Static Analysis of C Source Code

Laboratory for the class "Security Verification and Testing" (01TYASM/01TYAOV)
Politecnico di Torino – AY 2021/22
Prof. Riccardo Sisto

prepared by:
Riccardo Sisto (riccardo.sisto@polito.it)

v. 1.0 (22/11/2021)

Contents

Purpose of this laboratory

The purpose of this lab is to make experience with static source code analysis tools for the C/C++ languages. More specifically, two tools will be experimented: a simple lexical scanner (flawfinder) and a more sophisticated commercial static analysis tool (PVS-Studio). As the two tools have not only different features but also different coverage of vulnerabilities, their combined use is recommended. For the installation of the tools, please refer to the gettingStarted_v2.1.2.pdf guide.

All the material necessary for this lab can be found in the course web pages on didattica.polito.it, Materiale Didattico 2021/22 section, 04Lab_StaticAnalysisC folder.

Getting started with flawfinder and PVS-Studio

Before starting with the real exercises, let us make some tests to check the tools are properly set.

Running flawfinder to reproduce some of the results shown in the classroom

Run flawfinder on the CWE121.c file taken from the NIST Juliet Test Suite for C/C++, and check that you get the expected 4 hits that we saw in the classroom (you should get 4 hits if you use version 1.31. If you use later versions you may get a different number of hits).

Getting Started with PVS-Studio

Make sure you have run the following command to install the free academic license:

```
pvs-studio-analyzer credentials PVS-Studio Free FREE-FREE-FREE-FREE
```

Note that, when working with the Labinf VMs, this command has to be entered again after each re-start of the VM. With the Labinf physical machines, instead, the PVS-Studio license should remain stored in your home directory after the first execution of the command.

As we use PVS-Studio from the command line, some bash scripts are provided to simplify running PVS-Studio. They are included in the zip file named pvs-script.zip. Extract this archive in your home directory. The scripts will be copied to your bin directory, which will be created if not yet present. In order to complete the setup add the bin directory to the PATH if not yet included. This can be done by adding the following line to the .bashrc file in your home directory:

```
export PATH=$PATH:[home directory]/bin
```

where [home directory] is your home directory.

The pvs-addcomment script can be used to add the necessary comment to all the .c files in the current directory. The pvs-run script can be used to run PVS-Studio. You must run it with the same command-line arguments that you use for the 'make' command when you compile the program. The report is generated in HTML format (in the htmlreport directory). If you want to change the options used to run PVS-Studio you can edit the pvs-run script. The pvs-clean script makes a cleaning by removing the files generated by PVS-Studio, including the result files. It is called automatically by pvs-run before running PVS-Studio. Finally, pvs-setlicense can be used to set the license again (e.g. in the labinf VM).

Running PVS-Studio to reproduce some of the results shown in the classroom

Run PVS-Studio on the CWE121.c file taken from the NIST Juliet Test Suite for C/C++, by entering the following commands from the CWE121 directory (note that the makefile in this case requires no arguments):

```
pvs-addcomment
pvs-run
```

Check that the analysis proceeds without errors and that you get the html report containing a single entry, as shown in the classroom.

Now, try to run PVS-Studio from the demonstration web site:

```
https://pvs-studio.com/en/pvs-studio/godbolt/
```

Here, you can edit the C code that is in the left hand side text area. When you change the code, you can see the new results on the right hand side. If you prefer, you can open an alternative view, by clicking on 'Edit on Compiler Explorer'. Try to fix the sample code that is displayed and check that the errors reported disappear. The archive of the lab contains a very simple test file that contains a classical format string vulnerability. It is in the test1 directory. Copy the contents of the file and paste it in the left-hand size window, by overwriting the previous code. The format string vulnerability should be pointed out by PVS-Studio. Fix the code and check that PVS-Studio does not report the error after the fix.

1 Static Analysis with Flawfinder and PVS-Studio

1.1 Fixing and Re-analyzing CWE121.c

Fix the vulnerability found in CWE121.c and then re-run the tools (flawfinder and PVS-Studio). What is the result? How can it be explained?

Solution

Flawfinder still reports the error (now, as a new false positive) while PVS-Studio no longer reports it. This result can be explained because flawfinder is just a scanner, which looks at the code token by token, without being able to relate tokens, so it cannot discriminate the correct and the vulnerable versions of the program. Instead, PVS-Studio builds a behavioral model of the program, which is more precise.

1.2 Analyzing other simple examples

Use Flawfinder and PVS-Studio to analyze the other simple examples found in the lab material (test1, test2). For each one of them, run Flawfinder and PVS-Studio. Then, analyze each reported problem and decide if it is a true positive or a false positive. Explain your decision.

Solution

test1.c
flawfinder, line 23: FP (the array cannot overflow)
flawfinder line 25: FP (strncpy is used properly)
flawfinder, line 28 and PVS-Studio, line 30: TP (format string vulnerability if the command line arguments are not trusted)

test2.c
flawfinder, line 46: FP (the destination cannot overflow)
flawfinder, line 47: TP (system takes a command that is not trusted)
flawfinder, line 17: TP (at line 50, buf can overflow with untrusted data)
flawfinder, line 42: FP (the buffer cannot overflow)
flawfinder, line 43: TP (the buffer can overflow at line 48 with untrusted data)
flawfinder, line 45: FP (see line 42)
flawfinder, line 26: FP (read is used correctly)
flawfinder, line 48 and PVS-Studio 50: TP (see line 43)
flawfinder, line 50 and PVS-Studio 52: TP (see line 17)
flawfinder, line 51: FP (buf is 0-terminated)
flawfinder, line 51: FP (buf is 0-terminated)

1.3 Analyzing and Fixing vulnerability CVE-2017-1000249

CVE-2017-1000249 refers to a buffer overflow vulnerability that was found in an implementation of the UNIX file() command. In this exercise we try to find a vulnerability is a real application code. In the material for the lab, you can find a pdf document with the CVE description of the vulnerability and the package with the sources of a version of the software affected by the vulnerability. Read the CVE description and then run flawfinder on the file readelf.c, which is the one containing the vulnerability. Analyse the results of the run and find the vulnerability the CVE refers to. Classify the hits given by flawfinder into true positives (TP) and false positives (FP). How many TP and how many FP did you find? For each one of them, explain the reason for your classification.

Solution

lines 81,100,121,333: FP (the statically sized array is used consistently in the scope of the definition)
lines 391,437: FP (the destination can always hold the source data)
line 535: TP (the number of bytes copied by memcpy, i.e. variable descsz, is not properly checked because the condition (descsz >= 4 || descsz <= 20) is always true; the problem can be fixed by correcting the condition into (descsz >= 4 && descsz <= 20), which guarantees that descsz is less than 20, i.e. that less than 20 bytes are copied)

lines 559,626,656: FP (the destination can always hold the source data)
line 720: FP (the statically sized array is used consistently in the scope of the definition)
line 723: TP (this is a potential buffer overflow because the number of bytes copied by memcpy, i.e. variable descsz, is not checked in the function; depending on how the function is called, this could be a vulnerability)
line 954: FP (the destination can always hold the source data)
line 996: FP (the statically sized array is used consistently in the scope of the definition)
line 1040: FP (the destination can always hold the source data)
lines 1214,1327: FP (the statically sized array is used consistently in the scope of the definition)
lines 1340,1352,1366: FP (the destination can always hold the source data)
lines 1477,1478: FP (the statically sized array is used consistently in the scope of the definition)
line 1578: FP (the statically sized array is not used)
line 1331: FP (read cannot write outside the buffer)
line 1350: TP (as in the code there is no evidence that the string pointed to by p is null-terminated, this could be a vulnerability)

In summary, we have found 3 possible vulnerabilities. The one reported in the CVE is the vulnerability at line 535

Now, try to use PVS-Studio for the analysis of the code. Before being able to compile the code by the 'make' command it is necessary to generate the makefile, by running the following commands (see README.DEVELOPER):

```
autoreconf -f -i
./configure --disable-silent-rules
```

Then, you can check that the code can be compiled by running

```
make -j4
```

another preliminary operation before running PVS-Studio is to insert the two special comment lines at the beginning of each C file. This can be done by means of the pvs-addcomment script, after having moved into the src directory:

```
cd src
pvs-addcomment
```

Finally, PVS-Studio can be run from the main directory by running

```
make clean
pvs-run -j4
```

Note that whenever you want to repeat the analysis you need to clean the project, because PVS-Studio can only analyze the files that are actually compiled (make will automatically avoid the compilation of files if the result of compilation is up to date). Look at the problems reported by PVS-Studio on the readelf.c file. What can we say about the ability of PVS-Studio to find the known vulnerability in this file?

Solution

PVS-Studio did not report this vulnerability (so, technically, it is a false negative for PVS-Studio). However, PVS-Studio reports the cause of the vulnerability, i.e. the error in the boolean expression of the condition that controls the vulnerable memset operation. So, by fixing this error, the vulnerability is also fixed.

Find a fix for the vulnerability and write a patched version of the file.

Solution

The vulnerability reported in the CVE can be fixed by changing the or operator into an and operator (see readelf_fixed.c in the solution folder)