DLR Secure Software Engineering

Position and Vision Paper

Rohan Krishnamurthy
German Aerospace Center (DLR)
Jena, Germany
rohan.krishnamurthy@dlr.de

Michael Meinel
German Aerospace Center (DLR)
Berlin, Germany
michael.meinel@dlr.de

Carina Haupt
German Aerospace Center (DLR)
Berlin, Germany
carina.haupt@dlr.de

Andreas Schreiber
German Aerospace Center (DLR)
Cologne, Germany
andreas.schreiber@dlr.de

Patrick Mäder
Technical University Ilmenau
Ilmenau, Germany
patrick.maeder@tu-ilmenau.de

ABSTRACT

DLR as research organization increasingly faces the task to share its self-developed software with partners or publish openly. Hence, it is very important to harden the softwares to avoid opening attack vectors. Especially since DLR software is typically not developed by software engineering or security experts. In this paper we describe the data-oriented approach of our new found secure software engineering group to improve the software development process towards more secure software. Therefore, we have a look at the automated security evaluation of software as well as the possibilities to capture information about the development process. Our aim is to use our information sources to improve software development processes to produce high quality secure software.

CCS CONCEPTS

• **Security and privacy** → **Software security engineering**; *Systems security*; • **Software and its engineering** → *Software development process management*;

KEYWORDS

data science, it security, secure software engineering, code analysis, provenance

ACM Reference Format:
Rohan Krishnamurthy, Michael Meinel, Carina Haupt, Andreas Schreiber, and Patrick Mäder. 2018. DLR Secure Software Engineering : Position and Vision Paper. In *Proceedings of First International Workshop on Security Awareness from Design to Deployment (SEAD'18).* ACM, New York, NY, USA, Article 4, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Software engineering has been a conventional methodology that is followed for the development of software. It is still not well adopted by the scientific community so far. On top of that the topic of secure software is emerging without being handled appropriately.

Driven by the reproducible science paradigm and ever growing research networks, scientific software is made available to a broader range of users. Despite an alarming amount of vulnerabilities has been made public over the last years, software is still shared without being hardened with respect to security issues. Thus scientific software might introduce attack vectors to target research institutes.

Our goal is to come up with a collection of guidelines and tools to apply during development to create secure software without in-depth knowledge of software security.

At DLR we have access to a range of software repositories[1] as well as many ongoing research projects where software is developed.

In this paper we want to outline our research as follows:

- We characterize software development at DLR to illustrate the context of our research (Sect. 2).
- We describe the need to address the security issues in DLR with better insight into secure software engineering based on data analysis (Sect. 3).
- We present our strategies to analyze software development processes and to create a catalog with tools and guidelines that supports secure software development (Sect. 4).

2 SOFTWARE DEVELOPMENT AT DLR

The German Aerospace Center (DLR) is one of the largest research organizations in Germany. Over 8000 scientists are researching in the fields of aeronautics, space, transportation and energy. In these fields many tasks rely on computer systems. This involves individual software developed by domain experts in multiple programming languages across different platforms.

More than 2000 people at DLR are occupied with software development in part-time or full-time [1]. Most of the developers have no training in software development, only very few have deeper knowledge about systematic development of sustainable software, the means of software engineering, or secure software. Typical

[1] This includes but is not limited to version control systems as well as issue trackers and continuous integration systems.

SEAD'18, May 2018, Gothenburg, Sweden

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

development team sizes range from one up to 20 persons, in average being one scientist supported by interns and perhaps a Ph.D. student.

The combination of being domain scientists and small team sizes makes the amount of needed knowledge about software engineering and security a disproportional overhead for a project.

3 SOFTWARE SECURITY IN DLR

As DLR is well known for its expertise we have lots of cooperations with partners all over the world. Also the importance of reproducible results has an increasing necessity to publish not only the data but also the software used to produce the results [2]. Consequently our software needs to be shared with many different peers.

Sharing of software might open up security risks. As long as the *software*, *input data*, and the *execution environment* is under control of a single entity, security concerns are a minor issue. However as soon as one of these three factors gets externalized, security issues need to be considered. In many of our cases public interfaces to software are only added after the software is already in productive use. Known security issues are only handled in the added interfaces and not in the software itself. Examples for issues we already faced are: Missing validation of external datasets, information leaks over hidden channels, and outdated dependencies.

Unexperienced developers at DLR have seen the deployment of software in the cloud as a solution to decouple the execution of vulnerable code from internal resources. But this is not a security advantage. Hidden channels might be opened to internal DLR resources that are available to the cloud-hosted code. Leakage of information and data that was meant for internal use by the software is also a possible risk.

The lack of IT (security) experts leads to such problems. As a result internal software life cycles do not pay attention to basic activities like security updates of frameworks and libraries.

4 TOWARDS SECURE SOFTWARE DEVELOPMENT

Our focus is on improving processes and tools. We want to create a catalog with tools and guidelines that support secure software development. To accomplish this goal we apply methods from data science to analyze software development processes and the resulting software. As data we use the software projects of DLR.

Our strategy consists of the following steps:

(1) We select a number of projects with well-defined software engineering processes. Our position within the DLR gives us access to more than 300 projects on different version control systems and about 120 projects in issue trackers that can be mined for information. We want to record the actual processes that are carried out and compare them to the defines processes to identify deviations. To record the processes methods and technologies like repository mining, key loggers, IDE extensions and conducting surveys exist. Due to the privacy implications we refuse to use the key logger approach. While IDE extensions are our preferred way to capture provenance data, the variety of used IDEs at DLR, results is a high implementation overhead for a first approach. Hence in our first step we rely on the documented process and incorporate easy to adapt techniques like surveys and repository mining to augment this data.

(2) We monitor the quality of the software using manual and automated audits. Therefore we investigate and improve existing static and dynamic security analysis methods. The dynamic analysis provides the most universal and versatile way of automated security auditing, however they are mostly very time consuming and produce rather vague results in contrast to the static analysis approach. The static analysis can be evaluated based on manual audits, syntax tree analysis, and intermediate language analysis. We will focus on the latter-most analysis approach as the existing vulnerabilities from databases like CVE[2] or the exploitation frameworks can be transferred into intermediate language. These then can be used as examples for vulnerable or exploitable code.

(3) We conduct experiments to identify process properties that have an effect on the security of the resulting software. Factors such as security-focused requirements engineering or a special security testing phase promise a high impact. However we also want to experiment with other approaches such as *threat modeling* and special trainings for developers. To allow comparison of software quality across projects, we introduce a software security scoring system based on automated software analysis.

5 CONCLUSION

In order to improve software development processes we started a new research group. Our aim is to optimize process properties using approaches from data science. We include two main sources of data: the *provenance of software processes* and a *score for the software security* of that artifact.

We presented some strategies for collecting both of them:

- We plan to use repository mining as a source for process information. This should also help to identify missing information that needs to be recorded with another approach
- To augment the mined data, we plan to introduce developer surveys. In a later step we also plan to implement process recording extensions for IDEs.
- We plan to derive a common security scoring system based on existing dynamic and static analysis techniques.
- We develop a new data driven static analysis approach based on the intermediate language representation of source code.

We will apply these approaches in real projects in the environment of DLR, which gives us large datasets that can be used to improve results.

REFERENCES

[1] Carina Haupt and Tobias Schlauch. 2017. The Software Engineering Community at DLR: How we got where we are, Neil Chue Hong, Stephan Druskat, Robert Haines, Caroline Jay, Daniel S. Katz, and Shoaib Sufi (Eds.). *Proceedings of the Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE5.1).* http://elib.dlr.de/114050/

[2] Victoria Stodden and Sheila Miguez. 2014. Best Practices for Computational Science: Software Infrastructure and Environments for Reproducible and Extensible Research. *Journal of Open Research Software* 2, 1 (jul 2014). https://doi.org/10.5334/jors.ay

[2] https://cve.mitre.org/