CSC 405
Password Security

Alexandros Kapravelos
akaprav@ncsu.edu

CSC 405

How to NOT Store Passwords

Alexandros Kapravelos
akaprav@ncsu.edu

The Naive Approach - Just Store Them!

- Nothing stopping you
  - Except you clearly know better…

FirstName	LastName	Email	Password
Andrew	Adams	andrew@chinookcorp.com	password
Nancy	Edwards	nancy@chinookcorp.com	password1
Jane	Peacock	jane@chinookcorp.com	hunter22
Robert	King	robert@chinookcorp.com	robert123!@#

The Naive Approach - Just Store Them!

- Nothing stopping you
  - Except you clearly know better…

FirstName	LastName	Email	Password
Andrew	Adams	andrew@chinookcorp.com	password
Nancy	Edwards	nancy@chinookcorp.com	password1
Jane	Peacock	jane@chinookcorp.com	hunter22
Robert	King	robert@chinookcorp.com	robert123!@#

But there are still companies that use this approach!

Your Registration Information for [redacted]

[redacted]@corporate.[redacted].com
to me

Welcome Adam Gaweda,

Thank you for registering to take [redacted]'s Online Questionnaire!

Your username is: AMGaweda
Your password is: [redacted]

A few years old, but password was clearly plaintexted

You may use your username and password to return to the system at any time to check the status of your application. To return directly to [redacted] [redacted]'s Online Questionnaire, click on this link:
https://[redacted].com.

If you would like to apply for another job with [redacted], click here:
http://www.[redacted]careers.com/.

Thank you for your interest in [redacted].

The Human Resources Team

Storing Password in Plaintext is BAD

- So… **never** do it.

Name: Adams
Composer: andrew@chinookcorp.com
Unit Price: password1

Name: Edwards
Composer: nancy@chinookcorp.com
Unit Price: password

Name: Peacock
Composer: jane@chinookcorp.com
Unit Price: hunter22

Name: Park
Composer: margaret@chinookcorp.com
Unit Price: drowssap

Name: Johnson
Composer: steve@chinookcorp.com
Unit Price: qwertyuiop

Name: Mitchell
Composer: michael@chinookcorp.com
Unit Price: michaelchinookcorpcom

Name: King
Composer: robert@chinookcorp.com
Unit Price: robert123!@#

Name: Callahan
Composer: laura@chinookcorp.com
Unit Price: S3cur3P4$$w0rd

Less Naive Approach - Encrypt It

- Good intentions… bad execution

FirstName	LastName	Email	Password
Andrew	Adams	andrew@chinookcorp.com	cGFzc3dvcmQ=
Nancy	Edwards	nancy@chinookcorp.com	cGFzc3dvcmQx
Jane	Peacock	jane@chinookcorp.com	aHVudGVyMjI=
Robert	King	robert@chinookcorp.com	cm9iZXJ0MTIzIUAj

Base64

Less Naive Approach - Encrypt It

- Good intentions… bad execution
- Similar passwords will have similar encryptions

FirstName	LastName	Email	Password
Andrew	Adams	andrew@chinookcorp.com	cGFzc3dvcmQ= (password)
Nancy	Edwards	nancy@chinookcorp.com	cGFzc3dvcmQx (password1)
Jane	Peacock	jane@chinookcorp.com	aHVudGVyMjI=
Robert	King	robert@chinookcorp.com	cm9iZXJ0MTIzIUAj

Base64

Less Naive Approach - Encrypt It

- Good intentions… bad execution
- Similar passwords will have similar encryptions
- Also, common encryptions have decoders online

FirstName	LastName	Email	Password
Andrew	Adams	andrew@chinookcorp.com	cGFzc3dvcmQ= (password)
Nancy	Edwards	nancy@chinookcorp.com	cGFzc3dvcmQx (password1)
Jane	Peacock	jane@chinookcorp.com	aHVudGVyMjI=
Robert	King	robert@chinookcorp.com	cm9iZXJ0MTIzIUAj

Base64

Less Naive Approach - Encrypt It

- Good intentions… bad execution
- Similar passwords will have similar encryptions
- Also, common encryptions have decoders online

FirstName	LastName	Email	Password
Andrew	Adams	andrew@chinookcorp.com	cGFzc3dvcmQ= (password)
Nancy	Edwards	nancy@chinookcorp.com	cGFzc3dvcmQx (password1)
Jane	Peacock	jane@chinookcorp.com	aHVudGVyMjI=
Robert	King	robert@chinookcorp.com	cm9iZXJ0MTIzIUAj

- Another way to think about it: **Encryption = Reversible**

Still Naive Approach - Hash It

- Better…
- **Hashing = Irreversible***

FirstName	LastName	Email	Password
Andrew	Adams	andrew@chinookcorp.com	5f4dcc3b5aa7...
Nancy	Edwards	nancy@chinookcorp.com	7c6a180b3689...
Jane	Peacock	jane@chinookcorp.com	cb95015a436f...
Robert	King	robert@chinookcorp.com	3f94b11a9f70...

MD5

Password Cracking - Hashcat

```
$ hashcat --potfile-disable -m 0 pw.txt
          /usr/share/wordlists/rockyou.txt

hashcat (v6.2.6) starting
...
Dictionary cache hit:
* Filename..: /usr/share/wordlists/rockyou.txt
* Passwords.: 14344385
* Bytes.....: 139921507
...
5f4dcc3b5aa765d61d8327deb882cf99:password
7c6a180b36896a0a8c02787eeafb0e4c:password1
cb95015a436fe976eb38e45455372032:hunter22
```

derived from a data breach of the RockYou website in 2009. This breach exposed millions of plaintext passwords.

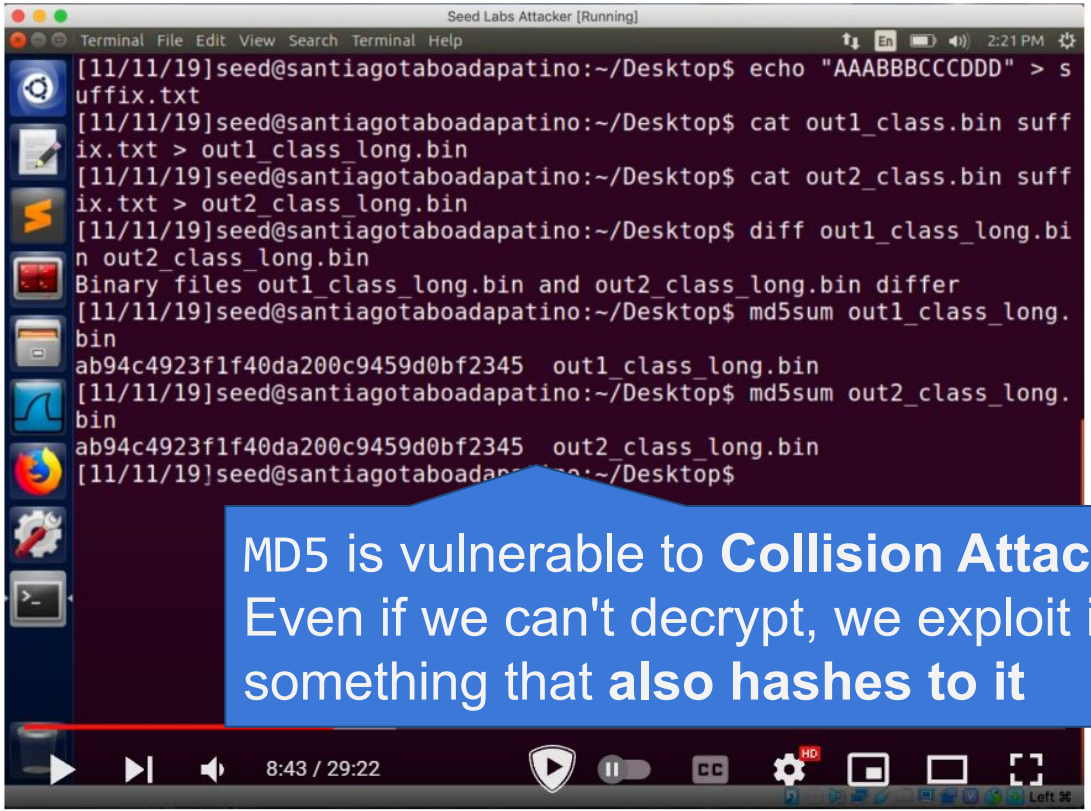
Password Cracking - Hashcat

```
$ hashcat --potfile-disable -m 0 pw.txt
          /usr/share/wordlists/rockyou.txt

hashcat (v6.2.6) starting
...
Dictionary cache hit:
* Filename..: /usr/share/wordlist
* Passwords.: 14344385
* Bytes.....: 139921507
...
5f4dcc3b5aa765d61d8327deb882cf99:password
7c6a180b36896a0a8c02787eeafb0e4c:password1
cb95015a436fe976eb38e45455372032:hunter22
```

Didn't catch robert123!@# but you can add rules to append numbers/ symbols to common words

MD5 = BAD



Seed Labs: MD5 Collision Attack https://www.youtube.com/watch?v=qi10gDfeoU4

Still Naive Approach - Hash It

- Obviously the issue was I used MD5 instead something like `SHA-128` or `SHA-256`!

FirstName	LastName	Email	Password
Andrew	Adams	andrew@chinookcorp.com	5e884898da28...
Nancy	Edwards	nancy@chinookcorp.com	0b14d501a594...
Jane	Peacock	jane@chinookcorp.com	20d2fe5e369d...
Robert	King	robert@chinookcorp.com	2feb713a06cd...

SHA-256

Still Naive Approach - Hash It

- O … or
  S…

FirstName	
Andrew	.
Nancy	.
Jane	.
Robert	.

…56

SHA-1 vs SHA-2

- The same but different (block ciphers)

- SHA-1
  - 160-bit hash
  - Can have a collision with 110 years of GPU time
    - Not super feasible for most entities, but possible

- SHA-2
  - Bit size can range from 256 to 512
  - Varying codes (SHA-224, SHA-256, SHA-384, SHA-512) refer to their output bit size

SHA-1 vs SHA-2

- The same but different (block ciphers)

- SHA-1
  - 160-bit hash
  - Can have a collision with 110 years of GPU time
    - Not super feasible for most entities, but possible
  - officially deprecated by NIST in 2011

- SHA-2
  - Bit size can range from 256 to 512
  - Varying codes (SHA-224, SHA-256, SHA-384, SHA-512) refer to their output bit size

- SHA-3 is now available

Dictionary Attacks FTW

```
$ hashcat --potfile-disable -m 1400
          pw_sha256.txt /usr/share/wordlists/rockyou.txt

hashcat (v6.2.6) starting
...
Dictionary cache hit:
* Filename..: /usr/share/wordlists/rockyou.txt
* Passwords.: 14344385
* Bytes.....: 139921507
...
5e884898da28...42d8:password
0b14d501a594...c94e:password1
20d2fe5e369d...eb0b:hunter22
```

Common Passwords are super easy to attack

Rainbow Tables

- However, passwords like `robert123!@#` can still avoid cracking…
- Unless Robert uses it somewhere else that was hacked.

Rainbow Tables

- However, passwords like `robert123!@#` can still avoid cracking…
- Unless Robert uses it somewhere else that was hacked.

- **Rainbow Tables** are stored hash decryptions done on other passwords and stored
  - Trades computational time for hard disk space
  - LARGE file sizes

Rainbow Table Specification

Algorithm	Table ID	Charset	Plaintext Length	Key Space	Success Rate	Table Size	Files
LM	lm_ascii-32-65-123-4#1-7	ascii-32-65-123-4	1 to 7	7,555,858,447,479 ≈ $2^{42.8}$	99.9 %	27 GB	Files
NTLM	ntlm_ascii-32-95#1-7	ascii-32-95	1 to 7	70,576,641,626,495 ≈ $2^{46.0}$	99.9 %	52 GB	Files
NTLM	ntlm_ascii-32-95#1-8	ascii-32-95	1 to 8	6,704,780,954,517,120 ≈ $2^{52.6}$	96.8 %	460 GB	Files
NTLM	ntlm_mixalpha-numeric#1-8	mixalpha-numeric	1 to 8	221,919,451,578,090 ≈ $2^{47.7}$	99.9 %	127 GB	Files
NTLM	ntlm_mixalpha-numeric#1-9	mixalpha-numeric	1 to 9	13,759,005,997,841,642 ≈ $2^{53.6}$	96.8 %	690 GB	Files
NTLM	ntlm_loweralpha-numeric#1-9	loweralpha-numeric	1 to 9	104,461,669,716,084 ≈ $2^{46.6}$	99.9 %	65 GB	Files
NTLM	ntlm_loweralpha-numeric#1-10	loweralpha-numeric	1 to 10	3,760,620,109,779,060 ≈ $2^{51.7}$	96.8 %	316 GB	Files

Rainbow Tables

- However, passwords like `robert123!@#` can still avoid cracking…
- Unless Robert uses it somewhere else that was hacked.

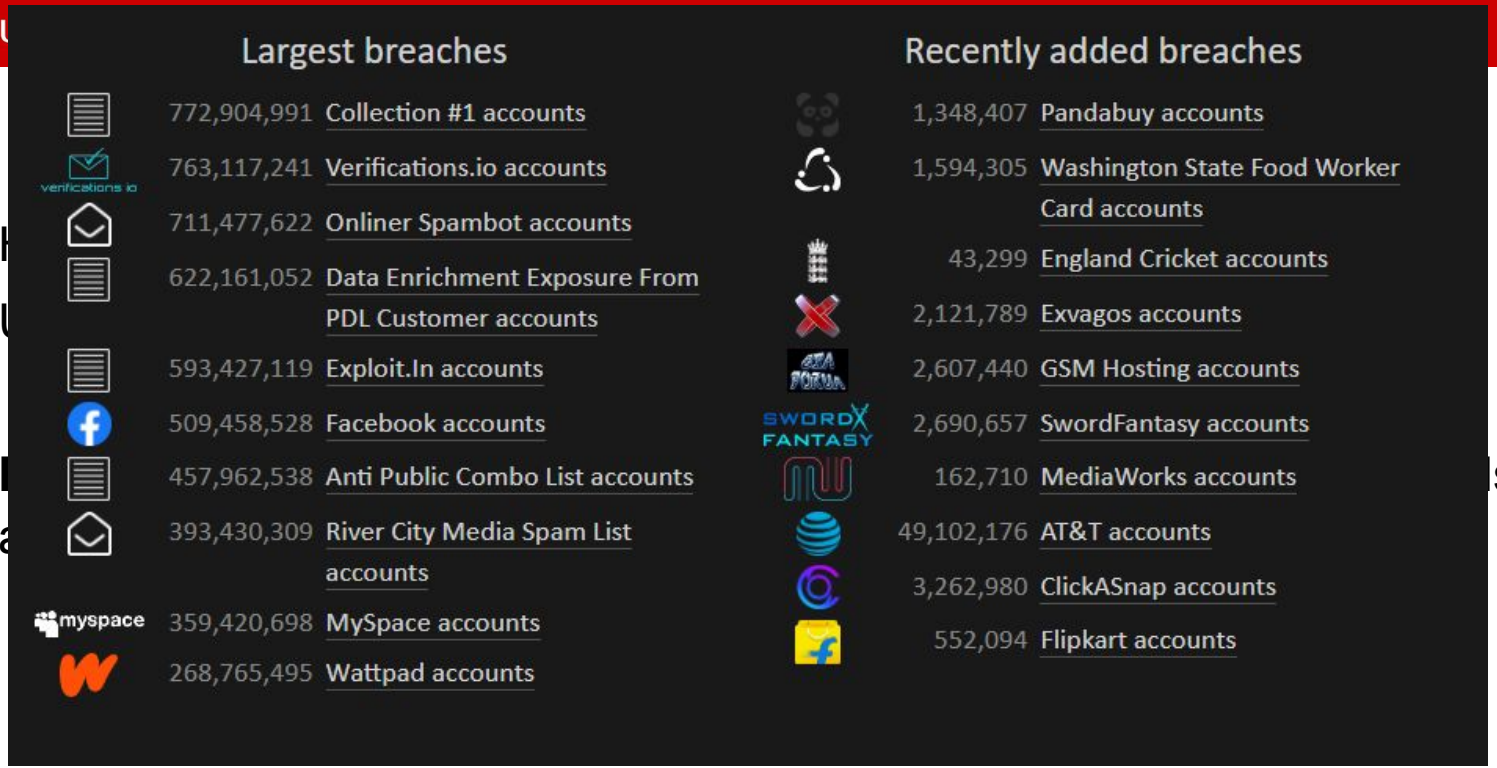
- **Rainbow Tables** are stored hash decryptions done on other passwords and stored
  - Trades computational time for hard disk space
  - LARGE file sizes

Rainbow Table Specification

Algorithm	Table ID	Charset	Plaintext Length	Key Space	Success Rate	Table Size	Files
NTLM	ntlm_loweralpha-numeric#1-10	loweralpha-numeric	1 to 10	3,760,620,109,779,060 ≈ $2^{51.7}$	96.8 %	316 GB	Files

But no one would ever reuse a password, …right?

- H
- U
- I s



haveibeenpwned.com

Current Best Approach - Salted Hash It

- Since SHA-256 will always encrypt robert123!@# to 2feb713a06…, we can mitigate this be **adding in some extra text**

FirstName	LastName	Email	Password	Salt
Andrew	Adams	andrew...	ae69caf5f4b4...	cxwnzrgwos
Nancy	Edwards	nancy...	c7bc75baf50a...	lgocdjosyn
Jane	Peacock	jane...	511dec4125ee...	bqkxuuqmbj
Robert	King	robert...	7ae0cd4700a3...	ctkwwudnyx

Current Best Approach - Salted Hash It

- Since SHA-256 will always encrypt robert123!@# to 2feb713a06…, we can mitigate this be **adding in some extra text**
- Storing the salt in the database is "fine"
  - Having the attacker know the salt does not make the task easier and still protects "`robert123!@#`" from other attacks

FirstName	LastName	Email	Password	Salt
Andrew	Adams	andrew...	ae69caf5f4b4...	cxwnzrgwos
Nancy	Edwards	nancy...	c7bc75baf50a...	lgocdjosyn
Jane	Peacock	jane...	511dec4125ee...	bqkxuuqmbj
Robert	King	robert...	7ae0cd4700a3...	ctkwwudnyx

Current Best Approach - Salted Hash It

- Since SHA-256 will always encrypt robert123!@# to 2feb713a06…, we can mitigate this be **adding in some extra text**
- Storing the salt in the database is "fine"
  - Having the attacker know the salt does not make the task easier and still protects "`robert123!@#`" from other attacks

FirstName	LastName	Email	Password	Salt
Andrew	Adams	andrew...	ac69caf5f4b4	cxupzngvos
Nancy	Edwards	nancy...		
Jane	Peacock	jane...		
Robert	King	robert...	7ae0cd4700a3...	ctkwwudnyx

Instead of hashing "**`robert123!@#`**", you hash "**`ctkwwudnyxrobert123!@#`**"

Making Salted Passwords

```
import hashlib, random, string

def make_salt(length=120):
  salt = ''
  for i in range(length):
    salt += random.choice(string.ascii_letters)
  return salt

def make_pw_hash(name, pw):
  salt = make_salt()
  to_encode = str(pw + salt).encode('utf-8')
  hashed = hashlib.sha256(to_encode).hexdigest()
  return hashed
```

Validating Salted Passwords

```
def valid_user(email, password):
    user = User.query.filter_by(email=email).first()
    salt = user.salt
    hashed_pw = make_pw_hash(password, salt)
```

Fine to store `salt` in DB, since we still need the user's input to make the hash

```
    if (user.password == hashed_pw):
        return user
    return False
```

If the hashed password doesn't equal the stored, hashed password, then invalid login

Clear Takeaways

- Salt passwords
  - Maybe add a little pepper

Clear Takeaways

- Salt passwords
  - Maybe add a little pepper

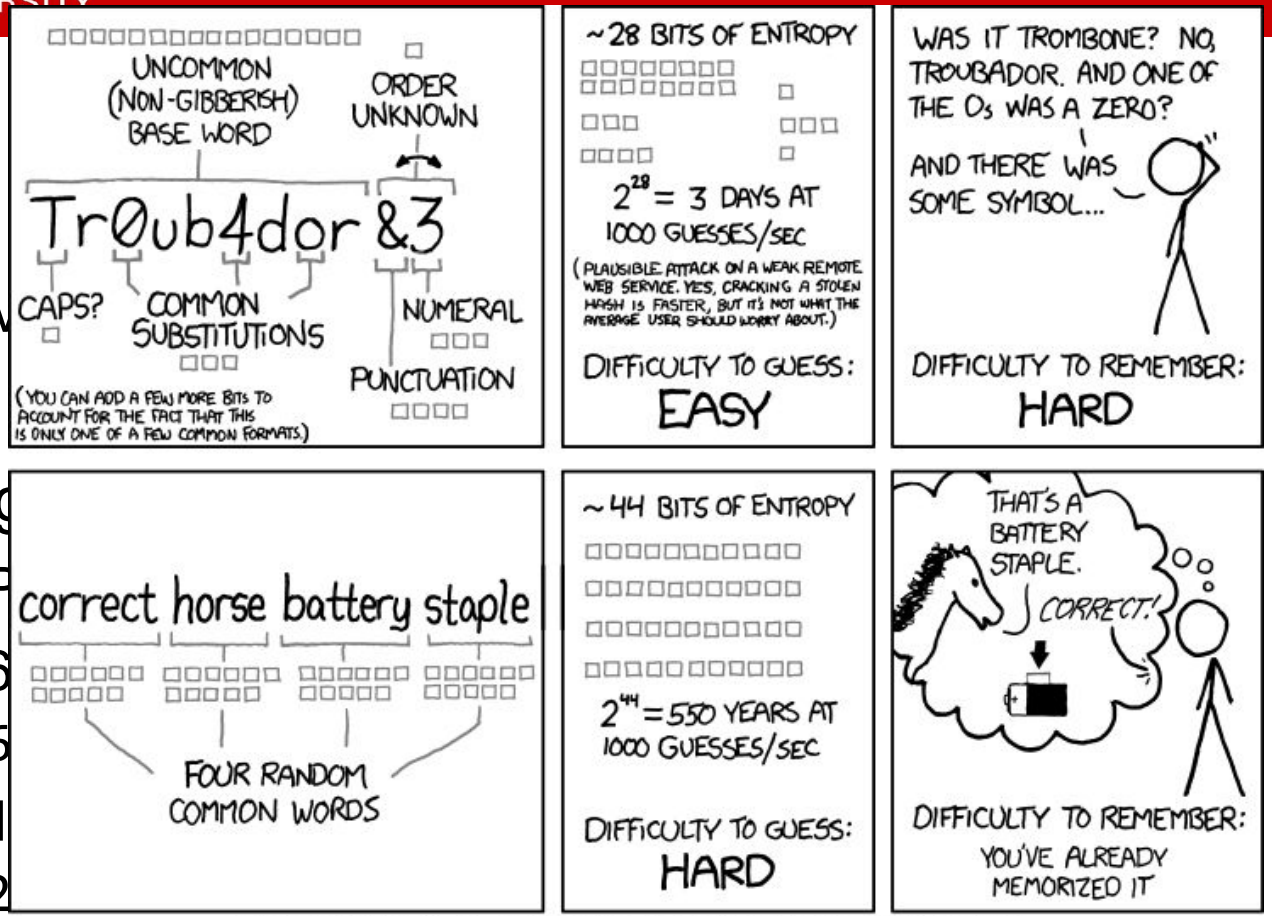
- Length > Complexity
  - Possibilities = complexity ^{length}
  - 6 character password with `a-z, A-Z, 0-9` characters
    62^6 = `56,800,235,584` possibilities

Clear Takeaways

- Salt passwords
  - Maybe add a little pepper

- Length > Complexity
  - Possibilities = complexity ^{length}
  - 6 character password with `a-z, A-Z, 0-9` characters
    62^6 = 56,800,235,584 possibilities
  - 10 character password with only `a-z` characters
    26^{10} = 141,167,095,653,376 possibilities

- Salt
  - M
- Leng
  - F
  - 6
    6
  - 1
    2



https://xkcd.com/936/