

# Web Security

# Why is web security important?

- Yahoo Data Breach affects 500 million

*“The stolen information includes people’s names, email addresses, telephone numbers, birth dates, passwords (most hashed with bcrypt), and, in some cases, encrypted or unencrypted responses to security questions and answers”*

- Researcher demonstrates that XSS exposed eBay users to Phishing Attacks

*“With the graphics of the eBay login page copied, an attacker simply needed to add a PHP script so that the submitted data would be sent to the attacker’s server instead of eBay. Once the phishing page was ready, the attacker could have injected it as an iframe into the URL of the vulnerable eBay page.”*

- <http://www.securityweek.com/xss-flaw-exposed-ebay-users-phishing-attacks>

- <https://techcrunch.com/2016/09/22/yahoo-confirms-state-sponsored-attacker-stole-personal-data-of-at-least-500-million-users/>

# Security is hard!

## Many opportunities for attackers!

### Potential Targets:

- Client-Server Connection
  - Session hijacking
- Browsers
  - Code injections
- Servers
  - Code injections, DOS
- Users
  - Phishing, Ransomware

# Securing Communications

# Same Origin Policy

URL	Outcome
<code>http://store.company.com/dir2/other.html</code>	Success
<code>http://store.company.com/dir/inner/another.html</code>	Success
<code>https://store.company.com/secure.html</code>	Failure
<code>http://store.company.com:81/dir/etc.html</code>	Failure
<code>http://news.company.com/dir/other.html</code>	Failure

- Scripts on one web page can access data on a second page only if the both web pages have the same origin
- Origin determined by protocol, port, and hostname

# Cross-Origin Resource Sharing (CORS)

- Allows resources from one webpage to be requested by another from a different domain.
  - Stylesheets, JavaScript, etc
- Defines a standard browsers and servers can follow that is safer than just allowing all cross-origin requests.

```
var app = express();
var allowCrossDomain = function(req, res, next) {
  res.header("Access-Control-Allow-Origin", "*");
  res.header("Access-Control-Allow-Headers", "Content-Type, Accept");
  res.header("Access-Control-Allow-Methods", "POST, GET, PUT, DELETE, OPTIONS");
  next();
};
app.use(allowCrossDomain);
```

# Cross-Origin Resource Sharing (CORS)

The screenshot shows a web browser's developer tools with the Network tab selected. The address bar shows a request to `domain:api.github.com`. The Network tab displays a list of requests, with the first one selected. The selected request is an `OPTIONS` request to `https://api.github.com/_private/browser/stats`, which returned a `204 No Content` status. The response headers show `Access-Control-Allow-Headers`, `Access-Control-Allow-Methods`, and `Access-Control-Allow-Origin: *`.

Elements Console Sources Network Timeline Profiles Application Security

View: ☐ Preserve log ☒ Disable cache ☐ Offline No throttling

domain:api.github.com sch ☐ Regex ☐ Hide data URLs **All** XHR JS CSS Img Media Font Doc

2000 ms 4000 ms 6000 ms 8000 ms 10000 ms 12000 ms 14000 ms 16000 ms

Name × Headers Preview Response Timing

☐ stats

☐ stats

▼ General

**Request URL:** `https://api.github.com/_private/browser/stats`  
**Request Method:** `OPTIONS`  
**Status Code:** 204 No Content  
**Remote Address:** `192.30.253.117:443`

▼ Response Headers [view source](#)

**Access-Control-Allow-Headers:** `Authorization, Content-Type, If-Match, ding, X-GitHub-OTP, X-Requested-With`  
**Access-Control-Allow-Methods:** `GET, POST, PATCH, PUT, DELETE`  
**Access-Control-Allow-Origin:** `*`

# HTTP

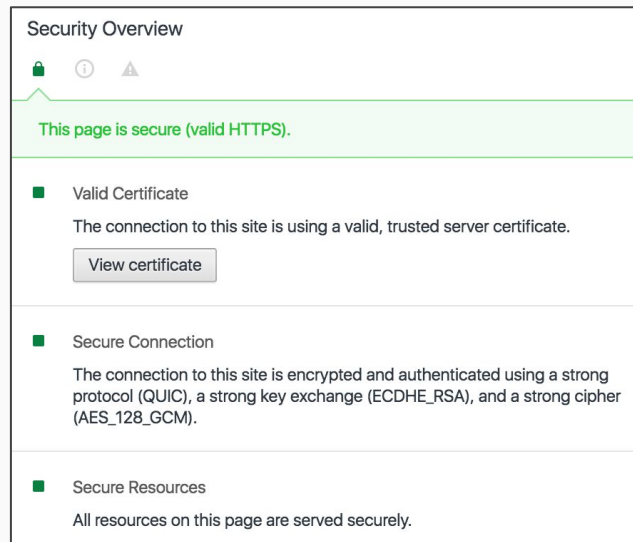


- Unencrypted messages between client and servers
- Vulnerable to attacks (man in the middle attacks, session hijacking, etc)



# HTTPS/TLS/SSL

- Transport Layer Security & Secure Socket Layer
- Another layer that messages pass through that allows for secure communication
- Messages are encrypted and cannot be read if captured
- Important Piece: SSL Certificates to verify you're connected to who they say they are.



# COMMON ATTACKS

# Code Injection

- Introduce code into vulnerable applications for malicious intent
  - SQL Injection
  - XSS (Cross-Site Scripting)

# SQL Injection

SQL statements are entered into text entry fields for execution on a server.

Query:

```
var sql = "SELECT * FROM users WHERE user = " + req.body.query;  
sendQuery(sql);
```

# Sanitize your Inputs



<https://gizmodo.com/>



<https://xkcd.com/327/>

# XSS

Malicious client-side scripts are injected into web pages viewed by others



The screenshot shows a web form titled "Llama Forum". It has two input fields: "Title" and "Text". Below the "Text" field is a green button labeled "Post".

```
router.post('/posts', function(req, res) {  
  var post = new Post()  
  post.title = req.body.title;  
  post.text = req.body.text  
  post.save(function(err) {  
    if ( err ) {  
      res.status(500).json({ message: 'Error occurred! '});  
    }  
    res.status(200).json({ message: 'success!', data: post });  
  });  
});
```

DEMO

# DDOS

Distributed  
Denial-of-Service attacks  
attempt to make web  
services unavailable by  
flooding them with  
requests



AWS Shield



**CLOUDFLARE**



# UI/UX and Security

Incorrect username or password. ✕

Username or email address

someemail@gmail.com


Password [Forgot password?](#)

Sign in

New to GitHub? [Create an account.](#)

<https://kev.inburke.com/kevin/invalid-username-or-password-useless/>

←



someemail@gmail.com

Password

Wrong password. Try again.

Sign in

☒ Stay signed in [Forgot password?](#)

<http://uxmag.com/articles/security-vs-design-standing-at-odds>

## Additional Resources

- UIUC Web Programming Security Resources
  - <https://github.com/uiuc-web-programming/web-security>
- Mozilla
  - <https://developer.mozilla.org/en-US/docs/Web/Security>

## Stay Informed

- OWASP (Open Web Application Security Project)
  - [https://www.owasp.org/index.php/Main\\_Page](https://www.owasp.org/index.php/Main_Page)
- ThreatPost (The Kaspersky Lab Security News)
  - <https://threatpost.com/category/web-security/>