CS 55:
Security and Privacy

Introduction; password cracking

Zoom poll

Where are you located?

With great power comes great responsibility …

OR

William Lamb, 2nd Viscount Melbourne

We will cover some techniques that could land you in jail if you use them… be wise

Spider Man's uncle Ben

Example: door entry

DO NOT show unless everyone agrees NOT to do this at Dartmouth!

Example: door entry

https://www.youtube.com/watch?v=xcA7iXSNmZE&start=06

Example: door entry

Countermeasures?

https://www.youtube.com/watch?v=xcA7iXSNmZE&start=06

Another example: door entry

https://www.youtube.com/watch?v=QSQo4bcfLbY&start=23

Another example: door entry

Countermeasures?

https://www.youtube.com/watch?v=QSQo4bcfLbY&start=23

Agenda

1. Course logistics
2. Passwords

We are meeting online this term

- We will use Zoom for class meetings during normal class hours (I will record and post each class session)
- We will see what works as we go, but here are some rules for starters:
  - Start with your camera on but microphone muted
  - If you would like to ask a question, use the “Raise hand” feature in Zoom, then when called on, ask your question and lower hand
  - We will make additional rules as we need them…
- I’ll assume you’ve done the reading for the day, in class I’ll expand/extend the material from the book, and will not simply repeat the book back to you
- A large portion of each day will be in-class discussions and demos on a live system

This class is about security and privacy; we will consider seven major topics

Seven main topics for the course:

1. Adversaries
2. Cryptography
3. Common attacks
4. Defensive tools
5. Security operations
6. Privacy and the law
7. The future

This course is *NOT*:
A course on how to hack

- **We will implement attacks to understand how they work**
- **We will also discuss countermeasures**

A course for script kiddies!

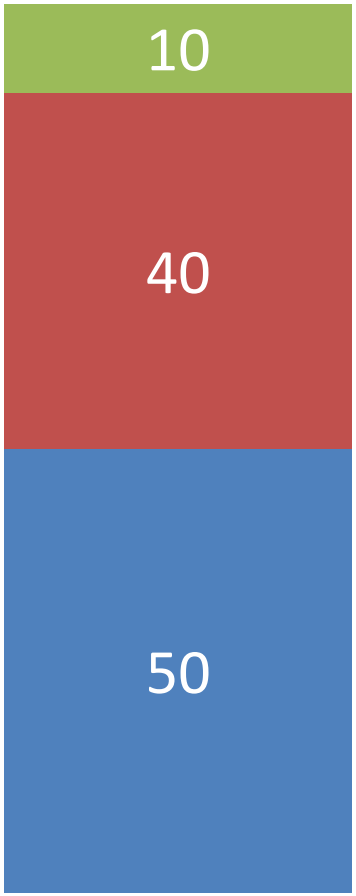
- **There are hundreds (thousands?) of scripts you can run to implement attacks without understanding how they work**
- **We will write our own code**
- **I'll assume you know Python and C**

We will not cover many network attacks

- **CS60 is not a prereq**

Assessment is based on five labs, two exams, and class participation

Syllabus: http://www.cs.dartmouth.edu/~tjp/cs55



Class participation

Two exams

- Midterm and final
- Each worth 20%

Five labs

- Each worth 10%
- Will work with two randomly assigned partners

Notes:

Readings from *Security in Computing,* 5th ed, by Pfleeger, Pfleeger, and Margulies

Laptop use in class is ~~encouraged~~ required – Google is your friend

NOTE: Lab 0 is simply to gather information

We will also be using Canvas and Slack for announcements and help

Online resources

Canvas

- Course announcements
- Lab submissions

Slack

- We will use Slack in place of Piazza
- Sign up via link on left side of Canvas
- Ask questions, but do not post solutions related to the labs or the exams

Lab 0 is out now

Lab 0

- Find it on Canvas
- Take course survey to understand your background
- Set Virtual Box and VM we will use for experiments
- Read and acknowledge course policies

Discussion

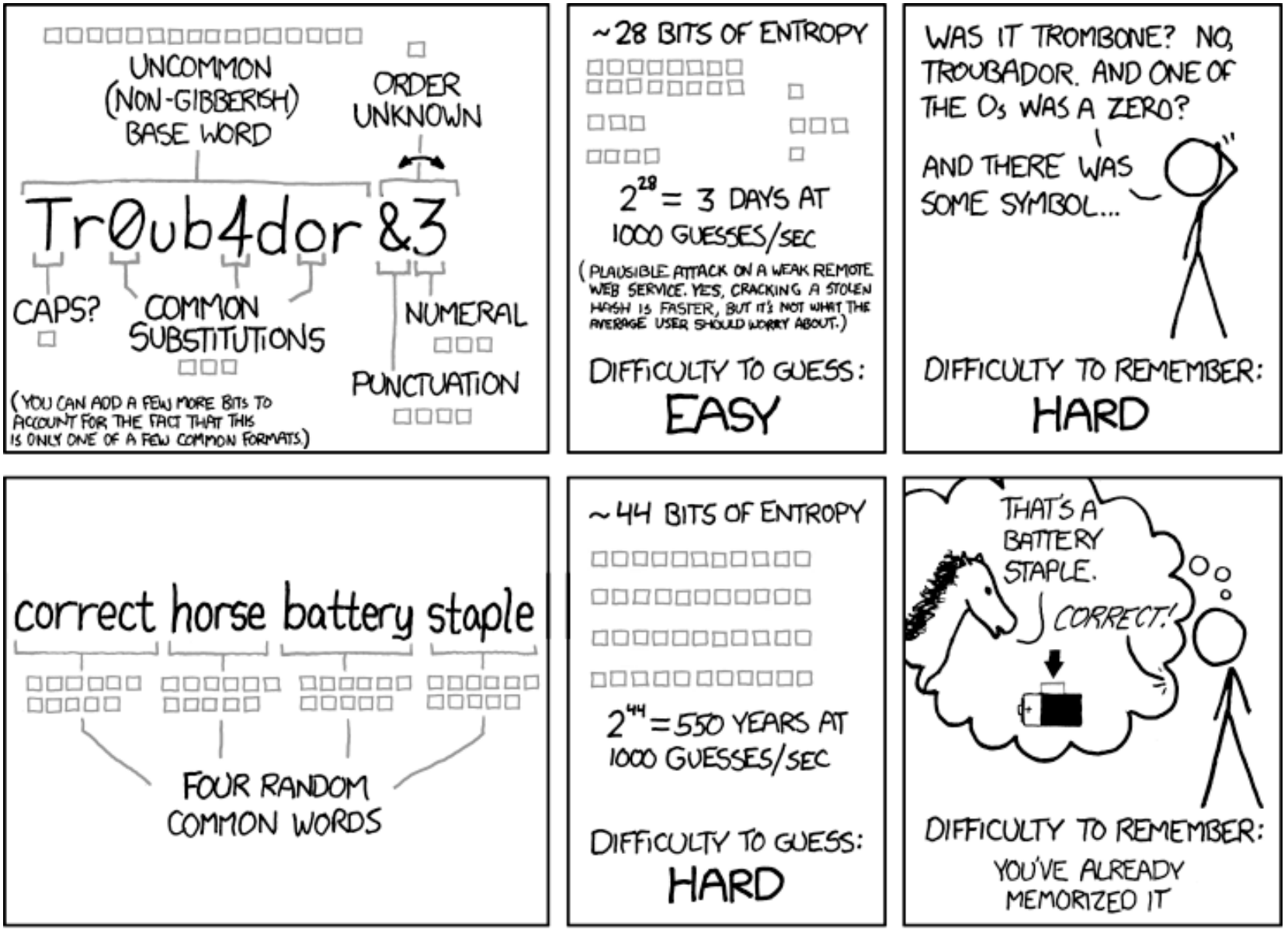
Survey: your background

My background

What is the difference between security and privacy?

Agenda

1. Course logistics
2. Passwords



THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

Source: https://xkcd.com/936/

Discussion:

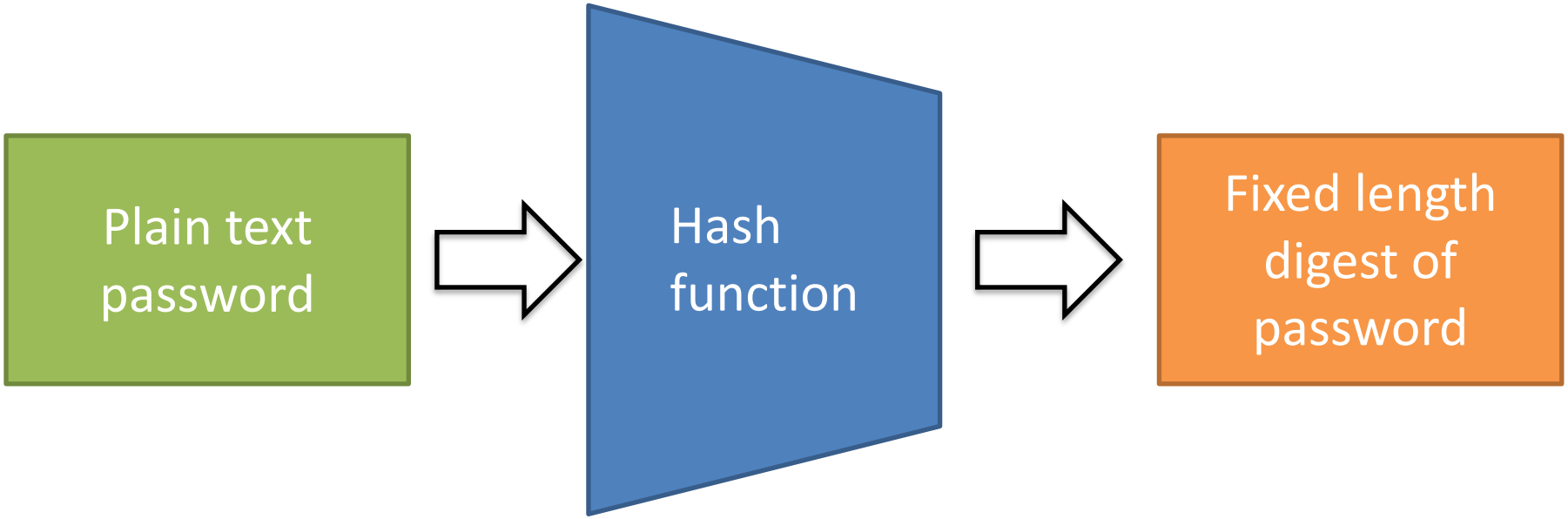
1. What is the difference between identification and authentication?

2. What are some of the problems with using passwords for authentication?

3. How do adversaries crack passwords?

Do not store passwords in plain text!

Use hashing to avoid storing passwords in plaintext



Input:
"my secret password"

Output:
a7303f3eee5f3ff1942bfbb1797ea0af

Hash function is a deterministic mathematical one-way trap door

- **Collision resistant – different plaintext do not produce same hash digest**
- **Choose hash functions that take a long time to computer (e.g., bcrypt)**
- **Cannot find plain text in "reasonable" amount of time given only the hash digest**
- **Or can we?**

DO NOT store user passwords in plain text!

UserID	UserName	UserPassword
1	testuser	password
2	testuser1	password
3	testuser2	password1
4	testuser3	my secret password

Do not store passwords in plain text

Note: same password results in same hash

If adversary steals passwords, cannot read plain-text password

UserID	UserName	HashedPassword
1	testuser	5f4dcc3b5aa765d61d8327deb882cf99
2	testuser1	5f4dcc3b5aa765d61d8327deb882cf99
3	testuser2	7c6a180b36896a0a8c02787eeafb0e4c
4	testuser3	a7303f3eee5f3ff1942bfbb1797ea0af

Instead store hash of password

On log in: hash plain text password and compare with database

Username: “testuser”
Password: “password”

Hash Password

Hash user’s plain text password and look for match in database

Because hash function is deterministic, same password will always result in same digest

Hashed password:
5f4dcc3b5aa765d61d8327deb882cf99

Hashes match for testuser

UserID	UserName	HashedPassword
1	testuser	5f4dcc3b5aa765d61d8327deb882cf99
2	testuser1	5f4dcc3b5aa765d61d8327deb882cf99
3	testuser2	7c6a180b36896a0a8c02787eeafb0e4c
4	testuser3	a7303f3eee5f3ff1942bfbb1797ea0af

Know user submitted valid password without storing in DB

Dictionary attack: try all words in a dictionary looking for a match

Username: “testuser”
Password: “password”

Hash Password

Hashed password:
5f4dcc3b5aa765d61d8327deb882cf99

Change your password if in dictionary!

Assume adversary steals hashed passwords

Password: “aardvark”
Password: “alice”
Password: “anteater”
…
Password: “password”

Dictionary attack:
Hash all words in a dictionary, if word hash matches database hash, password is “cracked”

UserID	UserName	HashedPassword
1	testuser	5f4dcc3b5aa765d61d8327deb882cf99
2	testuser1	5f4dcc3b5aa765d61d8327deb882cf99
3	testuser2	7c6a180b36896a0a8c02787eeafb0e4c
4	testuser3	a7303f3eee5f3ff1942bfbb1797ea0af

Will not crack if user’s password not in dictionary

Crack one, crack all with same password

Rainbow table attacks precompute all possible character combinations

Username: “testuser”
Password: “password”

Hash Password

Hashed password:
5f4dcc3b5aa765d61d8327deb882cf99

Password: “a”
Password: “aa”
Password: “aaa”
…
Password: “password”

Assume adversary steals hashed passwords

Rainbow table attack:
Precompute all character combinations up to certain length
Store resulting hash for each combo

UserID	UserName	HashedPassword
1	testuser	5f4dcc3b5aa765d61d8327deb882cf99
2	testuser1	5f4dcc3b5aa765d61d8327deb882cf99
3	testuser2	7c6a180b36896a0a8c02787eeafb0e4c
4	testuser3	a7303f3eee5f3ff1942bfbb1797ea0af

Look up database password in rainbow table

Lots of time and storage needed

Length limited

Use salt to prevent rainbow table attacks

Username: “testuser”
Password: “password”

Password + **Salt**: “password**.ef_ob’3**”
Salted hashed password:
62c21dd30b2d7e6e6671628458aeaf1f

Salt:

- **Random string of characters appended (or prepended or both) to password before hash**
- **Each user gets unique salt**
- **Salt stored in plain text in database**
- **User need not know value of salt, it is added on server side**

If salt is long (say 64 characters) rainbow table is impractical

Dictionary attack still possible

- **Password plus salt unlikely to be in dictionary**
- **Adding salt to each word slows down adversary**

UserID	UserName	Salt	SaltedPassword
1	testuser	.ef_ob'3	62c21dd30b2d7e6e6671628458aeaf1f
2	testuser1	\s#>2!x}	a9055805cb7e588dc27945cb95067f6b
3	testuser2	as=8KlA=	19e3385ed6bfe321b36b6bc4290bea0b
4	testuser3	n%lzA7QQ	a6fc8f715df44839b50cf31b639b961c

Adding pepper is even better

Username: “testuser”
Password: “password”

Password

Password + Salt **+ Pepper**: “password.ef_ob’3**Secret**”
Salted hashed password:
2811922850bbcd79683b58e43d1ab76f

UserID	UserName	Salt	SaltedPassword
1	testuser	.ef_ob'3	62c21dd30b2d7e6e6671628458aeaf1f
2	testuser1	\s#>2!x}	a9055805cb7e588dc27945cb95067f6b
3	testuser2	as=8KlA=	19e3385ed6bfe321b36b6bc4290bea0b
4	testuser3	n%lzA7QQ	a6fc8f715df44839b50cf31b639b961c

Pepper:

- **Random string of characters appended to password + salt before hash**
- **Pepper kept secret, not stored in database**
- **One pepper for all users**

Another variant

- **Pepper is one character chosen at random for each user**
- **Not stored**
- **On login, try ‘a’, then ‘b’**
  - **Will eventually find match**
  - **Slows adversary**

Do not use passwords that are in the dictionary!

Exercise

Enter username and password at phony sign up site:

http://vibrantcloud.org/cs55/saveUser.html

Site stores entries into Users table in a database

- Table has unique constraint on UserName (so choose something else if what you enter is already taken)
- NOTE: for demonstration purposes only, it stores the password in plain text! *You would not do this is production*!
- Also stores hashed and salted hash passwords

Assume an adversary does a SQL injection attack (or otherwise steals Users table) and gets usernames with hashed and salted passwords

- What can they do? They do not have the users' passwords
- Enter hashcat!

Hashcat is a password hashing tool

1. **Download usertable.csv from database**

2. **Extract hashes from usertable.csv**

   cat usertable.csv | awk "-F," '{print $4}' > unsalted.txt
   cat usertable.csv | awk "-F," '{print $6 ":" $5}' > salted.txt

-m is hash type:
- **0 = MD5**
- **10 = MD5(password+salt)**

-a is attack mode
- **0 = dictionary**
- **rockyou.txt is dictionary**

--potfile-disable means restart

3. **Crack passwords**

   Unsalted

   hashcat -m 0 -a 0 unsalted.txt ~/Downloads/rockyou.txt --potfile-disable

   Salted

   hashcat -m 10 -a 0 salted.txt ~/Downloads/rockyou.txt --potfile-disable

Exercise: crack passwords on local machine

Linux stores passwords in file **`/etc/shadow`**

- Take a look at them: **`cat /etc/shadow`**
  - Hmmm, need sudo privledges (we will see how to get them via a buffer overflow attack soon)
  - Assume we have sudo rights for now
  - **`sudo cat /etc/shadow`**
- Linux stores passwords in hashed form
  - Find out what form of hashing is used:
    `grep –A 18 ENCRYPT_METHOD /etc/login.defs`
  - Edit out username and end, save as shadow1.txt
    ~~`joeyjojo:`~~`$6$4Lh9NoaI$g3gJWZYhNaGC7FgiNHKci6izK5YJ7hAC9`
    `7LVN9tqJ07ULtOtZMyDJxQWzhBqHQ3CRsEU1ooSFx9A0eaOtac4m/`~~`÷`~~
    ~~`18463:0:99999:7:::`~~
- Crack with hashcat
  `hashcat –m 1800 –a 0 shadow1.txt ~/Downloads/rockyou.txt`
  `––potfile–disable –O`

Notice hashing is much slower with SHA512 than MD5!

Adapted from https://samsclass.info/123/proj10/p12-hashcat.htm

Before next class

1. Complete Lab 0

2. Read textbook: chapter 1.1-1.6

Security and privacy fails

https://brightside.me/wonder-curiosities/20-ridiculous-security-fails-that-are-too-good-to-be-true-426810/

Security and privacy fails

https://brightside.me/wonder-curiosities/20-ridiculous-security-fails-that-are-too-good-to-be-true-426810/

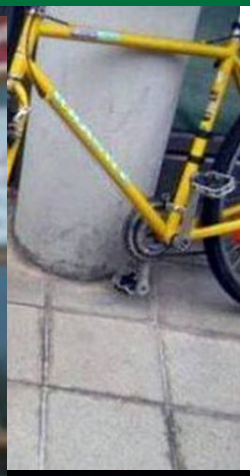
Security and privacy fails

https://brightside.me/wonder-curiosities/20-ridiculous-security-fails-that-are-too-good-to-be-true-426810/

Security and privacy fails

https://brightside.me/wonder-curiosities/20-ridiculous-security-fails-that-are-too-good-to-be-true-426810/

My favorite



https://brightside.me/

Discussion:

1. What is the difference between identification and authentication?
2. What are some of the problems with using passwords for authentication?
3. How do adversaries crack passwords?
4. How to choose a good password?
5. Should you write your passwords down?
6. Will passwords ever go away?