汎用的なアンパッキング手法の検討: 実行命令系列の類似度比較

伊沢 亮一 †　　森井 昌克 ‡　　井上 大介 †

† 国立研究開発法人情報通信研究機構
184-8795 東京都小金井市貫井北町 4-2-1
{isawa,dai}@nict.go.jp

‡ 神戸大学大学院工学研究科
657-8501 兵庫県神戸市灘区六甲台町 1-1
mmorii@kobe-u.ac.jp

あらまし パッキングされたマルウェアのコードを効率良く解析するため，パッキングツール（パッカー）の種類に依存しない汎用的なアンパッキング手法が求められている．本稿では汎用的なアンパッキング手法の検討を目的とし，マルウェアから得られる実行命令系列の類似度比較を行い，次の仮説を検証した: ある 2 つのパッキングされたマルウェアを比較したとき，a）それぞれの元のマルウェアが似ており，かつ異なるパッカーが使用されていれば，実行命令系列の類似度が高い部分がマルウェア本来のコードに該当し，b）異なるマルウェアかつ同じパッカーであれば，類似度が高い部分はパッカーのコードに該当する．本実験においてはこれらの仮説が成り立つことを確認した．実験の後，この仮説に基づく汎用アンパッキング手法について述べる．

Observing Instruction-Trace Similarity between Malware Samples for Generic Unpacking

Ryoichi Isawa †　　Masakatu Morii ‡　　Daisuke Inoue †

†National Institute of Information and Communications Technology.
4-2-1 Nukuikitamachi, Koganei, Tokyo 184-8795, JAPAN
{isawa,dai}@nict.go.jp

‡Graduate School of Engineering, Kobe University.
1-1 Rokkodai-cho, Nada-ku, Kobe-shi, Hyogo 657-8501, JAPAN
mmorii@kobe-u.ac.jp

Abstract Most malware samples emerging from the Internet are packed by packers to shield themselves from code analysis. For analyzing their original code efficiently, *generic unpacking* methods are strongly required. To invent this type of methods, we consider to compare the instruction traces of two given malware samples based on the following hypothesis: 1) the instruction sequences shared by them comprise the original binary if their original malware samples are in fact similar; 2) the shared sequences comprise the unpacking routine if the packers used for them are in fact similar. In this paper, we show that this hypothesis is correct at least in our experiments. After that, we discuss a generic unpacking method using this hypothesis.

1　はじめに

マルウェアの多くはパッカーと呼ばれるツールでパッキング（暗号化もしくは圧縮）されている．そのため，これらマルウェアのコードを解析するためにはマルウェア本来のコード（以下，オリジナルコード）を先に抽出しなければならない．しかし，UPX[1] や ASPack[2]，Molebox[3]，

telock[1]，Themida[4]，Armadillo[5] など，多種多様なパッカーが存在していることに加え，それらのパッキングアルゴリズムも未知なものが多いため，コード抽出は容易ではない．

コード抽出（以下，アンパッキング）を効率的に行うために，パッカーの種類に依存しない汎用的なアンパッキング手法の研究がなされてきた [6, 7, 8, 9, 10, 11, 12]．従来研究では研究者独自の着想で各パッカーに共通する特徴を発見・モデル化し，対象のマルウェアをメモリ上で実行，オリジナルコードの自己復号が終了したタイミングを検知するものが多い．その後，オリジナルコードをメモリ上から抽出もしくはメモリ上で直接解析する．しかしながら，Themida 等の複雑なパッカーに対しては，一般化されたモデルに沿った自己復号が行われず，自己復号のタイミングの検知精度が低くなることがある．

従来ではマルウェア 1 検体のみを入力することを前提としている手法が多いが，本稿ではマルウェアを 2 検体もしくは複数入力することによりアンパッキング精度の向上を目指す．ただし，それらマルウェアに用いられているパッカーの種類などの事前知識は必要ない手法を検討する．この手法では，マルウェアに対する事前知識は必要としないが，解析者が十分な量のマルウェア検体を保持していることを前提としている．

我々はあるマルウェア 2 検体の実行命令系列を比較したとき，次の仮説が成り立つと考えている: 実行命令系列は元になったマルウェアとパッカーの組で主に決定されるが，1) 2 検体の元のマルウェアが似ており，パッカーが同じであれば実行命令系列全体の類似度が高い，2) 元のマルウェアが似ており，パッカーが異なれば類似度の高い箇所がオリジナルコード，3) 元のマルウェアが大きく異なり，パッカーが同じであれば類似度の高い箇所がアンパッキングルーチン，4) 元のマルウェアが大きく異なり，パッカーも異なれば実行命令系列全体の類似度が低い．このうち，類似度が高い箇所が 2) によるものなのか 3) によるものなのか判定することができれば，オリジナルコードの抽出が可能となると考えている．本稿では，マルウェア 3 検体と 3 つのパッカーを組み合わせてサンプルを作成し，少なくとも本稿の実験では我々の仮説が成り立つことを確認した．実験の後，汎用アンパッキング手法の検討を行う．

本稿の貢献は，元のマルウェアとパッカーの組み合わせで比較したとき，どのように実行命令系列の類似度が変化するかを調査した点にあり，この調査は汎用アンパッキング手法の発展に寄与できると考えている．

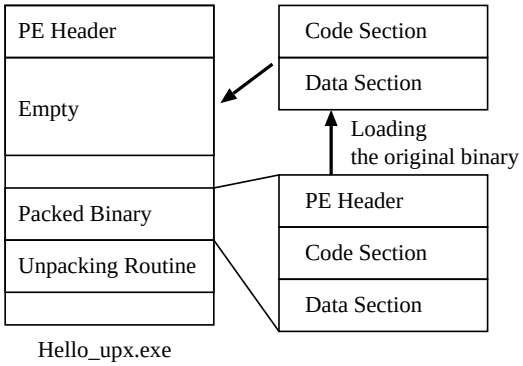

図 1: UPX でパッキングされたプログラムの構造

2 基礎知識

2.1 パッカー

パッカーの動作を説明するために，UPX でパッキングされたプログラムの構造を図 1 に示す [12]．パッキングされたプログラムは，PE ヘッダー，空のセクション，パッキングされたバイナリ，UPX のアンパッキングルーチンで構成される．このプログラムが実行されると，パッキングされたバイナリがアンパッキングルーチンによりメモリ上で復号される．そして，復号されたオリジナルコードが空のセクションに書き込まれる．その後，オリジナルコードが実行される．UPX はシンプルなパッカーであるが，Themida や telock 等，複雑なパッカーが多く存在する．

[1]Softpedia からダウンロード可能．http://www.softpedia.com/get/Programming/Packers-Crypters-Protectors/Telock.shtml

Address	Instruction
7c95be1b	push esi
7c95be1c	push dword ptr [ebp+0x8]
7c95be1f	call 0x7c95bdab
7c95bdab	mov edi, edi
...	...
0041081c	push edi
0041081d	or ebp, 0xffffffff
00410820	jmp 0x410832
00410832	mov ebx, dword ptr [esi]
...	...
00407f21	call 0x4017f0
004017f0	push ebp
004017f1	mov ebp, esp
004017f3	mov eax, 0x1f94
...	...

図 2: 実行命令系列の一例

2.2 実行命令系列

PIN[13] や Valgrind[14] などの Dynamic Binary Instrumentation Tool（DBI ツール）や，QEMU[15] や Xen[16] などの仮想環境上でマルウェアを実行することにより，実行された命令郡（実行命令系列）が取得できる．実行命令系列の一例を図 2 に示す．“push” や “call” のような実行された命令が取得できる．パッキングされたマルウェアであれば，この実行命令系列にアンパッキングルーチンとオリジナルコードが含まれる．

2.3 N-gram 類似度

ある 2 つの文字列の類似度を N-gram により比較する方法として Dice coefficient が提案されている [17]．N-gram とは，与えられた文字列のうち，連続した N 個の文字列のことを指す．本稿では実行命令系列を文字列に対応付け，オペコードを文字に対応付ける．例えば，“push push call mov” であれば，2-grams は “push push”，“push call” “call mov” となり，3-grams は “push push call”，“push call mov” となる．そして，以下の Dice coefficient の計算

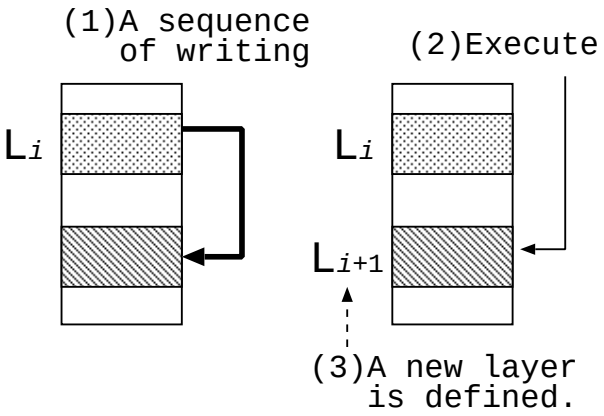

図 3: 新しいレイヤー L_{i+1} の出現条件 ($i = 0, 1, 2, \cdots$)

表 1: マルウェアとパッカーの組み合わせによる実行命令系列の類似度の変化

仮説	類似度	元検体	パッカー	
1	全体的に高	類似	同じ	
2	一部のみ高	類似	異なる	(a)
3	一部のみ高	異なる	同じ	(b)
4	全体的に低	異なる	異なる	

(a): 類似部分がオリジナルコード
(b): 類似部分がアンパッキングルーチン

方法 [17] により与えられた 2 つの実行命令系列の類似度を計算する．

$$S_{(A,B)} = \frac{2|N\text{-}grams(A) \cap N\text{-}grams(B)|}{|N\text{-}grams(A)| + |N\text{-}grams(B)|} \quad (1)$$

ここで，A と B は与えられた実行命令系列，$N\text{-}grams(\cdot)$ は N-gram の集合，$|\cdot|$ は N-gram の数をそれぞれ意味し，$0 \leq S_{(A,B)} \leq 1$ である．本稿では $S_{(A,B)}$ を N-gram 類似度と呼ぶ．

3 実行命令系列の類似度比較

3.1 類似度の比較方法

2 つの実行命令系列を比較するときに，全体を比較するのではなく，部分部分を比較するためにある基準で実行命令系列を分割する．このとき，アンパッキングルーチンとオリジナルコードがある程度事前に分かれていることが望まし

い．この分割方法として，Ugarte-Pedrero らが提案したレイヤー [18] を用いる．図 3 に示すように，プログラムが実行されているメモリにおいて，レイヤー L_i $(i = 0, 1, 2, ...)$ の命令がある領域にデータを書き込み，そのデータが実行されたときに新しいレイヤーが定義される．このとき，新しいレイヤーの番号は書き込み元のレイヤー番号に 1 を加えた値となる [2]．例えば，パッカーのアンパッキングルーチンが，次のアンパッキングルーチンを書き込み，実行するのであれば，次のレイヤーには動的に生成されたアンパッキングルーチンが格納される．

2 つのマルウェア A，B が与えられたときの比較方法を以下に示す．

1. マルウェア A，B の実行命令系列を取得する．
2. A と B の実行命令系列をそれぞれレイヤーに分割する．
3. A の各レイヤーと B の各レイヤーの全ての組に対して N-gram 類似度を計算する．

マルウェア A，B の実行命令系列を比較し，あるレイヤーのみ類似度が高いマルウェア A と B を見つける．これにより，表 1 の仮説 2 と 3 にあるように，そのレイヤーにはオリジナルコードもしくはアンパッキングルーチンが格納されていると考えられる．

ケーススタディ: マルウェア A の実行命令系列を分割したとき，3 つのレイヤー L_0^A,$\cdots$,L_2^A に分かれたとする．次に，マルウェア B の実行命令系列を分割したとき，4 つのレイヤー L_0^B,$\cdots$,L_3^B に分かれたとする．このとき，$S_{(L_0^A, L_0^B)}$, $S_{(L_0^A, L_1^B)}$, $\cdots$,$S_{(L_2^A, L_3^B)}$ の全ての組に対して N-gram 類似度を計算する．そして，仮に L_2^A, L_3^B のみ類似度が高ければ，これらの中にオリジナルコードもしくはアンパッキングルーチンが格納されていることが分かる．

[2] レイヤーの詳細な定義は文献 [18] の III 章 C 節を参照のこと．

表 2: 異なるマルウェア，同じパッカーにおける類似度（(A): Downloader with Molebox，(B): Backdoor.IRC.Bot with Molebox，L_i: i 番目のレイヤー）

		(B)		
		L_0	L_1	L_2
(A)	L_0	**0.88**	0.00	0.01
	L_1	0.00	**0.91**	0.00
	L_2	0.00	0.00	0.03

3.2 実験

我々が過去数年間に渡り収集したマルウェア数十万検体をシマンテック名で分類した．検体数で上位のカテゴリから無作為に Downloader を 1 検体，Backdoor.IRC.Bot を 1 検体を選んだ．そして，Backdoor.IRC.Bot に似ている検体として TrojanHorse のうちの 1 検体を選んだ．パッカーは Molebox，telock，Themida を用い，それぞれで独立に検体をパッキングし計 9 つのパッキングされた検体を作成した．

異なるマルウェアで同じパッカーにおける類似度を検証するために，“Downloader + Molebox”と “Backdoor.IRC.Bot + Molebox” の比較結果を表 2 に示す．L_0^A と L_0^B と L_1^A と L_1^B の類似度が高く，L_2^A と L_2^B の類似度が低い．手動で解析したところ，L_0 と L_1 にはアンパッキングルーチンが展開されており，表 1 の仮説 3 に該当する．表 3 も異なるマルウェアで同じパッカーにおける類似度を示している．L_0^A と L_0^B や L_1^A と L_1^B の類似度が高く，実際にアンパッキングルーチンが含まれていた．L_{22}^A にはオリジナルコードが含まれているが Backdoor.IRC.Bot の全てのレイヤーの類似度は 0.00 となった．

同じマルウェアで異なるパッカーにおける類似度を検証するために，“Downloader + Themida” と “Downloader + Molebox” の比較結果を表 4 に示す．L_{22}^A と L_2^B の類似度のみ高く，他は 0.00 となった．手動で確認したところ，L_{22}^A と L_2^B にオリジナルコードが展開されており，表 1 の仮説 2 に該当する．表 5 も同じマルウェアで異なるパッカーにおける類似度を示している．L_{22}^A と L_{13}^B の組および L_{22}^A と L_{14}^B の組以外は 0.00

表 3: 異なるマルウェア，同じパッカーにおける類似度（(A): Downloader with Themida，(B): Backdoor.IRC.Bot with Themida，L_i: i 番目のレイヤー，‘···’ の箇所は紙面の都合で省略.）

		(B)															
		L_0	L_1	L_2	L_3	L_4	L_5	L_6	···	L_{22}	L_{23}	L_{24}	L_{25}	L_{26}	L_{27}	L_{28}	L_{29}
(A)	L_0	**0.42**	0.00	0.00	0.00	0.00	0.00	0.00	···	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	L_1	0.00	**0.97**	0.00	0.00	0.00	0.00	0.00	···	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	L_2	0.00	0.00	**0.01**	**0.02**	0.00	0.00	0.00	···	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	L_3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	···	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	L_4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	···	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	L_5	0.00	0.00	0.00	**0.10**	0.00	**0.66**	0.00	···	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	L_6	0.00	0.00	0.00	0.00	**0.02**	0.00	**0.01**	···	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	···	···	···	···	···	···	···	···	···	···	···	···	···	···	···	···	···
	L_{22}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	···	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	L_{23}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	···	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	L_{24}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	···	0.00	0.00	0.00	**0.85**	0.00	0.00	0.00	0.00
	L_{25}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	···	0.00	**0.24**	0.00	**0.02**	0.00	0.00	0.00	0.00
	L_{26}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	···	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	L_{27}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	···	0.00	0.00	0.00	0.00	**0.03**	0.00	0.00	0.00
	L_{28}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	···	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	L_{29}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	···	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	L_{30}	0.00	0.00	0.00	0.00	**0.01**	0.00	0.00	···	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

表 4: 同じマルウェア，異なるパッカーにおける類似度（(A): Downloader with Themida，(B): Downloader with Molebox，L_i: i 番目のレイヤー，‘···’ の箇所は全て 0.00.）

		(B)		
		L_0	L_1	L_2
(A)	L_0	0.00	0.00	0.00
	···	···	···	···
	L_{21}	0.00	0.00	0.00
	L_{22}	0.00	0.00	**0.51**
	L_{23}	0.00	0.00	0.00
	···	···	···	···
	L_{30}	0.00	0.00	0.00

表 5: 同じマルウェア，異なるパッカーにおける類似度（(A): Downloader with Themida，(B): Downloader with telock，L_i: i 番目のレイヤー，‘···’ の箇所は全て 0.00.）

		(B)			
		L_0	···	L_{13}	L_{14}
(A)	L_0	0.00	···	0.00	0.00
	···	···	···	···	···
	L_{21}	0.00	···	0.00	0.00
	L_{22}	0.00	···	**0.01**	**0.36**
	L_{23}	0.00	···	0.00	0.00
	···	···	···	···	···
	L_{30}	0.00	···	0.00	0.00

となり，L^B_{13} と L^B_{14} にオリジナルコードが展開されていることを確認した．L^A_{22} と L^B_{13} の類似度は L^A_{22} と L^B_{14} と比べて低いが，0.00 に比べると偏差はでている．また，表 6 は類似したマルウェアで異なるパッカーにおける類似度を検証した．オリジナルコードのが展開されているレイヤーの類似度が高くなった．

最後に，異なるマルウェアで異なるパッカーの類似度を比較した結果を表 7 に示す．全てのレイヤーの組で類似度が 0.00 となった．

4 まとめ

本稿では，ある 2 つのマルウェアの実行命令系列を比較したとき，類似度が高い部分はオリジナルコードもしくはアンパッキングルーチンの可能性について検証し，本実験においてはこれが正しいことを確認した．ただし，本稿では，類似度が高い部分がオリジナルコードかアンパッキングルーチンかを判定する方法については言及していない．

この検証結果を基に次のようなアンパッキング手法が考えられる．あるマルウェアをアンパッキングするとき，すでに保有しているマルウェアと順に比較する．類似度の高いレイヤーをいくつか見つかるまで比較を繰り返し，それらレイヤーを総合的に見ることでオリジナルコードかアンパッキングルーチンかを判定できないか，今後検証していく．もしこの判定が可能であれば，汎用アンパッキングが実現できる．

実際に，マルウェアのソースは使い回されていることも多いため，解析対象のマルウェアと

表 6: 似ているマルウェア，異なるパッカーにおける類似度（(A): Backdoor.IRC.Bot with telock，(B): TrojanHorse with Molebox，L_i: i 番目のレイヤー，‘· · ·’ の箇所は全て 0.00.）

		(B)		
		L_0	L_1	L_2
(A)	L_0	0.00	0.00	0.00
	· · ·	· · ·	· · ·	· · ·
	L_{12}	0.00	0.00	0.00
	L_{13}	0.00	0.00	**0.17**
	L_{14}	0.00	0.00	**0.27**

表 7: 異なるマルウェア，異なるパッカーにおける類似度（(A): Backdoor.IRC.Bot with telock，(B): Downloader with Molebox，L_i: i 番目のレイヤー，‘· · ·’ の箇所は全て 0.00.）

		(B)		
		L_0	L_1	L_2
(A)	L_0	0.00	0.00	0.00
	· · ·	· · ·	· · ·	· · ·
	L_{14}	0.00	0.00	0.00

保有しているマルウェアのオリジナルコードが似ていることは十分に考えられる．また，解析対象のマルウェアが，保有していない未知のマルウェアであったとしても，パッカーが未知のものでなければアンパッキングルーチンの部分を判定することで結果的にオリジナルコードの部分を判定できることも考えられる．

参考文献

[1] Oberhumer, M. F., Molnar, L. and Reiser, J. F.: UPX: Ultimate Packer for eXecutables, `http://upx.sourceforge.net/` (2015).

[2] StarForce Technologies Ltd.: ASPack Software, `http://www.aspack.com/` (2015).

[3] DesaNova Ltda.: Virtualization and Protection Tools at Molebox.com, `http://www.molebox.com/` (2015).

[4] Oreans Technologies: Themida, `http://www.oreans.com/themida.php` (2015).

[5] The Silicon Realms Toolworks: Software protection, `http://www.siliconrealms.com/` (2015).

[6] Royal, P., Halpin, M., Dagon, D., Edmonds, R. and Lee, W.: PolyUnpack: Automating the Hidden-Code Extraction of Unpack-Executing Malware, *Proceedings of the 22nd Annual Computer Security Applications Conference (ACSAC'06)*, pp. 289–300 (2006).

[7] Martignoni, L., Christodorescu, M. and Jha, S.: OmniUnpack: Fast, Generic, and Safe Unpacking of Malware, *Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC'07)*, pp. 431–441 (2007).

[8] Kang, M. G., Poosankam, P. and Yin, H.: Renovo: A Hidden Code Extractor for Packed Executables, *Proceedings of the 5th ACM workshop on Recurring Malcode (WORM'07)*, New York, NY, USA, ACM, pp. 46–53 (2007).

[9] Guo, F., Ferrie, P. and Chiueh, T.-C.: A Study of the Packer Problem and Its Solutions, *Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection (RAID'08)*, Berlin, Heidelberg, Springer-Verlag, pp. 98–115 (2008).

[10] Kawakoya, Y., Iwamura, M. and Itoh, M.: Memory Behavior-Based Automatic Malware Unpacking in Stealth Debugging Environment, *Proceedings of the 5th International Conference on Malicious and Unwanted Software (MALWARE'10)*, pp. 39–46 (2010).

[11] Kim, H. C., Orii, T., Yoshioka, K., Inoue, D., Song, J., Eto, M., Shikata, J., Matsumoto, T. and Nakao, K.: An Empirical Evaluation of an Unpacking Method Implemented with Dynamic Binary Instrumentation, *IEICE Transactions*, Vol. 94-D, No. 9, pp. 1778–1791 (2011).

[12] Isawa, R., Inoue, D. and Nakao, K.: An Original Entry Point Detection Method with Candidate-Sorting for More Effective Generic Unpacking, *IEICE Transactions*, Vol. E98-D, No. 4, pp. 883–893 (2015).

[13] Intel Corporation: Pin - A Dynamic Binary Instrumentation Tool, available at *http://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool*.

[14] Nethercote, N. and Seward, J.: Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation, *SIGPLAN Not.*, Vol. 42, No. 6, pp. 89–100 (2007).

[15] Bellard, F.: QEMU, `www.qemu.org/`.

[16] Xen Project: The Xen Project, `http://www.xenproject.org/` (2013).

[17] Brew, C. and McKelvie, D.: Word-pair extraction for lexicography, *In Proc. of the 2nd Intl Conf. on New Methods in Language Processing*, pp. 45–55 (1996).

[18] Ugarte-Pedrero, X., Balzarotti, D., Santos, I. and Bringas, P. G.: SoK: Deep Packer Inspection: A Longitudinal Study of the Complexity of Run-Time Packers, *In Proc. of2015 IEEE Symposium on Security and Privacy*, pp. 659–673 (2015).