Mittuniversitetet
MID SWEDEN UNIVERSITY

Laboratory Assignment: Password Cracking and Social Engineering

Daniel Bosk and Lennart Franked*

–sourcefile– –revision– –time– –owner–

Contents

1 Introduction

The most common method of authentication used today is the use of 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? This falls under the category social engineering. This assignment considers some aspects of this together with Advanced Persistent Threats (APTs).

2 Aim

The aim of this assignment is that by completion you should:

- Have experience of how the chosing of a password affects its security.
- Be aware of different approaches to getting past password protection.
- Have awareness of Advanced Persistent Threats and knowing the meaning of this.

3 Theory

Before doing this laboratory assignment you should read Chapter 2 "Usability and Psychology" and Chapter 5 "Cryptography" in Anderson's *Security Engineering* [1]. You should also read the compendium "Grundläggande lösenordsanalys" [2] and the papers "Human Selection of Mnemonic Phrase-based Passwords" [3] and "Of Passwords and People" [4]. After that you should read about some recent incidents where password databases have leaked, e.g. [5–8].

You should also read about APTs. First you should read about an incident striking the security company RSA in [9]. Then you will read a paper on different approaches to APT, "Sherlock Holmes and The Case of Advanced Persistent Threat" by Juels and Yen [10].

4 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, it covers the social engineering aspects of security and you will construct a social engineering based scenario.

4.1 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 choses words, respectively.

Next, what happens if the randomly chosen words are not that randomly chose, what happens if they are rather a famous quote or similar.

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?

4.1.1 Cracking passwords

On the website

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

you can find a list of programs for password cracking. The two programs recommended for this assignment are "John the Ripper" and "Ophcrack". However, you are free to use any program to solve this. (These are already installed in the computer lab for this course.)

For a UNIX-like operating system the password hashes with corresponding salts stored in the file

`/etc/master.passwd`,

for BSD-based systems such as OpenBSD and FreeBSD, or, in the case of Linux-based systems such as Ubuntu, in the file

`/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 the following files:

`http://ver.miun.se/courses/security/labs/win-pwd.txt`,

and

`http://ver.miun.se/courses/security/labs/unix-passwd.txt`.

Thus, you do not have to use any program like `fgdump` or `unshadow(8)` to extract them.

4.1.2 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, the rainbow table. This reduces the problem om 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 times is what is saved from using a rainbow table. However, this comes with some compromises, the rainbow tables are huge and requires a lot of computational resources to produce. They also requires great resources to use, they must preferably fit in the computers primary memory.

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, 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, or on the website

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

(It is already installed in the course computer lab on campus.)

What you need to use this program are a few rainbow tables. You can find these on the website above. (These are not preinstalled.) Choose your tables carefully.

4.1.3 John the Ripper

John the Ripper, `john(1)`, 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 the website

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

You should use the "Community Enhanced Version". On Ubuntu the packages `john` and `john-data` are recommended. (John the Ripper is already preinstalled in the course computer lab on campus.)

To have a short summary of the possible arguments to pass to `john(1)`, just run `john` in the terminal without any arguments. See Listing 1 on the next page. (Note that on the computers in the lab on campus, you have to type the full search path in the terminal – i.e. `/usr/sbin/john` to run the program.)

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:

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

Choose your wordlists with care. You also have the script `pwdstream.py`[1] to help generate a stream of passwords, see `./pwdstream.py -h` for details.

[1] Available at URL: `http://ver.miun.se/courses/security/labs/pwdstream.py`.

```
$ 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 `john` in the terminal.

4.2 Social engineering and Advance Persistent Threats

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 based 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 theoretical. You should contribute your scenario by posting it in the forum in the course platform.

5 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:

1. All the passwords for the given hashes. You must also describe how you cracked them and how long it took you.
2. 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?
3. Your social engineering based APT-scenario. Note that this should also be published in the forum in the course platform.

References

[1] 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.

[2] Daniel Bosk. "Grundläggande lösenordsanalys". 2013. URL: http://ver.miun.se/courses/security/compendii/pwdanalysis.pdf.

[3] 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/.

[4] 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.

[5] Troy Hunt. *A brief Sony password analysis*. June 2011. URL: http://www.troyhunt.com/2011/06/brief-sony-password-analysis.html.

[6] Graham Cluley. *The worst passwords you could ever choose exposed by Yahoo Voices hack*. July 2012. URL: http://nakedsecurity.sophos.com/2012/07/13/yahoo-voices-poor-passwords/.

[7] Jon Oberheide. *Brief analysis of the Gawker password dump*. Dec. 2010. URL: https://blog.duosecurity.com/2010/12/brief-analysis-of-the-gawker-password-dump/.

[8] Nik Cubrilovic. *RockYou Hack: From Bad to Worse*. Dec. 2009. URL: http://techcrunch.com/2009/12/14/rockyou-hack-security-myspace-facebook-passwords/.

[9] 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.

[10] Ari Juels and Ting-Fang Yen. "Sherlock Holmes and The Case of the Advanced Persistent Threat". In: *LEET*. 2012. URL: https://www.rsa.com/rsalabs/staff/bios/ajuels/publications/SherlockHolmes.pdf.