# Code Audits

## How to increase the quality of the code

**Stefan Kelm**
*WP8-T1*

Webinar, 14th of July 2021

Public

www.geant.org

# Finding Vulnerabilities II - Looking into code

- Code Audits
  - How to increase the quality of the code
- Vulnerability Disclosure
  - How to deal with found vulnerabilities properly
- Breach and Attack Simulation
  - What happens if one or more vulnerabilities in your organisation are exploited

# What we will cover today

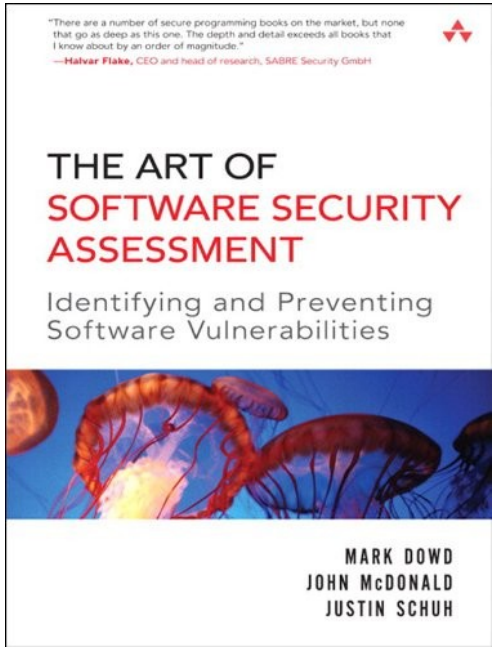
- Code Audits
  - What are code audits?
  - The different classes of audit tools
  - A closer look at static analyzers
    - SonarQube demo
  - Taking things further
    - Continuous integration
    - SDLC
  - Recommendations

# Introduction

www.geant.org

# Let’s start with a few definitions…

- Auditing
  - *“Auditing an application is the process of analyzing application code (in source or binary form) to uncover vulnerabilities that attackers might exploit. By going through this process, you can identify and close security holes that would otherwise put sensitive data and business resources at unnecessary risk.”*



# Source Code Security Analyzers

- According to NIST
  - *"For our purposes, a **source code security analyzer** examines source code to detect and report weaknesses that can lead to security vulnerabilities."*
  - *"They are one of the last lines of defense to eliminate software vulnerabilities **during development** or after deployment."*
  - *"**Byte Code Scanners** and **Binary Code Scanners** have similarities, but work at lower levels."*
- NIST Special Publication 500-268
  - Source Code Security Analysis Tool Functional Specification Version 1.1

# Security Testing Tools

- According to OWASP
  - SAST: **Static** Application Security Testing Tools
    - White box: examine the source code
  - DAST: **Dynamic** Application Security Testing Tools
    - Black box: primarily for web apps (e.g., “fuzzer”)
  - IAST: **Interactive** Application Security Testing Tools
    - “best of both worlds”
  - OSS: Open Source Software Security Tools
    - Keeping your libraries/dependencies updated
  - Static **Code Quality** Tools
    - *“Quality has a significant correlation to security.”*
- Automated vs. manual

GÉANT
Networks • Services • People

# We will be focussing on how to use static tools on our own code today

www.geant.org

GÉANT
Networks • Services • People

## Caveat

**auditing == testing == examining == analysing == analyzing == reviewing == scanning == ...**

**(at least for this talk ;-) )**

www.geant.org

# SAST: Strength & Weaknesses (Source: OWASP)

- Strengths
  - Scales well
    - can be run on lots of software, and can be run repeatedly (as with **nightly builds** or **continuous integration**)
  - Useful for things that such tools can automatically find with high confidence
    - such as buffer overflows, SQL Injection Flaws, …
  - Output is good for developers
    - highlights the precise source files, line numbers, and even subsections of lines that are affected

# SAST: Strength & Weaknesses (Source: OWASP)

- Weaknesses
  - Many types of security vulnerabilities are difficult to find automatically (“runtime issues”)
    - such as authentication problems, access control issues, insecure use of cryptography, etc.
    - frequently can’t find configuration issues, since they are not represented in the code
  - High numbers of **false positives**
  - Difficult to ‘prove’ that an **identified security issue is an actual vulnerability**

# But before we start: advertisement I

- Have you ever heard of GÉANT's WP9T2 services? ;-)
  - https://wiki.geant.org/display/GSD
  - https://wiki.geant.org/display/GSD/Software+Reviews

# The home of GÉANT Software Development Support

Erstellt von Linda Ness, zuletzt geändert von Stefan Kelm am Mai 07, 2021

GÉANT provides a wide range of **supporting services and activities** for **software development teams**. These activities are coordinated and promoted by GN4-3 Work Package 9 Task 2 (Software Governance and Support group).

## SECURE CODE TRAINING

Secure Code Training focuses on secure programming with the aim of minimizing the number of security bugs in the source code. Software developers and security specialists learn how to implement the self-defending applications, that are protected against cyberthreats to the maximum possible extent.

Learn more ->

## SCHOOL OF SOFTWARE ENGINEERING

School of software engineering focuses on code quality and management. Since its early editions, SSE has been introducing and exercising novel technologies and approaches in the software engineering domain, and has been well-accepted by the community of GÉANT developers.

Learn more ->

## KNOWLEDGE BASE FOR SOFTWARE TOOLS

GÉANT delivers a full suite of supporting tools and virtual machines (VMs) for software test and development. The tools and VMs include issue (bug) tracking, continuous integration, and software repositories for source code and binaries.

Learn more ->

## SOFTWARE BEST PRACTICES

The catalogue of software best practices contains a set of practices which are fit for purpose of the GÉANT teams. Task 2 team offers the teams in GÉANT the guidance and consultancy in software best practices adoption along with the process for their monitoring.

Learn more ->

## SOFTWARE CATALOGUE

The GÉANT Software Catalogue (GSC) is a production-grade service providing unified view on software teams and projects in GÉANT. The tools collects automatically relevant information from different data sources with software artefacts such as source code repositories (BitBucket, GitHub), issue tracking systems (Jira) or quality inspection tools (SonarQube)

Learn more ->

## SOFTWARE REVIEWS

Software code reviews are offered as a supporting activity performed by an independent testing team from WP9 Task 2. Code reviews help the GÉANT teams to make their code more robust against all kind of threats and to increase the quality of the code.

Learn more ->

Back to top

# Software Reviews

Erstellt von Marcin Wolski, zuletzt geändert von Branko Marovic am Apr 13, 2021

WP9 T2 provides a special service for GÉANT development teams to make their code more robust against all kinds of threats, to increase the quality of the code or to help them be compliant with the GÉANT Software IPR policy. Besides, the PLM process requires to pass a quality gate before the software can be put into production. A code assessment conducted by WP9 T2 or an IPR check are examples of such a quality gate. The prerequisite for an assessment is that the application or service is listed in the GÉANT Software Catalogue.



## Introduction

We offer several types of review services for code assessment. They vary concerning the method of review, scope and granularity of the report.

Each type of the above-mentioned services is a combination of various review procedures. Differences between the procedures are briefly discussed below:

- **Automated code analysis** concerns maintainability and reliability as the core quality characteristics. SonarQube (SQ) scans the source code of the subject system, identifying flaws and vulnerabilities in the source code, based on internally computed software metrics, and by comparing the subject source code with known anti-patterns. Additionally, SQ defines so-called Quality Gates that verify if the subject code meets requirements to be transitioned to another step in the Product Lifecycle Management (PLM) and can provide recommendations for the decision-makers.
- **Manual expert review** also concerns maintainability and reliability and has similar objectives to the automated code analysis, but it is conducted by domain experts, using pre-defined checklists and/or in an exploratory manner. Experts review and re-validate the results reported by the automated code analysis, and independently review the parts of code that require particular attention, e.g., classes that are complex and play important roles in the system, or may expose known vulnerabilities. The expert code review requires significantly more effort than automated analysis, so it is performed according to the priorities defined by the requestor.

Additionally, we offer a WhiteSource software scan supporting software's IPR check.

# What can static code analysis do for me ?

# → Let’s have a look at SonarQube

# SonarQube

- A web-based open-source platform used to measure and analyse the **quality** of source code
- Metrics on the following categories
  - Reliability
    - **Bugs** indicate that there something wrong in the code, even if the code currently works, it is broken
  - **Security**
    - **Vulnerabilities** include those from OWASP Top 10 and SANS Top 25
    - **Security hotspots** are security-sensitive pieces of code that need to be manually reviewed
  - Maintainability
    - **Debt** estimates time required to fix all issues
    - **Code smells** indicate that the code in question does not satisfy the basic design, implementation and quality principles that may […] increase the risks
  - Coverage
  - Duplications
  - Complexity

# SonarQube

- 5400+ static analysis rules across 27 programming languages
- **Quality Profiles** are sets of rules used by SQ to classify and describe issues
  - Whenever a rule is violated an issue is raised
  - Each language comes with its own Quality Profile (which can be changed)
- **Quality Gates** are an instrument to set a policy for shipping code to production
  - set(s) of conditions against which projects are measured, e.g.:
    - No *Blocker* or *Critical* issues on new code
    - Security Rating worse than *B*
    - Technical Debt greater than *1d*
- Lots of plugins to even enhance the functionality
- There's a free version ("community edition") available :-)

# Demo time

GEANT Software Development Support

BEREICHSVERKNÜPFUNGEN

Knowledge Base for software tools

SEITENHIERARCHIE

- Knowledge Base for software tools
  - Getting technical assistance for software tools
  - Artifactory Pro - Artifact Repository
  - Bamboo
  - Bitbucket
  - JIRA - issue tracking system
  - Quality Assurance Testbed
  - **SonarQube - source code analysis tool**
    - SonarQube Metrics
    - Typical SonarQube Use Cases
      - MANUAL: Adding Projects to SonarQube
      - MANUAL: Adding Source Code directly to a SonarQube project
      - MANUAL: Continuous Integration Setup with GitLab CI and SonarQube
      - MANUAL: Bitbucket and SonarQube
  - GitLab - Source Code Repository
- Software trainings
- Software Maturity and Best Practices
- Software Catalogue
- Software Reviews
- How-to articles
- Contact us

Seiten / The home of GÉANT Software Development Support / Knowledge Base for software tools

# SonarQube - source code analysis tool

Erstellt von Branko Marovic, zuletzt geändert von Stefan Kelm am Apr 12, 2021



## What is SonarQube

SonarQube is a web-based open-source platform used to measure and analyse the quality of source code. Its static code analysis provides insights into code issues and tec assess the code quality in a software project, but also to estimate the remaining effort needed for achieving the production level. SonarQube also helps with tracking code c tests. These features reduce the chances of deploying broken or untested code, particularly during the maintenance phase. Use of such a tool helps to identify many bugs a would otherwise stay undetected and cause damage. SonarQube's tracking of quality norms allows enforcing them and making the code more reliable and readable. Readab increases productivity and quality, as developers must read many lines of code before editing one; therefore, making the code easier to read makes it easier to write.

SonarQube can be used by the development team and in external reviews. It can analyse and manage the code in more than 25 programming languages, including Java, Pyt PHP, C, C++, C#, PL/SQL, Ruby, etc., but also HTML, XML, and CSS. More than 50 plugins extend its functionality.

## How it works

SonarQube reads the source code from the repositories or local files, analyses them with dedicated scanners, calculates metrics, stores the findings in a database, and show web dashboard. The outcomes of the analysis are quality measures and individually detected issues, which are instances where coding rules were broken. It can analyse so ways:

- On-demand:
  - The provided software may be a locally stored source code.
  - Code is stored in a Git, Bitbucket or SVN source code repository (including GÉANT Gitlab and Bitbucket) and code used by a build management system such as MSBuild or makefile.
- The recommended scenario: continuous inspection within a continuous integration and deployment integration (DevOps) lifecycle managed by a GÉANT tool such as:
  - GitLab (GitLab info)
  - Bamboo (Bamboo info)
  - Bitbucket (Bitbucket info)
  - Jenkins (Jenkins info)

The dashboard incorporates various gauges, grading scales, views, and reports and allows drilling into individual statistics to see exactly why things are flagged up to the ca This line is displayed with the corresponding issue and suggestion, but also the issue type, severity, status, and time when it was detected as well as the estimated effort.

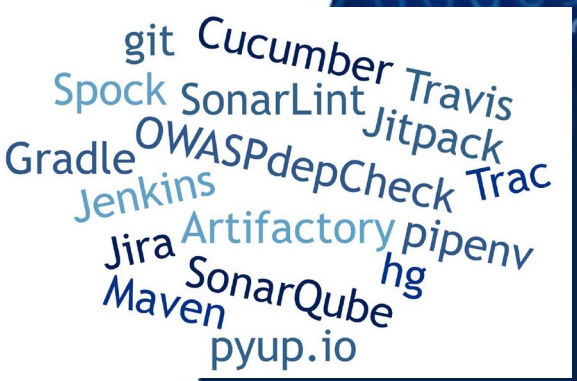
GÉANT
Networks • Services • People

# Analysing source code is all good but what you really want is...

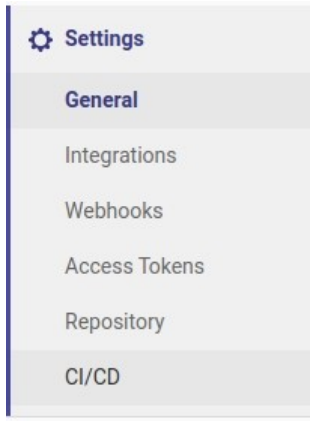
www.geant.org

# CI/CD integration

- According to redhat.com
  - *“CI/CD is a method to frequently deliver apps to customers by **introducing automation into the stages of app development**.”*
  - *“The main concepts attributed to CI/CD are **continuous integration**, **continuous delivery**, and **continuous deployment**. CI/CD is a solution to the problems integrating new code can cause for development and operations teams (AKA "integration hell").”*
  - *“Specifically, CI/CD introduces **ongoing automation** and **continuous monitoring** throughout the lifecycle of apps, from integration and testing phases to delivery and deployment.”*
- Scan the source code itself (SAST) **during the development** and/or **as part of the CI/CD pipeline**
  - Integration of testing tools into development frameworks
    - GitLab, GitHub, Bitbucket, Azure DevOps, Jenkins, …
    - SonarQube and many other SAST tools



# GitLab




## Shared runners

These runners are shared across this GitLab instance.

The same shared runner executes code from multiple projects, unless you configure autoscaling with MaxBuilds set to 1 (which it is on GitLab.com).

**Enable shared runners for this project**

## Available shared runners: 1

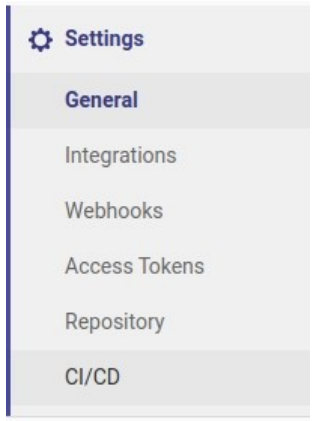
#4 (gi1cDQCR)

Runner dedicated for SonarQube activity

sonarqube

| Status | Job | Pipeline | Stage | Name | Duration | Coverage |
|---|---|---|---|---|---|---|
| passed | #53082 master deb6969f sonarqube | #29737 by | sonarqube | sonarqube | 00:00:23 3 weeks ago | |
| passed | #52921 master d8d4d28a sonarqube | #29691 by | sonarqube | sonarqube | 00:00:22 4 weeks ago | |

# GitLab




**SAST Analyzers**

By default, all analyzers are applied in order to cover all languages across your project, and only run if the language is detected in the Merge Request.

**Bandit** (Python)

**Paths to exclude from scan**

Comma-separated list of paths to exclude from scan. Uses Python's 'fnmatch' syntax; For example: '*/tests/*, */venv/*'

**Brakeman** (Ruby on Rails)

**Brakeman confidence level**

Ignore Brakeman vulnerabilities under given confidence level. Integer, 1=Low, 2=Medium, 3=High.

1

**ESLint** (JavaScript, TypeScript, React)

**Flawfinder** (C, C++)

**Flawfinder risk level**

Ignore Flawfinder vulnerabilities under given risk level. Integer, 0=No risk, 5=High risk.

1

**Kubesec** (Kubernetes manifests, Helm Charts)

| Security Control | Status | Manage |
|---|---|---|
| Static Application Security Testing (SAST)<br>Analyze your source code for known vulnerabilities. More information | Not enabled | Enable |

# GitLab

SAST detected **2** potential vulnerabilities 0 Critical **1 High** and **1 Other**

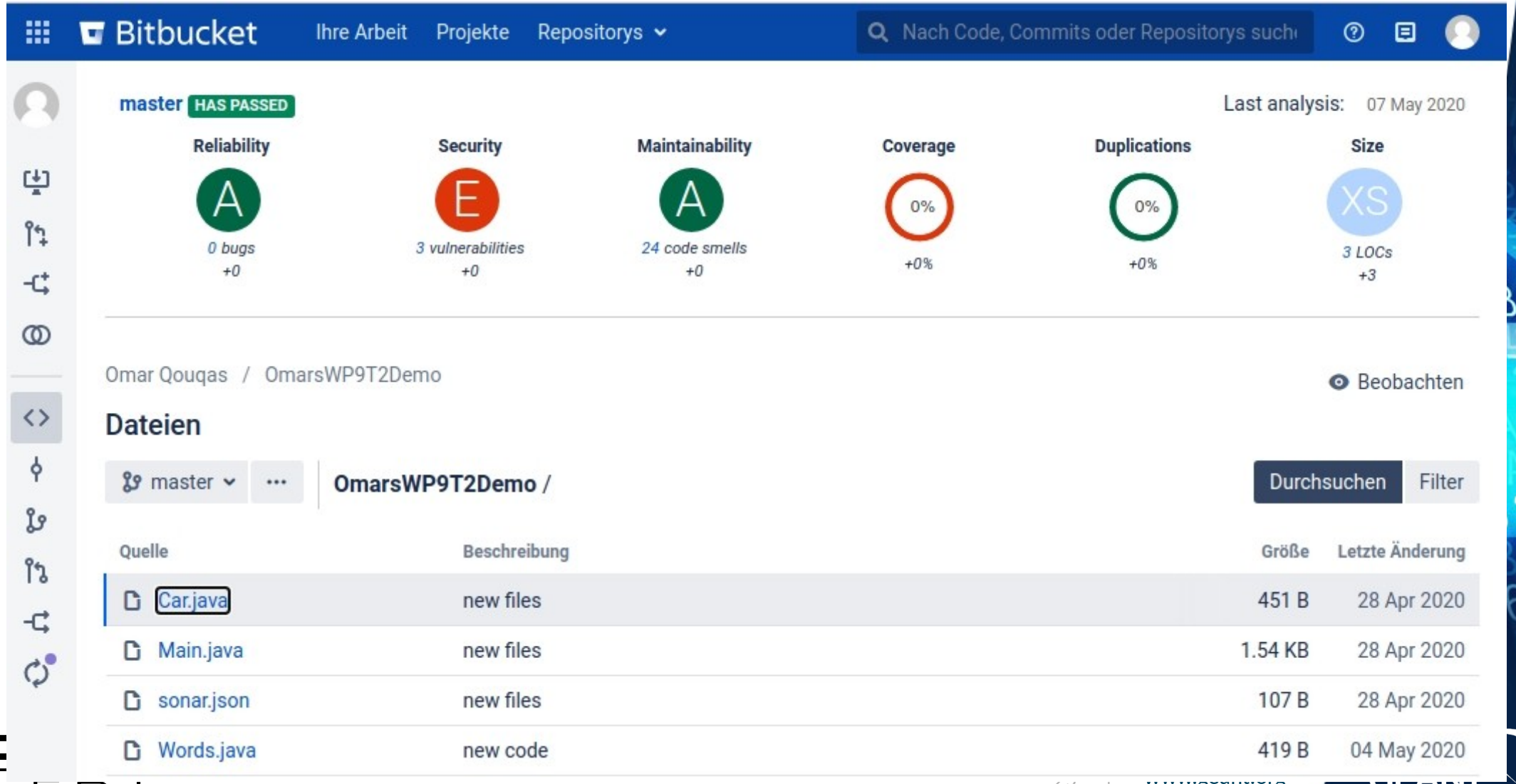
New

- High The Vue.js template has an unescaped variable. Untrusted user input passed to this variable results in Cross Site Scriptin...
- Medium Improper Control of Generation of Code ('Code Injection')

Fixed

- High The Vue.js template has an unescaped variable. Untrusted user input passed to this variable results in Cross Site Scriptin...
- Medium Improper Control of Generation of Code ('Code Injection')

# Bitbucket


Bitbucket
Ihre Arbeit
Projekte
Repositorys
Nach Code, Commits oder Repositorys such
master HAS PASSED
Last analysis: 07 May 2020
Reliability
A
0 bugs
+0
Security
E
3 vulnerabilities
+0
Maintainability
A
24 code smells
+0
Coverage
0%
+0%
Duplications
0%
+0%
Size
XS
3 LOCs
+3
Omar Qouqas / OmarsWP9T2Demo
Beobachten
Dateien
master
OmarsWP9T2Demo /
Durchsuchen
Filter
Quelle
Beschreibung
Größe
Letzte Änderung
Car.java
new files
451 B
28 Apr 2020
Main.java
new files
1.54 KB
28 Apr 2020
sonar.json
new files
107 B
28 Apr 2020
Words.java
new code
419 B
04 May 2020

GÉANT
Networks • Services • People

# CI/CD integration is fine!
# What you really, really want, though...

www.geant.org

# One step further: SDLC

- **Integration of tools/scanners into your (S)SDLC processes, especially at the early development stages**
- Do this in terms of
  - periodic/scheduled scanning
  - build-triggered scanning
  - manual scanning
- Another caveat: is it SDLC or SSDLC? ;-)
  - Systems Development Life Cycle
  - Software Development Life Cycle
  - **Secure Development Life Cycle**
  - Secure Software Development Life Cycle

# Integration of tools/scanners into SDLC

- How to chose the right scanner – does it…
  - report issues directly to (ticketing/bug tracking) systems such as JIRA, TFS, Bugzilla, OTRS, Trac, … ?
  - support CLI/API/plugin-based scanning through external CI/CD software (e.g., Jenkins) ?
    - (this is how I did it for the SonarQube demo)
  - analyze and present diffs between scans (gap analysis) ?
  - allow for extending the scanner with custom plugins, tests, and scripts ?
  - …

# Wrapping up...

www.geant.org

# General observations and recommendations

- Automated security scanners are very good in producing **lots of results**
  - Prioritize the results and make sure that the receivers are not overwhelmed with issues
    - A nice approach is to focus on a specific topic (e.g. *input validation* or *updating dependencies*) and first fix issues in that area before moving on to the next topic (remember SQ's "Security Category"?)
    - Another approach is to only send the "most critical" issues
  - Weeding out false positives from actual issues will require **time and effort**
- Do not underestimate the time required to configure automated scanners correctly!
- Don't trust vendors who claim their scanner will find all security issues in your application
  - There are vulnerabilities that no automated scanner can find

# General observations and recommendations

- Humans are needed with or without (static) analysis tools
  - less false positives/false negatives
  - may have insight into design and architecture
  - only a human who understands the application logic and its context can do a full security test
- Automated scanning cannot replace manual testing …
- … however, tools can cover more code in less time than a human
    - **faster**
    - **broader**
    - **repeatable**
    - …

# General observations and recommendations

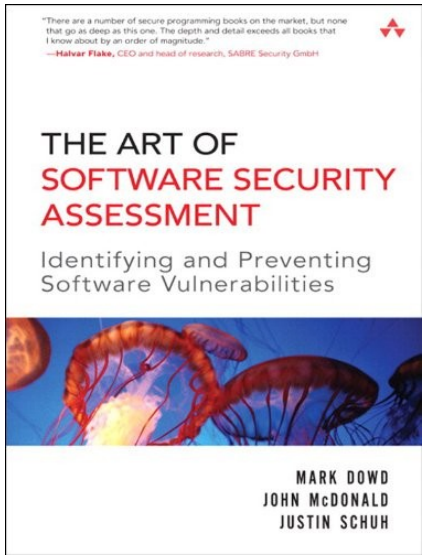
- If you have in-house developers try to increase awareness and get them on-board
  - Make sure to not overwhelm them with too many security issues and/or false positives
  - This allows for developers to also take this moment to learn more about a security topic
  - It is usually more difficult for them to motivate themselves when a whole range of different security issues need to be resolved at the same time
  - Develop test cases
  - Have a zero bug policy on your own code!
- Make code audits part of your risk management
  - the more high risk a system is the more manual testing should be done
- Consider to also scan for (hard-coded) credentials as part of the CI/CD pipeline
  - e.g., test accounts are being forgotten fairly often…
  - **Especially before pushing code to github, etc.**

# What have you learned?

- Code audits are Really Cool™
  - There's quite a few useful tools out there (check the references)
- Manual code audits are important and indeed needed but you want to automate things as much as possible
  - CI/CD integration is vital
- SDLC: the earlier you think about security, the better

# What was not covered today?

- Code Auditing Strategies
  - Code comprehension (CC) strategies
  - Candidate point (CP) strategies
  - Design generalization (DG) strategies
- Classifying Vulnerabilities
  - Design vulnerabilities, implementational vulnerabilities, operational vulnerabilites, …
  - Code weaknesses
    - XSS, CSRF, SQLI, overflows, race conditions, unchecked error conditions, …
    - Data flow, trust relationships, input validation, …
- DAST
  - OWASP ZAP (Zed Attack Proxy), Arachni, …
- Standards
  - Microsoft's Security Development Lifecycle (SDL)
  - OWASP Application Security Verification Standard (ASVS)



# But wait, we’re not done, yet: advertisement II

- Have a look at GÉANT’s Secure Coding Training :-)
  - https://wiki.geant.org/display/GSD/Secure+Code+Training

# Secure Code Training

Erstellt von Andrijana Todosijevic, zuletzt geändert von Gerard Frankowski am Jun 14, 2021

**MOTTO**

Producing secure code for applications is a key aspect of protecting GÉANT applications and systems. With the move towards multi-domain systems and services there is a greater emphasis on securing these multi-domain systems as well as ensuring secure deployment of them. The Secure Coding Training focuses on areas that affect the development and analysis of application's source code!

**Secure Coding Training 2021 (SCT 2021)**

***SCT 2021: let's meet virtually between 6th and 9th September!***

## DATE:

- 4 days, Monday - Thursday
- 06.09.2021 - 09.09.2021
- 10:00-14:00 CEST every day
- Fully virtual training

## REGISTRATION:

- Will be re-run soon after the final date is confirmed
- **https://events.geant.org/event/691/registrations/561/**

## TOPICS:

- Secure Development Life Cycle and Continuous Integration
- Writing Hacker Proof Code - Authentication
- Writing Hacker Proof Code - Authorization and Access Control
- Writing Hacker Proof Code - Logging Verification and Error Handling
- The most popular Web application vulnerabilities workshop (new edition)
- Review of the current static analysis tools
- HackMe contest

> *"Treat Input as Hostile"*
>
> (Dowd et al.)

# Thank you

Any questions?

Next Module: Vulnerability Disclosure

www.geant.org

> *"Passing static code analysis doesn't prove your code is safe... but failing it pretty much signals it isn't."*
>
> (Dana Epp)


The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 731122 (GN4-2).

# References

- GÉANT Software Reviews
  - https://wiki.geant.org/display/GSD/Software+Reviews
- GÉANT Secure Code Training
  - https://wiki.geant.org/display/GSD/Secure+Code+Training
- GÉANT SonarQube pages
  - https://wiki.geant.org/display/GSD/SonarQube+-+source+code+analysis+tool

# References

- Source Code Analysis Tools (OWASP)
  - https://owasp.org/www-community/Source_Code_Analysis_Tools
- Free for Open Source Application Security Tools (OWASP)
  - https://owasp.org/www-community/Free_for_Open_Source_Application_Security_Tools
- Source Code Security Analyzers (NIST)
  - https://www.nist.gov/itl/ssd/software-quality-group/source-code-security-analyzers

# References

- Static source code analysis tools recommended for CERN developers
  - https://security.web.cern.ch/recommendations/en/code_tools.shtml
- List of tools for static code analysis
  - https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis
- NIST Software Assurance Reference Dataset Project (SARD)
  - https://samate.nist.gov/SARD/

# References

- static analysis tools repository
  - https://github.com/analysis-tools-dev/static-analysis
- SonarQube
  - https://www.sonarqube.org/
  - https://github.com/SonarSource/sonarqube
  - https://docs.sonarqube.org/latest/analysis/scan/sonarscanner/
- WhiteSource
  - https://www.whitesourcesoftware.com/