Skipper

Generated by Doxygen 1.9.4

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 CodeSegmentDescriber Struct Reference

just pair-structure <code segment start offset, code segment end offset>

`#include <Tracer.h>`

Public Attributes

- size_t **start**
- size_t **end**

4.1.1 Detailed Description

just pair-structure <code segment start offset, code segment end offset>

The documentation for this struct was generated from the following file:

- /home/debian/Skipper/src/include/classes/Tracer.h

4.2 Configurator Class Reference

Contains and provides configuration settings.

`#include <Config.h>`

Public Member Functions

- **Configurator** (const std::string config_file_name="")
- void load_config (std::string config_file_name="settings.json")

  loads config directly from passed file_path
- void config (json config_data)

  updates config
- void config (std::string config_string)

  updates config
- json getTracerConfig () const

  get tracer's config part
- json getFuzzConfig () const

  get fuzzing' config part
- std::map< std::string, FuncConfig > getInspectionFunctions () const

  get information about functions for instrumentation
- bool logSymbolsEnabled () const

  shall client log all symbols found in program's modules
- std::string getLogSymbolsPath () const

  get logging path for symbols
- bool logFuzzingEnabled () const

  shall client log fuzzing events or not
- std::string getLogFuzzingPath () const

  get logging path for fuzzing events
- std::string getFuzzingCorpusPath () const

  where to store fuzzing corpus
- bool use_default_bounds () const

  shall client use passed default function bounds or not
- std::set< int > getInspectOpcodes () const

  get opcodes which will be traced
- std::set< std::string > get_modules_names () const

  Get the modules names, containing functions under instrumentation.
- std::set< std::pair< std::string, std::string > > get_modules_info () const

  Get the modules info.
- bool debugModeEnabled () const

  shall we log debug information or not

Public Attributes

- json **tracer_config**
- json **fuzzing_config**
- json **_config**

4.2.1 Detailed Description

Contains and provides configuration settings.

4.2.2 Member Function Documentation

4.2.2.1 config() [1/2]

```
void Configurator::config (
            json config_data )  [inline]
```

updates config

Parameters

config_data	- json-like object, containing neccessery configurations

4.2.2.2 config() [2/2]

```
void Configurator::config (
            std::string config_string )  [inline]
```

updates config

Parameters

config_string	- serialized json string with configuration

4.2.2.3 debugModeEnabled()

```
bool Configurator::debugModeEnabled ( ) const  [inline]
```

shall we log debug information or not

Returns

4.2.2.4 get_modules_info()

```
std::set< std::pair< std::string, std::string > > Configurator::get_modules_info ( ) const
[inline]
```

Get the modules info.

Returns

set of pairs< module_name, module_path >

4.2.2.5 get_modules_names()

```
std::set< std::string > Configurator::get_modules_names ( ) const  [inline]
```

Get the modules names, containing functions under instrumentation.

Just walk through all functions in config and collect involved program modules.

Returns

set of modules' names

4.2.2.6 getFuzzConfig()

```
json Configurator::getFuzzConfig ( ) const  [inline]
```

get fuzzing' config part

Returns

fuzzing' config part

4.2.2.7 getFuzzingCorpusPath()

```
std::string Configurator::getFuzzingCorpusPath ( ) const  [inline]
```

where to store fuzzing corpus

Returns

depricated?

4.2.2.8 getInspectionFunctions()

```
std::map< std::string, FuncConfig > Configurator::getInspectionFunctions ( ) const  [inline]
```

get information about functions for instrumentation

Returns

dict {func_name : {module_name, module_path, default_address}}

4.2.2.9 getInspectOpcodes()

```
std::set< int > Configurator::getInspectOpcodes ( ) const  [inline]
```

get opcodes which will be traced

Returns

set of DynamoRIO enum macros opcodes

4.2.2.10 getLogFuzzingPath()

```
std::string Configurator::getLogFuzzingPath ( ) const  [inline]
```

get logging path for fuzzing events

Returns

path to file for logs writing

4.2.2.11 getLogSymbolsPath()

```
std::string Configurator::getLogSymbolsPath ( ) const  [inline]
```

get logging path for symbols

Returns

path to file for logs writing

4.2.2.12 getTracerConfig()

```
json Configurator::getTracerConfig ( ) const  [inline]
```

get tracer's config part

Returns

tracer's config part

4.2.2.13 load_config()

```
void Configurator::load_config (
            std::string config_file_name = "settings.json" )  [inline]
```

loads config directly from passed file_path

Parameters

config_file_name	- path to configuration JSON file

4.2.2.14 logFuzzingEnabled()

```
bool Configurator::logFuzzingEnabled ( ) const  [inline]
```

shall client log fuzzing events or not

Returns

true is it must and false otherwise

4.2.2.15 logSymbolsEnabled()

```
bool Configurator::logSymbolsEnabled ( ) const  [inline]
```

shall client log all symbols found in program's modules

Returns

true is it must and false otherwise

4.2.2.16 use_default_bounds()

```
bool Configurator::use_default_bounds ( ) const  [inline]
```

shall client use passed default function bounds or not

Returns

true - shall use

false - shall not

The documentation for this class was generated from the following file:

- /home/debian/Skipper/src/include/classes/Config.h

4.3 FuncConfig Struct Reference

Contains function-under-test info.

`#include <types.h>`

Public Attributes

- std::string **module_name**

  name of the module, where it will be searched by client
- std::string **module_path**

  path to that module
- std::pair< size_t, size_t > **default_address**

  addresses, which will be assigned as a bounds of function in case of search failure and if user will not enter address manualy

4.3.1 Detailed Description

Contains function-under-test info.

The documentation for this struct was generated from the following file:

- /home/debian/Skipper/src/include/types.h

4.4 Guarder Class Reference

This class is responsible for tracking the opening and closing of the "gates".

`#include <Guarder.h>`

Public Member Functions

- void set_global_guards (std::map< std::string, std::vector< long long int > > guards)

  Set the global guards object.
- bool guards_opened () const

  wether LLVM "guards" are opened
- bool throw_instr (instr_t *instr)

  Update LLVM-guards state, using current instruction.

Protected Attributes

- bool **good_lea_met_** = false

  Tells wether we met lea instr with expected operands.
- bool **guards_opened_** = false

  Tells wether "guards" were opened or not.
- std::map< std::string, std::vector< long long int > > **global_guards_**

4.4.1 Detailed Description

This class is responsible for tracking the opening and closing of the "gates".

4.4.2 Member Function Documentation

4.4.2.1 guards_opened()

```
bool Guarder::guards_opened ( ) const  [inline]
```

wether LLVM "guards" are opened

Returns

4.4.2.2 set_global_guards()

```
void Guarder::set_global_guards (
            std::map< std::string, std::vector< long long int > > guards )  [inline]
```

Set the global guards object.

Parameters

guards	

4.4.2.3 throw_instr()

```
bool Guarder::throw_instr (
            instr_t * instr )  [inline]
```

Update LLVM-guards state, using current instruction.

Use fixed pattern:

- LEA guard-address
- MOV

If mov 1 - guards are opening here, else guards are closing

Parameters

instr	- current basic block instruction

Returns

wheter or not LLVM-gurds are opened now

The documentation for this class was generated from the following file:

- /home/debian/Skipper/src/include/classes/Guarder.h

4.5 Logger Class Reference

Logger class.

`#include <Logger.h>`

Public Member Functions

- **Logger** (const std::string &out_file)
- void set_log_file (const std::string &new_out_file)

  Set the file to logs collection.
- void start_logging (std::string new_out_file="", bool use_tid=false)

  Prepare logger for working. Opens log-file descriptor.
- void **stop_logging** ()

  Close file descriptor / stream.
- template<typename... Args>
  void log (std::string tag, const std::string &format, Args... args)

  arbitrary log function
- template<typename T , typename... Args>
  void **formatString** (std::ostringstream &oss, const std::string &format, T value, Args... args)
- void **formatString** (std::ostringstream &oss, const std::string &format)
- bool **is_open** () const

  checks whether logging stream is opened or not
- bool log_program_params ()

  Logs info about client program and passed args: path, number of arguments (argc), arguments from argv.
- void **log_line** ()
- template<typename... Args>
  void **log_debug** (const std::string &format, Args... args)

  Call this->log method with `tag = "DEBUG"`
- template<typename... Args>
  void **log_info** (const std::string &format, Args... args)

  Call this->log method with `tag = "INFO"`
- template<typename... Args>
  void **log_error** (const std::string &format, Args... args)

  Call this->log method with `tag = "ERROR"`
- template<typename... Args>
  void **log_warning** (const std::string &format, Args... args)

  Call this->log method with `tag = "WARNING"`
- void **log_modules** ()

  Logs all visible by DR modules.

4.5.1 Detailed Description

Logger class.

4.5.2 Member Function Documentation

4.5.2.1 log()

```
template<typename...  Args>
void Logger::log (
             std::string tag,
             const std::string & format,
             Args...  args )  [inline]
```

arbitrary log function

apply args to '{}' inside passed message string and write it into log-file with passed `tag`

Template Parameters

Args	- args for

Parameters

tag	- tag of message. message will be like "[tag] : message"
format	- message string with '{}' to insert passed args
args	- agrs to insert into format message string

4.5.2.2 log_program_params()

```
bool Logger::log_program_params ( )  [inline]
```

Logs info about client program and passed args: path, number of arguments (argc), arguments from argv.

Returns

4.5.2.3 set_log_file()

```
void Logger::set_log_file (
             const std::string & new_out_file )  [inline]
```

Set the file to logs collection.

Parameters

new_out_file	- file to write logs in

4.5.2.4 start_logging()

```
void Logger::start_logging (
            std::string new_out_file = "",
            bool use_tid = false )  [inline]
```

Prepare logger for working. Opens log-file descriptor.

Parameters

new_out_file	- optional new log-file name to set
use_tid	- shall logger log messages to file with thread-id prefix like '1234-log-file-name.txt'

The documentation for this class was generated from the following file:

- /home/debian/Skipper/src/include/classes/Logger.h

4.6 ModuleInfo Struct Reference

Just pair <module name, module path> struct.

`#include <types.h>`

Public Attributes

- std::string **name**
- std::string **path**

4.6.1 Detailed Description

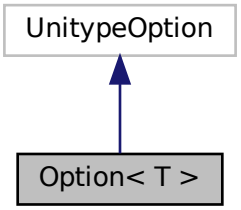
Just pair <module name, module path> struct.

The documentation for this struct was generated from the following file:

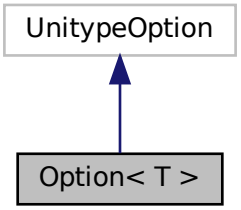
- /home/debian/Skipper/src/include/types.h

4.7 Option< T > Class Template Reference

Inheritance diagram for Option< T >:



Collaboration diagram for Option< T >:



Public Member Functions

- bool clamp_value () override
- **Option** (std::string name, T defval, std::string desc_short, std::string desc_long)
- **Option** (std::string name, T defval, T minval, T maxval, std::string desc_short, std::string desc_long)
- **Option** (const Option &option)
- Option & **operator=** (const Option &other)
- std::string get_value_str () const override
- std::string **get_value_separator** () const
- T **get_value** () const
- void clear_value () override
- void **set_value** (T new_value)
- bool convert_from_string (const std::string s) override
- bool option_takes_arg () const override
- bool convert_from_string (const std::string s)
- bool convert_from_string (const std::string s)
- bool convert_from_string (const std::string s)
- bool convert_from_string (const std::string s)

- bool convert_from_string (const std::string s)
- bool convert_from_string (const std::string s)
- bool convert_from_string (const std::string s)
- bool convert_from_string (const std::string s)
- bool option_takes_arg () const

Public Attributes

- T **value_**
- T **defval_**
- T **minval_**
- T **maxval_**
- bool **has_range_**

4.7.1 Member Function Documentation

4.7.1.1 clamp_value()

```
template<typename T >
bool Option< T >::clamp_value ( )  [inline], [override], [virtual]
```

Implements UnitypeOption.

4.7.1.2 clear_value()

```
template<typename T >
void Option< T >::clear_value ( )  [inline], [override], [virtual]
```

Reimplemented from UnitypeOption.

4.7.1.3 convert_from_string() [1/9]

```
bool Option< std::string >::convert_from_string (
            const std::string s )  [inline], [virtual]
```

Implements UnitypeOption.

4.7.1.4 convert_from_string() [2/9]

```
bool Option< int >::convert_from_string (
            const std::string s )  [inline], [virtual]
```

Implements UnitypeOption.

4.7.1.5 convert_from_string() [3/9]

```
bool Option< long >::convert_from_string (
            const std::string s )  [inline], [virtual]
```

Implements UnitypeOption.

4.7.1.6 convert_from_string() [4/9]

```
bool Option< longlong >::convert_from_string (
            const std::string s )  [inline], [virtual]
```

Implements UnitypeOption.

4.7.1.7 convert_from_string() [5/9]

```
bool Option< unsignedint >::convert_from_string (
            const std::string s )  [inline], [virtual]
```

Implements UnitypeOption.

4.7.1.8 convert_from_string() [6/9]

```
bool Option< unsignedlong >::convert_from_string (
            const std::string s )  [inline], [virtual]
```

Implements UnitypeOption.

4.7.1.9 convert_from_string() [7/9]

```
bool Option< unsignedlonglong >::convert_from_string (
            const std::string s )  [inline], [virtual]
```

Implements UnitypeOption.

4.7.1.10 convert_from_string() [8/9]

```
bool Option< double >::convert_from_string (
            const std::string s )  [inline], [virtual]
```

Implements UnitypeOption.

4.7.1.11 convert_from_string() [9/9]

```
template<typename T >
bool Option< T >::convert_from_string (
            const std::string s )  [override], [virtual]
```

Implements UnitypeOption.

4.7.1.12 get_value_str()

```
template<typename T >
std::string Option< T >::get_value_str ( ) const  [inline], [override], [virtual]
```

Implements UnitypeOption.

4.7.1.13 option_takes_arg() [1/2]

```
bool Option< bool >::option_takes_arg ( ) const  [inline], [virtual]
```

Implements UnitypeOption.

4.7.1.14 option_takes_arg() [2/2]

```
template<typename T >
bool Option< T >::option_takes_arg  [inline], [override], [virtual]
```

Implements UnitypeOption.

The documentation for this class was generated from the following file:

- /home/debian/Skipper/src/include/classes/Options.h

4.8 Parser Class Reference

Public Member Functions

- bool **parse_argv** (int argc, const char ∗argv[ ], std::string ∗error_msg, int ∗last_index)
- bool **add_option** (std::shared_ptr< UnitypeOption > option)
- std::vector< std::shared_ptr< UnitypeOption > > **get_options** ()
- std::shared_ptr< UnitypeOption > & **operator[ ]** (const std::string &name)
- const std::shared_ptr< UnitypeOption > & **operator[ ]** (const std::string &name) const

Public Attributes

- std::vector< std::shared_ptr< UnitypeOption > > **options_** = {}

The documentation for this class was generated from the following file:

- /home/debian/Skipper/src/include/classes/Options.h

4.9 sym_info_t Struct Reference

Contains all aviable info about some symbol.

`#include <types.h>`

Public Member Functions

- **sym_info_t** (const char ∗name, size_t off, bool imp=false)
- **sym_info_t** (const drsym_info_t ∗info, bool imp=false)
- **sym_info_t** (const sym_info_t &sym)

Public Attributes

- std::string **file**
- uint64 **line**
- size_t **line_offs**
- size_t **start_offs**
- size_t **end_offs**
- int **debug_kind**
- std::string **name**
- uint **type_id** = 0
- uint **flags**
- bool **ex** = false
- bool **imp** = true
- size_t **moffs** = 0

4.9.1 Detailed Description

Contains all aviable info about some symbol.

The documentation for this struct was generated from the following file:

- /home/debian/Skipper/src/include/types.h

4.10 thread_data Struct Reference

depricated

`#include <types.h>`

Public Attributes

- uint64_t **location**
- uint8_t **map** [MAP_SIZE]

4.10.1 Detailed Description

depricated

The documentation for this struct was generated from the following file:

- /home/debian/Skipper/src/include/types.h

4.11 TraceArea Struct Reference

extra-counters section (memory area for extra trace for libfuzzer), contains bounds and size.

`#include <Tracer.h>`

Public Attributes

- size_t **start**
- size_t **end**
- size_t **size**

4.11.1 Detailed Description

extra-counters section (memory area for extra trace for libfuzzer), contains bounds and size.

The documentation for this struct was generated from the following file:

- /home/debian/Skipper/src/include/classes/Tracer.h

4.12 Tracer Class Reference

Class, that inserts tracing function in basic blocks.

`#include <Tracer.h>`

Public Member Functions

- bool **set_config** (Configurator config)

  set up configuration for client tracing
- void **set_trace_area** (size_t start, size_t end)

  set up trace-memory (extra-counters section) address
- void traceOverflow (void ∗drcontext, void ∗tag, instrlist_t ∗bb, instr_t ∗instr)

  This method inserts instrumentation into basic-blocks.

Public Attributes

- json **tracer_config**

Protected Member Functions

- int **get_reg_id** (reg_id_t reg)

  Get register ID if submited and submit if it is not.

4.12.1 Detailed Description

Class, that inserts tracing function in basic blocks.

4.12.2 Member Function Documentation

4.12.2.1 traceOverflow()

```
void Tracer::traceOverflow (
            void * drcontext,
            void * tag,
            instrlist_t * bb,
            instr_t * instr )  [inline]
```

This method inserts instrumentation into basic-blocks.

`use_asm` parameter regulates whether client shall use inline ASM insertion or clean call DynamoRIO function to insert tracing.

Parameters

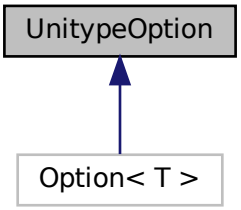
drcontext	- DR context
tag	- DR tag
bb	- current basic block
instr	- current ASM instruction

The documentation for this class was generated from the following file:

- /home/debian/Skipper/src/include/classes/Tracer.h

4.13 UnitypeOption Class Reference

Inheritance diagram for UnitypeOption:



Public Member Functions

- **UnitypeOption** (std::string name, std::string desc_short, std::string desc_long)
- std::string **get_name** () const
- bool **name_match** (const char ∗arg)
- void **set_is_specified** (bool x)
- bool **is_specified** () const
- virtual std::string **get_value_str** () const =0
- virtual bool **convert_from_string** (const std::string s)=0
- virtual bool **clamp_value** ()=0
- virtual bool **option_takes_arg** () const =0
- virtual void **clear_value** ()

Public Attributes

- std::string **valsep_**
- std::string **name_**
- bool **is_specified_**
- std::string **desc_short_**
- std::string **desc_long_**

The documentation for this class was generated from the following file:

- /home/debian/Skipper/src/include/classes/Options.h

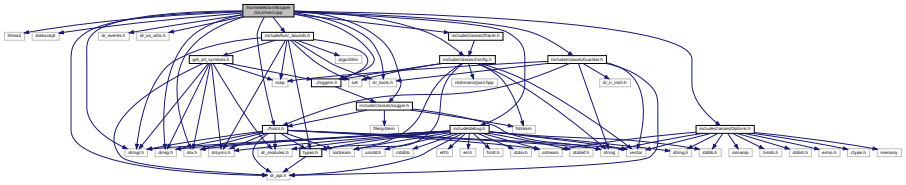
Chapter 5

File Documentation

5.1 /home/debian/Skipper/src/client.cpp File Reference

```
#include <thread>
#include <stdexcept>
#include "dr_api.h"
#include "dr_tools.h"
#include "dr_events.h"
#include "dr_os_utils.h"
#include "drmgr.h"
#include "drreg.h"
#include "drx.h"
#include "include/func_bounds.h"
#include "include/funcs.h"
#include "include/debug.h"
#include "include/classes/Config.h"
#include "include/classes/Logger.h"
#include "include/classes/Tracer.h"
#include "include/classes/Guarder.h"
#include "include/classes/Options.h"
#include "loggers.h"
```

Include dependency graph for client.cpp:



Functions

- void **init_tls** ()
- bool address_in_code_segment (void *tag, std::vector< CodeSegmentDescriber > &segments)

  Checks whether a piece of instruction is included in one of the instrumented segments.
- std::string **create_log_file_name** (std::thread::id id)
- void dr_client_main (client_id_t id, int argc, const char *argv[])

  Main client function.

5.1.1 Function Documentation

5.1.1.1 address_in_code_segment()

```
bool address_in_code_segment (
            void * tag,
            std::vector< CodeSegmentDescriber > & segments )
```

Checks whether a piece of instruction is included in one of the instrumented segments.

Parameters

tag	- DynamoRIO code segment tag
segments	- found segments (passed through the config)

5.1.1.2 dr_client_main()

```
void dr_client_main (
            client_id_t id,
            int argc,
            const char * argv[] )
```

Main client function.

Parameters

id	- client id
argc	- number of command line arguments
argv	- command line arguments

In this function, the client is configured and prepared for operation, and all handlers are posted for events occurring in the fuzzer-program process.

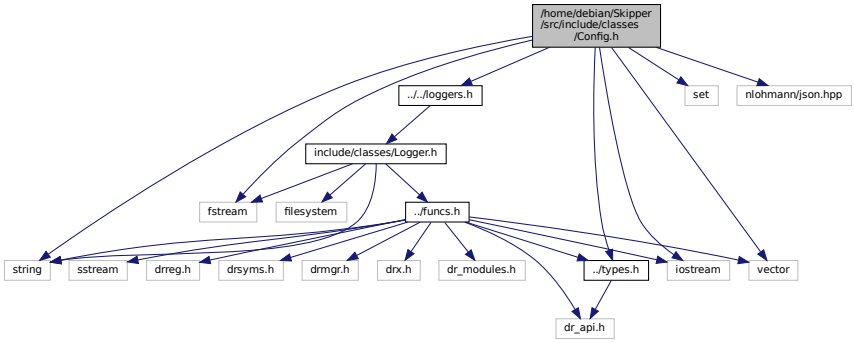
5.2 /home/debian/Skipper/src/include/classes/Config.h File Reference

Config class description.

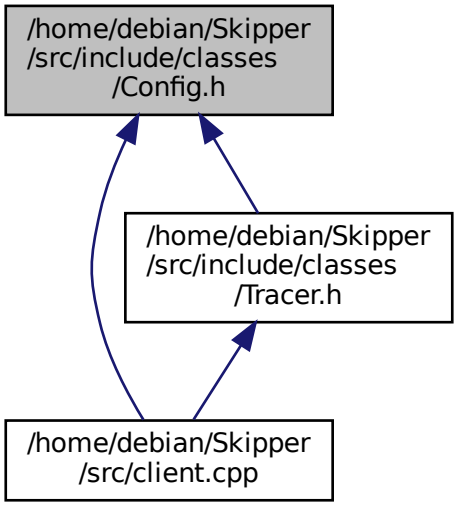
```
#include <fstream>
#include <iostream>
#include <set>
#include <string>
#include <vector>
#include <nlohmann/json.hpp>
#include "../types.h"
```

```
#include "../../loggers.h"
```

Include dependency graph for Config.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Configurator

  Contains and provides configuration settings.

Typedefs

- using **json** = nlohmann::json

5.2.1 Detailed Description

Config class description.

Author

Stepan Kafanov

Version

0.1

Date

2025-03-31

Copyright

Copyright (c) 2025

5.3 Config.h

Go to the documentation of this file.

```
1
11 #ifndef MY_CONFIG_header
12 #define MY_CONFIG_header
13
14 #include <fstream>
15 #include <iostream>
16 #include <set>
17 #include <string>
18 #include <vector>
19
20 #include <nlohmann/json.hpp>
21 #include "../types.h"
22 #include "../../loggers.h"
23
24 using json = nlohmann::json;
25
30 class Configurator {
31 private:
32     void config_with_json(json config_data) {
33         // ASM-inspection settings
34         // TODO
35
36         this->tracer_config = config_data["tracer_config"];
37     }
38
44     int get_opcode(std::string op_s)const {
45         if        (op_s == "add" ) {
46             return OP_add;
47         } else if (op_s == "adc" ) {
48             return OP_adc;
49         } else if (op_s == "sub" ) {
50             return OP_sub;
51         } else if (op_s == "sbb" ) {
52             return OP_sbb;
53         } else if (op_s == "adcx") {
54             return OP_adcx;
55         } else if (op_s == "adox") {
56             return OP_adox;
57         } else if (op_s == "cmp" ) {
58             return OP_cmp;
59         } else if (op_s == "movq") {
60             return OP_movq;
61         } else {
62             main_logger.log_error("cannot translate {} to DR opcode - unknown!", op_s);
63             throw std::runtime_error("we do not know this opcode's name!");
```

```
64              return -1;
65          }
66      }
67  public:
68      json tracer_config;
69      json fuzzing_config;
70      json _config;
71
72      Configurator(const std::string config_file_name = "") {
73          if (!config_file_name.empty()) {
74              this->load_config(config_file_name);
75          }
76      }
77
83      void load_config(std::string config_file_name = "settings.json") {
84          std::cout « "loading " « config_file_name « "..." « std::endl;
85          std::ifstream f(config_file_name, std::ifstream::in);
86          json config_data;
87          f » config_data;
88          this->_config = config_data;
89          std::cout « "configuration loaded!" « std::endl;
90      }
91
94      void config(json config_data) {
95          this->_config = config_data;
96      }
99      void config(std::string config_string) {
100         json config_data = json::parse(config_string);
101         this->_config = config_data;
102     }
103
106     json getTracerConfig()const {
107         return this->_config["tracer"];
108     }
109
112     json getFuzzConfig()const {
113         return this->_config["fuzzing"];
114     }
115
118     std::map<std::string, FuncConfig>
119     getInspectionFunctions()const {
120         std::cout « "inspection functions providing..." « std::endl;
121         auto f_info_json = this->_config["fuzzing"]["inspect_funcs"];
122         std::map<std::string, FuncConfig>
123         f_names;
124         for (auto f_info :  f_info_json) {
125             FuncConfig conf =   {
126                                     f_info["module_name"],
127                                     f_info["module_path"],
128                                     std::make_pair(
129                                         (size_t) std::stoull((std::string)
    f_info["default_address"]["start"], nullptr, 16),
130                                         (size_t) std::stoull((std::string)
    f_info["default_address"]["start"], nullptr, 16))
131                                 };
132             f_names[f_info["func_name"]] = conf;
133         }
134         std::cout « "functions collected successfuly!" « std::endl;
135         return f_names;
136     }
137
140     bool
141     logSymbolsEnabled()const {
142         if (!_config.contains("logging")) {
143             throw std::runtime_error("Missing 'logging' section");
144         }
145         return this->_config["logging"]["log_symbols"]["enable"];
146     }
147
150     std::string
151     getLogSymbolsPath()const {
152         if (!_config.contains("logging")) {
153             throw std::runtime_error("Missing 'logging' section");
154         }
155         bool enable = this->_config["logging"]["log_symbols"]["enable"];
156         if (!enable) {
157             std::cout « "[WARNING] : symbols logging is disabled" « std::endl;
158         }
159         return this->_config["logging"]["log_symbols"]["path"];
160     }
161
164     bool
165     logFuzzingEnabled()const {
166         if (!_config.contains("logging")) {
167             throw std::runtime_error("Missing 'logging' section");
168         }
169         return this->_config["logging"]["log_fuzzing"]["enable"];
```

```
170     }
171
174     std::string
175     getLogFuzzingPath()const {
176         if (!_config.contains("logging")) {
177             throw std::runtime_error("Missing 'logging' section");
178         }
179         bool enable = this->_config["logging"]["log_fuzzing"]["enable"];
180         if (!enable) {
181             std::cout << "[WARNING] : fuzzing logging is disabled" << std::endl;
182         }
183         return this->_config["logging"]["log_fuzzing"]["path"];
184     }
185
190     std::string
191     getFuzzingCorpusPath()const {
192         return this->_config["fuzzing"]["corpus_path"];
193     }
194
201     bool
202     use_default_bounds()const {
203         return this->_config["fuzzing"]["use_default"];
204     }
205
208     std::set<int> getInspectOpcodes()const {
209         dr_printf("getting opcodes to inspect...");
210         std::set<int> ops;
211         for (auto op :  this->_config["fuzzing"]["inspect_opcodes"]) {
212             auto op_dr = this->get_opcode((std::string) op);
213             if (op_dr < 0) {
214                 main_logger.log_error("cannot translate {} to DR opcode", (std::string) op);
215             }
216             ops.insert(op_dr);
217         }
218         return ops;
219     }
220
228     std::set <std::string> get_modules_names()const {
229         main_logger.log_info("getting modules names...");
230         std::set <std::string> module_names;
231         for (auto mn :  this->_config["fuzzing"]["inspect_funcs"]) {
232             std::string name = mn["module_name"];
233             module_names.insert((std::string) name);
234         }
235         return module_names;
236     }
237
243     std::set <std::pair<std::string, std::string>> get_modules_info()const {
244         dr_printf("getting modules names...\n");
245         std::set <std::pair<std::string, std::string>> modules_info;
246         for (auto mn :  this->_config["fuzzing"]["inspect_funcs"]) {
247             std::string name = mn["module_name"];
248             std::string path = mn["module_path"];
249             modules_info.insert(std::make_pair(name, path));
250         }
251         return modules_info;
252     }
253
256     bool debugModeEnabled()const {
257         return this->_config["debug"];
258     }
259 };
260
261 #endif // MY_CONFIG_header
```

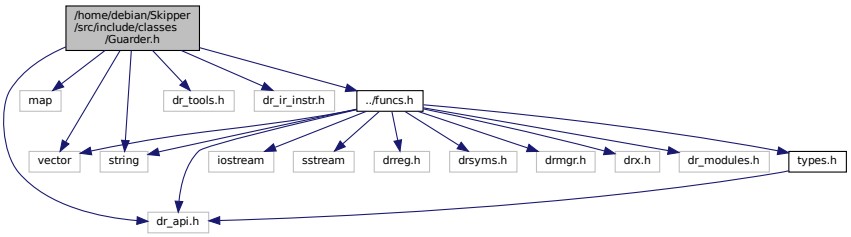
5.4 /home/debian/Skipper/src/include/classes/Guarder.h File Reference

Class for LLVM-guards tracking.

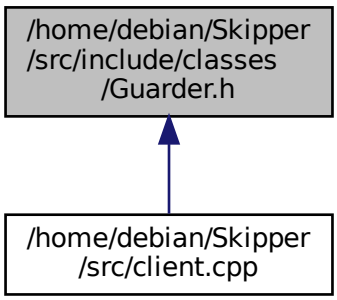
```
#include <vector>
#include <map>
#include <string>
#include "dr_api.h"
#include "dr_tools.h"
#include "dr_ir_instr.h"
```

```
#include "../funcs.h"
```

Include dependency graph for Guarder.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Guarder

  This class is responsible for tracking the opening and closing of the "gates".

5.4.1 Detailed Description

Class for LLVM-guards tracking.

Author

Stepan Kafanov

Version

0.1

Date

2025-03-31

Copyright

Copyright (c) 2025

5.5 Guarder.h

Go to the documentation of this file.

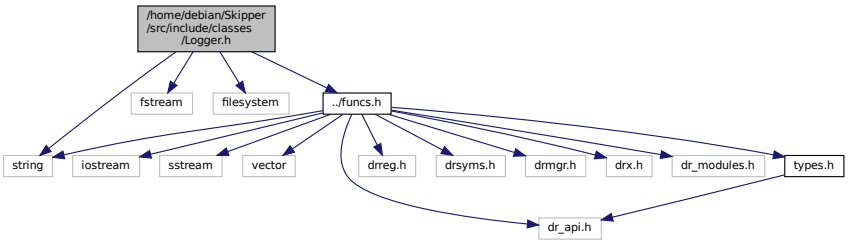
```
1
11 #ifndef MY_GUARDER_H
12 #define MY_GUARDER_H
13
14 #include <vector>
15 #include <map>
16 #include <string>
17
18 #define X86
19 #include "dr_api.h"
20 #include "dr_tools.h"
21 #include "dr_ir_instr.h"
22
23 #include "../funcs.h"
24
29 class Guarder {
30 protected:
32     bool good_lea_met_ = false;
34     bool guards_opened_ = false;
35     std::map<std::string, std::vector <long long int» global_guards_;
36
37 public:
38     Guarder() {}
39
45     void set_global_guards(std::map<std::string, std::vector <long long int» guards) {
46         this->global_guards_ = guards;
47     }
48
54     bool guards_opened()const {
55         return this->guards_opened_;
56     }
57
69     bool throw_instr(instr_t * instr) {
70         app_pc addr = instr_get_app_pc(instr);
71         module_data_t *mod = dr_lookup_module(addr);
72         auto module_name = std::string(dr_module_preferred_name(mod));
73         dr_free_module_data(mod);
74
75         int opcode = instr_get_opcode(instr);
76         if (instr_num_srcs(instr)) {
77             opnd_t src = instr_get_src(instr, 0);
78             if (opcode == (int) OP_lea && (opnd_is_far_memory_reference(src) ||
    opnd_is_near_memory_reference(src)) && opnd_is_abs_addr(src)) {
79                 dr_printf("2 opnd get addr\n");
80                 auto mem_addr = opnd_get_addr(src);
81                 dr_printf("addr has been gotten\n");
82
83                 if (std::find(  this->global_guards_[module_name].begin(),
84                                 this->global_guards_[module_name].end(),
85                                 (long long) mem_addr) != this->global_guards_[module_name].end()) {
86                     this->good_lea_met_ = true;
87                 } else {
88                     this->good_lea_met_ = false;
89                 }
90             } else {
91                 if (this->good_lea_met_ && instr_writes_memory(instr)) {
92                     if (opnd_is_immed_int64(src)) {
93                         long val = opnd_get_immed_int64(src);
94                         dr_printf("move opnd value is <%ld>\n", val);
95                         this->guards_opened_ = (val == 1);
96
97                         if (this->guards_opened_)
98                             dr_printf("open the gates!\n");
99                         else
100                             dr_printf("close the gates!\n");
101                     }
102                 }
103                 this->good_lea_met_ = false;
104             }
105         } else {
106             this->good_lea_met_ = false;
107         }
108         return this->guards_opened_;
109     }
110 };
111
112 #endif // MY_GUARDER_H
```

5.6 /home/debian/Skipper/src/include/classes/Logger.h File Reference

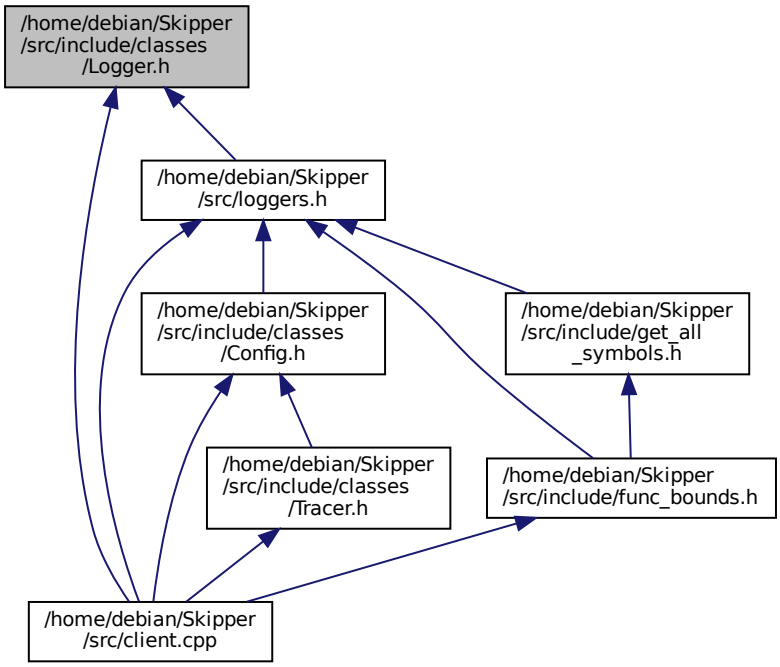
Logger class.

```
#include <string>
#include <fstream>
#include <filesystem>
#include "../funcs.h"
```

Include dependency graph for Logger.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Logger

  Logger class.

5.6.1 Detailed Description

Logger class.

Author

Stepan Kafanov

Version

0.1

Date

2025-03-31

Copyright

Copyright (c) 2025

5.7 Logger.h

Go to the documentation of this file.

```
1
11 #ifndef MY_LOGGER_header
12 #define MY_LOGGER_header
13
14 #include <string>
15 #include <fstream>
16 #include <filesystem>
17
18 #include "../funcs.h"
19
24 class Logger {
25 private:
26     std::string log_file;
27     std::ofstream stream;
28 public:
29     Logger() {}
30
31     Logger(const std::string & out_file) {
32         this->log_file = out_file;
33     }
34
40     void set_log_file(const std::string & new_out_file) {
41         if (this->stream.is_open()) {
42             this->stream.close();
43         }
44         this->log_file = new_out_file;
45     }
46
53     void start_logging(std::string new_out_file = "", bool use_tid = false) {
54         if (this->stream.is_open()) {
55             this->stream.close();
56         }
57         if (!new_out_file.empty()) {
58             this->log_file = new_out_file;
59         }
60
61         std::filesystem::path dir = std::filesystem::absolute(this->log_file).parent_path();
62         if (!dir.empty() && !std::filesystem::exists(dir)) {
63             std::filesystem::create_directories(dir);
64         }
65
66         std::cout « "stream opening:  " « this->log_file « std::endl;
67         try {
68             if (use_tid)
69                 this->stream.open(this->log_file + "-" + get_thread_id());
```

```
70              else
71                  this->stream.open(this->log_file);
72          } catch (...)  {
73              std::cout « "error with stream opening!" « std::endl;
74          }
75      }
76
81      void stop_logging() {
82          if (this->stream.is_open()) {
83              this->stream.close();
84          } else {
85              printf("logging is alreadry stopped!\n");
86          }
87      }
88
98      template<typename...  Args>
99      void log(std::string tag, const std::string& format, Args...  args) {
100         if (!this->stream) {
101             throw std::runtime_error("logging is impossible.  log stream is closed!");
102         }
103         std::ostringstream oss;
104         this->formatString(oss, format, args...);
105         this->stream « "[" « tag « "] :  " « oss.str() « std::endl;
106     }
107     template<typename T, typename...  Args>
108     void formatString(std::ostringstream& oss, const std::string& format, T value, Args...  args) {
109         size_t pos = format.find("{}");
110         if (pos != std::string::npos) {
111             oss « format.substr(0, pos) « value;
112             formatString(oss, format.substr(pos + 2), args...);
113         } else {
114             oss « format;
115         }
116     }
117     void formatString(std::ostringstream& oss, const std::string& format) {
118         oss « format;
119     }
120
122     bool is_open()const {
123         return this->stream.is_open();
124     }
125
126
129     bool log_program_params() {
130         this->log_line();
131         this->stream « "[DEBUG] : " « "app_name:  " « dr_get_application_name() « std::endl;
132         int num_args = dr_num_app_args();
133         this->stream « "[DEBUG] : " « "num_args:  " « num_args « std::endl;
134
135         if (num_args > 100) {
136             // num of args must be less or equal to 100
137             return false;
138         }
139
140         dr_app_arg_t args_array[100];
141         int err = dr_get_app_args(args_array, num_args);
142         if (err == -1) {
143             this->stream « "[ERROR] : " « "cannot get app args" « std::endl;
144             return false;
145         }
146
147         char buff[1000];
148         for (int i = 0; i < num_args; ++i) {
149             auto arg = dr_app_arg_as_cstring(&(args_array[i]), buff, sizeof(dr_app_arg_t)*10);
150             this->stream « "[DEBUG] :\t" « "arg-" « i « ":  " « arg « std::endl;
151         }
152         this->log_line();
153
154         return true;
155     }
156
157     void log_line() {
158         this->stream « "======================================================================" « std::endl;
159     }
160
162     template<typename...  Args>
163     void log_debug(const std::string& format, Args...  args) {
164         this->log("DEBUG", format, args...);
165     }
167     template<typename...  Args>
168     void log_info(const std::string& format, Args...  args) {
169         this->log("INFO", format, args...);
170     }
172     template<typename...  Args>
173     void log_error(const std::string& format, Args...  args) {
174         this->log("ERROR", format, args...);
175     }
```

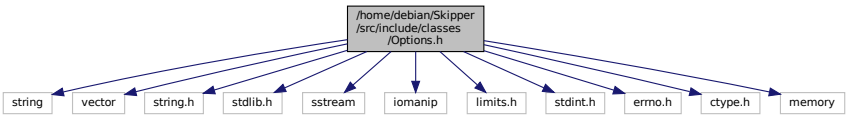
```
177     template<typename...  Args>
178     void log_warning(const std::string& format, Args...  args) {
179         this->log("WARNING", format, args...);
180     }
181 
183     void log_modules() {
184         auto modules = get_all_modules();
185         this->stream « "[DEBUG] : modules:" « std::endl;
186         for (auto module :  modules) {
187             this->stream « "\tmodule_name:  " « module.name « "; module_path:  " « module.path «
      std::endl;
188         }
189     }
190 };
191 
192 #endif // MY_LOGGER_header
```

5.8 /home/debian/Skipper/src/include/classes/Options.h File Reference

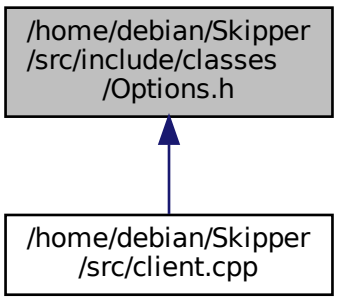
I adopted DynamoRIO parser for my project, because for some reason that parser didn't work.

```
#include <string>
#include <vector>
#include <string.h>
#include <stdlib.h>
#include <sstream>
#include <iomanip>
#include <limits.h>
#include <stdint.h>
#include <errno.h>
#include <ctype.h>
#include <memory>
```

Include dependency graph for Options.h:



This graph shows which files directly or indirectly include this file:



Classes

- class UnitypeOption
- class Option< T >
- class Parser

5.8.1 Detailed Description

I adopted DynamoRIO parser for my project, because for some reason that parser didn't work.

Author

Stepan Kafanov

Version

0.1

Date

2025-03-31

Copyright

Copyright (c) 2025

5.9 Options.h

Go to the documentation of this file.

```
1
11 #ifndef MY_OPTIONS_header
12 #define MY_OPTIONS_header
13
14 #include <string>
15 #include <vector>
16 #include <string.h>
17 #include <stdlib.h>
18 #include <sstream>
19 #include <iomanip>
20 #include <limits.h>
21 #include <stdint.h>
22 #include <errno.h>
23 #include <ctype.h>
24 #include <memory>
25
26 static bool
27 is_negative(const std::string &s)
28 {
29     for (size_t i = 0; i < s.size(); i++) {
30         if (isspace(s[i]))
31             continue;
32         if (s[i] == '-')
33             return true;
34         break;
35     }
36     return false;
37 }
38
39 class UnitypeOption {
40 public:
41     std::string valsep_;
42     std::string name_;
43     bool is_specified_;
```

```
44     std::string desc_short_;
45     std::string desc_long_;
46 public:
47     UnitypeOption(std::string name, std::string desc_short, std::string desc_long)
48         :  name_(name)
49         , desc_short_(desc_short)
50         , desc_long_(desc_long)
51         , valsep_(" ")
52         , is_specified_(false) {}
53     virtual ~UnitypeOption() {};
54 
55     std::string get_name()const {
56         return name_;
57     }
58     bool name_match(const char *arg) {
59         std::cout « "name matching" « std::endl;
60         std::cout « this->name_ « " " « arg « std::endl;
61         return std::string("-").append(this->name_) == arg || std::string("--").append(this->name_) ==
    arg;
62     }
63     void set_is_specified(bool x) {
64         is_specified_ = x;
65     }
66     bool is_specified()const {
67         return is_specified_;
68     }
69 
70     virtual std::string get_value_str() const = 0;
71     virtual bool convert_from_string(const std::string s) = 0;
72     virtual bool clamp_value() = 0;
73     virtual bool option_takes_arg() const = 0;
74     virtual void clear_value() {
75         is_specified_ = false;
76     }
77 };
78 
79 template <typename T>
80 class Option :  public UnitypeOption {
81 public:
82     T value_;
83     T defval_;
84     T minval_;
85     T maxval_;
86     bool has_range_;
87 
88     bool clamp_value()override {
89         if (has_range_) {
90             if (value_ < minval_) {
91                 value_ = minval_;
92                 return false;
93             } else if (value_ > maxval_) {
94                 value_ = maxval_;
95                 return false;
96             }
97         }
98         return true;
99     }
100 public:
101     Option(std::string name, T defval,                      std::string desc_short, std::string
    desc_long)
102         :  UnitypeOption(name, desc_short, desc_long)
103         , value_(defval)
104         , defval_(defval)
105         , has_range_(false) {}
106     Option(std::string name, T defval, T minval, T maxval, std::string desc_short, std::string
    desc_long)
107         :  UnitypeOption(name, desc_short, desc_long)
108         , value_(defval)
109         , defval_(defval)
110         , has_range_(true)
111         , minval_(minval)
112         , maxval_(maxval) {}
113 
114 
115     Option(const Option& option)
116         :  UnitypeOption(option.name_, option.desc_short, option.desc_long)
117         , value_(option.value_)
118         , defval_(option.defval_)
119         , has_range_(option.has_range_)
120         , minval_(option.minval_)
121         , maxval_(option.maxval_) {}
122     Option& operator=(const Option& other)
123     {
124         if (this != &other) {
125             value_ = other.value_;
126             defval_ = other.defval_;
127             valsep_ = other.valsep_;
```

```
128             has_range_ = other.has_range_;
129             minval_ = other.minval_;
130             maxval_ = other.maxval_;
131             name_ = other.name_;
132             is_specified_ = other.is_specified_;
133             desc_short_ = other.desc_short_;
134             desc_long_ = other.desc_long_;
135         }
136         return *this;
137     }
138     ~Option() override = default;
139 
140     std::string get_value_str()const override {
141         std::ostringstream ss;
142         ss « value_;
143         return ss.str();
144     }
145 
146     std::string get_value_separator()const {
147         return valsep_;
148     }
149 
150     T get_value()const {
151         return this->value_;
152     }
153 
154     void clear_value()override {
155         value_ = defval_;
156         is_specified_ = false;
157     }
158 
159     void set_value(T new_value) {
160         value_ = new_value;
161     }
162 
163 
164     bool convert_from_string(const std::string s) override;
165     bool option_takes_arg() const override;
166 };
167 
168 class Parser {
169 public:
170     std::vector <std::shared_ptr<UnitypeOption» options_ = {};
171 public:
172     Parser() {}
173 
174     bool parse_argv(int argc, const char *argv[], std::string *error_msg, int *last_index)
175     {
176         int i;
177         bool res = true;
178         for (i = 1 /*skip app*/; i < argc; ++i) {
179             // We support the universal "--" as a separator
180             if (strcmp(argv[i], "--") == 0) {
181                 ++i; // for last_index
182                 break;
183             }
184             // Also stop on a non-leading-dash token to support arguments without
185             // a separating "--".
186             if (argv[i][0] != '-') {
187                 break;
188             }
189             bool matched = false;
190             for (int j = 0; j < this->options_.size(); ++j) {
191                 auto op = this->options_[j];
192                 if (op->name_match(argv[i])) {
193                     matched = true;
194                     std::cout « "option takes arg..." « std::endl;
195                     if (op->option_takes_arg()) {
196                         ++i;
197                         if (i == argc) {
198                             if (error_msg != NULL) {
199                                 std::cout « "error";
200                                 *error_msg = "Option " + op->get_name() + " missing value";
201                             }
202                             res = false;
203                             goto parse_finished;
204                         }
205                         if (!op->convert_from_string(argv[i]) || !op->clamp_value()) {
206                             if (error_msg != NULL) {
207                                 std::cout « "error";
208                                 *error_msg = "Option " + op->get_name() + " value out of range";
209                             }
210                             res = false;
211                             goto parse_finished;
212                         }
213                     }
214                     std::cout « "setting is specified..." « std::endl;
```

```
215                     op->set_is_specified(true);
216                     // op->is_specified_ = true; // *after* convert_from_string()
217                 }
218             }
219             if (!matched) {
220                 if (error_msg != NULL) {
221                     std::cout << "error";
222                     *error_msg = std::string("Unknown option:  ") + argv[i];
223                 }
224                 res = false;
225                 goto parse_finished;
226             }
227         }
228     parse_finished:
229         if (last_index != NULL)
230             *last_index = i;
231         return res;
232     }
233 
234     bool add_option(std::shared_ptr<UnitypeOption> option) {
235         this->options_.push_back(option);
236         return true;
237     }
238 
239     std::vector <std::shared_ptr<UnitypeOption>> get_options() {
240         return this->options_;
241     }
242 
243     std::shared_ptr<UnitypeOption> & operator[](const std::string & name) {
244         for (int i = 0; i < this->options_.size(); ++i) {
245             if (this->options_[i]->get_name() == name) {
246                 return this->options_[i];
247             }
248         }
249         throw std::runtime_error(std::string("there is not operator with such name:  ") + name);
250     }
251     const std::shared_ptr<UnitypeOption> & operator[](const std::string & name)const {
252         for (int i = 0; i < this->options_.size(); ++i) {
253             if (this->options_[i]->get_name() == name) {
254                 return this->options_[i];
255             }
256         }
257         throw std::runtime_error(std::string("there is not operator with such name:  ") + name);
258     }
259 };
260 
261 template <>
262 inline bool
263 Option<std::string>::convert_from_string(const std::string s)
264 {
265     std::cout << "casting string with <" << s << ">..." << std::endl;
266     if (is_specified_)
267         value_ += valsep_ + s;
268     else
269         value_ = s;
270     return true;
271 }
272 template <>
273 inline bool
274 Option<int>::convert_from_string(const std::string s)
275 {
276     std::cout << "casting int with <" << s << ">..." << std::endl;
277     errno = 0;
278     // If we set 0 as the base, strtol() will automatically identify the base of the
279     // number to convert.  By default, it will assume the number to be converted is
280     // decimal, and a number starting with 0 or 0x is assumed to be octal or hexadecimal,
281     // respectively.
282     long input = strtol(s.c_str(), NULL, 0);
283     std::cout << "res:  " << input << std::endl;
284 
285     // strtol returns a long, but this may not always fit into an integer.
286     if (input >= (long)INT_MIN && input <= (long)INT_MAX) {
287         value_ = (int)input;
288         std::cout << "success" << std::endl;
289     } else
290         return false;
291 
292     return errno == 0;
293 }
294 template <>
295 inline bool
296 Option<long>::convert_from_string(const std::string s)
297 {
298     errno = 0;
299     value_ = strtol(s.c_str(), NULL, 0);
300     return errno == 0;
301 }
```

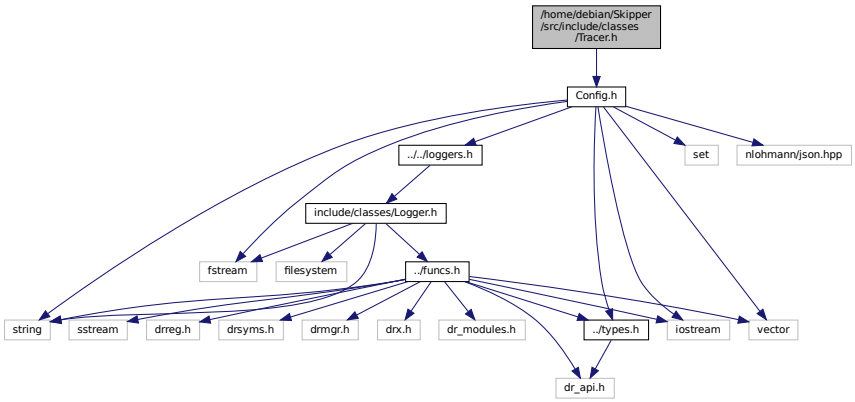
```
302 template <>
303 inline bool
304 Option<long long>::convert_from_string(const std::string s)
305 {
306     errno = 0;
307     // If we set 0 as the base, strtoll() will automatically identify the base.
308     value_ = strtoll(s.c_str(), NULL, 0);
309     return errno == 0;
310 }
311 template <>
312 inline bool
313 Option<unsigned int>::convert_from_string(const std::string s)
314 {
315     errno = 0;
316     long input = strtol(s.c_str(), NULL, 0);
317     // Is the value positive and fits into an unsigned integer?
318     if (input >= 0 && (unsigned long)input <= (unsigned long)UINT_MAX)
319         value_ = (unsigned int)input;
320     else
321         return false;
322 
323     return errno == 0;
324 }
325 template <>
326 inline bool
327 Option<unsigned long>::convert_from_string(const std::string s)
328 {
329     if (is_negative(s))
330         return false;
331 
332     errno = 0;
333     value_ = strtoul(s.c_str(), NULL, 0);
334     return errno == 0;
335 }
336 template <>
337 inline bool
338 Option<unsigned long long>::convert_from_string(const std::string s)
339 {
340     if (is_negative(s))
341         return false;
342 
343     errno = 0;
344     value_ = strtoull(s.c_str(), NULL, 0);
345     return errno == 0;
346 }
347 template <>
348 inline bool
349 Option<double>::convert_from_string(const std::string s)
350 {
351     // strtod will return 0.0 for invalid conversions
352     char *pEnd = NULL;
353     value_ = strtod(s.c_str(), &pEnd);
354     return true;
355 }
356 
357 
358 template <typename T>
359 inline bool
360 Option<T>::option_takes_arg()const
361 {
362     return true;
363 }
364 template <>
365 inline bool
366 Option<bool>::option_takes_arg()const
367 {
368     return false;
369 }
370 
371 
372 #endif // MY_OPTIONS_header
```

5.10 /home/debian/Skipper/src/include/classes/Tracer.h File Reference

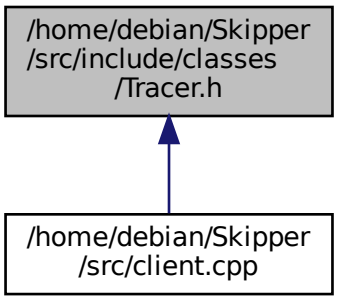
Tracer class description.

```
#include "Config.h"
```

Include dependency graph for Tracer.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct CodeSegmentDescriber

  just pair-structure <code segment start offset, code segment end offset>
- struct TraceArea

  extra-counters section (memory area for extra trace for libfuzzer), contains bounds and size.
- class Tracer

  Class, that inserts tracing function in basic blocks.

Functions

- int **get_msb_ind** (uint x)

  gets most signifant bit of x digit
- int trace_overflow (int *offset_int_ptr, uint32_t size, uint32_t ind, uint32_t reg_id)

trace function for clean-call

- void insert_tracing (void ∗drcontext, void ∗tag, instrlist_t ∗bb, instr_t ∗where, char ∗offset, uint32_t size, uint32_t ind, uint32_t reg_id)

  Function for inline ASM insertion.

5.10.1 Detailed Description

Tracer class description.

Author

Stepan Kafanov

Version

0.1

Date

2025-03-31

Copyright

Copyright (c) 2025

5.10.2 Function Documentation

5.10.2.1 insert_tracing()

```
void insert_tracing (
            void * drcontext,
            void * tag,
            instrlist_t * bb,
            instr_t * where,
            char * offset,
            uint32_t size,
            uint32_t ind,
            uint32_t reg_id )
```

Function for inline ASM insertion.

Inserts ASM instraction directly before tracing ASM-instruction. Functional is the same as in trace_overflow function.

Parameters

drcontext	- DR context
tag	- DR tag
bb	- current basic block
where	- current ASM instruction
offset	- trace-memory address
size	- trace-memory size
ind	- tracing instruction's index

5.10.2.2 trace_overflow()

```
int trace_overflow (
            int * offset_int_ptr,
            uint32_t size,
            uint32_t ind,
            uint32_t reg_id )
```

trace function for clean-call

This function will be called before each instruction client is tracing. It uses buffer of size 65 bytes to trace MSB (most significant bit) of tracing instruction operand and current carry flag from flag register.

Parameters

offset_int_ptr	- trace-memory address
size	- trace-memory size
ind	- tracing instruction's index
reg_id	- register ID in DynamoRIO

Returns

0 - error code

5.11 Tracer.h

Go to the documentation of this file.

```
1
11 #ifndef MY_TRACER_header
12 #define MY_TRACER_header
13
14 #define X86
15
16 #include "Config.h"
17
19 struct CodeSegmentDescriber {
20     size_t start;
21     size_t end;
22 };
23
25 struct TraceArea {
26     size_t start;
27     size_t end;
28     size_t size;
29 };
30
32 int get_msb_ind(uint x) {
33     if (x == 0)
34         return -1; // if there is not any rised bits (==0)
35
36     int msb_index = 0;
37     while (x >= 1) {
38         msb_index++;
39     }
40     return msb_index;
41 }
42
55 int trace_overflow(int* offset_int_ptr, uint32_t size, uint32_t ind, uint32_t reg_id) {
56     char* offset = (char*) offset_int_ptr;
57     reg_id_t dst_reg = (reg_id_t) reg_id;
58     if (size < 65*(ind+1)) {
```

```
59          // TODO : cheking
60          printf("memory is not enough for tracing\n");
61      }
62
63      // restore context
64      dr_mcontext_t mc = { sizeof(mc), DR_MC_ALL};
65      dr_get_mcontext(dr_get_current_drcontext(), &mc);
66
67      // flag register
68      reg_t xflags = mc.xflags;
69      // target register
70      reg_t reg = reg_get_value(dst_reg, &mc);
71
72      // most signigicant bit
73      int msb_ind_reg = get_msb_ind((uint) reg);
74
75      // trace
76      if (msb_ind_reg >= 0) {
77          ((char *)offset)[(ind*65+msb_ind_reg) % size] += 1;
78      }
79      ((char *)offset)[(ind*65+64) % size] += xflags & EFLAGS_CF;
80
81      return 0;
82  }
83
99  void insert_tracing(void *drcontext, void *tag, instrlist_t *bb, instr_t *where,
100                      char* offset, uint32_t size, uint32_t ind, uint32_t reg_id) {
101     auto xax = DR_REG_XAX;
102     auto xbx = DR_REG_XBX;
103     auto xcx = DR_REG_XCX;
104     auto xdx = DR_REG_XDX;
105
106     // save registers and flags
107     dr_save_arith_flags(drcontext, bb, where, SPILL_SLOT_2); // store into xax by default
108     dr_save_reg(drcontext, bb, where, xax, SPILL_SLOT_3);
109     dr_save_reg(drcontext, bb, where, xbx, SPILL_SLOT_4);
110     dr_save_reg(drcontext, bb, where, xcx, SPILL_SLOT_5);
111     dr_save_reg(drcontext, bb, where, xdx, SPILL_SLOT_6);
112
113     instr_t * instr;
114
115     // save interesting reg value into RCX
116     instr = XINST_CREATE_move(
117         drcontext,
118         opnd_create_reg(DR_REG_RCX),
119         opnd_create_reg(reg_id)
120     );
121     instrlist_meta_preinsert(bb, where, instr);
122
123     // ============================================================================
124     // get_msb_ind
125     // RCX - research register
126     // RAX - return result
127     // create labels mgb_loop, msb_finish, msb_ret_minus_1
128
129     instr = INSTR_CREATE_xor(
130         drcontext,
131         opnd_create_reg(DR_REG_RAX),
132         opnd_create_reg(DR_REG_RAX)
133     );
134     instrlist_meta_preinsert(bb, where, instr);
135     instr = INSTR_CREATE_bsr(
136         drcontext,
137         opnd_create_reg(DR_REG_RAX),
138         opnd_create_reg(DR_REG_RCX)
139     );
140     instrlist_meta_preinsert(bb, where, instr);
141     instr = INSTR_CREATE_mov_imm(
142         drcontext,
143         opnd_create_reg(DR_REG_RCX),
144         OPND_CREATE_INT64(0x3f)
145     );
146     instrlist_meta_preinsert(bb, where, instr);
147     instr = INSTR_CREATE_and(
148         drcontext,
149         opnd_create_reg(DR_REG_RAX),
150         opnd_create_reg(DR_REG_RCX)
151     );
152     instrlist_meta_preinsert(bb, where, instr);
153
154
155     // ============================================================================
156
157     instr = XINST_CREATE_move(
158         drcontext,
159         opnd_create_reg(DR_REG_RCX),
160         opnd_create_reg(DR_REG_RAX)
```

```
161     );
162     instrlist_meta_preinsert(bb, where, instr);
163     // answer in RCX
164     // write address into RAX
165     auto pos = ind*65;
166     instr = INSTR_CREATE_mov_imm(
167         drcontext,
168         opnd_create_reg(DR_REG_RAX),
169         OPND_CREATE_INT64(offset+pos)
170     );
171     instrlist_meta_preinsert(bb, where, instr);
172     instr = INSTR_CREATE_add(
173         drcontext,
174         opnd_create_reg(DR_REG_RAX),
175         opnd_create_reg(DR_REG_RCX)
176     );
177     instrlist_meta_preinsert(bb, where, instr);
178
179     instr = INSTR_CREATE_mov_ld(
180         drcontext,
181         opnd_create_reg(DR_REG_DL),
182         OPND_CREATE_MEM8(DR_REG_RAX, 0)
183     );
184     instrlist_meta_preinsert(bb, where, instr);
185     // add 1
186     instr = INSTR_CREATE_inc(
187         drcontext,
188         opnd_create_reg(DR_REG_DL)
189     );
190     instrlist_meta_preinsert(bb, where, instr);
191     // store value
192     instr = XINST_CREATE_store(
193         drcontext,
194         OPND_CREATE_MEM8(DR_REG_RAX, 0),
195         opnd_create_reg(DR_REG_DL)
196     );
197     instrlist_meta_preinsert(bb, where, instr);
198
199     // ========================================================================
200     // occupied RAX for address, RCX for flags, RDX - supportive
201     // read saved flags and store them to reg RCX
202     dr_restore_reg(
203         drcontext, bb, where, (reg_id_t) DR_REG_RCX, SPILL_SLOT_2
204     );
205     instr = INSTR_CREATE_and(
206         drcontext,
207         opnd_create_reg(DR_REG_ECX),
208         OPND_CREATE_INT32((uint32_t) EFLAGS_CF)
209     );
210     instrlist_meta_preinsert(bb, where, instr);
211     // store flag
212     auto i = (ind*65+64) % size;
213     // form address
214     instr = INSTR_CREATE_mov_imm(
215         drcontext,
216         opnd_create_reg(DR_REG_RAX),
217         OPND_CREATE_INT64(offset)
218     );
219     instrlist_meta_preinsert(bb, where, instr);
220     // read byte
221     instr = INSTR_CREATE_mov_ld(
222         drcontext,
223         opnd_create_reg(DR_REG_DL),
224         OPND_CREATE_MEM8(DR_REG_RAX, i)
225     );
226     instrlist_meta_preinsert(bb, where, instr);
227     // plus CF bit
228     instr = INSTR_CREATE_add(
229         drcontext,
230         opnd_create_reg(DR_REG_DL),
231         opnd_create_reg(DR_REG_CL)
232     );
233     instrlist_meta_preinsert(bb, where, instr);
234     // store new value
235     instr = XINST_CREATE_store(
236         drcontext,
237         OPND_CREATE_MEM8(DR_REG_RAX, i),
238         opnd_create_reg(DR_REG_DL)
239     );
240     instrlist_meta_preinsert(bb, where, instr);
241
242     // restore flags and registers
243     dr_restore_reg(drcontext, bb, where, xdx, SPILL_SLOT_6);
244     dr_restore_reg(drcontext, bb, where, xcx, SPILL_SLOT_5);
245     dr_restore_reg(drcontext, bb, where, xbx, SPILL_SLOT_4);
246     dr_restore_reg(drcontext, bb, where, xax, SPILL_SLOT_3);
247     dr_restore_arith_flags(drcontext, bb, where, SPILL_SLOT_2);
```

```
248 }
249
254 class Tracer {
255     TraceArea trace_area;
257     std::map<app_pc, size_t> pc_ind_map;
259     std::map<reg_id_t, size_t> reg_ind_map;
260 public:
261     json tracer_config;
262     Tracer() {}
263
265     bool set_config(Configurator config) {
266         std::cout « "setting config to tracer!" « std::endl;
267         this->tracer_config = config.getTracerConfig();
268         return true;
269     }
270
272     void set_trace_area(size_t start, size_t end) {
273         this->trace_area = {start, end, end-start};
274     }
275
276 protected:
278     int get_reg_id(reg_id_t reg) {
279         auto iter = this->reg_ind_map.find(reg);
280         if (iter != this->reg_ind_map.end()) {
281             return this->reg_ind_map[reg];
282         }
283         this->reg_ind_map[reg] = this->reg_ind_map.size();
284         return this->reg_ind_map[reg];
285     }
286 public:
296     void traceOverflow(void *drcontext, void *tag, instrlist_t *bb, instr_t *instr) {
297         if (!instr_num_dsts(instr)) {
298             return;
299         }
300
301         opnd_t dst = instr_get_dst(instr, 0);
302         if (!opnd_is_reg(dst)) {
303             return;
304         }
305
306         reg_id_t dst_reg = opnd_get_reg(dst);
307         int reg_ind = this->get_reg_id(dst_reg);
308
309         app_pc instr_pc = instr_get_app_pc(instr);
310         // new instruction?  - set index
311         if (this->pc_ind_map.find(instr_pc) == this->pc_ind_map.end()) {
312             this->pc_ind_map[instr_pc] = this->pc_ind_map.size();
313         }
314         size_t ind = this->pc_ind_map[instr_pc];
315
316         auto * module = dr_get_main_module();
317         auto pc = module->start;
318         dr_free_module_data(module);
319         size_t start_size_t = (size_t) this->trace_area.start + (size_t) pc;
320
321         instr_t *nxt = instr_get_next(instr);
322         if (this->tracer_config["use_asm"]) {
323             if (this->tracer_config["debug"]) {
324                 main_logger.log_debug("using inline-asm instrumentation!");
325                 dr_printf("[DEBUG] : using inline-asm instrumentation!\n");
326             }
327
328             insert_tracing(drcontext, tag, bb, nxt,
329                            (char*) start_size_t, this->trace_area.size, ind, dst_reg);
330         } else {
331             if (this->tracer_config["debug"]) {
332                 main_logger.log_debug("using clean_call instrumentation!");
333                 dr_printf("[DEBUG] : using clean_call instrumentation!\n");
334             }
335
336             dr_insert_clean_call_ex(drcontext,
337                                     bb, nxt,
338                                     (void *) trace_overflow,
339                                     (dr_cleancall_save_t) (DR_CLEANCALL_READS_APP_CONTEXT |
    DR_CLEANCALL_MULTIPATH),
340                                     4,
341                                     OPND_CREATE_INTPTR(start_size_t),
342                                     OPND_CREATE_INT32(this->trace_area.size),
343                                     OPND_CREATE_INT32(ind),
344                                     OPND_CREATE_INT32(dst_reg));
345         }
346         if (this->tracer_config["debug"]) {
347             main_logger.log_debug("add number:  {} | add index:  {} | thread id:  {}",
348                                   this->pc_ind_map.size(), ind, get_thread_id());
349             dr_printf("[DEBUG] : add number:  %d | add index:  %d | thread id:  %s\n",
350                       this->pc_ind_map.size(), ind, get_thread_id().c_str());
351         }
```

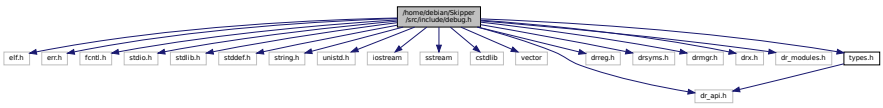
```
352     }
353 };
354
355 #endif // MY_TRACER_header
```

5.12 /home/debian/Skipper/src/include/debug.h File Reference

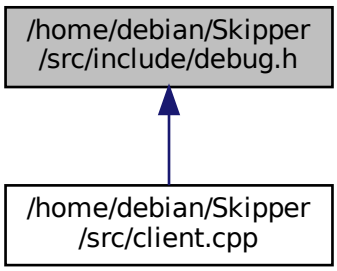
my debug functions for project

```
#include <elf.h>
#include <err.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <string.h>
#include <unistd.h>
#include <iostream>
#include <sstream>
#include <cstdlib>
#include <vector>
#include "dr_api.h"
#include "drreg.h"
#include "drsyms.h"
#include "drmgr.h"
#include "drx.h"
#include "dr_modules.h"
#include "types.h"
```

Include dependency graph for debug.h:



This graph shows which files directly or indirectly include this file:



Functions

- bool print_module_data (module_data_t ∗m)
  prints DR module info
- void **print_all_imported_symbols** ()
  prints all `imported` in DR opinion symbols
- void **print_modules** ()
  prints all visible modules
- void **print_instruction** (void ∗drcontext, instr_t ∗instr)
  prints ASM instruction
- void **print_argv** (int argc, const char ∗argv[ ])
  prints client's argv

5.12.1 Detailed Description

my debug functions for project

Author

Stepan kafanov

Version

0.1

Date

2025-03-31

Copyright

5.12.2 Function Documentation

5.12.2.1 print_module_data()

```
bool print_module_data (
            module_data_t * m )
```

prints DR module info

Parameters

m	- DR module-info pointer

Returns

whether `m` is not `NULL`

5.13 debug.h

Go to the documentation of this file.

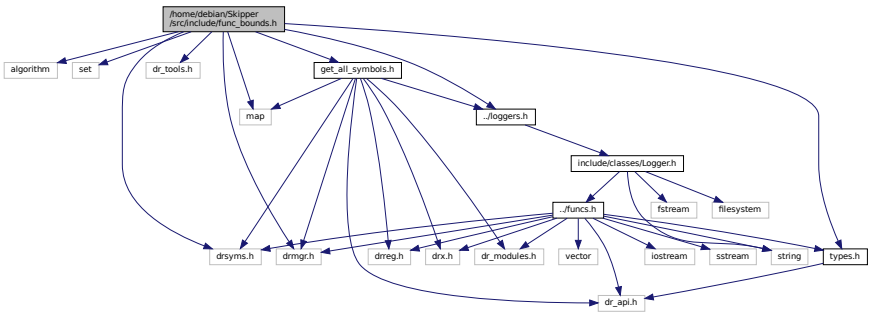
```
1
11 #ifndef MY_DEBUG_header
12 #define MY_DEBUG_header
13
14 #include <elf.h>
15 #include <err.h>
16 #include <fcntl.h>
17 #include <stdio.h>
18 #include <stdlib.h>
19 #include <stddef.h>
20 #include <string.h>
21 #include <unistd.h>
22 #include <iostream>
23 #include <sstream>
24 #include <cstdlib>
25 #include <vector>
26
27 #include "dr_api.h"
28 #include "drreg.h"
29 #include "drsyms.h"
30 #include "drmgr.h"
31 #include "drx.h"
32 #include "dr_modules.h"
33
34 #include "types.h"
35
39 bool print_module_data(module_data_t * m) {
40     if (m == NULL) {
41         return false;
42     }
43     bool err = printf(
44             "end                    :      %p\n"
45             "entry_point            :      %p\n"
46             "flags                  :      %u\n"
47             "name                   :      %s\n"
48             "full_path              :      %s\n"
49             // "file_version           :      %u"
50             // "product_version        :      %u"
51             // "checksum               :      %u"
52             "timestamp              :      %u\n"
53             // "module_internal_size   :      %lld"
54             "preferred_base         :      %p\n"
55             "start                  :      %p\n",
56             m->end,
57             m->entry_point,
58             m->flags,
59             dr_module_preferred_name(m),
60             m->full_path,
61             // m->file_version.version,
62             // m->product_version.version,
63             // m->checksum,
64             m->timestamp,
65             // m->module_internal_size,
66             m->start);
67     return true;
68 }
69
71 void print_all_imported_symbols() {
72     drsym_init(NULL);
73
74     auto modules = get_all_modules();
75     for (auto module_info :  modules) {
76         auto module_name = module_info.name;
77         dr_printf("module_name:  %s\n", module_name.c_str());
78         auto module = dr_lookup_module_by_name(module_name.c_str());
79         // dr_get_proc_address(module->handle, "New_G1");
80
81         auto iterator_im = dr_symbol_import_iterator_start(module->handle, NULL);
82         do {
83             auto * symbol = dr_symbol_import_iterator_next(iterator_im);
84             dr_printf("symbol:  %s\n", symbol->name);
85         } while (dr_symbol_import_iterator_hasnext(iterator_im));
86         dr_symbol_import_iterator_stop(iterator_im);
87     }
```

```
88
89     drsym_exit();
90 }
91
93 void print_modules() {
94     auto modules = get_all_modules();
95     dr_printf("modules:\n");
96     for (auto module :  modules) {
97         dr_printf("\tmodule_name:  %s; module_path:  %s\n", module.name.c_str(), module.path.c_str());
98     }
99 }
100
102 void print_instruction(void *drcontext, instr_t *instr) {
103     char instr_str[256];
104     instr_disassemble_to_buffer(drcontext, instr, instr_str, sizeof(instr_str));
105     dr_printf("Instruction:  %s\n", instr_str);
106 }
107
109 void print_argv(int argc, const char *argv[]) {
110     dr_printf("command line args:\n");
111     for (int i=0; i < argc; ++i) {
112         dr_printf("\t%d:  %s\n", i, argv[i]);
113     }
114 }
115
116 #endif // MY_DEBUG_header
```

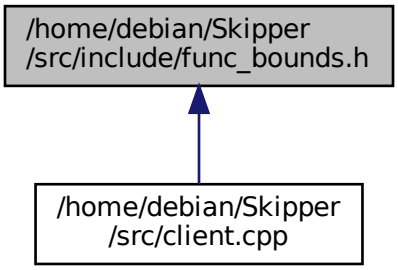
5.14 /home/debian/Skipper/src/include/func_bounds.h File Reference

```
#include <algorithm>
#include <set>
#include <map>
#include "dr_tools.h"
#include "drsyms.h"
#include "drmgr.h"
#include "get_all_symbols.h"
#include "types.h"
#include "../loggers.h"
```

Include dependency graph for func_bounds.h:



This graph shows which files directly or indirectly include this file:



Functions

- bool **get_func_bounds_callback** (drsym_info_t *info, drsym_error_t status, void *data)

  depricated
- std::map< std::string, std::pair< generic_func_t, generic_func_t > > get_func_bounds (std::map< std::string, FuncConfig > inspect_funcs, bool use_pattern, bool use_default_bounds)

  Get bounds for passed functions.

5.14.1 Detailed Description

Author

Stepan Kafanov

Version

0.1

Date

2025-03-31

Copyright

Copyright (c) 2025

5.14.2 Function Documentation

5.14.2.1 get_func_bounds()

```
std::map< std::string, std::pair< generic_func_t, generic_func_t > > get_func_bounds (
            std::map< std::string, FuncConfig > inspect_funcs,
            bool use_pattern,
            bool use_default_bounds )
```

Get bounds for passed functions.

Algorithm of search:

- collect all symbols and their addresses (offsets) from modules, poited in functions-under-test info
- sort all symbols with their offsets
- for every function-under-test name searching for coincidence
  - if there is not exact coincidence and `use_pattern = true` will search pattern coincidence
  - if failure and `use_default_bounds = true` and default-address is correct will take default-address as function bounds
  - else will ask user to enter bounds manualy
- return resulting dict with function bounds

Parameters

inspect_funcs	- list of inspecting functions (functions, bounds for we are searching)
use_pattern	- shall we try to search for not exact coincidence
use_default_bounds	- shall we use default functions bounds in case of search failure

Returns

std::map<std::string, std::pair<generic_func_t, generic_func_t>> - dict {'fucntion name' : (function-start-offset, function-end-offset)}

5.15 func_bounds.h

Go to the documentation of this file.

```
1
11 #ifndef FIND_FUNC_BOUNDS_header
12 #define FIND_FUNC_BOUNDS_header
13
14 #include <algorithm>
15 #include <set>
16 #include <map>
17
18 #include "dr_tools.h"
19 #include "drsyms.h"
20 #include "drmgr.h"
21 #include "get_all_symbols.h"
22
23 #include "types.h"
24 #include "../loggers.h"
25
26
28 bool get_func_bounds_callback(drsym_info_t *info, drsym_error_t status, void *data) {
29     if (info != NULL && data != NULL) {
30         // if (status != DRSYM_SUCCESS) {
31         //     ("line shit is not success!\n");
32         // }
```

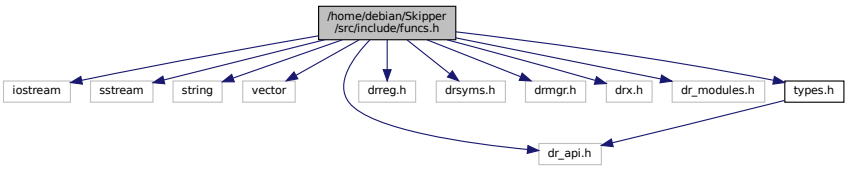
```
33          auto * d = (std::vector<std::pair<size_t, std::string> *) data;
34          d->push_back(std::make_pair(info->start_offs, info->name));
35      }
36      return true;
37  }
38
56  std::map<std::string, std::pair<generic_func_t, generic_func_t>
57  get_func_bounds(std::map<std::string, FuncConfig> inspect_funcs, bool use_pattern, bool
        use_default_bounds)
58  {
59      if (inspect_funcs.empty()) {
60          main_logger.log_error("empty instr function map!");
61          dr_printf("[ERROR] : empty instr function map!\n");
62          throw std::invalid_argument("[ERROR] : empty instr function map!");
63      }
64
65      std::set<std::pair<std::string, std::string> module_path;
66      for (auto & func :  inspect_funcs) {
67          module_path.insert(std::make_pair(func.second.module_name, func.second.module_path));
68      }
69
70      // with module offsets
71      std::map<std::string, generic_func_t> symbols;
72      for (auto & m_p :  module_path) {
73          auto symbols_offests = get_all_symbols_with_offsets(m_p.first, m_p.second, use_pattern);
74          symbols.merge(symbols_offests);
75      }
76
77      std::vector <std::pair<size_t, std::string> symbols_vector;
78      for (auto & symbol :  symbols) {
79          symbols_vector.push_back({(size_t) symbol.second, symbol.first});
80      }
81      std::sort(symbols_vector.begin(), symbols_vector.end());
82
83      std::map<std::string, std::pair<generic_func_t, generic_func_t> res;
84      for (auto & func :  inspect_funcs) {
85          std::string func_name = func.first;
86
87          auto iter = std::find_if(symbols_vector.begin(), symbols_vector.end(), [&func_name](const auto
        x){
88              return func_name == std::string(x.second);
89          });
90
91          if ((iter == symbols_vector.end()) && use_pattern) {
92              main_logger.log_debug("second try...");
93              iter = std::find_if(
94                  symbols_vector.begin(), symbols_vector.end(),
95                  [&func_name](const auto x){
96                      return std::string(x.second).find(func_name) != std::string::npos;
97                  });
98          }
99          main_logger.log_debug("searching complete!");
100
101          if (iter == symbols_vector.end()) {
102              main_logger.log_debug("cannot find such func_name =(");
103              char message[1024];
104              main_logger.log_debug("message:  there is not func name <{}> here", func_name);
105
106              std::string answer;
107              size_t addr = 0;
108              auto default_address = func.second.default_address;
109              if (default_address.first && default_address.first <= default_address.second) {
110                  if (use_default_bounds) {
111                      res[func_name] = std::make_pair((generic_func_t) default_address.first,
112                                                      (generic_func_t) default_address.second);
113                      continue;
114                  } else {
115                      main_logger.log("CONTROLE", "do you want to use default_address?[y/n] ");
116                      dr_printf("[CONTROLE] : do you want to use default_address?[y/n] ");
117                      std::cin » answer;
118                      main_logger.log("CONTROLE", "user answer:  ", answer);
119                      if (answer == "y" || answer == "yes") {
120                          res[func_name] = std::make_pair(
121                                                          (generic_func_t) default_address.first,
122                                                          (generic_func_t) default_address.second);
123                          continue;
124                      }
125                  }
126              }
127              {
128                  main_logger.log("CONTROLE", "do you want to enter address?[y/n] ");
129                  dr_printf("[CONTROLE] : do you want to enter address?[y/n] ");
130                  std::cin » answer;
131                  main_logger.log("CONTROLE", "user answer:  ", answer);
132                  if (answer == "n" || answer == "no") {
133                      res[func_name] = std::make_pair((generic_func_t)0, (generic_func_t)0);
134                      continue;
```

```
135                 }
136                 size_t start{}, stop{};
137                 main_logger.log("CONTROLE", "enter start address:  ");
138                 dr_printf("[CONTROLE] : enter start address:  ");
139                 std::cin » start;
140                 main_logger.log("CONTROLE", "user answer:  ", start);
141                 main_logger.log("CONTROLE", "enter stop address:  ");
142                 dr_printf("[CONTROLE] : enter stop address:  ");
143                 std::cin » stop;
144                 main_logger.log("CONTROLE", "user answer:  ", stop);
145                 res[func_name] = std::make_pair((generic_func_t)start, (generic_func_t)stop);
146                 continue;
147             }
148         }
149 
150         if (iter + 1 != symbols_vector.end()) {
151             main_logger.log_debug("find complete!\nnext_name:  {}", (iter+1)->second);
152             main_logger.log_debug("segment:  {} - {}", iter->first, (iter+1)->first);
153             res[func_name] = std::make_pair((generic_func_t)iter->first,
154                                             (generic_func_t)(iter+1)->first);
155         }
156     }
157 
158     return res;
159 }
160 
161 #endif // FIND_FUNC_BOUNDS_header
```

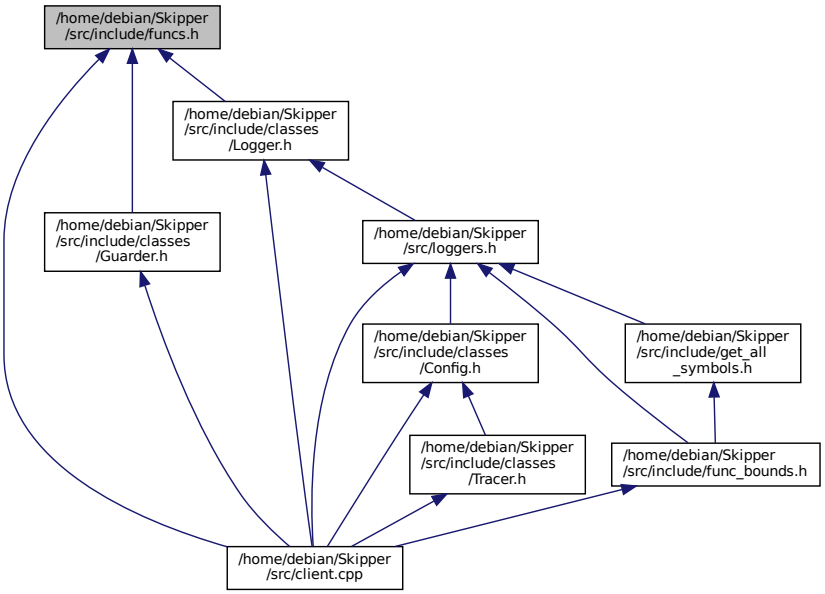
5.16 /home/debian/Skipper/src/include/funcs.h File Reference

```
#include <iostream>
#include <sstream>
#include <string>
#include <vector>
#include "dr_api.h"
#include "drreg.h"
#include "drsyms.h"
#include "drmgr.h"
#include "drx.h"
#include "dr_modules.h"
#include "types.h"
```

Include dependency graph for funcs.h:



This graph shows which files directly or indirectly include this file:



Functions

- size_t get_symbol_offset (std::string module_name, std::string module_path, std::string symbol_name)

  Get the symbol offset.
- std::vector< ModuleInfo > get_all_modules ()

  Get the all pairs <module_name, module_path>. Looks for a DynamoRIO-seen modules.
- std::vector< std::string > get_modules_names ()

  Get all modules names, which DynamoRIO can see.
- std::string int_to_hex (int my_integer)

  Converts int numbers to hex form.
- std::string get_thread_id ()

  Get current thread id for printing.

5.16.1 Function Documentation

5.16.1.1 get_all_modules()

```
std::vector< ModuleInfo > get_all_modules ( )
```

Get the all pairs <module_name, module_path>. Looks for a DynamoRIO-seen modules.

Returns

std::vector<ModuleInfo> list of module-pairs

5.16.1.2 get_modules_names()

```
std::vector< std::string > get_modules_names ( )
```

Get all modules names, which DynamoRIO can see.

Returns

std::vector <std::string>

5.16.1.3 get_symbol_offset()

```
size_t get_symbol_offset (
            std::string module_name,
            std::string module_path,
            std::string symbol_name )
```

Get the symbol offset.

Parameters

module_name	name of the module, where symbol contains
module_path	path to the module
symbol_name	name of the seeked symbol

Returns

size_t - offset of the symbol

5.16.1.4 get_thread_id()

```
std::string get_thread_id ( )
```

Get current thread id for printing.

Returns

std::string - return str-format current thread id

5.16.1.5 int_to_hex()

```
std::string int_to_hex (
            int my_integer )
```

Converts int numbers to hex form.

Parameters

my_integer	- converted number

Returns

std::string - hex form of number

5.17 funcs.h

Go to the documentation of this file.

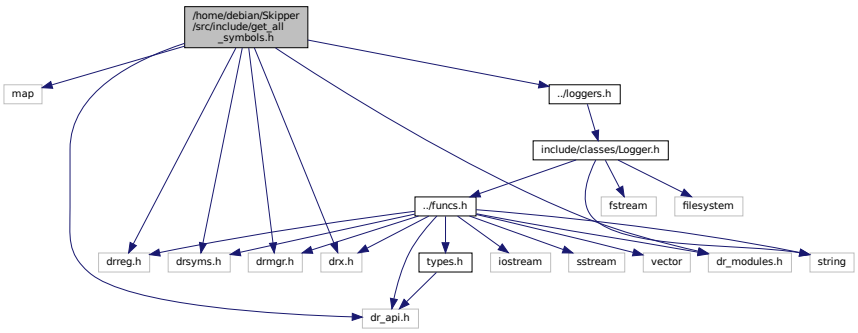
```
1
4 #ifndef FUNCS_DR_header
5 #define FUNCS_DR_header
6
7 #include <iostream>
8 #include <sstream>
9 #include <string>
10 #include <vector>
11
12 #include "dr_api.h"
13 #include "drreg.h"
14 #include "drsyms.h"
15 #include "drmgr.h"
16 #include "drx.h"
17 #include "dr_modules.h"
18
19 #include "types.h"
20
29 size_t get_symbol_offset(std::string module_name, std::string module_path, std::string symbol_name) {
30     drsym_init(NULL);
31     drsym_error_t error;
32     drsym_debug_kind_t kind;
33
34     error = drsym_get_module_debug_kind(module_path.c_str(), &kind);
35     if (error != DRSYM_SUCCESS) {
36         // main_logger.log_error("error in drsym_get_module_debug_kind() :  get_symbol_offset :  kind
      error");
37         dr_fprintf(STDERR, "ERROR: error in drsym_get_module_debug_kind() :  get_symbol_offset\n");
38         return 0;
39     } else {
40         // printf("kind:  %d\n", kind);
41     }
42
43     size_t offset = 0;
44     error = drsym_lookup_symbol(module_path.c_str(),
45                                 symbol_name.c_str(),
46                                 &offset,
47                                 DRSYM_DEMANGLE_FULL);
48     if (error != DRSYM_SUCCESS) {
49         // main_logger.log_error("error in drsym_lookup_symbol() :  get_symbol_offset");
50         dr_fprintf(STDERR, "ERROR: error in drsym_lookup_symbol() :  get_symbol_offset\n");
51         return 0;
52     } else {
53         // printf("offset:  %d\n", offset);
54     }
55
56     drsym_exit();
57
58     return offset;
59 }
60
67 std::vector<ModuleInfo> get_all_modules() {
68     auto iterator = dr_module_iterator_start();
69     std::vector<ModuleInfo> modules;
70     while (dr_module_iterator_hasnext(iterator)) {
71         auto * module = dr_module_iterator_next(iterator);
72         modules.push_back({dr_module_preferred_name(module), module->full_path});
73         dr_free_module_data(module);
74     }
75     dr_module_iterator_stop(iterator);
76     return modules;
77 }
78
84 std::vector <std::string> get_modules_names() {
85     auto modules = get_all_modules();
```

```
86      std::vector <std::string> modules_names;
87      for (auto module :  modules) {
88          auto mn = module.name;
89          modules_names.push_back(mn);
90      }
91      return modules_names;
92  }
93
100 std::string int_to_hex(int my_integer) {
101     std::stringstream sstream;
102     sstream « std::hex « my_integer;
103     std::string result = sstream.str();
104     return result;
105 }
106
112 std::string get_thread_id() {
113     std::stringstream ss;
114     ss « std::this_thread::get_id();
115     return ss.str();
116 }
117
118 #endif // FUNCS_DR_header
```

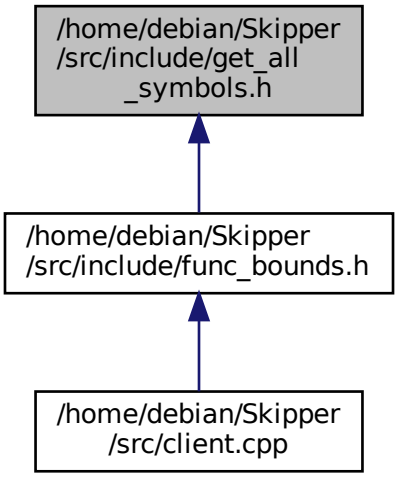
5.18 /home/debian/Skipper/src/include/get_all_symbols.h File Reference

```
#include <map>
#include "dr_api.h"
#include "drreg.h"
#include "drsyms.h"
#include "drmgr.h"
#include "drx.h"
#include "dr_modules.h"
#include "../loggers.h"
```

Include dependency graph for get_all_symbols.h:



This graph shows which files directly or indirectly include this file:



Functions

- bool get_all_symbols_with_offsets_callback (drsym_info_t ∗info, drsym_error_t status, void ∗data)

  just callback for get_all_symbols function
- std::map< std::string, generic_func_t > get_all_symbols_with_offsets (std::string module_name, std::string module_path)

  Get the all symbols with offsets object.

5.18.1 Detailed Description

Author

Stepan Kafanov

Version

0.1

Date

2025-03-31

Copyright

Copyright (c) 2025

5.18.2 Function Documentation

5.18.2.1 get_all_symbols_with_offsets()

```
std::map< std::string, generic_func_t > get_all_symbols_with_offsets (
            std::string module_name,
            std::string module_path )
```

Get the all symbols with offsets object.

Parameters

module_name	- module to look for symbols
module_path	- path to that module

Returns

std::map<std::string, generic_func_t> - dict {'symbol_name' : symbol_offset}

5.18.2.2 get_all_symbols_with_offsets_callback()

```
bool get_all_symbols_with_offsets_callback (
            drsym_info_t * info,
            drsym_error_t status,
            void * data )
```

just callback for get_all_symbols function

Parameters

info	- DR symbol info
status	- DR look through status
data	- Extra data, passsed to callback. Callback stores info about found symbols there.

Returns

always "true" to continue looking for new symbols

5.19 get_all_symbols.h

Go to the documentation of this file.

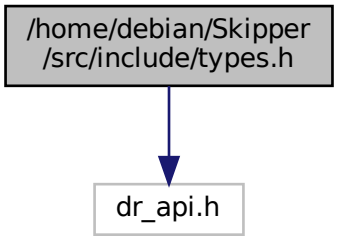
```
1
11 #ifndef GET_ALL_SYMBOLS_header
12 #define GET_ALL_SYMBOLS_header
```

```
13
14 #include <map>
15
16 #include "dr_api.h"
17 #include "drreg.h"
18 #include "drsyms.h"
19 #include "drmgr.h"
20 #include "drx.h"
21 #include "dr_modules.h"
22
23 #include "../loggers.h"
24
33 bool get_all_symbols_with_offsets_callback(
34     drsym_info_t *info,
35     drsym_error_t status,
36     void *data)
37 {
38     auto d = (std::map<std::string, generic_func_t> *)data;
39     if (info->name) {
40         (*d)[info->name] = (generic_func_t)info->start_offs;
41     }
42     return true;
43 }
44
52 std::map<std::string, generic_func_t>
53 get_all_symbols_with_offsets(
54     std::string module_name,
55     std::string module_path)
56 {
57     if (module_name.empty() || module_path.empty()) {
58         main_logger.log_error("should not be any empty args have been passed in
    <get_all_symbols_with_offsets> function!");
59         dr_fprintf(STDERR, "should not be any empty args have been passed in
    <get_all_symbols_with_offsets> function!\n");
60         throw std::runtime_error("empty arg has been passed!");
61     }
62
63     module_data_t *module = dr_lookup_module_by_name(module_name.c_str());
64     if (module == NULL) {
65         main_logger.log_error("cannot load module with name \"{}\" :  get_all_symbols", module_name);
66         dr_fprintf(STDERR, "cannot load module with name \"%s\" :  get_all_symbols\n",
    module_name.c_str());
67         throw std::runtime_error("cannot load module");
68     } else {
69         main_logger.log_info("module_path:  {}", module->full_path);
70     }
71
72     if (drsym_init(NULL) != DRSYM_SUCCESS) {
73         dr_printf("init dr_sym error.  exception throwen\n");
74         throw std::runtime_error("cannot init dr_mgr");
75     }
76     drsym_error_t error;
77     drsym_debug_kind_t kind;
78
79     error = drsym_get_module_debug_kind(module_path.c_str(), &kind);
80     if (error != DRSYM_SUCCESS) {
81         perror("error in drsym_get_module_debug_kind() :  get_all_symbols\n");
82         fprintf(stderr, "ERROR: %d\n", error);
83         throw std::runtime_error("[EXEPTION]: error in drsym_get_module_debug_kind() :
    get_all_symbols");
84     }
85     else {
86         // printf("kind:  %d\n", kind);
87     }
88     size_t base = (size_t)module->start;
89
90     // ################### job start ####################
91
92     // all pairs <symbol_name, offset>
93     std::map<std::string, generic_func_t> data;
94     error = drsym_enumerate_symbols_ex(module_path.c_str(),
95                                        get_all_symbols_with_offsets_callback,
96                                        sizeof(drsym_info_t),
97                                        &data,
98                                        DRSYM_DEMANGLE_FULL);
99
100     // + module offset
101     for (auto &symbol :  data) {
102         symbol.second = (generic_func_t)((size_t)symbol.second + base);
103     }
104
105     drsym_exit();
106     dr_free_module_data(module);
107
108     return data;
109 }
110
```

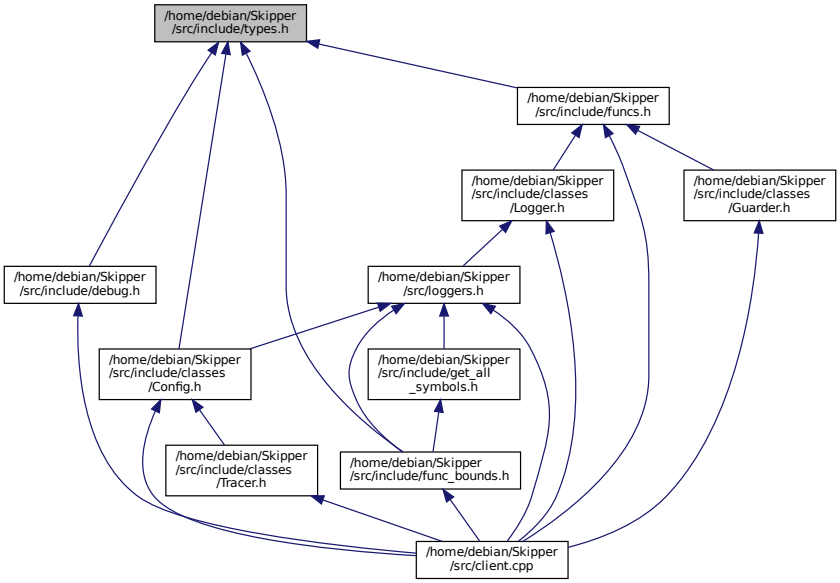
```
111 #endif // GET_ALL_SYMBOLS_header
```

5.20 /home/debian/Skipper/src/include/types.h File Reference

`#include "dr_api.h"`
Include dependency graph for types.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct sym_info_t

  Contains all aviable info about some symbol.
- struct ModuleInfo

Just pair <module name, module path> struct.

- struct FuncConfig

  Contains function-under-test info.

- struct thread_data

  depricated

5.21 types.h

Go to the documentation of this file.

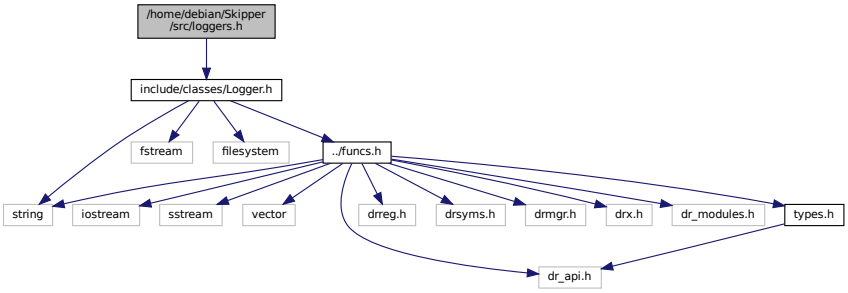
```
1
4 #ifndef MY_TYPES_header
5 #define MY_TYPES_header
6
7 #include "dr_api.h"
8
13 struct sym_info_t {
14     std::string file;
15
16     uint64 line;
17     size_t line_offs;
18     size_t start_offs;
19     size_t end_offs;
20
21     int debug_kind;
22
23     std::string name;
24     uint type_id = 0;
25     uint flags;
26
27     bool ex = false;
28     bool imp = true;
29     size_t moffs = 0;
30
31     sym_info_t(const char * name, size_t off, bool imp = false) {
32         if (name == NULL) {
33             dr_fprintf(STDERR, "NULL name in sym_info_t constructor\n");
34             throw std::runtime_error("NULL name in sym_info_t constructor");
35         }
36         this->ex = false;
37         this->imp = imp;
38         this->name = std::string(name);
39         this->moffs = off;
40     }
41     sym_info_t(const drsym_info_t * info, bool imp=false) {
42         if (!info) {
43             dr_fprintf(STDERR, "NULL drsym_info_t pointer in constructor!\n");
44             throw std::runtime_error("NULL drsym_info_t pointer in constructor");
45         }
46         this->ex = true;
47         this->imp = imp;
48         this->name = std::string(info->name, info->name_size);
49         this->type_id = info->type_id;
50         this->flags = info->flags;
51
52         this->file = std::string(info->file, info->file_size);
53
54         this->line = info->line;
55         this->line_offs = info->line_offs;
56         this->start_offs = info->start_offs;
57         this->end_offs = info->end_offs;
58     }
59
60     ~sym_info_t() {}
61
62     sym_info_t(const sym_info_t & sym) {
63         this->name = sym.name;
64         this->type_id = sym.type_id;
65         this->flags = sym.flags;
66
67         this->file = sym.file;
68
69         this->line = sym.line;
70         this->line_offs = sym.line_offs;
71         this->start_offs = sym.start_offs;
72         this->end_offs = sym.end_offs;
73
74         this->ex = sym.ex;
75         this->imp = sym.imp;
```

```
76     }
77 };
78
83 struct ModuleInfo {
84     std::string name;
85     std::string path;
86 };
87
92 struct FuncConfig {
94     std::string module_name;
96     std::string module_path;
98     std::pair<size_t, size_t> default_address;
99 };
100
102 static const size_t MAP_SIZE = 1025;
104 struct thread_data {
105     uint64_t location;
106     uint8_t map[MAP_SIZE];
107 };
108
109 #endif // MY_TYPES_header
```

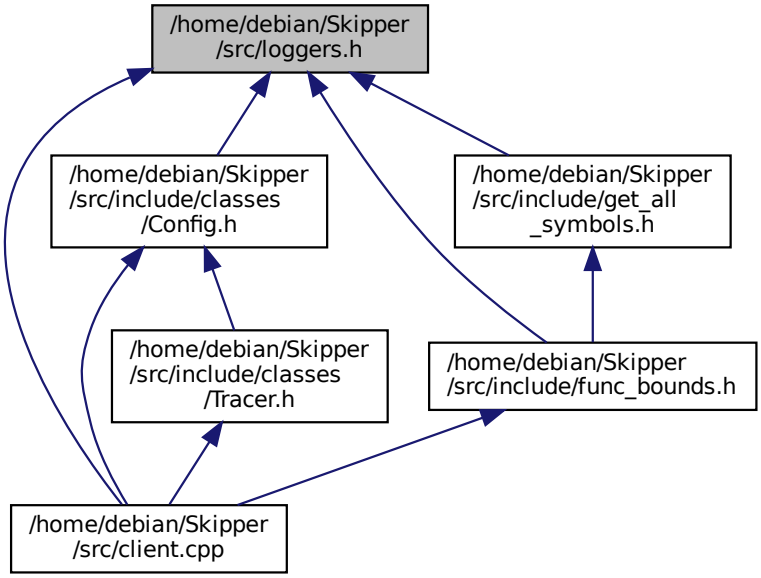
5.22 /home/debian/Skipper/src/loggers.h File Reference

`#include "include/classes/Logger.h"`

Include dependency graph for loggers.h:



This graph shows which files directly or indirectly include this file:



5.23 loggers.h

Go to the documentation of this file.

```
1
4 #ifndef MY_STATIC_LOGGERS_header
5 #define MY_STATIC_LOGGERS_header
6
7 #include "include/classes/Logger.h"
8
10 static Logger main_logger;
11
12 #endif // MY_STATIC_LOGGERS_header
```

Index