Brute-force and dictionary attack on hashed real-world passwords

L. Bošnjak*, J. Sreš* and B. Brumen*
* University of Maribor, Faculty of Electrical Engineering and Computer Science/Institute of Informatics, Maribor, Slovenia
leon.bosnjak@um.si, jure.sres@student.um.si, bostjan.brumen@um.si

Abstract - An information system is only as secure as its weakest point. In many information systems that remains to be the human factor, despite continuous attempts to educate the users about the importance of password security and enforcing password creation policies on them. Furthermore, not only do the average users' password creation and management habits remain more or less the same, but the password cracking tools, and more importantly, the computer hardware, keep improving as well. In this study, we performed a broad targeted attack combining several well-established cracking techniques, such as brute-force, dictionary, and hybrid attacks, on the passwords used by the students of a Slovenian university to access the online grading system. Our goal was to demonstrate how easy it is to crack most of the user-created passwords using simple and predictable patterns. To identify differences between them, we performed an analysis of the cracked and uncracked passwords and measured their strength. The results have shown that even a single low to mid-range modern GPU can crack over 95% of passwords in just few days, while a more dedicated system can crack all but the strongest 0.5% of them.

Keywords* – *password cracking; password security; brute force attack; dictionary attack; hashed passwords

I. INTRODUCTION

Password based authentication remains the most common way of granting access due to its simplicity, despite many well documented flaws. While progress has been made from the technical point of view, the weakest link remains the human factor. This was already pointed out nearly 40 years ago, when Morris and Thompson addressed the issue of UNIX password security, identifying numerous issues and proposing several countermeasures. They concluded that the algorithm (in that case DES), used in the encryption of plaintext passwords, was too fast, which made them prone to brute-force attacks. To put things into perspective, back then, cracking a 6 lowercase character password using an average CPU took 107 hours, while today it takes roughly 20 seconds. They also pointed out, that user-created passwords were not only short, but also predictable, and were thus likely to appear in dictionaries [1].

A decade later, a follow-up study by Feldmeier and Karn found out that password creation habits remained weak, while cracking methods and the algorithm speed improved. They concluded that in order to improve password security, password entropy should be increased. Several additional solutions were suggested, such as password meters, pre-assigned strong passwords and salts. An alternative login method in form of smart cards was proposed as well, but such method is plagued by another set of flaws [2].

Zviran and Haga observed that characteristics of user created passwords did not change in the internet era. Passwords are still weak and easy to guess, mainly because users keep employing predictable patterns and common words, such as names and birth dates. Furthermore, the level of data importance or sensitivity does not affect password composition. They concluded that the length, change frequency, and selection method are not related to writing down a password, while its composition is. A set of guidelines for selecting and implementing user-selected passwords and mechanisms that monitor their implementation was also proposed [3].

Klein went into details on what to put into dictionaries, suggesting permutations, keyboard patterns, names, changing character cases and covering foreign languages. Perhaps the most useful advice was using word pairs, although such dictionaries can become quite large. He used a relatively small, but diverse dictionary that contained phrases ranging from sport teams to names of asteroids. To improve security, smart cards and password checkers were suggested yet again [4].

For the past four decades, the battle against user habits and laziness has remained in full swing. With advances in technology, even longer passwords are becoming more vulnerable, the dictionaries grow larger each day and all precautions cannot deter users from making bad decisions. A study on password meters concluded that they increase password strength for the important accounts, while for the unimportant accounts they make no observable difference [5]. Additionally, password policies can be more of a help to the attacker, because they reveal the minimal search space. Given that the users most often try to satisfy rules with as little effort as possible and thus use predictable patterns, an adversary can prioritize such passwords in their attack while ignoring weaker ones. Another important aspect of password security is frequent password change, but despite educating the users about the problem, the majority still does not change their password once it has been set [6].

The aim of this study is to crack as many passwords used by the students to access the university's online grading system as possible, in order to demonstrate just how predictable and weak they really are. We were interested in the differences between the used cracking techniques, their success and time consumption. Finally, we performed the analysis and compared cracked and uncracked password characteristics and measured their strength with bits of entropy.

II. DATA ACQUISITION

185,643 passwords in MD5, SHA-1 and SHA-256 hashed form (and later in plaintext) were acquired, of which 151,136 were unique. They were created between 1980 and 2013 and were used to access the university's online grading system. Two default password patterns were identified. The main reason for duplicates are passwords, generated with the first pattern, consisting of two lowercase letters, derived from the students' first and last names, followed by four letters. The newer, improved pattern consisted of one uppercase, one lowercase letter, and six digits, all in random order. There were 154,505 (83.23%) default passwords, of which 144,250 (77.7%) matched the old pattern, and 10,255 (5.52%) the new one. Therefore, only 31,138 (16.77%) passwords were user-generated.

Previous works using this data found user habits to be in correlation with those described in [1], [2] and [3]. Password strength did somewhat improve over time, but unfortunately not enough to match the technology. Students who changed the default password mostly used short, simple, and predictable patterns [7]. An attempt to crack the passwords was made but was flawed in more than one way. Each attack mode was performed on the entire set of passwords, while the additional letters did not include 'đ' or 'ć', only Slovenian 'š', 'č' and 'ž'. Dictionary used was not adapted to suit the language used by the users but was instead build using large English wordlists. Lastly, only brute-force and dictionary attacks were performed, skipping attacks such as combinator or hybrid and rules [8].

Ethical concerns were addressed as well. Mentioned passwords are no longer in use, while additional data, such as gender, year of birth and course of study, had been anonymized by the university security service's personnel. The use of such personal data without prior or written consent of a subject (e.g. student) is allowed for the purpose of research under Article 11 (2), Article 13 (2), and Article 32 (3) of the Data Protection Directive [9].

III. EXECUTION OF THE ATTACK

User-generated passwords were predominantly created by Slovenian students, therefore some of them contained language-specific letters, that are considered special characters, such as "š" (used 181 times), "č" (used 160 times) and "ž" (used 74 times). Letters "ć" (used 13 times) and "đ" (used 4 times) were considered due to significant number of foreign students, mainly from former Yugoslav countries. In order to crack passwords that contained such letters, custom charsets were constructed.

All attacks were performed using hashcat v4.0.0 due to its flexibility and portability. In the first part of the experiment, a personal computer with a mid-range GPU, AMD Radeon R9 280X, 8GB of RAM (DDR3 1600, PC3 12800, 9-9-9-24, CAS9) and i5-4670k (@4GHz) with Windows 10 Pro (x64) OS was used. Benchmark hashrate for MD5 hash function using this system was approximately 8,900 MH/s. In the second part, we used a more dedicated system. With three nVidia GeForce Titan X Pascal, 64GB of RAM (DDR4 2400, PC4 19200, 16-16-16) and i7-6700k (@4GHz) with Linux Mint (x64) OS, the combined hashrate during benchmark reached approximately 88,500 MH/s.

All the executed attacks are gathered in Table I. They are grouped by the method of the approach and the results are also summed for each category. Sequence number denotes the order in which the attacks were executed, and the system column marks the computer used for the given attack. Some of the attacks were not completed and ran until they were interrupted. Few of the times are estimated due to the missing final report of the process, since they were paused and then resumed on another occasion. In the following chapters, each approach is described, and the more important results are singled out. It should also be pointed out, that most of the attacks could be combined during the more systematic approach.

A. Brute-force approach

Perhaps the most well-known, this attack mode heavily relies on the raw computing power instead of cleverness of the attacker. Cracking the old default pattern, for instance, took approximately 20 seconds for all 144,250 records (and some other passwords that matched the pattern), while the improved pattern was also later cracked in a single attack but took 10 hours and 20 minutes. This time could be significantly shorter if the attack covering all the possible arrangements of the letters and digits would be performed separately, because such detailed mask takes below one second to complete. Next, incremented attack was performed up to six characters in length, taking 2 minutes and 12 seconds and yielding 12,056 passwords. That was followed by the mask for digits only, from length seven to twelve characters, which took 3 minutes and 18 seconds and cracked 656 passwords. For further attacks on the remaining passwords, three custom charsets were constructed. First consisted of lowercase characters, including language specific letters mentioned earlier, digits and the most common special characters, while the second included uppercase letters as well. Third also contained letters from multiple less common foreign languages. All were missing a common special character asterisk ('*'), which later turned out to be a significant mistake, because the only two uncracked passwords left with the length of 7 contained said character, and there were 10 more in the longer uncracked passwords.

The first few attacks that cracked a large amount of the dataset provided an insight into user generated passwords and used patterns, which were later considered in the construction of the dictionaries. Weak passwords, like phone numbers, license plate numbers, birthdates and student IDs were all cracked as well.

TABLE I. ALL PERFORMED ATTACKS WITH UNIQUE PASSWORD COUNTS AND TIMES OF EXECUTION

Seq. no.	System	Attack	Count	%	Time
		ALL Attacks	**150,213**	**100.00**	**~35 hours 44 min**
		Brute-force	**147,401**	**98.13**	**~32 hours 21 min**
1	1	Old pattern (2 lowercase letters followed by 4 digits)	115,498	76.89	20 sec
2	1	Increment to length six (mixedalphaspecialnum)	12,056	8.03	2 min 12 sec
3	1	Digits only from length 7 to 12	656	0.44	3 min 17 sec
4	1	Increment to length 8 (lowercase custom charset)	7,094	4.72	22 min 6 sec
5	1	Length 9 and 10 (mixed custom charset, special pattern)	2,071	1.38	Aborted after ~30 min
6	1	Increment to length 9 (foreign languages charset)	36	0.03	Aborted after ~15 min
14	1	Improved pattern (mixedalphanum length 8)	9701	6.46	10 hours 20 min
16	2	Length 10 lowercase English alphabet	67	0.05	49 min 33 sec
17	2	Length 10 lowercase English alphabet and digits	222	0.15	~20 hours
		Dictionary	**284**	**0.19**	**2 sec**
8	1	674,096-word dictionary	248	0.17	1 sec
10	1	14,457,264-word dictionary	36	0.03	1 sec
		Combinator	**1,012**	**0.67**	**2 min 34 sec**
7	1	14,652-word dictionary combined with itself	202	0.13	1 sec
12	1	674,096-word dictionary combined with itself	810	0.54	2 min 33 sec
		Hybrid	**697**	**0.46**	**2 hours 20 min 6 sec**
9	1	14,652-word dictionary, length 4 mask, custom charset, both sides	73	0.05	6 min 40 sec
11	1	674,096-word dictionary, length 4 mask, custom charset, both sides	590	0.39	6 min 26 sec
14	2	2.15GB dictionary, incremented mask to length 5, both sides	34	0.02	2 hours 7 minutes
15	2	**Rules** (2.15GB dictionary, all included hashcat rules)	**127**	**0.08**	**50 min 13 sec**
13	1	**Online database**	**692**	**0.46**	**~10 min**

B. *Dictionary attacks*

A straightforward dictionary attack is limited to exact matches, but is still surprisingly successful, as users tend to choose simple and predictable passwords. As suggested in [3] and [4], used words included most common names and surnames from several languages, pet names, band names, car manufacturers, sports teams, famous people, and colors. These specific phrases were added to a large wordlist containing most common passwords in several languages, among others, obtained in several well-known leaks, forming a 14,457,264-word (134MB) dictionary for a straightforward attack. For the hybrid and combinator attacks, two more dictionaries were made. First contained 674,096 above mentioned common words from multiple languages (6.83MB), with leaked wordlists omitted. A second, smaller 14,652-word (92KB) dictionary contained the same common words (this time only Slovenian), all possible combinations of letters and numbers up to 4 characters (including full years) and special characters in order to build word pairs for use in other dictionary attack modes.

Hashcat enables concatenating two dictionaries on-the-fly, forming word pairs for the combinator attack. We avoided using two large wordlists because of processing power constraints. The main idea for building the previously mentioned smaller (674,096-word) dictionary was to construct such word pairs. By using this approach, some common patterns were covered, such as words followed or preceded by a year, or some other characters.

Another possibility when building a dictionary is to construct a single, larger wordlist by appending all words from one dictionary to all words in another. This newly constructed dictionary can then be used in any subsequent attacks. That allows us to cover passwords which contain combinations of three words by performing a combinator attack, or two words with additional characters before or after the passwords when executing a hybrid attack. The smallest (14,652-word), 92KB dictionary was combined with itself, resulting in a 2.15GB wordlist.

Dictionary attacks with the exception of larger hybrid approaches are much faster than the basic brute-force method. The majority of performed attach took only up to 15 minutes. Straight mode for instance, took only 1 second with the largest of the dictionaries, suggesting that a significantly larger dictionary could have also been used. Even the dictionary in the combinator attack could be larger when considering the time. Third approach using wordlists is hybrid with masks and therefore similar to brute-force method. Such attack takes existing wordlists and appends or prepends characters specified by the mask. With an option to increment said additions, this attack covers similar patterns as the combinator attack, but is more flexible, because it includes unusual diminutives and otherwise unpredictable strings.

C. Rule-based attack

Rules can be applied to words from dictionaries and are used with hybrid, combinator and straight dictionary attacks to provide even more options. They can be defined with a complex set of functions and some most popular are already included in hashcat. In our case, only the latter were used.

This attack was performed near the end, when only the strong password remained uncracked. By substituting certain letters with special characters or numbers (among many other rules), even some of the more clever user generated passwords were found. This method is fast even with large wordlists and can be performed in a matter of minutes or even seconds. However, in our case, the time of the combined attack was longer due to the number of rules and the complexity of a few of them. Some rules are more successful than others (such as those in d3ad0ne.rule and dive.rule), but in total they yield surprisingly good results.

D. Pre-computed wordlists

Perhaps the simplest approach is not to do any work but to simply look up cracked hashes. This method is also immune to calculation speeds of different hash functions. Online, freely available, precomputed wordlists and rainbow tables are huge, and they are growing each day. Among others, the 190GB, 15-billion-entry lookup table available on crackstation.net [10] or over 829.726 billion decrypted MD5 hashes on hashciler.co.uk [11] exist, with the latter also being used in our case. This method is easily prevented by using passwords in combination with salts, however no such procedure was used in this case, therefore enabling the use of this method. Unfortunately, such wordlists work best on passwords created by English speaking users due to language used in leaked databases.

IV. ANALYSIS

Out of the 185,643 passwords, 953 (0.51%) in total remained uncracked, out of which 923 (0.5%) are unique. This is a very unsettling result, especially considering the effort that was put into the attack. The number of cracked passwords could be even higher, if not for the mistakes described later in this chapter. Even the alarming number of uncracked passwords could be considered weak by most standards, although their measured strength was reasonably high.

We measured password strength with bits of entropy. Entropy is a measure of randomness of a system, and is one of the most common measures used in literature. [12-14] If, for instance, the passphrase is made of M symbols, each chosen at random from N possibilities, each equally likely, the entropy is $M*\log_2(N)$. Table II contains the values obtained for our set using the previously mentioned formula. Surprisingly, the password with the most bits of entropy was cracked, which is unusual at the first glance. Upon further inspection it became clear why: it consists of 29 lowercase letters ‘l’, making it the longest among all by 9 characters. It was obtained from the online database. Password with the most bits of entropy among the uncracked is “duiebvz fnikhrgh,l”, while the weakest is “ansm*63”. The latter remained uncracked due to the already mentioned mistake made while constructing the charsets. The weakest of all the passwords is a single zero with 3.3219 bits of entropy. Other passwords, containing only digits are extremely weak as well, since the search space is only ten characters.

Provided that the old, weak default pattern with the entropy of 32.1453 represents over three quarters of the passwords, the average for cracked passwords is skewed towards that value. Entropy of the improved default pattern is 49.3594 bits, making it better, but still not strong enough, given the fact that it was cracked in its entirety. A Mann-Whitney U test showed that the entropy of cracked passwords was significantly lower than the entropy of the uncracked passwords (U = 423,581, $p < 0.000$).

Another thing to consider while attempting to crack passwords stored in the hashed form is the used function. In our case, the original passwords were stored in the plaintext form, making the analysis of the uncracked passwords possible in the first place. Needless to say, this is the worst possible way of storing them. Another thing to add is that the uncracked passwords were only revealed after the attack was over. However, this bad practice also made it possible to compare the resilience of different hash functions, or in other words, their cracking times, as seen in Table III. Widely used MD5 function is among the fastest, but when it comes to cracking, that is all but desirable. Another very commonly used function is SHA-1. Despite being twice as slow by our measurements, it is just as unfit for storing passwords, as all passwords could still be cracked in less than a day. Third considered function was the bCrypt with a work factor 5. It is among the recommended functions when it comes to password hashing. The estimated time was on an entirely different level, as illustrated by the hashrates for each hash function while cracking the second pattern on a personal computer:

TABLE III. PASSWORD ENTROPY

Passwords	**Entropy**		
	Average (Std. Dev.)	***Min***	***Max***
All	33.3296 (6.8033)	3.3219	143.6717
Cracked	33.1495 (6.2913)	3.3219	143.6717
Uncracked	68.2320 (11.0076)	43.6017	118.5293

TABLE II. HASH FUNCTIONS ESTIMATED CRACKING TIME

Hash function	**Estimated time (desktop PC)**	
	Brute-force *(second pattern)*	***Dictionary*** *(6.83MB)*
MD5	10 hours 37 min	6 sec
SHA-1	21 hours 43 min	7 sec
bCrypt (5)	>10 years	308 days 18 hours

TABLE IV. UNIQUE CRACKED AND UNCRACKED PASSWORDS COMPARED BY PASSWORD LENGTH

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

- MD5: 5708.6 MH/s
- SHA-1: 2792.0 MH/s
- bCrypt: 4761 H/s

One more thing to mention is adding salt to a password before calculating the hash. Salt is a random string that is appended or prepended to a password and is usually stored in the database next to it. It should be long and not reused. When it comes to cracking, this approach prevents the use of lookup and rainbow tables, but the speed itself is not affected, due to hash function remaining unchanged and the salts being known.

Next, Table IV clearly shows the importance of the password length. If not for the previously mentioned mistake, every single password up to the eight characters in length would be cracked. A reasonable minimal length is at least 12 characters, provided that they are chosen randomly from a sufficiently large character pool. Additionally, Table V suggests, that using just lowercase letters and digits is only sufficient, when the length exceeds a certain threshold and the phrase is not predictable. Obviously it is not recommended, except in some specific cases, like in the method mentioned in conclusion.

On the other hand, uncracked passwords still contained many less common combinations of names and surnames, which exposes a flaw in the dictionaries. Results could be even better if the wordlists also contained a phone book or some similar larger list of names and surnames, since it appears that names are even more prevalent than previously thought. In cracked passwords, for instance, nine out of top ten base words are names (and the other is "nokia"). What's more, other common words were overlooked as well, such as dialect words, certain band names, less common car manufacturers or models and even movie titles. Those are generally already in the available dictionaries, but unfortunately mostly in English and therefore do not cross the language barriers. Nevertheless, a dictionary building method, described in [4] should be followed more closely.

With this in mind, the number of uncracked passwords could be even lower than it already is. Beforehand, a much higher password strength was expected, at least near the end of the attack. Because of that, some of the choices for attacks were a step in the wrong direction, like adding too many unused special characters in the character sets or including Greek letter passwords among others in the largest dictionary. After the surface examination of the uncracked passwords, an estimated 200 more passwords were somewhat weak and could be cracked with a better approach.

Sex of the users was considered as well, as seen in Table VI. In total, the ratio between male and female users is surprisingly even. A total of 91,902 (49.5%) passwords

TABLE V. UNIQUE CRACKED AND UNCRACKED PASSWORDS COMPARED BY CHARACTER SETS

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

TABLE VI. PASSWORDS COMPARED BY THE SEX OF THE USER

			Sex		Total
			Male	*Female*	
Cracked	Yes	*Count*	91,902	92,788	184,690
		% of total	49.50	49.98	99.49
	No	*Count*	593	360	953
		% of total	0.29	0.19	0.51
Total		*Count*	92,495	93,148	185,643
		% of total	48.83	50.17	100.00

by males and 92,788 (49.98%) passwords by females were cracked. Further, we performed a Mann-Whitney U test, and observed that the cracked passwords created by the male students had significantly higher entropy values than the passwords by female students (U = 4,227,514,906.5, p < 0.000). Among the uncracked passwords, the male users are better represented. Their passwords represent almost two thirds of the uncracked passwords compared to just a third for the female users. There was no significant difference in the entropy of uncracked passwords between the sexes, however (U = 102,806, p = 0.337).

Spearman's rank-order correlation test determined that there is a significant, albeit weak relationship between the year of the student's enrollment at the university, and the entropy of their password (r_s = 0.201, p < 0.000). This finding supports the conclusion that textual passwords are slowly improving over time, likely due to user education.

Finally, we divided our sample into five groups based on the students' field of study: humanistics, natural, social and technical studies, and transport. We discarded some of the older entries that did not include the field of study, bringing the total to 178,518 passwords. Then, we carried out the Kruskal-Wallis test, which showed that there are significant differences between the social studies and all the other groups (i.e. social-transport: $\chi^2(2)$ = -8,741.49, p < 0.000) in terms of password entropy, whereas there were no significant differences between any of the other groups. The mean rank for social studies was 32.56, followed by 33.93 for transport, 34.02 for humanistics, 34.037 and 34.044 for technical and natural sciences, respectively.

V. CONCLUSION

Even though additional sophisticated approaches could be employed, the results of the executed attack speak for themselves. Not only was the alarming amount of the passwords cracked, but the attack was also completed in a relatively short time period. Furthermore, the available wordlists grow larger by the day while the computers get faster, allowing for these attacks to become even easier over time.

Another key thing to remember is that these passwords were generated and used by the students, who are to some degree expected to be aware of security-related issues, and possess more cognitive ability than the average user. In other words, they should in theory pick stronger, more secure passwords. This is either not the case, or the passwords created by the average users are even weaker. Additionally, the age of the user and the year of the creation could also be considered in the analysis.

From the results it seems that the length is the main difference between the strong and weak passwords, so perhaps the solution to at least some textual password issues could be a less known method called Diceware. The goal is to create a long and hard to guess, yet secure passphrase using a random predefined 7776-word list available in multiple languages. Words are prepended with five numbers from one to six. Each word is then picked by rolling the dice five times. Number of words can vary, but the recommended minimum is six words, while for more security a length of up to nine words is advised. Each word in the passphrase yields 12.9 bits of entropy. This can be further increased by adding special characters in random word and random place by rolling the dice. The source also states, that a physical dice must be used due to true randomness [15]. Password policies could make picking such passphrases more complicated, because no uppercase letters or digits are used.

All things considered, it appears textual passwords are here to stay for foreseeable future due to their simplicity and widespread use, however ever-increasing computing power could raise the bar of reasonable security beyond what users would consider practical. While less practical but more secure alternatives like the graphical passwords or biometry could be the solution, the life of textual passwords could be extended if stronger hash functions such as bCrypt were more commonly employed to protect them.

REFERENCES

[1] R. Morris and K. Thompson, "Password Security: A Case History," *Commun. ACM*, vol. 22, no. 11, pp. 594–597, Nov. 1979.

[2] D. C. Feldmeier and P. R. Karn, "UNIX password security - ten years later," 9th Annual International Cryptology Conference on Advances in Cryptology, pp. 44-63, 1989.

[3] M. Zviran and W. J. Haga, "Password security: an empirical study," Journal of Management Information Systems, vol. 15, pp. 161–185, 1999.

[4] D. Klein, "Foiling the cracker: A survey of, and improvements to, password security," Proceedings of the 2nd USENIX Security Workshop, pp. 5-14, 1990.

[5] S. Egelman, A. Sotirakopoulos, I. Muslukhov, K. Beznosov, and C. Herley, "Does My Password Go Up to Eleven?: The Impact of Password Meters on Password Selection," in Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 2013, pp. 2379–2388.

[6] V. Taneski, M. Heričko and B. Brumen, "Impact of Security Education on Password Change," Information and Communication Technology, Electronics and Microelectronics (MIPRO) 38th International Convention, 2015, pp. 1351–1355.

[7] L. Bošnjak and B. Brumen, "What do students do with their assigned default passwords?," Information and Communication Technology, Electronics and Microelectronics (MIPRO) 39th International Convention, 2016, pp. 1430–1435.

[8] B. Brumen and T. Makari, "Resilience of students' passwords against attacks," Information and Communication Technology, Electronics and Microelectronics (MIPRO) 40th International Convention, 2017, pp. 1491–1495.

[9] "EUR-Lex," Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data, 1995, [Online]. Available: http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31995L0046:en:HTML, accessed: April 2016.

[10] CrackStation, [Online] Available: https://crackstation.net/, accessed December 2017

[11] HashKiller.co.uk, [Online] Available: https://hashkiller.co.uk/md5-decrypter.aspx, accessed December 2017

[12] M. L. Mazurek et al., "Measuring Password Guessability for an Entire University," ACM SIGSAC conference on Computer & Communications Security, 2013, pp. 173-186.

[13] P. G. Kelley, "Guess Again (and Again and Again): Measuring Password Strength by Simulating Password-Cracking Algorithms," IEEE Symposium on Security and Privacy, 2012, pp. 523-537.

[14] M. M. Taha, "On password strength measurements: Password entropy and password quality," International Conf. on Computing, Electrical and Electronics Engineering, 2013, pp. 497-501.

[15] Diceware, [Online] Available: http://world.std.com/~reinhold/diceware.html, accessed February 2018