

LESSON 07

Wordlist Generation

The quality of a dictionary attack depends entirely on the quality of the wordlist. In this lesson we explore where to find wordlists, how to generate custom ones with Crunch and CeWL, and how to supercharge them with Hashcat rules.

Objectives

- **Identify wordlist sources** — data breach leaks, generators, and common password lists
 - **Generate custom wordlists** using Crunch and CeWL
 - **Apply Hashcat rule files** to multiply the effectiveness of any wordlist
 - **Combine and deduplicate** multiple wordlists into optimized attack dictionaries
-

SOURCES

Where Do Wordlists Come From?

Not all wordlists are created equal. The best ones are derived from **real-world password leaks**, because they reflect how actual humans choose passwords.

Major Sources

- **Data breaches** — RockYou (14M passwords), LinkedIn (117M), Ashley Madison (36M). These are the gold standard for attack wordlists.
- **Common password lists** — SecLists, Probable Wordlists, CrackStation's dictionary. Curated and deduplicated.
- **Generators** — Crunch, Mentalist, CUPP. Create targeted wordlists based on patterns or personal info.
- **Web-scraped wordlists** — CeWL extracts words from websites to build target-specific dictionaries.
- **Multilingual wordlists** — Passwords from non-English speaking regions often follow different patterns and character sets.

Key insight: RockYou is included by default in Kali Linux at `/usr/share/wordlists/rockyou.txt`. It remains the most widely used wordlist in password cracking.

CRUNCH

Crunch — Custom Wordlist Generator

Crunch generates wordlists based on character sets and length ranges. It is exhaustive — it creates every possible combination within the specified parameters.

Basic Syntax

```
# Generate all passwords from 8 to 10 characters (default charset)
crunch 8 10 -o custom_dict.txt

# Specify a custom character set (lowercase + digits)
crunch 8 10 abcdefghijklmnopqrstuvwxyz0123456789 -o custom_dict2.txt

# Use a pattern: @=lowercase, =uppercase, %=digits, ^=symbols
crunch 8 8 -t @@@@%$$$ -o pattern_dict.txt
```

Warning: Crunch generates every combination. A wordlist of all 8-character lowercase+digit passwords is $36^8 = 2.8$ trillion entries. Be mindful of disk space and time.

CEWL

CeWL — Custom Word List Generator from Websites

CeWL spiders a target website and extracts unique words to build a wordlist. This is invaluable when targeting an organization — employees often use company-related terms as passwords.

```
# Basic usage — spider a website and save words
cewl -w web_dict.txt http://example.com

# Set minimum word length and spider depth
cewl -m 6 -d 3 -w web_dict.txt http://example.com

# Include email addresses found on the site
cewl -e -w web_dict.txt http://example.com

# Also extract words from metadata of downloadable files
cewl --meta -w web_dict.txt http://example.com
```

Pro tip: Combine CeWL output with Hashcat rules. A company name like "Acme" becomes "Acme123", "acme2024!", "ACME" and hundreds of other variants through rule-based mutations.

RULES

Hashcat Rules for Wordlist Manipulation

Rules transform each word in a wordlist into multiple candidates at runtime — without increasing the wordlist file size. This is one of Hashcat's most powerful features.

Built-in Rule Files

Rule File	Rules	Description
<code>best64.rule</code>	64	The most effective 64 rules. Great starting point.
<code>d3ad0ne.rule</code>	~34,000	Large, aggressive rule set for thorough attacks.
<code>dive.rule</code>	~99,000	Extremely comprehensive. Use when time is not a constraint.
<code>toggles1-5.rule</code>	varies	Toggle case at positions 1 through 5.

```
# Apply best64 rules to rockyou.txt  
hashcat -m 0 hashes.txt rockyou.txt -r /usr/share/hashcat/rules/best64.rule
```

SYNTAX

Rule Syntax Basics

Each line in a rule file is a sequence of operations applied to every word. Understanding the syntax lets you write your own custom rules.

Common Rule Operations

Rule	Name	Example: "password"
:	Do nothing (passthrough)	password
c	Capitalize first letter	Password
u	Uppercase all	PASSWORD
l	Lowercase all	password
t	Toggle case of all	PASSWORD
\$1	Append "1"	password1
\$!	Append "!"	password!

<code>^1</code>	Prepend "1"	1password
<code>r</code>	Reverse	drowssap
<code>d</code>	Duplicate	passwordpassword
<code>sa@</code>	Replace a with @	p@ssword

```
# Combined rule: capitalize + append "123"  
# Rule line: c $1 $2 $3  
# "password" → "Password123"
```

COMBINING

Combining & Optimizing Wordlists

Real-world attacks often use multiple wordlists. Combining them removes duplicates and produces a single, optimized file.

```
# Merge two wordlists, sort, and remove duplicates
cat dict1.txt dict2.txt | sort -u > combined.txt

# Count lines (unique passwords)
wc -l combined.txt

# Merge multiple files at once
cat rockyou.txt web_dict.txt custom_dict.txt | sort -u > mega_dict.txt

# Sort by frequency (if you have counts)
sort -t: -k2 -rn passwords_with_counts.txt | head -10000 > top10k.txt
```

Personalized wordlists

Use CUPP or Mentalist to generate wordlists from personal info (names, dates, pet names). Highly effective for targeted attacks.

Combined approach

Start with a breach list, add CeWL output for the target, mix in a personalized list, deduplicate, then apply rules during the attack.

NEXT STEPS

Lab & Homework

Lab 3 — Wordlist Generation

- **Step 1:** Install Crunch and CeWL on your system
- **Step 2:** Generate wordlists with both tools
- **Step 3:** Run Hashcat attacks using your custom wordlists with rule files
- **Step 4:** Analyze and compare results across different wordlist strategies

Homework

- Build a comprehensive attack wordlist from multiple sources
 - Write custom Hashcat rules and benchmark them against built-in rules
 - Profile CeWL against a test website and analyze the output
-

