Detection and Mitigation of Dependencies with Security Risks

Gleb Naumenko
University of British Columbia
naumenko@cs.ubc.ca

Puneet Mehrotra
University of British Columbia
puneet89@cs.ubc.ca

Anna Scholtz
University of British Columbia
ascholtz@cs.ubc.ca

1 INTRODUCTION

Code often relies on outdated dependencies or uses deprecated methods that might pose critical security issues [5]. The most common security vulnerabilities include: weak encryption, algorithms like TripleDES [1], weak hash functions, for example SHA-1 [1], or weak key length like RSA-768 [4]. As is often the case, these projects might have only a limited number of resources working on them – for example in a low priority legacy product – or might have no active developers, but a strong user-base, as is the case in many open source projects. Developers lose awareness of the libraries they use if they do not work on a project for too long. Also, most developers are not aware of the security vulnerabilities in the libraries they use [5]. It is tedious to look for updates, and one must remember to do so in the first place. In the absence of active monitoring of the project, its dependencies can stay undetected and outdated for long periods of time, increasing the risk of an attack. These problems can be alleviated with tools that notify the developers of the outdated dependencies in their code, suggest alternative safe methods, and look for updates and help with the application of fixes.

2 GOALS

The goal of our project is to build a tool that:

- Detects outdated dependencies
- Statically identifies locations where deprecated or unsafe methods are used in the code and suggests safe alternatives
- Dynamically updates dependencies in projects and then runs tests to check whether the code is broken and needs attention

The prototype will be able to detect risks in Python projects but will be designed in an extensible way to support more programming languages in the future. We plan to implement it as a command-line tool and as a plugin for the Sublime editor. The command-line tool will be able to analyze local projects as well as Github projects by providing a URL to the project code.

For detection, the tool will rely on pip which is a package management system for installing and managing Python software packages [5]. Additionally, it will check the code against a database containing methods with known weaknesses, such as methods that use hash collision weaknesses like MD5 [4]. The IDE plugin allows highlighting those issues directly in the Sublime editor. The tool will extract all dependencies from the provided source code, checks whether new versions have been released, and provides a summary to the developer. In addition, it also offers to update the dependencies if newer versions are available. The update will run on a clone of the source code, and then any available tests will be executed. After the build and tests have run, a report will be generated which will contain whether the update was successful and the coverage achieved by the tests that were run. In case the build fails, the report provides information about what kind of errors occurred. The report will also hint to the developer where methods with known weaknesses are used in the code and will suggest safer alternatives. The tool shall generate a report as plain text or as an HTML file. The tool will also allow creating pull-requests for successfully updated projects, with the generated report in their description.

3 RESEARCH QUESTIONS

Using the developed tool we want to analyze different projects on Github and address the following research questions:

(1) How many projects have dependencies with security risks?
(2) What are good strategies to update outdated dependencies?
(3) To what extent can static or dynamic verification be used to check if the code still runs after an update?

For the upgrading process, we will evaluate different strategies, such as updating all dependencies at once, updating dependencies one after the other. We will also examine what strategies to use when an upgrade fails.

4 EVALUATION

To evaluate the tool we plan to select trending Github repositories with unit tests, analyze them for their use of libraries. By this, we hope to glean information that helps us to answer (1). In the next step, we will randomly select projects to update and use different update strategies to resolve their outdated dependencies. We hope to identify an optimal strategy based on the ease of upgrade and success of the build process while accounting for the number of dependencies to answer (2). In the next step, we will update outdated dependencies of the selected repositories, issue pull-requests to the developers and examine the acceptance rate. We also plan to run a small user study in which we ask participants to use the tool and participate in an interview. The interview will focus on the usefulness and usability of the developed tool.

5 RELATED WORK

Requires.io [3] sends notifications if a Python dependency is expired. It monitors git repositories, however, a free plan is only available for public repositories. It also does not provide the possibility to update outdated dependencies automatically.

Greenkeeper.io [2] updates npm dependencies of Github JavaScript projects in real-time and runs unit tests.

Our developed tool will offer a command-line tool as well as an IDE integration to analyze locally stored Python projects as well as repositories on Github. Dependencies will get updated and the developer will see whether the updated code is broken by running all unit tests. In addition to dynamically checking whether the code breaks our tool will also employ static analysis to check where unsafe or deprecated methods are used.

REFERENCES

[1] Ca5350: Do not use weak cryptographic algorithms. https://msdn.microsoft.com/en-us/library/mt612872.aspx.

[2] greenkeeper.io. https://greenkeeper.io/.

[3] requires.io. https://requires.io/.

[4] T. Kleinjung, K. Aoki, J. Franke, A. K. Lenstra, E. Thomé, J. W. Bos, P. Gaudry, A. Kruppa, P. L. Montgomery, D. A. Osvik, et al. Factorization of a 768-bit rsa modulus. In *Annual Cryptology Conference*, pages 333–350. Springer, 2010.

[5] R. G. Kula, D. M. German, A. Ouni, T. Ishio, and K. Inoue. Do developers update their library dependencies? *Empirical Software Engineering*, pages 1–34, 2017.