GRAMMAR–BASED MALICIOUS CODE DETECTION CONCEPT

Ota Jirák
Doctoral Degree Programme (4), FIT BUT
E-mail: xjirak03@stud.fit.vutbr.cz

Supervised by: Dušan Kolář
E-mail: kolar@fit.vutbr.cz

Abstract: This paper describes two approaches of malware detection system based on scattered context grammars. The first solution uses the control flow graphs and a high-order language structure detection. The second concept uses the binary data as an input of the scattered context grammar compiler. This solution is based on pattern recognition.

Keywords: control flow graph, scattered context grammar, detection, decompiler, malware

1 INTRODUCTION

This paper describes a concept of grammar–based malicious code detection. There are two possible approaches. In the first part, an approach called Per Partes is described. This solution uses a sequence of data modifiers. In the second part, a solution called All-In-One is described. This approach uses the binary data as a compiler input. Finally, the comparison of both approaches is proposed.

2 PER PARTES

This solution has a straightforward approach to analyze the input data. It is composed from several parts:

- decompiler,
- control flow graph retrieval,
- high-order language structure detection,
- cfg normalization and serialization, and
- SCG compiler.

The relations between these parts are illustrated in Figure 1.

2.1 DECOMPILER

The input of the whole system is a file with PECOFF structure [1]. This structure is commonly used for files with various extensions (e.g. EXE, DLL, OCX, and others). There are two possible ways of assembler code retrieval:

- usage of an existing disassembler, or
- creation of a new, proprietary disassembler.

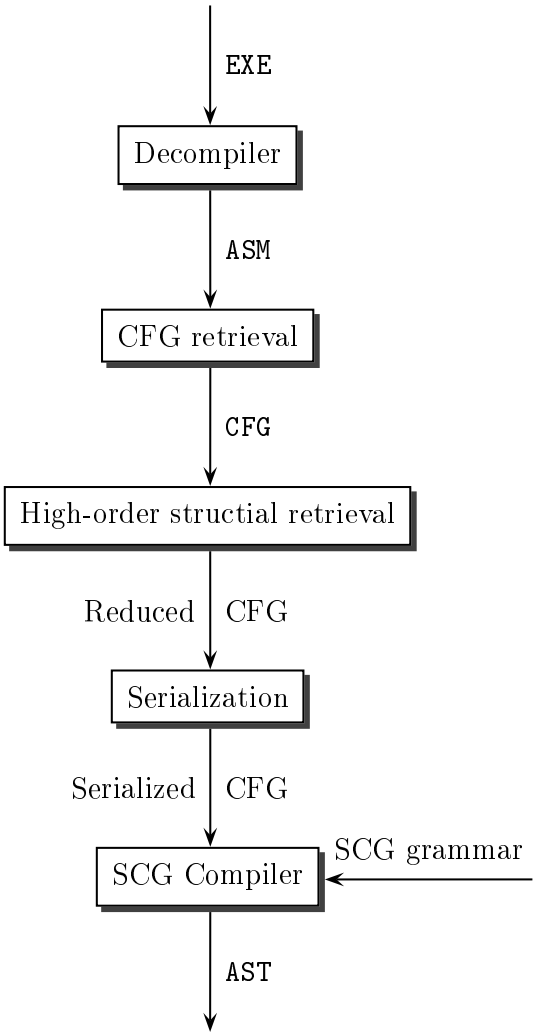

Figure 1: System structure - Per Partes

Every approach has its pros and cons. An advantage of the existing disassembler (e.g. OllyDbg, or IDA [4, 6]) is usage of well-established application.

On the other hand, the own disassembler is able to return a structure which can be used as an input for the other transformations. The custom disassembler can be modified to detect only jump instructions. For other instructions, it is necessary to only know the size of parameters. It is possible to skip an uninteresting code and speed the process of disassembling up.

For these reasons, the usage of the own implemented disassembler was chosen [7].

As an output of this disassembler, we get an assembler code corresponding to a given binary code.

2.2 Control Flow Graph retrieval

Control flow graph (CFG, more about CFG and CFG retrieval in [3]) is a directed graph in which the nodes represent basic blocks and the edges represent control flow paths.

The assembler code contains the information for CFG retrieving (e.g. target addresses of jumps, conditional jumps, etc.). From the definition of CFG, these instructions split the whole code into the interconnected basic blocks.

2.3 GRAPH REDUCTION

In the CFG, there are several recognizable patterns (e.g. cycles, conditions). A principle of the reduction is illustrated in Figure 2.

The process of the reduction has two phases:

- detection, and
- replacement.

The reduction algorithm investigates a given CFG and detects possible starts of some pattern. After that, it continues with a matching process (investigates related nodes).

When the pattern is found out, it is replaced by one node. This node has the name of the found pattern as an attribute. For more information see [5].

As an output of this reduction process is a reduced control flow graph.

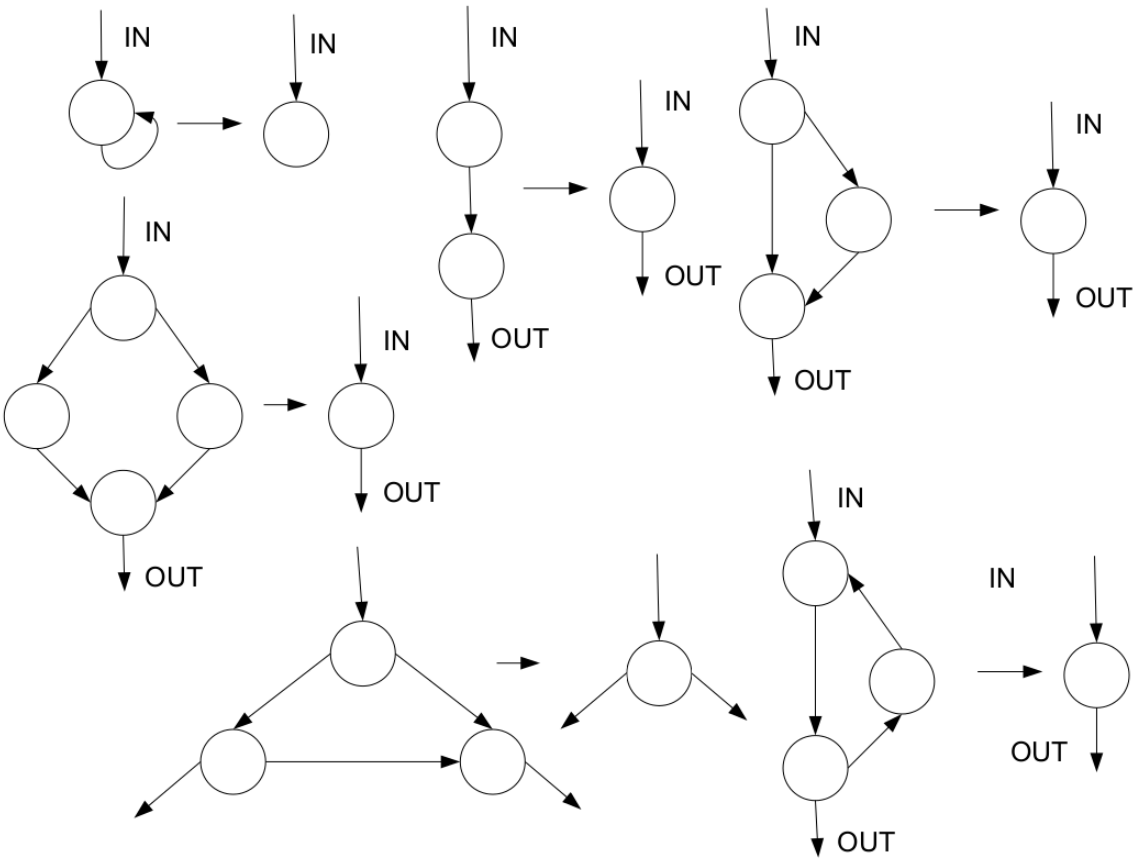

Figure 2: An illustration of reduction[5]

2.4 SERIALIZATION AND NORMALIZATION

It is easier to analyze uniformly described data. Therefore, a normalization and serialization are used.

The removing/replacing of patterns (CFG reduction) simplifies the process of serialization. An output of this process is a cfg serialized into one string.

2.5 GRAMMAR OF JEOPARDY

The malicious code is usually scattered over the whole code. From this fact, the usage of the scattered context grammars (SCG, see [2]) seems to be a natural approach. The SCG is able to describe context dependency.

The serialized cfg is given as an input string of scattered context grammar parser.

This part, and the whole system actually, returns an abstract syntax tree for the given malware grammar.

When the compiler returns a nonempty abstract syntax tree, there is something suspicious in the analyzed input. The existence of an abstract syntax tree means the existence of some patterns in a malware grammar.

3 ALL-IN-ONE

A structure of the All-In-One solution is illustrated in Figure 3. This solution is based on similar principle as commonly used pattern matching. These patterns are described by a malware scattered context grammar.

The malware grammar and binary data in the PECOFF format are given as inputs of the SCG compiler. A result of this system is an abstract syntax tree for the given grammar and input data. A nonempty abstract syntax tree indicates the existence of suspicious code.

This approach avoids several steps of conversion used in Per Partes solution. Conversions from Per Partes solution causes a loss of information contained in input data. Cfg retrieval produces static analysis. Static approach has several limitations, e.g. a target of jump can be given through a register. Problem is with exploration of all possible targets of these jumps.

One binary data can be interpreted as a parameter or an instruction. The only difference is a starting point of binary data interpretation.

This approach looks for predefined binary patterns. Therefore, it can skip unimportant input data without detailed analysis. A SCG is able to detect occurrence of several patterns in input code in the same time (there is one part and somewhere is the other). A SCG description of the exact target location is problematic (simulation of address space with grammar).

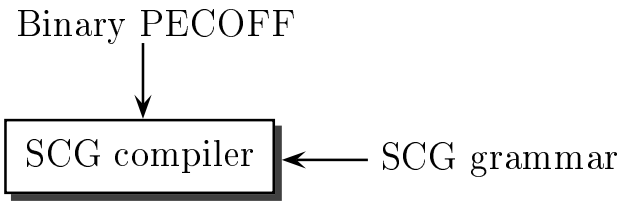

Figure 3: System structure - All-In-One solution

4 CONCLUSION AND FUTURE WORK

The first approach is easier to be used by programmer (human readable code of the scattered context grammar). The disadvantage of this approach is the possible loss of some information during conversion between several formats (binary $\rightarrow$ asm $\rightarrow \cdots \rightarrow$ abstract syntax tree). On the other hand, we can compare control flow structures (possible detection of the same structure described by different instructions).

The advantage of the second approach is its compactness. This approach avoids loss of information during conversion but it does not get information about similarity of control flow. The disadvantage is a human unreadable grammar definition (it works with binary code) caused by compactness.

Overall, the first solution gives us more advantages than the second one. The future work will consist of the implementation and improvement of the detection part of the Per Partes system. A possibility of a combination of the both approaches will be studied.

ACKNOWLEDGEMENT

This work was partially supported by the Czech Ministry of Education, Youth and Sports grant MŠMT 2C06008 "Virtual Laboratory of Microprocessor Technology Application" (visit `http://www.vlam.cz`), and the research plan "Security–Oriented Research in Information Technology", MSM 0021630528.

REFERENCES

[1] Microsoft PE and COFF Specification, `http://www.microsoft.com/whdc/system/platform/firmware/PECOFF.mspx` [cited 2011-03-03]

[2] Greibach, S., A., and Hopcroft, J., E.: Scattered context grammars. J. Comput. Syst. Sc., 3(3):233-247, 1969.

[3] Theiling, H.: Extracting safe and precise control flow from binaries, Real-Time Computing Systems and Applications, 2000. Proceedings. Seventh International Conference on, pp. 23-30, 2000.

[4] OllyDbg, `http://www.ollydbg.de/` [cited 2011-03-03]

[5] Ota Jirák and Dušan Kolář. Control Flow Graph Retrieval and Analysis via Simulation, In: MOSIS'08, Ostrava, CZ, MARQ, 2008, p.67–74, ISBN 978-8086840-40-6

[6] IDA Pro Disassembler and Debugger, `http://www.hex-rays.com/idapro` [cited 2011-03-02].

[7] Jirák, O.: Windows PE Transformation into control flow graph, Brno, FIT BUT Brno, 2007.