CSCI 331: Introduction to Computer Security

Lecture 6: Password Cracking, part 1

Instructor: Dan Barowy

Williams

Topics

Lab 1—what did we learn?

Resubmissions

Crypto refresher

Lab 2

Password database attacks

Your to-dos

1. Reading response (Oechslin) **due tonight**.
2. Project part 1 **due Sunday 9/29**.
3. Lab 2 **due Sunday 10/13**.

Lab 1—what did we learn?

Office hours:
Mondays 3-5pm (**TCL 307**)
Thursday 3-5pm (**TCL 312 UNIX lab**)
Fridays 4-6pm (**TCL 312 UNIX lab**)

Resubmissions

Cryptography refresher

Encryption is the **process of encoding a message** so that it can be read only by the sender and the **intended recipient**.

- A **plaintext** p is the original, unobfuscated data. This is information you want to protect.
- A **ciphertext** c is encoded, or encrypted, data.
- A **cipher** f is an algorithm that converts **plaintext** to **cipertext**. We sometimes call this function an **encryption function**.
  - ✳ More formally, a cipher is a function from plaintext to ciphertext, $f(p)=c$. The properties of this function determine what kind of encryption scheme is being used.
- A **sender** is the person (or entity) who enciphers or encrypts a message, i.e., the party that converts the plaintext into cipertext. $f(p)=c$
- A **receiver** is the person (or entity) who deciphers or decrypts a message, i.e., the party that converts the ciphertext back into plaintext. $f^{-1}(c)=p$

See the reading Why Stolen Password Databases are a Problem for a little more nuance.

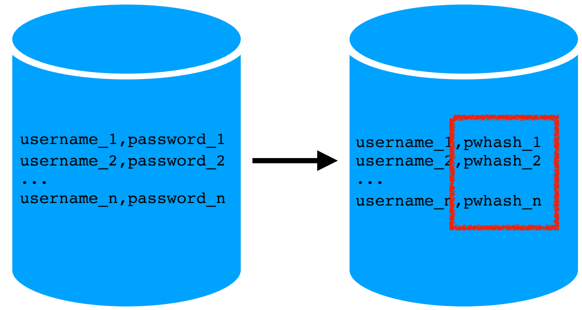
Lab 2

Password database attacks

- Random guessing attack
- Enumeration attack
- Dictionary attack
- Precomputed hash chain attack
- Rainbow table attack

Scenario

Entire password database **leaked** (bug; misconfiguration; theft by authorized personnel).



We keep passwords in **hashed** form.

Hashes are **not invertible**.

Random guessing


```
for each entry in database:
  not_found = true
  // try until found
  while not_found:
    // random plaintext
    p = randPassword()
    // create ciphertext
    c = hash(p)
    // compare
    if c = entry.pwhash:
      print entry.pwhash, p
      not_found = false
```

Complexity?

Random guessing: complexity (one pw)

m = # of possible passwords

p = probability that random guess is correct

= $1/m$

X = # guesses until success

$E[X] = 1/p$ (binomial experiment)

$= m$

$O(m)$ average **per pw** $O(mn)$ average for **all pw**

Enumeration: slightly better


```
for each entry in database:
  i = 0
  not_found = true
  // try until found or out of pt
  while not_found && i < NUM_PT:
    // gen ith possible plaintext
    p = genPassword(i)
    // create ciphertext
    c = hash(p)
    // compare
    if c = entry.pwhash:
      print entry.pwhash, p
      not_found = false
    i++
```

Complexity?

Enumeration: complexity

m = # of possible passwords

Average time to find **one pw**:

O(**m**/2)

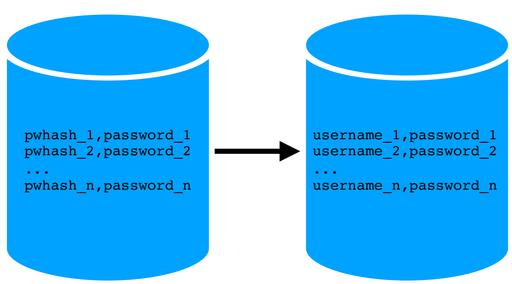
Average time to find **all pw**:

O(**n** x **m**/2)

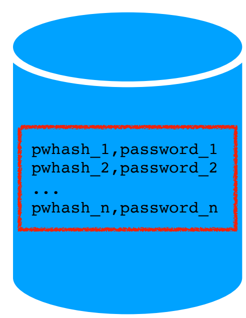
Dictionary attack

A **dictionary attack** is a form of **brute force attack** technique for **recovering passphrases** by systematically **trying all likely possibilities**, such as words in a dictionary.

Critically, a dictionary attack only tries each possibility once. It **trades space for time**.



Dictionary: much better



Ahead of time:

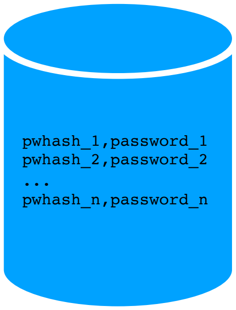
```
while i < NUM_PT:
    // gen ith possible plaintext
    p = genPassword(i)
    // create ciphertext
    c = hash(p)
    // save
    cracked_db[c] = p
```

Later:

```
for each entry in database:
  print cracked_db[entry.pwhash]
```

Complexity?

Dictionary attack: complexity



m = # of possible passwords

Time to compute dictionary:

O(**m**)

Time to lookup **one pw**:

O(**log m**)

Time to lookup **all pws**:

O(**n log m**)

Space needed:

O(**m**)

Activity: How much space?

It depends on the **number of possible passwords**.

Password scheme:

- Uppercase letters and numbers, except `O` and `I`.
- Up to 8 digits

How many passwords are there?

Activity: How much **space**?

m = # of passwords

$$= \sum_{k=1}^{8} 34^k = 1\,839\,908\,871\,710$$

≈ 1.8 trillion passwords

Suppose per-pw storage is **always 16 bytes**.
(8 bytes for cipertext, 8 bytes for plaintext)

16 x (1.8 x 10^{12}) bytes

≈ 26 terabytes

Is this a **feasible attack**?

https://www.amazon.com/Seagate-256MB-3-5-Inch-Enterprise-ST14000NM0018/dp/B07RQZJ347

Is this a feasible attack?

space: ≈ 26 terabytes

Time?

Suppose I can generate 1 million pw/sec

(1.8 x 10^{12}) / 10^6 ≈ (1.8 x 10^6) seconds

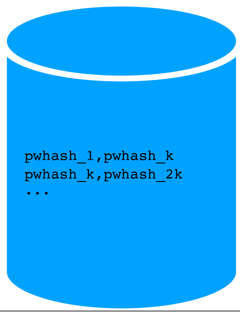
≈ 21 days with one computer.

This is **definitely feasible**!

Precomputed hash chains

A **PCHC attack** is a form of **brute force attack** technique for **recovering passphrases** by systematically **trying all likely possibilities**, such as words in a dictionary.

Critically, a PCHC attack only tries each possibility once. It **trades space for time, but it compresses the database.**



Thought experiment

Suppose we have:

$f(p)=c$, a **cipher** that maps **plaintexts** to **ciphertexts**; in this case, a **hash function**.

Because f is a hash function, there is **no inverse function** such that $f^{-1}(f(p))=p$.

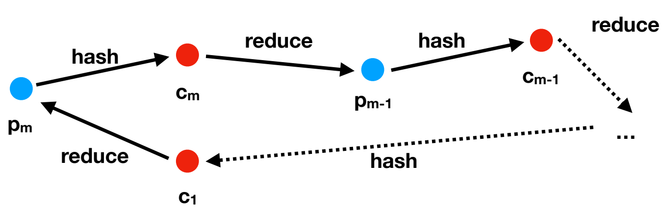
$r(c)=p$, that maps **cipertexts** to **plaintexts**, called a **reducer**.

A reducer is **not the inverse** of the hash!

Thought experiment

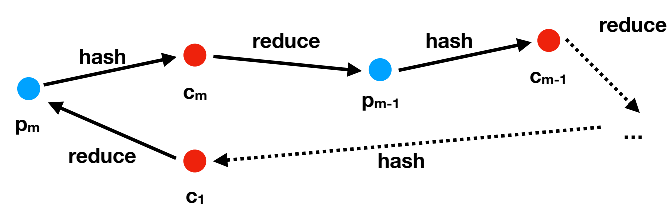
Suppose f(p_i) = c_i

Suppose $r(c_i)$ = p_{i-1} if i>1 otherwise p_m



Thought experiment

Such a scheme (a *hash chain*) lets us generate all plaintexts (and hashes) from a seed plaintext.



Only need to save the seed. **Drawbacks?**

Recap & Next Class

Today we learned:

Password attacks

Password attack complexity

Trading space for time

Next class:

PCHC algorithm