Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies in Bratislava

Principles of information security

Freely distributed password cracking tools

Ákos Lévárdy

April 30, 2024

Contents

1 Topic of the project: Freely distributed password cracking tools

In my topic I will be focused on freely distributed password cracking tools. Firstly, I will describe how passwords are encrypted or hashed. Passwords are used everywhere for user authentication, which is a widely adopted method due to its intuitive logic and ease of implementation for developers.

Despite their popularity, passwords pose security risks as password crackers are crafted to extract credentials from compromised data obtained through breaches or hacks.

2 What is password cracking?

Most password-based authentication systems do not store a user's actual password. Instead, they store a password hash, which is the result of sending the password and a random value called a salt through a hash function.

I will compare different hash functions and describe how they work when in use. Basically, hash functions are designed to be one way. This means that it is very difficult to get the original password from the hashed output.

It is important to know that hash functions are producing the same output for the same input. Comparing two hashes of a password is eventually the same as comparing two real passwords.

2.1 Password cracking tools

There are many different tools for password cracking.
The most popular ones are:

- Hashcat
- John the Ripper
- Brutus
- Wfuzz
- THC Hydra
- Meduza
- RainbowCrack
- OphCrack
- L0phtCrack
- Aircrack-ng

I will be comparing two of these password cracking tools – Hashcat and John the Ripper, analyze them, see how they work and then compare their performance.

3 Types of attacks on passwords

There are various types of attacks on passwords, and they can be categorized into different methods based on their techniques.
Most common types of password attacks:

- Brute Force Attack - This involves trying all possible combinations of passwords until the correct one is found.

- Dictionary Attack - Attackers use a predefined list of common words, dictionaries to try and guess the password.
- Rainbow Table Attack - A precomputed table containing the hash values of commonly used passwords. This attack compares these hashes with the target system's stored password hashes to find a match.
- Phishing – This attack tries to trick individuals into revealing their passwords by posing as a trustworthy entity.
- Keylogging - Malicious software or hardware records keystrokes without the user noticing it and capturing sensitive information such as usernames and passwords.
- Man-in-the-Middle (MitM) Attack - Intercepting communication between two sides to capture information, including passwords and usernames.
- Shoulder Surfing - Observing someone entering their password without their knowledge.

4 Progress report 1

For the first progress report I will analyse the different hashing algorithms and compare the types of attacks on passwords. I will write about the history of some tools for password cracking and about how to defend our passwords with password managers and other tools.

5 Progress report 2

For the second progress report I will test two password cracking tools and then compare how they work and their performance. I will use John the Ripper and Hashcat on Kali Linux after I set up a Virtual box for it.

6 Authentication

For authenticating a user there are 2 different types. One-factor authentication relies solely on one type of information for verification (usually a password), two-factor authentication enhances security by requiring two different types of information, making it harder for unauthorized users to gain access even if they have obtained one factor (password).

- One-Factor Authentication (1FA):
  - In one-factor authentication, only one type of information is required to verify the identity of the user.
  - Typically, this involves something the user knows, such as a password or a PIN.
  - It's the simplest form of authentication and is commonly used in many basic systems.
- Two-Factor Authentication (2FA):
  - Two-factor authentication requires two different types of information to authenticate the user.
  - These factors usually fall into one of three categories: something the user knows (password), something the user has (smartphone or a security token), or something the user is (biometric data like fingerprints or facial recognition).
  - By requiring two factors, 2FA adds an extra layer of security beyond just a password, making it more difficult for unauthorized users to gain access.
  - Common implementations of 2FA include receiving a text message with a code, using an authenticator app, or using biometric authentication along with a password.

7 Password-Hashing Schemes - PHS

Everyday tasks involve the use of computers, and many people use the services that are offered. The most popular method for authenticating users on the web is one-factor authentication, which consists of a password and a username.

Unfortunately, attackers take advantage of weak password management procedures to reveal users' credentials, which hurts both users and providers. In the majority of these cases, the user data was either processed directly by a cryptographic hash function or kept in cleartext.

7.1 Password-Hashing procedures

We use password-hashing procedures to secure this user-related data. Currently, the Password-Based Key Derivation Function 2 (PBKDF2) standard primitive is in use, however other popular schemes like Blowfish cipher - Bcrypt and Scrypt are also in use. The development of parallel computing has made it possible for multiple password-hash cracking attempts.

For the purpose of developing new, widely-acceptable password-hashing algorithms that are more safe and efficient, the global cryptography community organized the Password Hashing Competition (PHC). PHC improved our understanding of hashing passwords. Security flaws were discovered through more investigation, and new methods were subsequently developed.

7.2 Background Theory

A password is a secret that is straightforward for the user to remember and consists of just a few printed characters. In computer systems, passwords are frequently used for user authentication. For each account, the system keeps track of the user's identity and password. The individual then uses this information to log in and access the service.

The creation of cryptographic keys is another use for passwords. Based on an input password, the Key-Derivation Functions (KDF) generates a variety of cryptographic keys. KDFs primarily serve for session data encryption, and cryptographic keys are applied by encryption/decryption functions.

Cryptographic keys play a critical role in ensuring the security of data and communications in cryptographic systems. They enable encryption, decryption, digital signing, and authentication, helping to protect sensitive information from unauthorized access, tampering, or interception.
Proper key management practices are essential to maintaining the effectiveness and security of cryptographic systems.

Encryption converts plaintext into ciphertext using an encryption algorithm and a key, while decryption reverses this process to recover the original plaintext. These processes play a crucial role in protecting data confidentiality and integrity, ensuring that only authorized users can access and understand the encrypted information.

7.3 Key-Derivation Function

A key derivation function (KDF) is a cryptographic tool used to derive one or more secret keys from a single, fixed-length secret value, such as a master key or a passphrase.

- The KDF takes as input the secret value and possibly some additional parameters, such as a salt (random value) or context information.
- The KDF expands the input secret into one or more longer keys. This expansion process typically involves applying a cryptographic hash function or a more complex cryptographic algorithm repeatedly to the input, potentially mixing in the salt or other parameters. The goal is to generate keys that are unpredictable and have high entropy (randomness).

- The output of the KDF is one or more derived keys, which can be used for various cryptographic purposes, such as encryption, authentication, or generating other cryptographic parameters.
- Security Properties:
  A strong KDF should include the following security features:
  - Key Derivation: Both the derived keys and the input secret must be computationally independent of one another.
  - Resistance to Attacks: The KDF should resist various cryptographic attacks, such as brute-force attacks (trying all possible inputs) and pre-image attacks (deriving the input from the output).
  - Salt and Context Sensitivity: Incorporating a salt and context information can enhance the security of the derived keys by making them unique to each application or usage scenario.
- Essentially, a key derivation function offers a safe method for producing cryptographic keys from a single secret value, guaranteeing the confidentiality and integrity of the derived keys and making them appropriate for usage in cryptographic protocols and applications.

7.4 Password Security Risks and Vulnerabilities

When people create passwords for their accounts, they often use simple ones that are just eight characters long. These passwords are not very secure because they're easy to guess.

Hackers can use a method called exhaustive search to try every possible combination of characters until they find the right password. Once they do, they can access the account just like the real user. Even graphical passwords, where you draw patterns or pick images, can be easy for hackers to figure out because they don't provide much security.

7.5 Password-Based Key Derivation Function 2 - PBKDF2

PBKDF2 is a widely used key derivation function that's designed to derive cryptographic keys from a password or passphrase. Hashing-based techniques are the standard option for establishing password protection. The primary methods for making KDFs possible are keyed-hash message authentication codes (HMACs) and cryptographic hash functions that translate a password into one or more secret keys.

PBKDF2 uses pseudo-random functions (PRF), commonly implemented by HMACs. HMAC is a specific type of PRF that's widely used in cryptographic applications for generating message authentication codes. The SHA-256 function is the standard internal hash option for the HMAC.

A pseudo-random function (PRF) is a mathematical function that produces seemingly random outputs from given inputs. Unlike truly random functions, PRFs are deterministic, meaning that the same input always produces the same output. PRFs are designed to mimic the properties of random functions while being efficiently computable.

The scheme parses the password and the salt as inputs. Attacks utilizing precomputed data, such as dictionary attacks (which test hundreds of plausible combinations to determine the passphrase) and rainbow-table attacks (which make use of tables with precomputed hashes), are made more resilient by the salt. It typically has 8 bytes in size.

PBKDF2 steps:

- The salt and the password are initially processed by the HMAC.
- The derived key is then generated by processing the data multiple times.
- Although 1000 iterations is the minimum required amount, given that the guideline was set in 2000, it is no longer deemed sufficient.
- The password and the previous HMAC result are parsed by the HMAC at each repetition.
- The HMAC outcomes are then XORed.
- The outcome of the XOR operation in the last round is the final hash.

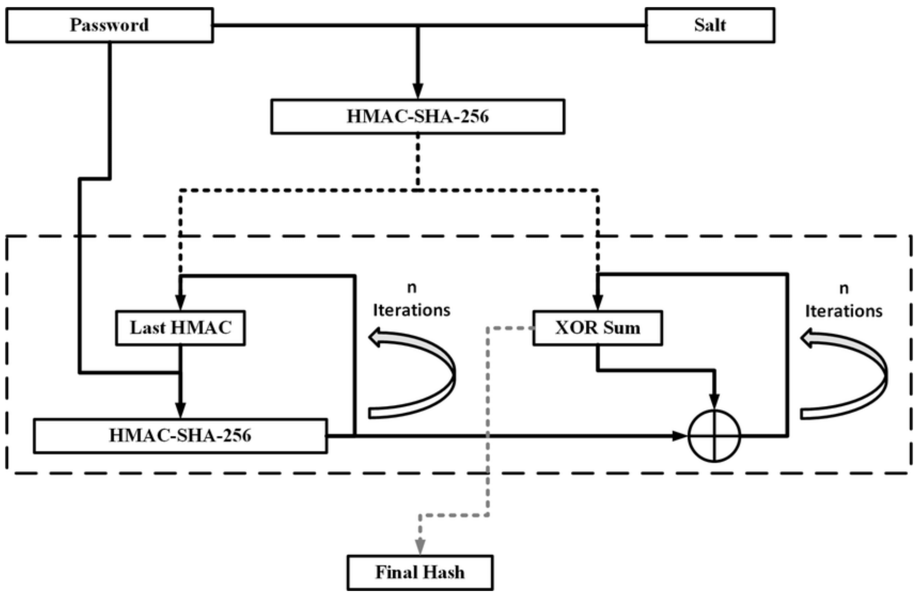


Figure 1: PBKDF2

The effective hardware-based cracking has a downside because it can be created as a small hardware implementation with little RAM requirements. Therefore, on GPUs, FPGAs, and ASICs, low-cost brute-force attacks are possible. In roughly one week, the most rapid attacks can crack about 65% of common passphrases. Approximately 245,000 passwords are obtained per second by attacks on multi-FPGA devices.

7.6 Bcrypt

Bcrypt is the chosen password hashing system for the BSD operating system. It makes use of the Blowfish block cipher. By default, it processes passwords that are 56 bytes long and produces hashes that are 24 bytes long. The number of iterations increases exponentially as the computational power of potential attackers grows, ensuring adequate protection against brute-force attacks. Additionally, a 16-byte salt value is employed to strengthen defenses against attacks relying on rainbow tables.

Bcrypt steps:

- Initialization: Bcrypt starts by setting up the Blowfish cipher with a cost parameter, a salt (a random value), and your password. This sets up a complex system of encryption.
- Key Schedule: Bcrypt then creates a set of subkeys based on your password. These subkeys are mixed with the original password in a special way.
- Encryption: The salt is encrypted using the key schedule. This process repeats several times, making it harder for attackers to reverse-engineer the password.
- Magic Value Encryption: A special value is encrypted multiple times using Blowfish. This further scrambles the data.
- Final Hash: The final hash is created by combining the salt, cost, and the result of the encryption process

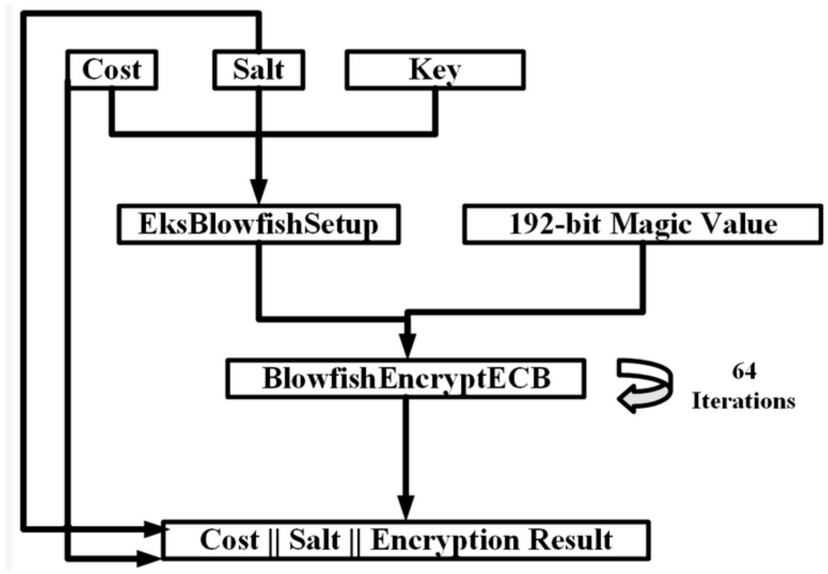


Figure 2: Bcrypt

The method uses 4KB of RAM and outperforms PBKDF2 in terms of blocking parallel cracking. While the best attack discovers 20,583 passwords per second, the least expensive attack only manages 1207 cracks per second at a cost of $99.

7.7 Scrypt

Scrypt is designed to be a memory-hard password hashing scheme, which means it makes it difficult for attackers to use large amounts of memory when trying to crack passwords.

Scrypt steps:

- It is based on the PBKDF2 algorithm and the Salsa stream cipher.
- It has features like MFcrypt (Memory-Hard Function), ROMix (Rounds Mixer), BlockMix, and SMix (Sequential Memory-Hard Function).
- To make sure all of the random values are kept in memory, ROMix creates a bunch of them and shuffles them around.
- BlockMix facilitates this procedure by utilizing another version of the Salsa cipher.
- SMix uses one or more ROMix functions to further complicate matters.
- The number of ROMix functions (parallelization parameters) determines how much time and space the process takes.
- MFcrypt is the final step, where the key-derivation function combines the mixing function with PBKDF2 using the SHA-256 hash function.

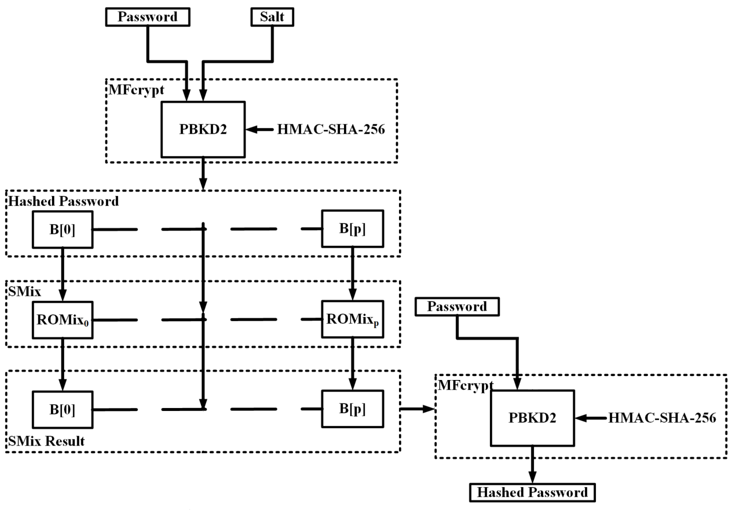


Figure 3: Scrypt

8 Password cracking tools

Passwords are created by humans using recurring patterns. These patterns are taken advantage of by data-driven password cracking techniques that rely on massive password breach collections.

Nowadays, it is common practice to simulate password cracking in order to determine the strength of a password. Numerous algorithms for breaking passwords are probabilistic, which means they build a model and then give each potential password a probability. The strength of a password is proportional to the number of passwords with higher probability based to that model, assuming the attacker logically guesses passwords in descending order of likelihood.

However, probabilistic tools are rarely used by real-world attackers; instead, they mostly employ programs like Hashcat and John the Ripper (JtR).

They give practical explanations. In offline attacks, the time it takes to make and check a guess includes the time to create a guess, hash it, and see if it matches the target password. Probabilistic algorithms are good at guessing passwords, but they take a lot of computer power to generate each guess.

So, for most fast hash functions, it's quicker to crack passwords using different software tools. While the likelihood of guessing a password with a probabilistic model might match the order of guessing with software tools, it's not always accurate.

The mangled-wordlist assaults of JtR and Hashcat are the most frequently used and intellectually engaging. These attacks take advantage of the fact that passwords typically vary in subtle, predictable ways; for example, a word may have a digit added to it by one person, while the same word may have a symbol added by another. In a mangled-wordlist attack, the attacker generates a rule list of mangling rules (such as appending a digit and replacing ”s” with ”$”) written in a transformation language given by the tool, together with a wordlist comprising popular passwords and natural language content. Every mangling rule is applied to every word in the order that the input lists specify during the whole attack.

8.1 HASHCAT

HASHCAT
Password Hash Cracking Tool

Hashcat is a popular password cracking tool used by security professionals and attackers to recover lost or forgotten passwords or to test the strength of passwords.

Types of attacks:

- Brute-Force Attack: Hashcat can perform a brute-force attack, where it systematically tries every possible combination of characters until it finds the correct password. This method is effective but can be time-consuming, especially for longer and more complex passwords.
- Dictionary Attack: Instead of trying every possible combination, Hashcat can use a predefined list of commonly used passwords, called a dictionary, to attempt to crack passwords. This method is faster than brute-force but relies on the likelihood that the password is contained in the dictionary.
- Combination Attack: Hashcat can also combine both brute-force and dictionary attacks. It starts with the words in the dictionary and then appends, prepends, or replaces characters to create variations of those words, effectively expanding the search space.
- Rule-Based Attack: Hashcat supports rule-based attacks where users can define custom rules to manipulate the dictionary words before attempting them as passwords. For example, rules can be created to add numbers or special characters to the end of dictionary words.
- Mask Attack: This method allows users to specify a mask or template for the password, defining the possible characters and their positions. Hashcat then generates and tests passwords based on this mask.
- Hybrid Attack: Hashcat can combine different attack modes to increase the chances of cracking passwords. For example, it can perform a dictionary attack with rule-based mutations.

8.2 John the Ripper - JTR

John the Ripper is another widely used password cracking tool utilized by security professionals and attackers alike. It operates using similar techniques as Hashcat, employing various methods to crack passwords. It is a versatile and powerful tool for cracking passwords, offering various techniques to suit different scenarios and types of passwords.

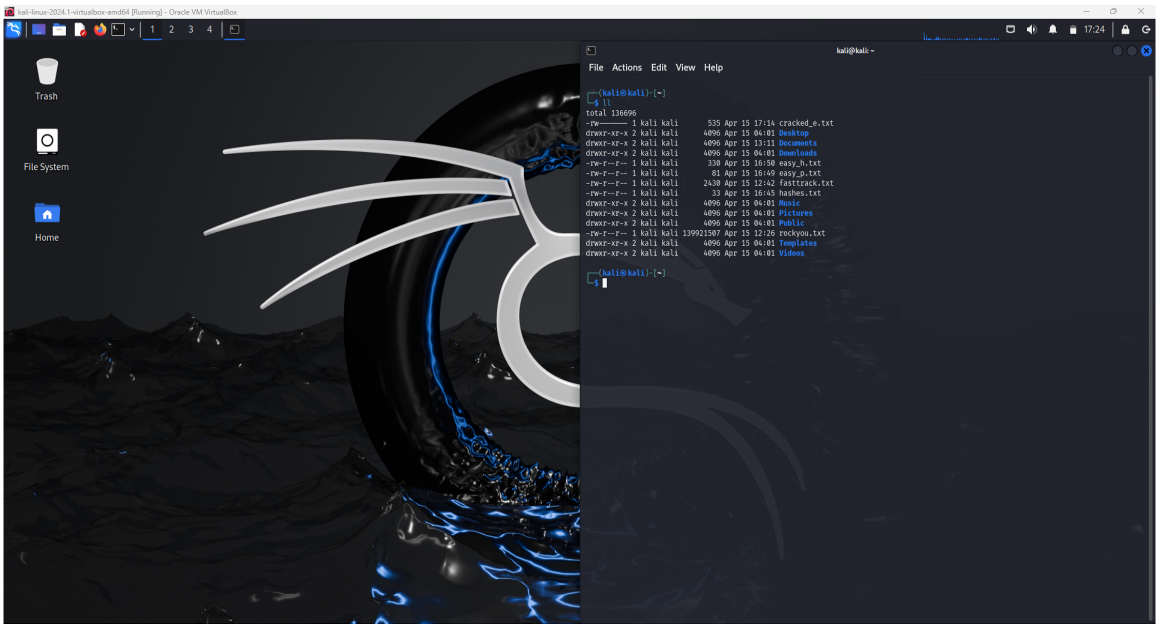
Types of attacks:

- Dictionary Attack: John the Ripper can perform a dictionary attack by trying words from a predefined list of common passwords or from a custom dictionary file. It systematically checks each word in the dictionary against the hashed passwords to see if there's a match.
- Brute-Force Attack: Like Hashcat, John the Ripper can also conduct brute-force attacks by systematically trying every possible combination of characters until it finds the correct password. This method is effective but can be time-consuming, especially for longer and more complex passwords.
- Rule-Based Attack: John the Ripper supports rule-based attacks where users can define custom rules to manipulate the dictionary words before attempting them as passwords. For example, rules can be created to add numbers or special characters to the end of dictionary words.
- Incremental Mode: This mode in John the Ripper generates passwords by incrementally increasing their length and complexity based on predefined rules. It starts with short passwords and gradually builds up to longer ones, trying different combinations along the way.
- Hybrid Attack: John the Ripper can combine different attack modes, such as dictionary attacks with rule-based mutations or brute-force attacks with specific character sets.
- Single Crack Mode: This mode allows John the Ripper to crack password hashes with just one attempt, using various techniques like dictionary, brute-force, and incremental attacks

9 Testing - Password cracking

Environment used:	Oracle VM VirtualBox
OS used:	Kali Linux
Version:	kali-linux-2024.1-virtualbox-amd64
Type of cracking:	Offline
Base Memory:	4096 MB
Processors:	2
Boot Order:	Hard Disk, Optical
Acceleration:	Nested Paging, PAE /NX, KVM Paravirtualization

View of Kali Linux:


kali-linux-2024.1-virtualbox-amd64 [Running] - Oracle VM VirtualBox
Trash
File System
Home
kali@kali: ~
File
Actions
Edit
View
Help
17:24

```
┌──(kali㉿kali)-[~]
└─$ ll
total 136696
-rw------- 1 kali kali       535 Apr 15 17:14 cracked_e.txt
drwxr-xr-x 2 kali kali      4096 Apr 15 04:01 Desktop
drwxr-xr-x 2 kali kali      4096 Apr 15 13:11 Documents
drwxr-xr-x 2 kali kali      4096 Apr 15 04:01 Downloads
-rw-r--r-- 1 kali kali       330 Apr 15 16:50 easy_h.txt
-rw-r--r-- 1 kali kali        81 Apr 15 16:49 easy_p.txt
-rw-r--r-- 1 kali kali      2430 Apr 15 12:42 fasttrack.txt
-rw-r--r-- 1 kali kali        33 Apr 15 16:45 hashes.txt
drwxr-xr-x 2 kali kali      4096 Apr 15 04:01 Music
drwxr-xr-x 2 kali kali      4096 Apr 15 04:01 Pictures
drwxr-xr-x 2 kali kali      4096 Apr 15 04:01 Public
-rw-r--r-- 1 kali kali 139921507 Apr 15 12:26 rockyou.txt
drwxr-xr-x 2 kali kali      4096 Apr 15 04:01 Templates
drwxr-xr-x 2 kali kali      4096 Apr 15 04:01 Videos

┌──(kali㉿kali)-[~]
└─$ 
```

- Kali Linux has more wordlists already stored and the largest of them is the rockyou.txt wordlist found in: /usr/share/wordlists/rockyou.txt.gz.
- After unzipping this file we have a wordlist with over 14 million (14,344,392) common passwords people use/used.
- The ”rockyou.txt” wordlist is compiled from data breaches, primarily from a significant breach of the social networking site RockYou in 2009. During this breach, hackers gained access to RockYou’s user database, which contained millions of usernames and passwords stored in plaintext. The breach exposed the weak security practices of the time, as RockYou stored passwords without proper encryption.
- Hashcat and John the Ripper are preinstalled with Kali, so no other installing is needed.

9.1 Hashcat

- Version: hashcat (v6.2.6)
- hashcat - Advanced CPU-based password recovery utility

- How to use Hashcat:

hashcat [options] hashfile [mask—wordfiles—directories]
-a, –attack-mode=NUM
-m, –hash-type=NUM
–potfile-disable, hashcat puts already cracked passwords in a pot file (basically saying: i dont need to crack these, i already cracked them, i am going to save time)

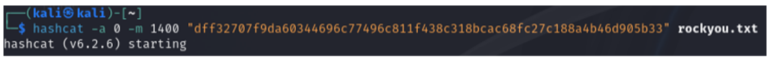
- Attack modes:

- 0 = Straight
- 1 = Combination
- 3 = Brute-force
- 6 = Hybrid Wordlist + Mask
- 7 = Hybrid Mask + Wordlist

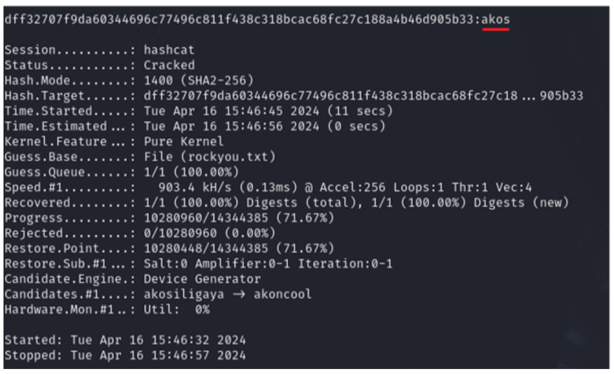
- Hash types: total 93 hash types to choose from

- for example: MD5, HMAC-MD5, SHA1, NTLM, SHA256, Plaintext ...

- Example usage - cracking password hash, hash type - SHA256, mode - straight:



```
┌──(kali㉿kali)-[~]
└─$ hashcat -a 0 -m 1400 "dff32707f9da60344696c77496c811f438c318bcac68fc27c188a4b46d905b33" rockyou.txt
hashcat (v6.2.6) starting
```



```
dff32707f9da60344696c77496c811f438c318bcac68fc27c188a4b46d905b33:akos

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 1400 (SHA2-256)
Hash.Target......: dff32707f9da60344696c77496c811f438c318bcac68fc27c18 ... 905b33
Time.Started.....: Tue Apr 16 15:46:45 2024 (11 secs)
Time.Estimated ...: Tue Apr 16 15:46:56 2024 (0 secs)
Kernel.Feature ...: Pure Kernel
Guess.Base.......: File (rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:   903.4 kH/s (0.13ms) @ Accel:256 Loops:1 Thr:1 Vec:4
Recovered........: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.........: 10280960/14344385 (71.67%)
Rejected.........: 0/10280960 (0.00%)
Restore.Point....: 10280448/14344385 (71.67%)
Restore.Sub.#1 ...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1....: akosiligaya → akoncool
Hardware.Mon.#1..: Util:  0%

Started: Tue Apr 16 15:46:32 2024
Stopped: Tue Apr 16 15:46:57 2024
```

- Different lengths for the passwords and different types of hashes used:

PASSWORD	LENGTH	MD5	NTLM
apple$	6	bbfb328d6962d6161d22c42182644ae8	0B4839664168D318107E1F2B70E00C87
sunrise7	8	6f3e5bbb3910d2f04cdf5d85b3a25a72	8855EC8615C785749896F0634F416EB5
moonlight#	10	3b38ed7ead43e363f7d7ec58f67a0644	8DD11A8AA7B34BDAF6769228798AC299
starrysky12	11	54ce50d4a1f51f96b80777991431b3ad	71F1F0EC37BD6CECC20E331F0FF72196
galacticcenter$	15	dfba3853df78358c3c9061c2aac508b6	CD0959434137CDC36168F0B7F2AE7789

PASSWORD	LENGTH	SHA-256
apple$	6	c68e29247ad5d104bbcc5eb57875964e009ec2eb67536df55e8ae8e92be3c812
sunrise7	8	fbba4f250fdf1e269661f29a5d5dd4e604621510c2288eeddc58308f603ddf62
moonlight#	10	4cea2d19b392c913e56884fbde565bc5a1bcd9e17d623b1e2950952c7ecf1423
starrysky12	11	fa0a7fea29a910d4c1f88007995ff3d14200cab2d0ec12d6e4439253973c708d
galacticcenter$	15	4b1107e19af2bff02942c0035126d9c7ebfeefdfd77e78130784b6a3c43bf99a

- Testing the passwords with hashcat:

HASHCAT - Cracking time in miliseconds				
PASSWORD	LENGTH	MD5	NTLM	SHA-256
apple$	6	0.46	0.12	0.64
sunrise7	8	0.56	0.72	0.90
moonlight#	10	1.10	0.90	1.10
starrysky12	11	1.17	1.19	1.44
galacticcenter$	15	1.46	1.34	1.46

- Testing the passwords with john the ripper:

JOHN THE RIPPER - Cracking time in miliseconds				
PASSWORD	LENGTH	MD5	NTLM	SHA-256
apple$	6	0.42	0.20	0.68
sunrise7	8	0.93	0.44	0.75
moonlight#	10	0.94	1.26	0.78
starrysky12	11	1.38	1.30	1.76
galacticcenter$	15	1.41	1.65	1.82

9.2 John the Ripper

- Version: John the Ripper 1.9.0-jumbo-1+bleeding-aec1328d6c
- john - a tool to find weak passwords of your users

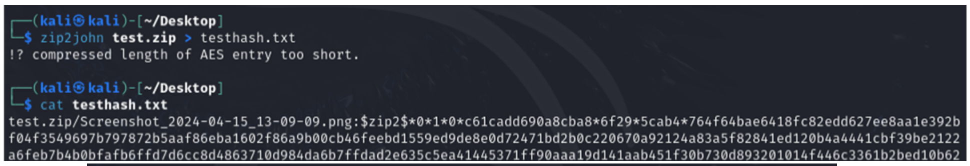
- How to use John the Ripper:

john [options] password-files
–format:NAME, Allows to override the ciphertext format detection. (Currently, valid format names are DES, BSDI, MD5, BF, AFS, LM)
–format=raw-[hash type]
–[MODE]

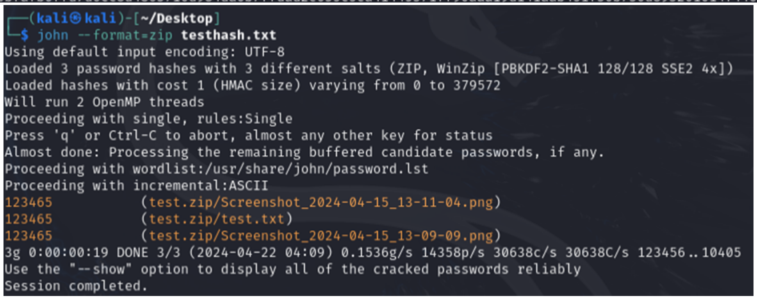
- Attack modes:

- –single
- –wordlist:FILE
- –incremental[:MODE]

- Example usage - cracking a password hash for a .zip file:



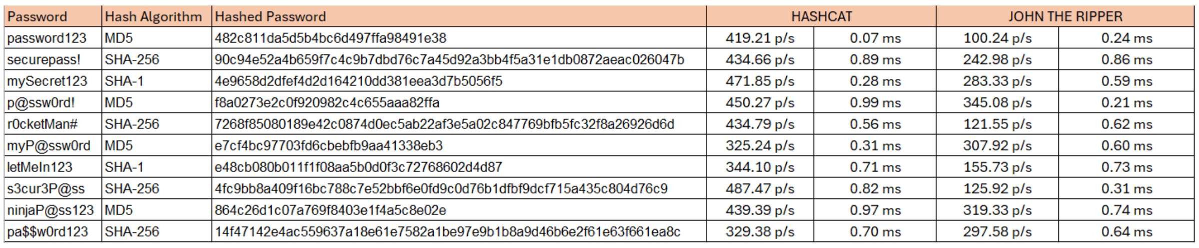
```
┌──(kali㉿kali)-[~/Desktop]
└─$ zip2john test.zip > testhash.txt
!? compressed length of AES entry too short.

┌──(kali㉿kali)-[~/Desktop]
└─$ cat testhash.txt
test.zip/Screenshot_2024-04-15_13-09-09.png:$zip2$*0*1*0*c61cadd690a8cba8*6f29*5cab4*764f64bae6418fc82edd627ee8aa1e392b
f04f3549697b797872b5aaf86eba1602f86a9b00cb46feebd1559ed9de8e0d72471bd2b0c220670a92124a83a5f82841ed120b4a4441cbf39be2122
a6feb7b4b0bfafb6ffd7d6cc8d4863710d984da6b7ffdad2e635c5ea41445371ff90aaa19d141aab451f30b730d893201014f446c3361b2bed10b62
```



```
┌──(kali㉿kali)-[~/Desktop]
└─$ john --format=zip testhash.txt
Using default input encoding: UTF-8
Loaded 3 password hashes with 3 different salts (ZIP, WinZip [PBKDF2-SHA1 128/128 SSE2 4x])
Loaded hashes with cost 1 (HMAC size) varying from 0 to 379572
Will run 2 OpenMP threads
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
Almost done: Processing the remaining buffered candidate passwords, if any.
Proceeding with wordlist:/usr/share/john/password.lst
Proceeding with incremental:ASCII
123465           (test.zip/Screenshot_2024-04-15_13-11-04.png)
123465           (test.zip/test.txt)
123465           (test.zip/Screenshot_2024-04-15_13-09-09.png)
3g 0:00:00:19 DONE 3/3 (2024-04-22 04:09) 0.1536g/s 14358p/s 30638c/s 30638C/s 123456..10405
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

9.3 Comparing Hashcat vs John the Ripper

- Testing the speed of Hashcat and John the Ripper with different hash types and different strengths for passwords:



Password	Hash Algorithm	Hashed Password	HASHCAT		JOHN THE RIPPER	
password123	MD5	482c811da5d5b4bc6d497ffa98491e38	419.21 p/s	0.07 ms	100.24 p/s	0.24 ms
securepass!	SHA-256	90c94e52a4b659f7c4c9b7dbd76c7a45d92a3bb4f5a31e1db0872aeac026047b	434.66 p/s	0.89 ms	242.98 p/s	0.86 ms
mySecret123	SHA-1	4e9658d2dfef4d2d164210dd381eea3d7b5056f5	471.85 p/s	0.28 ms	283.33 p/s	0.59 ms
p@ssw0rd!	MD5	f8a0273e2c0f920982c4c655aaa82ffa	450.27 p/s	0.99 ms	345.08 p/s	0.21 ms
r0cketMan#	SHA-256	7268f85080189e42c0874d0ec5ab22af3e5a02c847769bfb5fc32f8a26926d6d	434.79 p/s	0.56 ms	121.55 p/s	0.62 ms
myP@ssw0rd	MD5	e7cf4bc97703fd6cbebfb9aa41338eb3	325.24 p/s	0.31 ms	307.92 p/s	0.60 ms
letMeIn123	SHA-1	e48cb080b011f1f08aa5b0d0f3c72768602d4d87	344.10 p/s	0.71 ms	155.73 p/s	0.73 ms
s3cur3P@ss	SHA-256	4fc9bb8a409f16bc788c7e52bbf6e0fd9c0d76b1dfbf9dcf715a435c804d76c9	487.47 p/s	0.82 ms	125.92 p/s	0.31 ms
ninjaP@ss123	MD5	864c26d1c07a769f8403e1f4a5c8e02e	439.39 p/s	0.97 ms	319.33 p/s	0.74 ms
pa$$w0rd123	SHA-256	14f47142e4ac559637a18e61e7582a1be97e9b1b8a9d46b6e2f61e63f661ea8c	329.38 p/s	0.70 ms	297.58 p/s	0.64 ms

- Speed
  - Hashcat is generally faster than John the Ripper because it is optimized for working with a wide range of hardware accelerators like GPUs (graphics processing unit) and FPGAs (Field programmable gate array). This allows Hashcat to perform high-speed brute-force attacks, combinatorial attacks, and more, using the massive parallel processing power of modern GPUs.
  - John the Ripper has a mode called Jumbo that supports GPU acceleration, but its primary strength lies in CPU-based attacks. John's performance can be very good on CPUs, but typically, Hashcat outperforms John when it comes to GPU acceleration.
- Ease of Use
  - Hashcat has a steep learning curve due to its extensive range of options and configurations, which can be overwhelming for beginners. However, it has a consistent syntax and well-documented options, which, once mastered, can be highly effective and flexible.
  - John the Ripper is often considered easier for beginners, especially in its default mode. It automatically selects the best strategies and algorithms based on the hashes it detects, which simplifies the process for new users. However, making full use of its capabilities (especially in Jumbo mode) still requires learning a variety of options and configurations.
- When to Use Which
  - Use Hashcat when:
    You have access to powerful GPUs and need to perform high-speed, high-volume cracking.
    You are dealing with a very large set of hashes or need to use advanced attack modes that leverage GPU power.
    You need to crack hashes that are better supported by Hashcat's more modern algorithms and optimizations.

- Use John the Ripper when:
  You are working on a system with limited or no GPU resources.
  You need a tool that's good at automatically handling different hash types with minimal configuration.
  You are engaging in pentesting scenarios where you might benefit from the flexibility and scriptability of John the Ripper, especially with custom rules and modes designed to crack complex password patterns.

10 Passwords Strength

When passwords are compromised, it can lead to serious consequences such as personal information exposure, financial losses, and theft of sensitive data. As Internet technology advances, users are required to manage an increasing number of passwords. However, creating strong passwords that are also easy to remember is challenging. Consequently, users may resort to using personal information like names, birthdates, or phone numbers, making them vulnerable to targeted attacks and password reuse. Balancing security and usability has become a challenge for users.
Strength of a password is determined by a number of variables, such as length, complexity, randomness, and character set. The ways in which these elements affect password strength are broken down here:

- Length: Because they give hackers more room to search, longer passwords are typically more secure than shorter ones. The number of combinations that are feasible increases exponentially with each extra character.
  - With simply lowercase letters, there are 268268 potential permutations for an 8-character password.
  - There are 26122612 potential permutations for a 12-character password using the same character set, which is a far bigger number.
- Character Set: Adding a range of characters to your password, such as digits, special characters, lowercase and uppercase letters, and numbers, makes it more difficult to crack and less vulnerable to brute-force attacks.
  - Since "password" is a single lowercase letter, it is a weak password.
  - Because it combines capital and lowercase letters, numerals, and special characters, a password like "P@ssw0rd!" is more robust than others.
- Randomness: Passwords that are generated at random tend to be more secure than those that are based on popular terms or patterns. Dictionary or pattern-based attacks are less likely to guess or break random passwords.
- Predictability Avoiding: Keeping your password strong means staying away from patterns or sequences that are simple to figure out (like "123456" or "qwerty"). To further improve security, keep personal information like names, birthdays, and frequent words to a minimum.

Here are some examples demonstrating the impact of length and complexity on password strength:

- Weak Password: "password"
  - Length: 8 characters
  - Character Set: Lowercase letters only
  - Strength: Very weak, easily guessable or crackable
- Moderate Password: "P@ssw0rd"
  - Length: 8 characters
  - Character Set: Uppercase letters, lowercase letters, digits, special characters
  - Strength: Moderate, resistant to basic attacks but still vulnerable to advanced techniques
- Strong Password: "Tr0ub4dor&3"
  - Length: 12 characters
  - Character Set: Uppercase letters, lowercase letters, digits, special characters
  - Strength: Strong, significantly increases the search space for attackers and provides good resistance to various attack methods
- Very Strong Password: "x#uT7fr̂B2@j$LmP"
  - Length: 16 characters
  - Character Set: Uppercase letters, lowercase letters, digits, special characters
  - Strength: Very strong, extremely difficult to crack due to length and complexity

11 Password Managers

Password managers are software tools designed to securely store and manage passwords for various online accounts and services.
They work by encrypting and storing your passwords in a centralized vault, which is typically protected by a master password or passphrase.
When you need to log in to a website or app, the password manager can automatically fill in the credentials for you, saving you the trouble of remembering or typing them manually.

- Password Storage: The password manager safely saves your login information in an encrypted database or vault whenever you establish an account or change a password.
- Password Master: You just need to remember one master password or passphrase in order to access the password manager and recover your stored passwords. The password vault must be unlocked using this master password.
- Encryption: To protect your passwords and other private data, password managers employ robust encryption methods. This guarantees that the stored data cannot be accessed without the master password, even in the event of unwanted access.
- Auto-fill and Auto-generate: When you visit a website or open an app, password managers can automatically fill in login fields for you. This is a function that many of them come with. To provide increased protection, they can also create powerful, randomized passwords for brand-new accounts.
- Cross-Platform Syncing: You can access your passwords from your computer, smartphone, or tablet with the help of many password managers that support synchronization across various devices.
- Browser Extensions and Apps: To facilitate cross-platform and cross-device use, password managers usually provide browser extensions or standalone apps that interface with your operating system or web browser.

References

[1] John the ripper password cracker. Available online, 2019. URL: `https://www.openwall.com/john/doc/`.

[2] How to crack hashes with hashcat — a practical pentesting guide. Available online, 2022. URL: `https://www.freecodecamp.org/news/hacking-with-hashcat-a-practical-guide/`.

[3] How to crack passwords using john the ripper. Available online, 2022. URL: `https://www.freecodecamp.org/news/crack-passwords-using-john-the-ripper-pentesting-tutorial/`.

[4] Hashcat website. Available online, Accessed 2024. URL: `https://hashcat.net/hashcat/`.

[5] George Hatzivasilis. Password-hashing status. *Cryptography*, 1(2), 2017. `doi:10.3390/cryptography1020010`.

[6] Enze Liu, Amanda Nakanishi, Maximilian Golla, David Cash, and Blase Ur. Reasoning analytically about password-cracking software. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 380–397, 2019. `doi:10.1109/SP.2019.00070`.

[7] Jianhua Song, Degang Wang, Zhongyue Yun, and Xiao Han. Alphapwd: A password generation strategy based on mnemonic shape. *IEEE Access*, 7:119052–119059, 2019. `doi:10.1109/ACCESS.2019.2937030`.

[8] Hamza Touil, Nabil El Akkad, Khalid Satori, Naglaa F. Soliman, and Walid El-Shafai. Efficient braille transformation for secure password hashing. *IEEE Access*, 12:5212–5221, 2024. `doi:10.1109/ACCESS.2024.3349487`.