

Lab Guide

Three exercises: identify hash types, crack fast hashes, and attack a slow hash.

Prerequisites: Hashcat installed and working (verify with `hashcat --version`).
A wordlist available — `rockyou.txt` recommended. Install `hashid` with `pip install hashid`.

Exercise 1 — Hash Identification Challenge

EX 1 Identify 10 hashes using manual inspection and tools

Below are 10 hashes. For each one, determine the algorithm, the hashcat mode number, and whether it is salted. Use a combination of manual inspection (length, prefix) and automated tools.

The hashes

```
# Hash 1
5d41402abc4b2a76b9719d911017c592

# Hash 2
aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d

# Hash 3
$2a$10$N9qo8uLOickgx2ZMRZoMyeIjZAgcfl7p92ldGxad68LJZdL17lhWy

# Hash 4
$6$rounds=5000$saltsalt$kpa26CjNg2wMvYSGPHPNeq2VJoxTkN/0fPDJFSyYRwOuAgiNI

# Hash 5
a4f49c406510bdcab6824ee7c30fd852

# Hash 6
2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824
```

```
# Hash 7
$1$salt$qJH7.N4xYta3aEG/dfqo/0

# Hash 8
482c811da5d5b4bc6d497ffa98491e38:salt123

# Hash 9
$apr1$71850310$qh9m4xcAn3MGxoqwX/ztb.
```

Step 1: Manual inspection

For each hash, note the following:

- Does it have a prefix (`$2a$` , `$6$` , `$1$` , `$apr1$` , etc.)?
- How many characters is it? (Count only the hex/base64 portion)
- Is it pure hex, or does it contain other characters?
- Does it include a salt (colon-separated or embedded)?

Step 2: Verify with hashid

```
# Run hashid with -m flag to see hashcat modes
hashid -m '5d41402abc4b2a76b9719d911017c592'
hashid -m '$2a$10$N9qo8uLOickgx2ZMRZoMyeIjZAgcf17p92ldGxad68LJZdL17lhWy'
hashid -m '2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824'
```

Step 3: Confirm with hashcat

```
# Verify the format matches what hashcat expects
hashcat --example-hashes | grep -A4 'MODE: 0'
hashcat --example-hashes | grep -A4 'MODE: 3200'
```

Fill in the answer table

#	Algorithm	Mode (-m)	Salted?	How You Know
1				
2				
3				

#	Algorithm	Mode (-m)	Salted?	How You Know
4				
5				
6				
7				
8				
9				
10				

Expected answers: 1=MD5 (0), 2=SHA-1 (100), 3=bcrypt (3200), 4=SHA-512crypt (1800), 5=MD5 or NTLM (0 or 1000), 6=SHA-256 (1400), 7=MD5crypt (500), 8=md5(\$pass.\$salt) (10), 9=Apache MD5 (1600), 10=SHA-512 (1700).

Exercise 2 — Cracking Fast Hashes

EX 2 Run dictionary attacks on MD5 and SHA-1, compare speeds

Create test hashes and crack them with hashcat. Compare the cracking speed between hash types.

Step 1: Generate test hashes

```
# Generate MD5 hashes of known passwords
echo -n "password" | md5sum | cut -d' ' -f1 > md5_hashes.txt
echo -n "123456" | md5sum | cut -d' ' -f1 >> md5_hashes.txt
echo -n "letmein" | md5sum | cut -d' ' -f1 >> md5_hashes.txt
echo -n "dragon" | md5sum | cut -d' ' -f1 >> md5_hashes.txt
echo -n "monkey" | md5sum | cut -d' ' -f1 >> md5_hashes.txt

# Generate SHA-1 hashes of the same passwords
echo -n "password" | shasum | cut -d' ' -f1 > sha1_hashes.txt
echo -n "123456" | shasum | cut -d' ' -f1 >> sha1_hashes.txt
echo -n "letmein" | shasum | cut -d' ' -f1 >> sha1_hashes.txt
```

```

echo -n "dragon" | shasum | cut -d' ' -f1 >> sha1_hashes.txt
echo -n "monkey" | shasum | cut -d' ' -f1 >> sha1_hashes.txt

# Generate SHA-256 hashes
echo -n "password" | sha256sum | cut -d' ' -f1 > sha256_hashes.txt
echo -n "123456" | sha256sum | cut -d' ' -f1 >> sha256_hashes.txt
echo -n "letmein" | sha256sum | cut -d' ' -f1 >> sha256_hashes.txt
echo -n "dragon" | sha256sum | cut -d' ' -f1 >> sha256_hashes.txt
echo -n "monkey" | sha256sum | cut -d' ' -f1 >> sha256_hashes.txt

```

Step 2: Crack each set with rockyou.txt

```

# Crack MD5
hashcat -m 0 -a 0 md5_hashes.txt /path/to/rockyou.txt --force
# Note the speed (Speed.#1) and time to complete

# Crack SHA-1
hashcat -m 100 -a 0 sha1_hashes.txt /path/to/rockyou.txt --force
# Note the speed and time to complete

# Crack SHA-256
hashcat -m 1400 -a 0 sha256_hashes.txt /path/to/rockyou.txt --force
# Note the speed and time to complete

```

Step 3: Record and compare

Hash Type	Mode	Speed (H/s)	Time to Crack All 5	All Found?
MD5	0			
SHA-1	100			
SHA-256	1400			

Self-check: All 5 passwords should be found for each hash type. MD5 should be fastest, SHA-256 slowest. How many times faster is MD5 compared to SHA-256?

Exercise 3 — Cracking a Slow Hash (bcrypt)

EX 3 Attack a bcrypt hash and observe the dramatic speed difference

Generate a bcrypt hash and attempt to crack it. This exercise demonstrates why slow hashes are effective at resisting brute-force attacks.

Step 1: Generate a bcrypt hash

```
# Using Python to generate a bcrypt hash of "sunshine"
python3 -c "
import bcrypt
password = b'sunshine'
salt = bcrypt.gensalt(rounds=10)
hashed = bcrypt.hashpw(password, salt)
print(hashed.decode())
" > bcrypt_hash.txt

cat bcrypt_hash.txt
# Output: $2b$10$... (60-character bcrypt hash)
```

Step 2: Create a small targeted wordlist

```
# Create a small wordlist (bcrypt is too slow for rockyou.txt)
cat > small_wordlist.txt <<'EOF'
password
123456
sunshine
letmein
football
monkey
shadow
master
dragon
qwerty
EOF
```

Step 3: Crack the bcrypt hash

```
# Crack bcrypt with the small wordlist
hashcat -m 3200 -a 0 bcrypt_hash.txt small_wordlist.txt --force

# Compare: how long would rockyou.txt take?
# Check the estimated time if you started with the full list:
hashcat -m 3200 -a 0 bcrypt_hash.txt /path/to/rockyou.txt --force
```

Step 4: Benchmark comparison

```
# Run benchmarks for direct comparison
hashcat -b -m 0      # MD5 benchmark
hashcat -b -m 3200   # bcrypt benchmark

# Calculate the speed ratio
# MD5 speed / bcrypt speed = how many times faster MD5 is
```

Expected result: MD5 will be roughly 1,000,000x faster than bcrypt. The bcrypt hash should crack in seconds with the small wordlist but would take hours or days with rockyou.txt. This is exactly why bcrypt exists — it makes brute-force impractical.

Self-check: Did hashcat find "sunshine"? What was the bcrypt speed in H/s compared to the MD5 speed? Calculate the ratio and think about what this means for password security.