PASSWORD CRACKING 101

Understanding Hashes, Cracks, and Cyber Defense

PRESENTATION 2025

Presented By
Adrian Cisneros

Date
07/23/2025

Week 2

Password Cracking

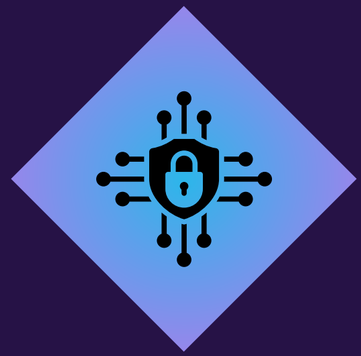
What & Why

💡 What is Password Cracking?

- The process of uncovering passwords by guessing or brute-forcing hashed credentials
- Often used to test password strength or breach systems

🎯 Why Do Attackers Use It?

- Credential Reuse:
  - One leaked password = access to multiple accounts (email, bank, company VPN)
- Privilege Escalation:
  - Start as a low-level user → crack an admin hash → gain full system control
- Persistence & Lateral Movement:
  - Cracked passwords help attackers move between machines undetected



1.1: Types of Threat Actors & Attacks

Brute-force, dictionary, hybrid attacks
Credential stuffing

2.2: Security Assessment Tools

Tools like Hashcat & John
the Ripper = assessment
tools for password policies

3.3: Implement Secure Authentication

Why strong hashing (e.g., bcrypt, PBKDF2)
and complexity rules matter

🔍 Security+ Objective 1.1 – Types of Threat Actors & Attacks

Attack Type	Description	Password Cracking Tie-In
Brute Force	Attempts every possible combination to guess a password	Time-intensive but effective against short/simple passwords
Dictionary Attack	Uses a predefined list of common words/passwords	Faster than brute force; often uses leaks like RockYou.txt
Hybrid Attack	Combines dictionary + brute force (e.g., word + number)	Useful for passwords like password123 or qwerty2024
Credential Stuffing	Reuses known leaked credentials across multiple services	Exploits reused passwords from prior breaches
Password Spraying	Tries common passwords (e.g., Spring2024!) across many users	Slower and stealthier than brute force; avoids account lockouts
Offline Attack	Attacker obtains the hash and cracks it without interacting with the system	Common in breaches — cracking occurs after data exfiltration
Online Attack	Repeated login attempts against a live system	Slower due to system limitations (rate-limiting, lockouts)

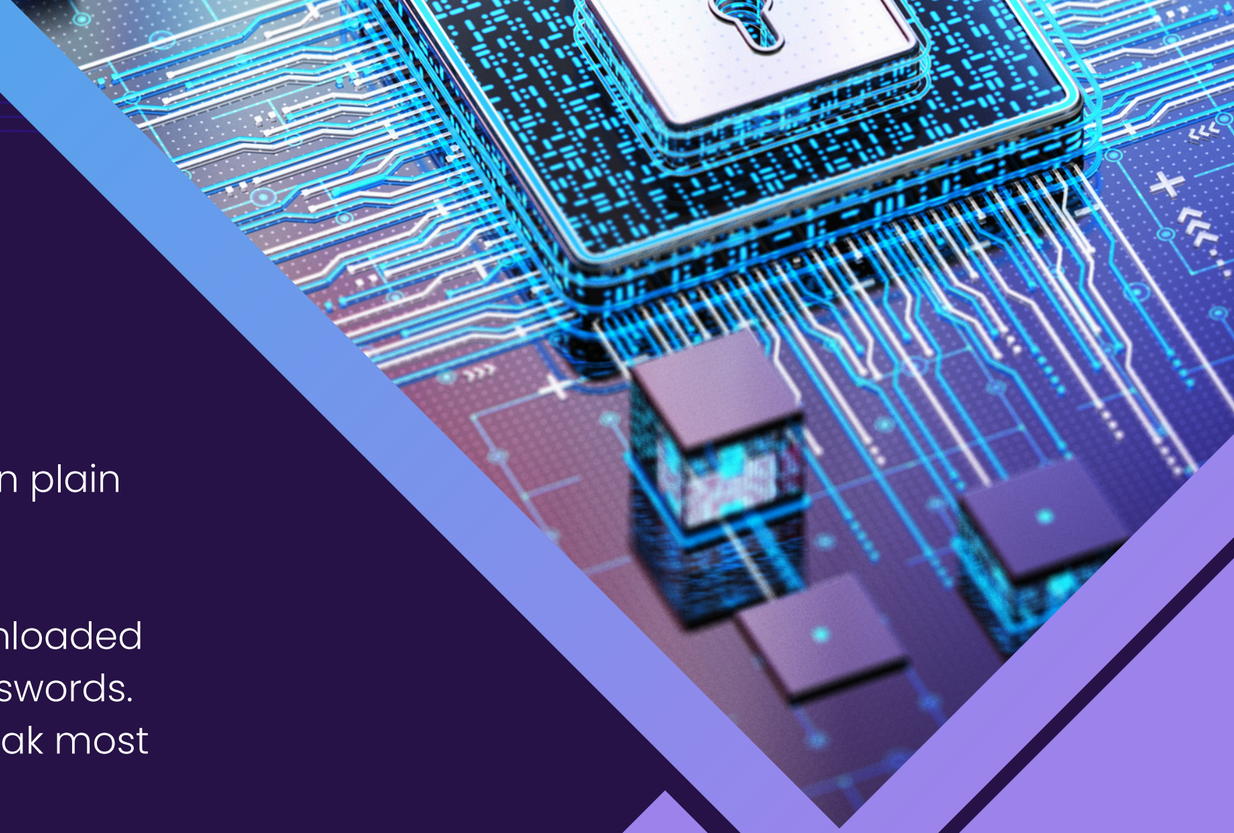
🧰 Security+ Objective 2.2 – Use of Security Tools

Tool	Primary Use	Password Cracking Role
Hashcat	High-performance hash cracking using GPU acceleration	Cracks hashes using various attack modes (dictionary, brute, hybrid)
John the Ripper	Flexible password cracker for Unix/Windows environments	Strong at rule-based cracking and format detection
CrackStation	Online hash lookup using precomputed databases	Quickly identifies weak, known hashes
Hydra / Medusa	Online password brute-force tools	Not used for hash cracking — targets login services like SSH, FTP
RainbowCrack	Uses rainbow tables (precomputed hashes)	Fast against unsalted hashes like old LM/NTLM
Strings / Binwalk	Used in reverse engineering binaries	Can extract hardcoded credentials from compiled software

🔐 Security+ Objective 3.3 – Implement Secure Authentication Controls

Control / Concept	Purpose	Relation to Password Cracking
Password Complexity	Require length, symbols, upper/lower case	Increases resistance to brute-force and dictionary attacks
Password History	Prevents reuse of old passwords	Slows down credential cycling and reuse
Account Lockout	Locks user after X failed attempts	Mitigates online brute force and spraying
Salting	Adds random data to password before hashing	Prevents use of rainbow tables and hash lookup attacks
Key Stretching	Uses algorithms like bcrypt, scrypt, PBKDF2 to slow hashing	Makes cracking computationally expensive
Multi-Factor Auth (MFA)	Adds a second verification method	Even cracked passwords won’t allow access without second factor
Federated Identity	Centralized authentication across services	May reduce the surface area for password-based attacks

RockYou Leak (2009)



📖 Imagine a small app developer storing 32 million passwords, in plain text.

One day in 2009, a hacker exploited a SQL vulnerability and downloaded their entire database. No encryption. No protection. Just raw passwords. These passwords, “123456,” “iloveyou,” “qwerty”, revealed how weak most users’ choices were.

🧠 Why it matters:

- These leaked passwords still live on in popular cracking wordlists (like rockyou.txt).
- They're now used in dictionary attacks against millions of other services.

📌 What is a Hash?

A hash is a fixed-length string of characters generated from input data (like a password) using a mathematical function.

You can think of it as a digital fingerprint:

- 🔁 It uniquely represents the original data
- 🎯 It's always the same for the same input
- 🔒 And it cannot be reversed to reveal the original

✨ Why It Matters

Hashing is a critical part of cybersecurity. It lets systems store and verify sensitive data (like passwords) without ever saving the actual password itself. If a database gets breached, attackers only see the hashes,

not the real passwords.

🧠 Key Properties of a Good Hash Function

Property	Meaning
One-Way	You *can't reverse it* — once it's hashed, the original is gone
Deterministic	Same input → same output every time
Fixed Size	No matter the input size, the output hash is always the same length
Fast to Compute	Ideal for quick comparisons, like checking if a password is correct
Collision-Resistant	Different inputs shouldn't result in the same hash

Password: sunshine123
SHA-256 Hash: ef92b778bafe771e89245b89ecbc08a44a4e166c066599l1881f383d4473e94f

Even the tiniest change (like capitalizing the "s") gives a totally different hash.

🧪 Popular Hashing Algorithms

Algorithm	Description	Vulnerabilities	Status
MD5	128-bit hash, very fast	Easily brute-forced, no built-in salting	❌ Broken
SHA-1	160-bit hash, more secure than MD5	Still vulnerable to collisions	⚠️ Weak
bcrypt	Adaptive, adds **salting**, slower by design	Designed to resist brute force and rainbow tables	✅ Recommended

🔑 Salting = Adding a random value to each password before hashing it.
This ensures that even if two users have the same password, their hashes will be completely different.
Without salting, attackers can precompute common hashes — called **rainbow tables** — and reverse-engineer weak hashes.

Adrian Cisneros

During login, the process is repeated:
The input password + the original salt → hashed again → if the hashes match, you're in.

🔍 Real-World Analogy:

Imagine feeding your password into a blender,
the hash is the smoothie that comes out.

You can always make the same smoothie
from the same ingredients,
but you can’t take the smoothie and get your
original apple, banana, and yogurt back.

🚨 Common Password Attack Methods

Cyber attackers don’t guess passwords like in the movies, they use smart, fast, and scalable techniques to crack hashes and gain access.

🧱 Brute Force

Try every possible combination until the right one is found.

📖 Dictionary Attacks

Uses lists of common or leaked passwords.

🧪 Hybrid Attacks

Combines dictionary words with patterns or rules.

🌈 Rainbow Table Attacks

Uses precomputed hash databases to instantly match hashes.

🔨 Brute Force Attack & 📖 Dictionary Attack

🔍 Aspect	💡 Details
⚙️ **How it works**	Tries **every possible combination** of characters until the correct one is
⏳ **Speed**	Very slow — increases **exponentially** with password length &
🎯 **Target**	Short or simple passwords (e.g., 3-4 characters)
🛡️ **Defense**	Enforce long & complex passwords; implement account lockout after failed
🧠 **Analogy**	Like trying every key on a keyring to unlock a door

🔍 Aspect	💡 Details
⚙️ **How it works**	Tries **words from a list** (e.g., rockyou.txt, common passwords, leaked lists)
⚡ **Speed**	Faster than brute force, especially with optimized wordlists
🎯 **Target**	Users who use real words or common passwords like password, 123456, etc.
🛡️ **Defense**	Encourage passphrases or non-dictionary combos; block common passwords
🧠 **Analogy**	Like guessing someone’s password by going through a list of most-used options

🔧 Hybrid Attack & 🌈 Rainbow Table Attacks

🔍 Aspect	💡 Details
⚙️ **How it works**	Combines **dictionary words + patterns** (e.g., Password2025!, Summer2024)
🧠 **Smartness**	Adds logic to guesses — e.g., capitalizing first letters, appending numbers
🎯 **Target**	Predictable human behavior (e.g., seasonal, name+number, company+year)
🛡️ **Defense**	Promote randomness and passphrase usage; avoid predictable structures
🧠 **Analogy**	Like knowing someone's birthday and trying combinations like

🔍 Aspect	💡 Details
⚙️ **How it works**	Matches hashes using precomputed lookup tables
⚡ **Speed**	Extremely fast — just search the table instead of recalculating each time
📁 **Size**	Requires large storage (often hundreds of GBs or more)
🎯 **Target**	Unsalted hashes (e.g., MD5, SHA-1 from older systems)
🔒 **Defense**	Salting the password before hashing makes these tables ineffective
🧠 **Analogy**	Like a massive cheat sheet for hashes — match instead of guess

Defending Against Cracks

✅ Good Password Hygiene

- Use long, complex, and unique passwords
- Avoid reused or predictable patterns

🧠 Password Policies

- Enforce minimum length (12+ characters)
- Require character variety (upper, lower, number, symbol)
- Regular updates (but avoid forced frequent resets unless needed)

🧂 Salting & Peppering

- Salt = random value added to each password before hashing → prevents rainbow table attacks
- Pepper = secret value added to all passwords (kept separately from database)

🔐 MFA (Multi-Factor Authentication)

- Even if a password is cracked, MFA blocks access
- Something you know + something you have (e.g., app code, device)

🚫 Lockout Policies

- Limit login attempts → slows brute force attacks
- Use progressive delays or CAPTCHA

📚 Security+ Tie-In:

- Authentication methods (passwords, MFA)
- Access control (who can do what)
- Security controls (preventing unauthorized access)

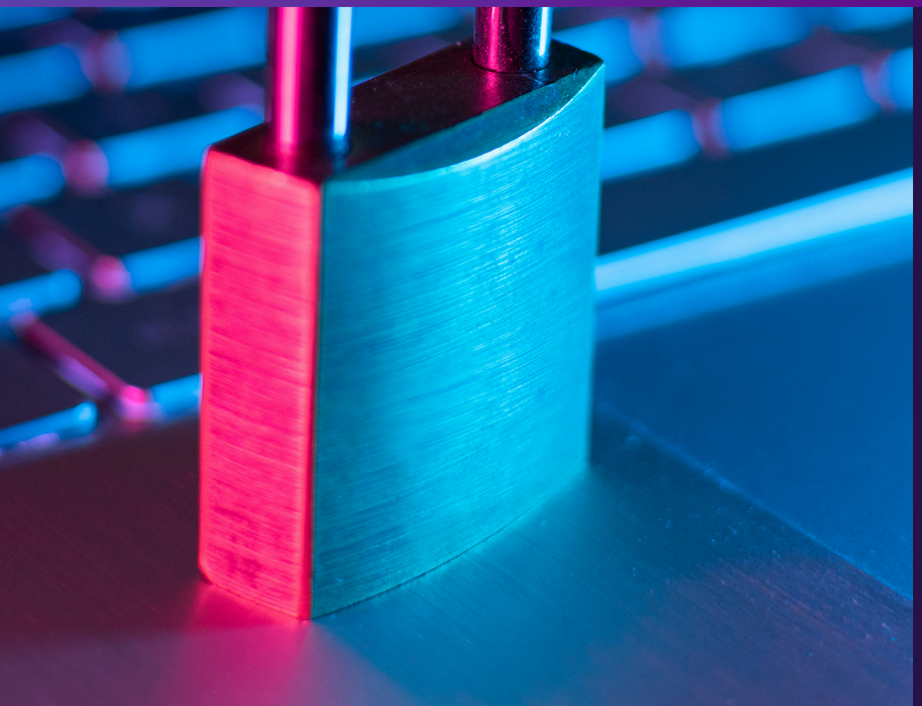
🧂 Salting & 🌶️ Peppering

Feature	Salting	Peppering
Value type	Random per user	Static or fixed (same for all users)
Stored with hash?	Yes, in the database	No, kept secret elsewhere
Purpose	Defeat rainbow/precomputed hashes	Defend against stolen hash + salt
Uniqueness	Unique for each user	Shared across all users

Together, salting and peppering make cracking passwords extremely hard, even if hashes are leaked:

- Salt thwarts precomputed attacks and makes mass cracking impractical.
- Pepper ensures even a leaked salt+hash database is still unusable without access to the app's internal secrets.

Tools of the Trade



💻 Hashcat

A high-speed, GPU-accelerated password recovery tool used by professionals. It supports multiple attack modes and over 300 hashing algorithms.

🧠 CrackStation

A browser-based hash cracker using precomputed rainbow tables. It’s great for quick demonstrations or showing the dangers of weak passwords.

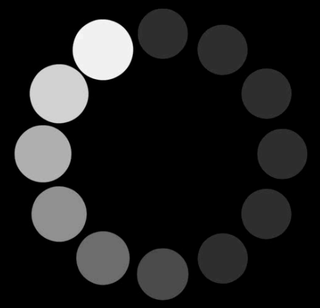
💻 John the Ripper (JtR)

A classic, flexible password cracker known for its speed on CPUs and scriptability. Great for learning and customizing attacks, with tools for Linux systems.

📊 Detailed Tool Comparison Table

Feature / Tool	Hashcat	John the Ripper (JtR)	CrackStation
Type	GPU-accelerated password cracker	CPU-based command-line cracker	Web-based hash cracker
Strength	Extremely fast with powerful GPUs	Very flexible, supports many attack types	Instant results for weak hashes
Attack Types	Brute-force, dictionary, hybrid, mask	Dictionary, hybrid, incremental, rule-based	Precomputed rainbow table lookups
Hash Support	300+ (bcrypt, SHA1/2, NTLM, WPA, etc.)	Many (LM, NTLM, bcrypt, DES, MD5, etc.)	Common ones (MD5, SHA1, NTLM, etc.)
Platform	Windows, Linux, macOS	Windows, Linux, macOS	Any (browser-based)
Use Case	CTFs, red teaming, password audits	Training, Linux systems, scripting	Demos, awareness training
Learning Curve	Medium to high (especially with masks)	Medium, good for scripting and tweaking	Low (plug and play)
Unique Feature	GPU parallelism + mask combinatorials	unshadow, rule engine, hash identification	Huge rainbow tables
Limitations	Needs good GPU, can be complex to set up	Slower than GPU tools on large hash sets	Only unsalted hashes, no custom hashes
Bonus Tip	Combine with rockyou.txt + masks for speed	Great for cracking Linux /etc/shadow files	Show "why salting matters" in seconds

<DEMO TIME>



🧱 BASICS

📌 General Syntax

hashcat [options] -m <hash-type> -a <attack-mode> <hashes> <wordlist/mask>

🔎 COMMONLY USED OPTIONS

Option	Purpose	Example
-m	Hash type	-m 0 (MD5), -m 1000 (NTLM)
-a	Attack mode	-a 0 (dictionary), -a 3 (mask)
--show	Show cracked passwords	--show
--username	Ignore usernames in hash files	--username
--force	Force Hashcat to run (unsafe GPU fix)	--force
-r	Apply rules file	-r rules/best64.rule
-o	Output cracked hashes	-o found.txt
--status	Show live cracking status	--status
--potfile-disable	Disable storing cracked hashes	--potfile-disable
--remove	Remove cracked hashes from file	--remove
--session	Name session for resuming	--session demo01
--restore	Resume a paused/crashed	--restore

	session	

💥 ATTACK MODES

Mode	Name	Description
0	Dictionary	Simple wordlist attack
1	Combinator	Combines words from 2 wordlists
3	Mask	Brute-force pattern attack
6	Hybrid (Dict + Mask)	Append mask to wordlist
7	Hybrid (Mask + Dict)	Prepend mask to wordlist

🔓 COMMON HASH TYPES

Hash Type	Hashcat Mode
MD5	0
SHA1	100
NTLM (Windows)	1000
bcrypt	3200
SHA256	1400
SHA512	1700
WPA/WPA2	22000 or legacy 2500

Use this to identify hash formats:

hashid <hash> or hash-identifier

🧪 EXAMPLES

1. 🗂️ Dictionary Attack (rockyou.txt on MD5)

hashcat -m 0 -a 0 hashes.txt rockyou.txt

2. 🔢 Brute-Force with Mask

hashcat -m 0 -a 3 hashes.txt ?l?l?l?l?d

Tries 5-char lowercase+digit passwords like 'abcd3'

3. 📜 Rule-Based Attack

hashcat -m 0 -a 0 hashes.txt rockyou.txt -r rules/best64.rule

4. 🧩 Hybrid Attack (Dict + 2-digit mask)

hashcat -m 0 -a 6 hashes.txt rockyou.txt ?d?d

5. 🛑 Pause & Resume

Pause

[Ctrl + C]

Resume session

hashcat --restore --session mysession

⚙️ ADVANCED USAGE

🧠 Use Multiple GPUs

hashcat -m 0 -a 0 -D 1,2 hashes.txt rockyou.txt

- **-D 1 = CPU**
- **-D 2 = GPU**

📈 Benchmark Mode

hashcat -b

🔍 Help Menu

hashcat --help

📂 BONUS: Save Cracked Passwords

hashcat -m 0 -a 0 hashes.txt rockyou.txt -o cracked.txt

Let me know what you're targeting (WPA2, shadow files, etc.) and I can build a full command for your use case.

Challenge of the Week (CTF)

🔐 **1. “secret” — Easy Hash Cracking**

https://ctflearn.com/challenge/1514

🗝️ **2. Brute Force is Fun!**

https://ctflearn.com/challenge/365

🔧 **3. “Jumping Chain Hash” — Basic Custom Hash Cracking**

https://ctflearn.com/challenge/436

THANK YOU

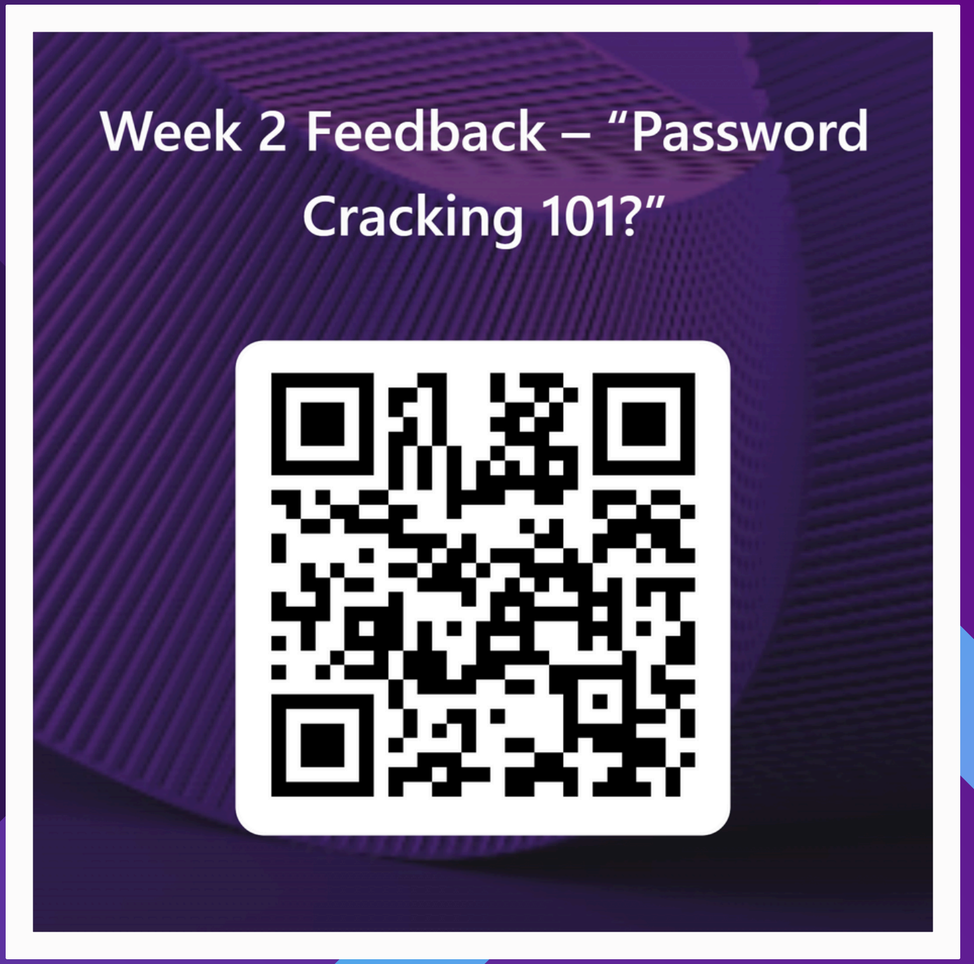
FOR YOUR ATTENTION

HTTPS://FORMS.OFFICE.COM/R/DCW0YLJ8QP

```
C:\VICEROY\Lab> unlock wisdom.exe

[✓] Use strong passwords
[✓] Salt & hash
[✓] Enable MFA

C:\VICEROY\Lab> echo Happy hacking!
Happy hacking!
```



acisn034@jaguar.tamu.edu

VICEROY

TEXAS A&M UNIVERSITY-SAN ANTONIO