GitHub Copilot Training

Andrew Scoppa

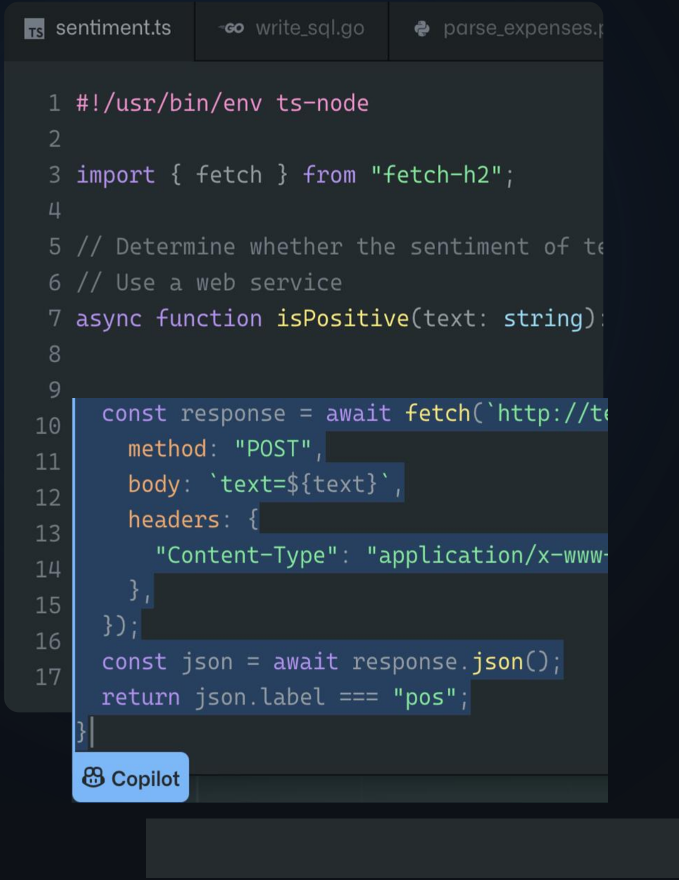
AGENDA

- GitHub Copilot - Introduction
- Best practices & prompt engineering
- In-class Demos
- Secure coding
- Wrap-up, Q&A

GitHub Copilot Fundamentals Recap

Let’s start with a high-level overview of GitHub Copilot

- GitHub Copilot is there to enhance daily work
  - Like a smart assistant or mentor by your side
- Draws context from text & code in open tabs
- Powered by OpenAI
  - Copilot uses a transformative model
  - Think of something like Google Translate
- Trained on large datasets to ensure accuracy
  - It even can help with HTML and markdown!
- Available as an extension to IDEs and editors

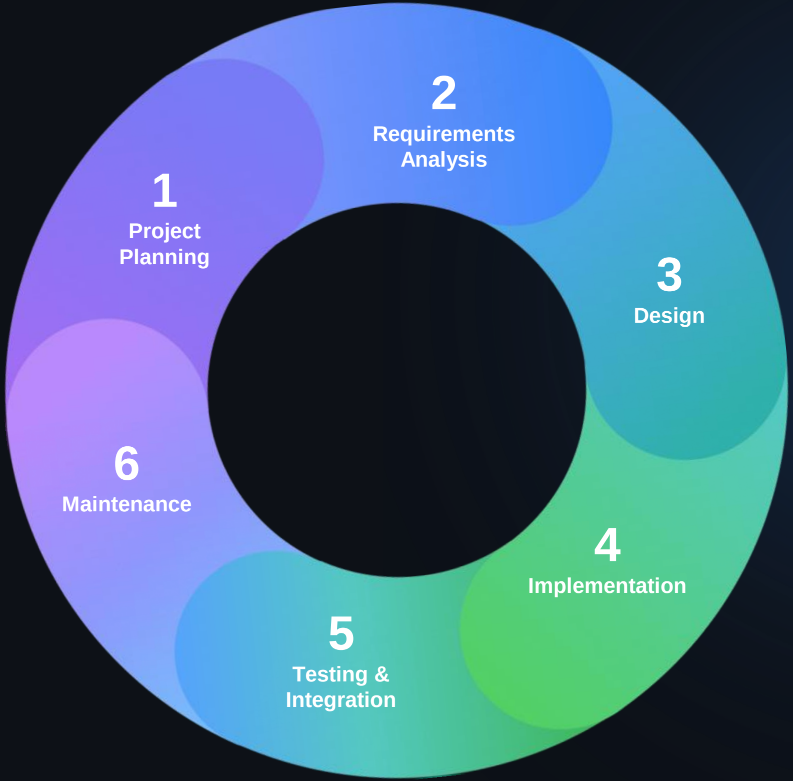


```
#!/usr/bin/env ts-node

import { fetch } from "fetch-h2";

// Determine whether the sentiment of te
// Use a web service
async function isPositive(text: string)

  const response = await fetch(`http://te
    method: "POST",
    body: `text=${text}`,
    headers: {
      "Content-Type": "application/x-www-
    },
  });
  const json = await response.json();
  return json.label === "pos";
}
```

Copilot vs Copilot Chat

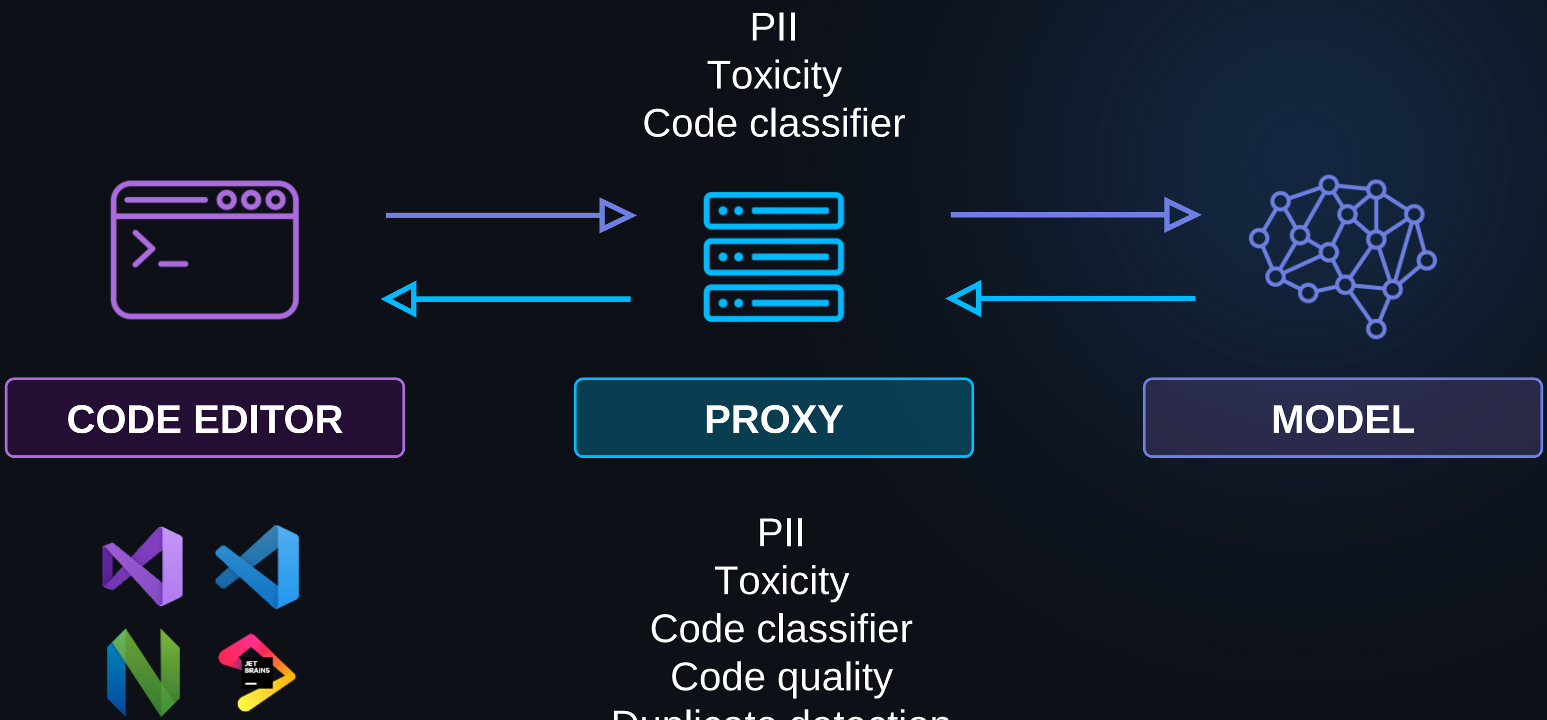
Copilot	Copilot Chat
Direct Code Writing	In-Depth Interactive Assistance
Seamless IDE Integration	Learning & Teaching
Solo Development	Collaborative Scenarios

GitHub Copilot + Chat



Helps developers stay in the flow throughout the entire SDLC

Data flow through the Copilot ecosystem



Prompt Engineering

What is Prompt Engineering?

Prompt engineering is the process of designing and creating high-quality prompts that can be used to generate accurate and useful code suggestions with Copilot.

- Copilot

AI Prompting Techniques

In the realm of GitHub Copilot, Zero-shot, One-shot, and Few-shot prompting refer to guiding the AI with varying levels of examples.

Zero-shot prompting doesn't provide any prior examples, expecting Copilot to understand and generate relevant code purely from the given task description.

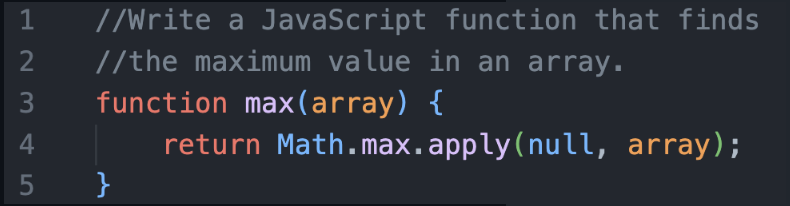
One-shot prompting provides a single example to set the context, assisting Copilot in generating a similar outcome.

Few-shot prompting involves offering multiple examples to establish a clearer pattern for the desired code output.

By understanding these techniques, developers can better instruct Copilot, ensuring more accurate and context-aware code suggestions.

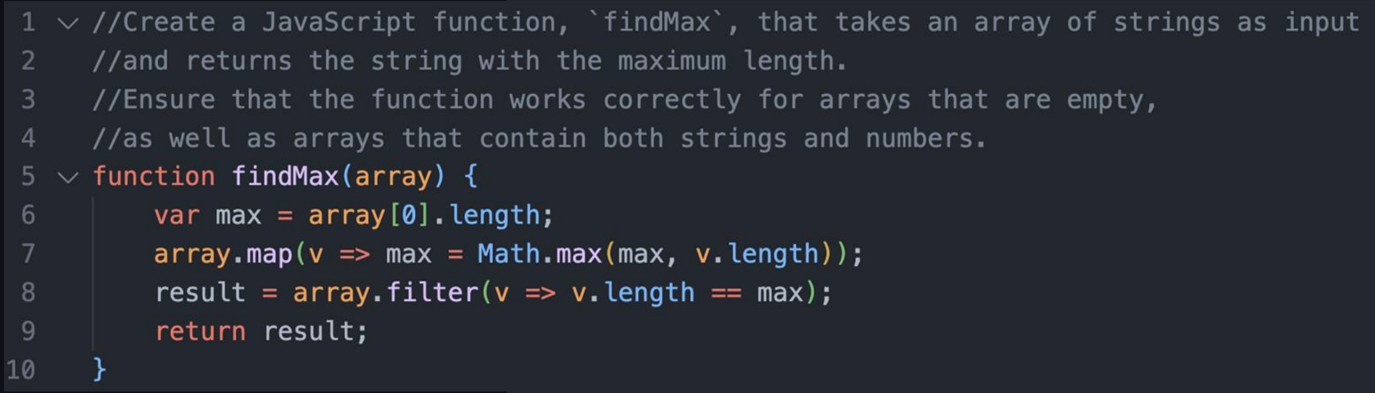
Zero Shot vs One Shot Prompting

Zero Shot Prompt



```javascript
//Write a JavaScript function that finds
//the maximum value in an array.
function max(array) {
    return Math.max.apply(null, array);
}
```

One-Shot Prompt



```javascript
//Create a JavaScript function, `findMax`, that takes an array of strings as input
//and returns the string with the maximum length.
//Ensure that the function works correctly for arrays that are empty,
//as well as arrays that contain both strings and numbers.
function findMax(array) {
    var max = array[0].length;
    array.map(v => max = Math.max(max, v.length));
    result = array.filter(v => v.length == max);
    return result;
}
```

Providing Context

To help Copilot generate accurate suggestions:

- Add a top-level comment block describing the purpose of the file
- Front load as many imports as needed (import / include / requires / etc.)
- Create a detailed comment block describing the purpose of an operation or UDT
- Use sample code as a starting point
- Have related content open in other tabs

Helpful Patterns

```
* Use descriptive variable names to make your intentions clear.

    totalSampleCount = 1000

* Maintain consistent naming conventions for variables and functions.

    i.e. using camelCase for variable names consistency

* Define method signatures with unambiguous parameter names and types.

    double calculateAverageSampleSize(unsigned long samples[], size_t size)

* Use comments to explain the purpose of the code.

    This function 'palindrome' takes an array of strings
    as input and returns  true if a palindrome is found.
```

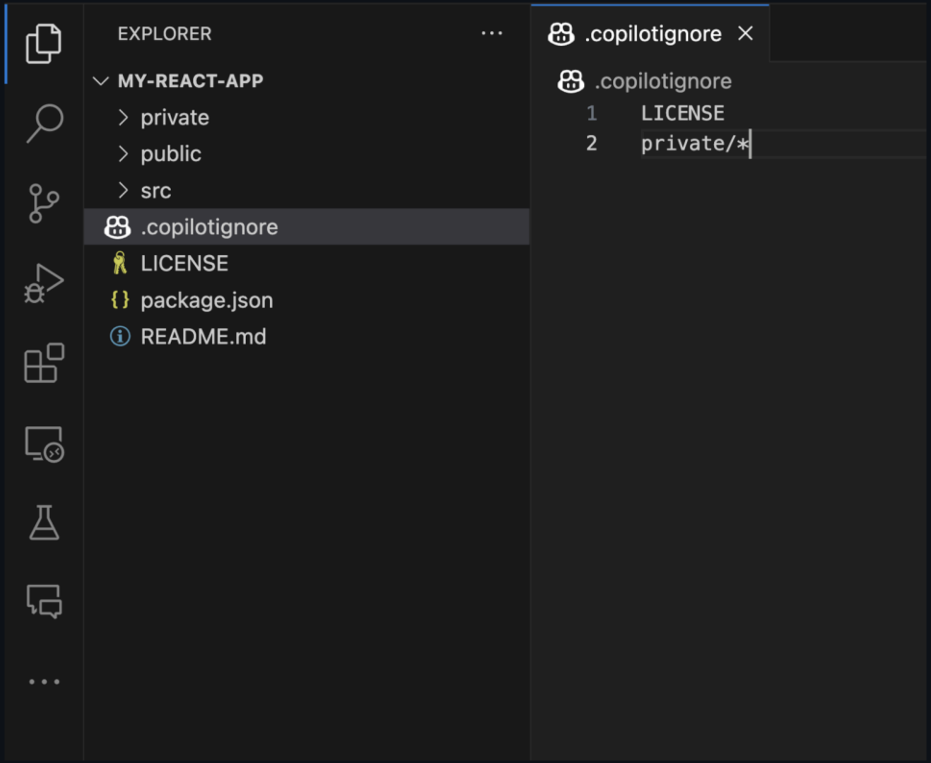
Helpful Patterns

```
* Specify error handling scenarios.

    Exit where the input is NULL and throw and
    exception if the input out of range [min, max] inclusive.

* Describe control flow structures.

    Write a while loop to print the first 'n' values in the fibonnaci sequence.

* Show examples of how to use the code.

    int samples[] = {1, 2, 3, 4, 5};
    double average = calculateAverageSampleSize(samples, 5);
    printf("Average: %f\n", average); // Average: 3.0

* Test your code.

    Write a test suite to verify the correctness of the code.
```

Secure coding

Block files from Copilot

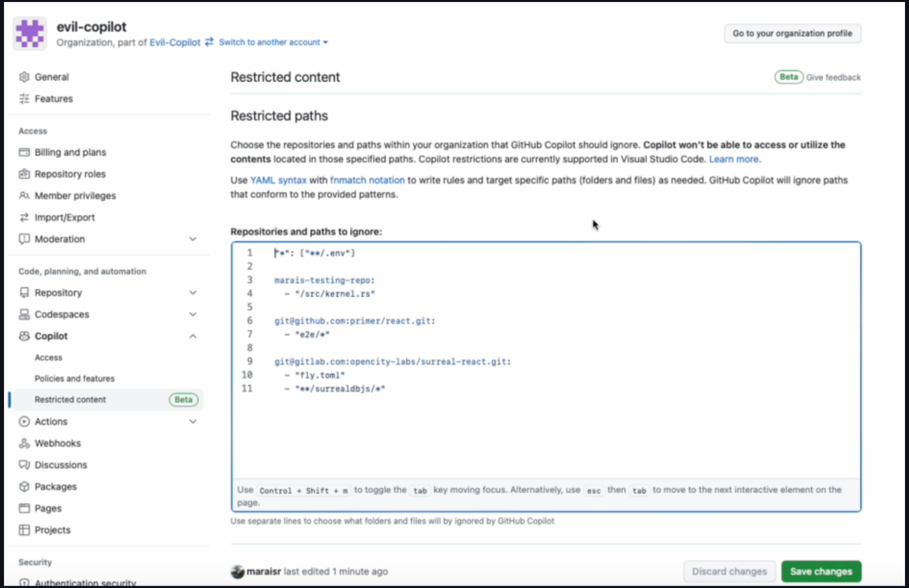
Use .copilotignore to block files and folders from being used by Github Copilot



```
LICENSE
private/*
```

Block files from Copilot

- The Copilot Restricted Content feature allows repository owners to control the usage of Copilot across their codebases.
- This includes specific files, folders, or entire repositories, even if they're not hosted on GitHub.
- The goal is to provide CfB customers a flexible way of controlling what content Copilot can access for prompt crafting and where it can insert code.



Restricted content

Restricted paths

Choose the repositories and paths within your organization that GitHub Copilot should ignore. **Copilot won't be able to access or utilize the contents** located in those specified paths. Copilot restrictions are currently supported in Visual Studio Code. Learn more.

Use YAML syntax with fnmatch notation to write rules and target specific paths (folders and files) as needed. GitHub Copilot will ignore paths that conform to the provided patterns.

Repositories and paths to ignore:

```yaml
"*": ["**/.env"]

marais-testing-repo:
  - "/src/kernel.rs"

git@github.com:primer/react.git:
  - "e2e/*"

git@gitlab.com:opencity-labs/surreal-react.git:
  - "fly.toml"
  - "**/surrealdbjs/*"
```

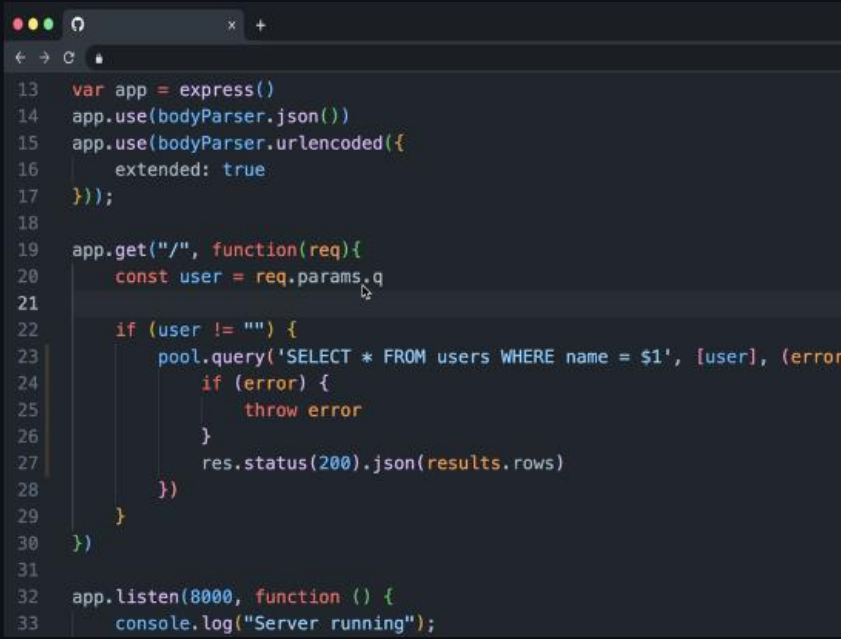
Use `Control + Shift + m` to toggle the `tab` key moving focus. Alternatively, use `esc` then `tab` to move to the next interactive element on the page.

Use separate lines to choose what folders and files will by ignored by GitHub Copilot

maraisr last edited 1 minute ago

Copilot and Secure Coding

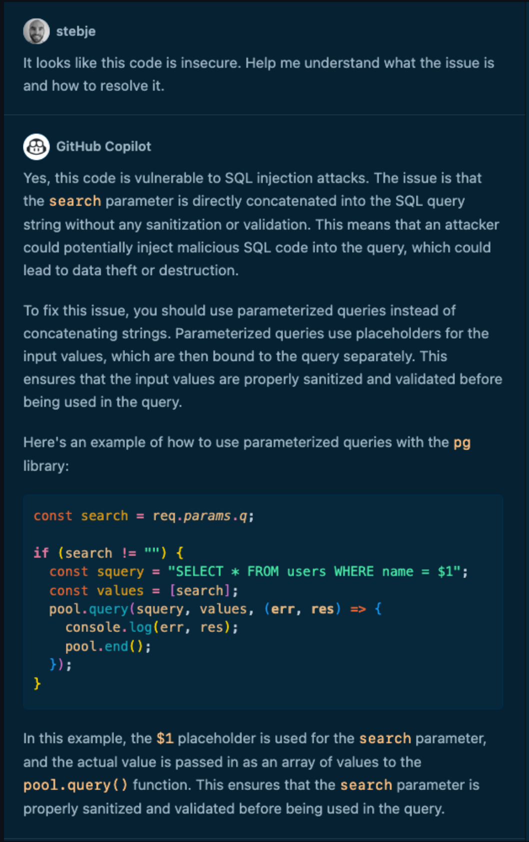
- AI-based vulnerability system that helps prevent insecure coding patterns (e.g. SQL script injection)
- Vulnerability filters are applied only to the Copilot-generated suggestion itself
    - It cannot detect downstream vulnerabilities introduced by the code e.g. on deployment infrastructure
    - **We recommend taking the same precautions you take with code written by engineers (linting, code scanning, etc.)**
- Copilot Chat can be used to query code for known vulnerabilities



```
var app = express()
app.use(bodyParser.json())
app.use(bodyParser.urlencoded({
    extended: true
}));

app.get("/", function(req){
    const user = req.params.q

    if (user != "") {
        pool.query('SELECT * FROM users WHERE name = $1', [user], (erro
            if (error) {
                throw error
            }
            res.status(200).json(results.rows)
        })
    }
})

app.listen(8000, function () {
    console.log("Server running");
```

Detection & Remediation

- Prompt for insecure coding patterns
- Use GHAS Code Scanning results
- Write custom CodeQL queries
- Increase your knowledge of secure coding patterns
- Create custom Secret Scanning patterns



stebje

It looks like this code is insecure. Help me understand what the issue is and how to resolve it.

GitHub Copilot

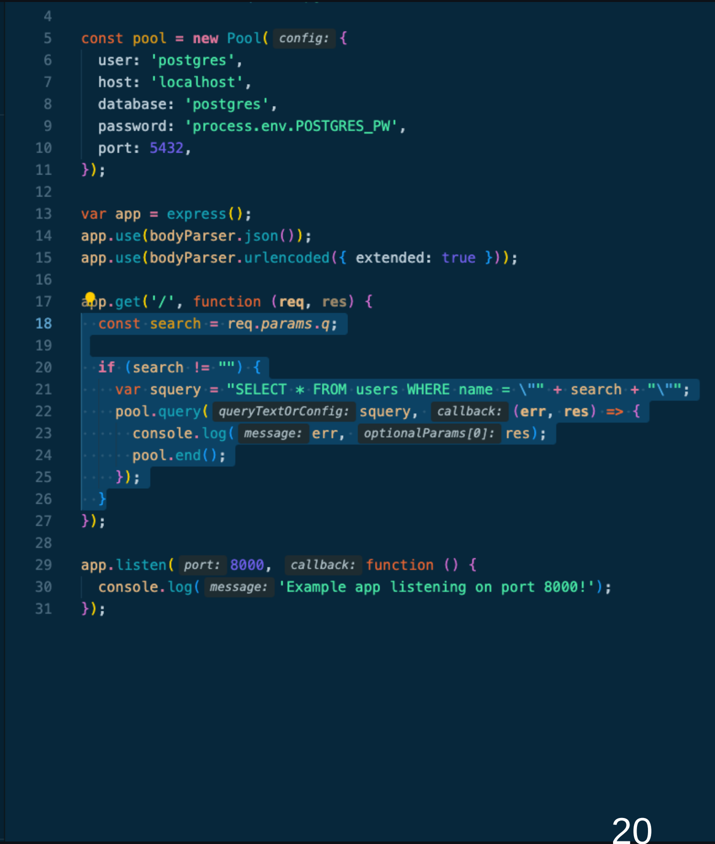
Yes, this code is vulnerable to SQL injection attacks. The issue is that the `search` parameter is directly concatenated into the SQL query string without any sanitization or validation. This means that an attacker could potentially inject malicious SQL code into the query, which could lead to data theft or destruction.

To fix this issue, you should use parameterized queries instead of concatenating strings. Parameterized queries use placeholders for the input values, which are then bound to the query separately. This ensures that the input values are properly sanitized and validated before being used in the query.

Here's an example of how to use parameterized queries with the `pg` library:

```
const search = req.params.q;

if (search != "") {
  const squery = "SELECT * FROM users WHERE name = $1";
  const values = [search];
  pool.query(squery, values, (err, res) => {
    console.log(err, res);
    pool.end();
  });
}
```

In this example, the `$1` placeholder is used for the `search` parameter, and the actual value is passed in as an array of values to the `pool.query()` function. This ensures that the `search` parameter is properly sanitized and validated before being used in the query.



```

const pool = new Pool( config: {
  user: 'postgres',
  host: 'localhost',
  database: 'postgres',
  password: 'process.env.POSTGRES_PW',
  port: 5432,
});

var app = express();
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

app.get('/', function (req, res) {
  const search = req.params.q;

  if (search != "") {
    var squery = "SELECT * FROM users WHERE name = \"" + search + "\"";
    pool.query( queryTextOrConfig: squery,  callback: (err, res) => {
      console.log( message: err,  optionalParams[0]: res);
      pool.end();
    });
  }
});

app.listen( port: 8000,  callback: function () {
  console.log( message: 'Example app listening on port 8000!');
});
```

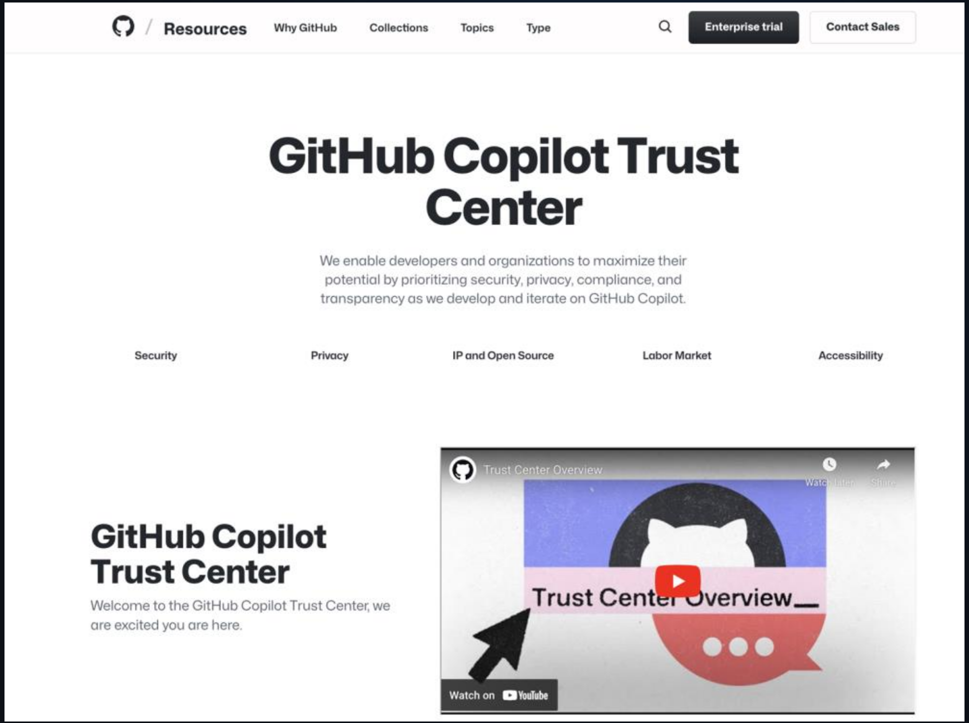
Copilot + GHAS

- Copilot is not a replacement of GHAS features
- Copilot can be used in tandem with GHAS features to detect and remediate vulnerabilities earlier during the SDLC
    - GHAS Code scanning results
    - GHAS Secret scanning

Security & Trust

Copilot Trust Center

- Security
- Privacy
- Data flow
- Copyright
- Labor market
- Accessibility
- Contracting



Resources Why GitHub Collections Topics Type Enterprise trial Contact Sales

GitHub Copilot Trust Center

We enable developers and organizations to maximize their potential by prioritizing security, privacy, compliance, and transparency as we develop and iterate on GitHub Copilot.

Security Privacy IP and Open Source Labor Market Accessibility

GitHub Copilot Trust Center

Welcome to the GitHub Copilot Trust Center, we are excited you are here.

Wrap Up

Thank you