Lab: Password cracking and social engineering

Breaking authentication mechanisms

Daniel Bosk Lennart Franked

13th March 2019

1 Introduction

The most common method of user authentication used today is passwords. This assignment treats the security of passwords: how they should be chosen to give any form of security and how easily different types of passwords are broken. In certain cases however, the strength of the password does not matter. Why crack the password when it is easier to manipulate the holder to giving up the password? Or simply doing the operation you want for you? This falls under the category social engineering. This assignment considers some aspects of this.

1.1 Aim

The aim of this assignment is that you should reflect on the strength of different types of passwords in general, but also when used with different protection strategies, e.g., with or without salts.

The intended learning outcomes are as follows, afterwards you should:

- *reflect* on how the choice of password affects its security.
- *know* of different approaches to getting past password protection.

1.2 Outline

The next section covers what you must read before you understand this assignment and how to do the work. Section 3 covers the work to be done, i.e. how you should learn this. Section 4 covers how it will be examined, i.e. how you show that you have fulfilled the intended learning outcomes given above.

2 Theory

Before doing this laboratory assignment you should read Chap. 2 "Usability and Psychology" and Chap. 5 "Cryptography" in *Security Engineering* [And08]. Further, you need a basic understanding of information theory [Sha48] for this assignment, for this you are recommended to read 'Chapter 6: Shannon entropy' [Uel].

Now that you have the basic theory, you should start reading the main material of this assignment. Start by reading the papers *Human Selection of*

Mnemonic Phrase-based Passwords [KRC06] and 'Of passwords and people: Measuring the effect of password-composition policies' [Kom+11]. You should then read the follow-up paper to the latter: 'Can long passwords be secure and usable?' [Sha+14]. Finally, you should read 'Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms' [Kel+12].

After that you should read about some recent incidents where password databases have leaked. There is a list of breaches maintained by the 'Have I been pwned' service[1], you can search for news articles to read the details. (And we encourage you to use this service.)

For a more in-depth treatment on password guessing, you are recommended to read 'Guessing human-chosen secrets' by Bonneau [Bon12]. However, this is not a mandatory part of the assignment.

The final part of the theory concerns social engineering. You should read about an incident striking the security company RSA, covered in 'RSA: SecurID Attack Was Phishing Via an Excel Spreadsheet' [Fis11].

3 Assignment

The assignment consists of two parts. The first part concerns the security of various types of passwords, you will first reflect and then break some passwords. The second part is about bypassing passwords without breaking them, i.e., it covers the social engineering aspects of security and you will construct a social-engineering-based scenario.

3.1 Introductory reflection

You are now going to reflect on the strength of different types of passwords. Use the theory to properly found your reasoning. Start by comparing how long a password consisting of lower- and upper-case letters, numbers and special characters must be to have the same strength as a password consisting of three and four randomly chooses words, respectively. Then continue by answering the following questions:

- What happens if the randomly chosen words are not that randomly chosen, what if they are rather a famous quote or similar?
- What happens if the 'traditional' password consisting of lower and upper case letters, digits and special characters, is not randomly chosen? What if it is based on a word?
- What is your estimate for password complexity to have a secure password today? Where is the limit in number of guesses needed to correctly guess the password?
- Does it matter where the password is used? Not for the value of the account that the password protects, but how well the service might protect the password.

[1] URL: `https://haveibeenpwned.com/`.

3.2 Cracking passwords

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.

The Windows hash is an old NTLM hash, which means that it is not salted[2]. The UNIX hash is salted and uses the Blowfish (OpenBSD).

3.2.1 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. The hashes are available in Sect. A. Thus, you do not have to use any program like fgdump or unshadow(8) to extract them.

3.3 Social engineering

In this part of the assignment you are going to help the University's security group think about social-engineering-based attack-scenarios. Your assignment is to develop a realistic scenario for a social engineering attack, the purpose of which is to use for educating the University staff. As inspiration you have the literature given in the theory section above.

Also please note, this is not an encouragement to perform this attack, it is strictly academic. You should contribute your scenario by posting it in the forum in the course platform.

4 Examination

The assignment may be solved in groups of up to two students. To get your work examined you should hand in a report (in PDF-format) containing the following:

- All the passwords for the given hashes. You must also describe how you cracked them and how long it took you.
- You should provide your reflections on password strength from above. Then you should relate these to the cracking of the passwords above. How does theory and practice relate?

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

- Your social-engineering-based scenario. Note that this should also be published in the forum in the course platform.

Acknowledements

This work was originally based on previous work by Rahim Rahmani and Curt-Olof Klasson. It has evolved since then and the only remains is the password hashes for Windows and the general idea of password cracking.

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`.

[Bon12] Joseph Bonneau. 'Guessing human-chosen secrets'. PhD thesis. University of Cambridge, May 2012. URL: `http://www.cl.cam.ac.uk/~jcb82/doc/2012-jbonneau-phd_thesis.pdf`.

[Fis11] Dennis Fisher. 'RSA: SecurID Attack Was Phishing Via an Excel Spreadsheet'. Apr. 2011. URL: `https://threatpost.com/en_us/blogs/rsa-securid-attack-was-phishing-excel-spreadsheet-040111`.

[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. URL: `http://ieeexplore.ieee.org/abstract/document/6234434/`.

[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`.

[KRC06] Cynthia Kuo, Sasha Romanosky and Lorrie Faith Cranor. *Human Selection of Mnemonic Phrase-based Passwords*. Tech. rep. 36. Institute of Software Research, 2006. URL: `http://repository.cmu.edu/isr/36/`.

[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.

[Sha48] C. E. Shannon. 'A Mathematical Theory of Communication'. In: *The Bell System Technical Journal* 27 (July 1948), pp. 379–423, 623–656.

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

A The password hashes

The password hashes used in this lab are included below. You can also find them downloadable from URL http://github.com/OpenSecEd/passwd/releases/tag/v1.1/. Note the line numbers and that everything should be on one line.

win-pwd.txt:

```
1 Administrator:500:eb7618163563325a1ca63770c434bf29
    :13242094842304ca9b9b14af2e7160d3:::
2 Guest:501:18c908ec4191f364e72c57ef50f76a05:5
    a30d98619b371f7762fa1dc74761237:::
```

unix-passwd.txt:

```
1 user1:
    $2a$06$0dwWHP6Yqgr6Kd20JJT19uzSDz4yk8pNEsn7pVwwtnvgngcg5e5G6
    :1011:10::/home/user1:/bin/ksh
```

B Password guess generator

For this lab there is also a password guess generator. This can be used with John the Ripper (JtR) 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 JtR 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.

C Instructions for Ophcrack and John the Ripper

C.0.1 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 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.

C.0.2 John the Ripper

JtR 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 JtR, just run the command 'john' in the terminal without any arguments. See List. 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 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:

```
$ 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
$
```

Listing 1: Output from JtR in the terminal.

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

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