TRACER: A Platform for Securing Legacy Code

Kostantinos Stroggylos[1], Dimitris Mitropoulos[1], Zacharias Tzermias[2], Panagiotis Papadopoulos[2], Fotios Rafailidis[3], Diomidis Spinellis[1], Sotiris Ioannidis[2], and Panagiotis Katsaros[3]

[1] Department of Management Science and Technology
Athens University of Economics and Business
{circular, dimitro, dds}@aueb.gr
[2] Institute of Computer Science
Foundation for Research and Technology - Hellas
{tzermias, panpap, sotiris}@ics.forth.gr
[3] Department of Informatics
Aristotle University of Thessaloniki
{katsaros@csd.auth.gr, frafaili@yahoo.gr}

Keywords: Static Analysis, Software Security, Trusted Applications, Legacy software.

Abstract

A security vulnerability is a programming error that introduces a potentially exploitable weakness into a computer system. Such a vulnerability can severely affect an organization's infrastructure and cause significant financial damage to it. Hence, one of the basic pursuits in every new software release should be to mitigate such defects.

A number of tools and techniques are available for performing vulnerability detection in software written in various programming platforms. One of the most common approaches to identify software vulnerabilities is *static analysis* [1]. This kind of analysis is performed by automated tools either on the program's source or object code and without actually executing it. However, since the formats in which static analysis tools store and present their results vary wildly, it is typically difficult to utilize many of them in the scope of a project. By automating the process of running a variety of vulnerability detectors and collecting their results in an efficient manner during development, the task of tracking security defects throughout the evolution history of software projects can be simplified.

In this paper we present TRACER, a framework to support the development of secure applications by constantly monitoring software projects for vulnerabilities. TRACER simplifies the integration of existing tools that detect software vulnerabilities and promotes their use during development and maintenance.

Instead of designing and implementing TRACER from the ground up, we built it on top of the open source *Alitheia Core* [2] platform, which is designed for facilitating large scale quantitative software engineering studies. While Alitheia Core aims for efficient estimation of the quality of software projects, TRACER

was designed with a focus on software security. To support the specific objectives of TRACER, a set of new components was added at each level of the Alitheia Core architecture. These include a model for representing software vulnerabilities, a mechanism for automatic vulnerability detection triggering, a REST API for accessing the analysis results, and an archetype for plug-ins to integrate new vulnerability detection tools in the platform. Like Alitheia Core, TRACER monitors multiple data sources associated with the development of a software project, such as the source code repository and bug tracking system, and automatically analyzes each revision. Therefore it can be used to track security defects throughout the evolution of a project.

In most cases, the detection of vulnerabilities on a software artifact involves only two steps: invoking an external tool created for this purpose with specific arguments as required, and evaluating the results it generates. There is a vast number of software vulnerability detection tools available, each one having different operating requirements. Such a tool can be integrated in TRACER by creating a corresponding driver that implements these two steps and stores the results using the data model provided by the platform. Thus we can leverage the functionality provided by existing tools, without duplicating it.

Such an external tool driver is called a vulnerability detector plug-in, and it uses the Alitheia Core infrastructure to handle automatic activation, as well as storage and retrieval of results. Each vulnerability detector is associated with the set of vulnerability types it can detect and the different types of software artifacts or programming constructs that it can analyze. This allows the platform to automatically trigger it when needing to check if a software project or artifact is vulnerable to a specific type of attacks or a new artifact is submitted to the system for evaluation.

To demonstrate the efficiency and usability of the platform, we have created plug-ins to integrate two different tools for vulnerability detection, namely: *FindBugs* [3], and *Frama*-C [4]. The former analyzes applications written in Java, while the latter examines applications written in C. This highlights the fact that our platform does not depend on the programming language used to develop the project that is being analyzed, and that the simplicity of integrating third party tools leads to high levels of expandability of the platform.

References

1. Chess, B., West, J.: Secure programming with static analysis. Addison-Wesley Professional (2007)
2. Gousios, G., Spinellis, D.: Alitheia core: An extensible software quality monitoring platform. In: Proceedings of the 31st International Conference on Software Engineering. ICSE '09, Washington, DC, USA, IEEE Computer Society (2009) 579–582
3. Hovemeyer, D., Pugh, W.: Finding bugs is easy. SIGPLAN Not. **39** (December 2004) 92–106
4. Cuoq, P., Kirchner, F., Kosmatov, N., Prevosto, V., Signoles, J., Yakobowski, B.: Frama-c: A software analysis perspective. In: Proceedings of the 10th International Conference on Software Engineering and Formal Methods. SEFM'12, Berlin, Heidelberg, Springer-Verlag (2012) 233–247