises both theoretical and hands-on approaches through lessons and practical exercises. Both platforms cater to beginners and those looking to advance their skills, offering flexible learning schedules and resources that are accessible from anywhere with an internet connection, making them popular choices for learners worldwide. Both cover various programming languages and computer science topics, including web security, however, they do not offer exercises where the users could perform real attacks.

3.3 Gamification

Deterding et al. [12] define gamification as the incorporation of game-based elements into non-game contexts, with the aim to boost motivation and involvement in tasks that do not naturally involve gaming. This strategy taps into the natural human predisposition to engage in gaming [24], using it to foster desirable actions in a way that's both fun and engaging. It involves more than merely adding points, badges, or rewards to an educational experience [24]. Instead, gamification embeds features such as puzzle solving, storytelling, and progression, enriching the user's engagement and immersion.

Gamification is widely used in many sectors, notably in education. Rabah, Cassidy and Bauchemain [20] conducted a second-order review to summarise and examine the effectiveness of gamification in education. They found that it enhances learning achievement, knowledge retention, and higher-order thinking skills by making courses more engaging and motivating. Moreover, it shows a positive effect on lower-risk assignments such as coursework, and overall course grades, but its impact on final exams is less pronounced. Due to these positive impacts on user experience and learning, I decided to attempt to incorporate gamification elements into my tool.

3.4 Summary

In this chapter we covered the concept of Capture the Flag and teaching tools for web security designed in its style. We saw that CTFs as teaching tools enhance the cybersecurity skills of individuals, but are not an ideal entry point for complete beginners. We discussed the benefits of integrating gamification elements in teaching tools used in higher education, and finally, we provided an overview of teaching tools for web security such as Google's Gruyere and Portswigger. We found out that while all these tools are of high quality and valuable to experienced users, they are not well-suited for beginners. They can, however, serve as a source of inspiration for this new tool.

Chapter 4

Requirements Gathering

This chapter outlines the requirements gathering process and methodologies used. We conducted a questionnaire among the University's students who took the Computer Security course to gain insights into the target group's knowledge, experience and demand. Furthermore, we use Human-Computer Interaction (HCI) methods called cognitive walkthrough with heuristic evaluation to analyse the strengths and weaknesses of an existing tool for teaching web application exploits and defences called Google's Gruyere.

4.1 Persona

Persona used in HCI methods is a fictional character created based on user studies to represent the different user types that might use a system in a similar way. Personas are used to understand and communicate the needs, behaviours, and goals of the target audience throughout the design process [9]. A persona representing our target group is a third-year undergraduate student who just attended the lectures about web security and they have little to no experience with performing XSS attacks. This persona serves as the central focus throughout all stages of requirements gathering, design, and implementation.

4.2 Student Questionnaire

The student questionnaire can be seen in Appendix B. It was conducted online using Google Forms and with a total of 14 respondents who took the Computer Security course.

In the first section my goal was to understand the students' confidence in theoretical and applied knowledge of Computer Security and see which of the five topics (Network Security, Cryptography, Secure Communications, Operating Systems Security, Web Security) they are confident in and with which they struggle.

The second section narrowed it down to web security and its goal was to understand the students' theoretical knowledge and hands-on experience with exploits of vulnerabilities outlined in Chapter 2 and compare the differences between them.

The final section focused on gathering information about the students' preferences regarding theoretical and hands-on learning, their experience with learning tools and CTFs, and their suggestions about what features should a tool for teaching web security contain.

4.2.1 Results

In the first section, 92.8% of students responded that their overall theoretical knowledge of Computer Security is either 'Good', 'Very Good' - which was the overall most popular answer with 57.1% - or 'Excellent'. The rest rated their knowledge as 'Fair' and no one answered 'Poor'. The web security topic was considered 'Easy' or 'Very Easy' by 78.6% of respondents. The rest was split between 'Difficult' and 'Neutral' and no one considered it 'Very Difficult'.

Furthermore, this trend of considering theoretical web security 'Easy' continued in the second section as well, with 'Easy' being the most common answer across all major topics taught in the web security part of the course. Additionally, 100% of the respondents said they chose the web security question as one of the two questions to answer in the final exam, empirically supporting the notion that theoretical web security is considered overall easy.

However, when asked about their confidence in performing attacks taught in the web security part of the course, the majority were either 'Not confident' or 'Not confident at all' at performing Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF) attacks. The attack with highest confidence was SQL Injection, with the majority of students saying they were 'Confident' or 'Very confident'.

In the final, Learning Tools section, 57.1% of respondents said they prefer learning hands-on about exploits and defences, 21.3% said they prefer a mix of hands-on and theoretical approach, and the rest prefers to learn only the theory.

This data shows the large disparity between confidence in theoretical and applied knowledge of computer and web security concepts among the students, while highlighting the demand for hands-on experience with exploits to accompany the theory and reinforce their learning, stressing the need for a learning tool catered to these students. Finally, Figure 4.1 shows specific suggestions and ideas for features gathered that could be included in such a tool and can be seen in Section 4.4.

4.3 Evaluation of Google's Gruyere

4.3.1 Methodology

As a part of the requirements gathering phase, I undertook an evaluation of the system to identify potential enhancements in usability for my tool. The evaluation has been done solely by me as an HCI expert using the cognitive walkthrough method combined with heuristic evaluation.

4.3.2 Heuristic Evaluation

Heuristic evaluation is an HCI method where one or more experts evaluate the general usability of a system against a set of criteria. We used the most common set of heuristics called Nielsen's 10 Heuristics [30] [31] while conducting the cognitive walkthrough.

4.3.3 Cognitive Walkthrough

Cognitive walkthrough is an evaluation method done by one or more experts without the to involve participants or end users, making it an ideal approach for an initial evaluation of a system [36]. The evaluators use the interface and try to accomplish multiple tasks a typical user of the system would do. Each task comprises multiple actions and responses which are evaluated against the goals and knowledge of the typical user. In order to capture the cognitive activities of the typical user, a persona from Section 4.1 is used [42].

Three tasks were devised for this cognitive walkthrough:

1. Start the coding lab
2. Complete the Reflected XSS challenge
3. Learn how to defend against it

During each action of each task we evaluated if the persona would know what to do next to get the desired response and used Nielsen's 10 Heuristics to capture the problems if the persona would be unsuccessful. A summarised paragraph was written for each usability aspect report, outlining the negative and positive aspects. The full reports can be seen in Appendix C.

4.3.4 Results

Four main negative and two positive aspects were found with the system. Each negative and positive aspect is associated with one heuristic from Nielsen's 10 Heuristics.

4.3.4.1 Negative Aspects

Visibility of system status - The system provides no information on the success or failure of a challenge. The user has to know the result they are looking for, which may be confusing for students learning these exploits. They may try to conduct an exploit and think what they did it is sufficient, although it is not, or vice versa. This will impact third-year students trying to learn about web vulnerabilities, as it is frustrating to not be able to validate the correctness of answers.

User control and freedom - When a user clicks the link to start the application, it replaces the instructions tab. This can be disorienting as all the instructions vanish abruptly, leaving the user without guidance and necessitating a return to the previous page. Furthermore, instead of featuring a straightforward "Start" button, the user is required to click a link embedded within a paragraph of text, alongside other similar links, which can potentially lead to confusion.

Recognition rather than recall - The set of instructions, explanations, challenges, hints and fixes are all in a separate tab from the system where the user performs all actions. This means that the user finds themselves constantly clicking between these tabs. It makes it very difficult to use the system because users forget the information given in the assignment and have to constantly go back and forth to remind themselves of what to do.

Aesthetic and minimalist design - The website is overwhelmingly cluttered with an extensive volume of text and information, much of which may be entirely irrelevant to the tasks a user aims to complete. Consequently, users often find themselves investing unnecessary time in going through content that holds little significance for them, resulting in frustration. With the exception of brief descriptions of hints, exploits, and solutions, the majority of pages are densely populated with small-sized text, making it likely that users will only engage in superficial skimming, potentially leading them to overlook genuinely crucial information.

4.3.4.2 Positive aspects

Match between system and the real world - The application where all the exploits happen is structured in a similar fashion as blogs or social media websites, which makes it intuitive to navigate, as it is something most students will have experience with. Having an intuitive user interface is important for users to navigate effectively, allowing them to focus on the matters at hand.

Help and documentation - Once the user orients themselves within the application, they find out it provides detailed and extensive documentation of how the app works, information about the vulnerabilities, as well as useful hints on how to exploit and fix them. Having an extensive documentation and help available is important for an easy access to official information.

Overall the system was confusing and overwhelming. The amount of information thrown at me from the start was too much and I had to search for tutorials and guides on how to use it. However, after the initial shock, realising what information is relevant to me, and understanding the vulnerable application, I understood the potential this sandbox-like system gives to an experienced user.

4.4 Requirements

Based on the usability aspect reports created from evaluating Google's Gruyere, and the data and suggestions gathered from the student questionnaire, I have extracted these requirements the tool must fulfill in order to achieve the persona's learning goals. XSS was chosen as the focus for this tool due to its high occurrence rates and the students' low confidence in exploiting the vulnerability.

Requirement 1: The tool shall teach our persona XSS attacks taught in the Computer Security course - reflected and stored.

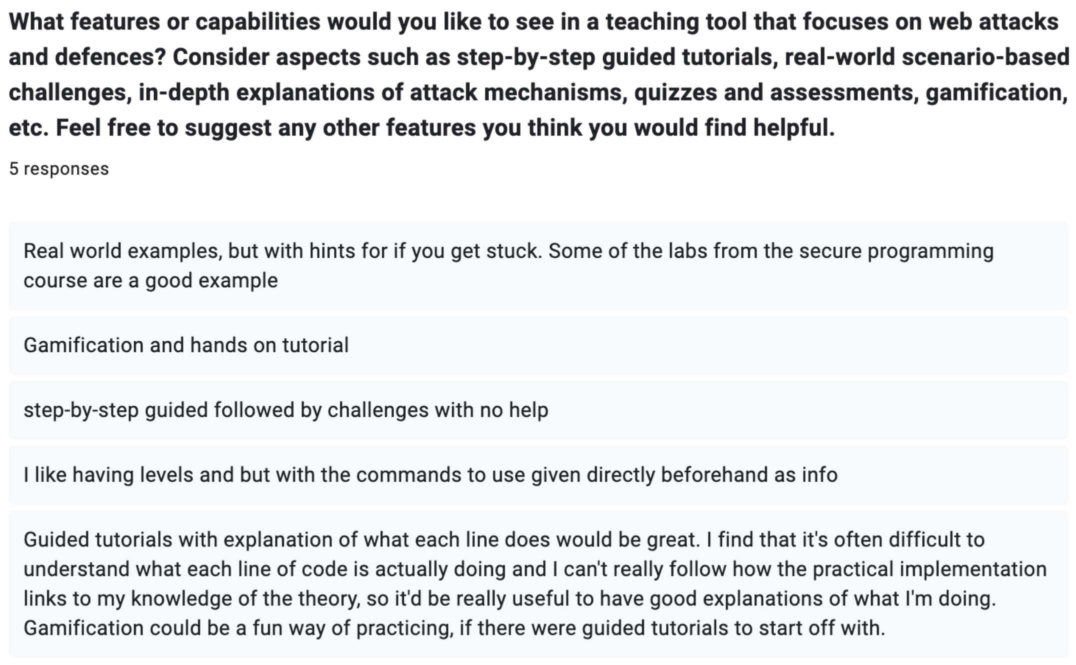
Requirement 2: The tool shall provide a theoretical background of XSS.

Requirement 3: The tool shall provide step-by-step guidance for performing XSS attacks.

Requirement 4: The tool shall provide hints, solutions, and explanations.

Requirement 5: The tool shall adhere to usability heuristics highlighted in Section 4.3.4.

Requirement 6: The tool shall contain aspects of gamification and CTFs.



What features or capabilities would you like to see in a teaching tool that focuses on web attacks and defences? Consider aspects such as step-by-step guided tutorials, real-world scenario-based challenges, in-depth explanations of attack mechanisms, quizzes and assessments, gamification, etc. Feel free to suggest any other features you think you would find helpful.

5 responses

Real world examples, but with hints for if you get stuck. Some of the labs from the secure programming course are a good example

Gamification and hands on tutorial

step-by-step guided followed by challenges with no help

I like having levels and but with the commands to use given directly beforehand as info

Guided tutorials with explanation of what each line does would be great. I find that it's often difficult to understand what each line of code is actually doing and I can't really follow how the practical implementation links to my knowledge of the theory, so it'd be really useful to have good explanations of what I'm doing. Gamification could be a fun way of practicing, if there were guided tutorials to start off with.

Figure 4.1: Suggestions for the tool gathered from the student questionnaire

4.5 Summary

The student questionnaire showed the void in place of applied knowledge of web security exploits and the demand to fill it. Furthermore, it revealed Cross-Site Scripting as a reasonable web vulnerability to be covered by the tool content-wise and gathered requested features the tool could have. The evaluation of Google's Gruyere exposed many usability problems that should be tackled and improved on when creating the new learning tool but highlighted some good aspects which could be built upon. This allowed me to determine the requirements for the tool both in terms of usability and content and set a good foundation for the design and implementation of the tool.

Chapter 5

Design and Implementation

This chapter describes the reasoning behind design choices made for the tool, encompassing the system's type, user interface, and content. It also details the technologies, methodologies, and tools employed in the tool's implementation.

5.1 Design Philosophy

The challenge this tool tries to tackle is the complete lack of applied experience with Cross-Site Scripting (XSS) attacks among the students of the Computer Security course. After analysing other similar existing tools and gathering requirements from the target group, my design philosophy is to eliminate the very high entry requirements and negative usability aspects of existing tools described in Section 3.2. This goal will be achieved by building upon the detected positive features of said tools, integrating aspects of CTFs and gamification, and lowering the entry requirements for inexperienced students by making the tool usable and accessible by improving the user interface and providing optional step-by-step guidance with hints, solutions, and explanations.

5.2 Technologies

Following my design philosophy, I decided that the tool will be a web application since the objective of the tool is to teach students XSS attacks and defences, which is best done in an actual web application where I can create real and purposeful vulnerabilities, rather than simulating them in a game. Furthermore, web applications are flexible regarding contents and design, allowing for a wide range of feature combinations. This makes the web application an ideal design choice for this tool as it can serve both as the source of information and assignments and as a real vulnerable application that should be targeted and exploited. Ease and flexibility of deployment is another aspect to be considered, as they are device agnostic and can run locally as code labs or be deployed online.

5.2.1 Back-end

The web application has to be vulnerable to both reflected and stored XSS attacks. While reflected attacks could be conducted in a purely front-end-based web application, stored attacks require access to persistent storage, traditionally a database. As such, the web application needs a back-end and a database. The choice of framework for the back-end is important because that is where most business logic is implemented and data is manipulated, so the features the framework provides will determine the capabilities that can be implemented. Finding a framework that has flexible security features is crucial, because without that, I could not create vulnerabilities on purpose. Different frameworks are also more suitable for monolithic web applications, whereas others are focused on the implementation of RESTful and micro-services [13]. Finally, I as the developer should preferably be familiar with the framework, or with the programming language it uses, as it saves a lot of time and allows me to use my time for other aspects of the implementation rather than learning new a framework from scratch.

Following the requirements for the framework, I recognised two ideal candidates for the tool, both of which I am familiar with: Java with Spring Boot and Python with Django [23]. Both languages are widely known and both frameworks are highly popular in both enterprise and open-source projects. Spring Boot is a very mature, robust, and well-maintained open-source framework. Its flexibility makes it a powerful framework, especially for Restful applications, however, it is not ideal for monolithic applications as it lacks in user interface development capabilities and is unnecessarily complex for this task. On the other hand, Django is much more straightforward, with a lot of diverse functionality and built-in front-end technologies in the form of Django Templates. It also provides security features such as CSRF tokens and output encoding by default, which can be customised and turned off.

After careful considerations, I have decided to use Django for its power, simplicity, and support for monolithic applications, allowing me to focus on the design and usability aspects of development, rather than the details of technical implementation.

5.2.2 Front-end

For the front-end I used Django Templates. Django Templates allow integration of the back-end with the front-end and dynamic generation of HTML files. It also allows the re-usability of code by creating a base HTML file that can be extended by other files, which helps with the readability and maintainability of the code.

Usability and minimalist, aesthetic design is the main focus of the user interface of the web application, which is why I have chosen to use Bootstrap [6] alongside Django Templates. Bootstrap is a web framework for HTML, CSS and JavaScript that offers pre-made CSS classes and designs. Using Bootstrap allowed me to focus on the general design of the web application rather than spending too much time creating my own. Nevertheless, custom CSS and JavaScript have been added as well for specific designs and functionality that was not covered by Django Templates and Bootstrap.

5.2.3 Containerisation

I chose Docker [14] for deploying and running the web application. Docker abstracts operating system-level virtualisation and utilises images as blueprints for containers - lightweight, standalone, and executable software packages. These images encompass all necessary components to run the application, including code, runtime and compile dependencies, and configuration files. This comprehensive packaging ensures consistent application performance across different environments, effectively resolving the “it works on my machine” issue and making it possible to run the application on any system that supports Docker. Additionally, Docker enhances security by isolating applications within containers, reducing the risk of unauthorised access or systemic vulnerabilities affecting other containers or the host system, thereby bolstering the application’s overall security. This is important in an application that contains vulnerabilities, as it protects the users’ machine from themselves.

5.2.4 Database

Database is needed to store data created by the users, so that they can perform stored XSS exploits. Django provides a built-in SQLite database that gets created automatically during the set-up of a new Django project. SQLite [41] is a lightweight relational database that contains all the data stored in a single file as part of the running application, rather than having a separate process running a server such as MySQL or PostreSQL. This makes SQLite ideal for the purposes of this tool, because the web application runs locally for every user with their own lightweight database. Additionally, migration from SQLite to other SQL based databases is supported and straightforward, allowing for potential expansion and online deployment in the future.

5.2.5 Version Control

I chose Git and GitHub for managing the version control of my software. Git is a robust version control system that enables tracking of source code modifications throughout the software development lifecycle. My previous experiences with Git made it the preferred choice, particularly due to its feature that allows reverting to earlier project versions, invaluable when dealing with software errors, and for managing multiple working versions via branches.

GitHub serves as the online platform complementing Git by facilitating version control and offering a backup solution. It ensures that the code is securely stored online, safeguarding against potential local system failures. Currently, my project on GitHub is set to public, making it accessible and open to anyone who would like to contribute.

5.3 Agile Development

To design and implement the tool, I chose to follow the Agile Development methodology [4], which is centred around the idea of iterative development, where requirements and solutions evolve over time. This approach is widely used in the industry and it is structured around multiple Sprints (iterations) during which I work through a full development cycle aimed at implementing a new working version of the tool. This emphasises user involvement and adaptability, as feedback from users is used to refactor requirements for the future Sprint. During this dynamic approach, the product evolves through stages, reflecting changing needs and new insights, thereby improving both product quality and user satisfaction. I have undertook two Sprints, each consisting of multiple phases:

1. Requirements Gathering
2. Development
3. Evaluation

Figure 5.1 shows the flow of the Agile Development process. The evaluation methodologies are described in Chapter 6.

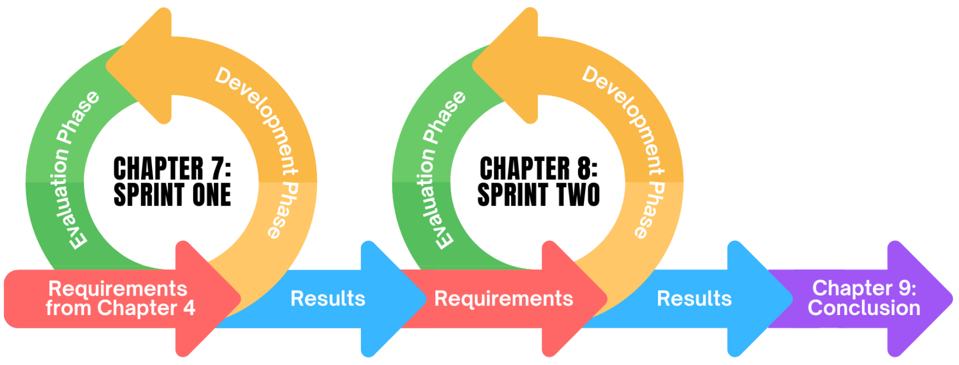


Figure 5.1: Agile Development flow diagram

5.4 Summary

In this chapter, we talked about the design philosophy and decisions made based on the background research done, and the requirements gathered from evaluating Google's Gruyere and conducting a target group questionnaire. As such, the tool was designed to be a web application. We also outlined all the technologies, tools and frameworks used in order to implement the web application and the rationale behind the choices. Finally, we describe the chosen methodology of iterative development called Agile Development and how we use it to deliver the tool.

Chapter 6

Evaluation Methodology

The goal of my evaluations is to measure the knowledge of XSS exploits and defences learned and experience in conducting XSS exploits by the participants after interacting with my tool. The evaluation also aims to determine the usability and learnability of the system in order to make it accessible to inexperienced Computer Security students.

6.1 Pre-tutorial Questionnaire

The pre-tutorial questionnaire can be seen in Appendix D. It was given to the participants before the start of the Think Aloud session and its purpose was to measure the prior knowledge and experience of the participants.

First, the questionnaire asks the participants about their background to see how well they match the persona defined in Section 4.1. The information of interest is their current year of study and if they have taken Computer Security or any other related course, such as Secure Programming. The questionnaire then proceeds to ask the participants to evaluate their theoretical knowledge of web security concepts, specifically of XSS, followed by a question asking them to evaluate their confidence in conducting XSS attacks.

The final section contains eight open-answer questions about XSS attacks and defences. The questions are designed to test the participants' prior knowledge of various theoretical and practical aspects of XSS. These questions are the same as questions in the post-lab questionnaire in order to measure how much the participants improved after completing the tutorial.

6.2 Think Aloud

Think Aloud sessions are a method used in usability testing where participants vocalise their thoughts and observations while interacting with the tool [10]. I chose this technique because it helps me understand the participants' thought process and identify issues that might not be evident through observation alone. Participants express what they are looking at, thinking, doing, and feeling as they navigate through the interface and complete tasks, making think aloud sessions useful for evaluation because they give direct access to the participants' experience, allowing me to make more informed decisions to enhance usability, improve user satisfaction, and ultimately ensure the tool satisfies its intended goals. Finally, in order to capture majority of issues with a system, three to five participants are required, depending on the size of the system, making it an effective method for this project.

6.2.1 Preparation

To run the Think Aloud sessions, I prepared a script that would be read to the participants before the start of the session. This script was borrowed with permission from the Human-Computer Interaction course at the University of Edinburgh and adjusted to my specific needs. It can be seen in Appendix E. The script explains what a Think Aloud session is and what is expected of the participants. It features a simple problem-solving Think Aloud example to be completed by me as the researcher, and then a different one by the participants to ensure their understanding. Additionally, I prepared three straightforward tasks that the participants have to complete:

1. Complete exercise 1 of the XSS tutorial
2. Complete exercise 2 of the XSS tutorial
3. Learn about defences against XSS exploits

6.2.2 Experiments

Every Think Aloud session was run in a quiet place where the participants could focus without any distractions. First I read the script to the participants and after reading the script, I presented the defined tasks and gave them space to ask questions about them. If they had no questions, the Think Aloud session properly started and the participants could start interacting with the system and completing the tasks at hand. During the entire session, I merely listened and observed whatever the participants were doing and took diligent notes. After the participants were done or their time ran out, the session ended and I could answer the participants' questions they may have had during the session. Lastly, the post-tutorial questionnaire was given to each participant.

6.3 Post-tutorial Questionnaire

The post-tutorial questionnaire can be seen in Appendix F. The post-tutorial questionnaire was given to the participants after they had finished the Think Aloud session and its purpose was to gather written feedback and measure their improvement from completing the Think Aloud session.

First, the questionnaire asks if the participant has enjoyed the tutorial, how difficult they consider the exercises, and what is their confidence in conducting XSS attacks after completing the tutorial. Then it proceeds to ask for open answer feedback about what the participants liked or disliked, and suggestions for what features and aspects of the tool could be added or improved.

The second section contains exactly the same set of eight open-answer questions as in the pre-tutorial questionnaire focused on measuring how much the participants have learned while completing the tutorial compared to their prior knowledge.

The final section asks the participants to complete the System Usability Scale survey in order to measure the usability of the tool.

6.4 System Usability Scale

The System Usability Scale (SUS) [7] has been proven to be a reliable, industry-standard tool used to evaluate the usability of a system [3]. It consists of a 10-item questionnaire with five response options for respondents; from ”Strongly agree” to ”Strongly disagree.” The SUS provides a quick and effective way to measure the perceived ease of use and learnability of a system. It generates a single number representing a composite measure of the overall usability. This number helps in comparing different systems or versions of a system and tracking improvements over time. The SUS is particularly useful because it is technology-agnostic, user-friendly, and can be applied to a wide variety of systems, making it a versatile tool in usability testing and user experience research.

The SUS survey was presented to each participant as part of the post-tutorial questionnaire right at the end. This way the participants have filled out the survey immediately after interacting with the tool and finishing all the exercises. The SUS survey can be seen in Appendix G. Figure 6.1 shows the scale for results interpretation.

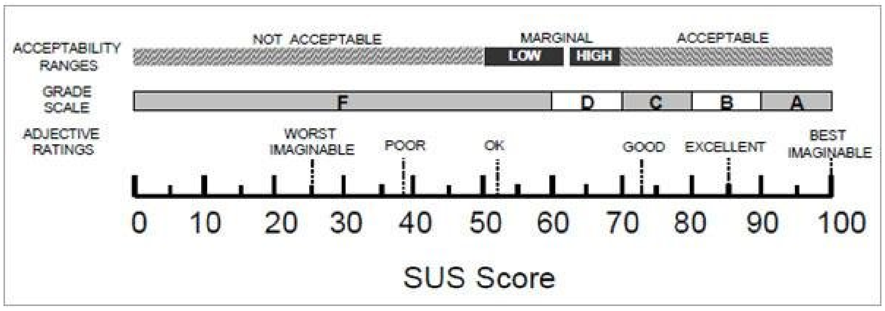


Figure 6.1: System Usability Scale

6.5 Summary

In this chapter we outlined the various methodologies used to evaluate the tool, which are used during Sprint One and Two in Chapters 7 and 8 respectively. We described the pre- and post-lab questionnaire designed for analysing the progress of participants’ confidence in conducting XSS attacks, and their prior and posterior knowledge of XSS topics, and finally gathered qualitative feedback and suggestions. We talked about Think Aloud sessions, a popular human-computer interaction evaluation method where participants interact with the tool and vocalise all their thoughts. Finally, we described the System Usability Scale, a simple but effective method that measures the usability and learnability of a system.

Chapter 7

Sprint One

For Sprint One, requirements were gathered as described in Chapter 4. Based on the design philosophy, the type of tool was decided to be a web application and the appropriate technologies were chosen for this task in Chapter 5. The evaluation methodologies used in this Sprint were described in Chapter 6. Appendix A contains additional images from the web application.

7.1 Development Phase

7.1.1 Interface Layout

The interface layout follows a standard web application theme and mimics a combination of a blog and tutorial, consisting of four main parts:

Navbar: The navbar at the top which can be used to navigate between pages and is always present.

Left Sidebar: The sidebar on the left can be used to navigate through a specific page to different chapters.

Right Sidebar: The sidebar on the right provides free space for possible future additions. Currently, it contains Gen Z Mode, a voluntary feature discussed in Chapter 9.

Main Content: The main content is in the middle. It is structured as a linear blog and tutorial that provides background information about XSS exploits, two exercises focusing on reflected and stored XSS exploits, and information on how to defend against them.

7.1.2 Background Introduction

The first part of the tutorial contains four sections that prepare the user for conducting reflected and stored XSS exploits as part of two exercises. It provides a summary of all the background information about XSS exploits and serves as a refresher for students who do not remember all the details about XSS, as well as a proper introduction to XSS for those with little to no knowledge about it.

Cross-Site Scripting (XSS): This section summarises all the general information about XSS.

Introduction: This section introduces the user to the tutorial and explains what are the main contents and what to expect from the tutorial.

Types of XSS attacks: This section introduces, compares and summarises the differences between the two most common XSS attacks, reflected and stored.

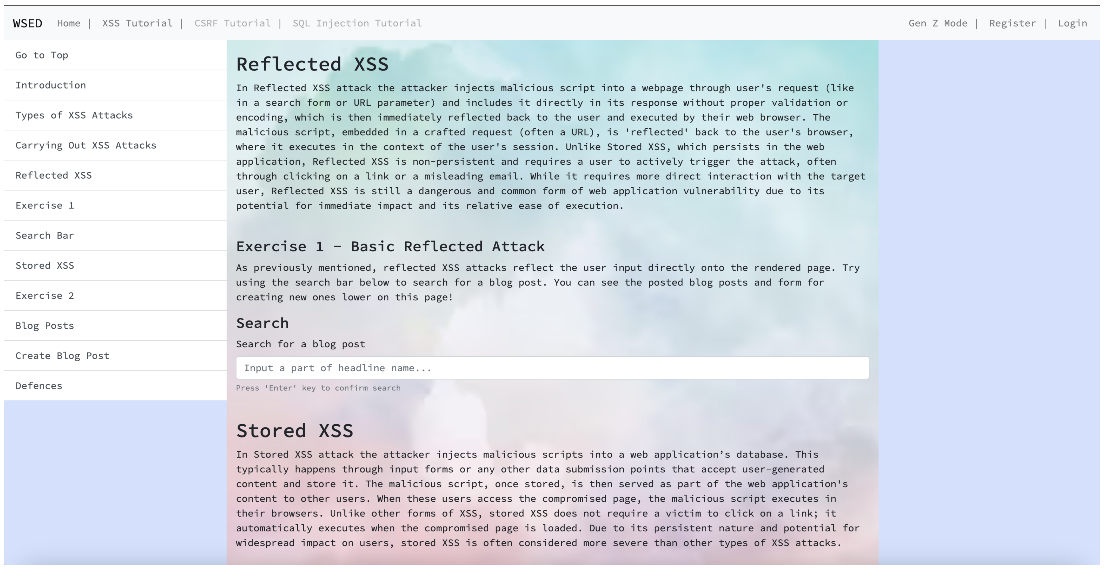
How to carry out XSS attacks: This section outlines three steps the user should take when trying to conduct an XSS attack, from answering the initial question of ”What am I trying to achieve?”, through conducting reconnaissance of the web application, to crafting the malicious injection code into the vulnerable entry point.

7.1.3 Exercises

After the tutorial introduction, two exercises follow. The first exercise focuses on performing a basic reflected XSS attack. The second exercise builds on the knowledge of the first exercise and focuses on performing a basic stored XSS attack. Alongside each exercise, there is a detailed description of the specific attack type associated with the exercise.

7.1.3.1 Exercise 1

In the first exercise, the user is guided to use a search bar in order to search for an arbitrary blog post (see Figure 7.1).



Reflected XSS

In Reflected XSS attack the attacker injects malicious script into a webpage through user's request (like in a search form or URL parameter) and includes it directly in its response without proper validation or encoding, which is then immediately reflected back to the user and executed by their web browser. The malicious script, embedded in a crafted request (often a URL), is 'reflected' back to the user's browser, where it executes in the context of the user's session. Unlike Stored XSS, which persists in the web application, Reflected XSS is non-persistent and requires a user to actively trigger the attack, often through clicking on a link or a misleading email. While it requires more direct interaction with the target user, Reflected XSS is still a dangerous and common form of web application vulnerability due to its potential for immediate impact and its relative ease of execution.

Exercise 1 - Basic Reflected Attack

As previously mentioned, reflected XSS attacks reflect the user input directly onto the rendered page. Try using the search bar below to search for a blog post. You can see the posted blog posts and form for creating new ones lower on this page!

Search

Search for a blog post

Input a part of headline name...

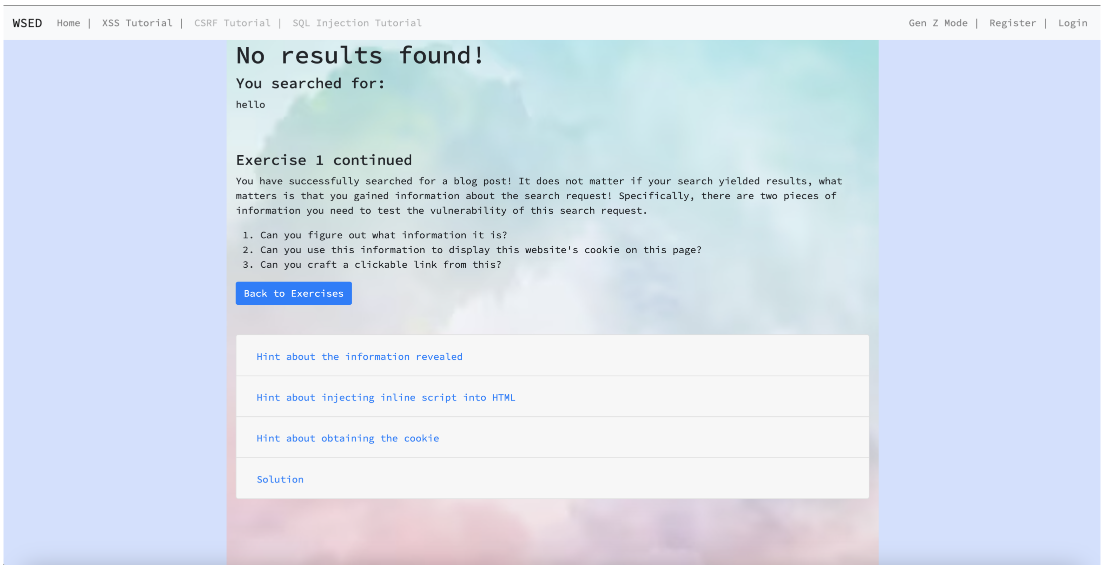
Press 'Enter' key to confirm search

Stored XSS

In Stored XSS attack the attacker injects malicious scripts into a web application’s database. This typically happens through input forms or any other data submission points that accept user-generated content and store it. The malicious script, once stored, is then served as part of the web application's content to other users. When these users access the compromised page, the malicious script executes in their browsers. Unlike other forms of XSS, stored XSS does not require a victim to click on a link; it automatically executes when the compromised page is loaded. Due to its persistent nature and potential for widespread impact on users, stored XSS is often considered more severe than other types of XSS attacks.

Figure 7.1: Exercise 1 description of reflected XSS and the assignment

Searching usually uses a GET request with a user input parameter that is visible in the URL and the input is often rendered in the response page. Figure 7.2 shows the continuation of exercise 1 once the user searches for anything.


WSED
Home | XSS Tutorial | CSRF Tutorial | SQL Injection Tutorial
Gen Z Mode | Register | Login
Back to Exercises
Hint about the information revealed
Hint about injecting inline script into HTML
Hint about obtaining the cookie
Solution

No results found!

You searched for:

hello

Exercise 1 continued

You have successfully searched for a blog post! It does not matter if your search yielded results, what matters is that you gained information about the search request! Specifically, there are two pieces of information you need to test the vulnerability of this search request.

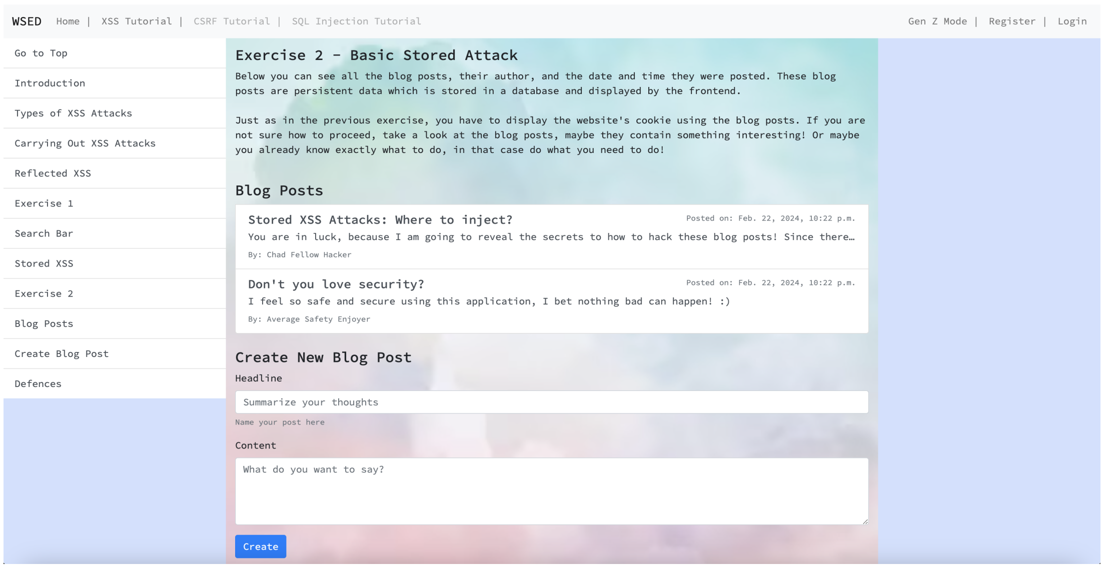
1. Can you figure out what information it is?
2. Can you use this information to display this website's cookie on this page?
3. Can you craft a clickable link from this?

Figure 7.2: Exercise 1 continued

Here the user is instructed to figure out the information needed to exploit the vulnerability and then execute it to display the browser's cookie information. If the user gets stuck, there are multiple hints available below, as well as the solution. Since the purpose of this tool is not to assess the user's knowledge but to teach the user, there is no method of validating the user's solution - they can check it against the provided solution and continue freely to the second exercise regardless of the correctness of their answer.

7.1.3.2 Exercise 2

In the second exercise, the user is instructed to exploit a vulnerability in the blog posts section and display the browser's cookie information again (see Figure 7.3).



Exercise 2 - Basic Stored Attack

Below you can see all the blog posts, their author, and the date and time they were posted. These blog posts are persistent data which is stored in a database and displayed by the frontend.

Just as in the previous exercise, you have to display the website's cookie using the blog posts. If you are not sure how to proceed, take a look at the blog posts, maybe they contain something interesting! Or maybe you already know exactly what to do, in that case do what you need to do!

Blog Posts

Stored XSS Attacks: Where to inject? Posted on: Feb. 22, 2024, 10:22 p.m.
You are in luck, because I am going to reveal the secrets to how to hack these blog posts! Since there…
By: Chad Fellow Hacker

Don't you love security? Posted on: Feb. 22, 2024, 10:22 p.m.
I feel so safe and secure using this application, I bet nothing bad can happen! :)
By: Average Safety Enjoyer

Create New Blog Post

Headline

Summarize your thoughts

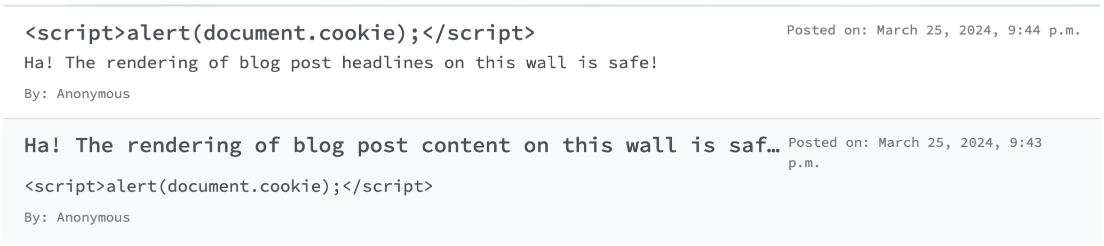
Name your post here

Content

What do you want to say?

Figure 7.3: Exercise 2 - existing blog posts and form for creation of blog posts

There are two pre-made blog posts, the first one serves as guidance, while the other is added there purely for immersion. The first post hints at trying to insert malicious scripts into the content field of the blog post. This however does not trigger the script and instead, the headline field is rewritten as a response to signalise that XSS encoding works when rendering the blog posts wall. However, if the user clicks on the blog post with malicious script in the content field, the detail of the blog post is dynamically rewritten to inform the user that the content field in the detail view of the blog post is protected as well and that they should try something else, hinting at inserting it into the headline. The same dynamic rewriting of blog posts happens when a malicious script is inserted into the headline input field (see Figure 7.4). Once the user clicks on the post with a malicious script in the headline, the attack is triggered, because the headline rendering of the blog post detail page is vulnerable.



```
<script>alert(document.cookie);</script>                    Posted on: March 25, 2024, 9:44 p.m.
Ha! The rendering of blog post headlines on this wall is safe!
By: Anonymous

Ha! The rendering of blog post content on this wall is saf…  Posted on: March 25, 2024, 9:43
                                                              p.m.
<script>alert(document.cookie);</script>
By: Anonymous
```

Figure 7.4: Dynamically rewritten blog posts when a malicious script is detected

The malicious script detection is done by checking if the input string contains <script> and </script> as this exercise assumes the user will use the solution from the previous exercise.

7.1.4 Defences

The final section of the tutorial contains information about the common ways of defending against XSS attacks. These include the methods described in Section 2.2.3. After reading through this section, the user has completed the XSS tutorial.

7.2 Evaluation Phase

7.2.1 Think Aloud Sessions

Following the design and implementation phase of Sprint One, the first set of evaluations was conducted with three participants, each one completing first the pre-tutorial questionnaire, then the Think Aloud session, and finally the post-tutorial questionnaire and System Usability Scale survey. Two of the participants fit the desired persona well and one participant represented an edge case as they had never taken any computer security-related course.

7.2.1.1 Participant 1

The first participant was a Masters student with a very limited knowledge of computer security and no knowledge of XSS, only vague knowledge of SQL injections, representing an edge case in these evaluations. As they started the Think Aloud session, they read thoroughly through the background introduction of XSS, nodding to themselves in understanding, and proceeding to correctly use parallels between XSS and SQL injections to fortify their understanding. Once they reached the section with the reflected XSS exercise, they read the assignment and immediately proceeded by using the search bar. However, once they reached the search results and read Exercise 1 continued, instead of attempting to perform the attack, they merely explained correctly how to do it and ignored the hints section. This unexpected behaviour could be attributed to the wording of the assignment, which was formulated as a ”Can you...?” question rather than an assignment. Then they returned to the main page and proceeded to the second, stored XSS exercise, with which they struggled, as they did not finish the first. After returning to the first exercise, they used the provided solution to attempt the second exercise, but injected only into the content field, which did not trigger the script. They assumed there was no vulnerability and proceeded to the defences section without injecting the headline. In this section, they struggled with understanding the difference between output encoding and input sanitisation, and what whitelisting is.

7.2.1.2 Participant 2

The second participant was a third-year student who took the Computer Security course and had no applied experience with XSS attacks, fitting our persona perfectly. They read the background introduction thoroughly and liked how it was written. It served as a good refresher, and the paragraphs about how to perform XSS attacks and threat comparison between reflected and stored types were interesting to them and less familiar from class. They completed Exercise 1 smoothly, with minor issues such as not knowing how to get the browser’s cookie and how to display values using JavaScript. They used a hint for the cookie, but had to search for the ”alert” function online. They then proceeded with Exercise 2 exactly as it was intended and completed it with no problems. They valued the guidance provided in the blog posts and found them entertaining. Reading about the defences, they appreciated the clarification of output encoding and input sanitisation, as they thought it was essentially the same. They liked the examples and explanations of Content Security Policy and HttpOnly.

7.2.1.3 Participant 3

The third participant was a fourth-year student who took the Computer Security course and had no applied experience with XSS attacks, fitting our persona closely. They read the background information promptly, seemingly understanding it all. When completing Exercise 1, they first explained how they would perform it before attempting it. However, since they could not recall much of JavaScript and wanted to be sure their steps were correct, they checked all hints before successfully completing the exercise. In Exercise 2, they went through the exercise as intended. However, when creating a blog post, they wrote a string starting with ”Ha!...” which is exactly the same start of the dynamically rewritten headline string as a response to the detection of a malicious script. Since the rewritten text was black, they did not notice it and were confused, which slowed down their progress. Eventually, they noticed and realised they should insert the script into the headline, successfully completing the exercise. However, due to the lack of visual feedback, they were briefly confused if they truly completed it. Despite this, they enjoyed the tutorial, and appreciated the blog posts and found them amusing, just as Participant 2. Finally, they went through the defences section, where everything made sense to them.

7.2.2 Sprint One Evaluation Conclusion

The three evaluation sessions highlighted a lot of positive aspects of the first version of the tool, but uncovered many issues that should be addressed during Sprint Two. Overall all three participants enjoyed using the tool. Table 7.1 shows the main positive and negative aspects of the tool uncovered from observations of the participants and their feedback.

Positive Aspects	**Negative Aspects**
• Informative background information • Well structured • Easy to navigate • Useful hints • Interactive and enjoyable • Informative defences section • Learning beyond scope of the Computer Security course • Logical transition between exercises • Entertaining blog posts	• No feedback on completion or failure of exercises • All content unlocked from start causing confusion • Same assignment for both exercises • Lack of visual feedback to user actions • Lack of JavaScript hints • Some unclear formulations • Lack of explanations

Table 7.1: Positive and negative aspects uncovered during Sprint One evaluation

Table 7.2 shows the summary of statistics gathered from evaluation of Sprint One. Time is given in minutes and measures how long it took the participants to complete the tasks given to them for the Think Aloud session. Pre and Post scores measure the amount of the eight open-answer questions about XSS from the pre and post-questionnaires that were answered correctly. Points were awarded as 0, 0.5, or 1 per question, depending if the answer was incorrect, partially correct, or fully correct, respectively. Pre conf. and Post conf. measure the participants' confidence in performing XSS attacks before and after using the tool, where 1 signals 'No confidence' and 10 signals 'Very high confidence'. SUS shows the System Usability Scale scores awarded.

We can see from the results that all participants improved their Post-score compared to Pre-score on average by 5.17 points, showing their learning progress after using the tool. Furthermore, the two participants closely fitting the persona completed both exercises without major problems. The edge case participant struggled a bit, but eventually managed it as well, highlighting the tool's accessibility beyond defined persona. This is also shown by the improvement of the Pre conf. and Post conf. scores, with an average confidence in conducting XSS attacks improvement of 3.33 points. Finally, the tool has achieved an average SUS score of 87.5, placing it into the highest, 'Excellent' tier. The difference in given SUS scores between participants two and three, and participant one, could be attributed to the difference in prior understanding of XSS, affecting their experience, as the lowest score was given by the edge case participant, while the higher scores were given by participants closely fitting the desired persona.

Participant	Year	Time	Pre score	Post score	Pre conf.	Post conf.	SUS
P1	5	49	0.5	7.0	2	5	75
P2	3	33	4.5	7.5	3	6	92.5
P3	4	25	1.0	7.0	3	7	95

Table 7.2: Summary of statistics from Sprint One evaluation

To conclude the evaluation of Sprint One, we will relate the results to the requirements defined in Section 4.4.

Requirement 1: The tool shall teach our persona XSS attacks taught in the Computer Security course - reflected and stored

Two participants completed the attacks without major problems. The edge case participant managed it with some struggles as well, with all participants improving their confidence in conducting XSS attacks substantially. As such I would consider this requirement fulfilled.

Requirement 2: The tool shall provide a theoretical background of Cross-Site Scripting

The tool contains not only background taught in the Computer Security course, but additional information beyond its scope. Furthermore, the background information section has achieved praise from the participants, and all participants saw substantial improvements in the post-tutorial questionnaire, which is why I would regard this requirement fulfilled.

Requirement 3: The tool shall provide step-by-step guidance for performing XSS attacks

The tool contains step-by-step guidance in the assignments and dynamically rewritten content, however, there is space for improvement. The wording was occasionally ambiguous and its visibility was often low, which is why I would consider this requirement partially fulfilled.

Requirement 4: The tool shall provide hints, solutions, and explanations

The Exercise 1 contains many hints and a solution, and Exercise 2 contains hints combined with guidance in the form of default blog posts. However, the results suggest there could be more hints present, especially about the usage of JavaScript, and more explanations about the usefulness and applications of the attacks. As such, I would consider this requirement partially fulfilled.

Requirement 5: The tool shall adhere to usability heuristics highlighted in section 4.3.4

The **Visibility of system status** heuristic could be improved, as it was a common feedback among the participants. The **User control and freedom** heuristic could be improved, as there was one moment when Participants 1 felt forced to go back to Exercise 1 to find the solution for Exercise 2. The **Recognition rather than recall** heuristic was fulfilled, as the information and assignments required to complete the exercises were present in designated places accordingly with no need for the participants to switch between different pages or tabs. The **Aesthetic and minimalist** heuristic design was achieved by having a clean, easy-to-use and easy-to-navigate user interface, which was recognised by the participants and reflected in high SUS scores. The **Match between system and the real world** heuristic was achieved by the tool mimicking a blog post website, which is a very common and known type of web application, and creating existing default blog posts to immerse the participants in the tool. The **Help and documentation** heuristic was achieved by having an extensive background information section that allowed even participants with no prior XSS knowledge to finish the exercises.

Requirement 6: The tool shall contain aspects of gamification and CTFs

The tool contained aspects of CTFs in the form of having to retrieve a specific value by exploiting the system, but it could be improved by having the participant to submit and validate the value. It also had aspects of gamification in the form of level difficulty scaling, however, there was still a lot of space for improvement.

7.3 Summary

We described the initial development results achieved during Sprint One. The web application was designed to work as the source of information and the vulnerable target at the same time, showcasing the user interface layout and the sections contained in the XSS tutorial. We then conducted the first set of evaluations. The participants managed to complete the exercises, increased their confidence in performing XSS attacks, improved their theoretical knowledge of XSS, and enjoyed interacting with the tool, thus meeting the learning goals defined in Chapter 1. However, despite the positive feedback and all the positive aspects present in the application, not all system requirements were fully met and many negative aspects were uncovered. These will be used in the next Chapter 8 to refine the original requirements and improve the application.

Chapter 8

Sprint Two

After the development of Sprint One, an initial evaluation with three users was conducted. The findings and users' feedback were refined into a set of additional requirements that should be fulfilled during Sprint Two:

Requirement 7: The system shall lock progression of exercises until they are completed

Requirement 8: The system shall have different goals for each exercise

Requirement 9: The system shall provide better visual feedback

Requirement 10: The system shall provide more JavaScript-specific hints

8.1 Development Phase

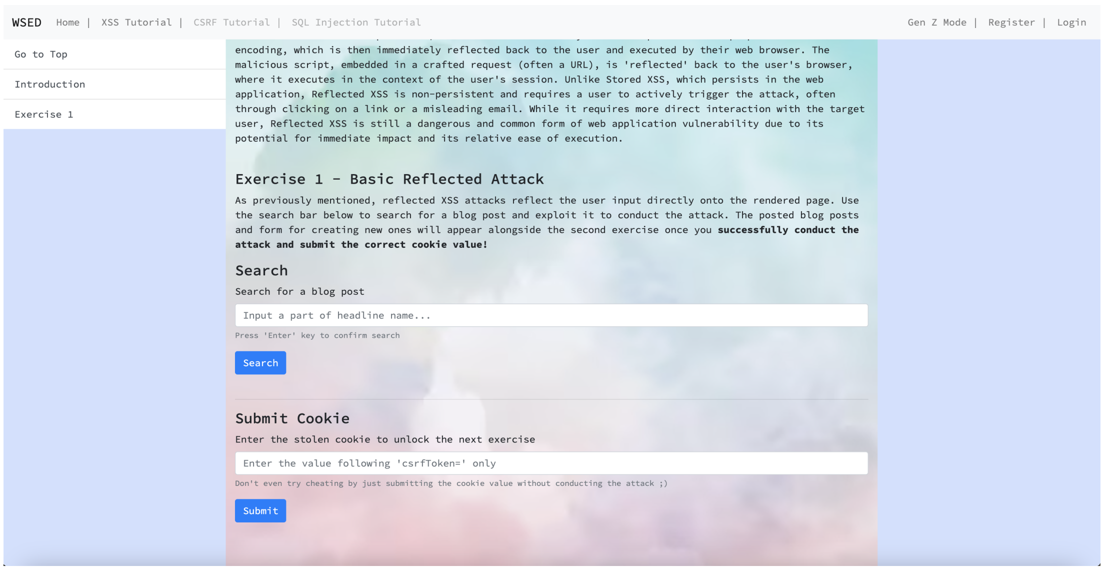
8.1.1 Requirement 7 Improvements

After Sprint One, there were no means for measuring progression, because both exercise and defence sections were unlocked from the start, allowing the users to proceed however they want, potentially confusing them regarding the completion of exercises and overwhelming them with all the content at the same time.

To improve this, during Sprint Two, I implemented a system that would require the user to complete an exercise in order for them to progress and see further content. This was, however, a difficult task, as it required the system to check if the user performed an XSS attack. Furthermore, recall that the overall goal of this tool is to **teach** the user how to perform XSS attacks, not **assess** them on their ability to do so. If the goal was to **assess** the users, the detection method could check the users' malicious input against a small set of viable solutions. However, since the goal is to **teach** the user how to conduct XSS attacks, a vast range of solutions is acceptable. Figure 8.1 shows the improved version of Exercise 1.

In order to tackle this problem, I decided that the method for detecting the completion of an exercise will validate two things. The first is the direct submission of some value, for example, the browser's cookie for Exercise 1. This check is straightforward, however, the problem is that the user could simply obtain the value by calling "document.cookie" in the browser's console, bypassing the exercise. To avoid this shortcut, I implemented a second check, which uses regex to detect if the input contains HTML tags, attributes or PHP code commonly used in XSS attacks. While this regex is not perfect (if it were, XSS vulnerabilities would cease to exist), its knowledge domain covers a wide range of solutions, much wider than what is covered by the Computer Security course and what is expected of the target audience.

This detection method used for unlocking progression ensures that the user will perform the XSS attacks and have a clear goal of retrieving specific value, rather than merely injecting "alert(hello);". The limitation of this approach is that it is still possible to inject a malicious script that does not uncover the desired value and the user then retrieves the cookie value manually, still passing the detection check. The most important thing is, however, that the user was able to conduct XSS attacks, and thus achieving the learning goal of this tool.


WSED
Home | XSS Tutorial | CSRF Tutorial | SQL Injection Tutorial
Gen Z Mode | Register | Login
Go to Top
Introduction
Exercise 1
Input a part of headline name...
Search
Enter the value following 'csrfToken=' only
Submit

encoding, which is then immediately reflected back to the user and executed by their web browser. The malicious script, embedded in a crafted request (often a URL), is 'reflected' back to the user's browser, where it executes in the context of the user's session. Unlike Stored XSS, which persists in the web application, Reflected XSS is non-persistent and requires a user to actively trigger the attack, often through clicking on a link or a misleading email. While it requires more direct interaction with the target user, Reflected XSS is still a dangerous and common form of web application vulnerability due to its potential for immediate impact and its relative ease of execution.

Exercise 1 - Basic Reflected Attack

As previously mentioned, reflected XSS attacks reflect the user input directly onto the rendered page. Use the search bar below to search for a blog post and exploit it to conduct the attack. The posted blog posts and form for creating new ones will appear alongside the second exercise once you **successfully conduct the attack and submit the correct cookie value!**

Search

Search for a blog post

Press 'Enter' key to confirm search

Submit Cookie

Enter the stolen cookie to unlock the next exercise

Don't even try cheating by just submitting the cookie value without conducting the attack ;)

Figure 8.1: Exercise 1 with submission box required for progression

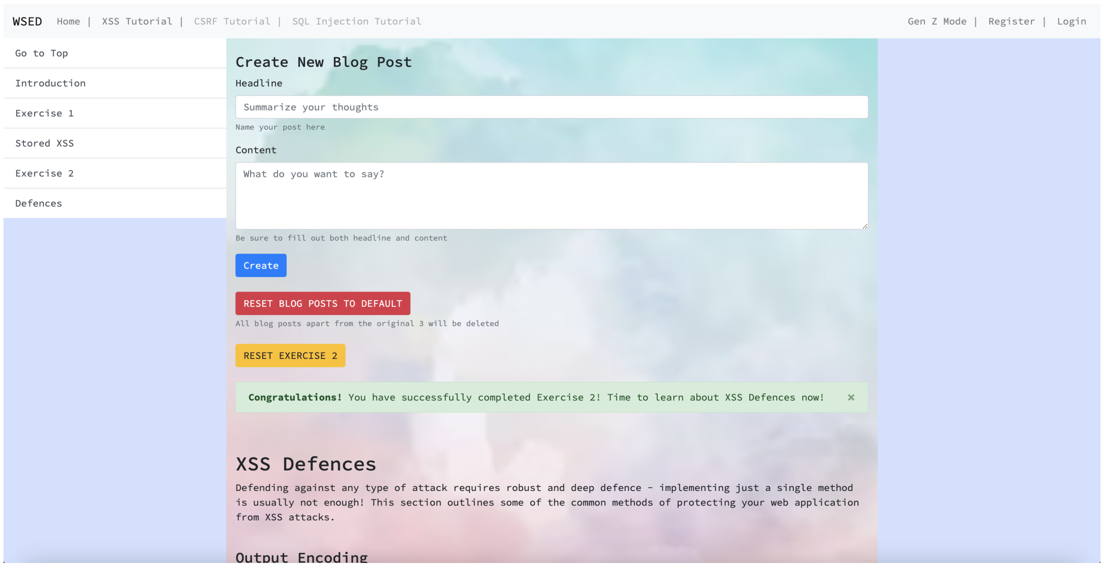
8.1.2 Requirement 8 Improvements

Both Exercise 1 and 2 had the same assignment and solution - the user was required to inject code that reveals the browser's cookie. This was, however, repetitive and did not present much of a new challenge after completing Exercise 1. It was also counterproductive for the new progression approach, because the user could copy the value from Exercise 1 and then inject any malicious code without attempting to complete Exercise 2 properly.

As such, the goal of Exercise 2 was changed to obtain a secret message contained in an HTML "div" element, hidden in the detail page of every blog post. The intended flow would be to scout the website by inspecting the page elements and once the hidden element was uncovered, the user would craft malicious code to access its value. This approach added variability to the exercises and showed the users the importance of not storing any sensitive information in the client itself. It shares the same limitation with Exercise 1 in terms of bypassing the exercise..

8.1.3 Requirement 9 Improvements

Exercise 1 did not provide any visual feedback apart from whatever the injected malicious script caused to happen, and the completion validation of the exercise had to be done by the user by checking that their injected code matches the given solution. The solution to Exercise 2 was the same as Exercise 1, with the only difference being that it focused on a stored XSS attack with two input fields and four places for output rendering. Additionally, a dynamic message rewriting was used for simple detection that checked if users' input string contains <script> and </script> to determine if an attack was performed. The new changes made use of the new XSS attack detection method introduced under Requirement 7. It displayed a success message if the user has successfully completed the exercise by passing both detection checks, and a failure message otherwise (see Figure 8.2). Furthermore, the dynamic rewriting of blog posts in Exercise 2 changed the colour either to green or red depending on the positive or negative tone of the message respectively (see Figure 8.3).


WSED
Home | XSS Tutorial | CSRF Tutorial | SQL Injection Tutorial
Gen Z Mode | Register | Login
Go to Top
Introduction
Exercise 1
Stored XSS
Exercise 2
Defences
Create
RESET BLOG POSTS TO DEFAULT
RESET EXERCISE 2

Create New Blog Post

Headline

Summarize your thoughts

Name your post here

Content

What do you want to say?

Be sure to fill out both headline and content

All blog posts apart from the original 3 will be deleted

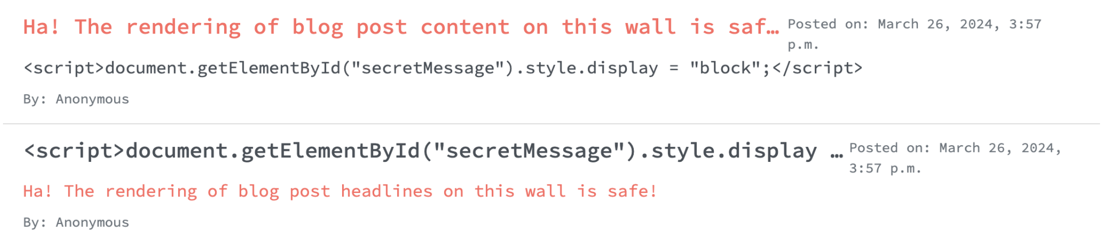
Congratulations! You have successfully completed Exercise 2! Time to learn about XSS Defences now! ×

XSS Defences

Defending against any type of attack requires robust and deep defence - implementing just a single method is usually not enough! This section outlines some of the common methods of protecting your web application from XSS attacks.

Output Encoding

Figure 8.2: Exercise 2 success feedback, progress unlocked, and example reset buttons



```
Ha! The rendering of blog post content on this wall is saf…   Posted on: March 26, 2024, 3:57
                                                                p.m.
<script>document.getElementById("secretMessage").style.display = "block";</script>
By: Anonymous

<script>document.getElementById("secretMessage").style.display …   Posted on: March 26, 2024,
                                                                     3:57 p.m.
Ha! The rendering of blog post headlines on this wall is safe!
By: Anonymous
```

Figure 8.3: Dynamic blog post rewriting with additional visual feedback

8.1.4 Requirement 10 Improvements

While there were many hints present, especially in Exercise 1, they did not cover everything required to complete it. To improve the flow of the exercise, I added a new hint on how to alert and log values into the console using JavaScript. In Exercise 2, I added a new default blog post which included two hints about accessing HTML elements and manipulating their visibility, as well as the solution to uncovering the secret message from Requirement 8.

8.1.5 General Improvements

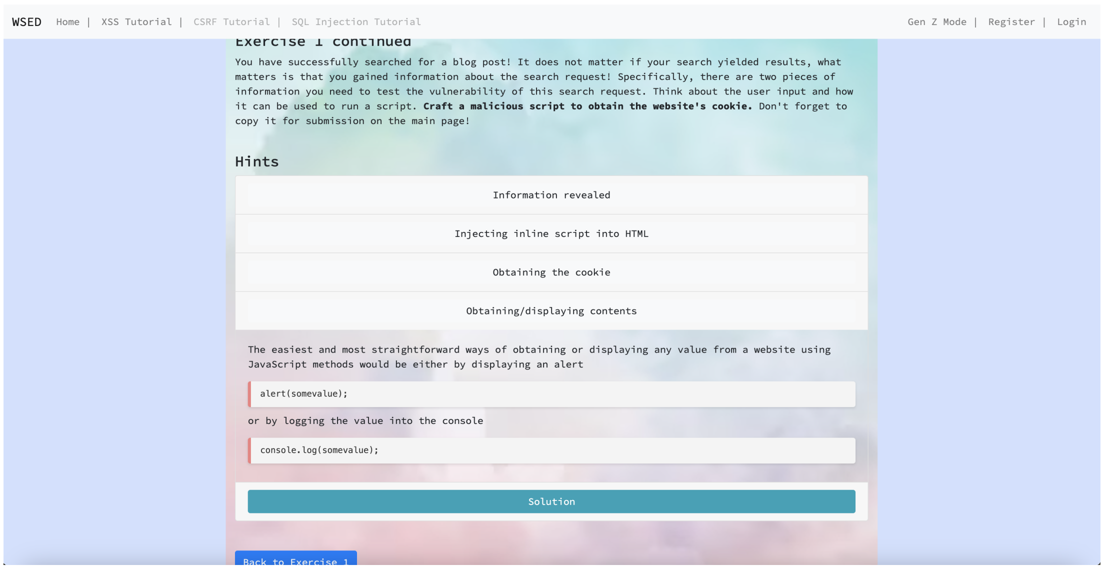
The four requirements highlight some of the major changes and improvements done during Sprint Two, but there are many minor improvements implemented as well.

Better Wording - Some sentences were phrased poorly in Sprint One and needed to be refactored. Some paragraphs were rewritten to properly reflect the changes to assignments and solutions.

Left Sidebar - The sidebar on the left was too cluttered with unnecessary links, so some of the links were removed and only the main checkpoints were kept.

Text Highlighting - The most important words, sentences and phrases were made bold in order to allow more experienced users to skip unnecessary text.

Hints UI - The hints were changed to properly separate themselves from the assignment in Exercise 1 and from the blog post text in Exercise 2. A headline was added to the division and "Back to Exercise 1" button was placed below the hints, rather than above, for a more natural layout (see Figure 8.4).


WSED
Home | XSS Tutorial | CSRF Tutorial | SQL Injection Tutorial
Gen Z Mode | Register | Login
Hints
Information revealed
Injecting inline script into HTML
Obtaining the cookie
Obtaining/displaying contents
Solution
Back to Exercise 1

Exercise 1 continued

You have successfully searched for a blog post! It does not matter if your search yielded results, what matters is that you gained information about the search request! Specifically, there are two pieces of information you need to test the vulnerability of this search request. Think about the user input and how it can be used to run a script. **Craft a malicious script to obtain the website's cookie.** Don't forget to copy it for submission on the main page!

The easiest and most straightforward ways of obtaining or displaying any value from a website using JavaScript methods would be either by displaying an alert

```
alert(somevalue);
```

or by logging the value into the console

```
console.log(somevalue);
```

Figure 8.4: Exercise 1 continued with improved UI and new hint

Explanations - Further explanations and reasoning behind why and how the attacks might be useful in the real world were added. Additional explanation about which defences are considered more effective were incorporated, and more reliable resource links to OWASP as well.

Exercise Resetting - Options to reset exercises were added in case the user wanted to redo them, making it possible by clicking on a single button rather than rebuilding the entire project.

Blog Posts Resetting - A feature was introduced to revert all blog posts to their default state by removing any user-created content.

Finish Page - A large finish button was added at the end of the tutorial, which redirects the user to a Thank You finish page and provides the user with a proper closure.

8.2 Evaluation Phase

8.2.1 Think Aloud Sessions

Following the development phase of Sprint Two, the second set of evaluations was conducted with another three, different participants. Each participant completed first the pre-tutorial questionnaire, followed by the Think Aloud session, and finally filled out the post-tutorial questionnaire and System Usability Scale survey. One participant was a close fit to the desired persona, while the other two were edge cases, as both of them had a lot of experience with applied computer security.

8.2.1.1 Participant 4

The fourth participant was a fourth-year student who did not take the Computer Security course, but worked as a penetration tester intern. As such, they had experience with a wide variety of computer security topics, both in theoretical and applied form, making them an edge case for this study. Their experience was, however, not the only reason why they were an edge case. Rather than reading or skimming through the background information, they skipped everything and went straight to Exercise 1. They skipped their assignment as well. They immediately figured out that they should insert malicious code into the search bar and proceeded to embed an HTML button tag instead of a script tag, which, upon being clicked, triggered an alert. The button worked, but since they did not read the assignment, they had to go back and reread it to figure out what to do. They could not remember how to obtain the browser's cookie and instead of using the hints, they played with Chrome development tools for ten minutes and searched the internet, finally figuring it out and successfully completing the exercise. Proceeding to Exercise 2, they did not read anything again, but they clicked through the blog posts, where they skimmed the text. They used the advice from the blog posts to try to insert malicious code into the content field, however, they used the HTML link tag, which was not covered by the detection mechanism. This error was fixed immediately after the session, before the next Think Aloud session. The dynamic rewriting of blog posts not being triggered confused them. They assumed it had to do something with encoding of special characters and tried searching Portswigger cheat sheets for help, with no success. Eventually they decided to use the HTML script tag, which triggered the dynamic rewriting and they managed to complete the exercise using the provided guidance. After completing Exercise 2, they ignored the defences section and completed the tutorial.

8.2.1.2 Participant 5

The fifth participant was a fourth-year student who took the Computer Security course, but had no applied experience and forgot most of XSS theory. While they fit the desired persona closely, their behaviour was quite unconventional. They fully read the background information without much reaction. They read the assignment of Exercise 1, and proceeded to explain how they could use the search bar for the attack before using it to search for an arbitrary word. Once they reached the Exercise 1 continued page, instead of attempting to solve the exercise on their own, they read through all the hints and used the given solution to complete the exercise. Proceeding to Exercise 2, they read everything carefully, but with seemingly no reaction. At this point it seemed as if they were reading everything without trying to understand it. During Exercise 2, they read through all the blog posts, and then through all the hints and the solution. They inserted the given solution into the content field, which was hinted by the first default blog post. The script was not triggered, so they tried inserting it into the headline field, which did not trigger the script, since the blog post wall was protected. They did not try clicking on the blog posts to go to their detail page. Thinking it is not working, they continued to try to insert new blog posts. After a couple minutes, they decided to click on the blog post with malicious headline, which triggered the script when rendering the detail page. They were confused why it happened this way even after reading the explanation. I explained it in detail to them after the evaluation was done, but they did not seem to understand my detailed explanation either. They went through the defences section without much reaction and completed the tutorial.

8.2.1.3 Participant 6

The sixth participant was a fourth-year student who took the Computer Security course and had some applied experience with web security, since they worked on a similar project as I did. They skimmed through the background introduction section, suggesting it would be nice to have a paragraph summarising the most important information for more experienced users, instead of having to read through it all. They went through Exercise 1 without any problems and proceeded straight to Exercise 2. They went through the second exercise as it was intended, using a malicious script as the blog post headline that merely caused an alert instead of grabbing the correct value, to explore the vulnerability. They saw that they have successfully exploited it and then they found the secret message in the detail of their blog post using the Chrome development tools. Instead of changing the script to uncover the value, they copied and submitted the value. Since the value was correct and they triggered an XSS attack, even though it was not the intended flow, they completed the exercise. Proceeding to the defences section, they were very impressed by it as they learned a lot from the HttpOnly and Content Security Policy sections. They took a photo of the HttpOnly section with my permission, finishing the tutorial.

8.2.2 Sprint Two Evaluation Conclusion

The further three evaluation sessions did not uncover any major usability issues, however, they collected suggestions for improving the tool in the future. All three participants stated that they enjoyed using the tool. Table 8.1 shows some aspects that could be improved upon in the future, based on the observations and feedback from the participants.

Improvement Suggestions
• Have a summary paragraph or ”Too Long Didn’t Read” mode of text for more experienced users • Make the two exercises more realistic and connected • Explain better the purpose of the browser’s cookie • Explain better if combination of all XSS defences prevents all XSS exploits

Table 8.1: Improvement suggestions observed and gathered from Sprint Two evaluation

Table 8.2 shows the summary of statistics gathered from the evaluation of Sprint Two. The legend is exactly the same as in Table 7.2. We can see that the edge case participant four took the longest time to finish the tutorial compared to the other two. Their SUS score was the lowest of the three, following the pattern of the edge case participant from Sprint One evaluation. We can see that their Pre and Post score have not improved, which was expected considering they barely read anything in the tutorial. Their confidence in conducting XSS attacks has not improved as well. On the other hand, we can see a substantial improvement in both pre and post-questionnaire and confidence scores for participant five, as well as a high SUS score, showing that despite using all the hints and solutions, they still learned a lot and their confidence in conducting XSS attacks increased greatly. The final participant had a lot of knowledge and experience before attempting the tutorial, which can be seen in their pre questionnaire and confidence scores. Even though both participants four and six had prior experience, participant six read the tutorial and assignments and thus was capable of achieving a perfect score in the post-questionnaire, improving their already high confidence in performing XSS attacks.

Participant	Year	Time	Pre score	Post score	Pre conf.	Post conf.	SUS
P4	4	42	6.5	6.5	7	7	77.5
P5	4	36	0.0	5.5	1	7	85
P6	4	30	6.0	8.0	6	7	85

Table 8.2: Summary of statistics from Sprint Two evaluation

To conclude the evaluation, we will relate the results to the original requirements defined in Section 4.4 which were not fully achieved during Sprint One, and to the newly refined requirements in Sprint Two.

Requirement 3: The tool shall provide step-by-step guidance for performing XSS attacks

During Sprint Two the wording of certain paragraphs and assignments regarding the steps to complete the exercises was adjusted to clearly reflect the goals of the exercises, and the dynamic rewriting of blog posts provided better visual feedback. As such, I would consider this requirement fulfilled.

Requirement 4: The tool shall provide hints, solutions, and explanations

During Sprint Two an extra hint regarding JavaScript was added to Exercise 1, making it possible to solve it by following all the hints (as seen with participant five). A third default blog post containing two hints and a solution was added to Exercise 2, making it possible to solve it using only the hints, and finally further explanations regarding the solutions to the exercises were added. Overall I would consider this requirement and **Requirement 10: The system shall provide more JavaScript-specific hints** fulfilled as well.

Requirement 5: The tool shall adhere to usability heuristics highlighted in Section 4.3.4

The only unfulfilled heuristic after Sprint One was **Visibility of system status**. During Sprint Two, message banners indicating the success or failure of completing an exercise were added. Additionally, dynamic rewriting of blog posts was given specific colours for clearer indication, and finally many important words and phrases in the text overall were made bold to emphasise the most important information, rendering this requirement and **Requirement 9: The system shall provide better visual feedback** fulfilled.

Requirement 6: The tool shall contain aspects of gamification and CTFs

Gamification in the form of progression, content locked behind each exercise (level), and improved storytelling through new default blog post, which was added during Sprint Two. Additionally, the values needed to obtain were made different and had to be submitted and validated by the system in order to proceed, thus adding more aspects of CTFs and fulfilling this requirement, **Requirement 7: The system shall lock progression of exercises until they are completed** and **Requirement 8: The system shall have different goals for each exercise** as well.

8.3 Summary

In this chapter we defined a new set of requirements refined from the evaluation results of Sprint One. Then we proceeded to describe the implementation of the major changes associated with the new requirements, as well as other minor changes that improved the quality of life features and minor usability issues. We then conducted the second set of evaluations. While this set of participants did generally not fit the desired persona well, they still managed to complete the XSS tutorial and enjoyed using the application. However, their confidence of performing XSS attacks and their theoretical knowledge saw lower increases compared to Sprint One, which could be attributed to participants' diversion from the persona. The System Usability Scale scores were also lower in Sprint Two. Despite all this, the new requirements were met, making the original ones fulfilled as well. Appendix A contains additional images from the web application.

Chapter 9

Conclusions

9.1 Overview

The aim if this project was to create a tool that teaches inexperienced informatics students web application exploits and defences, however, there are many types of web application vulnerabilities and attacks associated with them and covering them all would not be feasible. To decide which vulnerability should be the focus of this application, a student questionnaire was conducted. This student questionnaire showed that Cross-Site Scripting (XSS) and Cross-Site Request Forgery (XSRF) attacks are considered to be the most difficult among the students. Since XSS is a much more common vulnerability than XSRF, XSS was chosen as the topic for this tool. Additionally, the student questionnaire was used to gather explicit requirements, and a cognitive walkthrough evaluation of Google's Gruyere was used to gather requirements about the usability goals of this tool. Requirements were then defined based on the results of these requirements gathering methods.

I then went through two Agile development phases called Sprints. Each Sprint consisted of requirements gathering, development and evaluation phase. During Sprint One the design philosophy was shaped by the initial requirements gathering. I then implemented the first version of the tool and ran first set of evaluations with three participants. The evaluation highlighted many positive and negative aspects of the tool, which were then used for requirements gathering for sprint two. During Sprint Two I aimed to fulfill the new requirements by improving the negative aspects, and then ran the second set of evaluations with another three participants.

To evaluate the tool, I ran Think Aloud sessions with users and I created a pre and post-tutorial questionnaires designed to quiz the users' knowledge before and after they interact with the tool. These evaluations were focused on assessing if the system requirements were met. Lastly, I conducted a System Usability Scale (SUS) survey after the sessions to assess the usability aspects of the tool.

Overall, all six participants enjoyed using the tool and all requirements stated in Section 4.4 and Chapter 8 were met within Sprint Two. The evaluation has shown that the closer the participant fits the desired persona, the higher they awarded the tool in the System

Usability Score survey. The pre and post-questionnaire results further highlighted the successful achievement of the tool's educational goals, generally showing great improvements among the participants. Finally, the average System Usability Scale score of 85 across all six participants places the usability of this tool into the ”Excellent” category, proving high usability and accessibility to inexperienced users.

In conclusion, based on the results and observations gathered, participants enjoyed using the tool, were able to perform XSS attacks, and generally showed improvement in their theoretical knowledge of XSS, as well as their confidence in conducting the exploits. As such, I believe that this tool has fulfilled its learning goals stated in Section 1.1.

9.2 Discussion and Future Work

Further evaluation with a wider range of participants could enrich the tool's development, as many participants fell outside the intended user persona, representing edge cases. Engaging experts, particularly Computer Security professors, for evaluation could offer valuable insights, given they form a secondary audience with the potential to leverage the tool for educational purposes. However, this aspect was not fully realised due to challenges in participant recruitment.

The tool currently teaches basic reflected and stored XSS attacks and theoretical defences. Expanding its scope to more advanced exercises including a broader array of attacks, like injection attacks and Cross-Site Request Forgery (CSRF), is feasible and could be facilitated by the modular design afforded by Django. Additionally, introducing practical defence exercises could further improve the learning experience, though this would necessitate either keeping the local deployment for direct code access or the development of online simulations for realistic defence applications.

Designing effective XSS exercises proved challenging, needing to balance between the exercise complexity and the users' immersion without overwhelming those less confident in JavaScript. Efforts were made to enhance engagement through dynamic feedback to user actions rather than through complex JavaScript requirements. Improving the storytelling and adding more gamification elements such as scenario-based exercises with more complex goals, requiring users to assume different identities, could further increase this immersion. Future improvements could also focus on integrating more advanced attack detection methods such as Server-Side Detection of XSS attacks [22], beyond the limited regex-based mechanism.

There were two features that may be seen in the figures, but were not mentioned in the report, because they were voluntary and did not play a role in the evaluations. However, they provide a basis for future work. The first feature are basic mechanisms for user registration that have been added in early in the development when it was unsure if the web application will be deployed online. While requiring user accounts could potentially deter some users, it would enable online deployment and improve progress tracking, shifting from a global variable approach to a more personalised tracking system. The use of Django and Docker underpins the tool's extensibility and ease of deployment, indicating a viable path forward for these enhancements.

The second feature is the Gen Z Mode, which has been briefly mentioned in Section 7.1 as being a part of the right sidebar. This feature was inspired by Generation Z's high consumption of fast-paced media through TikTok or Instagram Reels, where users have to watch a simple, fast-paced video on the side to a main activity they are doing, in order to not get bored or lose attention. This concept is called ”Sludge” and in my application it can be turned on to play in the sidebar. I implemented it on a whim in 20 minutes and the reason why I did not include it in the report is because there is not enough research about it, since it has probably not reached the research community yet. Interestingly, similar features in the form of extensions for Visual Studio Code are on the market and work in a very similar way as my Gen Z Mode [21] [15]. To properly evaluate the Gen Z Mode, a large number of participants separated into control and experimental groups would have to be recruited and conduct a quantitative study, which was not the focus of the tool and it was not feasible for this project, as measuring the effectiveness of ”Sludge” videos could be a research project by itself.

Bibliography

[1] What is capture the flag? https://ctfd.io/whats-a-ctf/, 2019.

[2] Cwe top 25 most dangerous software weaknesses. https://cwe.mitre.org/top25/archive/2023/2023_top25_list.html, 2023.

[3] Philip T. Kortum Aaron Bangor and James T. Miller. An empirical evaluation of the system usability scale. *International Journal of Human–Computer Interaction*, 24(6):574–594, 2008.

[4] Atlassian Team. What is the agile methodology?, 2024.

[5] Adam Barth, Collin Jackson, and John C. Mitchell. Robust defenses for cross-site request forgery. In *To appear at the 15th ACM Conference on Computer and Communications Security (CCS 2008)*, 2008.

[6] Bootstrap Team. Bootstrap documentation, 2019. Version 4.3.

[7] John Brooke et al. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996.

[8] Catherine Holecko. How to play capture the flag. https://www.parents.com/capture-the-flag-rules-and-strategies-8607203, 2024.

[9] Yen-ning Chang, Youn-kyung Lim, and Erik Stolterman. Personas: from theory to practices. In *Proceedings of the 5th Nordic Conference on Human-Computer Interaction: Building Bridges*, NordiCHI '08, page 439–442, New York, NY, USA, 2008. Association for Computing Machinery.

[10] Elizabeth Charters. The use of think-aloud methods in qualitative research an introduction to think-aloud methods. *Brock Education Journal*, 12(2), 2003.

[11] Kevin Chung and Julian Cohen. Learning obstacles in the capture the flag model. In *2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14)*. USENIX Association, 2014.

[12] Sebastian Deterding, Rilla Khaled, Lennart Nacke, and Dan Dixon. Gamification: Toward a definition. pages 12–15, 01 2011.

[13] Hai Dinh-Tuan, Maria Mora-Martinez, Felix Beierle, and Sandro Rodriguez Garzon. Development frameworks for microservice-based applications: Evaluation and comparison. In *Proceedings of the 2020 European Symposium on Software*

Engineering, ESSE ’20, page 12–20, New York, NY, USA, 2020. Association for Computing Machinery.

[14] Docker Team. Docker documentation, 2013-2024. Version 4.28.

[15] Ethan Houseworth. Sludgevs. `https://marketplace.visualstudio.com/items?itemName=EthanHouseworth.sludgevs`.

[16] Google, Inc. Gruyere: Web application exploits and defenses. `https://google-gruyere.appspot.com/`, 2023.

[17] Shashank Gupta and Brij Bhooshan Gupta. Cross-site scripting (xss) attacks and defense mechanisms: classification and state-of-the-art. *International Journal of System Assurance Engineering and Management*, 8:512–530, 2017.

[18] Hack The Box. Hack the box. `https://www.hackthebox.com/`.

[19] J. Fox. Top cybersecurity statistics for 2024. Findings report, Cobalt Labs, Inc.

[20] Robert Cassidy Jihan Rabah and Robert Beauchemin. Gamification in education: Real benefits or edutainment. In *17th European Conference on e-Learning, Athens, Greece*, pages 489–497, 2018.

[21] Jiří Vrba. Subway surfers. `https://marketplace.visualstudio.com/items?itemName=jirkavrba.subway-surfers`.

[22] Martin Johns, Björn Engelmann, and Joachim Posegga. Xssds: Server-side detection of cross-site scripting attacks. In *2008 Annual Computer Security Applications Conference (ACSAC)*, pages 335–344, 2008.

[23] Marin Kaluža, Marijana Kalanj, and Bernard Vukelić. A comparison of back-end frameworks for web application development. *Zbornik veleučilišta u rijeci*, 7(1):317–332, 2019.

[24] Karl M. Kapp. *The Gamification of Learning and Instruction: Game-Based Methods and Strategies for Training and Education*, volume 1. ASTD, 2012.

[25] kingthorin. Sql injection. *OWASP*, 2023.

[26] KirstenS. Cross site request forgery (csrf). *OWASP*, 2023.

[27] KirstenS. Cross site scripting (xss). *OWASP*, 2023.

[28] Stela Kucek and Maria Leitner. An empirical survey of functions and configurations of open-source capture the flag (ctf) environments. *Journal of Network and Computer Applications*, 2020.

[29] Lucas McDaniel, Erik Talvi, and Brian Hay. Capture the flag as cyber security introduction. In *2016 49th Hawaii International Conference on System Sciences (HICSS)*, 2016.

[30] Jakob Nielsen. 10 usability heuristics for user interface design. *Nielsen Norman Group*, 1994.

[31] Jakob Nielsen and Rolf Molich. Heuristic evaluation of user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, 1990.

[32] OWASP. Sql injection prevention cheat sheet. *OWASP Cheat Sheet Series*, 2023.

[33] OWASP. Content security policy cheat sheet. *OWASP Cheat Sheet Series*, 2024.

[34] OWASP. Cross site scripting prevention cheat sheet. *OWASP Cheat Sheet Series*, 2024.

[35] OWASP. Input validation cheat sheet. *OWASP Cheat Sheet Series*, 2024.

[36] Peter G. Polson, Clayton Lewis, John Rieman, and Cathleen Wharton. Cognitive walkthroughs: a method for theory-based evaluation of user interfaces. *International Journal of Man-Machine Studies*, 36(5):741–773, 1992.

[37] Portswigger. Portswigger: Web security academy. `https://portswigger.net/web-security`.

[38] Jmanico Pawel Krawczyk Alan Hogan Eyal Lupu D0ubl3 h3lix Roberto Martelloni Tarin Gamberini Ali Khalfan Markgordon kingthorin mkost Grant Ongers Rknell, Wichers. Httponly. *OWASP*, 2024.

[39] S. Coutinho, A. Bollen, C. Weil, C. Sheerin, D. Silvera, Ipsos S. Donaldson, J. Rosborough, Perspective Economics. Cyber security skills in the uk labour market 2023. Findings report, UK Government - Department of Science, Innovation Technology.

[40] Jason H Sharp. Using codecademy interactive lessons as an instructional supplement in a python programming course. *Information Systems Education Journal*, 17(3):20, 2019.

[41] SQLite Team. Sqlite documentation, 2024.

[42] Mouldi Sagar Thomas Mahatody and Christophe Kolski. State of the art on the cognitive walkthrough method, its variants and evolutions. *International Journal of Human–Computer Interaction*, 2010.

[43] William J Triplett. Addressing cybersecurity challenges in education. *International Journal of STEM Education for Sustainability*, 3(1):47–67, 2023.

[44] Jan Vykopal, Valdemar Švábenský, and Ee-Chien Chang. Benefits and pitfalls of using capture the flag games in university courses. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, SIGCSE '20. Association for Computing Machinery, 2020.

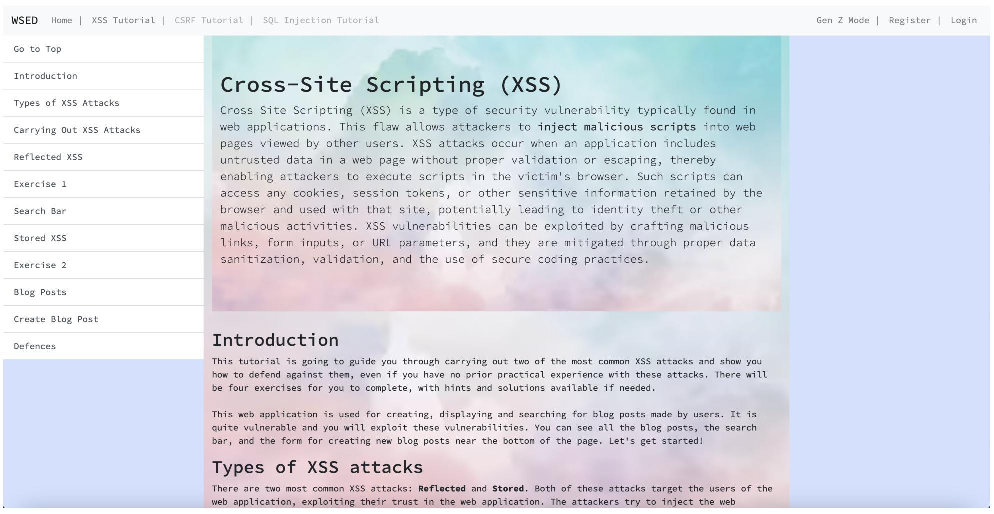
[45] Jeff Williams. Injection theory. *OWASP*, 2023.

[46] Weilin Zhong. Command injection. *OWASP*, 2023.

Appendix A

Web Application Images

A.1 Sprint One



WSED Home | XSS Tutorial | CSRF Tutorial | SQL Injection Tutorial   Gen Z Mode | Register | Login

Go to Top
Introduction
Types of XSS Attacks
Carrying Out XSS Attacks
Reflected XSS
Exercise 1
Search Bar
Stored XSS
Exercise 2
Blog Posts
Create Blog Post
Defences

Cross-Site Scripting (XSS)

Cross Site Scripting (XSS) is a type of security vulnerability typically found in web applications. This flaw allows attackers to **inject malicious scripts** into web pages viewed by other users. XSS attacks occur when an application includes untrusted data in a web page without proper validation or escaping, thereby enabling attackers to execute scripts in the victim's browser. Such scripts can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site, potentially leading to identity theft or other malicious activities. XSS vulnerabilities can be exploited by crafting malicious links, form inputs, or URL parameters, and they are mitigated through proper data sanitization, validation, and the use of secure coding practices.

Introduction

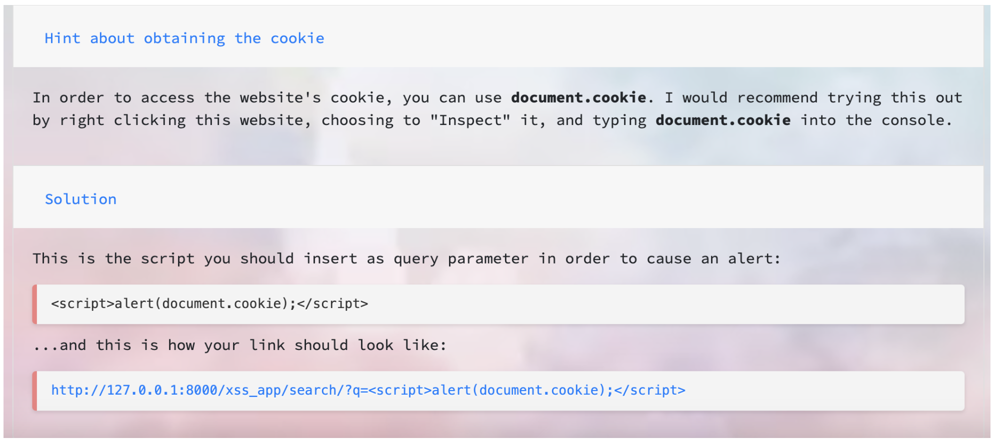
This tutorial is going to guide you through carrying out two of the most common XSS attacks and show you how to defend against them, even if you have no prior practical experience with these attacks. There will be four exercises for you to complete, with hints and solutions available if needed.

This web application is used for creating, displaying and searching for blog posts made by users. It is quite vulnerable and you will exploit these vulnerabilities. You can see all the blog posts, the search bar, and the form for creating new blog posts near the bottom of the page. Let's get started!

Types of XSS attacks

There are two most common XSS attacks: **Reflected** and **Stored**. Both of these attacks target the users of the web application, exploiting their trust in the web application. The attackers try to inject the web

Figure A.1: Example interface layout



Hint about obtaining the cookie

In order to access the website's cookie, you can use **document.cookie**. I would recommend trying this out by right clicking this website, choosing to "Inspect" it, and typing **document.cookie** into the console.

Solution

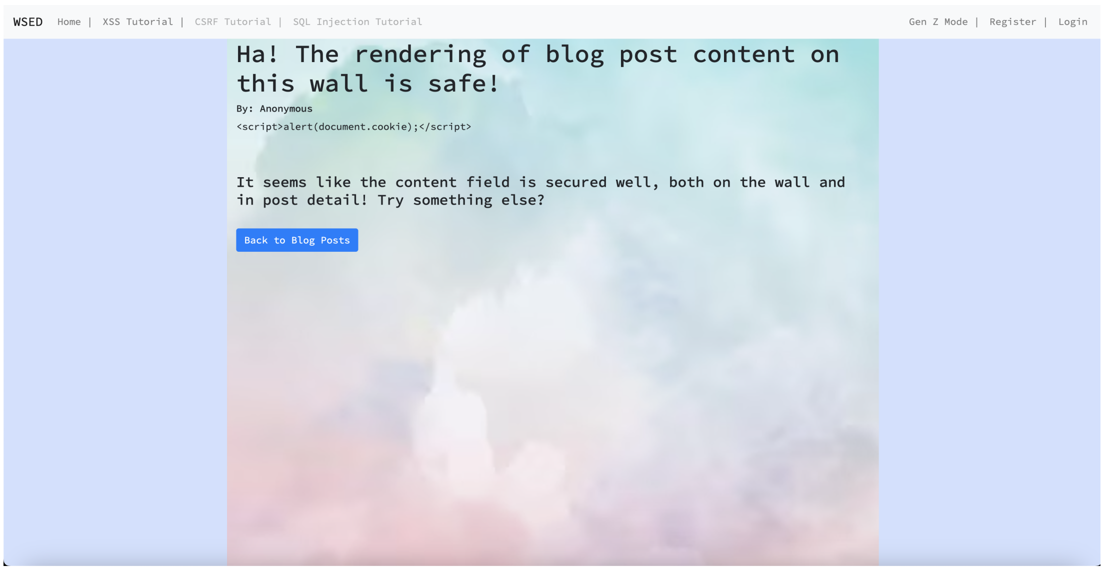
This is the script you should insert as query parameter in order to cause an alert:

```
<script>alert(document.cookie);</script>
```

...and this is how your link should look like:

```
http://127.0.0.1:8000/xss_app/search/?q=<script>alert(document.cookie);</script>
```

Figure A.2: Exercise 1 example hint and solution


WSED
Home | XSS Tutorial | CSRF Tutorial | SQL Injection Tutorial
Gen Z Mode | Register | Login
Back to Blog Posts

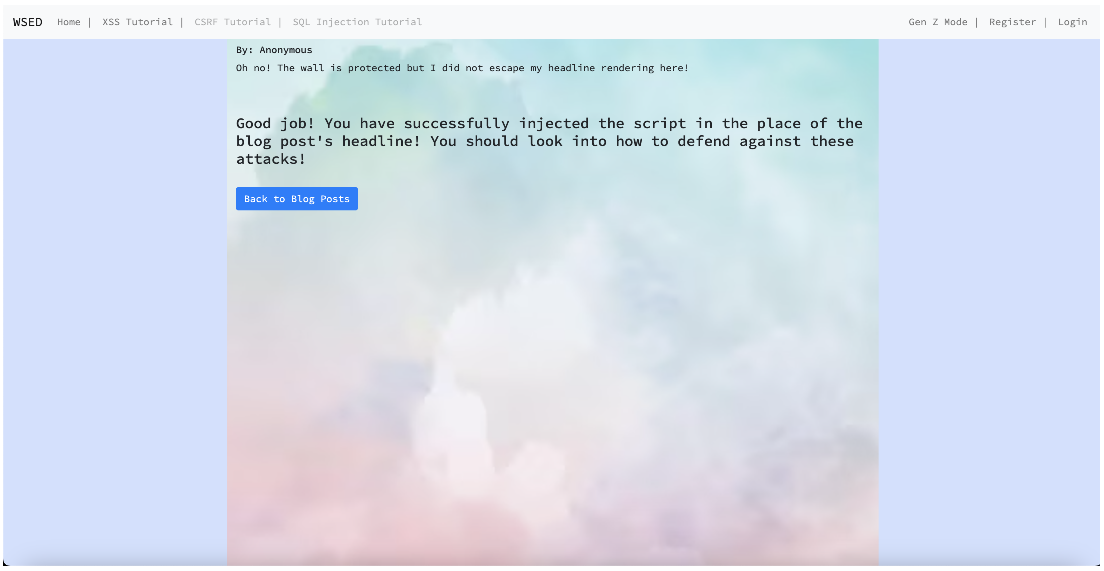
Ha! The rendering of blog post content on this wall is safe!

By: Anonymous

```
<script>alert(document.cookie);</script>
```

It seems like the content field is secured well, both on the wall and in post detail! Try something else?

Figure A.3: Exercise 2 dynamic rewriting of blog post detail when content is injected with malicious code

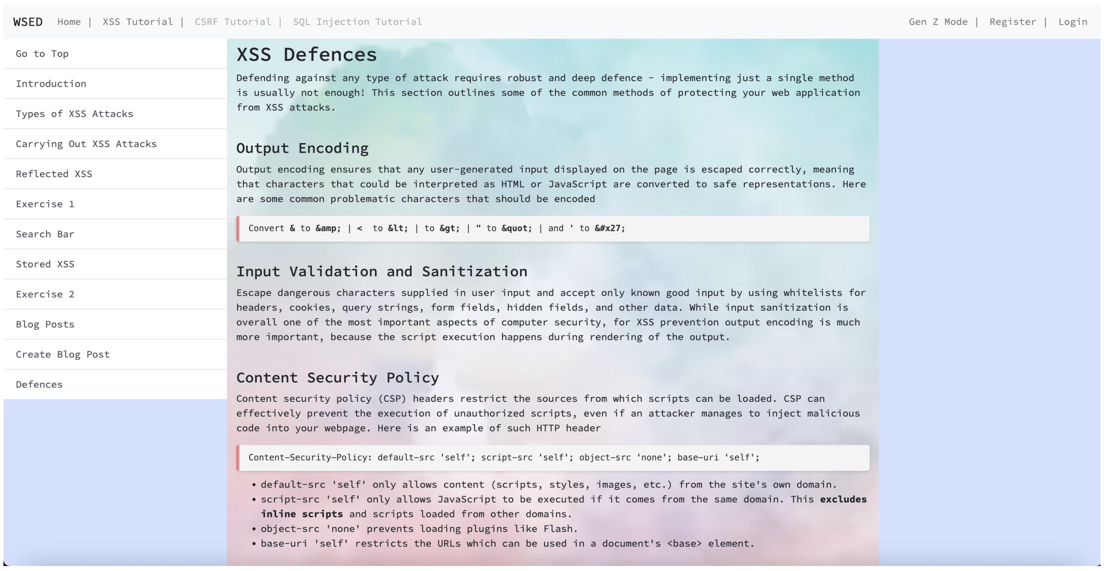

WSED
Home | XSS Tutorial | CSRF Tutorial | SQL Injection Tutorial
Gen Z Mode | Register | Login
Back to Blog Posts

By: Anonymous

Oh no! The wall is protected but I did not escape my headline rendering here!

Good job! You have successfully injected the script in the place of the blog post's headline! You should look into how to defend against these attacks!

Figure A.4: Exercise 2 dynamic rewriting of blog post detail when headline is injected with malicious code



XSS Defences

Defending against any type of attack requires robust and deep defence - implementing just a single method is usually not enough! This section outlines some of the common methods of protecting your web application from XSS attacks.

Output Encoding

Output encoding ensures that any user-generated input displayed on the page is escaped correctly, meaning that characters that could be interpreted as HTML or JavaScript are converted to safe representations. Here are some common problematic characters that should be encoded

```
Convert & to &amp; | < to &lt; | to &gt; | " to &quot; | and ' to &#x27;
```

Input Validation and Sanitization

Escape dangerous characters supplied in user input and accept only known good input by using whitelists for headers, cookies, query strings, form fields, hidden fields, and other data. While input sanitization is overall one of the most important aspects of computer security, for XSS prevention output encoding is much more important, because the script execution happens during rendering of the output.

Content Security Policy

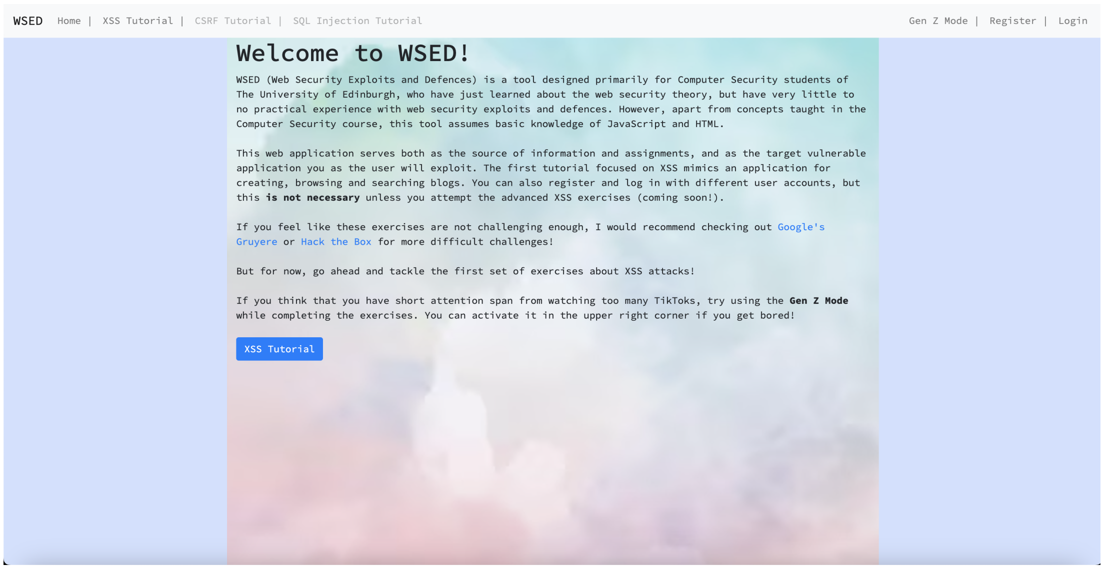
Content security policy (CSP) headers restrict the sources from which scripts can be loaded. CSP can effectively prevent the execution of unauthorized scripts, even if an attacker manages to inject malicious code into your webpage. Here is an example of such HTTP header

```
Content-Security-Policy: default-src 'self'; script-src 'self'; object-src 'none'; base-uri 'self';
```

- default-src 'self' only allows content (scripts, styles, images, etc.) from the site's own domain.
- script-src 'self' only allows JavaScript to be executed if it comes from the same domain. This **excludes inline scripts** and scripts loaded from other domains.
- object-src 'none' prevents loading plugins like Flash.
- base-uri 'self' restricts the URLs which can be used in a document's <base> element.

Figure A.5: Defences section example

A.2 Sprint Two


WSED
Home | XSS Tutorial | CSRF Tutorial | SQL Injection Tutorial
Gen Z Mode | Register | Login
XSS Tutorial

Welcome to WSED!

WSED (Web Security Exploits and Defences) is a tool designed primarily for Computer Security students of The University of Edinburgh, who have just learned about the web security theory, but have very little to no practical experience with web security exploits and defences. However, apart from concepts taught in the Computer Security course, this tool assumes basic knowledge of JavaScript and HTML.

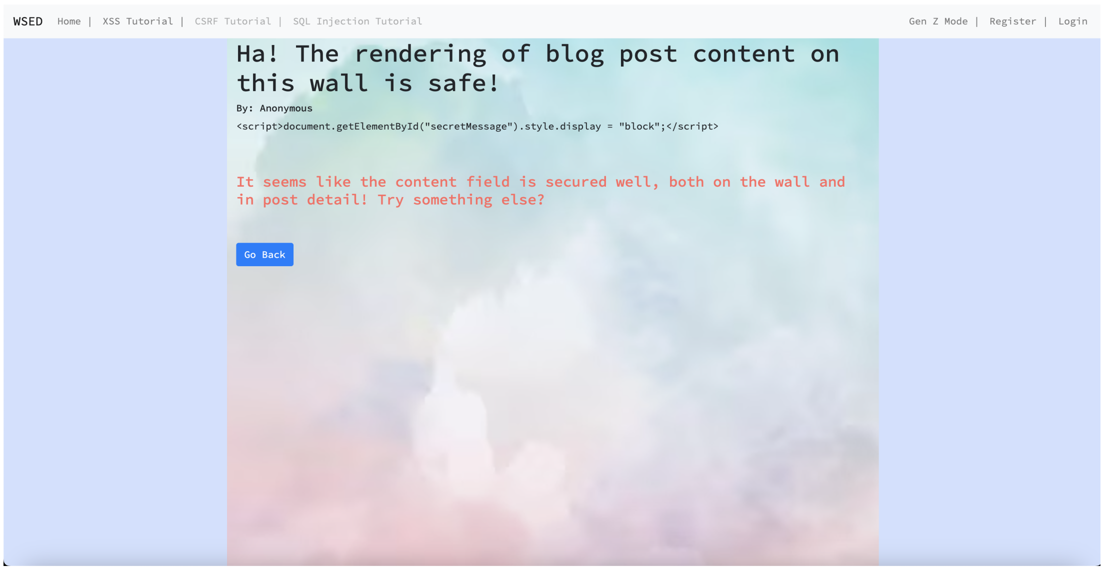
This web application serves both as the source of information and assignments, and as the target vulnerable application you as the user will exploit. The first tutorial focused on XSS mimics an application for creating, browsing and searching blogs. You can also register and log in with different user accounts, but this **is not necessary** unless you attempt the advanced XSS exercises (coming soon!).

If you feel like these exercises are not challenging enough, I would recommend checking out Google's Gruyere or Hack the Box for more difficult challenges!

But for now, go ahead and tackle the first set of exercises about XSS attacks!

If you think that you have short attention span from watching too many TikToks, try using the **Gen Z Mode** while completing the exercises. You can activate it in the upper right corner if you get bored!

Figure A.6: The welcome (landing) page of the application


WSED
Home | XSS Tutorial | CSRF Tutorial | SQL Injection Tutorial
Gen Z Mode | Register | Login
Go Back

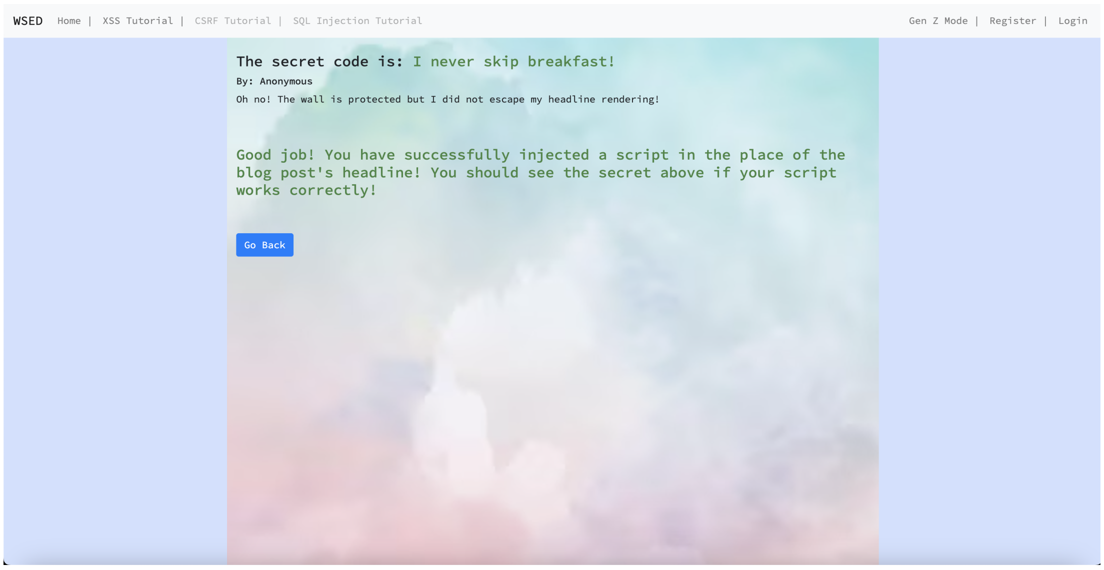
Ha! The rendering of blog post content on this wall is safe!

By: Anonymous

```
<script>document.getElementById("secretMessage").style.display = "block";</script>
```

It seems like the content field is secured well, both on the wall and in post detail! Try something else?

Figure A.7: Blog post detail page with malicious code in content


WSED
Home | XSS Tutorial | CSRF Tutorial | SQL Injection Tutorial
Gen Z Mode | Register | Login
Go Back

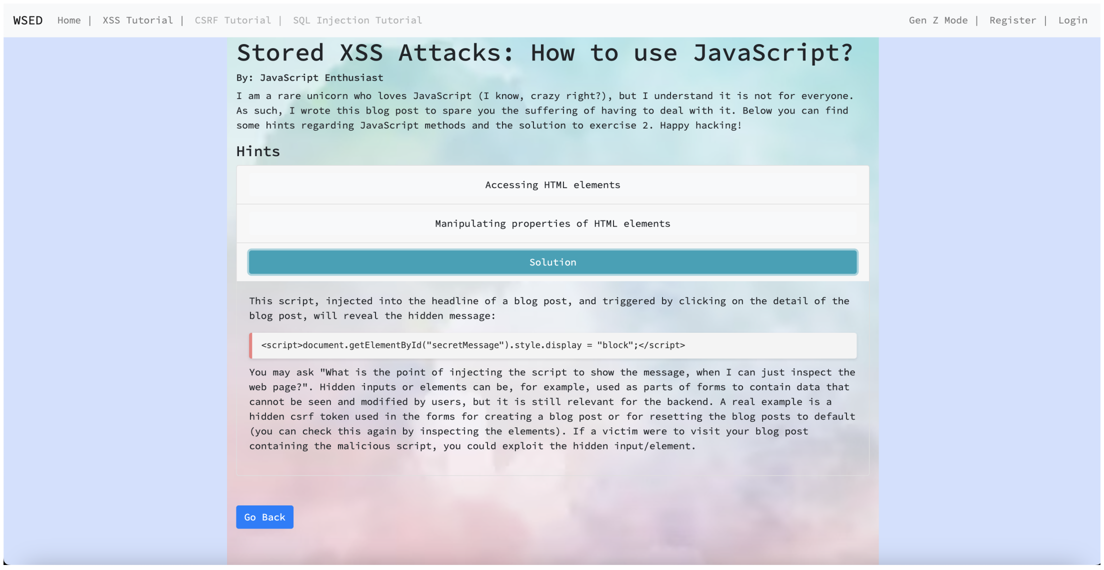
The secret code is: I never skip breakfast!

By: Anonymous

Oh no! The wall is protected but I did not escape my headline rendering!

Good job! You have successfully injected a script in the place of the blog post's headline! You should see the secret above if your script works correctly!

Figure A.8: Blog post detail page with malicious code in headline



Stored XSS Attacks: How to use JavaScript?

By: JavaScript Enthusiast

I am a rare unicorn who loves JavaScript (I know, crazy right?), but I understand it is not for everyone. As such, I wrote this blog post to spare you the suffering of having to deal with it. Below you can find some hints regarding JavaScript methods and the solution to exercise 2. Happy hacking!

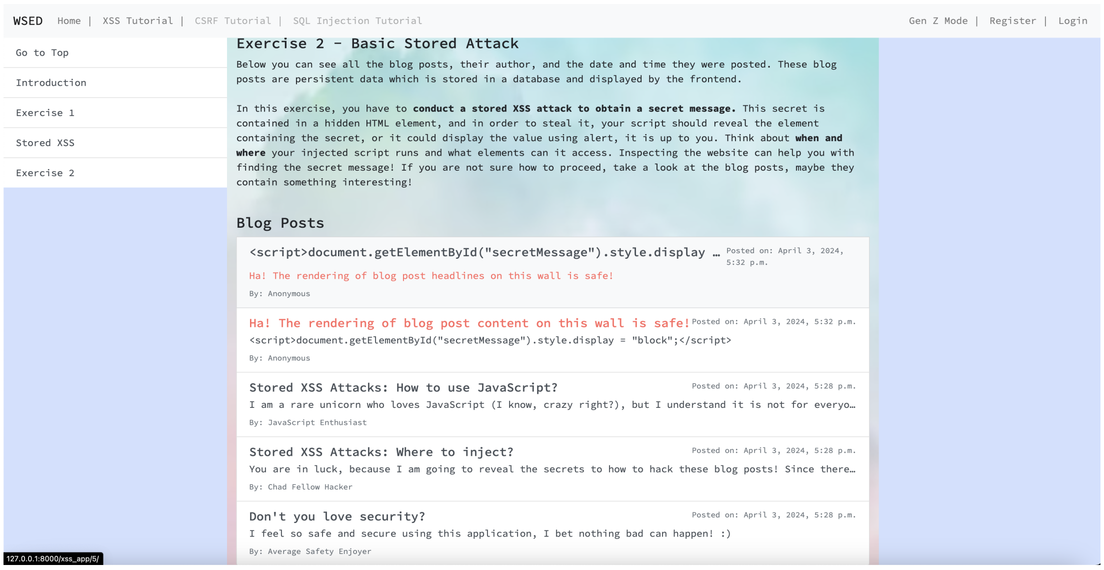
Hints

This script, injected into the headline of a blog post, and triggered by clicking on the detail of the blog post, will reveal the hidden message:

```
<script>document.getElementById("secretMessage").style.display = "block";</script>
```

You may ask "What is the point of injecting the script to show the message, when I can just inspect the web page?". Hidden inputs or elements can be, for example, used as parts of forms to contain data that cannot be seen and modified by users, but it is still relevant for the backend. A real example is a hidden csrf token used in the forms for creating a blog post or for resetting the blog posts to default (you can check this again by inspecting the elements). If a victim were to visit your blog post containing the malicious script, you could exploit the hidden input/element.

Figure A.9: Third default blog post with hints and JavaScript and the solution unrolled



Exercise 2 - Basic Stored Attack

Below you can see all the blog posts, their author, and the date and time they were posted. These blog posts are persistent data which is stored in a database and displayed by the frontend.

In this exercise, you have to **conduct a stored XSS attack to obtain a secret message.** This secret is contained in a hidden HTML element, and in order to steal it, your script should reveal the element containing the secret, or it could display the value using alert, it is up to you. Think about **when and where** your injected script runs and what elements can it access. Inspecting the website can help you with finding the secret message! If you are not sure how to proceed, take a look at the blog posts, maybe they contain something interesting!

Blog Posts

<script>document.getElementById("secretMessage").style.display … Posted on: April 3, 2024, 5:32 p.m.
Ha! The rendering of blog post headlines on this wall is safe!
By: Anonymous

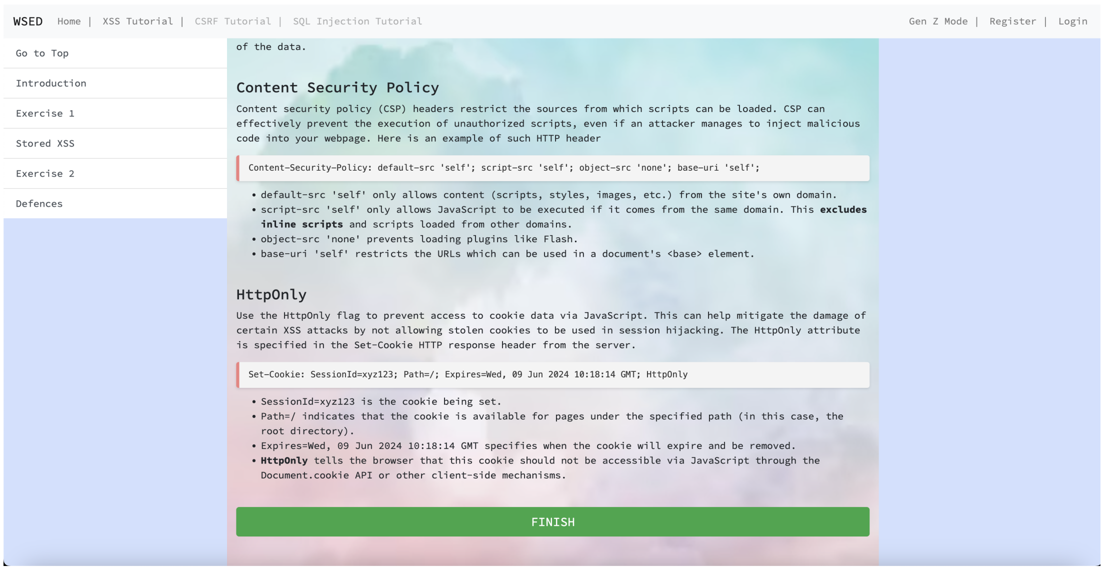
Ha! The rendering of blog post content on this wall is safe! Posted on: April 3, 2024, 5:32 p.m.
<script>document.getElementById("secretMessage").style.display = "block";</script>
By: Anonymous

Stored XSS Attacks: How to use JavaScript? Posted on: April 3, 2024, 5:28 p.m.
I am a rare unicorn who loves JavaScript (I know, crazy right?), but I understand it is not for everyo…
By: JavaScript Enthusiast

Stored XSS Attacks: Where to inject? Posted on: April 3, 2024, 5:28 p.m.
You are in luck, because I am going to reveal the secrets to how to hack these blog posts! Since there…
By: Chad Fellow Hacker

Don't you love security? Posted on: April 3, 2024, 5:28 p.m.
I feel so safe and secure using this application, I bet nothing bad can happen! :)
By: Average Safety Enjoyer

Figure A.10: Blog post wall after creating blog posts with malicious headlines and content


WSED
Home | XSS Tutorial | CSRF Tutorial | SQL Injection Tutorial
Gen Z Mode | Register | Login
Go to Top
Introduction
Exercise 1
Stored XSS
Exercise 2
Defences
FINISH

of the data.

Content Security Policy

Content security policy (CSP) headers restrict the sources from which scripts can be loaded. CSP can effectively prevent the execution of unauthorized scripts, even if an attacker manages to inject malicious code into your webpage. Here is an example of such HTTP header

```
Content-Security-Policy: default-src 'self'; script-src 'self'; object-src 'none'; base-uri 'self';
```

- default-src 'self' only allows content (scripts, styles, images, etc.) from the site's own domain.
- script-src 'self' only allows JavaScript to be executed if it comes from the same domain. This **excludes inline scripts** and scripts loaded from other domains.
- object-src 'none' prevents loading plugins like Flash.
- base-uri 'self' restricts the URLs which can be used in a document's <base> element.

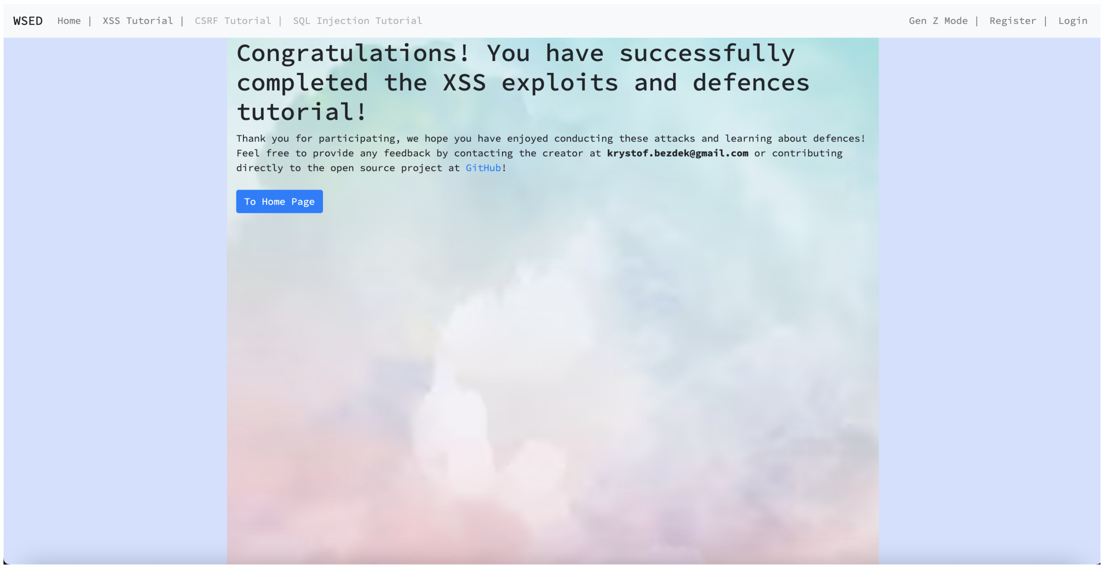
HttpOnly

Use the HttpOnly flag to prevent access to cookie data via JavaScript. This can help mitigate the damage of certain XSS attacks by not allowing stolen cookies to be used in session hijacking. The HttpOnly attribute is specified in the Set-Cookie HTTP response header from the server.

```
Set-Cookie: SessionId=xyz123; Path=/; Expires=Wed, 09 Jun 2024 10:18:14 GMT; HttpOnly
```

- SessionId=xyz123 is the cookie being set.
- Path=/ indicates that the cookie is available for pages under the specified path (in this case, the root directory).
- Expires=Wed, 09 Jun 2024 10:18:14 GMT specifies when the cookie will expire and be removed.
- **HttpOnly** tells the browser that this cookie should not be accessible via JavaScript through the Document.cookie API or other client-side mechanisms.

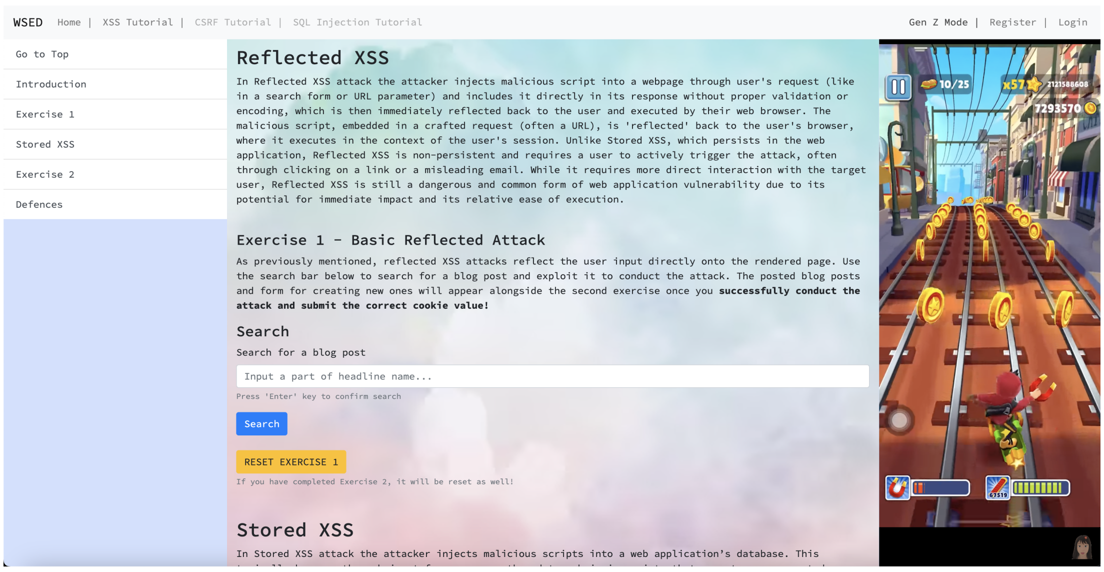
Figure A.11: Lower part of the defences section with Finish button


WSED
Home | XSS Tutorial | CSRF Tutorial | SQL Injection Tutorial
Gen Z Mode | Register | Login
To Home Page

Congratulations! You have successfully completed the XSS exploits and defences tutorial!

Thank you for participating, we hope you have enjoyed conducting these attacks and learning about defences! Feel free to provide any feedback by contacting the creator at **krystof.bezdek@gmail.com** or contributing directly to the open source project at GitHub!

Figure A.12: The final Thank You page after finishing the tutorial



Reflected XSS

In Reflected XSS attack the attacker injects malicious script into a webpage through user's request (like in a search form or URL parameter) and includes it directly in its response without proper validation or encoding, which is then immediately reflected back to the user and executed by their web browser. The malicious script, embedded in a crafted request (often a URL), is 'reflected' back to the user's browser, where it executes in the context of the user's session. Unlike Stored XSS, which persists in the web application, Reflected XSS is non-persistent and requires a user to actively trigger the attack, often through clicking on a link or a misleading email. While it requires more direct interaction with the target user, Reflected XSS is still a dangerous and common form of web application vulnerability due to its potential for immediate impact and its relative ease of execution.

Exercise 1 - Basic Reflected Attack

As previously mentioned, reflected XSS attacks reflect the user input directly onto the rendered page. Use the search bar below to search for a blog post and exploit it to conduct the attack. The posted blog posts and form for creating new ones will appear alongside the second exercise once you **successfully conduct the attack and submit the correct cookie value!**

Search

Search for a blog post

Input a part of headline name...

Press 'Enter' key to confirm search

Search

RESET EXERCISE 1

If you have completed Exercise 2, it will be reset as well!

Stored XSS

In Stored XSS attack the attacker injects malicious scripts into a web application’s database. This

Figure A.13: Extra: Activated Gen Z Mode, playing Subway Surfers in the right panel

Appendix B

Requirements Gathering Student Questionnaire

The Requirements Gathering Student Questionnaire starts on the next page.

Web Security Questionnaire

* Indicates required question

Web Security Questionnaire

This questionnaire is intended to gather information of what people found hard about the web security section of the Computer Security course taught at the University of Edinburgh and about their experience with teaching tools in the field of computer security.

This study will be used to gather requirements for a new teaching tool focused on web security exploits and defences, any feedback you provide will be used for this purpose along with anonymous use in related work. The data collected from this survey will be kept for a maximum of two months and then destroyed. Anonymised quotes may be retained longer for use by future students on this project.

1. **Are you a student at the University of Edinburgh studying in the School of Informatics?** *

   Mark only one oval.

   ◯ Yes

   ◯ No

2. **Have you taken the Computer Security course in the past, or are you currently taking the course?** *

   Mark only one oval.

   ◯ Yes

   ◯ No *Skip to section 7 (**Thank you for completing the questionnaire**)*

Computer Security Experience

This study will be used to evaluate the confidence of students in their general knowledge of concepts taught in the Computer Security course, any feedback you provide will be used for this purpose along with anonymous use in related work. The answers will be kept for a maximum of two months and then destroyed. Anonymised quotes may be retained longer for use by future students on this project.

3. **How would you describe your overall knowledge of theoretical concepts taught in the Computer Security course?** *

For example, would you be able to describe what an SQL Injection attack is and what are some common defence strategies?

Mark only one oval.

- Poor
- Fair
- Good
- Very Good
- Excellent

4. **How would you describe your overall confidence of applying concepts taught in the Computer Security course?** *

For example, would you be able to conduct an SQL Injection attack in a controlled environment, eg in a prepared virtual machine?

Mark only one oval.

- Poor
- Fair
- Good
- Very Good
- Excellent

5. **How difficult did you consider the five main topics taught in the Computer Security?** *

Mark only one oval per row.

	Very difficult	Difficult	Neutral	Easy	Very Easy
Network Security	⬭	⬭	⬭	⬭	⬭
Cryptography	⬭	⬭	⬭	⬭	⬭
Secure Communications	⬭	⬭	⬭	⬭	⬭
OS Security	⬭	⬭	⬭	⬭	⬭
Web Security	⬭	⬭	⬭	⬭	⬭

6. **Do you have any other comments or elaborations related to your answers?**

Web Security Experience

This study will be used to evaluate the confidence of students in their knowledge of web security concepts taught in the final section of the Computer Security course, any feedback you provide will be used for this purpose along with anonymous use in related work. The answers will be kept for a maximum of two months and then destroyed. Anonymised quotes may be retained longer for use by future students on this project.

7. **How difficult did you consider these topics taught in the Web Security part of the Computer Security course?** *

Mark only one oval per row.

	Very difficult	Difficult	Neutral	Easy	Very Easy
URLs and HTTP	⬭	⬭	⬭	⬭	⬭
Front-End technologies (HTML, JavaScript)	⬭	⬭	⬭	⬭	⬭
Cookies	⬭	⬭	⬭	⬭	⬭
Cross Site Scripting (XSS)	⬭	⬭	⬭	⬭	⬭
Cross Site Request Forgery (XSRF)	⬭	⬭	⬭	⬭	⬭
Injections	⬭	⬭	⬭	⬭	⬭

8. **Did you find anything else particularly difficult or easy to understand?**

9. **How confident would you be in performing attacks taught in the web security part of the Computer Security course?** *

The attacks would be in a controlled environment, such as a prepared Virtual Machine

Mark only one oval per row.

	Very confident	Confident	Neutral	Not confident	Not confident at all
Reflected Cross Site Scripting	◯	◯	◯	◯	◯
Stored Cross Site Scripting	◯	◯	◯	◯	◯
Cross Site Request Forgery	◯	◯	◯	◯	◯
SQL Injection	◯	◯	◯	◯	◯
Command Injection	◯	◯	◯	◯	◯

10. **Are there specific aspects or nuances of XSS/XSRF/Injection attacks that you find particularly confusing?**

11. **Computer Security exams offer three questions, and the students have to answer two. Each of these questions usually correlates to one of the five main parts of this course, theme wise. Did you choose the web security question last year as one of the two? If you have not taken the exam yet, are you planning on doing the web security question?** *

Mark only one oval.

- Yes
- No
- Other: ______________________

12. **Do you have any other comments or elaborations related to your answers?**

____

____

____

____

____

Teaching tools experience

This study will be used to evaluate the students' experience with teaching tools and other resources used to learn about Computer Security concepts, any feedback you provide will be used for this purpose along with anonymous use in related work. The answers will be kept for a maximum of two months and then destroyed. Anonymised quotes may be retained longer for use by future students on this project.

13. **Do you prefer learning about attacks and defences theoretically, or through hands-on practice?** *

Mark only one oval.

- Theoretically
- Hands-on
- Other: ______________________

14. **Have you heard about the concept of Capture the Flag (CTF) in cybersecurity?** *

Mark only one oval.

- ⬭ Yes
- ⬭ No

15. **Have you ever attempted to solve or competed in any CTFs?** *

Some examples are Google's Gruyere, Portswigger, Hack the Box

Mark only one oval.

- ⬭ Yes
- ⬭ No
- ⬭ Other: ______________________

16. **Did completing any CTF challenges help you with your studies? How?**

17. **What, if any, further resources did you use to learn about web security attacks and defences?**

18. **What features or capabilities would you like to see in a teaching tool that focuses on web attacks and defences? Consider aspects such as step-by-step guided tutorials, real-world scenario-based challenges, in-depth explanations of attack mechanisms, quizzes and assessments, gamification, etc. Feel free to suggest any other features you think you would find helpful.**

19. **Do you have any final comments or elaborations related to your answers?**

Final question

20. **Would you be interested in being contacted in the future regarding further studies and evaluation of this project? If so, enter your email please**

Thank you for completing the questionnaire

Appendix C

Cognitive Walkthrough Usability Aspect Reports

C.1 Report 1

Problem/good aspect: Problem

Name: No feedback when succeeding in challenges

Evidence

Heuristic: Visibility of system status

Interface aspect: There is no confirmation of succeeding in a challenge

Explanation: The system provides no information on success or failure of a task. The user has to know the result they are looking for, which may be confusing for students learning these exploits. They may try to conduct an exploit and think what they did it is enough, although they actually completed it, or vice versa.

Severity or Benefit

Rating: Medium

Justification: It can be very frustrating for users to not understand how they currently stand

Frequency: Medium

Impact: Medium

Persistence: High

How I weight the factors: This will impact novice 3rd year students trying to learn about web exploits, as it is very frustrating when you can' validate that what you are doing is actually correct.

Possible solutions: Implement a system that checks your actions and gives feedback

Relationships: None

C.2 Report 2

Problem/good aspect: Problem

Name: App window leaving from instructions

Evidence

Heuristic: User control and freedom

Interface aspect: Instead of opening new tab, the app replaces instructions tab

Explanation: When a user clicks on the link to start the app, it replaces the instructions tab. This causes confusion, because all instructions suddenly disappear and the user has no idea what to do, so they have to return back anyway, wasting time.

Severity or Benefit

Rating: Low

Justification: The user can just use back button and then open it in new tab

Frequency: Low

Impact: Low

Persistence: High

How I weight the factors: It does not cause major setbacks or frustration, however, it is unnecessary and annoying.

Possible solutions: Make it open in new tab

Relationships: None

C.3 Report 3

Problem/good aspect: Problem

Name: Having instructions with information, and the app in separate tabs

Evidence

Heuristic: Recognition rather than recall

Interface aspect: Having two separate tab to click between

Explanation: The set of instructions, explanations, challenges, hints and fixes are all in a separate tab than the system where you perform all actions. This means that the user finds themselves constantly click between these tabs, especially with the huge amount of text and information available on the website.

Severity or Benefit

Rating: High

Justification: Someone with little to no experience with web exploits has to perpetually go back to reread the assignment to ensure they understood it correctly and check the hints to be sure they are progressing

Frequency: High

Impact: Low

Persistence: High

How I weight the factors: It makes it very difficult to use the system, because people forget the information given in the assignment and have to constantly go back and forth to remind themselves of what to do.

Possible solutions: Have only one tab where all the information is displayed alongside the current challenge, or at least just the key things from the assignment

Relationships: Report 2

C.4 Report 4

Problem/good aspect: Problem

Name: Cluttered with text and unnecessary information

Evidence

Heuristic: Aesthetic and minimalist design

Interface aspect: The main instructions page

Explanation: The entire website is cluttered with large amounts of text and information, which often is not relevant at all to the tasks a user wants to accomplish.

Severity or Benefit

Rating: Low

Justification: The user may spend unnecessary time reading things that are not important to them, which can be frustrating.

Frequency: High

Impact: Low

Persistence: High

How I weight the factors: Apart from some descriptions of hints, exploits and fixes, most of the pages are just filled with small text which most users will only skim through, risking they will miss actual important information

Possible solutions: Reduce the amount of unnecessary text and highlight the important information

Relationships: None

C.5 Report 5

Problem/good aspect: Good aspect

Name: The vulnerable app is intuitive as other blogs

Evidence

Heuristic: Match between system and the real world

Interface aspect: The main vulnerable app

Explanation: The vulnerable app where all the exploits happen is structured in a similar way as blog or social media websites, which makes it intuitive to navigate as it is something most students will have experience with

Severity or Benefit

Rating: Medium

Justification: Having an intuitive user interface is important for users to navigate effectively, allowing them to focus on the matters at hand

Frequency: High

Impact: Medium

Persistence: High

How I weight the factors: The user is constantly affected by the quality of user interface and it has an impact on their experience

C.6 Report 6

Problem/good aspect: Good aspect

Name: Extensive and detailed documentation

Evidence

Heuristic: Help and documentation

Interface aspect: The main documentation and information page

Explanation: Once the user orients themselves within the web, they find out it provides detailed and extensive documentation of how the app works, information about the vulnerabilities, as well as useful hints how to exploit and fix them

Severity or Benefit

Rating: Low

Justification: Having important documentation and help is important, however, with tools for example like ChatGPT or YouTube, reading documentation is less common.

Frequency: Low

Impact: Low

Persistence: High

How I weight the factors: The user does not have to check the documentation often, and often will gravitate towards more concise and short resources rather than to official documentation

Appendix D

Pre-Tutorial Questionnaire

The Pre-Tutorial Questionnaire starts on the next page.

Pre-Tutorial Questionnaire

* Indicates required question

This questionnaire is designed collect information about your experience and knowledge of Cross-Site Scripting (XSS) exploits and defences before finishing the think aloud session.

1. What is your student ID (s1234567) *

____

2. What year are you in? *

Mark only one oval.

◯ Year 3
◯ Year 4
◯ Year 5

3. Have you taken the Computer Security, or any other course about computer security such as Secure Programming? *

Mark only one oval.

◯ Yes - Computer Security
◯ No - I have never taken any computer security related course
◯ Other: ______________________

4. How would you rate your theoretical knowledge of web security concepts, specifically of XSS? *

Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
No k	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Very high knowledge

5. How would you rate your confidence in conducting XSS attacks?

Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
No c	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Very high confidence

The following two questions are voluntary, do not feel pressured to answer if you are not comfortable with it!

6. Do you consider yourself to have a short attention span or become easily bored?

Mark only one oval.

◯ Yes

◯ No

◯ Other: ____________________

7. How many hours per week do you approximately spend scrolling TikTok, Instagram Reels, or YouTube Shorts?

____

The final two sections contain 8 open questions designed to test your knowledge of XSS exploits and defences before completing the think aloud session.

8. How does an XSS attack occur? *

9. What are the two most common XSS attack types? *

10. Which common mitigation technique is the most effective against XSS attacks? *

Questions continued

11. What is the main difference between Stored XSS and Reflected XSS? *

12. Which one of the two common XSS attacks is considered a more serious security threat and why? *

13. How would you access a website's cookie? *

14. How would you go about conducting reconnaissance of a website? *

15. What does HttpOnly cookie attribute do? *

Thank you for filling it out! Now onto the think aloud session

Appendix E

Think Aloud Script

Hello my name is Krystof.

Today we will be using the WSED website to learn about cross site scripting attacks and defences. Your participation today is purely voluntary, you may stop at any time. The purpose of this session is to identify issues with the WSED website. Please remember we are testing the website, we are not testing you.

In this observation, we are interested in what you think about as you perform the task we are asking you to do. In order to do this, I am going to ask you to talk aloud as you work on the task. What I mean by “talk aloud” is that I want you to tell me everything you are thinking from the first time you see the website. I would like you to talk aloud constantly from the time I give you the task till you have completed it. I do not want you to try and plan out what you say or try to explain to me what you are saying. Just act as if you were alone, speaking to yourself. It is most important that you keep talking. If you are silent for a long period of time, I will ask you to talk. There will be some reading involved as well. You are not required to read out loud, however, please vocalise any mental notes or thoughts you have while reading.

Do you understand what I want you to do?

[Wait for confirmation.]

Good.

Now we will begin with some practice problems. First, I will demonstrate by talking aloud while I solve a simple problem: “How many windows are there in this room?”

[Demonstrate talk aloud.]

Now it is your turn. Please talk aloud as you multiply 120 * 8.

[Let them finish]

Good. Now, those problems were solved all in our heads. However, when you are working on the computer you will also be looking for things, and seeing things that catch your attention. These things that you are searching for and things that you see are as important for our observation as thoughts you are thinking from memory. So please verbalise these too. As you are doing the tasks, I won't be able to answer any questions. But if you do have questions, go ahead and ask them anyway so I can learn more about what kinds of questions the website brings up. I will answer any questions after the session. Also, if you forget to talk aloud, I'll say, "please keep talking."

Do you have any questions about the talk aloud? Now I have some tasks prepared for you. I am going to go over them with you and see if you have any questions before we start.

Your task is to complete the Cross-Site Scripting (XSS) Tutorial Exercises 1 and 2, and learn about how to defend against XSS. Do you have any questions about the task?

You may begin.

Appendix F

Post-Tutorial Questionnaire

The Post-Tutorial Questionnaire starts on the next page.

Post-Tutorial Questionnaire

This questionnaire is designed collect information about your experience and knowledge of Cross-Site Scripting (XSS) exploits and defences after finishing the think aloud session. It also collects feedback on the usability aspects of the application.

* Indicates required question

1. What is your student ID (s1234567) *

2. Did you enjoy completing XSS Tutorial? *

3. How difficult did you find completing the exercises? *

Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
Easy	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Difficult

4. How would you rate your confidence in conducting XSS attacks? *

Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
No c	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Very high confidence

5. Did you find any specific aspects of the tool very good or very bad?

6. Do you have any other suggestions for the tool? It can be related to content, design, anything!

7. Did you use Gen Z Mode? *

Mark only one oval.

- ◯ Yes *Skip to question 8*
- ◯ No *Skip to question 9*

Gen Z Mode

8. Do you feel like it helped you with focus, or was it distracting? *

Mark only one oval.

	1	2	3	4	5	
It wa	◯	◯	◯	◯	◯	It helped with focus

This section contains 8 open questions designed to test your knowledge of XSS exploits and defences after completing the think aloud session.

9. How does an XSS attack occur? *

10. What are the two most common XSS attack types? *

11. Which common mitigation technique is the most effective against XSS attacks? *

12. What is the main difference between Stored XSS and Reflected XSS? *

13. Which one of the two common XSS attacks is considered a more serious security threat and why?

14. How would you access a website's cookie? *

15. How would you go about conducting reconnaissance of a website? *

16. What does HttpOnly cookie attribute do? *

Thank you for participation!

Appendix G

System Usability Scale (SUS)

The System Usability Scale starts on the next page.

System Usability Scale (SUS)

	Strongly Disagree				Strongly Agree
I think that I would like to use this system frequently.	1	2	3	4	5
I found the system unnecessarily complex.	1	2	3	4	5
I thought this system was easy to use.	1	2	3	4	5
I think that I would need the support of a technical person to be able to use this system.	1	2	3	4	5
I found the various functions in this system were well integrated.	1	2	3	4	5
I thought there was too much inconsistency in this system.	1	2	3	4	5
I would imagine that most people would learn to use this system very quickly.	1	2	3	4	5
I found this system very cumbersome to use.	1	2	3	4	5
I felt very confident using this system.	1	2	3	4	5
I needed to learn a lot of things before I could get going with this system.	1	2	3	4	5

Appendix H

Participants' information sheet

The Participants' information sheet can be seen on the next page.

Participant Information Sheet

Project title:	Tool for Teaching Web Application Exploits and Defences
Principal investigator:	Myrto D. Arapinis (marapini@inf.ed.ac.uk)
Researcher collecting data:	Krystof Bezdek (s2089135@ed.ac.uk)

This study was certified according to the Informatics Research Ethics Process, reference number 988079. Please take time to read the following information carefully. You should keep this page for your records.

Who are the researchers?

The project is supervised by Myrto D. Arapinis and conducted by Krystof Bezdek.

What is the purpose of the study?

The purpose of this study is to gather information of what people found hard about the web security section of the Computer Security course taught at the University of Edinburgh and about their experience with teaching tools in the field of computer security. This study will be used to gather requirements for a new teaching tool focused on web security exploits and defences, any feedback you provide will be used for this purpose along with anonymous use in related work. Furthermore, the study aims to evaluate the new teaching tool by running a Think Aloud session and presenting you with a pre- and post-tutorial questionnaire used to quiz you and measure your improvement, as well as gather feedback.

Why have I been asked to take part?

The target group are students at The University of Edinburgh who have taken the Computer Security course during their 3rd or 4th year.

Do I have to take part?

No – participation in this study is entirely up to you. You can withdraw from the study at any time, without giving a reason. Your rights will not be affected. If you wish to withdraw, contact the PI. We will stop using your data in any publications or presentations submitted after you have withdrawn consent. However, we will keep copies of your original consent, and of your withdrawal request.

What will happen if I decide to take part?

- Questions regarding your experience with web security, the Computer Security course, and your experience with teaching tools will be asked

- You will take a questionnaire

- It should take around 15 minutes at most

- Additionally, you will participate in Think Aloud session

- You will fill out the pre- and post-turoail questionnaires

- It should take around additional 30 to 60 minutes

Are there any risks associated with taking part?

There are no significant risks associated with participation.

Are there any benefits associated with taking part?

No.

What will happen to the results of this study?
The results of this study may be summarised in published articles, reports, and presentations. Quotes or key findings will be anonymized: We will remove any information that could, in our assessment, allow anyone to identify you. With your consent, information can also be used for future research. The data collected from this survey will be kept for a maximum of two months and then destroyed. Anonymised quotes may be retained longer for use by future students on this project.

Data protection and confidentiality.
Your data will be processed in accordance with Data Protection Law. All information collected about you will be kept strictly confidential. Your data will be referred to by a unique participant number rather than by name. Your data will only be viewed by the researcher/research team consisting of Myrto D. Arapinis and Krystof Bezdek.

All electronic data will be stored on a password-protected encrypted computer, on the School of Informatics' secure file servers, or on the University's secure encrypted cloud storage services (DataShare, ownCloud, or Sharepoint) and all paper records will be stored in a locked filing cabinet in the PI's office. Your consent information will be kept separately from your responses to minimise risk.

What are my data protection rights?

The University of Edinburgh is a Data Controller for the information you provide. You have the right to access information held about you. Your right of access can be exercised in accordance Data Protection Law. You also have other rights including rights of correction, erasure, and objection. For more details, including the right to lodge a complaint with the Information Commissioner's Office, please visit www.ico.org.uk. Questions, comments, and requests about your personal data can also be sent to the University Data Protection Officer at dpo@ed.ac.uk.
For general information about how we use your data, go to: edin.ac/privacy-research

Who can I contact?

If you have any further questions about the study, please contact the lead researcher, Krystof Bezdek (s2089135@ed.ac.uk).
If you wish to make a complaint about the study, please contact.
inf-ethics@inf.ed.ac.uk. When you contact us, please provide the study title, and detail the nature of your complaint.

Updated information.

If the research project changes in any way, an updated Participant Information Sheet will be made available on http://web.inf.ed.ac.uk/infweb/research/study-updates.

Consent

By proceeding with the study, I agree to all the following statements:

- I have read and understood the above information.
- I understand that my participation is voluntary, and I can withdraw at any time.
- I consent to my anonymised data being used in academic publications and presentations.

- I allow my data to be used in future ethically approved research.
  [Button here named “I agree” or “take me to the survey” as part of the online questionnaire]