Fast Automated Unpacking and Classification of Malware

Silvio Cesare

Deakin University

<silvio.cesare@gmail.com>

Who am I and where did this talk come from?

- PhD student at Deakin University.
  - DUPRA scholarship holder.
- Industry experience as a software architect.
  - 50,000+ LOC committed to the Qualys scanner.
- Presented at CanSecWest, Blackhat, Ruxcon, and academic conferences.
- This talk is on my Masters research at CQUniversity.

Introduction

Background to the Study

- Malware is a significant problem.
- Used for criminal enterprise and financial gain.
- Static and dynamic approaches to malware analysis and detection.
- Traditional detection systems have used static string signatures.
- Static string signatures perform poorly with malware variants.

Aim and Scope of the Research

- Aim
  - To efficiently and effectively detect and classify malware.

- Scope
  - Windows executable programs only.
  - Code unpacking, but not general deobfuscation.
  - Static analysis for the classification.

Major Contributions

- An application level emulation based unpacker.
- Entropy analysis to detect end-of-unpacking.
- The use of a graph invariant to fingerprint control flow graphs.
- Decompiling control flow graphs to generate a signature.
- Set similarity search to show malware similarity.
- System implementation and evaluation.

Structure of the Presentation

1. Related Work
2. Problem Definition and Our Approach
3. Automated Unpacking
4. Malware Feature Extraction
5. Malware Classification
6. Conclusions and Future Work

Related Work

Malware Polymorphism

- Syntactic Changes
  - Dead Code Insertion
  - Instruction Substitution
  - Variable Renaming
  - Code Reordering
  - Branch Obfuscation
  - Opaque Predication Insertion
  - Code Packing
  - etc

- Semantic Changes
  - Code Insertion, Deletion, Substitution, Transposition

Malware Obfuscation Using Code Packing

- Traditional Code Packing
- Shifting Decode Frame
- Instruction Virtualization and Malware Emulators
- Resistance to Dynamic Analysis

Static Program Features

- Object File Header Attributes
- Bytes
- Instructions
- Basic Blocks
- Control Flow Graphs
- Call Graph
- API Calls
- Data Flow
- Procedure Dependence Graphs
- System Dependence Graph

Control Flow Features

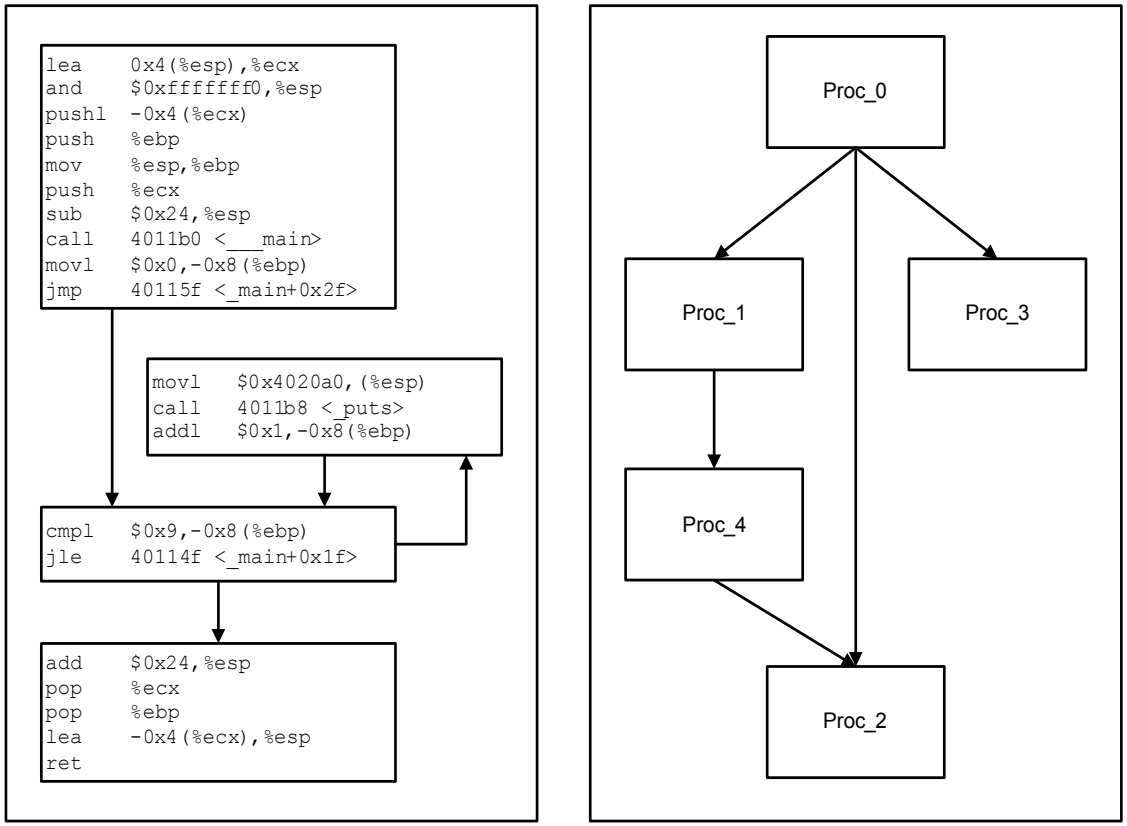

Figure . A control flow graph (left), and a call graph (right).

Comparison of Static Program Features

- Invariance under polymorphism.
- High level abstraction using program analysis.
- Robustness of feature extraction.

Classification of Program Features

- Vectors
- Strings
- Sets
- Graphs

Static Analysis of Malware

- Disassembly
- Control Flow Reconstruction
  - Opaque Predicate Detection
- Alias Analysis of Assembly Language
- Obfuscation and Limits to Static Analysis

Automated Unpacking of Obfuscated Malware

- Detecting the Code Packing Transformation
- Dynamic Approaches
  - Emulation and Virtualization
  - Dynamic Binary Instrumentation
  - Native Execution Hardware Paging
- Static Approaches
- Instruction Virtualization and Malware Emulators
  - Not completely automated.
  - Rotalume
  - Semi-automated Static Approaches

Detecting End of Unpacking

- Renovo
- Pandora’s Bochs
- OmniUnpack
- Uncover
- Hump-and-dump

Static Approaches to Malware Classification

- Classification Approaches
  - Statistical Classification
  - Instance-Based Learning
  - The Similarity Search Used in Instance-Based Learning

- Control Flow Based Classification Approaches
  - Control Flow Graphs
  - Call Graphs

Trends

- Malware Development
  - Polymorphism
  - Code packing using instruction virtualization and malware emulators
  - Anti-analysis
- Static Malware Detection and Classification
  - The use of program analysis techniques

Problem definition and our approach

Problem definition and our Approach

- Problem Definition
  - Access to a database of malware
  - Identify if a query is malicious by finding high similarity to an instance in the database.
    - This is instance-based learning.
  - Similarity is a real number [0,1].
  - Similarity greater than 0.6 indicates a variant.

- Our Approach
  - Automated Unpacking
  - Exact Flowgraph Matching
  - Approximate Flowgraph Matching

Automated Unpacking

Identifying Packed Binaries Using Entropy Analysis

- Entropy describes the amount ‘information’ in data.
- Compressed and encrypted content has high entropy.
- Typical packed malware is characterized as being compressed and encrypted.

Application Level Emulation

- Interpretation
- Windows API.
- Improvements to Emulation
- Emulating Known Sections of Code
- Verification of Emulation

Entropy Analysis to Detect End-of unpacking

- Standard approach is to exit unpacker when executing dynamically generated code.
- Doesn’t handle multiple layers.
- Solution is to check entropy of unread memory.
- Low entropy means no more data to unpack.

Discussion

- Instruction virtualization and malware emulators a problem.
- Anti-emulation a problem.
- Assumption that unpacking removes all significant obfuscations.
- Application level emulation good at working with many malware.
- Best at unpacking variants of known packers.

Evaluation

- Tested packing Windows programs hostname.exe (shown) and calc.exe prototype against 14 public packing tools.
- Results indicate accurate detection of the original entry point, and a speed suitable for adoption in real-time desktop Antivirus.

Name	Revealed code and data	Number of stages to real OEP	Stages unpacked	% of instr. to real OEP unpacked
upx	13107	1	1	100.00
rlpack	6947	1	1	100.00
mew	4808	1	1	100.00
fsg	12348	1	1	100.00
npack	10890	1	1	100.00
expressor	59212	1	1	100.00
packman	10313	2	1	99.99
pe compact	18039	4	3	99.98
acprotect	99900	46	39	98.81
winupack	41250	2	1	98.80
telock	3177	19	15	93.45
yoda's protector	3492	6	2	85.81
aspack	2453	6	1	43.41
pepsin	err	23	err	err

Name	Time (s)	Num.
mew	0.13	56042
fsg	0.13	58138
upx	0.11	61654
packman	0.13	123959
npack	0.14	129021
aspack	0.15	161183
pe compact	0.14	179664
expressor	0.20	620932
winupack	0.20	632056
yoda's protector	0.15	659401
rlpack	0.18	916590
telock	0.20	1304163
acprotect	0.67	3347105
pespin	0.64	10482466

Malware Feature Extraction

Static Analysis

- Built on a general platform for binary and program analysis.
- Disassembly
- Translation to intermediate language
- Control flow reconstruction
- Transformation of control flow to signatures

Exact Flowgraph Matching

- A graph invariant is invariant for isomorphic graphs.
- Useful as an estimate of graph isomorphism.
- Label nodes in graph using depth first ordering.
- List the nodes and their edges in order as a string.
- Hash the string to generate a signature.
- Assign a weight to the signature based on the number of basic blocks.

Approximate Flowgraph Matching

- Decompile control flow graphs to strings.
- Similar control flow graphs have similar strings.
- Edit distance between strings used to measure string similarity.
- Approximate dictionary search to index by distances uses metric trees.
- Assign a weight based on the length of the string.

The Approximate Flowgraph Signature

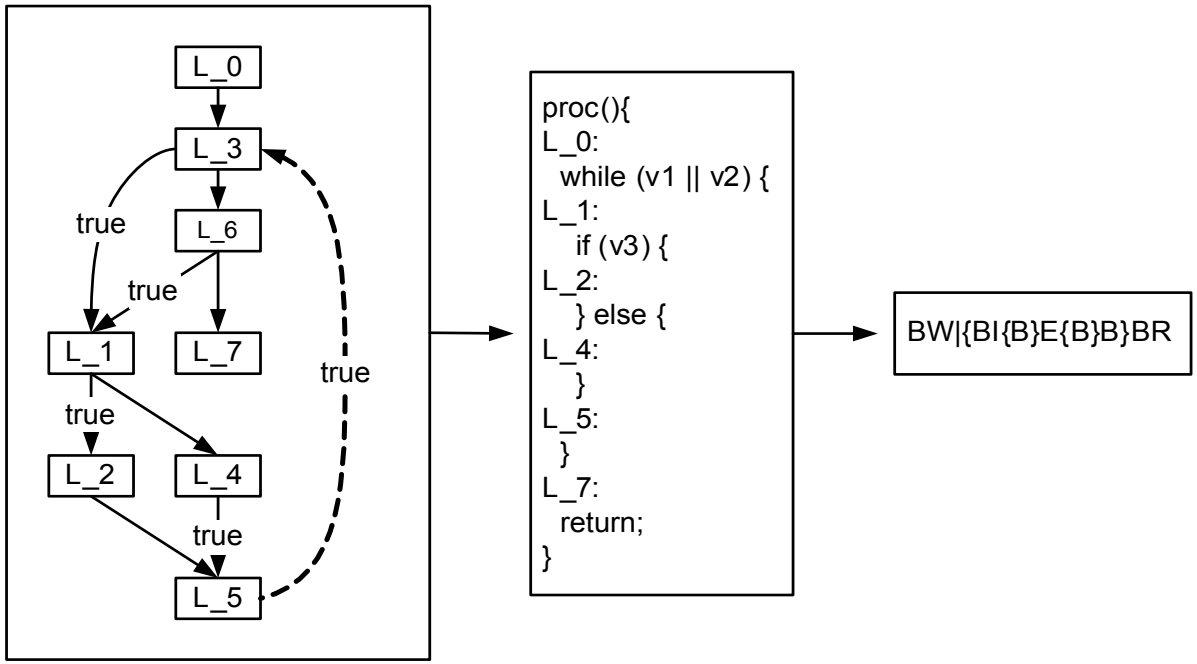

Figure . The relationship between a control flow graph, a high level structured graph, and a signature.

Discussion

- Static analysis required to successfully disassemble, and reconstruct flow graphs.
- Requires unpacking to work else the packer may be classified.
- Assumes code packing is the primary form of obfuscation.
- In most cases, if unpacking is OK, good flow graphs can be extracted.

Malware classification

Malware Classification Using Set Similarity

- Construct a mapping between features of programs.
- Based on greedy matching.
- Asymmetric similarity is sum of product of similarities between mapped features and weights.
- Similarity is product of asymmetric similarities.

$$S_x = \sum_i \begin{cases} 0, & w_{ed_i} < t \\ w_{ed_i} weight_{x_i}, & w_{ed_i} \geq t \end{cases}$$

$$S(i,d) = S_i S_d$$

The Set Similarity Search

- Find all sets from database that have a similarity > 0.6 to the query.
- Take advantage that many elements in set are fairly unique.
- Match these elements first.
- Cull sets.
- Then match frequently occurring elements.

Complexity Analysis

- Complexity of search is logarithmic to database size in expected case.
- If many duplicate or similar malware in database, complexity increases.
- Don’t store malware > 95% similar in database.

Implementation

- 100,000 LOC
- Mostly C++
- SQL Malware Database
- Python/Bash scripts for statistics and evaluation
- Java GUI
- HTML/PHP Web Interface

Effectiveness of Approximate Matching

- Tested classifying Klez (left) and Roron (right) families of malware.

	a	b	c	d	g	h
a		0.84	1.00	0.76	0.47	0.47
b	0.84		0.84	0.87	0.46	0.46
c	1.00	0.84		0.76	0.47	0.47
d	0.76	0.87	0.76		0.46	0.45
g	0.47	0.46	0.47	0.46		0.83
h	0.47	0.46	0.47	0.45	0.83	

	ao	b	d	e	g	k	m	q	a
ao		0.70	0.28	0.28	0.27	0.75	0.70	0.70	0.75
b	0.74		0.31	0.34	0.33	0.82	1.00	1.00	0.87
d	0.28	0.29		0.50	0.74	0.29	0.29	0.29	0.29
e	0.31	0.34	0.50		0.64	0.32	0.34	0.34	0.33
g	0.27	0.33	0.74	0.64		0.29	0.33	0.33	0.30
k	0.75	0.82	0.29	0.30	0.29		0.82	0.82	0.96
m	0.74	1.00	0.31	0.34	0.33	0.82		1.00	0.87
q	0.74	1.00	0.31	0.34	0.33	0.82	1.00		0.87
a	0.75	0.87	0.30	0.31	0.30	0.96	0.87	0.87	

Effectiveness of Exact Matching

- Klez, Netksky, Roron (Clockwise)

	a	b	c	d	g	h
a		0.76	0.82	0.69	0.52	0.51
b	0.76		0.83	0.80	0.52	0.51
c	0.82	0.83		0.69	0.51	0.51
d	0.69	0.80	0.69		0.51	0.50
g	0.52	0.52	0.51	0.51		0.85
h	0.51	0.51	0.51	0.50	0.85	

	aa	ac	f	j	p	t	x	y
aa		0.74	0.59	0.67	0.49	0.72	0.50	0.83
ac	0.74		0.69	0.78	0.40	0.55	0.37	0.63
f	0.59	0.69		0.88	0.44	0.61	0.41	0.70
j	0.67	0.78	0.88		0.49	0.69	0.46	0.79
p	0.49	0.40	0.44	0.49		0.68	0.85	0.58
t	0.72	0.55	0.61	0.69	0.68		0.63	0.86
x	0.50	0.37	0.41	0.46	0.85	0.63		0.54
y	0.83	0.63	0.70	0.79	0.58	0.86	0.54	

	ao	b	d	e	g	k	m	q	a
ao		0.44	0.28	0.27	0.28	0.55	0.44	0.44	0.47
b	0.44		0.27	0.27	0.27	0.51	1.00	1.00	0.58
d	0.28	0.27		0.48	0.56	0.27	0.27	0.27	0.27
e	0.27	0.27	0.48		0.59	0.27	0.27	0.27	0.27
g	0.28	0.27	0.56	0.59		0.27	0.27	0.27	0.27
k	0.55	0.51	0.27	0.27	0.27		0.51	0.51	0.75
m	0.44	1.00	0.27	0.27	0.27	0.51		1.00	0.58
q	0.44	1.00	0.27	0.27	0.27	0.51	1.00		0.58
a	0.47	0.58	0.27	0.27	0.27	0.75	0.58	0.58	

Effectiveness of Exact Matching (cont)

- 15,409 malware.
- Collected from mwcollect Alliance honeypots.
- 94% were more than 95% similar to existing malware.
- 35% were 100% identical after unpacking.

Efficiency of Exact Matching

- 809 malware.
- 86% malware processed in under 1.3 seconds.
- 0.25s median time for benign samples.

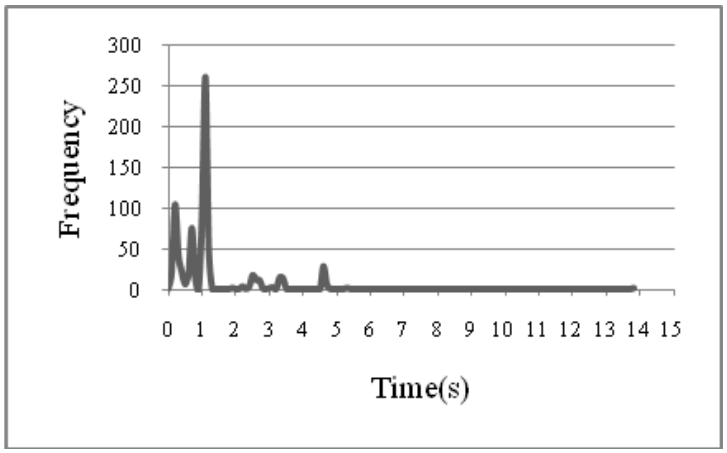

Figure . Malware processing time.

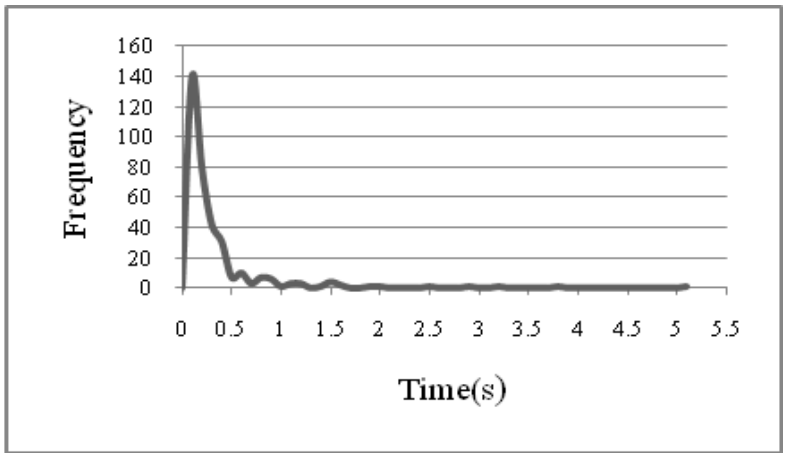

Figure . Benign processing time.

Efficiency of Exact Matching with a Synthetic Database

- Classification time only.
- Logarithmic trend line.

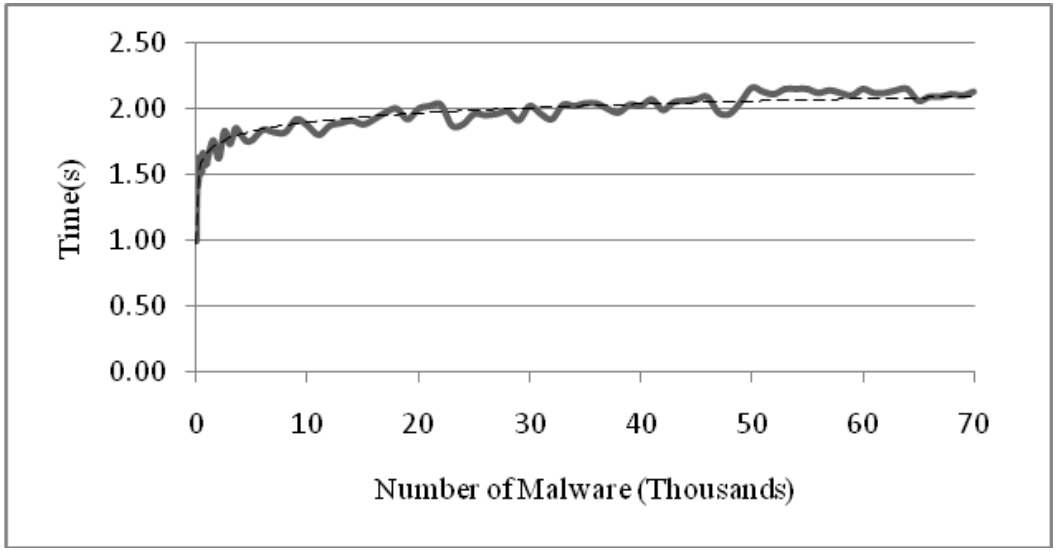

Figure . Scalability of classification.

Malwise's Resilience to False Positives

- Approximate matching (left), Exact matching (right).

	cmd.exe	calc.exe	netsky.aa	klez.a	roron.ao
cmd.exe		0.00	0.00	0.00	0.00
calc.exe	0.00		0.00	0.00	0.00
netsky.aa	0.00	0.00		0.19	0.08
klez.a	0.00	0.00	0.19		0.15
roron.ao	0.00	0.00	0.08	0.15	

	cmd.exe	calc.exe	netsky.aa	klez.a	roron.ao
cmd.exe		0.00	0.00		0.00
calc.exe	0.00		0.00	0.00	0.00
netsky.aa	0.00	0.00		0.15	0.09
klez.a		0.00	0.15		0.13
roron.ao	0.00	0.00	0.09	0.13	

Conclusions and Future Work

Future Work

- Submitted for publication a real-time approximate flowgraph matching system.
- The use of a more unpacking approaches (some work done), and packer classification (some work done).
- Ensemble learning approaches may better in practice than only flowgraph features.
- Evaluation with larger malware sets (1+ million).
- We will make a public web service next year.

Conclusions

- Thesis will be made public soon.
- Program analysis and flowgraph based classification can be performed in real or near real-time on the desktop.
- It is effective at identifying malware variants.

Questions?