

CPR E 431

## BASICS OF INFORMATION SYSTEM SECURITY

# User Authentication, Access Control, and Operating System

Rainbow Tables and Salting



# Video Summary

- Rainbow Tables
- Salting Passwords
- Storing Passwords with Salt
- Modern Approaches



# Cracking Passwords

- Store passwords and hash values in advance (instead of generating them)
- The question is how big is it?

Password is 8 Bytes + hash is 128 bits (if using MD5)

$$\underbrace{(8 \text{ Byte} + 16 \text{ Byte})} \times 94^8 = 1.4 \times 10^{17} \text{ Bytes} = 146 \text{ TB (approx.)}$$



# Cracking Passwords

- Store passwords and hash values in advance (instead of generating them)
- The question is how big is it?

Password is 8 Bytes + hash is 128 bits (if using MD5)

$(8 \text{ Byte} + 16 \text{ Byte}) \times 94^8 = 1.4 \times 10^{17} \text{ Bytes} = \underline{146 \text{ TB (approx.)}}$

- Instead of generating this huge amount of data we can use

## Rainbow Tables



# Rainbow Tables

- A rainbow table is a precomputed dictionary of plaintext passwords and their corresponding hash values that can be used to find out what plaintext password produces a particular hash.
- Unlike brute-forcing, performing the hash function isn't the problem here. With all of the values already computed, it's simplified to just a simple search-and-compare operation on the table.



# Cracking Passwords

## ➤ Rainbow Tables

### MD5 Rainbow Tables

Table ID	Charset	Plaintext Length	Key Space	Success Rate	Table Size	Files	Performance
md5_ascii-32-95#1-7	ascii-32-95	<u>1 to 7</u>	70,576,641,626,495	<u>99.9 %</u>	52 GB 64 GB	Perfect Non-perfect	Perfect Non-perfect
md5_ascii-32-95#1-8	ascii-32-95	<u>1 to 8</u>	6,704,780,954,517,120	<u>96.8 %</u>	460 GB 576 GB	Perfect Non-perfect	Perfect Non-perfect
md5_mixalpha-numeric#1-8	mixalpha-numeric	<u>1 to 8</u>	221,919,451,578,090	<u>99.9 %</u>	127 GB 160 GB	Perfect Non-perfect	Perfect Non-perfect
md5_mixalpha-numeric#1-9	mixalpha-numeric	<u>1 to 9</u>	13,759,005,997,841,642	<u>96.8 %</u>	690 GB 864 GB	Perfect Non-perfect	Perfect Non-perfect
md5_loweralpha-numeric#1-9	loweralpha-numeric	1 to 9	104,461,669,716,084	<u>99.9 %</u>	65 GB 80 GB	Perfect Non-perfect	Perfect Non-perfect
md5_loweralpha-numeric#1-10	loweralpha-numeric	<u>1 to 10</u>	3,760,620,109,779,060	<u>96.8 %</u>	316 GB 396 GB	Perfect Non-perfect	Perfect Non-perfect

➤ Lookup on 0.5 TB Rainbow Table will take only hours to find the password

 <http://project-rainbowcrack.com/table.htm>



# Pre-calculated Hashes & Rainbow Tables

- ▶ How big is such a database of pre-calculated hashes?
  - ▶ In raw form, generally too big to be practical (100's, 1000's of TB)
  - ▶ Using specialised data structures (e.g. Rainbow tables), can obtain manageable size, e.g. 1 TB
- ▶ Trade-off: reduce search time, but increase storage space
- ▶ Countermeasures:
  - ▶ Longer passwords
  - ▶ Slower hash algorithms
  - ▶ Salting the password before hashing



# Salting Passwords

$$ID, Salt, H(P||Salt)$$

- ▶ When ID and password initially created, generate random  $s$ -bit value (**salt**), concatenate with password and then hash
- ▶ When user submits password, salt from password database is concatenated, hashed and compared
- ▶ If attacker gains database, they know the salt; same effort to find password as brute force attack
- ▶ BUT pre-calculated values (e.g. Rainbow tables) are no longer feasible
  - ▶ Space required increased by factor of  $2^s$



# Salting Passwords

<b>username</b>	<b>salt</b>	<b>H(password    salt)</b>
john	a4H*1	ba586dcb7fe85064d7da80ea6361ddb6
sandy	U9(-f	816a425628d5dee17839fffeafb67144
daniel	5<as4	11842ced4203d4067ed6a6667f3f18d9
...	...	...
steve	LqM4^	184b7f9c6126c568ee50cd3364257973

- The attacker now knows the user name, the salt, and the hash
- How he is going to get the original password?
- How long this will take (worst case) ?

# Salting Passwords

<b>username</b>	<b>salt</b>	<b>H(password    salt)</b>
john	a4H*1	ba586dcb7fe85064d7da80ea6361ddb6
sandy	U9(-f	816a425628d5dee17839fffeafb67144
daniel	5<as4	11842ced4203d4067ed6a6667f3f18d9
...	...	...
steve	LqM4^	184b7f9c6126c568ee50cd3364257973

- The attacker now knows the user name, the salt, and the hash
- How he is going to get the original password?
- How long this will take (worst case) ?
- What is the benefit of using Salt?

# Salting Passwords

- If we used a 16-bits salt then the number of salts available are  $2^{16}$
- To use the Rainbow Table you have to generate a Rainbow table for each possible salt value
- 65536 Rainbow Tables x 0.5 TB per table = 23768 TB
- Another benefit of using salt is that if two users have the same password, they will have different hash values

username	password
john	mysecret
sandy	ld9a%23f
daniel	mysecret
...	...
steve	h31p_m3?

username	H(password)
john	06c219e5bc8378f3a8a3f83b4b7e4649
sandy	5fc2bb44573c7736badc8382b43fbeae
daniel	06c219e5bc8378f3a8a3f83b4b7e4649
...	...
steve	75127c78fd791c3f92a086c59c71ece0



# Password Storage: Best Practice

When storing user login information, always store a hash of a salted password

$$ID, Salt, H(P||Salt)$$

- ▶ Salt: random, generated when ID/password first stored; 32 bits or longer
- ▶ Hash function: slow, adaptive speed (work factor), e.g. bcrypt/scrypt, PBKDF2

Design for failure: assume password database will eventually be compromised

# Password Cracking

## Dictionary attacks

- Develop a large dictionary of possible passwords and try each against the password file
- Each password must be hashed using each salt value and then compared to stored hash values

## Rainbow table attacks

- Pre-compute tables of hash values for all salts
- Can be countered by using a sufficiently large salt value and a sufficiently large hash length

## Password crackers exploit the fact that people choose easily guessable passwords

- Shorter password lengths are also easier to crack

## John the Ripper

- Open-source password cracker first developed in 1996
- Uses a combination of brute-force and dictionary techniques



# Modern Approaches



- Complex password policy

- Forcing users to pick stronger passwords

- However password-cracking techniques have also improved

- The processing capacity available for password cracking has increased dramatically
- The use of sophisticated algorithms to generate potential passwords
- Studying examples and structures of actual passwords in use



# Password Selection Strategies

## User education

Users can be told the importance of using hard to guess passwords and can be provided with guidelines for selecting strong passwords



## Computer generated passwords

Users have trouble remembering them



## Reactive password checking

System periodically runs its own password cracker to find guessable passwords



## Complex password policy

User is allowed to select their own password, however the system checks to see if the password is allowable, and if not, rejects it

Goal is to eliminate guessable passwords while allowing the user to select a password that is memorable

# Video Summary

- Rainbow Tables
- Salting Passwords
- Storing Passwords with Salt
- Modern Approaches

