Plan for a Master's Project

Title	Data flow analysis to detect sql-injection vulnerabilities using Rascal
Document version	0.4.0 (fourth draft version)
Student	Ruud van der Weijde | 10453857
Host organization	Werkspot | www.werkspot.nl | Technology | Amsterdam
Contact person	Winfred Peereboom | Technology Director | winfred.peereboom@werkspot.nl | 06-46446888

Project summary (short version)	In my thesis, I will do a static source code analysis to find sql-injection vulnerabilities on the codebase of Werkspot. The implementation will be done in Rascal. It will contain the following main steps:
	• Parse PHP code to AST
	• Convert the AST to Object Flow Language (OFL)
	  ◦ OFL is used to get the data flows of an OO-program
	• Optimize the analysis
	  ◦ By using the PHP Include algorithm of Mark Hills.
	  ◦ By reading annotations to find object information.
	• Perform data flow analysis to find `sinks`.
	• Validation of the results.
	The data flow analysis will first focus on finding SQL injection (SQLi) and afterwards on finding XSS vulnerabilities.
	The scope of the research will be the legacy part of the system, which is using ZendFramework and Symfony (both first versions of the frameworks), and doctrine12 (all deprecated frameworks).

Research question	- How can vulnerabilities be found in PHP using Rascal, having as precise results as possible.
Research method	Implement in Rascal in the following steps: - Find out how the compiler works; check control flow graphs, check limitations (part of literature study). - Convert AST to OFL. - First OFL step is context-insensitive. - [Optional: make OFL context-sensitive.] - Apply flow propagation algorithm. - Define and add taint restrictions (what can be labeled as tainted?) - Define and add untain restrictions (how will tainted values be untainted) - Add application based settings to optimize results - `Include` optimization (try to resolve more includes using the study of mark hills) - Object construction optimization using annotations. - [YAML optimizations to improve analysis results.] - Validation of the analysis results: - - Use pre-defined files analyzed in different tools. - - Use analyzed tools in similar research that is still available for download. - - Compare results with Pixy, RIPS, SAFERPHP. - - Try to run the code in the sinks using a PHP interpreter. - Produce final results (this step will be iterated)
Expected results	Thesis, analysis tool, human readable results of analysis.
Required expertise	PHP, Rascal, taint + static code analysis, PHP compiler, OFL, SSA

Global timeline

- Jan:
- Finish the project plan.
- Feb:
- Thesis: background, problem description, motivation.
- Rascal: basic proof of concept in Rascal
- Mrt:
- Thesis: describe analysis method.
- Define taint and untaint definitions.
- Rascal: Extend the proof of concept to a working prototype. And compare the results with existing programs.
- Apr:
- Thesis: update analysis method, write about result analysis.
- Rascal: Improve prototype using dynamic include resolving.
- May:
- Thesis: update thesis, add first results to thesis.
- Rascal: Add annotations to analysis, for doctrine/symfony.
- Jun:
- Thesis: add results to thesis.
- Rascal: Handle type hinting annotations out of doctrine/symfony.
- Jul:
- Thesis: Finish, write conclusion, future work, limitations
- Aug:
- Room to fix delays in the progress.

Main risks

- Too ambitious: provide better scope on the project.
  - Added scope in vulnerabilities: SQLi first, maybe XSS later.
  - Added scope to source code, only old website code.
- Unclear project goals: define exactly what is expected.
  - Human readable output is expected.
  - This can be done by providing sinks, which are readable.
- Analysis is not measuring the right data.
  - Use dynamic analysis to validate the static analysis results.
- Too many false positives: better define what is a possible threats. Add more constraints during the processing
  - Use smt-solver or dynamic code execution of sinks using predefined tests to minimise the false positives.