CS 370: Introduction to Security

05.11: Advanced web security

Tu/Th 4:00 – 5:50 PM

Sanghyun Hong

sanghyun.hong@oregonstate.edu

Oregon State University

SAIL

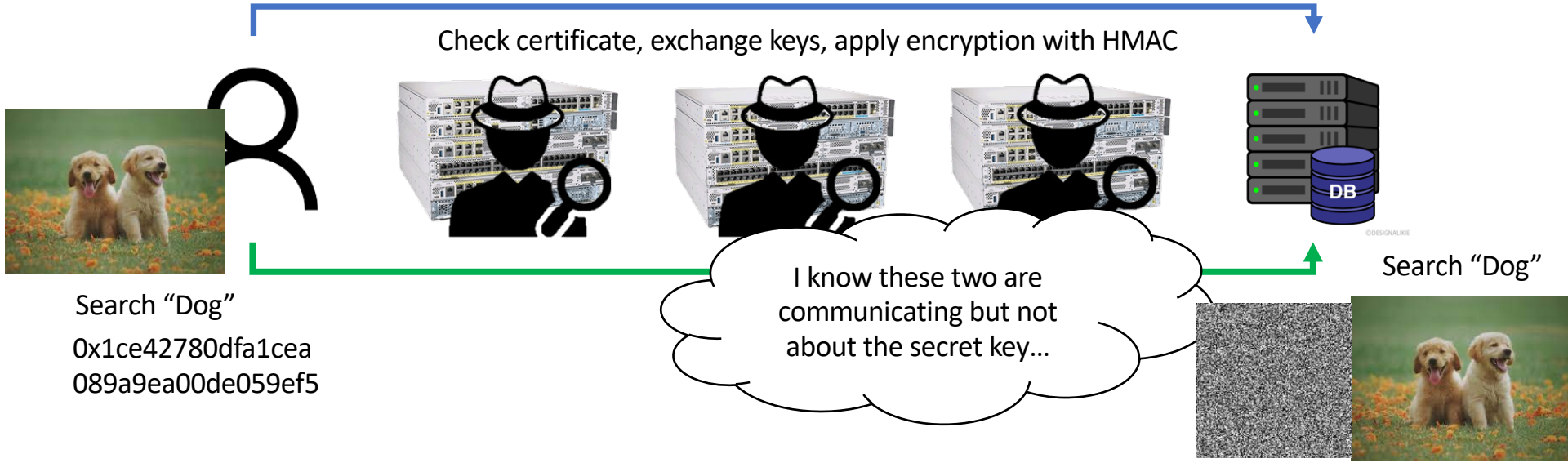
Secure AI Systems Lab

Topics for today

- Recap: TLS
- Recap: web security
  - Password (authentication)
  - Dictionary attack
  - SQL injection attack
- (Advanced) web security
  - Same-origin
  - Cookies
  - CSRF attacks

Recap: the Internet with a secure mechanism (SSL/TLS)

Middle mans never know
DH exchange keys!!



The Middlemen Will Only See the Encrypted Contents
They Will Never Know the Secret Key …

Recap: SSL/TLS Handshaking

Client (You**)**

- 1. Send ‘client hello’
  - Version
  - Random number
  - Cipher suites available

Server (oregonstate.edu**)**

- 2. Send ‘server hello’
  - Version
  - Random number
  - Cipher suites chosen
- 3. Send ‘server certificate’
  - Full chain of digital certificates

Recap: SSL/TLS Handshaking

Client (You**)**

- 1. Send ‘client hello’

Server (oregonstate.edu**)**

- 2. Send ‘server hello’
- 3. Send ‘server certificate’
- 4. Server key exchange
  - Send ECDHE public values
  - Signed by the server’s private key
- 5. ‘server hello’ done

Recap: SSL/TLS Handshaking

Client (You**)**

- 1. Send ‘client hello’
- 6. Client key exchange
  - Send ECDHE public values (client)

Server (oregonstate.edu**)**

- 2. Send ‘server hello’
- 3. Send ‘server certificate’
- 4. Server key exchange
  - Send ECDHE public values
  - Signed by the server’s private key
- 5. ‘server hello’ done

Recap: SSL/TLS Handshaking

Client (You**)**

- 1. Send ‘client hello’
- 6. Client key exchange
- 7. Change cipher spec
- 8. Handshake message (encrypted)

Server (oregonstate.edu**)**

- 2. Send ‘server hello’
- 3. Send ‘server certificate’
- 4. Server key exchange
- 5. ‘server hello’ done
- 9. Change cipher spec
- 10. Handshake message (encrypted)

Now, We Can Start Communicating with Encrypted MSG!

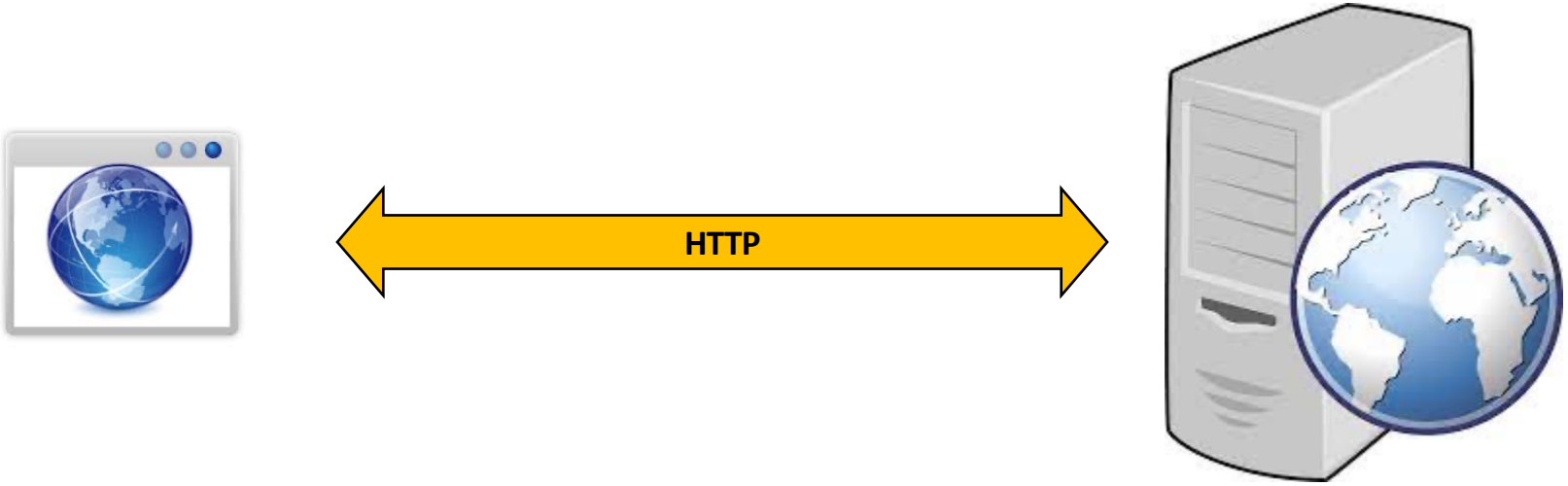
Recap: SSL/TLS Handshaking

- Send/receive application data
  - Both client and server will send encrypted data
  - [encrypted data] [MAC]
    - Server: server_write_key and server_write_mac_key
    - Client : client_write_key and client_write_mac_key

```
To generate the key material, compute

   key_block = PRF(SecurityParameters.master_secret,
                   "key expansion",
                   SecurityParameters.server_random +
                   SecurityParameters.client_random);

until enough output has been generated.  Then, the key_block is
partitioned as follows:

   client_write_MAC_key[SecurityParameters.mac_key_length]
   server_write_MAC_key[SecurityParameters.mac_key_length]
   client_write_key[SecurityParameters.enc_key_length]
   server_write_key[SecurityParameters.enc_key_length]
   client_write_IV[SecurityParameters.fixed_iv_length]
   server_write_IV[SecurityParameters.fixed_iv_length]
```

EXAMPLE: A WEB SERVER

- Suppose we talk to a webserver (HTTP)



Example: a web server

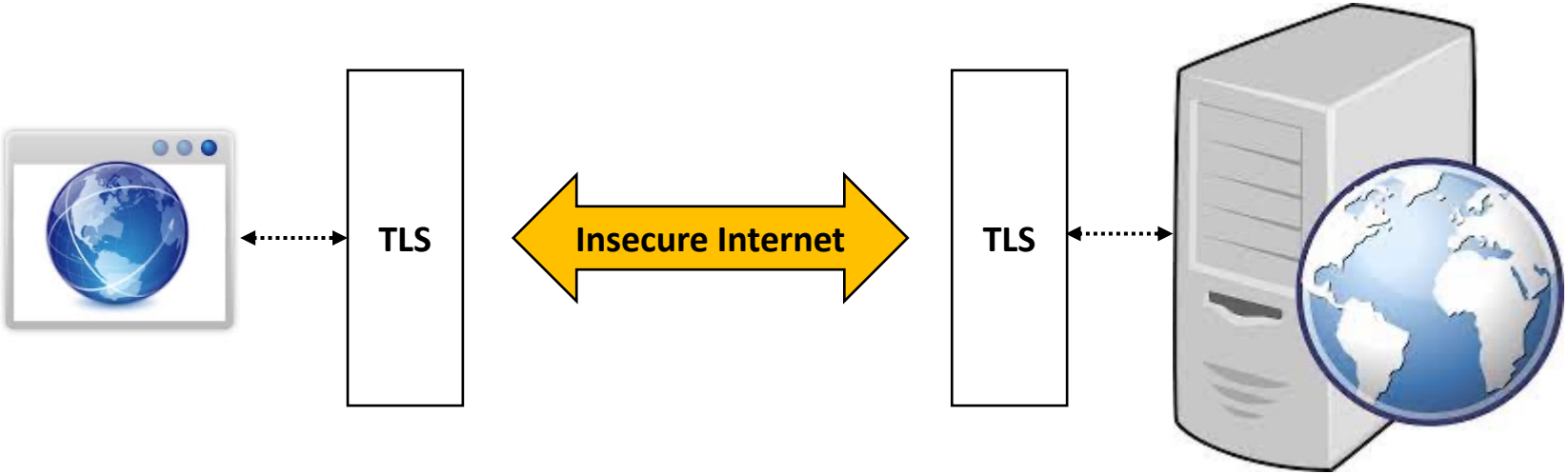
- Suppose we talk to a webserver (HTTP)

```
GET / HTTP/1.0
```


```
HTTP/1.0 200 OK
Date: Tue, 25 Oct 2022 12:53:12 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO
P3P: CP="This is not a P3P policy! S
Server: gws
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
```

EXAMPLE: A WEB SERVER

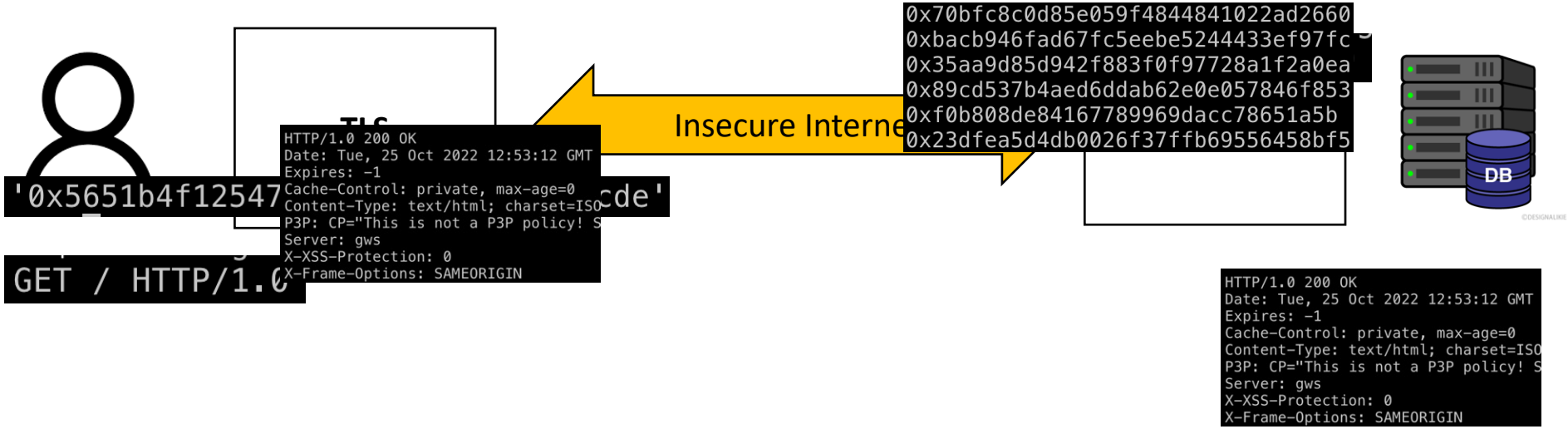
- Suppose we use HTTPs (instead of HTTP)



Example: a web server

Run **TLS handshake** to establish a secure channel

A Web Server Example



How can we use TLS?

- Many libraries are available
  - OpenSSL
  - libsodium
  - bouncycastle
  - SSL/TLS support in many other languages (Python, etc.)

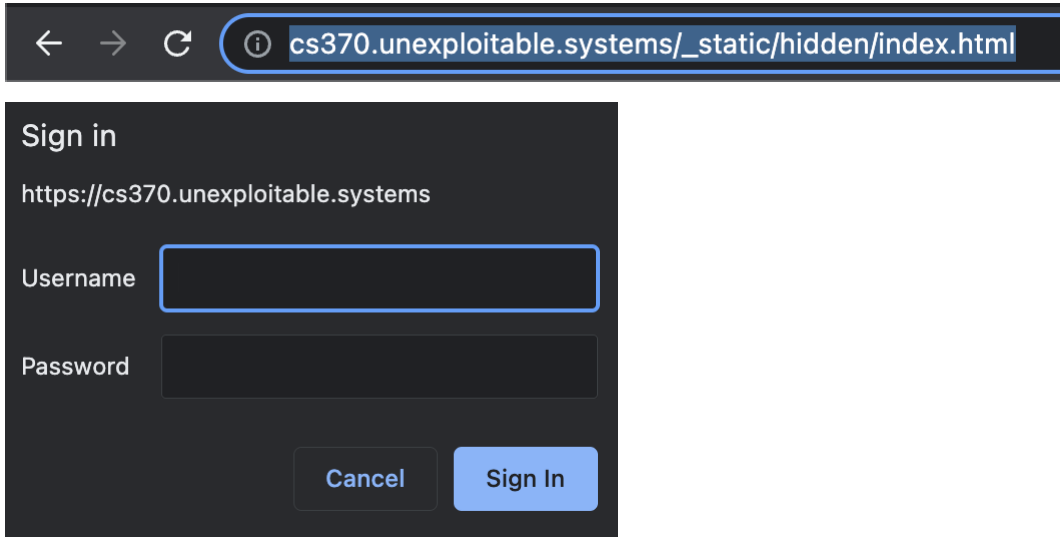
- You can “wrap” with them
  - We can easily convert non-TLS servers/clients into TLS servers/clients

Topics for today

- Recap: TLS
- Recap: web security
  - Password (authentication)
  - Dictionary attack
  - SQL injection attack
- (Advanced) web security
  - Same-origin
  - Cookies
  - CSRF attacks

How can we do access control on the web?

- "HTTP" basic authentication
  - A simple challenge and response mechanism
  - A server can request authentication information (ID and Password) from a client

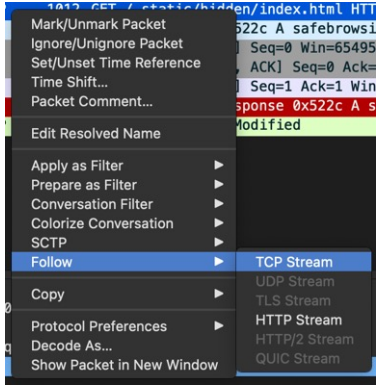


IT IS INSECURE!

- "HTTP" basic authentication
  - A simple challenge and response mechanism
  - A server can request authentication information (ID and Password) from a client

1	0.000000000	127.0.0.1	127.0.0.1	HTTP	1012	GET /_static/hidden/index.html HTTP/1.1
2	0.001963698	127.0.0.1	127.0.0.53	DNS	83	Standard query 0x522c A safebrowsing.google.com
3	0.002251748	127.0.0.1	127.0.0.1	TCP	74	53732 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1993954412 TSec…
4	0.002275826	127.0.0.1	127.0.0.1	TCP	74	8080 → 53732 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=1993…
5	0.002306468	127.0.0.1	127.0.0.1	TCP	66	53732 → 8080 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1993954413 TSecr=1993954413
6	0.017663091	127.0.0.53	127.0.0.1	DNS	118	Standard query response 0x522c A safebrowsing.google.com CNAME sb.l.google.com A 142…
7	0.025120028	127.0.0.1	127.0.0.1	HTTP	254	HTTP/1.1 304 Not Modified

- Monitor the stream:




```
GET /_static/hidden/index.html HTTP/1.1
Host: cs370.unexploitable.systems:8080
Connection: keep-alive
Cache-Control: max-age=0
Authorization: Basic Ymx1ZTkwNTc6Y3MzNzB7QjRzSWNfQXVUaF9JNV9OMHRfczNDdVIzfQ==
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/106.0.0.0
Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/
*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: _jsuid=1158429791;
experimentation_subject_id=eyJfcmFpbHMiOnsibWVzc2FnZSI6Iklqa3paak01WVdZekxUYzBOV0V0TkdJMU5pMDVZelpsTFRVd
05HUm1abVExTlRKak9DST0iLCJleHAiOm51bGwsInB1ciI6ImNvb2tpZS5leHBlcmltZW50YXRpb25fc3ViamVjdF9pZCJ9fQ%3D%3D-
-14df51e13094f383b80e4b21ff0c195dd82560ed; _jsuid=1158429791
If-None-Match: W/"6360b363-25"
If-Modified-Since: Tue, 01 Nov 2022 05:49:23 GMT

HTTP/1.1 304 Not Modified
Server: nginx/1.14.0 (Ubuntu)
Date: Tue, 01 Nov 2022 06:01:09 GMT
Last-Modified: Tue, 01 Nov 2022 05:49:23 GMT
Connection: keep-alive
ETag: "6360b363-25"
```

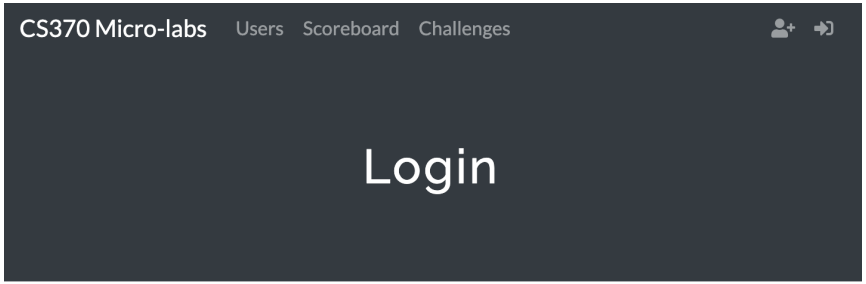
IT IS INSECURE!

- "HTTP" basic authentication
  - HTTP packets are unencrypted
  - base64Encode(username:password) is there …

- What about "HTTPS"?
  - We cannot use the HTTP basic authentication in HTTPs

SECURE AUTH.: DO NOT USE HTTP BASIC AUTH

- Let's use the login form



User Name or Email

Password

Forgot your password?

Submit

Secure auth.: do NOT store user passwords to servers

- So, any attacker who can access the server can see them

Home › Email Security

Bed Bath & Beyond Invest
After Employee Falls for P

By Eduard Kovacs on November 01, 2022

Share Tweet Recommend 0 RSS

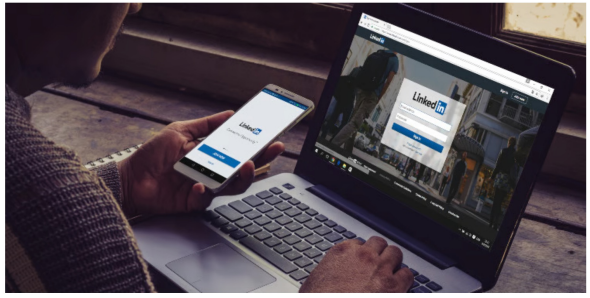
Bed Bath & Beyond revealed last week in an SEC filin
breach after an employee fell victim to a phishing at

This is not the first time Bed Bath & Beyond has disclosed
the retailer revealed that some customer accounts had b
hackers had obtained username and password combinatio
company and relied on the fact that
online accounts.

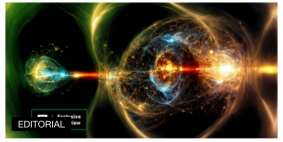
Scraped data of 500 million LinkedIn users being sold online, 2 million records leaked as proof

Updated on: 20 February 2023 9

Cybernews Team, Cybernews Team



Editor's choice



Quantum computing race explained: fast and furious

by Stefanie Schappert 05 May 2023

The World Economic Forum (WEF) published several think pieces this year describing a post-quantum computing world in which the global chasm between developed and underdeveloped populations only grows larger. But could the gloomy forecast be rosier than

Attackers put webservers on their radar; it can be **hacked**!
Passwords stored in the server **could also be leaked**

SECURE AUTH.: STORE HASHES!

- Hide the passwords from the server
  - Do not store the passwords directly
  - Do store SHA256("some_secret (salt)" + password)
  - Example:
    - SHA256("some_secret (salt)" + "my-super-secure-password!@#$11")
    - 59636881ab9bf34263cf3f4d90f25d2b91e74e8804b802d25c8f4bc5c80846ee

- Can an adversary reconstruct the password from the hash?
  - SHA256
    - One-way function
    - Many x exists that satisfies H(x) = y
    - SHA256('Hello, world') = SHA256('Something else')
  - Good luck!

Secure auth.: illustration of how hashing works

- Send ID/password but the server stores hash of the password

Secure auth.: how a server would process the log-in request?

- Once we submit the form
  - The server searches the database (e.g., SQL DB)
    - SELECT (username, password) FROM users WHERE username = ‘neuronoverflow’ and password = SHA256(secret + “my-super-secure-password!@#$11”)
  - Note:
    - The DB only stores the hash of the password
    - Suppose an adversary has access to the DB
    - They still need to compute the inverse to get the plaintext password

Topics for today

- Recap: TLS
- Recap: web security
  - Password (authentication)
  - Dictionary attack
  - SQL injection attack
- (Advanced) web security
  - Same-origin
  - Cookies
  - CSRF attacks

Dictionary attack

- The security guarantee assumes
  - We choose the password randomly!

- In reality
  - (12345678) Easy to memorize and type
  - (OregonBeaverRocks) Some phrases familiar
  - (Oregon1234) Add numbers on the phrase
  - (password1234!!) Add special characters at the end
  - …

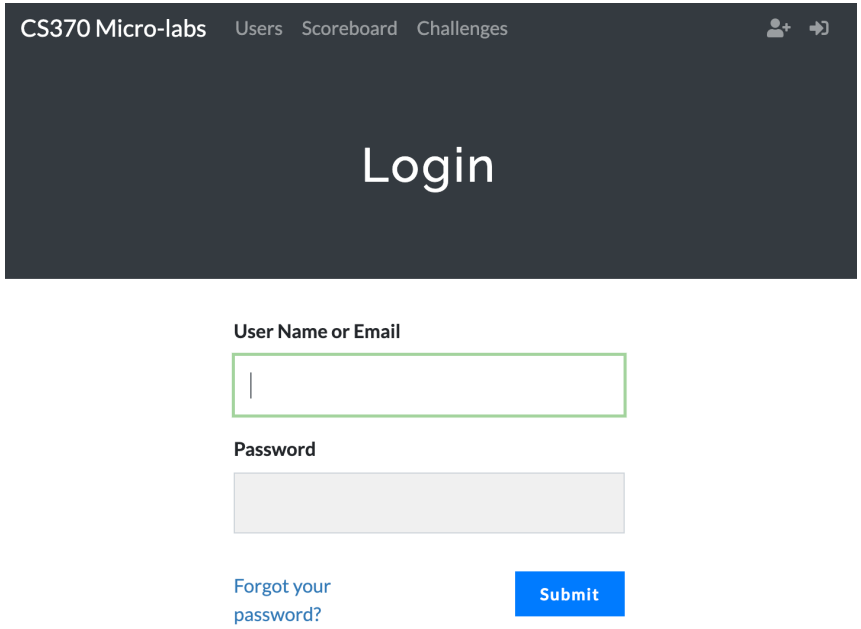
Dictionary attack

- Search space is significantly reduced
  - Suppose that the password is
    - 13 characters and consists of [A-Za-z0-9]
    - = 62^{13} possible combinations ($2.002854e^{23}$)
  - Suppose that
    - We know the password starts from ‘Portland’
    - = 62^{5} possible combinations ($9.1613283e^{8}$)
    - = 10^{15} smaller

SQL INJECTION

- Exploit the system's weakness
  - SELECT (username, password) FROM users WHERE username = 'neuronoverflow' and password = SHA256(secret + "my-super-secure-password!@#$11")

- SQL injection
  - We supply 'or 'a'='a as a password
  - SELECT (username, password) FROM users WHERE username = 'neuronoverflow' and password = '' or 'a' ='a'
  - THIS IS ALWAYS TRUE!!!



SQL INJECTION

- What if we supply ‘ union select (‘admin’, ‘a’) where ‘a’=‘a as a password?
  - SELECT (username, password) FROM users WHERE
  - username = ‘neuronoverflow’ and password = ‘’ union select (‘admin’, ‘a’) where ‘a’=‘a’

- You will have the admin
  - None for the first select statement
  - and the 2nd statement will query
    - Username = ‘admin’
    - Password = ‘a’
    - Always return true ‘a’ = ‘a’

Topics for today

- Recap: TLS
- Recap: web security
  - Password (authentication)
  - Dictionary attack
  - SQL injection attack
- (Advanced) web security
  - Same-origin
  - Cookies
  - CSRF attacks

Security principles

- Confidentiality
  - Malicious web sites should not be able to learn confidential information from our computers or other web sites
- Privacy
  - Malicious web sites should not be able to spy on us or our online activities
- Integrity
  - Malicious web sites should not be able to tamper with integrity of our computers or our information on other web sites
- Availability
  - Malicious parties should not be able to keep us from accessing our web resources

Security principles: confidentiality and integrity

- Security risk
  - Malicious web sites should not be able to tamper with our information or interactions on other websites
  - Example:
    - Suppose we visit “malware.com”
    - The attacker (who owns the website) should not be able to read our emails or buy things with our Amazon accounts

Confidentiality and integrity protection

- Same-origin policy
  - A rule that prevents one website from tampering with other *unrelated* websites
    - Enforced by the web browser
    - Prevents a malicious website from tampering with behavior on other websites
    - The key idea: webpages from the same site don't need to be isolated

Confidentiality and integrity protection

- Same-origin policy
  - A rule that prevents one website from tampering with other *unrelated* websites
    - Enforced by the web browser
    - Prevents a malicious website from tampering with behavior on other websites
    - The key idea: webpages from the same site don't need to be isolated
  - Every webpage has an origin defined by its URL with three parts:
    - Protocol: The protocol in the URL
    - Domain: The domain in the URL's location
    - Port: The port in the URL's location
      (If not specified, the default is 80 for HTTP and 443 for HTTPS)
    - Example:
      - https://computer.science.org/assets/photo.png (default: 443)
      - http://science.org:80/assets/new_photo.png

Confidentiality and integrity protection

- Same-origin policy
  - Two websites have the same origin *if and only if*
  - The protocol, domain, and port of the URL all match exactly

Domain I	Domain II	Same-origin?
https://cs.org	http://www.cs.org	
http://cs.org	https://cs.org	
http://cs.org:80	http://cs.org:8080	
https://cs.org/photo.png	https://cs.org/data/my.htm	

Confidentiality and integrity protection

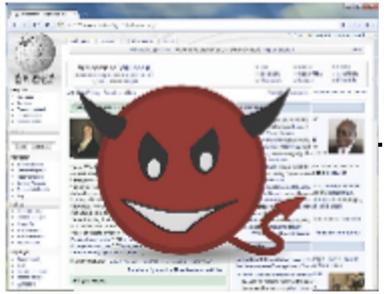
- Same-origin policy
  - Two websites have the same origin *if and only if*
  - The protocol, domain, and port of the URL all match exactly
  - Example:
    - **cs.org** embeds **google.com**
    - The inner frame cannot interact with the outer frame
    - The outer frame cannot interact with the inner one
  - Exception I:
    - JavaScript runs with the origin of the page that loads it
    - **cs.org** fetches JavaScript from **google.com**:
      - The JavaScript has the origin of **cs.org**
      - **cs.org** has “copy-pasted” JavaScript onto its webpage

Confidentiality and integrity protection

- Same-origin policy
  - Two websites have the same origin *if and only if*
  - The protocol, domain, and port of the URL all match exactly
  - Exception II:
    - Websites can fetch and display images/frames from other origins
    - But the website only knows about the image's size and dimensions (The website cannot manipulate the image)
    - The image and the frame has the origin of the page that it comes from

Confidentiality and integrity protection

- Same-origin policy
  - Two websites have the same origin *if and only if*
  - The protocol, domain, and port of the URL all match exactly
  - Exception III:
    - Websites can agree to allow some limited sharing
    - Cross-origin resource sharing (CORS)
    - Ex. the **postMessage** function in JavaScript
      - Receiving origin decides if to accept the message based on the origin
      - The correctness is enforced by the browser



postMessage(“run this script”, script)

Topics for today

- Recap: TLS
- Recap: web security
  - Password (authentication)
  - Dictionary attack
  - SQL injection attack
- (Advanced) web security
  - Same-origin
  - Cookies
  - CSRF attacks

Thank You!

Tu/Th 4:00 – 5:50 PM

Sanghyun Hong

sanghyun.hong@oregonstate.edu

Oregon State University

SAIL

Secure AI Systems Lab