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

大山恵弘[†]　川端祥龍[†]

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を推定する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を推定する。

Prediction of Data Structures Used in Unknown Binaries

Yoshihiro Oyama [†] and Yoshitatsu Kawabata[†]

1. はじめに

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。マルウェアの解析には、コード列を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。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。その結果、マルウェアが内部でどういうアルゴリズムやデータ構造を用いているかについては多くの情報が得られず、攻撃手法の理解に手間がかかる。

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を推定する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 に格納される値は必ず 0 から 31 までの範囲にある」といった情報を出力させることを現在検討している。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を推定することも検討している。

† 電気通信大学
The University of Electro-Communications

2. 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。

3. 提案システム

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。

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。

本システムは、保存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、ある値がスタック領域やヒープ領域を指すポインタとして解釈できる（その値がそれらの領域のアドレス範囲内にある）場合には、本システムはその値の「種類」を、「スタック領域へのポインタ」や「ヒープ領域へのポインタ」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。

解析は三段階で進める。第一段階では、平文のコード列がメモリ領域上に載っているメモリイメージを一つ選び、そのコード列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。なお、各フレームがどの関数のためのものかについては、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。次に、静的データ領域やヒープ領域に保存されている値をすべて取得する。取得した各々の値について、その値がヒープなどを指しているかどうかに従い、その値に割り当てる「種類」を決定する。

第三段階では、第二段階でポインタと推定された各値が指すデータ構造を推定する。まず、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。そうでない場合には、ポインタが指すデータ構造がどのメモリアドレスまで続くのかを以下の方法で推定する。まず、その時点でのメモリイメージの中から、ポインタと推定された値を全て集める。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、スタックのフレーム間境界まで続くと推定する。

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、システムコールの引数や返り値として渡されている値を調べる。そのシステムコールがポインタを受け渡すものである場合には、受け渡されているポインタの値の「種類」に、システムコールの仕様から分かる情報を加える（「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、ポインタと推定された値がデータ構造の中に含まれる場合に、そのポインタを再帰的に辿ることにより実現される。

4. 現状と今後の課題

我々は提案システムを現在実装中である。実装が完了したら、様々なアプリケーションを対象に、データ構造を正しく推定できるかどうかを評価する。また、データの値に関して常に成立する性質を推定する機能の追加も進める。この機能は Daikon[2] の手法を参考にして設計する予定である。さらに、Windows への移植も検討していきたい。

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, and 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, and Chen Xiao. The Daikon system for dynamic detection of likely invariants. *Science of Computer Programming*, 69(1–3):35–45, 2007.