Making Secure Easy-to-Remember Passwords

Philip Braunstein

December 14, 2015

Abstract

Password leaks are a notorious way that systems get compromised. Passwords are often leaked exactly because they are hard to remember, which prompts people to record them in insecure locations and avoid changing them. This report describes and evaluates three password generation strategies: random strings of characters, numbers, and symbols (RS); abbreviations of phrases mixed with meaningful numbers (MPM); and an adjective-noun-verb-adjective-noun string modeled after xkcd 936 (MX) [1]. Passwords were evaluated on ease of memorization, cryptographic strength against a password cracker, and bits of entropy. MPM passwords were easiest to remember by woefully insecure in terms of entropy per password as well as vulnerability to a password cracker. MX and MPM passwords were both sufficiently cryptographically secure. MX passwords were easy for participants to remember, and participants made quick learning gains when MX passwords were incorrectly remembered. This report recommends using MX passwords, as they are both cryptographically secure and easy to remember.

Introduction

People like to imagine that computer security is usually compromised by genius hackers exploiting inscrutable vulnerabilities. Often however, an attacker compromises a system through social engineering [2] or because of seemingly stupid reasons like the password being written on a note next to the computer. Shoring up technical security is a worthy cause, but this effort is rendered irrelevant unless the impact of password leaks is minimized.

People avoid changing their passwords, leave them written in plain text in a document on their computer or even on a sticky note on their computer for one reason only: secure passwords are hard to remember. Insecure and poorly-stored passwords are the cause of many security leaks [3]. For example, there was the case of the French TV network whose password was leaked because the password was on a sticky note on the wall in the background of a televised interview [4]. People have even been shown to share their passwords with others [5]. Passwords are also a common source of vulnerability because different websites have different

requirements for what they consider valid passwords [6], and passwords that are considered secure are obtuse and hard for people to remember [7].

In this report, three methods for making passwords are described and evaluated. The security and how easy each type of password to remember is evaluated and described in the Results section.

To the Community

I am particularly interested in passwords because passwords are at the crossroads of humans and computers. Humans and computers have varying strengths and weaknesses, which makes choosing the right passwords tricky. Passwords need to be secure enough so they don't get easily cracked with a tool like Hashcat, but they also need to be possible for people to remember. Passwords that are too simple get cracked by a program such as Hashcat, and passwords that are too complicated get hacked because people leave them written in obvious places or never change them. You can find all of the scripts for this project here: `https://github.com/pbraunstein/Security-Final-Project`

Methods

Overall Description of Procedure

Each of the password generation methods described below were tested on a total of 7 subjects. In order to prevent bias based on subjects getting used to the memorization task, the script `determineOrder.py` was used to dictate the order that each of the password methods was tested. For the random string and modified XKCD password style, participants were shown 10 options for passwords and were permitted to chose the preferred password. Participants were told there were no length requirements for the MPM password, and they were not informed of the length of the RS and MX passwords.

For each password method, participants chose the password they wanted to use, and participants were not permitted to write the password down. Participants were then instructed to focus on something else. Most went back to work, and some chatted with me during that time. When I was speaking with a participant, I tried to steer the conversation away from the topic of the password that they chose. After five minutes, participants were asked to write their password down. If they wrote the password down exactly correctly, they were informed that they remembered their password perfectly. If there was a mistake in the password, they were either told what was amiss with their guess (e.g. "the correct password is `casualpiebecomesonlyprofit` instead of `otherpiebecomesonlyprofit` or they were shown the correct password for another couple of seconds. Participants worked or chatted with me for another five minutes, and then wrote down their passwords again. Passwords were tested one at a time. It was at no time during the experiment necessary for participants to remember more than one password. As an example, Participant 1 first only

had to remember the RS password, then only the MX password, and then only the MPM password.

Learning gains were evaluated for each password type as follows: for each password type, the similarity of the first trial was subtracted from the similarity of the password trial. However, only those passwords, in which the first trial similarities were less than 1 were counted.

Error bars were not included for any of the figures in the experiment because 7 participants is not a large enough sample size. In order to get estimates on error, a larger study should be conducted.

Password Generation Schemes

Random String (RS)

A random string password consists of ten random characters, numbers, and symbols. A random string password must contain at least one of three of the four categories: uppercase letter, lowercase letter, number, or symbol. The script `randomPass.py` generates ten of these random strings passwords. Participants selected one of the ten passwords to use.

Memorable Phrase and Number (MPM)

Participants chose a memorable phrase and made an acronym of the first letter of each word. Every initial was lowercase in the password. They also chose a particular number and incorporated it after the initials of the memorable phrase. For example, I might choose the hook from a famous Rolling Stones song (**I** **c**an't **g**et **n**o **s**atisfaction, and the month and year of my graduation from college (May, 2012) to generate the following password: icgns0512.

Modified XKCD (MX)

Randall Munroe of XKCD suggests using several common words that can be used to create a more coherent memory as opposed to other password generation methods. I have improved on his idea by forcing passwords to obey the form adjective-noun-verb-adjective-noun, where the words from these parts of speech are drawn from publicly available word lists [8–10].

The script `humanPass.py` creates ten of these modified XKCD passwords, from which the participant chose one. The passwords were written with all lower case letters and no spaces between the words. Participants were allowed to alter the password to fix any grammatical mistakes. For example one password generated by `humanPass.py` was `availablenervebitemoleculararradish` (available nerve bite molecular radish). Participants were instructed to modify the passwords so that they make grammatical sense, but the exact modification was left to the discretion of the participant. For example, the previous password could be modified in one of two ways `availablenervebitesmoleculararradish` (available nerve bites molecular radish) or `availablenervesbitemoleculararradish` (available nerves bite molecular radish). Participants were encouraged to make whichever grammatical modification made the most sense to them.

Password Evaluation

The passwords were evaluated on three criteria: the success of a participant remembering the password, the strength of the password against password cracking, and the number of bits of entropy.

Success Remembering Password

At first glance, the success of a participant remembering a password should be measured by the percent similarity of the remembered password and the actual password. This could be calculated by comparing each character in the two passwords to see how similar they are. This method, however, is insufficient. Consider the following example:

```
0123456789
023456789
```

These two passwords result in a similarity of 0.1 since the '1' is omitted in the second password. However, nine of ten of the characters were successfully remembered. Therefore, this simple metric of similarity underrepresents how similar the two passwords actually are. In short, a simple percent similarity metric does not account for character insertions or deletions in the sequences.

To compensate for this problem, the script `smartFracSim.py` calculates the fraction similar of the two passwords from the forward direction *and* from the reverse direction and averages these values (hereafter: smart fraction similar). This raises the fraction similar of the example to 0.45 $(0.1 + 0.8/2.0)$.

While the smart fraction similar is a better metric than fraction similar of only one direction, this metric was still underrepresenting the similarity in the passwords. Consider the modified xkcd example password from above:

`availablenervebitesmolecularradish` (available nerve bites molecular radish)

Consider what would happen if a participant swapped the adjectives to make the following password:

`molecularnervebitesavailableradish` (molecular nerve bites available radish).

The original and adjective-swapped passwords have a smart fraction similar of 0.47. This significantly underrepresents how close the participant was to correctly remembering the password. The participant simply flipped the adjectives, and might score a 1.0 similarity if asked again. In order to compensate for this, the MX passwords were also analyzed for similarity using the fraction of the original words that were also present in the remembered password, regardless of the grammatical form (for example, `nerve` would count when matched with `nerves`). This word similarity

metric was averaged with the smart fraction similarity metric to compute the final similarity for the MX passwords. In the swapped adjectives example, the word fraction similarity is 1.0 (all words are present). Averaged with the smart fraction similarity, the final similarity fraction for this example is 0.74. The word fraction was determined by hand (by me), since it would have been difficult to write a script to recognize the varying valid grammatical forms of a word as well as where to break the password string into separate words.

The word fraction similarity metric is successful because it accounts for distinct parts of the password that a participant has successfully remembered. There is no exact corresponding metric for the RS and MPM passwords. So, the fraction of the same characters between the original password and remembered password was determined for these styles of password using the script `charFracSim.py`. As above, this metric is averaged with the smart fraction similar to calculate the final similarity for random string and memorable phrase and number passwords.

Strength Against Password Cracking

The MD5 hash of the passwords was determined using built in Python hashing functionality with the script `makeHashes.py`. A distribution of Kali Linux running in a virtual machine on my Mac was used for the password cracking. 2 GB of RAM was allocated to the virtual machine. The mask attack of the program Hashcat [11] was used for cracking the RS and the MPM passwords. The mask attack is a brute-force attack in which the user supplies a mask which describes the type of characters that should be attempted at each position where %d signifies a digit, %l signifies a lower-case letter, and %a signifies all possible characters. The mask makes the brute-force attack more efficient and more targeted.

The complete invocation of Hashcat to attempt to break RS password hashes was:

```
time /usr/bin/hashcat --hash 0 --attack-mode 3 --pw-min 10 hashes.txt
%a%a%a%a%a%a%a%a%a%a
```

where `--hash 0` indicates that MD5 without salt was used to has the passwords, `--attack-mode 3` indicates a mask attack should be used, `--pw-min` indicates the smallest length password guessed should be 10 (so Hashcat didn't increment through shorter length password guesses), `hashes.txt` is the file containing the hashes, and `%a%a%a%a%a%a%a%a%a%a` is the mask that indicates each of the 10 positions could be any character (equivalent to the deprecated brute-force attack).

The MPM passwords have the predictable structure of a number of lower case letters followed by a couple of digits. Several runs of Hashcat with masks corresponding to the structure of every MPM password created by the participants of this study. The masks used were `%l%l%l%l%d%d`, `%l%l%l%l%d%d%d%d`, `%l%l%l%l%l%d%d`, and `%l%l%l%l%l%d%d%d%d`. The standard Unix function `time` was used to record the user time it took to run each cracking attempt.

Password cracking was not attempted on the hashes of the MX passwords because these passwords are far too long to be cracked with a mask attack. Calculations to approximate resistance to hacking attacks of the MX passwords is described in the results section.

Bits of Entropy

The bits of entropy for each password was determined by calculating the number of possible passwords for a password of that style of that length. The entropy of the RS passwords (H_{RS}) was determined by the following formula:

$$H_{RS} = 94^{10}$$

in which there were 94 possibilities for each character and the overall length of the password was 10 characters. The 94 possibilities came from the following Python statement:

```
len(strings.punctuation + strings.digits + strings.letters)
```

The entropy $H_{MPM}(p)$ of a MPM password p was determined as follows:

$$H_{MPM}(p) = 26^l 10^d$$

where l is the number of lowercase letters in the MPM password and d is the number of digits in the password.

The entropy $H_{MX}(p)$ of a MX password p was determined as follows:

$$H_{MX}(p) = 26^{len(p)}$$

MX passwords only contained lowercase letters, which is the derivation of the 26 possibilities per character.

Results

MPM are Easiest to Remember

Of the 7 participants tested in the experiment, every single one of them remembered their MPM passwords perfectly for both trials of the experiment. MX passwords were remembered perfectly for every participant's second time being asked. A clear hierarchy of how easy each style password to remember emerges from these experiments: MPM passwords are the easiest to remember followed shortly by MX passwords, but the RS passwords are much harder to remember (Table 1 and Figure 1). Notice that RS passwords are clearly much more difficult to remember than either MPM or MX passwords (Figure 1).

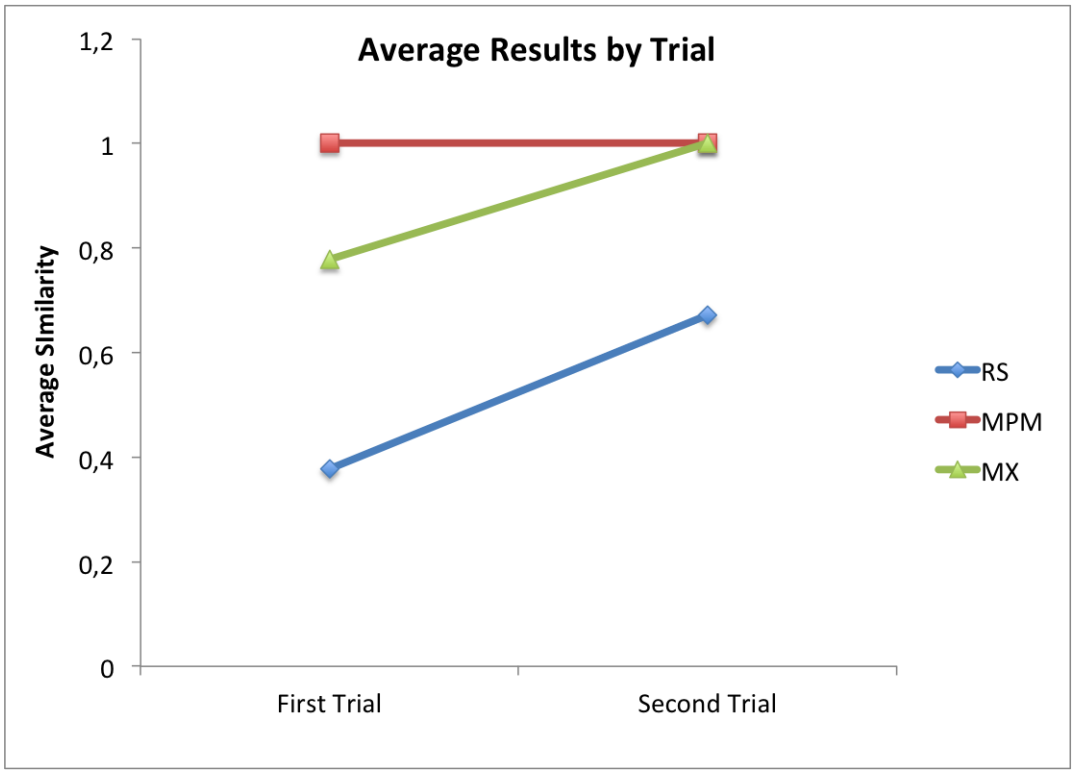

Figure 1: Average similarity by trial and password type

Password Type	First Trial	Second Trial
RS	0.34	0.67
MPM	1.0	1.0
MX	0.78	1.0

Table 1: Average similarity by trial and password type

MX Passwords are Easier to Learn than RS Passwords

Figure 2 shows that of the passwords that required learning (i.e. the participant did not remember the password perfectly the first time asked), MX passwords are easier to learn than the RS passwords.

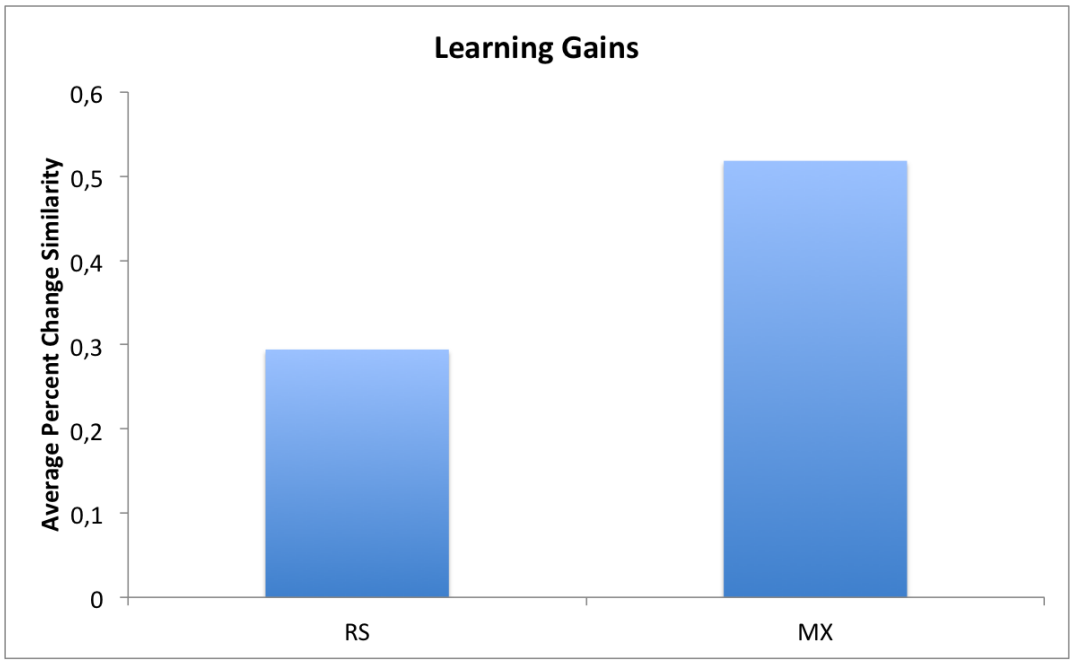

Figure 2: Learning gains for RS and MX

MX Passwords Have by Far the Highest Entropy

The average entropy and average log_{10} entropy for each password style is displayed in Table 2. Figure 3 and Table 3 shows the incredible gains of entropy seen by using MX passwords. MX passwords have on average 1.1×10^{36} times more entropy than RS passwords, and MX passwords have on average 1.2×10^{44} times more entropy than MPM passwords. MPM passwords had the lowest entropy.

Password Type	Entropy	log_{10} Entropy
RS	5.4×10^{19}	19
MPM	4.6×10^{11}	11
MX	5.7×10^{55}	56

Table 2: Average Entropy by password generation method

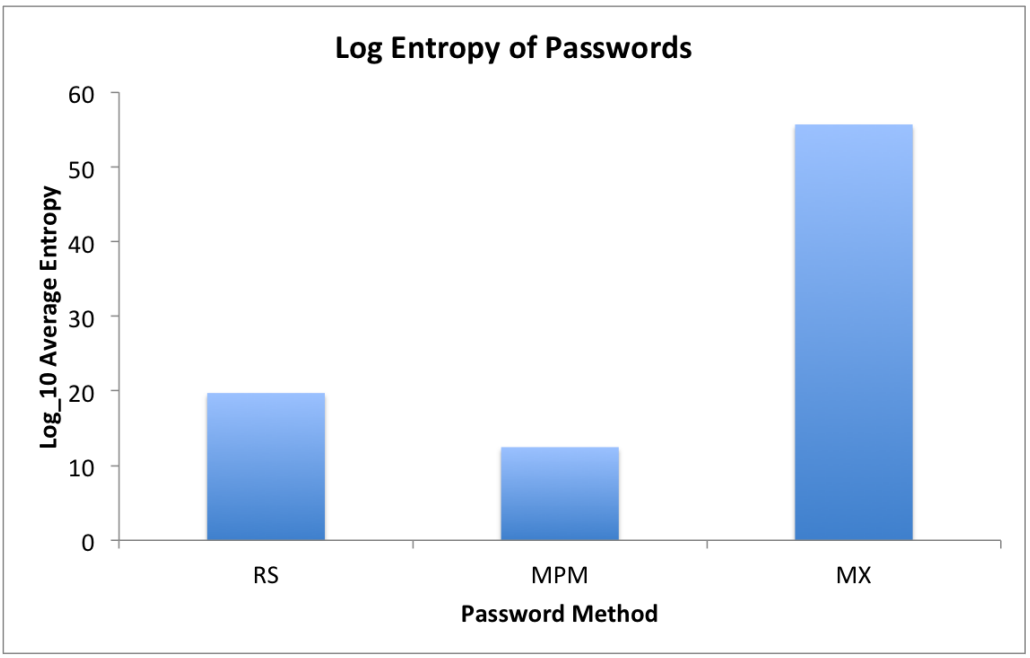

Figure 3: Log 10 average entropy of passwords by generation method

	RS	MPM	MX
RS	1	8.6×10^{-9}	1.1×10^{36}
MPM	1.2×10^{8}	1	1.2×10^{44}
MX	9.5×10^{-37}	8.2×10^{-45}	1

Table 3: Entry (x, y) is $\frac{\bar{H}_x}{\bar{H}_y}$ where $\bar{H}$ is average entropy

MPM Passwords are Woefully Insecure Against Password Crackers

Not only were the hashes of six of the seven MPM passwords cracked using Hashcat, but also these hashes were all cracked in under 10 minutes. The shortest of these passwords was cracked in only six seconds (Table 4).

RS Passwords and MX Passwords are Secure Against Password Crackers

It was not possible to use Hashcat's mask attack with RS passwords or MX passwords. The former because Hashcat predicted it would take more than 10 years to search the space, and the latter because Hashcat does not allow masks long enough to represent the MX passwords since searching the space of a mask 20-50 characters long is computationally intractable.

Mask	User Time (dd:hh:mm:ss)	Passwords Cracked
`%l%l%l%l%d%d`	6	MPM-3,MPM-5
`%l%l%l%l%d%d%d%d`	9:33	MPM-1,MPM-2
`%l%l%l%l%l%d%d`	2:41	MPM-6,MPM7
`%l%l%l%l%l%d%d%d%d`	5 days (pred.)	-
`%a%a%a%a%a%a%a%a%a%a`	$>$ 10 years (pred.)	-

Table 4: Hashcat user time by mask

Math approximating the feasibility of password cracking on RS and MX passwords was used in lieu of an actual password cracking run. From observing Hashcat running on my system, I have noticed that it generally runs at a speed of between 6.5 million to 7.5 million passwords attempted per second. In order to get a pessimistic lower-bound on the RS and MX passwords, 8 million passwords per second attempted is the speed used in the following calculations.

RS Passwords

The calculation for the amount of time to search the RS password space with a mask attack is shown below. Note that the number of passwords to search is the same as the entropy for the RS password.

$$5.4 \times 10^{19}\ passes \times \frac{sec.}{8.0 \times 10^{6}\ passes} \times \frac{year}{3.1 \times 10^{7}\ sec.} = 2.2 \times 10^{5}\ years$$

Thus it would take 220,000 years to search the space of RS passwords with a mask attack using Hashcat on my system.

MX Passwords

Using the same calculation as for the RS passwords yields:

$$5.7 \times 10^{55}\ passes \times \frac{sec.}{8.0 \times 10^{6}\ passes} \times \frac{year}{3.1 \times 10^{7}\ sec.} = 2.3 \times 10^{41}\ years$$

Thus using a mask attack, it would take 2.3×10^{41} years to search the space. This is a period trillions of trillions of ... (etc.) years longer than the age of the universe. While this number certainly excludes the possibility of using a mask attack to break the MX passwords, it is not the most accurate measurement security against a password cracker. This is because the letters in the MX passwords form coherent words, and are thus related to each other. In other words, the letter at position 1 in a password biases what letters could be at position 2. Note that this is not the case for the RS passwords.

A more appropriate style of attack to attempt to break the MX passwords would be a dictionary attack. Let us make a very generous assumption, and suppose the attacker had a list of all the words that could be put into the passwords (3893) as well as the knowledge that each password would be composed of exactly 5 words. In this scenario, the attacker *does*

not know that the passwords follow the pattern described in the Methods of this report (i.e. adj-noun-verb-adj-noun). In this case, the entropy is 3893^5 (there are 3893 options at each of the 5 word-positions in the password). In this case, the entire formula is multiplied by two because there are usually two ways to fix the grammar of an MX password (see Methods):

$$2 \times 3893^5 \ passes \times \frac{sec.}{8.0 \times 10^6 \ passes} \times \frac{year}{3.1 \times 10^7 \ sec.} = 7,200 \ years$$

Assuming the attacker new the exact set of words is a very generous assumption, and it would still require 7,200 years for the attacker to search the space with a dictionary attack of the MX passwords.

Taking this one step further, let us give the attacker one more piece of information. In this case, in addition to knowing the exact set of words, the attacker would also have the words segregated by type of word (i.e. a list of verbs, a list of adjectives, and a list of nouns). We will also give the attacker the knowledge of the structure of the password as adjective noun verb adjective noun. There are 1,466 adjectives; 2,327 nouns; and 100 verbs used to make the MX passwords. Thus, the number of passwords to search is $1466 \times 2327 \times 100 \times 1466 \times 2327 = 1.2 \times 10^{15}$:

$$2 \times 1.2 \times 10^{15} \ passes \times \frac{sec.}{8.0 \times 10^6 \ passes} \times \frac{year}{3.1 \times 10^7 \ sec.} = 9.4 \ years$$

Even under these extremely generous circumstances, it would still take 9.4 years for an attacker to crack the MX passwords.

Applications

This report demonstrates that RS and MX passwords are both acceptably secure against password crackers, but MX passwords are much easier to remember. Websites should not force users to come up with passwords with such draconian rules as "must contain three of the following: uppercase letter, lowercase letter, number, or digit." Rather, websites should allow users to choose a password consisting of only lowercase letters, as long as this password is sufficiently long. The impetus of change rests with those running websites, rather than with users of websites.

If users were allowed to use MX passwords rather than RS passwords, they would not forget their passwords as easily. Thus, users would be less likely to commit the cardinal sin of computer security of leaving passwords written in plain text on their computer. Furthermore, users wouldn't be so averse to changing their passwords since it is not so challenging to memorize an MX password.

Future Experiments

Future experiments should expand this study in two ways. First, more participants should be included so that the data is large enough to calculate statistical significance of the results. Second, the amount of time

between passwords should be expanded to more accurately simulate real password remembering conditions. With no research funding, it would not have been possible for me to ask for several hours of a participant's time, but more realistic conditions include asking participants for their passwords hours or even days later - these time frames more accurately reflect how passwords are actually used.

Conclusion

This report shows first and foremost that MPM passwords are woefully insecure (Table 4). Six of the seven MPM passwords were successfully cracked using Hashcat. The password cracking in this report underestimates the password cracking power an attacker would have. This is because an attack could dedicate more RAM to the VM on which the password cracker was running. Or, the attacker could have used Cudacat, which greatly improves the efficiency of Hashcat using GPU programming [12]. In general, the times reported in Table 4 underestimate the power of a password cracker.

This report also demonstrates that according to the entropy of passwords and the security of passwords against mask and dictionary attacks, both MX and RS passwords are acceptably secure (Figure 3, Table 3); however, the case can be made that RS passwords are more resistant to password crackers than MX passwords. Finally, the report demonstrates that of the two, MX passwords are much easier to remember than RS passwords (Figure 1). In the cases in which people did not remember the password perfectly on the first try, participants learned MX passwords much more effectively than RS passwords (Figure 2).

References

[1] Munroe, R. Password Strength. `https://xkcd.com/936/` (accessed December 10, 2015).

[2] Orgill, G. The urgency of effective user privacy-education to counter social engineering attacks on secure computer systems. CITC5 '04 Proceedings of the 5th conference on Information technology education, pages 177-181, 2004.

[3] Malenkovich, S. 10 Worst Password Ideas (As Seen In The Adobe Hack). `https://blog.kaspersky.com/10-worst-password-ideas-as-seen-in-the-adobe-hack/3198/` (accessed December 10, 2015).

[4] Machkovech, S. Hacked French network exposed its own passwords during TV interview. `http://arstechnica.com/security/2015/04/hacked-french-network-exposed-its-own-passwords-during-tv-interview/` (accessed December 10, 2015).

[5] Data Leakage Worldwide: Common Risks and Mistakes Employees Make. `http://www.cisco.com/c/en/us/solutions/collateral/enterprise-networks/data-loss-prevention/white_paper_c11-499060.html` (accessed December 10, 2015).

[6] Komanduri S. et al. Of Passwords and People: Measuring the Effect of Password-Composition Policies. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pages 2595-2604, 2011.

[7] Adams A. et al. Making Passwords Secure and Usable. People and Computers XII, pages 1-19, 1997.

[8] Quintans, D. The Great Noun List. `http://www.desiquintans.com/nounlist` (accessed December 10, 2015).

[9] Raveendranathan, P. adjectives1.txt. `https://www.d.umn.edu/~rave0029/research/adjectives1.txt` (accessed December 10, 2015).

[10] List of 100 Common Verbs. `http://stickyball.net/grammar/86.html` (accessed December 10, 2015).

[11] hashcat advanced password recovery. `http://hashcat.net/oclhashcat/` (accessed December 10, 2015).

[12] Automated password cracking with AWS GPU instances and CUDA hashcat. https://github.com/pabloav/aws-cudahashcat-auto (accessed December 10, 2015).

Appendix A: Generated Passwords by Password Type and Participant ID

Type-ID	Password
RS-1	`f,K~ym:}7j`
RS-2	`\fRRWMM>2,`
RS-3	`T\YJKwj3Ab`
RS-4	`Mhm@ujAPOZ`
RS-5	`"?za\HNy\n`
RS-6	`m!WA,SJBPZ`
RS-7	`);\U)dn>3^`

Table 5: RS passwords

Type-ID	Password
MPM-1	tmwt1988
MPM-2	hbty1205
MPM-3	mait10
MPM-4	mciagb5722
MPM-5	sapf11
MPM-6	tirwh31
MPM-7	tcsam16

Table 6: MPM passwords

Type-ID	Password
MX-1	nastyknightschangepoliteanteater
MX-2	dulldollarshaveshortscience
MX-3	englishrulerunloudwitness
MX-4	intelligentballoonsharessillyviolin
MX-5	casualpiebecomesonlyprofit
MX-6	plannedaardvarkdrinksbloodyriverbed
MX-7	unemployedshoemakersignorechangingviolin

Table 7: MX passwords