Week 11: Lecture A
Secure Authentication
Tuesday, November 7, 2023
Announcements

- **Project 3: WebSec** released
  - **Deadline:** November 9th by 11:59PM (**this Thursday**)

---

**Project 3: Web Security**

**Deadline:** Thursday, November 9 by 11:59PM.

Before you start, review the course syllabus for the Lateness, Collaboration, and Ethical Use policies.

You may optionally work alone, or in teams of **at most two** and submit **one project per team**. If you have difficulties forming a team, post on Piazza's Search for Teammates forum. Note that the final exam will cover project material, so you and your partner should collaborate on each part.

The code and other answers your group submits must be entirely your own work, and you are bound by the University's Student Code. You may consult with other students about the conceptualization of the project and the meaning of the questions, but you may not look at any part of someone else's solution or collaborate with anyone outside your group. You may consult published references, provided that you appropriately cite them (e.g., in your code comments). **Don't risk your grade and degree by cheating!**

Complete your work in the **CS 4440 VM** — we will use this same environment for grading. You may not use any **external dependencies**. Use only default Python 3 libraries and/or modules we provide you.
<table>
<thead>
<tr>
<th>Status</th>
<th>Progress</th>
</tr>
</thead>
<tbody>
<tr>
<td>Working on Part 1</td>
<td>0%</td>
</tr>
<tr>
<td>Finished Part 1, working on Part 2</td>
<td>0%</td>
</tr>
<tr>
<td>Finished Part 2, working on Part 3</td>
<td>0%</td>
</tr>
<tr>
<td>Finished with everything!</td>
<td>0%</td>
</tr>
<tr>
<td>Haven’t started yet :(</td>
<td>0%</td>
</tr>
</tbody>
</table>
Announcements

- **Project 4: NetSec** released
  - **Deadline:** Thursday, December 7th by 11:59PM

**Project 4: Network Security**

**Deadline:** Thursday, December 7 by 11:59PM.

Before you start, review the course syllabus for the Lateness, Collaboration, and Ethical Use policies.

You may optionally work alone, or in teams of at most two and submit one project per team. If you have difficulties forming a team, post on Piazza’s Search for Teammates forum. Note that the final exam will cover project material, so you and your partner should collaborate on each part.

The code and other answers your group submits must be entirely your own work, and you are bound by the University’s Student Code. You may consult with other students about the conceptualization of the project and the meaning of the questions, but you may not look at any part of someone else’s solution or collaborate with anyone outside your group. You may consult published references, provided that you appropriately cite them (e.g., in your code comments). **Don’t risk your grade and degree by cheating!**

Complete your work in the **CS 4440 VM** — we will use this same environment for grading. You may not use any external dependencies. Use only default Python 3 libraries and/or modules we provide you.
See Discord for meeting info!

www.utahsec.com
Interested in automated bug-finding?

  - Everything you’d ever want to know about fuzzing for finding **security bugs**!
  - Course project: team up to fuzz a **real program** (of your choice), and find and report its bugs!
  - [https://cs.utah.edu/~snagy/courses/cs5963/](https://cs.utah.edu/~snagy/courses/cs5963/)
Questions?
Last time on CS 4440...

Attacks on Security Properties
Denial of Service Attacks
Basic Security Properties

- Confidentiality: ???
- Authenticity: ???
- Integrity: ???
- Access Control: ???
- Availability: ???
Basic Security Properties

- **Confidentiality**: Concealment of information or resources
  - **Attacks**: intercept credentials, info

- **Authenticity**: Identification and assurance of info origin
  - **Attacks**: SMTP header spoofing

- **Integrity**: Preventing improper and unauthorized changes
  - **Attacks**: tampering HTML over HTTP

- **Access Control**: Enforce who is allowed access to what
  - **Attacks**: web app code injection

- **Availability**: Ability to use desired information or resource
  - **Attacks**: denial of service
DoS: Denial of Service

- Goal: ???
DoS: Denial of Service

- **Goal**: make a service unusable, usually by overloading the server or network

- **How?**
**DoS: Denial of Service**

- **Goal:** make a service unusable, usually by overloading the server or network

- **How?**
  - Trigger the host to **crash**
    - Application-based DoS
    - Memory corruption
  - Consume host’s **resources**
    - TCP SYN floods
    - ICMP ECHO (ping) floods
  - Consume host’s **bandwidth**
    - UDP floods
    - ICMP floods
Distributed DoS Attacks (DDoS)
Distributed DoS Attacks (DDoS)

- Attacker
- Handler
- Agent
- Victim
- Agent
- Agent
- Handler
- Agent
- Agent

???
Distributed DoS Attacks (DDoS)
Distributed DoS Attacks (DDoS)
Distributed DoS Attacks (DDoS)
Distributed DoS Attacks (DDoS)
Distributed DoS Attacks (DDoS)

- **Attacker**
- **Handler**
- **Agent**
- **Victim**

*Rootkit-owned victim systems*
Distributed DoS Attacks (DDoS)

Can the victim easily identify the attacker?
Distributed DoS Attacks (DDoS)

No—many layers of obfuscation!
Advanced DoS Strategies

- Reflection:
  - ???
Advanced DoS Strategies

- **Reflection:**
  - IP spoofing to redirect response to a victim

- **Amplification:**
  - ???
Advanced DoS Strategies

- **Reflection:**
  - IP spoofing to redirect response to a victim

- **Amplification:**
  - Technique that increases the amount of traffic or packet size that the victim sees versus what the attacker originally sent

- **How do these make detection harder?**
  - ???
Advanced DoS Strategies

- **Reflection:**
  - IP spoofing to redirect response to a victim

- **Amplification:**
  - Technique that increases the amount of traffic or packet size that the victim sees versus what the attacker originally sent

- **How do these make detection harder?**
  - Source remains **obfuscated**
  - Source constantly **changes**
DDoS or legitimate traffic?

u/TheRealAndyReid

“OMG... Joe’s in KC serves the BEST brisket sandwich”

“Ooooh!”
“Click!”
“Order now!”
“Hungry!”
“Yum!”
“I must try!”
“I’m from Cali and am clueless about BBQ!”

https://www.joeskc.com/
The TCP Three-way Handshake

- **Recall:** TCP is a *connection-oriented* protocol
  - Initiate with three-way “handshake”: SYN, SYN-ACK, ACK
  - Server *waits* until client responds with ACK

Client: SYN

Server: SYN-ACK

Server: *Hurry up!*

Client: ACK
SYN Flooding Attack

- Attack: ???
SYN Flooding Attack

- **Attack:** spam SYN packets to server, with spoofed origin address
  - Server’s resources completely reserved—now can’t serve legitimate clients
ICMP: Internet Control Message Protocol

- **ICMP**: pings to determine whether a system is connected to the Internet
  - Analogous to “Hello, are you still there?”

Client: ECHO REQUEST

Server: ECHO REPLY
ICMP Smurf Attacks

- Attack: ???

![Diagram](image-url)
ICMP Smurf Attacks

- **Attack**: takes advantage of broadcast-enabled hosts to amplify attack
- Attacker spams **spoofed-source** ICMP requests, reflected to victim’s IP

![Diagram showing an attacker sending spoofed ICMP requests to an Amplifier, which amplifies the requests and reflects them back to the victim's IP.](image_url)
ICMP Ping of Death Attack

- **Internet Protocol:** IPV4 packets should be **less than 65,536 bytes**
  - Packets can be sent in **fragments** and **reassembled** by receiver

- **Attack:** ???

```
<table>
<thead>
<tr>
<th>IP Header</th>
<th>ICMP Header</th>
<th>ICMP Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 bytes</td>
<td>8 bytes</td>
<td>65,508 bytes</td>
</tr>
</tbody>
</table>
```
ICMP Ping of Death Attack

- **Internet Protocol:** IPv4 packets should be less than **65,536 bytes**
  - Packets can be sent in **fragments** and **reassembled** by receiver

- **Attack:** send packet in fragments that **reassemble** to **64K+ bytes**
  - Many historical computer systems **could not handle larger packets**

- **Result:** crash by **buffer overflow**
  - Can’t serve clients until restart!
ARP: Address Resolution Protocol

- **ARP**: query to **resolve the MAC address** given a desired host IP
  - How we know which **physical** address to transmit data to from its logical address

**Client:** REQUEST MAC for IP 192.168.1.1

**Server:** SENDING MAC 00:B0:D0:63:C2:26
ARP Flooding Attack

- **Attack:** ???
ARP Flooding Attack

- **Attack**: same idea as **SYN flood**; **spoof source to victim** and spam away!
  - Victim gets overwhelmed by ARP replies and bandwidth crashes
**Physical Layer DoS**

**Russian Spy Submarines Are Tampering with Undersea Cables That Make the Internet Work. Should We Be Worried?**

A massive cable attack is probably an over-hyped scenario, at least for a country with as many redundant cables as the United States pitted against a limited number of Russian special-operations submarines.

**CNN Exclusive: FBI investigation determined Chinese-made Huawei equipment could disrupt US nuclear arsenal communications**
Thwarting DoS/DDoS Attacks

- How?
Thwarting DoS/DDoS Attacks

- **Limit connection rate**
  - Reduce to $N$ total requests

- **Detect anomalous activity**
  - IP geo-filtering
  - Packet similarity detection

- **Avoid holding connection state**
  - Don’t wait on “half-open” connections

- **Don’t be part of the problem!**
  - Disable potential amplifiers
  - Prevent botnet infection
Questions?
This time on CS 4440...

Authentication
Multiple Authentication Factors
One-time PINs
Secure Password Storage
What is authentication?

- What is it?
What is authentication?

What is it?
- That password you re-use for every website
- An ever-changing set of rules to frustrate you
- The most annoying thing about attending UofU

Sorry, but your password must contain an uppercase letter, a number, a hieroglyph, a feather from a hawk and the blood of a unicorn.

Why just 12 hours!?!?
What is authentication?

- **Goal:** ???
What is authentication?

- **Goal:** establish trust in the identity of another communicating party
- **Problem:** ???
What is authentication?

- **Goal**: establish trust in the identity of another communicating party

- **Problem**: cannot directly interact with them to verify their identity
  - Must be performed **remotely**

- **Challenge**: how can someone prove they are who they say they are?
The Three Factors of Authentication

- Something you ???
- Something you ???
- Something you ???
The Three Factors of Authentication

- **Something you have**
  - Smartphone
  - Laptop
  - Email account

- **Something you are**
  - Your fingerprint
  - Your DNA
  - Your iris, retina

- **Something you know**
  - Account password, banking PIN number
  - Nuclear strike challenge-response code
Single- vs. Multi-factor Authentication

- **N-factor authentication**: how many factors are used to authenticate
  - Password-only login is a single-factor authentication

- What are the **trade-offs**?
  - ???
Single- vs. Multi-factor Authentication

- **N-factor authentication**: how many factors are used to authenticate
  - **Password-only login** is a single-factor authentication

What are the **trade-offs**?

- **Fewer** factors = **worse** security
  - Compromise of one factor is total authentication violation
- **More** factors = **increased** security
  - To fully violate authentication, attacker must compromise all
- **Trade-off**: more annoying for user
  - Who cares? **Security >> UX**

Nowadays, most authentication is **at least 2-factor**
Questions?
One-time PINs
How can you prove—remotely—that you possess something?

**Proof of possession:** make the user perform some **object-specific action** that requires their **physical interaction**
One-time PINs

- **One-time PINs / Passwords:**
  - Password valid for only **one** login session or transaction

- **Delivering One-time PINs:**
  - ???

Stefan Nagy
One-time PINs

- **One-time PINs / Passwords:**
  - Password valid for only **one** login session or transaction

- **Delivering One-time PINs:**
  - **SMS**
    - Phone call
    - Text message
  - **Hardware**
    - Yubico YubiKey
    - RSA SecureID
  - **Application**
    - DUO Mobile
    - Google authenticator
Implementing OTPs

- **Idea:** call an API (e.g., `math.random`), send `random` to user, user re-enters it

---

**random** — Generate pseudo-random numbers

*Source code: Lib/random.py*

This module implements pseudo-random number generators for various distributions.

For integers, there is uniform selection from a range. For sequences, there is uniform selection of a random element, a function to generate a random permutation of a list in-place, and a function for random sampling without replacement.
Implementing OTPs

- **Idea:** call an API (e.g., `math.random`), send `random` to user, user re-enters it

- **Authentication offline? No!**
  - User needs internet to receive the OTP code
  - Without a connection, they can’t authenticate

- **Demonstrably secure? No!**
  - Most “random” APIs have small/predictable seeds
  - Also vulnerable to man-in-the-middle attacks

---

**random** — Generate pseudo-random numbers

Source code: `Lib/random.py`

**Warning:** The pseudo-random generators of this module should not be used for security purposes. For security or cryptographic uses, see the `secrets` module.

For integers, there is uniform selection from a range. For sequences, there is uniform selection of a random element, a function to generate a random permutation of a list in-place, and a function for random sampling without replacement.
**SIM: Subscriber Identity Module**
- A small card inserted into your phone
- Connects you to your carrier’s network
**Attack: SIM Swap**

- **SIM: Subscriber Identity Module**
  - A small card inserted into your phone
  - Connects you to your carrier’s network

- **Social engineering attack:**
  - Learn key info about victim. E.g.:
    - Mothers’ maiden name
    - Childhood street address
  - Trick carrier to issue new SIM card
    - “I’m Jeff Bezos, my phone broke!”
    - Attacker “appears to be” victim
**Attack: SIM Swap**

- **SIM: Subscriber Identity Module**
  - A small card inserted into your phone
  - Connects you to your carrier’s network

- **Social engineering attack:**
  - Learn key info about victim. E.g.:
    - Mothers’ maiden name
    - Childhood street address
  - Trick carrier to issue new SIM card
    - “I’m Jeff Bezos, my phone broke!”
    - Attacker “appears to be” victim

- **Result:** attacker is **man-in-the-middle**
  - Receives any OTPs transmitted by **SMS**!
Implementing OTPs

- **Better idea:** independently generate OTP codes based on a **moving factor**
  - E.g., intervals of **time**, unique session **count**, etc.

![Diagram showing the process of generating and verifying OTPs using HMAC(K, F) mod 10^D.]
Implementing OTPs

- **Better idea:** independently generate OTP codes based on a moving factor
  - E.g., intervals of time, unique session count, etc.

**Benefit:** higher entropy; and the client’s responses can be computed offline
Implementing OTPs

- **Better idea:** independently generate OTP codes based on a *moving factor*
  - E.g., intervals of *time*, unique session *count*, etc.

- **Common OTP protocols:**
  - HMAC-based OTP (*HOTP*)
    - Use *session count* as factor
  - Time-based OTP (*TOTP*)
    - Use *time interval* as factor
Implementing OTPs

- **Better idea:** independently generate OTP codes based on a moving factor
  - E.g., intervals of time, unique session count, etc.

- **Common OTP protocols:**
  - HMAC-based OTP (**HOTP**)
    - Use session count as factor
  - Time-based OTP (**TOTP**)
    - Use time interval as factor

- **Problem:** desynchronization
  - E.g., user hits “login” one too many times
Implementing OTPs

- **Better idea:** independently generate OTP codes based on a moving factor
  - E.g., intervals of time, unique session count, etc.

- **Common OTP protocols:**
  - HMAC-based OTP (**HOTP**)
    - Use session count as factor
  - Time-based OTP (**TOTP**)
    - Use time interval as factor

- **Problem:** desynchronization
  - E.g., user hits “login” one too many times
  - **Solution:** make a few OTPs; user matches once
Biometrics

- Provides proof of ???
Biometrics

- Provides proof of **physical identity**
Biometrics

- Provides proof of **physical identity**

- **Something unique to you** (hopefully)
  - Fingerprint, iris, retina, DNA

- **Security = unlikely match probability**
  - Fingerprint match chance: ???
  - Iris pattern match chance: ???
Biometrics

- Provides proof of **physical identity**
- **Something unique to you** (hopefully)
  - Fingerprint, iris, retina, DNA
- Security = **unlikely match probability**
  - Fingerprint match chance: $1 \text{ in } 64 \times 10^{13}$
  - Iris pattern match chance: $1 \text{ in } 10^{78}$
- **Trade-offs?**
  - ???
Biometrics

- Provides proof of **physical identity**

- **Something unique to you** (hopefully)
  - Fingerprint, iris, retina, DNA

- **Security = unlikely match probability**
  - Fingerprint match chance: \(1 \text{ in } 64 \times 10^{13}\)
  - Iris pattern match chance: \(1 \text{ in } 10^{78}\)

- **Trade-offs?**
  - Engineering effort, storage size, privacy concerns
Biometric Challenges

Downsides?
Biometric Challenges

- **Replay attacks**
  - Spoofs an enrolled user

- **Poisoning attacks**
  - Alter enrollment template
  - Alter one user’s enrollment

- **Noisy sensors**
  - Gives attackers “leeway” in crafting adversarial inputs

- **Change / loss of biometric**
  - **Change:** cataracts surgery
  - **Loss:** losing your finger

After an initial analysis, the Indian and American scientists used three iris sensors and two commercial iris biometric matchers to check if the new irises passed biometric authentication. They found that the iris sensors’ success rate dropped to 75% after surgery. The biometric matchers did better authenticating 93% of the irises.

Crane horror *Reg* reader uses his severed finger to unlock Samsung Galaxy phone

On the other hand he was fine
Questions?
Passwords
Passwords

- Something that you ???

Login

Caution: Before entering your uNID or password, verify that the address in the URL bar of your browser is directing you to a University of Utah web site.

Important security information: This login uses cookies to provide access to the site you requested and to other protected University of Utah websites. For your security, log out of the services you are using and exit your browser when you have finished your session. Some browsers, including Google Chrome, retain cookie information by default even after you close your browser. Review your browser’s support documentation to set your browser to clear cookies automatically upon exit. Instructions for Google Chrome.
Passwords

- **Something that you know**
  - Something that you forget?

- **A secret** string of data that confirms a user’s identity
Passwords

- **Something that you know**
  - Something that you forget?

- A **secret** string of data that confirms a user’s identity
  - **Letters** (ABCDEFGH)
  - **Digits** (0123456789)
  - **Other symbols** ($#%- _ !)

---

**Login**

- uNID: (e.g. u8675309)
- Password:

**Caution:** Before entering your uNID or password, verify that the address in the URL bar of your browser is directing you to a University of Utah web site.

**Important security information:** This login uses cookies to provide access to the site you requested and to other protected University of Utah websites. For your security, log out of the services you are using and exit your browser when you have finished your session. Some browsers, including Google Chrome, retain cookie information by default even after you close your browser. Review your browser’s support documentation to set your browser to clear cookies automatically upon exit. [Instructions for Google Chrome.](#)
Passwords

- **Something that you know**
  - Something that you forget?

- **A secret** string of data that confirms a user’s identity
  - **Letters** (ABCDEFGH)
  - **Digits** (0123456789)
  - **Other symbols** ($#%-_!)

- **Cryptographically secure?**
Passwords

- **Something that you know**
  - Something that you forget?

- A **secret** string of data that confirms a user’s identity
  - **Letters** (ABCDEFGHIJKLMNOPQRSTUVWXYZ)
  - **Digits** (0123456789)
  - **Other symbols** ($#%-_!)

- **Cryptographically secure?**
  - **Not at all!**

---

**Login**

- **uNID:** (e.g. u6875209)
- **Password:**

Caution: Before entering your uNID or password, verify that the address in the URL bar of your browser is directing you to a University of Utah web site.

Important security information: This login uses cookies to provide access to the site you requested and to other protected University of Utah websites. For your security, log out of the services you are using and exit your browser when you have finished your session. Some browsers, including Google Chrome, retain cookie information by default even after you close your browser. Review your browser’s support documentation to set your browser to clear cookies automatically upon exit. Instructions for Google Chrome.
Why aren’t passwords cryptographically secure?

- Cryptographically Secure = ???
Why aren’t passwords cryptographically secure?

- **Cryptographically Secure** = unbiased output, cannot be *predicted*
  - E.g., a cryptographically-secure pseudo-random number generator
Why aren’t passwords cryptographically secure?

- Are most passwords biased or predictable?
  - Analysis of Sony and Gawker breached passwords:

Patterns across all passwords
Why aren’t passwords cryptographically secure?

- **Are most passwords biased or predictable?**
  - Analysis of Sony and Gawker breached passwords:

  Patterns across **all** passwords

  Passwords derived from **people names**
Why aren’t passwords cryptographically secure?

- **Are most passwords biased or predictable?**
  - Analysis of Sony and Gawker breached passwords:

Patterns across all passwords

- Person name: 14%
- Place name: 8%
- Dictionary word: 8%
- Number: 7%
- Double word: 5%
- In email: 3%
- Short phrase: 2%
- Keyboard pattern: 2%
- Related to site: 1%
- No pattern: 1%

Passwords derived from location names

- Direct: 34%
- Numbers: 25%
- Symbols: 16%
- Reverse: 8%
Why aren’t passwords cryptographically secure?

- **Are most passwords biased or predictable?**
  - Analysis of Sony and Gawker breached passwords:

Patterns across all passwords

Passwords derived from **dictionary words**
Why aren’t passwords cryptographically secure?

- **Are most passwords biased or predictable?**
  - Analysis of Sony and Gawker breached passwords:

  **Patterns across all passwords**

  **Passwords derived from numbers**
Why aren’t passwords cryptographically secure?

- **Are most passwords biased or predictable?**
  - Analysis of Sony and Gawker breached passwords:
Why aren’t passwords cryptographically secure?

- Are most passwords biased or predictable?
  - Analysis of Sony and Gawker breached passwords:

Patterns across all passwords

Passwords derived from pop culture references

- thx1138
- gundam
- ncc1701
Known **default** passwords:
- Device manufacturers don’t care
- E.g., password, 12345, etc.
- How Mirai Botnet spread itself

<table>
<thead>
<tr>
<th>Username</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>666666</td>
<td>666666</td>
</tr>
<tr>
<td>888888</td>
<td>888888</td>
</tr>
<tr>
<td>admin</td>
<td>(none)</td>
</tr>
<tr>
<td>admin</td>
<td>1111</td>
</tr>
<tr>
<td>admin</td>
<td>1111111</td>
</tr>
<tr>
<td>admin</td>
<td>1234</td>
</tr>
<tr>
<td>admin</td>
<td>12345</td>
</tr>
<tr>
<td>admin</td>
<td>123456</td>
</tr>
<tr>
<td>admin</td>
<td>54321</td>
</tr>
<tr>
<td>admin</td>
<td>7ujMko0admin</td>
</tr>
<tr>
<td>admin</td>
<td>admin</td>
</tr>
</tbody>
</table>
Attack: Guessing Passwords

- Known **default** passwords:
  - Device manufacturers don’t care
  - E.g., password, 12345, etc.
  - How Mirai Botnet spread itself

- **Social engineering** attacks:
  - Trick victim to revealing key info
    - E.g., date of birth, nickname
    - pet’s name, favorite team
  - Try to guess their password
    - E.g., GoChiefs94, Chiefs1994

<table>
<thead>
<tr>
<th>Username</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>666666</td>
<td>666666</td>
</tr>
<tr>
<td>888888</td>
<td>888888</td>
</tr>
<tr>
<td>admin</td>
<td>(none)</td>
</tr>
<tr>
<td>admin</td>
<td>1111</td>
</tr>
<tr>
<td>admin</td>
<td>1111111</td>
</tr>
<tr>
<td>admin</td>
<td>1234</td>
</tr>
<tr>
<td>admin</td>
<td>12345</td>
</tr>
<tr>
<td>admin</td>
<td>123456</td>
</tr>
<tr>
<td>admin</td>
<td>54321</td>
</tr>
<tr>
<td>admin</td>
<td>7ujMko0admin</td>
</tr>
<tr>
<td>admin</td>
<td>admin</td>
</tr>
</tbody>
</table>
Server-side Password Storage

- Passwords stored server-side in a **database**

Why is storing passwords in **plaintext** problematic?
Passwords stored server-side in a database

Client: Register

Server: Store

Attacker: Login

If database breached, attacker has all passwords!
Server-side Password Storage

- Passwords stored server-side in a **database**
  - Increase security by only storing **hashed passwords**

If database **breached**, attacker has **zero plaintext passwords**!

### Password Database

<table>
<thead>
<tr>
<th>user</th>
<th>password</th>
</tr>
</thead>
<tbody>
<tr>
<td>bart</td>
<td>cowabunga</td>
</tr>
<tr>
<td>marge</td>
<td>p4$$w0rd</td>
</tr>
<tr>
<td>homer</td>
<td>donuts</td>
</tr>
</tbody>
</table>

### Hashed Password Database

<table>
<thead>
<tr>
<th>user</th>
<th>hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>bart</td>
<td>f0baf06…</td>
</tr>
<tr>
<td>marge</td>
<td>b3ea222…</td>
</tr>
<tr>
<td>homer</td>
<td>6c493f3…</td>
</tr>
</tbody>
</table>
Attacking Stored Passwords

- **Assumption**: attacker has **full access** to our database of **hashed passwords**
  - E.g., SQL injection, other web app attacks
Attacking Stored Passwords

- **Assumption:** attacker has **full access** to our database of **hashed passwords**
  - E.g., SQL injection, other web app attacks

- What if a **weak** hash function is used?
  - ???
**Attacking Stored Passwords**

- **Assumption:** attacker has **full access** to our database of **hashed passwords**
  - E.g., SQL injection, other web app attacks

- **What if a weak hash function is used?**
  - **Pre-image attacks:** find the original string
  - **Collision attacks:** find a different string that produces same hash as password
Attacking Stored Passwords

- **Assumption:** attacker has full access to our database of hashed passwords
  - E.g., SQL injection, other web app attacks

- What if a **weak** hash function is used?
  - **Pre-image attacks:** find the original string
  - **Collision attacks:** find a different string that produces same hash as password

- What if a **fast** hash function is used?
Attacking Stored Passwords

- **Assumption:** attacker has **full access** to our database of **hashed passwords**
  - E.g., SQL injection, other web app attacks

- What if a **weak** hash function is used?
  - **Pre-image attacks:** find the original string
  - **Collision attacks:** find a different string that produces same hash as password

- What if a **fast** hash function is used?
  - Attacker can **quickly pre-generate hashes** for all possible password possibilities

**Common Passwords**

<table>
<thead>
<tr>
<th>Password</th>
<th>SHA256(string)</th>
</tr>
</thead>
<tbody>
<tr>
<td>“password”</td>
<td>5e8848...</td>
</tr>
<tr>
<td>“123456”</td>
<td>8d969e...</td>
</tr>
<tr>
<td>“qwerty”</td>
<td>65e84b...</td>
</tr>
</tbody>
</table>

**Same hash function as the server**

**Attacker’s table of pw hash pairs**
Attack: Rainbow Tables

- Similar to a lookup table—attacker can trade-off disk space vs. CPU time
  - Attacker wants something that uses less time, less storage than a brute-force attack
Stefan Nagy

**Attack: Rainbow Tables**

- Similar to a **lookup table**—attacker can trade-off **disk space** vs. **CPU time**
  - Attacker wants something that uses **less time, less storage** than a **brute-force attack**

- **Idea:** iteratively **hash** and **reduce** to form a connected “**chain**” of hashes
  - **Simple reduction function:** truncate to just the first 10 characters of every hash

---

```
<table>
<thead>
<tr>
<th>&quot;password&quot;</th>
<th>Hash: 5e884898da 2804715...</th>
<th>Reduce: 5e884898da</th>
<th>Hash: 3a42beb21d d384f37...</th>
<th>Reduce: 3a42beb21d</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stored</td>
<td>Not stored</td>
<td>Stored</td>
<td>Not stored</td>
<td>Stored</td>
</tr>
</tbody>
</table>
```
To find a **password** from its hash, **perform reductions** and check for a match.

- For efficiency, only the **starting and ending links** are stored per each chain.

---

**Example:** Find password for hash **135addal**

\[
\begin{align*}
H, R (135addal) &= x1390214 \\
H, R (x1390214) &= 1p12iolk \quad \text{Found chain!}
\end{align*}
\]

Walk the chain from its start (**qwerty**) to find the original string:

\[
\text{qwerty} \rightarrow 1521ds11 \rightarrow 135addal
\]
Better Password Generation

- Why is reusing the **same password** bad practice?

Sorry, but your password must contain an uppercase letter, a number, a hieroglyph, a feather from a hawk and the blood of a unicorn.
Better Password Generation

- **Why is reusing the same password bad practice?**
  - If a breached server stores it in plaintext, your credentials are now stolen!

![Image showing a blackboard with 'I SHALL USE STRONG PASSWORDS. I SHALL USE STRONG PASSWORDS.']

Sorry, but your password must contain an uppercase letter, a number, a hieroglyph, a feather from a hawk and the blood of a unicorn.

![Image of a man holding his head in shock]
Better Server-side Password Storage

- **Slower** hash functions
  - ???
Better Server-side Password Storage

- **Slower hash functions**
  - Makes rainbow table generation **more computationally expensive** for attackers!
  - E.g., **Bcrypt, Scrypt**—perform multiple rounds of hashing (**much slower**)
Better Server-side Password Storage

- **Slower hash functions**
  - Makes rainbow table generation **more computationally expensive** for attackers!
  - E.g., *Bcrypt, Scrypt*—perform multiple rounds of hashing (**much slower**)

- **Salted passwords:**
  - Add **extra data** when generating hash
  - **Goal:** same input = different output
Better Server-side Password Storage

- **Slower hash functions**
  - Makes rainbow table generation **more computationally expensive** for attackers!
  - E.g., *bcrypt*, *scrypt*—perform multiple rounds of hashing (**much slower**)

- **Salted passwords:**
  - Add **extra data** when generating hash
  - **Goal:** same input = different output
  - Salting considerations:
    - Salt should **not be short**
    - Should be **unique** per user
Better Server-side Password Storage

- **Slower hash functions**
  - Makes rainbow table generation **more computationally expensive** for attackers!
  - E.g., **Bcrypt, Scrypt**—perform multiple rounds of hashing (**much slower**)

- **Salted passwords:**
  - Add **extra data** when generating hash
  - **Goal:** same input = different output
  - Salting considerations:
    - Salt should **not be short**
    - Should be **unique** per user

- **Better:** **salting + slow hashing**!
Assume attacker knows hash function and wants to **find a single password**
- Rapidly **becoming more doable** with advances in hardware!
Attack: Client-side Password Theft

- How?
Attack: Client-side Password Theft

How?
- Keyloggers, unencrypted transit, phishing, angry ex-partner
Forgetting and Recovering Passwords

- Security questions:
  - What’s your childhood pet?

- Password recovery email
  - Click here to reset your password!

- Send in plaintext to email
  - Your password is “in$3cur3”

Good security?
Forgetting and Recovering Passwords

- Security questions:
  - What’s your childhood pet?

- Password recovery email
  - Click here to reset your password!

- Send in plaintext to email
  - Your password is “in$3cur3”

Bad security! Attacker might have control of the victim’s email!
Forgetting and Recovering Passwords

- Security questions:
  - What’s your childhood pet?

- Password recovery email
  - Click here to reset your password!

- Send in plaintext to email
  - Your password is “in$3cur3”

- Other approaches:
  - Phone call
  - Session-specific PIN

**Bad security!** Attacker might have control of the victim’s email!

**Trade-offs?**
Questions?
Next time on CS 4440...

Tor: The Onion Router
Project 4 Tips