Using Formal Grammar and Genetic Operators to Evolve Malware

Sadia Noreen[1], Shafaq Murtaza[1], M. Zubair Shafiq[2], Muddassar Farooq[2]

[1] FAST National University, Islamabad, Pakistan
[2] nexGIN RC, FAST National University, Islamabad, Pakistan
{sadia.noreen,shafaq.murtaza}@nu.edu.pk,{zubair.shafiq,muddassar.farooq}@nexginrc.org

Abstract. In this paper, we leverage the concepts of *formal grammar* and *genetic operators* to evolve malware. As a case study, we take *COM infectors* and design their formal grammar with production rules in the BNF form. The chromosome (abstract representation) of an infector consists of genes (production rules). The code generator uses these production rules to derive the source code. The standard genetic operators – crossover and mutation – are applied to evolve population. The results of our experiments show that the evolved population contains a significant proportion of valid COM infectors. Moreover, approximately 7% of the evolved malware evade detection by COTS anti-virus software.

1 Evolutionary Malware Engine: an Empirical Study

Malware writers have developed *malware engines* which create different variants of a given malware – mostly by applying packing techniques. The developed variants essentially have the same functionality and semantics. In contrast, our methodology targets to create "new" malware. It consists of three phases: (1) design a formal grammar for malware and use it to create an abstract representation, (2) use standard genetic operators – crossover and mutation, and (3) generate assembly code from the evolved abstract representation.

The working principle of the proposed COM infector evolution framework is shown in Fig. 1. In the first step, it analyzes the source code of an infector and maps it to the production rules – defined in the formal grammar – to generate its chromosome. This step is initially done for 10 infectors (source code is obtained from [1]); resulting in a population of 10 chromosomes. We then apply genetic operators – crossover and mutation – to the population. Intuitively speaking, all individuals will not be legitimate infectors after genetic operators are applied. To test this hypothesis, we have a code generation unit which accepts these chromosomes and produces assembly code for them. Finally, we present the evolved malware to well-known COTS anti-virus products to check if the evolved infectors can evade detection.

We have observed that the evolved infectors fall into one of the three categories: (1) COM infectors which have turned benign, (2) COM infectors which are detected by anti-virus but as a different type than that of initial 10 infectors, and (3) unknown variants of COM infectors which have successfully evaded the

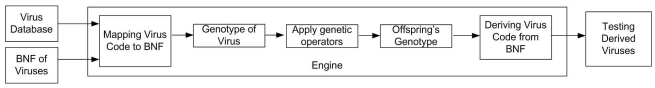

Fig. 1. Architecture of COM infector evolution framework

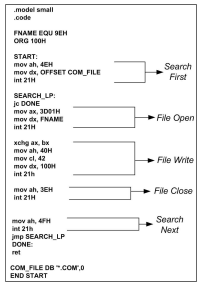

Fig. 2. Code of mini44

Fig. 3. BNF of COM infectors

detection mechanism. We manually execute the last category of the infectors on Windows XP machine to check if the evolved infectors truly do the damage. Our initial findings show that about 52% of evolved infectors have become benign; 41% are detected but with new names that are not included in the initial population; while remaining 7% still do their destructive job but remain undetected. The last category of infectors have achieved stealthiness in the true sense.

We now take an example of a simple `mini44` malware (see Fig. 2) to explain the evolution procedure. The common routines – Search First, Copy, Search Next – are labeled in Fig. 2. *Search First* routine searches for the first COM file in the current directory and it then opens it. After opening the file, the malware writes its code into the victim file and the file is closed. The next victim COM file is searched in *Search Next* function. Once our engine will read instruction *mov ah, 4EH* of `mini44`, it will lookup for the production rules that match with this instruction. The production rules are given in Fig. 3. The genotype of the instruction *mov ah, 4EH* may consist of following production rules: 1-2-7-8-9-10-11-12-16-15-22. In a similar fashion, the genotype of each insturction/routine in COM infector is generated. When we want to produce a new individual, we take abstract representation of two infectors and use crossover and mutation operators to evolve new individuals. Finally, the code generator does the reverse mapping to generate the source code of the evolved infector.

References

1. Virus Source Code Database, VSCDB, available at `http://www.totallygeek.com/vscdb/`.
2. S. Noreen, S. Murtaza, M.Z. Shafiq, M. Farooq, "Evolvable Malware", Genetic and Evolutionary Computation Conference (GECCO), ACM Press, 2009.
3. E. Filiol, "Metamorphism, Formal Grammars and Undecidable Code Mutation", International Journal of Computer Science, 2(1), pp. 70-75, 2007.