securityandtrust.lu

FAQAS Framework

SUM - Software User Manual

O. Cornejo, F. Pastore

Interdisciplinary Centre for Security, Reliability and Trust

University of Luxembourg

ITT-1-9873-ESA-FAQAS-SSS

Issue 1, Rev. 1

March 19, 2021

UNIVERSITÉ DU LUXEMBOURG

Revisions

Issue Number	Date	Authors	Description
ITT-1-9873-ESA-FAQAS-SUM Issue 1 Rev. 1	March 31th, 2021	Oscar Cornejo, Fabrizio Passtore	Initial release.

Contents

Chapter 1

Scope and content

This document is the deliverable SUM of the ESA activity ITT-1-9873-ESA. It concerns the software user manual for the *FAQAS framework* to be delivered by ITT-1-9873-ESA. Following the structure described in the SoW *AO9873-ws00pe_SOW.pdf*, it provides instructions for the users of the FAQAS framework according to ECSS-E-ST-40C Annex B.

1.1 Applicable and reference documents

- D1 - Mutation testing survey
- D2 - Study of mutation testing applicability to space software

Chapter 2

Terms, definitions and abbreviated terms

- FAQAS: activity ITT-1-9873-ESA
- FAQAS-framework: software system to be released at the end of WP4 of FAQAS
- D2: Deliverable D2 of FAQAS, *Study of mutation testing applicability to space software*
- KLEE: Third party test generation tool, details are provided in D2.
- SUT: Software under test, i.e, the software that should be mutated by means of mutation testing.
- WP: Work package

Chapter 3

External View of the Software

The FAQAS-framework is delivered as an archive consisting of the source files and a installer. The following is a depiction of the directory structure:

- `FAQASFramework/`
  - `SRCMutation/`: contains the source files of the component that performs source mutations.
  - `llvm-build.sh`: build script that compiles the SRCMutation component
  - `PythonWrappers/`: contains Python script wrappers that facilitate source code mutations.
  - `MASS/`
    * `FAQAS-Setup`: contains the Bash scripts necessary to install the FAQAS-Framework.
    * `FAQAS-GenerateCodeCoverageMatrixes`: contains the Bash scripts providing procedures to collect code coverage from the SUT.
    * `FAQAS-GenerateMutants`: contains a Bash script that invokes the `SRCMutation` to generate mutants.
    * `FAQAS-CompileOptimizedMutants`: contains the Python and Bash scripts that provides the procedures to compile mutants and filter equivalent and redundant mutants based on trivial compiler optimizations.
    * `FAQAS-CompileAndExecuteMutants`
      · `FAQAS-GeneratePrioritizedTestSuite`: contains the Python and Bash scripts that provides the procedures to generate prioritized and reduced test suites from the SUT.
      · `FAQAS-CompileAndExecute`: contains the Python and Bash scripts that provides the procedures to compile and execute the mutants against the SUT test suite. It also provides the procedures to determine the mutation stopping criterion (i.e., mutant sampling).
      · `FAQAS-IdentifyEquivalentAndRedundantMutants`: contains the Python and Bash scripts that provides the procedures to identify equivalent mutants based on code coverage.
    * `FAQAS-MutationScore`: contains the Python and Bash scripts that provides the procedures to compute the mutation score and provide summarized information about the code-driven mutation testing process.

Chapter 4

Operations Manual

4.1 Set-up and Initialization

MASS strongly depends on LLVM for source code mutation. For this reason, the set-up procedure consists of installing LLVM-3.8.1 and the `SRCMutation` component. For this procedure, a Bash script is provided.

The following shell command installs the corresponding dependencies and the `SRCMutation` component.

```
$ ./llvm-build.sh
```

4.1.1 Dependencies

- Linux packages: `r-base`, `jq`, `Python 3.7` or higher
- R packages: `binom`
- Python packages: `numpy`, `scipy`

4.2 Getting started

4.2.1 Initialization of the MASS workspace

MASS shall create a workspace folder where all the steps from the methodology shall be stored.

An installation Bash script is provided for the creation of this workspace, the script can be found on `$FAQAS/MASS/FAQAS-Setup/install.sh`

To use the installation script the shell variable `INSTALL_DIR` has to be set:

```
$ export INSTALL_DIR=/opt/DIRECTORY
```

If the `INSTALL_DIR` directory must be binded inside a container. Then, also the shell variable `EXECUTION_DIR` has to be set. This step is optional.

After setting the corresponding variables, the following commands are necessary to create the *MASS* workspace folder:

```
$ cd $FAQAS/MASS/FAQAS-Setup
$ ./install.sh
```

Once the installation folder has been created, the folder shall contain the following structure and files:

- `Launcher.sh`: *MASS* single launcher; the script executes all the steps of the methodology in one command.
- `mass_conf.sh`: *MASS* configuration file, has to be set before being able to execute *MASS*.
- `mutation_additional_functions.sh`: Bash script that must be filled by the application engineer before executing *MASS*.
- `MASS_STEPS_LAUNCHERS/`: folder containing all the single launchers for each step of the *MASS* methodology.
  - `MASS_STEPS_LAUNCHERS/PrepareSUT.sh`: launcher for the script that prepare the SUT and collects information about the SUT test suite.
  - `MASS_STEPS_LAUNCHERS/GenerateMutants.sh`: launcher for the generation of mutants.
  - `MASS_STEPS_LAUNCHERS/CompileOptimizedMutants.sh`: launcher for the trivial compiler optimization step.
  - `MASS_STEPS_LAUNCHERS/OptimizedPostProcessing.sh`: launcher for the post-processing of the trivial compiler optimization step.
  - `MASS_STEPS_LAUNCHERS/GeneratePTS.sh`: launcher for the generation of prioritized and reduced test suites.
  - `MASS_STEPS_LAUNCHERS/ExecuteMutants.sh`: launcher for the execution of mutants against the SUT test suite.
  - `MASS_STEPS_LAUNCHERS/IdentifyEquivalents.sh`: launcher for the identification of equivalent mutants based on code coverage.
  - `MASS_STEPS_LAUNCHERS/MutationScore.sh`: launcher for the computation of the mutation score and final reporting.
  - `MASS_STEPS_LAUNCHERS/PrepareMutants_HPC.sh`: launcher that prepare the mutants workspace for the execution on HPCs.
  - `MASS_STEPS_LAUNCHERS/ExecuteMutants_HPC.sh`: launcher that execute mutants on HPCs.
  - `MASS_STEPS_LAUNCHERS/PostMutation_HPC.sh`: launcher that assess past mutant executions, and decide whether more mutant executions are needed.

4.2.2 MASS Configuration

There are three Bash scripts that needs intervention from the SUT engineer. These three scripts enable *MASS* the correct identification of the SUT paths (e.g., source code folder, test suite folder), the SUT compilation commands, the SUT test suite execution commands, and the configuration of *MASS* itself (e.g., trivial compiler optimizations flags, mutant selection strategy, sampling rate).

The last intervention, regards providing a template of the original build script for the trivial compiler optimizations step. More details are provided in the following.

4.2.2.1 MASS Configuration File

The *MASS* configuration file is the Bash file `$INSTALL_DIR/mass_conf.sh`; within this file there are multiple environment variables that must be set:

```
# set SRCIROR path
export SRCIROR=

# set workspace directory path where MASS files can be stored (i.e., $INSTALL_DIR)
export APP_RUN_DIR=

# specifies the building system, available options are "Makefile" and "waf"
export BUILD_SYSTEM=

# directory root path of the SUT
export PROJ=

# directory source path of the SUT
export PROJ_SRC=

# directory test path of the SUT
export PROJ_TST=

# directory coverage path of the SUT
export PROJ_COV=

# directory path where the compiled binary is stored
export PROJ_BUILD=

# list of folders not to be included during coverage analysis, name folders shall be separated by '\|'
export COVERAGE_NOT_INCLUDE=

# filename of the compiled file or library
export COMPILED=

# path to the original build script
export ORIGINAL_MAKEFILE=

# compilation command of the SUT, the command shall be specified as a Bash array, e.g., (). Special characters shall be escaped.
export COMPILATION_CMD=

# additional commands for compiling the SUT (e.g., setup of workspace), the command shall be specified as a Bash array, e.g., (). Special characters shall be escaped.
export ADDITIONAL_CMD=

# command to be executed after each test case execution (optional), the command shall be specified as a Bash array, e.g., (). Special characters shall be escaped.
export ADDITIONAL_CMD_AFTER=

# compilation command for TCE analysis, the command shall be specified as a Bash array, e.g., (). Special characters shall be escaped.
export TCE_COMPILE_CMD=

# command to clean installation of the SUT, the command shall be specified as a Bash array, e.g., (). Special characters shall be escaped.
export CLEAN_CMD=

# relative path to location of gcov files (i.e., gcda and gcno files)
export GC_FILES_RELATIVE_PATH=
```

Furthermore, the following specific *MASS* variables must be set:

```
# specify if MASS will be executed on a HPC, possible values are "true" or "false"
export HPC=

# TCE flags to be tested, the flags shall be specified as a Bash array, e.g., ("-O0" "-O1").
export FLAGS=

# set if MASS should be executed with a prioritized and reduced test suite, possible values are "true" or "false"
```

```
8  export PRIORITIZED=
9
10 # set sampling technique, possible values are "uniform", "stratified", and "fsci"
11 # note: if "uniform" or "stratified" is set, $PRIORITIZED must be "false"
12 export SAMPLING=
13
14 # set sampling rate if whether "uniform" or "stratified" sampling has been selected
15 export RATE=
```

4.2.3 Prepare SUT Script Configuration

The prepare SUT configuration file is the Bash file `$INSTALL_DIR/MASS_STEPS_LAUNCHER/PrepareSUT.sh`; within this file the following actions must be provided by the engineer:

1. Provide commands to generate a compilation database file `compile_commands.json` of the SUT. Note that the paths defined within the database file must be full paths;
2. Provide commands to compile the SUT;
3. Provide additional commands to prepare SUT workspace (optional);
4. Provide commands to execute the SUT test suite iteratively over each test case;
    - After the execution of a test case add a call to `$MASS/FAQAS-GenerateCodeCoverageMatrixes/FAQAS-CollectCodeCoverage.sh` script;
    - the script `FAQAS-CollectCodeCoverage.sh` shall be invoked with three arguments: (i) the test case name, (ii) the time taken to run the test case in seconds, and (iii) the root folder where all the coverage files are being stored.

4.2.4 Mutation Script Configuration

The mutation script configuration file is the Bash file `mutation_additional_functions.sh`; within this file a Bash function `run_tst_case` must be provided by the engineer to guarantee a correct mutation testing process. This function shall receive as an argument the name of the test case to be executed, and then it should execute the command for running the test case. Particularly, the return should return 0, if the test case passes, and it should return 1 if the test fails.

4.2.5 Build Script for Compiler Optimizations

The SUT engineer shall provide a template of the original SUT build script. Such template shall be placed in the same folder where the original build script resides. Moreover, the template should have the following characteristics:

- The template shall not contain any debugging flags within compilation/linking commands;
- The template shall not contain any code coverage flags within compilation/linking commands;
- The template shall contain a placeholder for the compiler optimization option, specifically the placeholder `TCE`;

- The template shall contain a 'sort' command in the source dependency list to ensure that source files are always compiled in the same order;
- The template shall be named the same as the original build script, but with an ending '.template'.

4.2.6 Running MASS on single machines

4.2.6.1 One Step Launcher

One possible way to run *MASS* is to execute all the eight steps of the framework with one command. Such action will execute the following steps sequentially:

1. PrepareSUT
2. GenerateMutants
3. CompileOptimizedMutants
4. OptimizedPostProcessing
5. GeneratePTS
6. ExecuteMutants
7. IdentifyEquivalents
8. MutationScore

To execute the one step launcher, the following command shall be provided:

```
$ ./Launcher.sh
```

4.2.6.2 Multiple Step Launchers

Another possible way to run *MASS* is to execute all the eight steps of the framework through independent commands. The multiple steps of the methodology and its command are described in the following.

1. **Prepare the SUT**

```
    $ ./MASS_STEPS_LAUNCHER/PrepareSUT.sh
```

2. **Generate Mutants**

```
    $ ./MASS_STEPS_LAUNCHER/GenerateMutants.sh
```

3. **Compile Optimized Mutants**

```
    $ ./MASS_STEPS_LAUNCHER/CompileOptimizedMutants.sh
```

4. **Compile Optimized Mutants Post-Processing**

```
$ ./MASS_STEPS_LAUNCHER/OptimizedPostProcessing.sh
```

5. **Generate Prioritized and Reduced Test Suites**

```
$ ./MASS_STEPS_LAUNCHER/GeneratePTS.sh
```

6. **Execute mutants**

```
$ ./MASS_STEPS_LAUNCHER/ExecuteMutants.sh
```

7. **Identify Equivalent Mutants based on Code Coverage**

```
$ ./MASS_STEPS_LAUNCHER/IdentifyEquivalents.sh
```

8. **Computer Mutation Score**

```
$ ./MASS_STEPS_LAUNCHER/MutationScore.sh
```

4.2.7 Running MASS on HPC infrastructures

Given that resources from HPC infrastructures has to be requested for every performed tasks, it is not possible to run all the steps from *MASS* in one step. However, since resources can be requested accordingly, *MASS* can perform multiple steps simultaneously, enhancing the capabilities of the toolset. With a HPC, for example, *MASS* could analyze way more mutants than if *MASS* is executed on a single machine.

The multiple steps of the methodology and its command are described in the following.

1. **Prepare the SUT**

```
$ ./MASS_STEPS_LAUNCHER/PrepareSUT.sh
```

2. **Generate Mutants**

```
$ ./MASS_STEPS_LAUNCHER/GenerateMutants.sh
```

3. **Compile Optimized Mutants**: If the environment variable `HPC` is set to `true`. Then, a parameter can be passed to the launcher script indicating the optimization level to be processed. If six levels of optimizations are defined, then numbers between zero and five can be provided.

```
$ level=0
$ ./MASS_STEPS_LAUNCHER/CompileOptimizedMutants.sh $level
```

4. **Compile Optimized Mutants Post-Processing**

```
$ ./MASS_STEPS_LAUNCHER/OptimizedPostProcessing.sh
```

5. **Generate Prioritized and Reduced Test Suites**

```
$ ./MASS_STEPS_LAUNCHER/GeneratePTS.sh
```

6. **Prepare mutants**

```
$ ./MASS_STEPS_LAUNCHER/PrepareMutants_HPC.sh
```

7. **Execute mutants**: The launcher script receives two parameters: the first parameter is the number of the mutant to be executed, the number of mutant is defined as $1..M$, being M the total number of mutants. The total number of mutants can be derived from the folder `$INSTALL_DIR/hpc-src-mutants`. The second parameter defines if the test suite has to be executed in a reduced fashion or not. The possible values are “true” and “false”.

```
$ nr_mutant=1
$ reduced="false"
$ ./MASS_STEPS_LAUNCHER/ExecuteMutants_HPC.sh $nr_mutant $reduced
```

8. **Post-mutation execution**: The launcher script receives two numbers as parameters, a minimum and a maximum value, that represent the range of mutants to assess. The assessment consists of evaluating if more mutant executions are needed.

```
$ min=1
$ max=700
$ ./MASS_STEPS_LAUNCHER/PostMutation_HPC.sh $min $max
```

9. **Identify Equivalent Mutants based on Code Coverage**

```
$ ./MASS_STEPS_LAUNCHER/IdentifyEquivalents.sh
```

10. **Computer Mutation Score**

```
$ ./MASS_STEPS_LAUNCHER/MutationScore.sh
```

4.3 Mode selection and control

4.4 Normal Operations

4.5 Normal Termination

4.6 Error Conditions

4.7 Recover Runs

Chapter 5

Tutorial

5.1 Introduction

This tutorial presents how to use *MASS* on a typical case. Since *MASS* provides two modes of execution, we provide an example for running *MASS* on a single machine, and another example for running *MASS* on HPC infrastructures.

Both examples use the Mathematical Library for Flight Software as case study to exemplify the steps to follow for the use of FAQAS-framework.

5.2 Getting Started

5.3 Using the software on a typical task

5.3.1 Single Machine Example: Mathematical Library for Flight Software

The first step regards installing the *MASS* framework, please refer to Section 4.1.

The second step, consists of creating and installing a workspace folder for running *MASS* on the MLFS example. For this case, the workspace folder will be created on `/opt/MLFS`.

```
$ cd $FAQAS/MASS/FAQAS-Setup
$ export INSTALL_DIR=/opt/MLFS
$ ./install.sh
```

The third step consists of configuring the *MASS* configuration file `mass_conf.sh`. In the following, we provide excerpts of the file that require intervention from the engineer. The following excerpt contains the necessary configuration for the MLFS case study.

```
# set FAQAS path
export SRCIROR=/opt/srcirorfaqas

...

# set directory path where MASS files can be stored
export APP_RUN_DIR=/opt/MLFS

# specifies the building system, available options are "Makefile" and "waf"
```

```
export BUILD_SYSTEM="Makefile"

# directory root path of the software under test
export PROJ=$HOME/mlfs

# directory src path of the SUT
export PROJ_SRC=$PROJ/libm

# directory test path of the SUT
export PROJ_TST=$HOME/unit-test-suite

# directory coverage path of the SUT
export PROJ_COV=$HOME/blts_workspace

# directory path of the compiled binary
export PROJ_BUILD=$PROJ/build-host/bin

# filename of the compiled file/library
export COMPILED=libmlfs.a

# path to original Makefile
export ORIGINAL_MAKEFILE=$PROJ/Makefile

# compilation command of the SUT
export COMPILATION_CMD=(make all ARCH=host EXTRA_CFLAGS="-DNDEBUG" \&\& make all COVERAGE="true" ARCH=
    host_cov EXTRA_CFLAGS="-DNDEBUG")

# compilation additional commands of the SUT (e.g., setup of workspace)
export ADDITIONAL_CMD=(cd $HOME/blts/BLTSConfig \&\& make clean install INSTALL_PATH="$HOME/
    blts_install" \&\& cd $HOME/blts_workspace \&\& $HOME/blts_install/bin/blts_app --init)

# command to be executed after each test case (optional)
export ADDITIONAL_CMD_AFTER=(rm -rf $HOME/blts_workspace/*)

# compilation command for TCE analysis
export TCE_COMPILE_CMD=(make all ARCH=host EXTRA_CFLAGS="-DNDEBUG")

# command to clean installation of the SUT
export CLEAN_CMD=(make cleanall)

# relative path to location of gcov files (i.e., gcda and gcno files)
export GC_FILES_RELATIVE_PATH=Reports/Coverage/Data
```

Also *MASS* variables shall be configured within the same file. Particularly, we will run *MASS* with the following setup.

```
### MASS variables

# TCE flags to be tested
export FLAGS("-O0" "-O1" "-O2" "-O3" "-Ofast" "-Os")

# specify if MASS will be executed on a HPC, possible values are "true" or "false"
export HPC="false"

# set if MASS should be executed with a prioritized and reduced test suite
export PRIORITIZED="true"

# set sampling technique, possible values are "uniform", "stratified", and "fsci"
# note: if "uniform" or "stratified" is set, $PRIORITIZED must be "false"
export SAMPLING="fsci"

# set sampling rate if whether "uniform" or "stratified" sampling has been selected
export RATE=""
```

The fourth step consists of configuring the prepare SUT configuration file (`/opt/MLFS/MASS_STEPS_LAUNCHER/PrepareSUT.sh`; within this file the following actions must be provided by the engineer:

```
#!/bin/bash

# This file should be prepared by the engineer!
cd /opt/MLFS
```

```
. ./mass_conf.sh

# 1. Compile SUT
## example

cd $PROJ

# # generate compile_commands.json and delete build
bear make all && rm -rf build* && sed -i 's: libm: /home/mlfs/mlfs/libm:' compile_commands.json && mv compile_commands.json $MUTANTS_DIR
eval "${COMPILATION_CMD[@]}"

# 2. Prepare test scripts
# example

cd $HOME/blts/BLTSConfig
make clean install INSTALL_PATH="$HOME/blts_install"

# Preparing MLFS workspace (e.g., where test cases data is stored)
cd $HOME/blts_workspace
$HOME/blts_install/bin/blts_app --init

# 3. Execute test cases
# Note: execution time for each test case should be measured and passed as argument to FAQAS-CollectCodeCoverage.sh

# example
for tst in $(find $HOME/unit-test-suite -name '*.xml');do
    cd $HOME/blts_workspace

    tst_filename_wo_xml=$(basename -- $tst .xml)

    start=$(date +%s)
    $HOME/blts_install/bin/blts_app -gcrx $tst_filename_wo_xml -b coverage --nocsv -s $tst
    end=$(date +%s)

    # call to FAQAS-CollectCodeCoverage.sh
    # parameter should be test case name and the execution time
    source $MASS/FAQAS-GenerateCodeCoverageMatrixes/FAQAS-CollectCodeCoverage.sh $tst_filename_wo_xml "$(($end-$start))" $PROJ_COV
done
```

The fifth step consists of defining the function `run_tst_case` within the file `mutation_additional_functions.sh`:

```
run_tst_case() {

    tst_name=$1
    tst=$PROJ_TST/$tst_name.xml

    echo $tst_name $tst

    # run the test case
    cd $PROJ_COV
    $HOME/blts_install/bin/blts_app -gcrx $tst_name -b coverage --nocsv -s $tst

    # define if test case execution passed or failed
    summaryreport=$tst_name/Reports/SessionSummaryReport.xml
    originalreport=$HOME/unit-reports/$summaryreport

    test_cases_failed=`xmllint --xpath "//report_summary/test_set_summary/test_cases_failed/text()" $summaryreport`
    o_test_cases_failed=`xmllint --xpath "//report_summary/test_set_summary/test_cases_failed/text()" $originalreport`

    echo "comparing with original execution"
    echo $test_cases_failed $o_test_cases_failed

    if [ "$test_cases_failed" != "$o_test_cases_failed" ]; then
        return 1
    else
        return 0
    fi
}
```

The sixth step consists of providing a template for the build script for the trivial compiler optimizations step:

In particular, we replaced the following command:

```
CFLAGS = -c -Wall -std=gnu99 -pedantic -Wextra -frounding-math -fsignaling-nans -g O2 -fno-builtin $(
    EXTRA_CFLAGS)
```

By this one:

```
CFLAGS = -c -Wall -std=gnu99 -pedantic -Wextra -frounding-math -fsignaling-nans TCE -fno-builtin $(
    EXTRA_CFLAGS)
```

The seventh step consists of launching the one step launcher (see Section 4.2.6:

```
 $ /opt/MLFS/Launcher.sh
```

The results shall be reported at the end of the execution:

```
##### MASS Output #####
## Total mutants generated: 28071
## Total mutants filtered by TCE: 6918
## Sampling type: fsci
## Total mutants analyzed: 461
## Total killed mutants: 369
## Total live mutants: 92
## Total likely equivalent mutants: 53
## MASS mutation score (%): 90.44
## List A of useful undetected mutants: /opt/MLFS/RESULTS/useful_list_a
## List B of useful undetected mutants: /opt/MLFS/RESULTS/useful_list_b
## Number of statements covered: 1973
## Statement coverage (%): 100
## Minimum lines covered per source file: 2
## Maximum lines covered per source file: 138
```

5.3.2 HPC Infrastructure Example: Mathematical Library for Flight Software

The first step regards installing the *MASS* framework, please refer to Section 4.1.