Supporting Secure Coding with RefactorErl

Brigitta Baranyai
Eötvös Loránd University
Budapest, Hungary
Growing number of cyber threats in the era of Internet.
In order to improve the security of the systems, there are several standards and static analyser tools.
The lack of security analyser tools in case of Erlang.
Distributed processes run isolated with their own resources.

Immutable data structures.

Pure functions, modularity.

Fault tolerance as a core language concept.
Secure Coding in Erlang

- Interoperability mechanism related vulnerabilities
- Concurrent programming related issues
- Distributed programming related issues
- Injection
- Memory overload related attacks
INTEROPERABILITY MECHANISM

RELATED VULNERABILITIES

Using Erlang ports:

```erlang
-module(complex1).
-export([start/1, init/1]).

start(ExtPrg) ->
    spawn(?MODULE, init, [ExtPrg]).

init(ExtPrg) ->
    register(complex, self()),
    process_flag(trap_exit, true),
    loop(open_port([{spawn, ExtPrg}],
                   [{packet, 2}])).
```

Using dynamically loaded libraries (erl_ddll) or NIF:

```erlang
-module(complex2).
-export([foo/1]).

-on_load(init/0).

init() ->
    {ok, ExtPrg} = io:read("Provide a program..."),
    ok = erlang:load_nif(ExtPrg, 0).

foo(_X) -> exit(nif_library_not_loaded).
```
CONCURRENT PROGRAMMING RELATED ISSUES

Not connecting processes in an atomic way

Modifying process priority

ETS traversal without table fixes
DISTRIBUTED PROGRAMMING RELATED ISSUES

Using the network kernel related functions:

```c
net_kernel:allow/1,
net_kernel:connect_node/1,
net_kernel:start/1
```

SSL-3.0 and TLS-1.0 protocol configuration options for communication over sockets via the ssl module which can call forft Man-in-the-middle attacks:

```c
ssl:connect("example.net", 443, [
  {padding_check, false},
  {beast_mitigation, disabled},
  {fallback, true}
]);
```

With the evolution of the OpenSSL package some of the functions of the crypto module became obsolete:

```c
crypto:block_encrypt/3/4,
crypto:block_decrypt/3/4,
crypto:cmac/3/4, crypto:hmac/3/4, ...
```
OS commands called with unknown input:

- `module(injection).
- export([run_cmd/1]).`

`run_cmd(Input) ->
  os:cmd("cat " ++ Input).`

File related operations with unknown input data:

- `module(injection).
- export([eval/1]).`

`eval(Input) ->
  file:eval(Input).`

Dynamically loaded program code coming from unknown data source:

- `module(injection).
- export([load/1]).`

`load(Input) ->
  code:load_file(Input).`
MEMORY OVERLOAD RELATED ATTACKS

Dynamic atom creation related functions:

```prolog
parse_uri(Input) ->
  http_uri:parse(Input, [{ipv6_host_with_brackets, true}]).
```

XML parsing related functions without the usage of proper event handlers for preventing the internal or external entity expansion:

```prolog
parse_xml(Input) ->
  xmerl_sax_parser:stream(Input, []).
```
Static source code analyser tool.

Source code transformations without behaviour change.

Helps in understanding huge code bases, their maintenance or even investigating bugs by tracing back their origin.

Integrates well with editors like Emacs, Vim, Visual Studio Code and Eclipse.

Provides a web interface or command line tool through interactive shell.
• Determine the function call locations which are associated with unsecure operations.
• Select the functions parameters that can be associated with potential vulnerabilities.
• Run dataflow analysis on the sensitive parameters.
• Flag parameters with unknown source.
• Filter out functions provided by the users for input validation.
Provides syntactic and semantic information about Erlang programs by querying the call chains, function calls appearing in expressions, etc.

The units of the query language correspond to the semantic language elements of Erlang, which include the following: files, functions, function parameters, expressions, variables, etc.
RESULTS OF THE SECURITY CHECKER OF REFACTORERL

(defactorerl@localhost)31> ri:q("mods.funs.unsecure_calls").
coap_client:resolve_uri/1
   {ok, {Scheme, UserInfo, Host, PortNo, Path, Query}} =
   http_uri:parse(Uri, [{scheme_defaults, [{coap, ?DEFAULT_COAP_PORT},
   [coaps, ?DEFAULT_COAPS_PORT]]}])
coap_server_content:filter/2
   filter(
     case binary:split(Search, <<$>>) of
       [Name0, Value0] ->
         Name = list_to_atom(binary to list(Name0)),
         Value = wildcard_value(Value0),
         lists:filter(
           fun (Link) -> match_link(Link, Name, Value) end,
           Links);
       _Else -> Links
     end,
     Query)
ok
(refactorerl@localhost)32> 

RESULTS OF THE SECURITY CHECKER OF PEST

brigi@debVM:~$ /Projects/erlang$ ~Projects/pest/pest/pest.erl
  -r -s 0 relay/gen_coap/ build/default/lib/gen_coap/ebin/
  15: Keep OpenSSL updated for crypto module use (run with "-V
  crypto")
  coap_dtls_listen.beam:19 (ssl:_/)
  coap_dtls_socket.beam:[32,43,47,60,64] (ssl:_/)
brigi@debVM:~$ /Projects/erlang$
Additional rules to identify race conditions, obsolete cypher algorithms from the crypto module. Add security level related settings, configurable analysis to further enhance the user experience.
THANK YOU FOR YOUR ATTENTION!

This work was supported by the construction EFOP-3.6.3-VEKOP-16-2017-00002. The project was supported by the European Union, co-financed by the European Social Fund.