シンボリック実行による解析環境検知マルウェアの解析手法改善

加藤 俊樹[1,a)]　掛井 将平[1]　齋藤 彰一[1]

概要：マルウェアの一部は解析環境検知機能を持ち，これに対抗するにはその条件を特定することが重要である．その手法としてシンボリック実行があるが，精度に課題がある．精度低下の原因には，Windows API のシミュレーションと探索手法が挙げられる．シンボリック実行では，解析負荷軽減のために DLL をロードせずに各関数を模擬する関数を実装しているが，模擬関数がない場合には返り値にシンボル値を設定するのみである．したがって，シミュレーションの課題として模擬関数がない場合に返り値以外の条件抽出が不十分となることが挙げられる．また，探索手法の課題としては，短い経路の条件を優先的に抽出するため，詳細な検知回避条件が見落とされる場合があることが挙げられる．本論文では，これらの課題解決のために，まず，模擬関数がない場合のシミュレーションに関して，各 Windows API の個別対応から判明した引数へのシンボル値設定や引数が関数の場合の関数呼び出し処理を共通処理として拡張することを提案する．また，検知回避条件を反転して再解析することで，詳細な条件を抽出する手法を提案する．これら提案によって，マルウェア対策の精度と効率の向上を目指す．

キーワード：マルウェア，シンボリック実行，解析環境検知，angr

Improvement of Methods for Analysing Environment-Sensitive Malware Using Symbolic Execution

Toshiki Kato[1,a)]　Shohei Kakei[1]　Shoichi Saito[1]

Abstract: Some malware has an analysis environment detection function. It is essential to identify the conditions for such detection to counter it. Symbolic execution is one such method, but its accuracy has been challenged. The accuracy degradation stems from the simulation and search methods of Windows APIs. In symbolic execution, each function is simulated without loading the DLL to reduce the analysis load. However, only the return value is set symbolically without a simulated function, leading to insufficient condition extraction. Additionally, the search method sometimes cannot detect detailed detection evasion conditions because it prioritizes shorter paths. We propose two approaches to address these issues. For simulations without individual simulated functions, it suggests setting symbolic values for arguments and handling argument calls based on individual Windows API analysis. It also proposes extracting detailed conditions by inverting and reanalyzing detection evasion conditions. These approaches aim to improve the accuracy and efficiency of anti-malware countermeasures.

Keywords: malware, symbolic execution, anti-analysis, angr

1. はじめに

マルウェアの一部には環境に応じて動作を変えるマルウェアが存在している [1]．このようなマルウェアは，自

[1] 名古屋工業大学
Nagoya Institute of Technology
[a)] t.kato.991@nitech.jp

© 2024 Information Processing Society of Japan

This paper is work in progress and not peer-reviewed.

身が実行されている環境がデバッガや VM 等の解析環境であるか検知する機能を有している．この機能によりマルウェアが解析環境を検知した場合，悪性コードを実行しないことが多く，解析者はマルウェア本来の悪性の動きを解析することができない．このような解析環境検知マルウェアに対抗するには，マルウェアによって解析環境を検知されない条件を特定することが重要である．この条件を特定することで，検知されない解析環境を構築でき，詳細なマルウェアの動作の観測が可能となる．マルウェアの解析手法には大別して動的解析と静的解析の 2 種類がある．

動的解析は，マルウェアを実行して，その動作を観測することでマルウェアを解析する手法である．動的解析のメリットは，パッキングや難読化に耐性があることが挙げられる．一方，デメリットは，解析環境検知機能を持つマルウェアに検知されることや実行パスのカバレッジが低いことが挙げられる．動的解析を用いてマルウェアが解析環境を検知する条件を調べる研究として環境の設定が異なる複数の仮想環境を用いて，それぞれの環境においてマルウェアを実行することにより，動作の違いを観察し，どのような環境で検知されるのか突き止める研究 [2] がある．

一方，静的解析は，マルウェアを実行しないで解析する手法である．静的解析の代表的な手法の例としては，シンボリック実行による解析が挙げられる．静的解析のメリットは，マルウェアを実行しないため，解析環境を検知されないことや実際のプログラムフローでは到達しない経路や特定の環境に依存している経路を解析できることが挙げられる．このためコードカバレッジが高くなる．一方，デメリットは，パッキングや難読化が施されている場合に解析が困難になることが挙げられる．静的解析を用いて解析環境検知マルウェアによる検知を回避する条件を調べる研究として，シンボリック実行を用いた研究 [3] がある．

本論文では，環境構築の容易さと，コードカバレッジが高いという点で優れている静的解析の一種であるシンボリック実行を用いた，解析環境検知機能を有するマルウェアに対する検知回避条件の抽出に関して改善を目指す．また，シンボリック実行を行うためのツールとして angr [4–7] を採用する．angr による解析時の課題として，解析対象プログラムにおいて Windows API が呼び出される際のシミュレーション不足が挙げられる．シンボリック実行は解析負荷削減のために，DLL に含まれる本来の関数の中身を解析しない．その代わりにシミュレーションのための関数をあらかじめ用意しておき，そのシミュレーション関数に処理を移す仕組みである．このシミュレーション関数は Windows API ごとに用意されるが，正確にシミュレーションされていない関数や，そもそも用意されていない関数が多数ある．したがって，これらシミュレーション関数の実装が原因で解析環境検知マルウェアの検知回避条件が取得できない場合がある．また，別の課題として解析時の探索手法が挙げられる．静的解析のメリットとしてコードカバレッジが高いことを挙げたが，逆に経路が多すぎると解析負荷が高くなる．そのため，シンボリック実行では解析コストを低くしつつ，有用な情報を取得できる探索手法の実装が必要とされる．以上 2 つの課題に対して改善と効率化の提案を行う．

本論文の構成は以下のとおりである．まず第 2 章において，マルウェアの解析環境検知に対する検知回避条件抽出に関する関連技術について述べ，第 3 章で関連研究について述べる．第 4 章では，提案について述べ，第 5 章で提案手法の評価について述べる．第 6 章では，今後の課題について述べる．第 7 章でまとめる．

2. 関連技術

本章では，解析環境検知マルウェアの検知回避条件を抽出する手法で使用するシンボリック実行とそのツールである angr について概要を述べる．

2.1 シンボリック実行

シンボリック実行による解析では，プログラムへの入力値や変数に対して具体値を使用する代わりに，任意の値を表すシンボル値を使用することができる．シンボル値がプログラム内の分岐条件として利用されている場合，任意の経路に分岐する場合の満たすべき条件がシンボル値に付与される．これを事前に定義した終了地点まで行うことで，各経路を実行するために必要な条件を取得できる．

2.2 angr

angr はシンボリック実行を行うためのツールである．angr による基本的な解析の流れを述べる．まず解析のために開始アドレス，到達アドレス，回避アドレスを設定する．その後，開始アドレスから到達アドレスまで回避アドレスを通過せずに到達できる経路を探索し，その経路を通過するための条件（分岐処理時の変数の値域等）を抽出する．また，経路を探索する手法も様々用意されており，選択することが可能である．これを解析環境検知マルウェアに適用することで，解析環境検知を回避する条件を抽出することが可能となる．

angr は，Windows API をシミュレーションする際に SimProcedure クラスと呼ばれるクラスを使用する．各 Windows API に対するシミュレーション関数はこの SimProcedure クラスを継承した各 Windows API を模したクラスで定義される．シミュレーション関数がない Windows API に関しては，ReturnUnconstrained 関数にシミュレーションが任されており，返り値のシンボル値化等の処理が行われる．

© 2024 Information Processing Society of Japan

3. 関連研究

本章では，関連研究として，angr を利用して解析環境検知の回避条件抽出を行った研究について述べる．

3.1 angr による解析環境検知の回避条件抽出

angr を利用して解析対象プログラムに含まれる解析環境検知の回避条件を取得した研究 [3] がある．文献 [3] では，angr を使用して，解析環境検知手法が多数実装されたツールである Pafish [8] を解析して検知回避条件の抽出を行い，その精度について調査を行った．Pafish には，文献 [3] 発表時点において合計 55 個のデバッガ検知や VM 検知が実装されていた．文献 [3] に記された解析結果によると，55 個の検知手法の内，検知回避条件を抽出できた検知手法が 34 個，抽出できたが条件が不十分であった検知手法が 11 個，条件が抽出できなかった検知手法が 10 個である．したがって，全体の約 40%が条件抽出に課題があることが分かる．

3.2 Symba

angr のシミュレーション関数を動的に生成して解析を行うシステム Symba [9] がある．Symba では，解析時に注目する Windows API を決定し，その関数のシミュレーション関数では返り値だけでなく引数にもシンボル値を設定することで，より詳細な解析環境検知の回避条件抽出を試みている．

Symba では，まず，事前に解析対象プログラム内で利用されている Windows API の中で注目する関数を決めて，その関数について MSDN(Microsoft Developer Network) [10] から自動で引数や返り値の情報について取得する．そして解析に先立って取得した情報に基づいてシミュレーション関数を作成し，解析時に当該 Windows API が呼び出される際にそのシミュレーション関数に処理が移行するように設定する．その後，解析対象プログラムの制御フローグラフ情報を利用して，当該 Windows API が利用されているベーシックブロックに自動で開始アドレスを設定する．そして，この開始アドレスから条件抽出を開始し，規定のベーシックブロック個数内に条件を抽出できた場合はその地点から抽出を再開する．規定のベーシックブロック個数内で新たな条件を抽出できなかった場合は解析を終了する．以上のように，注目する Windows API の引数に関する条件の抽出に関して，自動的に開始アドレスの決定と終了条件を設定することで，従来の angr と比較してより詳細な条件の抽出と解析の簡単化を行っている．

4. 提案手法

本章では，angr による解析時の課題に関しての事前調査と，この調査結果に基づいて個々の Windows API 毎に改善した方法について述べる．さらに，これらの調査と個別対応を検討した結果を考察して得られた 2 つの提案方法について述べる．

表 1 従来の angr による Pafish の解析結果（文献 [3] の表 2 を参考に作成）

Table 1 Results of Pafish analysis by conventional angr. (Based on Table 2 in reference [3].)

解析環境検知機能の種類	条件の抽出結果			合計
	✓	△	×	
デバッガ検知	2	1		3
VM 検知（CPU 情報に基づく）	2	1	1	4
サンドボックス検知	5	3	3	11
フック検知			2	2
Sandboxie 検知			1	1
Wine 検知		1	1	2
VirtualBox 検知	1	15	1	17
VMWare 検知		7	1	8
Qemu 検知		2	1	3
Bochs 検知		1	2	3
Cuckoo 検知		1		1
rtt 検知	2	1	4	7
合計	12	33	17	62

表 2 個別対応による改善結果

Table 2 Improvement results from individualized attention.

改善結果		
改善	変化なし	悪化
37	23	2

4.1 事前調査

angr の解析時の課題として Windows API のシミュレーション関数の種類が不足していることが挙げられる．また，その実装が検知回避条件を抽出する目的には適していない実装の場合もある．したがって，シミュレーションに関して実際にどのような改善が必要かを Pafish を用いて調査した．調査の流れは，まず，本調査実施時点での最新バージョン（v0.6）の Pafish に対して angr で解析を行い，条件抽出を行う．抽出結果を**表 1** に示す．表 1 の分類は，検知手法の趣旨にある条件を抽出した場合（表 1 中の ✓），趣旨に合わない条件が抽出された場合（表 1 中の△），なにも抽出できなかった場合（表 1 中の×）である．結果は総数 62 個中 ✓ が 12 個，△が 33 個，×が 17 個である．したがって，△の 33 個と×の 17 個を合わせた 50 個の条件を本論文での改善対象とする．

4.2 個別対応による改善

改善方法の指針を得るために，効率は無視して各 Windows API に対して個別に行った改善方法について述べる．まず，具体的な改善方法について以下に示す．

- cpuid のシンボル値化

© 2024 Information Processing Society of Japan

angr では cpuid の値が固定値に設定されている．そのため，cpuid の値で分岐がある検知手法に関して，条件抽出ができない．そこで，angr のオプションによって cpuid のシンボル値化を行った．

- シミュレーション関数の追加
  主な変更内容は，シミュレーション関数に渡される引数に対するシンボル値の設定と，引数が関数の場合には当該関数内部も解析を行った上で，シミュレーション関数の解析を行う対応と，各 Windows API の固有の処理内容の簡易的なシミュレーションの実装である．
- フック検知への対応
  フック検知で参照される各 Windows API へのポインタが 0 の固定値である．それらアドレスをシンボル値に変更することで条件抽出を可能にした．
- 検知回避経路が複数ある場合への対応
  解析環境検知を回避する経路が複数存在している場合，標準状態では最初に実行される経路の条件抽出のみが可能となり，後段の条件を抽出することができない．そこで，一旦抽出した条件を反転させた条件を付与して再解析を行うことで，後段の条件までも抽出できるようにした．

この結果を**表 2** に示す．表 2 から，過半数で抽出した検知回避条件が改善していることが分かる．なお，条件抽出の精度が悪化した検知手法が 2 個存在するが，この原因は，新たに設定したシンボル値の影響で解析負荷が高くなったためである．

4.3　提案 1

提案 1 では，事前調査の個別対応から判明したシミュレーションの実装の改善に関して，共通する処理内容をまとめて処理する新たなシミュレーション関数，Return-SymbolUncosntrained 関数を提案する．提案 1 の概要図を**図 1** に示す．標準の angr のシミュレーションの実装は，各 Windows API に対するシミュレーション関数が実装されていない場合は ReturnUnconstrained と呼ばれるシミュレーション関数に処理が移行し，返り値としてシンボル値が返される．したがって，返り値が分岐の条件として利用されている場合は条件を抽出できるが，引数に関する条件抽出ができない．そこで，本提案 1 では，引数にシンボル値を設定する機能と引数が関数であった場合にその関数を解析する機能を追加した，ReturnUnconstraind 関数の拡張版である ReturnSybmolUnconstraind 関数を実装した（実際の実装では，引数の数毎に関数がある）．しかし，この実装により，扱うシンボル値の数が多くなり，解析の負荷が増加することが考えられる．そこで，angr の 2 つの機能，規定の残りメモリ使用量を下回った場合に解析を終了する機能と既定の解析時間を超えた場合に解析を終了する機能を利用して，解析の未終了や負荷による強制終了を防ぐ．この提案により，既存の angr と比較してより詳細な検知回避条件の抽出が期待でき，また，未知の解析手法に関しても，同様の改善を行うことなく，個別改善の効率の向上が期待できる．

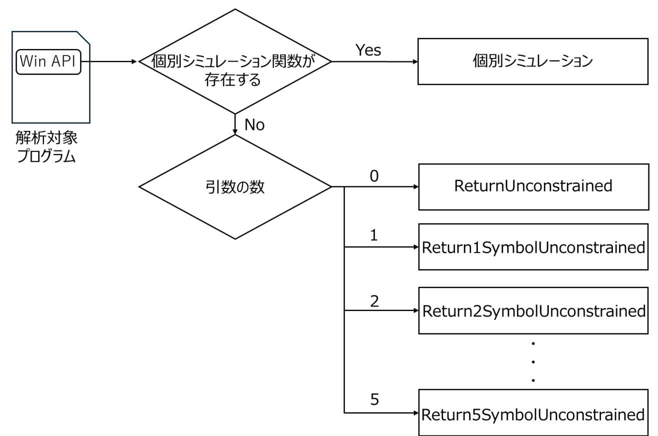

図 1　提案 1 の概要
Fig. 1　The Overview of Proposal 1.

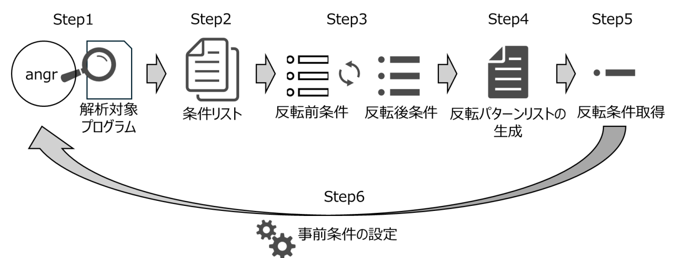

図 2　提案 2 の概要
Fig. 2　The Overview of Proposal 2.

4.4　提案 2

提案 2 では，条件抽出の漏れを防ぐために，事前条件を変えながら複数回の抽出を行う手法を提案する．具体的には，初回の条件抽出により取得した検知回避条件を反転した条件を 2 回目の条件抽出をする際の事前の条件として設定することで，1 回目の解析では探索できていない経路を探索し，その経路を通過するための条件を取得する．提案 2 の概要図について**図 2** に示す．提案 2 を用いた解析の流れについて図 2 の流れに沿って述べる．

Step 1. 通常の解析を行う

Step 2. 検知回避条件を抽出
- 例
  - unconstrained_ret_IsDebuggerPresent_1_32 == 0
  - mem_0xfffaad_1_32 <0

Step 3. 抽出した条件を反転した条件を求める
- 例
  - unconstrained_ret_IsDebuggerPresent_1_32 != 0
  - mem_0xfffaad_1_32 >= 0

Step 4. 反転条件の組み合わせのパターンを作成し，リストに保存する
- 例
  - unconstrained_ret_IsDebuggerPresent_1_32 != 0
  - mem_0xfffaad_1_32 >= 0
  - unconstrained_ret_IsDebuggerPresent_1_32 != 0，mem_0xfffaad_1_32 >= 0

Step 5. 反転条件の組み合わせパターンのリストの先頭から条件を取得 or リストが空の場合は解析終了
- 例
  - unconstrained_ret_IsDebuggerPresent_1_32 != 0

Step 6. 取り出した条件を次の解析の事前条件として設定し，Step1 へ戻る

5. 評価

本章では，提案 1，2 の評価について述べる．評価環境は，CPU は Intel Core i5-12400，メモリ 8GB，OS は Kali Linux である．また，使用するツールと解析対象プログラムのバージョンは，angr（v9.2.17），Pafish（v0.6），Al-Khaser（v0.81）である．

5.1 解析対象プログラム

解析対象プログラムとして Pafish と Al-Khaser [11] を用いる．Pafish は，マルウェアと同様に解析環境をさまざまな手法で検出するテストツールである．Pafish には，VirtualBox や VMware といった仮想環境検知を中心にデバッガ検知の機能が 62 個用意されている．al-kahser も Pafish と同様にさまざまな解析環境検知手法が用意されている．特に，Al-Khaser は 39 個のデバッグ検知手法を実装しているため，Pafish には存在しないさまざまなデバッグ検知手法に関して，評価を得ることが可能となる．

5.2 評価結果

5.2.1 提案 1 の評価結果

提案 1 を angr に実装した場合の解析結果を**表 3** に示す．提案 1 は標準状態の angr と比較して，引数に関する条件を抽出できる機能を追加している．表 3 では，まず検知手法の内，検知に Windows API を利用しているか否かよって分類している．その後，Windows API を利用している検知手法の内，引数を利用しているか否かによって分類している．そして，引数を利用している検知で文字列比較をしているか否かによって分類している．Windows API を利用していない検知と分岐条件に引数が利用されていない検知手法は，提案 1 によって改善することができないため，今回対象外である．つまり，提案 1 の改善対象は分岐条件に引数を利用している検知手法である．また，文字列比較については，解析処理の一部に文字列比較を含むか否かで結果が大きく変化したため分類している．表 3 の改善結果の分類は，標準状態の angr より多くの条件を取得できた場合（表 3 中の改善），標準状態の angr と同じ条件を抽出した場合（表 3 中の変化なし），標準状態の angr より抽出した条件が少なかった場合（表 3 中の悪化）である．

表 3 より，Pafish に関しては，対象の検知関数の総数は文字列比較がある検知関数が 21 個，文字列比較がない検知関数が 12 個である．文字列比較がある検知に関して，改善が 2 個，悪化が 19 個となっており，大幅に精度が低下している．しかし，この 19 個の検知手法は同じ関数，手法で検知を行っているため，実質問題は一つである．その他の文字列比較がない検知手法については，総数 12 個の内，改善が 5 個，変化なしが 5 個，悪化が 2 個である．

Al-Khaser に関しては文字列比較がない検知手法となっている．対象の検知関数 16 個の内，改善が 7 個，変化なしが 7 個，悪化が 2 個となっている．

まとめると，引数が関係ある検知手法に関して，Pafish と al-kahser を合わせた全体の結果としては，総数 49 個の内，14 個に関して改善した．

Pafish と Al-Khaser において改善できなかった検知手法について原因は以下の 4 つである．

- 文字列情報の消失
  Pafish の文字列比較がある 19 個の検知に関して悪化した原因は，文字列比較の際に利用されている関数にある．これらの検知に利用されている関数の本来の動作は引数で渡された文字列がシステムの特定の箇所に存在するか確認することである．提案 1 の場合，引数をシンボル値化したことにより比較する文字列がシンボル値化され，angr で特定できなくなり解析が終了する．
- シンボル値増加による解析負荷増加
  新たに設定したシンボル値の増加が原因となって改善しなかったパターンとして二つ挙げられる．一つ目は，新たに設定したシンボル値の取り得る経路が既定数以上存在しており，解析が困難であると判断されたため，事前に解析をスキップする処理が行われた結果，条件抽出ができない場合である．二つ目は，新たに設定するシンボル値が多すぎて解析負荷が高くなり，解析不可になる場合である．
- エラーの発生
  エラーの発生については，新たに設定したシンボル値を利用した処理で，シンボル値に対応していない処理がありエラーが生じる場合や，新たなシンボル値により到達した新たな経路でエラーが生じた場合が挙げられる．
- 探索手法の問題
  探索手法については，従来設定では短い経路について優先して条件を抽出するため，新たに設定したシンボル値が使用される経路に到達せずに解析が終了するこ

© 2024 Information Processing Society of Japan

表 3 提案 1 実装後の解析結果

Table 3 Analysis results after implementation of Proposal 1.

解析対象プログラム	検知手法	条件の分類		改善結果			合計
				改善	変化なし	悪化	
Pafish	Windows API	引数が関係ある	文字列比較あり	2	0	19	21
			文字列比較なし	5	5	2	12
		引数が関係ない（評価対象外）		-	19	-	19
	その他（評価対象外）	—		-	10	-	10
	合計			7	34	21	62
Al-Khaser	Windows API	引数が関係ある	文字列比較あり	0	0	0	0
			文字列比較なし	7	7	2	16
		引数が関係ない（評価対象外）		-	15	-	15
	その他（評価対象外）	—		-	8	-	8
	合計			7	30	2	39

表 4 提案 2 実装後の解析結果

Table 4 Analysis results after implementation of proposal 2.

解析対象プログラム	検知回避の経路	反転対象の条件の種類	改善結果			合計
			改善	変化なし	悪化	
Pafish	複数経路	単純な条件を含む	7	12	0	19
		複雑な条件のみで構成（評価対象外）	-	9	-	9
	単数経路（評価対象外）	—	-	34	-	34
	合計		7	55	0	62
Al-Khaser	複数経路	単純な条件を含む	11	7	0	18
		複雑な条件のみで構成（評価対象外）	-	7	-	7
	単数経路（評価対象外）	—	-	14	-	14
	合計		11	28	0	39

```
・unconstrained_ret_IsDebuggerPresent_1_32 == 0
・mem_31_16_64{UNINITIALIZED} == 0xfffffffffffffffe
```

図 3 単純な条件の例

Fig. 3 Examples of simple constraints

```
・(if 0x3f < mem_7ff_4_32{UNINITIALIZED} then 1 else 0)
・reg_ebp_3_32{UNINITIALIZED} + 0xfffffad8 == 0xfffffffe
```

図 4 複雑な条件の例

Fig. 4 Examples of complex constraints

とが原因として挙げられる．

5.2.2 提案 2 の評価結果

提案 2 を angr に実装した際の結果を**表 4** に示す．表 4 では，検知回避の経路が複数ある条件と一つしかない条件の 2 つに分類している．提案 2 の機能は検知回避の別経路の条件抽出であるため，複数経路存在する検知手法が対象となる．また，複数経路の中で，一度目の解析で抽出した条件，つまり反転対象の条件が単純な条件を含むか複雑な条件のみで構成されるかの 2 つに分類している．単純な条件とは**図 3** に示すような関数の返り値や特定のメモリアドレスに対する条件のみで構成されている条件を指す．一方，複雑な条件とは，**図 4** に示すような if 文や四則演算を含む条件を指す．提案 2 の手法は，複雑な条件のみで構成される条件に関しては対応しておらず，単純な条件を含む条件に対する改善である．表 4 の改善結果の分類は表 3 と同様である．

表 4 より，Pafish に関しては，検知回避の経路が複数ある検知手法が合計 28 個存在し，一度目の解析で抽出された条件の内，単純な条件を含む条件が 19 個，複雑な条件のみで構成された条件が 9 個となっている．今回対応している単純な条件を含む条件に関しての結果を見ると，19 個の検知手法の内 7 個について改善できたことが確認できる．

al-kahser に関しては，検知回避の経路が複数ある検知手法が合計 25 個存在し，一度目の解析で抽出した条件の内，単純な条件を含む条件が 16 個，複雑な条件のみで構成された条件が 9 個となっている．今回対応している単純な条件を含む条件に関しての結果を見ると，18 個の検知手法の内 11 個について改善できたことが確認できる．

まとめると，Pafish と Al-Khaser について今回改善対象である検知手法の総数 37 個の内 18 個に関して改善した．

Pafish と al-kahser において単純な条件を含むが，改善できなかった検知手法について，原因は以下の 5 つである．

- Windows API の引数に関する実装不足
  新たな条件，つまり分岐条件として利用されている条件が Windows API の引数に関する条件である場合，

© 2024 Information Processing Society of Japan

提案 1 を実装していない angr では，引数にシンボル値を設定する仕組みになっていないため，新たな条件を抽出することができない．単純な条件を含むが改善できていない検知手法の多くがこれが原因で改善できていない．

- 既存のシミュレーション関数の実装内容
  新たに到達した分岐の条件が，既存のシミュレーション関数が実装されている Windows API の返り値や引数であり，具体値を返すような実装になっていた場合，新たな条件を抽出することができない．
- 新たなエラー
  提案 2 による解析で新たな経路を解析できたが，そこで実装上の不備で新たなエラーが生じ，新たな条件を抽出することができない．
- Windosw API の返り値に関する条件の反転の限界
  Windows API の返り値が条件として抽出されて反転する場合，当該シミュレーション関数内で反転した条件を返り値のシンボル値に付与することになる．このため，1 つの検知で同じ Windows API が複数回呼ばれている場合，そのすべてに同じ条件を付与することになる．したがって，一回目と二回目の呼び出しで真反対の条件の付与が必要となる場合，矛盾が生じ，解析できない場合がある．
- ループ処理による負荷増加
  提案 2 により新たな経路に到達したが，その経路がループ処理を行う経路であった場合，ループ回数が多すぎると負荷が高くなり，解析できない場合がある．

5.3 Symba との比較

関連研究で挙げた Symba も目的の Windows API に関して引数にもシンボル値を設定することで詳細な条件を抽出している．Symba と本論文の提案 1 の違いについて述べる．Symba では，解析前に MSDN から情報を取得してシミュレーション関数の生成が必要である．一方，提案 1 の場合は，解析時にすべての処理が行われるため解析前の事前準備の必要性がない．また，Symba は MSDN を参照する都合上，ドキュメントが存在しない Windows API に関して対応していない．一方，提案 1 は angr で定義されている Windows API のプロトタイプ宣言を参照しているため，事前にプロトタイプが定義されている Windows API であれば引数や返り値について正確に扱うことができる．また，プロトタイプ宣言がない Windows API の場合に関しても正確性は低下するが，自動的に引数や返り値の設定を行い処理することができる．さらに，Symba では引数への対処はシンボル値を設定することのみであるが，提案 1 では，引数が関数であった場合に，その関数を呼び出して，呼び出し先も解析することでより詳細な条件を抽出することが可能である．このように，提案 1 は引数の型情報によって別の処理を行う実装であるため，今後の拡張性もある．最後に，引数へのシンボル値設定に関して，引数が構造体などへのポインタであった場合，Symba では手動で設定するシンボル値を工夫する必要があるが，提案 1 では，自動でポインタの指す先の構造体の情報も取得して，その要素の型や名前の情報を取得して，シンボル値を設定することが可能である．

6. 今後の課題

今後の課題について述べる．

- 文字列比較による検知への対応
  提案 1 では比較する文字列情報がシンボル値化されたため条件を抽出できない．この問題を解決するには，提案 1 の ReturnSymbolUnconstrained 関数を拡張して文字列の場合に別の処理を行うようにする必要がある．
- 解析負荷増加への対応
  解析負荷が増加して解析が失敗した場合が両提案で見られた．シンボル値増加による負荷増加に関しては，規定値以上のシンボル値が作成された場合には，シンボル値を評価して重要性の低いシンボル値を削除することで負荷を低減する必要がある．また，ループによる負荷増加に関しても，一定の使用メモリ量を超えたらループを強制的に脱出する仕組みが必要である．
- 提案 1 と提案 2 の組み合わせ
  今回の提案で改善しなかった検知手法に関して，提案 1 と 2 を組み合わせて解析を行うことで解決可能な原因も存在した．例えば，提案 1 では検知回避の経路に関して Windows API の返り値に関する条件が最短であった場合に改善できなかった．これは提案 2 の条件反転により解決できると考えられる．また，提案 2 では，Windows API の引数が新たな分岐条件として利用されている場合に改善できなかった．これは提案 1 の引数へのシンボル値設定により解決可能であると考えられる．したがって，提案 1 と 2 の組み合わせは必要であると考える．
- 解析時の開始，到達，回避アドレス設定
  シンボリック実行の根本的な課題として解析時の開始アドレスと到達アドレス，回避アドレスの決定方法が挙げられる．本論文では，それらアドレスの決定方法は事前に解析者が手動で設定しているため，実際のマルウェアの解析時には負担が大きい．そのため，自動的に解析箇所を決定する必要がある．自動的にアドレスを決めて解析を行う機能を含むシステムを提案している研究として，ベーシックブロック内の命令数に着目した研究 [12] がある．文献 [12] の提案システムでは，実行時に通らなかった経路かつ残りの経路で命令数が最も多いベーシックブロックにマルウェアの悪性

© 2024 Information Processing Society of Japan

部分が含まれると仮定して，その条件を満たす箇所を到達地点として登録している．

- パッキングや難読化への対応
  解析対象のプログラムに難読化が施されていた場合，解決すべきシンボル値が増え，リソースが消費されてシンボリック実行による解析は困難になると考えられる．また，パッキングが施されていた場合，プログラムを実行するわけではないためアンパッキングすることができず，本来の実行コードを解析することができないと考えられる．これらの課題は実際のマルウェアを解析するためには解決する必要がある．

7. まとめ

本論文では，シンボリック実行ツールである angr について 2 つの改善案を提案し，その評価を示した．従来の angr では，個別のシミュレーション関数が用意されていない Windows API に関して，返り値に関する条件しか抽出することができない．しかし，提案 1 により引数に関する処理を追加したことで，標準状態の angr 以上の条件を取得できることを示した．また，標準状態の angr の探索手法では，短い経路に関する条件から抽出していたため，検知の趣旨に沿った条件を抽出できないことが多々あった．しかし，提案 2 によって一度抽出した条件を反転させて，別経路を探索させることにより，従来より検知の趣旨に合った条件を抽出することが可能となった．

今後は，提案 1 で対策不足になっていた文字列比較による検知への対応や提案 1 と 2 を組み合わせる解析方法で，より詳細な条件を抽出することを可能とするとともに，より汎用性のある手法の実現を目指す．

参考文献

[1] Kim, M., Cho, H., and Jeong Hyun Yi：Large-Scale Analysis on Anti-Analysis Techniques in Real-World Malware — IEEE Journals & Magazine — IEEE Xplore, https://ieeexplore.ieee.org/abstract/document/9829724 (Accessed on 08/16/2024).

[2] Lindorfer, M., Kolbitsch, C., Comparetti, M., P.：Detecting Environment-Sensitive Malware., Recent Advances in Intrusion Detection (RAID) 2011. Lecture Notes in Computer Science, vol 6961. (2011).

[3] 窪 優司，大久保隆夫：シンボリック実行を活用したマルウェア解析環境 検知機能の回避条件自動抽出の研究，情報処理学会 Computer Security Symposium 2018 (2018).

[4] : angr, https://angr.io/ (Accessed on 08/16/2024).

[5] Shoshitaishvili, Yan, Wang, Ruoyu, Salls, Christopher, Stephens, Nick, Polino, Mario, Dutcher, Audrey, Grosen, Jessie, Feng, Siji, Hauser, Christophe, Kruegel, Christopher, Vigna, Giovanni：SoK: (State of) The Art of War: Offensive Techniques in Binary Analysis (2016).

[6] Stephens, Nick, Grosen, Jessie, Salls, Christopher, Dutcher, Audrey, Wang, Ruoyu, Corbetta, Jacopo, Shoshitaishvili, Yan, Kruegel, Christopher, Vigna, Giovanni：Driller: Augmenting Fuzzing Through Selective Symbolic Execution (2016).

[7] Shoshitaishvili, Yan, Wang, Ruoyu, Hauser, Christophe, Kruegel, Christopher, Vigna, Giovanni：Firmalice - Automatic Detection of Authentication Bypass Vulnerabilities in Binary Firmware (2015).

[8] : GitHub - a0rtega/pafish: Pafish is a testing tool that uses different techniques to detect virtual machines and malware analysis environments in the same way that malware families do, https://github.com/a0rtega/pafish.

[9] Tirenna, P. F., Basile, C., Lioy, A.：Techniques for malware analysis based on symbolic execution, https://webthesis.biblio.polito.it/15305/ (Accessed on 08/16/2024).

[10] : Microsoft Learn: キャリアの扉を開くスキルを身につける，https://learn.microsoft.com/ja-jp/ (Accessed on 08/16/2024).

[11] : LordNoteworthy/al-khaser: Public malware techniques used in the wild: Virtual Machine, Emulation, Debuggers, Sandbox detection., https://github.com/LordNoteworthy/al-khaser (Accessed on 08/16/2024).

[12] 本多恵佑，呉 謙， 金井敦：動的解析と静的解析を連携したマルウェアのサンドボックス回避条件抽出，*2024 Symposium on Cryptography and Information Security Nagasaki, Japan, Jan. 23– 26, 2024* (2024).

© 2024 Information Processing Society of Japan