UNIVERSITÀ DEGLI STUDI DI MILANO

DIPARTIMENTO DI INFORMATICA

Corso di Laurea in
Sicurezza dei Sistemi e delle Reti Informatiche

Security Fuzzing of ADS-B, NextGen protocols and avionics devices

RELATORE
Dott. Valerio Bellandi

TESI DI LAUREA DI
Giulio Ginesi
Matr. 872795

Anno Accademico 2017/2018

Ai miei genitori

Ringraziamenti

Contents

Chapter 1

Introduction

T0D0 1: Short introduction

The rest of the thesis is organized as follows:

T0D0 2:

- *Chapter 2: Software Testing and Fuzzing*
- *Chapter 3: Avionics Protocols*
- *Chapter 4: Experimental Setup*
- *Chapter 5: Results and Conclusions*
- *Chapter 6: Future work*

Chapter 2

Software Testing and Fuzzing

2.1 Software testing methods

T0D0 3: Short introduction on the various testing methodologies beside fuzzing

2.1.1 Black Box

For this metodology no access to the source code or knowledge of the design specifications and internal mechanisms of the program is required. The test is based only on what can be observed which means giving to the target different inputs and monitoring his behaviour. If we reflect on this definition for a moment it is one of the many ways to describe fuzzing. Although the definition of Black Box Testing fits quite well the one of fuzzing they are not the same thing in fact black box testing is widely used and includes all the testing techniques where the only available information comes from the user interaction with the application. One example among all is sql injection where the test is conducted against a plain web page without any previous knowledge about logic of the servers and the various scripts. Only interacting with them allows the user to get an idea of the inner working mechanisms, the interaction can be diect with the user via a manual testing or using automated testing tools for example SQLmap[1]. Black Box testing has many advantages: it does not require access to the source code so it is always applicable and even in the presence of source code it can still be effective. Since it is not source code dependant an effective test case against one program (for example a zip extractor) can be easyly reused to test any program that implements the same functionality. However due to his semplicity this method can only scratch the sourface of the application being tested, acheiving a extremely low code coverage. Moreover complex vulnerabilities which requires multiple attack vetros will never be triggered and can only be discovered with a White Box approach.

[1] http://sqlmap.org/

2.1.2 White Box

This metodology is nothing more than a source code review or analysis, it can be done either by a human or by automated tools. Human analysis in not always feasable and might be inaccurate since, especially in large projects, it is easy to miss one line or make simple mistakes. For this reasons automated source code analysis is the way to go in a white box approach. This method has many advantages mainly regarding code coverage since it is possible to test all the paths for vulnerabilities. However, source code rarely is available for projects outside the open source community and this approach still requires a human interaction to review the results and identify the parts that are actually related to security problems. Moreover the review might require significant time to complete, effectively bringing the performance down to the level of the BlackBox approach.

2.1.3 Gray Box

The definition of this technique is quite trivial because a lot of different methods can be seen as Gray Box approach. This takes the Black Box method as a starting point augmenting it with the use of common reverse engeneering and binary analysis techniques using disassemblers, debuggers and similar tools. The majority of those tools still requires human interaction and reverse engeneering a program can often be a tedious and hard job. However there are some automated tools that aims to facilitate this analysis, for example one of them is Ropper[2]. For this reasons the Gray Box technique is able to efficently improve the coverage produced by the classic Black Box approach still requiring access only to the compiled application. However reverse engeneering is a complex task and requires specific set of skills and tools which might not always be available.

2.2 The fuzzing world

"Fuzzing is the process of sending intentionally invalid data to a product in the hopes of triggering an error condition or fault." [1]

Fuzz Testing dates back to 1989 when it started as a project of the operating systems course at the University of Wisconsin-Madison. At his first stages it was nothing more than a simple black box technique to test the robustness of some UNIX utilities. The first two fuzzers *fuzz* and *ptyjig*, developed by two students, where incredibly successful in finding previously unknown vulnerabilities: they where able to crash around 30% of the considered utilities using the random inputs generated by the fuzzers[2]. It quickly become clear that this was a powerful and strong technique to test the robustness of a piece of software.

During the years fuzz testing evolved and developed gaining popularity and expanding to more and more fields (networks, files, wireless and more) until it became a proper field of research and engineering. As for today this fieeld is quite vast and, using this kind of tools in the software testing phase, is starting

[2]https://github.com/sashs/Ropper

to be a best practice. However it is still uncommon to find fuzzing outside the security world, for this reason is hard to give a precise and unique definition for this technique. I will now try to give an overview of the modern fuzzing world with particular reference to the local fuzzers.

First of all they can be divided in two main categories:

- **Mutation Based (or Dumb)**: apply random mutations to the given input without any knowledge of the data that are being manipulated. Common mutations include bitflipping, shortening or expanding and similar. This method is really simple and is a pure brute force approach, it can be applied to user provided inputs as well as to automatically generated ones to progressively generate a valid input.

- **Generation Based (or Smart)**: requires some knowledge of the data or protocol that is being fuzzed, this information are then used to generate proper inputs. For example generating valid CRC codes for mutated strings or keeping the correct structure for particular files (jpg). Obviously this approach has some advantages but in some cases a mutation based apprach can give better results as it can test programs even outside their functioning boundaries.

As always the best sits in the middle so a fuzzer that combines the two methods is the one that can give the most comprehensive results.

We can then relate fuzzers to the previously mentioned testing methods, in this way fuzzers can be further differentiated by the approach that they have to the problem and the information they require to test the binary. This divides the fuzzing world in three theoretical categories: **BlackBox Fuzzing, WhiteBox Fuzzing** and **GrayBox Fuzzing**. However fuzzing is not an exact science, in the real world choosing the right approach can be trivial and usually there is not a clear distinction between the different categories.

T0D0 4: Fuzzing approaches.

2.3 State of the art

There are many software available to do a moltitude of fuzzing jobs. The choice is vast and goes from the simple hobby project to the more complex and advanced commercial software however,

T0D0 5: choosing a good fuzzer is a crucial step for the success of the task.

The following two softwares represents the state of the art in term of binary fuzzers.

T0D0 6: describe afl, afl-unicorn and peach.

2.3.1 American Fuzzy Lop

We ended up choosing `American Fuzzy Lop` or `AFL` [3] as it represents the state of the art in this category, it has a high rate of vulnerabilities discovery as stated int the "bug-o-rama trophy case" section of the website, it is widely used and in active development.

AFL has many different features that makes it a quite sophisticated software. It is a gray box fuzzer which require access to the source code in order to perform what it calls "instrumentation". This process consist in adding some custom code to the program that is being compiled; this automatic operation, performed by the custom compilers `afl-gcc` and `afl-clang-fast`, adds small code snippets in critical positions without weighting down the program but allowing the fuzzer to obtain information on the execution flow at run time. However, even without access to the source code, it can leverage the QEMU emulator to obtain information about the execution state of the binary though this slows down the fuzzing speed.

The information gathered by the fuzzer will then be used to mutate the input: the tool uses a genetic algorithm approach keeping a queue of interesting inputs that will then be mutated and appending a file to the queue if it triggers a previously unknown internal state. This has been empirically proven to be a successful approach in sintetizing complex files or sentences form scarce or null information[**aflblog**].

One of the best characteristic of afl is that is open source meaning that every person can improve the code and modify it to meet different needs. For example one really active researcher in the fuzzing field, Dr. Marcel Böhme[**mbhome**], created some really interesting variations of afl trying to achieve better code coverage[**aflfast**][**greybf**] and better path discovery[**pythia**].

2.3.2 Peach

[3] http://lcamtuf.coredump.cx/afl/

Chapter 3

Avionics Protocols

T0D0 7: explain the main avionics protocols that will be discussed in the research.

Avionics telecomunications have undergone a major upgrade in the last years which is still in progress and new standards are set to became operative in the next decade. Such changes are made to bring the benefits of modern tecnology onboard of the aircrafts and in the **Air Traffic Management** (**ATM**) world. The classic radar will progressively be sobstituted by other systems based on transponders which require an active partecipation of the aircraft in the tracking procedure, meanwhile the classic radar will be kept as secondary or backup tracking method. Having an active data connection between the ground and the aircraft has many benefits, the most important is the ability to have a bilateral comunication where the ground can receive information on the status of the aircraft and of the flight and the aircraft can receive updated information on the weather, congestions and so on. All this informations, the majority of which are handled automatically by the systems, helps lowering the workload of both the pilots and the controllers while helping and enhancing both jobs.

T0D0 8: acars messages

3.1 OldGen

T0D0 9: explain how 1090mhz protocols works

3.2 NextGen

T0D0 10: explain how 978/nexrad/tis-b/fis-b protocols works

T0D0 11: maybe explain backwards compatibility with 1090 using 1090ES (extendes squitter) also put the theory about using fis-b tis-b in 1090mhz band. (Find some references and obtain some documents FREE)

Chapter 4

Experimental Setup

4.1 Hardware Setup

T0D0 12: Describe the hardware used, why we used that hardware and why it can be usefull to test also on such a hardware. Getting dedicated hw is difficult and expensive. This part is explained really well in the paper. copia and migliora.

4.2 Software Setup

T0D0 13: Explain why we used what we used. Same as before, copy from paper and final report.

4.3 Proceedings

T0D0 14: maybe we don't need this

Chapter 5

Results Analysis And Conclusions

5.1 Caveats

T0D0 15: Put all the problems we had here

5.2 Results

T0D0 16: Put the significative Results here and the state of the fuzzers

5.3 Conclusions

T0D0 17: I can think about doing a separate chapter for the conclusion

Chapter 6

Future Work

T0D0 18: Future work and proprietary fuzzer developement

Bibliography

[1] Pedram Amini Michael Sutton Adam Greene. *Fuzzing: Brute force vulnerability discovery*. Addison Wesley, 2007.

[2] Bryan So Barton P. Miller Lars Fredriksen. "An Empirical Study of the Reliability of UNIX Utilities". In: (1990). URL: `ftp://ftp.cs.wisc.edu/paradyn/technical_papers/fuzz.pdf`.