uc3m | Universidad Carlos III de Madrid

Cyber attack techniques

“The Simple Cracking Challenge”

Ramón Hernández León

CONTENTS

LIST OF FIGURES

1. TARGET

The goal of this project is to crack the password of an encrypted excel file.

You can find all the files used for this project in this repository

Keywords: wordlist, chain, hash, CeWL, Mentalist and Hashcat.

1.1. Intel

Password size: 9-11 characters

Structure: ['&']+[word]+[number]

Number: 2 digits

Word: 6 to 8 characters

- All lowercase OR All uppercase OR First uppercase
- Leetify characters: i = 1; e=3; o=0
- Based on target's personal information from

https://en.wikipedia.org/wiki/<Suspect>

1.2. Attack strategy

Based on the intel provided, the attack was divided in the following steps

1. Create a wordlist from the target's Wikipedia page fulfilling the word-length requirement
2. Generate variations of the words based on leetify characters, case-swapping and the static prefix
3. Execute the attack using the wordlist crafted against the hashed password of the file

2. CREATE THE BASE DICTIONARY

The base wordlist was created using the command line tool CeWL to scrap the target's Wikipedia page. Nevertheless, the name of that page in English did not match the name appearing in the encrypted file (*'Corinna_Larsen_enc.xlsx'*) but the Spanish page version did. Therefore, both pages were scrapped to generate a more complete wordlist.

The figure 2.1 shows the *cewl* commands used. Additionally, I created another script to discard those words longer than 8 characters, since cewl does not allow to specify the maximum word length. After executing both commands, I obtained a list of 671 words (see figure 2.3).

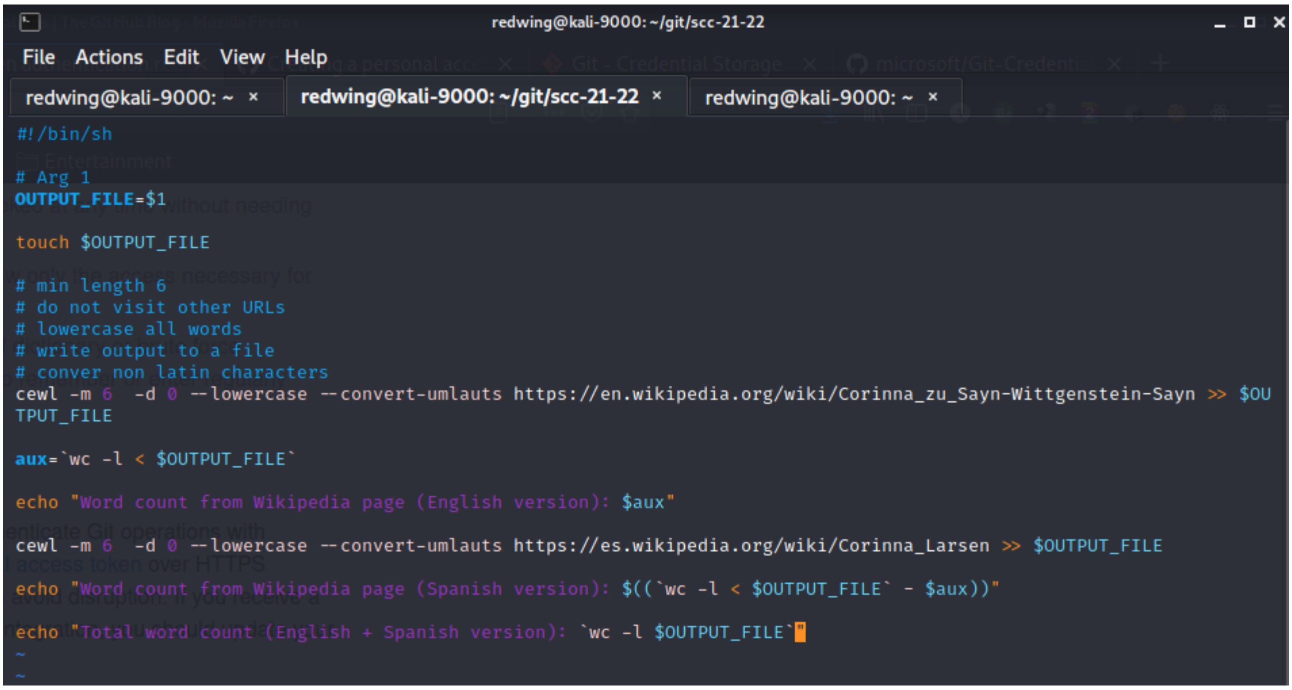


Fig. 2.1. Script to create a base wordlist using cewl


```
#!/bin/sh

# Arg 1
INPUT_FILE=$1
# Arg 2
OUTPUT_FILE=$2

# Arg 3, optional
PREFIX=$3

MIN_LEN=6
MAX_LEN=8

touch $OUTPUT_FILE

echo "Original wordlist size: `wc -l $INPUT_FILE`"
echo "\nSelecting words whose size is between $MIN_LEN and $MAX_LEN ...\n"

while read line; do
    # if line.length is valid save the word
    if [ ${#line} -gt $(($MIN_LEN - 1)) ] && [ ${#line} -lt $(($MAX_LEN + 1)) ]; then
            echo "$PREFIX$line" >> $OUTPUT_FILE
    fi
done < $INPUT_FILE

echo "Output wordlit size: `wc -l $OUTPUT_FILE`"
~
```

Fig. 2.2. Script to discard words not meeting the length requirement

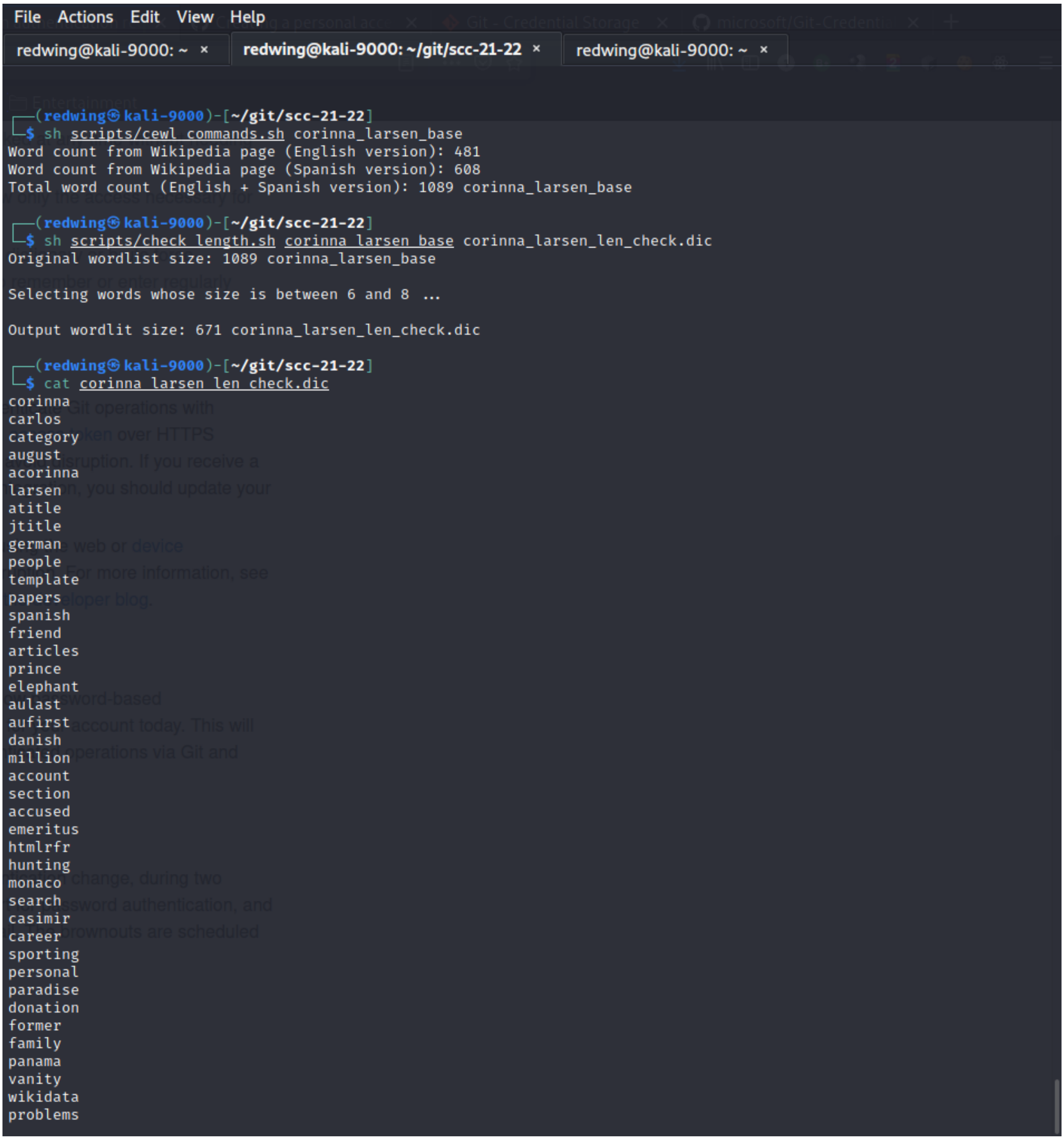


Fig. 2.3. Base word list obtained from the target's Wikipedia page

3. WORD MANGLING

According to the intel, there is an static characters in all possible passwords (&). Additionally, the characters of the word could be altered (case and leetify). In order to contemplate all this variations, a word chain was created using Mentalist GUI.

The password structure is the following: [&] + [word] + [number]. The character '&' is constant, all the possible passwords start with it. Regarding the word, the figure 3.1 shows the scheme of all the possible variations of a single word. Finally, all password end with a 2 digit number (XX). Taking all together, the approximated maximum password space size is 1,610,400 (see figure 3.2).

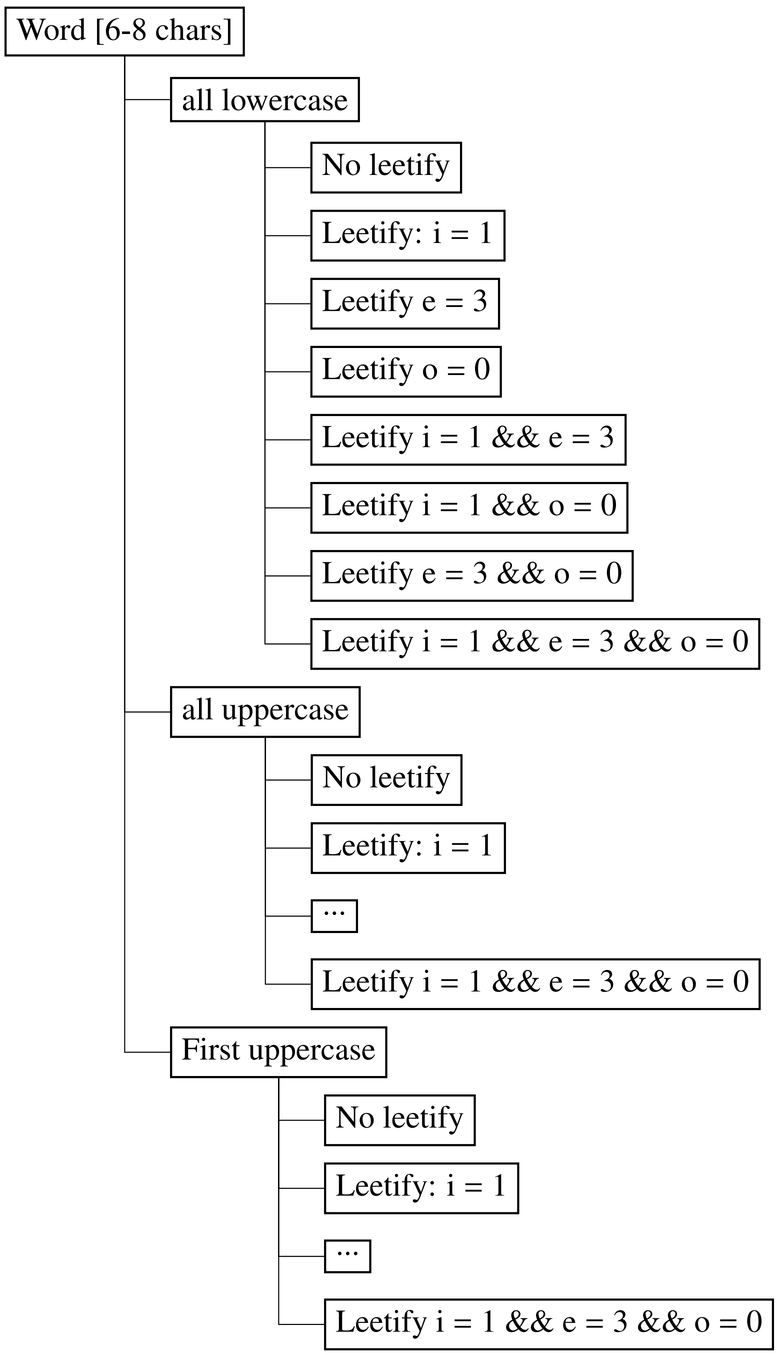


Fig. 3.1. Scheme of all the possible **word variations**. For each of the 3 cases there are 8 different combinations, making up a maximum of 24 variations from a single word (in case it has simultaneously e,i,o vowels)

$$base\_words \cdot word\_variations \cdot number\_variations = 671 \cdot 24 \cdot 100 = 1610400$$

Fig. 3.2. Estimation of the password space size

This requirements were translated to a chain rule using Mentalist GUI. The resulting chain (figure 3.3), composed of 5 nodes, contemplates every character combination mentioned above. The chain can be processed to obtain the **complete resulting wordlist** (figure 3.4), or to generate a set of **hashcat/jhon rules** (figure 3.5) to be used directly with the respective tool. The result is the same, just changes the approach to the crack: dictionary attack or rule based. I decided to take the dictionary approach since the estimated wordlist size is reasonably small to keep it in a file: 624,900 words (6.6MB)[1]

[1]Mentalist estimated 1,262,600 different passwords VS the 1,610,400 I estimated after the password structure analysis. This difference is because Mentalist checks duplicated cases and takes them out of the count (e.g. for the word "feel" the case leetify-i = leetify-o). Nevertheless, after computing all the possibilities and removing all the duplicates, the final list has 624,900 words (figure 3.4)

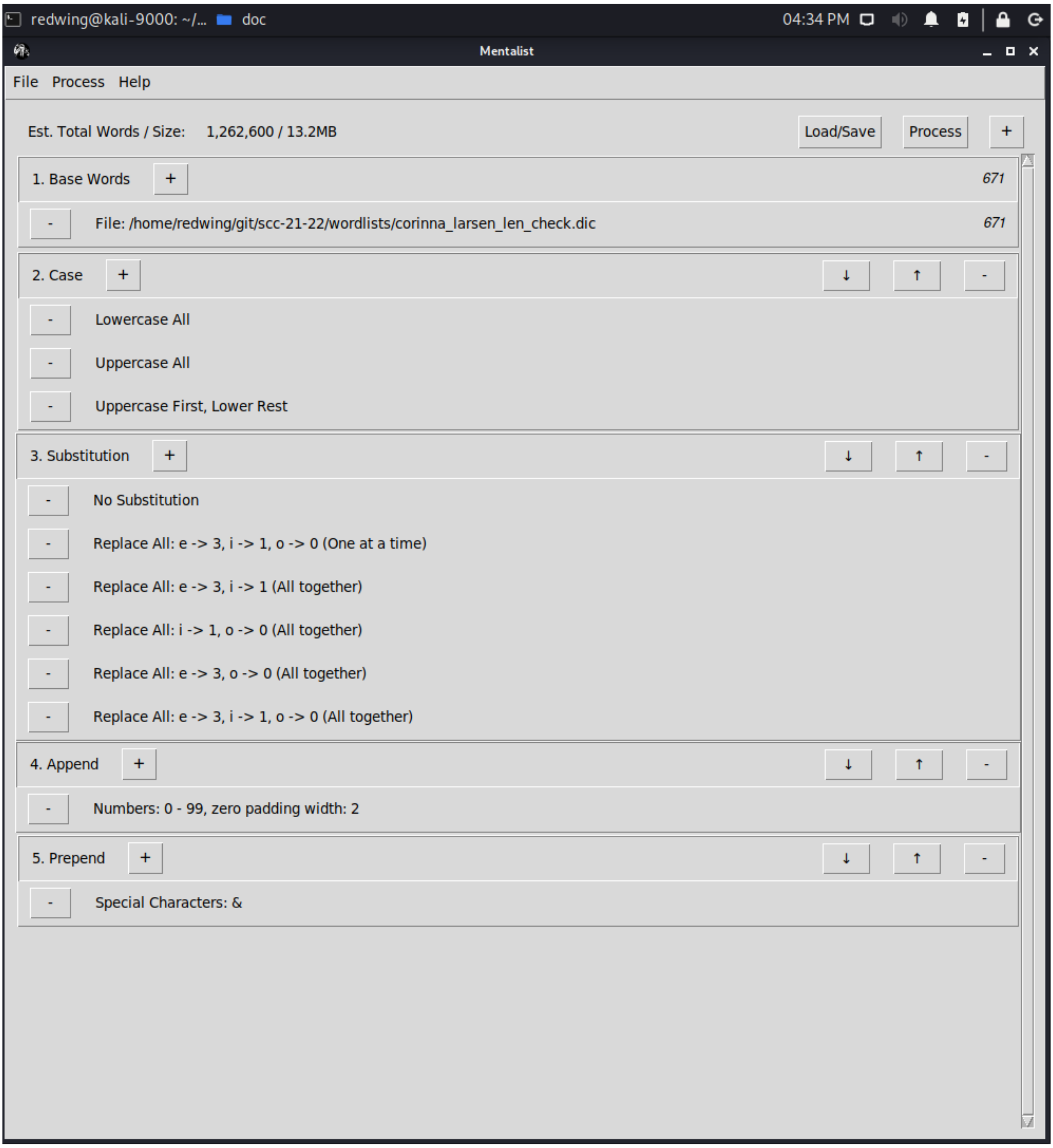


Fig. 3.3. Chain created to recreated all the possible passwords meeting the requirements mentioned in this chapter

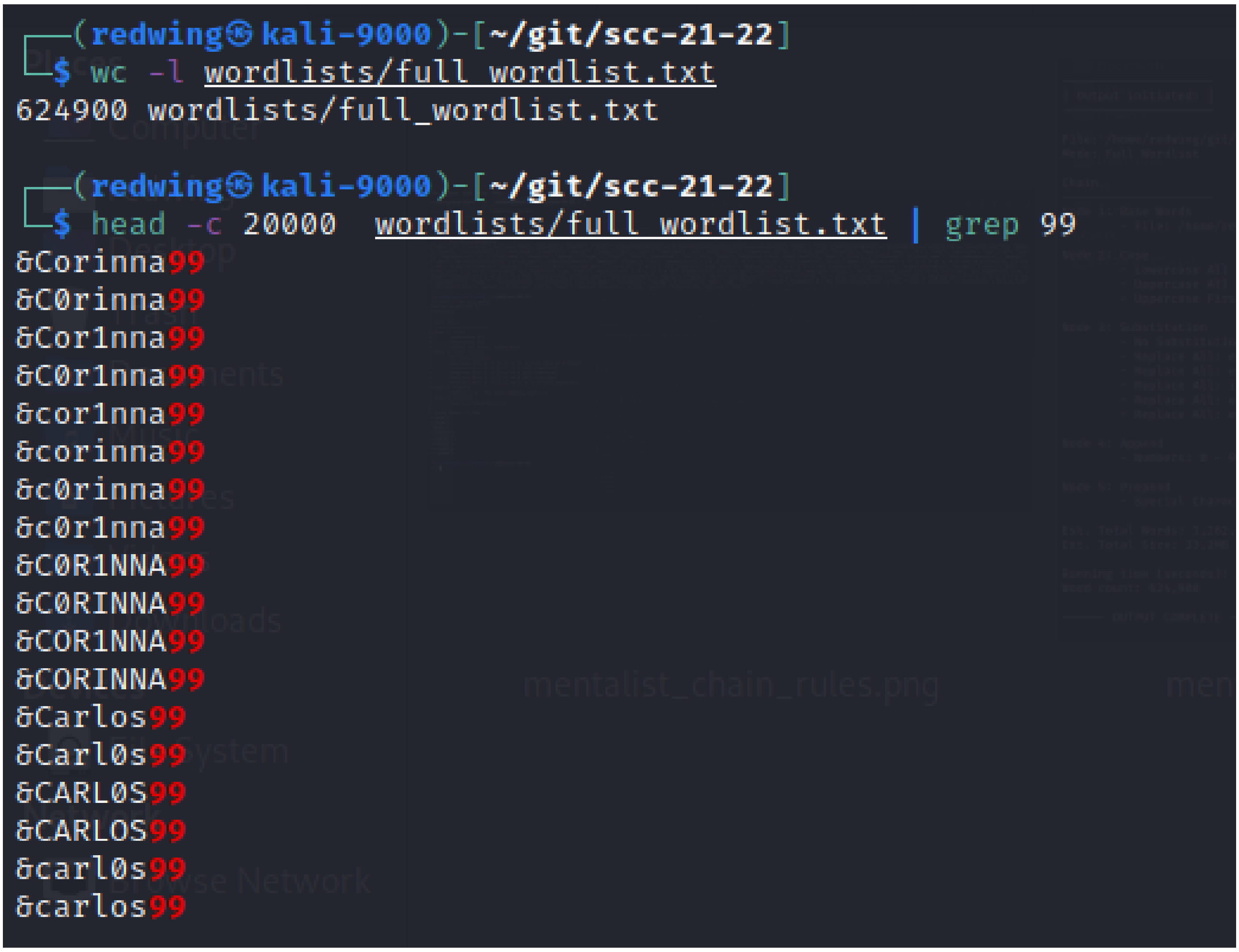


Fig. 3.4. Wordlist generated from the chain defined. Some examples are extracted, so that the word mutations mentioned

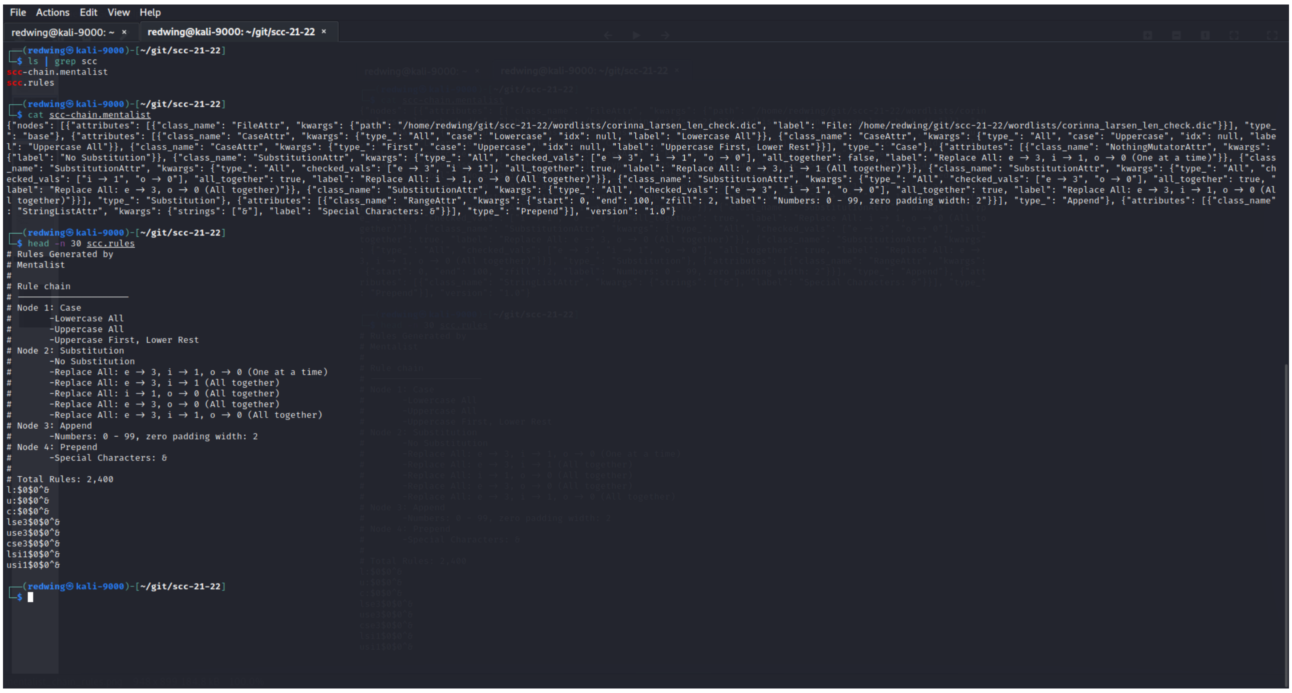


Fig. 3.5. Rules generated from the chain defined

4. CRACKING THE PASSWORD

Finally, the attack was performed using *Hashcat* tool. I selected *Hashcat* rather than *Jhon the Ripper* because I already have worked with Jhon and I wanted to try Hashcat.

I aimed for a dictionary attack, using the wordlist generated in the previous step with Mentalist. Since all the combinations are stored as a different word we do not need to add any additional rule. The other attack alternative was to use the set of rules generated from the chain with Mentalist and use them directly so that *Hashcat* generates all the passwords dynamically instead of reading them from a file.

First of all, we need to obtain the hash of the file's password, otherwise we cannot automate the attack. The steps, explained in the Hashcat FAQ [1], are the following

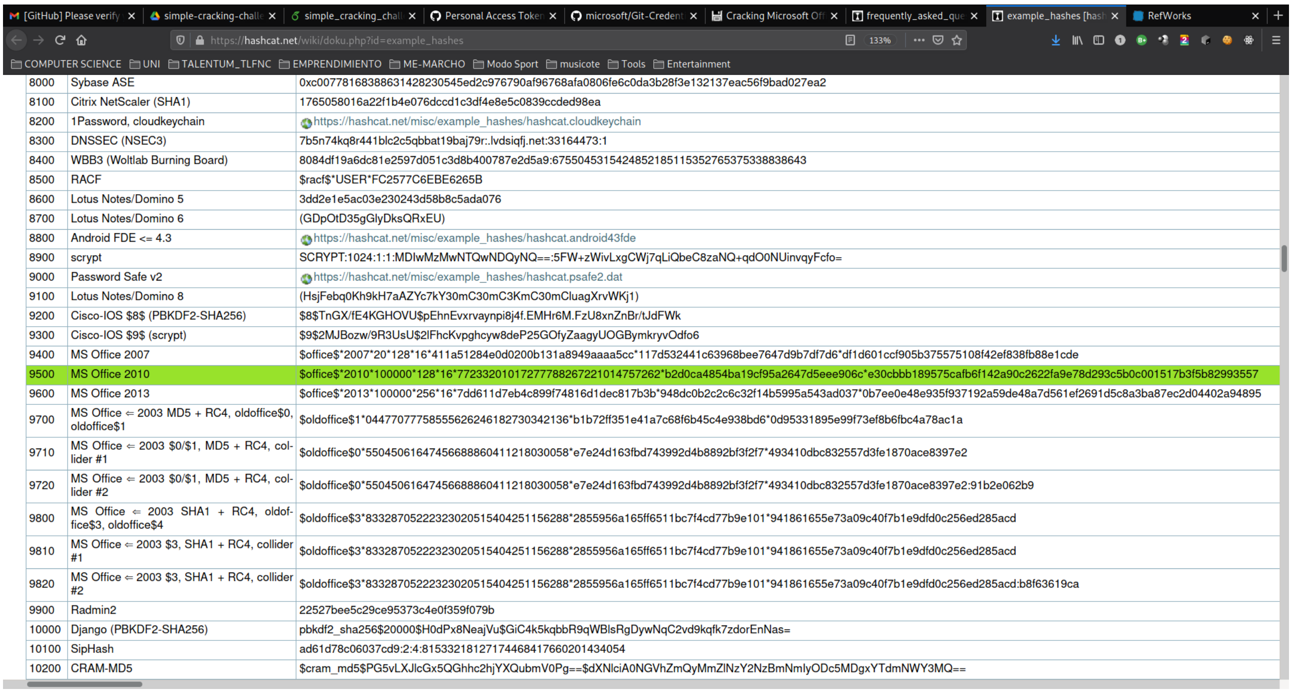
1. Extract hashed password from the file using the script [2] *office2hashcat.py* [4]. The output of the command is show in the figure 4.2

2. Deduce the hash type to select the appropiate hash mode. For that, I checked a common hashes table from the Hashcat wiki (figure 4.1). In that table there is an entry with the same format than our *hash.txt*: it is a **MS Office 2010 hash**, associated with the **Hashcat mode 9500**.

Once we know the password hash, the hash type we start the attack (fig 4.3: mode 0 (straight, wordlist), hash type 9500, input wordlist "full_wordlist.txt". The process starts and the GPU fans roar. After a little time, Hashcat shows some output in the terminal, telling us that it has found a match for the given hashed password: **&jun10r27**.

With the cracked password I could unlock the Excel file. The figure 4.4 shows the contents of the file once it has been decrypted. The flag contained inside it is: **Corinna_Larsen pandora 176 &jun10r27**

[2] There were some execution errors with the *office2hashcat.py* script. After some searches I managed to solve them by replacing 2 deprecated functions: ElementTree.getiterator() by ElementTree.iter() [2] and base64.decodestring() by base64.decodebytes() [3]



8000	Sybase ASE	0xc00778168388631428230545ed2c976790af96768afa0806fe6c0da3b28f3e132137eac56f9bad027ea2
8100	Citrix NetScaler (SHA1)	1765058016a22f1b4e076dccd1c3df4e8e5c0839ccded98ea
8200	1Password, cloudkeychain	https://hashcat.net/misc/example_hashes/hashcat.cloudkeychain
8300	DNSSEC (NSEC3)	7b5n74kq8r441blc2c5qbbat19baj79r:.lvdsiqfj.net:33164473:1
8400	WBB3 (Woltlab Burning Board)	8084df19a6dc81e2597d051c3d8b400787e2d5a9:6755045315424852185115352765375338838643
8500	RACF	$racf$*USER*FC2577C6EBE6265B
8600	Lotus Notes/Domino 5	3dd2e1e5ac03e230243d58b8c5ada076
8700	Lotus Notes/Domino 6	(GDpOtD35gGlyDksQRxEU)
8800	Android FDE <= 4.3	https://hashcat.net/misc/example_hashes/hashcat.android43fde
8900	scrypt	SCRYPT:1024:1:1:MDIwMzMwNTQwNDQyNQ==:5FW+zWivLxgCWj7qLiQbeC8zaNQ+qdO0NUinvqyFcfo=
9000	Password Safe v2	https://hashcat.net/misc/example_hashes/hashcat.psafe2.dat
9100	Lotus Notes/Domino 8	(HsjFebq0Kh9kH7aAZYc7kY30mC30mC3KmC30mCluagXrvWKj1)
9200	Cisco-IOS 8 (PBKDF2-SHA256)	8TnGX/fE4KGHOVU$pEhnEvxrvaynpi8j4f.EMHr6M.FzU8xnZnBr/tJdFWk
9300	Cisco-IOS 9 (scrypt)	$9$2MJBozw/9R3UsU$2lFhcKvpghcyw8deP25GOfyZaagyUOGBymkryvOdfo6
9400	MS Office 2007	$office$*2007*20*128*16*411a51284e0d0200b131a8949aaaa5cc*117d532441c63968bee7647d9b7df7d6*df1d601ccf905b375575108f42ef838fb88e1cde
9500	MS Office 2010	$office$*2010*100000*128*16*77233201017277788267221014757262*b2d0ca4854ba19cf95a2647d5eee906c*e30cbbb189575cafb6f142a90c2622fa9e78d293c5b0c001517b3f5b82993557
9600	MS Office 2013	$office$*2013*100000*256*16*7dd611d7eb4c899f74816d1dec817b3b*948dc0b2c2c6c32f14b5995a543ad037*0b7ee0e48e935f937192a59de48a7d561ef2691d5c8a3ba87ec2d04402a94895
9700	MS Office ⇐ 2003 MD5 + RC4, oldoffice$0, oldoffice$1	$oldoffice$1*04477077758555626246182730342136*b1b72ff351e41a7c68f6b45c4e938bd6*0d95331895e99f73ef8b6fbc4a78ac1a
9710	MS Office ⇐ 2003 $0/$1, MD5 + RC4, collider #1	$oldoffice$0*55045061647456688860411218030058*e7e24d163fbd743992d4b8892bf3f2f7*493410dbc832557d3fe1870ace8397e2
9720	MS Office ⇐ 2003 $0/$1, MD5 + RC4, collider #2	$oldoffice$0*55045061647456688860411218030058*e7e24d163fbd743992d4b8892bf3f2f7*493410dbc832557d3fe1870ace8397e2:91b2e062b9
9800	MS Office ⇐ 2003 SHA1 + RC4, oldoffice$3, oldoffice$4	$oldoffice$3*83328705222323020515404251156288*2855956a165ff6511bc7f4cd77b9e101*941861655e73a09c40f7b1e9dfd0c256ed285acd
9810	MS Office ⇐ 2003 $3, SHA1 + RC4, collider #1	$oldoffice$3*83328705222323020515404251156288*2855956a165ff6511bc7f4cd77b9e101*941861655e73a09c40f7b1e9dfd0c256ed285acd
9820	MS Office ⇐ 2003 $3, SHA1 + RC4, collider #2	$oldoffice$3*83328705222323020515404251156288*2855956a165ff6511bc7f4cd77b9e101*941861655e73a09c40f7b1e9dfd0c256ed285acd:b8f63619ca
9900	Radmin2	22527bee5c29ce95373c4e0f359f079b
10000	Django (PBKDF2-SHA256)	pbkdf2_sha256$20000$H0dPx8NeajVu$GiC4k5kqbbR9qWBlsRgDywNqC2vd9kqfk7zdorEnNas=
10100	SipHash	ad61d78c06037cd9:2:4:81533218127174468417660201434054
10200	CRAM-MD5	$cram_md5$PG5vLXJlcGx5QGhhc2hjYXQubmV0Pg==$dXNlciA0NGVhZmQyMmZlNzY2NzBmNmIyODc5MDgxYTdmNWY3MQ==

Fig. 4.1. Entry for the hash type 9500 "MS Office 2010". Extracted from [5]

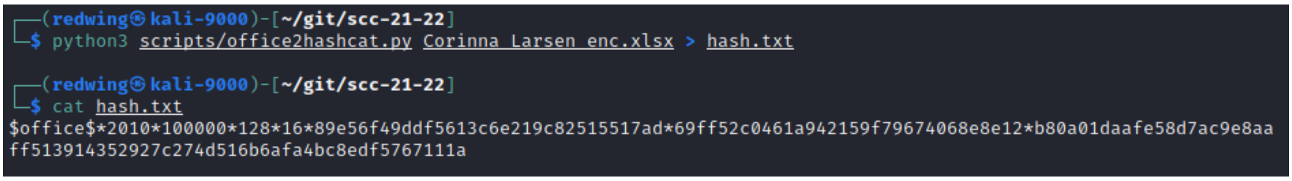


Fig. 4.2. Extraction of the hashed password from the Office Excel file using *Office2hashcat.py*[4]

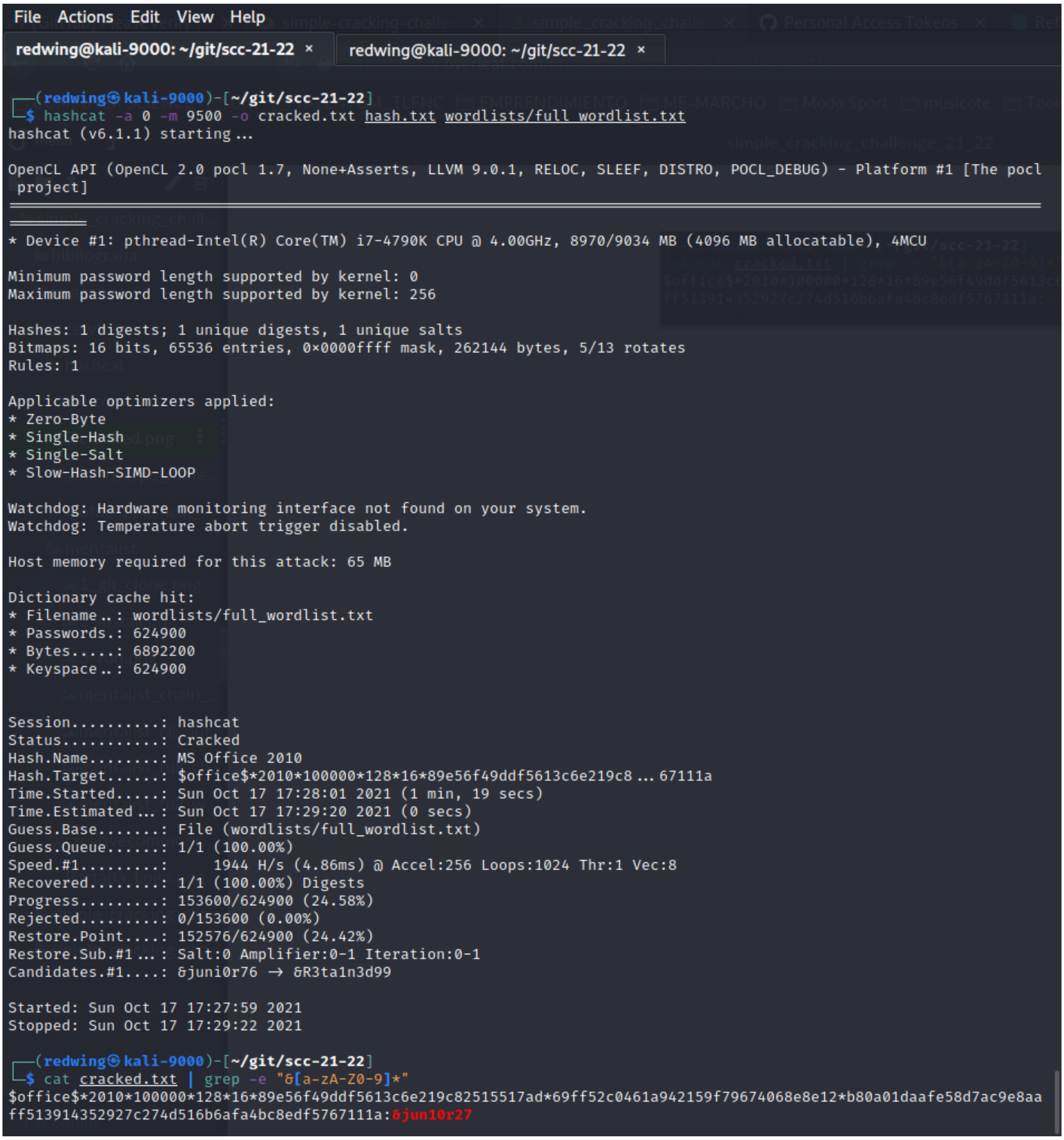


Fig. 4.3. Attack command used to crack the password

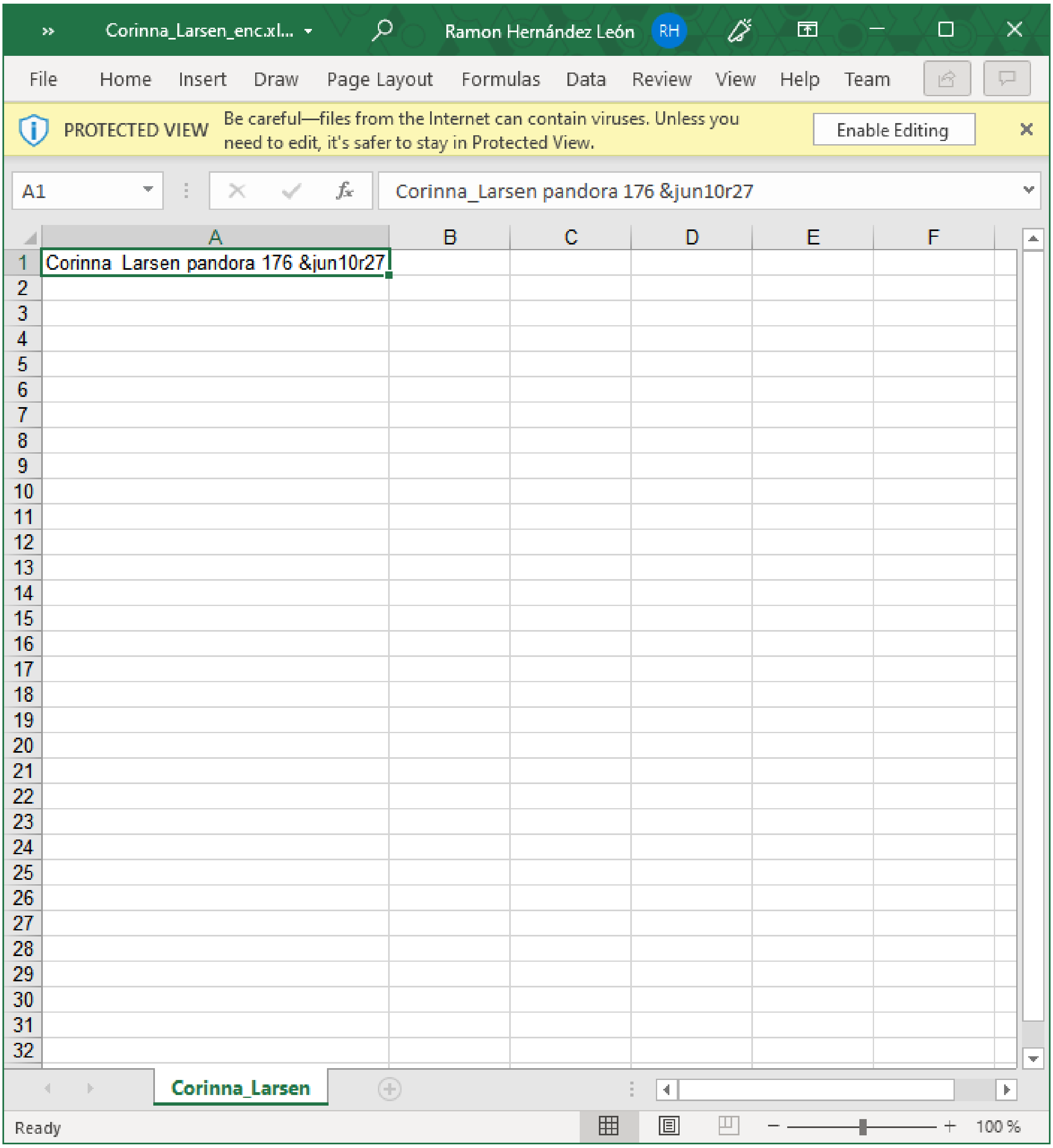


Fig. 4.4. Showing the contents of the file after cracking its password

BIBLIOGRAPHY

[1] Hashcat. (). “Frequently asked questions [hashcat wiki].” The question we are insterested in is "How do I extract the hashes from Office (Word, Excel, etc.) documents?" [Online]. Available: https://hashcat.net/wiki/doku.php?id=frequently%5C_asked%5C_questions%5C#how%5C_do%5C_i%5C_extract%5C_the%5C_hashes%5C_from%5C_office%5C_word%5C_excel%5C_etc%5C_documents.

[2] Philsmd. (2020). “Office2hashcat error [hashcat wiki],” [Online]. Available: https://hashcat.net/forum/thread-9560.html.

[3] BillyBBone. (Sep. 2021). “Getting attributeerror: Module ’base64’ has no attribute ’decodestring’ error while running on python 3.9.6 - stack overflow,” [Online]. Available: https://stackoverflow.com/questions/69187685/getting-attributeerror-module-base64-has-no-attribute-decodestring-error-wh.

[4] P. Lagadec, *Office2hashcat.py*, Jan. 2015. [Online]. Available: https://github.com/stricture/hashstack-server-plugin-hashcat/blob/master/scrapers/office2hashcat.py.

[5] Hashcat. (). “Example hashes [hashcat wiki],” [Online]. Available: https://hashcat.net/wiki/doku.php?id=example%5C_hashes.