Designing a Tool to Teach Password Security to Future Developers

Constance Crowe

MInf Project (Part 1) Report

Master of Informatics
School of Informatics
University of Edinburgh

2017

Abstract

I describe how I created a system which allows users to try out some basic password cracking techniques. By teaching the users how to break into a "secure" site by attacking the password, I demonstrate potential vulnerabilities from an attacker's point of view, and how they can be solved by a developer. The system is built for beginners, specifically first and second year students with some programming experience.

The system comprises six progressively more difficult levels which demonstrate common develop mistakes and how they might be fixed. Important features include introductory texts, hints, end-of-level explanations and sample solutions.

I evaluate this system with regard to its usability in order to ensure that I have a stable foundation on which to build next year. While the system was found to be simple and easy to use overall, further testing will be required.

Acknowledgements

Initially I would like to thank everyone who participated in the evaluation of my system, especially the five Think Aloud participants and the expert users.

I would like to thank my supervisor, Kami Vaniea, for her guidance and support throughout this project.

I would also like to thank Simon Rovder, Paul Sinclair, and my dad for taking the time to proofread this report. Thanks to my friends and family for all their support.

Finally, thanks to Mattias Appelgren, for all his help and support (and for cooking for me).

Table of Contents

Chapter 1

Introduction

With approximately 45% of the population of the world connected to the internet [9], cyber security is an increasingly important issue in everyday life. Last year, the Cyber Security Breaches Survey revealed that 65% of large companies detected that they were the target of cyber attacks, and furthermore that 25% of these companies confessed to succumbing to security breaches at least once a month [30]. These values are already considerable enough without even taking into account the no doubt substantial number of attacks which went undetected.

While it is true that some breaches can be traced back to carelessness on the part of the end-users, for example phishing attacks [20], ultimately the fault for the vast majority of large-scale breaches lies with the developers. Attacks which target users directly are not very common, so users tend to consider the effort required to secure their password too great in relation to the probability of an attack [27]. On the other hand, developers often fail to take appropriate steps towards securing their users' passwords. For example in 2012, 6.4 million LinkedIn passwords were leaked [7]. They were unsalted (see Section 3.2.2) and hashed only under SHA1. As a result it is easy for an attacker to obtain the plaintext of the passwords.

This apparent lack of functional security measures can largely be attributed to a general lack of security experts. A survey by Intel Security revealed that 82% of the respondents from within companies felt that the nation's workforce as a whole suffered from a shortage in cyber security skills. It went on to estimate that globally there will be from one to two million unfilled positions in the field of cyber security by 2019 [25].

One solution to this shortage would be to increase computer security education in schools and universities, which would in turn increase the potential number of skilled workers. There is a worldwide lack of computer security modules in university curricula. As was reported by CloudPassage, only one of the 36 top ranked US colleges required students to take a class on computer security, while three of the top ten do not even offer security classes [4]. This would suggest that a student who is interested in computer security would have to seek other sources in order to learn more about the subject.

While textbooks may offer theoretical explanations of core concepts, they do not allow for hands-on experience. This naturally leads us to explore online resources for learning about computer security, as online sources have the advantage of being more interactive and thus have the possibility of offering concrete demonstrations. Nevertheless, most readily available systems do not offer much guidance, and cover only more advanced material, as they assume some prior computer security background (see Section 2.4). This is why I aim to create a system for teaching computer security which requires little programming experience, and no prior knowledge of security.

1.1 Project Aim

The aim of my project is to create a system which allows users to try out some basic password cracking techniques. By teaching the users how to break into a "secure" site by attacking the password, I aim to demonstrate potential vulnerabilities from a hacker's point of view, and how they can be solved by a developer.

The target audience for my system is first and second year undergraduate (UG) students. Some experience in programming is expected of them, as some of the more advanced attacks inevitably involve writing scripts. This demographic represents potential future developers who have started to learn about software development but who are unlikely to have learned any computer security yet, as security modules are typically taught in later years [4]. I hope to create a system specifically to aid these beginners to learn more about the area, and the intricacies of developing with security in mind.

In light of this, I have defined a series of concrete learning outcomes which should be achieved by using my system:

- learning what is considered a good password, as opposed to a weak password
- understanding how cryptographic hash functions and salting work, and why they are important for secure password storage
- understanding and being able to perform a brute-force attack, a dictionary attack and a rainbow table attack, including the differences in time complexity of each of these attacks.

1.2 Report Structure

This report covers the progress I have made this year. The remaining 7 chapters are structured as follows:

Chapter 2: presents systems which have desirable features similar to my system, and how they differ from my own system.

Chapter 3: presents an overview of passwords and explains common attacks and developer mistakes which allow these to be possible.

Chapter 4: describes the iterative design process used to develop the requirements and design of the system.

Chapter 5: describes how I implemented the system, and presents the main features of the interface.

Chapter 6: critically evaluates the system with regards to its usability through the use of the System Usability Scale and Think Alouds.

Chapter 7: presents the results of the evaluation and discusses these in relation to the overall usability of the system and the design requirements established in Chapter 4.

Chapter 8: concludes with the further work required for this project.

Chapter 2

Related Work

2.1 Puzzle-based Learning

Falkner et al.[22] take the view that the form of problem-solving which is most often taught is far too constrained, and as such is not an efficient method for teaching students. The students never learn how to think about the problems in general, and therefore are never properly prepared for addressing real-world problems.

In response to this they suggest puzzle-based learning. They developed a course to be taught in university based on the textbook by Michalewicz and Michalewicz [33]. Using their definition, an educational puzzle should:

- be general enough so that strategies that the students discover can be used for future problems
- be simple and easy to remember, thereby increasing the probability of the students recalling its solution method
- be entertaining, so that the students will not lose interest
- have a "Eureka Factor", so the solution should not be obvious, but when the correct approach is found, there is a moment of revelation.

By using puzzles which are engaging, thought-provoking and educational, Falkner et al. were able to foster critical thinking skills in an engaging and interesting way. Ultimately, the students gained a deeper understanding of the material being covered. These are aspects that I capture in my own system, as I want the user to understand what they have learned and how this can be applied to other problems. However, in contrast to Falkner et al. [22], I have a more hands-on approach to learning as my system also captures elements of game-based learning.

2.2 Game-based Learning

Educational computer games are being developed to be used in a wide range of domains, including military training [28], anti-bullying education [16] and science education [32]. McQuiggan et al. [32] conducted an empirical study of one such system, *Crystal Island*. *Crystal Island* is a game based on the North Carolina eighth-grade microbiology syllabus. The player is tasked with discovering the cause of a mystery illness, in order to provide an engaging setting in which to learn microbiology.

McQuiggan et al. split their participants into groups, in order to compare the performance of the students who learned microbiology by using *Crystal Island* to those who learned through traditional classroom methods. To evaluate this, they carried out a series of pre- and post-tests, including a "Presence Questionnaire" and a set of 23 questions assessing their knowledge of the microbiology curriculum.

It was found that, overall, the students who played *Crystal Island* reported significant increases in both motivation and engagement when compared with the students in the traditional learning group. On the other hand, the traditional learning students did exhibit higher learning gains as they performed better in the final test than their counterparts, though this is not to say that the students who played *Crystal Island* did not also perform better in their post-tests than in their pre-tests. They found that these results are common to most game-based learning environments.

From here it becomes a decision between prioritising higher learning outcomes, or improved engagement and motivation. Since my target audience is made up of beginners, it will be crucial that the users are motivated to learn. As such, the high level goal of *Crystal Island*, teaching in an engaging environment, is the same as that of my system. My approach is therefore designed to be similar insofar as there is narrative and in-game explanations.

2.3 Learning Computer Security through Games

Chapman et al. [19] incorporate these game-based elements in their *PicoCTF* game. *PicoCTF* is a computer security competition aimed at high school students across a range of levels of experience. It is based on traditional security Capture-the-Flag (CTF) games in which the participants must obtain flags (a specific string of characters) by completing levels in order to score points. A narrative thread is upheld throughout the game in order to keep the players interested and engaged. I want to capture this same feeling of sustained engagement and learning, albeit without the pressure of competition as my system is first and foremost for learning.

Similarly, the *CPE123* gaming infrastructure, developed by Flushman et al. [24], is a system used in classroom settings to allow students to have hands-on, interactive experience with applying knowledge they have previously learned. It aims to encourage adversarial thinking through a set of increasingly difficult levels. By scaffolding the learning, they aim to achieve a deeper, more comprehensive understanding. Critically,

there is a clearly-defined safe space in which the students can experiment without fearing any negative repercussions. They found that the use of their system resulted in increased interest in the area of computer security, and that all the students felt that they had achieved significant learning gains.

This is the system I use as a basis for my project. It encompasses the features that I have identified as desirable, namely puzzle-based and game-based learning in a set of progressively more difficult levels, thus making it suitable for beginners. The main differences between this system and my own are the ultimate goals. While *CPE123* is a system for UG1 students to apply knowledge that they have previously gained, my system will be aimed at teaching UG1/UG2 within the system itself, with a specific focus on developer mistakes and solutions.

2.4 Other Similar Online Systems

There are a number of online computer security resources which present the user with security challenges. For example, Google's *Gruyère* [13] is a web application which demonstrates vulnerabilities and how to defend from them. The site offers good explanations of the concepts being taught, before allowing the user to try them out. While my system follows the same overall structure, the target audiences of the two systems differ. *Gruyère* expects a lot more prior knowledge and does not provide a great deal of guidance, as it is not primarily a system for teaching, but rather for exploration.

Similarly, *HackThis* [6] or *CyberChallenge UK* [11] offer challenges to be solved. They do not provide any surrounding explanations or tutorials but expect the users already to have this knowledge. The emphasis of these sites is placed on challenging the user, and it might be that not all users are expected to solve all challenges successfully. In contrast, my system focuses on teaching and ease of access for beginners, so while the puzzles are similar in style to those of these sites, all my users should be able to solve all my puzzles.

2.5 Summary

Puzzle-based learning has been proven effective in engaging students in such as way that they achieve a deep understanding of the subject material, along with essential critical thinking skills. I take this approach, as used by Falkner et al. [22], and adapt it to a more hands-on environment. Meanwhile, game-based learning is engaging in a different way and leads to students feeling more motivated to learn through interaction with an immersive system [32], so aspects of this are also incorporated.

The end result is a system similar to the *CPE123* [24] or *PicoCTF* [19] systems. They target UG1 and high school students respectively in environments which aim to apply knowledge which the students have previously learned. In contrast, I create a system based on similar principles, but which provides full explanations for each level, specifically from the point of view of a developer.

Chapter 3

Password Security

Text-based passwords are the most common method of authentication in computer and web systems [29]. While system-generated passwords are more secure than user-selected passwords, they are far harder to remember for the users. As a result, the majority of passwords are chosen by the users, and are as a result potentially insecure.

3.1 Password Strength

One of the main sources of weakness in passwords, alongside low entropy due to short and predictable passwords, is the use of easily found personal information or of words taken from a dictionary. According to a study conducted by Morris and Thompson [34], over 85% of the three thousand passwords they examined were either words from an English dictionary, the reverse spelling of these words, first or last names, cities, or telephone numbers. These passwords are weak as they can be easily compromised, either through the use of software or even through social engineering [34]. The NIST Digital Identity Guidelines provides instructions on how users should create their passwords [5]. By this definition, we consider a strong password to consist of at least 8 characters, a mix of uppercase and lowercase letters, digits and special characters.

However, as we saw in Chapter 1, Hearley suggests that leaving password security in the hands of the users is a waste of time [27]. According to statistics, most users care little about password security, and consider the effort required to secure their password too great in relation to the probability of an attack. Adams and Sasse present similar results in [14]. This is why we want to concentrate on teaching developers how to secure passwords, so that attackers are unable to break into sites and cause large-scale leaks, even in the event of user complacency.

3.2 Password Storage

Secure websites should never store user passwords in plaintext (a fully legible password), but should instead store the salted hashes of the passwords. In Linux systems, the salted hashes of the system passwords are stored in the `/etc/shadow` file, known as the shadow file.

3.2.1 Hash Functions

A cryptographic hash function takes a message of arbitrary length and outputs a fixed-length string. Any changes in the input message will result in a different output string. It also has the following properties [15]:

- It is a one-way function, that is, a function which is easy to compute but hard to invert. This means that if we have the hash value of a message, it should be very difficult to retrieve the original message from it.
- It is collision resistant, meaning that it is very difficult to find two different messages that produce the same hash value.

Since an attacker should not be able to reverse hash values to obtain the original messages, this is more secure than storing passwords in plaintext. The Rockyou leak is an example of a company not hashing their passwords. The contents of their databases were leaked and since the passwords were not hashed, 32 million passwords were released to the public [1].

The most common mistake when using hashes - other than not hashing passwords at all - is using a weak hash function. A weak hash function may be a function for which collisions have already been found or which is very fast to compute (eg. MD5 or SHA1 [12]). This is the mistake LinkedIn made, by only hashing the passwords under SHA1 [7]. Additionally, it should be ensured that the hash values cannot be accessed by the attacker. It is surprisingly common that attackers are able to acquire hash values through error messages due to developer mistakes, and this facilitates attacks [8].

However, hash functions are deterministic, so a given input will always produce the same output. As such, if two users choose the same password, this will be visible in the database where the hashes are stored, since the hash values will be identical.

3.2.2 Salting

A salt is a randomly generated string which we append to the password before hashing it. As a result, the hash values of identical passwords will end up being different.

This essentially negates the effectiveness of rainbow tables (see Section 3.3.3), as two users with the same password will no longer have the same hash. Since the salt is random, there is no way to predict it in advance and pre-compute the rainbow table, thus rendering this attack useless [12]. This is why salting hashes is so important - it dramatically increases the overall computational complexity of finding a match for the hash.

A common mistake when salting is using the same salt in each hash. This is clearly useless as two users with the same password will still have the same hash. Another oversight is using a salt which is too short. A short salt can be brute-forced, so an attacker could pre-compute a lookup table using all possible hashes.

3.3 Common Attacks

Password cracking is the practice of recovering passwords from data that has been stored or transferred by a computer system. Some common attacks include brute-force, dictionary and rainbow table attacks [29].

3.3.1 Brute-Force Attacks

Brute-force attacks are one of the most popular password cracking methods. They use a trial-and-error approach by systematically submitting all possible passwords combinations until the correct one is found [12]. The amount of time needed for a brute-force attack increases exponentially in relation to the length of the password.

One way of limiting the ability of an attacker to perform brute-force attacks is by implementing rate limiting. Rate limiting controls the rate of network traffic which is being sent or received. This reduces an attacker's ability to use online trial-and-error type attacks, where the attacker is actively submitting passwords to a website.

3.3.2 Dictionary Attacks

A dictionary attack is performed by running through a list of potential passwords until the correct one is found. Dictionary attacks rely on the fact that users often choose common passwords (or variations thereof - by appending a special character or digit to the beginning or the end of the password), and therefore enable us to search through the most likely options [12].

The list of leaked Rockyou password is often used as a wordlist for dictionary attacks [10]. Further modifications can be made by substituting common symbols for similar letters. The English dictionary is also often used.

The easiest way to defeat a dictionary attack is to implement rate limiting, and to not allow the user to choose a password which is likely to be included in one of these lists.

3.3.3 Rainbow Table Attacks

A rainbow table is essentially a dictionary which stores a list of common passwords alongside their hashes. It is optimised for hashes and passwords so maintains a fast look-up speed, despite the space it takes. An attack which uses this table is a rainbow

table attack. It is used when the attacker has somehow acquired the hash value of the user's password and needs to know what the plaintext is. Additionally, since the attacker does not have to keep submitting passwords, this attack is much faster than the trial-and-error type attacks we have seen previously. It is an offline attack as all the work is done offline, and only the correct password is submitted [12].

Using salts reduces the effectiveness of rainbow tables, as we outlined in Section 3.2.2.

3.4 Summary

In this chapter we covered some important points about passwords. We saw that password strength depends on length and that the passwords themselves should not be stored, but rather the salted hash value should be. The common developer flaws which are linked to this are:

- not hashing passwords
- using weak hash functions
- leaking information in the error message
- not using a salt
- reusing salts
- using short salts.

Brute-force attacks, dictionary attacks and rainbow table attacks can exploit these vulnerabilities.

Chapter 4

Design

4.1 Initial Concept Design

The issue that I attempt to address is the fact that developers do not know very much about password security. The lack of interactive, entry-level tools which teach the required skills exacerbates this problem. Thus the research question to which I aim to provide an answer to is:

> "How might we design an entry-level tool to teach developers about password security?"

My proposed solution is to create a system which teaches core password security concepts by first teaching how to crack these passwords. I take inspiration from the *CPE123* gaming infrastructure [24] presented in Chapter 2. I met with one of the authors of this paper, Zachary Peterson, and was shown their system in more detail.

A key point which arose as highly valuable from this demonstration was the presence of structured, increasingly difficult levels. The system starts with a very easy level which only requires a single digit as the password. It then becomes progressively more complex, and later requires 3 ascii characters. This was shown to be very effective for providing an easy introduction which still highlighted important security concepts. The more advanced levels required the users to script their own attacks using Python. Peterson explained that they considered basic knowledge of programming to be essential for computer security, which is why they did not shy away from including it.

The main point at which this system and my own diverge is at the learning outcomes. Peterson's game put more focus on exploration and application of prior knowledge than my own does. Accessibility and developer-oriented teaching are the core aspects of my system, which were not priorities in the *CPE123* system.

From here, a first set of high level goals for my system design can be drawn, inspired by the *CPE123* structure:

- present a set of progressively more difficult levels
- be accessible to beginners
- explain core concepts, specifically in terms of the impact for developers.

4.2 Paper Prototyping

Paper prototyping is a user-centred design process in which the designer draws up some rough sketches of possible product interfaces, and asks users to evaluate them. This process is helpful as it allows the designer to test possible designs without having to first build them. The users are also more likely to focus on the overall design rather than the details, as they are being presented with a hand-drawn draft and not something that looks like a finished product.

From the high level objectives given in the previous section, I developed a set of possible designs. By iterating on these design ideas, I refined the overall design to better fit the users' requirements. A sketch of each prototype interface was produced and I proceeded to survey members of my target audience. The users were asked to interact with the paper designs, as though they were fully functional interactive interfaces.

4.2.1 Design 1 - Graphical Keypad

The first design (see Fig. 4.1), was heavily influenced by Flushman et al.'s *CPE123* game [24]. It features a graphical keypad, with which the users could interact in order to input the level password. The test users enjoyed the game-like aspect and familiarity of this design, and the discovery which accompanied trying to figure out how to interact with it. However, some users did point out that, while they liked the keypad, they would much prefer a more standard input form. One further potential issue with this design is the lack of consistency across the levels. There would be no real way to make a keypad scale to represent alphanumeric passwords.

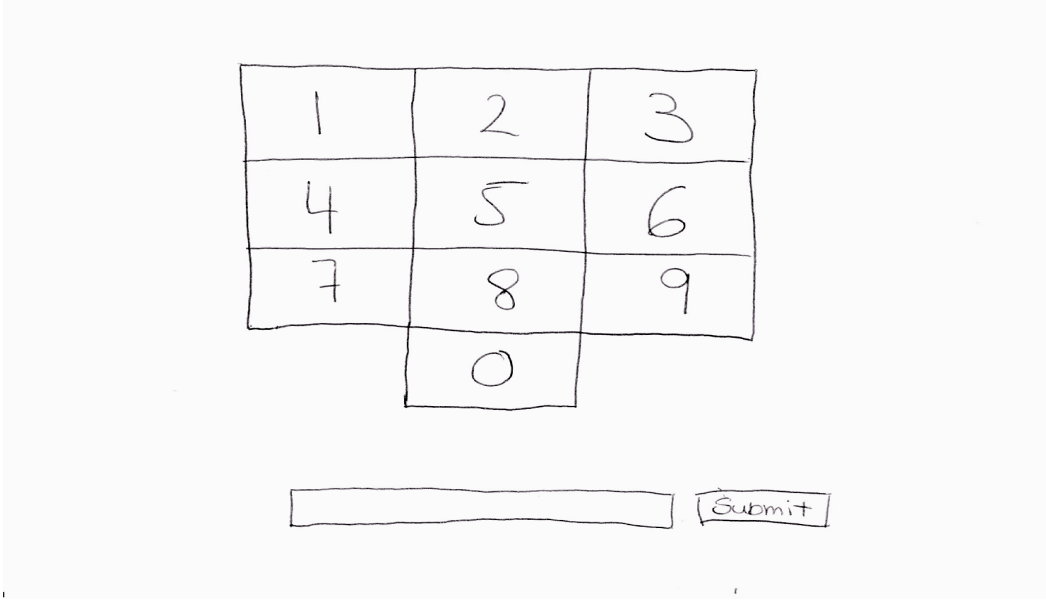

Figure 4.1: Sketch of possible Design 1, using a keypad

4.2.2 Design 2 - Storyline

The second design (see Fig. 4.2) is more minimalistic, and presents only the password input form alongside a short text. The text is intended to present the level, and to provide a small hint as to what is expected of the user. More importantly though, it should enable a clearly marked progression between the levels. The story should tie all the levels together and provide a better sense of continuity. The users were generally positive about the presence of this story, but did not like how it interfered with the main area of the interface.

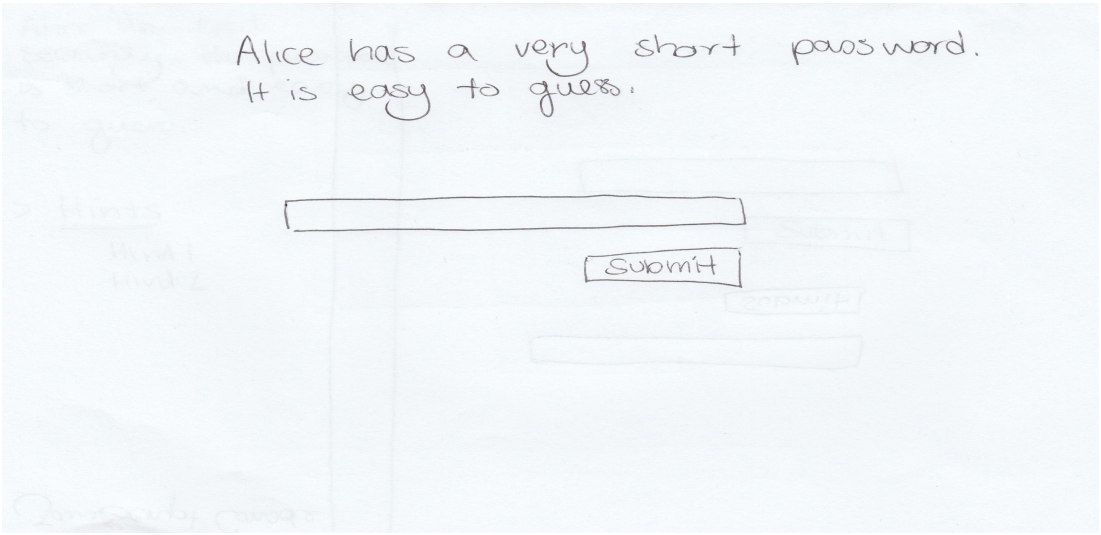

Figure 4.2: Sketch of possible design 2, with a short text above password input form

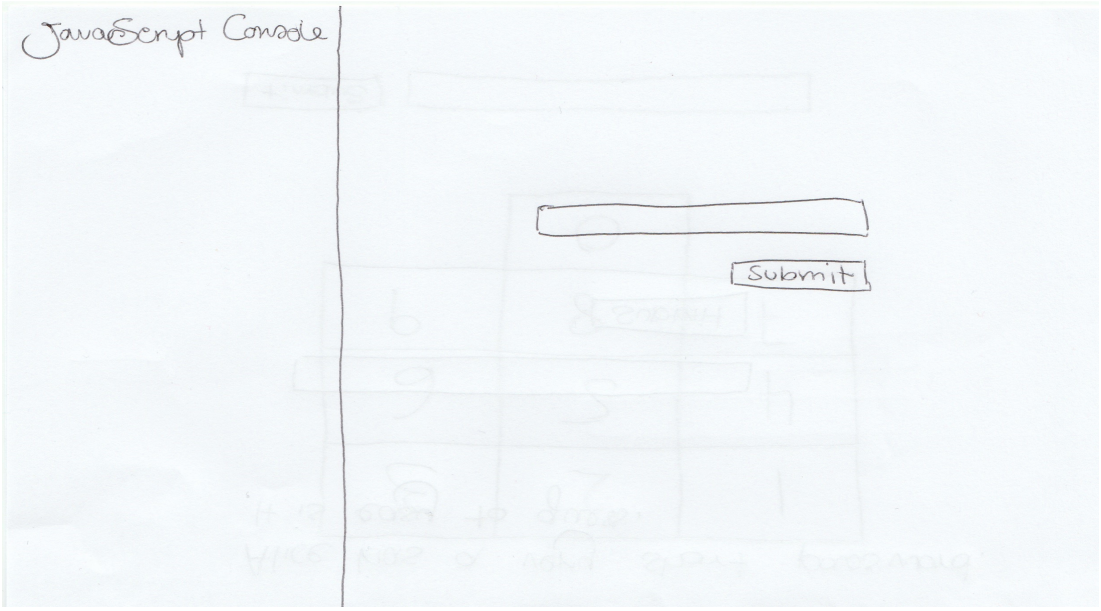

Figure 4.3: Sketch of possible design 3, with an on-screen console

4.2.3 Design 3 - On-screen Console

The third design (see Fig. 4.3) includes a Javascript console on the left hand side of the screen. This console would be present for levels which require some coding. The users liked the general idea of this, as it gave a small indication as to what the solution for the level was going to be. It also had a more obvious use than the keypad from Design 1 (section 4.2.1). However, many users found the prospect of using Javascript intimidating, and felt it left little space for additional information or support.

4.3 Requirements

Using the results of this prototyping, and the initial objectives, I was able to refine the users' requirements into a set of design requirements for my system:

Requirement 1: The system will be accessible to beginners.

As we mentioned in Section 1.1, the target audience of this system is UG1/UG2 students who have some experience in programming. As such, the system must be accessible and easy to use for beginners in computer security. This implies that the system must contain clear explanations of security terms and concepts.

Requirement 2: The system will present a set of puzzles/challenges.

By presenting a series of puzzles, I aim to promote independent critical thinking, and general problem-solving skills. Students tend to be more motivated when faced with a series of challenges and this should improve the overall engagement with the system [23] (see Section 2.1).

Requirement 3: The system will consist of progressively harder levels.

It has been demonstrated that scaffolded learning leads to an overall deeper understanding of the subject being taught [26]. With this in mind, and taking inspiration from Flushman et al. [24], the levels should be scaffolded. Each should build on the last and become progressively harder.

Requirement 4: The system will provide the user with hints.

When users are stuck, they tend to become frustrated or bored. These emotions can have a negative affect on concentration and enjoyment, so we want to regulate this as much as possible by providing help when needed [36].

Requirement 5: The system will provide a sample solution for each level.

As the users are beginners, in order to scaffold their learning efficiently (see Requirement 3), we should provide a suggested solution at the end of each level. This way they will learn what the "best" way to solve that level is and they can apply this knowledge to the next level.

4.4 Final Design

I developed a design which incorporates all of the user comments and the best features of each previous design from Section 4.2 to be used as the basis for the implementation. The graphical keypad was eliminated, due to the lack of consistency which would emerge across the levels. The console was eliminated for similar reasons, as well as those stated in Section 4.2.3. However, the storyline was included, as it represents an important feature of the system - the progression in difficulty and knowledge across levels. This feature was shifted to a sidebar on the left, in accordance with the users' desire for a more clearly defined space in which to work (see Fig. 4.4).

A set of hints have been added to the sidebar, to conform with Requirement 4. Requirement 5 is met by inserting an explanatory section between each level, which appears after completing one level, and before progressing to the next. This section explains the core concepts that have been demonstrated in each level in more detail, and why these concepts are important in the real world.

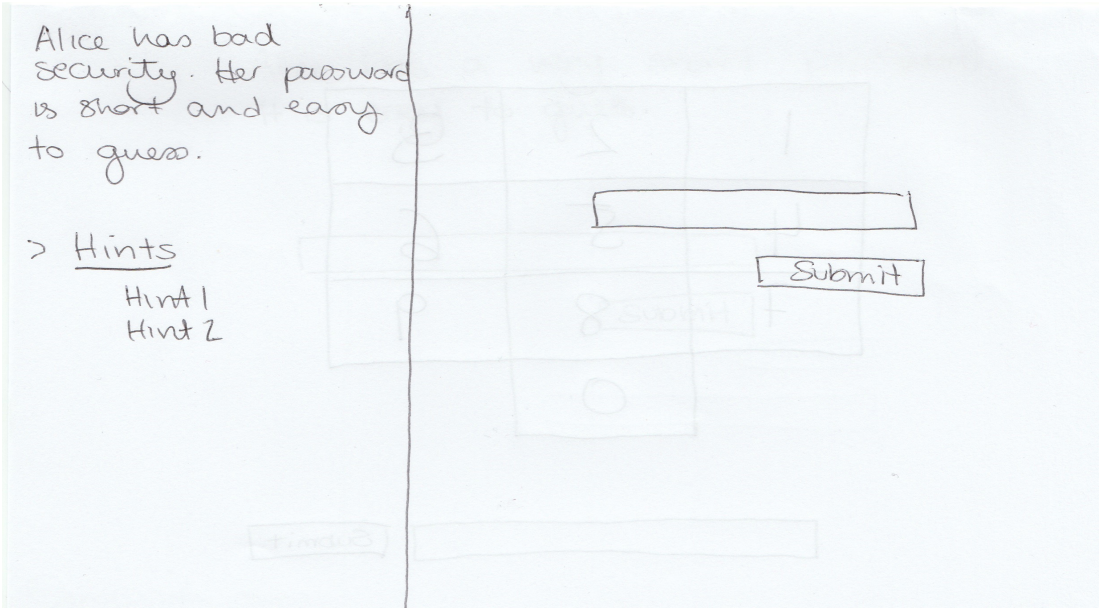

Figure 4.4: Sketch of final design after paper prototyping

4.5 Levels

Each level will feature its own input form, introductory text and hints. A level is completed when the user enters the correct password. They are then able to progress to the explanatory break-down of the level, and from there they can access the next level.

The levels will focus on covering the attacks and common developer issues explained in Chapter 3. As such, the main topics I want to cover are:

- not hashing passwords
- using weak hash functions
- leaking information in the error message
- not using a salt
- reusing salts
- using short salts.

From this, I designed six levels.

Level 1 - Manual Brute-Force Attack

Taking inspiration from Flushman et al. [24], the password for the first level is a single digit. This level should be very simple and not take long to complete. To clearly restrict the search space for this level, the input form only accepts digits.

From this, the user should realise that it is possible for an attacker simply to guess passwords (if the search space is fairly limited) by trying all possibilities. The level also highlights the fact that the time needed for a brute-force attack increases exponentially with respect to the length of the password. For example in this case the password is only a single digit, so there are a total of 10 possibilities.

Level 2 - Casing

This level does not have a password, but shows the password form of Level 3 alongside some questions.

Casing a website is the practice of exploring a website before attempting to compromise it. By doing this, an attacker can ascertain how the website functions, and identify potential vulnerabilities. In this level, the users are asked to case Level 3, and correctly answer a set of questions about it and the composition of the password before being allowed to progress. As such, they enter Level 3 with knowledge of how its flaws can be exploited.

Initially, I only planned for five levels, and this level did not exist. On reflection, it became evident that the increase in difficulty from Level 1 to the current Level 3 was too great. Instead of expecting the user to do the groundwork for Level 3 in that level, I added a new level (which became our current Level 2) which would be dedicated to casing the level.

Level 3 - Automated Brute-Force Attack

Having completed a manual brute-force attack, the next natural step would be to have the user automate a brute-force attack, as this is how attackers approach these problems in practice. This level reinforces the idea that brute-force attacks are ultimately effective but not very time efficient. As such, the level password comprises three digits. By limiting the search space to digits again, I ensure that the level should not take too long to complete, while still taking more time than Level 1.

Level 4 - Dictionary Attack

Moving on from the last level, the password is more realistic. The password for this level is a common - yet insecure - password, chosen from a list of real leaked passwords. The wordlist I use is the list of leaked Rockyou passwords [10], which I shortened and filtered for profanity.

In this level, I highlight the fact that even if a user does not choose an absurdly short and simple password, it is still possible for an attacker to find it by trial-and-error techniques. The level also emphasises the idea that developers should enforce some requirements with regards to the user's password. The entire burden of making a secure password should not lie with the users, as it has been shown that most users consider this to be too much effort [27].

Level 5 - Hashes

The passwords in the Rockyou list were easily acquired as the developers did not hash them but rather stored them in plaintext. To counteract developers storing their passwords in plaintext, I introduce the concept of hash functions in this level. However, a common developer mistake is to have the hash of the correct password accessible to the attacker. To demonstrate this, the level password is still one from our wordlist, but the hash of this password is visible in the input form's error message.

So that the user cannot repeat the solution used in the previous level, I implement rate limiting (see Section 5.1.3). The user will now have to use an offline attack to complete the level, by creating their own version of a rainbow table, and using it to look up the hash value given in the input form's error message.

Level 6 - Salted Hashed

Finally, I want to highlight the fact that simply hashing a password is not enough, and that passwords should be salted in order to increase the overall entropy. The password for this level is provided in the form of an extract of a shadow file. This also allows the user to learn about how passwords are stored on Linux machines.

4.6 Language

As I previously mentioned, some of the levels require writing scripts to automate attacks. I chose to recommend Python as the main scripting language, with the support of Peterson who also encouraged his students to use Python.

Python is a popular language among beginners, as it is simple and easy to learn thanks to its clear and concise syntax. As such, it is quite likely that the users already have some Python knowledge. Alternatively, concepts learned in other object-oriented languages are easily transferable to Python so it should not be very hard to pick up, especially due to its simple syntax and flexibility. The reverse is also true, so learning Python will not be a waste as it provides a good basis for other languages [37].

As a result, the support provided in the hints includes Python code snippets. The main source of concern with choosing Python is that it requires users to install Python libraries to complete the challenges. I pay particular attention to this point when evaluating the system in Chapter 6, in order to decide whether this causes a major issue or not.

4.7 Summary

In this chapter, we stated that the gap we are trying to fill is the lack of entry-level tools for developers to learn about computer security. Taking inspiration from Flushman et al.'s *CPE123* gaming infrastructure, I came up with a set of high level objectives. Using paper prototypes, and an iterative process, I was then able to design an interface which conforms to these objectives.

Thus, a set of requirements was defined:

1. The system will be accessible to beginners.
2. The system will present a set of puzzles/challenges.
3. The system will consist of progressively harder levels.
4. The system will provide the user with hints.
5. The system will provide a sample solution for each level.

I subsequently created a final design which encompasses all the previous points.

Finally, I explained the different levels that the system has, and justified the reasoning behind each level. The levels are as follows:

1. Manual Brute-Force Attack
2. Casing
3. Automated Brute-Force Attack
4. Dictionary Attack
5. Hashes
6. Salted Hashed

Chapter 5

Implementation

5.1 Back-end

5.1.1 Choice of Framework

The system should be resistant to attacks (other than those described in each level, which naturally are intentional). As such, the choice of framework is important as ideally it should make securing the website a simple task, so that the main focus of the implementation can actually be on building the levels. Python is generally known to be fairly secure (unlike PHP for example [35]) and has a variety of good and easy to use frameworks. It works well for small projects in particular but can easily be scaled up. I am also more familiar with it. As a result, I will be using Python to write my system. The two most popular Python frameworks are Django and Flask.

Flask is a microframework which is very easy to set-up and use. It could be considered more flexible than Django, as most functionalities can be added on with modules. On the other hand, Django has a lot more built-in functionality, especially in terms of security. For example it automatically sanitises user input (to protect against Cross-Site Scripting) and provides easy to use CSRF tokens to be placed in forms (to protect against Cross-Site Request Forgeries). The default values for these settings are well-tested and easily usable.

After considering both options, I decided that Django provided more built-in security, which has been widely tested. By using this framework, I can focus on building the levels themselves instead of worrying about potential unintentional vulnerabilities I might have created in my website.

5.1.2 Database

The solutions for each level are stored in a database. Each level is assigned a name and a password, which is stored as a SHA512 hash. I was unable to correctly salt the hashes as I had no sensible way of subsequently storing the salts. This should certainly be fixed in the upcoming year (see Section 8.2). I chose to use the Python `hashlib` library to hash my passwords.

Password hashing functions differ from cryptographic hash functions as the former will always have a salt and will be repeated for a certain number of rounds. This considerably slows down the computation of the hash and thus the computation speed of any potential attacker. However as a consequence it also takes longer for the legitimate website to verify the password. In most cases this is not an issue, as a user will not notice if logging into a website takes a couple of seconds. However in my website, the user will need to submit many passwords in quick succession, even if it is just to test the site's response. As a result, it was not viable to use password hashing functions (such as those in the Python `bcrypt` library), which is why I chose to use SHA512 from `hashlib`. This is also explained to the user - as it is important that they understand this nuance - at the end of Level 6.

5.1.3 Specific Level Constraints

In order to facilitate the brute-force and dictionary attacks in Levels 3 and 4, the webpage refreshes and the response returns a 401 (Unauthorised) status code whenever an incorrect password is submitted. By using this set-up, the users are able to submit passwords repeatedly until they obtain a 200 (Success) status code. The status code is not visible on the webpage itself, but the user is shown how to find it in the hints of Level 3.

However, to ensure that the users do not just brute-force the later levels, I implemented rate limiting. I used the Django `ratelimit` decorator in order to implement this on Levels 5 and 6. Now, no more than five password submission attempts can be made every minute. Any submissions above this threshold obtain a 403 (Forbidden) response. This can be seen in the browser, as it automatically redirects to a 403 error page. On this page I explain why this has happened to ensure that the user fully understands (see Fig. 5.1).

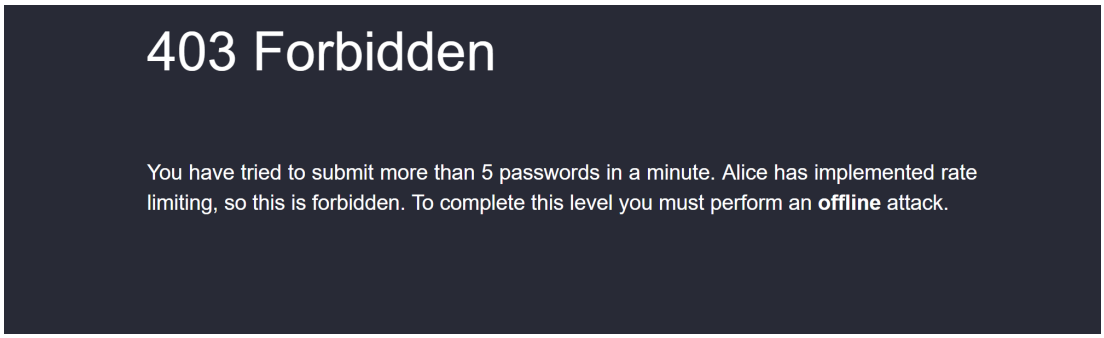

Figure 5.1: Screenshot of system showing 403 Error as a result of exceeding allowed number of submission per minute.

This reduces an attacker's ability to use online trial-and-error type attacks, where the attacker is actively submitting passwords to a website. Thus the user is forced to think differently about these levels.

5.2 Front-end

In order to make styling the website more manageable and consistent, I used Bootstrap [2]. Bootstrap is a web framework for HTML, CSS and JS, built for producing responsive websites. Using this framework enabled me to focus on the overall design of the system instead of having to concentrate on the smaller details, such as making input forms look professional, or having to code functioning modals (pop-up windows).

Each level has its own HTML file defining the actual contents of the level, but the overall style and layout is contained in a single file which is used as a base for all other files. This helps with the maintainability of the code, as it is simple to add new levels, while keeping the overall design consistent. The sidebar implementation is adapted from the "Bootstrap Responsive Sidebar" [3].

5.3 Website Content

5.3.1 Example Level Overview

Figure 5.2 shows the interface as it is shown at the start of Level 5. We can distinguish the same elements in each level. These are the level title, the introductory narrative (see Section 5.3.2), the hints (see Section 5.3.3), and the password input form. The "incorrect/correct password" message appears above this form, and indicates whether the password was correct or incorrect (see Fig. 5.3). Further examples of the system's interface can be found in Appendix A.

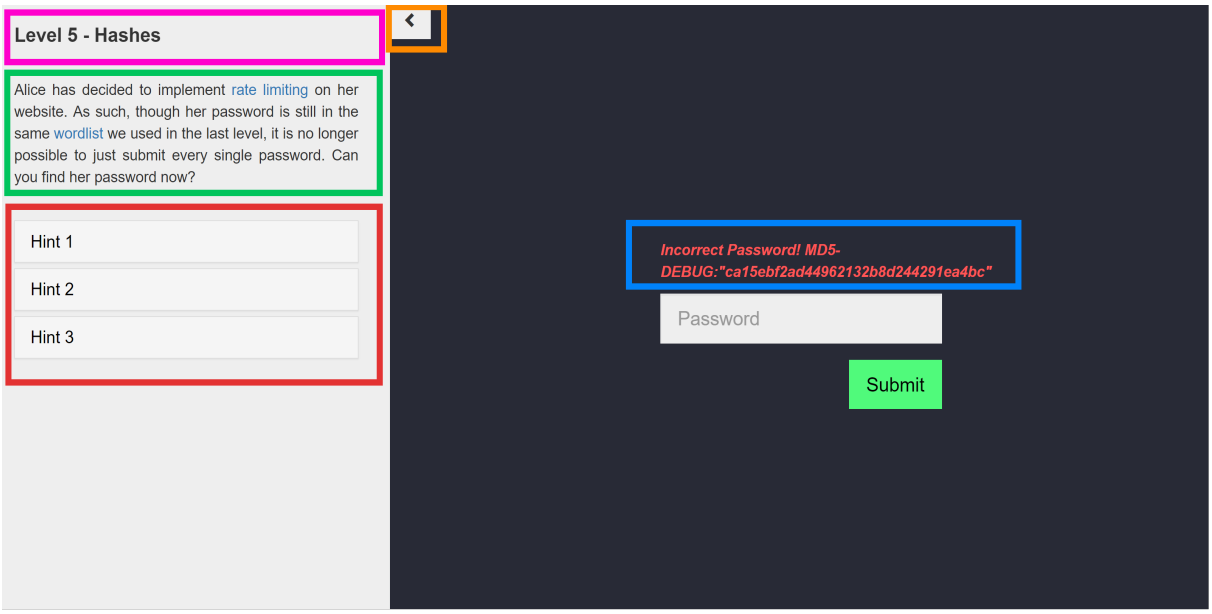

Figure 5.2: Screenshot of system showing the start of Level 5. In pink, the level title. In green, the level's introductory text. In red, the hints for the level. In blue, this level's error message. In orange, the button to collapse the sidebar.

Figure 5.3: Screenshots of possible messages. Top left, correct password message, displayed for each level when the correct password is inputted. Top right, default message when incorrect password is submitted. Bottom left, incorrect password message specific to Level 5 (leaks correct hash value). Bottom right, incorrect password message specific to Level 3 when submitted password was less than three digits.

5.3.2 Narrative Introduction

Each level has a short introduction in the sidebar. Through these introductory texts, we follow the progression of the website of our developer, Alice. Alice's website is hidden behind the level password, so as the user learns about potential vulnerabilities in her site, Alice learns how to protect against these exploits. As Alice is learning alongside the users, it is easy to summarise the learning outcomes of the previous level at the start of the next. This allows me to reinforce the knowledge gained and to provide a sense of progression and continuity between the levels.

Sections which I consider particularly important for the level (generally information about the password itself) are written in bold in order to bring them to the user's attention. These sections are the ones which provide the main hint for the level. Links are represented in blue, and lead either to documentation for a Python module which could be used in this level, or, more often than not, to my own explanations of important terminology (see Fig. 5.4).

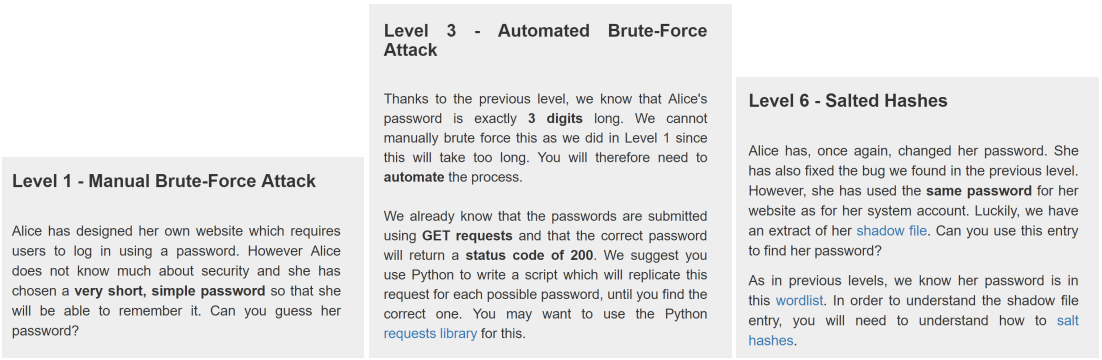

Figure 5.4: Screenshots of the introductions for levels 1, 3 and 6. Note important elements in bold, and links to my explanations/documentation in blue.

5.3.3 Hints

The hints were arguably the most challenging aspect of the interface. From the paper prototyping (see Section 4.2), it became apparent that there were drastically different approaches to how the users would want to use a hint system. Some students felt that in a learning environment, they do not want to get stuck and would happily look at the hints. Others felt that, given the game-like aspect of the system, it was a challenge to complete the level by using as few hints as possible.

As a result, I chose to write a set of numbered hints for each level. Each hint contains progressively more information about the level. The numbers should make it clear that the hints should be opened in order and the collapsible format obscures the hints until they are clicked (see Fig 5.5). In this way, if after the first hint the user feels confident to continue, they can do so without revealing too much additional information.

As an example, here are the hints for Level 1:

Hint 1: Not all characters are accepted by the input form. What kind of characters are accepted?
Hint 2: How many characters are accepted by the input form?
Hint 3: The input form accepts a single digit. This suggests Alice's password is a single digit. If you try all possible digits, you will eventually find the correct password.

Similarly to the introductory texts, key information was in bold and links to explanations I wrote about terms used in the hints were blue. Later levels which involved scripting included hints with snippets of sample code (see Fig. 5.5).

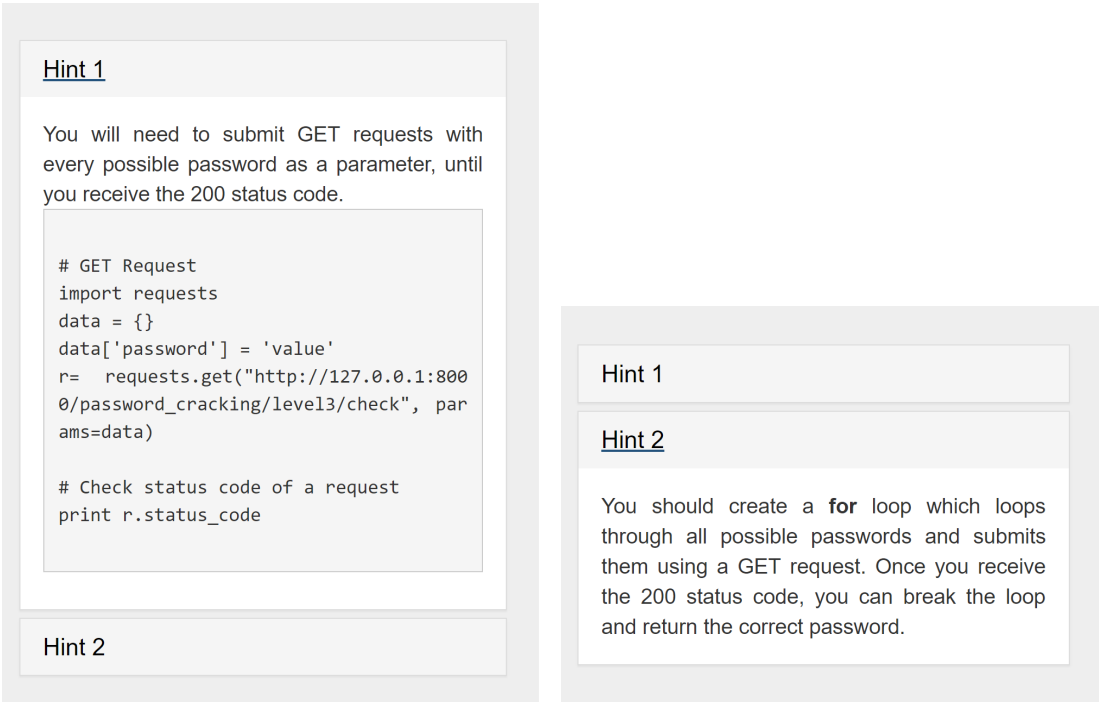

Figure 5.5: Screenshots of the hints for Level 3

5.3.4 Explanations

The explanation for each level appears after the level is completed. It contains an explanation of the attack, highlights the learning outcomes that I specified in Section 4.5, and provides a sample solution which follows the directions suggested by the provided hints (see Fig. 5.6). The user can still close this window and continue exploring the level, for example they may wish to play the level again while bearing in mind what they have just learned, or try out the sample solution and see how it compares with their own. I based my explanations on those I gave in Chapter 3. Similarly, any term which I believed the user might not know, such as "hash function", were explained in a modal, linked to from the webpage in blue.

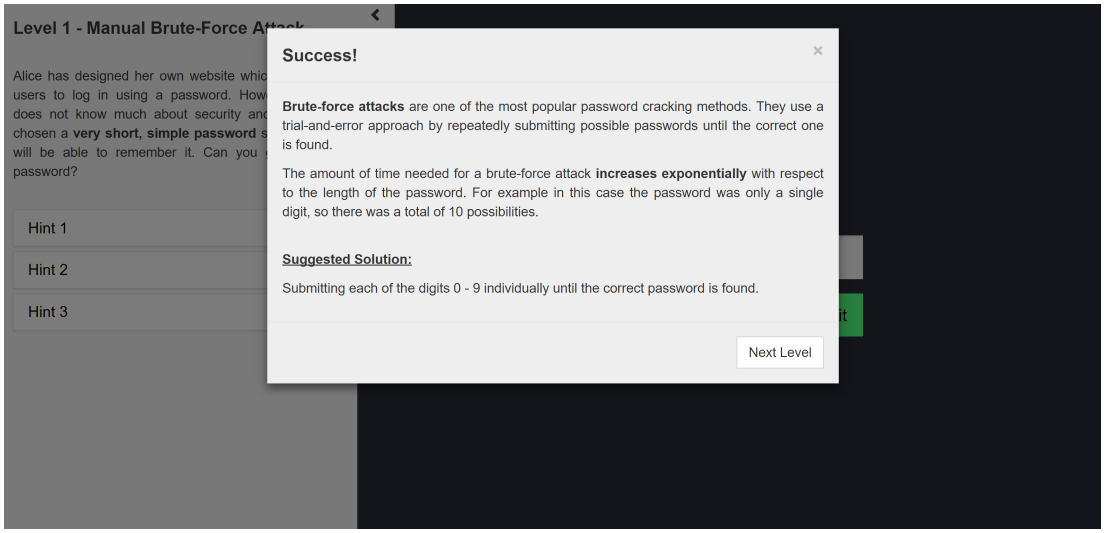

Figure 5.6: Screenshot of system showing the end-of-level explanation for Level 1

5.4 Summary

In this chapter, I explained how I built my interface using Django and Bootstrap, and how I implemented rate limiting and specific status codes to facilitate brute-force attacks.

I provide an overview of the interface of Level 5, highlighting the location of the title, introductory text, hints and error messages. These features are then described in further detail.

Chapter 6

Evaluation

The focus of my evaluations is determining the usability of the system, as I want to ensure that I have a stable foundation on which to build next year. The system must be easy to use and interact with for beginners, and the provided support should facilitate the completion of the levels.

6.1 System Usability Scale

Usability must be viewed in the context of the tool we are attempting to evaluate. As such, we define it as the appropriateness of the system to the context, in terms of effectiveness (the users can complete the tasks), efficiency, and user satisfaction [18].

The System Usability Scale (SUS) has been shown to be a reliable system of evaluation. It covers a variety of aspects which can contribute to usability, such as system complexity, or the need for support and training. It is used once the participants have had a chance to interact with the system which is being evaluated, but before any discussion [18].

Each statement in the SUS has a 5 point scale (a Likert Scale) on which the participant indicates their degree of agreement or disagreement with the statement. These statements vary between ”strong agreement” and ”strong disagreement” as the desired answers to indicate high usability. This variation ensures that the participants carefully read each statement. The individual results are not significant in themselves, but rather the SUS yields a score from 0 to 100 which can be used as a composite measure of the overall system usability.

I conducted a survey of 22 participants during the course of a project presentation day organised by the university, open to the entire student body. The participants were invited to interact with the finished system, until at least Level 3. They were then invited to anonymously fill out a printed copy of the SUS survey (see Appendix B), in accordance with the University of Edinburgh School of Informatics ethics process (RT1746).

It is worth noting that the participants came from a variety of year groups, and not solely from my target audience. Of the 22 participants, 11 were UG3 students, 5 were UG4, 2 UG1, 1 UG2, and 3 identified as “Other”. However, as we are mainly aiming to ascertain the potential of this system and the overall stability of the design - rather than the efficiency of its teaching methods - this will not negatively impact our results.

The SUS scores of the 22 participants ranged from 62.5 to 100 (see Fig. 6.2). It is important to remember that this is a score, and not a percentage. To give more context to these values, Bangor et al. suggested a way to associate SUS scores to letter grades or adjectives [17]. The average score given to my system was 87.25, which puts the system just above “Excellent” (see Fig. 6.1). This, alongside the overall distribution of scores, would suggest that the participants found my system to be very usable and not overly complicated for the task [17]. The comments I received confirmed this trend, as the participants considered the system attractive and simple to use. However, one point of concern was that most users experienced confusion over the fact that letters do not display in the input form of Level 1. This is taken into account during further evaluation (see Section 6.3).

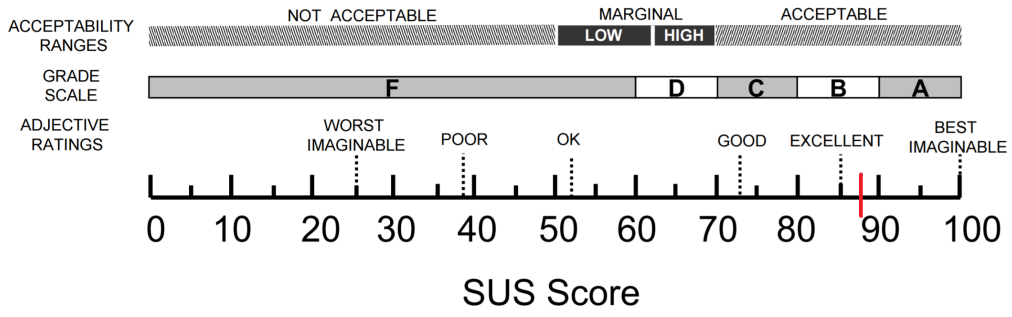

Figure 6.1: Comparison of average SUS scores to adjective ratings, acceptability scores and school grading scales, taken from [17]. Marked in red is my system’s average SUS score.

6.2 Expert Evaluation

I consulted both a lecturer in Computer Security and a security practitioner to evaluate my system from an expert’s point of view. Overall, both were generally positive, especially in regards to the goal of teaching developers. The security practitioner highlighted how important user awareness is, and suggested that this project could be a small step towards helping the public become more informed.

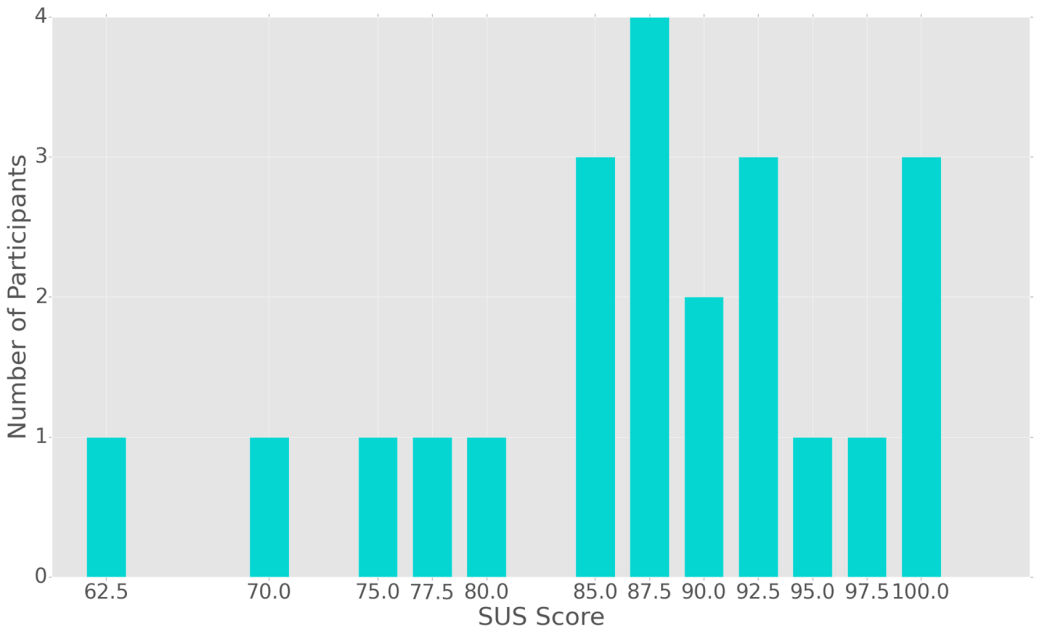

Figure 6.2: Histogram of SUS scores given by participants during the project presentation day

The lecturer was presented with the finished interface, and she interacted with it as an end-user might. Though she was positive overall, and did not find any major flaws, she did bring to my attention some points which may cause confusion or problems for the target users.

In Level 1, our expert user was briefly confused by the fact that letters do not appear in the input form (it only accepts digits). She suggested that this might not be obvious enough, and that it might be worth adding a clear indication of this constraint. Furthermore, the way to solve Level 5 might not be very obvious. She liked the idea of having the hash value revealed in the "Incorrect Password" error message, as finding leaked information in error messages is a key part of web security. However since there was no previous incentive to check the error messages, she pointed out that the users might not realise that they should check it in this level. I paid particular attention to these points when evaluating the system with real users in order to determine whether they negatively impacted the performance of the users.

Despite this, the expert user liked the use of a quiz in Level 2, as quizzes promote more active thinking so that the focus does not stray from the aim of the level. She also mentioned that she felt the explanations seemed very clear and approved of the inclusion of a suggested solution.

Finally, the expert user suggested some further improvements which could be considered for future work. In a first instance, having a Python interpreter within the system's webpage would be helpful for beginners as it would eliminate the need to install packages. Also, while this seems to have the potential to be a good teaching tool, it would be interesting to have a second marker section in which the users would apply the knowledge they have just acquired.

6.3 Think Aloud

Think Alouds are one of the most common evaluative methods in the usability community. They involve setting the participant a series of tasks, and asking them to vocalise what they are doing, thinking and feeling. It is particularly useful to identify aspects of a system which cause confusion, frustration or even delight. It also allows the researcher to view the process of task completion in its entirety [31].

Since I was focusing on the overall usability of the system, I chose to conduct a series of Think Alouds. I aim to locate areas of key problems by identifying which features of the system might frustrate or slow down a user. All the participants will use the exact same version of the system, and no modifications will be made from one participant to the next.

6.3.1 Preparation

In preparation for the Think Alouds, I wrote a script to be read aloud to each participant. The script includes a presentation of both myself and my project, followed by an explanation and demonstration of what is expected of the participant which has been adapted from Ericsson and Simon [21]. After obtaining the participants' consent, I present the task to be completed. The task for my Think Alouds was "complete them [the six levels in the system] all". The full script is included in Appendix C.

Particular attention was afforded to how the users interact with specific aspects of the system, namely the requirements and design issues outlined in Sections 4.3 and 6.2:

- the perceived difficulty of the levels (Req. 1, 2 and 3)
- the installation of Python libraries (Req. 1 and expert user advice)
- the reception/use of the hints (Req. 4)
- the understanding/user of explanations (Req. 5)
- the approach to error messages (expert user advice)
- confusion in Level 1 (expert user advice and comments from SUS survey)

6.3.2 Experiement Details

The participants were all UG1/UG2 students who volunteered to take part in the study. Each Think Aloud lasted around an hour, and I took notes on the participants' behaviour and thoughts. Table 6.1 shows the gender, year of study, time taken in minutes to complete all six levels, and the SUS score of the survey they subsequently filled out. The results of the experiments are presented in the next chapter.

Participant	Gender	Year of Study	Time taken (mins)	SUS score
P1	M	1	49	95
P2	M	1	50	95
P3	M	2	46	67.5
P4	F	2	55	90
P5	M	2	69	90

Table 6.1: Table showing the gender, year of study, total time taken to complete all six levels of the system (in minutes) and SUS score (from the survey they filled out) of all five participants of the Think Alouds

6.4 Summary

In this chapter, I presented the survey of 22 participants which I carried out during a university project presentation day. Each participant filled out the SUS survey after interacting with the first two levels. The average score of 87.25 seems to indicate that the participants found the design usable and suited to the task.

I then provide an overview of my meetings with expert users who gave me some feedback on the system. The lecturer in Computer Security highlighted some key points of which to be aware in future testing:

- Confusion over the lack of letters in Level 1
- Not enough attention being paid to error messages in previous levels to prepare for the debug hash present in the error message of Level 5.

However, overall she was generally positive and provided some ideas for extending the project in future years.

Finally, I conducted five Think Alouds with participants from my target audience. I paid special attention to the aspects pointed out by the expert user and those related to our original design requirements. The results are discussed in the next chapter.

Chapter 7

Results and Discussion

7.1 Progression of Difficulty

As we can see from Table 6.1, all the participants took similar amounts of time to complete all six levels. Table 7.1 provides a break-down of the time taken to complete each level for each student. Here too, the students have similar times. Figure 7.1 shows the average times taken to complete each level. We notice that Levels 1 and 2 are completed quickly in all cases, while Level 3 takes substantially longer. This is partly due to the fact that it features a brute-force attack that requires the participant to simply wait until their script finds the correct password. Level 4 took the longest for each student, with Levels 5 and 6 taking slightly less time (as they are offline attacks, the computation time is shorter).

From these times, it would be sensible to conclude that the progression of difficulty from one level to the next is reasonable. Importantly, the comments voiced by the participants during the Think Alouds confirm this, as no one level was deemed *too* difficult. Level 4 certainly took the most time overall, but this was mainly due to programming issues and the computation time (see Section 7.2). The participants orally confirmed that Level 6 was the most challenging, as it introduced the concept of shadow files. However they did not find it excessively complicated, but instead were interested to learn of a new concept.

Participant	Level 1	Level 2	Level 3	Level 4	Level 5	Level 6
P1	2	3	13	14	6	11
P2	2	3	10	15	11	9
P3	2	3	10	14	8	9
P4	3	3	13	14	12	10
P5	2	4	15	23	14	11

Table 7.1: Table showing the time taken in minutes for each Think Aloud participant to complete each level

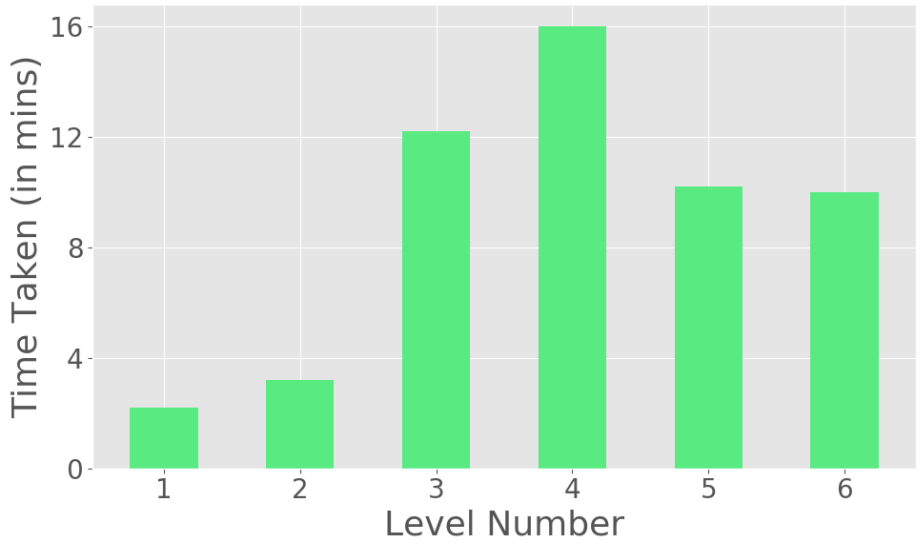

Figure 7.1: Plot of average time taken in minutes for each level of the system

7.2 Scripting the Attacks

None of the participants had any issues installing the suggested libraries for Python. **P4** did however point out that she believed that this would not be the case for all potential users, and felt certain that other students whom she knows would have issues. **P5** also suggested that a Python interpreter be integrated to the system, as it would make the coding easier in his opinion. This aligns with what our expert user highlighted, and while this change cannot be made immediately, it will certainly be considered for future work.

It should be noted that all the participants showed excitement and expressed a sense of accomplishment whenever their script found the correct password. This was most notably seen during Level 5, as, if the script is written correctly, the password should be output almost immediately since it is an offline attack, as opposed to the online attacks in the previous levels. This is the kind of emotional investment which was hoped for when creating the system.

7.2.1 Critical User Issues

The participants all were able to use Python, and all successfully completed all the levels. However, all but **P1** had some issues with opening files in order to get the possible passwords from the wordlist which is provided in Level 4. A code snippet demonstrating how to open files is included in Hint 2 of that level, but the users would of course want to attempt this for themselves without having to look at the hints.

Similarly, in Level 5, **P3**, **P4** and **P5** had a minor issue when using the Python `hashlib` library. All fully understood what they were attempting to do, the error arose only from a misunderstanding of the documentation. The exact behaviour of the `update` method is not very clear. The participants took it to mean that it would change the word to be hash, when in fact it concatenates it with the previous input. As such the users did not hash each possible password individually, but rather added the passwords one by one and hashed them all together.

Neither issue was caused by a lack of understanding of the problem. Nevertheless, a considerable amount of time was spent during these levels learning how to complete specific actions in Python. This is a problem as it suggests that the focus of these levels has strayed from understanding security concepts to learning Python. Since the users understand and know what they are trying to achieve, they do not use the hints - they justifiably do not consider that they need help.

Therefore, the solution I suggest for this would be to provide the users with a skeleton code file in each level containing the snippets found in the hints. Alternatively, the hints containing the sample code could be renamed to "Code Snippets" to clearly convey that they contain no additional information about the level, solely support for writing the script.

7.2.2 Minor User Issues

A mistake which all but **P2** made was forgetting that "000" to "099" are valid three-digit passwords. Arguably, this could indicate that the participants had not fully grasped the idea behind the level, but based on their thought process, it would seem that this was simply an oversight in all cases. They clearly understood that the input form had to take three digits, but forgot that looping over the numbers from 0 to 999 would not in fact cover "000" to "099". With this in mind, the best course of action is likely to just add a note pointing this out in the end of level explanation, to ensure that they notice their coding error.

Another issue discovered in Level 4 is that only **P1** remembered to change the URL to which the GET request was being submitted. The participants adapted their code from the previous level, in which the correct submission URL includes "`level3/check`". In Level 4 this should be changed to "`level4/check`". This contributed to the amount of time taken to complete Level 4 (see Fig. 7.1). This is an important element to notice, as it demonstrates that the user has understood how to case a website. However, ultimately, I would consider this a fairly minor issue as it does not cause the user to stray from the goal of the level, nor is it caused by a lack of programming knowledge. It is up to the user to be observant, as **P1** was. Nevertheless, since multiple participants made this mistake, I will add a hint to this level covering this point.

7.3 Hints

The main issue with the hints is that, while all students used the hints, not all were happy about it. **P2** and **P4** consulted the hints quite liberally as they considered the system to be a learning environment first and foremost. Conversely, **P1** and **P5** clearly felt a sense of defeat when they had to resort to using the hints, but they still preferred using the hints over becoming frustrated or giving up altogether. **P3**'s approach was somewhere between these two extremes, checking the hints more quickly than **P1** or **P5**, but still having a good independent attempt at solving the level first. He is also the only participant to check the hints in Level 1, not because he was stuck but because he wanted to verify what he had discovered so far. One could argue that this is not an issue, as it is up to the user how they want to approach the system. The only real problem arises from the fact that the hints are present in order to reduce the amount of frustration or boredom a student might feel from being stuck. If the term "Hints" discourage certain users from consulting them, this may backfire and the user may experience the negative affective states, in addition to a sense of defeat, I was hoping to avoid. As an initial response to this, it might be worth renaming the hints to "Tips", though the users' reactions to this new name would need to be tested.

On a more positive note, all the participants commented on how clear and well-written the hints were. In Level 4, both **P3** and **P4** opened the second hint hoping to find some information on how to open files in Python, which is in fact the contents of that hint. **P2**, **P3** and **P4** commented on how helpful it was to find the exact information they were unsure about. This suggests that the content of the hints was well-chosen and followed a logical progression.

7.4 Introductions and Explanations

All the participants read the introductions, though **P1** clearly skimmed through them, and all emphasised the elements in bold. This was most noticeable in Level 1 where the words "very short, simple password" are in bold. From this section, the participants started hypothesising about what Alice's password would be (a single character, entirely numerical, the word "password", etc.). This illustrates that highlighting important sections in bold was well received, as this was invariably the part that to which the users paid attention and said out loud.

Conversely, the links to explanatory modals were less well-received. For example, **P1** did not notice a lot of them, while **P3** was wary of clicking on them. He pointed out that it was clearly an explanation, but he was not sure if the link would navigate him away from the current page (a concern that **P1** also voiced). Ideally the links should be modified to indicate that they open a pop-up window and do not redirect the user.

On the other hand, **P2**, **P4** and **P5** clicked on all of the explanations and expressed that they understood the content being explained in each one. The only exception was the explanation of salting in Level 6. **P3** and **P4** both explicitly stated that that explanation was distinctly less clear than the others. Clearly this explanation will have to be reviewed.

The end-of-level explanations were similarly received. The participants expressed their understanding, generally accompanied by a variety of interjections as they understood the implications of what they have just learned. **P4** described the explanations as "well-written and easy to follow". They all seemed to understand and accept the sample solutions.

7.5 Overall Design and Usability

The overall average of the SUS scores given by the participants is 87.5. This matches the average found in Section 6.1. This, alongside the comments made by our participants, suggests that the system is easily usable. In fact, the only source of confusion that was directly linked to the system was in Level 1. The input form is coded to allow only a single digit to be typed in. As a result all the participants were confused, though they quickly figured out why this was happening. Other than this, the system design was well received, with **P3** and **P4** commenting that they liked the design and found it well-suited to the tasks.

With respect to the approach to error messages, all the participants successfully completed Level 5 without too much difficulty. However they did check all the first hint, which suggests that they check the error message. None of the participants seemed particularly upset by the fact they had to check the hint, though it is certainly worth bearing in mind as this is not an ideal outcome. More stress will be placed on checking for information in error messages during Level 2, in which the users case the website.

7.6 Summary

In this chapter, we discussed the results obtained from the five Think Alouds which I conducted. Table 7.2 summarises the main points.

I provided a possible solution for each issue. The main issue which arose was the fact that in Levels 4 and 5 the focus strays from learning about security to learning Python. However, overall it seems that the users enjoyed using the system and found it engaging and easy to use.

Positive Elements	Elements to be improved on
• Hints gave useful and desired information • Explanations are clear and succinct • Attractive design that seems both usable and suited to the task • Sense of achievement and excitement when script yielded the correct password • Bold sections were well-chosen and useful	• Using hints can inspire a sense of defeat • "Salting Hashes" explanation is confusing • Lack of letters in Level 1 is confusing • Opening files in Python in Level 4 • Using the `hashlib` library in Level 5 • Forgetting "000" to "099" in Level 3 • Links could be more obvious or indicate that they open a modal • Forgetting to change the URL in Level 4

Table 7.2: Summary of positive elements and elements to be improved on that arose as a result of my Think Alouds

Chapter 8

Conclusion

8.1 Overview

As we have seen, computer security is an increasingly important field, and yet despite this, there is a lack of support for future developers to learn about the basics of this topic. To fill this gap, my project was to create a system which teaches password security to future developers, to answer the question:

> "How might we design an entry-level tool to teach developers about password security?"

I set the following goals as the learning outcomes of my system:

- learning what is considered a good password, as opposed to a weak password
- understanding how cryptographic hash functions and salting work, and why they are important for secure password storage
- understanding and being able to perform a brute-force attack, a dictionary attack and a rainbow table attack, including the differences in time complexity of each of these attacks.

From this, I chose to design a system which teaches how common developer mistakes cause vulnerabilities which can be exploited by attackers. By demonstrating these exploits, I hoped to clearly illustrate the importance of the security concepts I defined as learning outcomes, and how these common developer mistakes can be avoided. As a result of my iterative design process, I defined the following requirements:

1. The system will be accessible to beginners.
2. The system will present a set of puzzles/challenges.
3. The system will consist of progressively harder levels.
4. The system will provide the user with hints.
5. The system will provide a sample solution for each level.

The system was evaluated with regards to its usability, and how well it met the system requirements. The goal was to assess how suitable and easy to use this type of system is for teaching the desired concepts. I conducted a survey of 22 participants who all filled out the SUS survey, and five Think Alouds.

In conclusion, it would seem that my system is usable and not overly complicated for the task. All the participants reported that they understood the material being taught. From this, I conclude that my system fulfils its goals, at least on the surface, though there are still some points which can be improved as we saw in Chapter 7. Nevertheless, to properly evaluate the system I will need to carry out a larger-scale evaluation.

8.2 Further Work

As we have said, the main point of improvement for my project is to conduct a larger-scale evaluation. Now that I know that the system itself is sound, I need to better evaluate the content and learning gains of the users. This will likely be done through the use of pre- and post-tests. Naturally, before this, I will need to implement the solutions to identified problems, such as the salting issue outlined in Section 5.1.2, and the points raised in Chapter 7.

The project has a lot of room for expansion and improvement. In a first instance, there are the suggestions of our expert user to consider (see Section 6.2). She suggested that a Python interpreter be included in the interface. This would be helpful as it would take the focus away from learning Python, as the required libraries and sample code will be provided within the system - the user will not have to install any additional material on their own computer. Furthermore, she pointed out that the project might benefit from a second set of levels, which would be marked. This would allow the students to learn the material, before applying their knowledge in an environment with less support and guidance.

I have only been focused on password security throughout this project. However, there are a variety of vulnerabilities which can arise as a result of developer mistakes, such as SQL Injections or Cross-Site Scripting attacks. The project could easily be extended in order to provide developers with a better knowledge of computer security.

Bibliography

[1] 32 million passwords show most users careless about security — Ars Technica. [Online]. https://arstechnica.com/security/2010/01/32-million-passwords-show-most-users-careless-about-security/ . [Accessed: 01-April-2017].

[2] Bootstrap. [Online]. http://getbootstrap.com/. [Accessed: 03-April-2017].

[3] Bootstrap Responsive Sidebar - CodePly. [Online]. https://www.codeply.com/go/ecE6qHNBOC. [Accessed: 27-March-2017].

[4] CloudPassage Study Finds U.S Universities Failing in Cybersecurity Education - CloudPassage. [Online]. https://www.cloudpassage.com/company/press-releases/cloudpassage-study-finds-u-s-universities-failing-cybersecurity-education/. [Accessed: 05-April-2017].

[5] Draft NIST SP 800-63-3 Digital Identity Guidelines. [Online]. https://pages.nist.gov/800-63-3/. [Accessed: 01-April-2017].

[6] HackThis!! - The Hackers Playground. [Online]. https://www.hackthis.co.uk/. [Accessed: 01-April-2017].

[7] How LinkedIns password sloppiness hurts us all — Ars Technica. [Online]. https://arstechnica.com/security/2016/06/how-linkedins-password-sloppiness-hurts-us-all/ . [Accessed: 01-April-2017].

[8] Improper Error Handling - OWASP. [Online]. https://www.owasp.org/index.php/Improper_Error_Handling/. [Accessed: 03-April-2017].

[9] Number of Internet Users (2016) - Internet Live Stats. [Online]. http://www.internetlivestats.com/internet-users/. [Accessed: 27-March-2017].

[10] Passwords - SkullSecurity. [Online]. https://wiki.skullsecurity.org/index.php?title=Passwords. [Accessed: 27-March-2017].

[11] Play on Demand & CyPhinx - Cyber Security Challenge UK . [Online]. https://cybersecuritychallenge.org.uk/competitions/play-demand-cyphinx . [Accessed: 01-April-2017].

[12] Secure Salted Password Hashing - How to do it Properly — CrackStation. [Online]. https://crackstation.net/hashing-security.htm. [Accessed: 01-April-2017].

[13] Web Application Exploits and Defenses - Gruyere. [Online]. https://google-gruyere.appspot.com/. [Accessed: 01-April-2017].

[14] Anne Adams and Martina Angela Sasse. Users are not the enemy. *Commun. ACM*, 42(12):40–46, 1999.

[15] Myrto Arapnis. Cryptography - lecture notes in computer security, October 2016.

[16] Ruth Aylett, Sandy Louchart, João Dias, Ana Paiva, and Marco Vala. Fearnot! - an experiment in emergent narrative. In *Intelligent Virtual Agents, 5th International Working Conference, IVA 2005, Kos, Greece, September 12-14, 2005, Proceedings*, pages 305–316, 2005.

[17] Aaron Bangor, Philip Kortum, and James Miller. Determining what individual sus scores mean: Adding an adjective rating scale. *Journal of usability studies*, 4(3):114–123, 2009.

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

[19] Peter Chapman, Jonathan Burket, and David Brumley. Picoctf: A game-based computer security competition for high school students. In *2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14)*, San Diego, CA, 2014. USENIX Association.

[20] Rachna Dhamija, J. D. Tygar, and Marti Hearst. Why phishing works. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '06, pages 581–590, New York, NY, USA, 2006. ACM.

[21] KA Ericsson and H Simon. Protocol analysis. verbal reports as data. 1993.

[22] Nickolas Falkner, Raja Sooriamurthi, and Zbigniew Michalewicz. Teaching puzzle-based learning: development of basic concepts. *Teaching Mathematics and Computer Science*, 10(1):183–204, 2012.

[23] Nickolas J. G. Falkner, Raja Sooriamurthi, and Zbigniew Michalewicz. Puzzle-based learning for engineering and computer science. *IEEE Computer*, 43(4):20–28, 2010.

[24] Tanya Flushman, Mark Gondree, and Zachary N. J. Peterson. This is not a game: Early observations on using alternate reality games for teaching security concepts to first-year undergraduates. In *8th Workshop on Cyber Security Experimentation and Test (CSET 15)*, Washington, D.C., 2015. USENIX Association.

[25] Center for Strategic and International Studies. Hacking the Skills Shortage. [Online]. http://www.mcafee.com/us/resources/reports/rp-hacking-skills-shortage.pdf. [Accessed: 05-April-2017].

[26] Alexander Ivan Games and Luke Kane. Exploring adolescent's STEM learning through scaffolded game design. In *Foundations of Digital Games, FDG'11, Bordeaux, France, June 28 - July 1, 2011*, pages 1–8, 2011.

[27] Cormac Herley. So long, and no thanks for the externalities: The rational rejection of security advice by users. In *Proceedings of the 2009 Workshop on New Security Paradigms Workshop*, NSPW '09, pages 133–144, New York, NY, USA, 2009. ACM.

[28] W. Lewis Johnson. Serious use of a serious game for language learning. In *Artificial Intelligence in Education, Building Technology Rich Learning Contexts That Work, Proceedings of the 13th International Conference on Artificial Intelligence in Education, AIED 2007, July 9-13, 2007, Los Angeles, California, USA*, pages 67–74, 2007.

[29] Patrick Gage Kelley, Saranga Komanduri, Michelle L. Mazurek, Richard Shay, Timothy Vidas, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, and Julio Lopez. Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms. In *IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA*, pages 523–537, 2012.

[30] Rebecca Klahr, Sophie Amili, Jayesh Shah, Mark Button, and Victoria Wang. Cyber Security Breaches Survey 2016. [Online]. https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/521465/Cyber_Security_Breaches_Survey_2016_main_report_FINAL.pdf. [Accessed: 05-April-2017].

[31] B. Martin, B. Hanington, and B.M. Hanington. *Universal Methods of Design: 100 Ways to Research Complex Problems, Develop Innovative Ideas, and Design Effective Solutions*. Rockport Publishers, 2012.

[32] Scott W. McQuiggan, Jonathan P. Rowe, Sunyoung Lee, and James C. Lester. Story-based learning: The impact of narrative on learning experiences and outcomes. In *Intelligent Tutoring Systems, 9th International Conference, ITS 2008, Montreal, Canada, June 23-27, 2008, Proceedings*, pages 530–539, 2008.

[33] Zbigniew Michalewicz and M Michalewicz. *Puzzle-based learning*. Hybrid Publishers, 2008.

[34] Robert Morris and Ken Thompson. Password security - A case history. *Commun. ACM*, 22(11):594–597, 1979.

[35] Klaus Purer. Php vs. python vs. ruby–the web scripting language shootout. *Vienna University of Technology*, 2009.

[36] Jonathan P. Rowe, Scott W. McQuiggan, Jennifer L. Robison, and James C. Lester. Off-task behavior in narrative-centered learning environments. In *Artificial Intelligence in Education: Building Learning Systems that Care: From Knowledge Representation to Affective Modelling, Proceedings of the 14th International Conference on Artificial Intelligence in Education, AIED 2009, July 6-10, 2009, Brighton, UK*, pages 99–106, 2009.

[37] John M Zelle. *Python programming: an introduction to computer science*. Franklin, Beedle & Associates, Inc., 2004.

Appendix A

System Screenshots

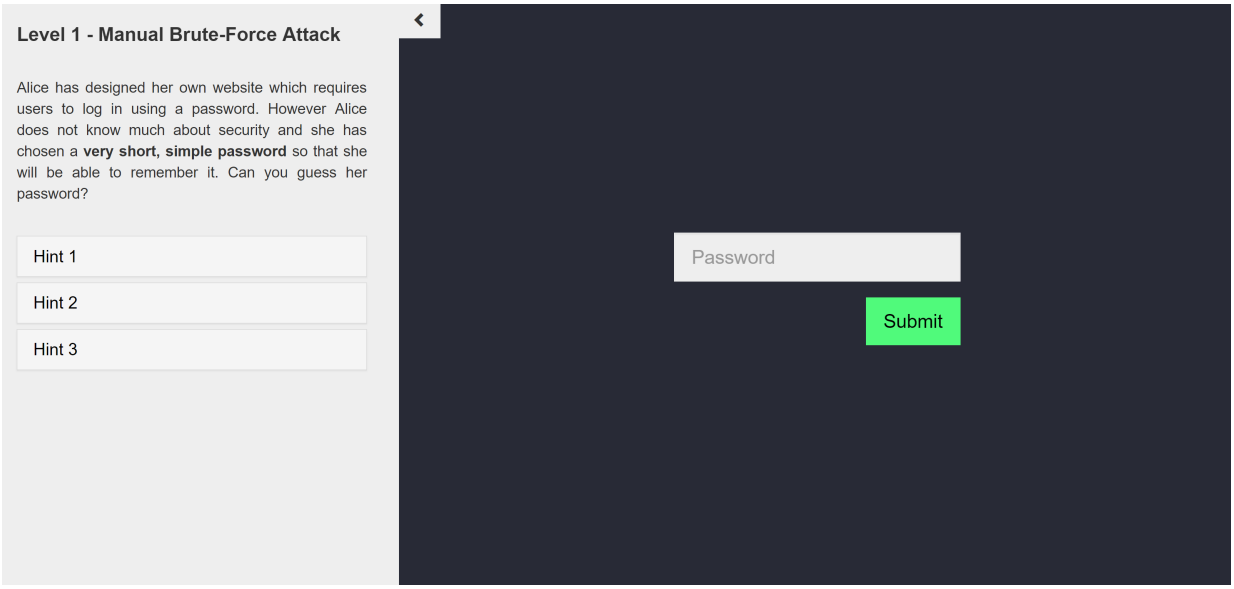

Figure A.1: Screenshot of Level 1

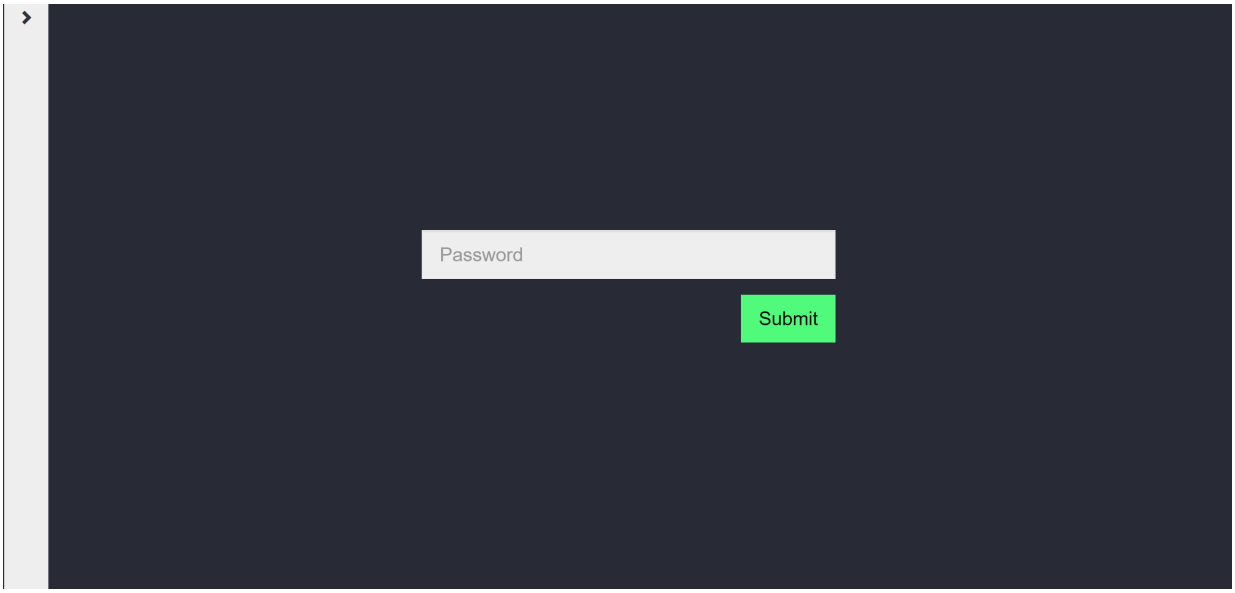

Figure A.2: Screenshot of Level 1, with the sidebar collapsed

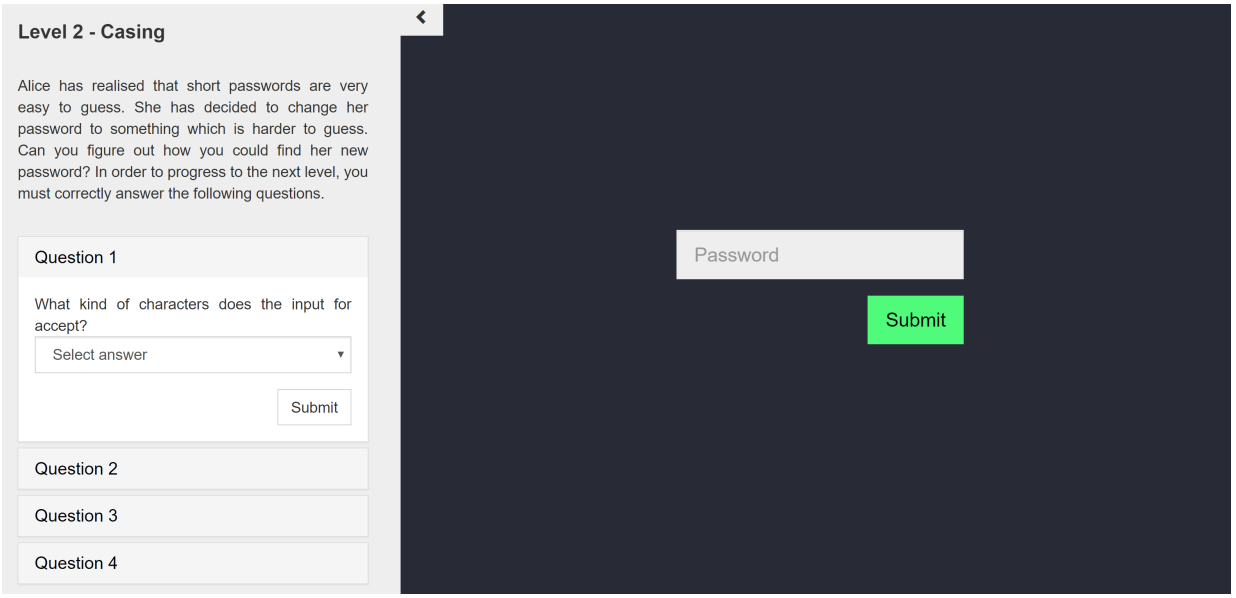

Figure A.3: Screenshot of Level 2

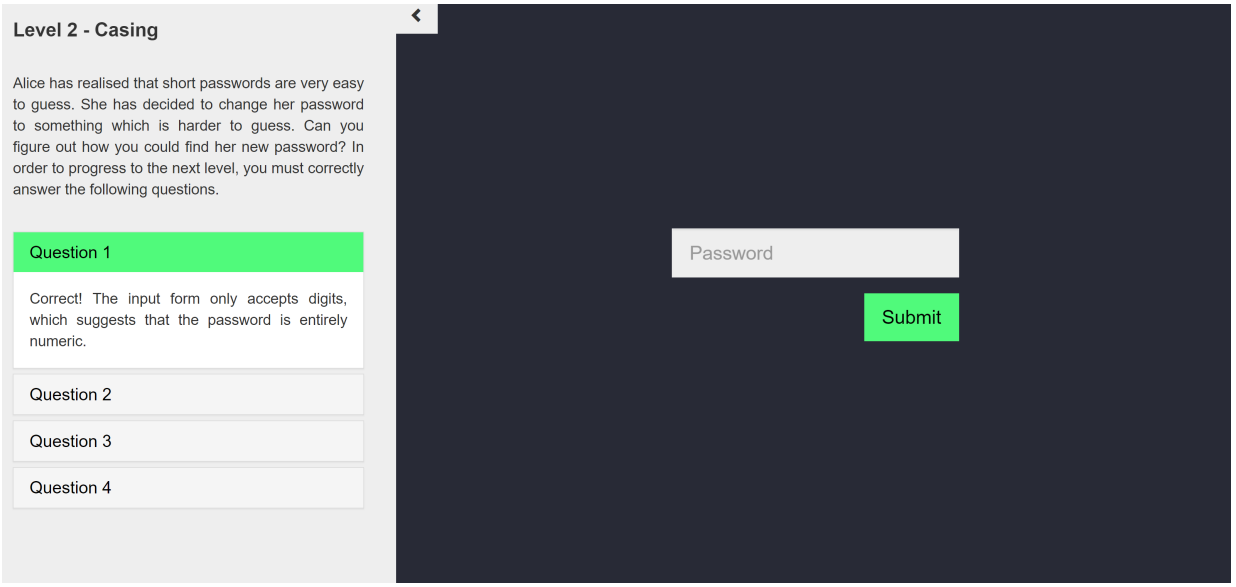

Figure A.4: Screenshot of Level 2, with first question answered correctly.

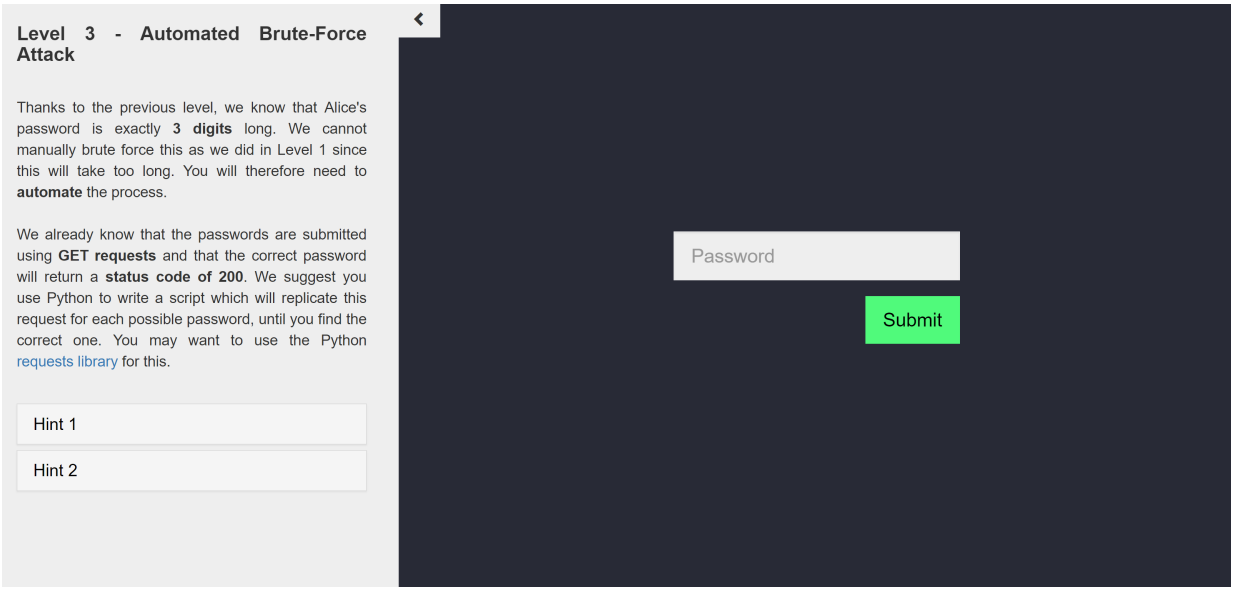

Figure A.5: Screenshot of Level 3

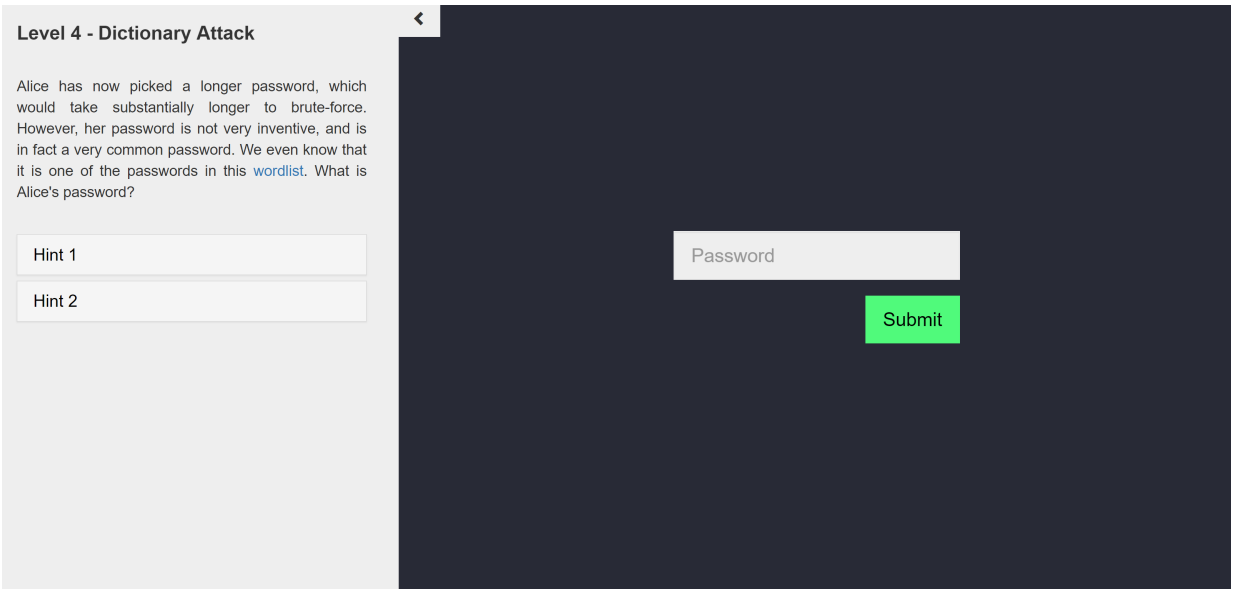

Figure A.6: Screenshot of Level 4

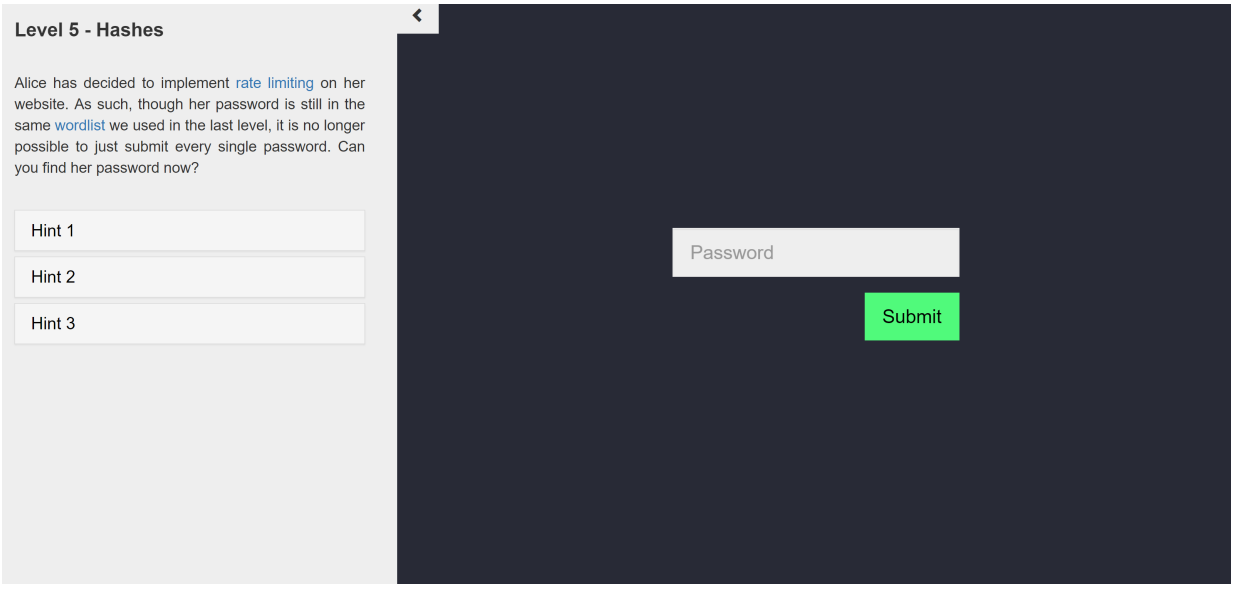

Figure A.7: Screenshot of Level 5

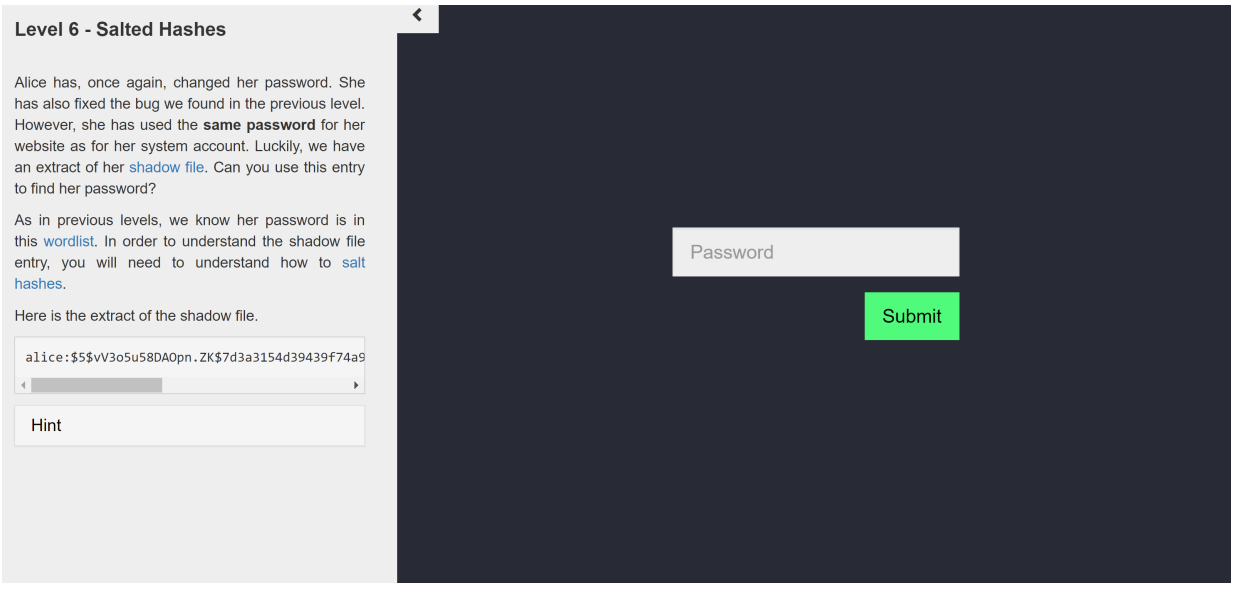

Figure A.8: Screenshot of Level 6

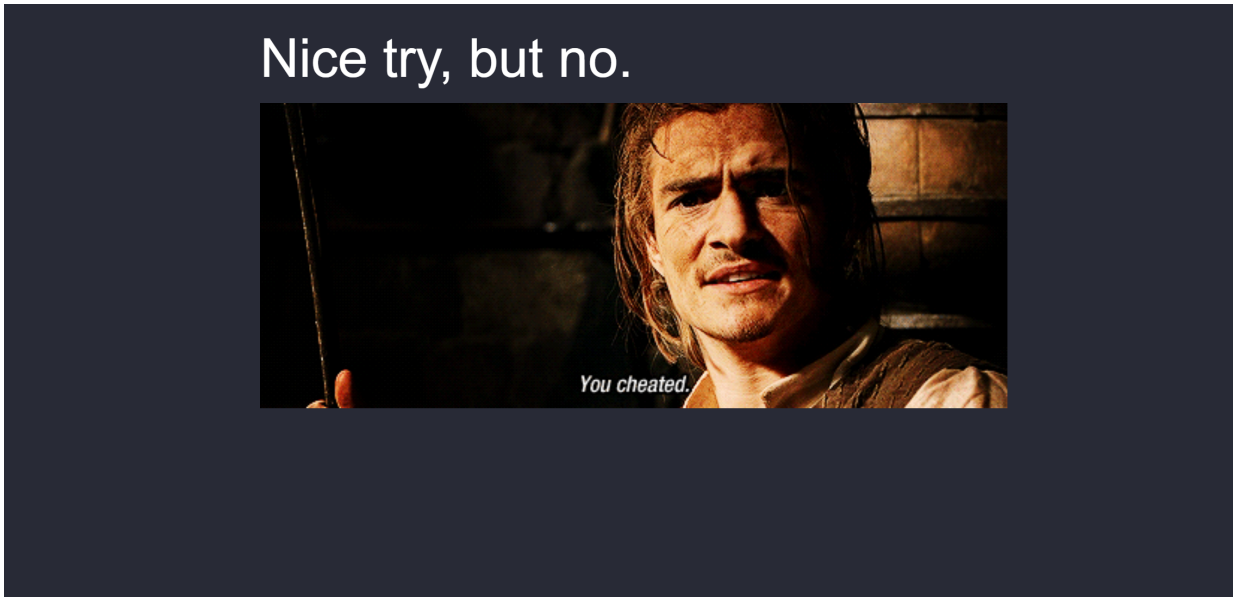

Figure A.9: Screenshot of page which appears if a user attempts to skip to the next level by changing the URL, rather than completing the level. Image from http://cdn3.gurl.com/wp-content/uploads/2015/09/cheated.gif.

Appendix B

System Usability Scale

Feedback Survey

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

What is your gender?

☐ Male
☐ Female
☐ Prefer not to answer
☐ Other .....

What year are you?

☐ UG1
☐ UG2
☐ UG3
☐ UG4 / MInf
☐ MSc
☐ Other .....

Appendix C

Think Aloud Script

Hello, my name is Connie. I'm a 4th year UG student, and I am conducting this study as part of the evaluation of my honours project. My honours project is about creating a system for teaching beginners, specifically future developers with some programming experience, about password security by teaching how various attacks can be carried out on an insecure website.

The purpose of this exercise is to evaluate the usability of my system. Please remember that I am testing the teaching system, and that I am not testing you. I will be taking notes during this, and I will use this data in my dissertation. Your name will not be included in the report at any point. You have the right to stop at any time. This project has undergone ethical screening in accordance with the University of Edinburgh School of Informatics ethics process (RT1748). Do you have any questions? Do you agree with all this?

[CONTINUE IF PARTICIPANT AGREES]

In this observation, I am interested in what you think about as you perform the tasks I am 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 each level until you finish it. 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. Do you understand what I want you to do?

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 my mother's house?

[DEMONSTRATE]

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

[WAIT UNTIL THEY HAVE FINISHED]

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 verbalize these too. As you are doing the tasks, I wont 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 system 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? There is a total of 6 levels in the system, and I would like to complete them all. Do you have any questions about the task? You may begin.