CS-E4350: Security Engineering, Spring 2021

Assignment 4

Passwords

Passwords have been one of the biggest practical problems facing security engineers since 1970s (Anderson, 2020). The first human-factors issue is that if the password is too long or complex, users might have difficulty entering it correctly. If the operation they're trying to perform is urgent, this might have safety complications.

Since the mid-1980s, people have studied what sort of passwords people choose, and found that they often use personal information to remember the password. This can include spouses' names, birthdates, hobbies in addition to single letters, or sometimes even empty fields.

Today, we have stricter restrictions regarding passwords. The National Institute of Standards and Technology (NIST) and Federal Bureau of Investigation (FBI) have released their yearly 2020 password recommendations. Some of these recommendations include:

1. Use long strings of words and characters at least 15 characters long
2. Change passwords only when they expire or are compromised
3. Take away complexity rules
4. Require screening of new logins against a list of commonly used or compromised logins
5. Monitor for reused logins
6. Limit authentication types
7. Biometrics
8. Limit authentication attempts

You can read more about these recommendations by following this link:

https://securityfirstit.com/wp-content/uploads/2020/03/New-2020-Password-Management-Recommendations-by-the-NIST-and-FBI-1.pdf

As the recommendations grow stricter in nature, we still face many risks regarding passwords and security of users. According to ENISA (European Union Agency for CyberSecurity, 2020). Passwords can be hijacked in many ways:

1. Social Engineering attacks such as phishing credentials using fake pages, voice phishing (so-called Vishing), shoulder surfing (e.g., peeping behind a person who is typing their password on a laptop) and even retrieving handwritten passwords from post-it notes.
2. Stealing using specialized software or physical keyloggers. Some of these attacks require a physical presence or proximity to a laptop or a device.
3. By intercepting communications, using fake access points or by leveraging man-in-the-middle attacks (MiTM) at a network level, more prevalent in public WiFis found in hotels, cafés, airports, etc.

4. Brute-force attacks on passwords by trying all the combinations, dictionary attacks or by simply guessing the password.
5. Retrieving passwords directly from data breaches and leveraging them using password spraying techniquesto other legitimate services.

This assignment consists of the following 3 parts (you should include **all** these parts in your answer):

1. Read section **3.4 Passwords** (Anderson, 2020). Please find a copy of the book by following the link:

   http://libproxy.aalto.fi/login?url=https://ebookcentral.proquest.com/lib/aalto-ebooks/detail.action?docID=6412239

2. Write down your thoughts to the following questions:
   1) Read the recommendations provided by NIST and the FBI and think about it. Do you agree with the recommendations listed? Can you think of other recommendations that might be useful?
   2) What do you think are the best authentication methods today? Do you think passwords are the best authentication technique today? Elaborate on your answer. If you disagree that passwords are the strongest authentication technique nowadays suggest other authentication methods to support your counter argument and explain why you think the other methods are stronger.

      Read the following paper Password Strength: An Empirical Analysis to help you answer this question:

      https://www.researchgate.net/profile/Matteo-Dellamico/publication/224136922_Password_Strength_An_Empirical_Analysis/links/00b7d52e6af00126ca000000/Password-Strength-An-Empirical-Analysis.pdf

   3) Scenario: Assume that you are pen-tester, and a vulnerability has been detected in a web application belonging to company X known as a major social media platform. You were able to access the data base of the clients' passwords and have cloned it. You notice that the original developer was not aware of how to store the passwords and used only SHA-1 to hash the passwords. Your task is to bruteforce the **hashed** password list that you have cloned, which contains 10 passwords. Implement a python password cracker script to crack these 10 passwords. The datasets provided for this assignment are: 1. The list of hashed passwords found in the file **hasheslist.** 2. Wordlist or a dictionary that will assist you with your hacking process. You can find the list of words in **wordlist** file.
   4) Code a password cracker using Python. You can use the Python library `hashlib` for this task. You will need to debug your code using the **hashed** password list. Some hints about password compositions to help you in your cracking process:
      a. Birthdates (1990 – 2000)

b. Combination of two words
c. Uppercase and lowercase
d. Symbols (@, !, %, & .. etc)

5) You will need to crack 80% or more of the 10 hashed passwords provided in order to get the maximum grade for this section.
6) Answer the following question: What is the issue with using SHA-1?

John the ripper was first released in 1996, JtR is a popular password cracking tool originally produced for UNIX-based systems. It was designed to test password strength, brute-force encrypted (hashed) passwords, and crack passwords via dictionary attacks.

You can access the tool from the following link:

https://www.openwall.com/john/

5) Analyze your findings after you run your code and compare with a more advanced solution (using JtR). **Scroll down for the manual to help you use JtR.** What do you notice? Note down your observations and explain. Your observations should include: a. **the time difference** in execution of the task between your solution and JtR, b. **the success rate** of cracking the passwords between your solution and JtR. How do you think your code could be improved? Note down your ideas.

(hint: try exploring `--wordlist` and `--format`)

Instructions:

The deadline for submitting this assignment is **9.04.2021 at 18:00.**

Implementation:

1. You can choose any coding language, but Python is recommended.
2. You can use https://jupyter.cs.aalto.fi/ (Python: General use (Jupyterlab)) or your local environment to code, but the final code submitted together with your PDF report. (.ipynb file + PDF in a zip file). Make sure to check your PDF report using the turnitin link provided before you upload your final zipped file.
3. Your code should be reproducible and readable with clear code commenting. Some basic rules are as follows:
   a. Comment each code block, using a uniform approach for each level. Include a description of its purpose, functions, parameters and results.
   b. Use a consistent style and be precise.

Report format:

1. Font size: 12 pt.
2. Spacing: Single Spacing
3. Use section titles as above in the report outline
4. 2-3 pages (including screenshots and references)

Notice: This assignment is individual work and will be checked by the Turnitin system. Cheating will be addressed with following Aalto rules. Cases will be reported to the appropriate Aalto officer.

Manual
John the Ripper password cracker, version 1.7.6-omp-des-jumbo-9
Copyright (c) 1996-2010 by Solar Designer and others
Homepage: http://www.openwall.com/john/

```
Usage: john [OPTIONS] [PASSWORD-FILES]
--config=FILE              use FILE instead of john.conf or john.ini
--single[=SECTION]         "single crack" mode
--wordlist=FILE --stdin    wordlist mode, read words from FILE or stdin
--rules[=SECTION]          enable word mangling rules for wordlist mode
--incremental[=MODE]       "incremental" mode [using section MODE]
--markov[=LEVEL[:START:END[:MAXLEN]]] "Markov" mode (see documentation)
--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[=LEFT]              show cracked passwords [if =LEFT, then uncracked]
--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
--salt-list=SALT[,SALT,..] load just the specified salt(s)
--salts=[-]COUNT[:MAX]     load salts with[out] at least COUNT passwords only
                           (or in range of COUNT to MAX)
--pot=NAME                 pot file to use
--format=NAME              force hash type NAME:
                           DES/BSDI/MD5/BF/AFS/LM/NT/XSHA/PO/raw-MD5/MD5-gen/
```

```
--format=NAME             force hash type NAME:
                          DES/BSDI/MD5/BF/AFS/LM/NT/XSHA/PO/raw-MD5/MD5-gen/
                          IPB2/raw-sha1/md5a/hmac-md5/phpass-md5/KRB5/bfegg/
                          nsldap/ssha/openssha/oracle/oracle11/MYSQL/
                          mysql-sha1/mscash/lotus5/DOMINOSEC/
                          NETLM/NETNTLM/NETLMv2/NETNTLMv2/NETHALFLM/MSCHAPv2/
                          mssql/mssql05/epi/phps/mysql-fast/pix-md5/sapG/
                          sapB/md5ns/HDAA/raw-md4/md4-gen/sha1-gen
--subformat=NAME          Some formats such as MD5-gen have subformats
                          (like md5_gen(0), md5_gen(7), etc).
                          This allows them to be specified.
                          If the name is LIST, then john will show all
                          subformats (help mode), and exit
--save-memory=LEVEL       enable memory saving, at LEVEL 1..3
--mem-file-size=SIZE      max size a wordlist file will preload into memory
                          (default 5,000,000 bytes)
--field-separator-char=c  Use 'c' instead of the ':' for processing fields
                          (input file, pot file, etc)
--fix-state-delay=N       only determine the wordlist offset every N times
                          It is a performance gain to delay a while
                          (say 100 loops for a fast algorithm).
                          For slow algorithms it should not be used
```