APPENDIX

A. Using SMODIC

Let us show how to use SMODIC. The commands to launch the tool are as follows:

- SMODIC <option1> <modelfile> <option2> <formula>

Option1 specifies the input file of SMODIC:

- M: the input is a SM-PDS model.
- B: the input is a binary program

Option2 specifies the model checking strategy:

- L: use the LTL model checking algorithm
- C: use the CTL model checking algorithm
- R1: perform reachability analysis using pre^*
- R2: perform reachability analysis using $post^*$

The model file can be either a binary program or a SM-PDS (.smpds file). The output have three files: one for the Control Flow Graph, one for assembly codes, and one for the generated SM-PDS. A SM-PDS consists of four parts: a finite set of standard PDS transition rules, a finite set of self-modifying transition rules, an initial phase (the initial set of transition rules) and an initial configuration (initial control location equipped with the stack contents).

Fig. 4. The Output of SMODIC

```
0x4ad0161a:     pushl   $0x4ad1f1b2     ; from: 0x4ad0504d(MAY),
0x4ad03fe4(MAY)
0x4ad0161f:     movl    %fs:0, %eax
0x4ad01625:     pushl   %eax
0x4ad01626:     movl    0x10(%esp), %eax
0x4ad0162a:     movl    %ebp, 0x10(%esp)
0x4ad0162e:     leal    0x10(%esp), %ebp
0x4ad01632:     subl    %eax, %esp
0x4ad01634:     pushl   %ebx
0x4ad01635:     pushl   %esi
0x4ad01636:     pushl   %edi
0x4ad01637:     movl    -8(%ebp), %eax
0x4ad0163a:     movl    %esp, -24(%ebp)
0x4ad0163d:     pushl   %eax
0x4ad0163e:     movl    -4(%ebp), %eax
0x4ad01641:     movl    $0xffffffff, -4(%ebp)
0x4ad01648:     movl    %eax, -8(%ebp)
0x4ad0164b:     leal    -16(%ebp), %eax
0x4ad0164e:     movl    %eax, %fs:0
0x4ad01654:     ret     ; targets: 0x4ad03fe9(MAY), 0x4ad05052(MAY)

0x4ad0165a:     movl    0x4ad33b90, %eax        ; from: 0x4ad03fef(MAY)
0x4ad0165f:     testl   %eax, %eax
0x4ad01661:     je      0x4ad04d48      ; targets: 0x4ad04d48(MAY),
0x4ad01667(MAY)
0x4ad01667:     pushl   $0x0    ; from: 0x4ad01661(MAY), 0x4ad04d75(MAY)
0x4ad01669:     call    %eax    ; targets: unresolved
0x4ad0166b:     ret     $0x4    ; targets: 0x4ad03ff4(MAY)      ; from:
0x4ad11d36(MAY)
```

Fig. 5. A Segment of Disassembly Codes

In order to show this, we will use the following command to check whether the program cmd.exe can eventually call the API function `GetModuleA` or not. For this case, we execute the following command:

- SMODIC B malware/cmd.exe L <>getmodulea

Figure 6 is the snapshot of the command to start SMODIC. In this command, "B" is Option1 specifying that the input is a binary program. "L" specifies that the strategy of model checking is LTL. <> getmodulea is the LTL formula $\mathbf{F}$ $(call\ GetModuleA)$.

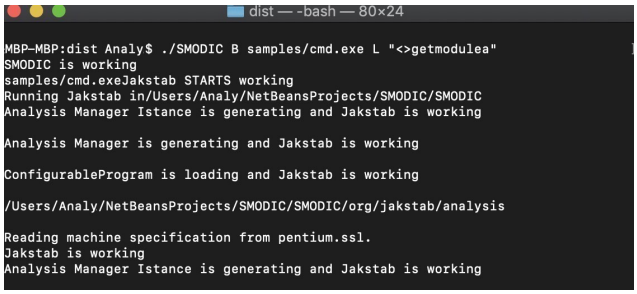

Fig. 6. An Example to Run SMODIC

The output have three files: cfg.dot contains the control flow graph (Figures 7 and 8 are two segments of cfg.dot: the control locations corresponding to the instructions are given in Figure 7, and the edges between control locations are given in Figure 8), cmd.asm contains the assembly code equipped with informations about the API functions (Figure 5 is a fragment of this file), and model.smpds contains the SM-PDS (Figure 9 is a segment of the SM-PDS transition rules). This file contains in addition an initial configuration (the initial control point with the stack contents and the initial set of transition rules). The three files are shown in Fig. 4.

```
"0x4ad050cc"[label="0x4ad050cc\nmovl 0x4ad34874, %ecx"];
"0x4ad04e9a"[label="0x4ad04e9a\npushl $0x4ad04eb8"];
"0x4ad04e9a"[label="0x4ad04e9a\npushl $0x4ad04eb8"];
"0x4ad03fe4"[label="0x4ad03fe4\ncall 0x4ad0161a"];
"0x4ad01637"[label="0x4ad01637\nmovl -8(%ebp), %eax"];
"0x4ad0511e"[label="0x4ad0511e\npushl %eax"];
"0x4ad0511e"[label="0x4ad0511e\npushl %eax"];
"0x4ad04e89"[label="0x4ad04e89\nmovl 0x8(%ebp), %esi"];
"0x4ad050aa"[label="0x4ad050aa\norl $0xffffffff, 0x4ad2fa50"];
"0x4ad050aa"[label="0x4ad050aa\norl $0xffffffff, 0x4ad2fa50"];
"0x4ad03fe4"[label="0x4ad03fe4\ncall 0x4ad0161a"];
"0x4ad0514f"[label="0x4ad0514f\ncall 0x4ad03fdd"];
"0x4ad04d70"[label="0x4ad04d70\nmovl %eax, 0x4ad33b90"];
"0x4ad0510d"[label="0x4ad0510d\nleal -36(%ebp), %eax"];
"0x4ad0514f"[label="0x4ad0514f\ncall 0x4ad03fdd"];
"0x4ad01626"[label="0x4ad01626\nmovl 0x10(%esp), %eax"];
"0x4ad03ff4"[label="0x4ad03ff4\nmovl %ebx, -4(%ebp)"];
"0x4ad19049"[label="0x4ad19049\ncmpl $0xe, 0x84(%ecx)"];
"0x4ad0513e"[label="0x4ad0513e\nmovl __initenv@msvcrt.dll, %ecx"];
"0x4ad19049"[label="0x4ad19049\ncmpl $0xe, 0x84(%ecx)"];
"0x4ad01636"[label="0x4ad01636\npushl %edi"];
"0x4ad050db"[label="0x4ad050db\nmovl %eax, 0x4ad2fa54"];
"0x4ad01636"[label="0x4ad01636\npushl %edi"];
"0x4ad0506b"[label="0x4ad0506b\ncmpl $0x4550, (%ecx)"];
"0x4ad04e8c"[label="0x4ad04e8c\nandl $0x0, (%esi)"];
```

Fig. 7. Control Locations with Instructions

B. Reachability Analysis in SMODIC

Let us show how to use SMODIC to perform reachability analysis on SM-PDSs and self-modifying code. To start the reachability analysis, we need to specify the options. Let us consider Option1 B, and Option2 R2(or R1). We also need to specify the sequence of API functions. For example, to perform the reachability analysis on the sequence of API functions "Call GetModuleA", "Call CopyFile" , "call SendFile", we put the names of the functions in lowercase and use the

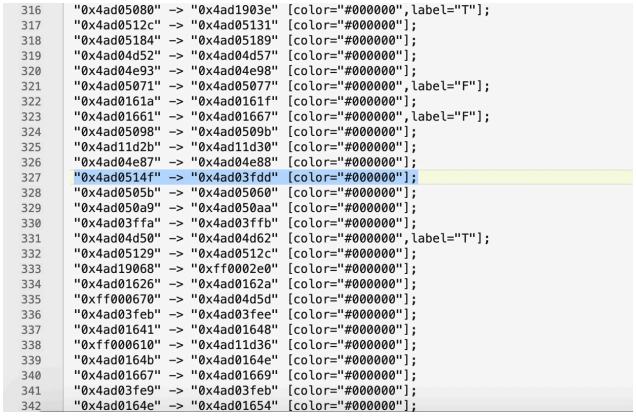

Fig. 8. A Segment of Edges between Locations

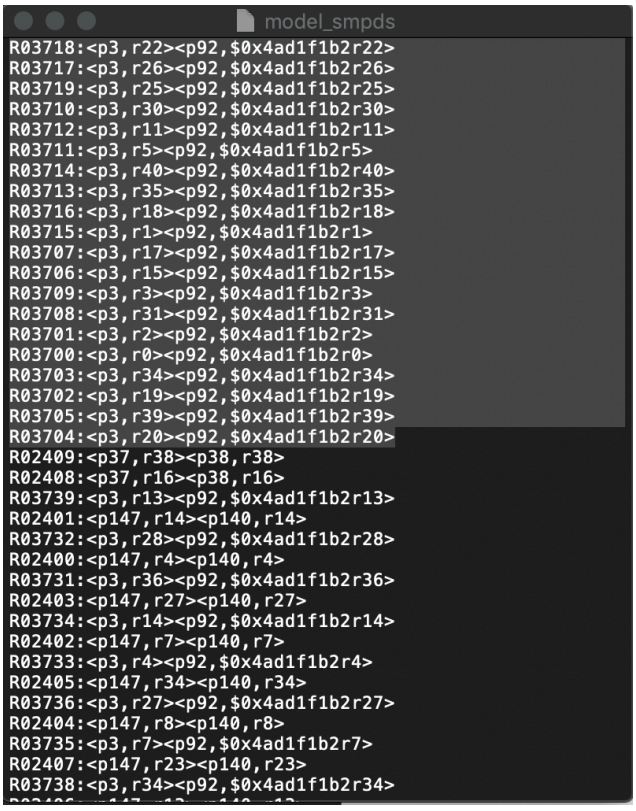

Fig. 9. A Segment of SM-PDS Transition Rules

symbol ";" to separate the names. To use the $post^*$ approach to check whether the above sequence of API functions can be reached or not, we use the following command (see Fig. 10):

- ./SMODIC B malware/cmd.exe R2 getmodulea;copyfile;sendfile

The result is shown in Fig. 11.

We also can run reachability analysis on SM-PDSs. Then,

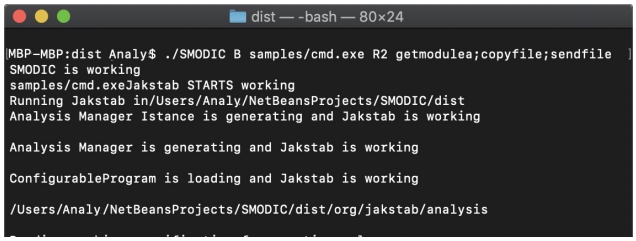

Fig. 10. An Example to Start SMODIC for Reachability Analysis

we need to specify the options. We make Option1 M, and Option2 R2(or R1). We also need to specify the target configuration. For example, we can execute reachability analysis using the $post^*$ approach to check if configuration $\langle p_0, r_0 \rangle$ (i.e., the configuration where the SM-PDS is in control point p_0 and has r_0 as content of the stack) can be reached or not by the following command:

- ./SMODIC M input.smpds R2 p0:r0

The output of SMODIC is the automaton representing the set of reachable configurations. SMODIC also tells whether the target configuration is reached or not.

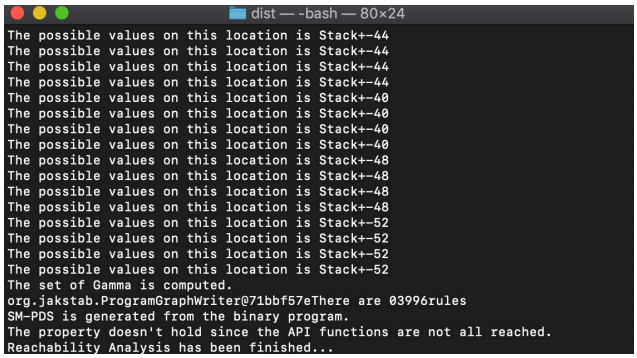

Fig. 11. The Result of the Example for Reachability Analysis

C. LTL and CTL in SMODIC

First, we will introduce the syntax of LTL/CTL used in SMODIC. To be able to use SMODIC, you need to be familiar with the syntax of the logics LTL and CTL. The implementation of LTL and CTL operators in SMODIC is as follows:

For LTL:

- Propositional Symbols: true, false and any lowercase string.
- Boolean operators: !(negation), -> (implication), <->(equivalence), && (and), || (or).
- Temporal operators: $[]p$(p always holds), $<> p$ (eventually p holds), pUq (p holds until q holds), and Xp (p holds next time).

For CTL:

- Propositional Symbols: tt(true),ff(false) and any lowercase string.
- Boolean operators: !(negation), -> (implication), <->(equivalence), && (and), || (or).
- Path quantifiers: A (for all paths) and E (there exists a path).
- Temporal operators: Xp (p holds next time), pRq (p holds until q does't hold), pUq (p holds until q holds).

D. Applying SMODIC for Malware Detection

We show how to apply LTL/CTL model checking to malware detection. Let us take a spy worm as example. Such a worm can record data and send it using the Socket API functions. For example, Keylogger is a spy worm that can record the keyboard states by calling the API functions GetAsyKeyState and GetKeyState and send this to the specific server by calling the socket function sendto. This behavior can be specified by the following LTL formula:

$$\phi_{sw} = \mathbf{F}\big((call\ GetAsyncKeyState \vee call\ GetRawInputData) \wedge \mathbf{F}(call\ sendto \vee call\ send)\big).$$

To check whether the program cmd.exe satisfies this formula or not, first, we need to rewrite this formula to the form supported by our tool SMODIC. Because all the propositions are lowercase strings, we rewrite API function calls (like call GetAsyncKeyState) by removing the word "*call*" and changing the name of the function in lowercase string. The operators are in spin syntax. Thus, formula ϕ_{sw} is rewritten as:

$$<> \big((getasynckeystate||getrawinputdata)\&\& <> (sendto||send)\big)$$

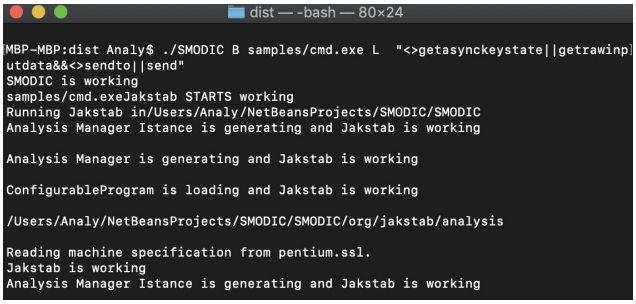

Fig. 12. Command for LTL Model Checking

So, we can check whether the program cmd.exe satisfies this formula or not by using the following command (shown in Figure 12):

./SMODIC B malware/cmd.exe L <> $\big((getasynckeystate||getrawinputdata)\&\& <> (sendto||send)\big)$

The result is shown in Figure 13. The result of the computation is that there is no accepting run. The output of the tool is No, i.e. **cmd.exe** is not a spyware.

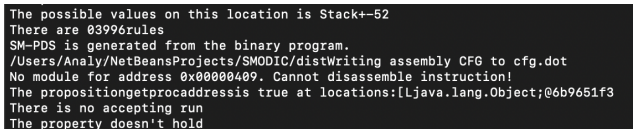

Fig. 13. Result of LTL Model Checking

E. Specifying Malicious Behaviours in LTL and CTL

We show in this section several malicious behaviors that can be described in LTL and/or CTL. These behaviors can be found in [10], [11], [13].

Data-Stealing: Stealing data from the host is a popular malicious behavior that intend to steal any valuable information including passwords, software codes, bank information, etc. To do this, the malware needs to scan the disk to find the interesting file that he wants to steal. After finding the file, the malware needs to locate it. To this aim, the malware first calls the API function GetModuleHandleA to get a base address to search for a location of the file. Then the malware starts looking for the interesting file by calling the API function FindFirstFileA. Then the API functions CreateFileMappingA and MapViewOfFile are called to access the file. Finally, the specific file can be copied by calling the API function CopyFileA. Thus, this data-stealing malicious behavior can be described by the following LTL formula as follows:

$$\phi_{ds} = \mathbf{F}(call\ GetModuleHandleA\ \wedge \mathbf{F}(call\ FindFirstFileA \wedge \mathbf{F}\ (call\ CreateFileMappingA\ \wedge \mathbf{F}\ (call\ MapViewofFile \wedge \mathbf{F}\ call\ CopyFileA))))$$

Spy-Worm: A spy worm is a malware that can record data and send it using the Socket API functions. For example, Keylogger is a spy worm that can record the keyboard states by calling the API functions GetAsyKeyState and GetKeyState and send that to the specific server by calling the socket function sendto. Another spy worm can also spy on the I/O device rather than the keyboard. For this, it can use the API function GetRawInputData to obtain input from the specified device, and then send this input by calling the socket functions send or sendto. Thus, this malicious behavior can be described by the following LTL formula:

$$\phi_{sw} = \mathbf{F}\big((call\ GetAsyncKeyState \vee call\ GetRawInputData) \wedge \mathbf{F}(call\ sendto \vee call\ send)\big)$$

Appending virus: An appending virus is a virus that inserts a copy of its code at the end of the target file. To achieve this, since the real OFFSET of the virus' variables depends on the size of the infected file, the virus has to first compute its real absolute address in the memory. To perform this, the virus has to call the sequence of instructions: l_1: call f; l_2: ....; f: pop eax;. The instruction call f will push the return address l_2 onto the stack. Then, the pop instruction in f will

put the value of this address into the register eax. Thus, the virus can get its real absolute address from the register eax. This malicious behavior can be described by the following LTL formula:

$$\phi_{av} = \bigvee \mathbf{F}\Big(call \wedge \mathbf{X}(\text{top-of-stack} = a) \wedge \mathbf{G}\neg\big(ret \wedge(\text{top-of-stack} = a)\big)\Big)$$

where the $\bigvee$ is taken over all possible return addresses a, and top-of-stack=a is a predicate that indicates that the top of the stack is a. The subformula $call \wedge \mathbf{X}(\text{top-of-stack} = a)$ means that there exists a procedure call having a as return address. Indeed, when a procedure call is made, the program pushes its corresponding return address a to the stack. Thus, at the next step, a will be on the top of the stack. Therefore, the formula above expresses that there exists a procedure call having a as return address, such that there is no ret instruction which will return to a.

Note that this formula uses predicates that indicate that the top of the stack is a. Our techniques work for this case as well: it suffices to encode the top of the stack in the control points of the SM-PDS. Our implementation works for this case as well and can handle appending viruses.

Kernel32.dll Base Address Malware: DLLs(Dynamic Link Libraries) are loaded into the process when a program starts. The kernel32.dll module is always loaded into every process handling memory management and interrupts. Therefore, it contains several API functions that can be used by malicious codes. If a malware wants to copy itself to the installation directory of some programs, it will make a call to LoadLibrary. This requires to know the location of LoadLibrary. Because Windows guarantees all DLLs get loaded at the same location, malicious codes only need to find the base address of Kernel32.dll to get the entrance (address) of the API functions. There are several kinds of approaches implemented in malwares to determine this base address. One techniques consists in finding the PE header of kernel32.dll in memory to locate Kernel32.dll export section. To this aim, the viruses seek first for the DOS header whose beginning word is MZ (5A4Dh in hex), and then look for the PE header whose beginning word is PE00 (4550h in hex). I.e., the malware will keep comparing the values of the registers with 5A4Dh and 4550h, until it succeeds. Thus, this behavior can be described in CTL as follows:

$$\phi_{bvi} = \bigvee \mathbf{EG}\big(\mathbf{EF}\ (cmp(r_1, 5A4Dh) \wedge \mathbf{EF}\ cmp(r_2, 4550h))\big)$$

where the $\bigvee$ is taken over all possible data registers r_1 and r_2, for instance, eax, ebx, ecx and edx.

Email-Worms: Email worms are worms that distribute copies of themselves using infected executables attached to fake email messages. $Klez.h$ is such a malicious program. A worm from the $Klez$ family first calls the API *GetModuleFileNameA* in order to get the name of its executable. For this, the worm needs to call this function with 0 as parameters, i.e., with 0 on the top of the stack since parameters to a function in assembly are passed through the stack. Then, the worm will copy its file to other locations using the function *CopyFileA*. This behavior can be expressed by a CTL formula as follows:

$$\Phi_{ew} = \mathbf{EF}\big(call\ GetModuleFileNameA \wedge (\text{top-of-stack} = 0) \wedge \mathbf{EF}\ call\ CopyFileA\big)$$

where "top-of-stack= 0" is a predicate that indicates that the top of the stack is 0. This formula expresses that there exists a call to the API function *GetModuleFileNameA* made with 0 on the top of the stack, followed by a call to the API function *CopyFileA*. Note that this formula uses predicates that indicate that the top of the stack is 0. Our techniques work for this case as well: it suffices to encode the top of the stack in the control points of the SM-PDS.

Spyware (Scanning the Disk): The aim of a spyware is to steal information from the host. To do this, it has to scan the disk of the host in order to find the interesting file that he wants to steal. If a file is found, it will run a payload to steal it, then continues searching the next file. If a directory is found, it will enter this path and continues scanning. This malicious behaviour is present e.g. in the notorious spyware Flame: It first calls the function FindFirstFileW to search the first object in the given path, then, it will check whether the function call succeeds or not. If the function call fails, it will call the function GetLastError. Otherwise it will call either the function FindFirstFileW again if it finds a directory or the function FindNextFileW to search for the next object. We can specify this behavior in CTL as follows:

$$\phi_{spy} = \mathbf{EF}\ \Big(call\ FindFirstFileW \wedge \mathbf{AF}\ \big(call\ GetLastError \vee call\ FindFirstFileW \vee call\ FindNextFileW\big)\Big)$$

This formula states that there exists a path where the function $FindFirstFileW$ is called, then, in all the future paths, the program either calls $GetLastError$ (if $FindFirstFileW$ failed) or calls $FindFirstFileW$ (if a directory is found) or calls $FindNextFileW$ (to search for the next file). Scanning a disk can be a behavior of a benign program. To avoid false alarms, we can combine this CTL formula with other formulas describing other malicious behaviors expressing the payload (such as sending a file) to determine whether the binary code is a malware or not. Note that, the formula is branching time and cannot be described as a LTL formula.

F. More Experiment Tables

1) Reachability Analysis: To compare the performance of SMODIC against the approach that consists in translating the SM-PDS into an equivalent PDS or symbolic PDS and then apply the standard $post^*$ and pre^* algorithms for PDSs and symbolic PDSs [**?**], [40], we first applied our tool on randomly generated SM-PDSs of various sizes. The results of the comparision using the pre^* (resp. $post^*$) algorithms are reported in Table III (resp. Table IV). In Table III, **Column** $|\Delta| + |\Delta_c|$ is the number of transitions of the SM-PDS (changing and non changing rules). **Column** SMODIC gives the cost it takes to apply our direct algorithm to compute the

pre^* for the given SM-PDS. **Column** PDS shows the cost it takes to get the equivalent PDS from the SM-PDS. **Column** Symbolic PDS reports the cost it takes to get the equivalent Symbolic PDS from the SM-PDS. **Column** Result1 reports the cost it takes to get the pre^* analysis of Moped [**?**] for the PDS we got. **Column** Total1 is the total cost it takes to translate the SM-PDS into a PDS and then apply the standard pre^* algorithm of Moped (Total1=PDS+Result1). **Column** Result2 reports the cost it takes to get the pre^* analysis of Moped for the symbolic PDS we got. **Column** Total2 is the total cost it takes to translate the SM-PDS into a symbolic PDS and then apply the standard pre^* algorithm of Moped (Total2=Symbolic PDS+Result2). "error" in the table means failure of Moped, because the size of the relations involved in the symbolic transitions is huge. Hence, we mark $-$ for the total execution time. You can see that our direct algorithm (**Column** SMODIC) is much more efficient.

Table IV shows the performance of the $post^*$ component of SMODIC. The meaning of the columns are exactly the same as for the pre^* case, but using the $post^*$ algorithms instead. You can see from this table that applying our direct $post^*$ algorithm on the SM-PDS is much better than translating the SM-PDS to an equivalent PDS or symbolic PDS, and then applying the standard $post^*$ algorithms of Moped. Going through PDSs or symbolic PDSs is less efficient and leads to memory out in several cases.

2) CTL model-checking: We compared the performance of SMODIC with the approach that consists in translating the SM-PDS to an equivalent standard PDS, and then applying the standard CTL model checking algorithm implemented in the PDS model-checker tool PuMoC [41]. We randomly generate several SM-PDSs and CTL formulas. The results are shown in Table V. **Column** $|\Delta|+|\Delta_c|$ indicates the size of the transition rules (the standard PDS rules and the self-modifying rules). **Column** formula size shows the size of the CTL formula. **Column** SM-PDS is the cost of SMODIC. **Column** To PDS reports the cost it takes to get the equivalent PDS from the SM-PDS. **Column** PDS is the cost used to run standard CTL model checking for the equivalent PDS in PuMoC. **Column** Total Time is the whole cost it takes to translate the SM-PDS into a PDS, and then apply the PDS CTL model-checking algorithm of PuMoC [41] (Total Time= To PDS + PDS). **Column** $Result1$ is the result of SMODIC and $Result2$ is the result of PuMoC [41], where Y means yes the formula is satisfied and N means no, the formula is not satisfied. "-" means out of memory. It can be seen that SMODIC is much more efficient, and that it terminates in all the cases, whereas going through CTL model-checking of PDSs gets out of memory in most of the cases.

G. Detecting Real Malwares

We applied SMODIC to detect several malwares. We report the results in Tables VI, VII and VIII. **Column** Size gives the number of control locations, **Column** Result shows the result of our tool SMODIC: Y means malicious and N means benign; and **Column** cost gives the cost in seconds.

$\|\Delta\| + \|\Delta_c\|$	SMODIC	PDS	Result1	Total1	Symbolic PDS	Result2	Total2
$10 + 3$	**0.08s & 2MB**	0.15s & 3MB	0.00s	0.15s	0.10s & 2MB	0.00s	0.10s
$13 + 3$	**0.10s & 2MB**	0.15s & 3MB	0.00s	0.15s	0.10s & 2MB	0.00s	0.10s
$13 + 3$	**0.12s & 2MB**	0.15s & 3MB	0.00s	0.15s	0.10s & 2MB	0.00s	0.10s
$43 + 7$	**0.24s &3MB**	3.44s &4MB	0.02s	3.46s	4.80s &5MB	0.01s	4.81s
$110 + 10$	**0.38s &7MB**	5.15s &6MB	0.01s	5.16s	2.71s &8MB	0.00s	2.71s
$120 + 10$	**0.42s &11MB**	5.20s &15MB	0.01s	5.21s	2.79s &10MB	0.01s	2.80s
$255 + 8$	**0.65s & 15MB**	295.41s & 86MB	0.05s	295.46s	21.41s & 76MB	0.02s	21.43s
$1009 + 10$	**1.49s &97MB**	11504.2s & 117MB	2.46s	11506.66s	14.10s &471MB	1.74s	15.84s
$1899 + 7$	**2.98s & 210MB**	6538s & 171MB	4.09s	6542.09s	124.10s & 558MB	2.71s	173.71s
$2059 + 8$	**3.82s &423MB**	19525.1s &113MB	4.19s	19529.29s	20.70s &713MB	error	-
$2099 + 8$	**4.05s & 32MB**	19031s & 192MB	4.19s	19035.19s	124.12s & 757MB	error	-
$2099 + 9$	**7.08s & 252MB**	29742s & 198MB	4.28s	29746.28s	128.12s & 760MB	error	-
$3060 + 9$	**11.36s & 282MB**	29993.05s & 241MB	18.72s	30011.77s	261.07s & 610MB	error	-
$3160 + 9$	**11.99s & 285MB**	29252.05s & 257MB	26.15s	29278.2s	162.55s & 611MB	error	-
$4058 + 7$	**18.06s & 332MB**	81408.51s &307MB	92.68s	81501.19s	802.07s &1013MB	error	-
$4058 + 8$	**19.42s & 397MB**	82812.51s &399MB	91.91s	82904.42s	899.07s & 1020MB	error	-
$4158 + 8$	**21.68s &491MB**	83112.51s &401MB	97.68s	83210.19	899.19s &1021MB	error	-
$5050 + 8$	**23.26s &499MB**	93912.51s &298MB	118.12	94030.63s	205.12s &375MB	error	-

TABLE III
SMODIC VS. STANDARD pre^* ALGORITHMS OF PDSS

$\|\Delta\| + \|\Delta_c\|$	SMODIC	PDS	Result1	Total1	Symbolic PDS	Result2	Total2
$10 + 3$	**0.12s & 2MB**	0.15s & 3MB	0.00s	0.15s	0.10s &2MB	0.00s	0.10s
$13 + 3$	**0.12s & 2MB**	0.15s &3MB	0.00s	0.15s	0.10s &2MB	0.00s	0.10s
$43 + 7$	**0.28s & 2MB**	3.44s & 4MB	0.04s	3.48s	4.80s & 5MB	0.02s	4.82s
$110 + 10$	**0.36s & 8MB**	5.15s &6MB	0.01s	5.16s	2.71s &8MB	0.00s	2.71s
$120 + 10$	**0.39s & 13MB**	5.20s &15MB	0.01s	5.21s	2.79s & 10MB	0.01s	2.80s
$255 + 8$	**0.44s & 15MB**	295.41s & 86MB	0.05s	295.46s	21.41s & 76MB	0.02s	21.43s
$1009 + 10$	**1.48s & 97MB**	11504.2s & 117MB	2.56s	11506.76s	14.10s & 471MB	1.75s	15.85s
$1899 + 7$	**3.47s & 212MB**	6538s & 171MB	3.89s	6541.89s	124.10s & 558MB	2.71s	126.81s
$2059 + 8$	**4.03s &323MB**	19525.1s & 113MB	3.99s	19528.99s	20.70s &713MB	error	-
$2099 + 8$	**4.15s &332MB**	19031s &192MB	3.99s	19034.99s	124.12s & 757MB	error	-
$2099 + 9$	**4.95s & 352MB**	29742s & 198MB	4.18s	29746.18s	128.12s & 760MB	error	-
$3060 + 9$	**5.71s & 388MB**	29993.05s & 241MB	18.12s	30011.17s	261.07s &610MB	error	-
$3160 + 9$	**5.79s & 415MB**	29252.05s & 257MB	26.10s	29278.15s	162.55s & 611MB	error	-
$4058 + 7$	**7.56s & 364MB**	81408.51s & 307MB	91.68s	81500.19s	802.07s & 1013MB	error	-
$4058 + 8$	**9.76s & 387MB**	82812.51s & 399MB	91.71s	82904.22s	899.07s & 1020MB	error	-
$4158 + 8$	**11.85s & 487MB**	83112.51s & 401MB	97.28s	83209.79s	899.19s & 1021MB	error	-
$5050 + 8$	**13.04s & 498MB**	93912.51s &498MB	112.53s	94025.04s	205.12s & 375MB	error	-

TABLE IV
SMODIC VS. STANDARD $post^*$ ALGORITHMS OF PDSS

	$\lvert\Delta\rvert + \lvert\Delta_c\rvert$	formula size	SMODIC time (s)	To PDS	PDS	Total Time	$Result1$	$Result2$
CTL Model Checking	$5+2$	2	**0.27s**	0.09s	0.25s	0.34s	Y	Y
	$5+3$	2	**0.29s**	0.09s	0.36s	0.45s	N	N
	$6+4$	5	**0.36s**	0.21s	0.45s	0.66s	Y	Y
	$8+4$	12	**2.88s**	0.35s	3.41s	3.76s	Y	Y
	$10+4$	18	**3.71s**	0.39s	3.85s	4.24s	Y	Y
	$20+4$	15	**3.84s**	0.62s	3.94s	4.56s	N	N
	$30+4$	8	**4.01s**	2.20s	4.79s	6.99s	Y	Y
	$35+4$	20	**5.13s**	2.36s	6.53s	8.89s	Y	Y
	$50+8$	6	**7.86s**	4.92s	8.04s	12.96s	N	N
	$80+8$	15	**8.46s**	5.06s	10.31s	15.37s	Y	Y
	$80+8$	20	**9.57s**	5.06s	10.79s	15.85s	Y	Y
	$110+8$	6	**8.83s**	5.25s	11.42s	16.64s	N	N
	$110+8$	15	**9.01s**	5.25s	12.98s	18.13s	N	N
	$110+8$	20	**10.24s**	5.25s	13.44s	18.69s	Y	Y
	$120+10$	10	**9.59s**	5.70s	12.32s	18.02s	N	N
	$120+10$	20	**11.48s**	5.70s	14.87s	20.57s	Y	Y
	$250+8$	6	**13.22s**	9.13s	18.94s	28.07s	N	N
	$250+8$	15	**18.37s**	9.13s	21.11s	30.24s	Y	Y
	$500+8$	6	**20.51s**	17.02s	29.25s	46.27s	N	N
	$600+9$	8	**23.34s**	295.24s	57.79s	353.03s	Y	Y
	$600+9$	15	**28.88s**	295.24s	63.16s	358.40s	Y	Y
	$600+9$	25	**35.39s**	295.24s	69.82s	365.06s	Y	Y
	$1000+10$	6	**35.11s**	3251.02s	7127.41s	10378.43s	N	N
	$1100+10$	8	**37.34s**	3251.02s	7319.82s	10570.84s	Y	Y
	$1100+10$	45	**83.63s**	3251.02s	-	-	N	-
	$1500+8$	30	**60.71s**	2182.78s	13821.34s	16004.12s	N	N
	$2000+10$	18	**49.48s**	5529.30s	-	-	Y	-
	$2000+10$	36	**61.13s**	5529.30s	-	-	N	-
	$2100+10$	15	**60.74s**	5544.69s	-	-	Y	-
	$2500+8$	30	**68.55s**	3981.93s	-	-	N	-
	$3000+7$	10	**65.84s**	5167.27s	-	-	Y	-
	$3000+7$	22	**78.51s**	5167.27s	-	-	N	-
	$3500+8$	6	**70.83s**	6105.60s	-	-	N	-
	$3500+8$	20	**83.14s**	6105.60s	-	-	Y	-
	$3500+10$	6	**75.91s**	9219.18s	-	-	N	-
	$3500+10$	20	**93.37s**	9219.18s	-	-	Y	-
	$3800+10$	30	**99.06s**	9295.24s	-	-	N	-
	$3850+10$	8	**93.20s**	9308.01s	-	-	Y	-
	$3850+10$	30	**115.52s**	9308.01s	-	-	N	-
	$4000+10$	8	**113.34s**	10002.28s	-	-	Y	-
	$4000+10$	20	**125.81s**	10002.28s	-	-	N	-
	$4200+8$	15	**121.16s**	9599.37s	-	-	Y	-
	$4500+8$	23	**136.72s**	9881.85s	-	-	Y	-
	$4500+11$	5	**139.95s**	40290.27s	-	-	Y	-
	$4800+11$	10	**142.13s**	42184.85s	-	-	Y	-
	$4800+11$	15	**153.22s**	42184.85s	-	-	Y	-
	$5500+10$	20	**196.46s**	45745.44s	-	-	Y	-

TABLE V
SMODIC VS. STANDARD ALGORITHMS FOR PDSS FOR CTL MODEL CHECKING

Example	Size	Result	$cost$	Example	Size	Result	$cost$	Example	Size	Result	$cost$
Tanatos.b	12315	Yes	16.261s	Netsky.c	45	Yes	0.002s	Win32.Happy	23	Yes	0.042s
Netsky.a	45	Yes	0.047s	Mydoom.c	155	Yes	0.014s	MyDoom-N	16980	Yes	30.231s
Mydoom.y	26902	Yes	12.462s	Mydoom.j	22355	Yes	11.262s	klez-N	6281	Yes	3.252s
klez.c	30	Yes	0.039s	Mydoom.v	5965	Yes	3.971s	Netsky.b	45	Yes	0.057s
Repah.b	221	Yes	2.428s	Gibe.b	5358	Yes	4.229s	Magistr.b	4670	Yes	3.699s
Netsky.d	45	Yes	0.083s	Ardurk.d	1913	Yes	0.482s	klez.f	27	Yes	0.054s
Kelino.l	495	Yes	0.326s	Kipis.t	20378	Yes	25.345s	klez.d	31	Yes	0.085s
Kelino.g	470	Yes	0.672s	Plage.b	395	Yes	0.291s	Urbe.a	123	Yes	0.376s
klez.e	27	Yes	0.094s	Magistr.b	4670	Yes	3.987s	Magistr.a.poly	36989	Yes	49.863s
Mydoom.M@mm	5965	Yes	5.633s	MyDoom.54464	5935	Yes	5.939s	MyDoom.N!worm	5970	Yes	6.152s
Win32.Runouce	51678	Yes	92.692s	Win32.Chur.A	51895	Yes	98.161s	Win32.CNHacker	51095	Yes	94.952s
Win32.Mydoom!O	215	Yes	0.481s	Mydoom.o@MM	257	Yes	0.298s	W.Mydoom.kZ2L	228	Yes	0.729s
Mydoom-EG [Trj]	230	Yes	0.242s	Email.Worm!c	220	Yes	0.249s	W32.Mydoom.L	235	Yes	0.288s
Worm.Mydoom-5	228	Yes	0.307s	Mydoom.CJDZ	225	Yes	0.392s	Mydoom.DN	220	Yes	0.299s
Win32.Mydoom.R	230	Yes	0.322s	Mydoom.dlnpqi	235	Yes	0.296s	Mydoom.o@MM	235	Yes	0.403s
Sramota.avf	240	Yes	0.383s	Mydoom	238	Yes	0.278s	Mydoom.288	248	Yes	0.410s
Mydoom.ACQ	19210	Yes	39.662s	Mydoom.ba	19423	Yes	38.269s	Mydoom.ftde	19495	Yes	39.583s
Worm.Anarxy	210	Yes	1.913s	Malware!15bf	220	Yes	2.017s	Anar.A.2	140	Yes	1.993s
Win32.Anar.a	215	Yes	1.631s	nar.24576	240	Yes	2.738s	Anar.S	155	Yes	2.093s
HLLW.NewApt	4230	Yes	6.954s	Win32.Worm.km	4405	Yes	7.396s	Newapt.Efbh	4550	Yes	7.254s
NewApt!generic	4815	Yes	9.002s	NewApt.A@mm	4485	Yes	8.159s	Newapt.Win32.1	4155	Yes	7.885s
W32.W.Newapt.A	5015	Yes	8.925s	Worm.NewApt.a	51550	Yes	9.083s	malicious.154966	5155	Yes	9.291s
Win32.Yanz	2250	Yes	4.357s	Yanzi.QTQX-0894	2120	Yes	4.109s	Win32.Yanz.a	2410	Yes	4.465s
Win32.Skybag	4180	Yes	6.891s	Skybag.A	4310	Yes	6.205s	Netsky.ah@MM	4480	Yes	6.991s
Skybag.b	4955	Yes	6.892s	Worm.Skybag-1	4820	Yes	7.119s	Win32.Agent.R	4490	Yes	7.898s
Skybag [Wrm]	4985	Yes	7.482s	Skybag.Dvgb	4830	Yes	7.564s	Netsky.CI.worm	4550	Yes	7.180s
Agent.xpro	533	Yes	0.352s	Vilsel.lhb	15036	Yes	4.972s	Generic.2026199	433	Yes	3.489s
Vilsel.lhb	15036	Yes	26.962s	Generic.DF	5358	Yes	7.821s	LdPinch.aoq	7695	Yes	6.290s
Jorik	837	Yes	4.159s	Bugbear-B	9278	Yes	17.737s	Tanatos.O	9284	Yes	21.481s
Gen.2	1510	Yes	5.632s	Gibe.b	5358	Yes	9.615s	Generic26.AXCN	837	Yes	3.792s
Androm	95	Yes	0.028s	Ardurk.d	1913	Yes	3.679s	Generic.12861	30183	Yes	72.264s
LdPinch.by	970	Yes	4.092s	Generic.2026199	433	Yes	2.402s	LdPinch.arr	1250	Yes	1.848s
Generic.12861	30183	Yes	88.294s	Generic.18017273	267	Yes	0.192s	LdPinch.mg	5957	Yes	9.297s
Script.489524	522	Yes	1.458s	Generic.DF	5358	Yes	8.291s	Zafi	433	Yes	1.028s
GenericKD4047	3495	Yes	4.646s	Win32.Agent.es	3500	Yes	6.083s	W32.HfsAutoB.	3398	Yes	5.092s
Trojan.Sivis-1	5351	Yes	7.029s	Win32.Siggen.28	5440	Yes	6.998s	Trojan/Cosmu.isk.	5345	Yes	6.273s
Trojan.17482-4	381	Yes	1.495s	Delphi.Gen	375	Yes	1.948s	Trojan.b5ac.	370	Yes	2.089s
Delfobfus	798	Yes	3.909s	Troj.Undef	790	Yes	4.068s	Trojan-Ransom.	805	Yes	5.119s
LDPinch.400	1783	Yes	4.893s	PSW.LdPinch.plt	1808	Yes	5.088s	PSW.Pinch.1	1905	Yes	5.757s
LdPinch.BX.DLL	2010	Yes	6.965s	LdPinch.fmye	1845	Yes	6.194s	LdPinch.5558	2015	Yes	6.907s
Lydra.a	3450	Yes	8.289s	Trojan.StartPage	2985	Yes	5.982s	PSWTroj.LdPinch	2985	Yes	6.198s
LdPinch-21	3180	Yes	6.917s	LdPinch-R	3025	Yes	7.005s	LdPinch.Gen.2	2990	Yes	6.992s
Graftor.46303	3230	Yes	5.898s	LdPinch-AIH [Trj]	3010	Yes	6.095s	Win32.Heur.k	2970	Yes	5.950s
LdPinch-15	580	Yes	1.008s	LdPinch.e	578	Yes	1.185s	Win32/Toga!rfn	590	Yes	2.023s
PSW.LdPinch.mj	595	Yes	1.078s	Gaobot.DIH.worm	590	Yes	1.482s	LDPinch.DF	588	Yes	1.736s
TrojanSpy.Zbot	610	Yes	1.610s	LDPinch.10639	605	Yes	1.185s	SillyProxy.AM	590	Yes	1.882s
LdPinch.mj!c	590	Yes	4.5345s	LdPinch.H.gen	605	Yes	3.955s	Generic!BT	615	Yes	2.085s
LdPnch-Fam	195	Yes	1.440s	Troj.LdPinch.er	205	Yes	2.529s	LdPinch.Gen.3	210	Yes	1.482s
Win32.Malware.wsc	150	Yes	2.843s	malicious.7aa9fd	185	Yes	2.189s	WS.LDPinch.400	195	Yes	1.898s

TABLE VI
MALWARE DETECTION USING SMODIC: PART 1

Example	Size	Result	*cost*	Example	Size	Result	*cost*	Example	Size	Result	*cost*
calculation	9952	No	18.352s	cisvc.exe	4105	No	3.631s	simple.exe	52	No	0.001s
shutdown.exe	2529	No	0.397s	loop.exe	529	No	9.249s	cmd.exe	1324	No	13.466s
notepad.exe	10529	No	24.583s	java.exe	800	No	15.852s	java.exe	21324	No	42.373s
sort.exe	8529	No	29.789s	bibDesk.exe	32800	No	50.279s	interface.exe	1005	No	8.462s
ipv4.exe	968	No	4.186s	TextWrangler	14675	No	45.221s	sogou.exe	45219	No	55.259s
game.exe	34325	No	82.424s	cycle.tex	9014	No	42.555s	calender.exe	892	No	35.039s
SdBot.zk	3430	Yes	23.242s	Virus.Gen	661	Yes	9.437s	AutoRun.PR	240	Yes	4.181s
Adon.1703	37	Yes	0.358s	Adon.1559	37	Yes	0.255s	Spam.Tedroo.AB	487	Yes	0.924s
Akez	273	Yes	0.136s	Alcaul.d	845	Yes	0.165s	Alaul.c	355	Yes	0.109s
Virus.klk	5235	Yes	15.863s	Virus.Agent	5340	Yes	15.968s	Hoax.Gen	5455	Yes	13.569s
eHeur.Virus02	420	Yes	4.985s	Akez.11255	440	Yes	3.985s	Akez.Win32.1	455	Yes	4.008s
Weird.10240.C	430	Yes	3.929s	PEAKEZ.A	450	Yes	2.998s	Virus.Weird.c	473	Yes	3.302s
W95/Kuang	435	Yes	2.985s	Radar01.Gen	465	Yes	4.005s	Akez.Win32.5	490	Yes	3.958s
Haharin.A	210	Yes	1.462s	fsAutoB.F026	245	Yes	1.698s	Haharin.dr	235	Yes	1.558s
NGVCK1	329	Yes	0.933s	NGVCK2	455	Yes	1.109s	NGVCK3	2300	Yes	1.388s
NGVCK4	550	Yes	1.149s	NGVCK5	1555	Yes	1.825s	NGVCK6	1698	Yes	1.689s
NGVCK7	6902	Yes	14.524s	NGVCK8	2355	Yes	4.254s	NGVCK9	281	Yes	13.301s
NGVCK10	2980	Yes	9.262s	NGVCK11	5965	Yes	11.456s	NGVCK12	4529	Yes	10.094s
NGVCK13	2210	Yes	8.902s	NGVCK14	5358	Yes	10.294s	NGVCK15	970	Yes	1.912s
NGVCK16	658	Yes	0.935s	NGVCK17	913	Yes	1.392s	NGVCK18	90	Yes	0.094s
NGVCK19	1295	Yes	6.958s	NGVCK20	4378	Yes	15.449s	NGVCK21	31	Yes	0.097s
NGVCK22	370	Yes	0.898s	NGVCK23	3955	Yes	9.498s	NGVCK24	6924	Yes	11.983s
NGVCK25	8127	Yes	15.018s	NGVCK26	4970	Yes	9.982s	NGVCK27	7989	Yes	13.197s
NGVCK28	227	Yes	0.098s	NGVCK29	960	Yes	0.692s	NGVCK30	89	Yes	0.088s
NGVCK31	550	Yes	0.875s	NGVCK32	60	Yes	0.059s	NGVCK33	65	Yes	0.069s
NGVCK34	5990	Yes	9.848s	NGVCK35	4590	Yes	10.178s	NGVCK36	825	Yes	2.934s
NGVCK37	80	Yes	0.998s	NGVCK38	150	Yes	1.093s	NGVCK39	395	Yes	1.048s
NGVCK40	40	Yes	0.921s	NGVCK41	950	Yes	0.704s	NGVCK42	8290	Yes	15.085s
NGVCK43	6220	Yes	2.930s	NGVCK44	5215	Yes	11.006s	NGVCK45	9290	Yes	14.595s
NGVCK46	320	Yes	0.928s	NGVCK47	834	Yes	2.958s	NGVCK48	9810	Yes	14.696s
NGVCK49	12320	Yes	25.395s	NGVCK50	8810	Yes	19.969s	NGVCK51	39810	Yes	68.283s
NGVCK52	520	Yes	0.289s	NGVCK53	15	Yes	0.089s	NGVCK54	8883	Yes	11.393s
NGVCK55	12520	Yes	38.768s	NGVCK56	6218	Yes	15.489s	NGVCK57	32562	Yes	83.482s
NGVCK58	9520	Yes	23.658s	NGVCK59	818	Yes	2.592s	NGVCK60	12962	Yes	38.025s
NGVCK61	10020	Yes	24.976s	NGVCK62	8818	Yes	19.299s	NGVCK63	2068	Yes	3.662s
NGVCK64	273	Yes	1.987s	NGVCK65	5855	Yes	8.995s	NGVCK66	68	Yes	1.002s
NGVCK69	4150	Yes	8.052s	NGVCK70	9860	Yes	24.199s	NGVCK71	3240	Yes	7.951s
NGVCK72	31	Yes	0.591s	NGVCK73	549	Yes	1.052s	NGVCK74	9078	Yes	29.078s
NGVCK75	90	Yes	1.002s	NGVCK76	5890	Yes	10.128s	NGVCK77	1958	Yes	9.559s
NGVCK78	33468	Yes	75.098s	NGVCK79	4735	Yes	10.980s	NGVCK80	45273	Yes	82.396s
NGVCK66	777	Yes	0.198s	NGVCK67	895	Yes	0.223s	NGVCK81	6939	Yes	2.726s
NGVCK82	2931	Yes	0.463s	NGVCK83	8759	Yes	10.316s	NGVCK84	34563	Yes	53.244s
NGVCK85	19024	Yes	29.220s	NGVCK86	1026	Yes	0.572s	NGVCK87	7929	Yes	5.671s
NGVCK88	6126	Yes	8.682s	NGVCK89	580	Yes	2.036s	NGVCK90	27843	Yes	17.353s
NGVCK91	20	Yes	0.001s	NGVCK92	59	Yes	0.903s	NGVCK93	98	Yes	0.021s
NGVCK94	150	Yes	0.146s	NGVCK95	1679	Yes	0.294s	NGVCK96	6299	Yes	5.196s
NGVCK97	4496	Yes	5.272s	NGVCK98	428	Yes	0.329s	NGVCK99	158	Yes	1.153s
NGVCK100	895	Yes	0.961s	NGVCK101	745	Yes	1.117s	NGVCK102	704	Yes	0.269s
NGVCK103	86	Yes	0.282s	NGVCK104	145	Yes	0.998s	NGVCK105	24124	Yes	68.816s

TABLE VII
MALWARE DETECTION USING SMODIC: PART 2

Example	Size	Result	$cost$	Example	Size	Result	$cost$	Example	Size	Result	$cost$
calculation.exe	9952	No	76.34s	cisvc.exe	4105	No	31.22s	simple.exe	52	No	3.17s
shutdown.exe	2529	No	23.52s	loop.exe	529	No	11.78s	cmd.exe	1324	No	19.36s
notepad.exe	10529	No	68.77s	java.exe	800	No	19.17s	java.exe	21324	No	122.07s
sort.exe	8529	No	74.12s	bibDesk.exe	32800	No	243.79s	interface.exe	1005	No	18.25s
ipv4.exe	968	No	24.43s	TextWrangler.exe	14675	No	65.09s	sogou.exe	45219	No	301.14s
game.exe	34325	No	234.14s	cycle.tex	9014	No	75.44s	calender.exe	892	No	25.39s
Adson.1651	39	Yes	0.44s	Adson.1734	42	Yes	0.43s	Alcaul.d	40	Yes	0.48s
Adon.1703	37	Yes	0.39s	Adon.1559	37	Yes	0.35s	Alcaul.i	48	Yes	0.44s
Alcaul.o	33	Yes	0.29s	Alcaul.d	845	Yes	0.165s	Alaul.c	355	Yes	0.109s
Alcaul.j	45	Yes	0.56s	Alcaul.m	23	Yes	0.19s	Evol.a	53	Yes	7.09s
Alcaul.e	32	Yes	1.93s	Alcaul.h	34	Yes	3.95s	Alcaul.g	25	Yes	4.18s
Alcaul.b	19	Yes	0.12s	Alcaul.f	23	Yes	1.99s	Alcaul.k	28	Yes	2.31s
Alcaul.l	27	Yes	0.95s	Klinge	45	Yes	64.15s	Akez.Win32.5	490	Yes	53.18s
Oroch.3982	31	Yes	1.49s	Anar.a	22	Yes	1.27s	Anar.b	25	Yes	1.58s
Bagle.dp	235	Yes	3.52s	Bagle.dv	175	Yes	6.14s	Bagle.ds	328	Yes	7.29s
Bagle.e	30	Yes	1.52s	Bagle.eb	185	Yes	3.87s	Bagle.ee	198	Yes	3.88s
Bagle.dn	198	Yes	6.07s	Bagle.ej	188	Yes	5.11s	Bagle.ff	355	Yes	8.07s
Bagle.ex	197	Yes	5.96s	Bagle.ev	183	Yes	6.50s	Bagle.en	192	Yes	9.99s
Predec.h	2650	Yes	58.34s	Predec.i	2855	Yes	63.58s	Predec.j	2835	Yes	62.37s
Predec.b	2830	Yes	61.77s	Predec.c	2858	Yes	64.02s	Predec.d	2826	Yes	61.11s
Predec.e	2850	Yes	67.97s	Predec.f	2895	Yes	69.50s	Predec.g	2829	Yes	66.59s
Haharin	355	Yes	13.52s	Ditex.a	25	Yes	1.46s	Oroch.5420	75	Yes	9.42s
Netsky.a	45	Yes	19.12s	Mydoom.c	155	Yes	4.14s	MyDoom-N	16980	Yes	343.93s
Netsky.x	55	Yes	21.85s	Netsky.y	68	Yes	29.06s	Netsky.z	115	Yes	43.37s
Netsky.gen	5508	Yes	59.24s	Netsky.p	6015	Yes	76.32s	Netsky.m	6805	Yes	73.77s
Netsky.r	230	Yes	8.83s	Netsky.k	6115	Yes	79.79s	Netsky.e	6245	Yes	79.44s
Mydoom.y	26902	Yes	452.77s	Mydoom.j	22355	Yes	211.93s	klez-N	6281	Yes	63.07s
klez.c	30	Yes	2.79s	Mydoom.v	5965	Yes	283.11s	Netsky.b	45	Yes	29.51s
Repah.b	221	Yes	12.76s	Gibe.b	5358	Yes	37.01s	Magistr.b	4670	Yes	43.59s
Netsky.d	45	Yes	1.87s	Ardurk.d	1913	Yes	12.08s	klez.f	27	Yes	0.73s
Kelino.l	495	Yes	21.01s	Kipis.t	20378	Yes	121.11s	klez.d	31	Yes	0.95s
Kelino.g	470	Yes	22.08s	Plage.b	395	Yes	1.96s	Urbe.a	123	Yes	9.17s
klez.e	27	Yes	3.94s	Magistr.b	4670	Yes	231.97s	Magistr.a.poly	36989	Yes	469.63s
Mydoom.M	5965	Yes	75.19s	MyDoom.54464	5935	Yes	45.78s	Mydoom.e	138	Yes	46.53s
Mydoom.R	230	Yes	30.22s	Mydoom.dlnpqi	235	Yes	1.99s	Mydoom.o	235	Yes	2.01s
Sramota.avf	240	Yes	11.01s	Mydoom	238	Yes	2.01s	Mydoom.288	248	Yes	3.12s
Mydoom.ACQ	19210	Yes	439.57s	Mydoom.ba	19423	Yes	238.77s	Mydoom.ftde	19495	Yes	339.29s
LdPinch.by	970	Yes	42.92s	Generic.2026199	433	Yes	32.83s	LdPinch.arr	1250	Yes	49.84s
Generic.12861	30183	Yes	188.94s	Generic.18017273	267	Yes	9.19s	LdPinch.mg	5957	Yes	69.77s
LDPinch.400	1783	Yes	54.93s	PSW.LdPinch.plt	1808	Yes	55.88s	PSW.Pinch.1	1905	Yes	57.07s
Newapt.F	11785	Yes	211.24s	Newapt.A	11715	Yes	205.79s	Newapt.E	11797	Yes	252.49s
LdPinch.bb	8145	Yes	63.13s	LdPinch.br	3645	Yes	33.52s	LdPinch.hb	1645	Yes	21.08s
LdPinch.v	7235	Yes	51.69s	LdPinch.fk	4906	Yes	47.11s	LdPinch.awp	195	Yes	17.97s
LdPinch.aaz	4145	Yes	41.05s	LdPinch.c0	8230	Yes	65.17s	LdPinch.ee	6501	Yes	71.30s
Bagle.m	5111	Yes	39.92s	Bagle.k	35	Yes	1.92s	Bagle.t	3345	Yes	45.64s
Newapt.C	11730	Yes	924.92s	Krynos.b	18370	Yes	893.45s	Jeans.a	6490	Yes	188.36s
Atak.f	2005	Yes	11.35s	Atak.g	2498	Yes	16.69s	Atak.l	1914	Yes	10.37s
Bagle.ab	5690	Yes	89.42s	Bagle.ef	995	Yes	54.11s	Bagle.eg	380	Yes	25.49s

TABLE VIII
MALWARE DETECTION USING SMODIC: PART 3