HOCHSCHULE
CAMPUS WIEN
UNIVERSITY OF
APPLIED SCIENCES

Exercise 1

VeraCrypt access recovery

Author:

Philip Magnus

Student identification number:

c2410537022

Date:

September 29, 2025

Contents

1 Introduction **3**
1.1 Task . . . 3

2 Recovering access **4**
2.1 Overview . . . 4
2.2 Tools . . . 4
2.3 Calculating the container header . . . 5
2.4 Cracking the password . . . 5
2.5 Mounting the container and extracting the content . . . 7
2.5.1 Mounting the container . . . 7
2.5.2 Extracting the files . . . 7
2.5.3 Container contents . . . 9
2.5.4 Dismounting container . . . 10

3 Findings **11**
3.1 Summary of container findings . . . 11
3.2 Questions . . . 11
3.2.1 How much time is needed for brute forcing different password lengths and character sets? . . . 11
3.2.2 What password complexity is needed for a 10-year secure container? . . . 11

Bibliography **13**

List of Figures **14**

List of Tables **15**

Listings **16**

1 Introduction

In the following document, we discuss the analysis steps and the resulting findings for the VeraCrypt exercise executed by Philip Magnus.

The report is structured in 3 sections. Section 1 recaps the task itself. Section 2 outlines the work performed to achieve a successful result. Section 3 provides an analysis of the findings and answers the additional questions provided by the lecturer.

1.1 Task

We received the following task from Spongebob Squarepants:

"Spongebob Squarepants lost the password for his Veracrypt container and hires you to crack it. You chuckle and expect something like 'password' or '123456'. You agree, sign the Bikini Bottom NDA and go directly to work."

The following details were provided so that we could access the container and try to gain access:

- The container could be obtained at http://seclva.ifs.tuwien.ac.at/forensics/mkveracrypt.php
- The VeraCrypt container uses a default PIM factor and AES with SHA256
- No hidden container inside
- Password character set and length are configurable while obtaining the container

Additionally two questions were asked by the lecturer:

- How much time is needed for brute forcing different password lengths and character sets?
- What password complexity is needed for a 10-year secure container?

To generate the container, an 8-digit student ID had to be provided to the website. We used **24105370** as our student ID.

2 Recovering access

2.1 Overview

The container generated by us, with the student ID **24105370** and a password consisting of 4 digits, generates the following sha256 hash:

```
1 sha256sum container_24105370.hc
```

Listing 2.1: container sha256 hash

```
a9a4937162520863c2b43c0336a3a7b9b530023ec6a92fb9cb4a999a59a296b3
```

Listing 2.2: container hash output

The container is protected by the password **6436** and contains the following files:

Hash	File
7d8355b740c5f07e4c4ed682374867dbbcd7921297bb6139a59d36ed94575949	awesome.jpg
098ce58c9e845f13af398bbda10aa448f154ae8bc852715b424b02236a5087a3	secret.txt
c21c67ec7d1c9e7a8c397acb68dd36af82b119ad1fe2a3c1d03cfacd1b433b3a	trippin.jpg
1a560c55c7b7c59882713876cf9460773451cf277621deb36cf9d65ed297a4f9	wasted.jpg

Table 2.1: Container contents

The *secret.txt* contains the following string of characters:

bdf74840911603bf25660c372d8204ce70803b7472

The rest of the files in the container are three pictures of Spongebob Squarepants and Patrick Star.

2.2 Tools

The following tools were used during the analyisis of the container:

- Fedora 42 Kernel: 6.16.7-200.fc42.x86_64
- Hashcat v6.2.6
- GPU: Mesa Intel(R) Iris(R) Xe Graphics (ADL GT2)
- VeraCrypt 1.26.24

2.3 Calculating the container header

First we calculated the hash of the full container for proof that our methods are non invasive and do not change the contents of the container.

```
1 sha256sum container_24105370.hc
```

Listing 2.3: sha256sum command

```
baf547c16dc52fa0cdd3140e62efb851cc5eb5ec70c928c4a9c29a59429e8b10 \\
container_24105370.hc
```

Listing 2.4: sha256sum command output

Executing the following command extracts the first 512 bytes of the container file to a new binary file. These 512 bytes contain the hash of the password needed to decrypt the container. The Hashcat algorithm can either be run against the newly extracted hash stored in a binary format or against the container itself.

```
1 dd if=container_24105370.hc of=24105370.hash bs=512 count=1
```

Listing 2.5: dd command

yields the following output:

```
1+0 Datenstze ein
1+0 Datenstze aus
512 Bytes kopiert, 7,9609e-05 s, 6,4 MB/s
```

Listing 2.6: dd command output

2.4 Cracking the password

Using the hashing algorithm no. 13751 [1] we can start cracking the hashed container password.

```
1 hashcat -d 1 -a3 -w3 -m 13751 --status container_24105370.hc "?d?d?d?d"
```

Listing 2.7: hashcat command

```
hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 ) - Platform #1 [Intel(R) Corporation]
============================================================
* Device #1: Intel(R) Iris(R) Xe Graphics, 14656/29394 MB (2047 MB allocatable),
    96MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 128

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates

Optimizers applied:
* Zero-Byte
* Single-Hash
* Single-Salt
* Brute-Force
* Slow-Hash-SIMD-LOOP
```

```
ATTENTION! Potfile storage is disabled for this hash mode.
Passwords cracked during this session will NOT be stored to the potfile.
Consider using -o to save cracked passwords.

Watchdog: Hardware monitoring interface not found on your system.
Watchdog: Temperature abort trigger disabled.

Host memory required for this attack: 281 MB

The wordlist or mask that you are using is too small.
This means that hashcat cannot use the full parallel power of your device(s).
Unless you supply more work, your cracking speed will drop.
For tips on supplying more work, see: https://hashcat.net/faq/morework

Approaching final keyspace - workload adjusted.

[s]tatus [p]ause [b]ypass [c]heckpoint [f]inish [q]uit =>

Session..........: hashcat
Status...........: Running
Hash.Mode........: 13751 (VeraCrypt SHA256 + XTS 512 bit (legacy))
Hash.Target......: container_24105370.hc
Time.Started.....: Sun Sep 28 00:40:12 2025 (7 secs)
Time.Estimated...: Sun Sep 28 00:55:13 2025 (14 mins, 54 secs)
Kernel.Feature...: Pure Kernel
Guess.Mask.......: ?d?d?d?d [4]
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........: 11 H/s (0.57ms) @ Accel:1024 Loops:31 Thr:256 Vec:1
Recovered........: 0/1 (0.00%) Digests (total), 0/1 (0.00%) Digests (new)
Progress.........: 0/10000 (0.00%)
Rejected.........: 0/0 (0.00%)
Restore.Point....: 0/1000 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:167679-167710
Candidate.Engine.: Device Generator
Candidates.#1....: 1234 -> 1551

[s]tatus [p]ause [b]ypass [c]heckpoint [f]inish [q]uit =>

[omitted for readability]

container_24105370.hc:6436
Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 13751 (VeraCrypt SHA256 + XTS 512 bit (legacy))
Hash.Target......: container_24105370.hc
Time.Started.....: Sun Sep 28 00:40:12 2025 (11 mins, 30 secs)
Time.Estimated...: Sun Sep 28 00:51:42 2025 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Mask.......: ?d?d?d?d [4]
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........: 11 H/s (0.59ms) @ Accel:1024 Loops:31 Thr:256 Vec:1
Recovered........: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.........: 7680/10000 (76.80%)
Rejected.........: 0/7680 (0.00%)
Restore.Point....: 512/1000 (51.20%)
Restore.Sub.#1...: Salt:0 Amplifier:9-10 Iteration:499968-499999
Candidate.Engine.: Device Generator
```

```
Candidates.#1....: 6813 -> 6360
Started: Sun Sep 28 00:39:39 2025
Stopped: Sun Sep 28 00:51:43 2025
```

Listing 2.8: hashcat command output

As you can see in the command output in Listing 2.8, with a fairly slow integrated graphics card the password was cracked in 11 minutes and 30 seconds.

After executing the Hashcat algorithm we can again verify the sha256 hash of the container to proof that none of the contents could have changed.

```
1 sha256sum container_24105370.hc
```

Listing 2.9: sha256sum command

```
baf547c16dc52fa0cdd3140e62efb851cc5eb5ec70c928c4a9c29a59429e8b10 \\
container_24105370.hc
```

Listing 2.10: sha256sum command output

2.5 Mounting the container and extracting the content

2.5.1 Mounting the container

To view the containers contents we need to mount it through the VeraCrypt software. We used the command line option to mount the container in a read-only state, as demonstrated in Listing 2.11.[2]
The same operation would be possible using the GUI for VeraCrypt.

```
1 sudo veracrypt --text --mount-options=ro -p 6436 container_24105370.hc
```

Listing 2.11: VeraCrypt mounting

```
Enter mount directory [default]:
Enter PIM for /../digitale_forensik/uebung_1/container_24105370.hc:
Enter keyfile [none]:
```

Listing 2.12: VeraCrypt mounting output

By default the VeraCrypt container is mounted in the */media/veracrypt1* directory.

2.5.2 Extracting the files

The following steps describe how we extracted the files contained in the decrypted and mounted VeraCrypt container.
First, we calculate the sha256 hashes of the files in the mounted container before copying.

```
1 sha256sum /media/veracrypt1/*
```

Listing 2.13: hashes extracted files

```
7d8355b740c5f07e4c4ed682374867dbbcd7921297bb6139a59d36ed94575949 awesome.jpg
098ce58c9e845f13af398bbda10aa448f154ae8bc852715b424b02236a5087a3 secret.txt
c21c67ec7d1c9e7a8c397acb68dd36af82b119ad1fe2a3c1d03cfacd1b433b3a trippin.jpg
1a560c55c7b7c59882713876cf9460773451cf277621deb36cf9d65ed297a4f9 wasted.jpg
```

Listing 2.14: hashes extracted files output

We create a new directory to store the extracted files.

```
mkdir extracted_files
```

Listing 2.15: create directory

We then copy the files from the mounted VeraCrypt container to the newly created directory:

```
cp /media/veracrypt1/* extracted_files/
```

Listing 2.16: copy files

Listing the directory contents shows us the contained files.

```
ls -la extracted_files/
```

Listing 2.17: ls extracted files

```
insgesamt 540
drwxr-xr-x. 1 philip philip 84 28. Sep 01:24 .
drwxr-xr-x. 1 philip philip 236 28. Sep 00:56 ..
-rwx------. 1 philip philip 362372 28. Sep 00:33 awesome.jpg
-rwx------. 1 philip philip 42 28. Sep 00:33 secret.txt
-rwx------. 1 philip philip 80961 28. Sep 00:33 trippin.jpg
-rwx------. 1 philip philip 98374 28. Sep 00:33 wasted.jpg
```

Listing 2.18: ls extracted files output

Finally we calculate the sha256 hash of all the extracted files.

```
cd extracted_files
sha256sum *
```

Listing 2.19: hashes extracted files

```
7d8355b740c5f07e4c4ed682374867dbbcd7921297bb6139a59d36ed94575949 awesome.jpg
098ce58c9e845f13af398bbda10aa448f154ae8bc852715b424b02236a5087a3 secret.txt
c21c67ec7d1c9e7a8c397acb68dd36af82b119ad1fe2a3c1d03cfacd1b433b3a trippin.jpg
1a560c55c7b7c59882713876cf9460773451cf277621deb36cf9d65ed297a4f9 wasted.jpg
```

Listing 2.20: hashes extracted files output

2.5.3 Container contents

The container contained the files as listed in subsection 2.5.2. The text file contained a string of random characters and numbers. The remaining files are three pictures of Spongebob Squarepants and Patrick Star as shown below.

```
1 cat secret.txt
```

Listing 2.21: cat secret

```
bdf74840911603bf25660c372d8204ce70803b7472
```

Listing 2.22: cat secret output

Figure 2.1: awesome.jpg

Figure 2.2: trippin.jpg

Figure 2.3: wasted.jpg

2.5.4 Dismounting container

To dismount the container we executed the following command.

```
1 veracrypt --unmount --force
```

Listing 2.23: cat secret

The command does not yield any output but unmounts all currently mounted containers. To finish our analysis we can check the sha256 hash of the container to proof that none of our methods altered any of its contents.

```
1 sha256sum container_24105370.hc
```

Listing 2.24: sha256sum after analysis command

```
baf547c16dc52fa0cdd3140e62efb851cc5eb5ec70c928c4a9c29a59429e8b10 \\
container_24105370.hc
```

Listing 2.25: sha256sum after analysis command output

3 Findings

3.1 Summary of container findings

The container was protected by the password **6436**.
Inside the container we found one text file and three pictures. The text file contained the following string of characters an digits: **bdf74840911603bf25660c372d8204ce70803b7472**. The remaining files, i.e. the three pictures, are pictures of Spongebob Squrepants by himself or with his best friend Patrick Star. A copy of each picture can be seen in the previous chapter at subsection 2.5.3.

3.2 Questions

3.2.1 How much time is needed for brute forcing different password lengths and character sets?

The amount of time to brute force different password length and char sets is heavily dependend on the type of hardware used for the attack as well as the algorithm that we try to brute force. For the sake of this answer we will assume the usage of modern hardware, i.e. a NVIDIA RTX 4090 graphics card, as well as the VeraCrypt AES algorithm used earlier in this exercise.

For the purpose of this analysis we will consult the Hashcat benchmark by user *Chick3nman*, who ran the hashcat benchmark on the aforementioned RTX 4090.[3]
The benchmark shows that a single RTX 4090 can achieve around $9000\frac{H}{s}$. This would result in the following speeds for different password length and character sets.

Length	Digits only 10 chars	uppercase & lowercase & digits 62 chars	Full set (+ symbols) 95 chars
8	3 hours	769 years	23374 years
9	30 hours	47695 years	2220564 years
10	12 days	2957112 years	210953597 years
11	128 days	183340946 years	20040591784 years
12	$3\frac{1}{2}$ years	11367138657 years	$1,90385622 \cdot 10^{12}$ years

Table 3.1: Time to bruteforce passwords RTX 4090

A password hash derived from 8 random characters selected from a 62-character set, protecting an AES-encrypted container, would take someone using a consumer-grade graphics card up to 769 years to crack in the worst case.

3.2.2 What password complexity is needed for a 10-year secure container?

To answer this question we will again use the parameters defined in subsection 3.2.1.
Looking at our Table 3.1 we can already see that an 8 character password with a 62 character sets overshoots the 10 year security quite a bit.

In Table 3.2 we listed the minimum password requirement for each set to last a non-stop brute force attack approximately 10 years.

Charset	Length	Hash Speed	Brute Force Time
Digits only 10 chars	13	$9000\frac{H}{s}$	35 years
uppercase & lowercase & digits 62 chars	7	$9000\frac{H}{s}$	12 years
Full set (+ symbols) 95 chars	7	$9000\frac{H}{s}$	246 years

Table 3.2: Password minimum requirements to last a minimum 10 years against consumer grade attack

We chose a minimum amount of 13 characters for digits only. This overshoots the 10 years by a bit but as we can see in Table 3.1 12 characters would result in a significantly lower time needed to brute force the password.
For the 62 character set, i.e. lower- uppercase numbers, we are pretty spot on with a time of 12 years for a password with a length of 7 characters.
The biggest jump in password securtiy can be seen for the full character set of 95 characters. If we were to use a password length of just 6 the password would be crackable in about $2\frac{1}{2}$ years. Adding just one character more to its length results in about 9840% increase in time to crack the password.

Table 3.2 can be viewed as the recommended password length for each charset to last a minimum of ten years against consumer grade attacks.

It should be noted that these numbers should not be viewed as a general guide, as hardware is becoming increasingly more efficient for its price even in the consumer space.[4]
It must also be noted that the numbers mentioned above are only valid in a scenario where an attacker would use one graphics card. In a normal attack more than one graphics card would be used in combination to try and crack the hash for a given password protected container. Attackers usually would use up to 8 graphics cards in combination to get the best performance possible. [5]

Bibliography

[1] Hashcat. "Hashcat documentation example hashes." Accessed: 2025-09-28. [Online]. Available: https://hashcat.net/wiki/doku.php?id=example_hashes (cit. on p. 5).

[2] VeraCrypt. "Veracrypt command line usage." Accessed: 2025-09-28. [Online]. Available: https://veracrypt.io/en/Command%20Line%20Usage.html (cit. on p. 7).

[3] Chick3nman. "Hashcat benchmark for rtx 4090." Accessed: 2025-09-28. [Online]. Available: https://gist.github.com/Chick3nman/32e662a5bb63bc4f51b847bb422222fd (cit. on p. 11).

[4] vast.ai. "Vast.ai pricing." Accessed: 2025-09-28. [Online]. Available: https://vast.ai/pricing (cit. on p. 12).

[5] Schwankner. "Example hash speeds for 8x h200 graphics cards." Accessed: 2025-09-28. [Online]. Available: https://gist.github.com/schwankner/087469e25458e0ff3bbc8aae02fae29a (cit. on p. 12).

List of Figures

2.1 awesome.jpg 9
2.2 trippin.jpg 9
2.3 wasted.jpg 10

List of Tables

2.1 Container contents 4

3.1 Time to bruteforce passwords RTX 4090 11
3.2 Password minimum requirements to last a minimum 10 years against consumer grade attack 12

Listings

2.1 container sha256 hash 4
2.2 container hash output 4
2.3 sha256sum command 5
2.4 sha256sum command output 5
2.5 dd command 5
2.6 dd command output 5
2.7 hashcat command 5
2.8 hashcat command output 5
2.9 sha256sum command 7
2.10 sha256sum command output 7
2.11 VeraCrypt mounting 7
2.12 VeraCrypt mounting output 7
2.13 hashes extracted files 7
2.14 hashes extracted files output 8
2.15 create directory 8
2.16 copy files 8
2.17 ls extracted files 8
2.18 ls extracted files output 8
2.19 hashes extracted files 8
2.20 hashes extracted files output 8
2.21 cat secret 9
2.22 cat secret output 9
2.23 cat secret 10
2.24 sha256sum after analysis command 10
2.25 sha256sum after analysis command output 10