SecurityBoat Community

Connect, Learn & Grow!!

# Secure Code Review

Ensuring Code Security with Effective Reviews

# About Speaker

Over 8 years of expertise in the field of Cybersecurity.

### Role

Product Security Consultant at Hitachi-GL.

### Skills

Product Security
Cloud Security
Threat Modeling
VAPT (Vulnerability Assessment and Penetration Testing)
Security Architecture

SecurityBoat Community
Connect, Learn & Grow!!

# Agenda

Exploring Secure Code Reviews: Importance, Techniques, Tips and Best Practices

**Introduction to Secure Code Review**

**Common Vulnerabilities OWASP Top 10**

**Tools and Techniques**

**Best Practices**

SecurityBoat Community
Connect, Learn & Grow!!

# Introduction

SecurityBoat Community
Connect, Learn & Grow!!

# What is Source Code Review

Secure code review is a manual or automated process that examines an application's source code.

### Objective

Identify existing security flaws or vulnerabilities.

### Focus

Detect logic errors, ensure proper implementation of specifications, and review adherence to style guidelines.

### Activities

Comprehensive evaluation of code to maintain security, functionality, and compliance.

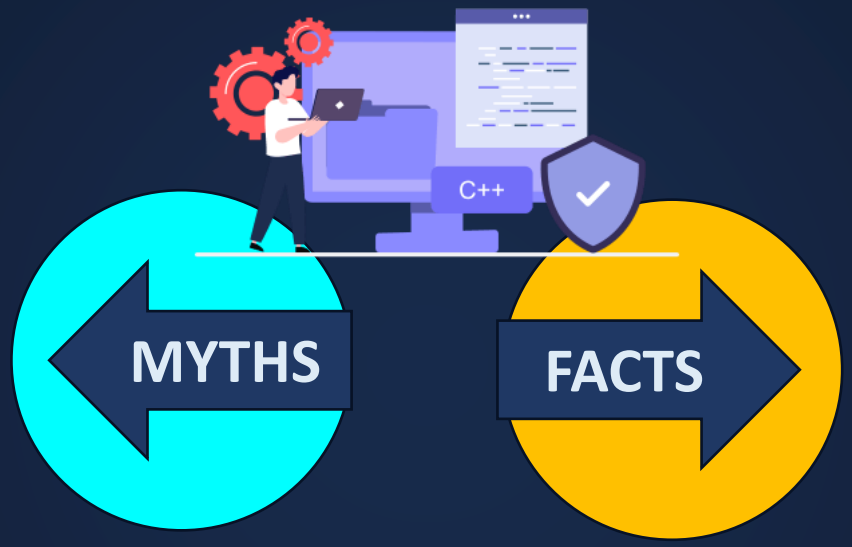
SecurityBoat Community
Connect, Learn & Grow!!

# SAST

SecurityBoat Community
Connect, Learn & Grow!!

# What Is SAST

Static Application Security Testing (SAST) is a frequently used Application Security (AppSec) tool, which scans an application's source, binary, or byte code. A white-box testing tool, it identifies the root cause of vulnerabilities and helps remediate the underlying security flaws. SAST solutions analyze an application from the "inside out" and do not reed a running system to perform a scan.




## Tools

Checkmarx
MICRO FOCUS Fortify
sonarqube

SecurityBoat Community
Connect, Learn & Grow!!

# Why Is Secure Code Review Important?

### Reduced Delivery Defects

Early identification of vulnerabilities lowers defects, minimizing costly rework and delays.

### Increased Developer Productivity

Resolving issues during development saves time and boosts productivity.

### Improved Consistency and Maintainability

Enforces coding standards and ensures long-term software maintainability.

### Cost Savings and ROI

Early fixes prevent costly security breaches and reduce rework expenses.

# SAST Analysis Types

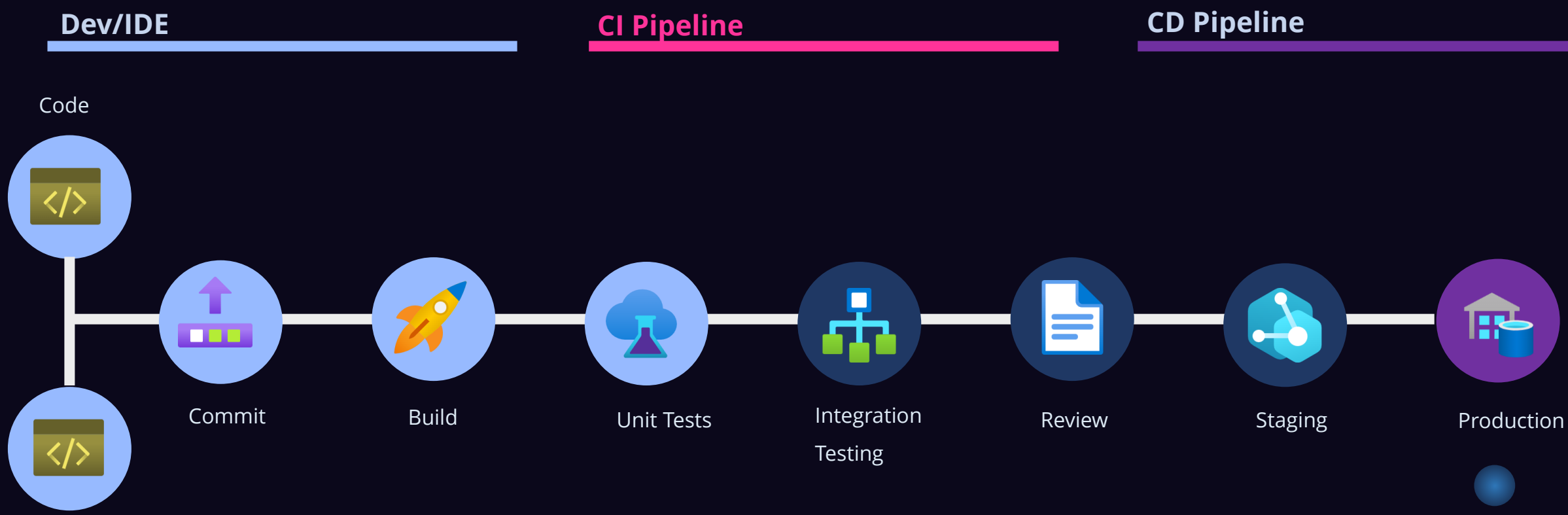
SecurityBoat Community
Connect, Learn & Grow!!

| | |
|---|---|
| 01 Pattern-Based Analysis | • **Description:** Identifies vulnerabilities by matching code patterns against known insecure coding practices or vulnerabilities.<br>• **Example:** Detecting hardcoded credentials, SQL injection patterns, or use of unsafe functions like strcpy in C/C++.<br>• **Strengths:** Fast and effective for detecting well-known vulnerabilities.<br>• **Limitations:** May produce false positives if the pattern is too generic. |
| 02 Data Flow Analysis | • **Description:** Tracks how data moves through the application to identify potential security risks, such as tainted data reaching sensitive sinks.<br>• **Example:** Identifying if user input (source) flows into an SQL query (sink) without proper sanitization.<br>• **Strengths:** Effective for detecting injection flaws, cross-site scripting (XSS), and similar vulnerabilities.<br>• **Limitations:** Can be computationally expensive and may miss complex data flows. |
| 03 Control Flow Analysis | • **Description:** Examines the order in which statements, function calls, and other control structures are executed to identify security issues.<br>• **Example:** Detecting unreachable code, infinite loops, or improper error handling.<br>• **Strengths:** Helps identify logical flaws and potential backdoors.<br>• **Limitations:** Limited to structural issues and may not detect data-related vulnerabilities. |
| 04 Semantic Analysis | • **Description:** Analyzes the meaning and context of code to identify vulnerabilities that may not be apparent through pattern matching alone.<br>• **Example:** Detecting insecure cryptographic implementations or improper use of APIs.<br>• **Strengths:** Provides deeper insights into the code's behavior.<br>• **Limitations:** Requires advanced tooling and may be slower than other methods. |
| 05 Configuration Analysis | • **Description:** Checks for insecure configurations in the code, such as weak encryption algorithms, insecure default settings, or misconfigured security controls.<br>• **Example:** Identifying use of deprecated or weak cryptographic algorithms like MD5 or DES.<br>• **Strengths:** Helps ensure compliance with security best practices.<br>• **Limitations:** Limited to configuration-related issues. |
| 06 Taint Analysis | • **Description:** A specialized form of data flow analysis that tracks "tainted" (untrusted) data from its source to its sink.<br>• **Example:** Detecting if user input is used in an SQL query without sanitization.<br>• **Strengths:** Highly effective for identifying injection vulnerabilities.<br>• **Limitations:** May produce false positives if the tool cannot accurately track taint propagation. |

# How Does SAST Work?

SecurityBoat Community
Connect, Learn & Grow!!

## CI/CD Pipeline



SecurityBoat Community
Connect, Learn & Grow!!

# Secure Coding Standards

Secure coding standards are rules and guidelines used to prevent security vulnerabilities. They prevent, detect, and eliminate errors that could compromise software security.

International Organization for Standardization ISO 27034

HIPAA COMPLIANCE

NIST National Institutes of Standards and Technology SP 800-53

OWASP

PCI DSS COMPLIANT

DISA

SecurityBoat Community
Connect, Learn & Grow!!

# OWASP TOP 10

2021

SecurityBoat Community
Connect, Learn & Grow!!

# A01 Broken Access Control

# Broken Authorization

SecurityBoat Community
Connect, Learn & Grow!!

## Vulnerable Code

BAC

```
public class BankAccountService {

    public BankAccount createBankAccount(String accountNumber, String
accountType, String accountName, String accountSSN, double balance, User
user) {
        // Check if the user is authenticated and has the necessary
permissions
        if (user == null || !user.isAuthenticated() || !user.hasPermission
("CREATE_ACCOUNT")) {
            throw new SecurityException("Unauthorized user");
        }

        // Broken Access Control: No check to ensure the user is creating
an account for their own SSN
        BankAccount account = new BankAccount();
        account.setAccountNumber(accountNumber);
        account.setAccountType(accountType);
        account.setAccountOwnerName(accountName);
        account.setAccountOwnerSSN(accountSSN); /
ownership
        account.setBalance(balance);

        return account;
    }
}
```

The method does not verify if the accountSSN belongs to the authenticated user.

An attacker with the CREATE_ACCOUNT permission can create a bank account for any SSN, even if it doesn't belong to them.

## Secure Code

BAC_Fix

```
public class BankAccountService {

    public BankAccount createBankAccount(String accountNumber, String
accountType, String accountName, String accountSSN, double balance, User
user) {
        // Check if the user is authenticated and has the necessary
permissions
        if (user == null || !user.isAuthenticated() || !user.hasPermission
("CREATE_ACCOUNT")) {
            throw new SecurityException("Unauthorized user");
        }

        // Fixed: Validate that the authenticated user is creating an
account for their own SSN
        if (!user.getSSN().equals(accountSSN)) {
            throw new SecurityException("You are not authorized to create
an account for this SSN.");
        }

        // Proceed with creating the bank account if authorization is
successful
        BankAccount account = new BankAccount();
        account.setAccountNumber(accountNumber);
        account.setAccountType(accountType);
        account.setAccountOwnerName(accountName);
        account.setAccountOwnerSSN(accountSSN); // SSN is validated
        account.setBalance(balance);

        return account;
    }
}
```

The method now checks if the accountSSN matches the authenticated user's SSN (user.getSSN()).
The accountSSN is validated to ensure it belongs to the authenticated user.

SecurityBoat Community
Connect, Learn & Grow!!

# A02 Cryptographic Failures

# Weak Algorithm

SecurityBoat Community
Connect, Learn & Grow!!

**Vulnerable Code**

```python
import hashlib

def hash_password(password):
    # Using MD5, which is insecure
    return hashlib.md5(password.encode()).hexdigest()

# Example usage
password = "my_secure_password"
hashed_password = hash_password(password)
print(f"Hashed Password: {hashed_password}")
```

MD5 is cryptographically broken and susceptible to collision attacks.

**Secure Code**

```python
def hash_password_secure(password):
    # Using SHA-256 (better than MD5 but still not ideal for passwords)
    return hashlib.sha256(password.encode()).hexdigest()

def hash_password_bcrypt(password):
    # Using bcrypt, which is designed for password hashing
    return bcrypt.hashpw(password.encode(), bcrypt.gensalt())

# Example usage
password = "my_secure_password"
hashed_password_sha256 = hash_password_secure(password)
hashed_password_bcrypt = hash_password_bcrypt(password)

print(f"SHA-256 Hashed Password: {hashed_password_sha256}")
print(f"bcrypt Hashed Password: {hashed_password_bcrypt.decode()}")
```

bcrypt is specifically designed for password hashing and includes a salt to prevent rainbow table attacks.

# Hardcoding Credentials

SecurityBoat Community
Connect, Learn & Grow!!

**Vulnerable Code**

```python
import boto3

# Hardcoded AWS credentials (INSECURE)
aws_access_key_id = "AKIAX3DFU6N9ZWQ9KIX"
aws_secret_access_key = "X1Gk+wC4N8geuhsA/47YwNhdua/quodnFensd+j"

# Create a session using hardcoded credentials
session = boto3.Session(
    aws_access_key_id=aws_access_key_id,
    aws_secret_access_key=aws_secret_access_key,
)

# Example: List S3 buckets
s3_client = session.client("s3")
response = s3_client.list_buckets()
print("S3 Buckets:", response["Buckets"])
```

AWS secret keys are exposed in the source code.

If the code is committed to a public repository (e.g., GitHub), attackers can easily find and misuse the keys.

**Secure Code**

```python
import os
import boto3
from dotenv import load_dotenv

# Load environment variables from .env file
load_dotenv()

# Retrieve AWS credentials from environment variables
aws_access_key_id = os.getenv("AWS_ACCESS_KEY_ID")
aws_secret_access_key = os.getenv("AWS_SECRET_ACCESS_KEY")

if not aws_access_key_id or not aws_secret_access_key:
    raise ValueError("AWS credentials not found in environment variables")

# Create a session using environment variables
session = boto3.Session(
    aws_access_key_id=aws_access_key_id,
    aws_secret_access_key=aws_secret_access_key,
)

# Example: List S3 buckets
s3_client = session.client("s3")
response = s3_client.list_buckets()
prin("S3 Buckets:", response["Buckets"])
```

Environment variables can be managed securely and are not included in version control.
[AWS CLI Configuration, IAM Roles, AWS Secrets Manager]

SecurityBoat Community
Connect, Learn & Grow!!

# A03 Injection

# spring SQLi

SecurityBoat Community
Connect, Learn & Grow!!

## Vulnerable Code

```java
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

@Repository
public class UserRepository {

    @Autowired
    private JdbcTemplate jdbcTemplate;

    public String getUserEmail(String username) {
        // Vulnerable SQL query with string concatenation
        String query = "SELECT email FROM users WHERE username = '" + username
+ "'";
        return jdbcTemplate.queryForObject(query, String.class);
    }
```

User input is directly used in SQL query (e.g., ' OR '1'='1) This query is then executed leading to SQL injection

## Secure Code

```java
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

@Repository
public class UserRepository {

    @Autowired
    private JdbcTemplate jdbcTemplate;

    public String getUserEmail(String username) {
        // Secure SQL query with parameterized input
        String query = "SELECT email FROM users WHERE username = ?";
        return jdbcTemplate.queryForObject(query, new Object[]{username},
String.class);
    }
}
```

- Parameterized queries ensure that user input is treated as data, not executable code.
- The database driver automatically escapes special characters, preventing injection.

# OS Command Injection

SecurityBoat Community
Connect, Learn & Grow!!

## Vulnerable Code

```java
import org.springframework.stereotype.Service;
import java.io.BufferedReader;
import java.io.InputStreamReader;

@Service
public class CommandService {

    public String executeCommand(String input) {
        try {
            // Vulnerable command execution
            Process process = Runtime.getRuntime().exec("ping " + input);
            BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStream()));
            StringBuilder output = new StringBuilder();
            String line;
            while ((line = reader.readLine()) != null) {
                output.append(line).append("\n");
            }
            return output.toString();
        } catch (Exception e) {
            throw new RuntimeException("Error executing command", e);
        }
    }
}
```

If input contains malicious commands (e.g., ; rm -rf /), it can execute arbitrary commands on the server.

## Secure Code

```java
import org.springframework.stereotype.Service;
import java.io.BufferedReader;
import java.io.InputStreamReader;

@Service
public class CommandService {

    public String executeCommand(String input) {
        // Validate input to allow only alphanumeric characters and dots
        if (!input.matches("[a-zA-Z0-9.]+")) {
            throw new IllegalArgumentException("Invalid input");
        }

        try {
            // Secure command execution with validated input
            Process process = Runtime.getRuntime().exec(new String[]{"ping", input});
            BufferedReader reader = new BufferedReader(new
InputStreamReader(process.getInputStream()));
            StringBuilder output = new StringBuilder();
            String line;
            while ((line = reader.readLine()) != null) {
                output.append(line).append("\n");
            }
            return output.toString();
        } catch (Exception e) {
            throw new RuntimeException("Error executing command", e);
        }
    }
}
```

- Input is validated to ensure it contains only allowed characters.
- The command is executed as an array of strings, preventing command chaining.

SecurityBoat Community
Connect, Learn & Grow!!

# A04 Insecure Design

SAST is unable to uncover this type of vulnerability

SecurityBoat Community
Connect, Learn & Grow!!

# A05 Security Misconfiguration

# Unrestricted File Upload Vulnerability

SecurityBoat Community
Connect, Learn & Grow!!

**Vulnerable Code**

- No File Type Validation: Attackers can upload malicious files (e.g., .php, .exe).
- No File Size Limit: Large files can cause DoS attacks.
- No File Name Sanitization: Malicious file names can cause issues on the server.
- No Virus Scanning: Malware can be uploaded and executed.

```php
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $target_dir = "uploads/";
    $target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
    $uploadOk = 1;

    // Check if file was uploaded
    if (isset($_POST["submit"])) {
        if (move_uploaded_file($_FILES["fileToUpload"]["tmp_name"], $target_file)) {
            echo "The file " . htmlspecialchars(basename($_FILES["fileToUpload"]["name"]))
. " has been uploaded.";
        } else {
            echo "Sorry, there was an error uploading your file.";
        }
    }
}
?>

<!DOCTYPE html>
<html>
<body>
    <form action="" method="post" enctype="multipart/form-data">
        Select file to upload:
        <input type="file" name="fileToUpload" id="fileToUpload">
        <input type="submit" value="Upload File" name="submit">
    </form>
</body>
</html>
```

# Unrestricted File Upload Vulnerability

Secure Code

SecurityBoat Community
Connect, Learn & Grow!!

```php
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $target_dir = "uploads/";
    $target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
    $uploadOk = 1;
    $fileType = strtolower(pathinfo($target_file, PATHINFO_EXTENSION));

    // Check if file is a valid upload
    if (isset($_POST["submit"])) {
        // Check file size (max 5MB)
        if ($_FILES["fileToUpload"]["size"] > 5000000) {
            echo "Sorry, your file is too large.";
            $uploadOk = 0;
        }

        // Allow only specific file types
        $allowedTypes = ["jpg", "jpeg", "png", "gif", "pdf"];
        if (!in_array($fileType, $allowedTypes)) {
            echo "Sorry, only JPG, JPEG, PNG, GIF, and PDF files are allowed.";
            $uploadOk = 0;
        }

        // Generate a random file name to prevent overwriting and malicious names
        $newFileName = uniqid() . "." . $fileType;
        $target_file = $target_dir . $newFileName;

        // Check if file already exists (unlikely due to unique name)
        if (file_exists($target_file)) {
            echo "Sorry, file already exists.";
            $uploadOk = 0;
        }

        // Move the file if all checks pass
        if ($uploadOk == 1) {
            if (move_uploaded_file($_FILES["fileToUpload"]["tmp_name"], $target_file)) {
                echo "The file " . htmlspecialchars(basename($_FILES["fileToUpload"]["name"])) . " has
been uploaded.";
            } else {
                echo "Sorry, there was an error uploading your file.";
            }
        }
    }
}
?>

<!DOCTYPE html>
<html>
<body>
    <form action="" method="post" enctype="multipart/form-data">
        Select file to upload:
        <input type="file" name="fileToUpload" id="fileToUpload">
        <input type="submit" value="Upload File" name="submit">
    </form>
</body>
</html>
```

File Type Validation:

- Only allow specific file types (e.g., jpg, jpeg, png, gif, pdf).
- Use pathinfo() to extract the file extension and validate it.

File Size Limit:

- Restrict file size to prevent DoS attacks (e.g., 5MB limit).

File Name Sanitization:

- Generate a unique file name using uniqid() to prevent overwriting and malicious file names.

File Existence Check:

- Ensure the file does not already exist (though unlikely due to unique naming).

Secure File Storage:

- Store uploaded files outside the web root directory if possible.
- Use .htaccess or server configuration to prevent execution of uploaded files.

# Error Handling Vulnerability

SecurityBoat Community
Connect, Learn & Grow!!

**Vulnerable Code**

```
using System;

public class InsecureErrorHandlingExample
{
    public void ProcessRequest()
    {
        try
        {
            // Simulate an error
            int result = 10 / int.Parse("0");
        }
        catch (Exception ex)
        {
            // Display detailed error message (INSECURE)
            Console.WriteLine("Error: " + ex.ToString());
        }
    }
}
```

- Detailed error messages (e.g., stack traces, file paths) are exposed to the user.
- Attackers can use this information to exploit vulnerabilities in the application.

**Secure Code**

```
using System;
using System.Diagnostics;

public class SecureErrorHandlingExample
{
    public void ProcessRequest()
    {
        try
        {
            // Simulate an error
            int result = 10 / int.Parse("0");
        }
        catch (Exception ex)
        {
            // Log the error securely
            LogError(ex);

            // Display a generic error message to the user
            Console.WriteLine("An error occurred. Please try again later.");
        }
    }

    private void LogError(Exception ex)
    {
        // Log the error to a secure file or logging service
        string logMessage = $"Error: {ex.Message}\nStack Trace: {ex.StackTrace}";
        Debug.WriteLine(logMessage); // Or use a logging framework like NLog or Serilog
    }
}
```

- Generic Error Messages: Users see a generic message, preventing exposure of sensitive information.
- Secure Logging: Errors are logged securely, either to a file or a logging service.
- No Stack Traces: Stack traces and other sensitive details are not displayed to the user.

# XXE/XEE Vulnerability

SecurityBoat Community
Connect, Learn & Grow!!

## Vulnerable Code

```php
<?php
// Simulate receiving XML input from a user
$xmlPayload = <<<XML
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [
    <!ENTITY xxe SYSTEM "file:///etc/passwd">
]>
<foo>&xxe;</foo>
XML;

// Load the XML payload
$dom = new DOMDocument();
$dom->loadXML($xmlPayload, LIBXML_NOENT | LIBXML_DTDLOAD);

// Extract data from the XML
echo $dom->textContent;
?>
```

- The LIBXML_NOENT and LIBXML_DTDLOAD options allow external entity loading.
- An attacker can inject a malicious external entity (e.g., file:///etc/passwd) to read sensitive files.

## Secure Code

```php
<?php
// Simulate receiving XML input from a user
$xmlPayload = <<<XML
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [
    <!ENTITY xxe SYSTEM "file:///etc/passwd">
]>
<foo>&xxe;</foo>
XML;

// Disable external entity loading
libxml_disable_entity_loader(true);

// Load the XML payload securely
$dom = new DOMDocument();
$dom->loadXML($xmlPayload, LIBXML_NOENT | LIBXML_DTDLOAD); // External entities are disabled

// Extract data from the XML
echo $dom->textContent;
?>
```

- libxml_disable_entity_loader(true) disables external entity loading, preventing XXE attacks.
- Even if the XML contains malicious external entities, they will not be processed.

SecurityBoat Community
Connect, Learn & Grow!!

# A06 Vulnerable & Outdated Components

SAST is unable to uncover this type of vulnerability

SecurityBoat Community
Connect, Learn & Grow!!

# A07 Identification & Authentication Failures

# Insecure Session Handling

SecurityBoat Community
Connect, Learn & Grow!!

**Vulnerable Code**

```
using System;
using System.Web;

public class InsecureSessionExample : IHttpHandler
{
    public void ProcessRequest(HttpContext context)
    {
        // Set session data without secure flags
        context.Session["UserId"] = 12345;

        // Send session ID in a non-secure cookie
        HttpCookie sessionCookie = new HttpCookie("ASP.NET_SessionId", context.Session.SessionID);
        context.Response.Cookies.Add(sessionCookie);

        context.Response.Write("Session data set.");
    }
    public bool IsReusable => false;
}
```

- Non-Secure Cookies: The session cookie is not marked as Secure or HttpOnly, making it accessible over non-HTTPS connections and via JavaScript.
- Session Fixation: The session ID is not regenerated after login, making it susceptible to session fixation attacks.
- No Expiration: The session does not have a timeout, increasing the risk of session hijacking.

**Secure Code**

```
using System;
using System.Web;

public class SecureSessionExample : IHttpHandler
{
    public void ProcessRequest(HttpContext context)
    {
        // Regenerate session ID to prevent session fixation
        context.Session.Abandon();
        context.Response.Cookies.Add(new HttpCookie("ASP.NET_SessionId", ""));

        // Set session data
        context.Session["UserId"] = 12345;

        // Configure secure session cookie
        HttpCookie sessionCookie = new HttpCookie("ASP.NET_SessionId", context.Session.SessionID)
        {
            Secure = true,    // Ensure the cookie is only sent over HTTPS
            HttpOnly = true,  // Prevent client-side script access to the cookie
            SameSite = SameSiteMode.Strict, // Prevent cross-site request forgery (CSRF)
            Expires = DateTime.Now.AddMinutes(20) // Set session timeout
        };
        context.Response.Cookies.Add(sessionCookie);

        context.Response.Write("Session data set securely.");
    }

    public bool IsReusable => false;
```

- Secure Cookies: The session cookie is marked as Secure and HttpOnly.
- Session Regeneration: The session ID is regenerated to prevent session fixation.
- Session Timeout: The session expires after 20 minutes of inactivity.
- SameSite Attribute: Prevents CSRF attacks by restricting cookie usage to same-site requests.

SecurityBoat Community
Connect, Learn & Grow!!

# A08 Software & Data Integrity Failures

# Insecure Deserialization Vulnerability

SecurityBoat Community
Connect, Learn & Grow!!

## Secure Code

```
const Ajv = require('ajv');
const ajv = new Ajv();

// Step 1: Define a schema for the expected data structure
const schema = {
    type: "object",
    properties: {
        name: { type: "string" },
        age: { type: "number" }
    },
    required: ["name", "age"],
    additionalProperties: false
};

// Step 2: Validate if the input is valid JSON
function isValidJSON(payload) {
    try {
        JSON.parse(payload);
        return true;
    } catch (e) {
        return false;
    }
}

// Step 3: Deserialize and validate the JSON payload
function safeDeserialize(jsonPayload) {
    // Validate if the payload is valid JSON
    if (!isValidJSON(jsonPayload)) {
        throw new Error("Invalid JSON payload");
    }

    // Deserialize the JSON payload
    const deserializedData = JSON.parse(jsonPayload);

    // Validate the deserialized data against the schema
    const validate = ajv.compile(schema);
    if (!validate(deserializedData)) {
        throw new Error("Invalid data structure: " +
JSON}stringify(validate.errors));

    return deserializedData;
}

// Example usage
try {
    const jsonPayload = '{"name":"John","age":30}';
    const deserializedData = safeDeserialize(jsonPayload);
    console.log("Deserialized Data:", deserializedData);
} catch (error) {
    console.error("Deserialization Error:", error.message);
}
```

## Vulnerable Code

```
const serialize = require('node-serialize');

// Simulate receiving serialized data from an untrusted source
const maliciousPayload = '{"rce":"_$$ND_FUNC$$_function(){require(\'child_process\').exec(\'rm -rf /\',
function(error, stdout, stderr) { console.log(stdout) });}()"}';

// Deserialize the payload (INSECURE)
const deserializedData = serialize.unserialize(maliciousPayload);
console.log(deserializedData);
```

Schema Definition: A schema is defined using ajv to specify the expected structure of the JSON data.

Input Validation:The isValidJSON function checks if the input is valid JSON using JSON.parse. If the input is not valid JSON, an error is thrown.

Deserialization and Schema Validation: The safeDeserialize function first validates the input as JSON.
It then deserializes the JSON payload using JSON.parse.

Error Handling:Errors are caught and logged, providing meaningful messages to the user without exposing sensitive details.

The node-serialize library allows deserialization of functions, which can execute arbitrary code.

An attacker can craft a malicious payload to execute commands on the server (e.g., rm -rf /).

SecurityBoat Community
Connect, Learn & Grow!!

# A09 Security Logging & Monitoring Failures

SAST is unable to uncover this type of vulnerability

SecurityBoat Community
Connect, Learn & Grow!!

# A10 Server-Side Request Forgery

# SSRF

SecurityBoat Community
Connect, Learn & Grow!!

## Vulnerable Code

```
const express = require('express');
const axios = require('axios');
const app = express();

app.get('/fetch', async (req, res) => {
    const url = req.query.url; // User-supplied URL

    try {
        const response = await axios.get(url); // Vulnerable to SSRF
        res.send(response.data);
    } catch (error) {
        res.status(500).send('Error fetching data');
    }
});

app.listen(3000, () => {
    console.log('Server running on port 3000');
});
```

- The server makes a request to a user-supplied URL (req.query.url).
- An attacker can provide a URL like http://localhost/admin or http://169.254.169.254/latest/meta-data/ to access internal resources.

## Secure Code

```
const express = require('express');
const axios = require('axios');
const app = express();
const { URL } = require('url');

// List of allowed domains
const ALLOWED_DOMAINS = ['example.com', 'trusted-domain.com'];

app.get('/fetch', async (req, res) => {
    const userUrl = req.query.url;

    try {
        // Parse and validate the URL
        const parsedUrl = new URL(userUrl);

        // Check if the domain is allowed
        if (!ALLOWED_DOMAINS.includes(parsedUrl.hostname)) {
            return res.status(400).send('Invalid URL');
        }

        // Fetch data from the validated URL
        const response = await axios.get(userUrl);
        res.send(response.data);
    } catch (error) {
        res.status(500).send('Error fetching data');
    }
});
```

- The URL is parsed and validated using the URL class.
- Only requests to allowed domains (ALLOWED_DOMAINS) are permitted.
- Prevents access to internal or unauthorized resources.

# Blind SSRF

SecurityBoat Community
Connect, Learn & Grow!!

## Vulnerable Code

```csharp
using System;
using System.Net.Http;
using System.Threading.Tasks;

public class VulnerableSSRF
{
    public static async Task Main(string[] args)
    {
        Console.WriteLine("Enter a URL to fetch:");
        string url = Console.ReadLine();

        using (HttpClient client = new HttpClient())
        {
            try
            {
                HttpResponseMessage response = await client.GetAsync(url);
                string content = await response.Content.ReadAsStringAsync();
                Console.WriteLine("Response received.");
            }
            catch (Exception ex)
            {
                Console.WriteLine("Error: " + ex.Message);
            }
        }
    }
}
```

- No URL Validation: The application does not validate or restrict the URL provided by the user.
- Internal Network Access: An attacker can provide a URL pointing to internal resources (e.g., http://localhost, http://169.254.169.254 for AWS metadata).

## Secure Code

- URL Whitelisting: Only URLs from a predefined list of trusted domains are allowed.
- URL Validation: The URL is validated using a regular expression to ensure it has a valid format (e.g., http:// or https://).
- Domain Extraction: The domain is extracted from the URL and checked against the whitelist.
- Error Handling: If the URL is not allowed, the application stops processing and displays an error message.

```csharp
using System;
using System.Net.Http;
using System.Text.RegularExpressions;
using System.Threading.Tasks;

public class SecureSSRF
{
    public static async Task Main(string[] args)
    {
        Console.WriteLine("Enter a URL to fetch:");
        string url = Console.ReadLine();

        // Validate the URL
        if (!IsUrlAllowed(url))
        {
            Console.WriteLine("Error: URL is not allowed.");
            return;
        }

        using (HttpClient client = new HttpClient())
        {
            try
            {
                HttpResponseMessage response = await client.GetAsync(url);
                string content = await response.Content.ReadAsStringAsync();
                Console.WriteLine("Response received.");
            }
            catch (Exception ex)
            {
                Console.WriteLine("Error: " + ex.Message);
            }
        }
    }

    // Validate the URL to prevent SSRF
    private static bool IsUrlAllowed(string url)
    {
        // Define a whitelist of allowed domains
        string[] allowedDomains = { "example.com", "trusted-domain.com" };

        // Use regex to extract the domain from the URL
        var match = Regex.Match(url, @"^https?://([^/]+)");
        if (!match.Success)
        {
            return false; // Invalid URL format
        }

        string domain = match.Groups[1].Value;

        // Check if the domain is in the whitelist
        foreach (string allowedDomain in allowedDomains)
        {
            if (domain.EndsWith(allowedDomain, StringComparison.OrdinalIgnoreCa
            {
                return true;
            }
        }

        return false; // Domain is not allowed
    }
}
```

SecurityBoat Community
Connect, Learn & Grow!!

# Best Practices

# Secure Code Guideline

SecurityBoat Community
Connect, Learn & Grow!!

## 1. Sanitize and validate all input

Consider all input potentially malicious and sanitize it accordingly using a vetted library. Next to direct user input think of:

- data streams
- events
- files
- cookies
- system properties
- command line parameters

## 2. Never store credentials as code/config

Some good practices:

- Block sensitive data being pushed to GitHub by adding git-secrets, or its likes, as a git pre-commit hook.
- Break the build using the same tools.
- Audit for slipped secrets with GitRob or truffleHog.
- Use ENV variables for secrets in CI/CD and secret managers like Vault in production.

## 3. Enforce the least privilege principle

People and automated processes should only have access to the data they actually need. Test if a module can perform operations they're not entitled to perform.

## 4. Enforce secure authentication

- Assume they're not who they say they are.
- Enforce TLS and TLS client authentication.
- Re-authenticate before sensitive operations.
- Enforce password complexity.

## 5. Test for new vulnerabilities in your OS app dependencies

Check your dependencies for known issues and don't introduce new vulnerabilities — implement tests on your local machine and connect your git repository. Snyk helps you with this by providing tooling to scan throughout every stage of your SDLC.

## 6. Handle sensitive data with care

- Only store the data you need.
- Encrypt data that is sensitive, for example, PII and financial data.
- Use the correct strong encryption algorithm.
- Transport sensitive data only over TLS.
- Check your cookies and session data for sensitive information.

## 7. Protect against well-known attacks

Know how common attacks — like SQL injections and Cross-site scripting (XSS) — work and take that with you in a review. Review the OWASP Top 10 vulnerabilities and learn how to spot them.

## 8. Statically test your source code

Use a Static Application Security Testing (SAST) tool — like Snyk Code — to find security issues in your code. It is recommended to automate these checks as part of your pipeline or build process.


**Brian Vermeer**
@BrianVerm
Developer Advocate at Snyk

**Trisha Gee**
@trisha_gee
Java Champion & Developer Advocate at JetBrains


www.snyk.io

SecurityBoat Community
Connect, Learn & Grow!!

# Secure Code Review Process

### Identify Vulnerabilities

Look for security flaws and misconfigurations

Review Business Logic

### Code Analysis

Manual review by security analyst

Automated scanning using tools


### Report Findings & Remediation

Document issues with severity levels

Prioritize Issues

Fix vulnerabilities and retest

### Preparation

Define scope and objectives

Establish Review Criteria

Gather tools and documentation

### Verification

Confirm fixes and ensure no new issues are introduced

Train Developers

Incorporate Feedback

# Code Review Best Practices for Code Reviewers

SecurityBoat Community
Connect, Learn & Grow!!

01 Focus on core issues; do less nit-picking

02 Talk in person if necessary

03 Document code review decisions

04 Integrate code review into your daily routine

05 Give respectful and constructive feedback

06 Reduce context switching as it kills productivity

07 Start by reviewing test code first

08 Use code review checklists

09 Fight code review bias

10 Explain your point of view

11 **Review Efficiency:** Timebox Reviews, Automate where possible

12 **Follow Up:** Verify fixes, Learn and improve

SecurityBoat Community
Connect, Learn & Grow!!

# Conclusion

SecurityBoat Community
Connect, Learn & Grow!!

# CONCLUSION

01 How you Inspect the data

02 Understand the source code

03 How is it protected

**Your Apps & Data Are Not Secure Until You Ask These Questions.**

SecurityBoat Community

Connect, Learn & Grow!!

# Thank You For Your Attention!

Do you have any questions?

# STAY CONNECTED

Thank you for taking the time to connect with us! We're eager to hear your thoughts and keep the conversation going. For the latest updates on security, follow us and stay engaged through our social media handles below

https://medium.com/@appsecwarrior

info@appsecwarrior.org

SecurityBoat Community
Connect, Learn & Grow!!

PRASAD K

Product Security Consultant at Hitachi-GlobalLogic