# PV181

## LABORATORY OF SECURITY AND APPLIED CRYPTOGRAPHY

ALESSIA MICHELA DI CAMPI (543922@MAIL.MUNI.CZ) & ŁUKASZ CHMIELEWSKI (CHMIEL@FI.MUNI.CZ)

Centre for Research on Cryptography and Security

Usability security

Passwords

Challenges

## INTRODUCTION

## COMPUTER SECURITY

The National Institute of Standards and Technology (NIST) defines Computer Security as:

“Measures and controls that ensure confidentiality, integrity, and availability of the information processed, stored (and communicated) by a computer”

**The so-called “CIA” triad: Confidentiality, Integrity, Availability**

Reference: https://csrc.nist.gov/glossary/term/computer_security

## C.I.A

**Confidentiality:**

*Definition:* Confidentiality ensures that information is **accessible only to those who have the authorized permissions to access it.**

*Objective:* The goal is to prevent unauthorized access, disclosure, or exposure of sensitive information. This principle is crucial for protecting private or classified data.

## C.I.A

**Integrity:**

*Definition:* Integrity focuses on **maintaining the accuracy and reliability of information**. It ensures that data remains unaltered and trustworthy throughout its lifecycle.

*Objective:* The objective is to prevent unauthorized or accidental modifications, deletions, or alterations to data. Integrity is essential for ensuring the trustworthiness of information.

## C.I.A

**Availability:**

*Definition:* Availability ensures that **information and resources are accessible and usable when needed by authorized users**.

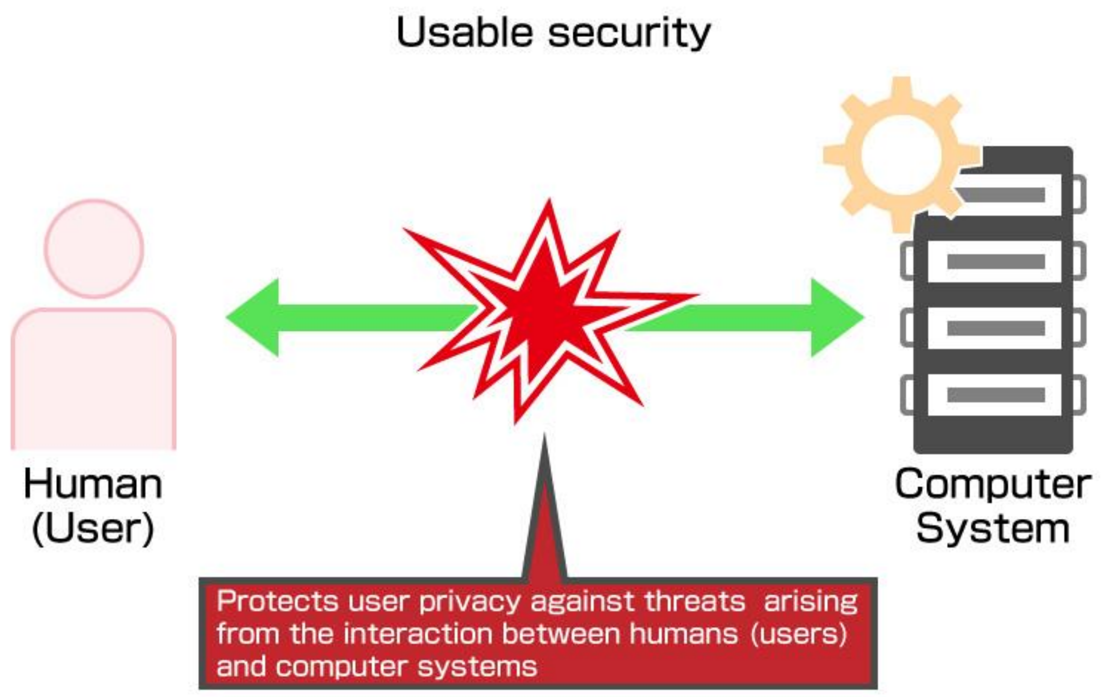
*Objective:* The goal is to prevent disruptions to the availability of data or services. This includes protection against denial-of-service attacks, system failures, or other events that could result in the unavailability of critical resources.

## SECURITY CHALLENGES DUE TO HUMANS

- Weak or Duplicate Passwords
- Lack of Security Awareness
- Phishing and Social Engineering
- Downloading Insecure Software
- Neglecting Patching and Updates
- Unauthorized Access
- Use of Insecure Devices
- Lack of Adequate Backups
- Privacy Violations
- Ignoring Security Indicators

## USABLE SECURITY

Usable security refers to the practice of preventing threats to user security and privacy that arise from the interaction of humans (users) with computer systems. Unlike traditional system and network security, it focuses on users, analyzing their behavior, mental models and decision-making processes.


Usable security
Human (User)
Computer System
Protects user privacy against threats arising from the interaction between humans (users) and computer systems

## USABLE SECURITY AND PASSWORDS

Good security is important online. Passwords are widely used and need to be easy to use and understand.

- **Challenges: Leakage and Guessing**

- *Preventing Leakage:* Addressing unauthorized password disclosure through phishing awareness, secure communication channels, encryption, and the implementation of multi-factor authentication (MFA).
- *Thwarting Guessing Attacks:* Countering unauthorized access attempts by promoting strong, unique passwords, enforcing complexity requirements, encouraging passphrase usage, and implementing account lockout mechanisms.

# USABLE SECURITY AND PASSWORDS

- **Usable Security Strategies:**
- *Intuitive Password Policies:* Designing user-friendly password creation policies.
- *Clear Guidance:* Providing concise instructions on security best practices.
- *Biometrics Integration:* Leveraging biometric technology to streamline authentication.

## WHAT IS A PASSWORD?

A password is a string of characters used to verify the identity of a user during the authentication process. Passwords are typically used in tandem with a username; they are designed to be known only to the user and allow that user to gain access to a device, application or website. Passwords can vary in length and can contain letters, numbers and special characters.

- A minimum length of eight characters and a maximum between 16 to 64 characters. While there is no limit to the length of a password, it does reach a point of diminishing returns.
- Include both uppercase and lowercase letters with case sensitivity. This increases the number of variables at play and, therefore, its difficulty.
- Use at least one number.
- Use at least one special character.
- Avoid using easily guessed elements such as names of children, pet names and birthdays.
- Consider using a password management tool.

## HOW DO PEOPLE CREATE PASSWORDS?

To improve user-friendly security, we need to **understand how people create and manage passwords**.

By studying their behavior, we can tailor security measures to fit natural inclinations, making it easier for individuals to adopt best practices.

This involves promoting the use of secure and easy-to-remember passphrases, personalized security measures, and accommodating user habits.

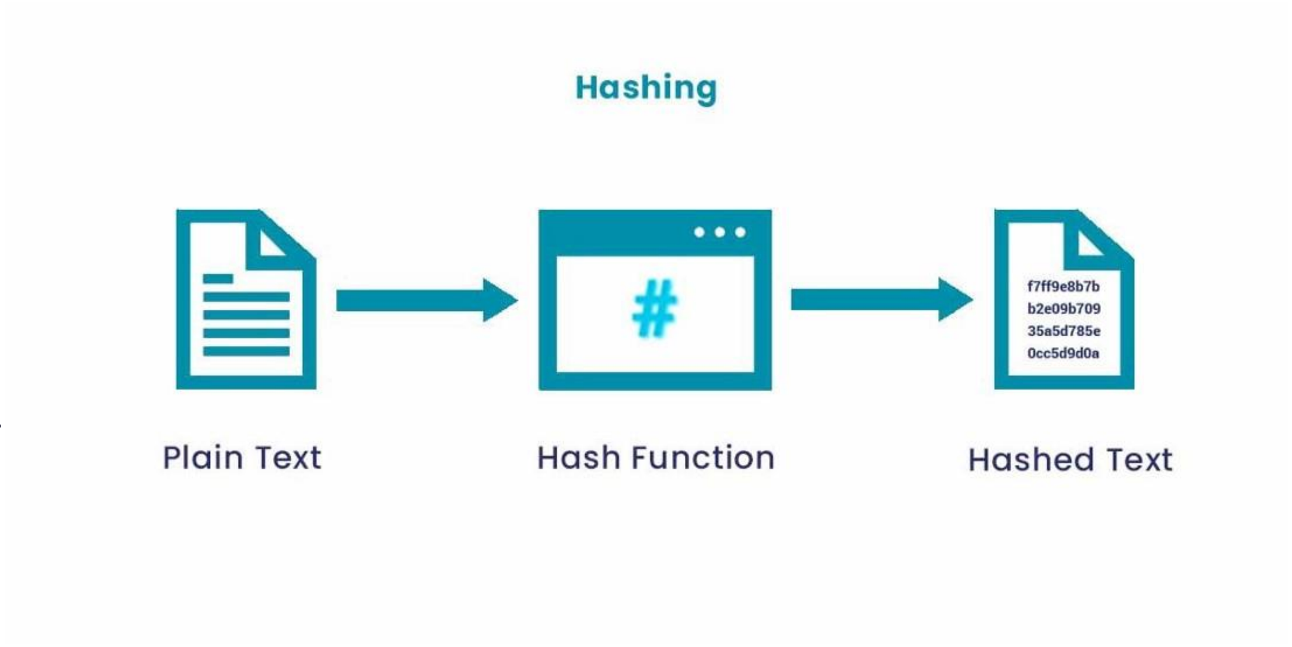
Ultimately, this approach empowers users to contribute to their own digital safety and fosters a more resilient and user-friendly online landscape.

## ATTACKS ON PASSWORDS

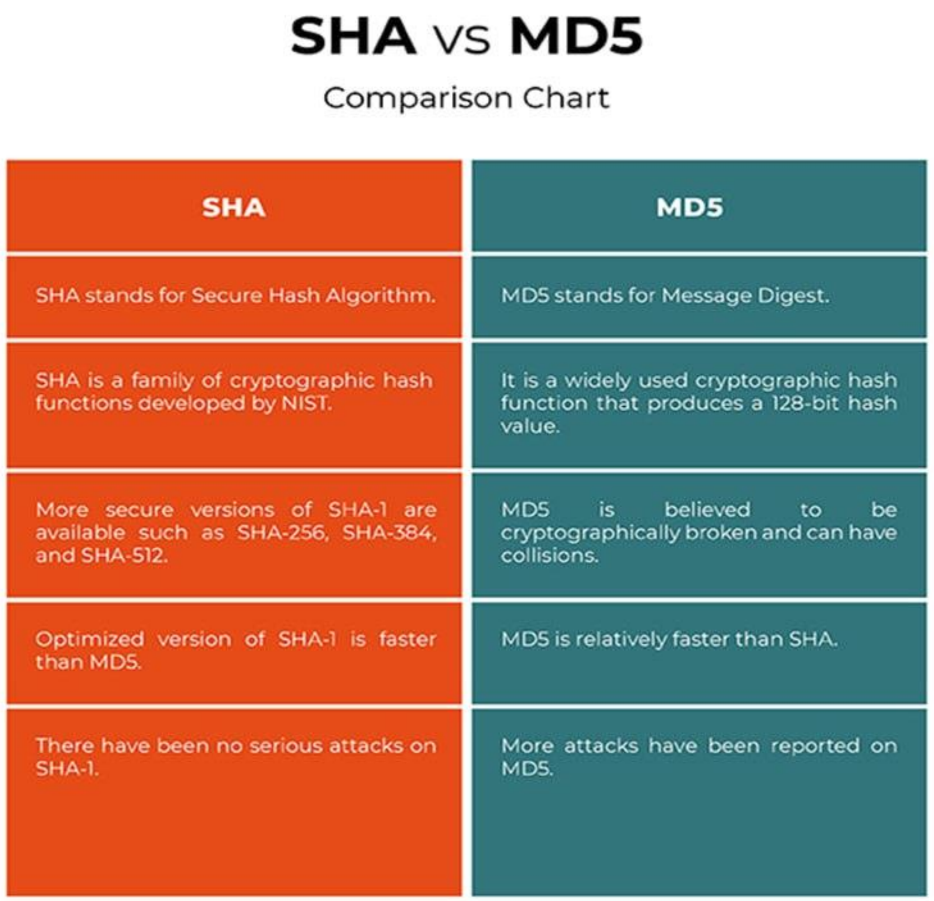
1. **Brute Force Attack:** In this type of attack, an assailant attempts to guess the password by trying all possible combinations of characters. This is a straightforward approach and often requires a considerable amount of time unless the password is weak.
2. **Dictionary Attack:** Attackers use a list of common words or a dictionary of words to try to guess the password. This is more efficient than brute force, as it relies on words that are more likely to be used as passwords.
3. **Phishing:** Phishing attacks involve deceiving users to obtain their login credentials. This can occur through emails, text messages, or fraudulent websites that appear authentic.
4. **Keylogging:** In a keylogging attack, an assailant secretly records all keystrokes made by the user, thereby capturing their login credentials.
5. **Rainbow Table Attack:** This attack involves using precomputed tables of password hashes. Attackers try to match stolen password hashes with those present in the rainbow table to obtain a match.
6. **Credential Stuffing:** Attackers use stolen credentials from one website on another site, taking advantage of the fact that many people reuse the same passwords across multiple platforms.
7. **Social Engineering:** This type of attack involves manipulating individuals through psychological means to divulge confidential information, such as passwords.

## PASSWORD HASHING

Hashing is the process of converting an alphanumeric string into a fixed-size string by using a hash function. A hash function is a mathematical function that takes in the input string and generates another alphanumeric string.



## DIFFERENT HASHES



**SHA vs MD5**

Comparison Chart

| SHA | MD5 |
|---|---|
| SHA stands for Secure Hash Algorithm. | MD5 stands for Message Digest. |
| SHA is a family of cryptographic hash functions developed by NIST. | It is a widely used cryptographic hash function that produces a 128-bit hash value. |
| More secure versions of SHA-1 are available such as SHA-256, SHA-384, and SHA-512. | MD5 is believed to be cryptographically broken and can have collisions. |
| Optimized version of SHA-1 is faster than MD5. | MD5 is relatively faster than SHA. |
| There have been no serious attacks on SHA-1. | More attacks have been reported on MD5. |

## SALTS

Password hashing and why salting is required
Hashing prevents passwords from being exposed or stolen by threat actors, since they are not stored as plaintext. For example, when users create an account with a username and password on a website, their password is hashed and stored in an internal file system in an encrypted form.

Although hashing is a safe way to store passwords compared to storing them in plaintext, the process is not without problems. One limitation is that, if two passwords are the same -- which is quite common because people tend to use common passwords, like "123456" and "password" -- the hashes generated are also identical.

## SALTS

This makes it easier for a bad actor to crack the passwords by brute-force attacks, dictionary attacks or rainbow attacks and compromise the accounts of multiple users, steal their data or cause other problems. To address this challenge, salting is required.

With password salting, a random piece of data is added to the password before it runs through the hashing algorithm, making it unique and harder to crack.

When using both hashing and salting, even if two users choose the same password, salting adds random characters to each password when the users enter them.

```
User Password 1 – orange
Salt 1 – y3Unz
Salt added to password 1 – orangey3Unz
Hash ("orange" + salt) 1 -
024ca8e38b33f9116c151123eb432d20
User Password 2 – orange
Salt 2 – 1Hx$
Salt added to password 2 – orange1Hx$
Hash ("orange" + salt) 2 -
5d41402abc4b2a76b9719d911017c592
```

## CRACKING PASSWORD

Hashcat is a fast password recovery tool that helps break complex password hashes. It is a flexible and feature-rich tool that offers many ways of finding passwords from hashes.

Hashcat offers multiple attack modes for obtaining effective and complex coverage over a hash's keyspace. These modes are:

- Brute-force attack
- Combinator attack
- Dictionary attack
- Fingerprint attack
- Hybrid attack
- Mask attack
- Permutation attack
- Rule-based attack
- Table-Lookup attack (CPU only)
- Toggle-Case attack
- PRINCE attack (in CPU version 0.48 and higher only)

## LAB

Installation:

1. hashcat inside docker
2. hashcat natively
3. hashcat inside virtualbox (or any virtual machine)

See the email from yesterday.

Did everyone do it? Any issues?

## TASK 1: GENERATING ONE-WAY HASHES

As studied in class, passwords are stored in hashed form, using particular cryptographic one-way hash functions. This functions are easy to compute but infeasible to invert: given a hash z there's no efficient algorithm to find a message m such that h(m) is z. Popular one-way hash functions are: sha1, sha2, md5

It is easy to hash passwords "by hand" from the linux shell:

```
$ echo -n 'pwd' | md5sum | tr -d ' -'
9003d1df22eb4d3820015070385194c8
```

Explanation:

- echo -n 'pwd' prints pwd omitting the trailing newline;
- | md5sum takes the output of previous command and computes md5 hash on it;
- | tr -d ' -' removes the trailing ' -' produced by md5sum, leaving just the hash

## TASK 1: GENERATING ONE-WAY HASHES

**Exercise 1**

1. Try yourself with other passwords
2. Try to run the single commands to see what happens
3. Compute the md5 hash of the md5 hash of let_me_go_to_the_next_level . In other words compute md5(md5(let_me_go_to_the_next_level)) . This will give you the password for task 2!

## TASK 2: CRACKING HASHES

We can use hashcat to crack the hash. For example: we easily manage to crack hash 9003d1df22eb4d3820015070385194c8 from the previous example:

```
$ hashcat -m 0 -a 3
9003d1df22eb4d3820015070385194c8
Session..........: hashcat
Status...........: Cracked
Hash.Type........: MD5
Hash.Target......:
9003d1df22eb4d3820015070385194c8
Time.Started.....: Wed Sep 25 15:36:23
2019 (0 secs) Time.Estimated...: Wed Sep
25 15:36:23 2019 (0 secs)
Guess.Mask.......: ?1?2?2 [3]
Guess.Charset....: -1 ?l?d?u, -2 ?l?d, -
3 ?l?d*!$@_, -4 Undefined ...
```

Explanation:
•-m 0 indicates that the hash is MD5
•-a 3 use Brute-Force mode, in which hashcat tries different guesses with fixed shapes
Note: run hashcat -h to see all the options, supported hashes and modes.

## TASK 2: CRACKING HASHES

From the output we can see what shape (mask) was used:

```
Guess.Mask.......: ?1?2?2
[3] Guess.Charset....: -1
?l?d?u, -2 ?l?d, -3
?l?d*!$@_, -4 Undefined
```

Mask is ?1?2?2 where 1 is ?l?d?u (lowercase,digit,uppercase) and 2 is ?l?d (lowercase,digit).Hashcat is brute-forcing all the possible passwords with this shape, such as our password pwd .

Note: hashcat stores the cracked hashes in the .local/share/hashcat/hashcat.potfile file (in some systems it is simply .hashcat/hashcat.potfile). To see the cracked password it is necessary to run the program again with the --show option or to run cat .local/share/hashcat/hashcat.potfile

## TASK 2: CRACKING HASHES

Exercises

- Find a password of length 4 that is easily cracked by simple brute-force (hint: check the masks used by hashcat);
- Find a password of length 4 that is NOT cracked by simple brute-force (hint: check that hashcat does not break the password with masks of length 4). Observe the increasing expected time when the mask length increases. Time depends on the hardware you are using, compare it with mates that have installed hashcat natively, so to observe the speedup using GPUs;
- You can specify on the command line a mask as in hashcat -m 0 -a 3 9003d1df22eb4d3820015070385194c8 ?u?l?l?l?l?l?d?s . This will try all the passwords of length 8 composed of 1 uppercase letter, 5 lowercase, 1 digit and 1 symbol. Charsets are available from hashcat help (option -h )

## TASK 2: CRACKING HASHES

Crack hash ad3f3432a2edf66b5d370958322652c5 exploiting the fact the the password "policy" requires 8 chars of this type: symbol, lowercase, digit, lowercase, uppercase, 3 digits (in this order!). This will give you the password for task 3!

## TASK 3

We can save many hashes in a file and ask hashcat to crack them all. We use >> to redirect and append hashes file hashes.txt :

```
$ echo -n ‘hello' | md5sum | tr -d ' -' >> hashes.txt
$ echo -n '1234' | md5sum | tr -d ' -' >> hashes.txt
$ echo -n 'nice' | md5sum | tr -d ' -' >> hashes.txt
$ echo -n '$password$' | md5sum | tr -d ' -' >> hashes.txt
$ echo -n 'r5g3EF%er' | md5sum | tr -d ' -' >> hashes.txt
$ echo -n 'pwd$' | md5sum | tr -d ' -' >> hashes.txt
$ cat hashes.txt 9003d1df22eb4d3820015070385194c8
81dc9bdb52d04dc20036dbd8313ed055 7c6483ddcd99eb112c060ecbe0543e86
666e549b95a5d5f8963cc7d5145347e0 7fd42a41096c71fc69fc0edc38ec4797
5342cd726d90d1e80e13cc5ec9358560 4684773b5d4e81256fae19e2cb799874
```

## TASK 3

The hashcat invocation is the same as before. We just pass the filename instead of the hash. If we stop it after a few seconds we see that hashcat is able to immediately break the short passwords.

```
$ hashcat -m 0 -a 3  hashes.txt
... ctrl-C
$ hashcat -m 0 -a 3  hashes.txt --show
7c6483ddcd99eb112c060ecbe0543e86:nice
9003d1df22eb4d3820015070385194c8:pwd
81dc9bdb52d04dc20036dbd8313ed055:1234
```

Explanation:

pwd fits the mask ?1?2?2 where 1 is ?l?d?u (lowercase, digit, uppercase) and 2 is ?l?d (lowercase, digit);
1234 and nice fit the similar, longer mask ?1?2?2?2 ;
$password$ , r5g3EF%er , riccardo123 are too complex to find quickly by simple brute-force;
Interestingly, pwd$ even if simple and short is not found because it does not fit the above mask ?1?2?2?2 .

## TASK 3

The file hashes.txt contains one weak password that will let you access Task 4!

```
wget https://secgroup.dais.unive.it/wp-content/uploads/2019/09/hashes2.txt
```

# TASK 4

Password cracking becomes much more effective when combined with dictionaries of know/easy passwords. A famous one is the rockyou list that came from the SQLi attack on rockyou.com.

The ~32M passwords were stored in the clear instead of hashed! File is available here (https://github.com/brannondorsey/naive-hashcat/releases/download/data/rockyou.txt) but is already in the docker image, if you are using it.

We can run again the cracking using the rockyou.txt dictionary as follows:

```
$ hashcat -m 0 -a 0 hashes.txt rockyou.txt ...
INFO: Removed 3 hashes found in potfile. ...
$ hashcat -m 0 -a 0 hashes.txt rockyou.txt --show
666e549b95a5d5f8963cc7d5145347e0:$password$
7c6483ddcd99eb112c060ecbe0543e86:nice
9003d1df22eb4d3820015070385194c8:pwd
81dc9bdb52d04dc20036dbd8313ed055:1234
5342cd726d90d1e80e13cc5ec9358560:riccardo123
```

Explanation:

Option -a 0 is used to run a dictionary attack: the dictionary file rockyou.txt is specified on the command line; hashcat remembers the hashes already cracked in file .hashcat/hashcat.potfile and does not try to crack them in next invocations. This is why it outputs INFO: Removed 3 hashes found in potfile . This is an important feature that allows for incremental cracking using different techniques; Using the dictionary we are able to crack two of the longer passwords: $password$ , riccardo123 but not yet r5g3EF%er and pwd$ .

## TASK 4

Exercise

- Remove .hashcat/hashcat.potfile and run again the dictionary attack to see what passwords are cracked;

- As done for previous task, you can specify on the command line a mask as in hashcat -m 0 -a 3 hashes.txt ?u?l?l?l?l?l?d?s . This will try all the passwords of length 8 composed of 1 uppercase letter, 5 lowercase,  1 digit and 1 symbol. Try to crack the missing passwords r5g3EF%er and pwd$ by specifying a suitable mask. (For  r5g3EF%er you will might need a GPU to crack it fast. Look at  the estimated time in the hashcat status to have an idea of the worst case.  In our experiments we could crack it in a matter of minutes in Docker.)

## TASK 4

Since you have root privileges in Docker you can create test users with weak/strong passwords and try to crack them. Use adduser  to create a few users and then try cracking their passwords with brute-force and dictionary:

```
(... after users have been added you will find their hashed passwords in
/etc/shadow ) $ cat /etc/shadow ...
paperino:$6$wClZL8Ij$T5o95har.hxfeBd3vtQCj/ivuVQPdfLxEnNhX3rhoYkK7irQVv/imbGPD9
rBhApTKj3c41hZ9cqC68Xo7eLNi1:18165:0:99999:7:::
pippo:$6$1KwvveJV$czDJksVHD2Ix56mN0HFsQdLWiEm811cuxrk9RCYWtUdFGnxXpJ27e3isdphgY
n3LZvWRQy7X33v4QEaI0phTE1:18165:0:99999:7:::
zio:$6$MSdL2emp$CHuArXopoKct6VWB7YXTO1q0VGpuWaUWEYzrI641QcxZFqZFKx7kAR8DQJIq/gR
tnQJOHqtrDvLuuGZqF.Ule/:18165:0:99999:7:::
$ hashcat -m 1800 -a 3 /etc/shadow # brute-force
$ hashcat -m 1800 -a 0 /etc/shadow rockyou.txt # dictionary
```

From hashcat help (-h  option) we find the suitable mode to pass to -m , in this case 1800, corresponding to sha512crypt:

1800 | sha512crypt $6$, SHA512 (Unix) | Operating Systems

Recall that the hash is iterated. This will make cracking much slower. Moreover, since there are salts, each hash needs to be recomputed for each given salt (no precomputation is possible).

## TASK 4

If you didn’t do it yet and you have a GPU, you can install hashcat natively and re-run the experiments to appreciate the speedup you can gain with highly parallelised hardware. In fact, brute-forcing amounts to repeating the same task many many times, which fits very well GPU parallelism!

## ASSIGNMENT 10 - RULES

This is a programming assignment. Please upload your scripts/code and the required analysis via the course webpage.

The deadline for submission is Dec. 28, 2023, 23:59.

-3 points for each started 24h after the deadline.

Your commands/scripts should be contained in reported in one text file with explanations. Please name the submission file as <uco_number>_hw10.zip. Put there both the commands/scripts, the analysis document, and all data produced during analysis (as long as the size is reasonable).

The scripts/code must contain comments so that it is reasonably easy to understand how to run the script for evaluating each answer.

## ASSIGNMENT 10 - TASKS

Hashcat allows a so-called rule-based attack.

- What we ask you to do is to perform the attack on a hashes file (choose one, maybe Battlefield) with a set of rules (for at least 5 sets of rules available), use the --debug-file=matched.rule (as stated in this guide https://www.4armed.com/blog/hashcat-rule-based-attack/, you can change the name of the output file). Analyse the time taken and the percentage of cracked passwords. [4 points]
- You can also create a script and send to hashcat a signal to have the information about the cracked percentage over the time and create a graph to show the behaviour. [BONUS +2]
- Report an analysis on how the output changes depending on the size of the rules file. [2 point]
- Create for each analysis most used rules file (for each rules set used). [2 point]
- Try to create your own rule set that intelligently uses the most used rules files (it does not have to perform better) and analyse the behaviour. [2 points]
- [SUPER BONUS! +2] What will happen if you try to combine attacks? Rules and masks for example.

- Any improvements or ideas are welcome!

THAT’S ALL! WE HOPE YOU ENJOYED PASSWORD CRACKING!