



JavaOne

Improving Software Quality with Static Analysis

William Pugh

Professor
Univ. of Maryland
<http://www.cs.umd.edu/~pugh>

TS-2007



You will believe...

Static analysis tools can find real bugs and real issues in your code.

You can and should effectively incorporate static analysis into your software development process.



Agenda

Introduction

Correctness issues

Bad Practice

Security defects

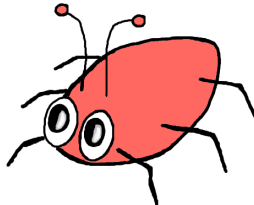

Demos (FindBugs, Fortify SCA)

Integrating static analysis

Wrap up



About Me

- Professor at Univ. of Maryland since 1988, doing research in programming languages, algorithms, software engineering
- Technical Lead on JSR-133 (Memory model), JSR-305 (Annotations for Software Defect Detection)
- Founder of the FindBugs™ project 
 - Open source static analysis tool for defect detection in the Java™ Programming Language
- Technical advisory board of 



Agenda

Introduction

Correctness issues

Bad Practice

Security defects

Demos (FindBugs, Fortify SCA)

Integrating static analysis

Wrap up

Static Analysis

- Analyzes your program without executing it
- Doesn't depend on having good test cases
 - or even any test cases
- Generally, doesn't know what your software is *supposed* to do
 - Looks for violations of reasonable programming
 - Shouldn't throw NPE
 - Shouldn't allow SQL injection
- Not a replacement for testing
 - Very good at finding problems on untested paths
 - But many defects can't be found with static analysis



Common Wisdom about Bugs and Static Analysis

- Programmers are smart
- Smart people don't make dumb mistakes
- We have good techniques (e.g., unit testing, pair programming, code inspections) for finding bugs early
- So, bugs remaining in production code must be subtle, and finding them must require sophisticated static analysis techniques
 - I tried `lint` and it sucked: lots of warnings, few real issues

Can You Find The Bug?

```
if (listeners == null)
    listeners.remove(listener);
```

- JDK1.6.0, b105, sun.awt.x11.XMSelection
 - lines 243-244

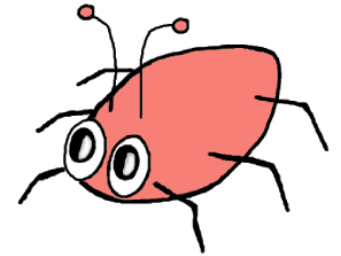
Why Do Bugs Occur?

- Nobody is perfect
- Common types of errors:
 - Misunderstood language features, API methods
 - Typos (using wrong boolean operator, forgetting parentheses or brackets, etc.)
 - Misunderstood class or method invariants
- Everyone makes syntax errors, but the compiler catches them
 - What about bugs one step removed from a syntax error?

Who Uses Static Analysis?

- Lots and lots of projects and companies
- Among many others, Glassfish and Google use FindBugs
 - Many companies are weird about letting you say they use your open source tool
- Lots of open source tools: PMD, CheckStyle, etc.
- IDEs include some: Eclipse, IntelliJ, Netbeans
- Commercial tools available from Fortify Software, KlocWork, Coverity, Parasoft, SureLogic
- Static analysis used even more widely/intensely for C/C++
 - More bugs to find
 - Bugs a lot scarier
 - Free tools not as good

FindBugs



- I'm mostly going to be talking about FindBugs
 - I know it best
- Some things will be specific to FindBugs
 - What we classify as a "correctness" issue
 - Which potential null pointer issues we report
- But most of the concepts apply to other tools

Bug Categories

Selected categories for today's discussion

- Correctness - the code seems to be clearly doing something the developer did not intend
- Bad practice - the code violates good practice
- Security defect
 - Vulnerability to malicious code
 - Vulnerability to malicious input
 - SQL injection, cross site scripting

Bug Patterns

- Some big, broad and common patterns
 - Dereferencing a null pointer
 - An impossible checked cast
 - Methods whose return value should not be ignored
- Lots of small, specific bug patterns, that together find lots of bugs
 - Every Programming Puzzler
 - Every chapter in *Effective Java*
 - Most postings to <http://thedailywtf.com/>

Analysis Techniques

Whatever you need to find the bugs

- Local pattern matching
 - If you invoke `String.toLowerCase()`, don't ignore the return value
- Intraprocedural dataflow analysis
 - Null pointer, type cast errors
- Interprocedural method summaries
 - This method always dereferences its parameter
- Context sensitive interprocedural analysis
 - Interprocedural flow of untrusted data
 - SQL injection, cross site scripting

Categories, ranking, use cases

- Every tool has categories, rules/patterns, priorities
- You can generally customize what you want to look at
- Sometimes, you want to do a code audit of a newly written module with 1,000 lines of code
 - and sometimes you want to scan 1,000,000 lines of code that has been in production for a year
- Different use cases require different tunings, different tools



Agenda

Introduction

Correctness issues

Bad Practice

Security defects

Demos (FindBugs, Fortify SCA)

Integrating static analysis

Wrap up

Correctness issues

Stuff you *really* want to look at

- In FindBugs, we reserve the Correctness category for issues we are most confident are wrong
 - code does something the developer didn't intend
- Many of the other categories reflect correctness issues
- But correctness issues are the things we think you should look at when scanning that million line code base
- low false positive rate, few low impact bugs

Infinite recursive loop

... Students are good bug generators

- Student came to office hours, was having trouble with his constructor:

```
/** Construct a WebSpider */  
public WebSpider() {  
    WebSpider w = new WebSpider();  
}
```

- A second student had the same bug
- Wrote a detector, found 3 other students with same bug

Double Check Against JDK1.6.0-b13

- Found 5 infinite recursive loops
- Including one written by Joshua Bloch

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

- Smart people make dumb mistakes
 - 27 across all versions of JDK, 31 in Google's Java code
- Embrace and fix your dumb mistakes



Finding Null Pointer Bugs with FindBugs

- FindBugs looks for a statement or branch that, if executed, guarantees a null pointer exception
- Either a null pointer exception could be thrown, or the program contains a statement/branch that can't be executed
- Could look for exceptions that only occur on a path
 - e.g., if the condition on line 29 is true and the condition on line 38 is false, then a NPE will be thrown
 - but would need to worry about whether that path is feasible



Null Pointer Bugs Found by FindBugs

JDK1.6.0-b105

- 109 statements/branches that, if executed, guarantee NPE
 - We judge at least 54 of them to be serious bugs that could generate a NPE on valid input
- Most of the others were deemed to be unreachable branches or statements, or reachable only with erroneous input
 - Only one case where the analysis was wrong



Examples of null pointer bugs

simple ones

```
//com.sun.corba.se.impl.naming.cosnaming.NamingContextImpl
```

```
if (name != null || name.length > 0)
```

```
//com.sun.xml.internal.ws.wSDL.parser.RuntimeWSDLParser
```

```
if (part == null | part.equals(""))
```

```
// sun.awt.x11.ScrollPanePeer
```

```
if (g != null)  
    paintScrollBars (g, colors) ;  
g.dispose () ;
```



Redundant Check For Null

Also known as a reverse null dereference error

- Checking a value to see if it is null
 - When it can't possibly be null

```
// java.awt.image.LoopupOp, lines 236-247
```

```
public final WritableRaster filter(  
    Raster src, WritableRaster dst) {  
    int dstLength = dst.getNumBands();  
    // Create a new destination Raster,  
    // if needed  
    if (dst == null)  
        dst = createCompatibleDestRaster(src);  
}
```

Redundant Check For Null

Is it a bug or a redundant check?

- Check the JavaDoc for the method
- Performs a lookup operation on a **Raster**.
 - If the destination **Raster** is **null**,
 - a new **Raster** will be created.
- Is this case, a bug
 - particularly look for those cases where we know it can't be null because there would have been a NPE if it were null

Bad Method Invocation

- Methods whose return value shouldn't be ignored
 - Strings are immutable, so functions like `trim()` and `toLowerCase()` return new String
- Dumb/useless methods
 - Invoking `toString` or `equals` on an array
- Lots of specific rules about particular API methods
 - Hard to memorize, easy to get wrong

Examples of bad method calls

```
// com.sun.rowset.CachedRowSetImpl  
if (type == Types.DECIMAL || type == Types.NUMERIC)  
    ((java.math.BigDecimal)x).setScale(scale);
```

```
// com.sun.xml.internal.txw2.output.XMLWriter  
try { ... }  
catch (IOException e) {  
    new SAXException("Server side Exception:" + e);  
}
```

Type Analysis

- Impossible checked casts
- Useless calls
 - `equals` takes an `Object` as a parameter
 - but comparing a `String` to `StringBuffer` with `equals(...)` is pointless, and almost certainly not what was intended
 - `Map<K, V>.get` also takes an `Object` as a parameter
 - supplying an object with the wrong type as a parameter to `get` doesn't generate a compile time error
 - just a `get` that always returns null



Lots of Little Bug Patterns

- checking if `d == Double.NaN`
- Bit shifting an `int` by a value greater than 31 bits
- Every Puzzer this year
 - more than half for most years



When Bad Code Isn't A Bug

- Static analysis tools will sometimes find ugly, nasty code
 - that can't cause your application to misbehave
- Cleaning this up is a good thing
 - makes the code easier to understand and maintain
- But for ugly code already in production
 - sometimes you just don't want to touch it
- We've found more cases like this than we expected



When Bad Code Isn't A Bug

bad code that does what it was intended to do

```
// com.sun.jndi.dns.DnsName, lines 345-347
```

```
if (n instanceof CompositeName) {  
    // force ClassCastException  
    n = (DnsName) n;  
}
```

```
// sun.jdbc.odbc.JdbcOdbcObject, lines 85-91
```

```
if ((b[offset] < 32) || (b[offset] > 128)) {  
    asciiLine += ".";  
}
```



When Bad Code Isn't A Bug

Code that shouldn't go wrong

```
// com.sun.corba.se.impl.dynamicany.DynAnyComplexImpl
String expectedMemberName = null;
try {
    expectedMemberName
        = expectedTypeCode.member_name(i);
} catch (BadKind badKind) { // impossible
} catch (Bounds bounds) { // impossible
}
if ( !(expectedMemberName.equals(memberName) ... ) ) {
```



When Bad Code Isn't A Bug

When you are already doomed

```
// com.sun.org.apache.xml.internal.security.encryption.XMLCipher
```

```
// lines 2224-2228
```

```
if (null == element) {
```

```
    //complain
```

```
}
```

```
String algorithm = element.getAttributeNS(...);
```




Overall Correctness Results From FindBugs

Evaluating Static Analysis Defect Warnings On Production Software, ACM 2007 Workshop on Program Analysis for Software Tools and Engineering

- JDK1.6.0-b105
 - 379 correctness warnings
 - we judge that at least 213 of these are serious issues that should be fixed
- Google's Java codebase
 - over a 6 month period, using various versions of FindBugs
 - 1,127 warnings
 - 807 filed as bugs
 - 518 fixed in code



Agenda

Introduction

Correctness issues

Bad Practice

Security defects

Demos (FindBugs, Fortify SCA)

Integrating static analysis

Wrap up

Bad Practice

- A class that defines an `equals` method but inherits `hashCode` from `Object`
 - Violates contract that any two equal objects have the same hash code
- `equals` method doesn't handle null argument
- `Serializable` class without a `serialVersionUID`
- Exception caught and ignored
- Broken out from the correctness category because I never want a developer to yawn when I show them a "correctness" bug

Fixing hashCode

- What if you want to define `equals`, but don't think your objects will ever get put into a `HashMap`?
- Suggestion:

```
public int hashCode() {  
    assert false  
        : "hashCode method not designed";  
    return 42;  
}
```

Use of Unhashable Classes

- FindBugs previously reported all classes that defined `equals` but not `hashCode` as a correctness problem
 - but some developers didn't care
- Now reported as bad practice
 - but separately report use of such a class in a `HashMap/HashTable` as a correctness warning



Agenda

Introduction

Correctness issues

Bad Practice

Security defects

Demos (FindBugs, Fortify SCA)

Integrating static analysis

Wrap up

Security defects

- Vulnerability to untrusted, malicious code
 - Do you have any public static non-final fields?
- Vulnerability to untrusted, malicious input
 - Can untrusted input, perhaps from user of a web application, force your program to things it shouldn't?



JavaOne

Vulnerability to Untrusted, Malicious Code

JDK1.6.0-b105

- 220 mutable public static fields
 - non-final fields
 - final references to mutable objects (e.g., arrays)
- 327 methods that return references to mutable internal components (e.g., arrays, **Date**)
 - caller can then change internal state
- 311 methods that take references to mutable objects as parameters and make them part of the internal state

Why haven't these been fixed?

Any untrusted applet can change the static fields

- Sun's security team is aware of the issue
 - They warn against public static non-final fields
 - TS-2594 - Secure Coding Guidelines, Continued: Preventing Attacks and Avoiding Antipatterns
- They say they will try to address it in JDK7
 - We'll see; hold their feet to the fire on this one.
- Is backwards compatibility a problem?
 - "We can't make `javax.swing.DefaultListCellRenderer.noFocusBorder` final, because some code might depend upon being able to change it?"
 - Some code deserves to be broken



Vulnerability to untrusted, malicious input

- Be glad you aren't working in C/C++
- But still lots of issues to be worried about
 - SQL Injection
 - Cross site scripting (XSS) - getting to be big issue
 - HTTP Response splitting
 - Path traversal
- If you write network facing code, and aren't worried/paranoid about these issues, you are being irresponsible

SQL Injection

- Forming SQL queries using string concatenation

```
String query = "SELECT cc_type, cc_number FROM "  
    + "user_data WHERE last_name = '" + user + "'";
```

- Can usually avoid by using SQL prepared statements *with constant Strings*
- Can just look for non-constant SQL query strings, or look deeper to find sources of data used to build query strings

Cross-Site Scripting

- Untrusted input from user included verbatim in HTML response
- Can be exploited by crafting a URL that a victim clicks on
- Generates a response from your web site
- that includes JavaScript that does nasty stuff
 - e.g., clicks "Buy now!"
- Also HTTP response splitting
 - Untrusted input included in HTTP response headers



Path Traversal

- Forming a file path using untrusted input
 - Not checking for "../.../yourSecrets.xml"

Building Security In

- Generally, need deeper, interprocedural analysis
 - to connect untrusted input to places that require trusted values
- Automated tools helpful, perhaps essential
 - but not sufficient
- Need to build security in, not bolt it on afterwards
 - risk analysis, architecture, abuse cases
- Need training, expertise, tools, effort
 - generally not cheap or easy
 - but necessary



Agenda

Introduction

Correctness issues

Bad Practice

Security defects

Demos (FindBugs, Fortify SCA)

Integrating static analysis

Wrap up



DEMO

FindBugs and Fortify SCA



Agenda

Introduction

Correctness issues

Bad Practice

Security defects

Demos (FindBugs, Fortify SCA)

Integrating static analysis

Wrap up

Integrating Static Analysis

- Want to make it part of your development process
 - Just like running unit tests
- Have to tune the tool to report what you are interested in
 - Different situations have different needs
- Need a workflow for issues
 - Almost all tools will report some issues that, after reviewing, you decide not to fix
 - Need to have a way to manage such issues

Running Static Analysis

- "We've got it in our IDE, so we're done, right?"
 - no, it really needs to also be done automatically as part of your build process
- Are you scanning 2 million lines of code?
 - You probably don't want 20,000 issues to examine

Defect/Issue Workflow

- How do issues get reviewed/audited?
- Can you do team auditing and assign issues?
- Once you've reviewed an issue, does the system remember your evaluation when it analyzes that code again?
 - even if it is now reported on a different line number?
- Can you identify new issues
 - since last build?
 - since last release to customer/production?

Learning from mistakes

- With FindBugs, we've always started from bugs
- We need API experts to feed us API-specific bugs
 - Swing, EJB, J2ME, localization, Hibernate, ...
- When you get bit by a bug
 - writing a test case is good
 - considering whether it can be generalized into a bug pattern is better
 - You'd be surprised at the number of times you make a mistake so stupid “no one else could possible make the same mistake”
 - but they do



Agenda

Introduction

Correctness issues

Bad Practice

Security defects


Demos (FindBugs, Fortify SCA)

Integrating static analysis

Wrap up



Getting Started

- If you do nothing else, try FindBugs 
 - No salesman will call
 - Check out medium/high priority correctness warnings
- That should provide some motivation to get started
- But you really want to take it to the next level, find out what tools work best for you, make it part of your development process
 - not a casual commitment
 - but quality never is



JSR-305: Annotations for Defect Detection

- Develop annotations that are useful for static analysis tools
 - perhaps dynamic tools as well
- For example, which parameters and return values are allowed to be null
- Standard annotations interpreted by multiple tools
- Targets Java 5+
 - but combines with JSR-307: Annotations on Java Types



For More Information

- Testing Java Code: Beyond the IDE
 - *today, 2:50pm*
- BOF-9587 - Pimp My Java Application: Applying Static Analysis Tools to Boost Java Code Quality
 - *today, 7:55pm*
- BOF-9231 - FindBugs BOF
 - *today, 8:55pm*
- TS-5711 - Developing Reliable Products: Static and Dynamic Code Analysis
 - *tomorrow, 6:35pm*
- On web: FindBugs, PMD, CheckStyle, JetBrains*, Klockwork*, Fortify Software*, Coverity*, Parasoft
 - ** - also in exhibit hall*



Q&A

William Pugh

Professor
Univ. of Maryland
<http://www.cs.umd.edu/~pugh>



Additional Information

Free Tools

- FindBugs
- Java Open Review: FindBugs + Fortify SCA
 - free service for open source projects
 - collaborative, distributed auditing
- PMD
 - great capabilities for writing custom rules
- CheckStyle
 - good for enforcing coding conventions

Development Environments

- Eclipse
 - has a fair number of checkers built in
 - but they don't seem to eat their own dog food
- IntelliJ
 - fairly smart set of checkers
- NetBeans
 - Jackpot system: makes it easy to write checkers and quick fixes

Commercial tools

- KlocWork K7
 - similar to FindBugs, lots of metrics and charts
- Fortify Software SCA
 - focuses on security issues
- Coverity Prevent
 - Java tool still top secret (at least, they won't talk to me)
- SureLogic Fluid
 - focuses on concurrency, design, user annotations
- Parasoft JTest
 - checks best practices