CS 4530: Fundamentals of Software Engineering

Lesson 9.3 Static Program Analysis

Jonathan Bell, Adeel Bhutta, Ferdinand Vesely, Mitch Wand
Khoury College of Computer Sciences

Learning Objectives for this Lesson

- By the end of this lesson, you should be able to:
  - Explain what a static analyzer is, and give some examples.
  - List some of the things that a static analyzer could do
  - Explain some limitations of static analyzers in practice.

A Static Analyzer checks the program without running it

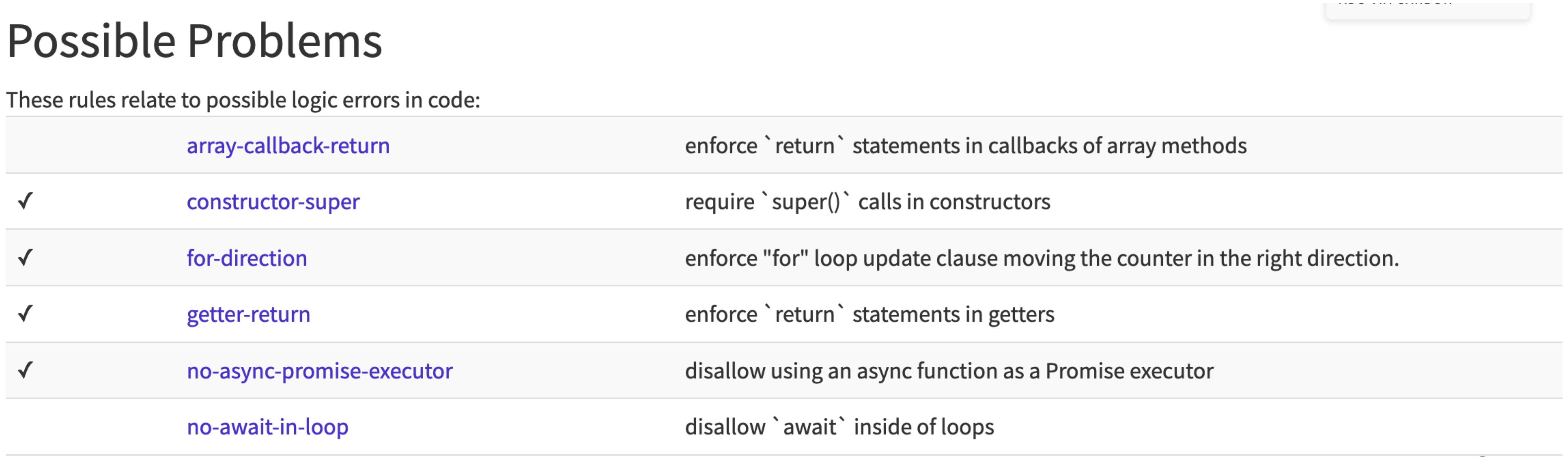
- It inspects the program to detect certain patterns in the code.
    - These could be patterns or anti-patterns.
    - Could be bugs or style conventions
- Examples:
    - The Typescript type checker
    - ESlint

The Type System is a Static Analyzer

- The type checker reads the program and checks to see that every place where a function is called, the given arguments match what the function expects.
- Benefits:
  - No pesky type errors at runtime
  - VSC runs the typechecker as you write, and provides explanations of the problems it finds.
  - Guaranteed 100% statement coverage, even in dead or rarely executed code.

ESLint is your (favorite|least favorite) static analyzer

- Linters = static analyzers for finding problematic patterns in code, stylistic errors, bugs
- ESLint is a popular linter for JavaScript
- Customizable via a set of rules



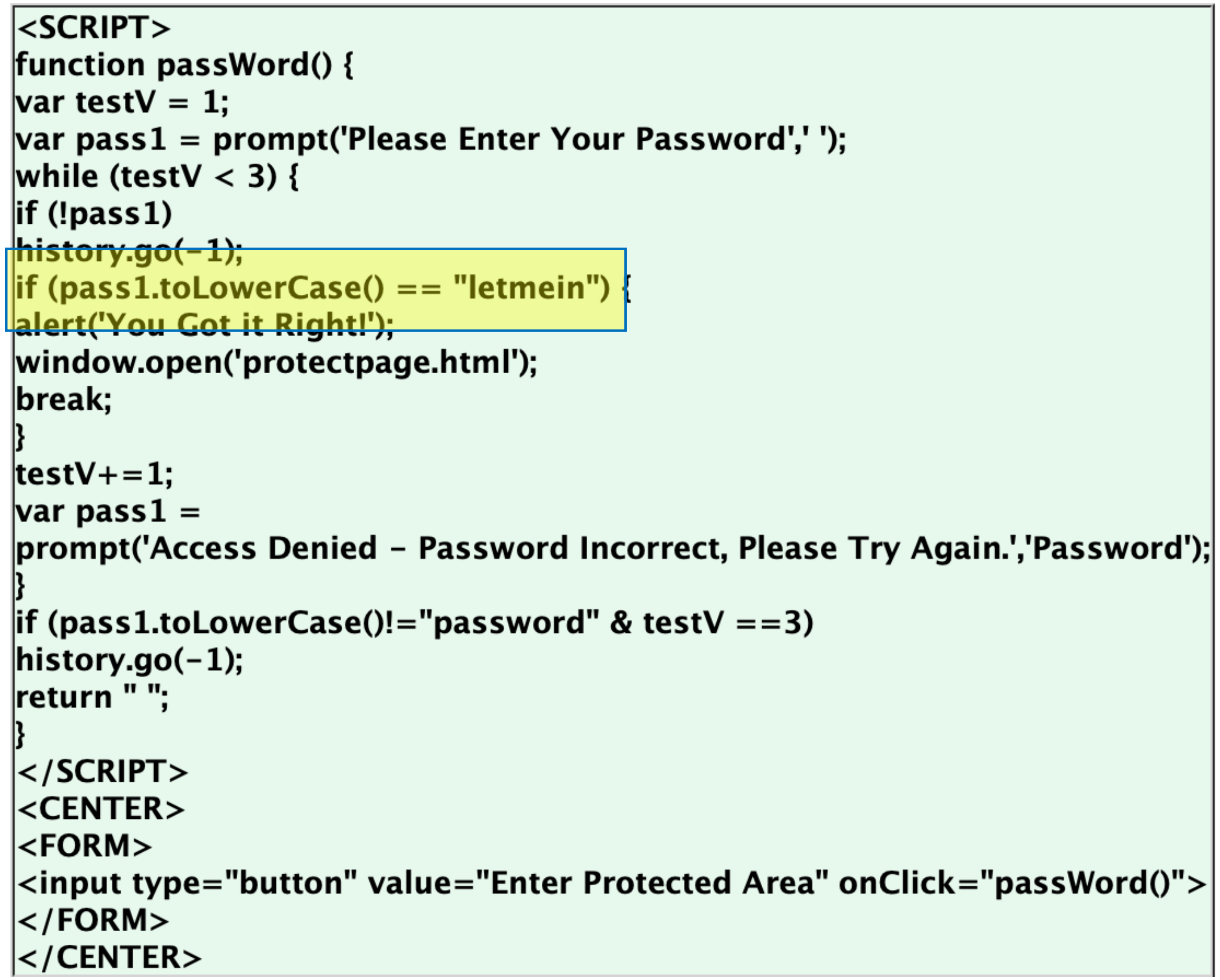
Possible Problems

These rules relate to possible logic errors in code:

	array-callback-return	enforce `return` statements in callbacks of array methods
✓	constructor-super	require `super()` calls in constructors
✓	for-direction	enforce "for" loop update clause moving the counter in the right direction.
✓	getter-return	enforce `return` statements in getters
✓	no-async-promise-executor	disallow using an async function as a Promise executor
	no-await-in-loop	disallow `await` inside of loops

Static Analyses can detect some dangerous anti-patterns

- CWE-798: Use of Hard-coded Credentials



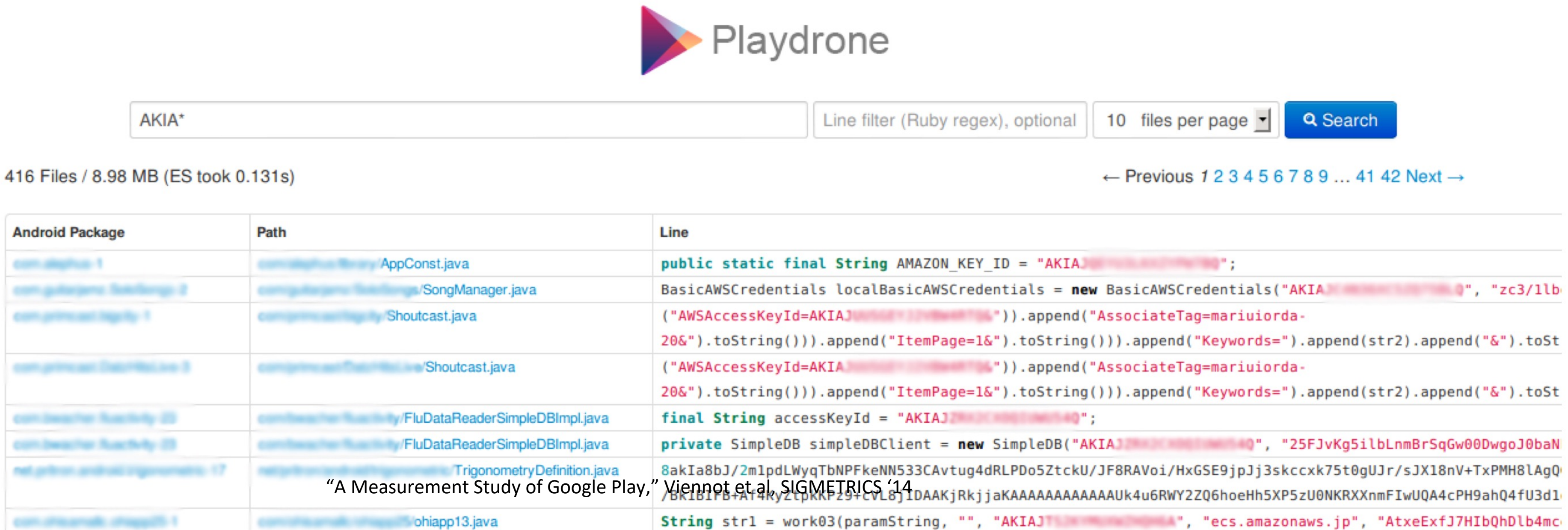
```
<SCRIPT>
function passWord() {
var testV = 1;
var pass1 = prompt('Please Enter Your Password',' ');
while (testV < 3) {
if (!pass1)
history.go(-1);
if (pass1.toLowerCase() == "letmein") {
alert('You Got it Right!');
window.open('protectpage.html');
break;
}
testV+=1;
var pass1 =
prompt('Access Denied - Password Incorrect, Please Try Again.','Password');
}
if (pass1.toLowerCase()!="password" & testV ==3)
history.go(-1);
return " ";
}
</SCRIPT>
<CENTER>
<FORM>
<input type="button" value="Enter Protected Area" onClick="passWord()">
</FORM>
</CENTER>
```

Static Analysis showed this kind of fault was widespread

- CWE-798: Use of Hard-coded Credentials: Study of 1.1m Android Apps

	Amazon	Facebook	Twitter	Bitly	Flickr	Foursquare	Google	LinkedIn	Titanium
Total candidates	1,241	1,477	28,235	3,132	159	326	414	1,434	1,914
Unique candidates	308	460	6,228	616	89	177	225	181	1,783
Unique % valid	93.5%	71.7%	95.2%	88.8%	100%	97.7%	96.0%	97.2%	99.8%

Table 5: Credentials statistics from June 22, 2013 and validated on November 11, 2013. A credential may consist of an ID token and secret authentication token.



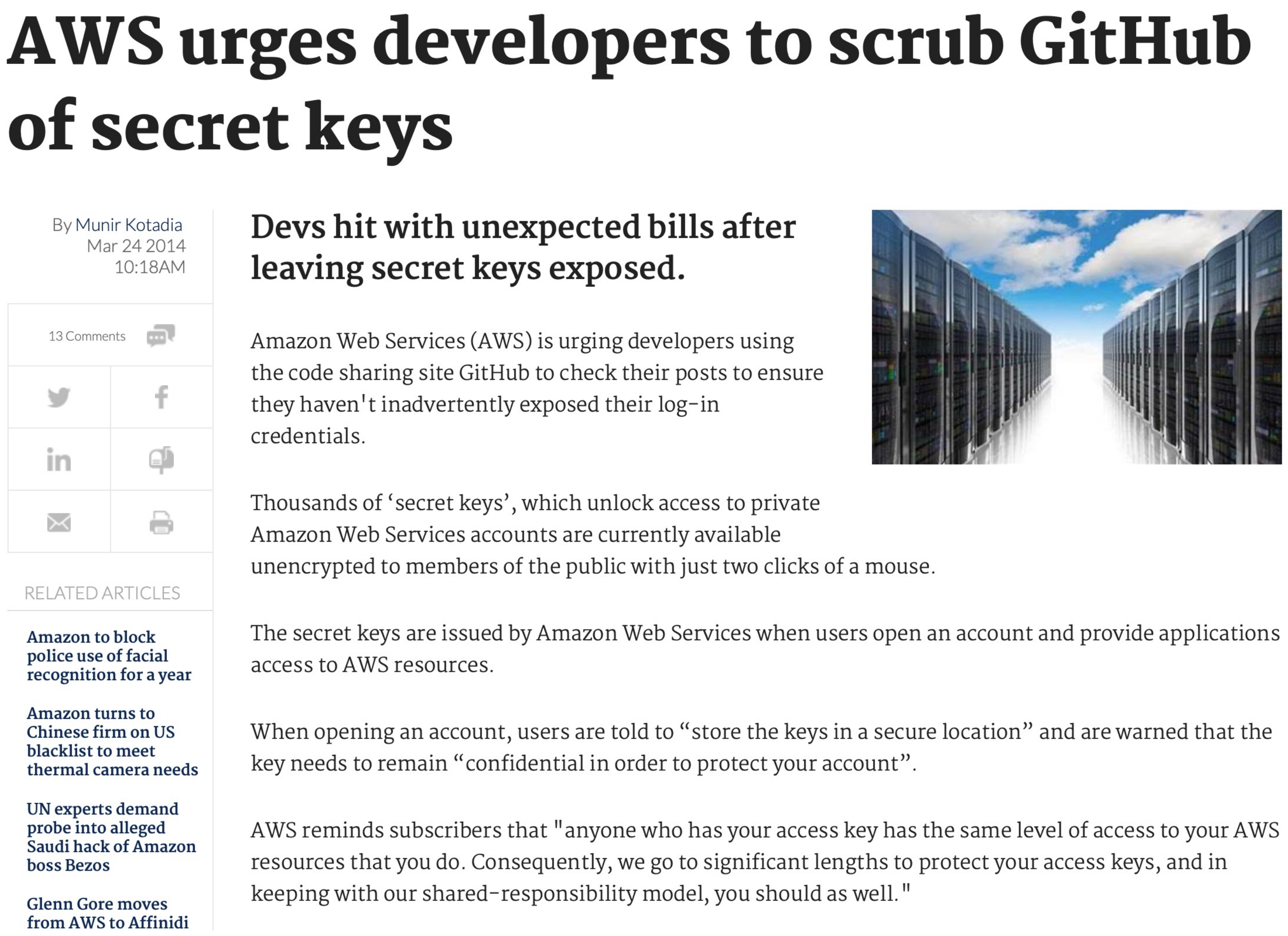
416 Files / 8.98 MB (ES took 0.131s)

← Previous 1 2 3 4 5 6 7 8 9 … 41 42 Next →

Android Package	Path	Line
[redacted]	[redacted]/AppConst.java	`public static final String AMAZON_KEY_ID = "AKIA[redacted]";`
[redacted]	[redacted]/SongManager.java	`BasicAWSCredentials localBasicAWSCredentials = new BasicAWSCredentials("AKIA[redacted]", "zc3/1lb`
[redacted]	[redacted]/Shoutcast.java	`("AWSAccessKeyId=AKIAJ[redacted]")).append("AssociateTag=mariuiorda-20&").toString())).append("ItemPage=1&").toString())).append("Keywords=").append(str2).append("&").toSt`
[redacted]	[redacted]/Shoutcast.java	`("AWSAccessKeyId=AKIA[redacted]")).append("AssociateTag=mariuiorda-20&").toString())).append("ItemPage=1&").toString())).append("Keywords=").append(str2).append("&").toSt`
[redacted]	[redacted]/FluDataReaderSimpleDBImpl.java	`final String accessKeyId = "AKIAJ[redacted]";`
[redacted]	[redacted]/FluDataReaderSimpleDBImpl.java	`private SimpleDB simpleDBClient = new SimpleDB("AKIAJ[redacted]", "25FJvKg5ilbLnmBrSqGw00DwgoJ0baN`
[redacted]	[redacted]/TrigonometryDefinition.java	`8akIa8bJ/2mlpdLWyqTbNPFkeNN533CAvtug4dRLPDo5ZtckU/JF8RAVoi/HxGSE9jpJj3skccxk75t0gUJr/sJX18nV+TxPMH8lAgQ /BkIBlFB+Af4kyZtpkKP29+cVL8j1DAAKjRkjjaKAAAAAAAAAAAAUk4u6RWY2ZQ6hoeHh5XP5zU0NKRXXnmFIwUQA4cPH9ahQ4fU3dl`
[redacted]	[redacted]/ohiapp13.java	`String str1 = work03(paramString, "", "AKIAJ[redacted]", "ecs.amazonaws.jp", "AtxeExfJ7HIbQhDlb4mc`

“A Measurement Study of Google Play,” Viennot et al, SIGMETRICS ‘14

This discovery led to action.



AWS urges developers to scrub GitHub of secret keys

By Munir Kotadia
Mar 24 2014
10:18AM

13 Comments

RELATED ARTICLES

Amazon to block police use of facial recognition for a year

Amazon turns to Chinese firm on US blacklist to meet thermal camera needs

UN experts demand probe into alleged Saudi hack of Amazon boss Bezos

Glenn Gore moves from AWS to Affinidi

Devs hit with unexpected bills after leaving secret keys exposed.

Amazon Web Services (AWS) is urging developers using the code sharing site GitHub to check their posts to ensure they haven't inadvertently exposed their log-in credentials.

Thousands of ‘secret keys’, which unlock access to private Amazon Web Services accounts are currently available unencrypted to members of the public with just two clicks of a mouse.

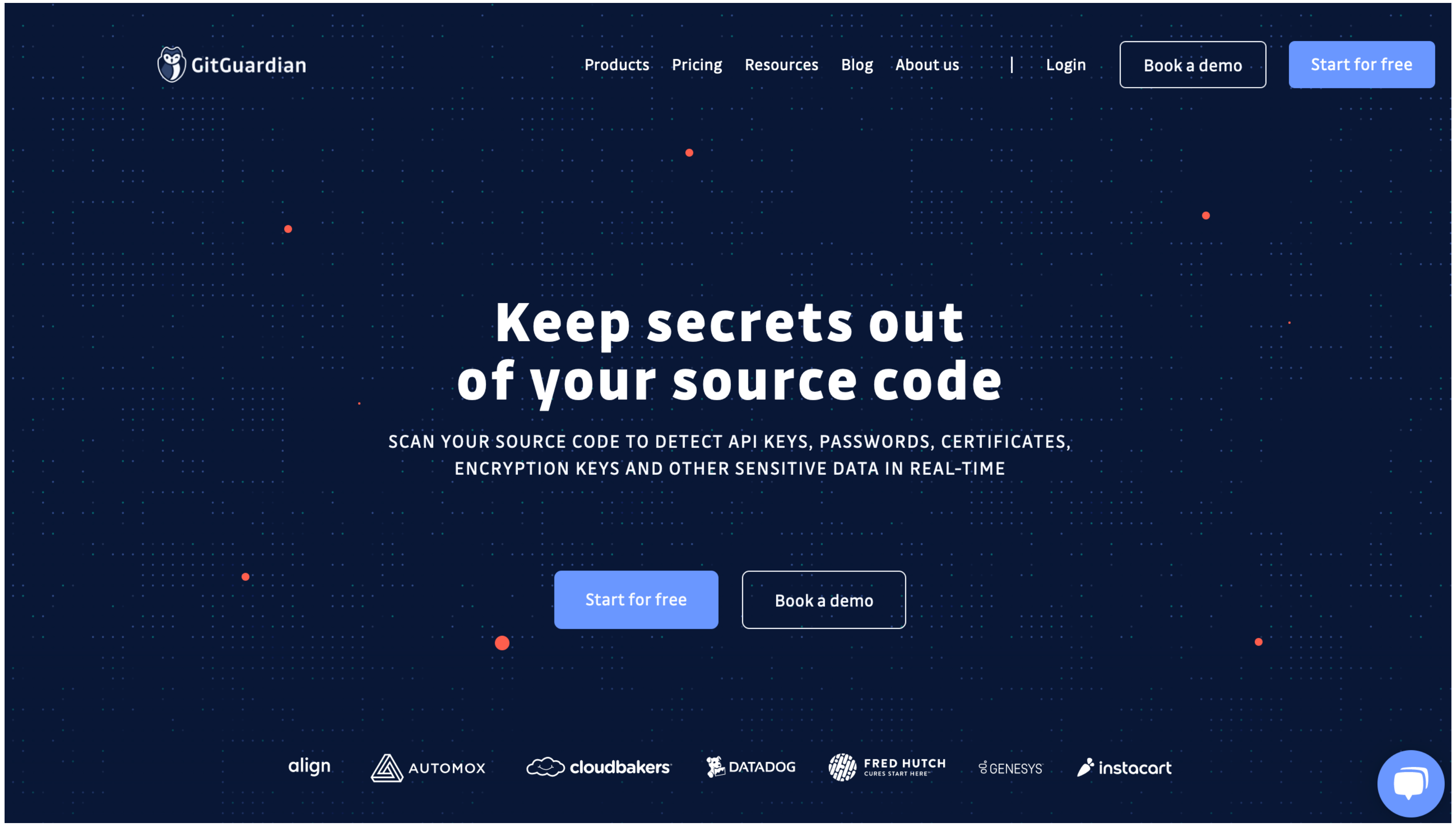
The secret keys are issued by Amazon Web Services when users open an account and provide applications access to AWS resources.

When opening an account, users are told to “store the keys in a secure location” and are warned that the key needs to remain “confidential in order to protect your account”.

AWS reminds subscribers that "anyone who has your access key has the same level of access to your AWS resources that you do. Consequently, we go to significant lengths to protect your access keys, and in keeping with our shared-responsibility model, you should as well."

That example wasn't hard: the tool just looked for a certain regex.

- GitGuardian (Launched in 2017)

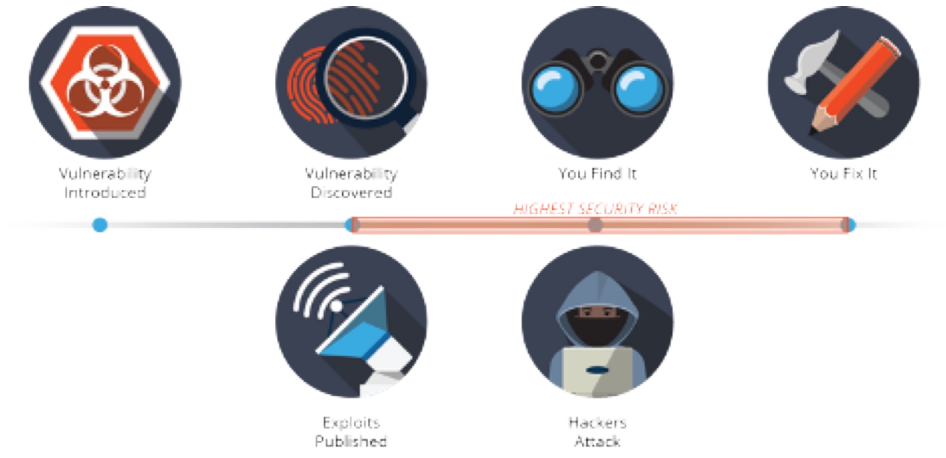


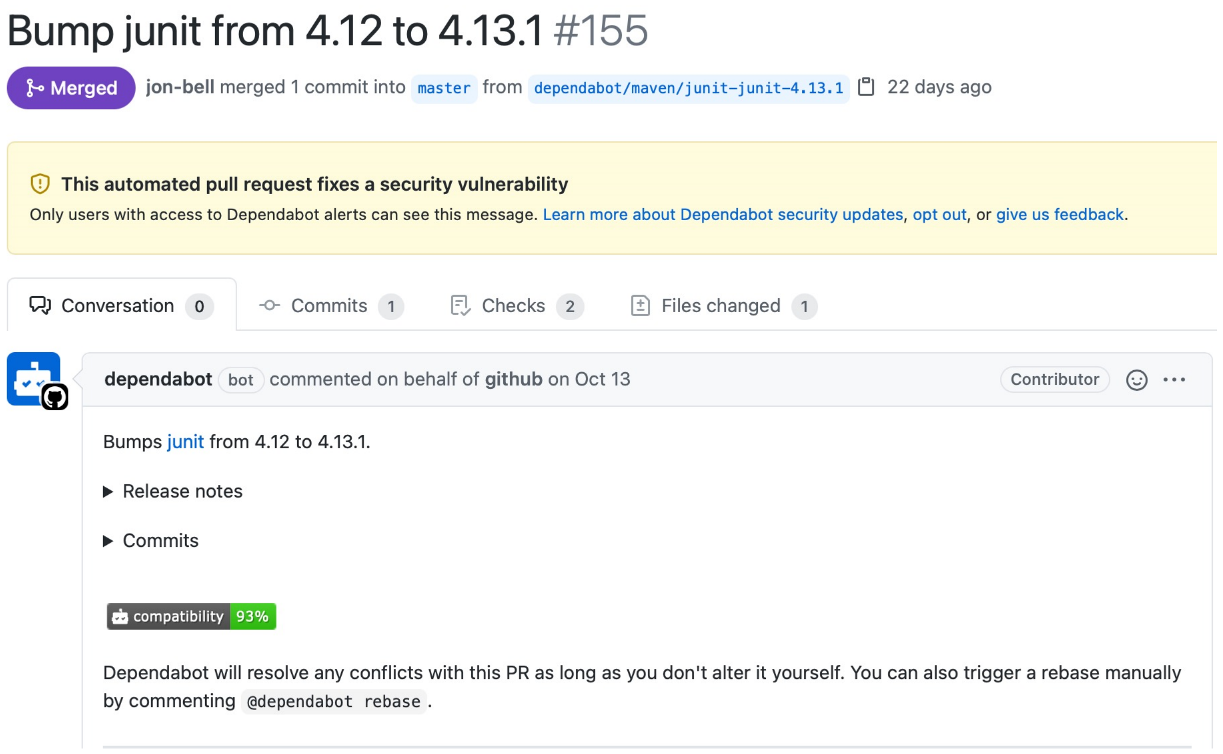
Keep secrets out of your source code

SCAN YOUR SOURCE CODE TO DETECT API KEYS, PASSWORDS, CERTIFICATES, ENCRYPTION KEYS AND OTHER SENSITIVE DATA IN REAL-TIME

Out-of-date dependencies may contain vulnerabilities

- A9:2017-Using Components with Known Vulnerabilities





Bump junit from 4.12 to 4.13.1 #155

Merged jon-bell merged 1 commit into `master` from `dependabot/maven/junit-junit-4.13.1` 22 days ago

This automated pull request fixes a security vulnerability

Only users with access to Dependabot alerts can see this message. Learn more about Dependabot security updates, opt out, or give us feedback.

Conversation 0 | Commits 1 | Checks 2 | Files changed 1

dependabot bot commented on behalf of **github** on Oct 13 — Contributor

Bumps junit from 4.12 to 4.13.1.

▶ Release notes

▶ Commits

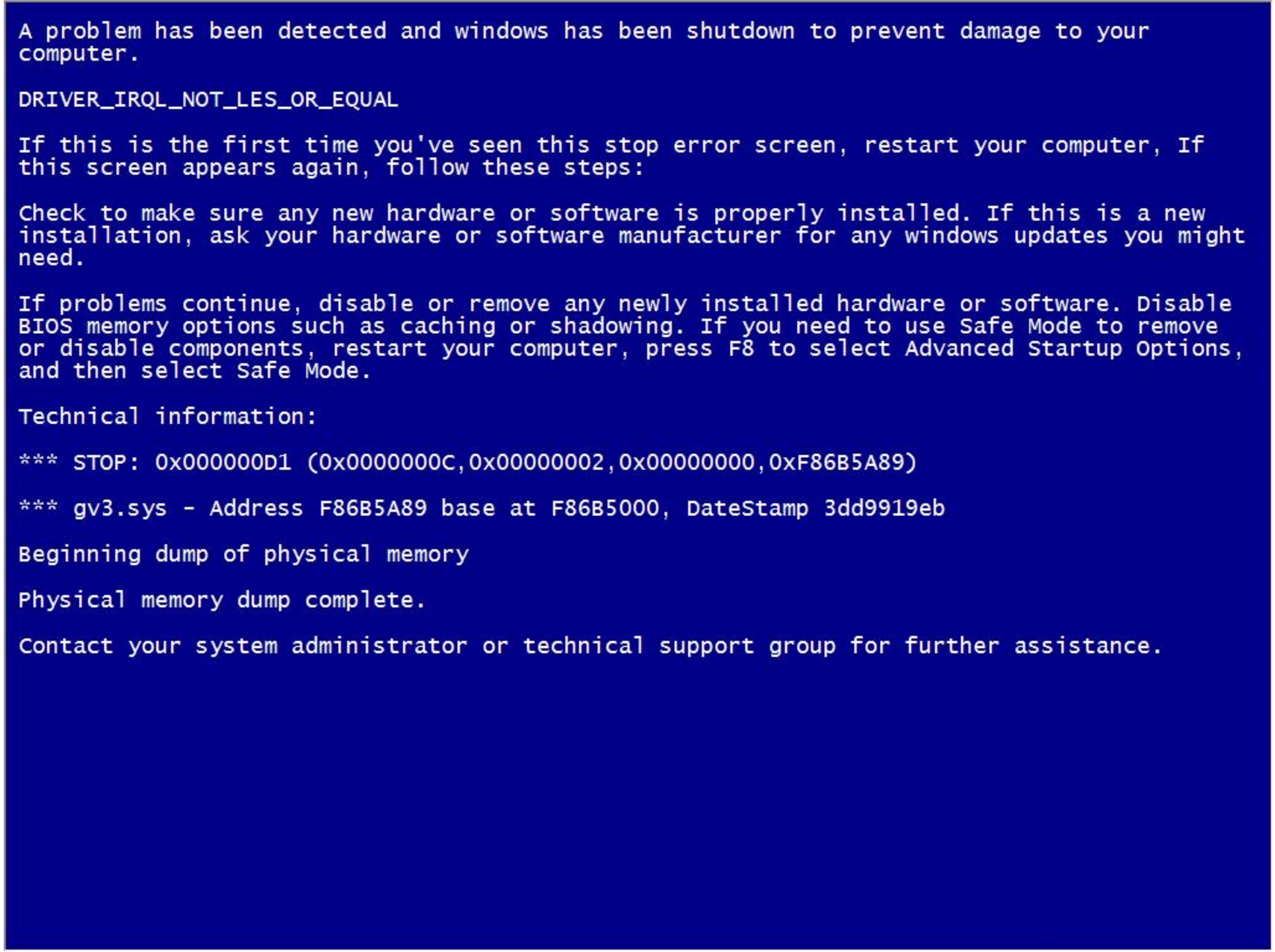
compatibility 93%

Dependabot will resolve any conflicts with this PR as long as you don't alter it yourself. You can also trigger a rebase manually by commenting `@dependabot rebase`.

Defects Static Analysis can Catch

- Defects that result from inconsistently following simple, mechanical design rules.
  - **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

Key: check compliance to simple, mechanical design rules



```
A problem has been detected and windows has been shutdown to prevent damage to your
computer.

DRIVER_IRQL_NOT_LES_OR_EQUAL

If this is the first time you've seen this stop error screen, restart your computer, If
this screen appears again, follow these steps:

Check to make sure any new hardware or software is properly installed. If this is a new
installation, ask your hardware or software manufacturer for any windows updates you might
need.

If problems continue, disable or remove any newly installed hardware or software. Disable
BIOS memory options such as caching or shadowing. If you need to use Safe Mode to remove
or disable components, restart your computer, press F8 to select Advanced Startup Options,
and then select Safe Mode.

Technical information:

*** STOP: 0x000000D1 (0x0000000C,0x00000002,0x00000000,0xF86B5A89)

*** gv3.sys - Address F86B5A89 base at F86B5000, DateStamp 3dd9919eb

Beginning dump of physical memory

Physical memory dump complete.

Contact your system administrator or technical support group for further assistance.
```

Static Analysis isn’t perfect

- Example: “every opened file is eventually closed”
- An analysis could miss some closes of opened files
- Or it could miss some open file not getting closed.

Getting precise results may require compromise

- Getting precise results may take **time**:
  - Some algorithms take exponential time.
  - Practical algorithms are limited to linear time
- Getting precise results may require **whole** program:
  - If parts of the program loaded at runtime:
    - Analysis results may be very imprecise, or (worse)
    - Incorrect, if they assume the whole program is available.
- Getting precise results may require intervention:
  - Code may need to be **annotated** with information:
    - E.g., “this method may return an open resource.”
    - E.g., type annotations

What happens when an analysis is imprecise?

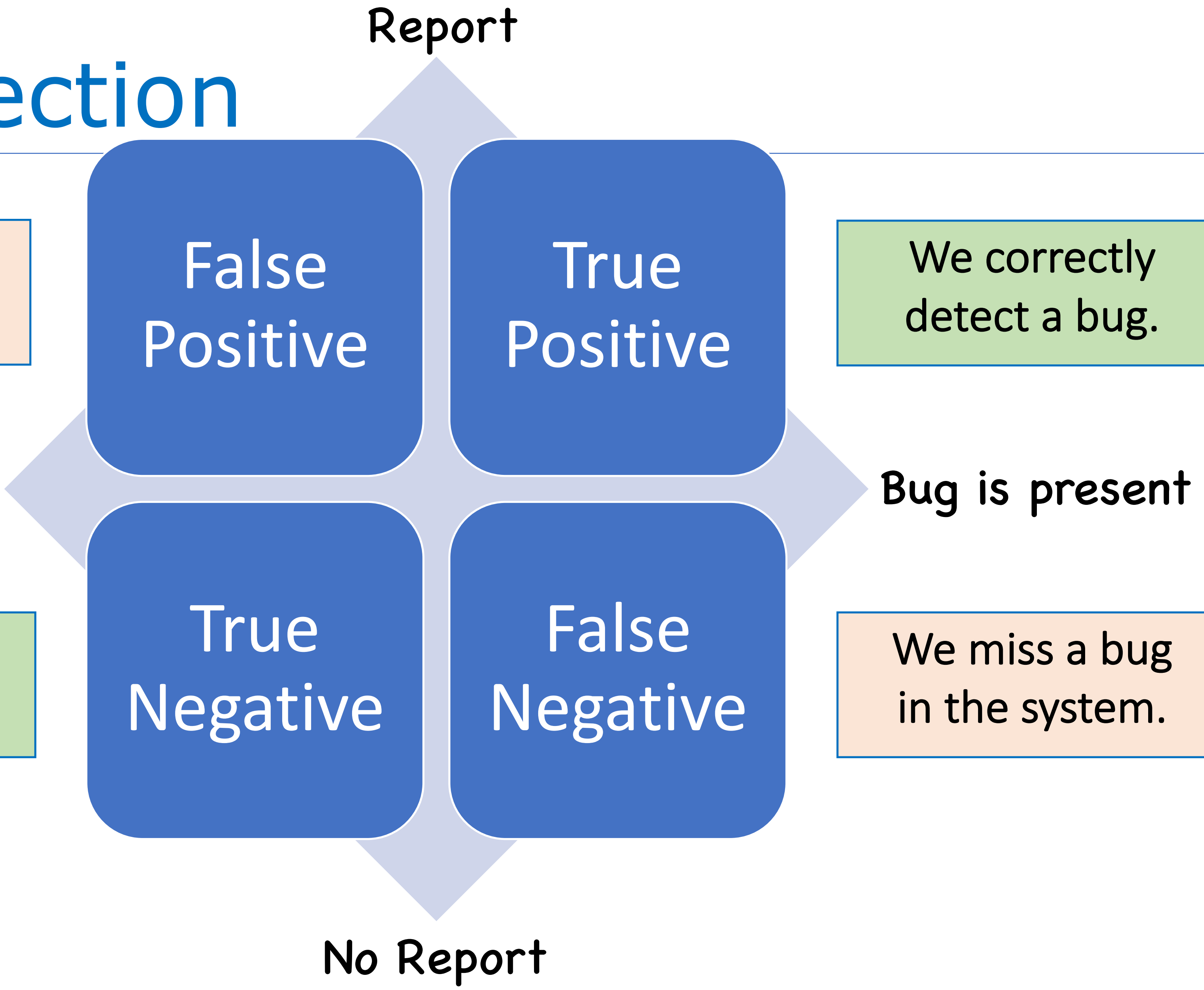
FALSE NEGATIVES

- The static analysis misses something ”bad” in program:
- Can give a false sense of security.
- Can be reduced, but at the cost of false positives!

FALSE POSITIVES

- The static analysis reports a problem that doesn’t exist:
- Real bugs can be swamped by a flood of spurious reports.
- Programmer time is wasted chasing down false leads.

Bug Detection



We generate a false alarm.

We correctly detect a bug.

We correctly find no problem.

We miss a bug in the system.

The fate of a bug report depends on more than whether or not it is accurate

- A report from static analysis is effectively false,
  - If it is ignored by developers;
  - Whether or not it represents a true bug.
- Even if the report is technically correct
  - It may refer to something considered unimportant:
    - E.g., who cares if all the files aren't closed, if the program is about to exit anyway.
    - E.g., yes, there is a race condition between two logging statements, but that's not important.
- Even if the report is technically wrong
  - Developers may see potential problem, and fix.

Criteria For Good Automated Program Analysis

- Efficient and Easy
  - Should not require whole program or annotations.
  - Should be automatically applied as part of workflow
- Rarely spurious
  - No more than 10% effectively false positive.
- Actionable
  - Should point out things easy to fix.
- Effective
  - Problems should be perceived as important.

Source: Software Engineering at Google, Chapter 20

Could static analyses replace testing?

- Short answer: no, not anytime soon.
- There are ways for mathematically proving that a program satisfies its specification
- But:
    1. You need a specification
    2. Existing proof techniques don’t scale
    3. And your proof is no better than your specification
- Go take a course in Formal Methods
    - Khoury has a whole research group on this

Review: Learning Objectives for this Lesson

- You should now be able to:
    - Explain what a static analyzer is, and give some examples.
    - List some of the things that a static analyzer could do
    - Explain some limitations of static analyzers in practice.