Password Security Gamification

Theo Koorehpaz

University of Sheffield

Supervisor: Ramsay Taylor

Module code: COM3610

This report is submitted in partial fulfilment of the requirement for the degree of Computer Science

13th May 2020

Declaration

All sentences or passages quoted in this report from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations that are not the work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this project and the degree examination as a whole.

COVID-19 Impact Statement

The lockdown imposed because of COVID-19 caused additional challenges for the completion of this project. In the second semester of the project, the university switched to online delivery of all teaching, and university buildings were closed. All project meetings were shifted to email correspondence and video meetings. In addition, my project plan was revised because I could no longer conduct the in-person evaluations that were originally planned.

Acknowledgements

First and foremost, I would like to thank my supervisor Ramsay Taylor for allowing me to conduct this project.

I would also like to thank friends, family and especially *the boys* for providing me with emotional support throughout.

I would also like to thank Marshall Bruce Mathers III for providing me with the lyrical motivation needed to complete this research.

Abstract

As we plunge deeper into the information era, society has become increasingly reliant on the internet for many services. The first line of defence to safeguard information resources from unauthorised access is password authentication. Despite it's prevalence, best practice is not often followed and the use of weak passwords persist.

Gamification is a recent technique that has emerged in the last decade and has proven to be particularly successful in improving user experience through simple psychology. It is a powerful tool that roots the idea of using game elements in non-game contexts to reach objectives while increasing user engagement and motivation.

Every day we are becoming more informed with the consequences of bad passwords, however those that are younger are yet to realise the insecurities. The aim of this project is to combine the benefits of Gamification with password security in order to reinforce the value and practice of strong passwords. It has been tailored especially towards the younger generation who are most in jeopardy of being attacked and are influenced by the burden of learning.

Contents

Declaration i

COVID-19 Impact Statement ii

Acknowledgements iii

Abstract iv

1 Introduction 1

2 Literature Review 3

2.1 Cracking Techniques . . . 4

2.1.1 Brute-Force . . . 4

2.1.2 Dictionary attacks . . . 6

2.1.3 Keyword attack . . . 7

2.2 Password Security . . . 8

2.2.1 Hashing . . . 8

2.2.2 Salts . . . 8

2.2.3 Peppers . . . 9

2.2.4 Security . . . 10

2.2.5 Hash Collisions . . . 11

2.3 Speed & Complexity . . . 12

2.3.1 Processes, Threads & Cores . . . 12

2.3.2 Keys per Second . . . 13

2.3.3 Password Complexity . . . 17

2.4 Existing Game Analysis . . . 19

2.4.1 S0urce.io . . . 19

2.4.2 Hacker Test . . . 20

2.4.3 Terminal Hacker . . . 21

2.4.4 Evaluation 22
2.5 Gamification 23
2.5.1 Gamification of Password Security 25

3 Requirements and Analysis **26**
3.1 Aim & Objectives 27
3.1.1 Aim 27
3.1.2 Objectives 27
3.2 Requirements 28
3.3 Analysis & Evaluation 29

4 Design **30**
4.1 Core GUI Components 31
4.1.1 Pre-Menu 31
4.1.2 Menu 32
4.1.3 Terminal 32
4.1.4 Store 33
4.1.5 Password Strength 33
4.1.6 Achievements 34
4.2 Game Mechanics & Modifications 35
4.2.1 Level Structure 35
4.2.2 Terminal Interface 37
4.2.3 Commands & Syntax 39
4.2.4 Entertainment Value & Reward System 39

5 Implementation & Testing **41**
5.1 Cracking Algorithms 42
5.1.1 Numerical Brute-Force 42
5.1.2 Alphanumerical Brute-Force 43
5.1.3 Dictionary attack 44
5.1.4 Combinator Dictionary Attack 45
5.1.5 Hybrid Dictionary Attack 47
5.1.6 Extension Parameter 48
5.1.7 Validation 51
5.1.8 Cracking Times & Multi-threading 52
5.1.9 Expert Level Generation 54
5.1.10 Libraries 56

5.2 Testing . . . 58
5.2.1 Unit Testing . . . 58
5.2.2 Beta-Testing & Amendments . . . 60

6 Results & Discussion **62**
6.1 Evaluation of Requirements . . . 63
6.1.1 Gamification Mechanics . . . 63
6.1.2 Educational Value . . . 64
6.1.3 Game Application . . . 66
6.2 Evaluation of Objectives . . . 67
6.3 Evaluation of Aim . . . 68
6.4 Future Work . . . 72

7 Conclusions **73**

A Expanded Information **75**
A.1 Disclaimer . . . 75
A.2 About . . . 75
A.3 How to Play . . . 76
A.4 Commands . . . 76
A.5 Algorithms . . . 76
A.6 Dictionaries . . . 77
A.7 Achievements . . . 78
A.8 Menu Items . . . 78

B Pre-game Questionnaire **79**
B.1 Round 1 Password Security . . . 79

C Post-game Questionnaire **85**
C.1 Password Security Round 2 . . . 85
C.2 User Experience . . . 90

D Questionnaire Responses **94**
D.1 Password Security Round 1 . . . 94
D.2 Password Security Round 2 . . . 98
D.3 User Experience . . . 103

Chapter 1

Introduction

Passwords are the forefront of security and confidentiality of data stored across almost infinite servers and workstations around the world. Obtaining them is an effective attack approach as they are the most commonly used mechanism to authenticate users in an information system – so, it would come as no surprise that a strong password is of utmost importance to prevent any unauthorised access to a computer. But what exactly constitutes to a 'strong' password? Many are aware of the importance of a strong password but do not fully understand why. Aside those who undertake information technology related studies, many are left exposed to the vulnerabilities of poor passwords and will continue to overlook the possibility of being attacked. A common trait amongst many individuals is that learning tends to be tedious, particularly for children that have very imaginative mindsets. Motivation will falter if a task is boring, so in the interest of achieving quality results it is necessary to find a way to increase engagement through enjoyable mediums. Gamification can prove to be ideal in this context by taking game design elements and applying them to password security as a means to educate players on an appealing platform.

In this paper, the development of a password security game is presented. The aim of the game is to harness the principles of gamification to increase password security and awareness for children through an engaging application. It will simulate accurate scenarios and methods that an attacker would perform to obtain passwords, whilst providing an immersive experience for the player that puts them in the driving seat of a hacker.

In the proceeding chapter, principles of password security, methodologies of gamification and analysis of existing games are researched and presented in the form of a literature review. Later chapters will explore requirements and analysis where the

project is broken down into aims that are achievable and objectives that can be met. Subsequent chapters will describe the technicalities of the application in the form of design and implementation where each feature will be documented. Finally, the last chapters will evaluate and assess the project as a whole and provide a summary discussing what went well and potential improvements that could have been made in hindsight.

Chapter 2

Literature Review

This chapter will explore popular password cracking tools and measure their effectiveness against standard security measures, as well as providing an in-depth review behind the psychology of gamification that motivates players. Existing hacker themed video games will also be analysed to capture the features that provide an engaging experience that can be moulded into the context of gamified password security.

2.1 Cracking Techniques

2.1.1 Brute-Force

The term 'brute-force' can be labelled on any algorithm that involves repeatedly trying different solutions until a condition is satisfied - all password cracking tools include this characteristic. A pure brute-force algorithm, or an exhaustive search, is a programming style that relies on sheer computing power to blindly iterate over an entire domain of possibilities (the domain being the length and character set of the password in this case). We only need to consider iterating over numerical, uppercase, lowercase and special character keys to crack a password - which is values 32 to 126 in the ASCII character set:

Dec	Bin	Hex	Char	Dec	Bin	Hex	Char	Dec	Bin	Hex	Char	Dec	Bin	Hex	Char
0	0000 0000	00	[NUL]	32	0010 0000	20	space	64	0100 0000	40	@	96	0110 0000	60	`
1	0000 0001	01	[SOH]	33	0010 0001	21	!	65	0100 0001	41	A	97	0110 0001	61	a
2	0000 0010	02	[STX]	34	0010 0010	22	"	66	0100 0010	42	B	98	0110 0010	62	b
3	0000 0011	03	[ETX]	35	0010 0011	23	#	67	0100 0011	43	C	99	0110 0011	63	c
4	0000 0100	04	[EOT]	36	0010 0100	24	$	68	0100 0100	44	D	100	0110 0100	64	d
5	0000 0101	05	[ENQ]	37	0010 0101	25	%	69	0100 0101	45	E	101	0110 0101	65	e
6	0000 0110	06	[ACK]	38	0010 0110	26	&	70	0100 0110	46	F	102	0110 0110	66	f
7	0000 0111	07	[BEL]	39	0010 0111	27	'	71	0100 0111	47	G	103	0110 0111	67	g
8	0000 1000	08	[BS]	40	0010 1000	28	(	72	0100 1000	48	H	104	0110 1000	68	h
9	0000 1001	09	[TAB]	41	0010 1001	29	)	73	0100 1001	49	I	105	0110 1001	69	i
10	0000 1010	0A	[LF]	42	0010 1010	2A	*	74	0100 1010	4A	J	106	0110 1010	6A	j
11	0000 1011	0B	[VT]	43	0010 1011	2B	+	75	0100 1011	4B	K	107	0110 1011	6B	k
12	0000 1100	0C	[FF]	44	0010 1100	2C	,	76	0100 1100	4C	L	108	0110 1100	6C	l
13	0000 1101	0D	[CR]	45	0010 1101	2D	-	77	0100 1101	4D	M	109	0110 1101	6D	m
14	0000 1110	0E	[SO]	46	0010 1110	2E	.	78	0100 1110	4E	N	110	0110 1110	6E	n
15	0000 1111	0F	[SI]	47	0010 1111	2F	/	79	0100 1111	4F	O	111	0110 1111	6F	o
16	0001 0000	10	[DLE]	48	0011 0000	30	0	80	0101 0000	50	P	112	0111 0000	70	p
17	0001 0001	11	[DC1]	49	0011 0001	31	1	81	0101 0001	51	Q	113	0111 0001	71	q
18	0001 0010	12	[DC2]	50	0011 0010	32	2	82	0101 0010	52	R	114	0111 0010	72	r
19	0001 0011	13	[DC3]	51	0011 0011	33	3	83	0101 0011	53	S	115	0111 0011	73	s
20	0001 0100	14	[DC4]	52	0011 0100	34	4	84	0101 0100	54	T	116	0111 0100	74	t
21	0001 0101	15	[NAK]	53	0011 0101	35	5	85	0101 0101	55	U	117	0111 0101	75	u
22	0001 0110	16	[SYN]	54	0011 0110	36	6	86	0101 0110	56	V	118	0111 0110	76	v
23	0001 0111	17	[ETB]	55	0011 0111	37	7	87	0101 0111	57	W	119	0111 0111	77	w
24	0001 1000	18	[CAN]	56	0011 1000	38	8	88	0101 1000	58	X	120	0111 1000	78	x
25	0001 1001	19	[EM]	57	0011 1001	39	9	89	0101 1001	59	Y	121	0111 1001	79	y
26	0001 1010	1A	[SUB]	58	0011 1010	3A	:	90	0101 1010	5A	Z	122	0111 1010	7A	z
27	0001 1011	1B	[ESC]	59	0011 1011	3B	;	91	0101 1011	5B	[	123	0111 1011	7B	{
28	0001 1100	1C	[FS]	60	0011 1100	3C	<	92	0101 1100	5C	\	124	0111 1100	7C	|
29	0001 1101	1D	[GS]	61	0011 1101	3D	=	93	0101 1101	5D	]	125	0111 1101	7D	}
30	0001 1110	1E	[RS]	62	0011 1110	3E	>	94	0101 1110	5E	^	126	0111 1110	7E	~
31	0001 1111	1F	[US]	63	0011 1111	3F	?	95	0101 1111	5F	_	127	0111 1111	7F	[DEL]

We can, of course, target specific character types for different scenarios. Disregarding certain character types will speed up the process dramatically, as restricting the search space will include much fewer possibilities for the brute-force to attempt. For instance, it would be unnecessary to brute-force a purely alphabetical password with the entire character set that includes numbers and special characters. Ironically, this technique can prove to be quite effective towards 'strong' passwords that must meet certain requirements.

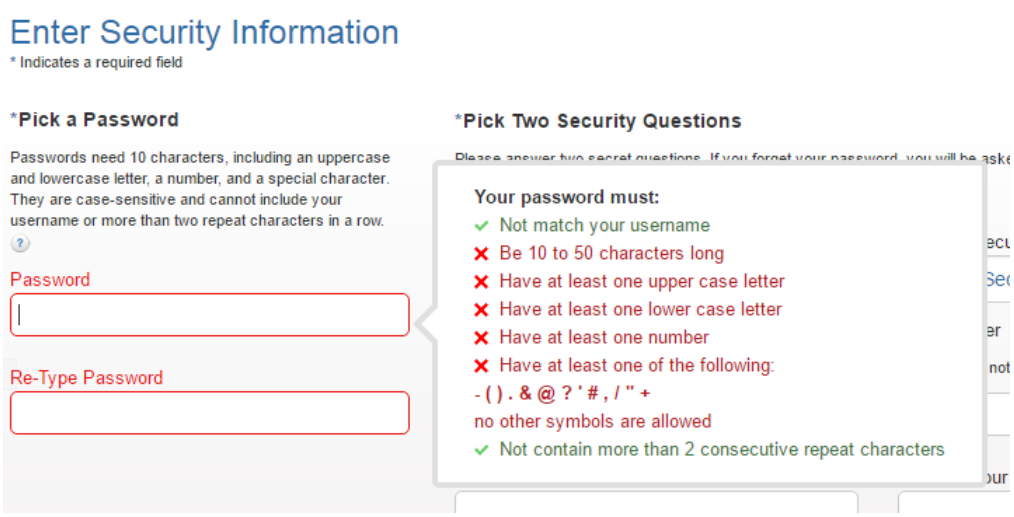


Figure 1: USPS Sign-up [1]

Ultimately, this does ensure a strong password by definition. But in practice hackers can take advantage of the password requirements to narrow down the search tree of possibilities. Another example where a brute-force approach may seem feasible is log-ins that require specific password characters or pin positions, provided that there is no maximum attempt mechanism in place:

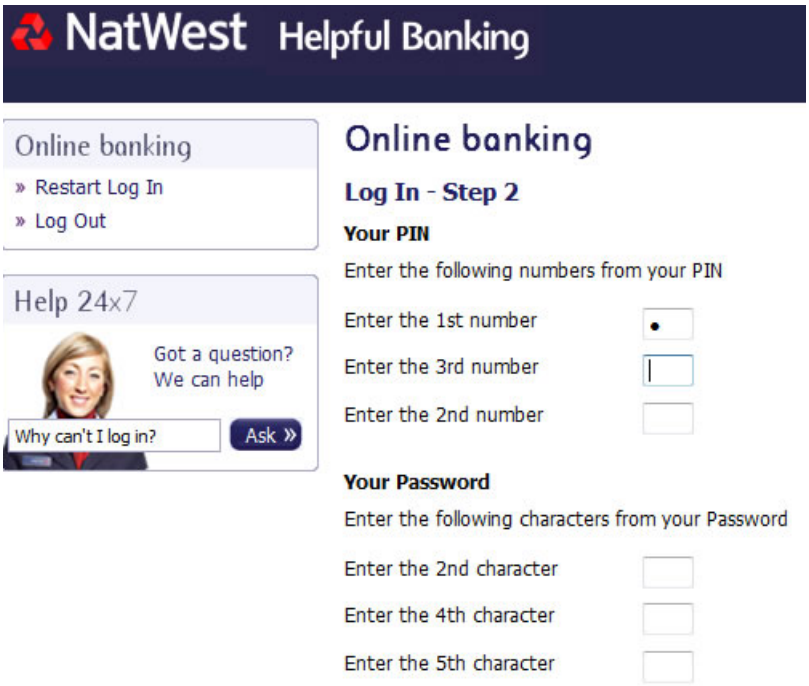


Figure 2: NatWest Online Banking login [2]

For purely numerical passwords, a simple for loop can be used to sequentially test numbers against a given input. For example, cracking a 4-digit numerical lock-screen passcode on a phone could require trying every number up to and including 9999. We must consider the fact that although numerical, these types of passwords are essentially PINs which are strings consisting of numbers. Therefore, in order to iterate through the possibilities, we need to append zeros for any unused digits:

```
1-9;          | 10-99;        | 100-999;      | 1000-9999;
0001, … 0009 → 0010, … 0099 → 0100, … 0999 → 1000, … 9999 …
```

2.1.2 Dictionary attacks

Dictionary attacks are a form of brute-force attack but iterate over commonly used words or phrases in lists to guess a password, rather than having to iterate every possible combination of it. They are considered to be a much more effective cracking tool due to the simple fact that most passwords consist of words rather than arbitrary characters. Pure brute-forcing the word 'password' for example, would produce an enormous search domain of 26^8 possibilities that would not be very feasible to search. On the other hand, a dictionary attack would more than likely be able to guess it due to its predetermined list of words that may contain the password. In the list of the 25 most popular passwords of 2018, 11 were made up of just common words and 4 were common words with numerical suffix/prefixes. [3] Dictionary attacks often succeed because people have a tendency to choose short passwords that are ordinary words or common passwords, even simple variations like appending a digit or substituting similar-looking characters are likely to be included.

Many types of dictionary attacks exist; combinator and hybrid adaptations are amongst the most popular. They exhibit different characteristics that can be more effective for cracking particular passwords - for instance, a combinator attack will append each word to every word in its own dictionary, or append each word from its own dictionary to another dictionary: [4]

```
pass   passpass testpass 1234pass
test → passtest testtest 1234test
1234   pass1234 test1234 12341234
```

Hybrid attacks are also another technique that utilise words from their own dictionary. They are similar to combinator attacks except a full brute-force keyspace is

either appended or prepended to each of the words from the dictionary: [5]

```
yell   yell0000, yell0001, yell0002, … yell9999,
been → been0000, been0001, been0002, … been9999,
seve   seve0000, seve0001, seve0002, … seve9999.
```

Both of these techniques tackle common password traits. Joining two words together or having a word followed by a sequence of numbers (e.g. a date) are examples of habitual pitfalls that attackers can take advantage of. However, a dictionary attack is only as effective as the dictionary it is using. There are extensive available lists which range from thousands to millions of terms that can be used for targeting specific subsets; such as social media lists, leaked passwords from sites and even actual dictionary of words. [6]

2.1.3 Keyword attack

The third and final cracking technique that will be explored in this research inherits a brute-force approach and is similar to that of a Mask attack that attempts all combinations from a predefined keyspace. We tend to know about humans and how they design passwords; a common pattern would be to include an individual's name along with their birth year for instance. The proposed technique will entail taking data inputted by the user which is then manipulated in a series of different ways to determine the password - the ways in which the data is manipulated can be categorised into three parts: substitution, concatenation and repetition. Substitution involves swapping uppercase with lowercase characters (and vice versa) and replacing letters with similar-looking numbers or symbols. Concatenation will join inputted strings together and append numerical or special characters to the end of the string, and repetition will truncate and join parts of the input together. It can be described similar to a hybrid of brute-force and dictionary attack that attempts multiple variations of specific words. The idea behind this technique is to generate password-like results given an input of terms, which could prove to be a valuable cracking technique further into this research.

```
User input: liverpool, suarez
Example of substitution property:  l1verp001, su4r3z, l1VeRpO0L …
Example of concatenation property: liverpool2?, liverpoolsuarez …
Example of repetition property:    livliv, rezrez, poolpool …

Example output: liverpool22, l1verp00l, L1VerPOOl, l1vl1v22,
liverp00lsuarez, suarez82, suasua?, suarezliverpool, l1v3rp0017 …
```

2.2 Password Security

2.2.1 Hashing

Hashing is a widely used technique in the field of computing and possesses various integral applications, from speeding up database lookups to verifying the integrity or corruption of data. In particular, it can be used for concealing sensitive data such as passwords. Passwords are almost never stored as plain-text on a computer system or database. Instead, they are usually passed through a hashing algorithm, which is a one-way pseudo-random function that transforms a string of data into a seemingly random output string of fixed length. The same input string will always produce the same output string, but changing the input string by a single character will produce a completely different output string. Not only is this technique impossible to reverse engineer but the password itself never has to be viewed in the process, making it somewhat secure. An example of a widely used and recognised hashing function is MD5, which takes an input message of arbitrary length and produces a 126-bit hexadecimal output string of length 32. [7]

```
Input string: "Hello World!"
MD5 Hash: ed076287532e86365e841e92bfc50d8c
```

Changing the W into lowercase 'w'

```
Input string: "Hello world!"
MD5 Hash: 86fb269d190d2c85f6e0468ceca42a20 -> very different
output
```

[8]

The problem here is if hashed passwords ever found their way onto the internet via data dumps or leaks, it would not take much effort for an attacker to crack them. All they would need to do is pass a list of words or phrases through the same hashing function until a match is found. This process can be made even easier by using a hash or rainbow table; which are precomputed dictionaries of plaintext passwords and their corresponding hash values [9]. One does not even have to compute the hashes in the first place, they can just be looked up in an extensive table and correlated to the plaintext passwords.

2.2.2 Salts

A salt is a non-secret value which is appended to the password before it gets hashed, thus producing a different hash from that of the password itself. It is important

to note that salts should be unique and never use any user-supplied information, to prevent two databases from returning the same hash for a given password + salt [10]. Although this technique renders hash tables useless, brute-force and dictionary attacks will still be an issue, assuming an attacker takes into account the salt and knows where to put it in their guesses.

```
Stored Hash = Hash(Password + Salt) ~
Password: p@ssw0rd
Salt: #&hL
∴ Stored Hash: Hash(p@ssw0rd#&hL) =
d04d53d29ee89f00978eea849b445477
```

Non-secret property of salts - usually stored as plain-text next to the hash:

```
[Database] -> Username, StoredHash, Salt ~
> cheekyqwerty, d04d53d29ee89f00978eea849b445477, #&hL
```

2.2.3 Peppers

A pepper is a secret, very short random string added to the password before it is hashed. The key difference between a salt and a pepper is that peppers are not stored, so in order to determine the hashed password + pepper, every combination of the pepper will have to be computed until a match is found. For instance, if a lowercase letter were to be used as a pepper, then there would be 26 possible hashes that would need to be cycled through; as a result, it would take an attacker 26 times longer to crack the password by brute-forcing it.

```
Stored Hash: Hash(Password + Pepper) ~
Password: p@ssw0rd
Pepper: x
Stored Hash: Hash(p@ssw0rdx) = f811ebdc440a4ef991ee8e3160009b43
```

Retrieving the password:

```
> Hash(p@ssw0rda) = d9d54905ffa112e98588a876dedce3bf
> Hash(p@ssw0rdb) = a6580233013d7a24d63ae123a383ab41
> Hash(p@ssw0rdc) = c46dafd8a9f6530db64ef2bbc1305bbc
> …
> Hash(p@ssw0rdx) = f811ebdc440a4ef991ee8e3160009b43 -> found
```

Secret property of peppers - not stored:

```
[User Login] ~
Username: cheekyqwerty
Password: p@ssw0rd -> [begin computing peppers]
```

2.2.4 Security

There are three main requirements that hashing algorithms must display in order to be deemed secure; speed, randomness and collision avoidance [11]. The MD5 algorithm is not deemed to be secure as it does not meet two of the requirements; speed and collision resistance. MD5 displays properties of randomness (as shown on page 8) - changing just one bit in the input string produces a completely different result. The security property that arises from the speed of a hashing algorithm is that it should neither be too slow nor too fast, in order to decrease the feasibility of distributed brute-force attacks on hashes. MD5 is considered to be very fast as it uses 32-bit operations, which are fast on x86 CPUs and current GPUs. On the other hand, other password hashing algorithms such as SHA-512 use 64-bit arithmetic operations which are slightly more resource-intensive for GPUs and 32-bit architectures. When tested with data, it was found that the SHA-512 performed 29% slower than MD5 for shorter input strings (36-49 characters), and 15.5% slower for longer strings (72-85 characters). [12]

2.2.5 Hash Collisions

Collision resistance is a property of cryptographic hash functions that determines how difficult it is to find two inputs that hash to the same output. More formally, given two inputs a and b such that:

$$H(a) = H(b), a \neq b$$

Collisions are a natural and unavoidable feature that arise from mapping a large value space to a much smaller value space, which is merely an instance of the pigeonhole principle. MD5 is not considered to be collision resistant due to its 'small' 128-bit digest size, where it is easier for a collision to occur compared to other functions such as SHA-512 which have a 512-bit digest. This is not necessarily a problem in the context of passwords as the probability of a collision is negligible and they do not require to be cryptographically secure. For applications such as document verification however, it should be impossible to create two messages that produce the same hash value and to generate a message matching a specific hash value [13]. This has huge implications for security as it is possible for someone to artificially create a hash collision. For instance, if someone could make a file that sums to a certain hash, they could forge documents that contain valid signatures that would verify the document. Or, they could intercept a file and change its data whilst keeping the hash the same by carefully tweaking the bytes.

2.3 Speed & Complexity

2.3.1 Processes, Threads & Cores

Computers have become very efficient at scheduling to simulate multiple processes and threads to run in parallel. There exists multiple technologies that adopt this idea to give the illusion that more hardware is being used than there actually is on a physical level, which can dramatically increase overall performance. A process is a program being executed which can be further divided into independent units known as threads [14].

Multiprocessing refers to the use of multiple CPUs within a single computer system which enables more instructions to be read and executed concurrently. Multi-core processors actually have additional hardware that is used to distribute and schedule more resources to processes:

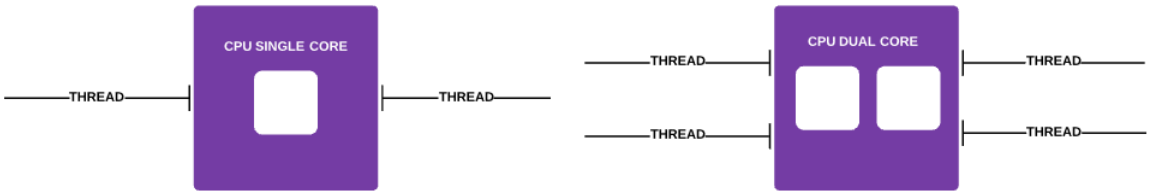


Figure 3: Multiprocessing

Multithreading is the ability of a CPU to provide multiple threads of execution so that more tasks can be executed simultaneously. It allows a single process to have multiple threads running in parallel whilst sharing resources:

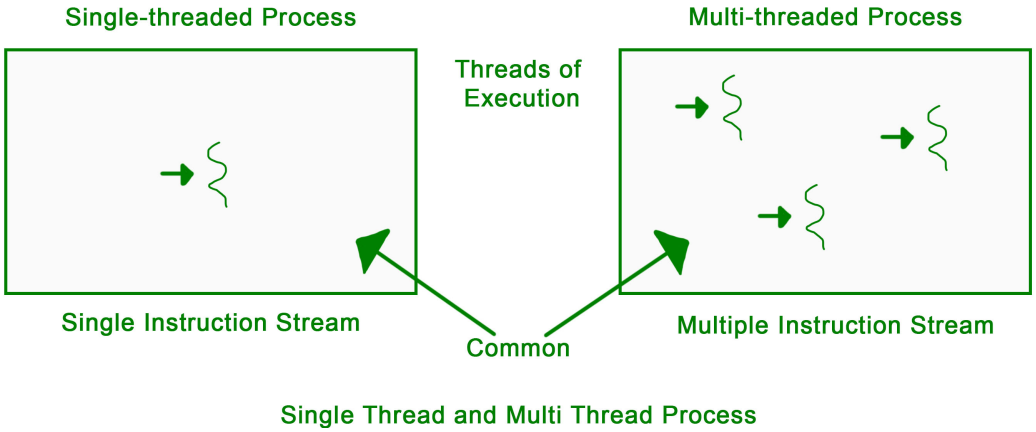


Figure 4: Multithreading [14]

Hyperthreading is a complex architecture developed by Intel that involves splitting each of the physical cores into virtual cores which simulates some degree of overlap in executing multiple instructions. The operating system will recognise each physical core as 2 logical cores and share workload between them, which greatly increases CPU performance and efficiency by allowing a greater number of independent instructions in the pipeline. Hyperthreading is an application of simultaneous multithreading and is only available on specific Intel processors and operating systems that support the technology. [15]

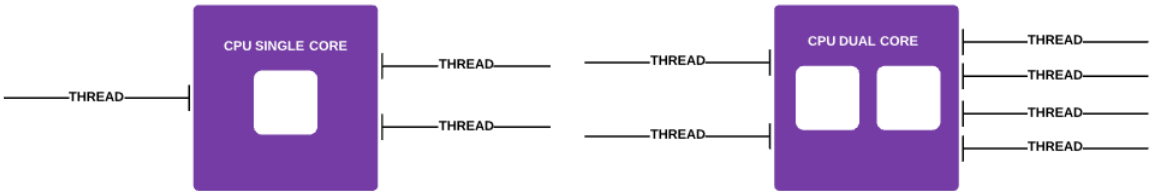


Figure 5: Hyperthreading

2.3.2 Keys per Second

Processing power is ultimately the driving force behind the speed of any computation, along with various other hardware and software components. In the context of passwords, the rate at which characters and sequence of characters can be attempted on a system is typically measured in keys-per-second. In order to calculate this constant and the time it would take to crack a given password (by brute-forcing it), a few factors need to be considered. [16]

1. Number of possible character combinations

The number of possible character combinations is calculated by raising the password type by the length of the password:

$$number\ of\ characters\ in\ the\ set^{\ password\ length}$$

Password length = 6:
Numeric:
[0123456789] ~ 10^6 = 1,000,000

Alphabetical:
[abcdefghijklmnopqrstuvwxyz] ~ 26^6 = 308,915,776

Alpha-numeric:
[ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789] ~ 36^6 = 2,176,782,336

Mixalpha:
[abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ] ~ 52^6 = 19,770,609,664

Mixalphanumeric:
[abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789] ~ 62^6 = 56,800,235,584

ASCII-32-126:
[All characters on standard US keyboard] ~ 94^6 = 689,869,781,056

2. Number of cores

Multi-core CPUs allow computers to carry out multiple operations at the same time, which in turn speed up computation. They are often considered to be the powerhouse of a computer, however GPUs can also excel when it comes to numerical operations. Graphics rendering is simply a series of complex mathematical calculations, which is essentially what hashing algorithms require to generate hashes. Although they operate at lower clock-speeds than CPUs, they make up for it in the number of cores that can run in parallel, making them exceptional hardware components for cracking passwords [17]. The processing cores and performance of each architecture can be seen below:

Figure 6: CPU vs GPU cores [18]

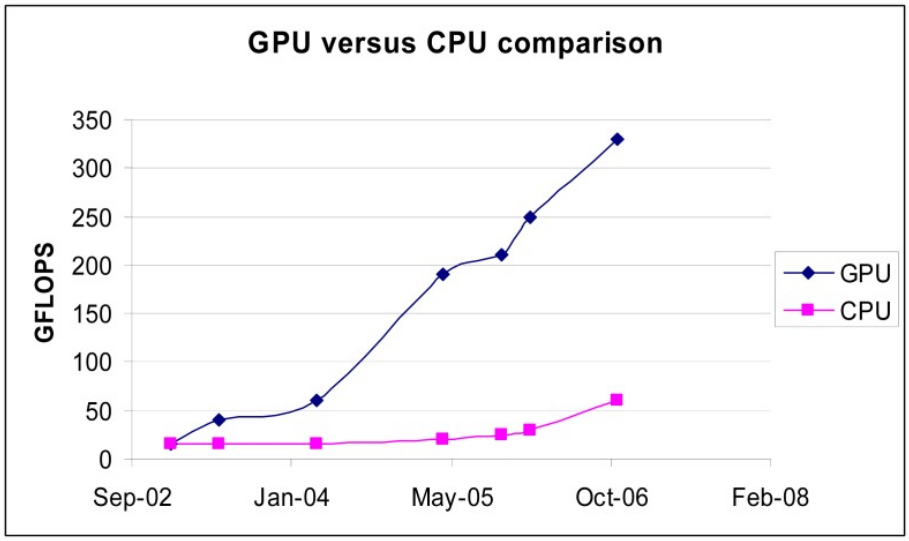


Figure 7: FLOP performance [19]

So, taking into account both CPU and GPU cores, we can calculate the number of effective cores to be used for hashing passwords - adding an efficiency constant to assume that a certain percentage of operations can be dedicated to the password crack:

$$Effective\ cores = Efficiency\ Constant \times Cores$$

3. Processor FLOPS and IPS

CPUs adopt single instruction, single data architectures (SISD), whereas GPUs have single instruction, multiple data architectures (SIMD). As previously discussed, to compensate for the limited number of cores a CPU can accommodate, techniques such as multithreading or hyperthreading can be used to allow more operations to be run simultaneously - this will not be considered in the calculation. The speed of a processor is usually measured in floating-point operations per second, or FLOPS. However, if we are using hashing functions like MD5 to perform computations on passwords that are broken down into 32-bit integers, we need to calculate instructions per second that don't require special floating-point operations: [20]

$$IPS = Instructions\ per\ clock\ cycle \times Clock\ rate \times Effective\ Cores$$

4. Keys-per-Second and Overall time

There are too many parameters to consider to calculate an exact KPS value for a

particular set of computer components, such as moving data around, communicating results between processors and setting up the next set of calculations for the arithmetic processors for applying hashes. We can summarise these internal factors by adding a constant: [21]

$$KPS = IPS \ / \ Hash\ Constant$$

Once we have calculated the KPS a processor can execute, we can calculate the overall time to guess the password given a particular character set:

$$Overall\ time = Combinations \ / \ KPS$$

Example theoretical cracking times for a desktop PC with an i7-980EE CPU and NVIDIA GTX295 GPU: [20]

	CPU	GPU
Cores	4	480
Effective Cores	3.4	456
GIPS	148	566
Hash/sec (MD5)	223,000	906,000

> KPS total: 1,129,000

`[123456] ->` $\frac{10^6}{1{,}129{,}000} \approx$ `0.89s`

`[password] ->` $\frac{26^8}{1{,}129{,}000} \approx$ `2 days`

`[pAsSw0rD] ->` $\frac{62^8}{1{,}129{,}000} \approx$ `5 years`

`[js92@!88?Km] ->` $\frac{94^{11}}{1{,}129{,}000} \approx$ `108378 millennia`

We can see that GPUs can be tailored to be very proficient in generating hashed passwords, however applying this technique into an application is far beyond the scope of the project and will only be discussed to illustrate the potential it has for password cracking. Moore's law predicts that the number of transistors in a dense integrated circuit doubles around every 18 months; which will result in faster computation and therefore faster cracking speeds [22]. This will become an issue for passwords that are not considered 'complex' enough for such computing power in years to come, especially with quantum computing over the horizon that is estimated to outperform Turing Machines by several orders of magnitude.

2.3.3 Password Complexity

As demonstrated above, the complexity of a password can be dictated by how long it would computationally take to guess it. This all depends on the search domain of possibilities and the predictability of the password, against brute-force and dictionary attacks respectively. We can elaborate even further by observing a dataset of 185,643 passwords obtained from a university's online grading system, that were used to demonstrate how unique cracked and uncracked passwords compared by length and character sets: [23]

Length	Passwords			
	Cracked		Uncracked	
	Count	*%*	*Count*	*%*
All lengths	150,213	100.00	923	100.00
1-4 characters	1471	0.98	0	0.00
5 characters	1316	0.88	0	0.00
6 characters	124999	83.21	0	0.00
7 characters	3047	2.03	2	0.22
8 characters	14964	9.96	3	0.33
9 characters	2044	1.36	13	1.41
10 characters	1357	0.90	31	3.36
11 characters	482	0.32	242	26.22
12 characters	276	0.18	207	22.43
13 characters	154	0.10	145	15.71
14 characters	60	0.04	118	12.78
15 characters	26	0.02	97	10.51
16 characters	14	0.01	22	2.38
17+ characters	3	0.002	43	4.66

Figure 8: Cracking times based on length

Character sets	Passwords			
	Cracked		Uncracked	
	Count	*%*	*Count*	*%*
loweralphanum	124,775	83.07	587	63.60
mixedalphanum	10,211	6.80	1	0.11
loweralpha	7912	5.27	196	21.24
numeric	6468	4.31	1	0.11
upperalphanum	272	0.18	0	0.00
loweralphaspecialnum	92	0.06	54	5.85
loweralphaspecial	76	0.05	18	1.95
specialnum	12	0.01	0	0.00
special	4	0.003	0	0.00
upperalpha	1	0.0007	0	0.00
mixedalphaspecialnum	0	0.00	7	0.76

Figure 9: Cracking times based on character sets

We can summarise what constitutes to a strong password with the following characteristics:

- Long passwords (8+ characters)
- Mixed character types (alphabetical/numerical/special)
- Non-dictionary terms
- Not a common password or password variation (e.g. p@55w0rd)
- Passwords that do not follow patterns (e.g. abcdefg)
- Arbitrary passwords (randomly generated)

In essence, the longer and more arbitrary a password is, the longer it will take for a computer to guess it. The security of a password is boosted exponentially each time a character is added as a result of the combinatorial explosion, so a password that is long enough would render a brute-force attack useless. Dictionary attacks and their multiple variations are often defeated by avoiding common words or phrases and variations of words in general.

2.4 Existing Game Analysis

Analysis and evaluation of existing games that revolve around hacking or password cracking, is a fundamental process for capturing valuable ideas and methodologies that can be implemented in the design of a password security game. Below are examples of three current games that demonstrate features to consider.

2.4.1 S0urce.io

S0urce.io is a game that revolves around hacking other real players in an online environment. You start off by choosing a player from a target list and selecting one of three 'ports' to attack. You are then displayed with a command-line terminal that indicates what to type in order to commence 'hacking' the player's firewall. After each successful hack, you are rewarded with bitcoin that can be used to buy data miners or defences for your own network.

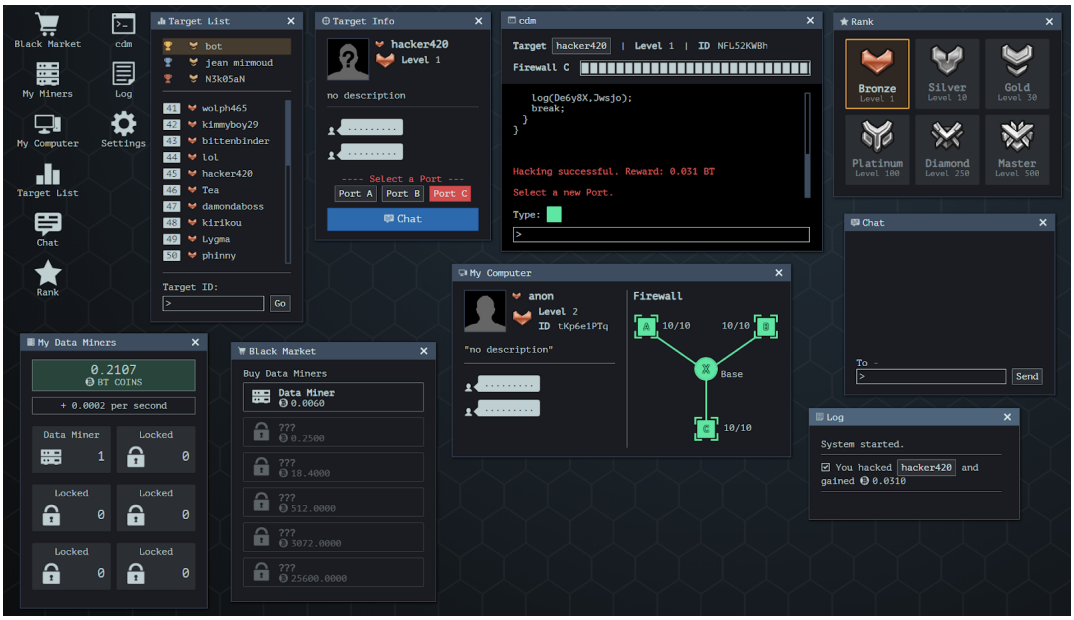


Figure 10: Screenshot of S0urce.io gameplay

Although some aspects of S0urce display elements of novelty, the overall experience feels somewhat tedious. It is a very oversimplified simulation of hacking that does not stimulate any sort of reward to the user. The journey to the goal of topping the leaderboard is a very linear challenge that essentially boils down to whoever can repeatedly type the same precomputed instructions the longest. Moreover, the bitcoin reward system does not serve any grand purpose in climbing the leaderboard and is a mere effort to add another layer to the game that attempts to encompass the feeling of

hacking. Despite the game's lack of engagement with the user, the online aspect as a whole performs well with responsive interactions with other users and live 'incoming' hacks being displayed in the logs. It also excels graphically with a sharp and intuitive interface that provides a satisfying experience when navigating between the windows.

2.4.2 Hacker Test

Hacker test is an online hacker simulation with 20 levels. The aim of the game is to crack the password on each page using the clues hidden in the source code. Each level gets progressively more difficult as the clues become more enveloped within the backend of the webpage that the player must try to uncover in order to advance.

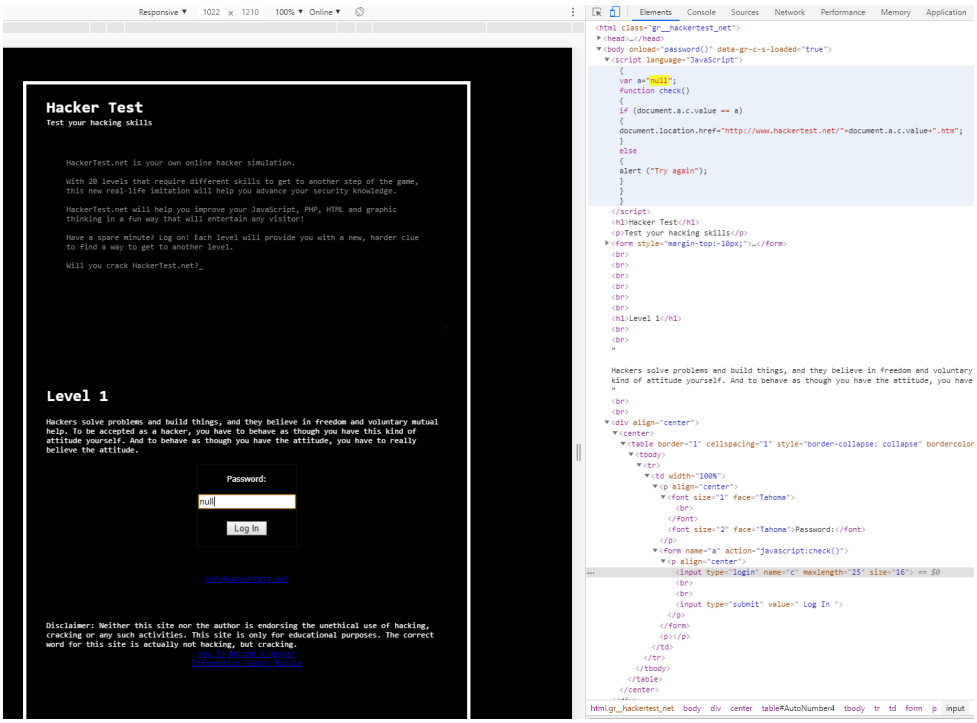


Figure 11: Screenshot of Hacker Test gameplay

Hacker Test is a thorough hacking simulator that requires meticulous attention to detail and an adequate understanding of web-based programming languages to play. Instead of reproducing interfaces, hacker test intentionally leaves gaps within the webpage itself to simulate scenarios for the player to exploit. This is a clever technique as it involves directly interacting with the page code, which creates an immersive experience to make the user feel like they are actually hacking a website. The downside to this game is that it can be deemed too difficult for the uninitiated, as it does require a certain degree of technical ability and problem solving. That being said, the main objective of the game is to improve one's understanding of JavaScript, PHP, HTML

as well as graphical thinking in a fun and interactive way, so it can be seen as either a challenge for competent programmers or a learning tool for beginners. The graphical appearance, although bare-boned, works in this setting as the game focuses purely on its technical objective that does not require a lot of front-end interaction.

2.4.3 Terminal Hacker

Terminal Hacker is a password cracking game that requires the player to guess as many passwords as they can before getting locked out of the terminal. It features three different game modes with four difficulties, and leaderboards.

Figure 12: Screenshot of Terminal Hacker gameplay

Terminal Hacker can be described as a glorified version of hangman. The hacker aesthetic has been built on the foundation of a word guessing game that does not require any technical ability from the player. Be that as it may, the simplistic design has been executed competently that proves to be effective in a number of ways. Simple mechanics such as time and incorrect guesses that contribute towards the final score, incentivises players to set personal records. Although being a word guessing game at its core, it does include a few different game modes and difficulties that tailor towards the players desires and abilities respectively. The interface does emulate that of a terminal window to some extent, displaying elements such as monospaced font, black background and only keyboard accessibility.

2.4.4 Evaluation

The table below summarises the pros and cons each game exhibits in the context of password security. This evaluation will heavily influence what mechanics and components shall be considered when designing the game, especially during the requirements and analysis section in the proceeding chapter.

S0urce	+ Attractive interface + Intuitive + Responsive interactions	- Oversimplified - Not challenging - Unsatisfactory reward system
Hacker Typer	+ Challenging + Improves understanding of various languages + Immersive experience	- (Very) technical - Basic understanding of web-based languages is required - No reward system
Terminal Hacker	+ Simple, effective game mechanics + Well designed interface + Saves progress	- Not technical - Not accurate in the context of password cracking - No reward system

2.5 Gamification

Gamification is a psychologically driven approach towards targeting motivation by incorporating game-design elements into non-game contexts [24]. In other words, a combination of work and play that allows for playful interactions whilst producing quality results. Research has indicated that it has shown mostly positive effects [25] and can be applied in a variety of contexts, such as fitness programs, reward programs for loyal customers, driving performance and personal goals. In particular, it is especially prominent in the field of education, as it can be used to ameliorate the burden of learning which can have significant implications for enriching intellectual ability. A study found that well designed, properly deployed gamification can lead to positive effects on cognitive learning outcomes, and improve lower-risk assignment performance such as quizzes and practical activities. Students perceived gamified courses to be more motivating, interesting and conducive to learning that other courses [26]. So, what exactly constitutes to a 'well designed, properly deployed' gamification? According to Fogg's behaviour model, it depends on three key ingredients: motivation, ability and a trigger:

Figure 13: Fogg's behaviour model

Without a doubt, video games inherently possess a high level of motivational potential. Given that gamification refers to the use of 'game-design elements into non-game contexts', it is not unreasonable to assume that it can harness this motivational power that can be applied in real-world applications. It is important to note, however, that gamification is not identical to the structure or objective of typical 'games' and differs in a number of ways. For instance, gamification adopts the idea of the concept of a game that involves rule-based explorative activities, rather than the definition of a

game that can be described as an artificial conflict between players that results in a quantifiable outcome [27]. Moreover, the design process of gamification focuses on game design elements as opposed to fully developed games that serve specific entertainment purposes. [28]

Points, badges and leaderboards are examples of components that are used as extrinsic motivation techniques to provide users with rewards during their journey on obtaining a goal [29]. Points are generally used to reward the player for performing specific tasks or for indicating progress towards a certain goal; a simple, yet effective mechanic that accumulates over time that can be used to quantify experience for the player. Badges are rewarded in a similar way to points, but are set at definitive milestones for completing specific or a collection of challenges. They can be seen as personal achievements in the eyes of the player and developers can use them to encourage the objectives of the gamified system. Leader-boards are graphical representations of an individual's performance against others, they add an element of competitiveness that can incentivise motivation to out-perform other players. A potential danger of leader-boards is that they are capable of de-motivating too, for instance if a player falls too far behind and sees it as an insurmountable task being able to catch up. Lastly, feedback is another crucial design element of gamification that can stimulate intrinsic motivation. It can be used to indicate to a player how close they are to reaching a goal, or even to provide constructive criticism on their learning performance. Feedback that is ongoing, immediate and meaningful can have a positive effect on learning outcomes and encourage players to keep playing. [30]

So, we know that gamification alleviates boredom by incorporating reward-based progression. Ultimately, this is dictated by the nature and difficulty of the goals themselves, which must be challenging yet achievable within the individual's ability. A task which is difficult can yield a larger reward for the player (e.g. points), but can also induce frustration. On the other hand, a task that is too easy can result in loss of interest - as a result, the challenge needs to increase as a player's skills improve over time:

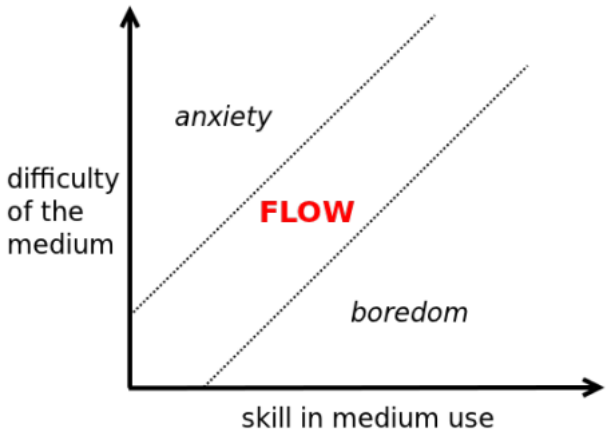


Figure 14: Anxiety boredom flow channel [31]

We can conclude that with the correct ratio of motivation and ability, gamification is able to positively influence human behaviour by driving players above the activation threshold and triggering them into specific actions. [32]

2.5.1 Gamification of Password Security

Despite its prevalence, password security is often overlooked and the use of weak passwords persist. The younger generation are especially vulnerable to the pitfalls of bad passwords as they are yet to fully realise the insecurities - creating a gamified application for this problem will hope to increase awareness. By taking into account the aforementioned sections regarding cracking techniques, storage measures and cracking times as elements in the system, players will be exposed to the processes involved in obtaining passwords through an interesting and engaging medium.

Summary

This chapter has explored how systematic password cracking approaches fair against various hashing countermeasures, along with a detailed analysis of relevant video games that demonstrate key components for developing an engaging gamified experience.

Chapter 3

Requirements and Analysis

The purpose of this chapter is to construct a set of requirements which will be used to plan the project. The aim and objectives are defined, as well as specific functionalities that will be provided in the application. The evaluation process will consider how the success of the project will be measured.

3.1 Aim & Objectives

3.1.1 Aim

- To increase password security and awareness in younger people by using an entertaining method and creating a password cracking game that adopts gamification techniques.

By providing an enjoyable and engaging platform, players will be motivated to continue playing and acquire a larger assimilation of the methods behind password security. To attain such a result, a precise balance of game features and educational elements must be achieved - applicability of gamification techniques in conjunction with password security will therefore need to be considered.

3.1.2 Objectives

- To understand password security and gamification by researching into current literature.
- To apply the principles of gamification to password security by developing a game that promotes the use of strong passwords.

Following the literature review, further knowledge of password security was gained which is necessary in order to create an accurate simulation of password cracking. Gamification also needs to be well understood and correctly utilised in order to provide the educational value through motivation - mechanics such as reward-based progression, achievements and challenging tasks set within the player's ability should be enough to trigger quality results.

The testing objectives will focus more on facilitating the development process for the developer with the use of code coverage and unit testing. Code coverage provides information on the lines of code have been hit given a particular execution trace, and unit tests are used to validate the function of individual components. Combining these two elements to clearly illustrate what areas of code have been tested will motivate the developer to write better tests which can result in more robust code. Beta-testing will also be considered as a way to enhance functionality during the final stages of development.

3.2 Requirements

The following table has been constructed to specify the full list of requirements for the project. Certain requirements are more important than others and should be prioritised given the timescale of the project; therefore each requirement will have an associated priority, M for mandatory, D for desirable and O for optional.

No.	Requirement	Priority
1	Improve understanding of password security	M
2	Increase entertainment value	M
3	Motivating reward system	D
4	Algorithm progress indicator	D
5	Linear difficulty increase	M
6	Universally playable	M
7	Intuitive and clean interface	D
8	Feasible cracking times	M
9	Utilise specifications of the system	O
10	Utilise hashing algorithm	D
11	Automatically generate random profiles	M
12	Automatically generate passwords from the profiles	M
13	Allows users to save progress	O
14	Allows users to unlock brute-force algorithms	M
15	Allows users to unlock dictionary attack algorithms	M
16	Allows users to unlock combinator dictionary attack algorithms	D
17	Allows users to unlock hybrid dictionary attack algorithms	D
18	Allows users to unlock keyword attack algorithms	D
19	Utilise Multithreading	D
20	Utilise Salt & Peppers in passwords	D
21	Utilise Hash tables	D
22	Allows users to unlock various dictionaries for dictionary attacks	M
23	Utilise Unit Tests	M
24	Utilise Code Coverage	M
25	Conduct Beta-tests	M

3.3 Analysis & Evaluation

Having studied and utilised Java for several years at the University of Sheffield, a strong familiarity and understanding of the language has been established. As a result, it will be the primary programming language in this project. The IntelliJ IDE in conjunction with code coverage and unit testing techniques will also be used which will prove to be helpful during development. Designing a hashing algorithm is beyond the scope for this project so code for a hashing algorithm will need to be imported. An analysis of the primary users involved must also be considered, as the game is especially catered towards year 13 pupils but can also be applicable to anyone who wishes to increase their understanding of password security.

Considering the nature and aim of the project, a target group will be asked to complete a short questionnaire that gauges their current understanding of password security. The same targeted group will then complete another questionnaire after playing the game that will be used to evaluate any increased understanding of password security - this will be the primary metric to determine whether the primary goal of the research has been fulfilled. Additional feedback will be collected from the participants asking about their general user experience, if they feel incentivised to use stronger passwords and any improvements that could be made.

Chapter 4

Design

This chapter will detail the final design of the application, covering the decisions made in the previous chapter as well as discussing modifications and improvements. The main focus will be giving an overview of the GUI components and mechanics of the game.

4.1 Core GUI Components

As per Requirement 7, the game must display a clean and intuitive interface in order for the user's journey to be smooth and as enjoyable as possible. It became very clear after 'Existing Game Analysis' that a well designed interface is of utmost importance when designing such an application, so a lot of resources were dedicated towards the front-end design. Combining the pros from the Existing Game Analysis Evaluation the general theme was decided to resemble a terminal emulator, similar to a Windows BIOS menu. The additional graphical layer and keyboard navigation will provide a much more attractive look and immersive feel to the game that maintains the aesthetic of a command line interface. It was also decided to not make the game full screen and keep the dimensions of the frame similar to that of a terminal window for a more compact view.

4.1.1 Pre-Menu

Before the player is taken to the Menu landing page, 3 message dialogs will be displayed as soon as the application has started. The player will be forced into reading a *Disclaimer*, *About* and *How to Play* dialog, that convey important information about the nature and objective of the application. Feedback received during the beta-testing phase of the game indicated that it was not immediately obvious how the game worked or where to start, so the *How to Play* dialog along with the other important messages are the first thing the player has to acknowledge [A.1-A.3].

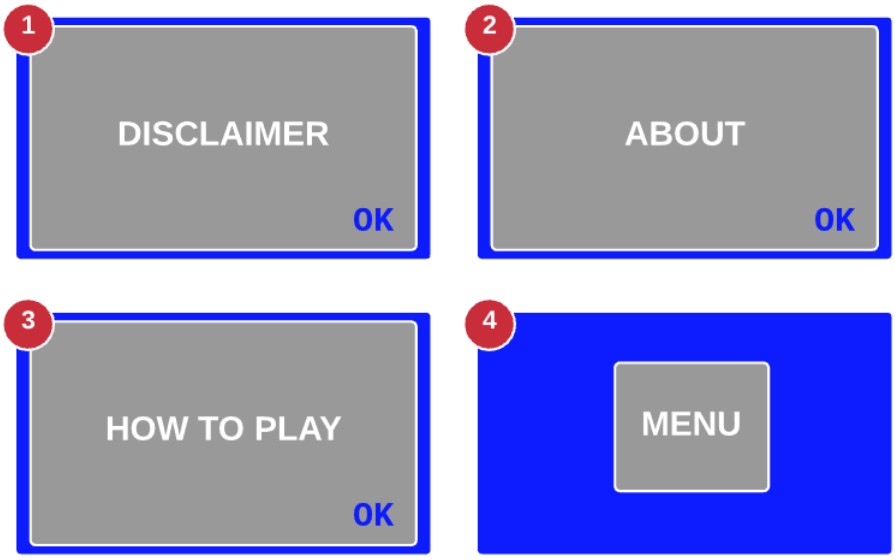


Figure 15: Pre-Menu sequence (made with Lucidchart [33])

4.1.2 Menu

The Menu page will then proceed after the player has read and accepted the sequence of messages. From here the player can choose to navigate to the Terminal, Store, Password Strength, Achievements pages and re-visit the individual dialogs from before. The core game is conducted within the Terminal and is the only section that differs from the rest in terms of functionality and does not follow the same graphical layout as the other pages.

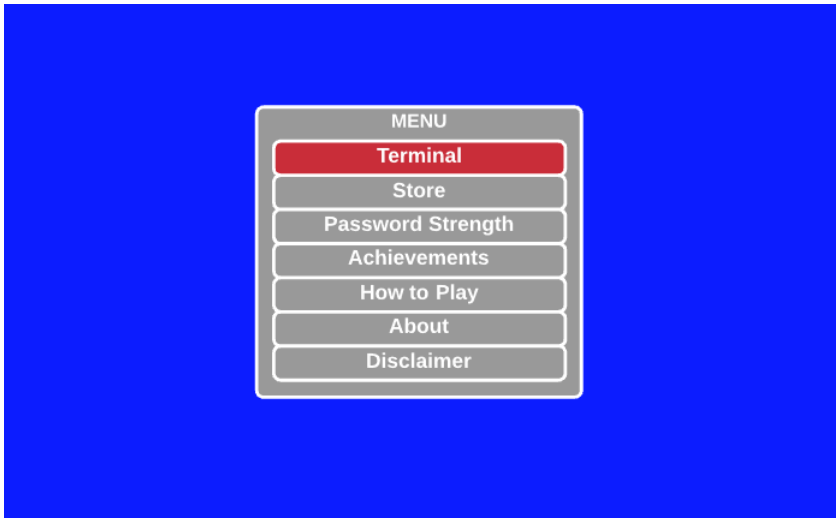


Figure 16: Menu page

4.1.3 Terminal

The Terminal is where the player will spend most of their time. From here they can accept profiles, execute password cracks, change the text colour of the terminal, view the list of available commands and the syntax that each algorithm has. Every action within the terminal can only be done through entering commands, except for navigating to the Menu that can be achieved by pressing the [Tab] key.

Figure 17: Terminal

4.1.4 Store

Pressing the [Tab] key from within the Terminal will bring up the Menu where the player can visit the store and buy all the necessary algorithms and dictionaries to crack passwords. Insufficient funds or selecting an already owned item will trigger an error message that will not permit the purchase.

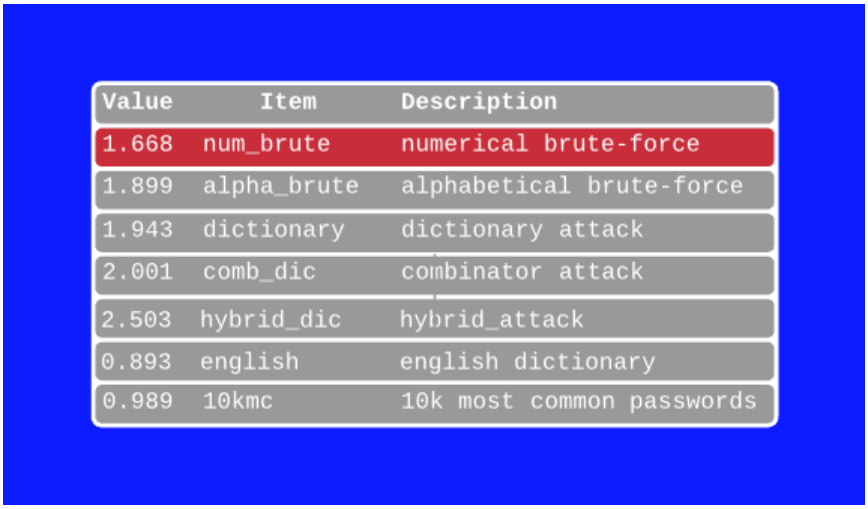


Figure 18: Store page

4.1.5 Password Strength

The Password Strength page is an additional feature that steers away from the core game. Here players can experiment with different passwords and see information on how secure they are, indicating how long it would take to crack them with different

attacks, the number of guesses it would take to crack it, its entropy value and a score out of 4. Bitcoin can also be awarded on this page if the player is able to construct a password that is deemed secure by having an entropy value greater than the threshold. A feedback option will also be available for passwords that have scored less than 4 which will inform the player on its inherent vulnerabilities and suggest how to improve its security.

Figure 19: Password Strength page

4.1.6 Achievements

The Achievements page is where the player can view a list of completed and uncompleted challenges, which can be unlocked by playing the core game as well as spending time creating passwords in the Password Strength page.

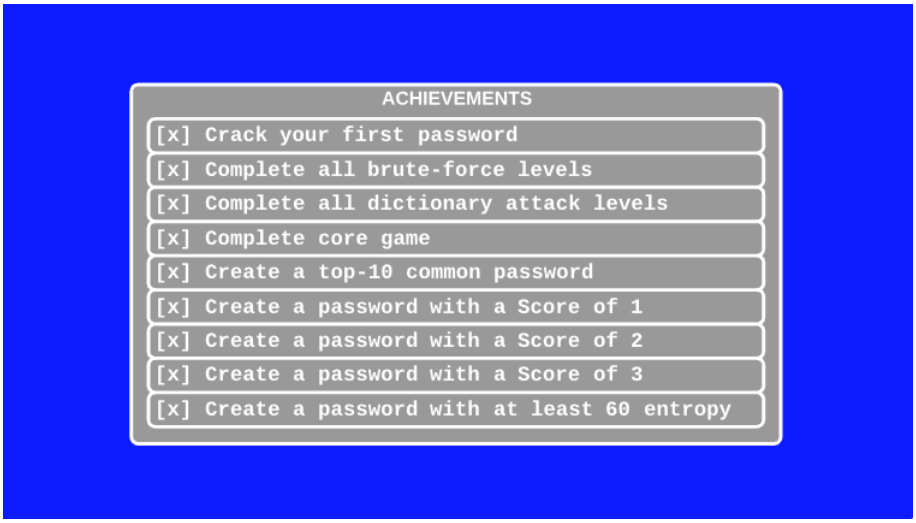


Figure 20: Achievements page

4.2 Game Mechanics & Modifications

As per Requirement No. 1 and perhaps the most important requirement to implement, the game must improve the player's understanding of password security. It was initially planned to simply make the player repeatedly crack passwords that get more difficult as they progress and unlock more cracking techniques, in hope that with a fully practical experience they would realise the insecurities of weak passwords. This approach did not make it into the final design as it exhibited several potential issues that impact user experience and therefore quality of results. Brute-forcing a task alone is not the most effective way to get results, nor will it trigger any motivation to continue playing which is a requirement in itself for the use of Gamification.

4.2.1 Level Structure

The core game consists of 24 levels, with the first 19 being more informative and tutorial orientated compared to the last 5, which revolve around more independent challenges. The idea is to cover all the content and familiarise the player with the mechanics of the algorithms during the first stage, which can then be applied to the final levels that can generate any sort of password based on what they have been taught. As per Requirement No. 6, designing the levels like this caters for a wide range of abilities as inexperienced players will be comfortable starting from scratch, and more experienced players can revise topics and anticipate the later levels.

Level	Topic	Requried Items	Reward Range	Hint Provided?
1	Written Passwords	None	0.5 - 0.6	No
2	Plain-text Passwords	None	0.6 - 0.7	No
3	Hash Functions	None	0.7 - 0.8	No
4-6	Numerical Brute-Force	num_brute	0.8 - 1.2	Yes
7-9	Alphanumerical Brute-Force	alpha_brute	1.1 - 1.6	Yes
10	Dictionary Attack	dictionary, english	1.4 - 1.8	Yes
11-12	Hash Tables	dictionary, english	1.5 - 2.0	Yes
13	Salts	dictionary, english	1.7 - 2.1	Yes
14	Combinator Dictionary Attack	comb_dic, fnames, snames	1.8 - 2.3	Yes
15-16	Peppers	comb_dic, fnames, snames	1.9 - 2.5	Yes
17-18	Hybrid Dictioanry Attack	hybrid_dic, 10kmc	2.1 - 2.8	Yes
19	Salt & Pepper	hybrid_dic, 10kmc	2.3 - 2.9	Yes
20-24	Random Profile Generation	ALL ITEMS	7.5 - 12.5	Limited

Table 4.1: Level Structure

As per Requirement No. 5, the difficulty of each level should should get more challenging as the player progresses through the game. This has been achieved by designing the levels such that the complexity of the password and algorithm required to crack it

increases for each level. The fundamental concepts of the topics also get more technical and will demand more intuition from the player in order to pass the level, such as knowing where to place a salt or guessing the type of pepper. This is especially the case for the final levels where the player has to crack passwords with little guidance using all of their acquired knowledge and items from the previous levels.

Levels 1-3: *introductory*

- Introduces the player to the vulnerabilities of plain-text passwords and the fundamentals of hashing functions for the safe storage of passwords. The password on these levels are static and do not require any algorithm to crack them and do not provide any hints.

Levels 4-9: *novice*

- Introduces the player to the application of brute-force techniques and gives the player a flavour of the systematic approach of obtaining a password. Hints involve telling the player how long the password is and how many character sets it consists of.

Levels 10-14: *intermediate*

- Introduces the player to dictionary attacks and the effectiveness of hash-tables and combinator attacks. Slightly more complex than the previous levels, the passwords on these levels are taken from random words in multiple dictionaries that can be salted. Hints provide information on what dictionaries have been used for the password.

Levels 15-19: *adept*

- Introduces the player to hybrid dictionary attacks as well as powerful pepper counter-measures. Passwords can include salts, peppers or a combination of both. Hints can sometimes inform what pepper was used and the integer range the password consists of.

Levels 20-24: *expert*

- Does not introduce the player to any new material. Passwords are associated with a generated profile and can be cracked with any of the acquired algorithms or dictionaries. Hints convey less information and suggest more on what algorithm should be used rather than characteristics of the password.

4.2.2 Terminal Interface

Requesting a profile from the Terminal Window will bring up a summary of the profile that informs the player of the proceeding topic, items required for the crack and the amount of Bitcoin awarded for completion. If the player owns all the necessary items they can accept by entering *y*.

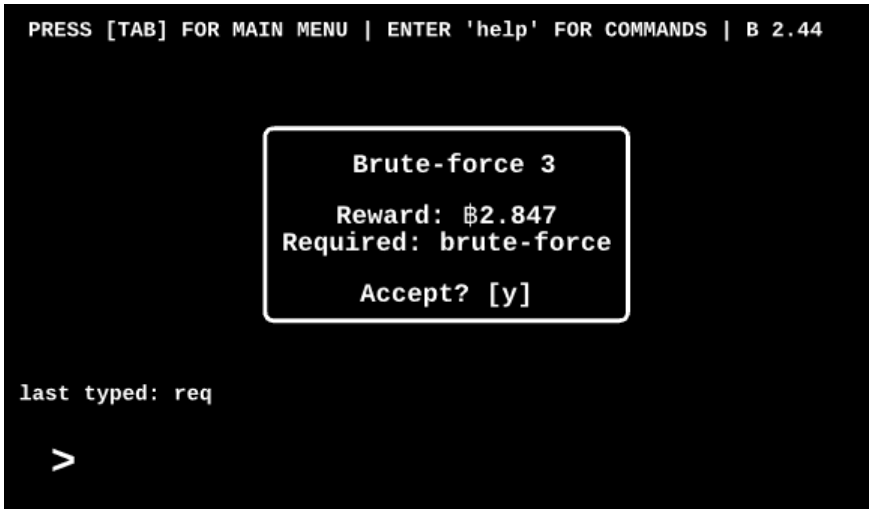


Figure 21: Profile Summary - all required items

However if the player does not own all the required items for this particular profile they will be instructed to visit the store before proceeding. For instance, it may be the case that the password associated with the profile can only be cracked with a dictionary attack with a list of commonly used passwords, therefore the player must purchase these items to continue.

Figure 22: Profile Summary - missing required items

Once the player has all the items required for the level and accepted the profile, they are either presented with an informative paragraph that describes in considerable detail the topic of the level (1-19), or a randomly generated profile (20-24) depending on what stage of the game they are on. A hint relating to the password followed by the hashed password itself will also be displayed. As per Requirement No. 4, an algorithm progress indicator would be a desirable feature that enables the player to actively view what stage the crack is on:

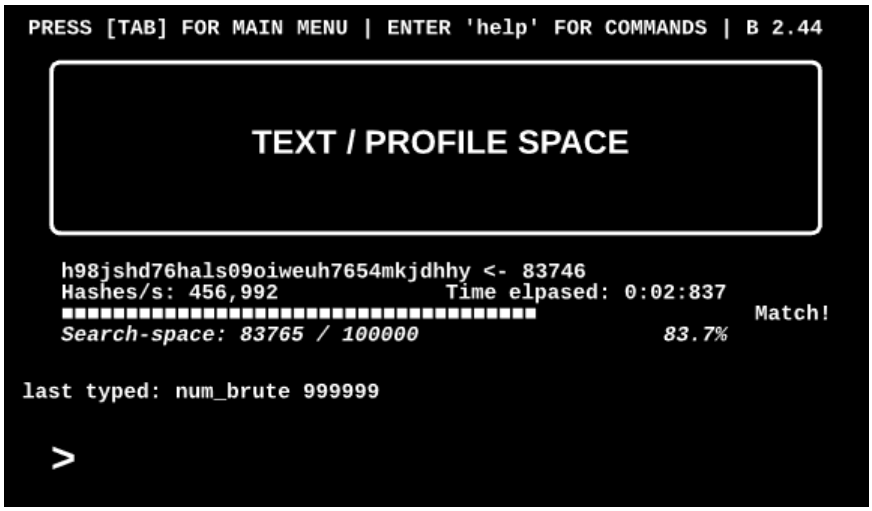


Figure 23: Level Template with Progress Indicator

The progress indicator will appear for any algorithm the player decides to use so long as they are not generating a hash table. Hash tables can only be generated for *dictionary, comb_dic* and *hybrid_dic* attacks that are un-salted or un-peppered and are presented differently:

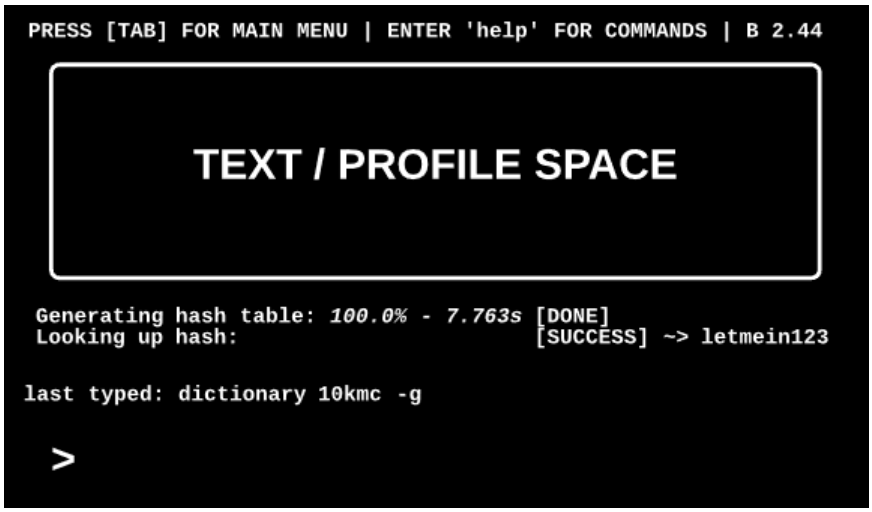


Figure 24: Level Template with Hash Table

If a match has been found then the terminal will instruct the player to enter the password to continue to the next level. If unsuccessful, they will have to try again and approach the crack from a different angle, adjusting the parameters of the algorithm.

4.2.3 Commands & Syntax

The header of the Terminal Window informs the user that the list of commands can be accessed by entering *help* [B.1]. The purpose of the up and down arrow key to grab and clear commands is to resemble the functionality of a terminal that facilitates entering commands. During the execution of an algorithm, typing will be disabled unless the algorithm is terminated by pressing the [ESC] key. From here (or anywhere within the Terminal) the player can enter *syntax* to bring up information on how to use the algorithms and the parameters they take [B.2].

Application of the algorithms and how they are used to crack hashed passwords will be explored in more detail in the next chapter. As per Requirement No. 19, it was stated earlier in the Literature Review that a custom *keyword* algorithm was going to be implemented into the game that would involve substitution, concatenation and repetition mechanics to crack a password. It has been decided to omit this algorithm from the final design as the player could feel over encumbered with all the other items they are using. The game already highlights the main aspects of password security - including another algorithm would only serve as an extension that will not provide the same educational value as the other techniques. Moreover, the techniques that have been covered are the concrete examples used in password cracking that are more relevant and hold a higher significance than one which is already similar to existing ones.

4.2.4 Entertainment Value & Reward System

As per Requirement No. 2, the game must increase entertainment value in order to provide the educational value through motivation - a crucial component to Gamification. This was achieved by incorporating reward-based mechanics and challenging tasks set within the player's ability, which ties in cohesively with Requirement No. 3; a reward system that encourages players to play at their own free will.

Evaluation of Existing Game Analysis suggests that a 1-dimensional reward system that can only accumulate points is not satisfying for the player and does not carry

a lot of intrinsic value. It was then considered to have a type of reward that can be expended as well as earned. The idea of having a currency-based reward system that players could invest their reward into items that actively contribute towards level progression sprung to mind. A problem then arose for how to ensure the player would never be in a situation where it is impossible for them to continue, because they do not have sufficient funds to purchase the items required for the level they are currently on. Increasing the lower-bound of the reward so that required items are always affordable was considered, but then it had to be made so that only those required items could be purchased, otherwise the player could mistakenly buy the wrong item and not have enough to buy the one they need. This was featured in the first design iteration where the player could only unlock (able to purchase) the item required based on what level they were on. However it did not make the final design as it did not offer much freedom of choice and made the reward system practically redundant. Players needed to somehow acquire additional Bitcoin that was not awarded from the core game.

The Password Strength feature was initially intended to increase entertainment value by including a section where the player could have a break from cracking passwords and instead test the strength of them. It was then considered that additional Bitcoin could be earned by encouraging the player to enter strong passwords determined from the entropy value. This mechanic actively makes the player think of strong passwords while getting paid, which serves a dual purpose of intrinsic motivation to create strong passwords and omit the possibility of running out of Bitcoin. Because of this, the lower-bound pay out from completing the levels has been changed so that it could work against the player, who is then forced to go in-between levels and utilise the Password Strength feature.

Another fundamental component of Gamification that can be used to trigger motivation is through the use of achievements. Achievements are a very effective mechanic as they can be designed to steer the player towards reaching a particular goal, which is why achievements for the game have been crafted to further encourage players to visit the Password Strength section as well as complete the core game [A.7]. Additionally, players are presented with a summary once they complete the core game that displays the total number of passwords they have cracked, total time taken to reach the end and how many achievements they have left to unlock - incentivising them to unlock all achievements and playing the game to its full potential.

Chapter 5

Implementation & Testing

This chapter will delve deeper into the process of putting the design into effect and discuss the technical challenges that were overcome.

5.1 Cracking Algorithms

The game exhibits the use of 5 different password cracking algorithms, including brute-force and several dictionary attack techniques. As stated in the previous section, the *keyword* algorithm has been omitted from the final design and has therefore not been implemented.

5.1.1 Numerical Brute-Force

The Numerical Brute-Force algorithm is the first algorithm the player unlocks and is the most intuitive. It only takes one integer parameter and will compute all hashes up to the specified length unless a match is found.

> num_brute [range]

```
// num_brute
private String num_brute()
{
    for (i = 0; i <= range; i++)
    {
        current = String.valueOf(i);
        if (MD5.getHashPassword(current).equals(hashedPassword))
            return generate.getPlainTextPassword();
    }
    return "No match!";
}
```

Figure 25: num_brute with range 9999

Figure 25 illustrates a successful crack that utilises the numerical brute-force algorithm by computing all the hashes from the range 0 - 9999 and locating a match at 9600.

5.1.2 Alphanumerical Brute-Force

As per Requirement No. 14, the Alphanumerical Brute-Force algorithm works in a similar way to Numerical Brute-Force but can be used to iterate over additional character sets. It can take either 1 or 2 character sets and will recursively compute combinations up to the specified length. It can also be used for cracking numerical passwords, but will prepend the unused character spaces with zeros.

> alpha_brute [cType] [cType] [length]

```
// available character sets
private char[] CHARSET; // populated by user [cType] [cType]
public final char[] LOWERCASE = "abcdefghijklmnopqrstuvwxyz".toCharArray();
public final char[] UPPERCASE = "ABCDEFGHIJKLMNOPQRSTUVWXYZ".toCharArray();
public final char[] NUMBERS = "0123456789".toCharArray();
public final char[] SPECIAL = "!@#$%^&*()-_+=~[]{}|:;<>,.?/".toCharArray();

// alpha_brute called with [length]
result = alpha_brute("", 0, length);

// recursively compute all combinations of CHARSET
private String alpha_brute(String str, int pos, int length)
{
    if (length == 0) // check string
    {
        current = str;
        i++;
        if (MD5.getHashPassword(current).equals(hashedPassword))
        {
            brute = true; // break condition
            match = str; // match found
        }
    }
    else
    {
        if (pos != 0) // reset pointer
            pos = 0;
        if (!brute)
        {
            for (int j = pos; j < CHARSET.length; j++)
                alpha_brute(str + CHARSET[j], j, length -1);
        }
    }
    return match;
}
```

Figure 26: alpha_brute with 1 character set

Figure 27: alpha_brute with 2 character sets

Figure 26 illustrates a successful crack that was achieved from utilising the alpha_brute algorithm by computing all the possible hashes of the uppercase character set of length 4 and locate the password 'VFLK'. Figure 27 illustrates the same behaviour but by applying a combination of lowercase and special characters of length 3 and locate the password 'wy+'.

5.1.3 Dictionary attack

As per Requirement No. 15 and 22, the Dictionary Attack algorithm will iterate over an entire supplied dictionary [B.3] unless a match is found. In these examples, we will be using the *standard (-s)* extension (described in more detail further into the chapter).

> dictionary [dictionary] [extension]

```
private String dictionary() throws Exception
{
    String st;
    if (genCheck) // generate hash table
      result = generateCheck(false);
    else // brute-force approach
    {
        while ((st = br1.readLine()) != null) // dictionary line
        {
            current = st.replaceAll("\\s+",""); // remove white-space
            i++;
            if (MD5.getHashPassword(current).equals(hashedPassword))
            {
                match = st; // match found
                return match;
            }
        }
    }
        return match;
```

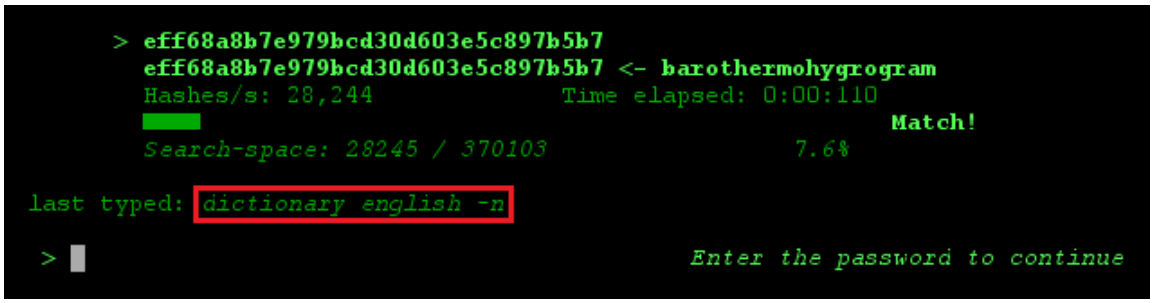


Figure 28: dictionary algorithm using English dictionary

Figure 28 illustrates a successful crack that utilises the dictionary algorithm with the English dictionary and locate the password 'barothermohygrogram'.

5.1.4 Combinator Dictionary Attack

As per Requirement No. 16, the Combinator Dictionary Attack concatenates all the words from one dictionary to another until a match is found. The order of the specified dictionaries is important as the algorithm will not compute the vice versa combinations.

> comb_dic [dictionary1] [dictionary2] [extension]

```
 private String combinator_dic() throws Exception
{
    if (genCheck) // generate hash table
      match = generateCheck(true);
    else
    {
        String st1;
        String st2;
        while ((st1 = br1.readLine()) != null) // dictionary 1 line
            {
                while ((st2 = br2.readLine()) != null) // dictionary 2 line
                {
                    i++;
                    current = (st1+st2).replaceAll("\\s+",""); // remove
                        white-space
                    if (MD5.getHashPassword(current).equals(hashedPassword))
                    {
                        match = current; // match found
                        return match;
                    }
                }
                resetBuffer(true); // point to first word in dictionary 2
                    after reaching EOF
            }
        }
    return match;
}
```

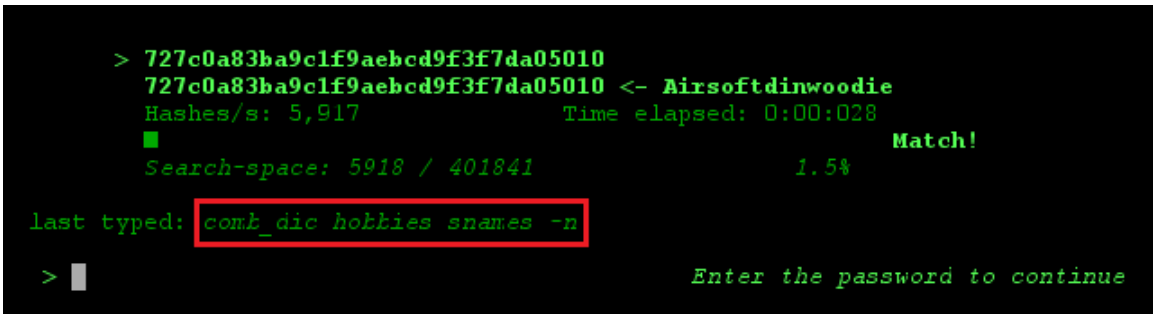


Figure 29: comb_dic algorithm using list of hobbies and list of surnames

Figure 19 illustrates a successful crack that utilises the comb_dic algorithm by combining a list of hobbies and surnames to locate the password 'Airsoftdinwoodie'.

5.1.5 Hybrid Dictionary Attack

As per Requirement No. 17, the Hybrid Dictionary Attack appends a brute-force keyspace to each word in the supplied dictionary.

> hybrid_dic [dictionary] [range] [extension]

```
private String hybrid() throws Exception
{
    if (genCheck) // generate hash table
      match = generateCheck(false);
    else
    {
        String st;
        while ((st = br1.readLine()) != null) // dictionary line
            {
                for (int k = 0; k <= hybridLength; k++) // numerical keyspace
                {
                    i++;
                    current = st.replaceAll("\\s+","") + String.valueOf(k);
                    if (MD5.getHashPassword(current).equals(hashedPassword))
                    {
                        match = current; // match found
                        return match;
                    }
                }
            }
        }
    return match;
}
```

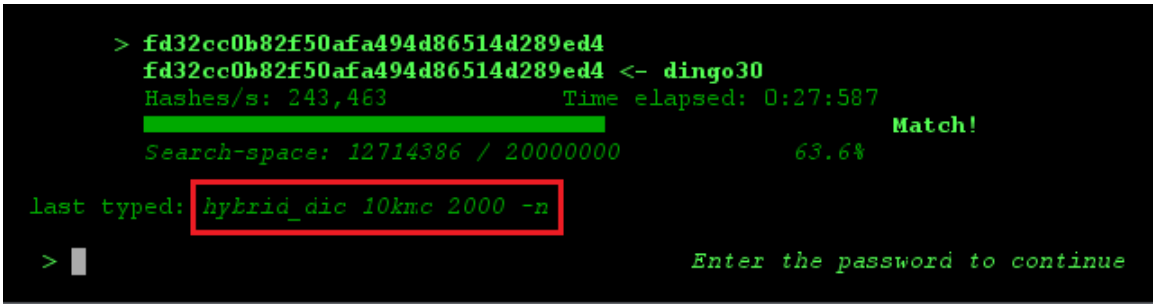


Figure 30: hybrid_dic attack

The figure above illustrates a successful crack that utilises the hybrid_dic algorithm to combine a list of words with a set of numbers and locate the password ‘dingo30’.

5.1.6 Extension Parameter

All of the dictionary attacks must include the extension parameter which tells the algorithm the type of attack to execute: *standard, generate hash table, salt, pepper* or *salt & pepper* [B.2].

Standard

The standard extension is used to execute the algorithm in a brute-force manner, as illustrated by the algorithms above by appending *-s*.

Generate Hash Table

As per Requirement No. 21, the Hash Table extension is used to generate a hash table and lookup a hash key inside it for a match, which is done by appending *-g*.

```
// look for password hash in generated hash table
private String genCheck(Hashtable h)
{
    if (h.containsKey(hashedPassword))
    {
        match = (String) h.get(hashedPassword);
        return match; // match found
    }
    else
        return match; // default value is "No match!"
}

// generate hash table
private Hashtable generateHashTable() throws Exception
{
    Hashtable hashTable = new Hashtable<>();
    String st;
    while ((st = br1.readLine()) != null) // read dictionary line
        {
            i++;
            hashTable.put(MD5.getHashPassword(st.replaceAll("\\s+","")),
                st.replaceAll("\\s+","")); // store pair
        }
    return hashTable;
}
```

Generating a hash table has a slightly different progress indicator than the standard extension. It has been stripped down to % completion and time taken with no progress bar or current hash attributes:

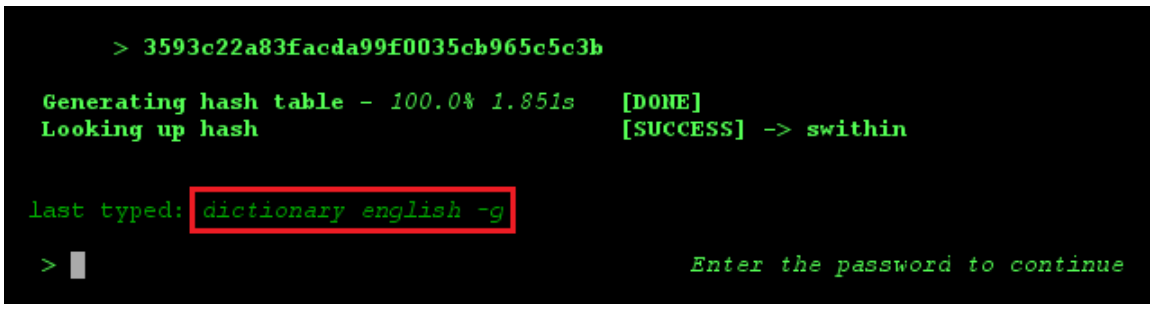


Figure 31: Successful Hash Table crack

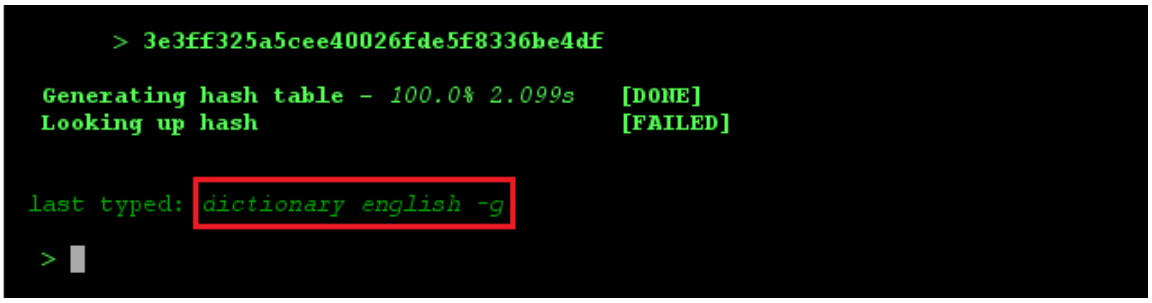


Figure 32: Unsuccessful Hash Table crack

Salts & Peppers

As per Requirement No. 20, certain levels will require the player to include a salt or pepper in their algorithm to crack a password. Salts are 4 characters long and consist of up to 2 character types, and peppers are 1 character long and consist of up to 2 character types:

```
// a salt would pass in length = 4 and sets = 2
// a pepper would pass in length = 1 and sets = 2
private String generateCharSet(int length, int sets)
{
    Random r = new Random();
    Algorithm tmp = new Algorithm(null,null,null);
    StringBuilder sb = new StringBuilder();
    StringBuilder bs = new StringBuilder();
    String g = randNum(); // generates 2 random unique numbers from 0-3
    int i;
    for (int x = 0; x < sets; x++)
    {
        i = Integer.valueOf(g.substring(x,x+1)); // grab first number
        switch (i) // append char space depending on grabbed number
        {
            case 0:
                sb.append(tmp.LOWERCASE);
                break;
            case 1:
```

```
                sb.append(tmp.UPPERCASE);
                break;
            case 2:
                sb.append(tmp.NUMBERS);
                break;
            case 3:
                sb.append(tmp.SPECIAL);
                break;
        }
    }
    for (int b = 0; b < length; b++)
        bs.append(sb.charAt(r.nextInt(sb.length()))); // grab random char
            from generated char space
    return bs.toString();
}
```

Given the fact that salts are non-secret, their value is stored next to the password hash which is used in the extension *-s [salt]*. Peppers on the other hand are secret values that are not stored and can be a lowercase *-l*, uppercase *-u*, special *-s* or numerical *-n* character added to the end of the password.

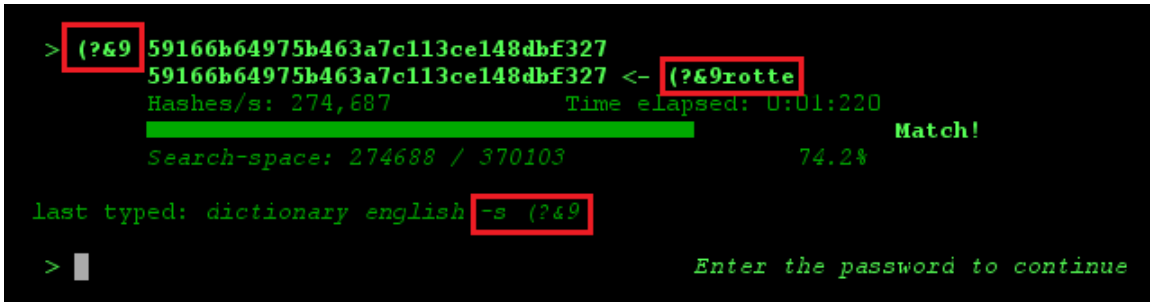


Figure 33: Using salts

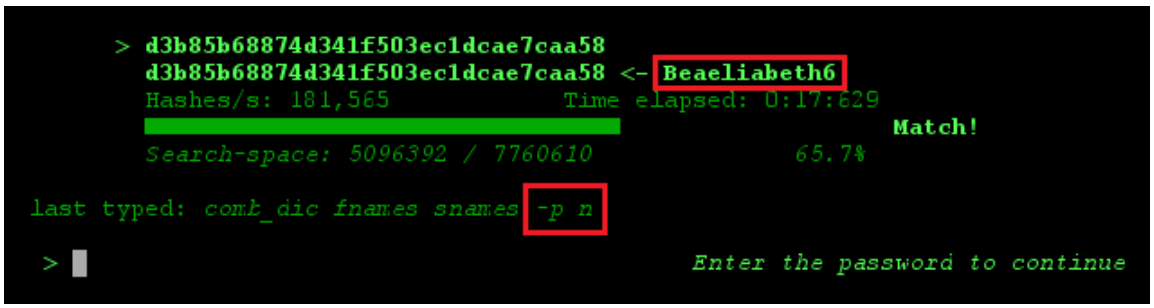


Figure 34: Using peppers

If the crack is successful, the plain-text output will also include the salt and/or pepper that was supplied with the extension. The password therefore will have to omit these values in order to continue to the next level. For instance, the password for the

salted password in Figure 33 would be 'rotte' and the peppered password in Figure 34 would be 'Beaeliabeth'. It is also possible to have both salt and pepper (or vice versa) included in the password:

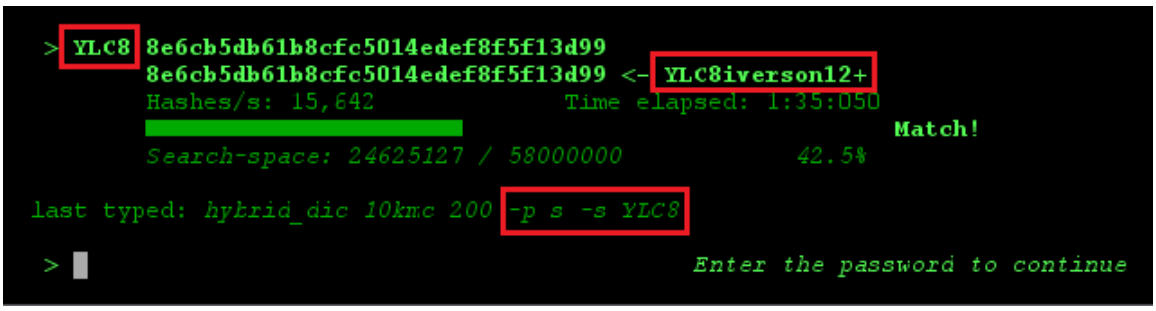


Figure 35: Using Salts & Peppers

The password in this case would be 'verson12' excluding the 'YLC8' salt and '+' pepper.

5.1.7 Validation

Validation is a very important requirement for an application of this nature. Entering an unrecognised command or incorrect algorithm syntax will output a *Syntax error* after the *last typed* text if it does not follow several conditions. First and foremost, the command is checked against a list of Terminal Commands that are always available, i.e. *req, help, syntax, setCol*. Failing these, it then checks for algorithm syntax by using a lot of string manipulation that returns false if an exception is caught at any point during the process.

```
// check if player owns algorithm
private static boolean validAlg()
{
    if (lastTyped.contains(" "))
    {
        int dlim = lastTyped.indexOf(" ");
        return ALGORITHMS.contains(lastTyped.substring(0, dlim));
    }
    else
    {
        return ALGORITHMS.contains(lastTyped);
    }
}

// validate algorithm (dictionary example)
public boolean validate()
{
```

```
    boolean valid = true;
    try
    {
        /* some variable definitions */
        switch (algorithm)
        {
            case ... :
            case "dictionary":
                    c = "x";
                    while (!c.equals(" ")) // grab inputted dictionary
                    {
                     c = command.substring(start+1+i,start+2+i); i++;
                    }
                    dictionary = command.substring(start + 1,
                        start+i).trim();
                    tmp = start + dictionary.length();
                    extension = String.valueOf(command.substring(tmp + 2,
                        tmp +4)); // grab extension
                    if (command.substring(tmp+2).equals("-g"))
                     genCheck = true; // generate hash table
                    else
                     valid = hasSaltAndPepper(tmp+2, extension); // validate
                         extension

                    if (!checkDictionary(dictionary)) // validate dictionary
                     valid = false;
                    else
                        resetBuffer(false); // initialise dictionary
                    break;
        }
    }
    catch (Exception e) // catch any extensions
        {
         valid = false;
        }
        return valid; // return true if no errors
}
```

The `validate()` method also initialises the necessary variables at the same time it validates the input. If it returns `true` then another method `execute()` is called that performs the crack.

5.1.8 Cracking Times & Multi-threading

As per requirement no. 8, feasible cracking times is a crucial property the game must have so that players are not forced to wait prolonged periods of time cracking

passwords, which has a risk of inducing boredom and as a result loss of motivation. There are two main mechanics that were implemented to ameliorate the time it takes to crack a password: multithreading and data size.

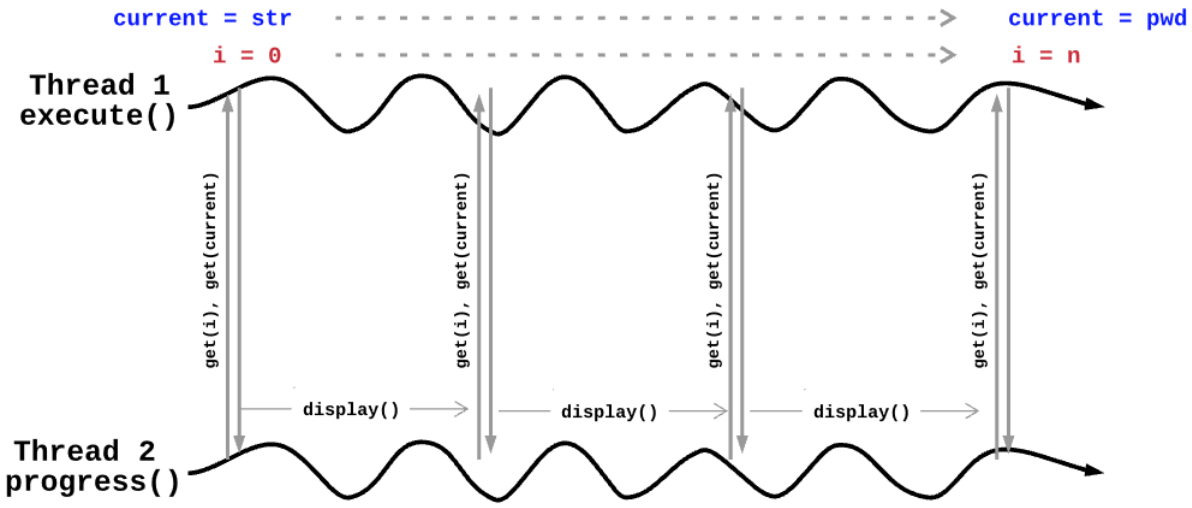


Figure 36: Multi-threading algorithm and GUI components

The figure above illustrates how multi-threading is used to separate algorithm execution from displaying algorithm progress. Thread 1 is independent from Thread 2 and purely handles the internal computation, providing the necessary getters for Thread 2 which handles the graphical output of the algorithm in progress. This approach is remarkably more effective than having just a single thread to handle both algorithmic computation and GUI components. However there is a negligible consideration that arises from this design, as theoretically the figures being displayed by Thread 2 are not updating at the same frequency as the variables in Thread 1. For instance, the new values for `i` and `current` would have gone through several iterations during the time it took for the `display()` method to output hashes/s, search-space, % completion, current hash and password, time elapsed and update the progress bar.

All of the passwords that are generated for each level are guaranteed to have a search-space $\leq$ 58,000,000 - given the worst case scenario on level 19 if the password is the last word in the dictionary followed by the maximum range value followed by the last character from the special character set. Tested on an Intel i5 7th Gen processor, the average number of hashes it could compute in a second is around 500,000 which would take just under 2 minutes to crack. It is possible for a player to enter a command

which produces a larger value, but does not effect the worst case time. As per Requirement No. 9, utilising the specifications of the system to maximise performance proved to be way beyond the scope of this project and has not been implemented.

5.1.9 Expert Level Generation

Unlike its predecessors, expert levels provide no theory and just display the relevant information to crack the password which is associated with the randomly generated profile. They are fully dynamic and will display different information and generate new passwords each time the application is run.

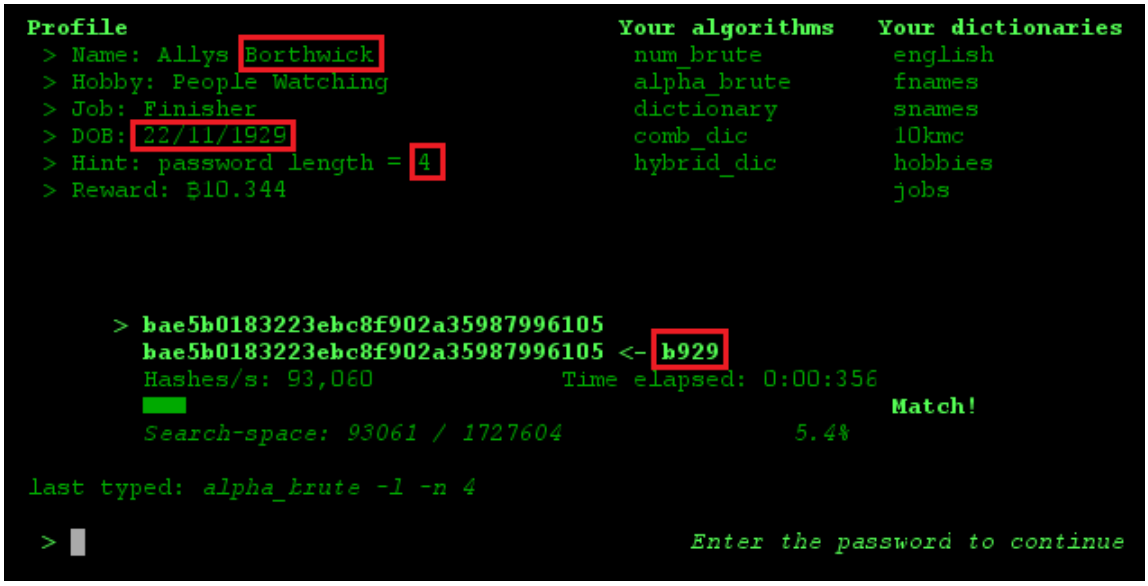


Figure 37: Password associated with Last Name and DOB

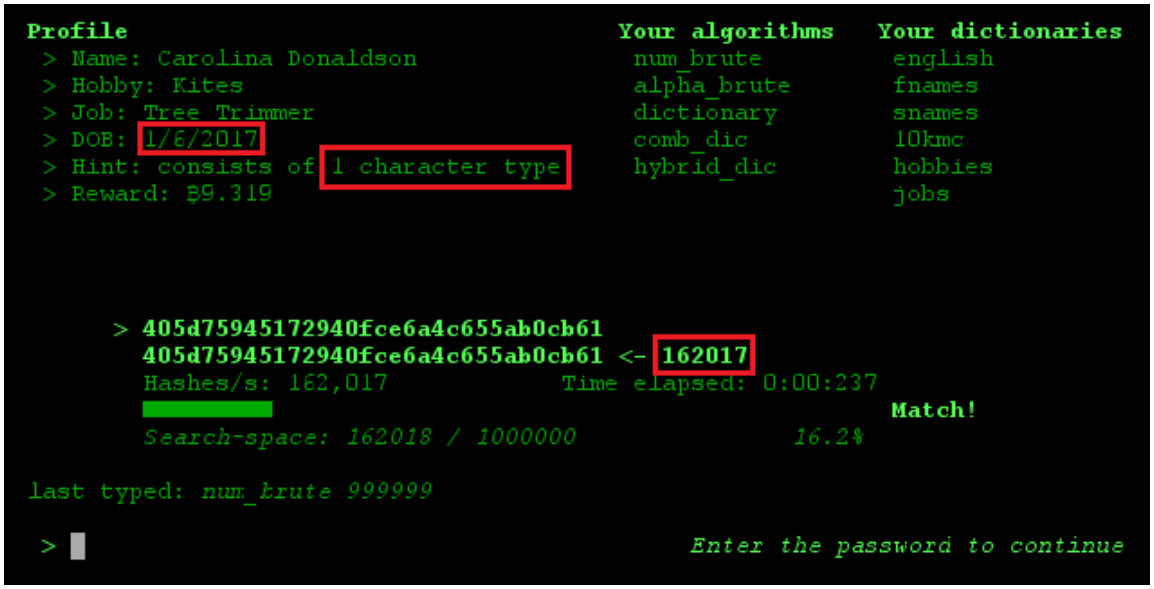


Figure 38: Password associated with DOB

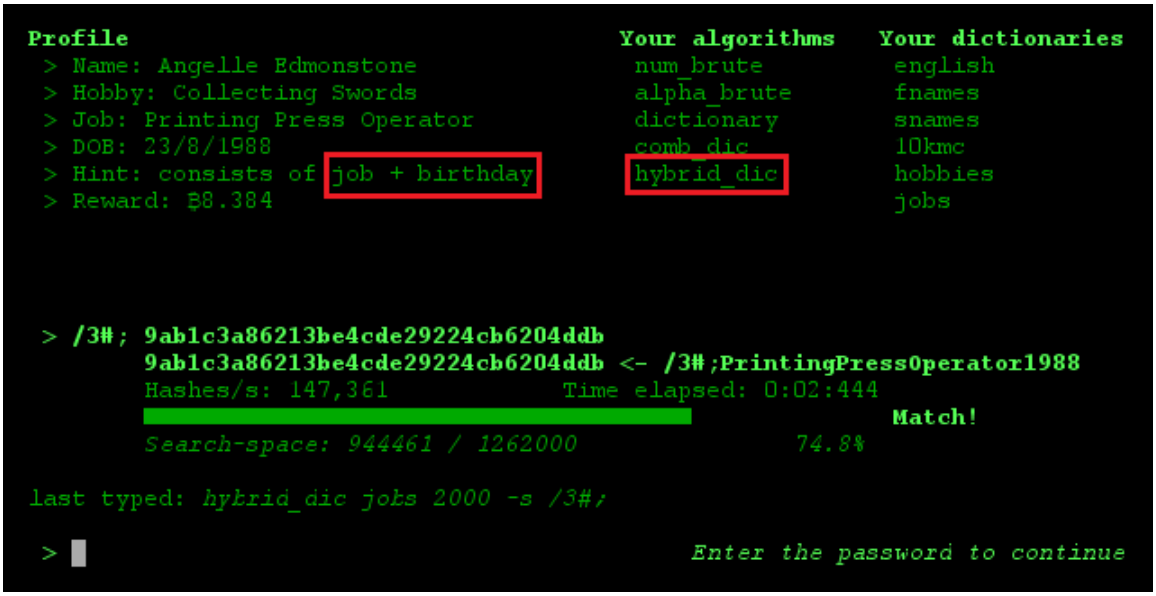


Figure 39: Password associated with Job and DOB

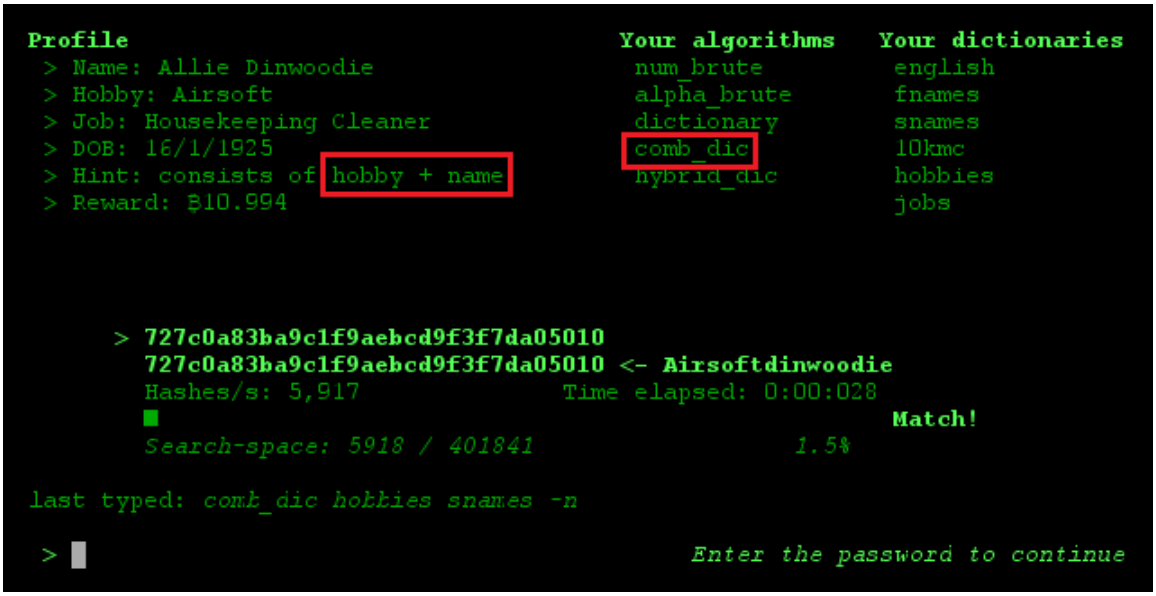


Figure 40: Password associated with Hobby and Name

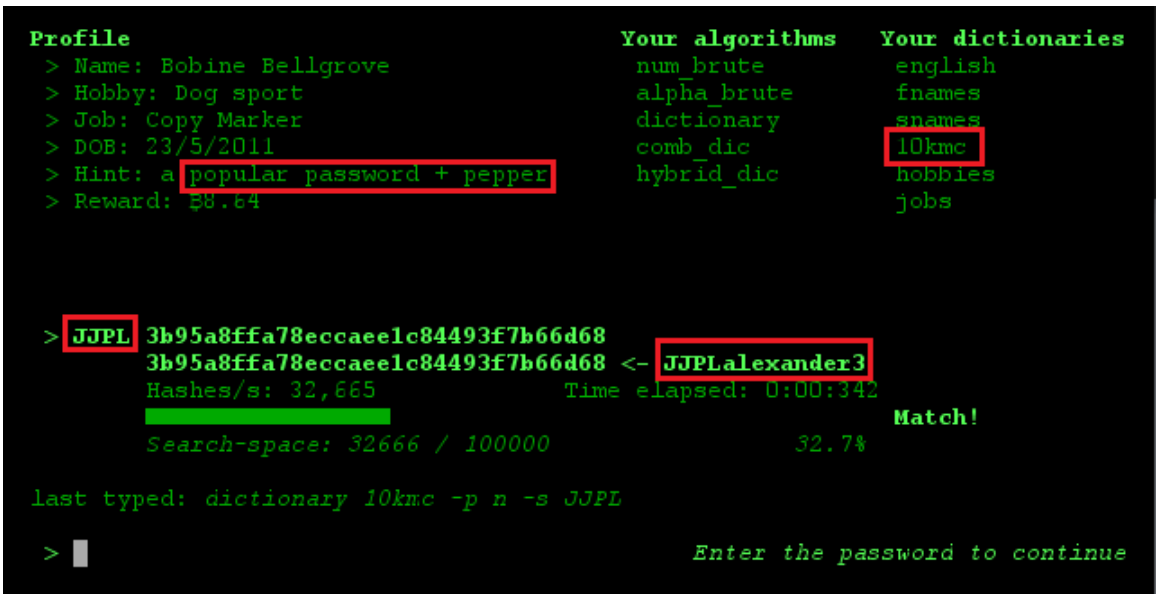


Figure 41: Common password with salt and pepper

Figure 44 demonstrates the possibility of a slight curve-ball the player has to overcome as the hint does not imply that the password is associated with the profile, but with a particular dictionary instead.

5.1.10 Libraries

Lanterna

Lanterna is a Java library allowing you to write semi-graphical user interfaces in a text-only environment [34]. Given the theme of the game it fits perfectly for the interface that was required, including the 'BIOS' looking Menu page as well as the Terminal emulator. Most of the development was done through the imported Maven library, except for implementing a few final amendments which effected the Swing Terminal Frame that Lanterna inherited. This included repositioning the window to the center of the screen, changing the title and icon of the window and disabling resizeability of the window. This could not be achieved though the imported library due to the read-only integrity of Maven, so the entire Lanterna repository was cloned into the working repository and changes were made there. A fat-jar file then had to be created with the edited library before being packaged into an executable JAR file.

nbvcxz

nbvcxz is a password strength estimator Java library that was the driving force behind the Password Strength page. The estimation is accomplished by running a password

through different algorithms looking for matches in any part of the password on: word lists, common dates, common years, special patterns, repeating characters, repeating sets of characters, and alphabetic sequences. [35].

5.2 Testing

Testing is a crucial phase in any system life-cycle that ensures quality by detecting software failures as well as gathering feedback from external users. This section will thoroughly explore how both of these approaches were implemented.

5.2.1 Unit Testing

As per Requirement No. 23, unit tests were written for individual components of the application to validate their function. Many logical cases were overlooked during development that Unit Testing was able to identify, which have since been patched and made the source code more robust.

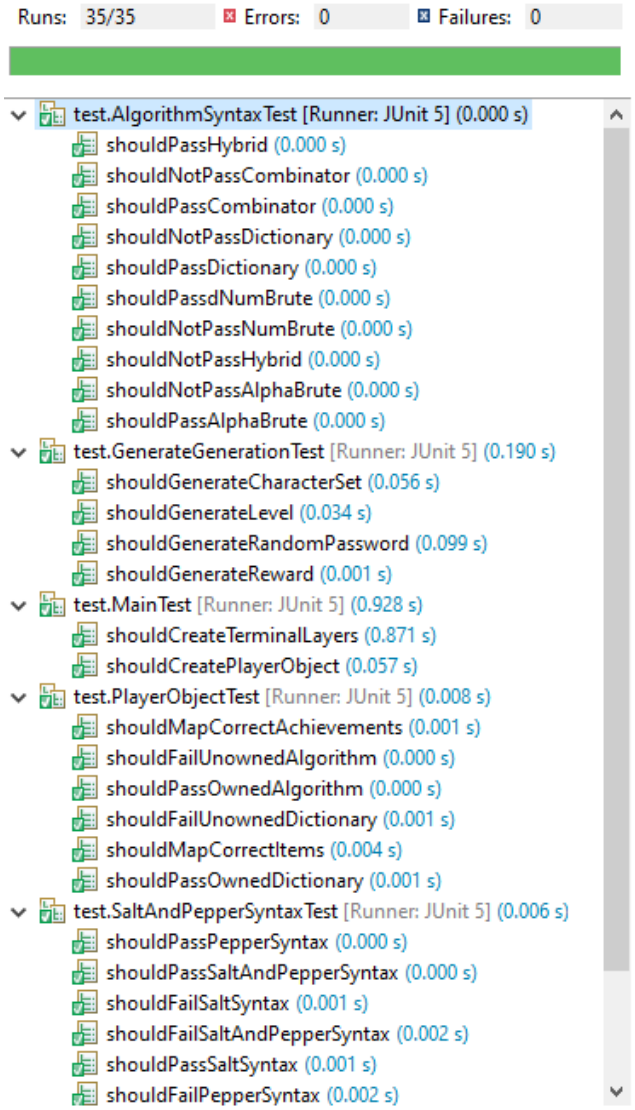


Figure 42: JUnit Test Cases

Figure 42 highlights some important test cases that ensures certain features are working as expected. Once the `validate()` method used to validate the algorithm syntax was working as expected, it became a very useful tool for testing inputs against their outputs that could be applied to all algorithm, salt and pepper syntaxes.

```
@Test
public void shouldNotPassDictionary()
{
   // Given a player owns the dictionary algorithm + English dictionary
   Player player = new Player();
   player.setPurchaced("00100100000");

   // When supplied with wrong extension
   String dictioanryAttack = "dictionary english -x";
   Algorithm alg = new Algorithm(null,dictioanryAttack,player);

   // Then output should be false
   Assert.assertTrue(alg.validate() == false);

   // When supplied with wrong extension
   dictioanryAttack = "dictionary english n";
   alg = new Algorithm(null,dictioanryAttack,player);

   // Then output should be false
   Assert.assertTrue(alg.validate() == false);

   // When supplied with no extension
   dictioanryAttack = "dictionary english";
   alg = new Algorithm(null,dictioanryAttack,player);

   // Then output should be false
   Assert.assertTrue(alg.validate() == false);
}
```

Example test case for invalid Dictionary algorithm syntax

As per Requirement No. 24, code coverage is an important metric to determine what areas of code have been used and tested, which could imply a lower chance of the application containing errors if the test cases support it. A healthy coverage report of 93% was achieved when running all the test cases which ensures a high level of confidence that the application is well tested.

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
Password Gamification	93.0 %	10,690	809	11,499
src	93.0 %	10,690	809	11,499
(default package)	0.0 %	0	34	34
algorithms	86.5 %	2,218	346	2,564
generate	99.9 %	2,629	2	2,631
metadata	99.6 %	1,051	4	1,055
player	98.3 %	237	4	241
test	93.5 %	1,253	87	1,340
ui	90.9 %	3,302	332	3,634

Figure 43: Coverage Report for Test cases

5.2.2 Beta-Testing & Amendments

As per Requirement No. 25, beta-testing is a great way to enhance the functionality and overall user experience of the application during the final stages of development. The following express challenges that were overcome as well as amendments made from user-testing.

Difficulty

Before hints were introduced on each level, a common theme in the feedback suggested that the levels were too difficult and did not provide enough guidance to crack the password. Hints on levels 3-19 convey information relating to the parameters the player should use for the algorithm in question, and hints on levels 20-14 describe more the type of algorithm to be used.

Password Strength

One of the beta-testers was able to construct a password so complex that the number of guesses exceeded the width of Terminal Window and broke the page. Additional validation that limits the password length has been implemented so that the maximum complexity one can achieve does not cause this issue.

Saving Progress

As per Requirement No. 13, an initial requirement was that a player could save their progress and pick up from where they left of if they closed the application. Due to the nature of the game being an executable JAR file, this was no longer possible as JAR is an archive which is meant to be unchanged.

Dictionary Terms

Another unforeseeable problem was the use of profanities included in the list of 10-thousand most common passwords. It is important however that the list remains in the game as it exemplifies how people think of passwords which is a key component to the research. This has been noted in the disclaimer [A.1].

Multi-threading

The use of multi-threading resulted in many interesting issues, especially when the player started to navigate or enter commands while a crack was in progress. To prevent any problems from occurring, keyboard input is disabled except for the [ESC] key that terminates the thread running the algorithm crack.

Chapter 6

Results & Discussion

This chapter will be analysing the success of the project by comparing the practical results from the questionnaires to the requirements, aim and objectives made in chapter 3. Further work including limitations and what could be improved upon will also be discussed.

6.1 Evaluation of Requirements

6.1.1 Gamification Mechanics

There are three requirements that fall under the mechanics of Gamification: **Increase entertainment value**, **Motivating reward system** and **Linear difficulty increase**. The User Experience section in the Post-game Questionnaire assess whether these requirements are met by providing a scale on how much they agree with the statement - included in this section users could also provide additional feedback on their overall experience such as:

"I thoroughly enjoyed this experience. I really liked the fact that you could build up bitcoins, this made it more fun, and provided a great incentive. My favorite part though would be the password security checker. I think this is a really useful tool!"

"Very nice game - surprisingly fun and satisfying, good explanations of concepts throughout and through provoking levels. Best of all, the fact you could change the colour of the terminal!"

"The Bitcoin reward system was a nice touch and somewhat satisfying to gain".

In summary, these responses in particular demonstrate how the Bitcoin reward system, Password Strength checker and being able to set a custom colour to the Terminal are fun and engaging features - **fulfilling Requirement No. 2**.

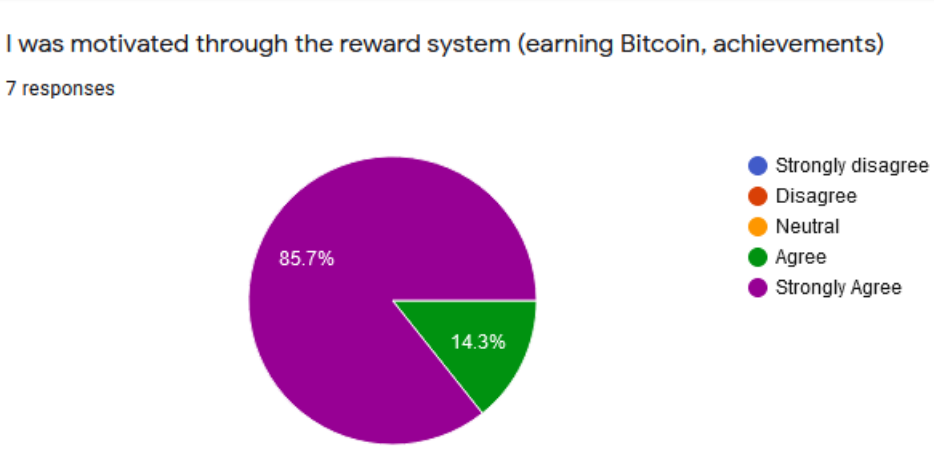


Figure 44: Reward System Response

85.7% of responses answered with "Strongly Agree" and the remaining 14.3% of responses answered with "Agree", indicating that all participants found they were indeed motivated through the reward system and **thus fulfilling Requirement No. 3**.

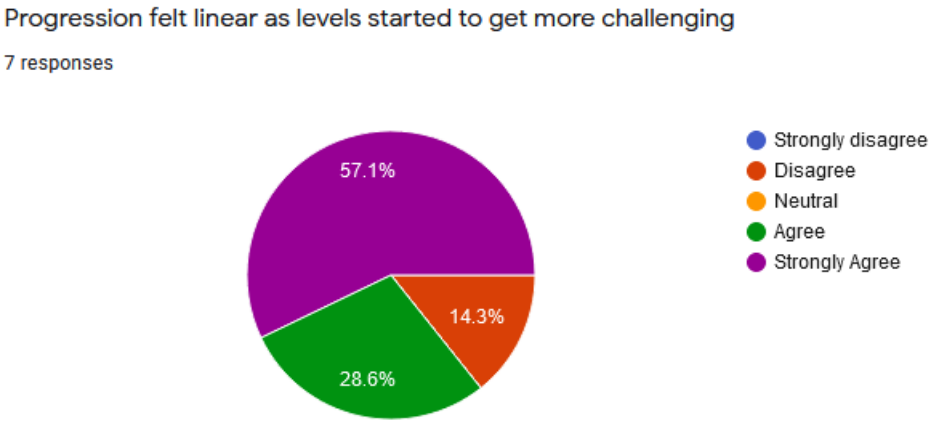


Figure 45: Linear Progression Response

Although mostly positive, linear progression was not a unanimous result amongst the users. One individual did not feel that the levels were increasing in difficulty at a steady rate, implying that **Requirement No. 5 was only partially met**.

6.1.2 Educational Value

Improve understanding of Password Security is the most crucial requirement of the project, so a thorough assessment must be conducted. There were several statements included in the Post-game Questionnaire that evaluate this requirement:

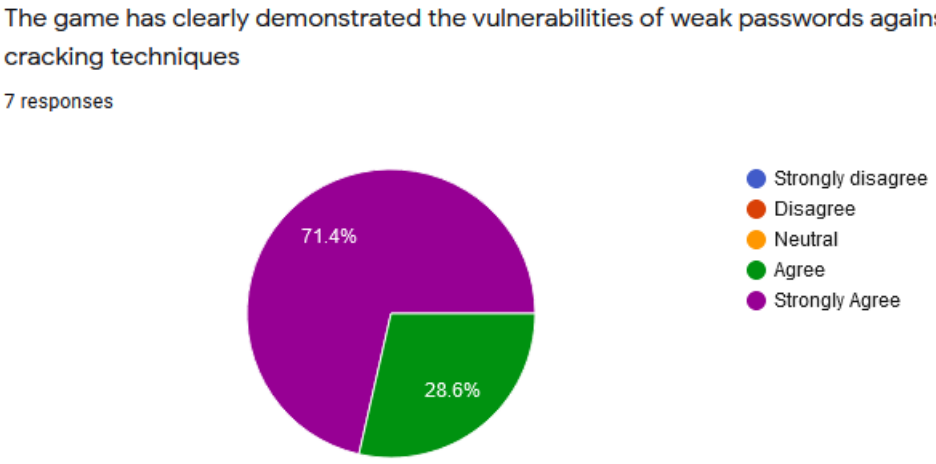


Figure 46: Demonstration of vulnerabilities of weak passwords response

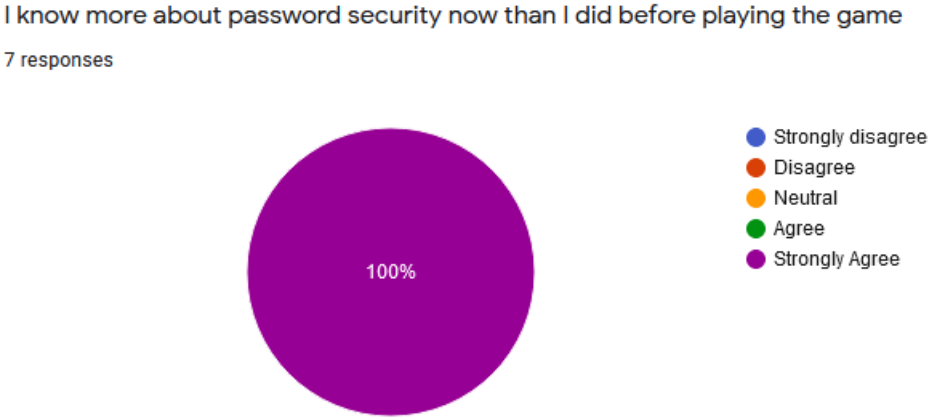


Figure 47: Increased understanding response

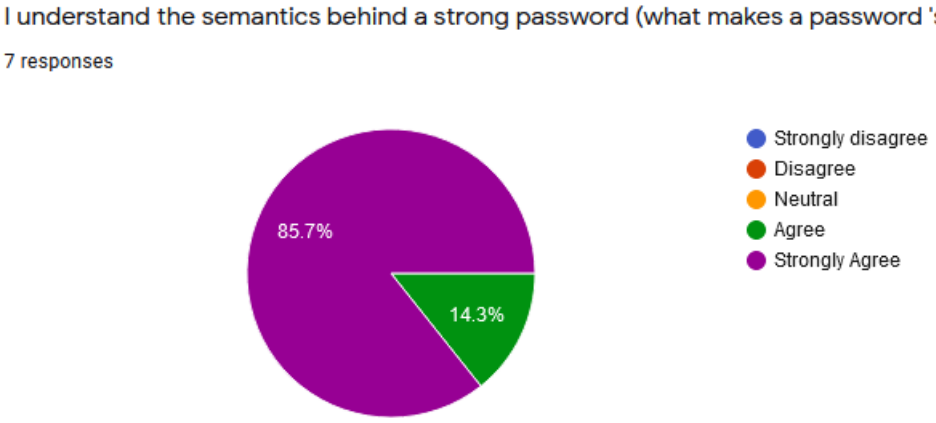


Figure 48: Password semantics response

The overwhelming positive feedback relating to the educational value that the game exhibited suggests that users feel they have an increased understanding of password security after playing the game - fulfilling Requirement No. 1.

6.1.3 Game Application

Requirements that revolve more around the user interface and functionality of the game were also evaluated through the User Experience section in the Post-game

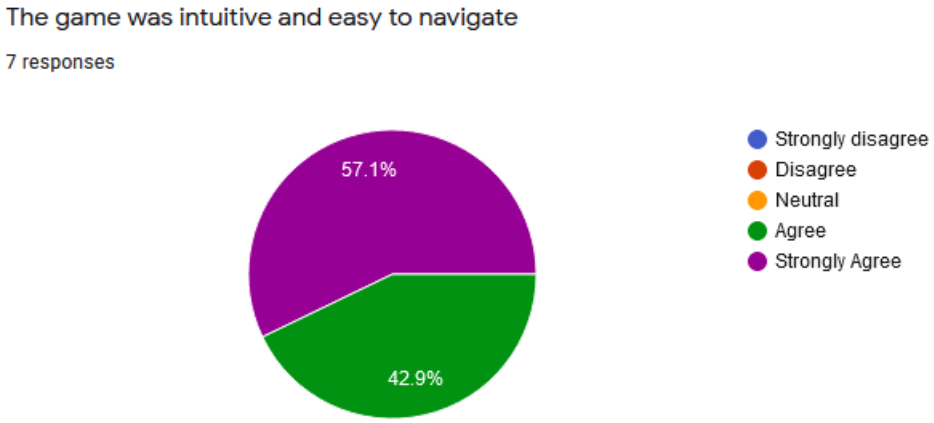


Figure 49: Intuitive design response

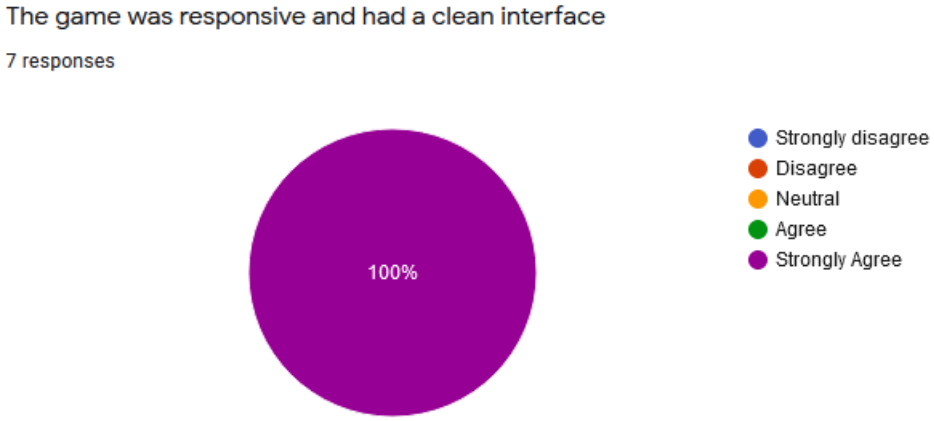


Figure 50: Clean interface response

The responses given for these statements satisfy Requirement No. 7 for a Clean and Intuitive interface.

6.2 Evaluation of Objectives

The first objective stated in Section 3.1.2 *"To understand password security and gamification by researching into current literature"* has been met in Section 2 with a Literature Review. Both topics were thoroughly researched which provided a more profound understanding of them, especially for Gamification which is a relatively novel technique. The extensive areas of password security that were documented allowed for the development of concise requirements that capture important requirements for the project goal.

The second objective *"To apply the principles of gamification to password security by developing a game that promotes the use of strong passwords"* is dependent on the first objective being met. A clear understanding of the topics is essential in order to correctly apply Gamification techniques to password security and provide educational value through motivation.

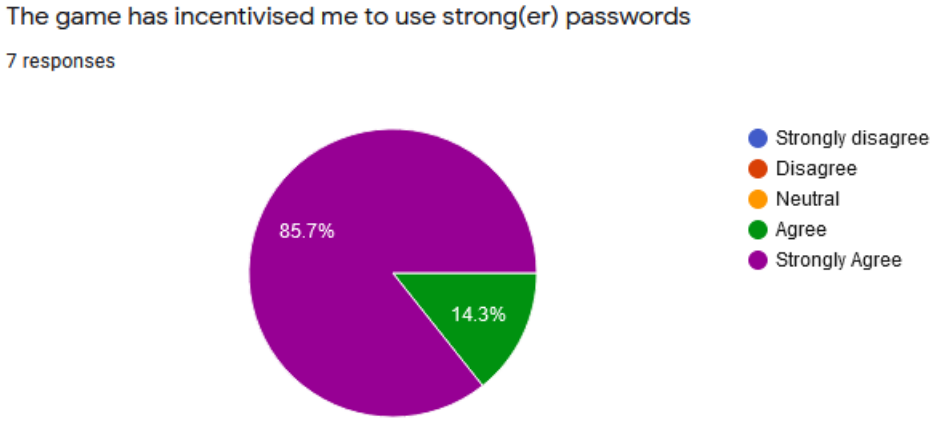
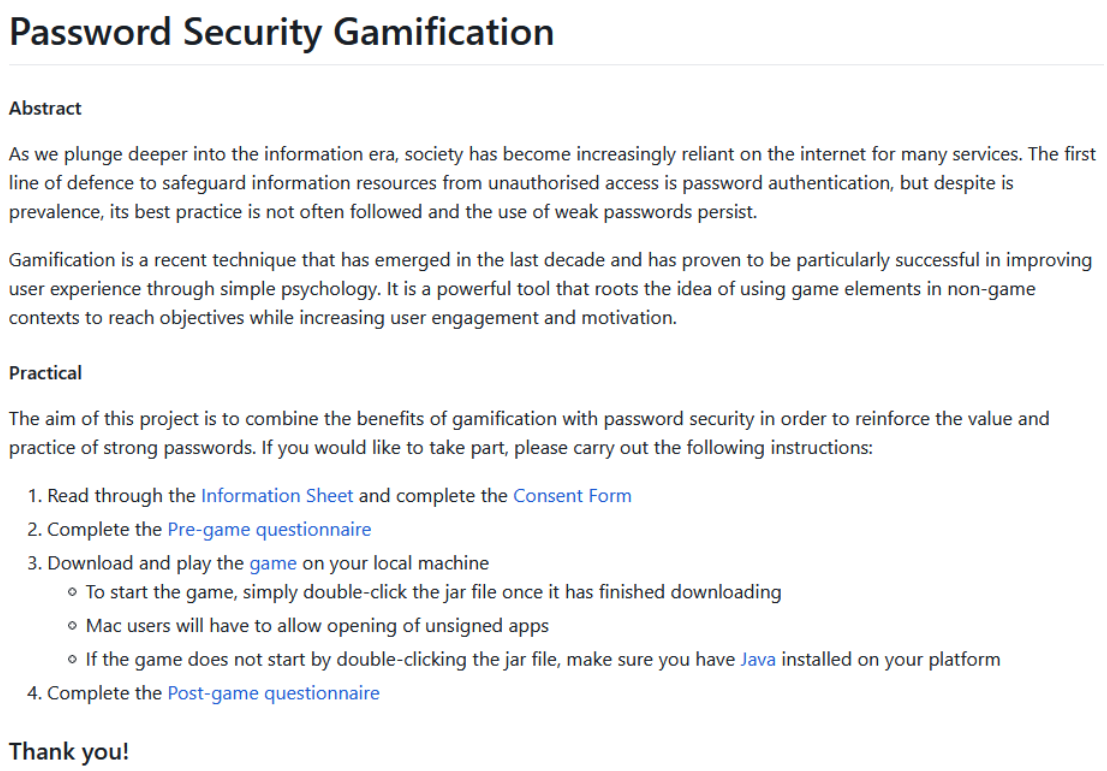


Figure 51: Incentivisation to use stronger passwords response

The second objective has been successfully met given the very positive response from users in the Post-game Questionnaire.

6.3 Evaluation of Aim

The primary goal and aim of the research stated in Section 3.1.1 *"To increase password security and awareness in younger people by using an entertaining method and creating a password cracking game that adopts gamification techniques."* has been slightly modified in light of the current situation (see COVID-19 Impact Statement between Declaration and Acknowledgements). The target group has since been changed to anyone that wishes to take part in the research and not just year 13 pupils. It had been planned to conduct short presentations behind the motivation of the research at two institutions to gather participants for the original target group, providing information sheets and consent forms in person. Alas, this evaluation could not be conducted so a more general and accessible method was implemented to gather participants for the project: through the University of Sheffield's Volunteer's List, friends and family. A Github link containing details of the research and how to take part was included in a formal email which was sent to the new target group. [36]



Password Security Gamification

Abstract

As we plunge deeper into the information era, society has become increasingly reliant on the internet for many services. The first line of defence to safeguard information resources from unauthorised access is password authentication, but despite is prevalence, its best practice is not often followed and the use of weak passwords persist.

Gamification is a recent technique that has emerged in the last decade and has proven to be particularly successful in improving user experience through simple psychology. It is a powerful tool that roots the idea of using game elements in non-game contexts to reach objectives while increasing user engagement and motivation.

Practical

The aim of this project is to combine the benefits of gamification with password security in order to reinforce the value and practice of strong passwords. If you would like to take part, please carry out the following instructions:

1. Read through the Information Sheet and complete the Consent Form
2. Complete the Pre-game questionnaire
3. Download and play the game on your local machine
   - To start the game, simply double-click the jar file once it has finished downloading
   - Mac users will have to allow opening of unsigned apps
   - If the game does not start by double-clicking the jar file, make sure you have Java installed on your platform
4. Complete the Post-game questionnaire

Thank you!

Figure 52: Github profile

Nevertheless, the research was completed and valuable results were collected. Evaluation based solely on how much users agree with statements is not sufficient enough to determine whether the aim was a success, which is why both questionnaires have

a 'quiz' style section that are used to evaluate any improved results.

The purpose of the Pre-game Questionnaire gauges the user's current understanding of password security by asking for the definition of certain terms that will later be presented in the game. These questions are single answer multiple choice, and include a 'Not sure' option that the user is encouraged to select if they do not know the answer for the question. The Post-game Questionnaire follows a similar theme by asking questions relating to the topics described in the game as well as the first questionnaire, but are more multi-answer orientated. Responses from both questionnaires are illustrated in section Appendix D and summaried below:

Q	Pre-game Questionnaire	Post-game Questionniare	Improvement
Hash Functions/Hash Tables	57.1% 28.6%	85.7% 0%	+28.6% -28.6%
Salts	0% 71.4%	71.4% 0%	+71.4% -71.4%
Peppers	0% 85.7%	71.4% 14.3%	+71.4% -71.4%
Dictionary Attack	71.4% 28.6%	100% 0%	+28.6% -28.6%
Brute-force Attack	57.1% 28.6%	100% 0%	+42.9% -28.6%
Combinator Attack	28.6% 57.1%	85.7% 0%	+57.1% -57.1%
Hybrid Attack	0% 85.7%	100% 0%	+100% -85.7%
Combinations/Time taken	14.3% 0%	85.7% 0%	+71.4% +0%

Table 6.1: Table comparing results from both questionnaires

- The top figure for the Pre-game Questionnaire is the ratio of users that answered the question correctly and the bottom figure is the ratio of users that selected the 'Not sure' option.
- The top figure for the Post-game Questionnaire is the ratio of correct answers given by the users and the bottom figure is the ratio of 'Not sure' answers given by the users.

It is important to note that naturally, there will always be an improvement in correct answers being selected because the Post-game Questionnaire is multiple-answer with

no negative marking, hence the dramatic improvement in scores. Nonetheless, the results show a promising increase in correct answers, and a decrease in uncertainty when answering questions - implying that users became more confident answering questions after having played the game.

The questionnaires also asked users to provide examples of a strong and weak password and to justify their choices. The results from both rounds were surprisingly similar, with the majority giving very good examples along with sound justifications. The final question asked the user to rank the presented passwords from least to most secure, which produced some interesting results:

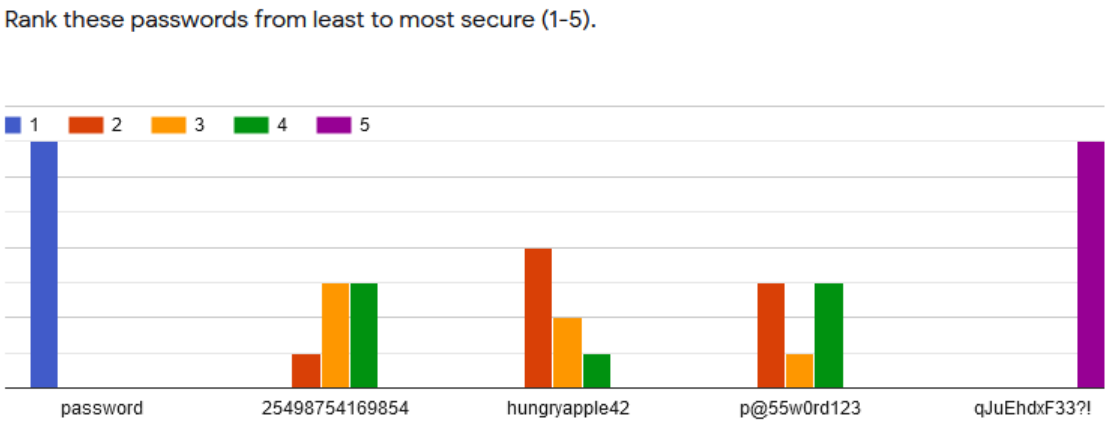


Figure 53: Password ranking Pre-game Questionnaire

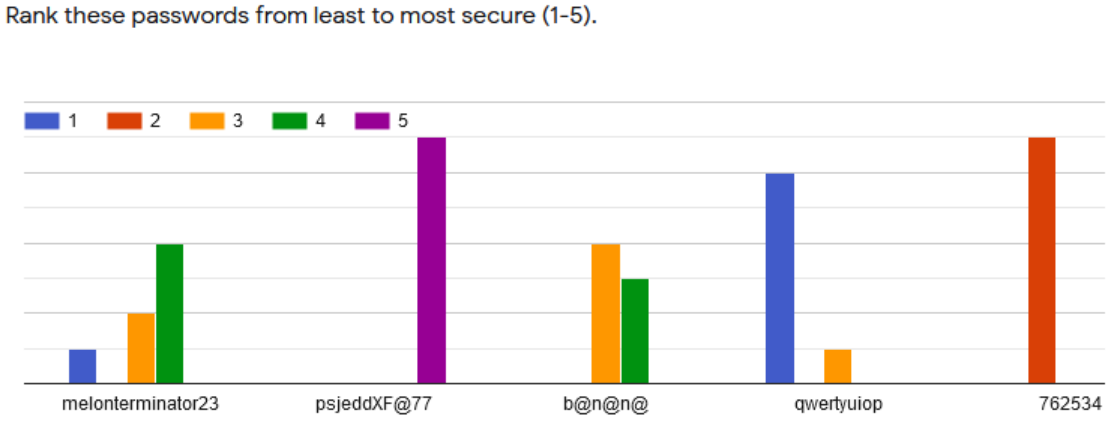


Figure 54: Password ranking Post-game Questionnaire

We can see that Figure 53 shows slightly more ambiguity for users ranking passwords from 2-4, whereas Figure 54 is more evenly distributed with users having similar results.

Overall, the gamified password cracking game made a difference. Evaluation of the Educational Value, Game Application, Gamification Mechanics and Objectives support this claim. The primary metric for determining the project's success has been fulfilled by demonstrating a substantial increase in questionnaire results, thus the aim of the project was met.

6.4 Future Work

There are so many variables to consider when creating a game of any sort. Clearly setting the objectives, planning the user journey, considering additional features on top of providing an engaging experience are just some of the things required to make a successful gaming experience - as a result, there exists new areas of investigation prompted by developments in this project. Moreover, features that were considered but could not be implemented due to time constraints or unforeseeable problems are also described below.

- In light of the current situation, gathering participants was not a hugely successful task. Through friends, family and the University's Volunteer's list no more than 7 participants were found to take part in the research. Had this not been the case, more feedback would have been provided and therefore more improvements suggested by the user from the last section in the Post-game Questionnaire. The current feedback lacks this criticism due to the simple fact that not enough participants were used in the evaluation, despite it being overly positive.

- Integrating the set of questions presented in both questionnaires into the game application was considered mid-way through the development process, but was not achieved due to other higher-priority functionalities. An all-in-one gamification system where the user does not have to leave the application is more convenient than having to navigate to and from online questionnaires.

- Saving user progress was an initial requirement that was actually achieved during development by R/W to a text file that logged the players rank, bitcoin, items purchased, achievements and terminal text colour. This was not achieved due to the nature of the game being an executable JAR file that heavily advises against writing to a text file once an archive. This problem could be overcome by R/W to a separate text file that is not stored within the packaged JAR and referencing it as an external source.

- Implementing a level select, infinite levels or hidden achievements could increase replayability and set more challenging tasks for the player.

- Tight time constraints meant that better programming etiquette could have been achieved. Although implementation demonstrated strong testing coverage, the source code itself could be more efficient in places.

Chapter 7

Conclusions

Originally planned to be aimed towards year 13 pupils, this project has been successful in developing a gamified application that increases password security and awareness in an entertaining way for the average player.

First and foremost, a profound understanding of both password security and gamification techniques would be needed. The Literature Review documented thorough research into these topics as well as a detailed analysis of relevant video games that demonstrate key components for motivation. This acquired knowledge allowed for the development and capture of important objectives for the project goal, which was defined in the proceeding chapter. The Requirements & Analysis Chapter was able to construct a set of detailed requirements, aim and objectives so that features could be designed for the game into manageable steps. The overall theme and design of the game was inspired by the evaluation of existing games in the Literature Review. These game-design elements were then combined with Gamification techniques to create an engaging application that motivates players through reward-based progression, and unlocking achievements. The design section can be seen as a living chapter due to the constant design tweaks and modifications that arose from the feasibility of requirements. The Implementation and Testing chapter put the design into effect, elaborating on the structure of novel algorithms and how testing techniques were conducted. The Results & Discussion Chapter evaluated the primary mandatory requirements, objectives and aim which determined the success of the project. Throughout the Design and Implementation Chapters, other desirable and optional requirements have been addressed that cover all requirements in Section 3.2 - three of which have not been implemented; Allows users to save progress, Utilise Keyword algorithm and Utilise Specifications of System.

In conclusion, the project has achieved the primary aim and objectives to claim it is successful. The project as a whole progressed fluidly with various modifications and technical challenges being handled competently. Undertaking the project has provided the developer with an abundance of valuable experience that will be applied in future projects.

Appendix A

Expanded Information

A.1 Disclaimer

This is an educational simulation of password cracking and does not promote computer misuse in any way. Passwords are generated randomly from legitimate data-sets which may include profanities. When a crack is in progress, a lot of resources will be in use. It is advised to only have this window open and no other processes running unless you have exceptional hardware on your computer system, otherwise you may experience some performance issues. Progress is NOT saved. Closing and re-opening the application will reset your progress. You do not have to complete the game. Play as much as you desire.

A.2 About

This artifact is part of an undergraduate research project created by Theo Koorehpaz that aims to increase password security and awareness by exposing participants to powerful password cracking techniques and counter-measures.

Project title: Password Security Gamification
Lead Researcher: Theo Koorehpaz
Institute: University of Sheffield
Supervisor: Ramsay Taylor
Module code: COM3610
Department: Computer Science

A.3 How to Play

The objective of this game is to learn about password security by cracking various types of passwords with specialised algorithms. The first 19 levels will walk you through how to crack certain passwords as well as explaining in detail how particular counter- measures can be used to mitigate them.

Bitcoin is earned for each completed level, and can be used to purchace essential items from the store that you will need for later levels.

The Password Strength page is an additional feature where you can experiment with different passwords to see how long it would take to crack them and recieve feedback on how to make them more secure.

You can start the game by selecting 'Terminal' from the Menu and requesting a profile from the help page.

A.4 Commands

'help'

- [Up Arrow] ->Grab last command
- [Down Arrow] ->Clear command
- [ENTER] ->Execute command
- syntax ->Display algorithm syntax
- req ->Request a profile / return to profile page
- setCol (COL) ->Set terminal text colour
  - CYAN | RED | GREEN | MAGENTA | YELLOW | WHITE

A.5 Algorithms

'syntax'

- num_brute [range] \ num_brute 9999

- alpha_brute [cType] [cType] [len] \ alpha_brute -l 4
- dictionary [dic] [extn] \ dictionary english -n
- comb_dic [dic1] [dic2] [extn] \ comb_dic fnames snames -n
- hybrid_dic [dic] [range] [extn] \ hybrid english 200 -n

Parameter information:

- cType ->character space = {-l, -u, -n, -s}
- len ->password length = all combinations of specified length
- range ->computes from 0 - specified range
- dic ->dictionary name = {english ,10kmc ,fnames, snames, jobs,hobbies}
- exten ->extension = {-n, -g, -s, -p}
  standard, generate hash table, add salt, add pepper
  -n and -p require additional parameters:
  -s [salt], -p ? = {l, u, n, s}
- -l/l, -u/u, -n/n, -p/p = lowercase, uppercase, numerical, special characters

A.6 Dictionaries

- english ->English dictionary
- fnames ->List of common English first names
- snames ->List of common English surnames
- 10kmc ->List of 10 thousand most common passwords
- jobs ->List of common professions
- hobbies ->List of common hobbies

A.7 Achievements

- Crack your first password
- Complete all brute-force levels
- Complete all dictionary attack levels
- Complete core game
- Create a top-10 common password
- Create a password with a Score of 1
- Create a password with a Score of 2
- Create a password with a Score of 3
- Create a password with at least 60 entropy

A.8 Menu Items

Bitcoin	Item	Description
1.800	num_brute	Numerical brute-force algorithm
2.752	alpha_brute	Alphanumerical brute-force algorithm
2.211	dictionary	Dictionary attack algorithm
3.998	comb_dic	Multi-dictionary attack algorithm
4.566	hybrid_dic	Brute force & Dictionary attack algorithm
1.422	english	English language dictionary
1.101	fnames	English first names dictionary
1.009	snames	English last names dictionary
1.567	10kmc	10-thousand most common passwords
1.691	hobbies	Hobbies dictionary
1.704	jobs	Professions dictionary

Appendix B

Pre-game Questionnaire

B.1 Round 1 Password Security

What is a hash? *

- ○ A non-secret value appended to a password
- ○ A secret value appended to a password
- ○ An algorithm that iterates through a list of words
- ○ An algorithm that iterates through every possible combination of characters
- ○ A seemingly random string of fixed length
- ○ An algorithm that appends a numerical keyspace to a list of words
- ○ An algorithm that combines lists of words
- ○ Not sure

What is a salt? *

- A non-secret value appended to a password
- A secret value appended to a password
- An algorithm that iterates through a list of words
- An algorithm that iterates through every possible combination of characters
- A seemingly random string of fixed length
- An algorithm that appends a numerical keyspace to a list of words
- An algorithm that combines lists of words
- Not sure

What is a pepper? *

- A non-secret value appended to a password
- A secret value appended to a password
- An algorithm that iterates through a list of words
- An algorithm that iterates through every possible combination of characters
- A seemingly random string of fixed length
- An algorithm that appends a numerical keyspace to a list of words
- An algorithm that combines lists of words
- Not sure

What is a dictionary attack? *

- A non-secret value appended to a password
- A secret value appended to a password
- An algorithm that iterates through a list of words
- An algorithm that iterates through every possible combination of characters
- A seemingly random string of fixed length
- An algorithm that appends a numerical keyspace to a list of words
- An algorithm that combines lists of words
- Not sure

What is a brute-force attack? *

- A non-secret value appended to a password
- A secret value appended to a password
- An algorithm that iterates through a list of words
- An algorithm that iterates through every possible combination of characters
- A seemingly random string of fixed length
- An algorithm that appends a numerical keyspace to a list of words
- An algorithm that combines lists of words
- Not sure

What is a combinator attack? *

- ○ A non-secret value appended to a password
- ○ A secret value appended to a password
- ○ An algorithm that iterates through a list of words
- ○ An algorithm that iterates through every possible combination of characters
- ○ A seemingly random string of fixed length
- ○ An algorithm that appends a numerical keyspace to a list of words
- ○ An algorithm that combines lists of words
- ○ Not sure

What is a hybrid attack? *

- ○ A non-secret value appended to a password
- ○ A secret value appended to a password
- ○ An algorithm that iterates through a list of words
- ○ An algorithm that iterates through every possible combination of characters
- ○ A seemingly random string of fixed length
- ○ An algorithm that appends a numerical keyspace to a list of words
- ○ An algorithm that combines lists of words
- ○ Not sure

How many possible combinations are there for a 6 digit alphabetical password? *

- ○ 52^6
- ○ 26^6
- ○ 52 x 6
- ○ 26 x 6
- ○ Not sure

Please provide an example of a 1) Strong password, 2) Weak password and briefly justify your answers. *

Long-answer text

Rank these passwords from least to most secure (1-5). *

	1	2	3	4	5
password	○	○	○	○	○
254987541698...	○	○	○	○	○
hungryapple42	○	○	○	○	○
p@55w0rd123	○	○	○	○	○
qJuEhdxF33?!	○	○	○	○	○

How much do you agree with the following statement: "I struggle to find moti-vation to learn something relating to password security"? *

- ○ Strongly dissagree
- ○ Dissagree
- ○ Neutral
- ○ Agree
- ○ Strongly agree

Appendix C

Post-game Questionnaire

C.1 Password Security Round 2

Which of the following techniques would be best suited for cracking commonly used passwords? *

- [ ] Brute-force attack
- [ ] Dictionary attack
- [ ] Combinator dictionary attack
- [ ] Hybrid dictionary attack
- [ ] Hash tables
- [ ] Not sure

In principle, which of the following techniques cannot be used if the password is salted? *

- [ ] Brute-force attack
- [ ] Dictionary attack
- [ ] Combinator dictionary attack
- [ ] Hybrid dictionary attack
- [ ] Hash tables
- [ ] Not sure

In principle, which of the following techniques cannot be used if the password has been peppered? *

- [ ] Brute-force attack
- [ ] Dictionary attack
- [ ] Combinator dictionary attack
- [ ] Hybrid dictionary attack
- [ ] Hash tables
- [ ] Not sure

Which of the following techniques takes advantage of common password traits? *

- [ ] Brute-force attack
- [ ] Dictionary attack
- [ ] Combinator dictionary attack
- [ ] Hybrid dictionary attack
- [ ] Hash tables
- [ ] Not sure

Which of the following techniques would be best suited for cracking a passcode (numerical password)? *

- [ ] Brute-force attack
- [ ] Dictionary attack
- [ ] Combinator dictionary attack
- [ ] Hybrid dictionary attack
- [ ] Hash tables
- [ ] Not sure

In theory, which of the following techniques is guaranteed to eventually find any password? *

- [ ] Brute-force attack
- [ ] Dictionary attack
- [ ] Combinator dictionary attack
- [ ] Hybrid dictionary attack
- [ ] Hash tables
- [ ] Not sure

Which of the following properties of hash functions makes them most suitable for storing passwords? *

- [ ] Randomness
- [ ] Collision-resistance
- [ ] Speed
- [ ] One-way
- [ ] Not sure

The value of a salt is typically stored with the hashed password *

- ( ) True
- ( ) False

Adding a 1-digit numerical pepper to a password will increase the time it takes to brute-force it by *

- ○ 1-fold (nothing)
- ○ 2-fold
- ○ 10-fold
- ○ 100-fold
- ○ Not sure

Please provide an example of a 1) Strong password, 2) Weak password and briefly justify your answers. *

Long-answer text

Rank these passwords from least to most secure (1-5). *

	1	2	3	4	5
melonterminat...	○	○	○	○	○
psjeddXF@77	○	○	○	○	○
b@n@n@	○	○	○	○	○
qwertyuiop	○	○	○	○	○
762534	○	○	○	○	○

C.2 User Experience

The game exhibited an engaging and hands on experience of password cracking *

- ○ Strongly disagree
- ○ Disagree
- ○ Neutral
- ○ Agree
- ○ Strongly Agree

Progression felt linear as levels started to get more challenging *

- ○ Strongly disagree
- ○ Disagree
- ○ Neutral
- ○ Agree
- ○ Strongly Agree

The game was intuitive and easy to navigate *

- ○ Strongly disagree
- ○ Disagree
- ○ Neutral
- ○ Agree
- ○ Strongly Agree

The game was responsive and had a clean interface *

- Strongly disagree
- Disagree
- Neutral
- Agree
- Strongly Agree

The game has clearly demonstrated the vulnerabilities of weak passwords against various cracking techniques *

- Strongly disagree
- Disagree
- Neutral
- Agree
- Strongly Agree

The game has incentivised me to use strong(er) passwords *

- Strongly disagree
- Disagree
- Neutral
- Agree
- Strongly Agree

I know more about password security now than I did before playing the game *

- ◯ Strongly disagree
- ◯ Disagree
- ◯ Neutral
- ◯ Agree
- ◯ Strongly Agree

I was motivated through the reward system (earning Bitcoin, achievements) *

- ◯ Strongly disagree
- ◯ Disagree
- ◯ Neutral
- ◯ Agree
- ◯ Strongly Agree

I understand the semantics behind a strong password (what makes a password 'strong') *

- ◯ Strongly disagree
- ◯ Disagree
- ◯ Neutral
- ◯ Agree
- ◯ Strongly Agree

I understand how various counter-measures (hashes, salts, peppers) are used to increase the security of passwords *

- Strongly disagree
- Disagree
- Neutral
- Agree
- Strongly Agree

I thought the Password Strength was a useful tool to demonstrate how passwords can be improved (by providing an entropy metric, feedback for weak passwords etc) *

- Strongly disagree
- Disagree
- Neutral
- Agree
- Strongly Agree

Please provide any additional feedback you would like to share, including you your overall experience or any improvements that can be made

Long-answer text

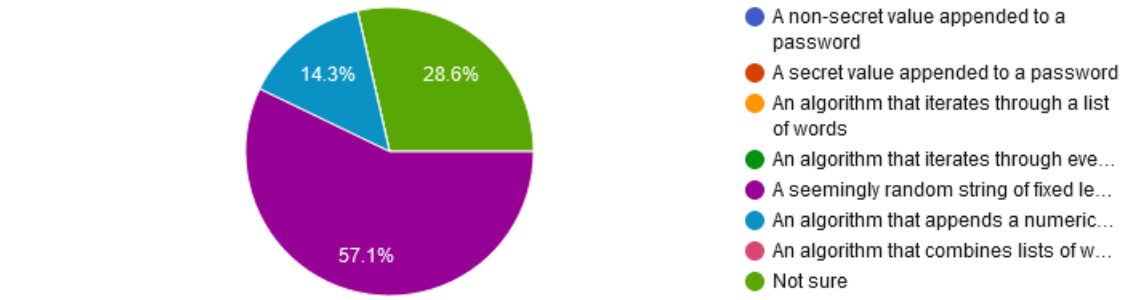
Appendix D

Questionnaire Responses

D.1 Password Security Round 1

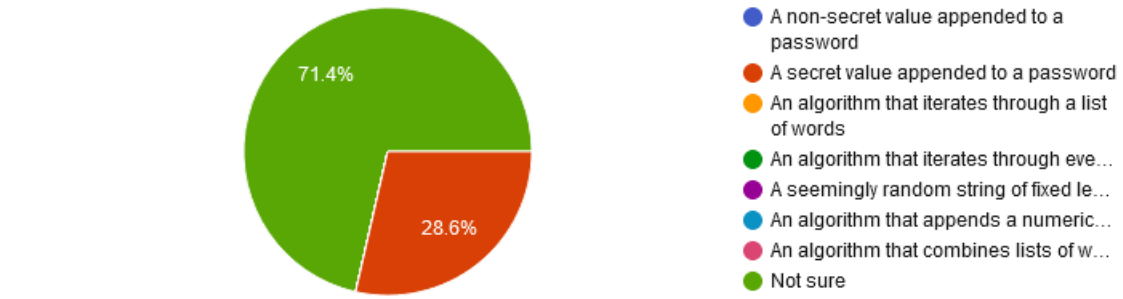
What is a hash?

7 responses



What is a salt?

7 responses



What is a pepper?

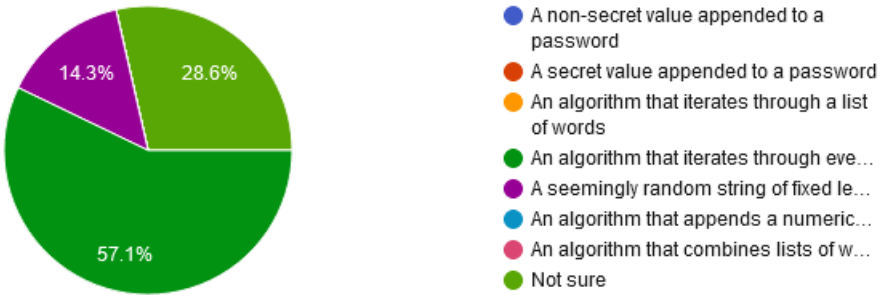
7 responses

What is a dictionary attack?

7 responses

What is a brute-force attack?

7 responses



What is a combinator attack?

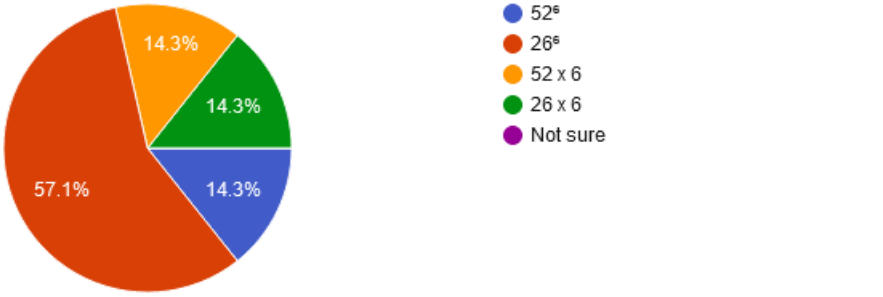
7 responses

What is a hybrid attack?

7 responses

How many possible combinations are there for a 6 digit alphabetical password?

7 responses



Please provide an example of a 1) Strong password, 2) Weak password and briefly justify your answers.

7 responses

1) GaRdEn123 2) garden - Mixture of upper and lower and some numbers

fdsHA123n-14njfd__whj1, CatName1. First password is random, long and made up of no whole words. Password 2 is a frequent structure and easily guessable.

strong password - jeLLy8642 - mixture of lower, upper and numbers, weak password - jelly1234 - just lower case and numbers in numerical order

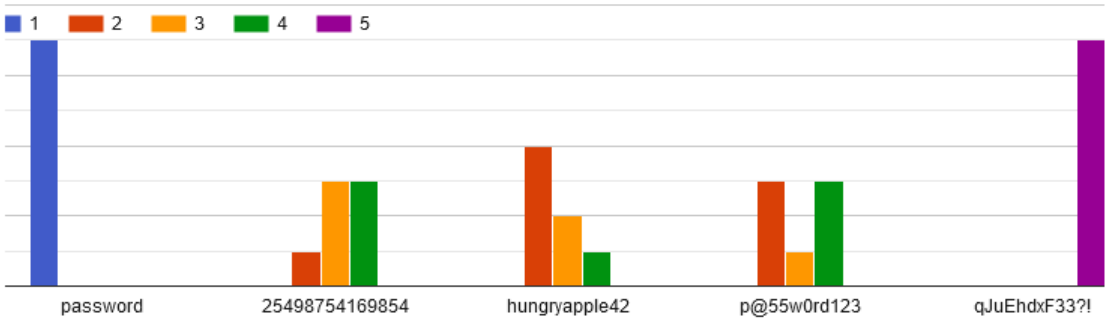
1) obligatorymeaninglesssousaphonegenerator - loads of letters but easy to remember
2) P@55word! - short and obvious

Strong: aJ-9&^Hat2/1n$1- mix of various types of characters, with no pattern between them and long
Weak:password - one word, obvious choice for password, only lower case letters

1) urjsmcOKJSd13 - Random, lower and uppercase with numbers
2) davidgodardd98 - my name, personal info contained - not random

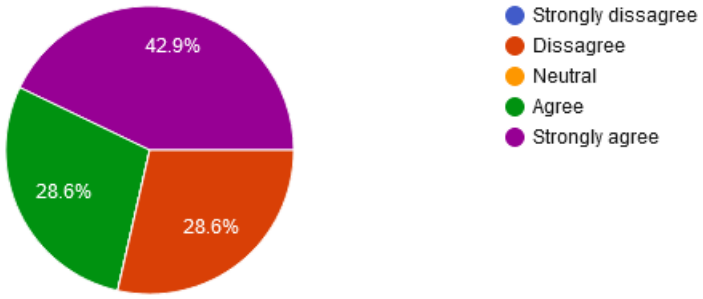
strong - PeanutbutterJELLY3, long password with mix of uppercase lowercase and numbers
weak - password, commonly used password

Rank these passwords from least to most secure (1-5).



How much do you agree with the following statement: "I struggle to find moti-vation to learn something relating to password security"?

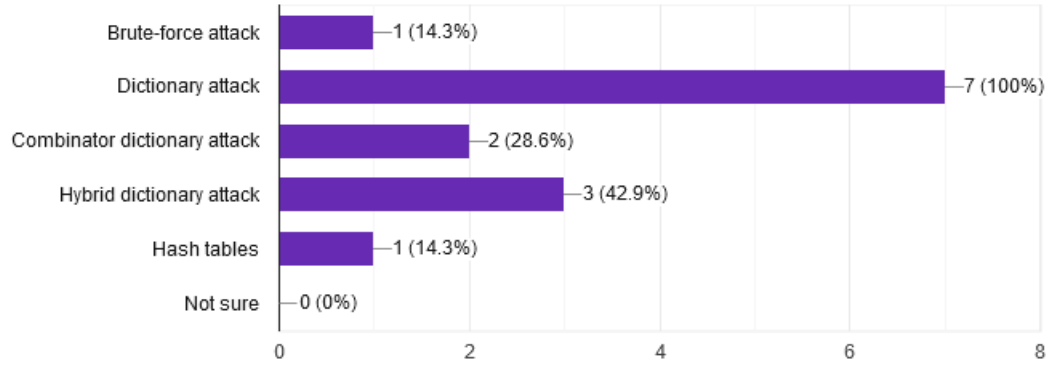
7 responses



D.2 Password Security Round 2

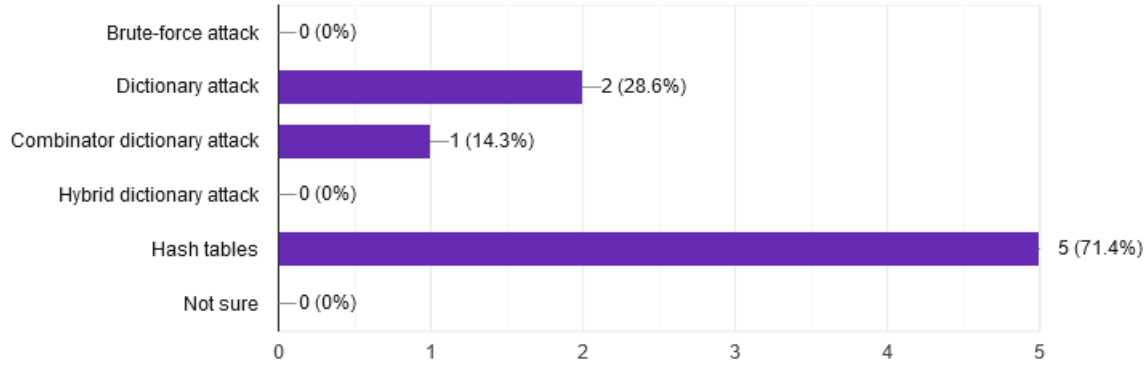
Which of the following techniques would be best suited for cracking commonly used passwords?

7 responses



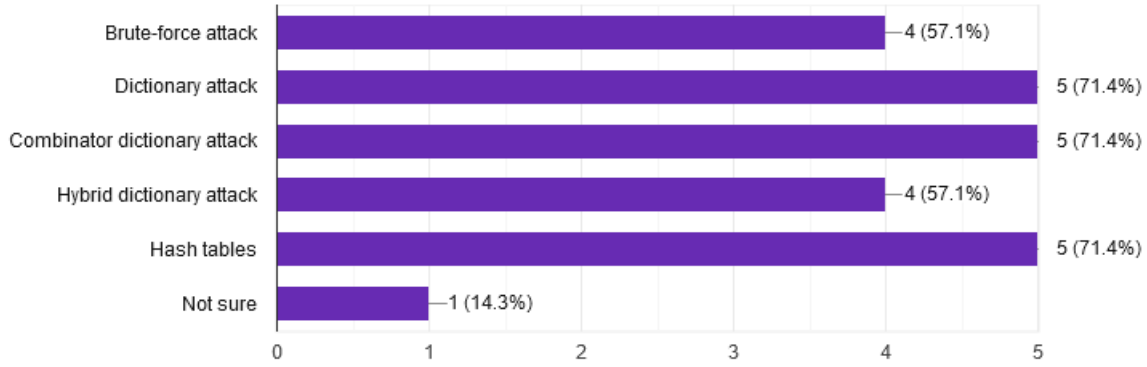
In principle, which of the following techniques cannot be used if the password is salted?

7 responses



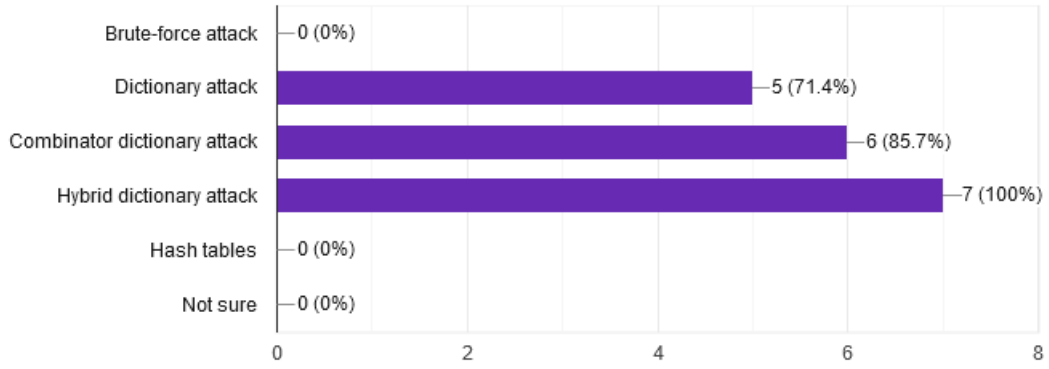
In principle, which of the following techniques cannot be used if the password has been peppered?

7 responses



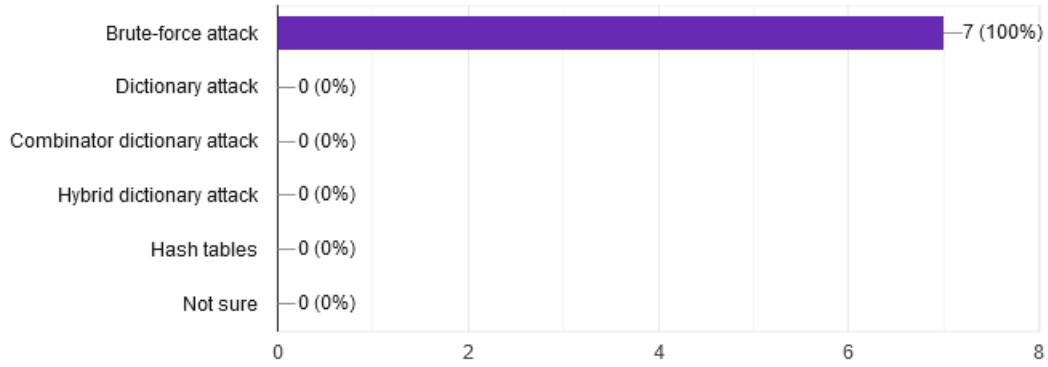
Which of the following techniques takes advantage of common password traits?

7 responses



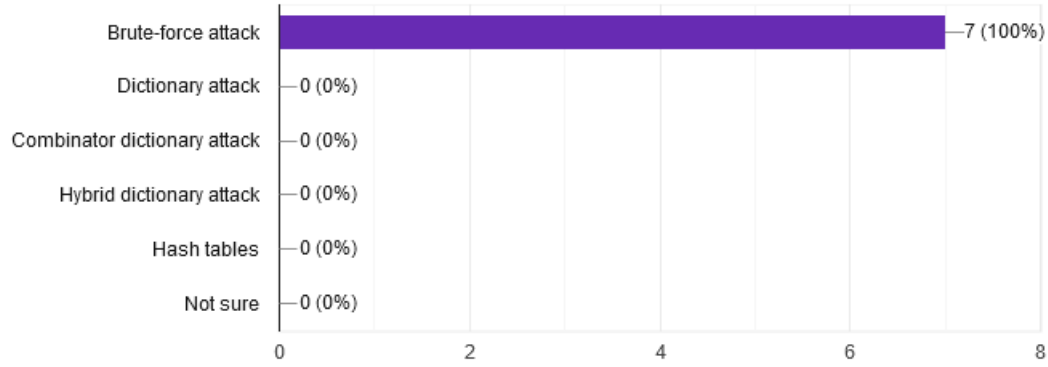
Which of the following techniques would be best suited for cracking a passcode (numerical password)?

7 responses



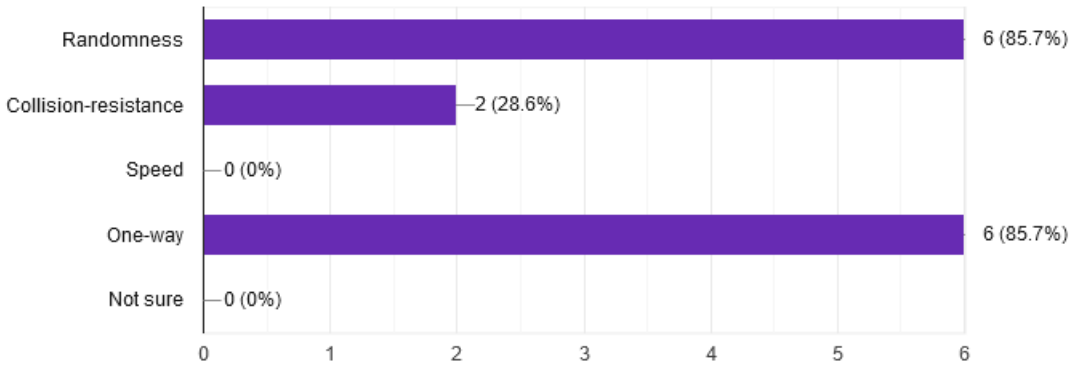
In theory, which of the following techniques is guaranteed to eventually find any password?

7 responses



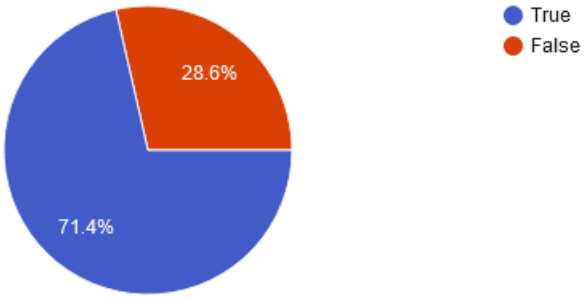
Which of the following properties of hash functions makes them most suitable for storing passwords?

7 responses



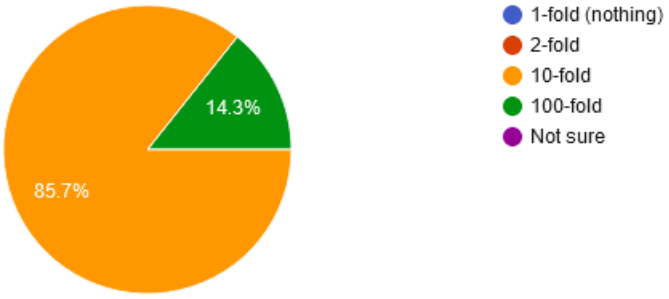
The value of a salt is typically stored with the hashed password

7 responses



Adding a 1-digit numerical pepper to a password will increase the time it takes to brute-force it by

7 responses



Please provide an example of a 1) Strong password, 2) Weak password and briefly justify your answers.

7 responses

@L5ie694 alfie123 - combination of upper, lower, numerical and special

kj190319213~'@~2312 strong password because its random, uncommon, hard to bruteforce and contains no words. Tom1 not secure, easy to break with a dictionary attack.

1) @pP1e5/o0 - combination of special, lower, upper, all mixed up. 2) apples100 - all lower case no special and easy number

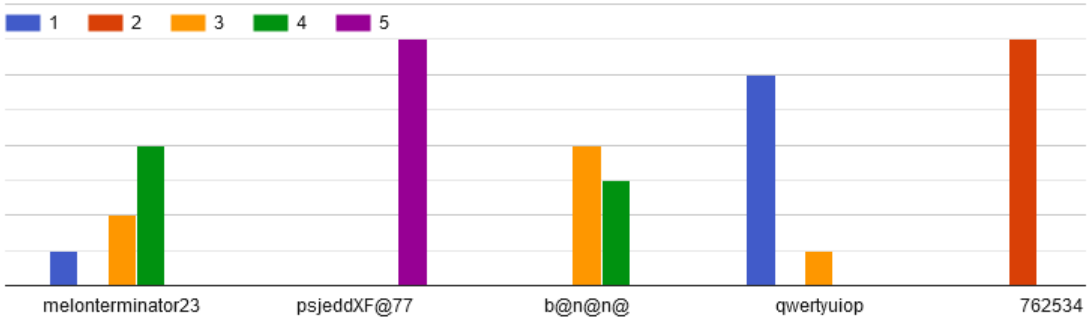
weak - password: only lower case letters(no numbers/symbols), commonly used, only one word,obvious choice for a password strong - aT2*@$gW4Q1j no discernible pattern, mix of characters, no words

1) K@*er&68~)xE - random numbers, letters, uppercase and symbols and no words
2) password - just a simple word

Strong - hJhsdYW?78, mix of uppercase lowercase numbers and special
weak - 234895, easy for brute-force

sP0jHDGfs@?? - mix of character types, hello123 - common word and pattern

Rank these passwords from least to most secure (1-5).



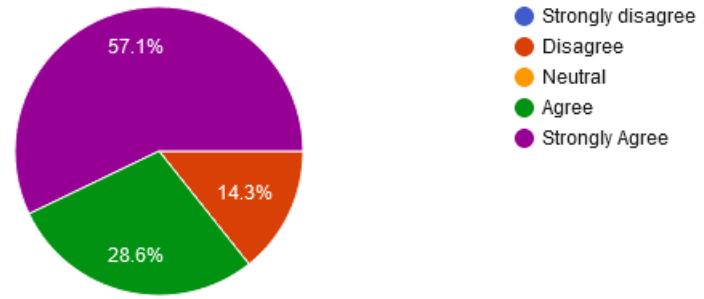
D.3 User Experience

The game exhibited an engaging and hands on experience of password cracking

7 responses

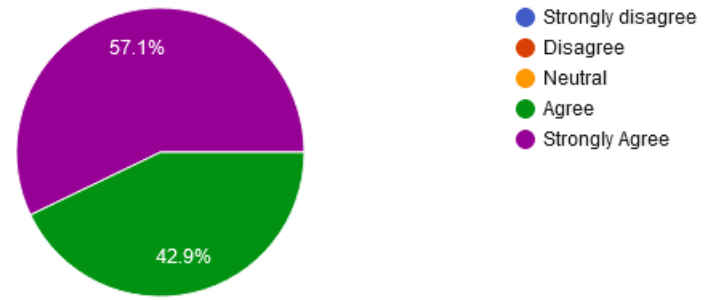
Progression felt linear as levels started to get more challenging

7 responses



The game was intuitive and easy to navigate

7 responses

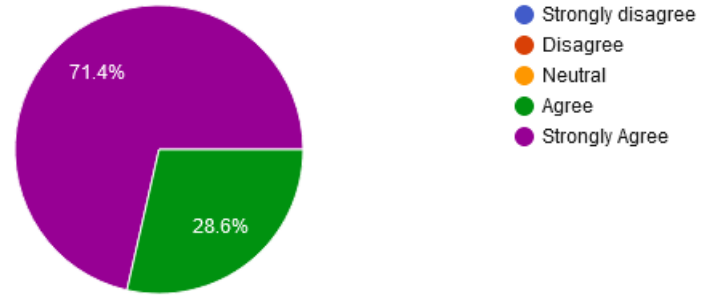


The game was responsive and had a clean interface

7 responses

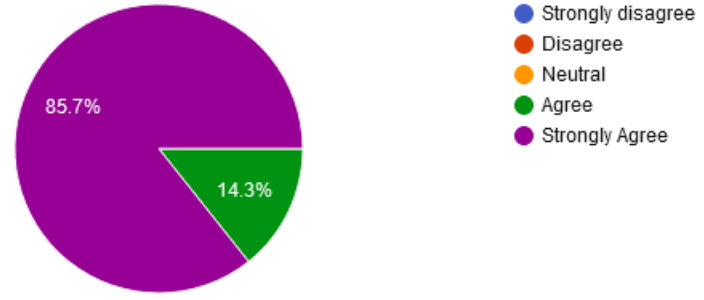
The game has clearly demonstrated the vulnerabilities of weak passwords against various cracking techniques

7 responses



The game has incentivised me to use strong(er) passwords

7 responses

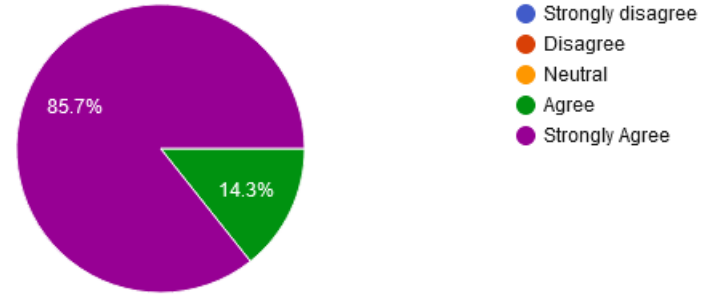


I know more about password security now than I did before playing the game

7 responses

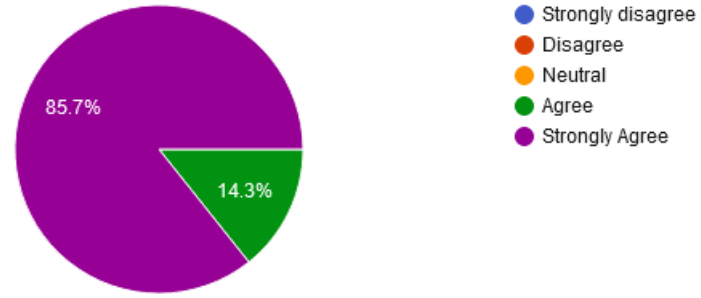
I was motivated through the reward system (earning Bitcoin, achievements)

7 responses



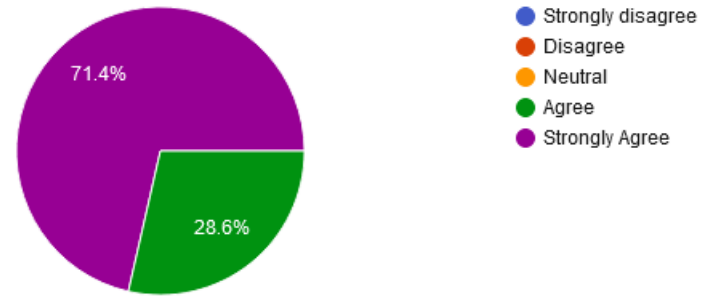
I understand the semantics behind a strong password (what makes a password 'strong')

7 responses



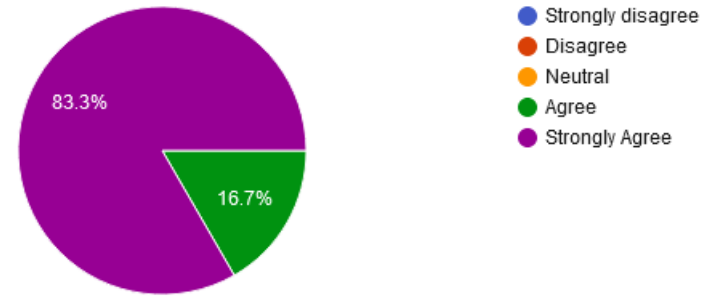
I understand how various counter-measures (hashes, salts, peppers) are used to increase the security of passwords

7 responses



I thought the Password Strength was a useful tool to demonstrate how passwords can be improved (by providing an entropy metric, feedback for weak passwords etc)

6 responses



Please provide any additional feedback you would like to share, including you your overall experience or any improvements that can be made

6 responses

love that you are forced to go to the password testing page in-between some levels. forces you to actively think about being secure as you are getting paid for it.

I thoroughly enjoyed this experience. I really liked the fact that you could build up bitcoins, this made it more fun, and provided a great incentive. My favorite part though would be the password security checker. I think this is a really useful tool!

Very nice game - surprisingly fun and satisfying, good explanations of concepts throughout and though provoking levels. best of all, the fact you could change the colour the terminal!

Very well though out game, loved every second.

I was faily confident with my current knowledge of password security before playing the game but turns out there was so much more to learn! the password strength feature was especially interesting and I might even use it when creating a new password

the bitcoin reward system was a nice touch and somewhat satisfying to gain

References

[1] Natwest, Online Banking. [Online]. Access date: 15 February 2020. Available from: https://www.nwolb.com

[2] USPS. [Online]. Access date: 15 February 2020. Available from: https://www.usps.com/

[3] Ehrenkranz, M. 2018. The 25 Most Popular Passwords of 2018. Gizmodo. [Online]. Access date: 7 December 2019. Available from: https://gizmodo.com/the-25-most-popular-passwords-of-2018-will-make-you-fee-1831052705

[4] Hashcat, Hybrid Attack. [Online]. Access date: 7 December 2019. Available from: https://hashcat.net/wiki/doku.php?id=hybrid_attack

[5] Hashcat, Combinator Attack. [Online]. Access date: 7 December 2019. Available from: https://hashcat.net/wiki/doku.php?id=combinator_attack

[6] Skull Security, Passwords. [Online]. Access date: 7 December 2019. Available from: https://wiki.skullsecurity.org/Passwords[35]

[7] Rouse, M. MD5. TechTarget. [Online]. Access date: 7 December 2019. Available from: https://searchsecurity.techtarget.com/definition/MD5

[8] Walker, S. MD5 Hash Generator. [Online]. Access date: 7 December 2019. Available from: https://www.miraclesalad.com/webtools/md5.php

[9] Dutt, P. Understanding Rainbow Table Attack. GeeksforGeeks. [Online]. Access date: 7 December 2019. Available from: https://www.geeksforgeeks.org/understanding-rainbow-table-attack/

[10] Kauffman, L. About Secure Password Hashing. [Online]. Access date: 7 December 2019. Available from: https://security.blogoverflow.com/2013/09/about-secure-password-hashing/

[11] Habeeb, A. Introduction to Secure Hash Algorithms [Article]

[12] Latinov, L. MD5, SHA-1, SHA-256 and SHA-512 speed performance. Automation Rhapsody. [Online]. Access date 7 December 2019. Available from: https://automationrhapsody.com/md5-sha-1-sha-256-sha-512-speed-performance/

[13] Komargodski, I & Yogev, E. On Distributional Collision Resistant Hashing. [Article]

[14] GeeksforGeeks, Multithreading in Operating System. [Online]. Access date: 16 February 2020. Available from:
https://www.geeksforgeeks.org/multithreading-in-operating-system/

[15] Intel, Hyper-Threading Technology. [Online]. Access date: 16 February 2020. Available from: https://www.intel.co.uk/content/www/uk/en/architecture-and-technology/hyper-threading/hyper-threading-technology.html

[16] Better Buys. Estimating Password-Cracking Times. [Online]. Access date: 7 December 2019. Available from:
https://www.betterbuys.com/estimating-password-cracking-times/

[17] Kamal, P. A Study on the Security of Password HashingBased on GPU Based, Password Cracking usingHigh-Performance Cloud Computing. [Article]

[18] Ferreira, JR. & Oliveira, MC. GPU-Optimized Pulmonary Nodule Retrieval Based on 3D Margin Sharpness Descriptors [Article]

[19] Pelletier, M. Parallel Algorithm for GPU Processing; for use in High Speed Machine Vision Sensing of Cotton Lint Trash [Article]

[20] Sprengers, M. GPU-based Password Cracking. [Article]

[21] John the Ripper benchmarks. 2018. [Online]. Access date: 7 December 2019. Available from:
https://openwall.info/wiki/john/benchmarksJohn-the-Ripper-benchmarks

[22] Moore's Law. [Online] Access date: 7 December 2019. Available from:
http://www.mooreslaw.org/

[23] Bošnjak, L. Brute-force and dictionary attack on hashed real-world passwords. [Article]

[24] Oxford Dictionary. Lexico. [Online]. Access date: 7 December 2019. Available from: https://www.lexico.com/en/definition/gamification

[25] Hamari, J. Koivisto, J. Sarsa, H. Does Gamification Work?—A Literature

Review of Empirical Studies on Gamification. [Article]

[26] Rabah, J. Cassidy, R. Beauchemin, R. Gamification in education: Real benefits or edutainment? [Article]

[27] Fogg, BJ. Fogg Behaviour Model. [Online] Access date: 7 December 2019. Availbale from https://www.behaviormodel.org/

[28] Rules of Play: Game Design Fundamentals Salen and Zimmerman [Book]

[29] Mandl, H. Mayr, S. Hense, JU. Sailer, M. How gamification motivates: An experimental study of the effects of specific game design elements on psychological need satisfaction. [Article]

[30] Tan, RH. PBL. Learn About Gamification. [Online]. Access date: 7 December 2019. Available from:
https://gamification21.wordpress.com/learning-content-2/10-pbl/

[31] Poornikoo M. Gamification: A platform for transitioning from Goods-dominant logic to Service-dominant logic. [Article]

[32] Sips, R-J. Aroyo, L. Welty, C. Dumitrache, A. "Dr. Detective": Combining gamification techniques and crowdsourcing to create a gold standard in medical text [Article]

[33] Lucidchart, Web-based proprietary platform for creating drawrings and charts. [Online]. Access date: 5 May 2020. Available from: https://lucidchart.com

[34] mabe02, Lanterna. Java library for creating text-based GUIs. [Online] Access date: 9 May 2020. Available from: https://github.com/mabe02/lanterna

[35] GoSimpleLLC, nbvcxz. Password Strength Estimator. [Online]. Access date: 9 May 2020. Available from: https://github.com/GoSimpleLLC/nbvcxz

[36] Theo Koorehpaz, Password Security Gamification. [Online]. Access date: 12 May 2020. Available from:
https://github.com/tpazz/Password-Security-Gamification