OWASP Find Security Bugs
The community static code analyzer
Agenda

- Introduction to Find Security Bugs
  - Why use it?
  - How does it work?
- Integrations
- “Hidden” features
- Vulnerabilities found
- Conclusion
Who I am

Philippe Arteau
Security Researcher at GoSecure

Past experiences:
- Developer
- Pentester
- Security Code Review

Open-source developer
- Find Security Bugs (SpotBugs - Static Analysis for Java)
- Burp and ZAP Plugins (Retire.js, CSP Auditor, Request Reissue Scripter) Security Code Scan (Roslyn – Static Analysis for .NET)
Introduction
Find Security Bugs in a nutshell

• Detectors built around the SpotBugs engine with a focus on security issues
• Open-source
• OWASP project since 2019
• 131 bug patterns
• Works great with Java, Kotlin and JSP
  • Works ok with Groovy and Scala
How does it work?
Vulnerability types

- SQL/HQL Injection
- Command Injection
- Cryptography Weaknesses
- Cross-Site Scripting
- Path Traversal
- Template Injection
- Hard Coded Password
- Insecure Configuration
- XML External Entity
- Predictable Random Generator
Advantages

• High code coverage
• Source code level identification
• Help find vulnerabilities early in the SDLC
• Consistency

Disadvantages

• Does not cover:
  • Logic flaws
  • Sensitive information leakage
  • Production configuration
• Technology specific
• False positive (Potential vulnerabilities only)
Integration
Integration In IDE

- IntelliJ
- Eclipse
- NetBeans
Continuous Integration

Try
{
    String userInput = s.getParameter(NAME, "");
    if (userInput.equals(""))
    {
        userInput = SELECT_ST + userInput;
    }
    String sql = userInput.replaceAll("\s+", "");
    Connection conn = DatabaseUtilities.getConnections().orElseThrow()
    Statement statement = conn.createStatement()
    ResultSet rs = statement.executeQuery(sql);
    if (rs.getRow() == 2)
    {
        statement.executeUpdate(sql1);
    }

This use of prepared Statement.executeUpdate(String sql) can be vulnerable to SQL injection.

The input values included in SQL queries need to be passed in safely. Bind variables in prepared statements can be used to easily mitigate the risk of SQL injection.

Vulnerable Code:

Connection conn = [...];
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("UPDATE OFFERS set OFFER_ID = " + values + " where OFFER_NAME = " + "defaultOffer")

Solution:

Connection conn = [...];
conn.prepareStatement("UPDATE OFFERS set OFFER_ID = ? where OFFER_NAME = ?");
updateValues.setInt(1, 93.50);
updateValues.setString(2, "defaultOffer");

References (JDBC)

Oracle Documentation, Java Tutorials > Prepared Statements
References (SQL injection)

WASC-IS: SQL Injection
OWASP Top 10 2013A Injection
OWASP: SQL Injection Prevention Cheat Sheet
OWASP: Query Paramertization Cheat Sheet

getPasswordCracker(s), setStage(2);
Continuous Integration

Many free and open-source options
- SonarQube (with Sonar-FindBugs)
- Jenkins (with Warnings-NG)

Integrated in many commercial solutions
- Gitlab
- CodeDX
Demonstration

Scanning the WebGoat project with Spotbugs
In Continuous Integration
Hidden Features

Much more than source code scanning...
Analyzing compiled libraries

Allows rapid assessment of potential risks
  • Does not require original source code

Able to scan classes from:
  • Android APK files (dex to jar required)
  • WAR or EAR files

```
findsecbugs.bat -html -output report.htm third-party-lib.jar
```
Scanning without build configuration

Complex builds are common in large enterprises

The code reviewer can end up with

- Missing dependencies or dependencies hosted on a private repository
- Custom build steps
- Use of a proprietary tool

Solution:

Ask the developer to provide pre-built code
Import inside IntelliJ (No need to recompile it)
Vulnerabilities Found
Struts CSRF Token Prediction
CVE-2014-7809
Code sample from Struts 2.3.17

```java
public class TokenHelper {

    private static final Random RANDOM = new Random();

    public static String getToken( String tokenName ) {
        String token = generateGUID();
        setSessionToken(tokenName, token);
        return token;
    }

    public static String generateGUID() {
        return new BigInteger(165, RANDOM).toString(36).toUpperCase();
    }
}
```
Struts 2.3.17: FSB report

Predictable pseudorandom number generator

The use of a predictable random value can lead to vulnerabilities when used in certain security critical contexts. For example, when the value is used as:

- a CSRF token: a predictable token can lead to a CSRF attack as an attacker will know the value of the token
- a password reset token (sent by email): a predictable password token can lead to an account takeover, since an attacker will guess the URL of the ‘change password’ form
- any other similar value

A quick fix could be to replace the use of `java.util.Random` with something stronger, such as `java.security.SecureRandom`.

Vulnerable Code:

```java
String generateSecretToken() {  
    Random r = new Random();  
    return long.toString(r.nextLong());  
}
```

Solution:

```java
import org.apache.commons.codec.binary.Hex;
String generateSecretToken() {  
    SecureRandom secRandom = new SecureRandom();  
    byte[] result = new byte[32];  
    secRandom.nextBytes(result);  
    return Hex.encodeHexString(result);  
}
```

References:
- Cracking Random Number Generators - Part 1 (http://joezd.id.au)
- CRIT-MC02:1 Generating strong random numbers
- CRIT-X00: Use of Insufficiently Random Values
- Predicting Struts CSRF Token (Example of real-life vulnerability and exploitation)

public class TokenHelper {

    private static final Random RANDOM = new Random();

    public static String setToken(String tokenName) {
        String token = generateGUID();
        setSessionToken(tokenName, token);
        return token;
    }

    public static String generateGUID() {
        return new BigInteger(165, RANDOM).toString(36).toUpperCase();
    }
}
Java PRNG (java.util.Random)
DerbyDB XXE
CVE-2015-1832
Code sample from DerbyDB 10.12.1.1

```java
/**
 * Fault in the list of rows.
 */

private void readRows() throws Exception {
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    _builder = factory.newDocumentBuilder();
    Document doc = _builder.parse(_xmlResource);
    Element root = doc.getDocumentElement();

    _rawRows = root.getElementsByTagName(_rowTag);
    _rowCount = _rawRows.getLength();

    _xmlResource.close();
}

https://apache.googlesource.com/derby/+/6f55de19d898430fec96d3041a03b25fd218454f/java/engine/org/apache/derby/vti/XmlVTI.java
DerbyDB 10.12.1.1: Exploitation

Impact: Privilege escalation from basic SQL access to file access and directory listing
Spring Expression Language (SPEL) injection
CVE-2018-1273
public void setPropertyValue(String propertyName, @Nullable Object value) throws BeansException {
    if (!isWritableProperty(propertyName)) {
        throw new NotWritablePropertyException(type, propertyName);
    }
    StandardEvaluationContext context = new StandardEvaluationContext();
    context.addPropertyAccessor(new PropertyTraversingMapAccessor(type, conversionService));
    context.setTypeConverter(new StandardTypeConverter(conversionService));
    context.setRootObject(map);
    Expression expression = PARSER.parseExpression(propertyName);
Expected property path:
property1.property2

- In practice:
  - property[0].property
  - property[code()].property

```
@Override
public boolean isWritableProperty(String propertyName) {
    try {
        return getPropertyPath(propertyName) != null;
    } catch (PropertyReferenceException e) {
        return false;
    }
}
```

```
private PropertyPath getPropertyPath(String propertyName) {
    String plainPropertyPath = propertyName.replaceAll("\[\.*\\\]", "");
    return PropertyPath.from(plainPropertyPath, type);
}
```
Spring Data Commons: Exploitation

Request
POST /users?size=5 HTTP/1.1
Host: localhost:3001
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:57.0) Gecko/20100101 Firefox/57.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 110
Cookie: RememberMe=284a3e01e46e47b4c8c7c69f83b7a773d0c31e908f0f10801e789ce80

Password: abc

Response
Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fall

Calculator

Standard

0

Math

7 8 9 ×

4 5 6 −

1 2 3 +

± 0 . =

[...]

password[T(java.lang.Runtime).getRuntime().exec("calc")]=abc
For more vulnerabilities found with Find Security Bugs

- JavaScript Injection
- XSLT Injection
- Deserialization
- Command injection / Ghostscript
Lessons learned (What worked)

Unit testing is key for a static code analysis tool
- Regression tests with samples for every detector and heuristic
- Make test cases easy to write with DSL

Documentation
- Code has to be obvious (naming, structure, comments)
- Developer guide to contribute

Find existing tool before building a new one
- Shopping for existing frameworks
How to contribute?

Code contribution
- Bug fixes
- New vulnerability patterns
- Code samples for new bug patterns

Help others
- Answer question on StackOverflow [find-sec-bugs] and [spotbugs]

Improve the documentation
- Improve the English descriptions
- (If really really motivated) Translate descriptions
Different language different OS tool

- **Brakeman**: Ruby
- **SonarSource**: Java, PHP, ...
- **Bandit**: Python
- **.NET Security Code Scan**: C#, VB.net
- **NuGet package**
- **Visual Studio extension**
Questions?

- parteau@gosecure.ca
- @GoSecure_Inc
- @h3xStream

https://www.gosecure.net/blog
Find Security Bugs related


SpotBugs website: [https://spotbugs.github.io/](https://spotbugs.github.io/)

SonarQube plugin [https://github.com/spotbugs/sonar-findbugs](https://github.com/spotbugs/sonar-findbugs)
Vulnerabilities found


XXE in DebyDB: https://issues.apache.org/jira/browse/DERBY-6807

Spring Data Commons Vulnerability: https://www.gosecure.net/blog/2018/05/15/beware-of-the-magic-spell-part-1-cve-2018-1273