Passfault: an Open Source Tool for Measuring Password Complexity and Strength

B. A. Rodrigues
EMC, Federal University of Goiás
Goiânia, GO, 74605-010, Brazil

J. R. B. Paiva
Federal Institute of Goiás
Goiânia, GO, 74055-100, Brazil

V. M. Gomes
EMC, Federal University of Goiás
Goiânia, GO, 74605-010, Brazil

C. Morris
OWASP
Saratoga Springs, UT, 84045, USA

W. P. Calixto
EMC, Federal University of Goiás
Goiânia, GO, 74605-010, Brazil

ABSTRACT

Most intelligent systems need a reliable authentication method in order to assure the security of sensitive information and devices, and vulnerable combinations of username and password make it easy for devices and accounts to be compromised by attackers. Big corporations have had their databases stolen, leaving millions of accounts compromised. Although many systems encourage users to select good passwords by enforcing policies, modern cracking techniques have proved such strategies to be insufficient. Along with easily guessable passwords, password reutilization is a common practice among average users, which might lead to a wide range of social engineering and privilege escalation techniques. An accurate evaluation of a password's resistance to being cracked has proved to be a critical challenge. This paper describes Passfault, which is an Open Source tool maintained by OWASP that poses as a promising method of making sure passwords will not be cracked. A solid mathematical foundation is presented, and Passfault's results over the famous password list RockYou are described. Then, the most popular password rules found in the list are used to simulate an attack against a recently leaked hash list, providing statistical insight about how corporations and users are managing their passwords.

Keywords: Acess Control and Authorization, Authentication, Cyber-crime, Information Security Culture, Password Strength, Password Complexity, Password Cracking.

1. INTRODUCTION

With topics such as Internet of Things (IoT), Cloud Computing and Social Networks playing an increasingly important role in the social and economic development of today's society, the theme of Information Security has received a lot of attention [1]. Any vulnerability that is inherent to a widely used technology deserves a lot of attention from both the scientific community and the general public.

A reliable authentication technique is crucial to most computer systems, being essential to assure the security of sensitive information and devices [2]. Although many types of authentication methods have been implemented [3]–[6], a character based pair of username and password is still the most widely used technique in the majority of systems. Vulnerable combinations of username and password make it easy for devices and accounts to be compromised by malicious players [7]–[9]. One example is the Hajime worm, which propagates itself in IoT devices by exploiting weak telnet credentials [10]. Along with easily guessable passwords, password reutilization is a common practice among average users [11], which might lead to a wide range of social engineering and privilege escalation techniques, such as the ones used in the Dropbox leak [12].

Cryptographic Hash Functions (CHF) are functions that take a password as input and generate a long and complex string of characters (hash). Most password authentication systems use CHFs to store password hashes in their databases. CHFs are unidirectional functions, which means the only way to obtain a specific hash is by providing its original password as input [13]. This allows the system to verify authentication attempts without keeping a record of what the password actually is. In the authentication process, the password typed by the user generates a hash that is compared to the hash stored in the database, and access is granted in case both values are equal [13].

In the context of Threat Modelling, the malicious subject that is trying to get hold of the password can be referred to as the attacker, or cracker. The system administrator or specialist that seeks to prevent the password from being compromised is referred to as the defender [14]. In order to get hold of a password, the cracker must find the sequence of characters whose CHF matches the hash stolen from the database. This process is called password cracking and there are specific tools able to make millions of guesses per second, such as John The Ripper [15], Cain and Abel [16] and Hashcat [17]. Hashcat is the most advanced and popular tool, making it possible to use OpenCL [18] based parallelization techniques on many hardware platforms, such as CPUs, GPUs, DSPs and FPGAs. Password cracking techniques have been widely researched, not only by the scientific community [9], [19], [20], but also by online password cracking communities and forums [21]–[23]. In 2016, more than 96% of the SHA1 hashes leaked from LinkedIn's databases [24] were cracked in less than five months after they were released online [25]. If passwords are simple enough, even robust CHFs like bcrypt are not unaffected by modern cracking techniques [26].

When users are creating a password, most of them tend to use a predictable pattern that can be easily remembered. That also makes the password easily guessable by a cracker. Some authors have tried to find patterns in the way users create their passwords. Weir et al. [19] used probabilistic context-free grammars as input to a cracking software. Bishop and Klein [9] searched for common patterns such as number of characters and words from dictionaries. Researchers have also tried to elaborate a reliable metric that is able to quantify how robust a password is against password cracking. Castelluccia et al. used Markov models to model password strength [20]. The standards defined by NIST [27], [28], propose character entropy as the metrics for password guessability. Kelley et al. [30] evaluates NIST's metrics as useful, although limited in most cases. Kelley et al. also proposes [29] and evaluates [30] as a metric the empirical entropy based on previously collected passwords, also with limited results.

In order to establish a general framework for quantifying a password's resistance against cracking techniques, Sahin, Lychev and Wagner [31] formalized the concepts of password strength and complexity with concrete mathematical definitions that emphasize their differences. These concepts are very important in the context of password cracking because they draw on the key insight that "an attacker success at cracking a password must be defined by its available computational resources, time, CHF, as well as the topology that bounds their search space" [31].

Computer systems need a reliable way of making sure their users' passwords are not vulnerable to being cracked. This paper describes a tool called Passfault [32]–[33], which is an Open Source project mantained by OWASP [34] and released under the Apache License 2.0 [35]. Passfault poses as an alternative metric for password complexity and strength, making use of the mathematical foundation provided by Sahin et al. [31] to give users and system administrators a better sense of security about their passwords. For this, two experiments were performed, providing statistical insights about Passfault's results over the famous leaked password list RockYou [36] and another recently leaked list, which is kept unnamed for ethical reasons. The paper is structured as follows: First, the mathematical concepts defined by Sahin et al. [31] are revised. Then, Passfault's implementation is discussed. The methodologies for the experiments are then described. Results are then presented and discussed, followed by the Conclusions section.

2. DEFINITIONS

This section revises the mathematical definitions established by Sahin et al. [31]. These definitions establish the mathematical foundation behind Passfault's implementation. Some definitions were simplified or adapted for this paper.

2.1. Alphabet

An alphabet α is a finite set of characters.

2.2. Password

A password p is a finite string of characters over an alphabet α. The profinite set $\mathbb{X}(\alpha)$ of all finite strings over α is defined as the set of all possible passwords over α.

2.3. Parsing

A parsing of a finite string σ is a partition of its constituent characters in α. For example, possible parsings of the string *psword1* are *psword|1, ps|word|1,* and *p|s|word|1.*

2.4. Rule

A rule, denoted by $\xi(\alpha, aux)$, is a function that takes as input a finite alphabet α together with some auxiliary information *aux*, and outputs a subset of $\mathbb{X}(\alpha)$.

The auxiliary information aux can be viewed as a logical formula that specifies the requirements the password must follow. For example, auxiliary information might be the minimal number of characters or the mandatory use of special characters.

One possible example consists of a rule ξ that takes only strings that belong to a dictionary of English words, α is the English alphabet and *aux* defines that the string must have at least eight characters. The strings *homework* and *explanation* are examples of strings that belong to the subset of $\mathbb{X}(\alpha)$ generated by $\xi(\alpha, aux)$.

2.5. Rule Chain

A rule chain is a rule defined by an ordered set of rules $\{\xi_1, \xi_2, \dots, \xi_n\}$ such that its output consists of the outputs $p_1, p_2, \dots, p_n$ merged together in the same order to form the string $p_1|p_2| \dots |p_n$. Note that a rule chain also complies with the definition of a rule.

For example, take rules ξ_a and ξ_b, alphabets α_a and α_b, and auxiliary information aux_a and aux_b. Rule ξ_a takes names of American citizens, α_a is the English alphabet and aux_a defines that the string must have only lower case characters. Rule ξ_b takes number sequences of possible dates (such as ddmmyy or mmdd), α_b is {0-9} and aux_b doesn't define any requirement. One possible output of ξ_a is *john*, one possible output of ξ_b is *073190*, and one possible output of the rule chain $\{\xi_a, \xi_b\}$ is *john073190.*

2.6. Complexity

The complexity of a password p is defined over some alphabet α, with respect to a finite set of rules $\Xi = \{\xi_1, \xi_2, \dots, \xi_k\}$. It is defined as the size of the smallest subset of $\mathbb{X}(\alpha)$ containing p that can be generated by some rule ξ_k over α, with any auxiliary inputs. If no rule in Ξ can generate p, then p's complexity is the cardinality of $\mathbb{X}(\alpha)$.

2.7. Strength

The strength of a password p is defined over some alphabet α, a cryptographic hash function F over α, p's hash $F(p)$, a time period T, and a description of the attacker A, which includes all of its computational resources, its rules and auxiliary information. The strength is defined in terms of either success or failure when $F(p)$ is subjected to a cracking experiment. The experiment consists of A using its rules and auxiliary information to generate candidate passwords c_k and their hashes $F(c_k)$, and it is over when either: a) A found some c_k such that $F(c_k) = F(p)$ or b) the time after the experiment starts exceeds T. Case a) means success in the experiment, and p is labeled as *weak*. Case b) means A was not able to recover the password, and p is then labeled as *strong*.

3. PASSFAULT'S IMPLEMENTATION

Passfault is an Open Source tool implemented in Java and released under the Apache License 2.0. Its development started in 2011 with the intent of making the complexity of passwords measurable and easily understood. This section briefly describes how Passfault works, relating the aspects of its implementation with the concepts presented in last section.

Passfault receives a password as input, as well as the CHF and the attacker's hardware (either the number of GPUs or the cracking speed) as optional information. It starts its execution by configuring several modules that will be responsible for finding rules in the password's structure. These modules do their analysis in parallel threads, which allows Passfault's execution to be practially instantaneous. In order to find rule chains, the modules analyse all possible parsings of the password. When the analysis is finished Passfault compares all the returned rule chains, trying to find the one with the smallest complexity. In case the user has also provided the CHF and attacker's hardware information, Passfault estimates how long it would take for the password to be cracked.

Table 1 summarizes most rules Passfault is able to find. Besides recognizing rules, Passfault also identifies the presence of upper and lower case characters, increasing the size of the reported search space accordingly. Note that the same rule might have different search spaces, such as Random Numbers and Random Latin Characters, or Exact Matches for different dictionaries.

TABLE 1
Rules in Passfault's Implementation

Rule	Description	Example
Dictionary Exact Match	Substring matches some dictionary's entry.	*john*
Dictionary Backwards Match	Substring matches the backwards version of some dictionary's entry.	*nhoj*
Dictionary Misspelling	Substring matches a misspelled version of some dictionary's entry.	*johm*
Dictionary Augmentation	Substring matches an augmented version of some dictionary's entry.	*jo.hn*
Dictionary l337 Substitution	Substring matches a l337 version of some dictionary's entry.	*j0hn*
Date Format	Substring has numbers in a date format.	*073190*
Random Characters	Substring is a sequence of random characters.	*a7mk0s*
Keyboard Patterns	Substring has a character sequence commonly found in keyboard layouts.	*zxcvbn*
Repeated Keys	Substring is a sequence of repeated characters.	*aaaa*
String Repetition	Substring repeats another substring previously found in the password.	*johnjohn*

The results are presented to the user in a structured way, listing each rule in the rule chain it found and their respective complexities, as well as the estimated cracking speed and time to crack for the optional CHF and number of GPUs.

Passfault's attacker's hardware option works under the assumption that the cracking speed grows linearly with the number of devices the attacker chooses to use in the cracking process. For example, one NVidia GRID K520 is able to calculate 423 million SHA1 hashes per second, while ten instances of the same device working together are able to calculate 4.23 billion SHA1 hashes per second.

It is important to emphasize that Passfault gives the smallest search space as the complexity of a password. That means that Passfault is evaluating the worst case scenario, where the attacker is going use the smallest rule chain that matches the password.

4. METHODOLOGY

4.1. Experiment 1

Experiment 1 was designed to evaluate the rule chains present in the famous password list RockYou. RockYou is a gaming service that had their databases hacked in 2009, and a list of usernames and plaint text passwords was exposed to the web. The list has 36,965,286 passwords, including repetitions, and a total of 14,344,391 different passwords and their frequencies were analysed.

Wordlists from 7 different languages were used to form dictionaries for the experiment. The languages were German, English, Spanish, French, Italian, Dutch and Portuguese. These wordlists are derived from several different sources, which are described in Appendix A. It is known that most languages follow the Zipf distribution, meaning the frequency of each word is inversely proportional to its rank in the frequency table. So for each language, two dictionaries were used: one accounting for the distribution's head (most frequent 80% of the words); and one accounting for the long tail of the word distribution.

Besides the language wordlists, dictionaries commonly used for password cracking were also used in the experiment. The list of dictionaries is described in Table 2. For simplicity, the language wordlists are not mentioned in the table. The US Names and Surnames lists also follow the Zipf distribution, and they were divided between head and long tail as well.

TABLE 2
Dictionaries

Dictionary	Description	Size
Pet Name List	List of pet names	400
500 worst passwords	List of 500 worst passwords according to the password research group WhatsMyPass [37].	500
10k worst passwords	List of 10,000 worst passwords according to OWASP's Project SecLists [38].	10,000
John the Ripper	Dictionary used by the password recovery tool John the Ripper.	3,545
Cain and Abel	Dictionary used by the password recovery tool Cain and Abel.	306,706
US Names Popular	List of the top 80% most popular american names, according to the US Social Security [39].	1,127
US Names Long Tail	Long tail of the american names, according to the US Social Security [39].	85,782
US Surnames Popular	List of the top 80% most popular american surames, according to the US Census of 2000 [40].	14,214
US Surnames Long Tail	Long tail of the american surnames, according to the US Census of 2000 [40]	137,457

4.2. Experiment 2

Experiment 2 was designed to evaluate the performance of the 20 most frequent rule chains found in last experiment when cracking a famous hash list leaked in 2016, which is kept unnamed for ethical reasons. The list contains a total of 176,751,930 SHA1 hashes, including repeated ones.

The cracking process was executed using Hashcat v3.10 on an NVidia GRID K520.

5. RESULTS

5.1. Experiment 1

The histogram of the rule chain complexities is plotted in Fig. 1. The average complexity of RockYou's passwords was 3.34x10^8, a relatively low value that could easily be cracked in a small amount of time for most CHFs in use today. The peak in the right side of the histogram is due to a high number of passwords corresponding to URLs (probably browser bookmarks that users decided to use as passwords), which is a rule Passfault was not able to identify.

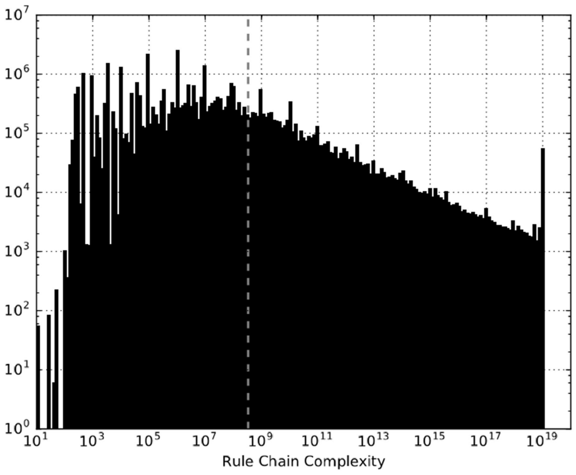

Fig. 1. Histogram of rule chain complexities in RockYou password list. Average Complexity is indicated by the gray dashed line.

A total of 848,645 different rule chains were found by Passfault in the RockYou list. These rule chains are permutations of 1,533 different isolated rules. Table 3 shows a list with the 20 most frequent rule chains found in the list. The Time to Crack column corresponds to the case where a SHA1 hash is attacked by one NVidia GRID K520 at 423.4 million hashes/s. These 20 rule chains were found in nearly 10 million passwords, which correspond to around 27.13% of the total 36.9 million passwords in the list. The total search space covered by these rule chains spans only 206.8 million different passwords, which means that for obsolete CHFs like SHA1, a good attacker could get the credentials of about one fourth of the users in a very short period of time.

The frequency rank of the rule chains is shown in Fig. 2. The rule chains seem to follow a Zipf distribution. One hypothesis to explain this behavior could be related to the fact that many passwords follow linguistic patterns, which are known to obey Zipf distributions.

If a password's length is small enough, such as 6 characters or less, its complexity is bound to relatively low values because even character based brute force attacks of this magnitude are feasible in short periods of time. The histogram of password lengths is plotted in Fig. 3. The average length in RockYou password list is 7.86 characters. A total of 9,970,733 passwords (26.97% of the entire list) are 6 characters long or less.

As described in Subsection 2.7, a password's strength depends on its complexity, the CHF used to generate its hash, a predefined time period, and the attacker's resources. If a time period of three months is considered, the CHF is SHA1, the attacker's hardware is one NVidia GRID K520, and the same rule chains that Passfault found in Experiment 1 are being used, then only 190,378 passwords (0.51% of the total) could be considered strong in the RockYou list.

TABLE 3
20 Most Frequent Rule Chains in RockYou Password List

Rule Chain	**Frequency**	**Complexity**	**Time to Crack**
Dictionary Match: John The Ripper	1,423,805	3,545	8.3727 µs
Date Format	1,098,073	930,000	2.1965 µs
Dictionary Match: 500 Worst Passwords	998,980	500	1.1809 µs
Dictionary Match: 10k Worst Passwords	939,639	10,000	23.6180 µs
Dicionary Match: US Names Popular	736,001	926	2.1871 µs
6 Random Numbers	510,875	1,000,000	2.3618 ms
Dicionary Match: Pet Name List	419,748	400	0.9447 µs
Dicionary Match: German Long Tail	412,485	97,212	0.2296 ms
Dicionary Match: John The Ripper — 2 Random Numbers	354,587	354,500	0.8373 ms
7 Random Numbers	351,678	10,000,000	23.6183 ms
6 English Keyboard Horizontal Characters	335,550	278	0.6567 µs
Dicionary Match: 500 Worst Passwords — 2 Random Numbers	319,673	50,000	0.1181 ms
Dicionary Match: 500 Worst Passwords — 1 Random Number	315,149	35,450	83.7270 µs
Date Format — 2 Random Numbers	314,824	93,000,000	219.6504 ms
Dictionary Match: US Names Popular — 2 Random Numbers	290,586	92,600	0.2187 ms
Dictionary Match: 10k Worst Passwords — 2 Random Numbers	286,861	1,000,000	2.3618 ms
Dictionary Match: Pet Name List — 2 Random Numbers	244,682	24,700	58.3372 µs
Dictionary Match: 10k Worst Passwords — 1 Random Number	237,737	100,000	0.2362 ms
8 Random Numbers	229,563	100,000,000	236.1833 ms
Dictionary Match: Italian Long Tail	210,625	97,292	0.2298 ms
Total	**10,031,121**	**206,797,403**	**488.4208 ms**

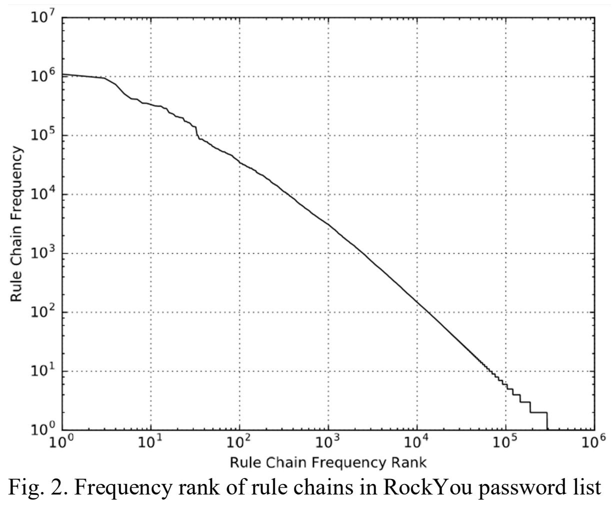

Fig. 2. Frequency rank of rule chains in RockYou password list

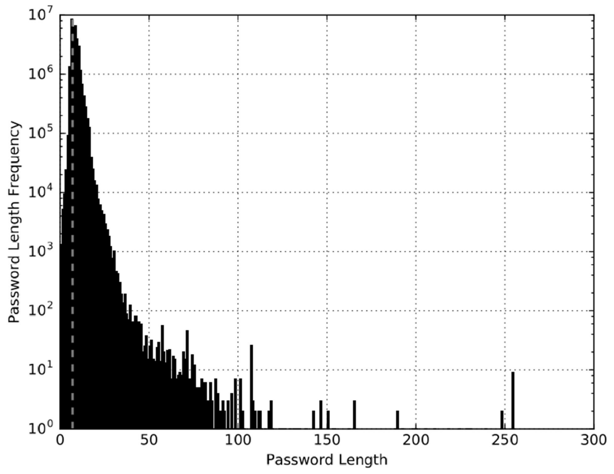

Fig. 3. Histogram of password lengths in RockYou password list. Average length is indicated by the gray dashed line.

5.2. Experiment 2

The 20 most frequent rule chains found in RockYou list were used to crack a list of 176.7 million SHA1 hashes, which include repetitions. The results are shown in Table 4. The hashes cracked by each rule chain also seemed to follow a Zipf Distribution, although not in the same order as in Experiment 1. A total of 44,896,578 hashes were cracked, accounting for around 25.40% of the whole list.

For each rule chain, the average time Hashcat spent was 3.5 minutes, with little variance. Most of this time was spent by Hashcat in setting up internal configurations, such as loading target hashes into memory and generating bitmap tables.

6. CONCLUSIONS

This paper described Passfault, an Open Source tool that measures password complexity and strength. A mathematical foundation was revised and Passfault's implementation aspects were discussed. Passfault's was tested over the famous password list RockYou, and the most frequent chain rules were used to perform an attack against a recently leaked SHA1 hash list.

Each day more corporations' databases are being hacked, exposing password hashes from millions of accounts. Entropy based password strength metrics are becoming inefficient against recent password cracking techniques. New metrics for estimating password robustness to such attacks, like the one proposed in the present paper, are essential to make sure accounts and devices are safe from unauthorized access.

The rule chains Passfault found in RockYou's list follow a Zipf distribution, which means small numbers of frequent rule chains account for major portions of the list. If these frequent rule chains have low complexities, a large number of passwords can be easily compromised, even when strong CHFs are used.

TABLE 4
Cracking SHA1 Hashes with Hashcat

Rule Chain	Frequency
Dictionary Match: John the Ripper	9,562,427
Date Format	1,947,585
Dictionary Match: 500 Worst Passwords	60,092
Dictionary Match: 10k Worst Passwords	4,672,643
Dictionary Match: US Names Popular	61,786
6 Random Numbers	6,661,096
Dictionary Match: Pet Name List	8,829
Dictionary Match: German Long Tail	3,572,018
Dictionary Match: John The Ripper — 2 Random Numbers	4,591,111
7 Random Numbers	1,525,511
6 English Keyboard Horizontal Characters	2,406
Dictionary Match: 500 Worst Passwords — 2 Random Numbers	28,377
Dictionary Match: 500 Worst Passwords — 1 Random Numbers	46,812
Date Format — 2 Random Numbers	146,016
Dictionary Match: US Names Popular — 2 Random Numbers	298,497
Dictionary Match: 10k Worst Passwords — 2 Random Numbers	2,017,542
Dictionary Match: Pet Name List — 2 Random Numbers	7,627
Dictionary Match: 10k Worst Passwords — 1 Random Number	1,041,068
8 Random Numbers	2,519,963
Dictionary Match: Italian Long Tail	1,490,744
Total	**44,896,578**

RockYou's rule chains show that most users tend to create low complexity passwords. If the RockYou password list was protected by SHA1 hashes, the attacker was using a relatively cheap GPU, and the cracking process took three months, then only less than 1% of the passwords would remain uncracked. This proves that a combination of high complexity passwords and good CHFs (like bcrypt) are essential to secure user data. Even though the distribution of the number of hashes cracked by each rule chain in Experiment 2 did not follow the same order as observed in Experiment 1, the percentage of cracked hashes was also near to 25% of the whole list. This emphasizes the fact that few popular rule chains account for big portions of the total passwords.

APPENDIX

The language wordlists used for Experiment 1 were obtained with help of the Python module called *wordfreq* [41], which is mantained by Luminoso Technologies, Inc. The module gathers information about word usage on different topics at different levels of formality, using data collected from the following sources:

- **LeedsIC**: The Leeds Internet Corpus.
- **SUBTLEX**: The SUBTLEX word frequency lists.
- **OpenSub**: Data derived from OpenSubtitles.
- **Twitter**: Messages sampled from Twitter's public stream.
- **Wikipedia**: The full text of Wikipedia in 2015.
- **Reddit**: The corpus of Reddit comments through May 2015.
- **CCrawl**: Text extracted from the Common Crawl and language-detected with cld2.

ACKNOWLEDGMENTS

The authors would like to thank Raymond Stone from Microsoft Corporation for his valuable help during this research.

Also, special thanks to CAPES for providing financial aid to this project.

REFERENCES

[1] J. Hutchens. **Kali Linux network scanning cookbook**, Birmingham, UK: Packt Pub., 2014.
[2] Cisar and S.Cisar. **Password - a Form of Authentication**, 5th International Symposium on Intelligent Systems and Informatics, 2007.
[3] R. Wildes. **Iris recognition: an emerging biometric technology**, Proceedings of the IEEE, v. 85, n. 9, p. 1348-1363, 1997.
[4] S. Singh and M. Yamini. **Voice based login authentication for Linux**, 2013 International Conference on Recent Trends in Information Technology (ICRTIT), 2013.
[5] R. Jayamaha et al. VoizLock, **Human Voice Authentication System using Hidden Markov Model**, 2008 4th International Conference on Information and Automation for Sustainability, 2008.
[6] F. Koushanfar Y. Zhang. **Robust privacy-preserving fingerprint authentication**, 2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), 2016.
[7] D. Florêncio and C. Herley. **A large-scale study of web password habits**, Proc. WWW'07, 2007.
[8] M. Dell'Amico, P. Michiardi, and Y. Roudier. **Password strength: An empirical analysis,** Proc. INFOCOM 2010, 2010.
[9] M. Bishop and D. V. Klein. **Improving system security via proactive password checking**, Computers and Security, vol. 14, no. 3, pp. 233-249, 1995.
[10] S. Edwards and I. Profetis. **Hajime: Analysis of a decentralized internet worm for IoT devices**, Rapidity Networks Security Research Group, 2016.
[11] S. Gaw and E. W. Felten. **Password management strategies for online accounts**, Proc. SOUPS, 2006.
[12][Online: 11/20/16] M. Lynley. **Dropbox employee's password reuse led to theft of 60M+ user credentials**, TechCrunch, 2016. https://techcrunch.com/2016/08/30/dropbox-employees-password-reuse-led-to-theft-of-60m-user-credentials
[13] P. C. van Oorschot, A. J. Menezes and S. A. Vanstone. **The Handbook of Applied Cryptography**, CRC Press, 5th edition, 2001.
[14] A. Shostack. **Threat modeling: designing for security**, Wiley; 1 edition, 2014.
[15][Online: 11/20/16] http://www.openwall.com/john/
[16][Online: 11/20/16] http://www.oxid.it/cain.html
.
[17] J. Steube. **Hashcat Advanced Password Recovery**, 2013.
[18] A. Munshi. **The OpenCL specification**, 2009 IEEE Hot Chips 21 Symposium (HCS), 2009.
[19] M. Weir, A. Sudhir, M. Breno, and G. Bill. **Password cracking using probabilistic context-free grammars**, Security and Privacy, 30th IEEE Symposium, 2009.
[20] C. Castelluccia, M. Durmuth, and D. Perito. **Adaptive Password-Strength meters from markov models**, Proc. NDSS 2012, 2012.
[21][Online: 11/20/16] https://hashcat.net/forum/
[22][Online: 11/20/16] http://crackingforum.com/
[23][Online: 11/20/16] https://www.webcracking.com/.
[24][Online: 11/20/16] J. M. Gosney. **How LinkedIn's password sloppiness hurts us all**, Ars Technica, 2016. http://arstechnica.com/security/2016/06/how-linkedins-password-sloppiness-hurts-us-all/
[25][Online: 11/20/16] https://hashes.org/public.php.
[26][Online: 11/20/16] T. Fox-Brewster. **Why You Shouldn't Panic About Dropbox Leaking 68 Million Passwords**, Forbes, 2016. http://www.forbes.com/sites/thomasbrewster/2016/08/31/dropbox-hacked-but-its-not-that-bad
[27] W. E. Burr et al. **Electronic authentication guideline**, Technical report, nov 2013.
[28] W. E. Burr et al. **Electronic authentication guideline**, NIST Special Publication, 2004.
[29] P. Kelley et al. **Encountering stronger password requirements: user attitudes and behaviors**, Proc. SOUPS '10, 2010.
[30] P. Kelley et al. **Guess Again (and Again and Again): Measuring Password Strength by Simulating Password-Cracking Algorithms**, 2012 IEEE Symposium on Security and Privacy, 2012.
[31] C. S. Sahin, R. Lychev and N. Wagner. **General Framework for Evaluating Password Complexity and Strength**, arXiv:1512.05814v1 [cs.CR], 2015
[32][Online: 11/20/16] http://www.passfault.com/
[33][Online: 11/20/16] https://github.com/OWASP/passfault
[34][Online: 11/20/16] https://www.owasp.org/
[35] L. Rosen. **Open source licensing**, Vol. 692. Prentice Hall, 2005.
[36] M. Weir et al. **Testing metrics for password creation policies by attacking large sets of revealed passwords**, Proceedings of the 17th ACM conference on Computer and communications security - CCS '10, 2010.
[37][Online: 11/20/16] http://www.whatsmypass.com/
[38][Online: 11/20/16] https://www.owasp.org/index.php/Projects/OWASP_SecLists_Project
[39][Online: 11/20/16] https://www.ssa.gov/oact/babynames/limits.html
[40][Online: 11/20/16] http://www.census.gov/topics/population/genealogy/data/2000_surnames.html
[41][Online: 11/20/16] https://github.com/LuminosoInsight/wordfreq