Investigating the resistance of passwords to an 'everyday' attacker

Alice Henshaw

Word count, excluding figures: 9,998

Abstract

The most common authentication method in modern day computer systems is still text-based username and password pairs [19,28]. This motivates many attacks to be developed against them based on complex mathematical and probabilistic algorithms, which have been developed to perform efficient and accurate password cracking [27,33,54,55]. However, they require both extensive development time, and mathematical knowledge and therefore are not a major threat to the general population (although rapid proliferation of easily acquired weaponised toolkits may transform this threat in the near future) [4,12,18]. While state-of-the-art attack developments are important for theoretical research, little has been done to measure the extent of threat from an 'everyday' attacker.

In this project an 'everyday' attacker and their capabilities are described, and attacks simulated on a real-world database, successfully cracking over 80% of the 14.3 million passwords. This allows us to assess the level of security of user-created passwords. We then combine the consideration of the effectiveness of these attacks with an analysis of the different hash functions used to protect passwords, and also of the password policies implemented by commonly visited websites, allowing us to shed light upon the confusion regarding what constitutes a secure password. From here we provide informed recommendations for both user password-construction, and for set-up of password policies and hash functions in defending systems.

Acknowledgements

Many thanks to my supervisors Jassim Happa, Michael Goldsmith and tutor Thomas Melham for all their help and advice with the completion of my project. Thanks also to Mike Giles and Wesley Armour for helping me with use of the ARC system [2], and allowing me to join their research group so that I could use ARC's GPUs for my research. Finally, thanks to the support team at ARC for helping me set up my account, and looking into any problems I faced with their system exceptionally quickly.

Table of Contents

Table of Figures

1 Introduction

1.1 Introduction to the problem

Text-based usernames and passwords are the most common authentication method in computer systems [19,28]. Individuals use them to protect everything from emails to bank accounts, meaning that the security they provide is of vital importance; however this also provides motivation for attackers wanting to break these passwords in order to access the data or systems they are protecting. This risk is heightened by the fact that users use the same password for an average of 3.9 different online accounts [11]; if one account is compromised, the others may be too.

The secrecy of password databases cannot be assumed; there are many instances of hackers gaining access to them in large systems [7,9,13,14,23,57,58], in fact in some systems they are not really hidden at all [33,40]. Related work has been carried out to look at developing state-of-the-art mathematical algorithms for cracking passwords efficiently [27,33,54,55]. However, with password-cracking tools readily available online [4,12,18,38], an attacker need not be an expert to set about attacking a system, and little or no work has been done to look at the threats imposed by such an 'everyday' attacker (defined in Section 4.1).

There is a trade-off between password memorability and security [52]; if humans were able to remember long strings of random symbols, password cracking would be significantly more difficult. For passwords of length n, there are 95^n different possible passwords (as there are 95 possible characters), which for large n would take hundreds of years to crack. However, this is not the case; not only do humans struggle to memorise strings of random characters, studies have shown that easily memorable passwords are those based on biographical information or simple words [26,52].

In some cases, companies try to ensure the security of passwords by providing users with passwords and not allowing them to create their own. However, a result known as *the generation effect* shows that there is a far greater chance of a password being forgotten if it was not generated by the user themselves, which therefore leads to passwords being written down [56]. In fact 55% of users admit to having written down passwords, with 8% writing down *all* passwords [39].

It is important to educate the population as to what a secure password is to empower them to generate them themselves, thereby increasing both security and memorability. Password policies and measures of strength are put in place by the majority of systems to force users to create passwords that follow a certain set of rules (details given in Section 5.3), however few provide any information as to why these guidelines are valid.

This project defines the capabilities of an 'everyday' attacker and analyses their potential in password-cracking. From this, conclusions can be drawn about the strength of users' passwords, and recommendations made about how to create passwords secure against our 'everyday' attacker. Further analyses are carried out of the effectiveness of different security measures put in place to protect users' passwords, including choice of hash function and password policies, and additional recommendations made for implementation of these in a system.

1.2 Contributions to the field

While thorough experimental research has been carried out looking at how easily passwords can be cracked, many of the methods developed involve complex probability, Markov models, and 'intelligent' algorithms, which can train themselves from a set of passwords to be better at cracking another [27,33,54,55]. While these prove theoretical possibilities, and may be considered a threat against systems of great importance, they are not representative of the threats against everyday systems and websites that the general population use. This project provides detail into how a typical, 'everyday' attack might be formed, and how successful it is against the actual passwords created by users. This therefore can be used to inform users as to how secure their passwords actually are against such an attack.

Furthermore, research has been carried out looking into the effects of different password policies on user password creation, proving that stronger policies do in fact lead to users creating less 'crackable' passwords [19], but policies currently implemented in the world's password-protected systems are not comprehensively considered. In this project, an analysis is provided of the password policies set by the most widely known systems, and insight provided into whether these policies are helping to guide users to create passwords that are secure against 'everyday' attacks.

2 Background

When a user enters a password to authenticate themselves to an entity, the password entered is compared to the password which they have previously created for their account, and access granted if the passwords match. However storing a database of users' passwords in plaintext risks attacks from people trying to gain access to a user's account. Furthermore, for remote access to a system, transmission of passwords as plaintext provides further opportunities for an attacker.

Many systems use cryptographic hash functions to considerably reduce these security risks. When a user sets up a password, the password is hashed, and the hash itself is stored in a database along with the user's identification. When they later wish to be authenticated, they enter a password, the hash of which is sent to the client storing the hash database. If the two hashes match then access is granted to the system, otherwise it is refused.

2.1 Hash functions

The versatility of hash functions due to the large number of properties they have means there is a degree of disagreement on their exact definition. For the remainder of this report, the following definitions will be assumed.

A *hash function* $f\colon \{0,1\}^* \to \{0,1\}^n$ is a deterministic function which takes an arbitrary length input bit string, and outputs a bit string of a fixed and finite length, *n*, known as a *hash*.

2.1.1 Cryptographic hash functions

Cryptographic hash functions are a subset of hash functions, exhibiting additional properties. Before defining a cryptographic hash function, the following terms must be defined:

A problem is said to be *computationally infeasible* if the best known algorithm to solve it requires an unreasonable amount of computational time or resources. More precisely, the worst case time to compute for large enough input will eventually exceed any given polynomial function of the input size [36].

A function $f\colon D \to R$ is *preimage resistant* if, for a non-negligible proportion of possible outputs $y \in R$, it is computationally infeasible to calculate an input $x \in D$ such that $f(x) = y$ [20]. This 'non-negligible' quantification is important – one cannot claim no single input can be found for any output, as an attacker can easily calculate $h(x) = y$ for some x, and immediately know a preimage for $y \in R$. Our quantification instead implies that our function cannot be reverse engineered to find an x such that $h(x) = y$.

A *collision* of a function $f\colon D \to R$ occurs when f yields the same value when applied to two distinct inputs, i.e. $x, x' \in D$, are a collision if and only if $x \neq x'$ and $f(x) = f(x')$ [36]. Infinitely many collisions are guaranteed to exist for every hash function, as they have an infinite domain and a fixed and finite range.

A function $f\colon D \to R$ is *second-preimage resistant* if, for any given input, it is computationally infeasible to calculate a collision with that input, i.e. for any $x \in D$, it is computationally infeasible to find a distinct $x' \neq x \in D$ such that $f(x) = f(x')$ [36].

A function $f: D \rightarrow R$ is *collision resistant* if it is computationally infeasible to calculate any collisions for f, i.e. it is computationally infeasible to find a pair $x, x' \in D$ such that $f(x) = f(x')$ [36].

A *cryptographic hash function* $chf: D \rightarrow R$ is a hash function additionally exhibiting preimage resistance, and some protection against collisions. This could come in the form of either second-preimage resistance, or full collision resistance. These properties together mean that a cryptographic hash function has good diffusion – each input has a seemingly random hash, and a small change in the input bit string causes a large change in this hash.

2.1.2 Salted hash functions

A *salted* hash function $shf: \{0,1\}^{s+*} \rightarrow \{0,1\}^n$ also has an argument called a *salt*. The salt is a bit string of fixed length, s, which is appended or prepended to x before the hash function is applied.

When a user enters their password, to ensure the correct password yields the correct hash, the salt used must always be identical. For this reason, the salt must be stored along with $h(x||s)$ in the database, and therefore cannot be assumed to be secret. Although an attacker with access to the hashes has access to the salts, they can have numerous advantages when it comes to protecting attacks. for example, if a different salt is used for each user, two users with the same password will have different hashes, thereby limiting the knowledge an attacker can glean about passwords from just the hashes. Further advantages in Section 2.2.

2.1.3 Key-stretched hash functions

Key-stretched hash functions are another method by which some hash functions are strengthened against some forms of attack. Here the hash function is iterated n times, usually for a large n. This makes it approximately n times slower to calculate the hash for any given input, thereby making it take longer to authenticate a user. However, this also slows an attacker calculating hashes in attacks against the system. This is discussed further in Section 2.2.

2.2 Methods of attack

Cryptographic hash functions are often used in systems to protect important and secret data, passwords being just one example of this. A lot of time and effort has therefore been put into developing attacks against hash functions, by both attackers hoping to crack them and by researchers verifying the extent of their security.

For this project, it is assumed that the hash functions in question are fully preimage resistant. This means that attempts will not be made to reverse or exploit flaws in hash functions to find passwords, but instead attacks against preimage resistant hash functions are explored. Attacks against such hash functions require the hash of different possible plaintexts to be calculated in turn, and each compared with the hash(es) hoping to be cracked. Such attacks are limited by the speed at which hashes can be calculated, meaning that they can be slowed by a time-intensive hash function, like a key-stretched hash function.

2.2.1 Brute-force attacks

A brute-force attack hashes every single possible input over a given character set – no possible strings are missed. These attacks tend to start with all strings of length 1, incrementing this until either a string with the desired hash is found, or all passwords up to a certain length have been tried.

By Kerckhoff's Principle[1], in the case of salted hash functions, we must assume the attacker not only knows the hash function, but also the salt stored in the database as well. The attacker can therefore append it to each guessed string before hashing, meaning the number of guesses needed to crack a single hash is not increased. However, one level of security that using salts introduces is that, when trying to crack say 5 hashes at once, if the system has used a different salt for each hash then each must be attacked separately.

Although a brute-force attack is guaranteed to crack every possible password, the time required to hash such a large set of values makes this a time-consuming task. Instead, other methods of attack can be used, which reduce this set of values to be hashed.

2.2.2 Dictionary attacks

When brute-forcing passwords, strings such as `a^W` or `&q0` may be tried before common English words such as `cat` or `dog`, and all these 3 letter strings will be tried long before `password`. Instead, a dictionary attack can be performed. In a dictionary attack, a list of words called a dictionary is compiled by the attacker, and each word is hashed and compared to the hash to be cracked. This dictionary could be a list of all the English words, or a list of commonly known passwords (including for example, `123456` and `qwerty`).

Dictionary attacks are like brute-force attacks, in that they are limited by the speed at which the attacker can calculate the hashes, and so salted and key-stretched hash functions have similar effects in slowing an attacker.

2.2.3 Other attack methods

Many other methods of attack against hashed passwords exist, including creation of lookup tables storing plaintext and hash pairs for a set of plaintexts, or rainbow tables that balance the time taken to carry out a brute-force attack, with the space required to store a lookup table. However these attack methods have been researched in detail [43,50,58], and will not be discussed in this project.

2.3 Hardware for attacks

When using a brute-force attack against a hash, where the corresponding password could be up to length 3, and calculation of up to $95 + 95^2 + 95^3$ hashes is necessary, totalling 866,495 passwords to hash and compare. If that length is then increased to *8*, there are *6.7 thousand trillion* passwords to hash. However, the calculation of so many hashes is easily parallelisable by calculating the hashes for multiple different strings at once. Computers have two key forms of processor that can be used to carry out such parallel operations, CPUs and GPUs.

[1] Kerckoff's Principle states, that a cryptographic system should be secure even if everything about the system, except the private key, is public knowledge. [6]

The CPU of a computer is responsible for executing sequences instructions, and allows parallelisation through the use of multiple cores, each of which can execute a sequence of instructions independently of the others. Some of the less modern CPUs have just two cores (e.g. Intel Core i3), but it is common for more modern CPUs to have four or more cores (e.g. Intel Core i5/i7) [15]. This allows the computer to execute four instructions at a time, reducing *6.7 thousand trillion* to *1.7 thousand trillion* passwords for each core to hash.

Computers moreover have another processing unit built in; the GPU. These were created to perform mathematical calculations involved with a computer's display. The sheer volume of calculations involved in graphic calculations lead to GPUs having a large number of cores, which can be exploited by giving them computations unrelated to the graphical display.

The number of cores in a GPU varies greatly – a standard laptop may have an integrated GPU built into its motherboard (for example, the Intel Core i5-5200U has an integrated GPU with 24 cores), or may have a distinct, more powerful GPU (for example, the Toshiba Satellite L50 has a GPU with 384 cores) [15]. Dedicated GPUs have been built for the purpose of carrying out highly-parallelisable general-purpose GPU programs (GPGPU), for example the NVIDIA Tesla K80 has 4992 cores [34], which reduces our *6.7 thousand trillion* passwords to be hashed to just *1.34 trillion* per core.

An analysis of the speeds at which these different hardware devices can carry out attacks can be found in Section 4.4.

3 Related work

A number of groups have studied the effects of using probability to predict and generate different possibilities of passwords [27,33,54,55]. One method of doing this is using Markov models, which use the structure of natural language to assign probabilities to different combinations of letters appearing consecutively, and use this to generate the more probable strings of characters. In 2005, Narayanan and Shmatikov [33] paired this technique with a finite automaton to further eliminate strings of lower probability, and found the technique achieved similar cracking percentages to previous methods but using a smaller search space (and therefore faster running time).

In 2009, Weir et al. [55] published a new style of password attack - the Probabilistic Context-Free Grammar (CFG). This also uses probability to generate higher-probability strings using knowledge of common password structures (e.g. six letters followed by two numbers). These templates are then filled in with the most common substrings. They found their approach cracked 28% to 129% more passwords than cracking software John the Ripper [18].

As recently as 2016, a further development of the use of probability was published by Melicher et al. [28]. They developed a neural network for password guessing – a memory-efficient machine learning technique that guesses which character is likely to follow after a certain string of characters. It was shown to regularly crack more passwords than both Markov models and CFGs.

Kelley et al. [19] researched the effects of different password policies on the complexity of passwords created by users, and on the memorability of these passwords. They found that passwords created under stricter policies are less likely to be cracked, however they more notably found that the length of password enforced is more important than 'randomness' against an attacker. In a second paper (2011), they analyse the memorability of the passwords created under different policies, and discovered that policies enforcing passwords to be more 'random' lead to users writing them down or forgetting them [22].

These models all have a complex theoretical basis, using probability and machine learning to create algorithms that eliminate strings of low probability to increase efficiency of the algorithm. They provide useful evidence for what an attacker can theoretically achieve, but require a large amount of research and time to develop. These methods therefore are likely not to be used by attackers unless they want to gain access to highly-valuable information. Little work exists on detailing the severity of threat from an 'everyday' attack – an attack that could be carried out with considerably less knowledge and time.

Furthermore, the results gathered in these papers are not used to further inform how a password should be created to best secure any information protected under its protection. With passwords remaining the most common computerised-authentication method, users need to be provided with clear information detailing what constitutes vulnerability of passwords, and how to construct passwords resilient against potential attacks.

This project is a study to provide additional insight into resistance to password cracking. It further provides recommendations for construction of secure passwords, and for the set-up of systems to provide maximum possible security to its users' passwords.

4 Methodology

4.1 The 'everyday' attacker

Due to the reasons outlined in Section 3, this project looks at simulating a systematic 'everyday' attacker, who poses a very real threat against the world's computer systems, and the general population. More specifically, we model an attacker who:

- has obtained access to a database of hashed passwords –from a leak online, or by hacking a system himself
- need not have *detailed* knowledge about the different styles of attack, but has read information available online, and is logical and systematic in his approach
- has enough knowledge of computers and programming to download a package, install the necessary drivers and carry out the (sometimes complex) commands on the command line
- has money with which to purchase efficient hardware on which to carry out his attack - hardware costing approximately £15,000 (as of 01/07/2017) in our model.

4.2 The passwords

To model an 'everyday' attack on passwords, a set of passwords to hash and attack is necessary. Due to the secrecy of passwords and the purpose they serve, there is a distinct lack of information known about their structure. Creating a sample of passwords that accurately represents the passwords chosen by the population is therefore exceedingly difficult.

As discussed in Section 1.1, users have a tendency not to generate random passwords, and instead base their passwords on biographical information and simple words to help with memorability [26]. This, and other available information about password habits (discussed in Section 4.6.2), is helpful for structuring an attack against a set of passwords. However, it does not provide enough detail to inform construction of a password set that accurately represents those created by the population. Instead passwords must be collected from the population.

There are 2 main methods of collecting a set of passwords representative of the population's passwords. The first of these uses a survey to collect opinions from individuals about their personal password habits [19,22]. However, an accurate sample is difficult to achieve as actual passwords will not be revealed. Alternatively, password databases are sometimes hacked, and the data publicised [7,9,13,14,57,58]. Often such databases are hashed and so the passwords themselves are not known and so the hashes must be attacked, and only the recovered passwords known. However some systems do not have password hashing implemented and all passwords are accessible; these are the most accurate representation of the population's passwords that's available.

In 2009 the password database for RockYou.com was successfully hacked, and the passwords were found to be stored in plaintext. This database included over 32 million passwords, 14,343,644 of which were unique [42]; these are the passwords chosen for use in our model of attack. For the purpose of these tests, any replicated passwords have been removed from the list.

4.2.1 Ethical considerations

Although many researchers have used the leaked passwords of the RockYou hack in their works [21,28,54], there are undoubtedly ethical considerations related to using information acquired in an illegal fashion. This consideration is heightened by the knowledge that the average user reuses the same password for 3.9 accounts [11,21], thereby meaning if their username and password combination is discovered from one attack, the same password is likely to be used for another system by that user.

The data acquired for this project is already in the public domain, copies of the list can easily be accessed online by anyone, and complex analyses of the passwords are also readily available [23]. This thereby means that our use of the passwords does not increase the hurt caused to the victims of the attack, and further does not increase any knowledge of the passwords to the wider community.

However in consideration of the users affected, permission was sought and granted from CUREC for use of the passwords. Throughout the project, all information relating each password to its user is not considered, any copies of the passwords remain encrypted or hashed at all times, and no information about individual passwords is revealed.

4.2.2 Reliability

The RockYou password set was acquired illegally by a hacker, and published online for anyone to see. This calls into question the reliability of the passwords, as proof that the list downloaded is in fact the exact, unaltered set of passwords from RockYou is hard to come by. However the statistics in analyses of this set have shown it to possess very similar properties to other sets of passwords leaked from other companies, showing it to be representative of generic user passwords [14,23,54]. Furthermore, this exact list has been used in many previous research studies, as training sets for intelligent algorithms, or for testing the success of a state-of-the-art password cracking technique, and has shown successful results [21,54].

4.3 Choice of hash function

A large number of hash functions exist, for many different purposes throughout computing, including protecting the integrity of messages. However, we are only interested in those commonly used for password hashing, so as to create an 'everyday' attack model. Many companies do not publicise the security in place to protect the passwords used in their systems. This could be due to there being a lack of security in place, or an attempt to keep it secret from possible attackers. However information about the following large systems is known:

- Microsoft used the LM hash function to hash passwords in a huge number of their Windows systems [31]. This was later proven to be insecure and replaced with the NTLM hash, which was also proven to be insecure. Microsoft also use versions of SHA and MD5 within their systems, though not for the hashing of passwords [32].
- Apple are known instead to use a large variety of different hash functions through their products, including SHA-1, SHA-256 and SHA-512 [3]. For password hashing on Macs, different hash functions are used in different versions, these again include SHA-1, SHA-256 and NTLM [35].

- UNIX-based operating systems support a variety of different hash functions for password hashing, allowing the user to choose their preferred function. These include MD5, Blowfish, SHA-256 and SHA-512 [37,54].

PBKDF2 is an iterated key-derivation function, aiming to make keys less vulnerable to brute-force attacks [16]. PBKDF2 allows you to specify the function you would like to iterate, and the number, *n*, of iterations, making it possible to use with a hash function to create a key-stretched hash function. The specification of your own function allows PBKDF2 to be used with functions that are trusted to be secure (such as SHA256), to build a system even more resilient against attacks. A number of different systems utilise PBKDF2 with different numbers of iterations for a variety of purposes; these include Kerberos (4,096), iOS3 (2,000), iOS4 (10,000), blackberry (1) and Last Pass (100,000) [10,17,24]. However no systems currently appear to be using PBKDF2 to hash passwords.

Clearly many different hash functions are used for the purpose of password hashing. In our model we will be using the SHA-256 hash function, however a comparison of the security of different hash functions against our attack is available in Section 5.2.

4.4 Hardware

In order to decide which hardware an attacker would be best to purchase, an analysis of the speeds of an everyday laptop CPU and GPU was carried out. This was followed by a comparison with a purpose-built GPU to compare their performance when attacking a set of hashes. The results follow.

The laptop used was a Dell Latitude E5550, with the following specification [15]:

- **CPU:** Intel Core i5-5300U (2 cores, 4 threads, maximum 2.9GHz)
- **GPU:** Intel HD graphics 5500 (24 cores, maximum 950MHz)

This is an integrated GPU, meaning it is found on the motherboard, and is built purely for processing the simple graphics for a laptop. The purpose-built GPU chosen was a NVIDIA Tesla K80, with the following specification [34]:

- **GPU:** Tesla K80 (4096 cores, base clock 560MHz, booster 875MHz)

70 tests were carried out on each of the first two devices, consisting of 10 repetitions of 7 different brute-force attacks (on passwords of lengths 2 through to 8). For each test the number of hashes the processor could calculate per second was recorded. The results of these tests can be seen in Figure 4.4a. The table of values corresponding to the graph can be found in the Appendix, Section 11.1

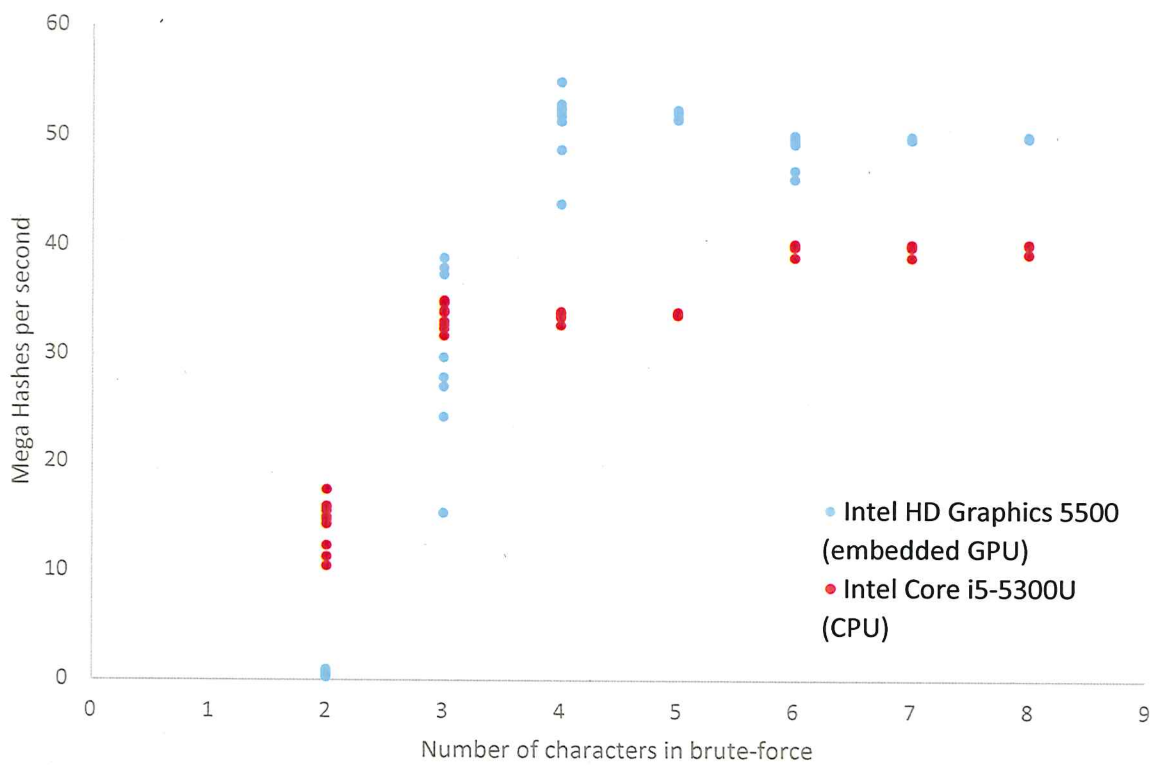


FIGURE 4.4a: Graph of CPU vs. GPU hashing rates.

The number of hashes to calculate in each test is equal to 95^x, where x is the number of characters in the possible passwords. When x is small, the two processors do not reach their full capacity, as with fewer hashes the program is less parallelisable, and so fewer are calculated per second. However, as x increases it can be seen that both the CPU and GPU trend towards their capacity.

The results show that, at their full capacity, the GPU can calculate a larger number of hashes per second than the CPU. This is because, despite the fact that the GPU has a far slower clock speed in this particular device, its larger number of cores benefits it greatly in such a parallelisable computation.

Undoubtedly such a GPU is created to process the graphics for a laptop, and therefore only has the maximum specification required for this simple process. A comparison of this integrated GPU with a NVIDIA Tesla K80 built specifically for highly parallelisable general purpose GPU programming was therefore subsequently carried out. The 70 results previously calculated using the Intel GPU were used, and the same tests carried out on the K80.

The drastically different results can be seen in Figure 4.4b, with the table of results found in the Appendix, Section 11.2. These results clearly show that the purpose-built K80 is capable of calculating nearly 16 times as many hashes per second, meaning it would be far more successful in a password-cracking scenario, as each attack chosen could be carried out nearly 16 times faster.

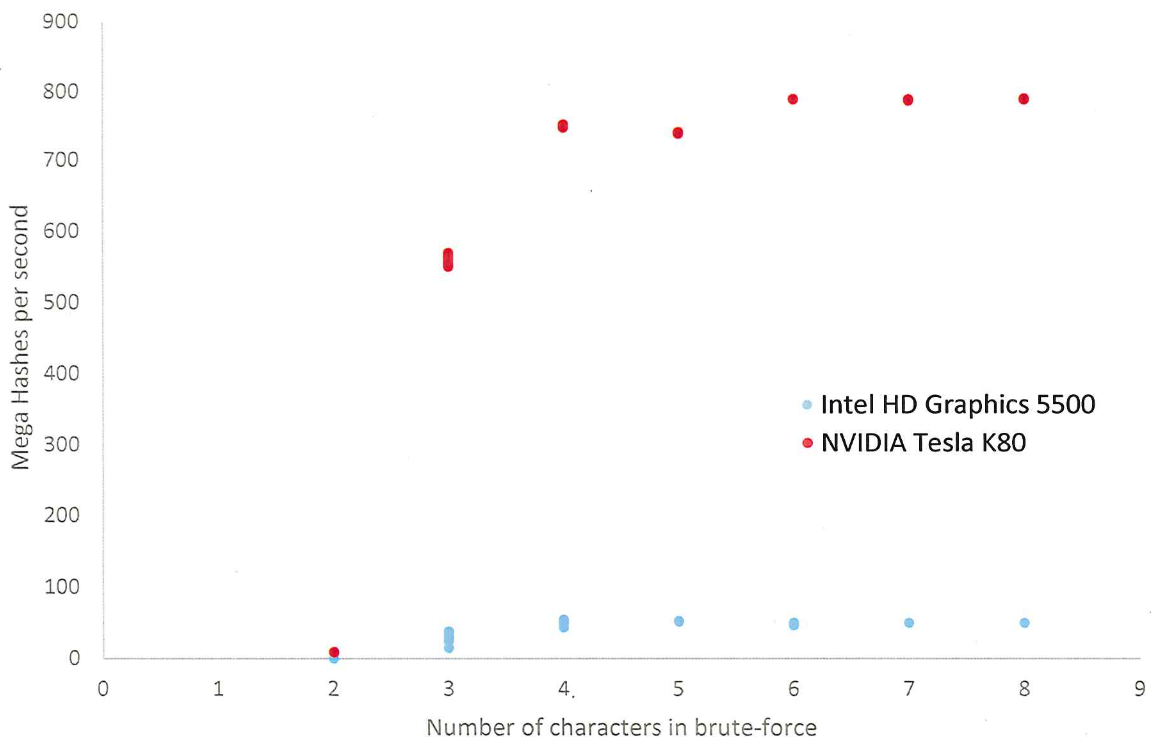


FIGURE 4.4b: Graph of Intel HD Graphics 5500 vs. NVIDIA Tesla K80 hashing rates

4.5 Software

Many hash-cracking programs and websites are available online for free. These include downloadable programs such as John the Ripper, Hashcat, and Cain and Abel [4,12,18], or websites employing lookup or rainbow tables [25,38]. This means that to set about cracking a hashed password, knowledge of hash functions is not necessary, let alone any expert programming knowledge or experience.

For the purposes of this project, we make use of the freely downloadable software, Hashcat [12]. Hashcat is compatible with over 190 different hash functions and has versions available for both Linux and Windows operating systems, and for any AMD, Intel or NVIDIA GPU, making it highly versatile for anyone to use. It has further been used by many previous researchers in their papers, has extensive documentation, an online advice forum, and many step by step tutorials online [12,28]. Together these attributes make Hashcat an accessible piece of software and suitable for use by our 'everyday' attacker.

4.6 Attack method

To effectively use Hashcat to attack a set of hashes, the different incorporated attack modes need to be explored to discover what the software is capable of. These modes are described in Section 4.6.1, with examples given in Figure 4.6.1a, and screenshots provided in the Appendix, Section 11.3. Furthermore, knowledge of the trends in how users construct their passwords is important to ensure the attacks carried out are targeted and time is not wasted hashing unlikely strings.

4.6.1 Hashcat attack modes

The first attack mode, known as 'straight' attack mode in Hashcat, implements a dictionary attack. The user provides a dictionary of their choosing, and every word in the provided dictionary is hashed and checked against the provided file of hashes. A more detailed and knowledgeable use of Hashcat allows users to further provide rules to be applied to each dictionary word in turn before the attack is carried out.

The second provided attack mode, known as 'combination' attack mode combines dictionary words in a most sophisticated version of the dictionary attack. Users provide 2 dictionary files, possibly the same file, and for each x in Dict_1 and y in Dict_2, the software hashes xy and compares this to the provided file of hashes.

The next attack mode is a brute-force attack. This mode allows the brute-force attack described in Section 2.2.1 to be carried out, however it also allows a more sophisticated brute-force to be carried out over a particular *mask* of characters. For example, all possible strings consisting of one uppercase letter followed by three lowercase letters could be brute-forced. This reduced character-set brute-force attack can dramatically reduce the search space: a brute-force of *all* four character strings must hash 81.45 million strings, compared to less than 457,000 for our example mask.

The final two possible styles of attack are a hybrid attack that combine a dictionary attack, with a masked brute-force attack. Users can either append or prepend a mask to every word in the provided dictionary, as shown in Figure 4.6.1a.

Mode	Dictionaries	Mask / Rule	Passwords generated
'Straight' dictionary	`three, example, words`		`three, example, words`
Dictionary + rule	`three, example, words`	Rule: `c; se3` (capitalise, then substitute 3 for e)	`Thr33, Exampl3, Words`
Combination	1: `blue, red` 2: `cat, dog`		`bluecat, bluedog, redcat, reddog`
Masked brute-force		Mask: `?l?l?d` (2 lowercase then 1 digit)	`aa0, aa1, …, aa9, ab0, …, zz8, zz9`
Hybrid: dictionary + mask	cat, dog	Mask: ?d (1 digit)	cat0, …, cat9, dog0, …, dog9
Hybrid: mask + dictionary	cat, dog	Mask: ?s (1 symbol)	!cat, "cat, #cat, …, !dog, "dog, #dog, …

FIGURE 4.6.1a: Example usage of each Hashcat attack mode

Although it is not possible to apply rules in the hybrid or combination modes, a program can be written to apply the rule to the dictionary. This new dictionary is then piped into the hybrid or combination attacks, to produce new untried password attempts. The details of the attack carried out are provided with the results in Section 5.1, with further detail provided in the Appendix, Section 11.4.

4.6.2 Password construction

The attack modes of Hashcat gives a good basis of an attack. However, to make the attack more efficient and increase the chance of success, investigation into the common structures of passwords allows attacks to be targeted towards the most probable password choices.

Vu et al. (2007) gave an analysis of user password construction habits and showed that the majority of users tend to base their password on biographical information or on a simple word [26,52]. This information allows us to utilise the dictionary attack from Hashcat to test whether each word was chosen - a discussion of dictionary choice can be found in Section 4.6.3. A second analysis, by Weir et al. (2010), looked at the most common alterations of dictionary words, finding that four of the five most common alterations were adding 1, 2, 3 or 4 digits to the end of the word, the other being leaving the word unaltered [54]. Other common rules included capitalising the word and appending digits, replacing `'i'` with `'1'`, replacing `'o'` with `'0'` and replacing `'a'` with `'@'`. The full list of common alterations is available in [54].

From these analyses, it can be seen that Hashcat's dictionary attack is likely to be effective, both with and without substitution and case altering rules. The hybrid attack of appending masks can also be used to create many of the other common word-alterations found, e.g. appending one digit. A number of the common alterations include both applying a rule and appending a mask to the dictionary word. While Hashcat does not permit a rule to be applied to the word during a hybrid attack, a new dictionary can be created containing words transformed by the rule, and this new dictionary used in the hybrid attack.

Once dictionary attacks have been exhausted, brute-force attacks can be used to try to catch the passwords that were cracked. Using a mask instead of a full brute-force attack is a useful tool to speed up the attack against a password set by decreasing the size of the search space, and therefore decreasing the time taken to carry out each attack. However this attack is only efficient if masks are chosen effectively.

For this stage of the attack model, masks are chosen using information gathered from a number of different password analyses of other leaks of password dictionaries. These analyses are of leaks of passwords from Sony, MySpace, phpBB, Hotmail and Gawker [7,9,14,54]; two analyses of passwords from different Finnish and Italian websites are also used. A summary of these results can be seen in Figure 4.6.2a.

	Sony	MySpace	phpBB	Hotmail	Gawker	Finnish 1	Finnish 2	Italian
Average Length		8.3	7.3	8.8		7.6	7.6	7.9
Only lowercase	45.0%	7.3%	50.6%	43.0%	61.0%	51.6%	53.1%	51.2%
Only lower and digit		84.1%		87.0%		92.8%		
Contain a symbol	1.0%	9.5%	1.4%	5.2%	0.7%	1.8%	4.6%	6.6%
Contain an uppercase		6.9%	7.2%	8.1%		3.0%		

FIGURE 4.6.2a: Summary of results from eight password dump analyses

Unfortunately, without carrying out an analysis ourselves of the individual password sets, not all information in the table could be gathered. However from the information gathered from these eight password analyses, some clear trends can be seen and used for attacking the RockYou password set:

- For all password sets except MySpace, 43-61% of passwords only consisted of lowercase letters. The anomaly of the MySpace set (7.3%) is due to the fact that MySpace had password policy enforcing no passwords consisted of only lowercase letters. The passwords without this requirement could show a bug in the policy enforcement, or passwords created before the policy was brought in.
- From the sets where information is available, 84-93% of passwords consisted of only lowercase characters and digits. This data is extremely informative towards an attack, as it allows masks to be created from a far smaller set of characters and therefore improves the efficiency of an attack. This is backed up by the percentages of passwords containing symbols or lowercase letters being small in comparison.
- An average password length of between 7 and 9 also informs mask composition, as masks of these lengths can be constructed and prioritised over others.

As mentioned, the MySpace password set was collected under a password policy and therefore does not share the high percentage of all lowercase passwords that the other lists do. However, a list of the top 10 masks in the MySpace set can be found from an analysis by Weir [54], giving a clear breakdown of the most popular password structures when a password of all lowercase has not been chosen.

This list of 10 popular masks, found in Figure 4.6.2b, makes up 48.6% of the MySpace password set, and inform the choice of masks used in the attack in section 5.1. Further to being useful to inform masks for an attack, they also provide insight into how users choose their passwords; all 10 of these masks consist of a string of lowercase letters, followed by one or two digits. Password memorability studies on how users construct passwords show that largely passwords are based on a memorable word or phrase [26,52], implying many of these passwords will actually consist of a word or collection of words as the string of letters.

Rank	Mask
1	L6D1
2	L6D2
3	L7D1
4	L8D1
5	L5D2
6	L9D1
7	L7D2
8	L5D1
9	L4D2
10	L8D2

FIGURE 4.6.2b: Top 10 MySpace masks

4.6.3 Dictionaries

For many of these attacks, the choice of dictionary is key. The larger the dictionary size, the greater the chance of it containing words that a user has chosen for their password, but the longer the attack takes. For an attacker working with GPUs as efficient as Tesla K80s, the priority becomes looking for a dictionary of the greatest size, many of which can easily be located online [29,42].

For this model of attack, a large English word corpus was selected containing 3,160,120 words, along with 4 smaller dictionaries each containing common passwords such as `123456`, or passwords from other attacked websites like MySpace or phpBB [42]. These password dictionaries are easily locatable on websites that provide dictionaries for hacking, and contain a total of 225,579 entries.

4.7 Attack summary

After carrying out the investigation summarised in Sections 4.1-4.6, our 'everyday' attacker model is set up as follows:

- The RockYou password list hashed under SHA-256, represents the database of hashes an attacker may get their hands on, with all 14 million distinct hashes stored in a file together.
- Hashcat used to crack as many hashes as possible, using two Tesla K80 GPUs. Each hash that is successfully cracked is removed from the file, so that if future attacks have overlapping attempts with the currently attempt, it is known how many distinct hashes have been cracked.
- The attacks outlined in Section 4.6, implemented on Hashcat, recording both the number of passwords cracked, and the length of time taken for each attack.

More details, the results of the attack, and its consequences are given in the following sections.

5 Results

This section gives the results produced from the main attack, plus further results comparing the effectiveness of different hash functions in protecting a password, and an analysis of the password policies implemented by the top 50 most popular websites [2]. A discussion of the consequences of these results can be found in Section 6.

5.1 Attack results

Over the course of the attack, 142 individual attacks were run against our password set. However, these attacks can be grouped into nine main groups of consecutive attacks of the same style. For example, dictionary attacks were carried out with 13 different substitutive rules (e.g. all e's substituted with 3's), and these can be grouped into one main 'Dictionary with letter substitution' attack.

A table summarising the results of these 9 main attacks can be found in Figure 5.1a, and a graph containing data points from all 142 attacks can be seen in Figure 5.1b. A complete table describing all 142 attacks, with a breakdown of their individual results, can be found in the Appendix, Section 11.4.

As the results show, after 20 hours and 3 minutes of attacking, over 80% of all passwords had been successfully cracked. The first 48.77% of these took only 3 hours and 50 minutes; in fact the first 40% were cracked in just over an hour. With an attack running as such, within one day of acquiring a hashed set of passwords, an attacker could gain access to four in every five users' accounts, and have access to all information the user has protected by this. This is clearly a major threat against password-protected systems as we know them, and a greater awareness of what constitutes a strong password must be shared with the population. A discussion of the results found through this attack can be found in Section 6.

Attack method	Number cracked	Time (seconds)	% cracked (cumulative)
Brute-force (*all* 1-6 character passwords)	2,227,076	2,310	15.53
Dictionary	325,597	713	17.80
Dictionary with letter substitution	46,742	515	18.12
Hybrid: Dictionary + mask	3,972,588	5,704	45.82
Hybrid: Mask + dictionary	423,510	4,572	48.77
Combination (each dictionary with itself)	2,527,488	11,893	66.39
Brute-force (common masks)	1,743,324	29,096	78.55
Combination (pair of different dictionaries)	16,012	197	78.66
Brute-force (further different masks)	207,068	17,199	80.10
Total	11,489,405	20h 3m 19s	

FIGURE 5.1a: Summary of main attack results

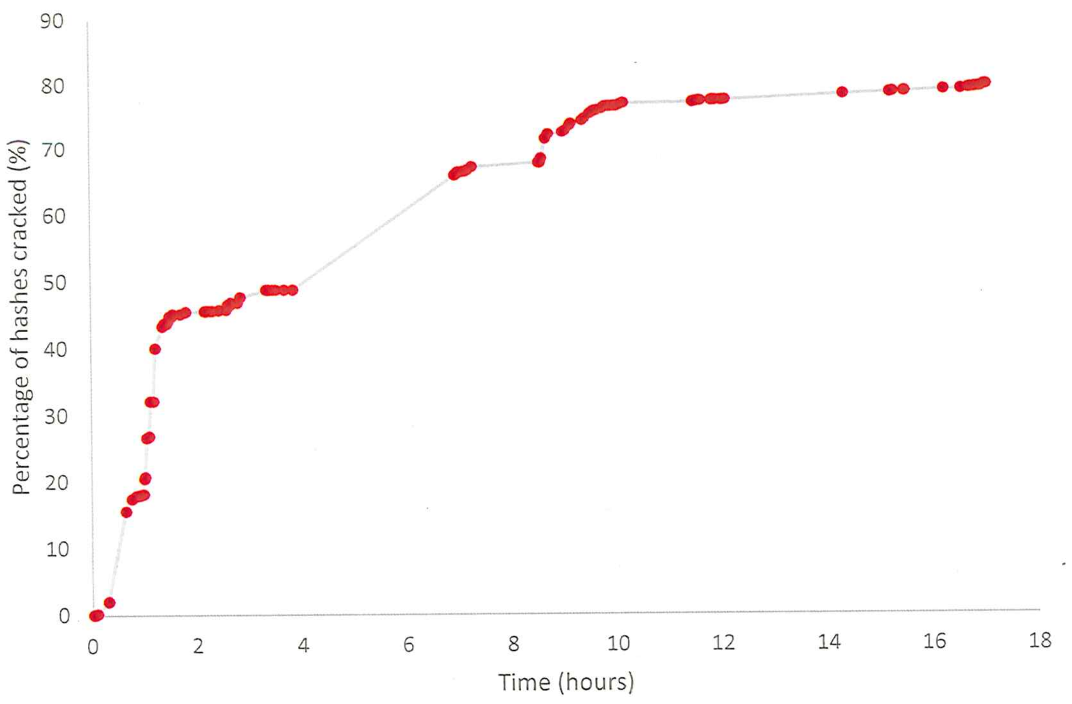


FIGURE 5.1b: Graph of all results from main attack, showing the increasing percentage of passwords cracked throughout the time of the attack.

5.2 Hash function comparison

As well as the main attack against the hashed RockYou set, an analysis of different hash functions was carried out, looking at the cracking speeds of a single Tesla K80 running Hashcat on the same passwords, purely changing which hash function they are hashed under, and therefore also changing which function is being calculated. This style of attack does not take into account potential weaknesses of the hash functions themselves, but merely looks at attacks of the style in previous section. This allows analysis of whether different hash functions provide greater security against *this style* of attack, and by what degree. Separate consideration of whether each function truly is preimage resistant is of course necessary.

The specific functions were chosen either due to their wide usage, or their recommended usage – as discussed in Section 4.3. The results are shown in Figure 5.2a.

Hash function		Hashes per second	Time for attack
LM		2,125,530,000	7h 26m
SHA-256		788,610,000	20h 3m
SHA-512		83,519,480	7d 21h
Salted SHA-256		704,180,000	22h 27m
PBKDF2-HMAC-SHA256	1 round	21,969,590	29d 24h
	1,000 rounds	252,060	7y 59d
	10,000 rounds	25,484	70y 302d
	100,000 rounds	2,593	696y 36d

FIGURE 5.2a: Comparison of the hashing rates of 8 different hash functions on a NVIDIA Tesla K80

Due to the slight variation in speeds between tests, the values given are the average values recorded across 50 different attacks for each hash function. Although 50 is not a large number of tests, the results for each hash function had a relative standard deviation[2] of less than 1%, and so no further tests were required to calculate a reliable average.

The right-hand-most column is purely to add perspective to these hashing rates and is not experimental; it shows how long the tests carried out in Section 5.1 would have taken if the password set were hashed by a different function. These differences in calculation speeds have a clear advantage in being able to improve the security of a system. A further discussion of these results is given in Section 6.1.

5.3 Password policy analysis

Further to the tests run in Sections 5.1 and 5.2, an analysis of the password policies implemented by various websites was carried out. Work has previously been carried out looking at password policies, as described in Section 3 [19,22]. However little analysis of the policies implemented by leading websites has been carried out, therefore, although we know that a stronger password policy can lead to less 'crackable' passwords being created, it is not known whether these are actually being used in practice.

To carry out this analysis, the top 50 most visited websites (as of 30/03/2017) in the UK were recorded, and a new account created for each [2]. Information about the structure they enforce on new passwords was then collected. The most popular UK websites were chosen as opposed to the top 50 global websites, as many of the top 50 global websites correspond to the same accounts (e.g. www.google.com, www.google.co.uk, www.google.fr, etc.), and so a smaller number of distinct accounts would be considered.

Of the top 50 UK websites, four had no option to create an account. The remaining 46 websites corresponded to 37 distinct accounts, and were the password policies analysed. Figures 5.3a-d contain a summary of data collected; a more comprehensive and detailed table can be found in the Appendix, Section 11.5.

Required characters	Number of websites
No restriction	25
1 lowercase, 1 number	6
1 lowercase, 1 number/symbol	2
At least 2 types of character	2
1 number /symbol	1
1 lowercase, 1 uppercase, 1 number	1

FIGURE 5.3a – Table of character restrictions in passwords of the top 50 websites

[2] Relative standard deviation is obtained by dividing the standard deviation by the mean, and multiplying by 100; it is expressed as a percentage [48].

Minimum length	Number of websites
1	2
2	0
3	1
4	2
5	1
6	18
7	1
8	11
9	1
10	0

FIGURE 5.3b, above – Table of enforced minimum password lengths of the top 50 websites

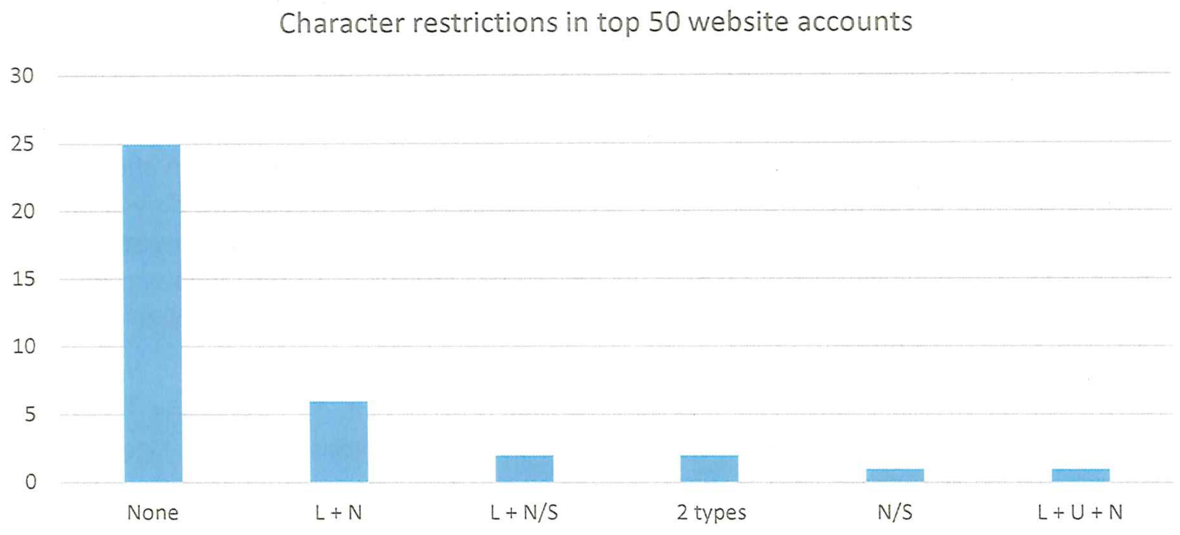


FIGURE 5.3c, above – Character restrictions in passwords of the top 50 websites (data in 5.3a)
L – 1 lowercase, **S** – 1 symbol, **N** – 1 number, **U** – 1 uppercase, **2 types** – any 2 character sets

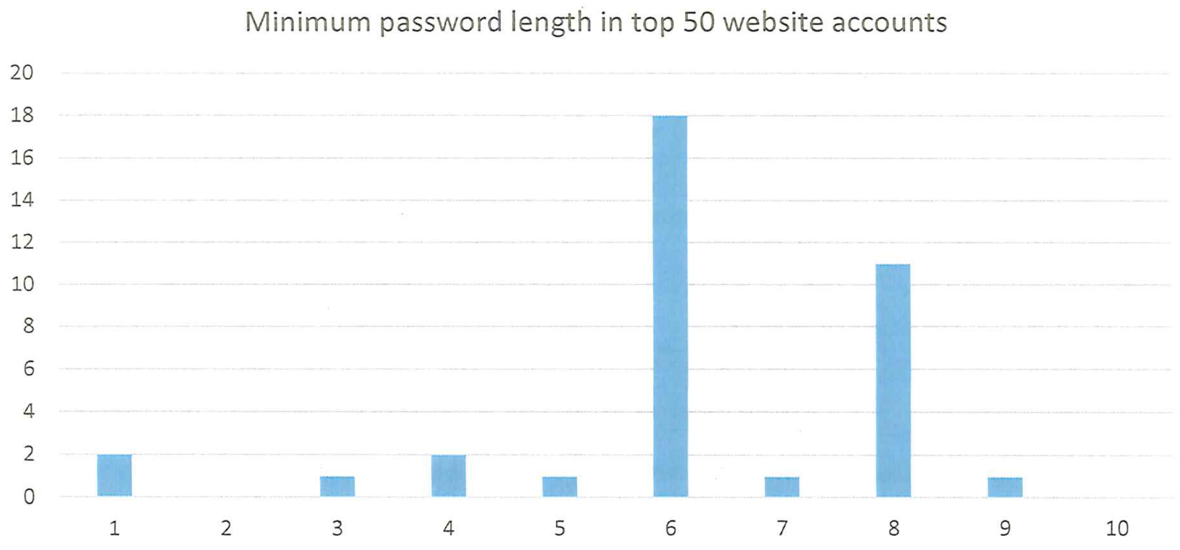


FIGURE 5.3d, above –enforced minimum password length of the top 50 websites (data in 5.3b)

Despite the knowledge from previous research that a stronger password policy leads to creation of less 'crackable' passwords [19,22], our results show that the majority of websites still do not enforce any restrictions over the choices of characters included in a password and most only enforce passwords to be of 6 or 8 characters in length. Furthermore, 25 of the 37 websites enforced no restrictions on the characters within the passwords.

A further service provided by some websites is an indication of password strength, this is usually a scale from 'weak' to 'strong' given on creation of a new password. However, for all the top 50 websites [2], whenever such a scale is given, no reference is provided to illustrate why they are qualified to provide this information. While some websites seem to have a rough idea as to what would constitute a strong password, others provide a scale that misleads users into thinking passwords are secure when they unequivocally are not.

Take, for example, www.facebook.com – a website trusted by millions of users, which has a reputation of being run by some of the world's leading computing brains. On creation of a password on Facebook, the password `$gP&2s` is deemed as 'strong' and a password of `gefajsyctnnupasoqvbpdgftofpyls` is deemed as 'weak'. The former of these passwords can be cracked by the set-up described in this paper in just 20 minutes by carrying out a brute-force of all 6 character passwords. The latter is a pseudorandom string of lowercase letters, requiring a brute-force of all 30 character strings of lowercase letters taking roughly 255.5 years to crack. A further discussion of the results presented in this section is given in Section 6.1.

6 Discussion

6.1 Discussion of attack results

The results of the attack, given in Section 5.1, give some clear insight into the security of the passwords chosen by users: The vast majority of these passwords are not secure, even against an 'everyday' attacker. The first 40% of passwords were cracked in less than 75 minutes meaning realistically even an attacker with a cheaper and less efficient (yet still dedicated) GPU (for example the Tesla M4 costing just £2,000 [45]) could crack a large proportion of the passwords within just a few hours.

The insecurity of so many passwords could be down to many factors, however it is fair to assume that the vast majority of the users who created these passwords do not realise the extent of the vulnerability of their accounts. The password policies implemented by the majority of websites do not require users to construct secure passwords, but *do* put some limitations on the password construction (e.g. minimum of 6 characters), and so this enforcement may lead users to believe that this limitation is a satisfactory level of security.

Many companies, much like the example given of Facebook in Section 5.3, put no weighting on the length of a password, and instead consider the complexity of the character sets they contain. Trusted websites like this are giving their users a false sense of security, leading inexperienced users to believe that their personal information is protected when it is not. Furthermore these 'secure' passwords are likely to then be used for other accounts by this user [11,21], increasing their vulnerability to attack.

Of the 37 websites, 25 enforced no restrictions on the characters within the passwords set. Considering the research shown in Figure 5.1a, we can see that around 50% of the passwords created for all 25 of these unrestricted websites will likely be constructed purely from lowercase letters, decreasing the size of a brute-force attack from 95^n to just 26^n, where n is the length of the password chosen. Also in Figure 5.1a we can see the average password length chosen is 7-9. Using our attack set-up, all nine character lowercase passwords (26^9) can be cracked in less than 77 minutes, clearly showing more information needs to be given to users in order to help them protect their personal information and accounts.

The attacker simulated in this project has been described as an 'everyday' attacker, however the attack would require research online and logical planning. The word 'everyday' is used because all hardware, software, and background understanding are readily available online, and the attack can be performed without detailed knowledge of its workings. Some attackers may structure their attack with less research and logical structure, leading them to perform attacks in a less efficient order, possibly with unnecessary testing in between. However the efficiency of the attack in Section 5.1 means an attacker could be 8 times less efficient than our model, and still crack 80% of passwords in less than a week; this is therefore still a severe threat.

In terms of the hash function used by these systems, the results in Figure 5.2a show clearly that the choice of hash function can greatly dictate the security of the passwords it protects. If a PBKDF2-HMAC-SHA256 hash were to be calculated with 100,000 rounds for every password entered, then attacking the hashed password database would be an incredibly difficult and lengthy task.

However, the increased length of time for an attacker to calculate each hash is also an increased time for the system employing the hash function to calculate each hash. For some systems this may not be a problem, however consider a large system like Facebook, with nearly 2 billion users active every month, and over 1.25 billion users active daily [46,47]. For such systems, both security and efficiency are key considerations, and so a balance must be found between the length of time taken to calculate the hashes, and allowing the system to run efficiently.

6.2 Recommendations for password-protected systems

The form of attack described in this paper is known as an offline attack; the attacker gets hold of the hashed password database, and has as much time and as many attempts as they like to try to attack the data. Although companies believe they have stored their database in a secure location, it is still important that they do not assume the database is safe from attackers, and therefore must assume that an attacker has access to the database itself. For this reason, the hash function applied is often the only level of security that protects each password in the database, and therefore this must be carefully chosen – along with carefully chosen passwords.

PBKDF2-HMAC-SHA256 is a variant of the commonly used hash function SHA256, which, as shown in Section 5.2, allows calculation of fewer hashes per second due to its computational intensity. For maximum protection of users' passwords, this hash function should be used with as many rounds as possible, while still allowing the system to run as efficiently as required. In practice, from the results of the hash functions provided in Section 5.2, it can be seen that more than 10,000 rounds is not necessary unless the system contains security-critical information – even if the attacker had access to 140 Tesla K80 GPUs (costing roughly £1m) our attack would still take them an entire year to carry out against PBKDF2-HMAC-SHA256.

In addition to the hash function, the most important security is the password itself. Although the defending system should not choose the users' passwords themselves – as this leads users to writing down and forgetting passwords, they can help to ensure that users create secure passwords by implementing strong password policies, and giving more accurate measurements of password strength [19,22,56].

Password policies should ideally enforce all passwords to:

- Not be formed from only digits, as these can be efficiently cracked
- Have *at least* 8 characters if a mixture of 3 or 4 character sets are used. This is due to the fact that a brute-force of 8 character passwords takes our attacker 61 days to complete, giving the accounts security against this.
- Have *at least* 12 characters for passwords using just 1 or 2 character sets. Popular masks can be generated, as they were when the MySpace password database were leaked, and so if just 1 or 2 character sets are used, usually this would be just lowercase and digits, and therefore additional length is required so that even popular masks take time to crack.

In addition to protection against offline attacks, defending systems can put measures in place to help protect themselves from online attacks. Attackers try each password in turn into the log-in form, and are therefore limited by the frequency with which each attempt can be tried; implementing a timeout after a fixed number of incorrect attempts helps to protect against these.

6.3 Recommendations for passwords

Using a password set that was leaked in plaintext means that the set of uncracked passwords can easily be recovered by performing a dictionary attack using the set of all RockYou passwords against the remaining hashes. This allows an analysis of the strongest passwords to be done, found in Section 6.3.1, which can then be used to form recommendations of how strong passwords should be structured in Section 6.3.2.

Characteristic	RockYou set	Uncracked set
Average password length	8.74	11.31
Single character type	44%	37.4%
Only lowercase letters	26%	35.6%
Only digits	16.4%	0.2%
Only uppercase letters	1.6%	1.6%
Only symbols	0.04%	0.2%
Contain a symbol	6.6%	25.2%
7 characters or less	33%	6.5%
9 characters or less	69.0%	29.7%

FIGURE 6.3.1a: Comparison of statistics for RockYou vs uncracked password sets

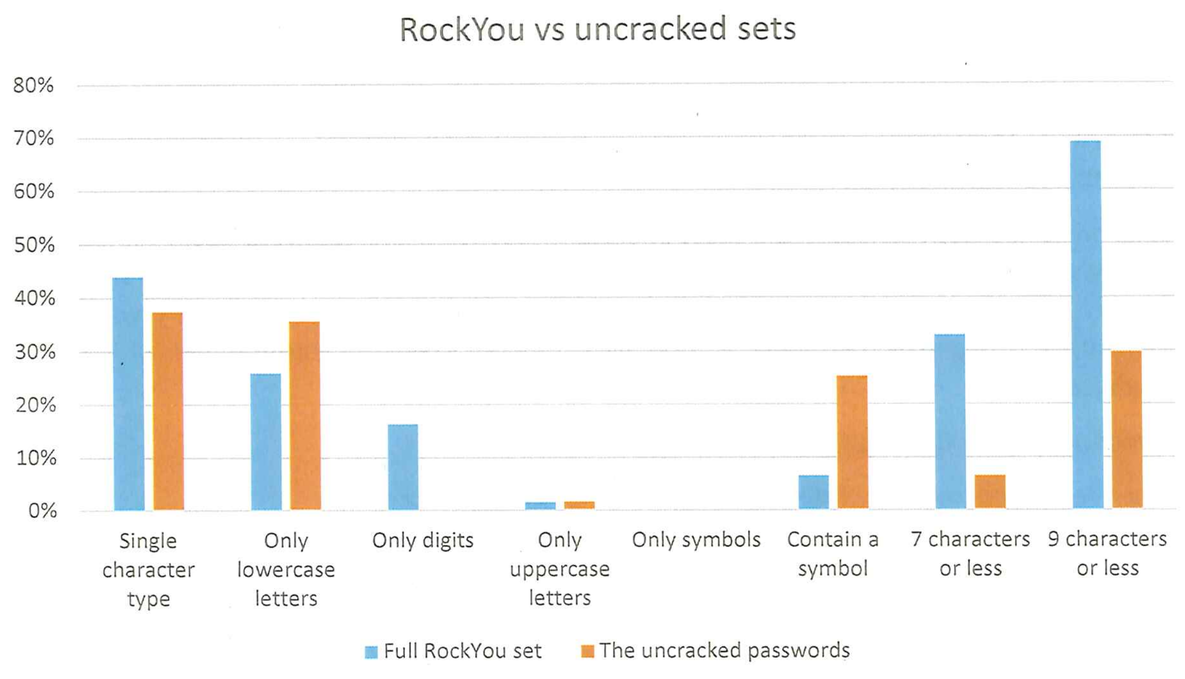


FIGURE 6.3.1b: Graph showing the comparison in 6.3.1a

6.3.1 The uncracked passwords

An analysis of the passwords in the entire RockYou password set, and their composition of characters was carried out by Korelogic [23]. This allows the uncracked passwords in our attack to be compared to the original set, thereby shedding light on how these passwords differ from the cracked ones. In Section 5.1, 2,854,239 passwords were left uncracked after the 20 hour attack – this is 19.9% of the original set of passwords. Figures 6.3.1a and 6.3.1b show an analysis of the RockYou passwords vs the uncracked passwords.

These results confirm the research that say longer passwords are less 'crackable' [19,22], through an increase from an average of 8.74 to 11.31 characters. Furthermore, the percentage of passwords containing a symbol has increased from 6.6% to 23.4% showing that passwords with a symbol in them are less likely to be cracked than those without a symbol. These properties cannot be considered separately, as all passwords of length less than 7, including those containing symbols, were cracked within 77 minutes.

These two results have been discussed by previous research trying to formulate the creation of secure passwords, however Figure 6.3.1a also contains some results which are more surprising. The percentage of passwords constructed from only lowercase letters shows an increase from 26% to nearly 36% in the uncracked set. A further analysis of the uncracked dataset shows that the average length of the uncracked passwords which were created using only lowercase letters is 13.24 – significantly longer than the average length of all uncracked passwords.

An additional result of note is the percentage of passwords constructed only from digits. This decreases from 16.4% to 0.2%. This result is due to the number of possible digits being only 10 – compared to 26 for lowercase or uppercase letters and 33 for symbols. This means that a brute-force attack against passwords containing only digits is far more efficient than one against passwords consisting of a different character type.

6.3.2 Generation of strong passwords

From the previous section, informed conclusions can be drawn as to how users should create passwords to withstand our 'everyday' attacker. It is easy to say that secure passwords are long random strings, however, as discussed in Sections 1.1 and 3, this leads to passwords being written down or being forgotten leading to new passwords requested, both of which lead to security problems.

The most important feature of the passwords leading to them remaining uncracked is their length. Any form of mask or brute-force attack increases exponentially in time as the length of the password increases. Additional attack styles do allow attacks on passwords of a greater length without carrying out the exponential brute-force, however these attacks target passwords of a set structure:

- One dictionary entry, possibly with rules applied to switch the case of letters and/or replace occurrences of one character with another specified character.
- Two concatenated dictionary entries
- One dictionary entry with a mask applied to the beginning or end

To combine both password memorability, and security, long passwords can be created as a combination of four or more words appended to one another, possibly with numbers/symbols interspersed. These numbers and symbols could either be as a substitution for letters, or as part of the memorable phrase – for example Ex4mpleP4sswordForSecurity, or ThisIsInThe20%Uncracked.

Comparable advice has been given by individuals previously, however the severity of the risk of weaker passwords is not emphasised and is not evidenced by simulation of an attack [41].

The use of four or more words not only ensures password length is greater than the average user had created for RockYou, but also ensure it is resistant to combinations of dictionary attacks. For a dictionary of over 3 million words, trying a combination of n words leads to $3{,}000{,}000^n$ possible passwords needing to be hashed, meaning that only combinations of two words is a feasible attack.

An extension to this structure of password generation is to add spaces between the words. A space is in the 'symbol' character set, and so removes the possibility of a brute-force against passwords with just letters. Of the uncracked passwords 11.1% contained at least one space in them – these make up nearly half of the uncracked passwords containing a symbol. However, some systems prohibit use of a number of symbols, which limits the range of passwords that can be created – a handful of systems do not allow any symbols to be used at all [30].

7 Reflection

Initially this project started with the intention of carrying out a large-scale password hacking comparison on both a CPU and a GPU. However it quickly became apparent that a large-scale attack was not going to be practical on a CPU which attacks just one hash far slower than a GPU. I therefore decided it would be better to focus on the vulnerability of passwords to an 'everyday' attacker using GPUs, with a minor consideration of CPUs. This would have received far less focus in the original project plan, but has far more practical implications for protecting users' data, whilst the original plan would have been more of a technological investigation.

Setting up Hashcat to run on my laptop was a lot harder than I initially thought it would be. The error messages provided by Hashcat are not very meaningful, and only after days of research did I discover that the laptop in question was not actually capable of running the necessary drivers for Hashcat to work, and so a new plan of action was needed.

The use of the state-of-the-art GPUs belonging to ARC [1] ran far more smoothly than the attempt with an old laptop. Unfortunately, Hashcat had to be built from scratch from the source code, as the set-up in ARC means that there is no operating system available to download the software onto. However once this was set up, the tests began running smoothly and efficiently.

The results of the attack came as a surprise to me; I was expecting to crack a number of the RockYou passwords, however the speed with which the attack cracked such a large proportion of passwords was staggering. Brute-force and dictionary attacks against hashed passwords were briefly mentioned in Computer Security lectures, however the extent of the efficiency of parallelising these attacks against a set of passwords was not made apparent.

When the time came to start running the longer tests, taking over an hour apiece, it became apparent that it was not possible for a job to run for this long without being automatically halted by ARC, with no progress saved. On speaking to the support at ARC they were not immediately sure as to what was causing the problem, so I instead had to find a way of creating restore points, so that after being logged off automatically, I could easily log in and resume the test without losing progress.

Before completing this project, I had never had to deal with large-scale data before, and so I've always been able to carry out small alterations to data in editing software, or by writing a simple program. However the scale of the passwords in this project meant that many text editing programs could not even open the file, and so other ways of editing the contents had to be found. I discovered many efficient command-line tools exist for dealing with such large stream of data, and learnt to adapt these to perform the necessary alterations. Hashing of passwords, removal of extra data and analysis of uncracked passwords were all carried out through use of bash scripts and command-line commands.

In hindsight, progress of the project could only take-off after permission had been granted to use the RockYou password set by approval of a CUREC 1A form. The application for permission was only sent in when I wanted to start work using the passwords, without realising that the approval can take 30-60 days to be confirmed. If I were to carry out similar work in future, I would therefore consider whether CUREC approval was needed earlier in the process.

8 Conclusions

The 'everyday' attacker simulated in this project demonstrated a new angle on the art of password cracking, by successfully cracking the vast majority of the passwords in a large corpus using a simple set-up. The efficiency and success of the attack shed light on the extent of the vulnerability of real-world password-protected accounts, and exposed flaws in the understanding of what it means for a password to be secure. Education of the general population, and update of both hash functions used and password-creation policies are vital next steps if accounts are to become more secure against attackers in the near future.

9 Future work

Although probabilistic password-cracking methods are not currently accessible to those without detailed knowledge of probabilistic methods, once an algorithm has been designed it is easily replicated and run without knowledge of how it works. This means it is only a matter of time before weaponised toolkits implementing password-cracking with probabilistic CFGs, Markov models or neural networks are available for the general population to download, and when this happens our 'everyday' attacker will suddenly be capable of far more complex and efficient attacks. For this reason, to continue to educate users about what makes a password secure, an analysis of the password structures most resilient against these more complex attacks should be carried out.

Furthermore, although recommendations for constructing a secure password have been given in this report, it is important that further analysis is carried out on the human-factor side of password generation. As shown by our attack, humans have a tendency to follow trends and therefore generate similarly structured passwords. If a large proportion of the population now moves to create more secure passwords, but all follow similar trends in doing this, efficient attacks may be developed to target a new, specific structure of password.

10 References

[1] Advanced Research Computing. (2017). http://www.arc.ox.ac.uk/

[2] Alexa. (2017). Top sites in United Kingdom. http://www.alexa.com/topsites/countries/GB accessed on (30/03/17)

[3] Apple. (2017). iOS Security guide: iOS 10.

[4] Cain and Abel. (2014). Password recovery tool for Microsoft Operating Systems. http://www.o xid.it/cain.html

[5] Crack Station. (2016). Salted password hashing – doing it right. https://crackstation.net/hashing-security.htm

[6] Crypto-IT. (2017). Kerckoff's Principle. http://www.crypto-it.net/eng/theory/kerckhoffs.html

[7] Dell'Amico, M., Michiardi, P. and Roudier, Y. (2010). Password strength: An empirical analysis. In *International conference on computer communications*

[8] Dark Net. (2016). IGHASHGPU – GPU based hash cracking – SHA1, MD5 & MD4. https://www.darkn et.org.uk/2016/08/ighashgpu-gpu-based-hash-cracking-sha1-md5-md4/.

[9] Duo. (2010). Brief analysis of the Gawker password dump. https://duo.com/blog/brief-analysis-of-the-gawker-password-dump.

[10] ElcomSoft. (2010). Smartphone forensics: Cracking blackberry backup passwords. https://blog.el comsoft.com/2010/09/smartphone-forensics-cracking-blackberry-backup-passwords/

[11] Florencio, D. and Herley, C. (2007). A large-scale study of web password habits. In *International world wide web conferences.*

[12] Hashcat. (2016). Advanced password recovery; the world's fastest password cracker. https://hashca t.net/hashcat/

[13] How To Geek. (2011). Analysis of Sony leak shows weak and duplicate passwords. *Blog.* https://www.howtogeek.com/94174/analysis-of-sony-leak-shows-weak-and-duplicate-passwords/.

[14] Hunt, T. (2011). A brief Sony password analysis. *Blog.* https://www.troyhunt.com/brief-sony-password-analysis/.

[15] Intel. (2017). Intel core processors. https://www.intel.co.uk/content/www/uk/en/homepage.html

[16] Internet Engineering Task Force. (2000). Request for comments 2898: PKCS #5: Password-based cryptography specification, version 2.0

[17] Internet Engineering Task Force. (2005). Request for comments 3962: Advanced Encryption Standard encryption for Kerberos 5.

[18] John the Ripper. Password cracker. http://www.openwall.com/john/

[19] Kelley, P. G., Komanduri, S., Mazurek, M. L., Shay, R., Vidas, T., Bauer, L., Christin, N., Cranor, L. F. and Lopez, J. (2012). Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms. In *IEEE symposium on security and privacy.*

[20] Ker, A. D. (2014). Computer Security. *Lecture notes, University of Oxford.*

[21] Keszthelyi, A. (2013). About Passwords.

[22] Komanduri, S., Shay, R., Kelley, P. G., Mazurek, M. L., Bauer, L., Christin, N., Cranor, L. F. and Egelman, S. (2011). Of passwords and people: Measuring the effect of passwords-composition policies. In *Human factors in computing systems.*

[23] KoreLogic. (2016). LinkedIn revisited – full 2012 hash dump analysis. https://blog.korelogic.com/bl
og/2016/05/19/linkedin_passwords_2016.

[24] LastPass. (2011). LastPass Security Notification. https://blog.lastpass.com/2011/05/lastpass-security-notification.html/

[25] Lemos, R. (2003). Cracking Windows passwords in seconds. https://www.cnet.com/news/cracking-windows-passwords-in-seconds/

[26] Leyden, J. (2003). Office workers give away passwords for a cheap pen. https://www.theregister.c

o.uk/2003/04/18/office_workers_give_away_passwords/

[27] Ma, J., Yang, W., Luo, M. and Li, N. (2014). A study of probabilistic password models. In *IEEE symposium on security and privacy.*

[28] Melicher, W., Ur, B., Segreti, S. M., Komanduri, S., Bauer, L., Christin, N. and Cranor, L. F. (2016). Fast, Lean and Accurate: Modelling password guessability using neural networks. *USENIX Security, Best paper.*

[29] Miessler, D. (2017). Dictionary collection. https://github.com/danielmiessler/SecLists/tree/m
aster/Passwords

[30] Miessler, D. (2007). The list of shame: websites that don't allow special characters in their passwords. https://danielmiessler.com/blog/the-list-of-shame-websites-that-dont-allow-special-chara
cters-in-their-passwords/.

[31] Microsoft. (2012). Passwords technical overview. https://technet.microsoft.com/en-us/libr
ary/hh994558(v=ws.10).aspx

[32] Microsoft. (2017). Integrity with hash functions. https://technet.microsoft.com/en-gb/libr
ary/cc959516.aspx

[33] Narayanan, A. and Shmatikov, V. (2005). Fast dictionary attack on passwords using time-space tradeoff. In *Computer and communications security.*

[34] NVIDIA. (2017). Tesla high-performance computing. http://www.nvidia.co.uk/object/tesla-high-performance-computing-uk.html

[35] Online Hash Crack. (2017). Extract hashes and crack Mac OS X passwords. https://www.onli
nehashcrack.com/how-to-extract-hashes-crack-mac-osx-passwords.php

[36] Page, T. (2009). The application of hash chains and hash structures to cryptography.

[37] Pillai, S. (2013). Understanding hashing with shadow utils.

[38] Project Ranbow Crack. (2017). Rainbow Tables. http://project-rainbowcrack.com/table.htm

[39] Rainbow Technologies. (2003). Password survey results. http://mktg.rainbow.com/mk/get/pws

urvey03

[40] Rankin, K. (2012). Hack and / - password cracking with GPUs. In *Linux Journal.* http://www.linuxjo urnal.com/content/hack-and-password-cracking-gpus.

[41] SANS Institute for the internet community. (2014). Password construction guidelines.

[42] Skull Security. (2015). Password dictionaries and leaked passwords. https://wiki.skullsecu rity.org/Passwords

[43] Source Forge. (2015). Magical rainbow table generator. https://sourceforge.net/projects/m rtg/?sourc e=recommended.

[44] Sprengers, M. and Batina, L. (2012). Speeding up GPU-based password cracking. *SHARCS 2012.*

[45] Software Direct. (2017). Tesla M4 Module. http://www.cadsoftwaredirect.com/tesla-m4-mod ule.html

[46] Statista. (2017). Facebook: number of monthly active users worldwide 2008-2017. https://www.s tatista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/

[47] Statista. (2017). Facebook: number of daily active users worldwide 2008-2017. https://www.statis ta.com/statistics/346167/facebook-global-dau/

[48] Texas A&M University. (2015). Average, standard deviation and relative standard deviation. http://www.chem.tamu.edu/class/fyp/mathrev/std-dev.pdf

[49] TrueCrack. (2015). Code repository for TrueCrack. https://github.com/lvaccaro/truecrack.

[50] Theocharoulis, K. and Papaefstathiou, I. (2010). Implementing rainbow tables in high-end FPGAs for super-fast password cracking. In *2010 International conference on Field Programmable Logic and Applications.*

[51] VeraCrypt. (2017). Free disk encryption software. https://veracrypt.codeplex.com/.

[52] Vu, K. P. L., Proctor, R. W., Bhargav-Spantzel, A., Tai, B. L., Cook, J. and Schultz, E. E. (2007). Improving password security and memorability to protect personal and organisational information. In *International journal of human-computer studies.*

[53] Vulnerability Assessment. (2010). Download for IGHASHGPU. http://www.vulnerabilityassess ment.co.uk/ighashpu.htm

[54] Weir, C. M. and Aggarmal, S. (2010). Using probabilistic techniques to aid in password cracking attacks. *Florida State University Libraries.*

[55] Weir, C. M., Aggarwal, S., Medeiros, B. de. and Glodek, B. (2009). Password cracking using probabilistic context-free grammars. In *IEEE symposium on security and privacy.*

[56] Winstanley, P. A., (2004). Processing strategies and the generation effect: implications for making a better reader. *Memory and cognition.*

[57] WP Engine. (2015). Unmasked: What 10 million passwords reveal about the people who choose them. http://wpengine.com/unmasked/

[58] Burnett, M. (2015). A glimpse into the world of internet password dumps. For *XATO.* https://xato.n et/a-glimpse-into-the-world-of-internet-password-dumps-5ee4609da237

11 Appendix

11.1 Table of CPU vs. GPU hashing rates

FIGURE 11.1a: Table of results from Intel HD Graphics 5500 vs Intel Core i5-5200U tests. Measured in Mega Hashes per second.

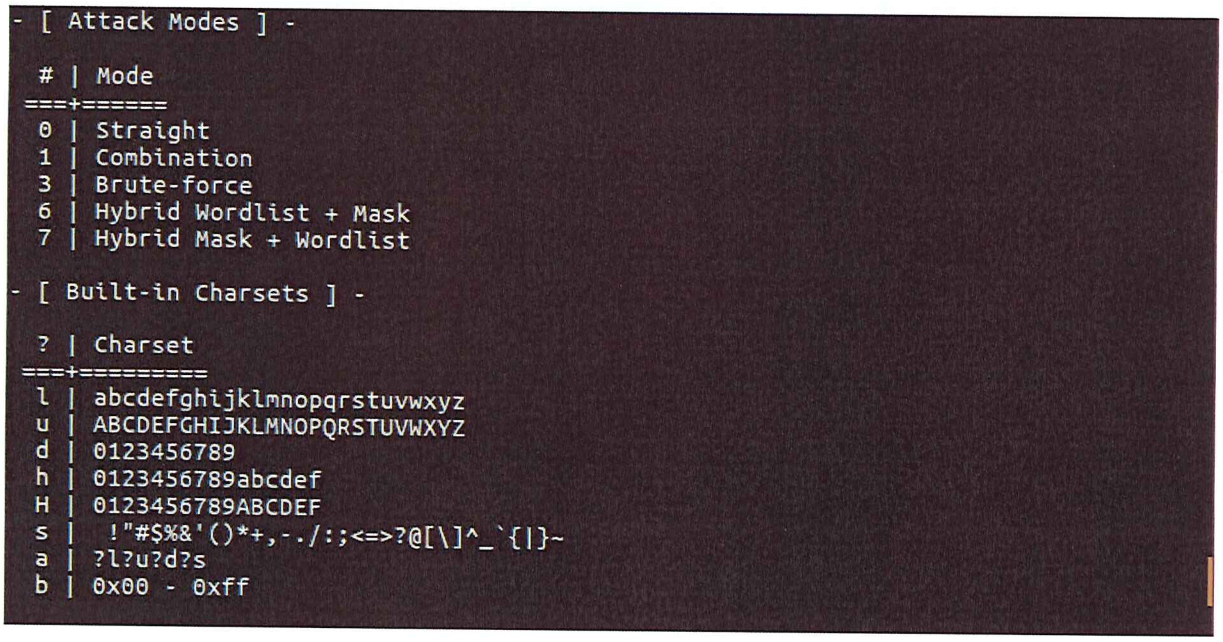
Brute-force length	**2**	**3**	**4**	**5**	**6**	**7**	**8**
GPU test 1	0.5333	27.0415	52.4512	52.103	49.5202	49.8839	49.9061
GPU test 2	0.9095	27.8881	54.9547	52.2149	49.7022	49.9936	49.9458
GPU test 3	0.8068	15.3384	52.5475	51.5109	49.3323	49.828	49.9268
GPU test 4	0.5223	37.4036	52.931	52.2339	46.8806	50.0073	50.0441
GPU test 5	0.986	29.7611	52.473	52.116	50.1052	50.0389	49.9834
GPU test 6	0.296	24.2335	43.7936	52.4198	49.9107	49.9027	49.9086
GPU test 7	0.3529	32.7773	51.3805	52.2134	49.9275	49.9055	50.0125
GPU test 8	0.8624	38.0058	48.7751	51.6564	49.9341	50.004	50.0433
GPU test 9	0.6659	38.9054	51.8683	52.2541	46.0839	49.8421	49.9398
GPU test 10	0.6154	32.8572	52.0984	52.2691	49.9376	49.7483	49.975
CPU test 1	10.4262	32.4037	33.7115	33.8168	40.1038	40.1038	40.1348
CPU test 2	15.7388	32.7723	33.4835	33.7334	39.0024	39.0024	40.1986
CPU test 3	15.9289	33.9148	32.7542	33.8653	40.2004	40.2004	40.231
CPU test 4	11.286	35.0107	33.8699	33.898	40.1542	40.1542	40.1472
CPU test 5	15.4923	31.7499	33.9049	33.852	40.1504	40.1504	39.3444
CPU test 6	12.3056	33.9282	33.6039	33.957	39.944	39.944	39.3093
CPU test 7	14.72	34.081	33.9952	33.8511	40.208	40.208	39.3053
CPU test 8	14.928	33.0924	33.6748	33.6887	40.1329	40.1329	40.2322
CPU test 9	14.273	34.898	33.8685	33.9875	40.2095	40.2095	40.2206
CPU test 10	17.4459	34.81	33.953	33.9758	40.0327	40.0327	39.3015

11.2 Table of Intel HD Graphics vs. NVIDIA Tesla K80 hashing rates

FIGURE 11.2a: Table of results from Intel HD Graphics 5500 vs NVIDIA Tesla K80 tests. Measured in Mega Hashes per second.

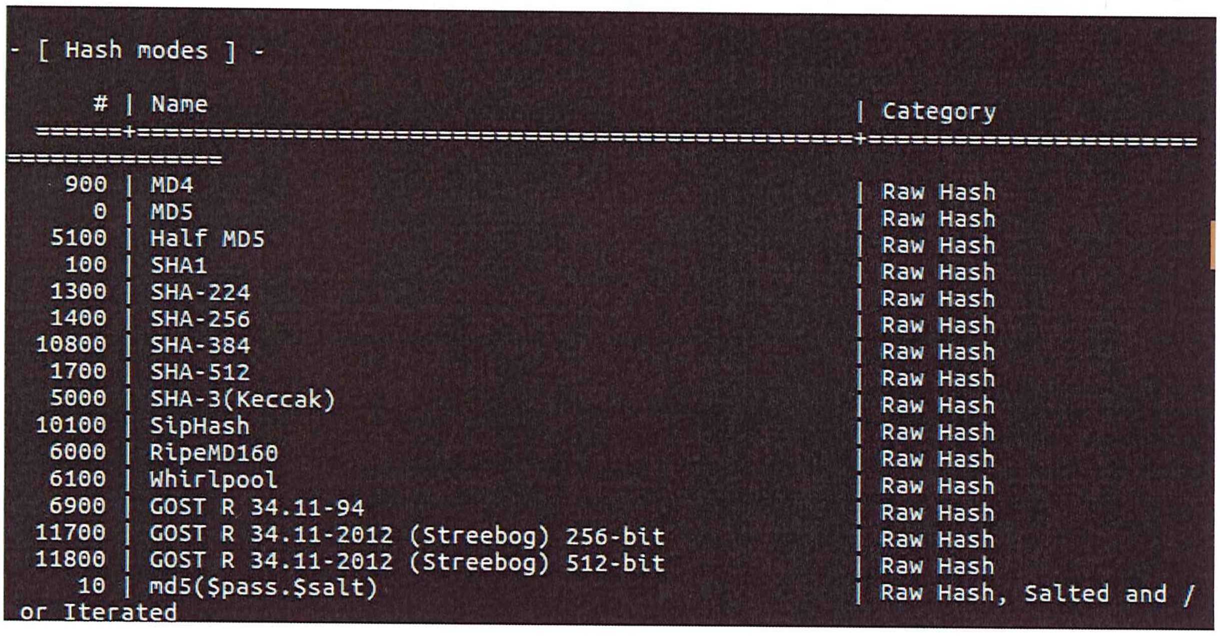
Brute-force length	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Intel test 1	0.5333	27.0415	52.4512	52.103	49.5202	49.8839	49.9061
Intel test 2	0.9095	27.8881	54.9547	52.2149	49.7022	49.9936	49.9458
Intel test 3	0.8068	15.3384	52.5475	51.5109	49.3323	49.828	49.9268
Intel test 4	0.5223	37.4036	52.931	52.2339	46.8806	50.0073	50.0441
Intel test 5	0.986	29.7611	52.473	52.116	50.1052	50.0389	49.9834
Intel test 6	0.296	24.2335	43.7936	52.4198	49.9107	49.9027	49.9086
Intel test 7	0.3529	32.7773	51.3805	52.2134	49.9275	49.9055	50.0125
Intel test 8	0.8624	38.0058	48.7751	51.6564	49.9341	50.004	50.0433
Intel test 9	0.6659	38.9054	51.8683	52.2541	46.0839	49.8421	49.9398
Intel test 10	0.6154	32.8572	52.0984	52.2691	49.9376	49.7483	49.975
K80 test 1	8.1194	550	746.6	738.5	788.4	786	788.9
K80 test 2	8.8276	569.3	747.7	740.1	789.5	788.4	788.3
K80 test 3	9.029	551.4	749.7	737.3	788.9	788.2	789.2
K80 test 4	8.9363	566.2	751	739.7	789.3	788.4	788.7
K80 test 5	8.1196	554.3	747.2	737.4	788.5	788.6	788.3
K80 test 6	8.4174	563.9	746.6	737.9	788.9	788.4	788.5
K80 test 7	9.1237	561.7	746.3	738.6	787.9	788.5	788.6
K80 test 8	9.0104	560.9	748.5	737	788.5	788.6	787.5
K80 test 9	9.0478	558	752.3	741	789	789	788.4
K80 test 10	8.9917	553.6	746.3	739.9	788.1	788.8	789.7

11.3 Screenshots of Hashcat



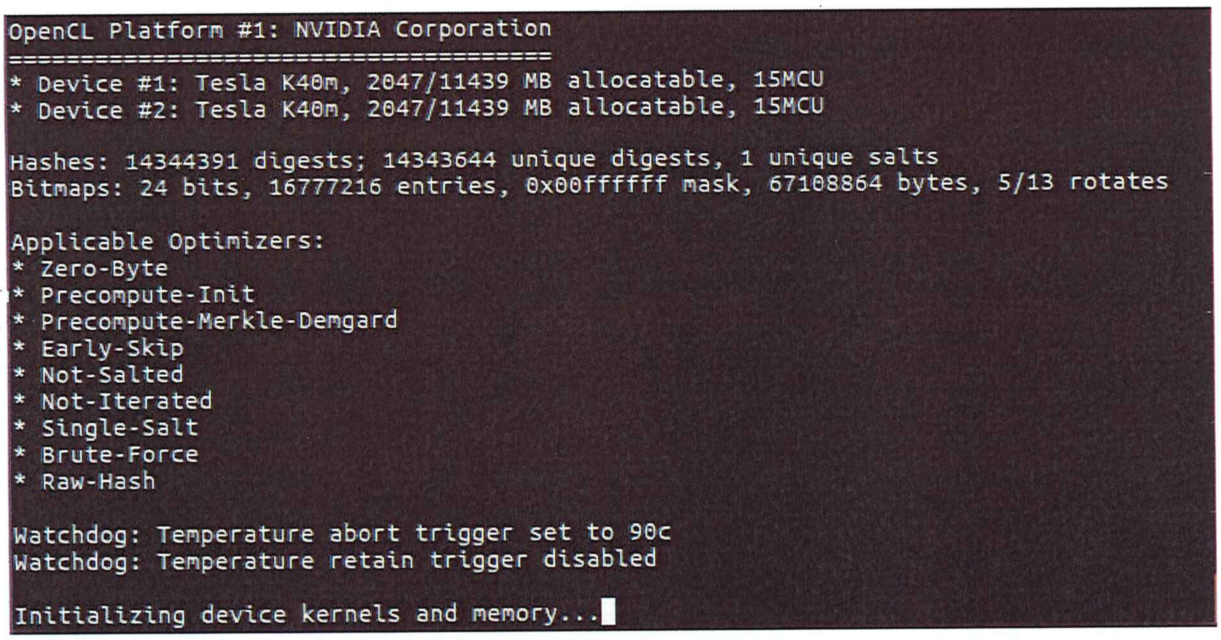
```
- [ Attack Modes ] -

  # | Mode
 ===+======
  0 | Straight
  1 | Combination
  3 | Brute-force
  6 | Hybrid Wordlist + Mask
  7 | Hybrid Mask + Wordlist

- [ Built-in Charsets ] -

  ? | Charset
 ===+=========
  l | abcdefghijklmnopqrstuvwxyz
  u | ABCDEFGHIJKLMNOPQRSTUVWXYZ
  d | 0123456789
  h | 0123456789abcdef
  H | 0123456789ABCDEF
  s |  !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
  a | ?l?u?d?s
  b | 0x00 - 0xff
```

FIGURE 11.3a: Screenshot showing Hashcat attack modes, and character sets. Retrieved by command `hashcat --help`



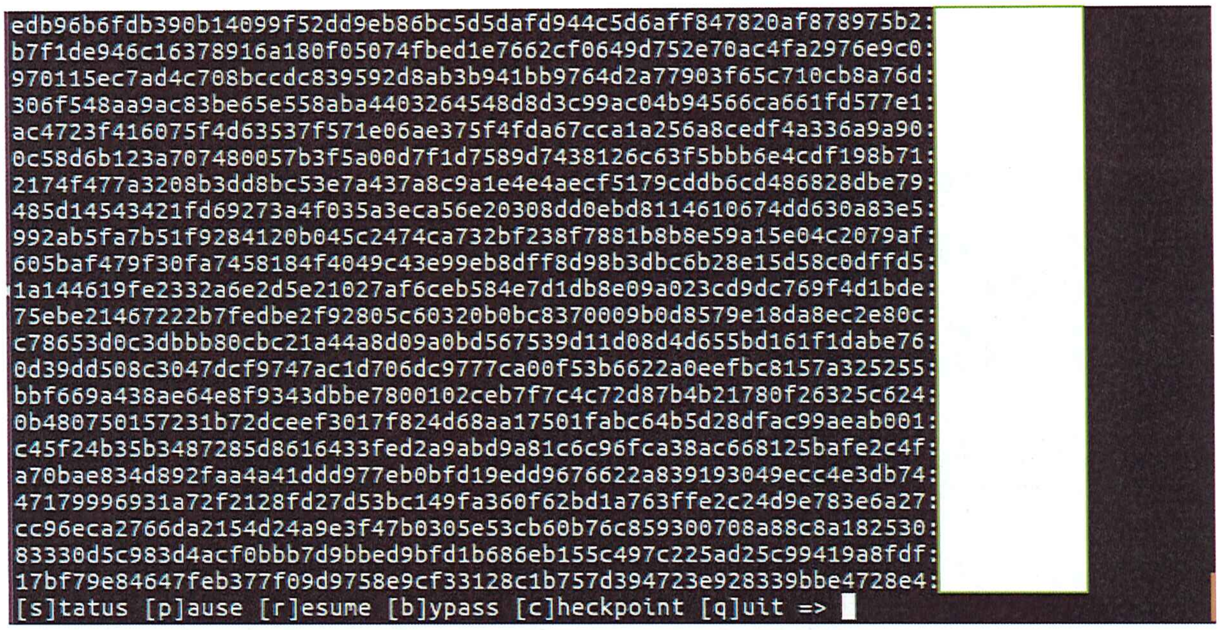
```
- [ Hash modes ] -

      # | Name                                             | Category
  ======+==================================================+======================
===============
    900 | MD4                                              | Raw Hash
      0 | MD5                                              | Raw Hash
   5100 | Half MD5                                         | Raw Hash
    100 | SHA1                                             | Raw Hash
   1300 | SHA-224                                          | Raw Hash
   1400 | SHA-256                                          | Raw Hash
  10800 | SHA-384                                          | Raw Hash
   1700 | SHA-512                                          | Raw Hash
   5000 | SHA-3(Keccak)                                    | Raw Hash
  10100 | SipHash                                          | Raw Hash
   6000 | RipeMD160                                        | Raw Hash
   6100 | Whirlpool                                        | Raw Hash
   6900 | GOST R 34.11-94                                  | Raw Hash
  11700 | GOST R 34.11-2012 (Streebog) 256-bit             | Raw Hash
  11800 | GOST R 34.11-2012 (Streebog) 512-bit             | Raw Hash
     10 | md5($pass.$salt)                                 | Raw Hash, Salted and /
 or Iterated
```

FIGURE 11.3b: Screenshot showing *some* of the Hashcat hash functions. Retrieved by command `hashcat --help`



```
OpenCL Platform #1: NVIDIA Corporation
======================================
* Device #1: Tesla K40m, 2047/11439 MB allocatable, 15MCU
* Device #2: Tesla K40m, 2047/11439 MB allocatable, 15MCU

Hashes: 14344391 digests; 14343644 unique digests, 1 unique salts
Bitmaps: 24 bits, 16777216 entries, 0x00ffffff mask, 67108864 bytes, 5/13 rotates

Applicable Optimizers:
* Zero-Byte
* Precompute-Init
* Precompute-Merkle-Demgard
* Early-Skip
* Not-Salted
* Not-Iterated
* Single-Salt
* Brute-Force
* Raw-Hash

Watchdog: Temperature abort trigger set to 90c
Watchdog: Temperature retain trigger disabled

Initializing device kernels and memory...
```

FIGURE 11.3c: Screenshot showing Hashcat initialising before an attack. Retrieved by command `hashcat -m 1400 -a 3 rockyou.txt ?l?l?l?l?l?l`

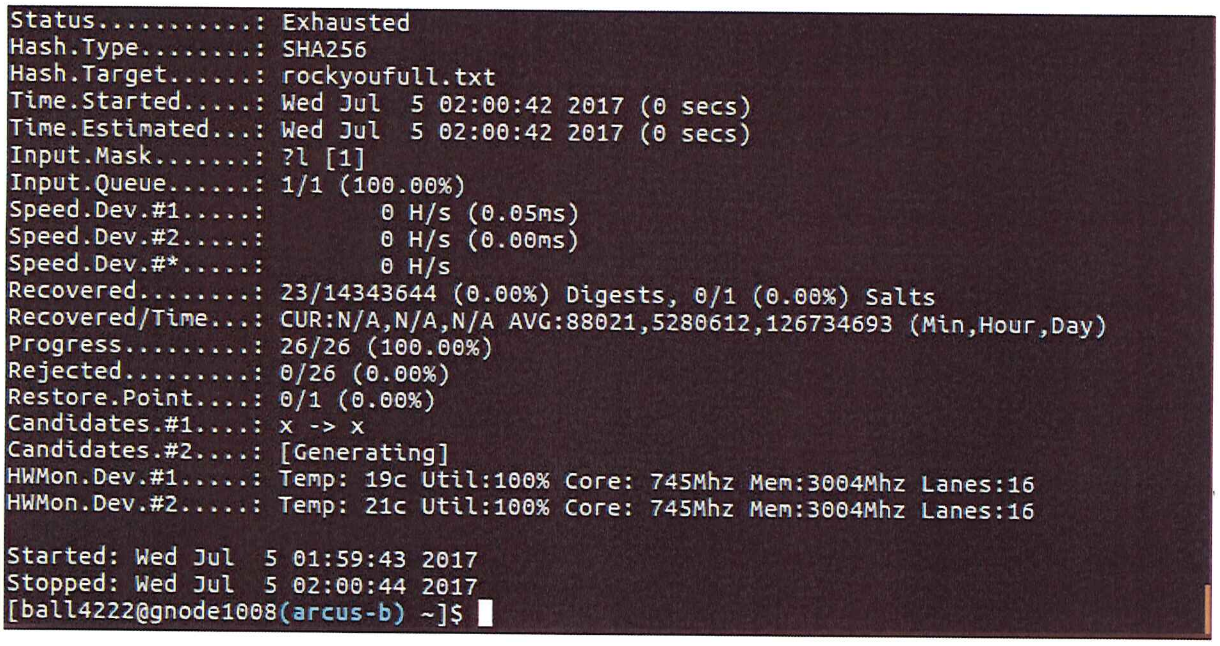


```
edb96b6fdb390b14099f52dd9eb86bc5d5dafd944c5d6aff847820af878975b2:
b7f1de946c16378916a180f05074fbed1e7662cf0649d752e70ac4fa2976e9c0:
970115ec7ad4c708bccdc839592d8ab3b941bb9764d2a77903f65c710cb8a76d:
306f548aa9ac83be65e558aba4403264548d8d3c99ac04b94566ca661fd577e1:
ac4723f416075f4d63537f571e06ae375f4fda67cca1a256a8cedf4a336a9a90:
0c58d6b123a707480057b3f5a00d7f1d7589d7438126c63f5bbb6e4cdf198b71:
2174f477a3208b3dd8bc53e7a437a8c9a1e4e4aecf5179cddb6cd486828dbe79:
485d14543421fd69273a4f035a3eca56e20308dd0ebd8114610674dd630a83e5:
992ab5fa7b51f9284120b045c2474ca732bf238f7881b8b8e59a15e04c2079af:
605baf479f30fa7458184f4049c43e99eb8dff8d98b3dbc6b28e15d58c0dffd5:
1a144619fe2332a6e2d5e21027af6ceb584e7d1db8e09a023cd9dc769f4d1bde:
75ebe21467222b7fedbe2f92805c60320b0bc8370009b0d8579e18da8ec2e80c:
c78653d0c3dbbb80cbc21a44a8d09a0bd567539d11d08d4d655bd161f1dabe76:
0d39dd508c3047dcf9747ac1d706dc9777ca00f53b6622a0eefbc8157a325255:
bbf669a438ae64e8f9343dbbe7800102ceb7f7c4c72d87b4b21780f26325c624:
0b480750157231b72dceef3017f824d68aa17501fabc64b5d28dfac99aeab001:
c45f24b35b3487285d8616433fed2a9abd9a81c6c96fca38ac668125bafe2c4f:
a70bae834d892faa4a41ddd977eb0bfd19edd9676622a839193049ecc4e3db74:
47179996931a72f2128fd27d53bc149fa360f62bd1a763ffe2c24d9e783e6a27:
cc96eca2766da2154d24a9e3f47b0305e53cb60b76c859300708a88c8a182530:
83330d5c983d4acf0bbb7d9bbed9bfd1b686eb155c497c225ad25c99419a8fdf:
17bf79e84647feb377f09d9758e9cf33128c1b757d394723e928339bbe4728e4:
[s]tatus [p]ause [r]esume [b]ypass [c]heckpoint [q]uit =>
```

[redacted]

FIGURE 11.3d: Screenshot showing Hashcat during an attack. Retrieved by command `hashcat -m 1400 -a 3 rockyou.txt ?l?l?l?l?l?l?l?l?l?l`

The passwords are obscured for privacy of users.



```
Status...........: Exhausted
Hash.Type........: SHA256
Hash.Target......: rockyoufull.txt
Time.Started.....: Wed Jul  5 02:00:42 2017 (0 secs)
Time.Estimated...: Wed Jul  5 02:00:42 2017 (0 secs)
Input.Mask.......: ?l [1]
Input.Queue......: 1/1 (100.00%)
Speed.Dev.#1.....:        0 H/s (0.05ms)
Speed.Dev.#2.....:        0 H/s (0.00ms)
Speed.Dev.#*.....:        0 H/s
Recovered........: 23/14343644 (0.00%) Digests, 0/1 (0.00%) Salts
Recovered/Time...: CUR:N/A,N/A,N/A AVG:88021,5280612,126734693 (Min,Hour,Day)
Progress.........: 26/26 (100.00%)
Rejected.........: 0/26 (0.00%)
Restore.Point....: 0/1 (0.00%)
Candidates.#1....: x -> x
Candidates.#2....: [Generating]
HWMon.Dev.#1.....: Temp: 19c Util:100% Core: 745Mhz Mem:3004Mhz Lanes:16
HWMon.Dev.#2.....: Temp: 21c Util:100% Core: 745Mhz Mem:3004Mhz Lanes:16

Started: Wed Jul  5 01:59:43 2017
Stopped: Wed Jul  5 02:00:44 2017
[ball4222@gnode1008(arcus-b) ~]$
```

FIGURE 11.3e: Screenshot showing Hashcat after completion of an attack. Retrieved by command `hashcat -m 1400 -a 3 rockyou.txt ?l`

11.4 Break-down of attack results

Key

:	Do nothing to the dictionary word
l	dictionary words all lowercase
u	dictionary words all uppercase
c	dictionary words with first letter uppercase and the rest lowercase
?d	1 digit – 0123456789
?l	1 lowercase letter – abcdefghijklmnopqrstuvwxyz
?u	1 uppercase letter – ABCDEFGHIJKLMNOPQRSTUVWXYZ
?s	1 symbol – !”#$%&’()*+,-./:;<=>?@ [\]^_`{¦}~ or space
L7	7 lowercase letters – an abbreviation of ?l?l?l?l?l?l?l
D5	8 digits – an abbreviation of ?d?d?d?d?d
U6	6 uppercase letters – an abbreviation of ?u?u?u?u?u?u
S4	4 symbols – an abbreviation of ?s?s?s?s

FIGURE 11.4a Table showing a breakdown of all 142 of the tests run in the attack in section 5.1.

Attack	Cracked	Time	Left to crack	Total cracked	Percentage cracked
			14343644		
Brute-force 1 character, all	46	71	14343598	46	0.0003207
Brute-force 2 character, all	336	65	14343262	382	0.002663201
Brute-force 3 character, all	2475	70	14340787	2857	0.01991823
Brute-force 4 character, all	17961	113	14322826	20818	0.145137456
Brute-force 5 character, all	258952	761	14063874	279770	1.950480645
Brute-force 6 character, all	1947306	1230	12116568	2227076	15.52657051
English dictionary	265624	424	11850944	2492700	17.37842908
All other dictionaries	59973	289	11790971	2552673	17.79654459
All dicts + e to 3 (under :, l, u, c)	7098	60	11783873	2559771	17.84602992
All dicts + l to 1 (under :, l, u, c)	4486	58	11779387	2564257	17.8773051
All dicts + o to 0 (under :, l, u, c)	6067	57	11773320	2570324	17.91960258
All dicts + a to 4 (under :, l, u, c)	2314	57	11771006	2572638	17.93573516
All dicts + a to @ (under :, l, u, c)	1921	57	11769085	2574559	17.94912785
All dicts + s to $ (under :, l, u, c)	1253	56	11767832	2575812	17.95786343
All dicts + s to 5 (under :, l, u, c)	7196	56	11760636	2583008	18.00803199
All dicts + i to 1/g to 9/i to !/z to 2/b to 8/l to !	10976	58	11749660	2593984	18.08455369
Combinations of substitutions	5431	56	11744229	2599415	18.12241715
All dicts + ?d	331774	66	11412455	2931189	20.43545559
All dicts + ?s	42070	56	11370385	2973259	20.72875624
All dicts + ?d?d	842743	107	10527642	3816002	26.60413212
All dicts + ?s?s	8676	57	10518966	3824678	26.66461884
All dicts + ?d?s	8188	52	10510778	3832866	26.72170336
All dicts + ?s?d	21335	52	10489443	3854201	26.8704452
All dicts + ?d?d?d	753112	98	9736331	4607313	32.12093803
All dicts + ?s?s?s	4339	213	9731992	4611652	32.15118836

All dicts + ?d?d?d?d	1139936	136	8592056	5751588	40.09851332
All dicts + ?d?d?d?d?d	469337	497	8122719	6220925	43.37060373
All dicts + ?d?d?s	12137	57	8110582	6233062	43.45521961
All dicts + ?s?d?d	39349	60	8071233	6272411	43.72955018
All dicts + ?s?s?d	837	91	8070396	6273248	43.73538551
All dicts + ?d?s?s	1039	89	8069357	6274287	43.74262914
All dicts +l/u/c + ?d	58446	69	8010911	6332733	44.15009882
All dicts +l/u/c + ?s	6188	42	8004723	6338921	44.19323988
All dicts +l/u/c + ?d?d	91332	85	7913391	6430253	44.82998184
All dicts +l/u/c + ?s?s	1093	54	7912298	6431346	44.83760194
All dicts +l/u/c + ?d?s	1038	51	7911260	6432384	44.84483859
All dicts +l/u/c + ?s?d	2927	53	7908333	6435311	44.86524484
All dicts +l/u/c + ?d?d?d	41964	52	7866369	6477275	45.15780648
All dicts +l/u/c + ?s?s?s	537	532	7865832	6477812	45.1615503
All dicts +l/u/c + ?d?d?d?d	50363	382	7815469	6528175	45.51266749
All dicts +l/u/c + ?d?d?d?d?d	15039	1301	7800430	6543214	45.61751533
All dicts +l/u/c + ?d?d?s	1323	82	7799107	6544537	45.62673892
All dicts +l/u/c + ?s?d?d	3904	79	7795203	6548441	45.65395655
All dicts +l/u/c + ?s?s?d	87	178	7795116	6548528	45.65456309
All dicts +l/u/c + ?d?s?s	132	178	7794984	6548660	45.65548336
All dicts +l/u/c + ?s?d?d?d	15406	467	7779578	6564066	45.76288982
All dicts +l/u/c + ?d?d?d?s	7937	468	7771641	6572003	45.81822443
?d + all dicts (under :/l/u/c)	41800	44	7729841	6613803	46.10964271
?s + all dicts (under :/l/u/c)	5646	39	7724195	6619449	46.14900509
?d?d + all dicts (under :/l/u/c)	51254	43	7672941	6670703	46.50633409
?s?s + all dicts (under :/l/u/c)	821	60	7672120	6671524	46.51205788
?d?s + all dicts (under :/l/u/c)	1039	46	7671081	6672563	46.51930151
?s?d + all dicts (under :/l/u/c)	2404	46	7668677	6674967	46.53606155
?d?d?d + all dicts (under :/l/u/c)	45052	59	7623625	6720019	46.85015189
?s?s?s + all dicts (under :/l/u/c)	512	433	7623113	6720531	46.85372141
?d?d?d?d + all dicts (under :/l/u/c)	113606	207	7509507	6834137	47.64575167
?d?d?d?d?d + all dicts (under :/l/u/c)	155590	1786	7353917	6989727	48.73048299
?d?d?s + all dicts (under :/l/u/c)	2000	106	7351917	6991727	48.74442645
?s?d?d + all dicts (under :/l/u/c)	746	105	7351171	6992473	48.74962736
?s?s?d + all dicts (under :/l/u/c)	52	218	7351119	6992525	48.74998989
?d?s?s + all dicts (under :/l/u/c)	34	218	7351085	6992559	48.75022693
?s?d?d?d + all dicts (under :/l/u/c)	568	585	7350517	6993127	48.75418687
?d?d?d?s + all dicts (under :/l/u/c)	2386	577	7348131	6995513	48.77082142
Combination, 2 from English words	2447116	11160	4901015	9442629	65.8314512
Combination, 2 from passwords.txt	6532	50	4894483	9449161	65.87699053
Combination, 2 from MySpace.txt	5511	49	4888972	9454672	65.91541173
Combination, 2 from phpBB.txt	52640	111	4836332	9507312	66.28240355
Combination, 2 from 500worst.txt	5	44	4836327	9507317	66.28243841
Combination, 2 from English (l)	9338	271	4826989	9516655	66.34754042

Combination, 2 from English (u)	6346	208	4820643	9523001	66.39178301
Mask L7	26135	140	4794508	9549136	66.57398915
Mask L8	71549	310	4722959	9620685	67.07280939
Mask L9	88811	4602	4634148	9709496	67.69197562
Mask D7	0	45	4634148	9709496	67.69197562
Mask D8	0	45	4634148	9709496	67.69197562
Mask D9	89112	89	4545036	9798608	68.31324035
Mask D10	443869	273	4101167	10242477	71.40777476
Mask D11	102357	198	3998810	10344834	72.12138003
Mask D12	36629	989	3962181	10381463	72.3767475
Mask L6D1	35229	152	3926952	10416692	72.62235454
Mask L6D2	117127	330	3809825	10533819	73.43893225
Mask L7D1	56957	102	3752868	10590776	73.83602103
Mask L8D1	69423	761	3683445	10660199	74.32001938
Mask L5D2	42761	147	3640684	10702960	74.61813748
Mask L7D2	108843	382	3531841	10811803	75.37696139
Mask L4D4	27966	132	3503875	10839769	75.57193277
Mask L5D3	21928	63	3481947	10861697	75.72480884
Mask L3D5	0	65	3481947	10861697	75.72480884
Mask L2D6	19856	87	3462091	10881553	75.86323949
Mask L1D7	10209	114	3451882	10891762	75.93441388
Mask L6D3	41652	372	3410230	10933414	76.22480034
Mask L5D4	20713	75	3389517	10954127	76.36920576
Mask L4D3	10455	63	3379062	10964582	76.44209519
Mask D2L6	4470	54	3374592	10969052	76.47325882
Mask D1L7	6570	177	3368022	10975622	76.51906308
Mask D3L5	2586	118	3365436	10978208	76.53709197
Mask D4L4	3632	106	3361804	10981840	76.56241329
Mask D1L6	3621	105	3358183	10985461	76.58765792
Mask D2L5	1778	104	3356405	10987239	76.60005365
Mask D3L4	1257	104	3355148	10988496	76.60881712
Mask D4L3	0	90	3355148	10988496	76.60881712
Mask U7	26199	193	3328949	11014695	76.79146945
Mask U8	29458	267	3299491	11044153	76.99684264
Mask U9	22226	4738	3277265	11066379	77.15179629
Mask U6D1	6445	102	3270820	11072824	77.19672909
Mask U6D2	7231	79	3263589	11080055	77.24714166
Mask U7D1	7086	123	3256503	11087141	77.29654333
Mask U8D1	6625	93	3249878	11093766	77.34273104
Mask U5D2	3473	118	3246405	11097239	77.36694385
Mask U7D2	6286	762	3240119	11103525	77.41076814
Mask U4D4	1467	54	3238652	11104992	77.42099567
Mask U5D3	1151	55	3237501	11106143	77.42902013
Mask U3D5	0	52	3237501	11106143	77.42902013

Mask U2D6	1478	53	3236023	11107621	77.43932434
Mask U1D7	926	56	3235097	11108547	77.44578017
Mask U6D3	1574	364	3233523	11110121	77.45675367
Mask U5D4	856	104	3232667	11110977	77.46272147
Mask U4D3	729	51	3231938	11111706	77.46780386
Mask S7	162	153	3231776	11111868	77.46893328
Mask L8D2	116500	8100	3115276	11228368	78.28113972
Mask L9D1 - stopped	37957	3185	3077319	11266325	78.54576564
Combination password.txt + English	1570	54	3075749	11267895	78.55671125
Combination English + password.txt	5060	59	3070689	11272955	78.5919882
Combo English (l/u/c) + password.txt	9382	84	3061307	11282337	78.65739696
Mask S8	203	763	3061104	11282540	78.65881222
Mask L4D5	2473	64	3058631	11285013	78.67605331
Mask L6D4	36420	2676	3022211	11321433	78.92996368
Mask L5D5	2428	1172	3019783	11323861	78.94689104
Mask L4D6	22759	502	2997024	11346620	79.10556062
Mask L3D8	6410	129	2990614	11353030	79.15024941
Mask L5D6 - stopped	398	65	2990216	11353428	79.15302415
Mask D4L5	4187	212	2986029	11357615	79.18221478
Mask D6L4	7669	45	2978360	11365284	79.23568097
Mask D5L5	5073	89	2973287	11370357	79.27104856
Mask D4L6	2593	151	2970694	11372950	79.28912625
Mask U4D5	132	52	2970562	11373082	79.29004652
Mask U6D4	175	102	2970387	11373257	79.29126657
Masks U5D5 L3D7 L3D6 U3D8 U3D7 U3D6	47692	248	2922695	11420949	79.62376227
Mask L5D2S1	501	65	2922194	11421450	79.62725511
Mask L5S1D2	1690	64	2920504	11423140	79.63903733
Mask L10 - stopped	66265	10800	2854239	11489405	80.10101896
Total		72199			
		20:03:19			

11.5 Analysis of password policies

FIGURE 11.5a: Table detailing the password policies of the top 50 UK websites

Rank	Website	Policy		
		Min length	Character restrictions	Other information
1	google.co.uk	8	None	Advice given: a strong password contains a mix of numbers, letters and symbols. It is hard to guess, does not resemble a real word, and is only used for this account. 8 chars with 3 types of char is classed as 'Strong'
2	youtube.com	Same account as google (1)		
3	google.com	Same account as google (1)		
4	facebook.com	6	None	Password of all lowercase of ANY length and randomness is classed as weak. Compared to a password length 6 of all 4 character types is strong
5	reddit.com	6	None	Password of all lowercase of ANY length and randomness is classed as weak. Compared to a password length 6 of all 4 character types is 'okay'
6	bbc.co.uk	8	Min 1 letter & min 1 number or symbol	No measure of strength
7	amazon.co.uk	6	None	No measure of strength
8	wikipedia.org	1	None	No measure of strength
9	ebay.co.uk	6	2 different types of chars needed	Password length 6 of all 4 types is strong, password with 1U followed by any number of L is medium
10	twitter.com	6	None	All lower 6 (weak) 12 (medium) 16 (strong) 22(very strong) all 4 types of char length 6 very strong
11	live.com	8	None	No measure of strength
12	yahoo.com	9	None	No measure of strength
13	theladbible.com	No accounts possible		
14	linkedin.com	6	None	all lower 6 (3/5 strength) 11 (4/5 strength) 20 (5/5 strength) - passwords that pass cant be less than 3 though
15	diply.com	No accounts possible		
16	livejasmin.com	6	None	No measure of strength
17	instagram.com	6	None	No measure of strength
18	imgur.com	6	Min 1 letter & min 1 number	No measure of strength
19	t.co	Same account as twitter (10)		

20	theguardian.com	6	None	No measure of strength
21	netflix.com	4	None	No measure of strength
22	pornhub.com	6	None	No measure of strength
23	office.com	Same account as live (11)		
24	dailymail.co.uk	5	None	all lower 5 (weak) 10 (medium) 15 (strong) all diff char types 5 (medium) 6(strong)
25	microsoftonline.com	Same account as live (11)		
26	imdb.com	8	None	No measure of strength
27	paypal.com	8	1 number or symbol required	No measure of strength Can be all numbers or all symbols
28	twitch.tv	8	None	8 length lower is fair strength, longer strength depends on whether it contains words
29	tumblr.com	Same account as yahoo (12)		
30	ntd.tv	No accounts possible		
31	www.gov.uk	8	Min 1 number & min 1 letter	No measure of strength
32	givemesport.com	3	None	No measure of strength
33	amazon.com	6	None	No measure of strength
34	wikia.com	1	None	No measure of strength
35	thesportbible.com	No accounts possible		
36	wordpress.com	6	8 if just lower, and 8 if any word can be found	just disallows if It deems it weak but doesn't say why it is weak
37	rightmove.co.uk	6	None	No measure of strength
38	bing.com	Same account as live (11)		
39	telegraph.co.uk	8	Min 1 number & min 1 letter	No measure of strength
40	stackoverflow.com	8	Min 1 number & min 1 letter	No measure of strength
41	microsoft.com	Same account as live (11)		
42	gumtree.com	6	Min 1 letter & min 1 number or symbol	there's an indication of strength
43	xhamster.com	4	None	No measure of strength
44	msn.com	Same account as live (11)		

45	pinterest.com	6	None	No measure of strength
46	github.com	7	Min 1 letter & min 1 number	No measure of strength
47	apple.com	8	Min 1 lower, 1 upper and 1 number	there's an indication of strength
48	tripadvisor.co.uk	6	None	No measure of strength
49	lloydsbank.co.uk	8	Min 1 letter & min 1 number	No measure of strength
50	independent.co.uk	6	None	Any number of lowercase characters does not make the strength better - 60 lowercase is weaker than 5 lowercase plus 1 number