Metamorphic programming

23RD DECEMBER 2020

BIHANIC BENJAMIN AND NÉRAUD CHRISTOPHE

ENSIMAG - UGA

UNIVERSITY YEAR 2020 - 2021

1 Abstract

This document aims at presenting our research about metamorphic programming, an obfuscation method.

First, we will present obfuscation methods in general, then we will describe how metamorphic programming works. We will give its definition, talk about how it is different from polymorphic programming and give some examples of existing metamorphic viruses. Then, we will illustrate this concept with an example. Finally, we will talk about possible defenses against this kind of attack.

2 Obfuscation

Obfuscation is a process used to make a code or an executable hardly readable and understandable by a human. Code obfuscation is generally used to protect an intellectual property, to make a reverse-engineering analysis more complex or to make a binary difficult to detect by an antivirus by modifying its signature.

The obfuscation may take several forms:

- by encoding strings: for instance, the malware *HerpesNet* encoded all strings with the ROT 13 algorithm, making the string less readable and the program less understandable ;
- by using a packer: a software that compresses and obfuscates a binary. The packer also adds a wrapper which will unobfuscate the binary during the execution. Thanks to the packer, it is really hard to analyse the binary with a static analysis ;
- by adding directives to block the execution of the program if it is executed in a virtual machine or through a debugger: this process makes the dynamic analysis a lot harder ;
- by modifying the binary such that its signature is different: it makes an antivirus detection process harder.

A metamorphic program is a program that is able to change its binary content at each run, but without altering its behaviour. Thus, its signature changes with every execution, which makes it invisible to an antivirus.

An antivirus works by comparing the signature of programs — a hash of the executable for instance — against a database filled with signatures of known viruses. This method cannot work against viruses whose signature changes often.

The concept of metamorphism must be separated from polymorphism. A polymorphic program contains a *polymorphic engine* whose goal is to mutate the code in order to get a **semantically** equivalent program but with a different code [1]. A common way of implementing polymorphism is through encryption: the code is ciphered so that an antivirus would not be able to detect it, and it is unciphered when executed. With this definition, a polymorphic virus is unable to mutate the code of its polymorphic engine.

On the other hand, a metamorphic code uses a *metamorphic engine* that creates a **logically** equivalent code. It operates on its own binary representation, hence it is able to mutate its metamorphic engine as well, making it a lot harder for an antivirus to detect it [2].

Here is a list of some well known metamorphic viruses:

- Zmist: released in 2001, it was the most challenging virus at this time because the concept of polymorphism and metamorphism was totally new [3];
- Zperm: this virus changes the order of its instructions and keeps a logically equivalent code using jump instructions, it was one of the first virus of its kind [4];
- MetaPHOR: mostly known as Win32/Simile, it has the particularity of having a very short encryptor/decryptor compared to the length of the virus's body [5].

3 Metamorphic program

Building a metamorphic program is quite simple. First, the source code adds sequences of assembly instructions doing nothing in the program. Then, when the program is run, those sequences are randomly replaced with a **different** code that does nothing as well. In that way, the signature will be randomized at each run.

In this part, we will explain how to create a metamorphic code with some pseudo-code. The produced metamorphic code is strongly inspired by the first answer of the following StackOverflow post: `https://stackoverflow.com/questions/10113254/metamorphic-code-examples`

First, let's declare a function `junk` that puts some assembly instruction inside the program.

junk()

```
junk():
   add_asm(PUSH, NOP, NOP, NOP, NOP, NOP, NOP, POP)
```

This function adds three different assembly instructions in the code. The first one, PUSH, saves the value of a variable by pushing it onto the stack. The second one, NOP, does nothing and skips to the next instruction. The last one, POP, recovers the value of the variable at the top of the stack.

Then, let's write a function that reads a binary file.

read_file()

```
read_file():
    open_binary_file()
    junk()
    str = read_binary_file()
    junk()
    return str
```

In this function we can notice several calls to the function `junk`. Those calls do not change the expected behaviour — opening and reading a file — as they simply add assembly code that does not interfere with the opening and reading process. Those instructions will be randomly modified later.

Now, let's write a simple function that writes a binary code.

write_file()

```
write_file(str):
    open_binary_file()
    junk()
    write_binary_file(str)
```

Let's now modify the binary with the following function.

metamorphic()

```
metamorphic(str):
    junk()
    for_each junk in str:
        junk()
        randomly_modify_NOP(str)
```

This function looks for all assembly instructions added by the function `junk()` and randomly modifies the NOP instructions. The instructions PUSH and POP will by default save and load the registers $EAX or $RAX (in 32 or 64 bits architectures respectively), so the randomly added instructions will be operations on this register, such as adding or subtracting a random value. The add instructions have to be of the same length as the one in the `junk()` in order not to invalidate the binary code by changing all the offsets.

Finally we can write the main function.

main()

```
meta():
    read_file()
    metamorphic()
    write_file()
```

This pseudo-code shows how a metamorphic program works: during each run, it reads the content of the targeted executable, then it replaces all the junk instructions by random instructions that do not break the behaviour of the code but still change its signature, and writes it to a new executable.

Here is a GitHub link to a real and simple implementation that we made of this metamorphic program in modern C++:

`https://github.com/Doyko/metamorphic`

4 Defenses

Even though the job of an antivirus is harder on a metamorphic virus, there are still some other techniques that allow to detect those.

For instance, metamorphic code does not offer protection against heuristic analysis. This kind of analysis operates by executing the targeted program in a virtual sandbox environment, and analysing its behaviour [6]. The antivirus then looks for suspicious operations such as replication or attempted access to sensible files, which are usually performed by malwares. A metamorphic virus might then be detected by an antivirus, as it uses replication-like mechanisms.

Moreover, a metamorphic virus will need to analyze its own structure in order to apply the randomization mechanisms. An antivirus could then look for such instructions.

As a metamorphic virus will usually appear unknown to an antivirus, artificial intelligence based detection methods are being developed as well in order to detect a new virus as fast as possible.

An article from Moustafa Saleh suggests a statistical solution for detecting metamorphic codes, but it needs some beforehand preparation [7]. The basic idea is, for each known metamorphic virus, we start by making a database of some of its replications. Then, we organize it using a mathematical model based on the study of the eigenvalues of the code. Then, an unknown input file would be compared with the already known data, and if its behaviour is similar enough, it would be classified as a virus. This method is not perfect as it requires samples of already known metamorphic viruses, but it allows to detect a metamorphic virus we already know quite efficiently.

Finally, it is worth mentioning that those kind of viruses are able to propagate thanks to users' lack of knowledge on security. The vast majority of viruses are transmitted through e-mail, as fake Word attachments, rogue ZIP files or simple phishing for instance. It is thus important for a company to raise awareness amongst its employees about these methods, as well as implementing access control policies in order to isolate the most sensible parts of their infrastructure.

5 Conclusion

Metamorphic programming has since a few years become a really challenging field of study. Unlike classical self-replicating malwares and unlike polymorphic codes, metamorphic viruses are able to fully mutate each part of their code, which makes the detection process for antiviruses really difficult.

However, it is still possible to detect those viruses to a certain extent. Heuristic analysis or statistical study of the most well known metamorphic codes gives a lot of information about their behaviour and helps finding new ways to classify them.

Although one could expect that current progress in the field of artificial intelligence would lead to the emergence of new methods of detection, it is nonetheless essential to continue raising awareness amongst end users to ensure they are less likely to open an obviously infected file.

Références

[1] M. Rouse, "Metamorphic virus," April 2018. [Online]. Available: https://searchsecurity.techtarget.com/definition/Metamorphic-virus

[2] T. M. Driller, "Metamorphism in practice or "how i made metaphor and what i've learnt"," February 2002. [Online]. Available: https://web.archive.org/web/20070602061547/http://vx.netlux.org/lib/vmd01.html

[3] P. Szor, "The art of computer virus research and defense," *Symantec Press*, pp. 278–280, 2005. [Online]. Available: http://index-of.es/Viruses/T/The%20Art%20of%20Computer%20Virus%20Research%20and%20Defense.pdf

[4] "Z0mbie's home page." [Online]. Available: https://z0mbie.dreamhosters.com

[5] "W32/etap-a," *Sophos*, February 2013. [Online]. Available: https://www.sophos.com/en-us/threat-center/threat-analyses/viruses-and-spyware/W32~Etap-A.aspx

[6] W. Wong and M. Stamp, "Hunting for metamorphic engines," *Journal in Computer Virology*, 2006.

[7] M. Saleh, "Towards metamorphic viruses recognition using eigenviruses," July 2011. [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1206/1206.5871.pdf