
Tim Sonnekalb  
Thomas S. Heinze  
German Aerospace Center (DLR), Institute of Data Science  
Jena, Germany  
tim.sonnekalb@dlr.de  
thomas.heinze@dlr.de

Lynn von Kurnatowski  
German Aerospace Center (DLR), Institute for Software Technology  
Weßling, Germany  
lynn.kurnatowski@dlr.de

Andreas Schreiber  
German Aerospace Center (DLR), Institute for Software Technology  
Köln, Germany  
andreas.schreiber@dlr.de

Jesus M. Gonzalez-Barahona  
Universidad Rey Juan Carlos  
Fuenlabrada, Spain  
jgb@gsyc.urjc.es

Heather Packer  
University of Southampton  
Southampton, United Kingdom  
hp3@ecs.soton.ac.uk

ABSTRACT
Software repositories contain information about source code, software development processes, and team interactions. We combine provenance of the development process with code security analysis to automatically discover insights. This provides fast feedback on the software’s design and security issues, which we evaluate on projects that are developed under time pressure, such as Germany’s COVID-19 contact tracing app ‘Corona-Warn-App’.

CCS CONCEPTS
• Security and privacy → Software security engineering; • Software and its engineering → Software libraries and repositories; Software defect analysis; • Information systems → Data mining; • Human-centered computing → Open source software.

KEYWORDS
program analysis, provenance, software security, repository mining, open source software, covid-19

1 INTRODUCTION
Software repositories contain much information besides the source code itself. Especially for Open Source projects, the team composition and development process is transparent and traceable and can be evaluated at any point of time by, for example, continuous evaluation with regards to security by automated analysis [8].

The COVID-19 pandemic raises challenges for scientists of many disciplines. Computer scientists and software developers help to fight the pandemic with software systems, which must be developed under time pressure [2], with high quality, and with accepted concepts for data security and privacy.

For example, apps for mobile devices that support contact tracing of infected persons are useful to identify local COVID-19 hot-spots and find other persons, who are potentially infected, too. For contact tracing, several architectures are possible and have been discussed—sometimes very controversial—in many countries. Two favoured approaches are centralized and decentralized architectures; both using Bluetooth Low Energy for contact identification. Apple and Google developed an Exposure Notification API1 as extension of their operating systems iOS and Android, which developers of exposure notification apps can use for privacy-preserving contact tracing. We focus on the German decentralized exposure notification app Corona-Warn-App2 (CWA; see Section 2).

Our main contributions towards our vision of an automated, provenance-driven security audit infrastructure for Open Source software are:
• We give an overview of static code analysis, which we use for our purpose (Section 3).
• We describe our method for querying the development process by using provenance (Section 4).
• We outline our ongoing efforts on combining information from process provenance with static code analysis for some specific revisions of the source code (Section 5).

1https://www.apple.com/covid19/contacttracing/  
2https://github.com/corona-warn-app
2 DEVELOPMENT OF THE “CORONA-WARN-APP”

The development of the Corona-Warn-App gets special attention during the COVID-19 pandemic; the development had to be done in a short time frame: development started in April 2020 and the app was released on 16th June, 2020 for Android and iOS. CWA is developed by SAP and Telekom using a transparent and open development process. CWA has a decentralized architecture, accompanied by centrally-managed Java-based server applications to distribute findings about infected users and store test results uploaded by the laboratories.

CWA development history is publicly available from 12 repositories (some of them auxiliary), including data since April 29th, for source code changes (5,624 git commits; Figure 1), issue tracking (1,397 GitHub issues) and code review (2,144 GitHub pull requests)\(^3\). The human team participating in the development is composed of 306 persons authoring code changes. Having into account the short time span, this amounts to a considerable effort, and suggests that most of the real activity is shown in these public repositories.

The analysis of the software development context for applications, by retrieving metadata from software repositories, has been an active area of research since the early 2000s [14, 21]. During this time several tools have been developed to get some metrics about the software development process and the team building it. We use GrimoireLab\(^4\), a toolset for retrieving data from software development repositories, store it, and perform some analytics via its SaaS instance Cauldron\(^5\) to produce statistics for the CWA. In this case, the context analysis ensures that the data analyzed for provenance is likely real (e.g., it is not likely that the analyzed repositories are not “dump repositories”, where code is copied from time to time, while the real activity happens elsewhere), and gives an idea of the volume of activity caused by the project. In a more complete analysis, software development analytics may complement our provenance analysis by providing insights about how the different actors behave in the project, and how their contributions are related and processed.

Table 1: Used static analysis tools.

<table>
<thead>
<tr>
<th>Static analysis tool</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xanitizer</td>
<td>taint analysis</td>
</tr>
<tr>
<td>infer</td>
<td>formal verification</td>
</tr>
<tr>
<td>Spotbugs</td>
<td>coding rules</td>
</tr>
<tr>
<td>Detekt</td>
<td>coding rules</td>
</tr>
<tr>
<td>Checkstyle</td>
<td>linter, coding rules</td>
</tr>
<tr>
<td>Flowdroid</td>
<td>taint analysis</td>
</tr>
<tr>
<td>SonarQube</td>
<td>linter, coding rules</td>
</tr>
</tbody>
</table>

Static analysis tools can be integrated at various points in a developer’s lifecycle, while coding in terms of IDE plugins, when committing to a developer repository, either in batch mode or at diff-time, or when conducting quality insurance.

The usability of static analysis is known to be influenced by factors such as false-positive ratio, understandable and actionable analysis results, and integration with developer workflow [7, 16]. Experiences in large-scale application of static analysis shows, that integration with developer workflow and reporting bugs as soon as possible is important.

For example, SonarCloud found a bug, which was introduced to the repository “cwa-app-android” by the pull request #876 (Figure 2)\(^6\). The bug was found by SonarScanner before the pull request was accepted and the appropriate line should be deleted. The variable denomination fakeHeader gives a further hint, that this code lines are probably debug code and should not be part of production code.

Figure 2: Introduced bug ‘CWE-561 – Dead code’, ‘CWE-570 – expression is always false’, detected by SonarQube Scanner.

3 All numbers are for July 20th.
4 GrimoireLab: http://chaoss.github.io/grimoirelab
5 Cauldron: https://cauldron.io
6 Pull request #876 was no longer available at the time of publication. Other issues found in the repository cwa-app-android by Sonarcloud are here: https://sonarcloud.io/project/issues?id=corona-warn-app_cwa-app-android
4 PROVENANCE OF REPOSITORIES

Software development is a highly complex process involving a wide range of responsibilities and people. In addition the complexity of the software itself grows over time. To cope with this different tools are used to support the development process. During the entire software development process, all these support tools produce several types of data. These large amounts of data, which are generated before, during, and after the development of a software, can be analyzed using provenance [10].

Provenance analysis focusing on the development of open source software projects provides insight into the interactions of people. These interactions can fall into different categories. The most notable interactions in the development of track and trace software for COVID-19 are those that scrutinise the nature of the data collected and stored, which is hard for automated processes alone to evaluate ethical considerations. This can be evident in the provenance by the number of people collaborating outside of the development team, the number of developers, and the issues reported. While these types of measures cannot guarantee the ethics of the software, it does provide an indication that it has been evaluated by humans.

4.1 Generating Retrospective Provenance for git Repositories

To analyze the development process, we extract retrospective provenance [9] from repositories and store it in a graph database for further analysis (Figure 3) [15]. To extract provenance from git-based projects we use tools, which crawl the git repositories and additional information, such as issues or pull requests (Git2PROV [3, 19] and GitHub2PROV [13]). The provenance is generated as a file in JSON format and then stored in a Neo4j graph database. We note that while GitHub already provides visualisations for their hosted projects, the GitHub2PROV model supports bespoke visualisations that benefit from complex queries across the model’s graph structure, which are not achievable using GitHub’s API.

4.2 Using and Analyzing Provenance

To analyze the provenance graph, many visual and analytical methods exist; including semantic reasoning. For example, we illustrate querying and using the provenance graph for a simple example query for the CWA repository “cwa-server”: “Which files have commits by team members as well as external contributors?”

We generate a Cypher query, that adds information about contributors roles. We retrieve member information via the GitHub API and store it in Python lists of team members and external contributors, which we insert in a Cypher template. This Cypher query creates new directed relations between persons (PROV Agents) and files (PROV Entities); for example, the relation for team members is: (Agent)−[::CONTRIBUTES_TO {role : 'team'}]→(Entity).

Then we query for files, where team members and external contributor made changes at any of the files revisions (Listing 1).

Listing 1: Find all files where a team member AND an external contributor contributed changes.

```
MATCH (team_member:Agent)−[r1:CONTRIBUTES_TO {role : 'team'}]→(f:Entity)−[r2:CONTRIBUTES_TO {role : 'contributor'}]→(external_contributor:Agent)
RETURN team_member,f,external_contributor
```

5 RETROSPECTIVE CODE ANALYSIS FOR OPEN SOURCE SOFTWARE PROJECTS

For conducting a security analysis of the CWA and its development process, we integrate the extracted provenance (Section 4) with bugs or vulnerabilities as reported by the selection of static analysis tools (Section 3). In our infrastructure (Figure 4), we therefore consider individual commit snapshots in the history of the CWA repositories. According to the respective repository, we run certain static analysis tools on a snapshot, track their reported findings and save them into a database for later analysis.

Due to the various involved static analysis tools and their differing report formatting and output granularity, the tools’ findings need to be consolidated such that, for example, duplicated findings can be identified. The tools’ reports are therefore parsed to extract the locations and types of found bugs or vulnerabilities; the latter is additionally normalized using the Common Weakness Enumeration (CWE)7 and other bug ontologies. Interlinking the tools findings with provenance information is done via the respective snapshot’s commit hash.

```
MATCH (team_member:Agent)−[r1:CONTRIBUTES_TO {role : 'team'}]→(f:Entity)−[r2:CONTRIBUTES_TO {role : 'contributor'}]→(external_contributor:Agent)
RETURN team_member,f,external_contributor
```

5 RETROSPECTIVE CODE ANALYSIS FOR OPEN SOURCE SOFTWARE PROJECTS

For conducting a security analysis of the CWA and its development process, we integrate the extracted provenance (Section 4) with bugs or vulnerabilities as reported by the selection of static analysis tools (Section 3). In our infrastructure (Figure 4), we therefore consider individual commit snapshots in the history of the CWA repositories. According to the respective repository, we run certain static analysis tools on a snapshot, track their reported findings and save them into a database for later analysis.

Due to the various involved static analysis tools and their differing report formatting and output granularity, the tools’ findings need to be consolidated such that, for example, duplicated findings can be identified. The tools’ reports are therefore parsed to extract the locations and types of found bugs or vulnerabilities; the latter is additionally normalized using the Common Weakness Enumeration (CWE)7 and other bug ontologies. Interlinking the tools findings with provenance information is done via the respective snapshot’s commit hash.

```
MATCH (team_member:Agent)−[r1:CONTRIBUTES_TO {role : 'team'}]→(f:Entity)−[r2:CONTRIBUTES_TO {role : 'contributor'}]→(external_contributor:Agent)
RETURN team_member,f,external_contributor
```

7https://cwe.mitre.org/
the number of found vulnerabilities or bugs [12], can be tested for the CWA case study.

- The usage of static analysis tools can be investigated, answering questions like how effective certain tools—or combinations thereof—were in uncovering bugs or vulnerabilities [5] or how understandable and usable their reports were [7].

- Characteristics of the vulnerability management in the CWA app development process can be analyzed quantitatively, using metrics like mean time to fix [6], or qualitatively, using fault tree analysis.

6 RELATED WORK

Baumgartner et al. [1] categorized occurring security and privacy risks of existing contact tracing app solutions from a methodological point of view. They discussed different architectures, conducted an experimental study, and created a movement profile of an infected person with an early version of the DP3T app. A similar work by Vaudenay [18] describes the data exchange of the decentralized DP3T solution and possible attack scenarios on the communication, which is always possible without hyperlocal data. He concludes, that there are down sides in the design of decentralized apps and shows improvements. Both works did not focus on the software development itself.

Sun et al. [17] investigate the security of contact tracing applications by the use of static and dynamic analysis tools. They criticised, that not all contact tracing app developers make their code publicly available.

Wang et al. [20] analyzed the activities of much-contributing developers to open source projects in an empirical study and looked also on other repository artifacts besides the code. They investigated the communication between developers and quality of software with increasing contribution.

7 CONCLUSIONS AND FUTURE WORK

We described our vision for automated, provenance-driven security where security of the software product is essential. This includes developing tools and visualizations for developers to investigate how software is developed, the processes used, and the details around how security issues are identified and fixed.

Another future work is to capture code insertions and deletions of individual commits by diff trees [4]. This would enable us to enrich the provenance information; not just with the static code view, via the analysis of commit snapshots, but also with a dynamic view. As a result, sources and fixes of vulnerabilities identified by static analysis could be better researched.

ACKNOWLEDGMENTS

We thank DLR’s student research assistant Claas de Boer (TU Dresden) for his tool prov2neo.

REFERENCES


