Software Vulnerabilities: Exploitation and Mitigation

—

Lab 9

Prof. Jacques Klein & Pedro Jesús Ruiz Jiménez
(inspired from Prof. Alexandre Bartel's course)

9 Fuzzing (20 P.)

In this lab you will find software vulnerabilities using the AFL fuzzer.

9.1 Setup Labs Environment

9.1.1 Download resources

Download the Debian virtual image here (user:user, password:user; user:root, password:root).

9.2 Launch Emulated Environment

On your host machine, go to the directory where `DebianAFL.qcow2` is located. Then, create the virtual machine by using the following command:

```
$ qemu-system-x86_64 -hda DebianAFL.qcow2 -m 1024 -smp 3
```

9.3 Game Application

You want to fuzz an old but very famous DOS 3D engine, the Build engine. However, there is no port of AFL to DOS, so you decided to find a Unix port of the 3D engine to be able to fuzz the program on your Unix machine.

Download as `user` the source code of the port:

```
 $ svn co svn://svn.icculus.org/buildengine/trunk buildengine
```

Check that the latest modification happened in 2009:

```
 $ cd buildengine
 $ svn info
Path: .
Working Copy Root Path: /home/user/buildengine
URL: svn://svn.icculus.org/buildengine/trunk
```

```
Relative URL: ^/trunk
Repository Root: svn://svn.icculus.org/buildengine
Repository UUID: 93e08484-711e-0410-a0a6-aab9f2357333
Revision: 370
Node Kind: directory
Schedule: normal
Last Changed Author: icculus
Last Changed Rev: 370
Last Changed Date: 2009-04-15 00:21:36 +0200 (Wed, 15 Apr 2009)
```

Compile the program (you might need to install dependencies with apt-get):

```
 $ make
```

We will try to run AFL on the generated `build` program. Create the following directories: `build.afl` and `build.afl/input` and `build.afl/generated` . Copy file `nukeland.map` to `build.afl/input`. Run AFL:

```
#!/bin/bash

TARGET_BIN="/home/user/buildengine/build"

afl-fuzz -i ./build.afl/input/ \
  -f ./build.afl/generated/board.map \
  -o ./build.afl/findings \
  -- $TARGET_BIN ./build.afl/generated/board.map
```

Question 9.1 AFL generates an error message. Explain why the binary needs to be instrumented. 2 P.

For AFL to run, the binary needs to be instrumented. Update the make file according to the following diff:

```
Index: Makefile
===================================================================
--- Makefile    (revision 370)
+++ Makefile    (working copy)
@@ -38,8 +38,8 @@
   SDL_LIB_DIR := please_set_me_cygwin_users
 endif

-CC = gcc
-LINKER = gcc
+CC = afl-gcc
+LINKER = afl-gcc

 #-----------------------------------------------------------------------------#
 # To use a different platform's ASM or portable C, change this.
@@ -77,6 +77,8 @@
   USE_ASM :=
 endif
```

```
+  CFLAGS += -m32
+  LDFLAGS +=-m32 -L/emul/linux/x86/usr/lib
 ifeq ($(strip $(linux64)),true)
   CFLAGS += -m32
   LDFLAGS +=-m32 -L/emul/linux/x86/usr/lib
```

You might need the 32-bit version of some libraries.
Install them as follows (the `su` command will ask you for the `root` password):

```
# su
# dpkg --add-architecture i386
# apt-get update
# apt-get install libsdl1.2-dev:i386
# apt-get install g++-multilib
# apt-get install libstdc++-6-dev:i386
# exit
```

Compile the program again:

```
 $ make clean
 $ make
```

The generated binary should be 32-bit:

```
 $ file build
build: ELF 32-bit LSB pie executable, Intel 80386, version 1
↪   (SYSV), dynamically linked, interpreter /lib/ld-linux.so.2,
↪   for GNU/Linux 3.2.0,
↪   BuildID[sha1]=8900b49cb05c22976f050bbe7883fb62bbd61905, with
↪   debug_info, not stripped
```

At this point you have an instrumented AFL binary. However, AFL still ”fails”.

Question 9.2 Why does AFL still ”fail”? 2 P.

9.4 Updating the Code to Only Test the Map Parser

The code loading a `map` file is method `loadboard` in file `build.c`.

Question 9.3 Update `build.c` to bypass the code initializing the 3D engine. 4 P.

Question 9.4 Update the code to quit the program once the code parsing the map file has finished. 4 P.

9.5 Finding Crashes

Compile the modified program. At this point, we have an AFL-instrumented binary which only parses `map` files and then returns. If AFL detects a crash it probably means that the crash occurred in the parsing code. Launch AFL [1]. It should detect crashes in less than a few minutes.

Question 9.5 Let AFL run for a few minutes. How many crashes did it generate? How many hangs? hang=it times out 2 P.

9.6 Analyzing Crashes

AFL stores input files which generated a crash in `build.afl/findings/crashes/`. Select a single crash.

Question 9.6 Run the modified `build` binary in gdb to analyze the crash. Locate where the crash happens. Identify what part of the input file triggers the crash. Identify the type of bug and/or vulnerability. Identify what can the attacker control (return address, etc.). Is the crash you have selected exploitable or not? If yes, how would you exploit it? Explain everything IN DETAILS. Note that you do not need to write an exploit for the vulnerability. 6 P.

Note on plagiarism

Plagiarism is the misrepresentation of the work of another as your own. It is a serious infraction. Instances of plagiarism or any other cheating will at the very least result in failure of this course. To avoid plagiarism, always cite the source from which you obtained the text.

[1] you should have a number of executions between 1 and 40 per second. If you have 1000 executions per seconds or more it probably means that something is wrong, e.g., maybe the input file is not correctly specified and the program immediately stops at every execution