PoMMaDe: Pushdown Model-checking for Malware Detection

Fu Song  
Shanghai Key Laboratory of Trustworthy Computing  
East China Normal University  
Shanghai, P.R.China  
fsong@sei.ecnu.edu.cn

Tayssir Touili  
LIAFA, CNRS and University Paris Diderot  
Paris, France  
touili@liafa.univ-paris-diderot.fr

ABSTRACT
We present PoMMaDe, a Pushdow Model-checking based Malware Detector. In PoMMaDe, a binary program is modeled as a pushdown system (PDS) which allows to track the stack of the program, and malicious behaviors are specified in SCTPL or SLTPL, where SCTPL (resp. SLTPL) is an extension of CTL (resp. LTL) with variables, quantifiers, and predicates over the stack (needed for malware specification). The malware detection problem is reduced to SCTPL/SLTPL model-checking for PDSs. PoMMaDe allows us to detect 600 real malwares, 200 new malwares generated by two malware generators NGVCK and VCL32, and prove benign programs are benign. In particular, PoMMaDe was able to detect several malwares that could not be detected by well-known anti-viruses such as Avira, Avast, Kaspersky, McAfee, AVG, BitDefender, Eset Nod32, F-Secure, Norton, Panda, Trend Micro and Qihoo 360.

Categories and Subject Descriptors
F.3.1 [Theory of Computation]: Specifying and Verifying and Reasoning about Programs—Malware Detection

General Terms
Security, verification

Keywords
Model-Checking, Malware Detection, Pushdown Systems

1. INTRODUCTION
Malwares are one of the most prevalent security threats on the Internet. These programs are used to attack organizations and countries, e.g., the notorious malware Flame has

 Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.  
ESEC/FSE ’13, August 18-26, 2013, Saint Petersburg, Russia  
Copyright 2013 ACM 978-1-4503-2237-9/13/08 ...$15.00.

ABSTRACT
We present PoMMaDe, a Pushdown Model-checking based Malware Detector. In PoMMaDe, a binary program is modeled as a pushdown system (PDS) which allows to track the stack of the program, and malicious behaviors are specified in SCTPL or SLTPL, where SCTPL (resp. SLTPL) is an extension of CTL (resp. LTL) with variables, quantifiers, and predicates over the stack (needed for malware specification). The malware detection problem is reduced to SCTPL/SLTPL model-checking for PDSs. PoMMaDe allows us to detect 600 real malwares, 200 new malwares generated by two malware generators NGVCK and VCL32, and prove benign programs are benign. In particular, PoMMaDe was able to detect several malwares that could not be detected by well-known anti-viruses such as Avira, Avast, Kaspersky, McAfee, AVG, BitDefender, Eset Nod32, F-Secure, Norton, Panda, Trend Micro and Qihoo 360.

Categories and Subject Descriptors
F.3.1 [Theory of Computation]: Specifying and Verifying and Reasoning about Programs—Malware Detection

General Terms
Security, verification

Keywords
Model-Checking, Malware Detection, Pushdown Systems

1. INTRODUCTION
Malwares are one of the most prevalent security threats on the Internet. These programs are used to attack organizations and countries, e.g., the notorious malware Flame has

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 2013 ACM 978-1-4503-2237-9/13/08 ...$15.00.

F.3.1 [Theory of Computation]: Specifying and Verifying and Reasoning about Programs—Malware Detection

1. INTRODUCTION
Malwares are one of the most prevalent security threats on the Internet. These programs are used to attack organizations and countries, e.g., the notorious malware Flame has been used for targeted cyber espionage in Middle Eastern countries and was not detected by any anti-virus for more than five years. Thus, malware detection is a critical topic in computer security. Classic antivirus techniques: code emulation (dynamic analysis) and signature (pattern)-based techniques become insufficient. Indeed, code emulation techniques monitoring only few traces of programs in a limited time span may miss some malicious behaviors. Signature-based techniques using patterns of programs’ codes to characterize malwares can detect only known malwares and are easy to get around.

Addressing these limitations, many works proposed to apply model-checking to detect malware [5, 9, 10, 14, 13, 18, 8, 3, 4], as model-checking checks the behavior (not the syntax) of the program in a static way. However, all these works are not able to model the stack of the programs. Being able to track the stack is important for malware detection, since many obfuscation techniques rely on operations over the stack to evade anti-viruses [16].

To solve this problem, in our previous works [21, 19, 22], we proposed to model a binary program as a pushdown system (PDS). This allows to track the stack of the program. We introduced two new logics SCTPL [21, 19] and SLTPL [22] which are extensions of CTL and LTL with variables, quantifiers, and predicates over the stack. We reduced malware detection to SCTPL/SLTPL model-checking for PDSs. SCTPL/SLTPL allow us to express malicious behaviors in a more succinct manner and specify properties on the stack content which is important for malware detection. In [21, 19, 22], we showed that SCTPL/SLTPL model-checking for PDSs allow to have efficient and robust malware detection techniques.

In this paper, we present PoMMaDe, a malware detector based on SCTPL/SLTPL model-checking for PDSs. PoMMaDe takes as inputs a binary program and a set of SCTPL/SLTPL formulas specifying malicious behaviors. PoMMaDe outputs Yes if the binary program satisfies one of the formulas. It means that the program may be a malware. Otherwise, PoMMaDe outputs No, meaning that the program is benign.

PoMMaDe first uses a preprocessor to automatically unpack the program if it is packed. The (resulting) program is disassembled based on Jakstab [15] and IDA Pro [12]. Jakstab performs static analysis of the binary program, provides an assembly program and the values of the registers and memory addresses. We use IDA Pro to obtain API functions’ informations which cannot be obtained by Jakstab. Then, the assembly program is translated into a PDS. Nex-
t, PoMMADE applies the SCTPL/SLTPL model-checking algorithms of [21, 19, 22] to check whether or not the PDS satisfies one of the formulas. If this is the case, PoMMADE outputs Yes, meaning that the program may be a malware. Otherwise, PoMMADE outputs No, meaning that the program is benign. Moreover, in PoMMADE, we implement three optimization strategies which improve the performance when checking benign programs.

Our tool PoMMADE allows to detect 600 real malwares, all of 200 new malwares generated by the best two malware generators NGVCK and VCL32. Several of these new malwares could not be detected by several well-known anti-viruses such as: Avira, Avast, Kaspersky, McAfee, AVG, BitDefender, Eset Nod32, F-Secure, Norton, Panda, Trend Micro and Qiho 360. NGVCK and VCL32 are the best malware generators as shown in [24]. Moreover, PoMMADE can prove that benign programs are benign and detect the notorious malware Flame which was not detected for more than 5 years. The results show the efficiency and the applicability of our tool. PoMMADE is downloadable at http://www.liafa.univ-paris-diderot.fr/~song/pommade.

Related Work. BitScope [7] is applied to detect trigger-based behaviors in malware. Panorama [11] is a tool for spyware detection. However, [7, 11] are dynamic analysis based tools. Thus, they can miss malware behaviors if they appear after the fixed time interval of the dynamic analysis. PyEA [8] is a malware detector that uses control flow graph matching. PyEA cannot track the stack behaviors which is important for malware detection. Other software model-checkers, such as SLAM [2], Blast [6] and PuMoC [20], dedicated to source code verification, cannot detect malware, since malwares are usually given in binary codes. Binary code analyzers such as CodeSurfer/x86 [1] and McVETO [23] check reachability and cannot detect malware. Indeed, many malicious behaviors are more complex than reachability. Jakstab [15] only considers disassembly and static analysis of binary codes rather than malware detection. In PoMMADE, we implemented the model-checking algorithms of [21, 19, 22]. In these previous works, we had a preliminary implementation that was able to detect 270 malwares. We had to manually check whether a binary program is packed or not. If this is the case, we have to manually unpack the binary program. This task is completely automatic in PoMMADE. Moreover, PoMMADE is able to detect more than 600 real malwares.

2. BACKGROUND

2.1 Pushdown Systems

A pushdown system (PDS) \( P \) is a tuple \((P, \Gamma, \Delta)\), where \( P \) is a finite set of control locations, \( \Gamma \) is the stack alphabet, \( \Delta \subseteq (P \times \Gamma) \times (P \times \Gamma^*) \) is a finite set of transition rules. A configuration of \( P \) is a pair \((p, \omega)\) where \( p \in P \) and \( \omega \in \Gamma^* \). The successor relation \( \sim \) on \((P \times \Gamma^*) \times (P \times \Gamma^*) \) is defined as follows: if \((p, \gamma, q, \omega) \in \Delta\), then \((p, \gamma \omega') \sim (q, \omega')\) for every \( \omega' \in \Gamma^* \). An execution (path) of \( P \) from \( c_0 \) is a sequence of configurations \( \pi = c_0 c_1 \cdots \) s.t. for every \( i \geq 0\), \( c_i \sim c_{i+1} \).

We use the approach of [19] to translate a binary program into a PDS. Intuitively, a binary program is first disassembled into an assembly program by Jakstab and IDA Pro. Then, the assembly program is translated into a PDS in which the control locations correspond to the control points of the program, the stack alphabet consists of the return addresses and the values of the operands of all the push statements. Each transition rule encodes a statement of the program. Thus, a path of the PDS mimics an execution trace of the program.

2.2 SCTPL and SLTPL

In PoMMADE, malicious behaviors are specified in SLTPL or SCTPL. SLTPL (resp. SCTPL) can be seen as an extension of LTL (resp. CTL) with variables, quantifiers and predicates over the stack content. Variables are parameters of atomic predicates and can be quantified by the existential and universal quantifiers. We use regular expressions over the stack alphabet and variables to represent predicates over the stack content. The formal definitions can be found in [21, 19, 22]. For example, the SCTPL formula \( \Psi_{sp} = \exists m \text{EF}(\text{call(GetModuleFileNameA)} \land 0 \text{ml} \land \text{EF}(\text{CopyFileA} \land m^*)) \) is a malicious behavior of worms, where \( 0 \text{ml} \) (resp. \( m^* \)) is a regular expression specifying that the top of the stack is either \( 0m \) or \( m \) as parameters, since parameters are passed through the stack in assembly). Later, \( \text{CopyFileA} \) is called when \( m \) is on the top of the stack (i.e., this function is called with 0 and \( m \) as parameters, since parameters are passed through the stack in assembly). PoMMADE is able to detect more than 600 real malwares.

2.3 The Model-Checking Algorithms

PoMMADE is based on the model-checking algorithms of [21, 22], where SCTPL/SLTPL model checking for PDSs is reduced to the emptiness problem for Symbolic (Alternating) Büchi Pushdown Systems. We refer the reader to [21, 22] for more details.

2.4 Example: Data-stealing Malware

We show how to use SLTPL to express the malicious behavior of a data-stealing malware. More malicious behaviors are described in Appendix B. Note that there exist malicious behaviors that can be expressed in SLTPL but not in SCTPL, and vice versa.

The main purpose of a data-stealing malware is to steal the user’s personal confidential data such as username, password, credit card number, etc and send it to another computer (usually the malware writer). The typical behavior of data-stealing malware can be summarized as follows: the
malware will first call the API function `ReadFile` in order to load some file of the victim into memory. To do this, it needs to call this function with a file pointer `f` (i.e., the return value of the calling function `OpenFile`) as the first parameter and a buffer `m` as the second parameter (`m` corresponds to the address of a memory location), i.e., with `f m` on the top of the stack since in assembly, function parameters are passed through the stack. Then, the malware will send its file (whose data is pointed by `m`) to another computer using the function `send`. It needs to call `send` with a connection `c` (i.e., the return value of the calling function `socket`) as first parameter and the buffer `m` as the second parameter, i.e., with `c m` on the top of the stack. Figure 2 shows a disassembled fragment of a malware corresponding to this typical behavior. Before calling the function `ReadFile`, it pushes the two parameters `m` and `f` onto the stack. Later it calls the function `send` after pushing the two parameters `m` and `c` onto the stack. This behavior can be expressed by the following SLTPL formula \( \Phi_d = \exists m \ F (\text{call} (\text{ReadFile}) \land \Gamma m \Gamma^* \land F (\text{call} (\text{send}) \land \Gamma m \Gamma^*) ) \), where the expression \( \Gamma m \Gamma^* \) states that the second value of the stack is `m` (corresponding to the second parameter of the function `ReadFile` and `send`). \( \Phi_d \) states that there exists an address `m` which is the second parameter when calling `ReadFile`, and such that later, eventually, `send` will be called with `m` as its second parameter.

3. ARCHITECTURE AND IMPLEMENTATION

Given a binary program, and a set of malicious behaviors expressed by SCTPL or SLTPL formulas, PoMMADE outputs Yes if the program satisfies one of the formulas. It means that the binary program may be a malware. Otherwise, PoMMADE outputs No, meaning that the binary program is benign. As shown in Figure 1, PoMMADE consists of five components: Preprocessor, Oracle, Filter, Model Builder and Model-Checking Engines.

Preprocessor uses PEfile [17] to check whether the binary program is packed or not. If this is the case, it uses the corresponding unpacker (if it exists) to unpack the binary code and feed the resulting binary program to Oracle. Otherwise, it directly passes the binary code to Oracle. So far, PoMMADE supports dozens of popular packers for Windows, and hundreds of packers for Linux. Moreover, users can easily add a new unpacking tool by modifying the database file. Oracle takes as input a (unpacked) binary program and outputs the assembly program, and the informations of API functions and the states (values of the registers and memory addresses at each control point). Oracle relies on Jakstab [15] and IDA Pro [12]. Jakstab performs static analysis of the binary program, provides an assembly program and the states. However, it does not allow to extract the informations of API functions and some indirect calls to the API functions. PoMMADE uses IDA Pro to get these informations. The outputs of Oracle are used by Filter to filter out benign programs according to the given optimization strategy by “syntactically” checking the assembly program. Indeed, suppose we would like to check whether a program `P` satisfies the formula \( \Phi_d \) described above. Then if `P` does not contain any call to the functions `ReadFile` or `send`, we can deduce that `P` does not contain this malicious behavior \( \Phi_d \) only by performing a syntactical check over it. We don’t need to apply model-checking to reach this conclusion. PoMMADE provides three strategies: (1) keywords strategy, (2) sequence strategy and (3) direct strategy. When “keywords strategy” is chosen, the user has to provide a set of instructions to PoMMADE (in our example, the user can provide \{ `call (ReadFile), call (send)` \}). Filter will syntactically check whether or not the assembly program contains these instructions. If this is not the case, PoMMADE outputs No (we know that `P` does not contain this malicious behavior \( \Phi_d \)) as first parameter when calling `ReadFile`, and such that later, eventually, `send` will be called with `m` as its second parameter.

4. EXPERIMENTS

All the experiments were run on a Linux platform (Fedora 13) with a 2.4GHz CPU, 2GB of memory. The time limit is fixed to 20 minutes.

Detecting Real Malwares: PoMMADE is applied to check more than 600 real malwares and 400 benign programs tak-
en from Microsoft Windows System. PoMMaDe can detect all the real malwares and prove benign program are benign with only three false positives. PoMMaDe was able to detect several families of malwares using only one formula. In particular, our tool can detect the malware Flame, the most complex attack toolkit which can record audio conversations, intercept the keyboard, etc. Flame has been active for more than 5 years and was not detected by any antivirus.

The average time (resp. memory) of checking one malware against the satisfiable formula is 24.73 seconds (resp. 21.86 MB) (Thanks to Filter, PoMMaDe does not need to apply model-checking for many benign programs).

PoMMaDe vs Existing Antiviruses: To compare our tool with the well-known existing anti-viruses, and show its robustness, we automatically created 200 new malwares using the generators NGVCK and VCL32. [24] showed that these systems are the best malware generators. These programs use very sophisticated features such as anti-disassembly, anti-debugging, anti-emulation, and anti-behavior blocking and come equipped with code morphing ability which allows them to produce different-looking viruses. Our results are reported in Table 1. PoMMaDe was able to detect all these 200 malwares, whereas several well-known and widely used anti-viruses such as Avira, Avast, Kaspersky, McAfee, AVG, BitDefender, Eset Nod32, F-Secure, Norton, Panda, Trend Micro and Qihoo 360 were not able to detect several of them.

5. REFERENCES


<table>
<thead>
<tr>
<th>Generator</th>
<th>No. of Variants</th>
<th>NGVCK</th>
<th>Avira</th>
<th>Kaspersky</th>
<th>Avast</th>
<th>Qihoo 360</th>
<th>McAfee</th>
<th>AVG</th>
<th>BitDefender</th>
<th>Eset Nod32</th>
<th>F-Secure</th>
<th>Norton</th>
<th>Panda</th>
<th>Trend Micro</th>
</tr>
</thead>
<tbody>
<tr>
<td>NGVCK</td>
<td>100</td>
<td>100%</td>
<td>0%</td>
<td>29%</td>
<td>18%</td>
<td>68%</td>
<td>100%</td>
<td>11%</td>
<td>99%</td>
<td>81%</td>
<td>0%</td>
<td>46%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>VCL32</td>
<td>100</td>
<td>100%</td>
<td>0%</td>
<td>2%</td>
<td>100%</td>
<td>99%</td>
<td>0%</td>
<td>100%</td>
<td>100%</td>
<td>76%</td>
<td>0%</td>
<td>30%</td>
<td>0%</td>
<td>0%</td>
</tr>
</tbody>
</table>

Table 1: Detection rates of new malwares generated by NGVCK and VCL32.