Metamorphic testing in a Cybersecurity context

Tobias Mattsson

May 7 2023

Abstract

The oracle problem is a well-known problem within software testing. Situations where it is hard or even impossible to know what the expected outputs for some inputs should be are quite frequent, which can make it hard to generate test cases. Since its birth, metamorphic testing has become a popular technique for dealing with this problem, since it compares the output of multiple test cases rather than relying on knowledge of the exact outputs. This testing technique has been employed in plenty of contexts, one of which is cybersecurity.

The purpose of this essay is to explore some real-world applications of metamorphic testing for cybersecurity solutions, which will give insight into how it can be used and to what extent it is useful. The essay covers two specific topics for which this is relevant: 1) detecting obfuscator bugs, and 2) network scanning. Some studies are discussed in terms of how the technique is employed and what the results showed. Lastly, a reflection regarding all of the above will follow.

I certify that generative AI, incl. ChatGPT, has not been used to write this essay. Using generative AI without permission is considered academic misconduct.

Table of contents

1. Introduction

Perhaps one of the key hurdles in the topic of Software Testing is the so-called *oracle problem*. As test input generation has advanced over the years, the need for better oracles which can determine whether the respective outputs are correct has become apparent [1]. This is of particular importance when it comes to automated testing; it has been proven rather difficult to automate suitable oracles for any given test cases since it can be hard to know the expected output of a non-specific input.

The principle of *metamorphic testing* has been around for quite some time, and has since been found to be quite relevant for addressing the aforementioned oracle problem. By utilizing properties of the input data, the need for a test oracle can be bypassed by comparing two separate input instances and their respective outputs [7]. In this way, metamorphic testing is a useful tool for the process of automatically generating test suites.

Not only has metamorphic testing been found useful for things like automating test cases, but its usefulness has also been explored in topics such as *cybersecurity*. This is because tests for security-enhancing features of software may involve the oracle problem becoming very difficult, which can be attributed to aspects such as the use of cryptographic algorithms and other complex functionality [2]. In recent times, metamorphic testing has been cleverly employed to solve various problems in the areas of cybersecurity.

The purpose of this essay is to explore the ways in which metamorphic testing has been applied in the context of cybersecurity in more recent times. First off, the concept of metamorphic testing will be covered in great detail. After this, some real-life applications of metamorphic testing in cybersecurity problems will be brought up.

2. Metamorphic testing

The concept of metamorphic testing was first introduced in a publication from 1998, and was mainly posed as an approach to selecting test cases based on successful test cases with the aim of revealing further software errors [4]. However, it was also found to be useful in scenarios where a test oracle is absent. For this reason, it became a very useful technique for dealing with one of the most prominent hurdles in software testing: the oracle problem. What makes metamorphic very worthwhile compared to other software testing techniques is the fact that it can not only deal with the oracle problem, but also the *reliable test set problem* which is another fundamental problem in software testing [3]. This problem is also relevant on the topic of software testing, as it states that it is difficult to effectively find a subset of test cases (called the *reliable test set*) which determine the correctness of a program. However, the oracle problem remains more interesting in this regard since metamorphic testing was not considered as being a big deal for such a topic initially, yet has found great relevance since few techniques exist for dealing with this problem.

The basis for metamorphic testing is to make use of *metamorphic relations* of the target functions, algorithms etc. These are essentially important properties which determine the relation between multiple input instances and the respective outputs [3]. The premise of this testing approach is to first generate a set of input instances, which may be addressed as the *source test cases*. Given these inputs, a set of *follow-up test cases* is generated by calculating new inputs via the given metamorphic relation/s. Instead of verifying each test case individually, metamorphic testing will compare the source test cases with the follow-up ones to determine whether the metamorphic relations hold in these cases (i.e. the properties correctly apply for the given inputs). In this sense, there is no need for figuring out what the expected output should be for every given input; all that is needed is to gather pairs of test cases and compare the outputs to see if they satisfy the properties in place.

A simple example of a metamorphic relation can be found in areas like trigonometry, for which e.g. the values of different functions follow some properties. Let's say we wish to test the sine function. Consider the trigonometric identity $sin(-\theta) = -sin(\theta)$. Let's say that $sin(\theta)$ is the source test case. In this case, we can generate a follow-up test case where we get the value of $sin(-\theta)$ and compare it to the value of $sin(\theta)$ in the manner described by the identity. Since we know that the given trigonometric identity should hold, we can be sure that the function is correct for an instance where $sin(-\theta) = -sin(\theta)$ for our given input θ. Thus, we are able to test the correctness of the function without knowing what the actual output values are.

3. Applications in Cybersecurity

In this section, a few solutions involving metamorphic relations to problems within the topic of cybersecurity are discussed. First is a solution for finding bugs in various obfuscators, and second is a solution for testing network scanners.

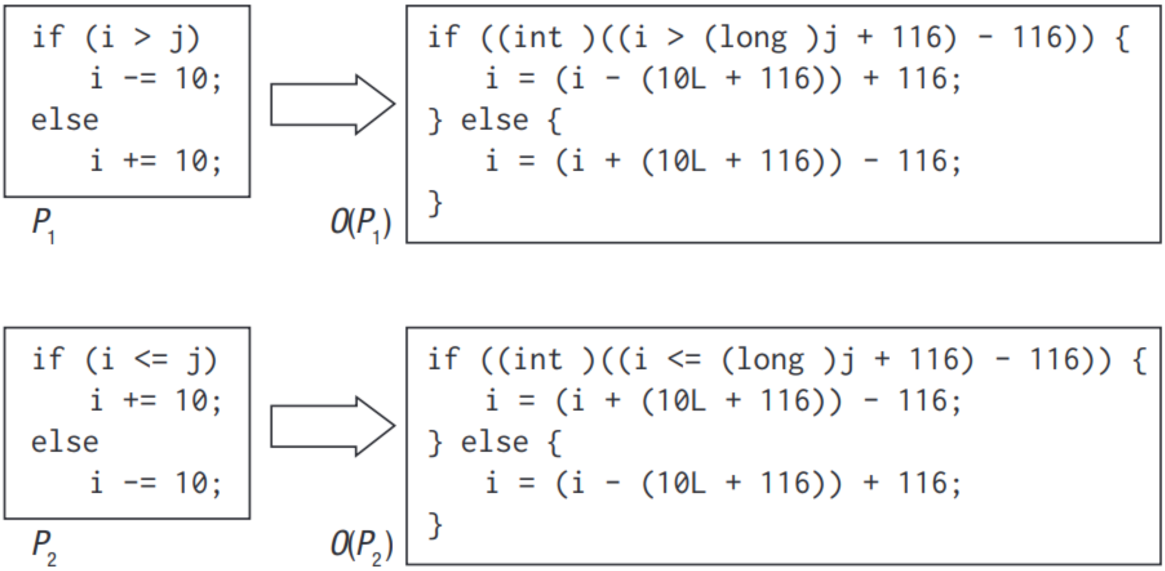
3.1. Detecting obfuscator bugs

When it comes to protecting confidential elements in software, an *obfuscator* is an important tool for ensuring such. Their purpose is to transform code in order to protect the confidentiality of the intellectual property, and have been used in real-world projects by companies such as Google and Facebook [5]. As any other software, however, such a tool may contain bugs of its own which prevent it from functioning as intended in some scenarios. In the case of obfuscators, these bugs can be difficult to locate as regular tests may not be able to replicate them. In a study by Chen et al., some real-world obfuscators are tested on by using metamorphic testing with the intent of finding some of these difficult-to-locate bugs. The obfuscators tested were *Cobfusc*, *CXX-Obfus*, *Tigress* and *Obfuscator-LLVM*.

As explained in section 2, metamorphic testing relies on having defined metamorphic relations for the function or algorithm in question. In the case of an obfuscator program, these relations would be between various executable programs. Chen et al. defined three such relations for the obfuscators under scrutiny, namely the following [2]:

- **MR_1:** If two different programs are functionally equivalent, then their obfuscated versions should also be functionally equivalent.
- **MR_2:** An obfuscator should generate behaviourally equivalent programs for the same input, regardless of execution environment.
- **MR_3:** When obfuscating the same code multiple times, the same obfuscated code should be generated.

With these metamorphic relations in place, all that is needed is a set of initial test cases (in this case, initial programs to test). Chen et al. generated 500 source test cases randomly, whereafter they ran the different obfuscators on them and inspected whether the relations were adhered to. The results obtained in this study indicate that all of these obfuscators contain bugs which prevent all these metamorphic relations from applying to every conceivable test case. What the authors found was that different relations detected different kinds of bugs, which were spread out over all the four obfuscators tested. Figure 1 shows an example test case which fails on MR_1, where two functionally equivalent programs P_1 and P_2 do not result in two functionally equivalent obfuscated versions (which was revealed upon execution) [2].



```
if (i > j)
    i -= 10;
else
    i += 10;
```

```
if ((int )((i > (long )j + 116) - 116)) {
    i = (i - (10L + 116)) + 116;
} else {
    i = (i + (10L + 116)) - 116;
}
```

```
if (i <= j)
    i += 10;
else
    i -= 10;
```

```
if ((int )((i <= (long )j + 116) - 116)) {
    i = (i + (10L + 116)) - 116;
} else {
    i = (i - (10L + 116)) + 116;
}
```

Figure 1: Example test case failing on MR_1. Image source: Chen et al. [2].

3.2. Network scanning

One topic of cybersecurity which could be considered as being greatly significant is *network security*. With attacks happening constantly around the world, disrupting all kinds of networks, the need for detecting potential risks is vital. For this reason, network scanning tools are of great importance as they can scan networks and find security risks in real-time [6]. However, just as for any other software, a network scanner must be tested for correctness so that one can be sure that it works. This has proven to be a rather difficult ordeal, since the network environment changes constantly and since different results may be obtained from scanning the same network in a short period of time.

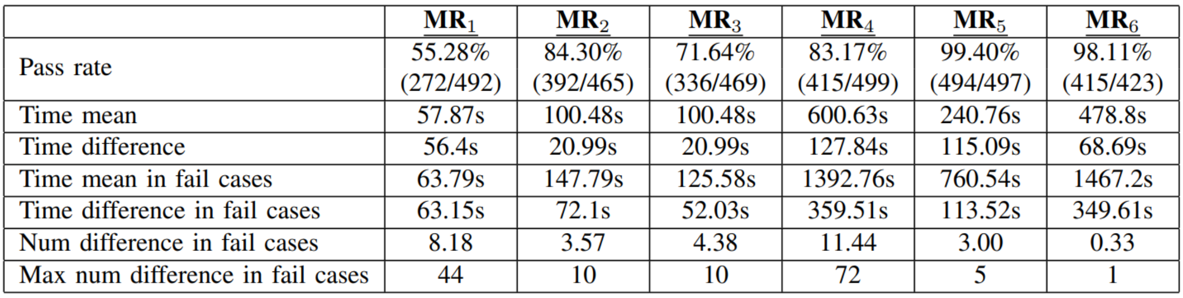
In a study by Zhang et al., metamorphic testing is utilized to evaluate the effectiveness of two network scanners, namely *Nmap* and *Masscan*. They were able to identify seven different metamorphic relations, which were used in the testing process. These were the following:

- **MR_1:** Given one task (t_0) scanning network segments A and B, and then two tasks (t_1 and t_2) scanning A and B respectively, the amount of *up-hosts* found by t_0 should equal the sum of up-hosts found by t_1 and t_2 together.
- **MR_2:** Given one task (t_0) scanning ports x_0-x_i of some network, and another task (t_1) scanning ports x_0-x_i AND x_j-x_n of the same network, the amount of up-hosts found by t_1 should be greater than or equal to the amount t_0 finds.
- **MR_3:** In a similar scenario as that of MR_2, the output of t_0 should be a subset of t_1 (all up-hosts found by t_0 are also found by t_1).
- **MR_4:** If two tasks scan the same network segment, one in order and one at random, they should return the same result.

- **MR_5:** With one task (t_0) scanning a network in TCP SYN ping mode, and another task (t_1) scanning the same network in TCP SYN & ACK mode, the number of hosts found by t_0 should be less than or equal to the amount which t_1 finds.
- **MR_6:** With two tasks performing an OS scan on the same network segment, one in order and one at random, they should detect the same OS for the same host.
- **MR_7:** With two tasks performing a service scan on the same network segment, one in order and one at random, they should detect the same service versions for each port.

With these in place, Zhang et al. implemented a test case generator which produced random network addresses to be used for the test scanning. They then proceeded to test all of MR_1-MR_7 on Nmap, after which they tested MR_1 and MR_2 on Masscan. Table 1 shows the results they obtained with Nmap. What they could conclude from this was that, when scanning two segments, this network scanner has a considerably lower accuracy compared to when only one segment is scanned. The pass rate in the table clearly indicates that this is the case. The test results for Masscan which they gathered indicated that it has a significantly lower pass rate all around for MR_1 and MR_2. This can be explained as being due to this scanner being asynchronous in the sense that it employs a short time-out to speed up the scanning process, but missing a large number of response packets as a result [6].

Table 1: Nmap test results. Image source: Zhang et al. [6].



	MR_1	MR_2	MR_3	MR_4	MR_5	MR_6
Pass rate	55.28% (272/492)	84.30% (392/465)	71.64% (336/469)	83.17% (415/499)	99.40% (494/497)	98.11% (415/423)
Time mean	57.87s	100.48s	100.48s	600.63s	240.76s	478.8s
Time difference	56.4s	20.99s	20.99s	127.84s	115.09s	68.69s
Time mean in fail cases	63.79s	147.79s	125.58s	1392.76s	760.54s	1467.2s
Time difference in fail cases	63.15s	72.1s	52.03s	359.51s	113.52s	349.61s
Num difference in fail cases	8.18	3.57	4.38	11.44	3.00	0.33
Max num difference in fail cases	44	10	10	72	5	1

4. Reflection

Quite clearly, metamorphic testing appears to be a very useful approach for making sure various programs function correctly. Based on the described real-world applications for this technique in the context of cybersecurity, many different kinds of applications and programs could benefit from being tested using it. What's interesting to note is the amount of metamorphic relations one can find for these kinds of tools. It gives a great example of how such can be formulated and utilized for testing the correctness of the respective programs under testing in practice, and it just goes to show how this testing technique can find relevance when testing practically anything.

It can be stated that, simply by being fundamentally different from other, more traditional, testing techniques, metamorphic testing is capable of finding faults that may go unnoticed otherwise. As we could see in the case of detecting obfuscator bugs, this method helped in finding issues which would otherwise go unnoticed and could potentially lead to considerable consequences. Furthermore, it can help in figuring out how effective some tool is at what it does, such as in the case with network scanners.

Of course, one should perhaps not expect too much of metamorphic testing. It may prove to be capable of what other techniques are not, but that does not mean it can do everything. What it does show, however, is the value in testing via different kinds of perspectives in order to detect a more diverse set of faults.

5. Conclusions

Both of the presented applications of metamorphic testing in cybersecurity seem to exemplify the value of such a testing technique. What appears to be a common occurrence is that it is capable of finding various faults or behaviors in different kinds of software which other testing techniques cannot do. When it comes to cybersecurity-focused applications, this is of particular importance since unintentional holes in the security may otherwise be present.

What one can take away from this is that, although metamorphic testing cannot do everything for you, it is an excellent testing technique to employ in order to make your security-based applications as reliable as they can be.

References

[1] Jahangirova, G. (2017, July). Oracle problem in software testing. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis* (pp. 444-447).

[2] Chen, T. Y., Kuo, F. C., Ma, W., Susilo, W., Towey, D., Voas, J., & Zhou, Z. Q. (2016). Metamorphic testing for cybersecurity. *Computer*, *49*(6), 48-55.

[3] Chen, T. Y., Kuo, F. C., Liu, H., Poon, P. L., Towey, D., Tse, T. H., & Zhou, Z. Q. (2018). Metamorphic testing: A review of challenges and opportunities. *ACM Computing Surveys (CSUR)*, *51*(1), 1-27.

[4] Chen, T. Y., Cheung, S. C., & Yiu, S. M. (2020). Metamorphic testing: a new approach for generating next test cases. *arXiv preprint arXiv:2002.12543*.

[5] Velez, M. (2013). Finding and Understanding Bugs in Obfuscators. *UC Davis*.

[6] Zhang, Z., Towey, D., Ying, Z., Zhang, Y., & Zhou, Z. Q. (2021, June). MT4NS: Metamorphic testing for network scanning. In *2021 IEEE/ACM 6th International Workshop on Metamorphic Testing (MET)* (pp. 17-23). IEEE.

[7] Segura, S., Fraser, G., Sanchez, A. B., & Ruiz-Cortes, A. (2016). A Survey on Metamorphic Testing. *IEEE Transactions on Software Engineering*, *42*(9), 805-824.