riscure

True Code

Frequently Asked Questions

Version	2021.4
Date	1 October 2021

Frequently Asked Questions

Developed by

Riscure BV

Delftechpark 49, 2628 XJ Delft, The Netherlands
Phone: +31 15 251 40 90, Fax: +31 15 251 40 99
Email: inforequest@riscure.com
Web: www.riscure.com

Contents

1 Frequently Asked Questions (FAQ)

1.1 General Questions

1.1.1 How many parallel True Code instances can be opened at the same time?

Multiple instances of True Code can run in parallel, read from and write to the same annotation database. For this situation, you must store the annotation database (SQLite-file) locally at your machine or use PostgreSQL.

True Code may require quite some CPU power and memory, so multiple running instances of True Code may not lead to an overall performance increase.

It is not advised to run the same code checker on multiple instances. This may lead to duplication of code checker findings.

1.2 Compilation Configuration

1.2.1 Why do I need to configure True Code for compiling my code base?

True Code embeds a set of code checkers and a Fault Injection Simulation for performing the analysis of your code base. These technologies *will not analyze your source code* directly, and depending on the feature you would like to use, True Code will use different techniques for performing the analysis.

In the case of the Code Checkers, True Code will create an *intermediate representation* of the code to perform the analysis. True Code relies on clang to generate LLVM IR as the intermediate representation. The list of supported target platforms (chips) is limited; please check the pull down list in True Code at 'Workspace settings' | 'Target' | 'Architecture'. Being this said, it is imperative to create a proper compilation database, so True Code can create the intermediate representation when analyzing your code. To this end, True Code will help you to create a compilation database using the Compilation Wizard. Please note that the

In the case of the Fault Injection Simulator, True Code will use an RISC-V or ARM compiler to create a binary of a stubbed file containing the function(s) you want to simulate. By default, True Code will use the RISC-V compiler, creating a binary using the RV32I instruction set.

1.2.2 Can True Code use my own compiler to create an Intermediate Representation of my source code and run the Code Checkers using my proprietary compiler?

True Code uses LLVM IR to run the code checkers. The True Code LLVM IR parser relies on a specific version of LLVM IR. For this reason, you need to use the Clang/LLVM which is delivered together with True Code.

1.2.3 How do I know if I need to click on “clear host platform macros”?

Usually, code bases that need to be *cross-compiled* need to use this feature. Since a clear answer is highly dependent on the source code you are trying to import, we advise you to get support from a developer from your team who is familiar with the compilation of your code base.

1.2.4 What is the “predefined macros file”? How do I supply this file to True Code? What information needs to be in this file?

True Code relies on the *clang* compiler to create an *Intermediate Representation* of your source code. To do this, *clang* needs to correctly set the configuration corresponding to the target platform. This configuration is supplied in the “Target” stage of the “Compiler Settings” window. To reveal this option, you need to click on the “Show optional fields” in the bottom of that window. In the case you are compiling your code base to run on another platform (target platform), it is likely that you need to adjust these configurations to match what it is expected on the target.

The *predefined macros* file is a file that includes C-like definitions. In general, these definitions include (but are not limited to) the size of integers (char, int, long), size of pointers (void *) among others. To get a clear idea about what are de definitions that need to be set, please get the help from a developer from your team who is familiar with the compilation of your code base.

1.2.5 What are the "additional flags" and how do I know if I need to set them?

In the "Target" stage of the "Compiler Settings" window, and as an optional field, it is possible to set *additional flags* when configuring the target platform. For some platforms, it is required to specify some extra configurations to clang, in form of additional options that are set in the compiler. You can use this option to correctly set these options and make your source code compile. If you are in doubt whether to set these additional flags or not, please get help from a developer from your team who is familiar with the compilation of your code base.

1.2.6 In the "Database" stage of the "Compiler Settings" window, what are the difference between "import", "generate" and "parse build log" options?

To create the compilation database, True Code can use three strategies:

- Import: If your codebase uses a CMake compilation system, you can directly import the compilation database created by CMake. Use this option only if your codebase uses CMake and you have the compilation database (JSON file). If your codebase uses CMake but do not have a compilation database, please check the CMake documentation to create it.
- Generate: With this option, True Code will try to discover your source files (according to the configuration) and create compilation database for them. Please note that this process is based on "best effort", and does not

guarantee that the generated compilation database will be complete or accurate.

- Parse build log: If your codebase uses a Make/IAR compilation system, True Code can process a compilation build log created by your compilation system and create a compilation database with it. This option requires a “verbose” build log. Please refer to the documentation of your build system to get information about how to generate such verbose build log.

If you are unsure about which option you need to choose, please get help from a developer from your team who is familiar with the compilation of your code base.

1.2.7 In the “Database” stage of the “Compiler Settings” window, what is the difference between the “Imported” / “Parsed” and the “Fixed” compilation databases?

When selecting the “Import” or “Parse build log” options, True Code will go through two stages to complete the import or parsing.

The first stage is the import or parsing, and it will simply read the given JSON file or parse the lines in your build log to create entries on a JSON compilation database (respectively). The file to be read or the parsed compilation database is referred as the “Imported” or “Parsed” compilation database (respectively) in this first stage.

In the second stage, True Code will try to perform an automatic fix of the “imported” or “parsed” database. During this fix, True Code will try fixing the paths referred in the “imported” or “parsed” database, and try to match the files with the ones present in your codebase folder. The result of this fixing stage is referred as the “fixed” database.

1.2.8 In the “Database” stage of the “Compiler Settings” window, what are the “String replacements”?

When parsing a build log, True Code will analyze line by line the file you feed as the build log. In the case you want to automatically perform some “search and replace” actions on the lines of your build log, you can use the “String replacements” to define the text to find and the text to replace.

1.2.9 In the “Database” stage of the “Compiler Settings” window, what are the “Unsupported arguments”?

Some proprietary compilers have options that *clang* does not support, which will make *clang* to fail when invoked by True Code. If this is the case, you can use the “Unsupported arguments” option to define a list of compiler arguments that are known to be incompatible with *clang.*

When running the “Fix” or “Diagnostics” stages later in the “Compiler settings”, it is possible that some files do not compile with an *“unsupported argument”* error (check the “Logging” tab to search for these errors). If this is the case, you can add the reported unsupported argument to this configuration and try again.

If you are in doubt about which is the list of known unsupported arguments, please get help from a developer from your team who is familiar with the compilation of your code base.

1.2.10 In the “Compiler Settings” window, the “Fix” and “Diagnostics” stage seem to perform the same analysis. Is this normal?

True Code has been designed to provide you with the most help possible to create a valid compilation database. During the “Fix” stage, True Code tries to fix the compilation lines and will try to invoke the compiler to check if the fixed line is correct. This process does not guarantee that the fixed line is correct.

In the next stage, the “Diagnostics” will then use the fixed compilation database and try to compile the source files. If the file can be compiled correctly, it is completely guaranteed that the fixed compilation line is correct, and can be used by True Code to create the intermediate representation of your code.

Despite its similar behavior, please note that the “Fix” stage is mandatory when using the “import” and “parse build log” modes of the wizard. Additionally, in all the three modes (“import”, “generate” and “parse build log”) the “Diagnostics” stage is mandatory, ensuring that True Code can use the compilation database.

1.2.11 In the “Parse” stage of the “Compiler settings” window, what is the “Compiler type” configuration? How do I know which one I have to select?

Your code base uses a compilation system to create the binaries. This compilation system invokes the compiler to do so. Depending on the type of compiler, the format of the compilation line might vary. True Code needs to be aware of this, so it can correctly parse the arguments and values to create a compilation database. To know which the correct type is, please get help from a developer from your team who is familiar with the compilation of your code base.

1.2.12 In the “Parse” stage of the “Compiler settings” window, what is the “Compiler string” configuration?

To parse your build log, True Code will analyze each line of the build log and parse the arguments. To correctly identify the invocation to the compiler (and parse its options), True Code needs to know which is the text that call your compiler. For example, in the following line:

```
arm-linux-gnueabihf-gcc –march=m32 –DSOME_MACRO=1 –Isystem .. –c
file.c –o file.o
```

“arm-linux-gnueabihf-gcc” is the compiler string, and is this text (without the quotations) that needs to be configured in True Code.

Please note that if your code base uses more than one compiler, you can add multiple texts that True Code will match to parse the compilation lines.

If you are unsure about which is the compiler string in your compilation line, please get help from a developer from your team who is familiar with the compilation of your code base.

1.2.13 In the “Compile settings” wizard, what is the difference between the “Generate” and the “Diagnostics” step?

The “Generate” stage is only shown when you instruct True Code to create a compilation database. This stage will scan your code base, discover all the files that contain source code, and try to use generic compilation flags to obtain an intermediate representation file. Since most of the code bases use particular compilation flags, it is likely that these compilation flags will not be enough to compile all your source code. In any case, the “Generate” stage will try its best to create a valid compilation database.

On the contrary, the “Diagnostics” will make use of an already-created (and fixed) compilation database, and use all of the entries to compile your source codes. In other words, in this stage True Code will make sure that the codebase that you provided (either by importing, parsing or creating) can be used to create the intermediate representation and, thus, useful to run the code checkers on your code base.

1.3 Code Indexing and Context Creation

1.3.1 What is "Code base indexing"? Why do I need to index my code base?

Before being able to analyze your code base, True Code needs to create an index with your functions definitions. True Code needs this information, and it will use it to know where to start the analyses for both the code checkers and the Fault Injection Simulators.

1.3.2 What is "Context creation"? Why do I need to create contexts?

For using the Code Checkers, True Code also needs to know the functions each code checker need to analyze. To this end, True Code makes use of "Contexts". A Context can be understood as a set of functions, on which a code checker can run. When configuring the execution of a code checker, you need to specify a "Context" to run on. When executing, True Code will analyze *every function in the Context* using the checker. Being this said, it is important to keep this in mind when defining Contexts; try to include significant functions in a Context that is meant to run an *integer overflow validation* code check.

1.3.3 I get a blue "Outdated codebase index" message. What does it mean? How can I fix this?

With this warning, True Code is informing you that it detected a change in your source files, so the function index needs to be updated, or it needs to be created, in case you have never indexed the codebase.

To fix this warning, please go to menu "Workspace settings" and then to the "Context settings" option to open the "Context settings" window. In the "Configure codebase indexing" section select your configurations and then click on the "Run" button, next to "Index codebase". Depending on the size of your code base, this process might take a long time and significant resources from your computer. We recommend running this indexing overnight.

1.3.4 What is the difference between “Full” and “Partial” indexing mode?

When True Code detects that one or more files in your code base were modified, it needs to update the index of function definitions. Since creating such index from the scratch might be a process that takes a long time and significant resources from your computer, True Code offers two options to run the indexing. On one hand, you can always run a “Full” indexing, which will re-create the index from scratch. However, if only a few files were changed, True Code will detect this and suggest to run a “Partial” index of the code base. This partial indexing will scan only the files that were changed, reducing significantly the time and resources spent in updating the code base index.

1.3.5 What is the difference between indexing files in “Db only” and “All files”?

By default, True Code creates a function definition index only considering the files that are present in the compilation database. This is referred as indexing file “Db only”. In some cases, you would like to include all the source files in your code base. If this is the case, select “All files” to index all the source files.

1.3.6 C++ support: What are the limitations of True Code when analyzing C++ codebases?

True Code is mostly used and tested for C codebases. There are some known limitations with regard to C++ codebase:

- The codebase index will not contain entries for the template definitions. It will contain entries for all explicit template instantiations. For more information, please see Manual, chapter “Context settings”, section “Index limitations”.
- Function stubbing for FI simulation and Fuzzing has been developed and tested for C codebase.

1.3.7 C++ support: What should I select: “Expand and parse templates” or “Ignore templates”?

By default, True Code will try to expand C++ templates, creating new functions definitions and it will add these new functions to the code base index. We recommend you keep this default configuration. However, some C++ templates usages are not fully supported by True Code, which makes the indexing failing. If this is the case, you can try using the “Ignore templates” option to skip the expansion and indexing of C++ templates. With this configuration, you will be able to run Code Checkers and Fault Injection Simulation on all functions not derived by a C++ template.

1.3.8 What are “Contexts”? How does True Code use them?

When running the Code Checkers, you need to tell True Code which are the functions on which the check needs to be run. To ease this process, True Code makes use of a “Context”, which groups a set of function definitions into a single set. When configuring the execution of a Code Checker, you can configure the function to analyze by simply selecting the Context on which you would like to run the checker. True Code will then analyze every function in the configured Context.

1.3.9 How can I select the “right” set of functions for a Context? How this selection can impact the results of the Code Checkers?

Unfortunately, there is no generic way to have the right selection of functions to a context. In question 1.3.11 you can find some guidelines about which function to add to which context. However, the final answer needs to be assessed by you development team.

Since the creation of contexts is a complicated process, a “right” or “wrong” selection can heavily impact the results, and therefore, the conclusions of the analysis. For example, if you do not select the right functions in your context, you can have detections of issues whose countermeasure was introduced in other parts of the code, which leads to false positives. To minimize this risk, we encourage you to get help from your development team to decide the list of functions to start the analysis, defining the Context for each code checker.

1.3.10 How can I define my own True Code contexts to analyze a set of functions?

To define your customized Contexts, go to the “Workspace settings” menu, and then to “Context settings”. On the left panel, you can find the “Add context type” button, to create a new Context. In the “New Context type” window, you can give a name and a color to the new context. Then click on “OK” to create the new empty context. It should now appear on the left side of the window.

To add functions to the newly created context, click on the “Configure” button identified by a gear icon. The “Generate context” window appears, offering you the options about how to add functions to this context. For further reference how to use this interface, please refer to True Code’s manual.

After you configured the rules for include functions, click on the “Save” button to accept your changes. Finally, click on the “Run” button next to “Generate context”. The newly created context will be populated with function definitions that match the rules you defined.

Later, you can use the context you defined when running any of the Code Checkers.

1.3.11 How can I know which function I need to include/exclude in/from a Context?

In paragraph “Context settings” of the manual, three context types are described: “Untrusted interfaces”, “Fault Injection sensitive” and “All code”, together with a description of the type of functions that belong to a context type.

Chapter “Code checks”, section “Common configuration field and buttons” of the manual, describes how each code checker is linked to a context type.

Despite these suggestions, we recommend getting in contact with your development team to know more about what work do the functions perform, to which context they belong and which code checkers to run on each of them.

1.3.12 What is the difference between creating a context from the “Context settings” window and from the button “Create new context for the selected item” from the file editor?

No differences.

1.3.13 Are True Code contexts being used in the Fault Injection simulator?

The main goal of the Fault Injection simulator is to simulate the execution of a single function, up to a given call depth. In this sense, the True Code Contexts cannot be used, since they can contain more than just one function definition.

1.4 Code Checkers

1.4.1 When running the code checkers, does the tool only compile or also links and create object files?

In the “Compilation configuration” wizard, True Code will create a compilation database, which will tell True Code how to compile each one of your source files. When you run a code checker, true Code will use the compilation database to create an Intermediate Representation file, containing LLVM IR code (IR file). Since this representation is used by True Code to perform the analysis, it will create one IR file per source file that contains either function being analyzed (entry function) or functions called by the entry function up to the configured depth of the analysis.

When running the code checkers, these files are loaded by True Code, but no linkage is performed, nor is any object file created. True Code will simply load the LLVM IR files, and create internal structures to perform the analysis of your code.

1.4.2 What is the “Symbolic execution limit” parameter found in some code checkers? What is the impact it has on the execution of those code checkers?

Some code checkers search for vulnerabilities that depend on specific values that can be assigned to variables so that they lead to a security vulnerability (for example an integer overflow or an array out of bounds access). There may be only specific execution paths leading to a sensitive operation where the variables that are used in the sensitive operation can have values that lead to the security vulnerability. For a specific execution path, TrueCode applies symbolic execution to determine if there is a combination of values that can be assigned to the unknown variable, such that the security vulnerability is present. “Symbolic execution limit” parameter is used to limit the number of execution paths on which the symbolic execution search is applied.

1.4.3 What is the “Timeout (seconds)” parameter found in some code checkers? What is the impact it has on the execution of those code checkers?

This limits the number of seconds that the code checker will spend analyzing each function. If it doesn’t find a security vulnerability during that time, it will print a message in the logging window that the timeout limit was reached, and it will move on to analyze the next function (without creating any annotation for the current function)

1.4.4 In the “Time of Check – Time of Use”, what is the “Assumed dereference” parameter? What is the impact it has on the execution of this code checker?

When an unknown function (a function for which TrueCode doesn’t have the source file – for example library functions) is encountered during analysis, TrueCode by default assumes that the function will not dereference any function arguments that are pointers.

However, in some case it may be possible that the user knows that one or more functions dereference the pointer function arguments. For example this may happen for the memcpy function.

In this “Assumed dereference” parameter the user can set a list of such functions. TrueCode will then treat the function call as a dereference of the pointer function parameter, which may lead to reporting a vulnerability of the type “Time of Check – Time of Use” – depending on the usage of the pointer in the rest of the analyzed function.

1.5 True Code known issues

1.5.1 I installed the Eclipse plugin, but when I click on True Code's buttons, nothing happens (no windows are shown, not even an error message).

Starting from version 2020-12, Eclipse come with an embedded Java version that is known to have compatibility issues with True Code. We are currently looking for a suitable solution for this issue. In the meantime, you can work around the compatibility issues by inserting a small change in the Eclipse configuration files. Locate your 'eclipse.ini' file and edit it by inserting the following lines after the line that contains '-vmargs':

```
--add-exports=java.desktop/sun.java2d=ALL-UNNAMED
--illegal-access=permit
```

Save the file, and restart Eclipse if it was running. The next time you try to open True Code, it should open correctly.

An alternative solution is to use an external JRE to run Eclipse. Please check the minimal JRE version for the Eclipse version you want to run in https://wiki.eclipse.org/Eclipse/Installation, then install and JRE/JDK that fits the requirement. Please note that any version Java equal or higher than 15 is known to have compatibility issues with True Code. To configure Eclipse to use this external Java installation, please follow the official instructions provided by The Eclipse Foundation: https://wiki.eclipse.org/Eclipse/Installation#Configure_Eclipse_to_use_the_JVM.

Contact

Please send a mail via support@riscure.com or ‘Login to the support portal’ and follow the procedure under the portal, for any of the following reasons:

- To ask for a new license file.
- To request a new feature that you may like to have in True Code
- To rise a support ticket related to an issue you encounter during the usage of True Code
- To notice as about a bug you encounter during the usage of True Code

For any other requests please send a mail via inforequest@riscure.com.