
HOPE
X V


Security
in Development
Security

Introduction to the workshop

MODUS CREATE

- Team Intro
  - Andy Dennis - VP Platform and Cloud
  - Bill Reyor - Director of Security
  - Sean Clayton - Security Architect
- What is Modus Create?
- Course structure
  - PART 1 (IDE / LOCAL SECURITY)
  - BREAK (15 Minutes)
  - PART 2 (PIPELINES SECURITY)

2600

A Quick Class Survey

MODUS CREATE

Have you ever?

> Cloned a repository with Git?
> Merged a pull request
> Exploited an SQL injection vulnerability? (or had to fix one)

2600

If you need help raise your hand


Getting
Setup

Getting setup - The tools

MODUS CREATE

You will need

- **Git**
  https://git-scm.com/book/en/v2/Getting-Started-Installing-Git
- **IDE (if you don't already)**
  https://code.visualstudio.com/
- **GitHub account**
  https://www.github.com
- **SonarLint/SonarCloud**
  https://docs.sonarsource.com/sonarcloud/improving/sonarlint/
- **CodeQL**
  https://codeql.github.com/
- **Optional: Copilot**
  https://github.com/features/copilot

During the course we will use

- **AWS git-secrets**
  https://github.com/awslabs/git-secrets
- **Talisman**
  https://github.com/thoughtworks/talisman
- **BFG**
  https://rtyley.github.io/bfg-repo-cleaner/

Getting setup - repository

MODUS CREATE

We've setup a repository for you to work with during this course.

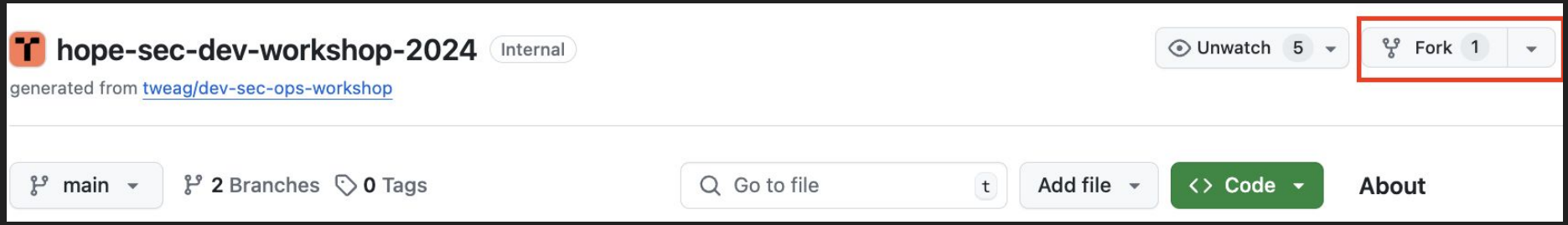
> **Located at Modus Create OSPO (Tweag) repository**
> https://github.com/tweag/hope-sec-dev-workshop-2024

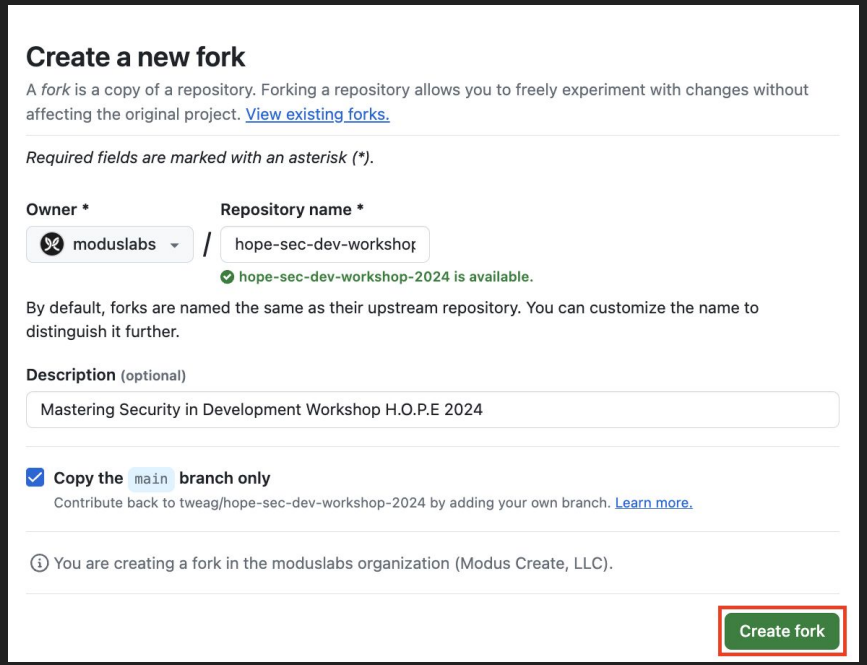
> **Fork it** 🍴
> https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/working-with-forks/fork-a-repo

2600

Fork It!

MODUS CREATE





2600

A few words about Code Spaces

MODUS CREATE

2600

Add an SSH key to your Github account

MODUS CREATE

1. **Generate a new SSH key pair on your local machine if you don't have one**

```
> ssh-keygen -t ed25519 -C "your_email@example.com"
```

2. **Copy the public key file (e.g. id_ed25519.pub) to your clipboard**

```
> clip < ~/.ssh/id_ed25519.pub     # On Windows

> pbcopy < ~/.ssh/id_ed25519.pub   # On macOS

> cat ~/.ssh/id_ed25519.pub        # On Linux, then manually copy
```

3. **Login to your github account and goto**

   A. Settings -> Access -> SSH & GPG keys -> New SSH key

   B. Enter a title for the key, then paste the public key we generated in step 2.

2600

Follow Along with the Demos

MODUS CREATE

The lab instructions are divided into two parts, each with its own README file:

Part 1

1. Navigate to the **part1** directory in the main repository
2. Open the README.md file
3. Follow the step-by-step instructions to complete the Part 1 demo

Part 2

1. Navigate to the **part2** directory in the main repository
2. Open the README.md file
3. Follow the step-by-step instructions to complete the Part 2 demo

The **README** files contain detailed guidance, code snippets, and examples to help you follow along with the live demos during the workshop.

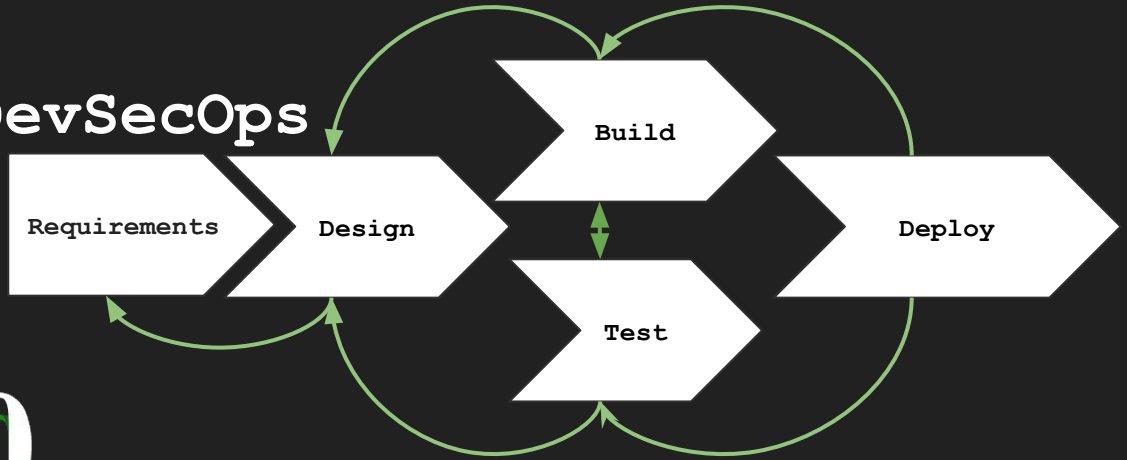
2600

What is DevSecOps?

MODUS CREATE

Traditional

DevSecOps

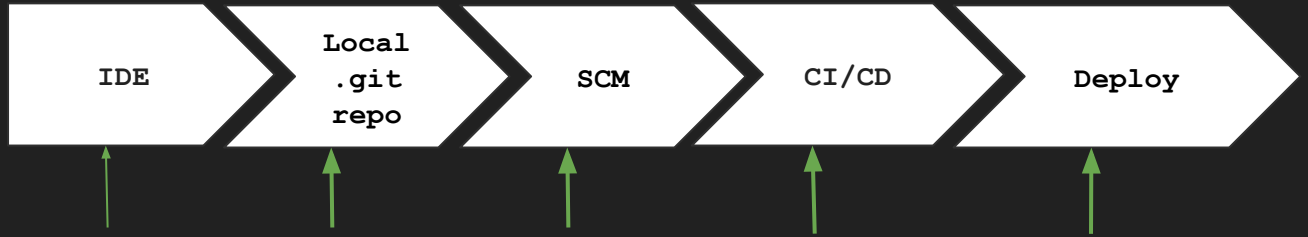


Security feedback loops

2600

What is Shifting Left?

MODUS CREATE



Security is constantly pushed further left in SDLC

2600

MODUS CREATE

Part 1

Local Security
IDEs and CodeSpace

Introduction

MODUS CREATE

The first part of the course is dedicated to integrating security into your local development environment.

This covers:

> Security integration into the IDE
> Ignore files and clean up
> Pre-commit hooks

Security in the IDE

MODUS CREATE

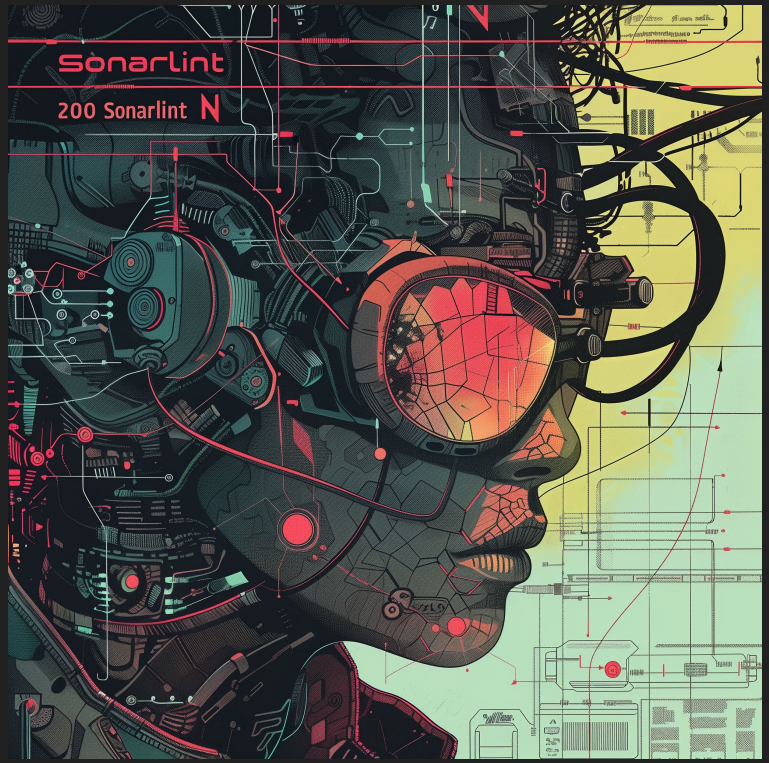
2600

IDE plugins

- Acts as a first line of defense
- Catch errors in the code
- Detect secrets in the code e.g. passwords and API keys
- Uncover vulnerabilities like XSS and code that allows for SQL Injections

SonarLint

MODUS CREATE



Setup instructions

SonarLint works with a number of popular IDEs.

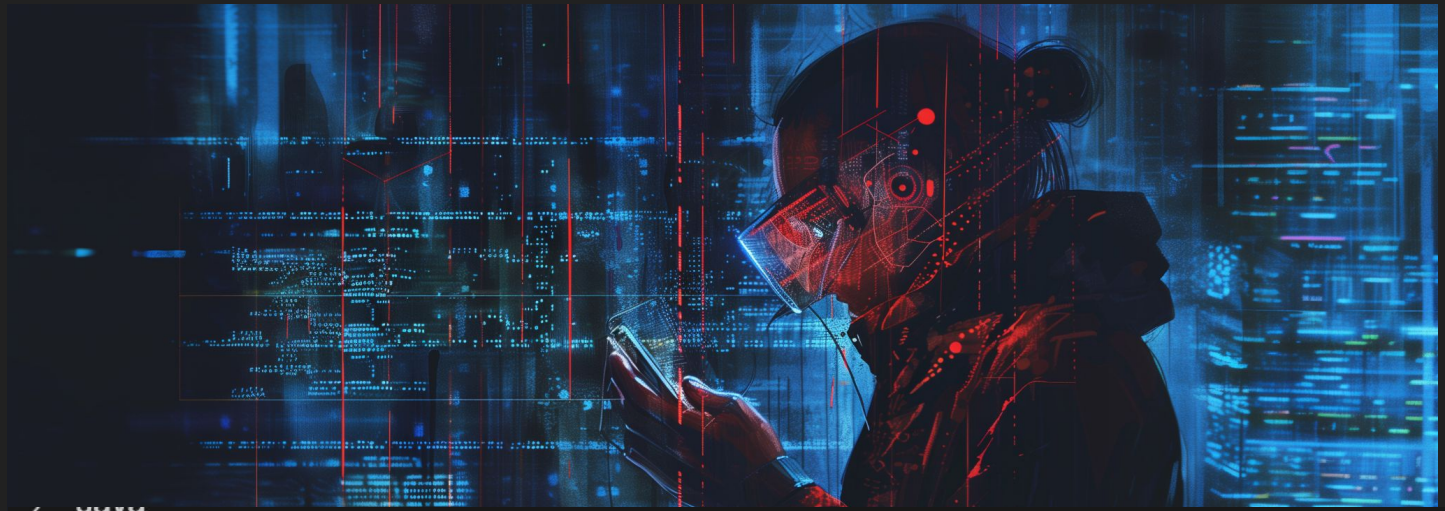
1. Follow the steps for your IDE to install it.
2. Setup a SonarCloud account via your GitHub user if you don't have one

(See README in REPO)

2600

SonarLint - Rules

MODUS CREATE



JavaScript

__proto__ property should not be used on

"===" and "!==" should be used instead of "==" and "!=" on

"alert(...)" should not be used on

2600

2600

Demo 1 - Sonarlint

MODUS CREATE

SonarLint

MODUS CREATE

2600

We can see the plugin identified:

- Use of the alert statement
- Useless variable assignments
- Use of the debugger statement
- Use of var

Some of these have security implications!

MODUS CREATE

Install the plugin now into your IDE:

https://marketplace.visualstudio.com/items?itemName=github.vscode-codeql

Or if you would prefer to use it from the CLI follow the instructions here:

https://docs.github.com/en/code-security/codeql-cli/getting-started-with-the-codeql-cli/setting-up-the-codeql-cli

2600

Demo 2 - CodeQL

MODUS CREATE

CodeQL

MODUS CREATE

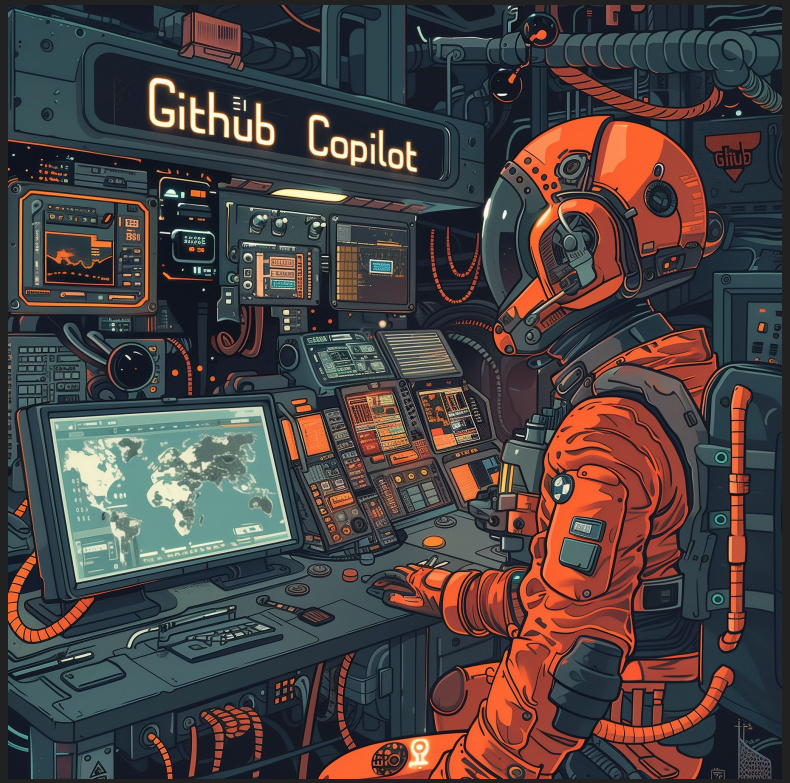
- Execute CodeQL IDE
- Detects Coding Issues
- You can write custom detections
- Deploy at scale (used in research)

How is it different from a typical SAST tool?

2600

GitHub Copilot

MODUS CREATE



GitHub Copilot

Used 1 reference >

To prevent SQL injection, you should use parameterized queries or prepared statements. This ensures that user input is always treated as literal data and not part of the SQL command. Here's how you can modify your `saveToDatabase` method to use parameterized queries:

```
static saveToDatabase(blogPost) {
    const sql = `INSERT INTO blog_posts (title, content, author) VALUES (?, ?, ?
    // Pretend database call
    console.log(`Executing SQL: ${sql}`);
    var db = window.openDatabase("sqlcachedb", "1.0", "Cache result", 200000);
    db.transaction(function(tx){
        tx.executeSql(sql, [blogPost.title, blogPost.content, blogPost.author],
            console.log("Data inserted successfully");
        }, function(tx, error){
            console.log("Error occurred while inserting data");
        });
    });
}
```

In this code, the `?` are placeholders for the actual values. The values are passed in as an array in the second argument to `executeSql`. This ensures that the values are properly escaped, preventing SQL injection attacks.

2600

Demo 3 - Github Copilot vs DuckDuckGo

MODUS CREATE

2600

Pre-commit Hooks

MODUS CREATE

Pre-commit hooks

MODUS CREATE

- Scripts run before commiting
- Enforce code quality, catch issues
- Customizable, but can be skipped

One of the best ways to prevent secrets from being committed.

2600

Pre-commit hooks - key features

MODUS CREATE

2600

Used to perform tasks like

- Quality enforcement
- Prevent commits with problems e.g. security vulnerabilities

Key features include

- Exist in the .git/hooks https://git-scm.com/docs/githooks
- Not transferred with the repo
- Can be bypassed (--no-verify flag)

AWS Git Secrets

MODUS CREATE

Let's open the readme, and follow along. We are now going to:

- Install the hook in the repo
- Register some patterns to block
- Run a manual scan
- Explore what blocking a commit looks like

Fire up your command line!

https://github.com/awslabs/git-secrets

2600

Demo 4 - Pre-commit hook

MODUS CREATE

2600

Demo 5 - Talisman

MODUS CREATE

Talisman

MODUS CREATE

Suite of pre-commit hooks w/security

- **Download** https://github.com/thoughtworks/talisman

Features of Talisman

- **Overview of key features**

Exploring Talisman

- Switch to the **README** file
- Command line experiments

Talisman

2600

2600

.gitignore

Using .gitignore files

MODUS CREATE

Pre-commit hooks are very handy for catching secrets and other code we do not want to push up to our repository.

How about if there are whole classes of file we want to avoid adding?

Well we can add an extra layer of defense using the **.gitignore** file.

The **.gitignore** file allows us to:

- Create an exclusion list of file types and directories so these are not tracked
- Apply fine grained controls over this exclusion list by adding to each repository
- Can be version controlled so every user inherits it. This is unlike the hooks we just explored

2600

Using .gitignore files

MODUS CREATE

2600

2600

Demo 6 - .gitignore

MODUS CREATE

Future project: Cleaning up secrets and files

BFG can be obtained from:

https://rtyley.github.io/bfg-repo-cleaner/

Remember to rename the file to bfg.jar (drop the version from it).

The bfg.jar is pre-installed in your workspace

The readme provides an example of how we can use BFG to:

- Remove certificates
- Remove files with passwords
- Remove specific strings (passwords, API keys) from files

Feel free to try this out in your own time.

2600

Part 1 wrap up

MODUS CREATE

Congratulations, you've now reached the end of **Part 1** and have a basic understanding of IDE plugins, pre-commit hooks and .gitignore files.

Some of the tools such as Talisman have many features we couldn't cover today. You may be interested in taking the time to dig into these after the course.

Let's take a short break, then we will move onto **Part 2**.

2600


15
Minute Break

Part 2

Security in the CI/CD pipeline

MODUS CREATE

Welcome to Part 2

In this section of the workshop we will cover:

- Secrets scanning
- Handling secrets in GitHub
- Detecting security vulnerabilities in the repository
- Vulnerable dependencies
- Static Analysis (SAST)
- Branch protection rules

2600

2600

Secrets Scanning

MODUS CREATE

MODUS CREATE

Secrets Scanning

We saw how we can detect secrets locally using git-secrets and Talisman.

We also saw how we can prevent them being committed via pre-commit hooks and .gitignore files.

We are now going to look how we can do this in GitHub, before looking more broadly at how to handle vulnerabilities.

Useful Documentation

- TruffleHog: https://github.com/trufflesecurity/trufflehog
- GitHub Secrets Scanning: https://docs.github.com/en/code-security/secret-scanning/about-secret-scanning
- Horusec: https://horusec.io/site/

2600

Demo 7 - Secrets Scanning

MODUS CREATE

2600

Handling Secrets in GitHub

MODUS CREATE

MODUS CREATE

Secrets in GitHub

So in summary some best practices for handling secrets in GitHub include:

- **Prepare** - use a secret manager (GitHub has one)
- **Prevent** - git-secrets, and the other tools we saw with the IDE/Pre-commit hooks
- **Detect** - pipeline checks, we can also use TruffleHog
- **Respond** - BFG / Git-filter

Let’s look at secrets storage in GitHub quickly

2600

Demo 8 - Handling Secrets

MODUS CREATE

2600

Detecting Security Vulnerabilities

MODUS CREATE

Detecting security vulnerabilities

Multiple tools for detecting vulnerabilities.

Secrets scanning - pattern/entropy detection

SAST - Static Analysis Security Testing

SCA - Software Composition Analysis

SBOMs - Software Bills of Materials

GitHub has the following features:

> SECURITY.md (Provide instructions for how to report a security vulnerability)
> Secrets scanning (which we just saw)
> Dependency analysis (Dependabot)
> CodeQL(SAST)via Actions
> GitHub SBOM

2600

2600

Dependency Vulnerabilities

MODUS CREATE

Dependency vulnerabilities

MODUS CREATE

GitHub supports detection of vulnerable dependencies through **Dependabot**.

This tool allows us to check files such as package.json and pom.xml to understand if our application is including vulnerable components

Features include:

- Detect vulnerable and out of date packages
- Create automatic pull requests to remediate issues
- List vulnerabilities associated with the packages

Let's take a look at an example of it in action.

2600

2600

Demo 9 - Dependencies

2600

Static Analysis

MODUS CREATE

Static Analysis (SAST)

GitHub comes integrated with CodeQL which we experimented with in Part 1. This supports static analysis security testing of a number of programming languages via Actions.

CodeQL is free to use for public repositories.

You can learn more here:

https://docs.github.com/en/code-security/code-scanning/introduction-to-code-scanning/about-code-scanning

Features:

- Multiple language support (Java, JS, Ruby etc.)
- Highlights security vulnerabilities
- Allows for custom queries/query packs
- Execute as a gating mechanism in your CI/CD pipeline

Demo time!

2600

2600

Demo 10 - SAST

MODUS CREATE

2600

Branch Protection Rules

MODUS CREATE

Branch protection rules

Branch protection rules can be used to prevent code being merged until it passess all the relevant tests.

It also allows us to lock down who can commit to the branch.

Features include:

- Block merges until security checks are passed
- Lockdown branch ownership via **CODEOWNERS** files
- Require multiple individuals to approved a pull request before it can be merged

2600

Demo 11 - Branch Protection

MODUS CREATE

MODUS CREATE

SBOMs

SBOM stands for Software Bill of Materials.

It’s a great way of getting a snapshot of all the components in your application.

GitHub supports SBOMs out of the box for repositories stored there.

Features of SBOM include:

- CycloneDX format https://cyclonedx.org/
- SPDX format https://spdx.dev/
- VEX https://www.cisa.gov/sites/default/files/2023-01/VEX_Use_Cases_Aprill2022.pdf

2600

2600

Demo 12 - SBOMS

MODUS CREATE

Conclusion

MODUS CREATE

MODUS CREATE

Recap

In this workshop we covered a number of areas related to mastering security in development.

- Security in the IDE
- Pre-commit hooks
- Secrets scanning
- SAST
- Dependency scanning
- Branch protection rules
- SBOMS

2600

Next Steps

MODUS CREATE

After this workshop you can take the repository you forked and use it as a starting to point for your own projects.

Some ideas for other areas to explore are listed on the right.

- Try integrating with Infrastructure-as-Code. Check out Checkov: https://www.checkov.io/
- Experiment with other third party tooling for SAST. For example, the open source Horusec: https://github.com/ZupIT/horusec
- Look into container scanning: https://github.com/quay/clair

2600

Thanks for attending!

MODUS CREATE

Thank you for attending the workshop today. A big thank you to the HOPE Staff and volunteers as well.

Questions?

andy.dennis@moduscreate.com

Remember to check out:

https://moduscreate.com/careers/

For our latest job posts!

2600