Finding Bugs in Software
(Static Analysis)

Object Oriented Software Engineering
Lecture 5

Dr. Graham McDonald
graham.mcdonald@glasgow.ac.uk

Outline

- Introduction to debugging techniques
- What is a Bug Pattern?
- Automated analysis
  - Soundness
  - Precision
- The architecture of a debugger

Debugging

- Debugging Technique:
  - **Threat Modeling**: Look at design, write out and/or diagram what could go wrong.
  - **Manual** code reviews and inspection
  - **Automated Tools**, e.g., static analysis

What is a Bug Pattern:

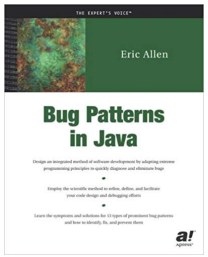
Definition: (*Bug Pattern*)

Bug patterns are recurring correlations between signalled errors and underlying bugs in a program.

What is a Bug Pattern:

- Common pitfalls of a programming language that are documented so that developers can learn to avoid them.

- Challenge:
  - How to identify them
  - How to treat them,
  - and how to prevent them.



Example:

- Bug- **Predictable random number generator**:

Vulnerable Code:

```
String generateSecretToken() {
    Random r = new Random();
    return Long.toHexString(r.nextLong());
}
```

Example:

- Bug- **Predictable random number generator**:

Solution:

```
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);
}
```

Replace the use of **java.util.Random** with something stronger, such as **java.security.SecureRandom**.

Example:

- Bug- **Using Object Deserialization**:

  - Object deserialization of untrusted data can lead to remote code execution.
  - Deserialization is a sensible operation that has a great history of vulnerabilities.

Example:

- Bug- **Using Object Deserialization**:

Code at risk:

```
public UserData deserializeObject(InputStream receivedFile) throws
IOException, ClassNotFoundException {
    try (ObjectInputStream in = new ObjectInputStream(receivedFile)) {
        return (UserData) in.readObject();
    }
}
```

Example:

- Bug- **Using Object Deserialization**:

Solution:

- Avoid deserializing object provided by remote users.
- If deserialization of objects from remote users cannot be avoided, then ensure that **domain input validation** is robustly applied for sanitisation.

Example:

- Bug- **Trust Boundary Violation**:
  - A trust boundary can be thought of as line drawn through a program.
  - On one side of the line, data is untrusted. On the other side of the line, data is assumed to be trustworthy.
  - The purpose of validation logic is to allow data to safely cross the trust boundary - to move from untrusted to trusted.
  - A trust boundary violation occurs when a program blurs the line between what is trusted and what is untrusted.

Example:

- Bug- **Trust Boundary Violation**:
  - The following code accepts an HTTP request and stores the username parameter in the HTTP session object before checking to ensure that the user has been authenticated.

Code at risk:

```
usrname = request.getParameter("usrname");
if (session.getAttribute(ATTR_USR) == null) {
        session.setAttribute(ATTR_USR, usrname);
}
```

Example:

- Bug- **Trust Boundary Violation**:
  - The following code accepts an HTTP request and stores the username parameter in the HTTP session object before checking to ensure that the user has been authenticated.

Code at risk:

```
usrname = request.getParameter("usrname");
if (session.getAttribute(ATTR_USR) == null) {
        session.setAttribute(ATTR_USR, usrname);
}
```

Solution:
Add validation prior to setting a new session attribute. When possible, prefer data from safe location rather than using direct user input.

Examples

Bug- **Null Pointer Exception**:

- Eclipse 3.0.1, org.eclipse.update.internal.core.ConfiguredSite

```
if (in == null)
    try {
        in.close();
    } catch (IOException e1) {
    }
```

Examples

Bug- **Infinite recursion** :

- J2SE version 1.5 build 63 (released version), java.lang.annotation.AnnotationTypeMismatchException

```
public String foundType() {
   return this.foundType();
}
```

Bug Patterns

- Not all bugs are subtle and unique
- Many bugs share common characteristics
- A bug pattern is a code idiom that is usually a bug
  - Detection of many bug patterns can be automated using simple analysis techniques

Finding Bugs in Software

(Static Analysis)

Object Oriented Software Engineering

Lecture 5: Part 2

Dr. Graham McDonald

graham.mcdonald@glasgow.ac.uk

Bug Patterns

- Categories
  - **Correctness** of the program
  - Not conforming to best practice (a **bad practice**)
  - **Internationalization** problems
  - Malicious code **vulnerability**
  - **Multithreaded** correctness
  - **Performance**
  - Security
  - Dodgy code
  - etc

http://findbugs.sourceforge.net/bugDescriptions.html

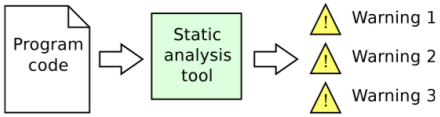
Code Inspection

- Manually examine source code to look for bugs
- Limitations:
  - Labor intensive
  - Subjective: Source code might appear to be correct when it is is not.
    - Can you spot the typo in this slide?
    - People have similar blind spots reading source code

Debugger

- A debugger is a special program used to **analyse** other programs in order to find bugs.
- A debugger is used to automatically detect **bug patterns** in program code

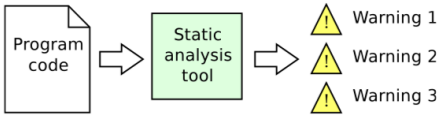
Debugger



Idea:

- Use a program to analyse your program for bugs
- Analyse statements, control flow, method calls, etc

Debugger



Advantages over testing and manual code inspection:

- Can analyse many potential program behaviours
- Doesn't get bored
- Relatively objective

Can Automated Program Analysis Work?

"Everything interesting about the behaviour of programs is undecidable"

- H. G. Rice [1953]

- In general, we can't tell whether a program P has some property Q.
- Instead: Approximate Q in analysis of P

The limits of static analysis

(The Halting Problem)

1. Does program P have bug X?"
2. Can program P reach state X?"

Soundness

- A bug detection system is sound if whenever there is a bug in the program an alert is raised
- Unsound means the bug detection system can generate false negative outputs

Precision

- A bug detection system is precise if every bug alert in the program is actually a bug
- Imprecise means the bug detection system can generate false positive outputs

Program Analysis Trade-offs

- Generally, most program analysis are conservative (i.e they are sound and imprecise)
- But, the detection of bugs in a program is an approximation that involves trade-off between **soundness**, **precision** and **execution time**

Approximation towards completeness

Challenge:

- Can a bug detection system be designed such that it always overestimate possible program behaviours
  - Never misses a bug, but might report some false warnings
- Problem: The analysis may report so many false warnings that the real bugs cannot be found!

Approximation towards Soundness

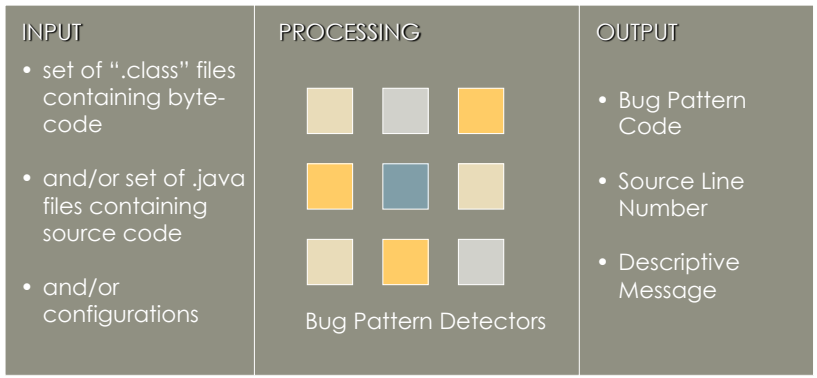
Challenge:

- Can a bug detection system be designed such that it always underestimate possible program behaviours
  - Never reports a false warning, but might miss some real bugs
- Problem: The analysis may not find as many bugs as we would like

Balanced Approximation

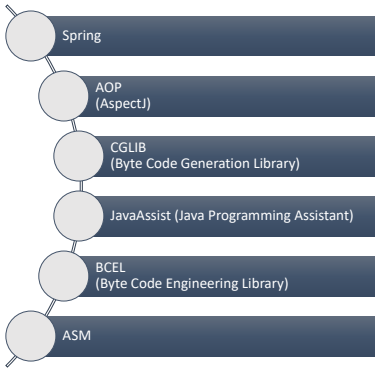
- A static analysis to find bugs does not need to be consistent in its approximations
  - Neither sound nor complete: miss some real bugs and report some false warnings
- This gives the analysis more flexibility to estimate likely program behaviours

The Architecture of a Debugger



Debugger: Byte Code Input

- Frameworks for byte code analysis



Debugger: Source Code Input

- Frameworks for source code analysis

Java Parser: A Java Parser with AST generation and visitor support
(https://github.com/javaparser/javaparser)