WEB SECURITY

- Prevent fake logins
- Protect user data
- Protect access to the system
- Protect information on the system

CORE RULES OF WEB SECURITY

- Never trust user input
- You can never be more clever than all of them
- The Front End cannot add security, only convenience
- Your data IS of interest

XSS

Cross-Site scripting (XSS)

(Why "X"? English is weird, and CSS already has a meaning)

Running a client-side script that is NOT yours

SIMPLE XSS DEMO

```javascript
const express = require('express');
const app = express();
const PORT = 3000;

app.get('/', (req, res) => {
  const name = req.query.username;
  res.send(`<p>Hello ${name}</p>`);
});

app.listen(PORT, () => console.log(`Running`));
```

Seems reasonable enough, right?

HOW TO ABUSE WITH XSS

```javascript
app.get('/', (req, res) => {
  const name = req.query.username;
  res.send(`<p>Hello ${name}</p>`);
});
```

```
?username=%3Cimg+src=%27%27+onerror=alert(%27pwned%27)%22%3E
```

A user (not you!) can run JS on your page

If we save that data and show it to others (like username), a user can run JS on the pages of OTHER USERS

WHY IS XSS BAD?

- They can inject ads (incl. popups)
- They can scrape any data off the page and send it elsewhere
  - Including personal/private data
  - Including passwords
- They can alter any data on the page
- They can perform actions on the page
  - Enter data
  - Click buttons

HOW TO DEFEND AGAINST XSS

Rule #1: Never trust data from the user

- "whitelist" permitted data, block anything else
- Whitelisting isn't always practical, but should always be the first choice

Rule #2: Never assume the user isn't clever enough

- Attempts to "Blacklist" bad data eventually fail

Examples:

- (whitelist) Phone is only 0-9, parens, dot, and '-'
- (blacklist) Phone can't contain `*` or `<` or `>`

NEVER TRUST THE FRONT END

NEVER TRUST FRONT END JS TO ENFORCE SECURITY

- They can alter it, or even just not use a browser

Rule #3: **client-side JS provides convenience, not security. Security MUST be backend**

Rule #4:

- Your data IS of interest
    - Inject malware
    - Inject ads
    - Grab reused account info

SQL INJECTION

Just like XSS is inserting javascript into your HTML, SQL Injection is inserting SQL commands into your SQL.
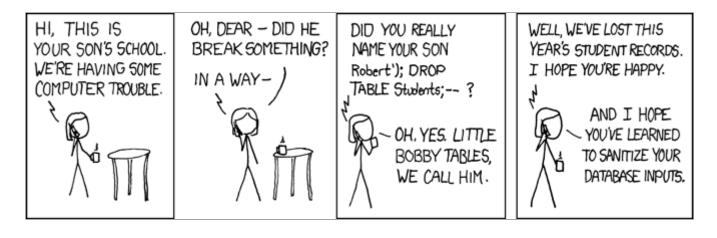
Consider:

- `SELECT age FROM people WHERE name = "${name}"`

What happens is all based on what `name` is:

- Send `Bao` and it works fine.
- Send `Bao"; DELETE * FROM people WHERE "1" = "1` and you just deleted everything

LITTLE BOBBY TABLES

XKCD (**http://xkcd.com**)

WHY IS SQL INJECTION BAD?

- They inject ads
- They inject scripts for XSS/XSRF
- They can delete your data
- They can copy your data
- They can encrypt your data (ransomware)
- They can alter your data (incl. theft)

DEFENSE AGAINST SQL INJECTION

- Never craft your SQL from user input
- Always use "bound" variables when possible
- If not possible to use bound, use the escaping libraries from your vendor
  - AND whitelist your data

POOR PASSWORD SECURITY

If someone can read your DB...

- Malicious employee
- Security Hole

...it is much better if they can't actually get passwords

HASHING

You never need to store the password directly.

You don't care about the password itself, all you need is proof the user KNOWS it.

When given a password, run it through a one-way hashing algorithm, store resulting hash.

To later confirm, when they give you the password you can again hash it and compare to the stored hash.

You never store the password.

SALTING

Hashing can pre-calculate options (a "rainbow table")

- Add a random "salt". Store the salt with the hash.
- user "bob" has password "123456"
- Hashed that might end up as "ip9awlnfhiorwijeqds"
- But let's pick a random salt of "ih7g57r"
- So we hash "ih7g57r-123456"
- That gives us "hhncdhluxhluxhlu3xl2"
- So we store "ih7g57r-hhncdhluxhluxhlu3xl2"
- Next time "bob" logs in, he gives us "123456"
- We hash the salt(from stored value)+password
- We compare the salt+hash to the stored value

DON'T DO LOGINS

My advice: Don't try to do logins

- cryptographically secure hashing algorithms?
- how get a large enough salt?
- these will remain secure as technology advances?

Your stuff may not be a big deal, the user's password IS

- They reuse it somewhere else more important

Use an alternative

- External Password (Google, Facebook, Github, etc)
- OIDC (Okta, Auth0, etc)

SUMMARY

- Never trust input
  - don't store it
  - don't display it
  - don't use it in string commands
- Web requests/responses are all visible to the user
  - and points in-between
- You will not be smarter than the bad people
  - You win by not giving them the chance to try
- No site is too small to be a target of the bad people

SECURITY RULES

- Rule #1: Never trust data from the user
- Rule #2: Never assume the user isn't clever enough
- Rule #3: Security MUST be server-side
- Rule #4: Your site WILL be a target