Seminar: Evaluating and designing authentication

Daniel Bosk Lennart Franked

19th March 2020

Abstract

A lot of user authentication is based on passwords. We use password policies to aid users in selecting a secure password. Unfortunately, research has shown that the common password-polices do not have the expected effect: users can still choose easy-to-guess passwords and the policies actually makes guessing easier. It is thus important to *scientifically* evaluate the actual effects of any user-authentication mechanism, otherwise our security might be at risk. Here we will focus on exactly that. More specifically, after this lab you should be able to

- *evaluate* the effective security by considering security and usability.
- *analyse* research results in usable security and *apply* those relevant to a given situation.
- *design* security policies aligned with usability.

To do this, we must be familiar with several topics: usability [And08, Ch. 2], cryptography [And08, Ch. 5] [Bos16], information theory [Uel] and the scientific method [PB07]. The main contents is some research papers on password security and usability: 'Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms' [Kel+12], 'Of passwords and people: Measuring the effect of password-composition policies' [Kom+11], 'Can long passwords be secure and usable?' [Sha+14] and 'The Password Life Cycle' [SB18]; complemented by a paper on the usability of password managers: 'A comparative usability evaluation of traditional password managers' [KSC10].

1 Introduction

User authentication is present in most systems. There is one security mechanism which can be found almost everywhere, which is intended to solve this problem: passwords. From a usability perspective, passwords generally perform very poorly. This will, of course, also yield security implications.

Usually, in systems using passwords, the password selection of the users are governed by some password-composition policy to help (or force) the users to select strong passwords. Thus, how these policies are designed has great impact on the resulting passwords the users choose. This impact is not always what is expected, in fact, sometimes a password policy can result in weaker passwords.

There is no indication that passwords will be replaced any time soon, so if we must use passwords, we would better use them well. This is the goal of this assignment.

2 Assignment

The seminar is divided into two seminar sessions. In the first (section 2.1), we will evaluate how password-composition policies, both their security and their usability. In the second (section 2.2), we will evaluate how users cope with passwords and evaluate the usability of password managers.

You will read research papers for these sessions. While reading, write down your thoughts. Let these questions guide your reading:

- What are the main results of the research paper?
- How did they conclude them? I.e., what is the research method?

Discuss your thoughts in the working groups before each seminar. The groups at the seminars will be randomly chosen to maximize exchanges of ideas.

2.1 Password-composition policies

Before the seminar session First you must read the chapter 'Usability and Psychology' [And08, Ch. 2]. Further, you need a basic understanding of information theory, for this you are recommended to read 'Chapter 6: Shannon entropy' [Uel].

Start by estimating the strength of the following two password-composition policies:

- At least eight characters chosen uniformly randomly. The characters include lower and upper case, digits and special characters.
- Four words chosen uniformly randomly. Say the words are Swedish, then there are approximately 125 000 words in SAOL (the common dictionary of Swedish).
- Add to the first policy the requirement of at least one character from each character group.

How strong are they? Order them according to their strengths.

Read the papers 'Of passwords and people: Measuring the effect of password-composition policies' [Kom+11], 'Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms' [Kel+12], 'Can long passwords be secure and usable?' [Sha+14] and, finally, 'The Password Life Cycle' [SB18]. In these papers the authors studied how password-composition policies affects the interplay between password security and usability and how users cope with passwords. Focus on these questions:

- What are the main results of the research paper?
- How did they conclude them? I.e., what is the research method?

After reading, reflect on what you have read, think about the following questions:

- How do these results compare to your experience of what is used in practice?
- How are your password strategies compared to those in the papers?

- What would be a good password policy? Why would that be good?
- Is it fine to write down passwords or not? It depends?
- How to recover from failed states; e.g., forgotten passwords, lost keys?
- What problems do you perceive with authentication of users in general?
- In which situations are passwords suitable and in which are they not?

Crack the following three passwords available in the two files:

`https://github.com/OpenSecEd/passwd/releases/download/v1.1/unix-passwd.txt`

and

`https://github.com/OpenSecEd/passwd/releases/download/v1.1/win-pwd.txt`

There are instructions for how to get going with this in appendix A. (You should work in your groups and make this as fast as possible.)

Discuss the papers and your reflections in the group, how hard are passwords to crack? For the papers, try to answer: what questions does each paper try to answer, how do they do this?

During the seminar session During the seminar we will first let the groups summarize their discussions from before the seminar: the papers, the reflections and experience from password cracking. (We will take 30 minutes for these summaries.)

After a short break, each group will design a password policy for one of the following:

- BankID and Mobile BankID,
- a web-mail account,
- a company login account,
- an encrypted hard-drive;

drawing from both the papers and the discussions. (We allocate 20 minutes for this.) Finally, every group will present their policy and analysis, then we will evaluate it together. (We spend the last 20 minutes on this.)

2.2 Password managers and other tools

Before the seminar session Read the paper 'A comparative usability evaluation of traditional password managers' [KSC10]. In this paper the authors examines the usability of three password managers and compares them. Discuss the paper in the group: what questions does the paper answer, how do they do this?

The group should then evaluate one password manager. Discuss in the group what interesting factors to look at. There are numerous password managers, some of the most popular are e.g.:

- KeePass{X,XC},
- LastPass,
- 1Password,
- LessPass,
- BitWarden,
- PassBolt,
- Password Safe.

(Remember, the group only needs to evaluate one.)

Then the group should also evaluate the following authentication mechanisms:

- (Mobile) BankID[1],
- WebAuthn[2] and
- IRMA[3].

After this, think of a few services, tools or devices that you frequently use and where you must authenticate. Think about the authentication in those situations: are they properly designed, how would you like to change them and why?

During the seminar session During the second part of the seminar we will discuss your experiences, thoughts and suggestions for improvements. We will do this in groups (randomly chosen to spread the ideas).

First, we will discuss (in groups) your experience of the password managers and how those relate to the results of the paper [KSC10]. How did you 'evaluate' the password manager? What were your conclusions? (The groups will have 30 minutes to discuss this.)

We will summarize the discussions and, particularly, differences between groups in full class. (We will spend 15 minutes on this.)

After a short break, we will discuss the advantages and disadvantages of the other authentication mechanisms that you tried. How did you 'evaluate' them, what did you do, how was that experience? What service did you want to improve, how and why? (The groups will have 30 minutes for discussions.) Then each group will summarize the most important parts of their discussion in class: any changes of mind, diverging opinions? (We allocate 15 minutes for this.)

3 Examination

To pass this assignment you need to come prepared and actively participate throughout the activities.

[1]URL: `https://www.bankid.com`.

[2]URL: `https://webauthn.io`. Note that you might have to use a smartphone as those have more hardware support than most laptops.

[3]URL: `https://irma.app`.

References

[And08] Ross J. Anderson. *Security Engineering. A guide to building dependable distributed systems.* 2nd ed. Indianapolis, IN: Wiley, 2008. ISBN: 978-0-470-06852-6 (hbk.) URL: `http://www.cl.cam.ac.uk/~rja14/book.html`.

[Bos16] Daniel Bosk. 'A high-level overview of cryptography'. Lecture. 2016. URL: `https://github.com/OpenSecEd/appliedcrypto/releases/tag/v1.1`.

[Kel+12] Patrick Gage Kelley, Saranga Komanduri, Michelle L Mazurek, Richard Shay, Timothy Vidas, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor and Julio Lopez. 'Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms'. In: *Security and Privacy (SP), 2012 IEEE Symposium on.* IEEE. 2012, pp. 523–537. DOI: `10.1109/SP.2012.38`.

[Kom+11] Saranga Komanduri, Richard Shay, Patrick Gage Kelley, Michelle L. Mazurek, Lujo Bauer, Christin Nicolas, Lorrie Faith Cranor and Serge Egelman. 'Of passwords and people: Measuring the effect of password-composition policies'. In: *CHI.* 2011. URL: `http://cups.cs.cmu.edu/rshay/pubs/passwords_and_people2011.pdf`.

[KSC10] Ambarish Karole, Nitesh Saxena and Nicolas Christin. 'A comparative usability evaluation of traditional password managers'. In: *International Conference on Information Security and Cryptology.* Springer. 2010, pp. 233–251.

[PB07] Sean Peisert and Matt Bishop. 'How to Design Computer Security Experiments'. In: *Fifth World Conference on Information Security Education: Proceedings of the IFIP TC11 WG 11.8, WISE 5, 19 to 21 June 2007, United States Military Academy, West Point, New York, USA.* Ed. by Lynn Futcher and Ronald Dodge. Boston, MA: Springer US, 2007, pp. 141–148. ISBN: 978-0-387-73269-5. DOI: `10.1007/978-0-387-73269-5_19`. URL: `http://web.cs.ucdavis.edu/~peisert/research/Peisert-WISE2007-SecurityExperiments.pdf`.

[SB18] Elizabeth Stobert and Robert Biddle. 'The Password Life Cycle'. In: *ACM Trans. Priv. Secur.* 21.3 (Apr. 2018), 13:1–13:32. ISSN: 2471-2566. DOI: `10.1145/3183341`.

[Sha+14] Richard Shay, Saranga Komanduri, Adam L Durity, Phillip Seyoung Huh, Michelle L Mazurek, Sean M Segreti, Blase Ur, Lujo Bauer, Nicolas Christin and Lorrie Faith Cranor. 'Can long passwords be secure and usable?' In: *Proceedings of the 32nd annual ACM conference on Human factors in computing systems.* ACM. 2014, pp. 2927–2936. URL: `http://lorrie.cranor.org/pubs/longpass-chi2014.pdf`.

[Uel] Daniel Ueltschi. 'Chapter 6: Shannon entropy'. URL: `http://www.ueltschi.org/teaching/chapShannon.pdf`.

A How to crack passwords

Cracking programs The papers above used some password-cracking software. In addition, on the website

`http://sectools.org/tag/crackers/`

you can find a list of programs for password cracking. You are free to use any program to solve this, however, there is a Docker image available to make things easy.

The Windows hash is an old NTLM hash, which means that it is not salted[4]. The UNIX hash is salted and uses Blowfish (OpenBSD).

How to obtain the password hashes For a UNIX-like operating system the password hashes with corresponding salts are stored in the file '/etc/master.passwd' on BSD-based systems such as OpenBSD and FreeBSD. In the case of Linux-based systems such as Ubuntu, the file used is '/etc/shadow'. You need privileges (root) to read this file.

The hashes on a Windows system can be acquired by the program fgdump. This is available from URL

`http://www.foofus.net/~fizzgig/fgdump/`.

The hashes in this assignment are already extracted from these files for your convenience. You are going to find the passwords for both Windows and UNIX-like systems. Thus, *you do not have to use any program like fgdump or unshadow(8) to extract them.*

A.1 A Docker image

If you have Docker on your computer you can use the container provided here. It has John the Ripper and Ophcrack preinstalled. To use it, create a directory on your computer where you have all the files you would like to have available. Then start the container from that directory.

```
docker run -it -v $(pwd):/pwdeval dbosk/pwdeval
```

This will map the current directory to the working directory (`/pwdeval`) inside the container. The first time you run it, it will automatically download the image from Docker Hub.

A.2 Instructions for Ophcrack and John the Ripper

Ophcrack The ophcrack(1) program uses a technique called rainbow tables. What this means is that all password and hash-value combinations are precomputed and stored in a huge table. This is called a hash table. The rainbow table is a special case of hash table, the benefit is that it is smaller than a conventional hash table. The hash table reduces the problem of cracking the password to searching this huge table.

The alternative approach is to compute the hash value for each guess, this takes time and this time is what is saved by using a hash table. However, this

[4]Consider this when choosing your method for cracking.

comes with some compromises, the hash tables (and even rainbow tables) requires a lot of computational resources to produce. They also requires great resources to use, they must preferably fit in the computers primary memory. Hence the use of hash tables and rainbow tables is a trade-off between computational and storage resources.

Because of the space limitations of this method it can easily be countered by adding a salt to the hash. This means that the rainbow table must increase too much in size to be feasible. Unfortunately, some Windows hashes are not salted, so this method can be used on those hashes (at least in some cases). UNIX-like systems has a longer tradition of using salts, so this method is not feasible on those hashes.

You will find ophcrack(1) in the package manager of most UNIX-like systems. You can also find it on URL

`http://ophcrack.sourceforge.net/`.

You also need a few rainbow tables to be able to use the program. You can find these on the website above. Choose your tables carefully.

John the Ripper John the Ripper is a terminal-based program using many different ways of cracking passwords. It has the possibility of brute-force attacks, dictionary attacks, and the possibility of using rules to modify the words in the dictionary (e.g., 'leet-speak'). Naturally, these methods takes much longer time to use than a rainbow table, since all computations are done in real-time.

The program can be found in the package manager of most UNIX-like systems, or on URL

`http://www.openwall.com/john/`.

You are recommended to use the 'Community Enhanced Version'.

To have a short summary of the possible arguments to pass to John the Ripper, just run the command 'john' in the terminal without any arguments. See listing 1. You can also read the manual page john(1).

The most interesting arguments are

- –show,
- –wordlist,
- –rules, and
- –incremental=all.

Note that '–wordlist' and '–stdin' are separate arguments. The first reads words from a file (provided a filename) while the latter reads words from standard input. You can read more about this in the manual by typing 'man 1 john' in the terminal.

You will find links to different wordlists to use in the following URL:

`http://sectools.org/tag/crackers/`.

Choose your wordlists with care. You also have the script 'pwdstream.py' (appendix A.3) to help generate a stream of passwords, see './pwdstream.py -h' for details.

Listing 1: Output from John the Ripper in the terminal.

```
$ john
John the Ripper password cracker, version 1.7.8
Copyright (c) 1996-2011 by Solar Designer
Homepage: http://www.openwall.com/john/

Usage: john [OPTIONS] [PASSWORD-FILES]
--single                  "single crack" mode
--wordlist=FILE --stdin   wordlist mode, read words from FILE or stdin
--rules                   enable word mangling rules for wordlist mode
--incremental[=MODE]      "incremental" mode [using section MODE]
--external=MODE           external mode or word filter
--stdout[=LENGTH]         just output candidate passwords [cut at LENGTH]
--restore[=NAME]          restore an interrupted session [called NAME]
--session=NAME            give a new session the NAME
--status[=NAME]           print status of a session [called NAME]
--make-charset=FILE       make a charset, FILE will be overwritten
--show                    show cracked passwords
--test[=TIME]             run tests and benchmarks for TIME seconds each
--users=[-]LOGIN|UID[,..] [do not] load this (these) user(s) only
--groups=[-]GID[,..]      load users [not] of this (these) group(s) only
--shells=[-]SHELL[,..]    load users with[out] this (these) shell(s) only
--salts=[-]COUNT          load salts with[out] at least COUNT passwords only
--format=NAME             force hash type NAME: DES/BSDI/MD5/BF/AFS/LM/crypt
--save-memory=LEVEL       enable memory saving, at LEVEL 1..3
$
```

A.3 Password guess generator

For this lab there is also a password guess generator. This can be used to better control what guesses are used while cracking. It will output a stream of passwords, one per line, on standard out, hence you can pipe this to John the Ripper using the '–stdin' option.

You can find its source code downloadable from the URL

`https://github.com/OpenSecEd/passwd/releases/download/v1.1/pwdstream.py`.

Acknowledgement

This work was originally based on previous work by Rahim Rahmani and Curt-Olof Klasson. It has evolved much since then, essentially only the Windows password hash is the same.