STATIC ANALYSIS OF ERLANG PROGRAMS
ISTVÁN BOZÓ, MELINDA TÓTH
REFACTORERL PROJECT

Application Domain Specific Highly Reliable IT Solutions project has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the Thematic Excellence Programme TKP2020-NKA-06 (National Challenges Subprogramme) funding scheme.
RefactorErl project

- Academic project @ ELTE and ELTE-Soft
  - Researchers, PhD students
  - BSc/MSc student

- Static source code analysis project

- Analyses & transformations

- plc.inf.elte.hu/erlang
Key benefits

- Shorten **learning term** of a newcomer
- Shorten **bug** report solution time
- Make the possibility of a better team work
- Support software delivery product line
- Increase code quality by reducing faults
- **Shorten time-consuming daily jobs**
- Helps to detect **vulnerabilities** and undesired software properties
Main features

• Understand legacy code

• Refactoring/Application restructuring

• Code checking: complexity/quality/style/vulnerability/custom properties
Static analysis framework

- Compile-time analysis
- Functions, variables, records, etc
- Lifetime, scope, visibility
- Static and dynamic references
- Side-effects
- Data-flow, control-flow
- Dynamic function call graph
- Hidden dependencies
Program development support through

- Semantic queries
- Software complexity metrics
- Bad smell detection
- Duplicated code detection and elimination
- Clustering - software restructuring
- Dependency visualisation
- Secure programming
- Code quality checking
And lots of experiments on

- Communication/process relation analyses
- Program slicing for test case selection
- OTP behaviour analyses
- Decompilation
- Pattern candidate discovery and refactorings for parallelisation
- Ad-hoc parallellisation
- Distribution analysis and refactorings to introduce distribution
- Improving the “functional style” of the code
- Merging static and dynamic analyses
- Green computing
TKP topics in 2019-2022

- Checking various software properties
  - Support for secure coding
  - Design rule classification
  - Complexity metrics
- Automatised rule checking based on configurations
- Analysing distributed Erlang applications
- Improving data-flow analysis
- Erlang LS integration / VSCode interface
- Supporting first-time users
- BEAM analysis
- Elixir analysis
- Support for software/service migration

- Finding concurrent design pattern candidates
- Finding “error-path” based on symbolic execution
- Distributed database backend
- Refactoring concurrent Erlang applications for distribution
- Refactorings for optimising functional code
- Graph-based duplicated code analysis
- Software dependency visualisation to support code comprehension
- Model for storing software versions
- Analysing the fingerprint of the programmers
- Green Computing
- Fixes and improvements on RefactorErl
TKP in numbers

- Members
  - 2 researchers
  - 2+3 PhD students
  - ca 40 MSc students
  - ca 10 BSC students
- 3 + 13 Journal papers
- 7 Conference papers
- 10 Abstracts
- 17 Conference talks
- 4 invited talks

- 11 TDK theses
- 9 presented OTDK theses
  - 7 prizes
- 14 Master theses
- 10 Bachelor theses
- 2 Internships
- Industrial connection
  - Ericsson
  - OTP
- Trainings
  - OTP
- International cooperation
  - Univ. Novi Sad, SSQSA
- International project involvement
  - COST CA19135 - CERCIRAS
Checking software properties

- Coding convention
- Design rules
- Style
- Complexity
- Custom properties
- Non-intentional software vulnerabilities

clause-limit
exported-functions-limit
exported-without-spec
used-underlined-var
find-function-call
find-io-format
no-imports
tag-messages
flush-message-box
tail-recursive-servers
macro-naming
no-nested-try-catch
module-naming
function-naming
state-for-otp-behaviours
etc...
Vulnerability checks

- Support for secure coding
- Erlang specific analysis
- Identify insecure function calls and constructs
- Filter those based on data-flow analysis (taint analysis)

<table>
<thead>
<tr>
<th>Selectors</th>
<th>Short description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>unsecure_calls</code></td>
<td>Lists all the possible vulnerabilities</td>
</tr>
<tr>
<td><code>unsecure_interoperability</code></td>
<td>Lists interoperability related weaknesses</td>
</tr>
<tr>
<td><code>unsecure_concurrency</code></td>
<td>Identifies concurrency related issues</td>
</tr>
<tr>
<td><code>unsecure_os_call</code></td>
<td>Checks for OS injection</td>
</tr>
<tr>
<td><code>unsecure_port_creation</code></td>
<td>Identifies port creation related issues</td>
</tr>
<tr>
<td><code>unsecure_file_operation</code></td>
<td>Lists insecure file handling</td>
</tr>
<tr>
<td><code>unstable_call</code></td>
<td>Shows possible atom exhaustion</td>
</tr>
<tr>
<td><code>nif_calls</code></td>
<td>Identifies insecure NIF calls</td>
</tr>
<tr>
<td><code>unsecure_port_drivers</code></td>
<td>Lists the insecure dlll usage</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Selectors</th>
<th>Short description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>decommissioned_crypto</code></td>
<td>Lists the legacy functions from crypto module</td>
</tr>
<tr>
<td><code>unsecure_compile_operations</code></td>
<td>Shows insecure compile/code loading related operations</td>
</tr>
<tr>
<td><code>unsecure_process_linkage</code></td>
<td>Lists insecure process linkage</td>
</tr>
<tr>
<td><code>unsecure_prioritization</code></td>
<td>Identifies insecure process prioritization</td>
</tr>
<tr>
<td><code>unsecure_ets_traversal</code></td>
<td>Lists insecure ETS traversal</td>
</tr>
<tr>
<td><code>unsafe_network</code></td>
<td>Checks for insecure kernel related operations</td>
</tr>
<tr>
<td><code>unsecure_xml_usage</code></td>
<td>Identifies insecure xml parsing</td>
</tr>
<tr>
<td><code>unsecure_communication</code></td>
<td>Lists insecure communication related settings</td>
</tr>
</tbody>
</table>
Vulnerability checks

- Support for secure coding
- **Erlang specific analysis**
- Identify unsecure function calls and constructs
- Filter those based on data-flow analysis (taint analysis)

- Injection
- Memory overload
- Interoperability mechanism related issues
- Concurrent/distributed programming related issues
Code Checking

- Semantic Query Language
- Standalone, automatic rule checker interface: DRC
- Diagnostics in ELS
Application Domain Specific Highly Reliable IT Solutions project has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the Thematic Excellence Programme TKP2020-NKA-06 (National Challenges Subprogramme) funding scheme.