Final Project Report

Dr.Rajshekar K, Diya Ilinani

April 30, 2021

Abstract

In this report, ideas involving static and dynamic analysis have been proposed. The attempt is to help distinguish between benign and malware files using the features extracted from the files using control flow graphs for dynamic analysis and statistics of system calls for static analysis, after which machine learning classifying algorithms are applied to classify the same.

1 INTRODUCTION

Malware is any malicious program that attempts to perform an unwanted action such as intentionally harming the system or stealing confidential information without the knowledge of the user. Antivirus tools in this age rely on gathering an extensive amount of data in the form of already detected malicious signatures in order to predict similar programs. Although this works fairly effectively for most variants, it is much harder to do so with obfuscated malware variants. In this case, we must identify features that are relevant to the behavior of malicious programs and reduce the reliance on an enormous signature database.

The use of control flow graph is not new in malware detection method but the focus has been on the how to improve comparison of said graphs or optimize them. There have not been many papers in which the graphs themselves have been subjected to feature selection.
In [1], the authors built a control flow graph database to use as the signature of the malware.Assembly code was extracted using an .exe file was extracted using a disassembler. The assembly code is optimized by removing a number of indirect paths and unwanted instructions. This is useful because obfuscated versions of malware are often longer due to the inclusion of irrelevant statements in the code. The optimization process is done by using the same methods that compilers use to optimize code and increase efficiency.
In [2], the authors extract features from the control flow graph of executables and create training data. Then, generate classifiers according to specific machine learning algorithms then detect virus with these classifiers.The features extracted from the graphs include the number of nodes, edges, number of isolated nodes, terminal

nodes, maximum degree and average degree of each node. The paper establishes that there is a significant difference in the graph size of malicious and benign files and CFGs can therefore be useful features to classify them.

2 PROPOSED METHOD

STATIC ANALYSIS:
In static analysis, a binary is analysed without actually executing the program and this is typically done with the help of a disassembler. In the proposed method, we construct a basic block control flow graph of an executable and extract features from it. Basic blocks are sections of code that execute in the order that each line is written consecutively. This means that there are no branch instructions in the basic block code except at the point of entry and point of exit. After the execution of a basic block is complete it moves on to another such block. A control flow graph is a directed graph describing the transfer of control. A control flow graph of these blocks would describe all the possible paths that could be taken in the assembly program.

Radare2 [3] is a framework used to reverse-engineer and analyse binaries. It comes with a number of predefined commands to make the extraction of graphs and analysis easier. The command 'aaa' directly executes a number of commands : analyse all function calls , analyse len bytes of instructions for references , type matching analysis for all functions , constructing function names , analysing all flags starting from where the execution of the program starts. The basic block control flow graph consists of nodes which are the basic blocks with a block ID and instructions in each block, the edges denote the blocks to which jumps occur due to conditional branches, jmp instructions or function calls.

A script was written using the r2pipe module in python ,used to extract the basic block control flow graph in the form of a dot graph for all the malicious and benign programs.

In the dot graph program, each node has a node ID which is a string followed by all the instructions in the block. The list of nodes is followed by all the edges that form the graph where the order of the nodes of each edge indicates the direction and the node is depicted by the same string as used above.

In the proposed method, each node is mapped to an integer starting from zero, till all basic blocks are marked by a number. This ensures that there is some symmetry between two different programs instead of randomly labelled strings. After that in each basic block only some opcodes are selected and separated.

In [4], 992 x86 and x64 malicious PE-files were downloaded from virusshare. For the benign data set, 771 files were taken from Windows Vista system exe files and popular application files. In the analysis performed on these files, there were 62 assembly instructions that occurred in malware samples but were not present in

benign files. A lot of these instructions were virtual machine operations which were presumably being used to check if programs were being executed in a sandbox to detect if a program is malware or not. It should also be noted that fast system calls were only used in malware samples. The paper recorded the most common instructions in both malware and benign samples. It also has a list of opcodes that were only in the malware file data set. The tables below indicate the instructions that were used in the feature vector and were taken from [4].

Opcode	Description	Malware %	Benign %
mov	Move	36,88	36,98
call	Call Procedure	8,89	8,02
lea	Load Effective Address	7,76	4,37
cmp	Compare Two Operands	5,94	5,30
push	Push Word/DW/QW Onto the Stack	2,16	8,49
jz	Jump short if zero/equal (ZF=0)	4,41	4,58
test	Logical Compare	3,27	3,97
jmp	Jump	3,59	3,06
add	Add	3,34	3,08
jnz	Jump short if not zero/not equal	2,81	2,89
pop	Pop a Value from the Stack	2,15	3,52
xor	Logical Exclusive OR	3,24	2,05
sub	Subtract	2,57	1,55
retn	Return from procedure	1,68	1,66
movzx	Move with Zero-Extend	1,33	1,06
and	Logical AND	1,13	0,95
or	Logical Inclusive OR	0,68	0,51
inc	Increment by 1	0,50	0,54

Figure 1: Top 18 opcodes found in malware and benign files as referenced from [4]

In the proposed method only the opcodes in these tables were separated in the basic blocks in the sequence that they are written in each block. A feature vector is made from the graph program, it consists of all the nodes with their mapped names with the instructions that were selected according to the tables followed by all the edges in the graph. This feature vector is then trained using machine learning algorithms.

DYNAMIC ANALYSIS:

In dynamic analysis, a binary is analysed by running it in a secure environment and examining the log files. This is typically done with the help of a virtual machine or emulator. Cuckoo sandbox is a popular tool used to perform dynamic analysis on binaries. For dynamic analysis, in the proposed method, features were extracted from cuckoo sandbox reports of executables. The cuckoo JSON reports are divided

Opcode	Description
stosq	Store String
syscall	Fast System Call
setno	Set Byte on Condition - not overflow (OF=0)
cvtsd2si	Convert Scalar Double-FP Value to DW Integer
movmskpd	Extract Packed Double-FP Sign Mask
prefetcht1	Prefetch Data Into Caches
fprem	Partial Remainder (for,compatibility with i8087 and i287)
cmpsq	Compare String Operands
lodsq	Load String
scasq	Scan String
cvtss2si	Convert Scalar Single-FP Value to DW Integer
fnsave	Store x87 FPU State
orpd	Bitwise Logical OR of Double-FP Values
fxsave	Save x87 FPU, MMX, XMM, and,MXCSR State
movmskps	Extract Packed Single-FP Sign Mask

Figure 2: Top 15 opcodes that were found exclusively in the 992 malicous files that were analysed as referenced from [4]

into sections for strings, target, apistats, processes and pe imports.A script was written to parse these sections and only record the APStats section with each call and frequency in python.For the feature vector each API call in the APIStats sections followed by its frequency was taken as an array. These vectors were used to train various machine learning algorithms.

A second feature vector was selected which also included the pe imports in the JSON report, following the same processes and ultimately training using the same machine learning algorithms.

3 RESULTS

The dataset used in static analysis [5] consists of 2054 malicious files and 323 benign files found on github, out of which 800 and 300 malicious and benign files were used to train and test the machine learning algorithms. The result for various algorithms are as follows:

The dataset used in dynamic analysis [6] consists of 2025 malicious cuckoo reports and 1195 benign cuckoo reports. Out of these 800 and 320 malicious and benign reports were used in the machine learning algorithms respectively.

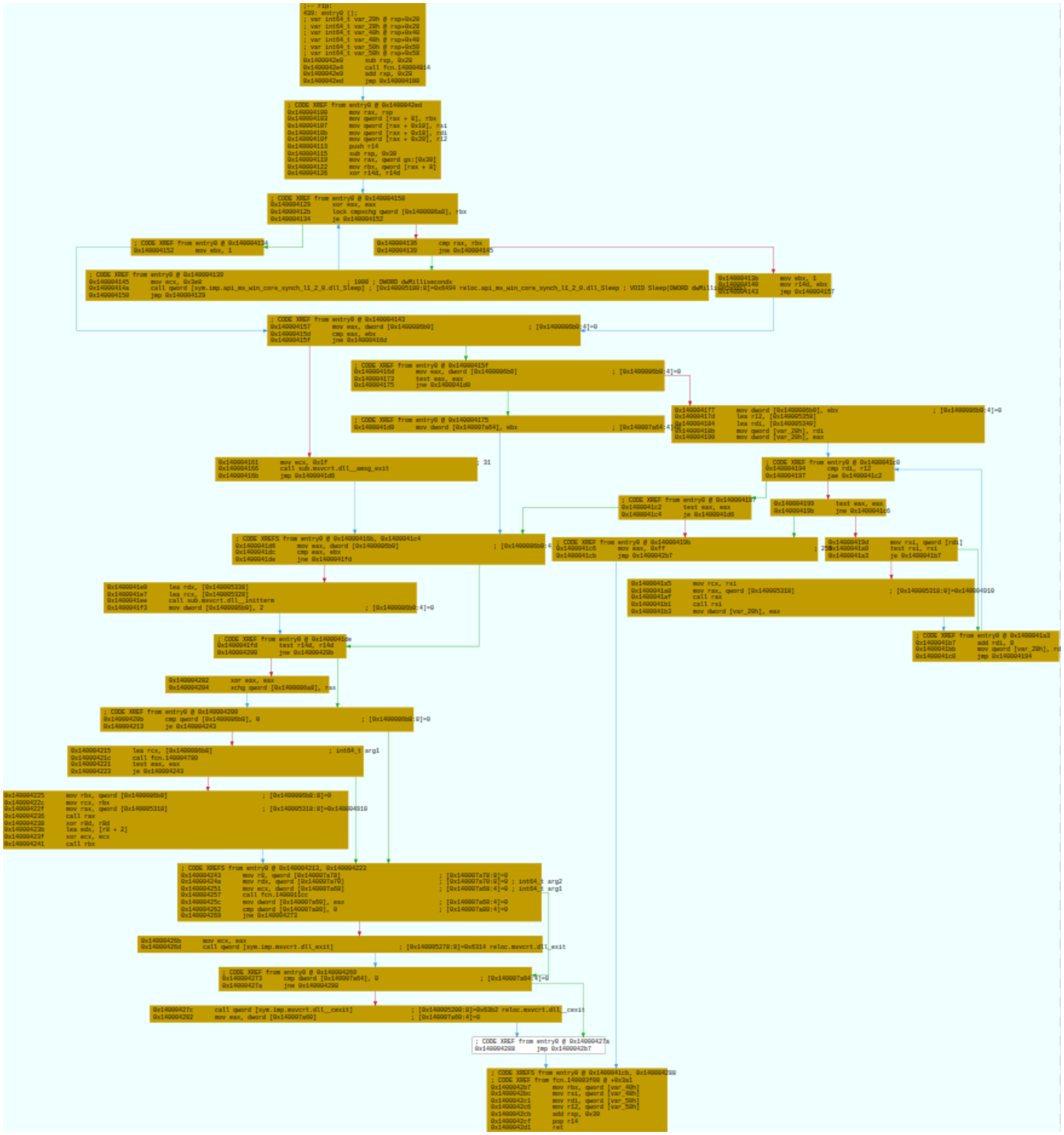

Figure 3: Control flow graph of a program

0	test	1	push	pop	2	push	push	push	push	call	mov	cmp	3	push	call	4	call	push	push	call	xor	mov	mov	lea
push	call	mov	mov	mov	mov	mov	mov	5	push	push	push	call	cmp	6	xor	inc	7	mov	cmp	je	8	mov	test	9
mov	push	push	call	pop	pop	test	10	mov	cmp	11	push	push	call	sub.msvcrt.dll__initterm\l0x01006909				pop	pop	mov	12	cmp	13	
push	push	call	14	cmp	15	mov	cmp	je	16	mov	17	mov	movzx	cmp	18	cmp	je	19	inc	inc	jmp	20	cmp	je	
21	cmp	je	22	jmp	23	mov	jmp	24	movzx	cmp	je	25	cmp	26	inc	inc	mov	jmp	27	cmp	28	xor	inc	mov	
jmp	29	push	call	jmp	30	push	call	sub.msvcrt.dll__amsg_exit\l0x0100d8a4					pop	jmp	31	push	call	pop	test	je	32	push	push	push	
call	jmp	33	movzx	jmp	34	xor	cmp	mov	jmp	35	cmp	36	call	37	mov	mov	jmp	38	mov	mov	39	call	0	->	
33	0	->	1	1	->	2	2	->	35	2	->	3	3	->	41	4	->	5	5	->	27	5	->	6	
6	->	7	7	->	30	7	->	8	8	->	23	8	->	9	9	->	38	9	->	10	10	->	12	10	
>	11	11	->	12	12	->	14	12	->	13	13	->	14	14	->	31	14	->	15	15	->	38	15	->	
16	16	->	17	17	->	20	17	->	18	18	->	34	18	->	19	19	->	17	20	->	24	20	->	21	
21	->	24	21	->	22	22	->	18	23	->	10	24	->	0	24	->	25	25	->	0	25	->	26	26	
>	24	27	->	29	27	->	28	28	->	7	29	->	5	30	->	10	31	->	15	31	->	32	32	->	
15	33	->	2	34	->	19	35	->	37	35	->	36	36	->	37	37	->	39	38	->	39				

Figure 4: Feature vector used in static analysis

ALGORITHM	PRECISION	RECALL	ACCURACY
KNEIGHBOURS CLASSIFIER	0.99	0.96	98.0
SVC	0.99	0.97	98.18
LOGISTIC REGRESSION	0.97	0.96	97.27
MLP CLASSIFIER	0.97	0.96	97.09

Figure 5: Static analysis

ALGORITHM	PRECISION	RECALL	ACCURACY
KNEIGHBOURS CLASSIFIER	0.88	0.84	88.66
SVC	0.84	0.85	86.33
LOGISTIC REGRESSION	0.83	0.83	85.66
MLP CLASSIFIER	0.82	0.81	84.66

Figure 6: Feature vector with API STATS (DYNAMIC ANALYSIS)

ALGORITHM	PRECISION	RECALL	ACCURACY
KNEIGHBOURS CLASSIFIER	0.83	0.83	85.66
SVC	0.83	0.86	86.00
MLP CLASSIFIER	0.87	0.87	88.66

Figure 7: Feature vector with API STATS and PE Imports(DYNAMIC ANALYSIS)

4 CONCLUSION AND FUTURE WORK

It is clear that the results are much better for the static analysis method even though there is no indicator that helps us predict the behavior of the program. In the dynamic analysis method, since the feature vector only takes into account the frequency of each API call made during the execution and does not consider the sequence of the calls themselves, the information is not sufficient to yield higher results. In the static analysis method, both the sequence of calls, transfer of control and the selection of instructions that are important to the benign and malicious programs leads to better results.

The sequence of API Calls could be used to better predict the behavior of malware in dynamic analysis in addition to other parameters like frequency. In the static analysis method, though the direction of each edge was taken into consideration, the condition upon which each branch occurs was not included in the feature vector, The inclusion of this information could give a more comprehensive feature vector.

References

[1] Anju, SS and Harmya, P and Jagadeesh, Noopa and Darsana, R. *Malware detection using assembly code and control flow graph optimization.* Proceedings of the 1st Amrita ACM-W Celebration on Women in Computing in India

[2] Zhao, Zongqu. *A virus detection scheme based on features of control flow graph.* 2011 2nd International Conference on Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC)

[3] *https://github.com/radareorg/radare2.*

[4] Bragen, Simen Rune. *Malware detection through opcode sequence analysis using machine learning.* 2015

[5] *https://github.com/tgrzinic/phd-dataset.*

[6] *https://drive.google.com/drive/folders/1juNrSMY8lQHzfwI7YVlW1yiJId8j1H-9.*