Control Flow Graphs as Malware Signatures

Guillaume Bonfante, Matthieu Kaczmarek and Jean-Yves Marion
Nancy-Université - Loria - INPL - Ecole Nationale Supérieure des Mines de Nancy
B.P. 239, 54506 Vandœuvre-lès-Nancy Cédex, France

Abstract

This study proposes a malware detection strategy based on control flow graphs. It carries out experiments to evaluates the false-positive ratios of the proposed methods. Moreover, it presents some insight to establish detection methods sound with respect to some obfuscation techniques.

Introduction

In this paper, we propose a detection strategy based on *control flow graphs (CFGs)*. More precisely, we show how flow graphs can be used as signatures. This is the first step of a broader semantics based approach to detection. For the last twenty years, commercial software has treated this problem with the help of detection strategies based on string signatures. This method requires a database of signatures continuously updated. To maintain the database, some experts analyse every new malicious programs and try to reverse engineer a corresponding signature, with the constraint to produce as least as possible false-positives. The complexity of such an analysis is illustrated by [7]. The amount of time and manpower induced by this strategy is now challenged by advanced obfuscation techniques [5, 1, 2].

In order to enhance this strategy, current detection research interest in detecting semantic aspects. Different approaches have been proposed. In [14, 15], assembler instructions are specified in an equational logic and a prover is used to identify malicious behaviour. The methods exposed in [4, 6] are based on abstract interpretation, the considered signatures are built on control flow graphs and on traces of system states. In [13, 10], CFGs and data flow are used to detect malware.

This study follows the same vein. It carryies out some experiments using CFGs as signatures. This detection strategy is motivated by the fact that the domain of CFGs is at least as complex as the domain of strings. That is for any integer n, there are more than 2^n differents CFGs composed of n nodes. It is worth to mention that the CFG extraction is fully automatic.

This paper goes through the following roadmap. The Sect. 1 defines some notions about detection and places this work in a sound theoretical context. The Sect. 2 describes the CFGs induced by `x86` assembler which is the architecture target by our experiment. The experiments are presented in the Sect. 3. Finally, we question some problems opened by this work in the Sect. 4.

1 Detection

Detectors. We consider the detection notion defined in [6]. Let $\mathbf{P}$ be the set of programs and $\mathbf{M} \subset \mathbf{P}$ the set of malicious programs. We associate $\mathbf{M}$ to a set of signatures $\mathbb{M}$. A detector on $\mathbb{M}$ is a function $D : \mathbf{P} \times \mathbb{M} \to \{0, 1\}$. Then we say that a program $\mathbf{p}$ is detected if there is a signature $m \in \mathbb{M}$ such that $D(\mathbf{p}, m) = 1$.

Applying this definition on the classical signature scanning method, the set $\mathbb{M}$ would consist in the list of all known malware string signatures and the associated detector would be defined by $D(\mathbf{p}, m) = 1$ if m is a substring of $\mathbf{p}$ and $D(\mathbf{p}, m) = 0$ otherwise.

Evaluation. A detector aims at the discrimination of sane programs from malware. As a result a detector can only be evaluated w.r.t a set of sane programs $\mathbf{S} \subset \mathbf{P}$ and set of malicious programs $\mathbf{M} \subset \mathbf{P}$. We suppose that $\mathbf{M} \cap \mathbf{S} = \emptyset$. We define w.r.t D, $\mathbf{M}$ and $\mathbf{S}$ *the set of false-positives* $\mathbf{F}^+ = \{\mathbf{p} \mid \mathbf{p} \in \mathbf{M} \wedge \forall m \in \mathbb{M} : D(\mathbf{p}, m) = 0\}$. We associate to $\mathbf{F}^+$ the false-positive ratio defined by the ratio between the cardinals of $\mathbf{F}^+$ and of $\mathbf{S}$.

2 CFGs in assembler `x86`

Description. A *control flow graph (CFG)* is composed of linked nodes. We define six kinds of node `jmp`, `jcc`, `call`, `ret`, `inst` and `end` which correspond to the control flow structure. First, `x86` assembler programs have four kinds of flow instruction: non-conditional jumps (`jmp`), conditional jumps (`jcc`), function calls (`call`) and function returns (`ret`). Then, we abstract any contiguous sequence of instructions without flow modification as a node of kind `inst`. Finally, the kind `end` corresponds to the end of the program.

This induces the following properties on the graph. Any CFG has a unique node of kind `end`. Nodes of kind `jmp`, `call` and `inst` have only one successor. Nodes of kind `jcc` have exactly two successors. Nodes of kind `call` and `ret` are well parenthesised. If a node of kind `ret` has no corresponding node of kind `call` then it is linked to the node of kind `end`. We present two examples of CFG extraction in Figure 1.

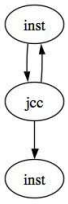
We use the following structure to represent nodes.

```
typedef struct node_t{
  add_t node_add;
  add_t ret_add;
  kind_t kind;
  node_t *children;
} node_t;
```

Reduction. While experimenting, we have observed that nodes of kind `jmp` and `inst` do not really provide information about the control flow. Indeed, they have a unique successor and they do not influence the successing flow – as do `call` nodes. Then, we propose to also experiment with reduced graph.

We propose the following graph reduction. For any node of kind `inst` or `jmp` remove the node from the graph and link all its predecessors to its unique successor. The Figure 2 presents a CFG and its reduction.

```
0x1288 add edi, 0x4
0x128b mul dword [0x4056c9]
0x1291 mov [edi], eax
0x1293 inc dword [0x4056c5]
0x1299 cmp dword [0x4056c5], 0x270
0x12a3 jnz 0x1288
0x12a5 pop edi
```




```
0x12d1 call 0x126f
0x12d5 mov dword [ebp−0x4], 0x0
0x126f push ebp
0x1270 mov ebp, esp
0x1272 push edi
0x1273 lea edi, [0x405814]
0x1279 mov eax, [ebp+0x8]
0x127c mov [edi], eax
0x127e mov dword [0x4056c5], 0x1
0x1288 add edi, 0x4
0x128b mul dword [0x4056c9]
0x1291 mov [edi], eax
0x1293 inc dword [0x4056c5]
0x1299 cmp dword [0x4056c5], 0x270
0x12a3 jnz 0x1288
0x12a5 pop edi
0x12a6 ret
```

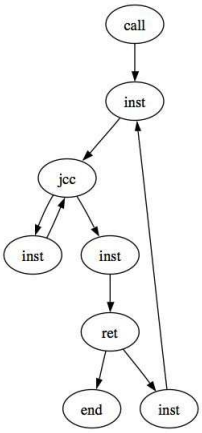


Figure 1: CFG extraction

3 Experiments

Methodology. These experiments examines CFGs as signatures for malware detection. Our experiment target `win32 x86` architecture. A collection of 750 programs has been gathered and the `ClamAV` scanner has been used to check that they are sane programs. Then, a collection of 2278 has been collected from public sources. CFGs have been automatically extracted from malware and constitute the signatures. Finally, two algorithm detection have been implemented, we evaluates their false-positive ratios.

Detection based on CFGs. The first algorithm computes the detector D defined by $D(\mathbf{p}, m) = 1$ if m if m is isomorphic to the CFG of $\mathbf{p}$ and $D(\mathbf{p}, m) = 0$ otherwise. Informally speaking, the detector D search for an exact malware CFG. We divide our collections of sane programs and malware in four category depending on the size of the CFGs. In the first category the CFGs of the programs have less that 50 nodes, in the second they have between 50 and 100 nodes, in the third they have between 100 and 5000 nodes, and in last category they have more that 5000 nodes. We interest in false-positives for each category. The Table 1 presents the results.

We observe that the false-positive ratio decrease w.r.t the size of CFGs. Under 100 nodes the CFG is not reliable to discriminate malware from sane programs, but over 100 nodes the results are much better. This is really encour-

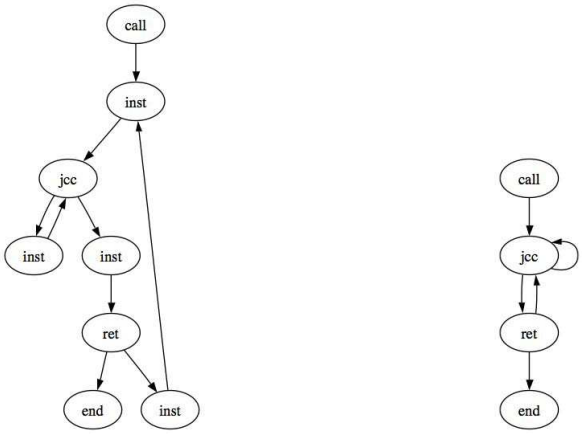


Figure 2: CFG reduction

Size of CFGs	< 50	$50 - 100$	$100 - 5000$	> 5000	Overall
Sane programs (**S**)	15	51	383	301	750
Malware (**M**)	440	635	1033	170	2278
False-positives ($\mathbf{F}^+$)	10	15	7	1	33
Ratios	66.7%	29.4%	1.8%	0.3%	4.4%

Table 1: Results of the experiments

aging because the detection accuracy increase with the size of programs where classical signature scanning has more false-positives when the size increases. As a result, CFG detection could enhance classical detection and it would be interesting to associate the two methods.

Detection based on reduced CFGs. The second method uses the detector R defined by $R(\mathbf{p}, m) = 1$ if m is the reduction of the CFG of $\mathbf{p}$ and $R(\mathbf{p}, m) = 0$ otherwise. In other terms, the detector R does not consider flow without branch. We do the same experiments as previously considering reduced CFGs. The Table 2 presents the results.

Size of CFGs	< 50	$50 - 100$	$100 - 5000$	> 5000	Overall
Sane programs (**S**)	74	39	528	109	750
Malware (**M**)	1138	228	850	62	2278
False-positives ($\mathbf{F}^+$)	26	0	8	0	34
Ratios	35.1%	0.0%	1.5%	0.0%	4.5%

Table 2: Results of the experiments with reduction

This detection method add one new false-positive, but it has the advantage to group most of false-positives under 50 nodes. Over 50 nodes the accuracy is good. This experiment corroborates the hypothesis that nodes of kind `inst` and `jmp` do not provide information about control flow.

Moreover, using reduced graph, the detection becomes sound w.r.t some obfuscation techniques such as code reordering. By code reordering, we consider

the technique that consists in three steps. Split the program in blocks, permutes them and link the block with jumps to recover the original flow.

Comparison. For comparison, statistical methods used in [9] induce false-positive ratios between $0,5\%$ and 34%. A detector based on artificial neural networks developed at IBM [12] presents a false-positive ratio lower that 1%. The data mining methods surveyed in [11] present false-positive ratios between $2,2\%$ and $47,5\%$. Heuristics methods from antivirus industry tested in [8] present false-positive ratios lower than $0,2\%$.

4 Further Research

We have not considered host infection. In our framework, this detection would be associated to sub-graph search. In [3], such a study has been carried out for the virus `MetaPHOR`. The exposed results are promising.

It would be interesting to face our methods to metamorphic engines in order to evaluate the soundness w.r.t program obfuscation. We already know that the detector R is sound w.r.t code permutation, but it would be of worth to study the obfuscation techniques described in [11, 1, 2].

We should also consider the specific problem of packed and encrypted malware. In our study the signature of packed malware only target the decompression procedure. As a result if the used packer is public, such as `UPX`, this cause false-positive.

References

[1] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. On the (im) possibility of obfuscating programs. *Lecture Notes in Computer Science*, 2139:19–23, 2001.

[2] P. Beaucamps and E. Filiol. On the possibility of practically obfuscating programs towards a unified perspective of code protection. *Journal in Computer Virology*, 3(1):3–21, April 2007.

[3] D. Bruschi, Martignoni, L., and M. Monga. Detecting self-mutating malware using control-flow graph matching. Technical report, Universit degli Studi di Milano, September 2006.

[4] M. Christodorescu, S. Jha, S. Seshia, D. Song, and R. Bryant. Semantics-aware malware detection. *IEEE Symposium on Security and Privacy*, 2005.

[5] C. Collberg, C. Thomborson, and D. Low. A taxonomy of obfuscating transformations. *University of Auckland Technical Report*, 170, 1997.

[6] M. Dalla Preda, M. Christodorescu, S. Jha, and S. Debray. A Semantics-Based Approach to Malware Detection. In *POPL'07*, 2007.

[7] P. Ferrie and P. Ször. Hunting for metamorphism. *Virus Bulletin*, march 2001.

[8] D. Gryaznov. Scanners of the Year 2000: Heuristics. *Proceedings of the 5th International Virus Bulletin*, 1999.

[9] J. Kephart and W. Arnold. Automatic Extraction of Computer Virus Signatures. *4th Virus Bulletin International Conference*, pages 178–184, 1994.

[10] A. Lakhotia and M. Mohammed. Imposing Order on Program Statements to Assist Anti-Virus Scanners. *Proceedings of Eleventh Working Conference on Reverse Engineering, IEEE Computer Society Press*, 2004.

[11] M. Schultz, E. Eskin, E. Zadok, and S. Stolfo. Data Mining Methods for Detection of New Malicious Executables. *Proceedings of the IEEE Symposium on Security and Privacy*, page 38, 2001.

[12] G. Tesauro, J. Kephart, and G. Sorkin. Neural networks for computer virus recognition. *Expert, IEEE [see also IEEE Intelligent Systems and Their Applications]*, 11(4):5–6, 1996.

[13] M. Venable, M. Chouchane, M. Karim, and A. Lakhotia. Analyzing Memory Accesses in Obfuscated x86 Executables. *Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, pages 1–18, 2005.

[14] M. Webster. Algebraic Specification of Computer Viruses and Their Environments. *Selected Papers from the First Conference on Algebra and Coalgebra in Computer Science Young Researchers Workshop*, pages 99–113, 2005.

[15] M. Webster and G. Malcolm. Detection of metamorphic computer viruses using algebraic specification. *Journal in Computer Virology*, 2(3):149–161, 2006.