

# Secure Angular Programming

presented by Spencer Sundrud

A dark blue diagonal gradient bar that starts from the bottom left and extends towards the top right, covering the lower half of the slide.

# Overview

- In-built methods and features
- XSS
- Sanitization
- Error Messages
- Session Hijacking

# Assumptions

- Working knowledge of Angular
- Basic secure programming practices
- Scope of project is only on the front-end - no backend coding

# In-built methods and features

- Consistent updates - make sure to use the latest libraries available
- Automatic sanitation - does not trust user input, escapes untrusted values
- Very few vectors of attack

# XSS

Treat all templates as executable code! All inputs, including custom user names, should be sanitized before being used to render any element.

Don't break the system - use sanitizer libraries instead of disabling security features.

## **Best practice:**

Use [innerHTML] or {{ }} tags to remove scripting, etc.

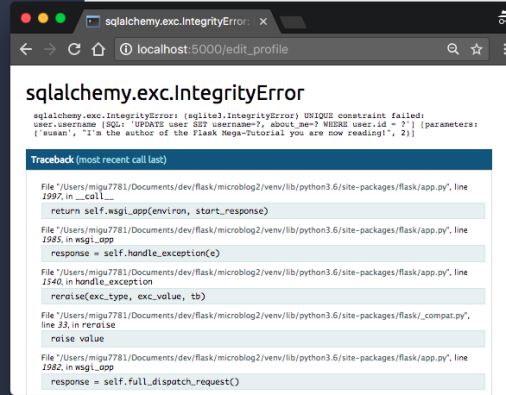
# Sanitation

As mentioned before, use things like DOMSanitizer APIs to sanitize data. Assume the backend code has horrible security, and never passed an untrusted string back to them.

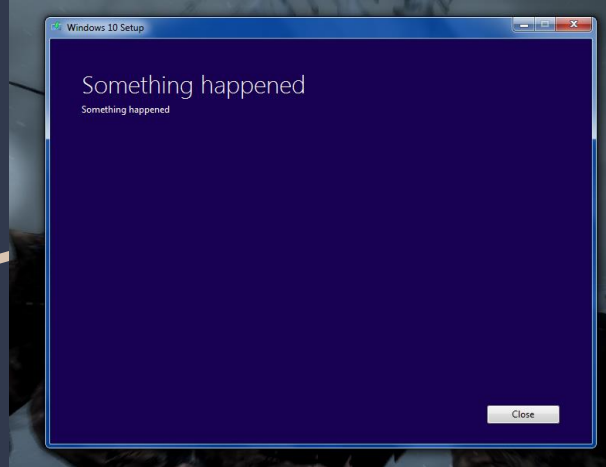
## **Best practice:**

Use the `bypassSecurity` function sparingly, if at all. It is literally a bypass to all inbuilt sanitation measures.

# Error Messages



The screenshot shows a web browser window with the address bar at `localhost:5000/edit_profile`. The page title is `sqlalchemy.exc.IntegrityError`. The main content area displays the error message: `sqlalchemy.exc.IntegrityError (sqlalchemy.exc.IntegrityError) UNIQUE constraint failed: user.username (SQL: 'UPDATE user SET username=?, about=?, weight user.id = ?' [parameters: {'username', 'I'm the author of the Flask Mega-Tutorial you are now reading!', 2}])`. Below the error message is a 'Traceback (most recent call last)' section with a blue header. The traceback lists several file paths and line numbers, including `File "/Users/migu7781/Documents/dev/flask/microblog2/venv/lib/python3.6/site-packages/flask/app.py", line 1997, in __call__`, `return self.wsgi_app(environ, start_response)`, `File "/Users/migu7781/Documents/dev/flask/microblog2/venv/lib/python3.6/site-packages/flask/app.py", line 1985, in wsgi_app`, `response = self.handle_exception(e)`, `File "/Users/migu7781/Documents/dev/flask/microblog2/venv/lib/python3.6/site-packages/flask/app.py", line 1940, in handle_exception`, `reraise(exc_type, exc_value, tb)`, `File "/Users/migu7781/Documents/dev/flask/microblog2/venv/lib/python3.6/site-packages/flask_compat.py", line 33, in reraise`, `raise value`, and `File "/Users/migu7781/Documents/dev/flask/microblog2/venv/lib/python3.6/site-packages/flask/app.py", line 1982, in wsgi_app`, `response = self.full_dispatch_request()`.



## Best practice:

Something in between these two. Give enough information for a user to be able to troubleshoot, but not enough for an attacker to get confidential information.

# Session Hijacking

My responsibility to work with OpenID Connect. In short, we will use open-source algorithms already implemented in production-level projects to ensure that authentications are secure, and the session cannot be replicated by a separate party.

## **Best practice:**

Don't write your own code - we will rely on others' expertise on actual authentication protocol, modified for our system.



# Questions?



## Sources and resources

<https://angular.io/guide/security>

<https://www.youtube.com/watch?v=WK2qc4U405I>

<https://ripcordsystems.com/2019/03/09/angular-how-to-prevent-xss-attacks-code-examples/>

<https://www.hindawi.com/journals/scn/2018/6315039/>