QA: Analysis Tools

17-313 Spring 202

Foundations of Software Engineering

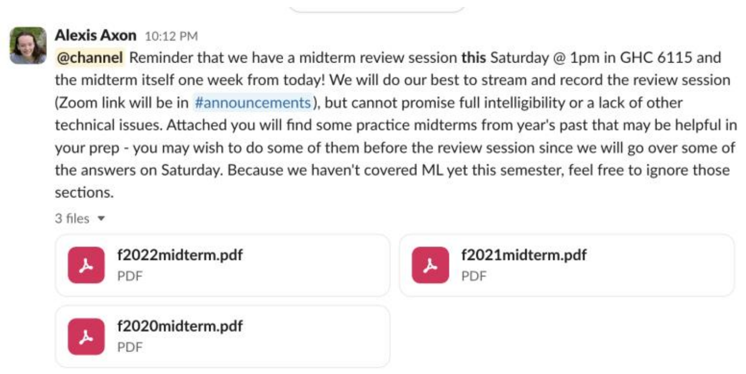
https://cmu-313.github.io

Michael Hilton and Edwardo Feo Flushing

Learning Goals

- Gain an understanding of the relative strengths and weaknesses of static and dynamic analysis
- Examine several popular analysis tools and understand their use cases
- Understand how analysis tools are used in large open-source software

Administrivia



Alexis Axon 10:12 PM
@channel Reminder that we have a midterm review session **this** Saturday @ 1pm in GHC 6115 and the midterm itself one week from today! We will do our best to stream and record the review session (Zoom link will be in #announcements), but cannot promise full intelligibility or a lack of other technical issues. Attached you will find some practice midterms from year's past that may be helpful in your prep - you may wish to do some of them before the review session since we will go over some of the answers on Saturday. Because we haven't covered ML yet this semester, feel free to ignore those sections.

3 files

f2022midterm.pdf PDF
f2021midterm.pdf PDF
f2020midterm.pdf PDF

- EXAM NEXT WEEK!
  - Practice exams are posted to slack
  - Review session: Saturday @ 1pm in GHC 6115
  - Review the previous exams, and bring questions to the review session
  - ODR Requests (should) have been approved
  - NOTE: Bonus points for handwritten cheat sheet, but printed is allowed to be used
- Please review grades on canvas.
  - Don't forget to do the teamwork survey. Participation point.

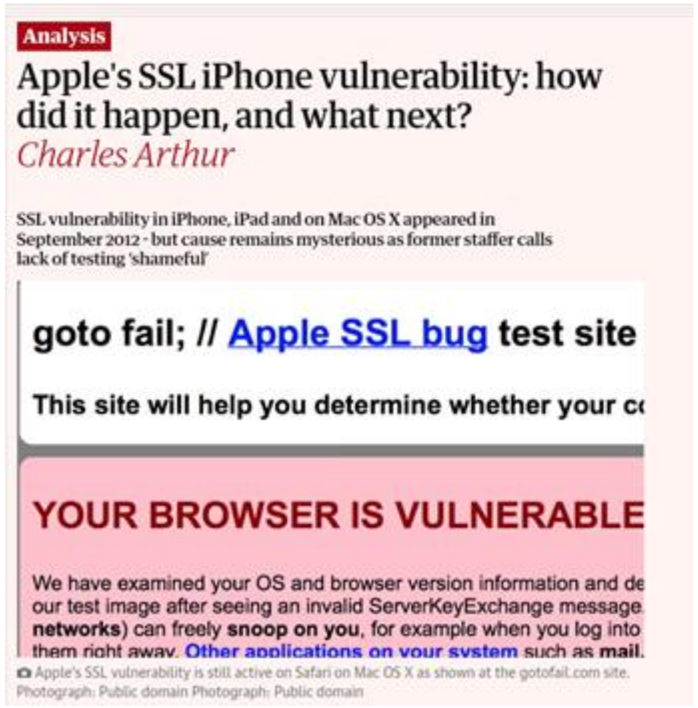
Software can be hard

- https://www.destroyallsoftware.com/talks/wat

```
static OSStatus
SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa,
                                 SSLBuffer signedParams,
                                 uint8_t *signature,
                                 UInt16 signatureLen) {
    OSStatus err;
     .…
    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
            goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
            goto fail;
            goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
            goto fail;
    …
fail:
    SSLFreeBuffer(&signedHashes);
    SSLFreeBuffer(&hashCtx);
    return err;
}
```

```c
static OSStatus
SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa,
                                 SSLBuffer signedParams,
                                 uint8_t *signature,
                                 UInt16 signatureLen) {
    OSStatus err;
     .…
    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
        goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;
    …
fail:
    SSLFreeBuffer(&signedHashes);
    SSLFreeBuffer(&hashCtx);
    return err;
}
```

goto fail;



Analysis

Apple's SSL iPhone vulnerability: how did it happen, and what next?

Charles Arthur

SSL vulnerability in iPhone, iPad and on Mac OS X appeared in September 2012 - but cause remains mysterious as former staffer calls lack of testing 'shameful'

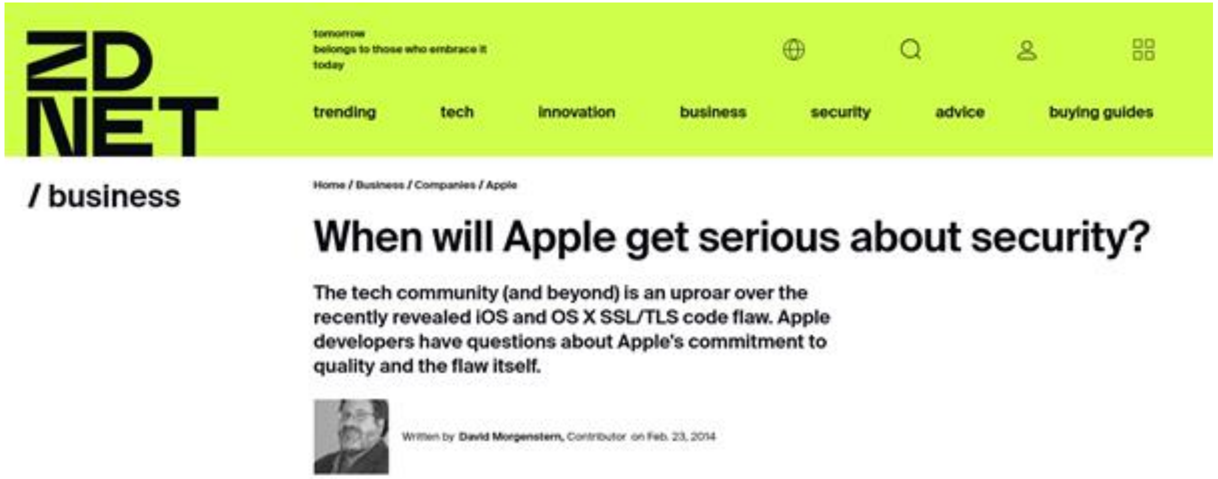
goto fail; // Apple SSL bug test site

This site will help you determine whether your c

YOUR BROWSER IS VULNERABLE

We have examined your OS and browser version information and de
our test image after seeing an invalid ServerKeyExchange message
networks) can freely **snoop on you**, for example when you log into
them right away. Other applications on your system such as **mail**.

Apple's SSL vulnerability is still active on Safari on Mac OS X as shown at the gotofail.com site. Photograph: Public domain Photograph: Public domain



/ business

Home / Business / Companies / Apple

When will Apple get serious about security?

The tech community (and beyond) is an uproar over the recently revealed iOS and OS X SSL/TLS code flaw. Apple developers have questions about Apple's commitment to quality and the flaw itself.

Written by **David Morgenstern,** Contributor on Feb. 23, 2014

```c
/* from Linux 2.3.99 drivers/block/raid5.c */
static struct buffer_head *
get_free_buffer(struct stripe_head * sh,
                int b_size) {
  struct buffer_head *bh;
  unsigned long flags;
  save_flags(flags);
  cli(); // disables interrupt
  if ((bh = sh->buffer_pool) == N   L)
    return NULL;
  sh->buffer_pool = bh -> b_next;
  bh->b_size = b_size;
  restore_flags(flags); // re-enables interrupts
  return bh;
}
```

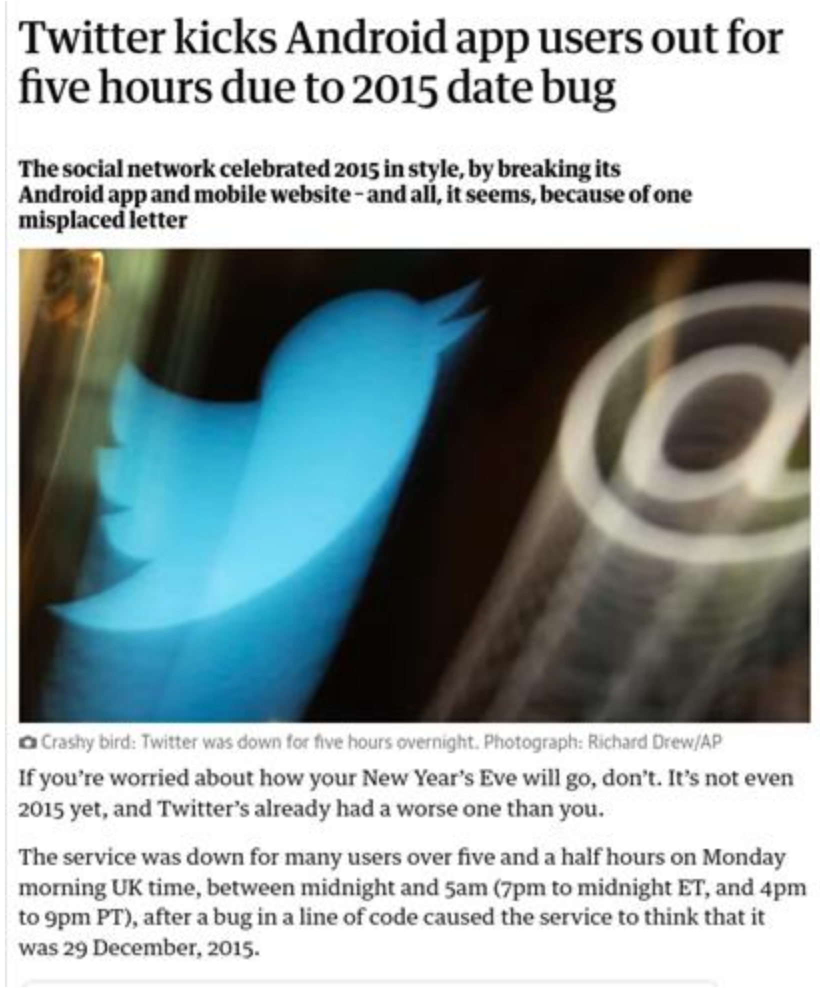
ERROR: function returns with interrupts disabled!

Twitter’s week year bug

ISO 8601 rule: *The first week of the year is the week containing the first Thursday.*
“So if January 1 falls on a Friday, it belongs to the last week of the previous year. If December 31 falls on a Wednesday, it belongs to week 01 of the following year.”

```
DateTimeFormatter.ofPattern("dd MMM YYYY").format(zonedDateTime)
```

Use yyyy instead of YYYY



Twitter kicks Android app users out for five hours due to 2015 date bug

The social network celebrated 2015 in style, by breaking its Android app and mobile website - and all, it seems, because of one misplaced letter

Crashy bird: Twitter was down for five hours overnight. Photograph: Richard Drew/AP

If you’re worried about how your New Year’s Eve will go, don’t. It’s not even 2015 yet, and Twitter’s already had a worse one than you.

The service was down for many users over five and a half hours on Monday morning UK time, between midnight and 5am (7pm to midnight ET, and 4pm to 9pm PT), after a bug in a line of code caused the service to think that it was 29 December, 2015.

Could you have found them?

- How often would those bugs trigger?
- Driver bug:
- What happens if you return from a driver with interrupts disabled?
- Consider: that’s one function
- …in a 2000 LOC file
- …in a module with 60,000 LOC
- …IN THE LINUX KERNEL
- *Some defects are very difficult to find via testing, inspection.*

Defects of interest…

- Are on uncommon or difficult-to-force execution paths. (vs testing)
- Executing (or interpreting/otherwise analyzing) all paths concretely to find such defects is <u>infeasible</u>.
- What we really want to do is check the <u>**entire possible state space**</u> of the program for <u>particular properties</u>.
- What we **CAN** do is check an <u>**abstract state space**</u> of the program for particular properties.

What is Static Analysis?

- **Systematic** examination of an **abstraction** of program **state space**.
- Does not execute code! (like code review)
- **Abstraction:** produce a representation of a program that is simpler to analyze.
- Results in fewer states to explore; makes difficult problems tractable.
- Check if a **particular property** holds over the entire state space:
- Liveness: “something good eventually happens.”
- Safety: “this bad thing can’t ever happen.”
- Compliance with mechanical design rules.

Activity: Analyze the Python program statically (Yes/No/Maybe)

```
def n2s(n: int, b: int):
 if n <= 0: return '0'
 r = ''
 while n > 0:
  u = n % b
  if u >= 10:
   u = chr(ord('A') + u-10)
  n = n // b
  r = str(u) + r
 return r
```

1. What are the set of data types taken by variable `u` at any point in the program?
2. Can the variable `u` be a negative number?
3. Will this function always return a value?
4. Can there ever be a division by zero?
5. Will the returned value ever contain a minus sign '-'?

What static analysis can and cannot do

- Type-checking is well established
  - Set of data types taken by variables at any point
  - Can be used to prevent type errors (e.g. Java) or warn about potential type errors (e.g. Python)
- Checking for problematic patterns in syntax is easy and fast
  - Is there a comparison of two Java strings using `==`?
  - Is there an array access `a[i]` without an enclosing bounds check for `i`?
- Reasoning about termination is impossible in general
  - Halting problem
- Reasoning about exact values is hard, but conservative analysis via abstraction is possible
  - Is the bounds check before `a[i]` guaranteeing that `I` is within bounds?
  - Can the divisor ever take on a zero value?
  - Could the result of a function call be `42`?
  - Will this multi-threaded program give me a deterministic result?
  - Be prepared for "MAYBE"
- Verifying some advanced properties is possible but expensive
  - CI-based static analysis usually over-approximates conservatively

The Bad News: Rice's Theorem
Every static analysis is necessarily incomplete, unsound, undecidable, or a combination thereof

"Any nontrivial property about the language recognized by a Turing machine is undecidable."

Henry Gordon Rice, 1953

Static Analysis is well suited to detecting certain defects

- Security: Buffer overruns, improperly validated input…
- Memory safety: Null dereference, uninitialized data…
- Resource leaks: Memory, OS resources…
- API Protocols: Device drivers; real time libraries; GUI frameworks
- Exceptions: Arithmetic/library/user-defined
- Encapsulation:
  - Accessing internal data, calling private functions…
- Data races:
  - Two threads access the same data without synchronization

Static Analysis: Broad classification

- Linters
  - Shallow syntax analysis for enforcing code styles and formatting
- Pattern-based bug detectors
  - Simple syntax or API-based rules for identifying common programming mistakes
- Type-annotation validators
  - Check conformance to user-defined types
  - Types can be complex (e.g., “Nullable”)
- Data-flow analysis / Abstract interpretation)
  - Deep program analysis to find complex error conditions (e.g., ”can array index be out of bounds?”)

Static analysis can be applied to all attributes

- Find bugs
- Refactor code
- Keep your code stylish!
- Identify code smells
- Measure quality
- Find usability and accessibility issues
- Identify bottlenecks and improve performance

Activity: Analyze the Python program dynamically

```
def n2s(n: int, b: int):
  if n <= 0: return '0'
  r = ''
  while n > 0:
    u = n % b
    if u >= 10:
      u = chr(ord('A') + u-10)
    n = n // b
    r = str(u) + r
  return r

print(n2s(12, 10))
```

1. What are the set of data types taken by variable `u` at any point in the program?
2. Did the variable `u` ever contain a negative number?
3. For how many iterations did the while loop execute?
4. Was there ever be a division by zero?
5. Did the returned value ever contain a minus sign ‘-’?

Dynamic analysis reasons about program executions

- Tells you properties of the program that were definitely observed
  - Code coverage
  - Performance profiling
  - Type profiling
  - Testing

- In practice, implemented by program instrumentation
  - Think "Automated logging"
  - Slows down execution speed by a small amount

Static Analysis vs Dynamic Analysis

Static Analysis	Dynamic Analysis
Requires only source code	Requires successful build + test inputs
Conservatively reasons about all possible inputs and program paths	Observes individual executions
Reported warnings may contain false positives	Reported problems are real, as observed by a witness input
Can report all warnings of a particular class of problems	Can only report problems that are seen. Highly dependent on test inputs. Subject to false negatives
Advanced techniques like verification can prove certain complex properties, but rarely run in CI due to cost	Advanced techniques like symbolic execution can prove certain complex properties, but rarely run in CI due to cost

Static Analysis

Tools for Static Analysis

CS

UNIVERSITY OF MARYLAND 18 56

FindBugs™

because it's easy

: my[py]

snyk

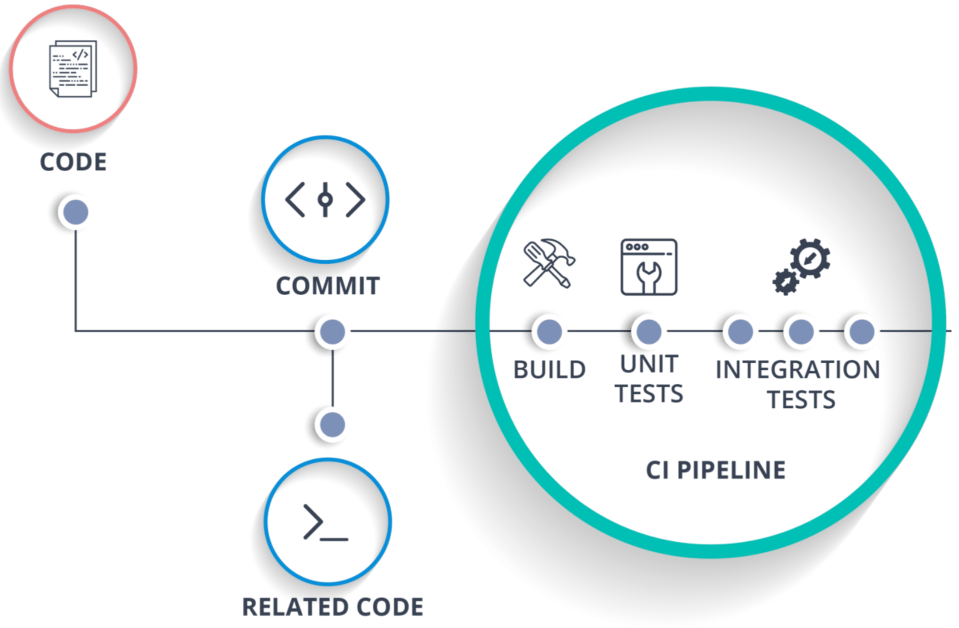
sonarqube

coverity®

SYNOPSYS®

squale

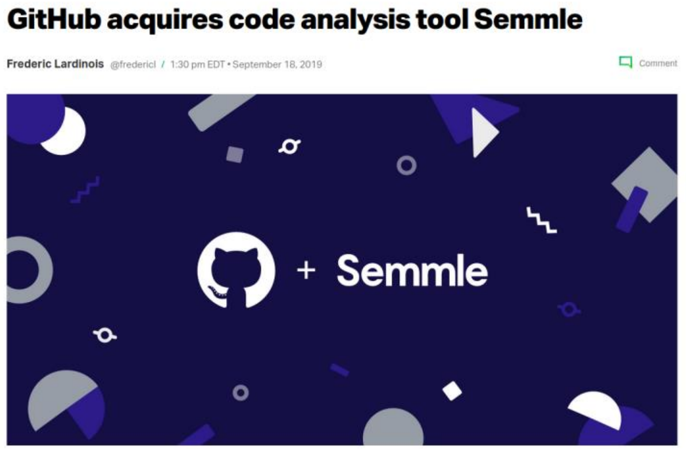
Static analysis is a key part of continuous integration



Travis CI

GitHub Actions

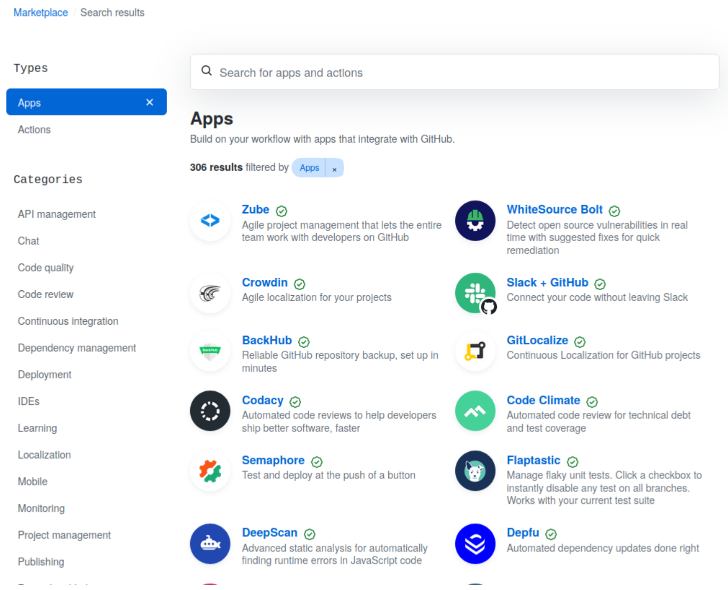
Static analysis used to be an academic amusement; now it's heavily commercialized



GitHub acquires code analysis tool Semmle

Frederic Lardinois @fredericl / 1:30 pm EDT • September 18, 2019

Comment



Marketplace / Search results

Types

- Apps ×
- Actions

Categories

- API management
- Chat
- Code quality
- Code review
- Continuous integration
- Dependency management
- Deployment
- IDEs
- Learning
- Localization
- Mobile
- Monitoring
- Project management
- Publishing

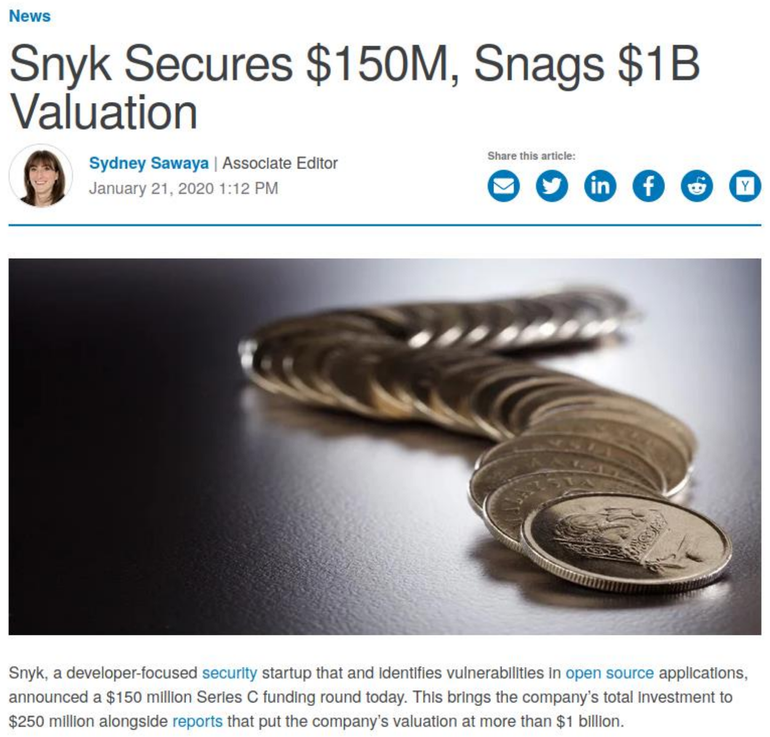
Search for apps and actions

Apps

Build on your workflow with apps that integrate with GitHub.

306 results filtered by Apps ×

- **Zube** — Agile project management that lets the entire team work with developers on GitHub
- **WhiteSource Bolt** — Detect open source vulnerabilities in real time with suggested fixes for quick remediation
- **Crowdin** — Agile localization for your projects
- **Slack + GitHub** — Connect your code without leaving Slack
- **BackHub** — Reliable GitHub repository backup, set up in minutes
- **GitLocalize** — Continuous Localization for GitHub projects
- **Codacy** — Automated code reviews to help developers ship better software, faster
- **Code Climate** — Automated code review for technical debt and test coverage
- **Semaphore** — Test and deploy at the push of a button
- **Flaptastic** — Manage flaky unit tests. Click a checkbox to instantly disable any test on all branches. Works with your current test suite
- **DeepScan** — Advanced static analysis for automatically finding runtime errors in JavaScript code
- **Depfu** — Automated dependency updates done right



News

Snyk Secures $150M, Snags $1B Valuation

Sydney Sawaya | Associate Editor

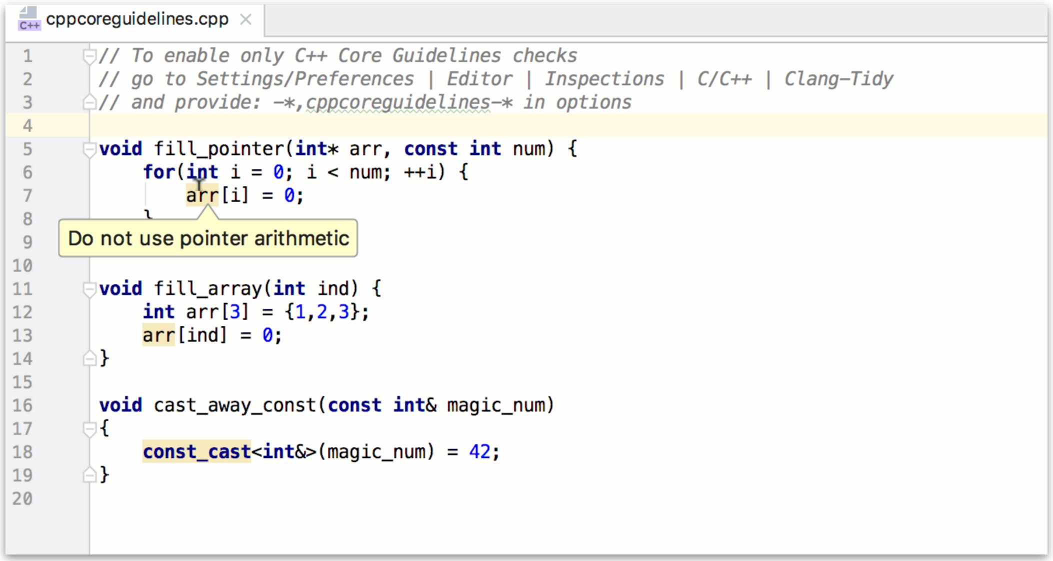
January 21, 2020 1:12 PM

Share this article:

Snyk, a developer-focused security startup that and identifies vulnerabilities in open source applications, announced a $150 million Series C funding round today. This brings the company's total investment to $250 million alongside reports that put the company's valuation at more than $1 billion.

https://www.sdxcentral.com/articles/news/snyk-secures-150m-snags-1b-valuation/2020/01/
https://techcrunch.com/2019/09/18/github-acquires-code-analysis-tool-semmle/
https://github.com/marketplace

Static analysis is also integrated into IDEs



cppcoreguidelines.cpp

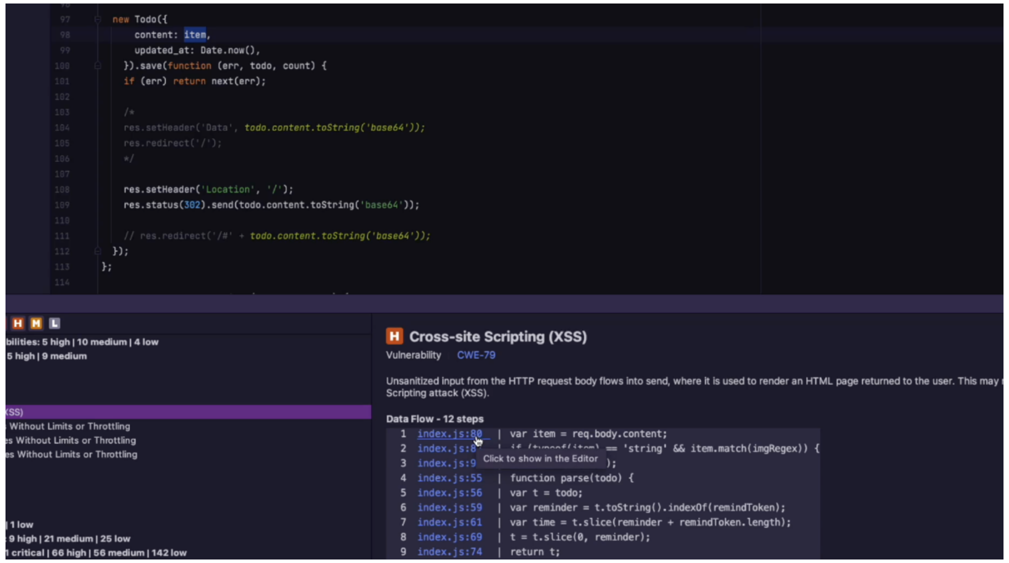
```
// To enable only C++ Core Guidelines checks
// go to Settings/Preferences | Editor | Inspections | C/C++ | Clang-Tidy
// and provide: -*,cppcoreguidelines-* in options

void fill_pointer(int* arr, const int num) {
    for(int i = 0; i < num; ++i) {
        arr[i] = 0;
    }
}

void fill_array(int ind) {
    int arr[3] = {1,2,3};
    arr[ind] = 0;
}

void cast_away_const(const int& magic_num)
{
    const_cast<int&>(magic_num) = 42;
}
```

Do not use pointer arithmetic



```
    new Todo({
      content: item,
      updated_at: Date.now(),
    }).save(function (err, todo, count) {
    if (err) return next(err);

    /*
    res.setHeader('Data', todo.content.toString('base64'));
    res.redirect('/');
    */

    res.setHeader('Location', '/');
    res.status(302).send(todo.content.toString('base64'));

    // res.redirect('/#' + todo.content.toString('base64'));
  });
};
```

bilities: 5 high | 10 medium | 4 low
5 high | 9 medium
XSS)
Without Limits or Throttling
es Without Limits or Throttling
es Without Limits or Throttling
| 1 low
9 high | 21 medium | 25 low
1 critical | 66 high | 56 medium | 142 low

Cross-site Scripting (XSS)

Vulnerability CWE-79

Unsanitized input from the HTTP request body flows into send, where it is used to render an HTML page returned to the user. This may
Scripting attack (XSS).

Data Flow - 12 steps

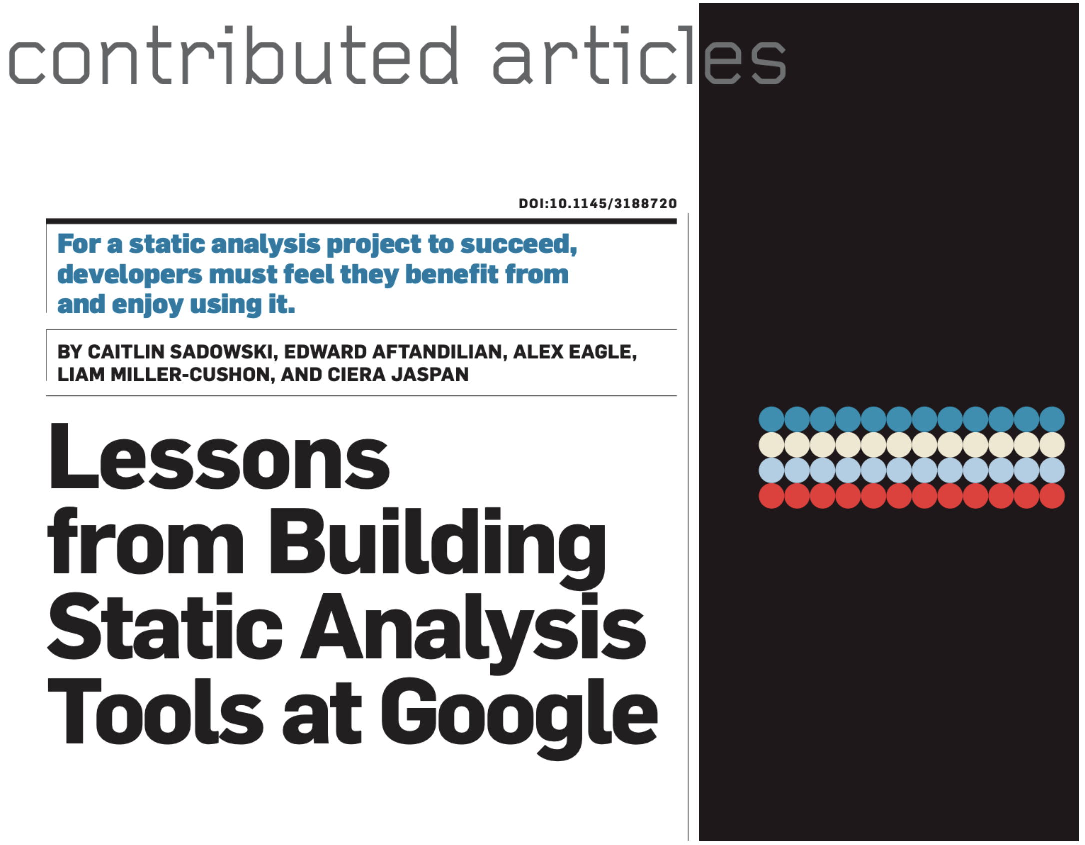
```
1  index.js:80  | var item = req.body.content;
2  index.js:8   | if (typeof(item) == 'string' && item.match(imgRegex)) {
3  index.js:9   Click to show in the Editor );
4  index.js:55  | function parse(todo) {
5  index.js:56  | var t = todo;
6  index.js:59  | var reminder = t.toString().indexOf(remindToken);
7  index.js:61  | var time = t.slice(reminder + remindToken.length);
8  index.js:69  | t = t.slice(0, reminder);
9  index.js:74  | return t;
```

What makes a good static analysis tool?

- Static analysis should be **fast**
    - Don't hold up development velocity
    - This becomes more important as code scales
- Static analysis should report **few false positives**
    - Otherwise developers will start to ignore warnings and alerts, and quality will decline
- Static analysis should be **continuous**
    - Should be part of your continuous integration pipeline
    - Diff-based analysis is even better -- don't analyse the entire codebase; just the changes
- Static analysis should be **informative**
    - Messages that help the developer to quickly locate and address the issue
    - Ideally, it should suggest or automatically apply fixes

Lessons for Static Analysis Tools at Google

- Make It a Compiler Workflow
- Value of compiler checks.
- Reporting issues sooner is better
- Warn During Code Review
- Engineers working on static analysis must demonstrate impact through hard data.



contributed articles

DOI:10.1145/3188720

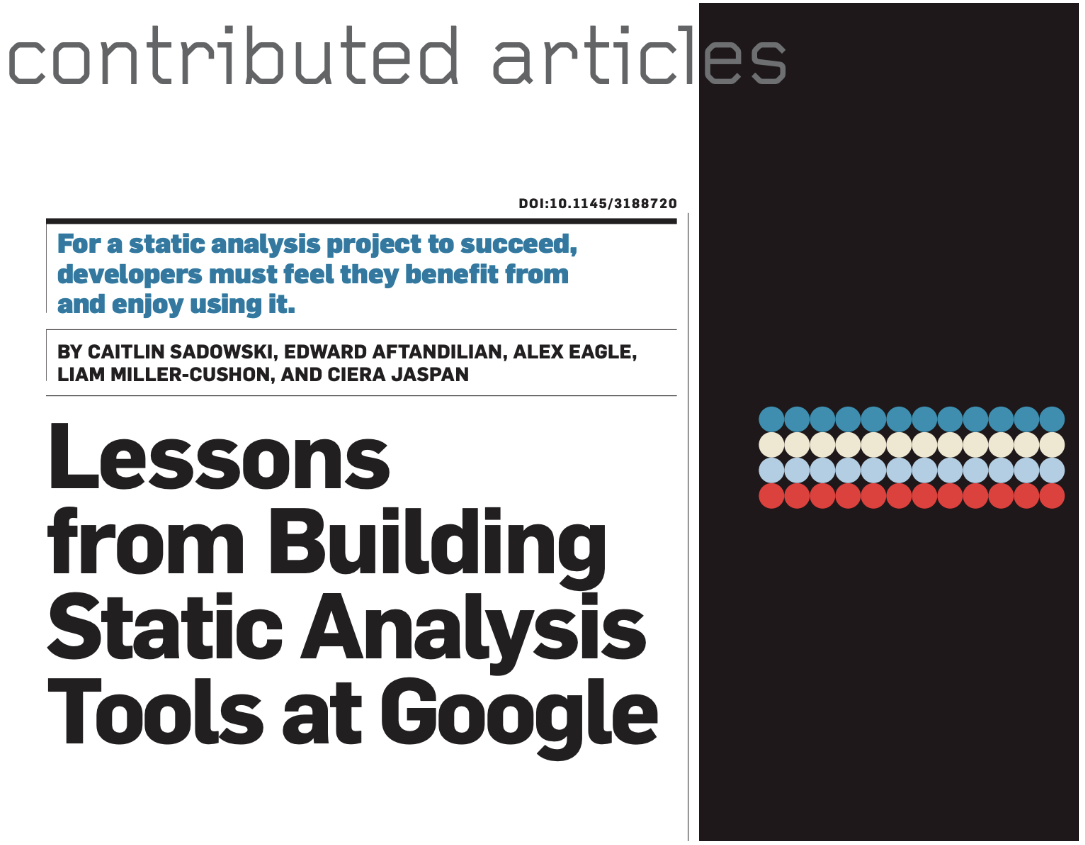
For a static analysis project to succeed, developers must feel they benefit from and enjoy using it.

BY CAITLIN SADOWSKI, EDWARD AFTANDILIAN, ALEX EAGLE, LIAM MILLER-CUSHON, AND CIERA JASPAN

Lessons from Building Static Analysis Tools at Google

Lessons learned

- Finding bugs is easy
- Most developers will not go out of their way to use static analysis tools.
- Developer happiness is key.
- Do not just find bugs, fix them.
- Crowdsource analysis development.


contributed articles

DOI:10.1145/3188720

For a static analysis project to succeed, developers must feel they benefit from and enjoy using it.

BY CAITLIN SADOWSKI, EDWARD AFTANDILIAN, ALEX EAGLE, LIAM MILLER-CUSHON, AND CIERA JASPAN

Lessons from Building Static Analysis Tools at Google

(1) Linters: *Cheap, fast, and lightweight static source analysis*

Use linters to enforce style guidelines

Don’t rely on manual inspection during code review!

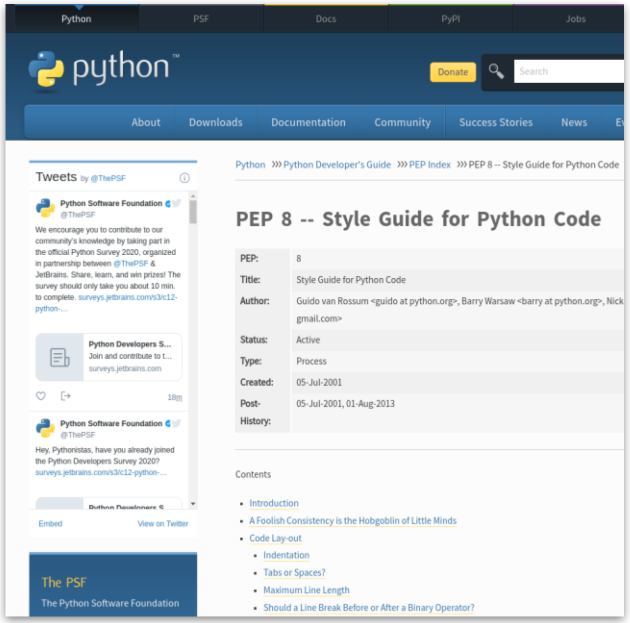
Linters use very “shallow” static analysis to enforce formatting rules

- Ensure proper indentation
- Naming convention
- Line sizes
- Class nesting
- Documenting public functions
- Parenthesis around expressions
- What else?

Use linters to improve maintainability

- Why? We spend more time reading code than writing it.
  - Various estimates of the exact %, some as high as 80%
- Code is ownership is usually shared
- The original owner of some code may move on
- Code conventions make it easier for other developers to quickly understand your code

Use Style Guidelines to facilitate communication



Tweets by @ThePSF

Python Software Foundation
@ThePSF
We encourage you to contribute to our community's knowledge by taking part in the official Python Survey 2020, organized in partnership between @ThePSF & JetBrains. Share, learn, and win prizes! The survey should only take you about 10 min. to complete. surveys.jetbrains.com/s3/c12-python-…

Python Developers S…
Join and contribute to t…
surveys.jetbrains.com

18m

Python Software Foundation
@ThePSF
Hey, Pythonistas, have you already joined the Python Developers Survey 2020?
surveys.jetbrains.com/s3/c12-python-…

Embed View on Twitter

The PSF
The Python Software Foundation

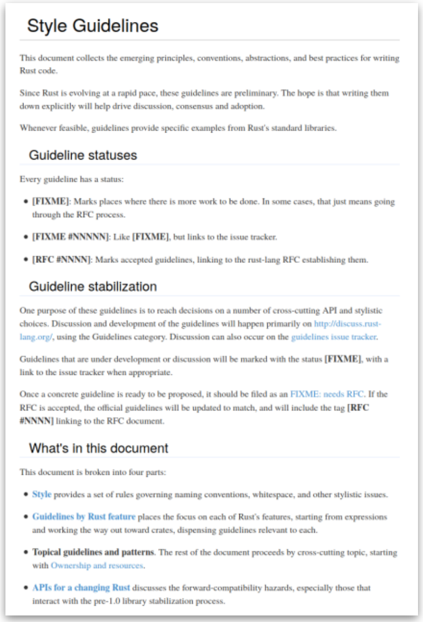
Python ››› Python Developer's Guide ››› PEP Index ››› PEP 8 -- Style Guide for Python Code

PEP 8 -- Style Guide for Python Code

PEP:	8
Title:	Style Guide for Python Code
Author:	Guido van Rossum <guido at python.org>, Barry Warsaw <barry at python.org>, Nick gmail.com>
Status:	Active
Type:	Process
Created:	05-Jul-2001
Post-History:	05-Jul-2001, 01-Aug-2013

Contents

- Introduction
- A Foolish Consistency is the Hobgoblin of Little Minds
- Code Lay-out
  - Indentation
  - Tabs or Spaces?
  - Maximum Line Length
  - Should a Line Break Before or After a Binary Operator?



Style Guidelines

This document collects the emerging principles, conventions, abstractions, and best practices for writing Rust code.

Since Rust is evolving at a rapid pace, these guidelines are preliminary. The hope is that writing them down explicitly will help drive discussion, consensus and adoption.

Whenever feasible, guidelines provide specific examples from Rust's standard libraries.

Guideline statuses

Every guideline has a status:

- **[FIXME]**: Marks places where there is more work to be done. In some cases, that just means going through the RFC process.
- **[FIXME #NNNNN]**: Like **[FIXME]**, but links to the issue tracker.
- **[RFC #NNNN]**: Marks accepted guidelines, linking to the rust-lang RFC establishing them.

Guideline stabilization

One purpose of these guidelines is to reach decisions on a number of cross-cutting API and stylistic choices. Discussion and development of the guidelines will happen primarily on http://discuss.rust-lang.org/, using the Guidelines category. Discussion can also occur on the guidelines issue tracker.

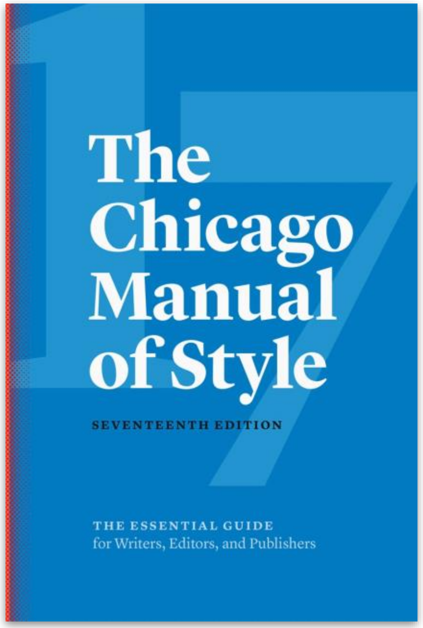
Guidelines that are under development or discussion will be marked with the status **[FIXME]**, with a link to the issue tracker when appropriate.

Once a concrete guideline is ready to be proposed, it should be filed as an FIXME: needs RFC. If the RFC is accepted, the official guidelines will be updated to match, and will include the tag **[RFC #NNNN]** linking to the RFC document.

What's in this document

This document is broken into four parts:

- Style provides a set of rules governing naming conventions, whitespace, and other stylistic issues.
- Guidelines by Rust feature places the focus on each of Rust's features, starting from expressions and working the way out toward crates, dispensing guidelines relevant to each.
- **Topical guidelines and patterns**. The rest of the document proceeds by cross-cutting topic, starting with Ownership and resources.
- APIs for a changing Rust discusses the forward-compatibility hazards, especially those that interact with the pre-1.0 library stabilization process.


The Chicago Manual of Style
SEVENTEENTH EDITION
THE ESSENTIAL GUIDE
for Writers, Editors, and Publishers

Guidelines are inherently opinionated, but **consistency** is the important point.

Agree to a set of conventions and stick to them.

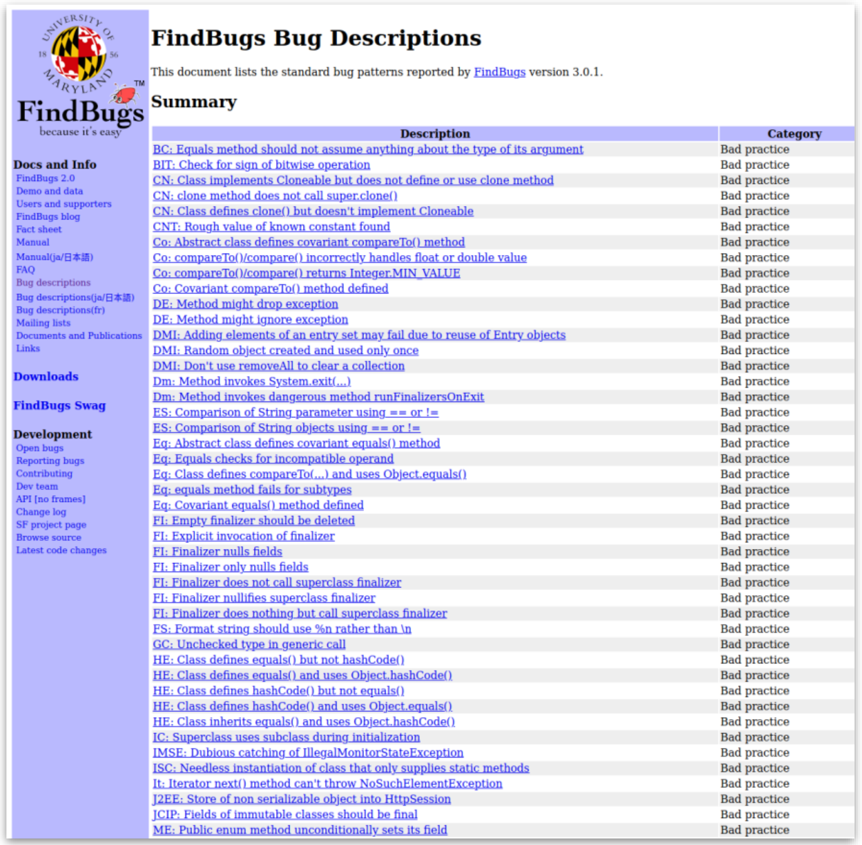
https://www.chicagomanualofstyle.org/ | https://google.github.io/styleguide/ | https://www.python.org/dev/peps/pep-0008

Take Home Message: Style is an easy way to improve readability

- Everyone has their own opinion (e.g., tabs vs. spaces)
- Agree to a convention and stick to it
  - Use continuous integration to enforce it
- Use automated tools to fix issues in existing code

(2) Patten-based Static Analysis Tools

- Bad Practice
- Correctness
- Performance
- Internationalization
- Malicious Code
- Multithreaded Correctness
- Security
- Dodgy Code



FindBugs
because it's easy

Docs and Info
FindBugs 2.0
Demo and data
Users and supporters
FindBugs blog
Fact sheet
Manual
Manual(ja/日本語)
FAQ
Bug descriptions
Bug descriptions(ja/日本語)
Bug descriptions(fr)
Mailing lists
Documents and Publications
Links

Downloads

FindBugs Swag

Development
Open bugs
Reporting bugs
Contributing
Dev team
API [no frames]
Change log
SF project page
Browse source
Latest code changes

FindBugs Bug Descriptions

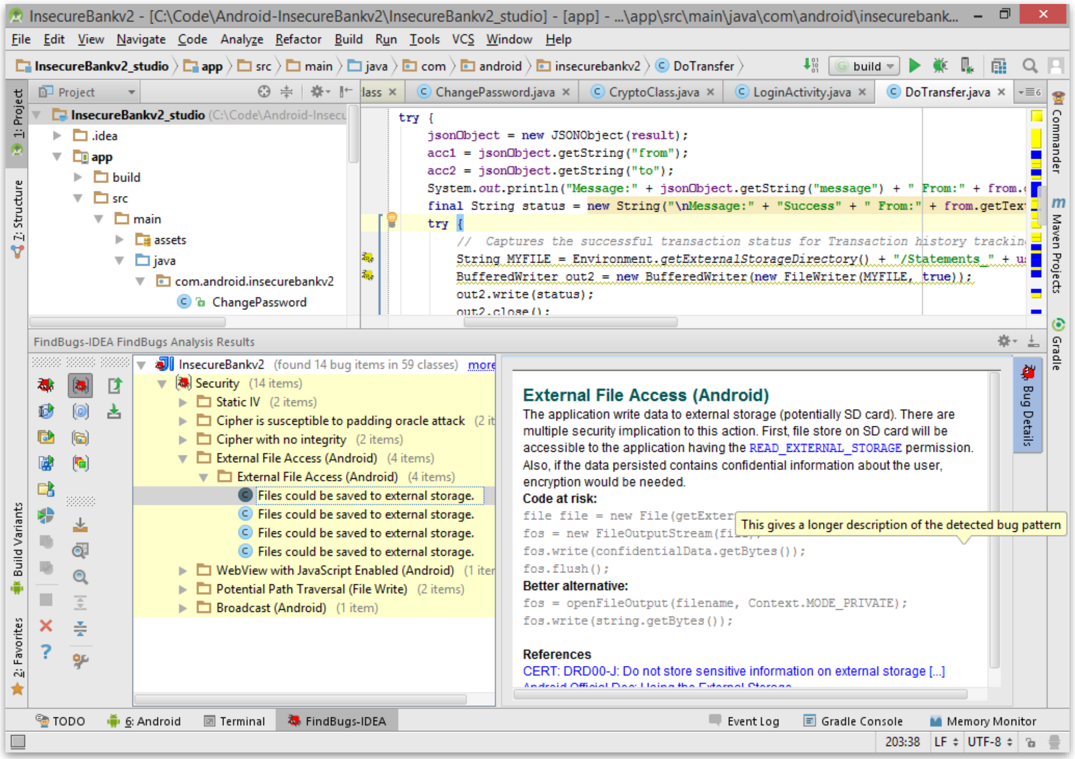
This document lists the standard bug patterns reported by FindBugs version 3.0.1.

Summary

Description	Category
BC: Equals method should not assume anything about the type of its argument	Bad practice
BIT: Check for sign of bitwise operation	Bad practice
CN: Class implements Cloneable but does not define or use clone method	Bad practice
CN: clone method does not call super.clone()	Bad practice
CN: Class defines clone() but doesn't implement Cloneable	Bad practice
CNT: Rough value of known constant found	Bad practice
Co: Abstract class defines covariant compareTo() method	Bad practice
Co: compareTo()/compare() incorrectly handles float or double value	Bad practice
Co: compareTo()/compare() returns Integer.MIN_VALUE	Bad practice
Co: Covariant compareTo() method defined	Bad practice
DE: Method might drop exception	Bad practice
DE: Method might ignore exception	Bad practice
DMI: Adding elements of an entry set may fail due to reuse of Entry objects	Bad practice
DMI: Random object created and used only once	Bad practice
DMI: Don't use removeAll to clear a collection	Bad practice
Dm: Method invokes System.exit(...)	Bad practice
Dm: Method invokes dangerous method runFinalizersOnExit	Bad practice
ES: Comparison of String parameter using == or !=	Bad practice
ES: Comparison of String objects using == or !=	Bad practice
Eq: Abstract class defines covariant equals() method	Bad practice
Eq: Equals checks for incompatible operand	Bad practice
Eq: Class defines compareTo(...) and uses Object.equals()	Bad practice
Eq: equals method fails for subtypes	Bad practice
Eq: Covariant equals() method defined	Bad practice
FI: Empty finalizer should be deleted	Bad practice
FI: Explicit invocation of finalizer	Bad practice
FI: Finalizer nulls fields	Bad practice
FI: Finalizer only nulls fields	Bad practice
FI: Finalizer does not call superclass finalizer	Bad practice
FI: Finalizer nullifies superclass finalizer	Bad practice
FI: Finalizer does nothing but call superclass finalizer	Bad practice
FS: Format string should use %n rather than \n	Bad practice
GC: Unchecked type in generic call	Bad practice
HE: Class defines equals() but not hashCode()	Bad practice
HE: Class defines equals() and uses Object.hashCode()	Bad practice
HE: Class defines hashCode() but not equals()	Bad practice
HE: Class defines hashCode() and uses Object.equals()	Bad practice
HE: Class inherits equals() and uses Object.hashCode()	Bad practice
IC: Superclass uses subclass during initialization	Bad practice
IMSE: Dubious catching of IllegalMonitorStateException	Bad practice
ISC: Needless instantiation of class that only supplies static methods	Bad practice
It: Iterator next() method can't throw NoSuchElementException	Bad practice
J2EE: Store of non serializable object into HttpSession	Bad practice
JCIP: Fields of immutable classes should be final	Bad practice
ME: Public enum method unconditionally sets its field	Bad practice

http://findbugs.sourceforge.net/bugDescriptions.html
https://spotbugs.readthedocs.io/en/latest/bugDescriptions.html

SpotBugs can be extended with plugins



InsecureBankv2 - [C:\Code\Android-InsecureBankv2\InsecureBankv2_studio] - [app] - ...\app\src\main\java\com\android\insecurebank...

```
try {
    jsonObject = new JSONObject(result);
    acc1 = jsonObject.getString("from");
    acc2 = jsonObject.getString("to");
    System.out.println("Message:" + jsonObject.getString("message") + " From:" + from.
    final String status = new String("\nMessage:" + "Success" + " From:" + from.getTex
    try {
        //  Captures the successful transaction status for Transaction history trackin
        String MYFILE = Environment.getExternalStorageDirectory() + "/Statements_" + u
        BufferedWriter out2 = new BufferedWriter(new FileWriter(MYFILE, true));
        out2.write(status);
        out2.close();
```

FindBugs-IDEA FindBugs Analysis Results

- InsecureBankv2 (found 14 bug items in 59 classes) more
  - Security (14 items)
    - Static IV (2 items)
    - Cipher is susceptible to padding oracle attack (2 it
    - Cipher with no integrity (2 items)
    - External File Access (Android) (4 items)
      - External File Access (Android) (4 items)
        - Files could be saved to external storage.
        - Files could be saved to external storage.
        - Files could be saved to external storage.
        - Files could be saved to external storage.
    - WebView with JavaScript Enabled (Android) (1 iter
    - Potential Path Traversal (File Write) (2 items)
    - Broadcast (Android) (1 item)

External File Access (Android)

The application write data to external storage (potentially SD card). There are multiple security implication to this action. First, file store on SD card will be accessible to the application having the READ_EXTERNAL_STORAGE permission. Also, if the data persisted contains confidential information about the user, encryption would be needed.

Code at risk:

```
file file = new File(getExter
fos = new FileOutputStream(fi
fos.write(confidentialData.getBytes());
fos.flush();
```

This gives a longer description of the detected bug pattern

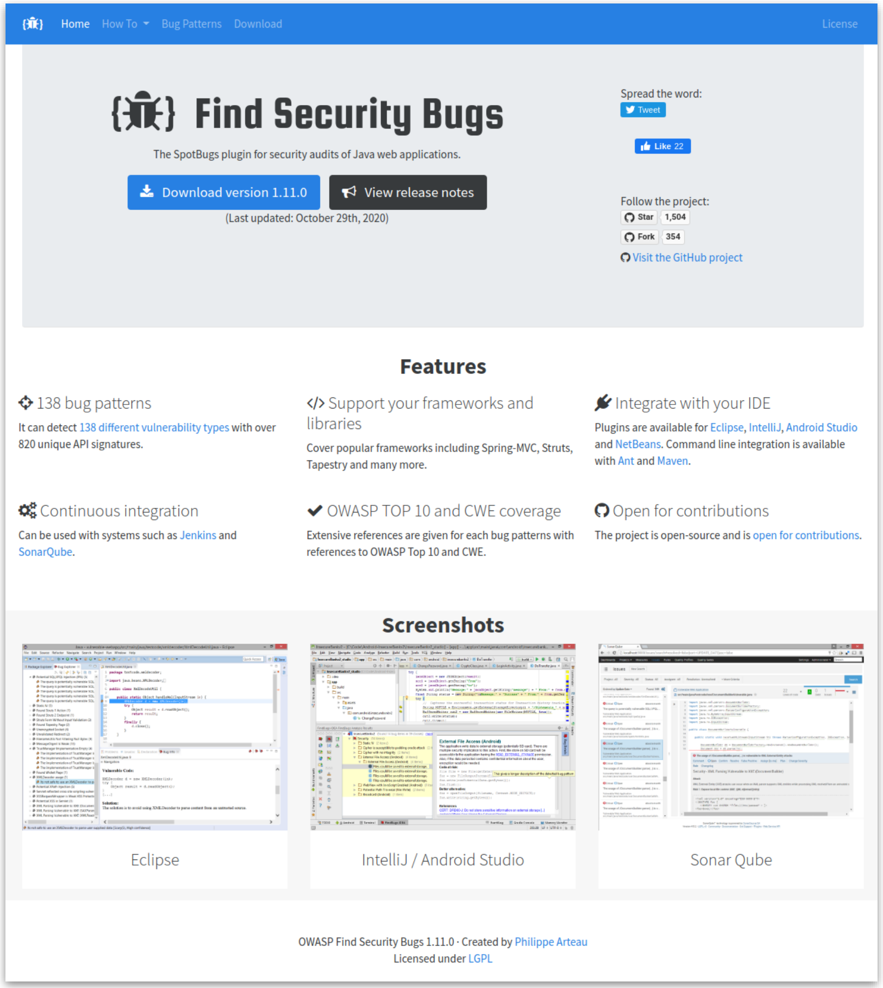
Better alternative:

```
fos = openFileOutput(filename, Context.MODE_PRIVATE);
fos.write(string.getBytes());
```

References

CERT: DRD00-J: Do not store sensitive information on external storage [...]

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
InsecureBankv2_studio app src main java com android insecurebankv2 DoTransfer build
Project ChangePassword.java CryptoClass.java LoginActivity.java DoTransfer.java
InsecureBankv2_studio (C:\Code\Android-Insecu
.idea app build src main assets java com.android.insecurebankv2 ChangePassword
Bug Details
TODO 6: Android Terminal FindBugs-IDEA Event Log Gradle Console Memory Monitor
203:38 LF UTF-8



Home How To Bug Patterns Download License

Find Security Bugs

The SpotBugs plugin for security audits of Java web applications.

Download version 1.11.0 | View release notes

(Last updated: October 29th, 2020)

Spread the word: Tweet Like 22

Follow the project: Star 1,504 Fork 354

Visit the GitHub project

Features

138 bug patterns
It can detect 138 different vulnerability types with over 820 unique API signatures.

Support your frameworks and libraries
Cover popular frameworks including Spring-MVC, Struts, Tapestry and many more.

Integrate with your IDE
Plugins are available for Eclipse, IntelliJ, Android Studio and NetBeans. Command line integration is available with Ant and Maven.

Continuous integration
Can be used with systems such as Jenkins and SonarQube.

OWASP TOP 10 and CWE coverage
Extensive references are given for each bug patterns with references to OWASP Top 10 and CWE.

Open for contributions
The project is open-source and is open for contributions.

Screenshots

Eclipse | IntelliJ / Android Studio | Sonar Qube

OWASP Find Security Bugs 1.11.0 - Created by Philippe Arteau
Licensed under LGPL

Bad Practice:

```
String x = new String("Foo");
String y = new String("Foo");

if (x == y) {
  System.out.println("x and y are the same!");
} else {
  System.out.println("x and y are different!");

}
```

Bad Practice: ES_COMPARING_STRINGS_WITH_EQ Comparing strings with ==

```
String x = new String("Foo");
String y = new String("Foo");

~~if (x == y) {~~
if (x.equals(y)) {
  System.out.println("x and y are the same!");
} else {
  System.out.println("x and y are different!");
}
```

Performance:

```
public static String repeat(String string, int times)
{
  String output = string;
  for (int i = 1; i < times; ++i) {
    output = output + string;
  }
  return output;
}
```

Performance: SBSC_USE_STRINGBUFFER_CONCATENATION Method concatenates strings using + in a loop

```
public static String repeat(String string, int times)
{
  String output = string;
  for (int i = 1; i < times; ++i) {
    output = output + string;
  }
  return output;
}
```

The method seems to be building a String using concatenation in a loop. In each iteration, the String is converted to a StringBuffer/StringBuilder, appended to, and converted back to a String. **This can lead to a cost quadratic in the number of iterations, as the growing string is recopied in each iteration.**

Performance: SBSC_USE_STRINGBUFFER_CONCATENATION
Method concatenates strings using + in a loop

```
public static String repeat(String string, int times)
{
 int length = string.length() * times;
 StringBuffer output = new StringBuffer(length);
 for (int i = 0; i < times; ++i) {
  output.append(string);
 }
 return output.toString();
}
```

Challenges

- The analysis must produce zero false positives
  - Otherwise developers won’t be able to build the code!
- The analysis needs to be really fast
  - Ideally < 100 ms
  - If it takes longer, developers will become irritated and lose productivity
- You can’t just “turn on” a particular check
  - Every instance where that check fails will prevent existing code from building
  - There could be thousands of violations for a single check across large codebases

(3) Use type annotations to detect common errors

- Uses a conservative analysis to prove the absence of certain defects
  - Null pointer errors, uninitialized fields, certain liveness issues, information leaks, SQL injections, bad regular expressions, incorrect physical units, bad format strings, …
  - C.f. SpotBugs which makes no safety guarantees
  - Assuming that code is annotated and those annotations are correct
- Uses annotations to enhance type system
- Example: Java Checker Framework or MyPy

Annotations can be applied to types and declarations

```
// return value
@NonNull String toString() { ... }

// parameter
int compareTo(@NonNull String other) { ... }

// receiver ("this" parameter)
String toString(@Tainted MyClass this) { ... }
```

Detecting null pointer exceptions

- @Nullable indicates that an expression may be null
- @NonNull indicates that an expression must never be null
  - Rarely used because @NonNull is assumed by default
  - See documentation for other nullness annotations
- Guarantees that expressions annotated with @NonNull will never evaluate to null, forbids other expressions from being dereferenced

- import org.checkerframework.checker.nullness.qual.*;

- public class NullnessExampleWithWarnings {
-     public void example() {
-         **@NonNull** String foo = "foo";
-         String bar = null;

-         foo = bar;
-     }
- }

- import org.checkerframework.checker.nullness.qual.*;

- public class NullnessExampleWithWarnings {
-     public void example() {
-         **@NonNull** String foo = "foo";
-         String bar = null;

-         foo = bar;
-     }
- }

`@Nullable` is applied by default

```
import org.checkerframework.checker.nullness.qual.*;

public class NullnessExampleWithWarnings  {
   public void example() {
      @NonNull String foo = "foo";
      String bar = null;

      foo = bar;
   }
}
```

`@Nullable` is applied by default

```
Error: [assignment.type.incompatible] incompatible types in assignment.
  found    : @Initialized @Nullable String
  required: @UnknownInitialization @NonNull String
```

```
import org.checkerframework.checker.nullness.qual.*;

public class NullnessExampleWithWarnings {
  public void example() {
    @NonNull String foo = "foo";
    String bar = null;   // @Nullable

    if (bar != null) {
      foo = bar;
    }
  }
}
```

bar is refined to `@NonNull`

Is there a bug?

```
public String getDay(int dayIndex) {
  String day = null;
  switch (dayIndex) {
    case 0: day = "Monday";
    case 1: day = "Tuesday";
    case 2: day = "Wednesday";
    case 3: day = "Thursday";
  }
  return day;
}

public void example() {
  @NonNull String dayName = getDay(4);
  System.out.println("Today is " + dayName);
}
```

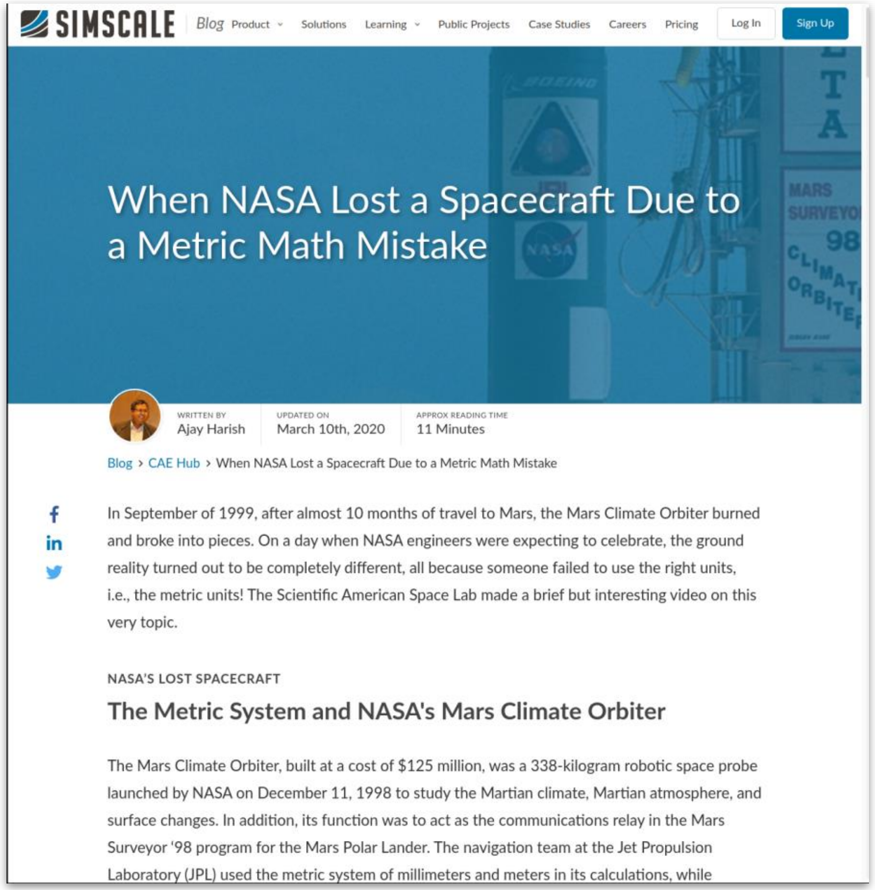
Is there a bug? Yes.

```
public String getDay(int dayIndex) {
  String day = null;
  switch (dayIndex) {
    case 0: day = "Monday";
    case 1: day = "Tuesday";
    case 2: day = "Wednesday";
    case 3: day = "Thursday";
  }
  return day;
}

public void example() {
  @NonNull String dayName = getDay(4);
  System.out.println("Today is " + dayName);
}
```

```
Error: [return.type.incompatible] incompatible types in return.
  type of expression: @Initialized @Nullable String
  method return type: @Initialized @NonNull String
```

Remember the Mars Climate Orbiter incident from 1999?



When NASA Lost a Spacecraft Due to a Metric Math Mistake

WRITTEN BY Ajay Harish | UPDATED ON March 10th, 2020 | APPROX READING TIME 11 Minutes

Blog › CAE Hub › When NASA Lost a Spacecraft Due to a Metric Math Mistake

In September of 1999, after almost 10 months of travel to Mars, the Mars Climate Orbiter burned and broke into pieces. On a day when NASA engineers were expecting to celebrate, the ground reality turned out to be completely different, all because someone failed to use the right units, i.e., the metric units! The Scientific American Space Lab made a brief but interesting video on this very topic.

NASA'S LOST SPACECRAFT

The Metric System and NASA's Mars Climate Orbiter

The Mars Climate Orbiter, built at a cost of $125 million, was a 338-kilogram robotic space probe launched by NASA on December 11, 1998 to study the Martian climate, Martian atmosphere, and surface changes. In addition, its function was to act as the communications relay in the Mars Surveyor '98 program for the Mars Polar Lander. The navigation team at the Jet Propulsion Laboratory (JPL) used the metric system of millimeters and meters in its calculations, while

NASA's Mars Climate Orbiter (cost of $327 million) was lost because of a discrepancy between use of metric unit Newtons and imperial measure Pound-force.

Units Checker identifies physical unit inconsistencies

- Guarantees that operations are performed on the same kinds and units
- Kind annotations
  - @Acceleration, @Angle, @Area, @Current, @Length, @Luminance, @Mass, @Speed, @Substance, @Temperature, @Time
- SI unit annotation
  - @m, @km, @mm, @kg, @mPERs, @mPERs2, @radians, @degrees @A, …

```
import static org.checkerframework.checker.units.UnitsTools.m;
import static org.checkerframework.checker.units.UnitsTools.mPERs;
import static org.checkerframework.checker.units.UnitsTools.s;

void demo() {
  @m int x;
  x = 5 * m;

  @m int meters = 5 * m;
  @s int seconds = 2 * s;

  @mPERs int speed = meters / seconds;
  @m int foo = meters + seconds;
  @s int bar = seconds - meters;
}
```

```
import static org.checkerframework.checker.units.UnitsTools.m;
import static org.checkerframework.checker.units.UnitsTools.mPERs;
import static org.checkerframework.checker.units.UnitsTools.s;

void demo() {
  @m int x;
  x = 5 * m;

  @m int meters = 5 * m;
  @s int seconds = 2 * s;

  @mPERs int speed = meters / seconds;
  @m int foo = meters + seconds;
  @s int bar = seconds - meters;
}
```

`@m` indicates that `x` represents meters

To assign a unit, multiply appropriate unit constant from `UnitTools`

Does this program compile?

```
import static org.checkerframework.checker.units.UnitsTools.m;
import static org.checkerframework.checker.units.UnitsTools.mPERs;
import static org.checkerframework.checker.units.UnitsTools.s;

void demo() {
  @m int x;
  x = 5 * m;

  @m int meters = 5 * m;
  @s int seconds = 2 * s;

  @mPERs int speed = meters / seconds;
  @m int foo = meters + seconds;
  @s int bar = seconds - meters;
}
```

`@m` indicates that `x` represents meters

To assign a unit, multiply appropriate unit constant from `UnitTools`

Does this program compile? No.

```
import static org.checkerframework.checker.units.UnitsTools.m;
import static org.checkerframework.checker.units.UnitsTools.mPERs;
import static org.checkerframework.checker.units.UnitsTools.s;

void demo() {
  @m int x;
  x = 5 * m;

  @m int meters = 5 * m;
  @s int seconds = 2 * s;

  @mPERs int speed = meters / seconds;
  @m int foo = meters + seconds;
  @s int bar = seconds - meters;
}
```

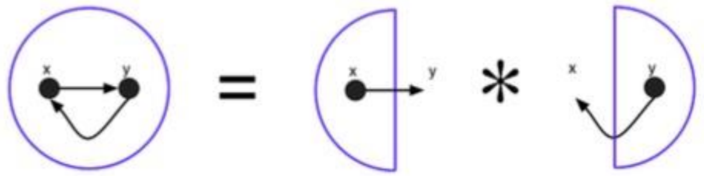
Addition and subtraction between meters and seconds is physically meaningless

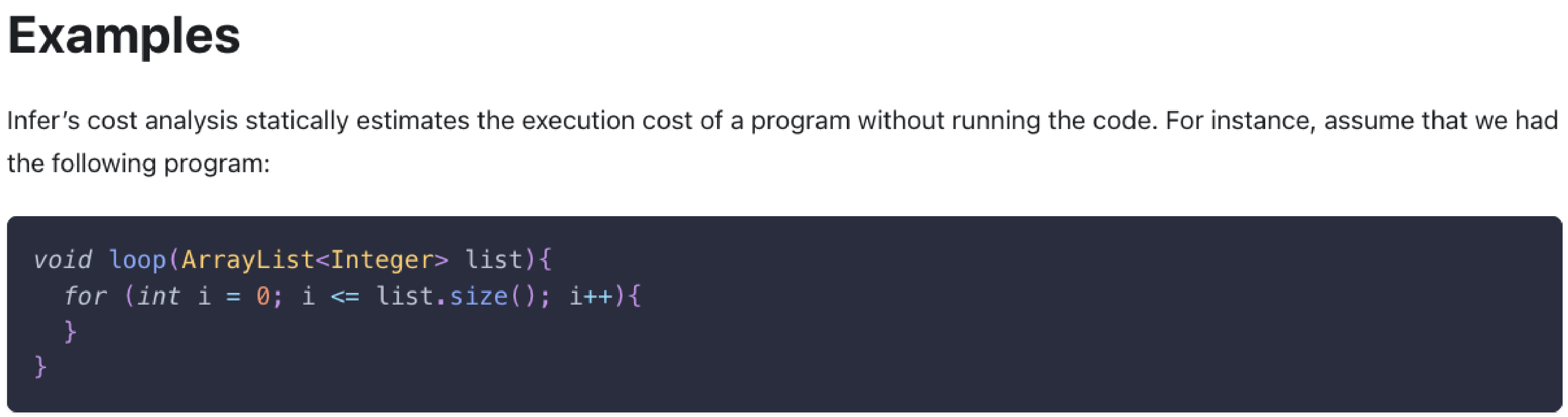
Checker Framework: Limitations

- Can only analyze code that is annotated
  - Requires that dependent libraries are also annotated
  - Can be tricky, but not impossible, to retrofit annotations into existing codebases
- Only considers the signature and annotations of methods
  - Doesn't look at the implementation of methods that are being called
- Dynamically generated code
  - Spring Framework
- Can produce false positives!
  - Byproduct of necessary approximations

Infer: What if we didn't want annotations?

- Focused on memory safety bugs
  - Null pointer dereferences, memory leaks, resource leaks, ...
- Compositional interprocedural reasoning
  - Based on separation logic and bi-abduction
- Scalable and fast
  - Can run incremental analysis on changed code
- Does not require annotations
- Supports multiple languages
  - Java, C, C++, Objective-C
  - Programs are compiled to an intermediate representation

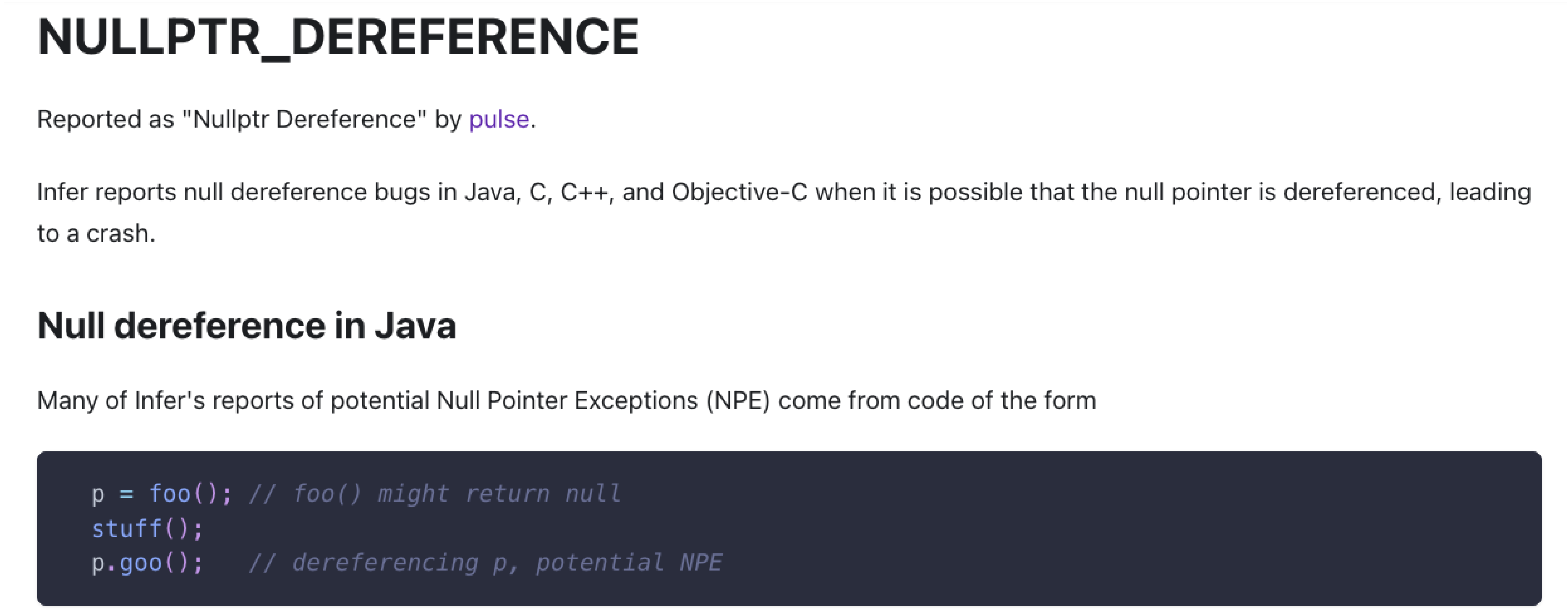




Examples

Infer's cost analysis statically estimates the execution cost of a program without running the code. For instance, assume that we had the following program:

```
void loop(ArrayList<Integer> list){
  for (int i = 0; i <= list.size(); i++){
  }
}
```



NULLPTR_DEREFERENCE

Reported as "Nullptr Dereference" by pulse.

Infer reports null dereference bugs in Java, C, C++, and Objective-C when it is possible that the null pointer is dereferenced, leading to a crash.

Null dereference in Java

Many of Infer's reports of potential Null Pointer Exceptions (NPE) come from code of the form

```
  p = foo(); // foo() might return null
  stuff();
  p.goo();   // dereferencing p, potential NPE
```

INVARIANT_CALL

Reported as "Invariant Call" by loop-hoisting.

We report this issue type when a function call is loop-invariant and hoistable, i.e.

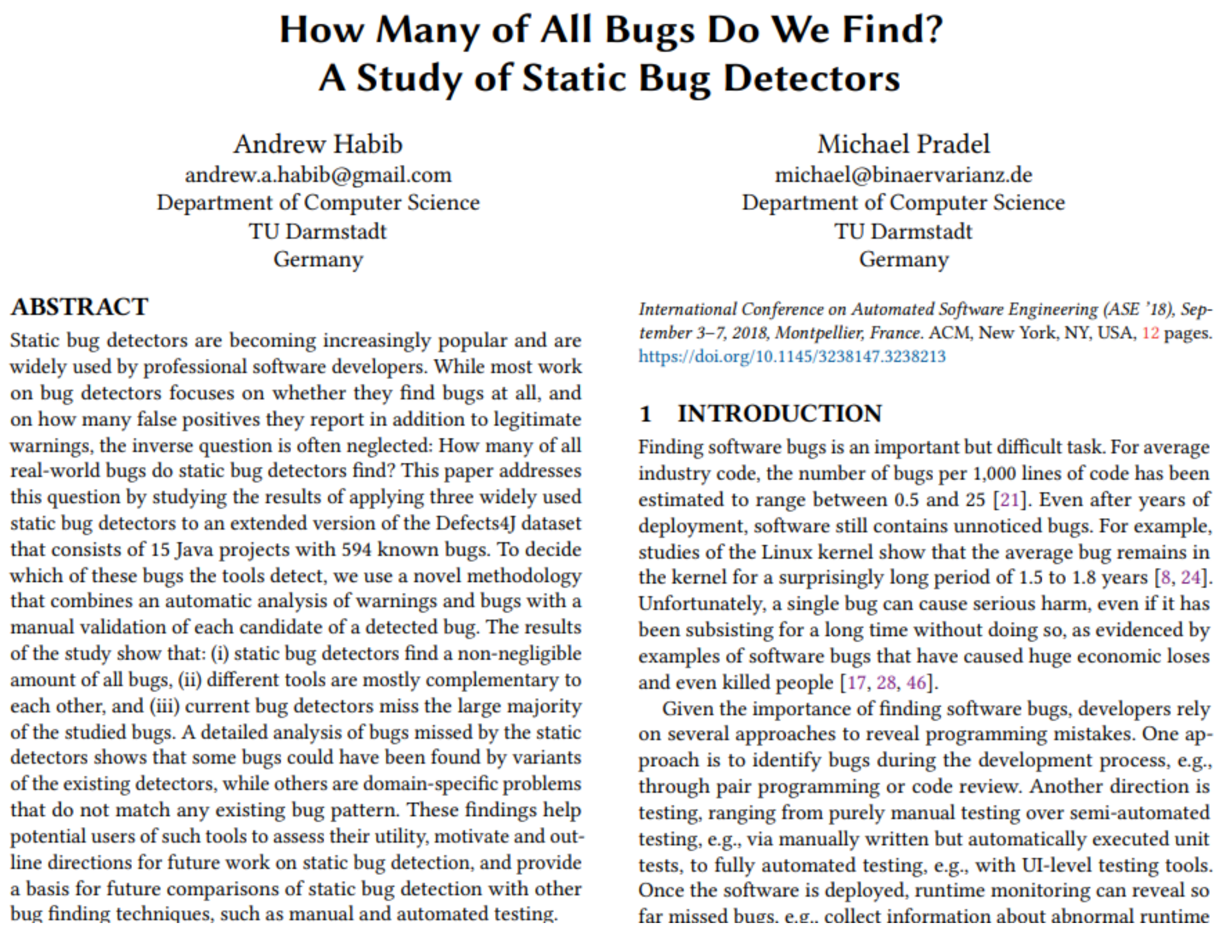
- the function has no side side effects (pure)
- has invariant arguments and result (i.e. have the same value in all loop iterations)
- it is guaranteed to execute, i.e. it dominates all loop sources

```
int foo(int x, int y) {
 return x + y;
}


void invariant_hoist(int size) {
    int x = 10;
    int y = 5;
    for (int i = 0; i < size; i++) {
      foo(x, y); // hoistable
    }
  }
```

Which tool to use?

The best QA strategies employ a combination of tools



How Many of All Bugs Do We Find? A Study of Static Bug Detectors

Andrew Habib
andrew.a.habib@gmail.com
Department of Computer Science
TU Darmstadt
Germany

Michael Pradel
michael@binaervarianz.de
Department of Computer Science
TU Darmstadt
Germany

ABSTRACT

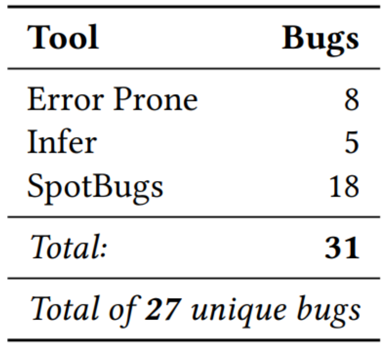
Static bug detectors are becoming increasingly popular and are widely used by professional software developers. While most work on bug detectors focuses on whether they find bugs at all, and on how many false positives they report in addition to legitimate warnings, the inverse question is often neglected: How many of all real-world bugs do static bug detectors find? This paper addresses this question by studying the results of applying three widely used static bug detectors to an extended version of the Defects4J dataset that consists of 15 Java projects with 594 known bugs. To decide which of these bugs the tools detect, we use a novel methodology that combines an automatic analysis of warnings and bugs with a manual validation of each candidate of a detected bug. The results of the study show that: (i) static bug detectors find a non-negligible amount of all bugs, (ii) different tools are mostly complementary to each other, and (iii) current bug detectors miss the large majority of the studied bugs. A detailed analysis of bugs missed by the static detectors shows that some bugs could have been found by variants of the existing detectors, while others are domain-specific problems that do not match any existing bug pattern. These findings help potential users of such tools to assess their utility, motivate and outline directions for future work on static bug detection, and provide a basis for future comparisons of static bug detection with other bug finding techniques, such as manual and automated testing.

International Conference on Automated Software Engineering (ASE '18), September 3–7, 2018, Montpellier, France. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3238147.3238213

1 INTRODUCTION

Finding software bugs is an important but difficult task. For average industry code, the number of bugs per 1,000 lines of code has been estimated to range between 0.5 and 25 [21]. Even after years of deployment, software still contains unnoticed bugs. For example, studies of the Linux kernel show that the average bug remains in the kernel for a surprisingly long period of 1.5 to 1.8 years [8, 24]. Unfortunately, a single bug can cause serious harm, even if it has been subsisting for a long time without doing so, as evidenced by examples of software bugs that have caused huge economic loses and even killed people [17, 28, 46].

Given the importance of finding software bugs, developers rely on several approaches to reveal programming mistakes. One approach is to identify bugs during the development process, e.g., through pair programming or code review. Another direction is testing, ranging from purely manual testing over semi-automated testing, e.g., via manually written but automatically executed unit tests, to fully automated testing, e.g., with UI-level testing tools. Once the software is deployed, runtime monitoring can reveal so far missed bugs, e.g., collect information about abnormal runtime



Tool	Bugs
Error Prone	8
Infer	5
SpotBugs	18
Total:	**31**
Total of 27 unique bugs	

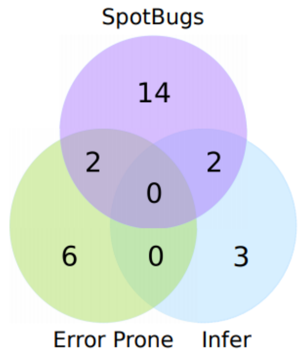


Figure 4: Total number of bugs found by all three static checkers and their overlap.

Which tool to use?

- Depends on use case, available resources
- **Linters**: Fast, cheap, easy to address issues or set ignore rules
- **Pattern-based bugs**: Very insightful, need to deal with false positives based on project domain
- **Type-annotation-based checkers**: More manual effort required; needs overall project/team commitment. But good payoff once adopted
- **Deep analysis tools**: Can find really tricky issues, but can be costly. Might need some understanding of how the tool works to deal with false positives.
- The best QA strategy involves multiple analysis and testing techniques