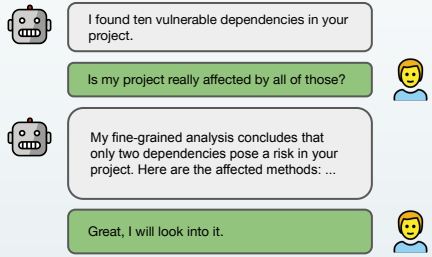
Effectiveness of using call graphs to detect propagated vulnerabilities

Jakub Nguyen
J.H.Nguyen@student.tudelft.nl

Supervised by Mehdi Keshani and Sebastian Proksch

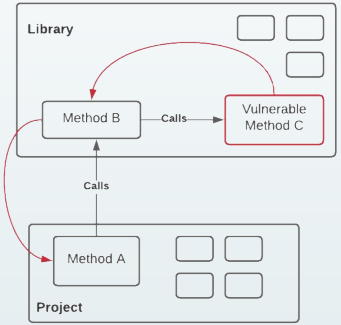


Background

- **Package management** tools like Maven[1] are an integral part of software development nowadays
- Tools like Dependabot[2] can alert developers about **vulnerabilities** in **dependencies** they use
- Such alerts are oftentimes false positives because the vulnerable part of the dependency is not touched
- Current research conducted within the FASTEN Project[3] revolves around **call graphs**
  - Source code is statically analysed to generate **call graphs**
  - **Call graphs** are directed graphs where vertices represent methods and edges possible calls between them
- **Call graphs** can be used to analyse dependency networks on the method-level ➔ detecting whether **vulnerable methods** of your **dependencies** are actually reachable

The research questions:
How effective are call graphs to detect propagated vulnerabilities?
How well do such analysis results reflect the reproducibility of vulnerabilities?

Example vulnerability detection through call graphs:



The project is considered vulnerable because Method A can call a Method C.

Methodology

Goals:

- Find projects with vulnerable dependencies (**package-level vulnerabilities**)
- Perform **call graph analysis** on many projects
- Investigate the reproducibility of the vulnerabilities manually

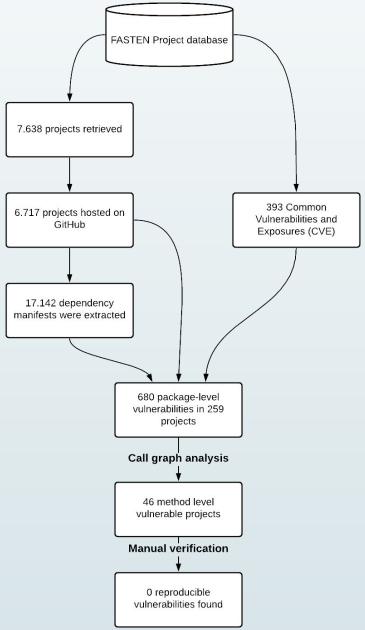
The FASTEN Project collects data about public software repositories and vulnerabilities. This data was used for this research.

The diagram in the results section depicts the overall procedure.

Throughout the experiment, ideas to improve the detection of vulnerabilities through call graphs were developed.

Results

The overall procedure on a high level:



Conclusion

- No reproducible vulnerabilities were found
  - ➢ A greater set of projects needs to be analysed to draw meaningful conclusions
- Descriptions of vulnerabilities oftentimes do not explicitly name related methods ➔ difficult to extract associated method names
  - ➢ The initial data was over-approximating related methods resulting in precision loss
- Possible improvements to such detection revolve around
  - ➢ Considering method parameters
  - ➢ Order of execution of methods
  - ➢ Certainty of method calls
  - ➢ Precise vulnerability information

➔ Call graphs yield **great potential** for detecting propagated vulnerabilities.

Future work

- An experiment on a greater scale with an improved methodology
- Implementing the proposed improvements and testing performance in practice
- Eventually, a tool that conducts method-level analysis and gives recommendations just like Dependabot. (Possibly working on top of Dependabot)

References

[1] https://maven.apache.org/
[2] https://dependabot.com/
[3] https://www.fasten-project.eu/