OOPSLA'19
Compiler Fuzzing: How Much Does It Matter?
Overview of the Artifact

Michaël Marcozzi, Qiyi Tang, Cristian Cadar
Imperial College London, UK

Alastair Donaldson
Google and Imperial College London, UK

09/07/2019

1 Getting Started Guide

1.1 Install Virtualbox

The artifact has been fully loaded into a virtual machine. For running this virtual machine, you need to install Virtualbox first, which can be downloaded from `www.virtualbox.org` for Windows, Linux, Macintosh and Solaris. If your computer is running Ubuntu Linux, you can alternatively install it using apt:

```
sudo apt install virtualbox
```

We have tested the artifact under Virtualbox version 6.0, but it should work just fine with any other recent version.

1.2 Import the artifact's virtual machine

Download the artifact's virtual machine file using the link `https://bit.ly/2xAjnqN` (about 1GB, shortened URL to Google Drive). The virtual machine can use up to 100GB of disk space and has a default RAM usage of 8GB. You can reduce the RAM available to the virtual machine during import if necessary, but it may lead to out of memory failures for the most demanding experiments. The virtual machine's host should have at least 20GB of free disk space and a broad, fast and reliable internet connection, as the artifact downloads and builds compilers and applications on the fly.

To import the virtual machine file into Virtualbox using the graphical interface, simply follow these instructions: `https://www.virtualbox.org/manual/UserManual.html#ovf-import-appliance` (we recommend you to keep the default VM settings). If your computer is running Ubuntu Linux, you can alternatively import the virtual machine using the command line:

```
vboxmanage import compiler_bugs.ova
```

1.3 Start and connect to the artifact's virtual machine

Start the imported virtual machine either using the graphical interface (`https://www.virtualbox.org/manual/UserManual.html#intro-running`) or the Ubuntu Linux terminal:

```
VBoxHeadless --startvm debian
```

Taking advantage of the graphical interface, you can directly login into the virtual machine window, using the following credentials:

USER user42

PASSWORD user42user42

With the same credentials, you can also connect to the virtual machine from the host using ssh:

```
ssh -p 2222 user42@localhost
```

1.4 Generate the paper's tables using the data collected during the study

Once logged on the virtual machine, go to the data directory:

cd /home/user42/compiler-bug-impact/data

All the data that we collected while running the experiments detailed in the section 5 of the paper are available in this folder. The logs of building and testing our 309 Debian applications with our 45 versions of Clang/LLVM are stored in the Build_Logs subfolder. The results of our assembly-level static analysis of the built applications, aiming at computing the number of syntactically different functions, are stored in the Function_Logs subfolder.

From these data, you can generate the tables 2 to 6 presented in the paper, using the five genTableX.sh scripts available in the folder (where X refers to the table number). For example, running command ./genTable6.sh will generate table 6 as a .csv file in the results subfolder. You can then pretty-print this .csv file using the displayResults.sh script:

```
./displayResults.sh /home/user42/compiler-bug-impact/data/results/table6.csv
```

1.5 Run the impact analysis for one compiler bug and two applications

Finally, you can run by yourself a tiny fraction of the experiments detailed in the section 5 of the paper and compare the results with the ones stored in the data directory. To do so, simply go to the example directory and run the run_example.sh script (you will have to enter the sudo password "user42user42" after some time):

```
cd /home/user42/compiler-bug-impact/example
./run_example.sh
```

This script runs our impact analysis experiment for the Clang/LLVM bug #26323 over the afl and libraw Debian applications and saves the results, which can by displayed by running

```
cat /home/user42/compiler-bug-impact/example/results/26323/new-26323.txt
cat /home/user42/compiler-bug-impact/example/results/26323/26323-func.txt
```

Theses results can then be compared to the ones stored in the data directory, which can be gathered by running the following commands:

```
grep -A4 "afl" /home/user42/compiler-bug-impact/data/Build_Logs/EMI/new-26323.txt
grep -A4 "libraw" /home/user42/compiler-bug-impact/data/Build_Logs/EMI/new-26323.txt
grep -A2 "libraw" /home/user42/compiler-bug-impact/data/Function_Logs/EMI/26323-func.txt
```

2 Step-by-Step Instructions

Our paper presents an empirical study of the impact of 45 Clang/LLVM bugs over 309 Debian applications. This artifact enables one to reproduce the automated part of this study for each of the bugs, either by using prebuilt versions of Clang/LLVM affected by the bug, or by first downloading, preparing and building the corresponding Clang/LLVM sources, as explained

in the paper. The second alternative is optional: it gives more control and insight over the experimental process, but requires much more time and could not be fully automated.

It should be noted that running the full impact analysis for all the bugs and applications considered in the paper took us five months on a range of virtual machines dispatched on a dozen of servers. If you are just aiming at quickly evaluating this artifact or understanding our experimental infrastructure and process, we recommend you to run the experiments only for one or two bugs over a fraction of our 309 packages.

2.1 Run the impact analysis for any bug and applications (prebuilt compilers)

The list of Debian applications over which the analysis should be performed is stored in the following file (which you can display using the **cat** command):

```
cat /home/user42/compiler-bug-impact/scripts/build/tasks-small.json
```

By default, it only contains the afl and libraw Debian applications used in the first part of this overview. To load our full list of 309 Debian applications instead, simply run the following command:

```
mv /home/user42/compiler-bug-impact/scripts/build/tasks-full.json
   /home/user42/compiler-bug-impact/scripts/build/tasks-small.json
```

Alternatively, you can edit the file by hand to customize the list of Debian applications over which the analysis should be performed. In particular, to add a new application, you just need to add a new element to the list, which should contain the targeted application name and version (to be extracted from `https://packages.debian.org/source/stretch/`), while the other fields — chroot, esttime, modes and type — should remain untouched.

To perform the impact analysis for one of the bugs studied in the paper over the list of applications that you have chosen, go to the example directory and run the run_example_bug.sh script with the bug ID number as a parameter (you will have to enter the sudo password "user42user42" after some time). For example, for EMI bug #26323, you need to run:

```
cd /home/user42/compiler-bug-impact/example
./run_example_bug.sh 26323
```

The run_example_bug.sh script works in the same way as the run_example.sh script from the first section of this overview. In particular, it downloads the prebuilt compilers from the Internet and stores them in the /home/user42/compilers folder. The obtained results are then stored in the /home/user42/compiler−bug−impact/example/results folder. If you run experiments for many bugs and applications, the size of these two folders might become very large, so that they might need to be purged at some point.

2.2 Run the impact analysis for any bug and applications (source compilers)

In the previous subsection, the run_example_bug.sh script was just fetching prebuilt versions of the buggy, fixed and warning-laden compilers for the analyzed bug. In this subsection, we detail how these compilers could be built on the virtual machine from the Clang/LLVM source code.

First, go to the compiler build scripts folder:

```
cd /home/user42/compiler-bug-impact/scripts/compilers
```

There, you can run the download−sources.sh script to download the source code of the buggy and fixed compilers for the bug. This script requires two parameters: first, the ID number of the considered bug and secondly the SVN revision number corresponding to the fixing patch for this bug. For example, for EMI bug #26323, you need to run (you may have to enter the sudo password "user42user42" after some time):

```
./download-sources.sh 26323 258904
```

given that bug #26323 has been fixed during revision #258904 as one can check in the corresponding bug report: `https://bugs.llvm.org/show_bug.cgi?id=26323`. We have collected in a file the SVN revision numbers corresponding to the fixing patch of all the bugs analyzed in the paper. You can display them by issuing the following command:

```
cat /home/user42/compiler-bug-impact/scripts/compilers/revisions.txt
```

The downloaded compiler sources are stored in the user42's home folder. For example, for EMI bug #26323, the source code of the buggy and fixed compilers will be stored in the /home/user42/26323/buggy and /home/user42/26323/fixed folders.

REMARKS about the download-sources.sh script:

- The script may take a while to download the sources without giving any terminal feedback and you should ignore any warning messages. If running the script in a tmux session (see `https://github.com/tmux/tmux/wiki`), the progress can be monitored by checking the log file /home/user42/compilers.log.
- For bugs affecting LLVM versions < 3.8, you also need to download compiler-rt. Simply uncomment the part in the script that takes care of compiler-rt.
- For bug 20189, as the fixing patch was incorporated in two contiguous revisions of the compiler sources, the revision number for the buggy compiler should be $revision - 2$ instead of $revision - 1$. See the script to (un)comment the appropriate part.
- For bug 27903, as explained in Section 3.1, its fixes were applied together with other code modifications and/or via a series of non-contiguous compiler revisions. In the downloaded source code of the buggy compiler of this bug, we need to modify line 61 of llvm/lib/CodeGen/StackColoring.cpp from cl::init(**false**), cl::Hidden to cl::init(**true**), cl::Hidden to turn on the buggy optimization.

Once the buggy and fixed compilers sources have been downloaded, a patch should be prepared so that, when applied to the fixed compiler's source code, it would produce the warning-laden compiler described in the paper. We have prepared such patches for all the bugs investigated in our study and they can be found in the following folder:

```
/home/user42/compiler-bug-impact/scripts/compilers/patches
```

To build the buggy, fixed and warning-laden compilers, copy the warning-laden patch in the compilers' source folder and run the build-compiler.sh script with the bug's ID number as a parameter (this script is stored in the same folder as the download-sources.sh script). For example, for EMI bug #26323, you need to run:

```
cp /home/user42/compiler-bug-impact/scripts/compilers/patches/EMI/26323.txt
   /home/user42/26323/patch.txt
./build-compiler.sh 26323
```

REMARKS about the build-compiler.sh script:

- The script may take a while to build the compilers without giving any terminal feedback and you should ignore any warning messages. If running the script in a tmux session, the progress can be monitored by checking the log file /home/user42/compilers.log.
- For bugs affecting LLVM versions < 3.1, you need to use make instead of cmake to build the compilers. (Un)comment the appropriate part in build-compiler.sh to take care of this.
- The virtual machine is set up to build the compilers for most of the bugs, except the following ones: 11964, 11977, 12189, 12855, 12899, 12901 and 13326. Building the compilers for these old bugs indeed requires either to use gcc-4.4, g++-4.4 libraries or to modify the source code of the compilers.

Once the buggy, fixed and warning-laden compilers have been compiled, move them to the appropriate folder so that the run_example_bug.sh script can use them instead of downloading

prebuilt versions from the Internet. For example, for EMI bug #26323, you need to issue the following commands (the first line deletes any prebuilt versions that would have been downloaded earlier by the run_example_bug.sh script):

```
rm -rf /home/user42/compilers/26323/
mv /home/user42/26323/ /home/user42/compilers
```

2.3 Clean up the mess

When you complete the evaluation of this artifact, you can erase any track of it from your computer by deleting the virtual machine and all its associated files. To do so, simply follow these instructions: `https://www.virtualbox.org/manual/UserManual.html#intro-removing`. Using the Ubuntu linux terminal, you can also do it by issuing the following command:

```
VBoxManage unregistervm --delete debian
```
