未知のバイナリプログラムが用いているデータ構造の推定

川端祥龍† 大山恵弘†

1. 背景

マルウェアの脅威は依然深刻である．マルウェア対策においては，マルウェアの動作の解析が，被害の分析や拡散の防止に不可欠である．しかし，マルウェアの解析は困難であることが多い．その要因として，マルウェアのソースコードが入手困難であることや，マルウェアのコードはしばしば難読化されているということが挙げられる．

マルウェアの解析には，マルウェアを動作させずに解析する静的解析と，マルウェアを実行して動作を解析する動的解析がある．単純なマルウェアに対しては，静的解析はある程度有効である．しかし，複雑なマルウェアや難読化されたマルウェアに対しては，解析が失敗したり，解析に大きな手間がかかるなど，静的解析が有効でないことも多い．そのようなマルウェアの解析には動的解析が有効である．

従来の動的解析手法の多くは，呼び出されるOS APIの列や，難読化前のコードを抽出する[3]．しかし，残念ながら，それらの手法で得られる情報は，OSとアプリケーションの間の界面の挙動の情報や，コード列の情報に限られる．このような情報のみから，マルウェアの解析を行うことは容易ではない．効率よくマルウェアの解析を行うためには，マルウェアが内部で用いているアルゴリズムやデータ構造に関する情報が必要である．

2. 目的と方針

本研究では，マルウェアに代表される未知のバイナリプログラムが内部で用いているデータ構造の推定を行うシステムを提案する．

本システムは，未知のプログラムを監視しながら動作させ，様々なタイミングでそのプログラムを停止させてスタックやヒープの内容を取得する．その内容の解析により，そのプログラムが用いているデータ構造の種類や，データに関して常に成立する性質（インバリアント）を推定する．たとえば，推定結果として「アドレス0x08048374から始まる関数の3個目の局所変数は，ヒープ上の文字列を指すポインタである」とか「アドレス0x08052ca8に格納されている値は必ず0から31までの範囲の数である」といった情報を出力する．また，用いているデータ構造がリストやツリーであるなどの情報も推定する．

† 電気通信大学大学院 電気通信学研究科 情報工学専攻

3. 関連研究

Cozzie らによる Laika[1] は，プロセスのメモリイメージを入力として，そのプロセスが用いているデータ構造を推定するシステムである．本研究の方法はLaikaの方法をベースにしている．ただし本研究では，以下の2点により，Laikaよりも高い精度で推定することを目指している．第一に，複数のメモリイメージを利用する．提案システムでは，未知のプログラムを実行しているプロセスを複数のタイミングで停止させ，多数のメモリイメージを取得して解析に用いる．第二に，システムコールやライブラリ関数の引数の仕様に関する知識を利用する．例えばある値がstatシステムコールの第二引数に渡されていたら，その値のアドレスから始まるメモリ領域は構造体を格納するために用いられていると推測する．

Ernst らによる Daikon[2] は，プログラムのソースコードを入力として，そのプログラムに対して成り立つインバリアントを推定するシステムである．本研究とDaikonの相違点は、主に以下の2つである．1つ目は，Daikonはインバリアントの推定にソースコードを必要とするが，本研究のシステムではソースコードを必要としない．2つ目は，Daikonではリスト構造やツリー構造等のデータ構造の推定は行わない．これに対し，本研究ではデータ構造と，データ構造について成り立つインバリアントも推定する．

4. 提案システム

提案システムは，バイナリプログラムを入力として，それに関する解析結果を出力する．本システムはLinux/IA-32上に実装され，Linux/IA- 32用バイナ

リプログラムを解析対象とする．解析対象バイナリにはシンボル情報は含まれていないとする．プログラムの解析は，メモリイメージの収集と，メモリイメージの解析の，2 つのフェイズからなる．

4.1 メモリイメージの収集

未知のプログラムを実行するプロセス（アプリケーションプロセス）1 つにつき，それを監視するためのプロセス（モニタプロセス）を 1 つ作る．モニタプロセスはアプリケーションプロセスを，全ての関数の先頭と末尾，および，全てのシステムコールの呼び出しと復帰のタイミングで停止させる．停止後，アプリケーショプロセスのメモリイメージ，全レジスタの値，ファイル`/proc/pid/maps`の内容を取得し，保存する．このファイルには，プロセス番号 pid のプロセスが使うメモリのどの範囲がスタックやヒープなどであるかについてのメモリマップ情報が書かれている．関数の先頭と末尾での停止は，コード領域へのブレークポイントの挿入により実現し，システムコールでの停止は`ptrace`システムコールにより実現する．

4.2 メモリイメージの解析

本システムは，収集されたメモリイメージなどから，そのプログラムがどうメモリを利用するかを推定する．具体的には，ヒープ領域と静的データ領域の各アドレス，および，各関数のスタックフレーム内の各場所に格納されるデータの「種類」を推定する．「種類」とは，整数やポインタなどの，プログラム内での用途に応じて分類された値集合の各々に付与される情報である．例えば，ある値がヒープ領域上のリストを指すポインタとして解釈できる場合には，本システムはその値の「種類」を，「ヒープ領域にあるリストへのポインタ」であると推定する．なお，当然ながら，大きい整数をポインタであると解釈してしまい，推定を誤る可能性はある．

メモリイメージの解析は三段階で進める．第一段階では，最初の一つのメモリイメージを選び，そのコード領域のデータを逆アセンブルして`call`命令を抽出する．そして`call`命令のオペランドに含まれるコードアドレスの集合を元に，各関数のコードが置かれたメモリ範囲を推定する．

第二段階では，各メモリイメージに対して以下の処理を行う．まず，スタックを底に向けて辿りながら，各フレームに格納されている値を取得する．そして，各フレームがどの関数のためのものかについて，第一段階で求めたメモリ範囲情報と，各フレームに格納されているリターンアドレスを照合して調べる．

第三段階では，各メモリイメージの全てのメモリ領域を走査し，ポインタと推定される値を集める．そして，それらのポインタの値のアドレス全てから別個のデータ構造が始まると推定する．そして，各データ構造は，次のデータ構造が始まるアドレスの直前や，メモリマップ情報から得たメモリ境界まで続くと推定する．

また，推定されたポインタの先が，NUL 文字で終わる ASCII 文字列である場合には，「文字列を指す」という情報を，そのポインタの「種類」に付加する．

さらに，システムコールの呼び出し前後のメモリイメージの解析では，システムコールの仕様から分かる情報を利用する（「16 バイトの構造体を指す」など）．

これらの処理の後，リストやツリーといった，ポインタの組み合わせによって構築されるデータ構造を検出する．検出処理は，データ構造が中にポインタを含む場合に，そのポインタを再帰的に辿ることによって行う．

5. 現状と今後

現状は，基本設計に基づきシステムの実装を行った．また，簡単なプログラムに対して，データ構造の推定が行えることを確認した．

今後は，データ構造やインバリアント推定における精度や解析に必要な時間について，実在のマルウェアも含む様々なプログラムを対象として，評価を行っていく予定である．また，実験結果を元にデータ解析アルゴリズムの改良やその評価を行う予定である．

謝辞 本研究の一部は総務省戦略的情報通信研究開発推進制度（SCOPE）の支援を受けて行われた．

参 考 文 献

1) Anthony Cozzie, Frank Stratton, Hui Xue, Samuel T. King. Digging For Data Structures. In *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation*, 2008.
2) Michael D. Ernst, Jeff H. Perkins, Philip J. Guo, Stephen McCamant, Carlos Pacheco, Matthew S. Tschantz, Chen Xiao. The Daikon system for dynamic detection of likely invariants. *Science of Computer Programming*, 69(1-3):35-45, 2007.
3) Min Gyung Kang, Pongsin Poosankam, Heng Yin. Renovo: A Hidden Code Extractor for Packed Executables. In *Proceedings of the 2007 ACM Workshop on Recurring Malcode*, 2007.