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.

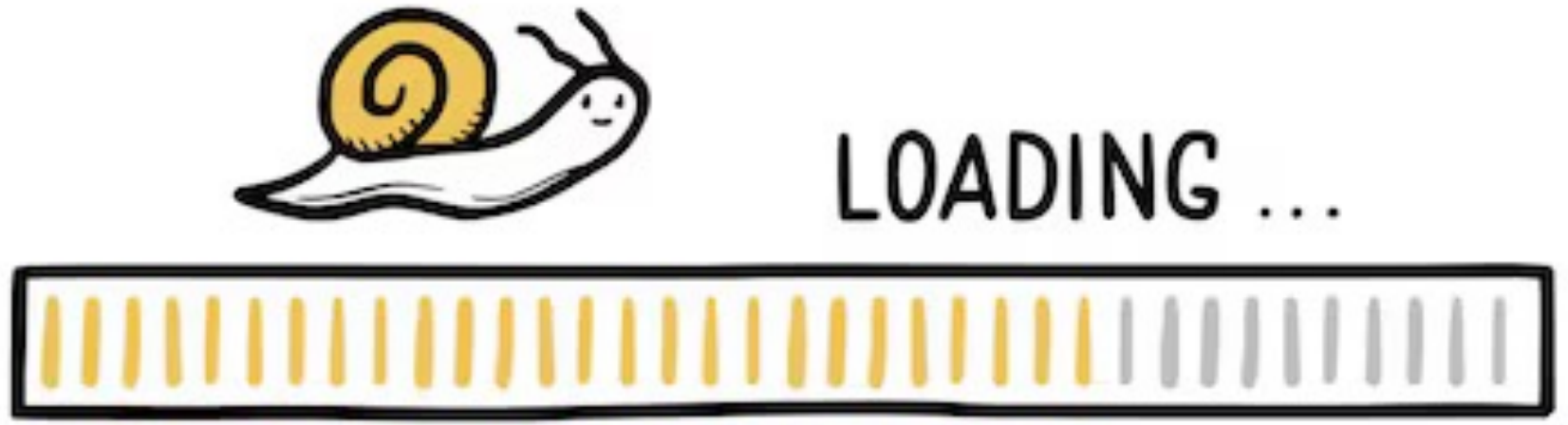
Authorized Use Only: Students must only use the methods, techniques, and tools taught in this course on systems and networks for which they have explicit authorization to test and analyze.

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 plan

- **First lesson: Cryptographic Failures**
  - We know everything about it by now!
- **This & next lessons: Broken Authentication**
  - Offline (hashed password cracking)
  - Online (login forms cracking)
- **Next & last lessons: Malware Analysis**



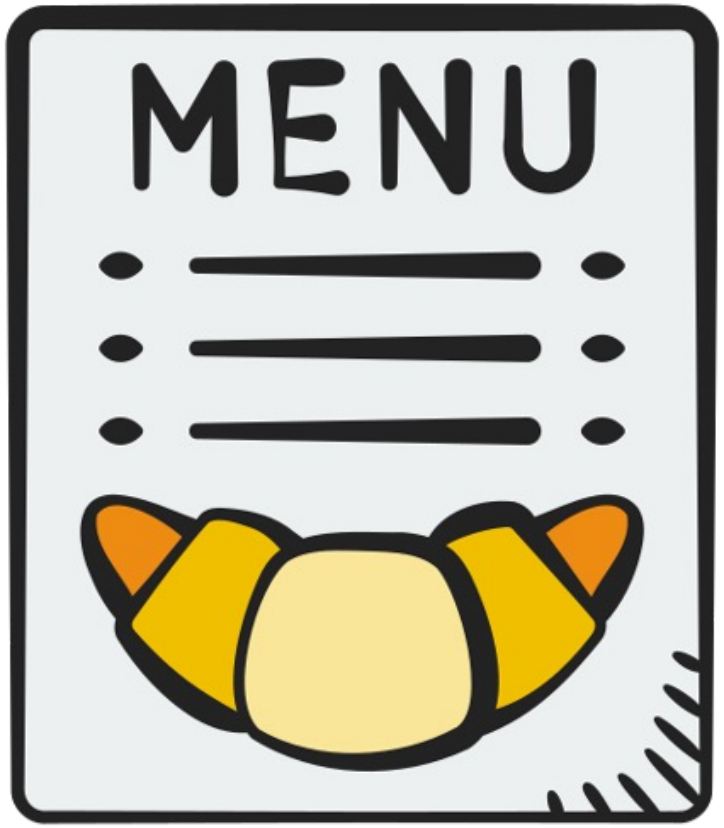
ETHICAL HACKER

* * * * * * *

YOUR PASSWORD IS TOO SHORT SO I CHANGED IT

Summary

- **Password training**
  - Are you (re)using terrible passwords?
- **Cracking with automated tools (John, Hashcat)**
  - Hashed passwords
  - Unix and Windows passwords
  - Password-protected Excel and ZIP files
- **Guessing passwords with automated tools (Hydra)**
  - Services (ssh, ftp, etc.)
  - Login pages
- **Bypassing login using SQL injection (SQLi)**



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**
  - Reused passwords from breaches
  - No rate-limiting or account lockout
- **Session hijacking**
  - Tokens over HTTP or XSS stealing session cookies
- **Brute force vulnerabilities**
  - No CAPTCHA, lockout, or throttling
  - Unlimited login attempts
- **Token leakage**
  - JWT/API tokens in localStorage or URLs
  - 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
- **Improper session management**
  - Long-lived sessions or logout doesn't invalidate session
- **Auth logic bypass**
  - Poor backend checks (e.g., role from client-side)
- **Client-side Auth enforcement**
  - Hidden admin UI elements without backend restrictions
- **Hardcoded credentials**
  - Secrets exposed in code or app binaries
- **Insecure "Remember Me"**
  - Plaintext passwords in cookies or no expiration

Password attacks

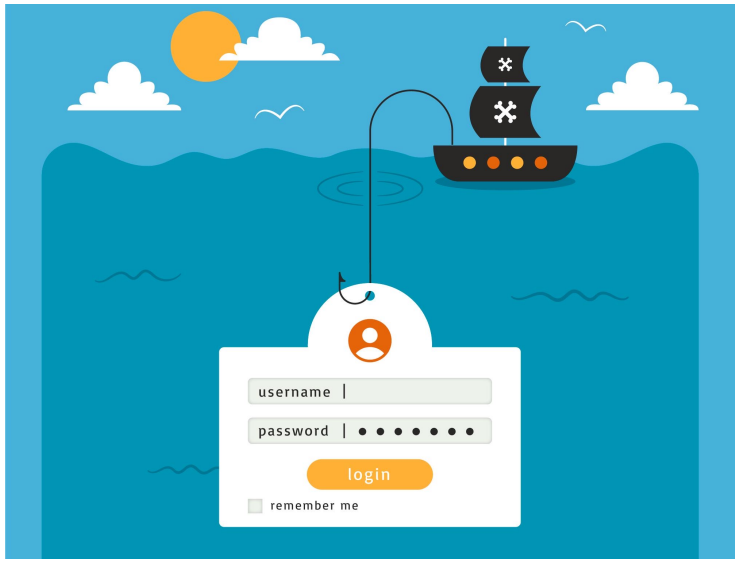
The tough life of a password

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 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

John has no GUI: a lot of command line text is coming your way!

Bring your best reading glasses.

John the Ripper: help and supported formats

john –h

```
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 John directory of 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`

Using Python to crack hashes with dictionary (from previous lesson)

(re)Check exercise in CryptographicFailures/CryptFail_dictionary_md5.py

- We have seen this in the previous lesson: **simply read the wordlist into an array and then loop all the words by computing their hash and comparing it with the to-be-cracked hash. End when the hashes are the same**
  - This approach clearly misses all the optimizations of tools like John and it is extremely slower

```
def crack_hash(hash_to_crack, hash_function, hash_name):

    password_list = [line.rstrip() for line in open('Resources/xato-net-10-million-passwords-10000.txt')]

    start = time()
    print(f"Cracking {hash_name} hash: {hash_to_crack}")
    for password in password_list:
        # Generate the hash of the password using the specified hash function
        hashed_password = hash_function(password)
        print(f"Trying password: {password} -> {hashed_password}")
        if hashed_password == hash_to_crack:
            print(f"Success! The password is: {password}. Cracked in {time() - start} seconds")
            return password
    print("Failed to crack the hash.\n")
    return None
```

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

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 and used to build a custom list of potential passwords

- Meet **CUPP - Common User Passwords Profiler (https://github.com/Mebus/cupp)**
  - You can find this Python tool in the BrokenAuthentication\Online\cupp directory of lab materials

```
(.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!
```

peter.txt — Modificato

```
janeMary!!@
janeMary!$
janeMary!$!
janeMary!$$
janeMary!$%
janeMary!$&
janeMary!$*
janeMary!$@
janeMary!%
janeMary!%!
janeMary!%$
janeMary!%%
janeMary!%&
janeMary!%*
janeMary!%@
janeMary!&
janeMary!&!
janeMary!&$
janeMary!&%
janeMary!&&
janeMary!&*
janeMary!&@
janeMary!*
janeMary!*!
janeMary!*$
janeMary!*%
janeMary!*&
janeMary!**
janeMary!*@
janeMary!@
janeMary!@!
janeMary!@$
janeMary!@%
janeMary!@&
janeMary!@*
janeMary!@@
```

peter.txt — Modificato

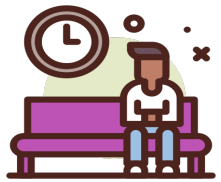
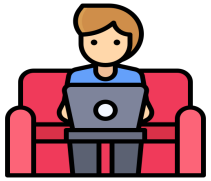
```
Benjamin
Benjamin!
Benjamin!!
Benjamin!!!
Benjamin!!$
Benjamin!!%
Benjamin!!&
Benjamin!!*
Benjamin!!@
Benjamin!$
Benjamin!$!
Benjamin!$$
Benjamin!$%
Benjamin!$&
Benjamin!$*
Benjamin!$@
Benjamin!%
Benjamin!%!
Benjamin!%$
Benjamin!%%
Benjamin!%&
Benjamin!%*
Benjamin!%@
Benjamin!&
Benjamin!&!
Benjamin!&$
Benjamin!&%
Benjamin!&&
Benjamin!&*
Benjamin!&@
Benjamin!*
Benjamin!*!
Benjamin!*$
Benjamin!*%
Benjamin!*&
Benjamin!**
```

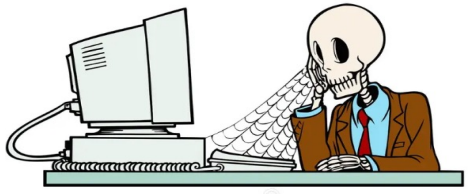
CUPP - Common User Passwords Profiler

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.

Accelerating bruteforce with GPU

- Bruteforce can be accelerated with GPU
  - GPU is ~10× to 500× faster than CPU, depending on the type of hash
- To list available devices use the command:

```
john --list=opencl-devices
```

- The output will look like this if there is a GPU on the machine:

```
Platform #0 name: NVIDIA CUDA, version: OpenCL 3.0 CUDA 11.4.557
    Device #0 (1) name:     NVIDIA RTX A5000
    Device vendor:          NVIDIA Corporation
    Device type:            GPU (LE)
    Device version:         OpenCL 3.0 CUDA
    OpenCL version support: OpenCL C 1.2
    Driver version:         470.256.02 [recommended]
```

- Bruteforcing command will then become:

```
john --incremental --format=raw-md5-opencl --device=1 hashes.txt
```

Accelerating bruteforce with GPU

- Using an Nvidia RTX A5000, assuming an alphanumeric password of known length = 8

```
john --pot=clear --incremental:Alnum --format=raw-md5-opencl --device=1 --min-length=8 --max-length=8 hashes.txt
```

```
Device 1: NVIDIA RTX A5000
Using default input encoding: UTF-8
Loaded 1 password hash (raw-MD5-opencl [MD5 OpenCL])
Note: This format may be a lot faster with --mask acceleration (see doc/MASK).
LWS=256 GWS=4194304

Press 'q' or Ctrl-C to abort, 'h' for help, almost any other key for status
0g 0:00:00:06 0.00% (ETA: 2025-03-10 17:59) 0g/s 46832Kp/s 46832Kc/s 46832KC/s Dev#1:35°C 1712580j..aivvan9c
0g 0:00:00:14 0.00% (ETA: 2025-03-09 23:29) 0g/s 48028Kp/s 48028Kc/s 48028KC/s Dev#1:35°C tr0gusa1..24djdn07
0g 0:00:00:16 0.00% (ETA: 2025-03-09 16:41) 0g/s 48436Kp/s 48436Kc/s 48436KC/s Dev#1:36°C atears74..ljanud17
0g 0:00:00:18 0.00% (ETA: 2025-03-09 11:27) 0g/s 48646Kp/s 48646Kc/s 48646KC/s Dev#1:36°C rodr114m..ramorto3
0g 0:00:00:20 0.00% (ETA: 2025-03-09 01:59) 0g/s 48902Kp/s 48902Kc/s 48902KC/s Dev#1:36°C lj100amp..jcimigml
0g 0:00:00:22 0.00% (ETA: 2025-03-08 23:07) 0g/s 49120Kp/s 49120Kc/s 49120KC/s Dev#1:36°C kublyal7..revstgoy
0g 0:00:01:40 0.00% (ETA: 2025-03-08 15:31) 0g/s 49468Kp/s 49468Kc/s 49468KC/s Dev#1:39°C delup2a1..dryecd93
0g 0:00:44:45 0.06% (ETA: 2025-03-07 19:59) 0g/s 50293Kp/s 50293Kc/s 50293KC/s Dev#1:41°C g0b45ta9..gcauni8K
0g 0:00:44:47 0.06% (ETA: 2025-03-07 20:11) 0g/s 50284Kp/s 50284Kc/s 50284KC/s Dev#1:41°C frcgkq0d..fan4arr8
s3cret7b         (?)
1g 0:06:51:33 DONE (2025-01-16 20:57) 0.000040g/s 49713Kp/s 49713Kc/s 49713KC/s Dev#1:40°C s3cret7b..s3clkm8y

Use the "--show --format=raw-MD5-opencl" options to display all of the cracked passwords reliably
Session completed.
```

Accelerating bruteforce with GPU using Hashcat

- Hashcat is an alternative to John. It does not have as many features but it is better optimized over GPUs and thus faster at bruteforcing
  - Numeric password `08193388301745` (MD5: `5a17d62fff439a83ce3ee1c11538285a`)

```
hashcat -a 3 -m 0 hash.txt ?d?d?d?d?d?d?d?d?d?d?d?d --session=bruteforce --hwmon-disable -O --force
```

```
hashcat (v6.2.5) starting

CUDA API (CUDA 11.4)
====================
* Device #1: NVIDIA RTX A5000, 24048/24256 MB, 64MCU
* Device #2: NVIDIA RTX A5000, 24048/24256 MB, 64MCU
* Device #3: NVIDIA RTX A5000, 24048/24256 MB, 64MCU
* Device #4: NVIDIA RTX A5000, 24048/24256 MB, 64MCU
* Device #5: NVIDIA RTX A5000, 24048/24256 MB, 64MCU
* Device #6: NVIDIA RTX A5000, 24048/24256 MB, 64MCU
* Device #7: NVIDIA RTX A5000, 24048/24256 MB, 64MCU
*  Device #8: NVIDIA RTX A5000, 24048/24256 MB, 64MCU

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

Optimizers applied:
* Optimized-Kernel
* Zero-Byte
* Precompute-Init
* Meet-In-The-Middle
* Early-Skip
* Not-Salted
* Not-Iterated
* Single-Hash
* Single-Salt
* Brute-Force
*  Raw-Hash
```

```
Session..........: bruteforce
Hash.Mode........: 0 (MD5)
Hash.Target......: 5a17d62fff439a83ce3ee1c11538285a
Time.Started.....: Mon Jan 20 09:27:13 2025, (3 secs)
Time.Estimated...: Mon Jan 20 09:27:16 2025, (0 secs)
Kernel.Feature...: Optimized Kernel
Guess.Mask.......: ?d?d?d?d?d?d?d?d?d?d?d?d [13]
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........: 41902.4 MH/s (10.60ms) @ Accel:64 Loops:250 Thr:512 Vec:1
Speed.#2.........: 41564.3 MH/s (10.68ms) @ Accel:64 Loops:250 Thr:512 Vec:1
Speed.#3.........: 41816.2 MH/s (10.66ms) @ Accel:64 Loops:250 Thr:512 Vec:1
Speed.#4.........: 41710.4 MH/s (10.64ms) @ Accel:64 Loops:250 Thr:512 Vec:1
Speed.#5.........: 42281.5 MH/s (10.51ms) @ Accel:64 Loops:250 Thr:512 Vec:1
Speed.#6.........: 42206.8 MH/s (10.51ms) @ Accel:64 Loops:250 Thr:512 Vec:1
Speed.#7.........: 42651.0 MH/s (10.40ms) @ Accel:64 Loops:250 Thr:512 Vec:1
Speed.#8.........: 42017.2 MH/s (10.59ms) @ Accel:64 Loops:250 Thr:512 Vec:1
Speed.#*.........:   336.1 GH/s
Recovered........: 1/1 (100.00%) Digests
Candidate.Engine.: Device Generator
Candidates.#1....: 1384806934845 -> 6881643491634
Candidates.#2....: 1328221773845 -> 6857439555745
Candidates.#3....: 1383893944845 -> 6889414996378
Candidates.#4....: 1245414996378 -> 6296221773845
Candidates.#5....: 1232954808378 -> 6210091635745
Candidates.#6....: 1234539555745 -> 6211386311745
Candidates.#7....: 1382643491634 -> 6880993944845
Candidates.#8....: 1233291635745 -> 6219771626466

Started: Mon Jan 20 09:27:00 2025
Stopped: Mon Jan 20 09:27:18 2025
```

Using Python to bruteforce MD5 hashes (from previous lesson)

- We have seen this in the previous lesson: **given a password length, simply try all possible combinations of an alphabet of symbols: compute the hash of each combination and compare to the to-be-cracked**
  - This approach clearly misses all the optimizations of tools like John and it is extremely slower

```
def bruteforce(target_hash, pwd_length):

    seed = "aeosrnidlctumpbgqvyhfzjxwk"                    # lowercase
    # seed = "aeosrnidlctumpbgqvyhfzjxwk" + "1234567890"      # uppercase + numbers
    seed_bytes = list(map(ord, seed))

    # Possible are: permutations, combinations or product
    attempts = 0
    start = time()
    for word_bytes in itertools.product(seed_bytes, repeat=pwd_length):

        word_string = reduce(lambda x, y: x+y, map(chr, word_bytes))          # word_bytes to string
        hash_ = hashlib.md5(word_string.encode('utf-8')).hexdigest()          # MD5 of word_bytes

        if hash_ == target_hash:
            print("\n==> PASSWORD CRACKED: word = %s | hash = %s" % (word_string, hash_))
            break

        attempts += 1
```

Hashcat has a better implementation over GPU

Number of characters	Numbers Only	Lowercase Only	Upper and Lower Case	Number, Upper, Lower	Number, Upper, Lower, Symbols
8	Instantly	Instantly	2 minutes	5 minutes	3 hours
9	Instantly	9 seconds	2 hours	5 hours	12 days
10	Instantly	4 minutes	2 days	14 days	3 years
11	Instantly	2 hours	132 days	3 years	279 years
12	Instantly	2 days	19 years	159 years	26.5 thousand years
13	Instantly	6 weeks	995 years	10 thousand years	3 million years
14	3 minutes	3 years	51 thousand years	608 thousand years	239 million years
15	26 minutes	82 years	2 million years	37 million years	22.7 billion years
16	5 hours	2136 years	140 million years	3 billion years	3 trillion years
17	43 hours	56 thousand years	8 billion years	145 billion years	205 trillion years
18	18 days	2 million years	379 billion years	9 trillion years	20 quadrillion years
19	6 months	38 million years	20 trillion years	557 trillion years	2 quintillion years
20	5 years	977 million years	2 quadrillion years	35 quadrillion years	176 quintillion years
21	49 years	26 billion years	54 quadrillion years	3 quintillion years	17 sextillion years
22	490 years	660 trillion years	3 quintillion years	133 quintillion years	2 septillion years

https://specopssoft.com/blog/best-password-practices-to-defend-against-modern-cracking-attacks/

- Time to crack via brute-forcing given hashes with modern hardware with 4x Nvidia RTX 4090s
- Generally, a stock RTX 4090 will achieve approximately 164 GH/s in Hashcat
  - 164.000.000.000 guesses/second
- With Prof. Barni's main research server powered up by 8x RTX A5000
  - We reached a peak of 330GH/s

Hashcat has a better implementation over GPU

Number of characters	Numbers Only	Lowercase Only	Upper and Lower Case	Number, Upper, Lower	Number, Upper, Lower, Symbols
8	Instantly	Instantly	2 minutes	5 minutes	3 hours
9	Instantly	9 seconds	2 hours	5 hours	12 days
10	Instantly	4 minutes	2 days	14 days	3 years
11	Instantly	2 hours	132 days	3 years	279 years
12	Instantly	2 days	19 years	159 years	26.5 thousand years
13	Instantly	6 weeks	995 years	10 thousand years	3 million years
14	3 minutes	3 years	51 thousand years	608 thousand years	239 million years
15	26 minutes	82 years	2 million years	37 million years	22.7 billion years
16	5 hours	2136 years	140 million years	3 billion years	3 trillion years
17	43 hours	56 thousand years	8 billion years	145 billion years	205 trillion years
18	18 days	2 million years	379 billion years	9 trillion years	20 quadrillion years
19	6 months	38 million years	20 trillion years	557 trillion years	2 quintillion years
20	5 years	977 million years	2 quadrillion years	35 quadrillion years	176 quintillion years
21	49 years	26 billion years	54 quadrillion years	3 quintillion years	17 sextillion years
22	490 years	660 trillion years	3 quintillion years	133 quintillion years	2 septillion years

https://specopssoft.com/blog/best-password-practices-to-defend-against-modern-cracking-attacks/

- Time to crack via brute-forcing given hashes with modern hardware with 4x Nvidia RTX 4090s

ed

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

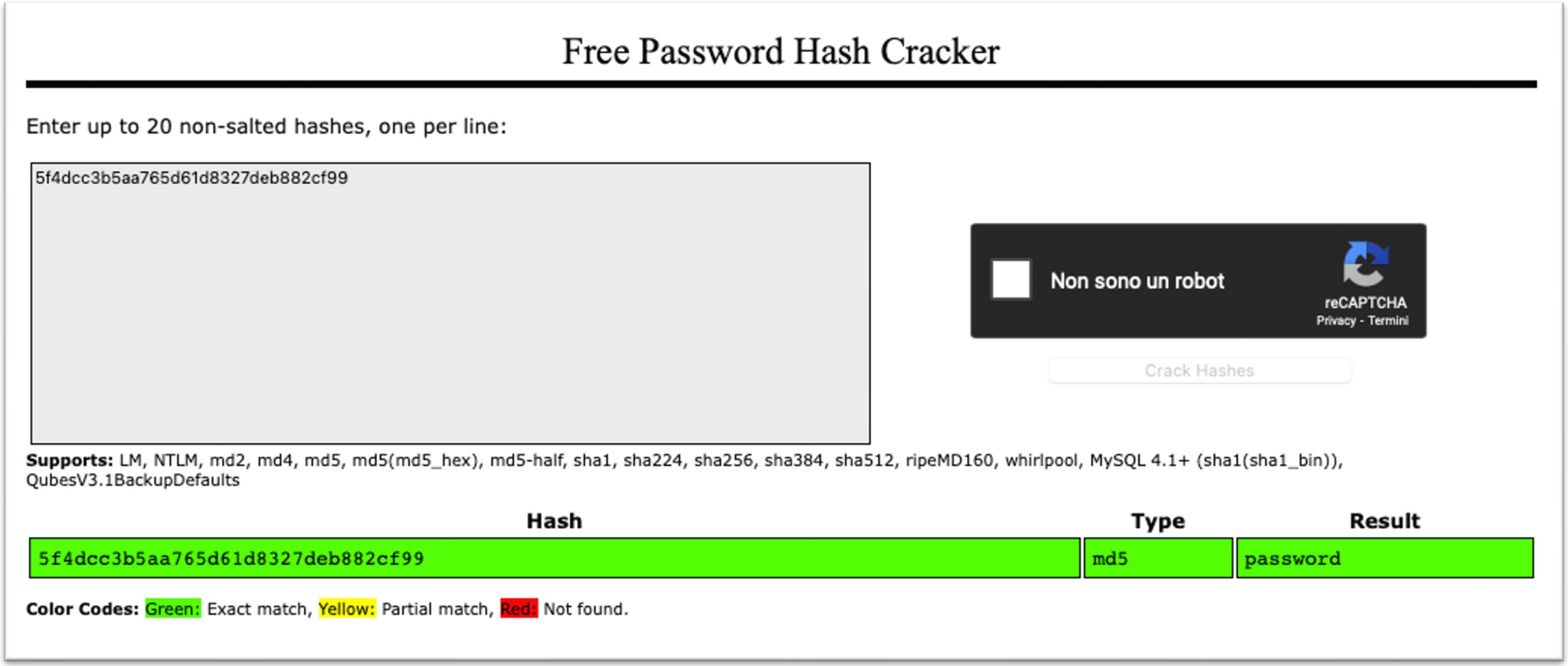
- Why it's useful?
  - You can see how fast your system cracks different hash types. This helps you compare systems (e.g., laptop vs server) or spot misconfigurations (e.g., SIMD not enabled)
  - Hardware verification
  - **It's also a practical demonstration of hashing algorithm complexity and their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

The lazy way. Using hash cracking tools online: https://crackstation.net

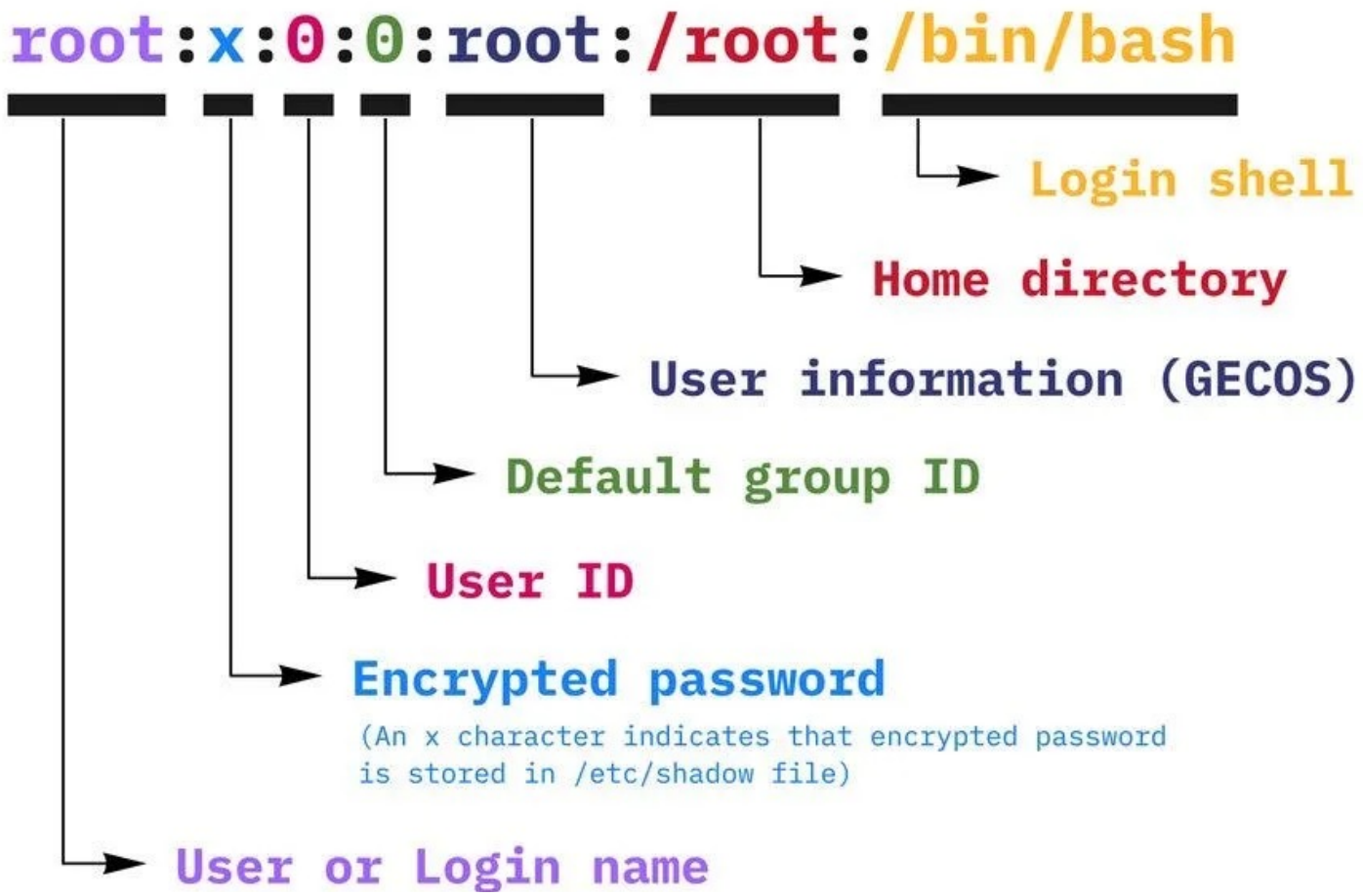
- This website sometimes is useful to crack weak password hashes
  - Try here before running more sophisticated tools
  - It uses a massive wordlist: 1.5 billions hashes, 15GB (link)
  - If the password is not in there, then it will not be able to break the hash



Cracking UNIX password hashes

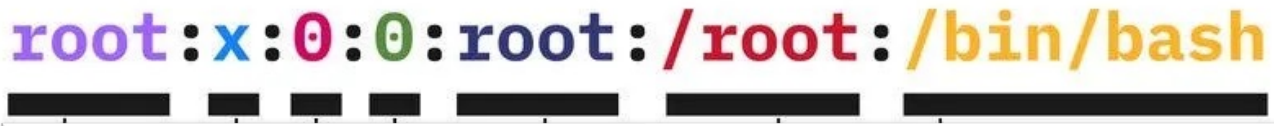
Cracking UNIX passwords: /etc/passwd

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 (:)



Cracking UNIX passwords: /etc/passwd

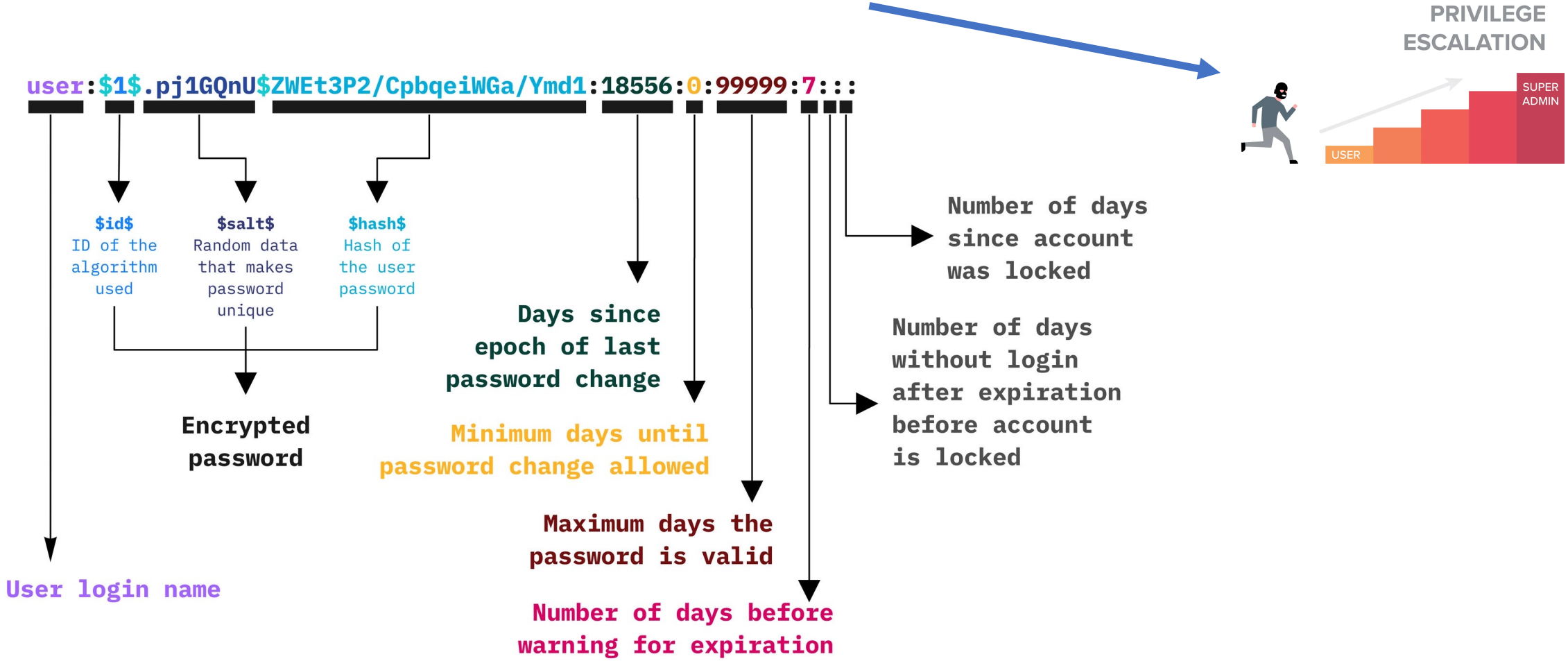
The `/etc/passwd` file is a plain-text database housing fundamental user information. Each line in the file represents a user account and is divided into fields separated by colons (:)




```
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
```

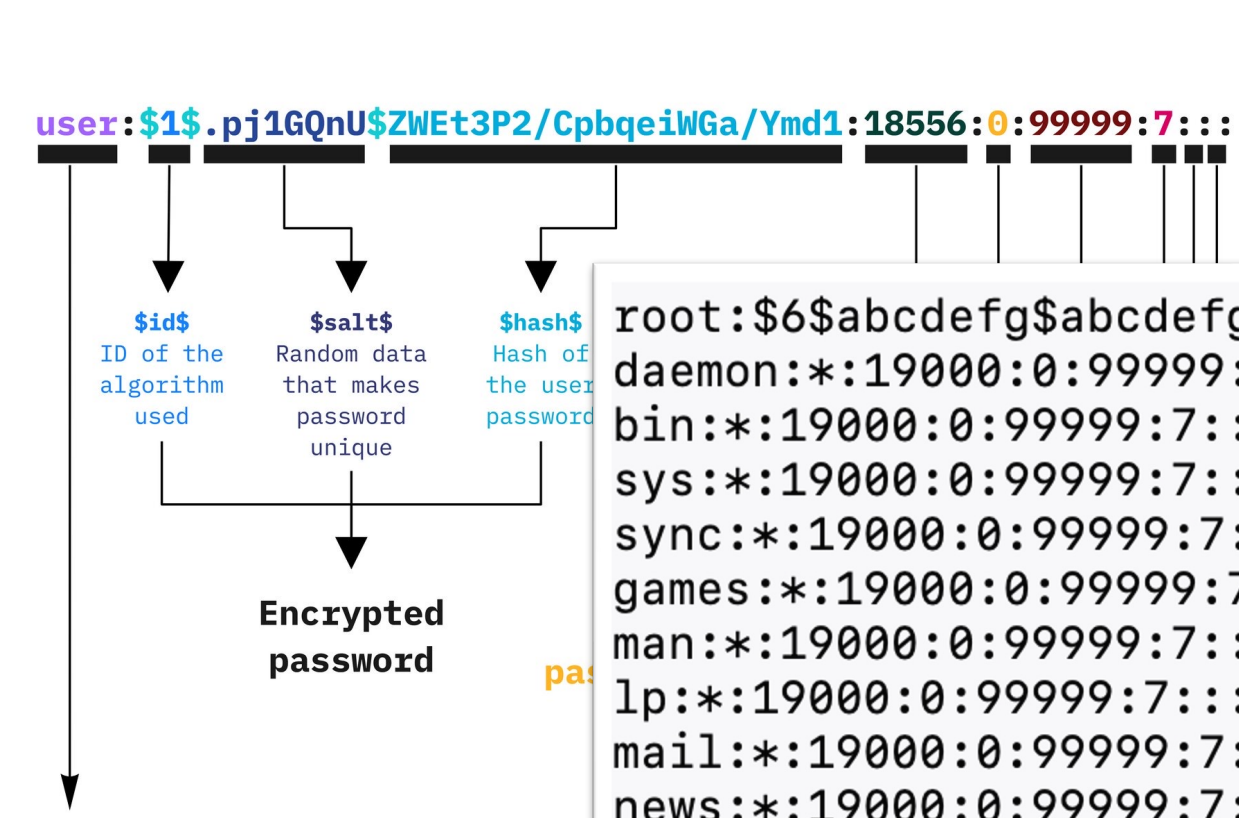
Cracking UNIX passwords: /etc/shadow

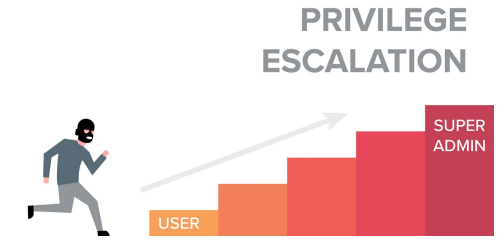
For heightened security, critical user authentication information, particularly hashed passwords, resides in the `/etc/shadow` file, **accessible exclusively to privileged users**.



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

Cracking MS Windows NTLM hashes

Cracking Microsoft Windows passwords

- **Windows uses a secure hashing algorithm to hash passwords**
  - Starting from Windows Vista: NTLM (NT LAN Manager)
- **NTLM hashes are unsalted by default**
  - The same password will always produce the same hash
  - Vulnerable to precomputed attacks (like rainbow tables)
  - Windows mitigates with additional security measures
    - Account lockout policies and password complexity requirements
- **Password hashes for local accounts are stored in the Security Account Manager (SAM) database**
  - The SAM database is in `C:\Windows\System32\config\SAM`
  - The SAM file is protected by the OS
  - It is encrypted to prevent unauthorized access
- **There are several tools to dump the hashed passwords from the SAM**
  - Mimikatz (https://github.com/ParrotSec/mimikatz, https://github.com/skelsec/pypykatz)
  - Pwdump (https://www.openwall.com/passwords/windows-pwdump)
  - Meterpreter (https://www.metasploit.com)

See also: https://www.microsoft.com/en-us/wdsi/threats/threat-search?query=HackTool:Win32/Dump
https://www.microsoft.com/en-us/wdsi/threats/threat-search?query=Hacktool:Win32/Mimikatz

Cracking Microsoft Windows passwords

You need to have elevated privileges to run hash dumping tools, but it's not unusual for a hacker to get lucky with a power user who falls for a well-crafted phish.

```
Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
john.doe:1001:aad3b435b51404eeaad3b435b51404ee:b45cffe084dd3d20d928bee85e7b0f21:::
jane.smith:1002:aad3b435b51404eeaad3b435b51404ee:5f4dcc3b5aa765d61d8327deb882cf99:::
service.account:1003:aad3b435b51404eeaad3b435b51404ee:098f6bcd4621d373cade4e832627b4f6:::
```

Fields:

1. Username: the account name (e.g., `Administrator`, `john.doe`).
2. RID: relative Identifier for the account, unique within the system (e.g., `500` for Administrator).
3. LM Hash: the legacy LAN Manager hash of the password. In modern systems, this is often disabled and represented as a placeholder (`aad3b435b51404eeaad3b435b51404ee`).
4. NTLM Hash: the NT hash of the password, used for authentication (e.g.,`31d6cfe0d16ae931b73c59d7e0c089c0`).
5. Blank Fields (:::): reserved for additional data like domain information or history (empty in the example)


```
john --format=NT ntlm_hashes.txt --wordlist=rockyou.txt
john --show --format=NT ntlm_hashes.txt
```

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

```
# Create a file and zip it
echo "supersecret data" > secret.txt
zip --password <password> file.zip secret.txt

# Try to unzip it, and it will ask for a password
unzip file.zip

# Extract the hash
./zip2john file.zip > zip_hash.txt

# Crack
john --wordlist rockyou.txt --format=pkzip zip_hash.txt
```

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`)

JTR CHEAT SHEET

This cheat sheet presents tips and tricks for using JtR

JtR Community Edition - Linux

Download the JtR Bleeding Jumbo edition with improved capabilities and other goodies.

```
git clone https://github.com/magnumripper/JohnTheRipper -b bleeding-jumbo
```

Compile JtR and enable/disable required features

```
cd JohnTheRipper/
cd src/
./configure
make clean && make -s
```

Enable bash completion. add the following line to your ~/.bashrc

```
. <JtR path>/run/john.bash_completion
```

Cracking Modes

Wordlist Mode (dictionary attack)

```
./john --wordlist=password.lst hashfile
```

Mangling Rules Mode (hybrid)

```
./john --wordlist=password.lst –rules:<rulename> hashfile
```

Incremental mode (Brute Force)

```
./john --incremental hashfile
```

External mode (use a program to generate guesses)

```
./john --external: <rulename> hashfile
```

Loopback mode (use POT as wordlist)

```
./john --loopback hashfile
```

Mask mode (read MASK under /doc)

```
./john --mask=?1?1?1?1?1?1?1 -1=[A-Z] hashfile -min-len=8
```

Hybrid Mask mode

```
./john -w=password.lst –mask='?l?l?w?l?l' hashfile
```

Markov mode (Read MARKOV under /doc). First generate Markov stats:

```
./calc_stat wordlist markovstats
```

Then run:

```
./john -markov:200 -max-len:12 hashfile --mkv-stats=markovstats
```

Prince mode (Read PRINCE under /doc)

```
./john --prince=wordlist hashfile
```

Most modes have Maxlen=13 in John.conf but it can be overwritten with `-max-len=N` up to 24

Multiple CPU or GPU

List OpenCL devices and get the device id

```
./john --list=opencl-devices
```

List formats supported by OpenCL

```
./john --list=formats --format=opencl
```

Multiple GPU's

```
./john hashes --format:<openclformat> --wordlist:<> --rules:<> --dev=0,1 --fork=2
```

Multiple CPU's (e.g., 4 cores)

```
./john hashes --wordlist:<> --rules:<> --dev=2 --fork=4
```

Rules

`--rules:Single`

`--rules:Wordlist`

`--rules:Extra`

`--rules:Jumbo` (all the above)

`--rules:KoreLogic`

`--rules:All` (all the above)

Incremental Modes (Brute Force)

`--incremental:Lower (26 char)`

`--incremental:Alpha (52 char)`

`--incremental:Digits (10 char)`

`--incremental:Alnum (62 char)`

Incremental mode with new charsets

Create a new charset based on john.pot

```
./john --make-charset=charset.chr
```

Create a new entry in John.conf to accommodate the new charset

```
# Incremental modes
[Incremental:charset]
File = $JOHN/charset.chr
MinLen = 0
MaxLen = 31
CharCount = 95
```

Run JtR with the new charset

```
./john --incremental=charset hashfile
```

Wordlists

Sort a wordlist to use with wordlist rule mode

```
$tr A-Z a-z < SOURCE | sort -u > TARGET
```

Use a POT file to generate a new wordlist

```
cut -d: -f2 john.pot | sort -u > pot.dic
```

Generate candidate passwords for slow hashes.

```
./john --wordlist= password.lst --stdout --rules:Jumbo | ./unique -mem=25 wordlist.uniq
```

Use external mode for complex rules

http://www.lanmaster53.com/2011/02/creating-complex-password-lists-with-john-the-ripper/

Generate a wordlist that meets the complexity specified in the complex filter

./john --wordlist=[path to word list] --stdout --external:[filter name] > [path to output list]

Try sequences of adjacent keys on a keyboard as candidate passwords

```
john --external:Keyboard hashfile
```

Configuration Items on John.conf

When using both CPU and GPU set this flag

Idle = N

Hidden Options

```
./john --list=hidden-options
```

Display guesses

```
./john --incremental:Alpha -stdout –session=s1
```

Generate guesses with external program

```
crunch 1 6 abcdefg | ./john hashes –stdin –session=s1
```

Session and Restore

```
./john hashes -session=name
```

./john --restore:name

Show cracked passwords

```
./john hashes --pot=<>  --show
```

Resources

John-Users Mailing List

http://www.openwall.com/lists/john-users/

JtR Community Wiki
http://openwall.info/wiki/john

```
Documentation under doc folder
```

Matt Weir Blog
http://reusablesec.blogspot.ch/

Simple Rule in John.conf

```
[List.Rules:Tryout]
l
u
c
l r
l Az"2015"
d
l A0"2015"
A0"#"Az"#"
```

Details

convert to lowercase
```
l
```

convert to uppercase
```
u
```

#capitalize
```
c
```

#lowercase the word and reverse it (palindrome)
```
l r
```

#lowercase the word and append at end of the word (Az) the number 2015
```
l Az"2015"
```

duplicate
```
d
```

lowercase the word and prepend at beggining of the word (A0) the number 2015
```
l A0"2015"
```

Add # to the beginning and end of the word
```
A0"#"Az"#"
```

Use the Wordlist Rule

Display the password candidates generated with the mangling rule
```
./john --wordlist=password.lst --stdout --rules:Tryout
```

Generate password candidates max length of 8
```
./john --wordlist=password.lst --stdout=8 --rules:Tryout
```

```
./john hashes --wordlist=password.lst --rules:Tryout
```

Simple Wordlist Rules

#lowercase the first character, and uppercase the rest
```
C
```

#toggle case of all characters in the word
```
t
```

#toggle case of the character in position N
```
TN
```

#reverse: "Fred" -> "derF"
```
r
```

#duplicate: "Fred" -> "FredFred"
```
d
```

#reflect: "Fred" -> "FredderF"
```
f
```

#rotate the word left: "jsmith" -> "smithj"
```
{
```

#rotate the word right: "smithj" -> "jsmith"
```
}
```

#append character X to the word
```
$X
```

#prefix the word with character X
```
^X
```

Insert and Delete Wordlist Rules

#Remove the first char from the word
```
[
```

#Remove the last char from the word
```
]
```

#delete the character in position N
```
DN
```

#extract substring from position N for up to M characters
```
xNM
```

#insert character X in position N and shift the rest right
```
iNX
```

#overstrike character in position N with character X
```
oNX
```

Charset and Conversion Wordlist Rules

#shift case: "Crack96" -> "cRACK(^"
```
S
```

#lowercase vowels, uppercase consonants: "Crack96" -> "CRaCK96"
```
V
```

#shift each character right, by keyboard: "Crack96" -> "Vtsvl07"
```
R
```

#shift each character left, by keyboard: "Crack96" -> "Xeaxj85"
```
L
```

Length control

#reject the word unless it is less than N characters long
```
<N
```

#reject the word unless it is greater than N characters long
```
>N
```

#truncate the word at length N
```
'N
```

Dictionaries

Generate wordlists from Wikipedia pages: `wget https://raw.githubusercontent.com/zombiesam/wikigen/master/wwg.py`

```
python wwg.py -u http://pt.wikipedia.org/wiki/Fernando_Pessoa -t 5 -o fernandopessoa -m3
```

Generate wordlists from Aspell Dict's

```
aspell dump dicts
```

```
sudo apt-get install aspell-es
```

```
aspell -d es dump master | aspell -l es expand | awk 1 RS=" |\n" > Spanish.dic
```

Resources

Full Rules Documentation
`http://www.openwall.com/john/doc/RULES.shtml`

Password Analysis and Cracking Kit
`https://thesprawl.org/projects/pack/`

Mangling Rules Generation by Simon Marechal
`http://www.openwall.com/presentations/Passwords12-Mangling-Rules-Generation/`

Thank you!

Next lab: Broken Authentication online (plug your cable)