

SonarQube Training

Session #1 - Overview

A journey in the land of code quality and security



Agenda

Session #1 - Overview

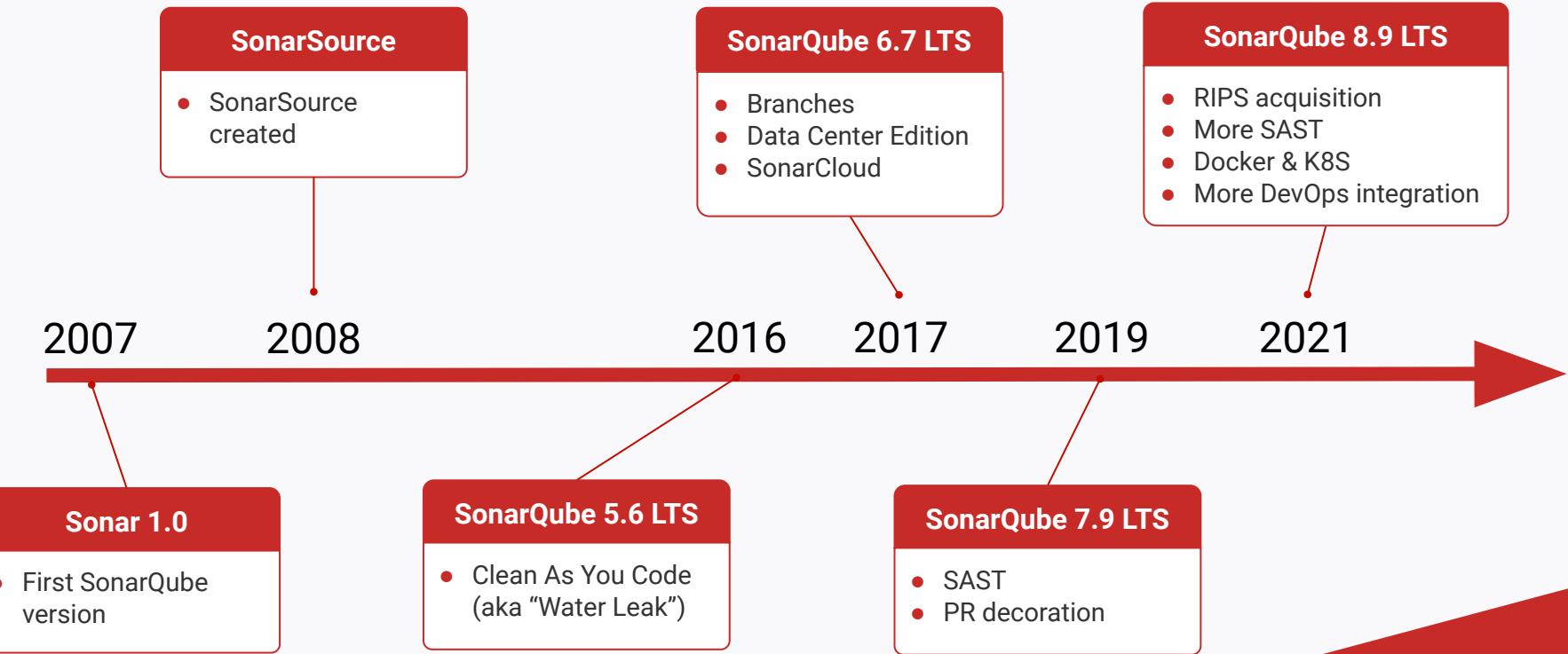
- SonarSource's approach to Code Quality and Security
- The *Clean As You Code* paradigm
- Security overview
- Demo of SonarSource's own production SonarQube platform
- Demo of SonarLint
- Platform walkthrough: Show your production platform, how you configured and used it so far
- Q&A

SonarSource's approach

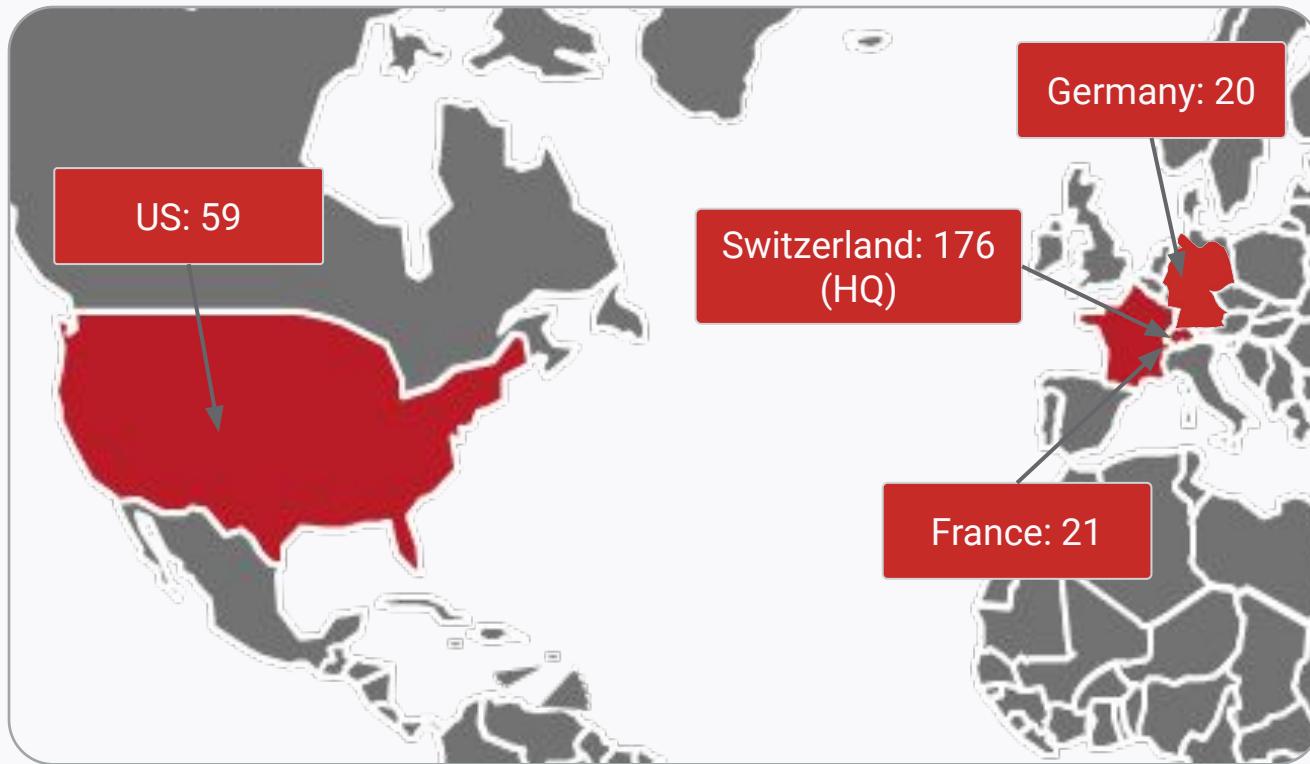
To code quality and security



History



Our Team





A lively community

	Today	Yesterday	Last 7	Last 30
Pageviews	9.1k	30.7k ▲	140k ▲	609k ▼
User Visits	133	311 ▲	1.6k ▲	7.0k ▲
Time to first response	1	5 ▼	26 ▼	37 ▼
Likes	17	41 ▲	173 ▲	516
Flags	0	1 ▲	1 ▲	2
User-to-User (with r...)	4	4 ▲	43 ▲	129



Users per Trust Level ?

0 7.4k 1 3.2k 2 69 3 1 4 177

<https://community.sonarsource.com>



Software (and SonarQube) is everywhere

Customer diversity is good for our rules

Aerospace & Defence



Automotive



E-commerce



Energy



15'000+ commercial customers

Financial Services



Healthcare



Media



Public



Transport & Logistics



Retail



Technology



Telecommunications



Our mission

**Every developer and development
team uses SonarSource for
code quality and security**

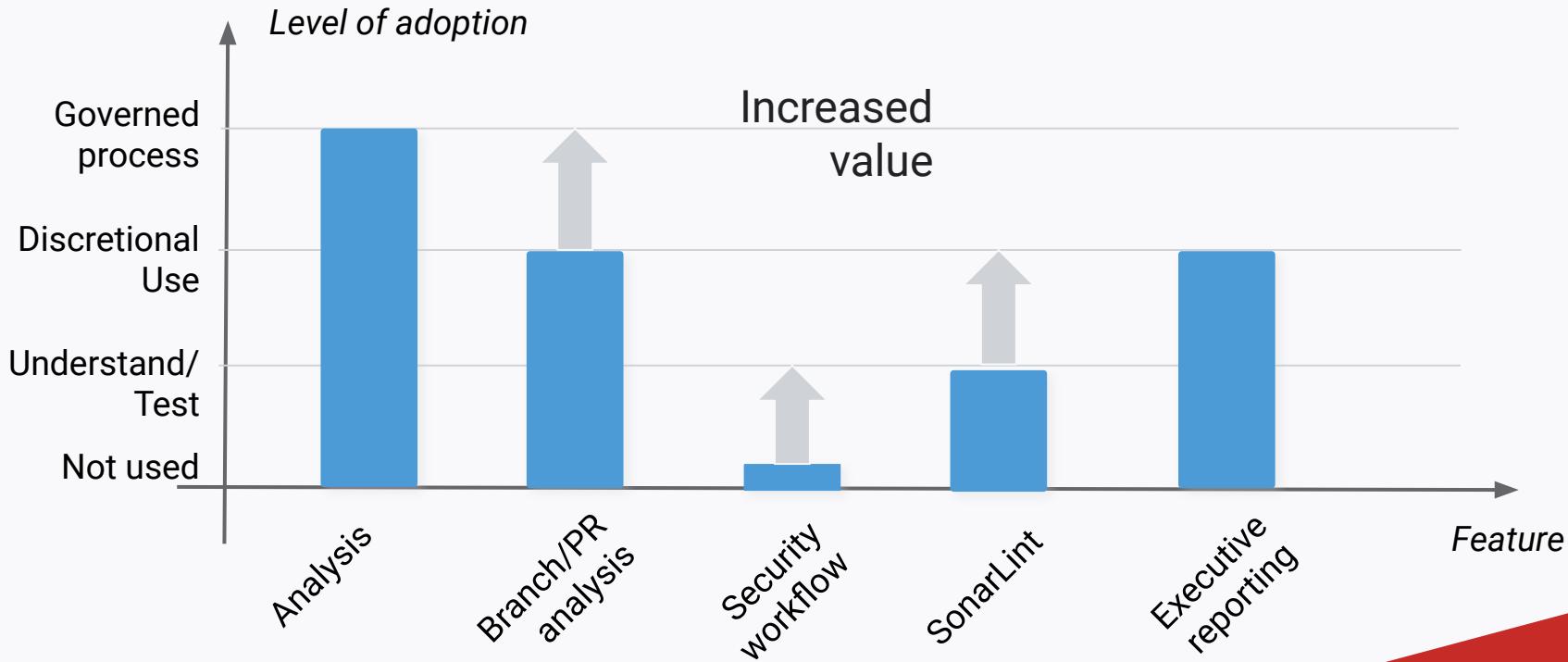
SonarQube Quality Model





Getting the most of SonarQube

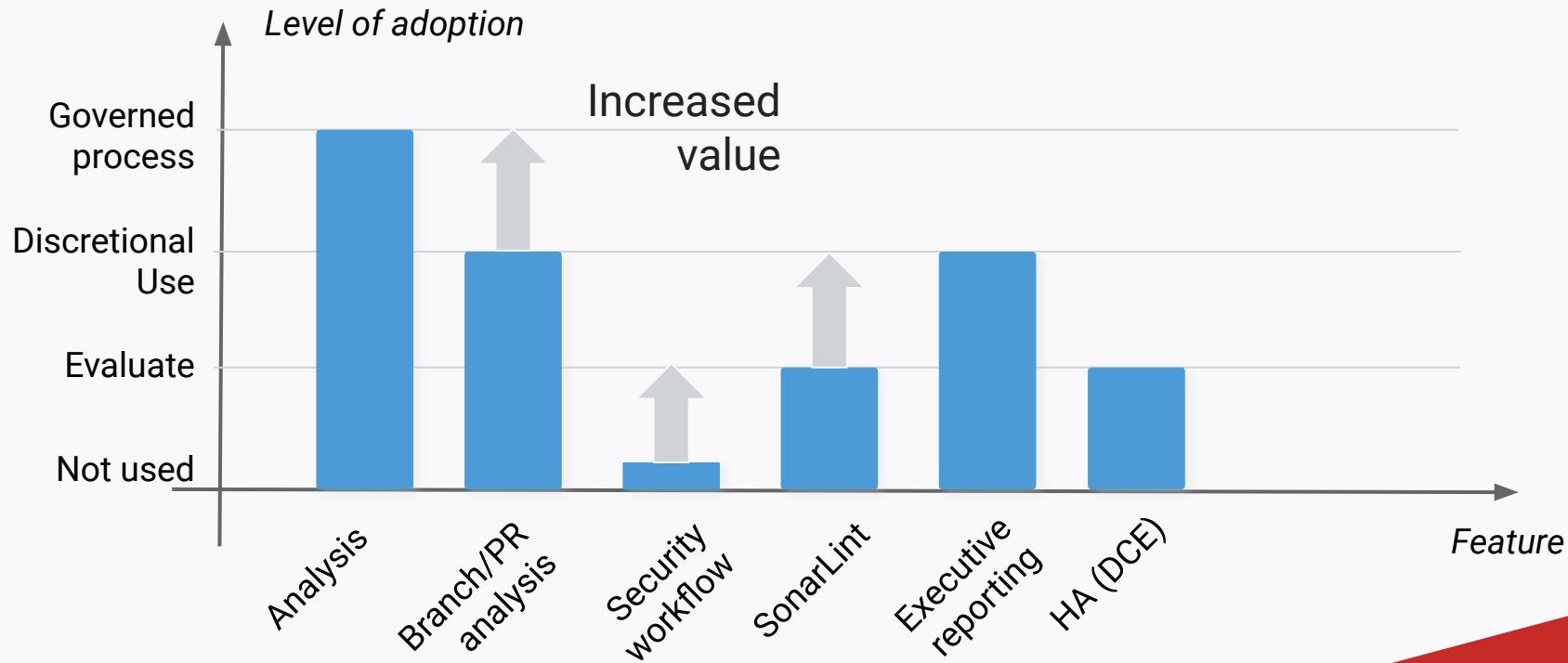
What you get from SonarQube depends on what you make of it





Getting the most of SonarQube

What you get from SonarQube depends on what you make of it





Key Feature - Quality Gate

PLEASE STAND BEHIND
THE YELLOW LINE

© Alexander Dudley 2004



Key Feature - Clean As You Code

- Track **new** issues
- Compute metrics on **new** code
 - Coverage
 - Duplications

```
    return this;
}
public String[] getReturningAttributes() {
    return returningAttributes;
}
/**
 * @throws NamingException if unable to perform search
 */
public NamingEnumeration<SearchResult> find(
    LOG.debug("Search: {}", this);
    NamingEnumeration<SearchResult> result;
    InitialDirContext context = null;
    boolean threw = false;
    try {
        context = contextFactory.createBindContext(principal, password);
        SearchControls controls = new SearchControls();
        controls.setSearchScope(SearchControls.SUBTREE_SCOPE);
        controls.setReturningAttributes(returningAttributes);
        result = context.search(baseDN, request, controls);
        threw = true;
    } finally {
        ContextHelper.close(context, threw);
    }
    return result;
}
```

Legacy Code

```
private boolean checkPasswordUsingBind(String principal, String password, String ldapKey) {
    if (StringUtils.isEmpty(password)) {
        LOG.debug("Password is blank.");
        return false;
    }
    InitialDirContext context = null;
    try {
        context = contextFactories.get(ldapKey).createUserContext(principal, password);
        result = context.search(baseDN, request, controls);
    } catch (NamingException e) {
        LOG.debug("Password not valid for user {} in server {}", principal, ldapKey, e.getMessage());
    } finally {
        ContextHelper.closeQuietly(context);
    }
}
private boolean checkPasswordUsingGssapi(String principal, String password, String ldapKey) {
    // Use our custom configuration to avoid reliance on external config
    Configuration.setConfiguration(new Krb5LoginConfiguration());
    LoginContext lc;
    int i;
    for (i = 0; i < 10; i++) {
        // do Something
    }
    try {
        lc = new LoginContext(getClass().getName(), new CallbackHandlerImpl(principal, password));
        lc.login();
    } catch (LoginException e) {
        // Bad username: Client not found in Kerberos database
        // Bad password: Integrity check on decrypted field failed
        LOG.debug("Password not valid for {} in server {}", principal, ldapKey, e.getMessage());
        return false;
    }
    try {
        lc.logout();
    } catch (LoginException e) {
        LOG.warn("Logout fails", e);
    }
}
```

New Code



Key Feature - SAST

- Detect Injection vulnerabilities
- SonarQube UI shows tainted data flow
- OWASP / CWE reporting
- Other vulnerabilities
(data exposure, poor practices)
- Security Hotspots

The screenshot shows two code review panels from SonarQube. The top panel displays code for `Servlet.java` with several annotations:

- Line 26: `String name = 1 taintedRequest.getParameter(FIELD_NAME);` annotated with **Vulnerability +6**.
- Line 27: `taintedString = name;` annotated with **Vulnerability +6**.
- Line 31: `taintedString = 4 java.net.URLDecoder.decode(taintedString);` annotated with **Vulnerability +6**.
- Line 34: `try { 5 new SQLInjectionVulnerability(taintedString);` annotated with **Vulnerability +6**.
- Line 35: `new CommandInjectionVulnerability(taintedString);` annotated with **Vulnerability +6**.
- Line 36: `new RegexInjectionVulnerability(taintedString);` annotated with **Vulnerability +6**.

The bottom panel displays code for `SQLInjectionVulnerability.java` with annotations:

- Line 18: `stmt.execute(6 SELECT_SQL + uname);` annotated with **Vulnerability Blocker Open Not assigned 30min effort**.

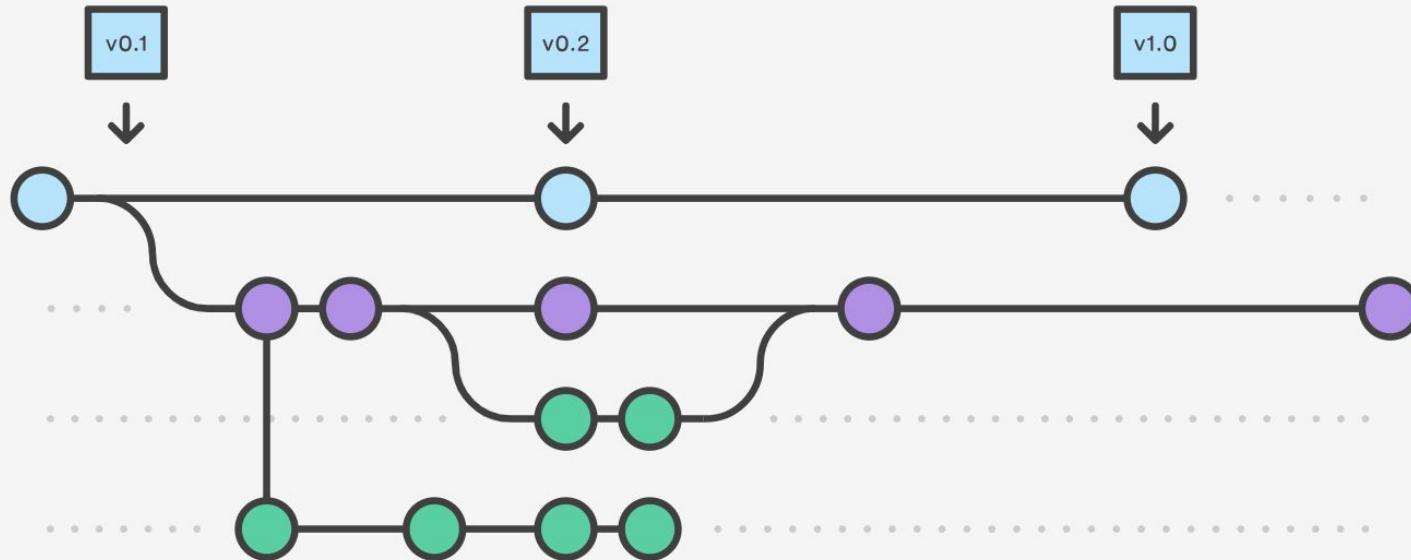
A sidebar on the right provides a summary of taint propagation:

- source: taint value is propagated
- taint value is propagated

At the bottom, a note says "Refactor this code to not construct SQL queries directly from tainted user-controlled data." with a link "See Rule".



Key Feature - Branch & PR support





Key Feature - Documented rules

Vulnerability

Critical

cert, cwe, owasp-a6, sans-top25-porous ▾

Available Since June 13, 2018

SonarAnalyzer (Java)

Constant/issue: 2min

The Advanced Encryption Standard (AES) encryption algorithm can be used with various modes. Some combinations are not secured:

- Electronic Codebook (ECB) mode: Under a given key, any given plaintext block always gets encrypted to the same ciphertext block. Thus, it does not hide data patterns well. In some senses, it doesn't provide serious message confidentiality, and it is not recommended for use in cryptographic protocols at all.
- Cipher Block Chaining (CBC) with PKCS#5 padding (or PKCS#7) is susceptible to padding oracle attacks.

In both cases, Galois/Counter Mode (GCM) with no padding should be preferred.

This rule raises an issue when a `Cipher` instance is created with either ECB or CBC/PKCS5Padding mode.

Noncompliant Code Example

```
Cipher c1 = Cipher.getInstance("AES/ECB/NoPadding"); // Noncompliant
Cipher c2 = Cipher.getInstance("AES/CBC/PKCS5Padding"); // Noncompliant
```

Compliant Solution

```
Cipher c = Cipher.getInstance("AES/GCM/NoPadding");
```

See

- [MITRE, CWE-327](#) - Use of a Broken or Risky Cryptographic Algorithm
- OWASP Top 10 2017 Category A6 - Security Misconfiguration

- [CERT, MSC61-J](#). - Do not use insecure or weak cryptographic algorithms
- [SANS Top 25](#) - Porous Defenses



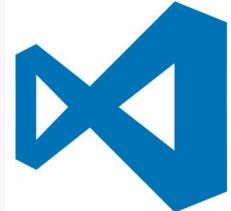
Key Feature - Language coverage

 Java	 C#	 C	 C++	 JS	TypeScript
 Python	 GO	 Swift	COBOL	Apex	 php
 Kotlin	 Ruby	 Scala	 HTML	 CSS	ABAP
 Flex	 Objective-C	PL/I	PL/SQL	RPG	T-SQL
 VB	VB6	<XML/>			



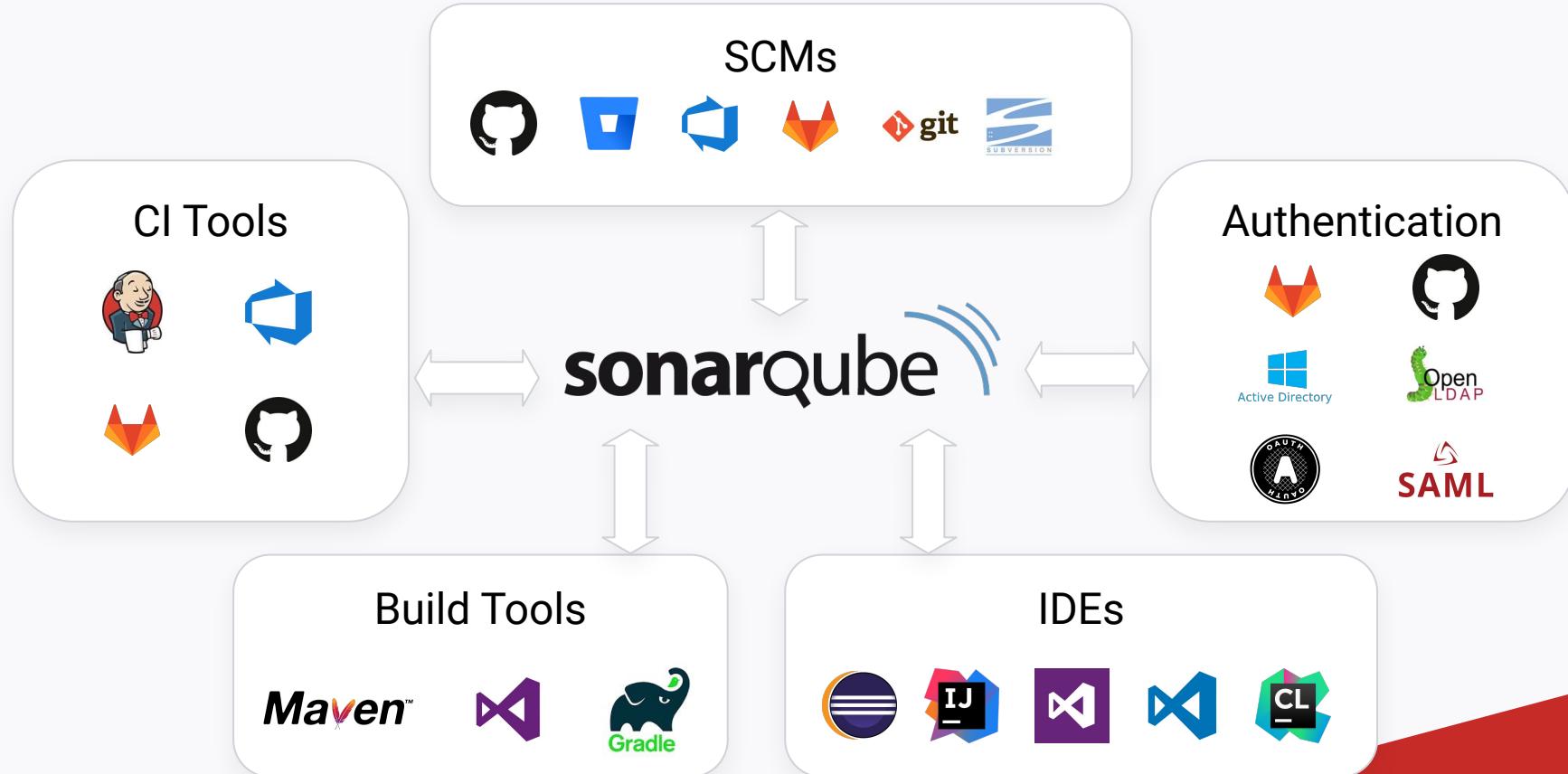
Key Feature - In IDE analysis

sonarlint





Key Feature - DevOps integration

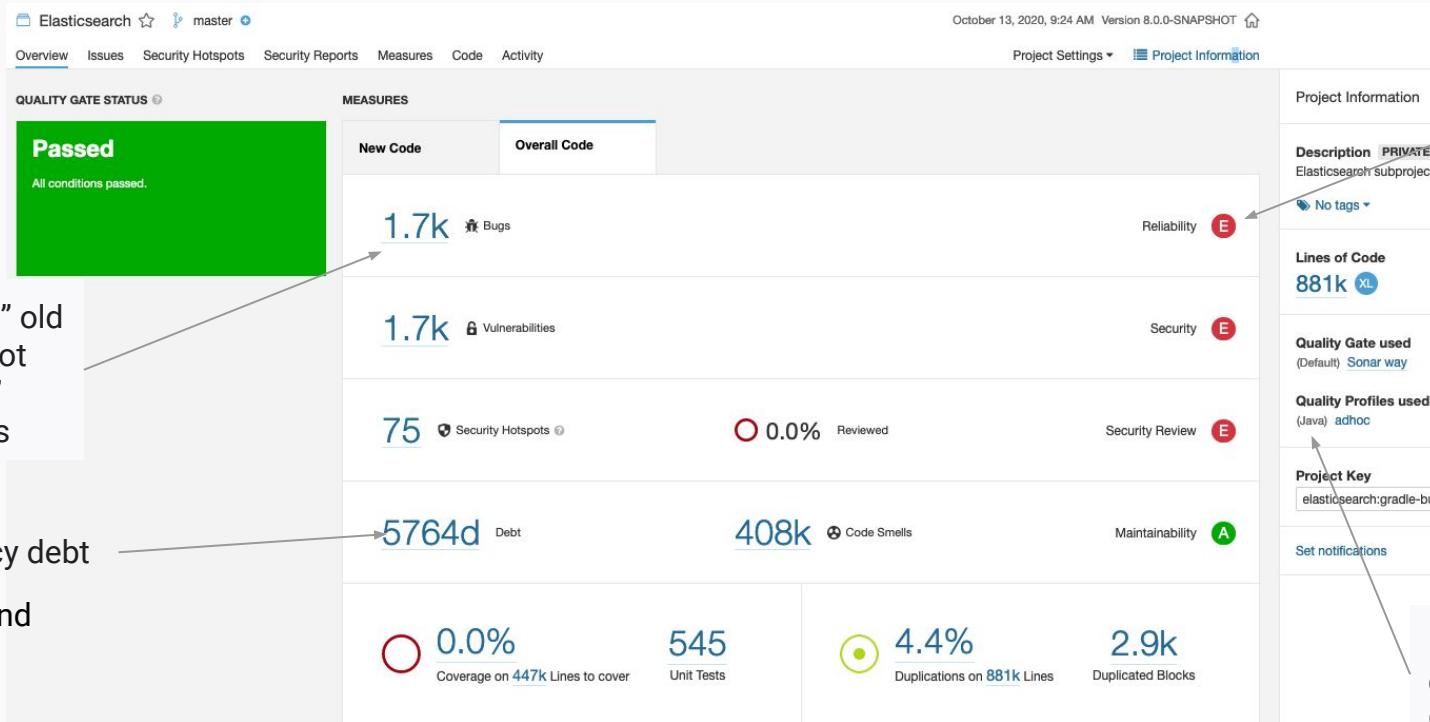


Clean As You Code

Fundamentals



Caveats of traditional approaches



Risk of functional regression

Developer pushback

- Not my code
- Too many issues
- Boring!

Resistance to evolution of Quality Profiles



What is best practice?

You take personal responsibility for your code

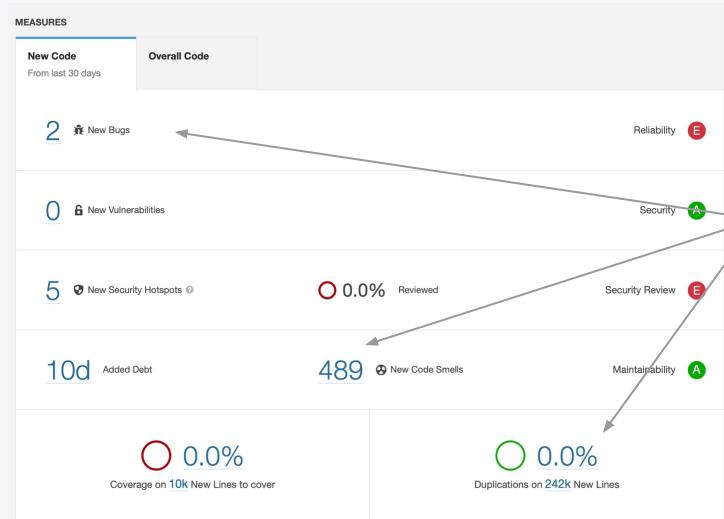
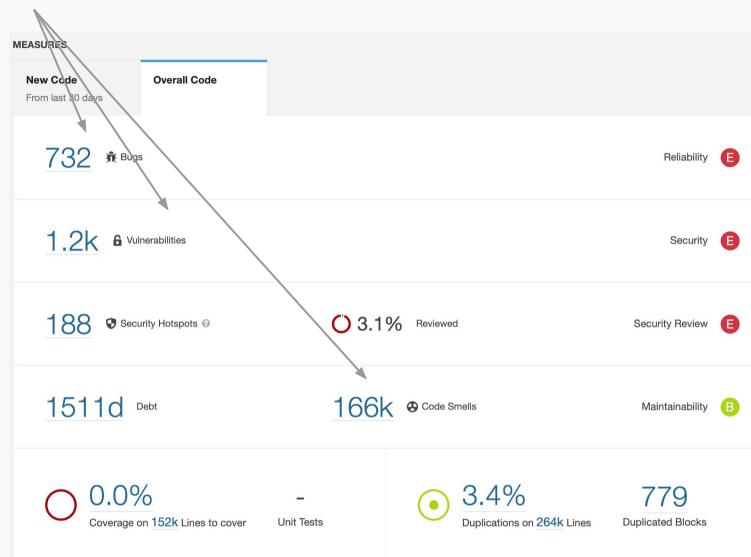




Huge legacy debt

Avoid being overwhelmed

A lot of
work here

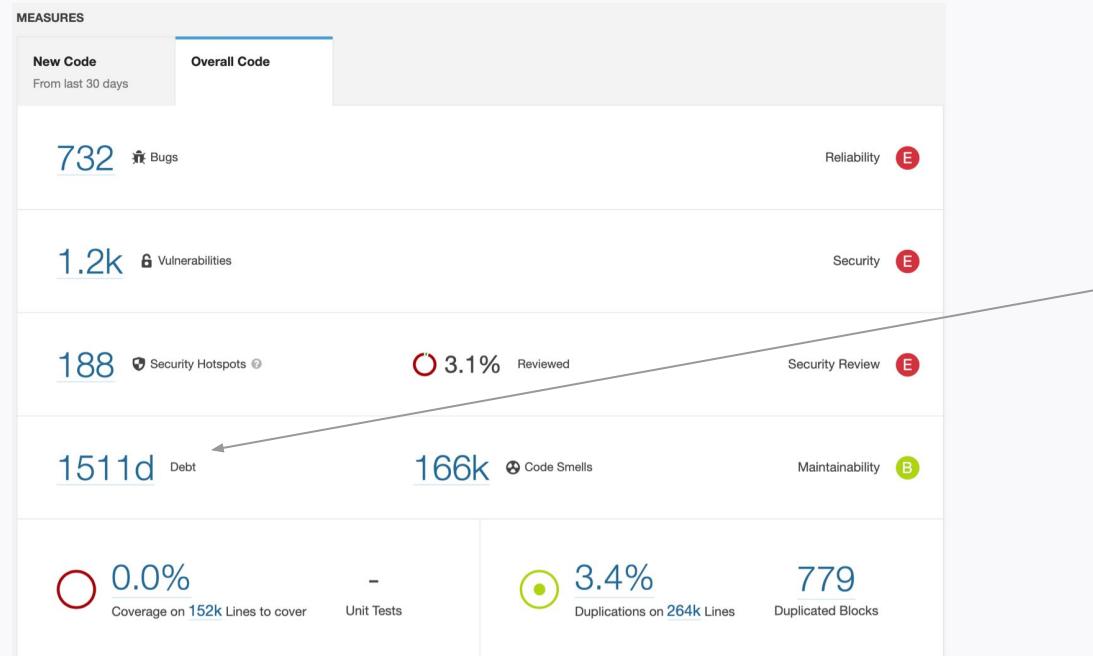


Not so much
there



Huge legacy debt

Get rid of budget considerations



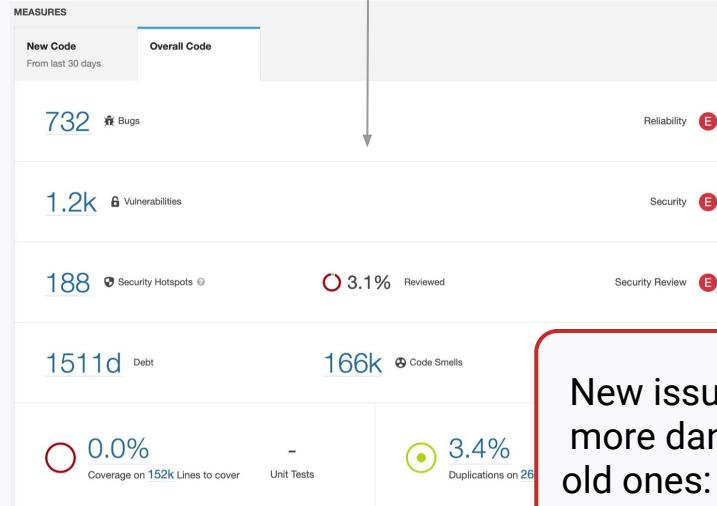
1511 days =
between \$400K
and \$1200K



Compensation

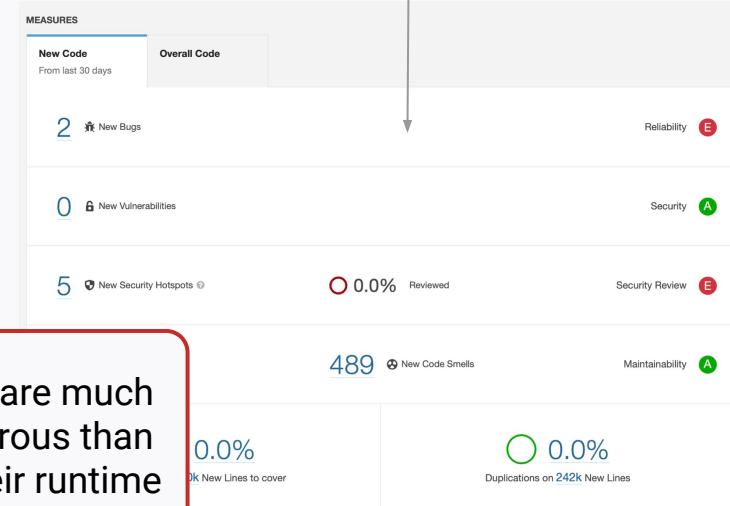
SonarQube will not let you fix an old issue to compensate for a new one

You **may** fix
some of these



...also, be
careful of
functional
regression
here!

But you **MUST** fix
these



New issues are much
more dangerous than
old ones: their runtime
effect is unpredictable



Developer Push Back

Issues assigned to developers that creates them

Blame data
determines who
added or modified
this code

The screenshot shows a SonarQube code review interface. On the left, a blame history indicates changes were made by 'pierr...' on line 55. The main area displays Java code with a SonarQube issue highlighted:

```
    return astNode;
}

@Override
public AstNode createTerminal(Input input, int startIndex, int endIndex, List<Trivia> trivias, TokenType 1 type) {
    int[] lineAndColumn = input.lineAndColumnAt(startIndex);
    Token token = Token.builder()
        .setType(2 type == null ? UNDEFINED_TOKEN_TYPE : type)
        .setLine(lineAndColumn[0])
        .setColumn(lineAndColumn[1] - 1)
        .setValueAndOriginalValue(input.substring(startIndex, endIndex))
        .setURI(input.uri())
        .setGeneratedCode(false)
        .setTrivia(trivias)
        .build();
    AstNode astNode = new AstNode(token);
    astNode.setFromIndex(startIndex);
```

A callout from the blame data points to the condition in line 63. A callout from the SonarQube issue points to the same condition. The issue details are as follows:

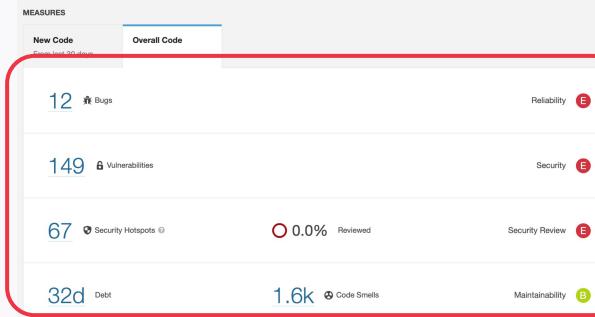
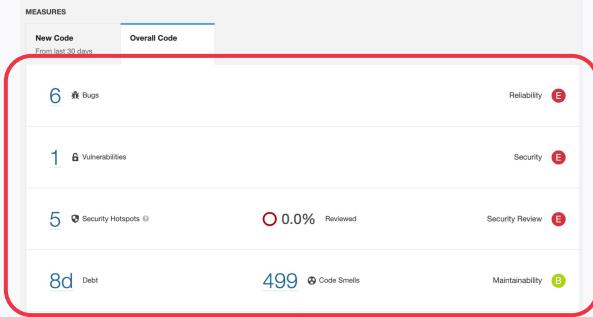
Change this condition so that it does not always evaluate to "false" Why is this an issue?
Bug ▾ Major ▾ Open ▾ pynicolas ▾ 15min effort Comment 3 years ago ▾ L63 %
cert, cwe, misra, pitfall ▾

Issue assigned to corresponding SonarQube user

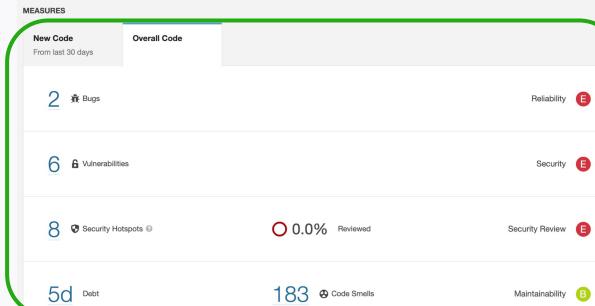
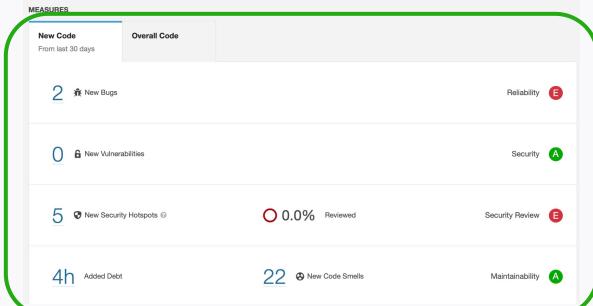


Quality Profile evolution

Making it easier to raise the bar or use new rules from new SQ versions



Significant increase in old code issues



Low or no increase in new code issues

My project with old quality profile

Same project with upgraded profile



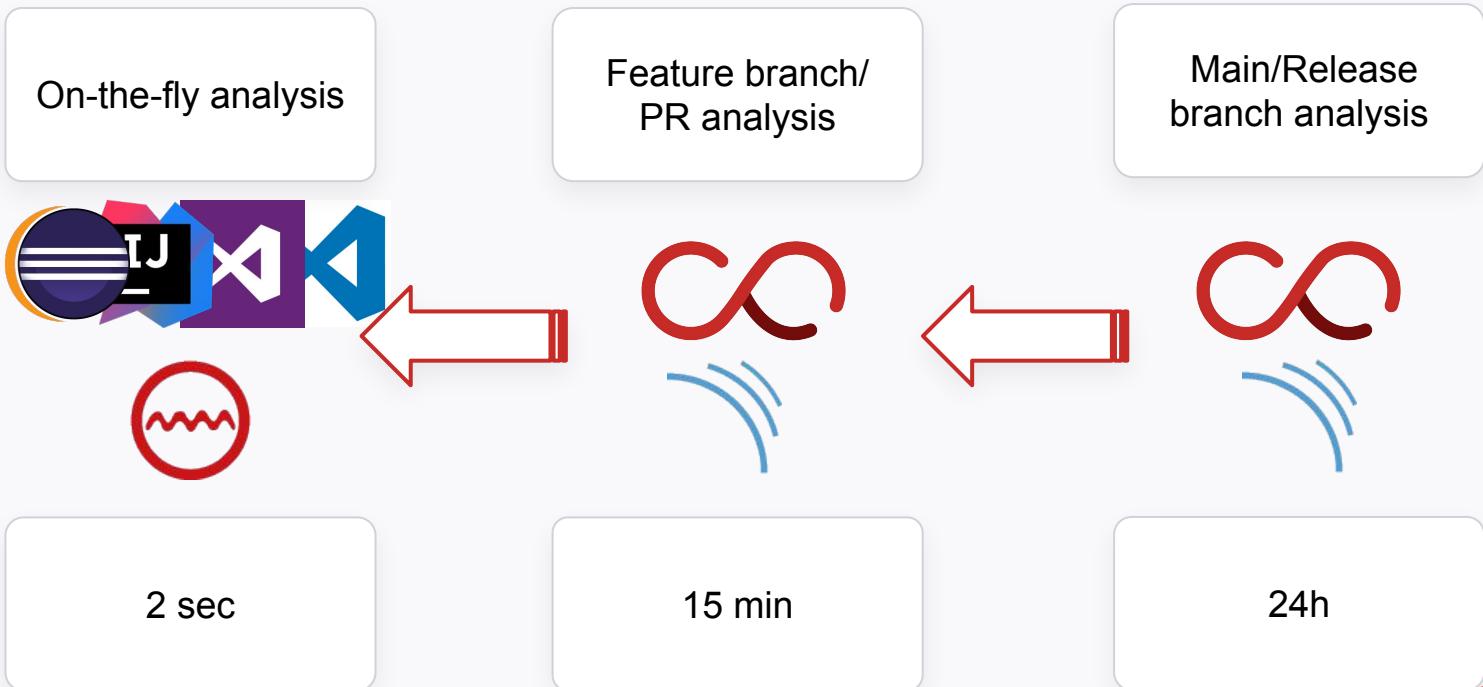
Cleaning up the legacy code

Development lifecycle will renew old code into cleaner new code



Going further

Shift left and shorten the feedback loop



Security overview



Security

What we do and don't do

SAST

Primarily for
Developers

Auditors

DAST

SCA

In the
DevSecOps
pipeline

Specific
process

Fast!





Security: What we detect

OWASP Top 10 Security Risk Categories

A1 Injection	A2 Broken Auth	A3 Data Exposure	A4 XXE
A5 Broken Access Control	A6 Security Misconfiguration	A7 XSS	A8 Insecure Deserialization
	A9 Components with Vulnerabilities	A10 Insufficient Logging & Monitoring	



Security in 8.9 LTS

Languages coverage and evolution

Web and Common Apps

- Injections (taint) vulnerabilities
- Non injection vulnerabilities
- Security Hotspots



System & Embedded

- Buffer overflow Vulnerabilities
- Other non-injection Vulnerabilities
- Security Hotspots



🔓 Vulnerabilities

Fix Security Risks

The screenshot shows two code editor panes and a central analysis interface.

Top Editor: Shows Java code in `Servlet.java`. Lines 28-36 are highlighted with red boxes numbered 1 through 5, indicating tainted data flow. A tooltip box highlights issue 5: "Refactor this code to not construct SQL queries directly from tainted user-controlled data." It also lists 6 vulnerabilities in total.

Bottom Editor: Shows Java code in `SQLInjectionVulnerability.java`. Line 18 is highlighted with a red box numbered 6, indicating a tainted query construction. A tooltip box highlights issue 6: "Refactor this code to not construct SQL queries directly from tainted user-controlled data." It also lists 6 vulnerabilities in total.

Analysis Interface: A central panel displays the following information:

- Vulnerability Summary:** 6 Vulnerabilities +6
- Servlet.java Issues:** 1 source: taint value is propagated, 2 taint value is propagated, 3 taint value is propagated, 4 taint value is propagated, 5 taint value is propagated
- SQLInjectionVulnerability.java Issues:** 6 sink: taint value is propagated
- Navigation:** alt + ↑ ↓ to navigate issue locations
- Details:** See Rule, 3 months ago, L18 %, cert, cwe, owasp-a1, sans-top25-inse...

?

Security Hotspots

Review Security-sensitive code

5 Security Hotspots to review

Review priority: HIGH

Cross-Site Request Forgery (CSRF) 1

Make sure disabling Spring Security's CSRF protection is safe here.

TO REVIEW

Review priority: MEDIUM

Weak Cryptography 1

Review priority: LOW

Insecure Configuration 1

Make sure disabling Spring Security's CSRF protection is safe here.

Add Comment Get Permalink

Category Cross-Site Request Forgery (CSRF)

Review priority HIGH

Assignee Alexandre Gigleux

Status: To review
This Security Hotspot needs to be reviewed to assess whether the code poses a risk.

src/main/java/foo/security/hotspots/CSRFProtectionSpring.java

```
7 @EnableWebSecurity
8 public class CSRFProtectionSpring extends WebSecurityConfigurerAdapter {
9
10    @Override
11    protected void configure(HttpSecurity http) throws Exception {
12        http.csrf().disable();
13    }
14 }
```

sonarqube

sonarlint

Demo



Platform walkthrough

