Mining malware specifications through static reachability analysis

Hugo Daniel Macedo[1] Tayssir Touili[2]

[1]INRIA Rocqencourt

[2]LIAFA Univ. Paris 7

November 4, 2013

Motivation

Our goal: Malware detection!

Why? Social impact!

- Malware in the news!
- We are all collateral damage!

Huge technological challenge!

- 286 million new malware variants in 2010 ([Fossi et al.])

Motivation

Our goal: Malware detection!

Why? Social impact!

- Malware in the news!
- We are all collateral damage!

Huge technological challenge!

- 286 million new malware variants in 2010 ([Fossi et al.])

We need automation!

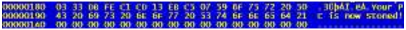
Existing anti-malware technology

Emulation based

- Time limited
- Behavior hiding

Signature matching based

- Easy to avoid detection by syntactic manipulation!



Malware detection

More robust techniques

Solution

One needs to analyse the behavior not the syntax of the program without executing it!

Malware detection

More robust techniques

Solution

One needs to analyse the behavior not the syntax of the program without executing it!

Model checking is a good candidate!

Model checking for malware detection

$$\text{Program} \quad \models \quad \text{Malicious behavior}$$

Model checking for malware detection

Program $\models$ Malicious behavior

Model?

Model checking for malware detection

Program $\models$ Malicious behavior

Model?

Specification formalism
to describe behaviors?

Previous approaches on model checking for malware detection

Use finite state models

- (E.g. Kinder et al. [2010],Bonfante et al. [2008])
- But the model fails to capture stack behavior!

Why is the stack important?

Malware writers use the stack to obfuscate their behaviour.

Example of obfuscation

E.g. call obfuscation:

l_1 : push m
l_2 : push 0
l_3 : call *GetModuleFileName*
l_r : ...

l_1 : push m
l_2 : push 0
l_3 : push l_r
l_4 : jmp l_g
l_r : ...

Import address table	
l_g	*GetModuleFileName*

Example of obfuscation

E.g. call obfuscation:

l_1 : push m
l_2 : push 0
l_3 : call *GetModuleFileName*
l_r : ...

l_1 : push m
l_2 : push 0
l_3 : push l_r
l_4 : jmp l_g
l_r : ...

Import address table	
l_g	*GetModuleFileName*

Our solution is:

To use pushdown systems that is a finite state system + a stack

We use PDS (FSS + stack!)

Pushdown systems (**PDS**)

A **PDS** is a triple $\mathcal{P} = (P, \Gamma, \Delta)$ where:

- P is a finite set of control points,
- Γ is a finite alphabet of stack symbols, and
- $\Delta \subseteq (P \times \Gamma) \times (P \times \Gamma^*)$ is a finite set of transition rules.

Configurations

- A configuration $\langle p, \omega \rangle$ of $\mathcal{P}$ is an element of $P \times \Gamma^*$

PDS for malware detection

Since 2012 PDS have been used to perform malware detection!

- FM [Song and Touili, 2012b]
- TACAS [Song and Touili, 2012a]

POMMADE tool (FSEN [Song and Touili, 2013])

- Logic to specify malicious behaviors.
- Few malicious behaviors (discovered manually!)

PDS for malware detection

Since 2012 PDS have been used to perform malware detection!

- FM [Song and Touili, 2012b]
- TACAS [Song and Touili, 2012a]

POMMADE tool (FSEN [Song and Touili, 2013])

- Logic to specify malicious behaviors.
- Few malicious behaviors (discovered manually!)

Our contribution in this work is to
Show how to automatically extract the malicious behaviors from a set of malware!

Model checking for malware detection

Program $\models$ Malicious behavior

PDS

Specification??

Example of an email worm behavior

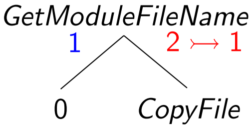
Assembly fragment from Bagle malware

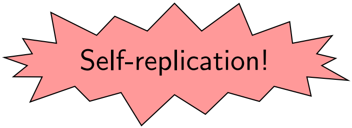
$$
\begin{aligned}
l_1 &: \text{push } \mathrm{m} \\
l_2 &: \text{push } 0 \\
l_3 &: \text{call } \mathit{GetModuleFileName} \\
&\quad\vdots \\
l_4 &: \text{push } \mathrm{m} \\
l_5 &: \text{call } \mathit{CopyFile}
\end{aligned}
$$

Self-replication!

System call dependency trees (SCDT)

l_1 : push m
l_2 : push 0
l_3 : call *GetModuleFileName*
⋮
l_4 : push m
l_5 : call *CopyFile*





Model checking for malware detection

To summarize

Program $\models$ Malicious behavior

PDS

SCDT

Roadmap

How to automatically discover malicious SCDTs from programs?

Approach

Given a:

- set of already known malicious programs
- set of already known benign programs

The goal is

To extract **SCDT**s and use statistical machinery to distinguish the malicious ones!

How to extract SCDTs from a program?

1. Model binaries as pushdown systems (mimic program behaviors)

How to extract SCDTs from a program?

1. Model binaries as pushdown systems (mimic program behaviors)
2. Static reachability analysis (discover system calls)

How to extract SCDTs from a program?

1. Model binaries as pushdown systems (mimic program behaviors)
2. Static reachability analysis (discover system calls)
3. Extract behaviors (discover data flows encoded as trees)

Learning malicious trees

MalSCDT malicious behavior trees

A malicious behavior tree is a tree that occurs frequently in malware extracted **SCDT**s!

To compute frequent "subtrees" we use gSpan!

We specialize the frequent subgraph algorithm presented in [Yan and Han, 2002] to the case of trees.

Roadmap

Model checking for malware detection

In summary we want to verify that:

$$\text{Program} \models \text{Malicious behavior}$$

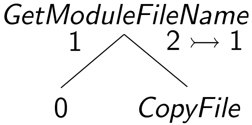
PDS

MalSCDT

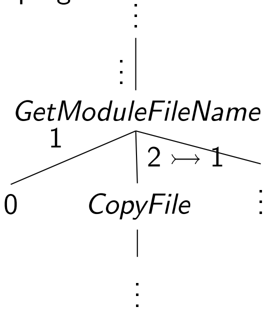
Recognizing **MalSCDT**

A problem!

MalSCDT



SCDT extracted from program under test



Use automata with regexps!

$$GetModuleFileName(q^{*}1(0)q^{*}2 \rightarrowtail 1(\mathrm{CopyFile})\ q^{*}) \rightarrow q_{fin}$$

Teaching computers to detect malware

Build malicious behaviors database

1. Build an hedge automaton $\mathcal{A}$ (recognizing **MalSCDT**)

Teaching computers to detect malware

Build malicious behaviors database

1. Build an hedge automaton $\mathcal{A}$ (recognizing **MalSCDT**)

Malware detection

1. Model binary as **PDS** (mimic program behavior)

Teaching computers to detect malware

Build malicious behaviors database

1. Build an hedge automaton $\mathcal{A}$ (recognizing **MalSCDT**)

Malware detection

1. Model binary as **PDS** (mimic program behavior)
2. Static reachability analysis (discover system calls)

Teaching computers to detect malware

Build malicious behaviors database

1. Build an hedge automaton $\mathcal{A}$ (recognizing **MalSCDT**)

Malware detection

1. Model binary as **PDS** (mimic program behavior)
2. Static reachability analysis (discover system calls)
3. Extract **SCDT** (discover data flows encoded as a tree)

Teaching computers to detect malware

Build malicious behaviors database

1. Build an hedge automaton $\mathcal{A}$ (recognizing **MalSCDT**)

Malware detection

1. Model binary as **PDS** (mimic program behavior)
2. Static reachability analysis (discover system calls)
3. Extract **SCDT** (discover data flows encoded as a tree)
4. Check wether **SCDT** belongs to $\mathcal{A}$

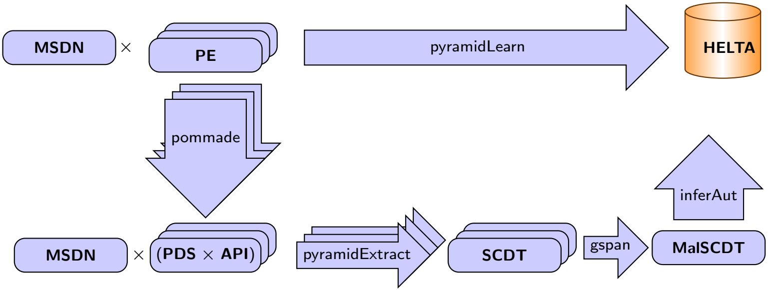
Roadmap

Results

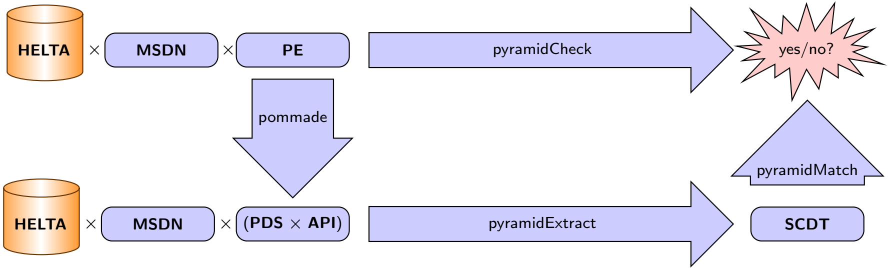
- Implemented the approach in a tool called **PYRAMID**
- Learned **MalSCDT** from a set of malware
- Tested them on another set of malware
- Compared the results with traditional antivirus

Implementation

PYRAMID in learning mode



PYRAMID in detection mode



Experimental results

Learning experimental phase

From 193 malware files we obtained 1026 **MalSCDT**

Detection experimental phase

- Detected 983 malware instances from 330 families ($5\times$ bigger)
- Detection in 2.15s in average
- Correctly classified as non-malware 250 benign programs files

Results comparison

Procedure

- Submitted the “malware” files to 48 antivirus tools
- Categorized the antivirus performance in 4 classes

Results comparison

Procedure

- Submitted the “malware” files to 48 antivirus tools
- Categorized the antivirus performance in 4 classes

Outcome

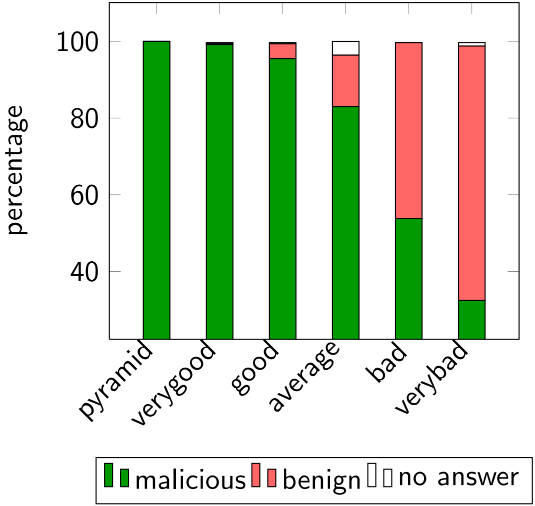
- 99% of the malware files were detected by the top 10% tools!
- Our tool detects real malware!
- In average the tools only detected 80% of the files!

Results comparison

Performance	#Antivirus	Detection range
very good	5	99.1% to 99.5%
good	19	95.0% to 99.1%
bad	19	40.0% to 95.0%
very bad	5	8.0% to 40.0%

Table: Performance categories

Results comparison



Thank you for your attention!

Bibliography

Guillaume Bonfante, Matthieu Kaczmarek, and Jean-Yves Marion. Morphological detection of malware. In *International Conference on Malicious and Unwanted Software*, 2008. doi: 10.1109/MALWARE.2008.4690851.

M. Fossi, G. Egan, K. Haley, E. Johnson, T. Mack, T. Adams, J. Blackbird, M.K. Low, D. Mazurek, D. McKinney, et al. Symantec internet security threat report trends for 2010.

Johannes Kinder, Stefan Katzenbeisser, Christian Schallhart, and Helmut Veith. Proactive Detection of Computer Worms Using Model Checking. *IEEE Trans. on Dependable and Secure Computing*, 2010.

Fu Song and Tayssir Touili. Pushdown model checking for malware detection. In *TACAS*, 2012a.

Fu Song and Tayssir Touili. Efficient malware detection using model-checking. In *FM*, 2012b.

Fu Song and Tayssir Touili. PoMMaDe: Pushdown model-checking for malware detection, 2013.

Experiments

Learning

From 193 malware files we obtained 1026 **MalSCDT**

Detection

- Detected 983 malware instances from 330 families ($5\times$ larger)
- Detection in 2.15s in average
- Correctly classified as non-malware 250 benign programs files