A static, packer-agnostic filter to detect similar malware samples

Grégoire Jacob[1,3], Paolo Milani Comparetti[2], Matthias Neugschwandtner[2], Christopher Kruegel[1], Giovanni Vigna[1]
The authors thank **André Grégio** for presenting this paper

[1] University of California, Santa Barbara / [2] Vienna University of Technology
[3] Télécom SudParis

Fri Jul 27 2012

Introduction: the malware proliferation

How many unique malware samples are we dealing with?

- Few original malware families (large portions of shared source code)
- Humongous number of distinct samples in each family
- Sample generation by re-packing (compression, encryption)

Why does it hinder our actual techniques?

- The number of samples makes any manual analysis impossible
- Solutions based on static analysis?
  - Packing make static and signature-based approaches intractable
  - Generic unpacking mainly relies on dynamic approaches
- Solutions based on dynamic analysis?
  - Packing becomes transparent in dynamic analysis
  - Increasing needs in resources to instrument the samples (infrastructures based on virtual machines *e.g. Anubis, CWSandbox, Norman Sandbox, ThreatExpert*)

Introduction: prioritizing submissions

How to prioritize submissions to dynamic analysis systems?

- **Detection of similar malware samples:** malware samples from the same family exhibit an almost identical behavior while running
- **Priority Policy:**
  - analyze new samples first to identify new techniques
  - re-analyze samples from a same family to find evolutions (*e.g.* new C&C servers)
- **Requirement: a static and packer-agnostic similarity measure**

Our approach: code signals similarity

- The executable structure is easily tampered with
- The executable code is more reliable but hidden by packing
- Packing algorithms (compression, encryption) have weaknesses: **similarity in the code signal (distribution) is preserved**

Introduction: packing weaknesses

Packing algorithms

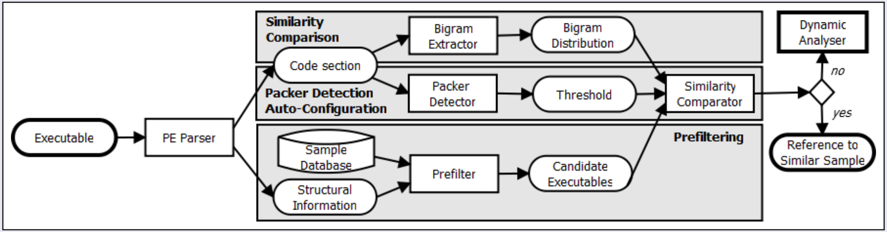
- Compression: dictionary-based (*e.g.* $LZ77$), range or entropy encoding
- Encryption: block encryption by arithmetic operations (*e.g.* $+$, $\oplus$)

	Compression	Encryption
Process	Deterministic	Non-deterministic (key)
Byte-sequences	Substitution by compressed symbols	Substitution by encrypted symbols
Byte-distributions	Preserved over compressed symbols	Permuted over encrypted symbol
Byte-alignment	Broken	Preserved

- Properties:
  - Destroyed similarity between an original code and its packed version
  - **Similarity between similar codes preserved by packed versions:** two similar code distributions remain similar after packing if the byte alignment and permutation are handled

System: filter overview

System architecture



System: similarity comparison

Code signal extraction

- Code signal = Bigram distribution of raw bytes over the code section
  Distribution is extracted **without disassembly nor unpacking**
- **Bit-shifting window** to handle the **byte-aligment destruction**
- **Sorted distribution** to handle the **encryption permutation**

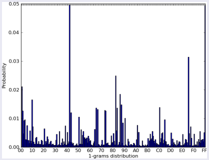
Original data:	{ 1 0 0 0 0 1 0 1 (85), 1 0 1 1 1 1 1 0 (BE), 1 1 1 1 1 1 1 1 (FF), 0 0 0 1 0 1 0 1 (15) }
Byte shifting window:	{ 1 0 0 0 0 1 0 1 1 0 1 1 1 1 1 0 (85BE), 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 (BEFF) }
Bit shifting window:	{ 1 0 0 0 0 1 0 1 1 0 1 1 1 1 1 0 (85BE), 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 (BEFF),
	0 0 0 0 1 0 1 1 0 1 1 1 1 1 0 1 (0B7D), 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 (7DFE),
	0 0 0 1 0 1 1 0 1 1 1 1 1 0 1 1 (16FB), 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 0 (FBFC),
	0 0 1 0 1 1 0 1 1 1 1 1 0 1 0 1 (2DF7), 1 1 1 1 0 1 1 1 1 1 1 1 1 0 0 0 (F7F8),
	0 1 0 1 1 0 1 1 1 1 1 0 1 0 1 1 (5BEF), 1 1 1 0 1 1 1 1 1 1 1 1 0 0 0 1 (EFF1),
	1 0 1 1 0 1 1 1 1 1 0 1 0 1 1 1 (B7DF), 1 1 0 1 1 1 1 1 1 1 1 0 0 0 1 0 (DFE2),
	0 1 1 0 1 1 1 1 1 0 1 0 1 1 1 1 (6FBF), 1 0 1 1 1 1 1 1 1 1 0 0 0 1 0 1 (BFC5),
	1 1 0 1 1 1 1 1 0 1 0 1 1 1 1 1 (DF7F), 0 1 1 1 1 1 1 1 1 0 0 0 1 0 1 0 (7F8A) ... }

- Heuristic over name, access rights and size to locate the code section

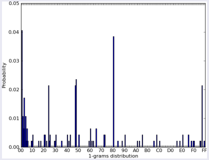
System: similarity comparison

Code signal extraction

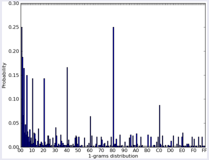
- **Bit-shifting window** to handle the **byte-aligment destruction**
- *NsPack*: LZMA (dictionary-based with range encoding)



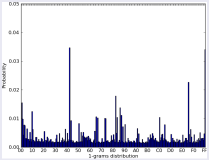
Original sample A



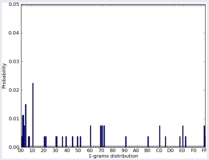
LZMA compressed A (byte-shifting window)



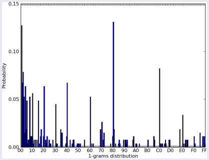
LZMA compressed A (bit-shifting window)



Original sample B



LZMA compressed B (byte-shifting window)

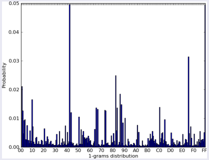


LZMA compressed B (bit-shifting window)

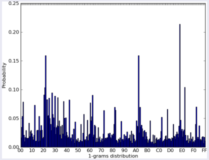
System: similarity comparison

Code signal extraction

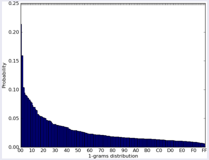
- **Sorted distribution** to handle the **encryption permutation**
- *PolyENE*: Arithmetic encryption (random operation: xor, add, rot)



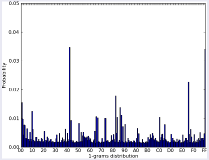
Original sample A



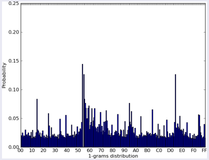
XOR encrypted A
(unsorted distribution)



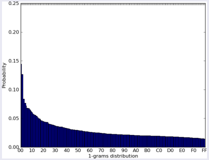
XOR encrypted A
(sorted distribution)



Original sample B



ADD encrypted B
(unsorted distribution)



ADD encrypted B
(sorted distribution)

System: similarity comparison

Code signal comparison

- Chi-square test between code signal
- Similarity threshold determined according to the packer detector
- Similarity candidates determined by the sample prefilter

System: packer detection

Detection heuristics

- Packers tend to closer to a random signals:
- Statistical tests similar to the evaluation of PRNG:
  - T1 - Uncertainty: Code entropy.
  - T2 - Uniformity: χ^2 between the code and an equiprobable distribution.
  - T3 - Run: Longest sequence of identical bytes in the code.
  - T4 - 1^{st}-order dependency: Autocorrelation coeff. of the code at lag 1.
- Detection and coarse-grain classification:
  unpacked , compressed, encrypted, multi-layer encrypted code

Threshold tuning

- Packers generate code signals closer to random signals
- Similarity of sorted signals increases with the level of packing
- Similarity threshold is tightened according to the level of packing

System: candidate selection

Sample pre-filtering

- Reduction in the number of computation
- Pre-filter rules based on characteristic features
  Features with high entropy and resilience to packing:
  - Size-based filter: range of binary size
  - PE-based filter: PE header fields

Location	Name	H	$Card$
DOS Header	`AddressNewExeHeader`	1.9	13
NT Header	`Characteristics`	0.7	7
Optional Header	`(min/maj)LinkerVersion`	0.7	6
	`CodeBase`	0.9	6
	`ImageBase`	0.4	5
	`(min/maj)OSVersion`	0.4	4
	`(min/maj)ImageVersion`	0.5	4
	`(min/maj)SubsystemVersion`	0.5	4
	`Subsystem`	0.2	2
	`DllCharacteristics`	0.8	7
	`SizeStackReserve`	0.3	4
	`SizeStackCommit`	0.4	5

Evaluation: dataset presentation

Controlled experiments

- **Goal:** tune the filter and determine the thresholds
- S_1 : 384 distinct samples from *Windows*, *OpenOffice*, shareware
- S_2 : 65 bots from the *SdBot* and rBot families, with version numbers
- P : *UPX*, *FSG*, *NsPack*, *WinUPack* (compressors), *Yoda's Cryptor*, *PolyENE* (cryptors), *tElock*, *Allaple* (multi-layer cryptors)

Large-scale experiments

- **Goal:** verify the precision, scalability and robustness of the filters
- 794,665 malware samples from *Anubis*
- 91,522 behavioral clusters from dynamic analysis
- **Ground truth:**
  structural similarity (sections sizes and hashes)
  and behavioral similarity (system call profiles) combined

Evaluation: metrics presentation

Precision metrics

- **Metrics:**

$$TH = nb\ similar\ samples\ flagged\ as\ similar$$
$$\frac{+\quad nb\ unique\ samples\ flagged\ as\ unique}{nb\ submitted\ samples}$$
$$FH = \frac{nb\ dissimilar\ samples\ flagged\ as\ similar}{nb\ submitted\ samples}$$
$$M = \frac{nb\ similar\ samples\ flagged\ as\ dissimilar}{nb\ submitted\ samples}$$

- **Granularity:**

$(f)-$ *two samples are similar if they belong to the same family*

$(v)-$ *two samples are similar if they belong to the same family and have the same version*

Evaluation: controlled experiments

Packer detection

- S_1 packed with packers from P plus *Allaple*
- **Results:**

Name	Unpacked	Packed	Compr.	Crypt.	MLCrypt.
Unpacked	99.74%	00.26%	00.26%	00.00%	00.00%
Compressors	11.80%	88.20%	87.21%	00.72%	00.27%
Crypters	12.00%	88.00%	17.53%	68.51%	01.96%
Multi-layers	02.42%	97.58%	00.28%	71.58%	25.72%
Packed	08.74%	91.26%	N/A	N/A	N/A

Evaluation: controlled experiments

Threshold selection

- S_1 and S_2 packed with packers from P plus *Allaple*
- **Selection:**
  - Minimizing false positive (FP) while maximizing true hits (TH)
  - Two sets of thresholds depending on the granularity
- **Results:**

	Family granularity thresholds			
Packer	Thrsh.	TH(f)	FH(f)	M(f)
None	0.0020	99.8%	00.2%	00.0%
Comp.	0.0018	97.5%	00.3%	02.2%
Crypt.	0.0015	89.7%	00.2%	10.1%
MLCrypt	0.0013	93.7%	00.3%	06.0%
Average	-	95.2%	00.3%	04.5%

- Lower precision for crypters:
  Encryption blocks larger than bigrams and additional key variations introduce some diffusions between bigrams:

  Encryption is no longer a perfect permutation

Evaluation: controlled experiments

Threshold selection

- S_1 and S_2 packed with packers from P plus *Allaple*
- **Selection:**
  - Minimizing false positive (FP) while maximizing true hits (TH)
  - Two sets of thresholds depending on the granularity
- **Results:**

	Version granularity thresholds			
Packer	Thrsh.	TH(f)	FH(f)	M(f)
None	0.0012	98.0%	00.2%	01.8%
Comp.	0.0008	93.7%	00.3%	06.0%
Crypt.	0.0006	90.0%	00.2%	09.8%
MLCrypt	0.0004	84.2%	00.1%	15.7%
Average	-	91.3%	00.2%	08.5%

- Lower precision for crypters:
  Encryption blocks larger than bigrams and additional key variations introduce some diffusions between bigrams:
  Encryption is no longer a perfect permutation

Evaluation: large-scale experiments

Precision and reduction factor

- Maintained precision:

Similarity Thresholds				Accuracy			Reduction
U	C	E	MLE	TH	FH	M	Factor
0.0020	0.0018	0.0015	0.00130	91.1%	00.7%	09.2%	4.84
0.0012	0.0008	0.0006	0.00040	84.6%	00.5%	14.9%	3.79
0.0005	0.0003	0.0002	0.00008	74.4%	00.3%	25.3%	2.71

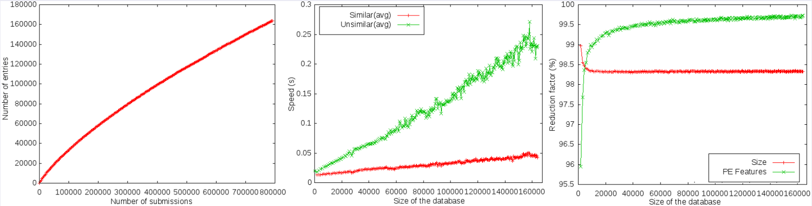
- Comparison to other approaches:

Systems	TH	FH	M
No prerequisite on the code			
Distance-based(*Filter*)	80.8%	00.7%	18.5%
Hash-based(*peHash*)	81.1%	00.6%	18.3%
Unpacked and disassembled code			
Distance-based(*Disasm*)	84.3%	00.5%	15.2%
Graph-based(*Graph*)	83.4%	00.4%	16.2%

Evaluation: large-scale experiments

Scalability

- Database growth, time per submission and prefilter efficiency:



- Comparison to other approaches (20,000 samples):

Filter	*PeHash*	*Disasm*	*Disasm*
6min	9min	239 min*	847 min*
* without unpacking			

Evaluation: large-scale experiments

Robustness

- Comparison to other approaches:

Modifications	*Disasm*	*Graph*	*peHash*	*Filter*
Modifying section permissions	✓	✓	×	✓
Changing section sizes	✓	✓	×	✓
Injecting data in sections	✓	✓	×	*
Appending new sections	✓	✓	×	*
Compression	×	×	✓	✓
Arithmetic encryption	×	×	✓	✓
Chained encryption	×	×	×	×
Strong encryption	×	×	×	×

Conclusion: static filter

Contributions

- A fast and static similarity measure not requiring disassembly
- A robust and packer-agnostic similarity measure
- A coarse-grained packer detection method based on statistical tests
- A large scale evaluation of the measure to build a submission filter
- **A reduction in analysis of submissions by a factor 3 to 5**