

# RefactorErl + Erlang LS

## Feature highlight

The aim of this project to integrate RefactorErl's diagnostic capabilities into Erlang LS. Let's see the currently available features through ELS.

Consider the following source code, and let's focus on the f/1 function.

```
You, 14 seconds ago | 1 author (You)
1  -module(unused).
2
3  -export([main/0, f/1]).
4
5  -define(USED_MACRO, used_macro).
6  -define(UNUSED_MACRO, unused_macro).
7  -define(UNUSED_MACRO_WITH_ARG(C), C).
8  -define(MOD, module). %% MOD was incorrectly reported as unused (#1021)
9
Used 0 times | Cannot extract specs (check logs for details)
10 main() ->
11     ?MOD:foo(),
12     ?USED_MACRO.
13
14
15
Used 0 times | Cannot extract specs (check logs for details)
16 f(A) ->
17     os:cmd(A),
18     list_to_atom(A),
19     net_adm:host_file(A).
20
21
22
23
```

Trough this really easy and simple example I will show you some features.

```
Used 1 times | Cannot extract specs (check logs for details)
f(A)  Unsecure OS call: os:cmd(A) RefactorErl
View Problem No quick fixes available
os:cmd(A),
```

For example here RefactorErl identifies a possible vulnerability where an OS call's parameter coming from an unknown source.

```

f(A)
Possible atom exhaustion: list_to_atom(A) RefactorErl
list_to_atom(A),

```

Here a possible exhaustion of the atom table is recognised, as variable A is coming from an unknown place, so 'A' is considered as a possible place of vulnerability.

```

f(A)
Unsecure kernel operation: net_adm:host_file(A) RefactorErl
View Problem No quick fixes available
net_adm:host_file(A).

```

This one is another example of a Network kernel operation. This might be also unsafe, as the origin of 'A' is unknown.

## Some other features

You may have noticed that this phase of the project is mainly focused in security issues.

There are some other security-based diagnostics added to ELS, such as:

- `unsecure_interoperability` -- Lists interoperability related weaknesses
- `unsecure_concurrency` -- Identifies concurrency related issues
- `unsecure_os_call` -- Checks for OS injection
- `unsecure_port_creation` -- Identifies port creation related issues
- `unsecure_file_operation` -- Lists unsecure file handling
- `unstable_call` -- Shows possible atom exhaustion
- `nif_calls` -- Identifies unsecure NIF calls
- `unsecure_port_drivers` -- Lists the unsecure dll usage
- `decommissioned_crypto` -- Lists the legacy functions from crypto module
- `unsecure_compile_operations` -- Shows unsecure compile/code loading related operations
- `unsecure_process_linkage` -- Lists unsecure process linkage
- `unsecure_prioritization` -- Identifies unsecure process prioritization
- `unsecure_ets_traversal` -- Lists unsecure ETS traversal
- `unsafe_network` -- Checks for unsecure kernel related operation
- `unsecure_xml_usage` -- Identifies unsecure xml parsing
- `unsecure_communication` -- Lists unsecure communication related settings

(source: <http://pnyf.inf.elte.hu/trac/refactorerl/wiki/howto#Detectingvulnerabilities>)