実行可能プログラムのコード変換による脆弱性の検知手法の提案

境　顕宏 †‡　　堀　良彰 †‡　　櫻井　幸一 †‡

† 九州大学大学院システム情報科学府
819-0395 福岡市西区元岡 744

‡（財）九州先端科学技術研究所
814-0001 福岡市早良区百道浜 2-1-22 SRP ビル 7F

あらまし　既存のバイナリコードの振る舞い解析には特殊な技術を必要とする．さらに，近年では自身のコードを暗号化するなどし，バイナリコードから悪意のある振る舞いを静的に検出することは難しくなってきている．今回は，バイナリコード中の振る舞いを実行時に変更することで，プログラム中の脆弱性回避や悪意のあるコードの実行を防ぐ方法について提案する．また，振る舞い変更を再利用性の高いプログラミング手法として一般化する．

Vulnerability Detection in Native Code based on Code Translation

Akihiro Sakai†‡　　Yoshiaki Hori†‡　　Kouichi Sakurai†‡

†Graduate School of Information Science and Electrical Engineering, Kyushu University
744 Motooka, Nishi-ku, Fukuoka, 819-0395, Japan

‡Institute of Systems, Information Technologies and Nanotechnologies
2-1-22 Momochihama, Sawara-ku, Fukuoka, 814-0001, Japan

Abstract　When analyzing the behaviors of an existing binary code, it is difficult because need to learn about special technologies. Moreover, static analysis is also difficult to detect the malicious behaviors if codes are encrypted by themselves. In this paper, we propose the method of dynamic change of program behavior, and avoid executing vulnerability codes and malicious codes intentionally. In order to make it possible, we plan to generalize it as reusable programming approach.

1　はじめに

近年のソフトウェアの汎用化に伴い，同じソフトウェアが様々な場所で利用されるようになってきた．さらに，ソフトウェアの機能自体も大規模かつ複雑化しており，開発現場では既存のコードの流用などによるソフトウェアの再利用化が進み，ネットワークを介したサービス等も登場した．このような中，ソフトウェアは様々な入力に対し正しく動作することが求められている．

しかし，意図しないソフトウェアの振る舞いが問題となることがある．例えば，大規模化したソフトウェアから脆弱性の原因となる不具合を完全に取り除いてしまうことは難しく，ソフトウェア設計に関わる問題や未知の脆弱性などを悪用されたゼロデイ攻撃が問題となっており，様々な取り組みが成されている [1][9][10]．一方で，ソフトウェア自体が攻撃コードで構成されているマルウェア等も存在する．さらに，近年のマルウェアは，自身のコードを暗号化し，シグネチャベースでの検知ソフトを逃れるケースも存在する．このようなプログラムは単純に逆

アセンブルするだけで攻撃コードが含まれるかを静的に調べることは難しく，隔離環境で実行させて動的に振る舞いを調べる試みもなされている [11].

しかし，実行時に意図しないコードが含まれていないかを調べることは容易ではない．これはネイティブコードの動的な解析には OS に関する専門的な知識や独自の実験環境が必要だからである．例えば，UNIX 系 OS においてシステムコールをフックするためには，カーネルとして動作するドライバを記述する必要がある．したがって，システム管理者や開発者が動的解析プログラムを実装することは容易ではなかった．

この問題に対し本稿では，プログラム中で外部から観測可能な実行位置に着目し，振る舞いを変化させるための枠組みを提案する．この枠組みでは，実行位置の意味集合からなるセマンティックスペース部分と，振る舞いを変化させるコードからなり，実行時にコード変換が行われる．また，実行位置の抽出および振る舞いを変化させるコードを定義するにあたって，モジュール化機構の 1 つであるアスペクト指向プログラミング (AOP: Aspect-Oriented Programming)[4][7][8] の概念を用い，記述方法も似たような表記とした．

本提案手法を利用することにより振る舞い変化を行うためのコード記述が可能になる．また，AOP の概念を用いた記述を行うことで記述したコードが再利用性の高いものとなり，他のプログラムに対してもコード変換が流用可能になる．

本稿の残りの構成は次のようになっている．2 節で問題提起を行い，3 節で提案する枠組みについて説明する．4 節で記述例を説明し，5 節では提案手法を実現するための実装について述べる．6 節で提案手法を議論し，7 節で関連研究について述べ，8 節で本稿を締める．

2 問題意識

本節では，プログラムの動的振る舞い変化を代表する方法として仲介ライブラリを用いた変更について説明し，その問題点ついて述べ研究の目的を明らかにする．

2.1 仲介ライブラリを用いた変更

プログラムの実行時の動作を変更する方法として仲介ライブラリを用いる方法がある．Windows オペレーティングシステムでは，ファイルを開く API として `CreateFile` という関数等が用意されている．この関数は Kernel32.dll に定義されており，関数を呼ぶとシステムフォルダ内の Kernel32.dll にある関数が実行される．ところが，動的ライブラリのロードにはディレクトリの探索順番があり，実行ファイルと同じディレクトリに同名のライブラリがある場合，そちらが優先してロードされる．仲介ライブラリはこの性質を利用した振る舞い変更の方法である．以下のプログラムは，アクセスしようとするファイル名を調べ特定の領域へのファイルアクセスを別の場所にリダイレクトする仲介ライブラリの例である．

```
HANDLE CreateFile(
        LPCTSTR lpFileName,
        DWORD dwDesiredAccess,
        DWORD dwShareMode,
        LPSECURITY_ATTRIBUTES lpSecAttr,
        DWORD dwCreationDisposition,
        DWORD dwFlagsAndAttributes,
        HANDLE hTemplateFile
  ){
    lpFunc = (CreateFile)GetProcAddress
       (hK32DLL,"CreateFile");
    if (fileName のチェック) {
      return lpFunc(renamedFileName);
    } else {
      return lpFunc(fileName);
    }
  }
…(同ライブラリ内の他関数のスタブ)…
```

仲介ライブラリを用いる方法では，この例のようにファイルアクセス用の関数と同じ名前と引数を持った `CreateFile` 関数を用意しコンパイルする．作成されたライブラリを振る舞いを変更したい実行ファイルと同じディレクトリにおいて実行するとプログラムが API を呼び出す位置で振る舞いが変更される．

しかし，仲介ライブラリを用いる方法には次のような問題点がある．

再利用性の問題 仲介ライブラリは本来の API

呼び出しでロードされる動的ライブラリの動作をフックしたものとなる．したがって，仲介ライブラリでは本来のライブラリ内に定義されている全ての関数を含めなければならない．一般に Kernel32.dll 等に代表されるライブラリは沢山の関数をもっており，その中にはファイルアクセスとは関係のないものも存在する．よって，ライブラリがエクスポートする他の関数スタブが別の関心毎としてプログラム内に書かれるため，再利用性が低下する．

類似関数の問題 ファイルを開く関数として，`CreateFile` を上げたが，利便性のため引数を減らした関数や，OS の下位互換性を保つための関数等，同じ意味をもつ API 関数が複数存在する．上のプログラムをコンパイルしてそのまま動かした場合，`OpenFile` 関数等では元々の処理が実行されてしまい，ファイルを開くという同じ意味を持つ動作でも時によって振る舞いが変化しないことがあり，不具合混入の元となる．

2.2 本研究の目的

前節で述べたように，仲介ライブラリを用いる方法では再利用性の問題や類似関数の問題で振る舞い変化をうまく表現できない問題があった．したがって，プログラムの意味に応じて振る舞いを変化させることを余分な記述なしで表現することはできなかった．

この問題に対し本稿では，セマンティックスペースを用いた AOP 言語を用い，バイナリコードの実行時に処理を埋め込むことで振る舞いを変える方法を提案する．この方法を用いることで，プログラム動作の意味に対応した振る舞いの変更を行う記述が可能になり，前に述べた問題が解決できる．

3 動的コード変換手法: DynaAO

本節では，プログラム実行点を集合として捉え，動的な処理変更が行える AOP 言語，DynaAO について説明する．

3.1 セマンティックスペース

はじめに AOP の概念について述べる．AOP とはログ処理等の横断的関心事を分離するためのモジュール化機構で，横断的関心事はアスペクトと呼ばれるモジュールに記述する．中でも，AspectJ[2] はジョインポイント機構 (join point mechanism) を採用しており，プログラムの実行位置を表すジョインポイント (join point)，ジョインポイントの中から特定のジョインポイントを選ぶためのポイントカット (pointcut)，ポイントカットにより選び出された箇所に影響を与えるアドバイス (advice) の 3 つの要素からなる．アスペクトは，織り込み (weaving) を行うことによって，既存のプログラムと合成される．DynaAO は，このジョインポイントモデルを用いた言語設計を行っている．

セマンティックスペースは，プログラムの動作を特徴づける関心毎を記述したモジュールである．また，セマンティックスペースは名前を持っており，ファイル操作等といった意味集合から名前を付ける．次にセマンティックスペース内では，ファイルを開く，ファイルを閉じる等の振る舞いの意味を記述する．

このセマンティックスペースは，振る舞いを変更するコード，つまりアドバイスを記述する際に参照される．

3.2 ジョインポイントとポイントカット

DynaAO におけるジョインポイントは，ネイティブコード中の特定のアドレスを指す．ジョインポイントとして選ぶアドレスはポイントカットとして記述する．

DynaAO におけるポイントカットは，セマンティックスペース内に記述し，各意味が示す外部から観測可能なプログラムの実行点を指定する．例えば，ファイルを開く場合，`CreateFile` や `OpenFile` 関数の呼び出しとして表現する．この表現は AspectJ のポイントカット定義に似ているが，DynaAO ではライブラリ名を明示する必要がある．

3.3 アドバイス

アドバイスは，前節で指定したポイントカットで選択したジョインポイントに織り込むコードの中身を指す．また，アドバイスはジョインポイントの前にコードを挿入する before アドバイス，ジョインポイントの後にコードを挿入する after アドバイス，ジョインポイントそのものを別のコードで置き換える around アドバイスに分けられる．

4 記述例

この節では，DynaAO を用いた 2.1 節で上げた例題に対する記述例を示す．以下にコード例を示す．

```
semanticspace FileIO{
    int open(LPCTSTR lpFileName);
    int close(HANDLE fileHandle);

    /* (1) */
    int open(LPCTSTR lpFileName){
      call HANDLE Kernel32.CreateFile(
        LPCTSTR lpFileName,
        DWORD dwDesiredAccess,
        DWORD dwShareMode,
        LPSECURITY_ATTRIBUTES lpSecAttr,
        DWORD dwCreationDisposition,
        DWORD dwFlagsAndAttributes,
        HANDLE hTemplateFile
      );
      call  HANDLE Kernel32.OpenFile(…);
      …
    }
}

using FileIO;
/* (2) */
around FileIO.open(LPCTSTR fileName){
  if (fileName のチェック) {
    /* (3) */
    return proceed(renamedFileName);
  }
  else{
    /* (4) */
    return proceed(fileName);
  }
}
```

このプログラムにおいて (1) はポイントカットの記述である．これは，プログラム中の kernel32.dll に含まれる CreateFile 関数呼び出し場所をジョインポイントとして選択することを意味する．さらに，このポイントカットではアクセスするファイル名を lpFileName 変数として取得している．

次に (2) は，(1) で選択されたジョインポイントにおいて実行するアドバイスの記述である．around アドバイスとして記述されているので，アドバイスは CreateFile 関数を置き換える処理として実行される．このアドバイスは，書き込もうとするファイル名を調べ，Windows ディレクトリへの書き込みであれば (3) が実行，そうでなければ (4) が実行される．(3)(4) では共に proceed が呼ばれ，これにより本来実行すべき kernel32.dll 内の CreateFile 関数呼び出しが行われ，その結果を返す．

5 実装

本節では実装の流れについて説明する．

図 1 は，実装と実行の流れを表している．まず，DynaAO 言語を DynaAO コンパイラによって C++ソースコードを生成する．この C++ソースコードはリンク先の DLL がエクスポートする関数をすべて含む完全なラッパーに相当する．次に生成された C++ソースコードを OS 上でロード可能な DLL へと変換する．この変換には既存の C++コンパイラを利用する．最終的には，生成された DLL と解析対象の実行ファイルを同じディレクトリ内に配置し実行することで，API 呼び出しが生成された DLL がエクスポートした関数で置き換わり，DynaAO 言語で記述した内容に沿ってプログラムの振る舞いが変更される．

また，本稿では Windows OS を対象とした記述例と実装方法を述べたが，UNIX 系の OS においても LD_LIBRARY_PATH 環境変数等を用いることにより同様な実装を与えることができる．

6 議論

本節では提案手法を議論する．

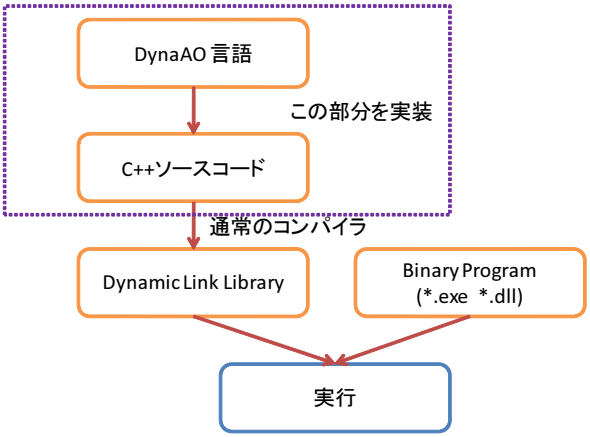

図 1: 実装と実行のイメージ

6.1 サンド ボックスアプローチとの比較

JavaVM では，ブラウザ経由でのプログラム起動の際，ファイルへの IO 等を制限するサンドボックスのメカニズムがある．本研究で提案しているコード 変換も，特定条件下以外では何の処理もせずに制御を返す関数として記述することができる．しかし，サンドボックスはアクセス制御の実現例であり，パーミッションのチェックを行い本来の処理の可否を決定する静的な関数が呼ばれるに過ぎない．したがって，例えばアクセスログを記録して処理は許可する等といったことを表現できない．一方，本提案手法では振る舞い変更を一般のプログラミング手法として与えることにより柔軟な記述が可能になる．

6.2 他のコード 変換手法との比較

実行時に振る舞いを変化させる方法とは別に，実行前にバイナリコードを変換させる静的コード 変換 [3] がある．実行前にコード 変換を行う場合，仲介 DLL 等を用いないため，プログラム側からデバッガ検知等に代表される手法によりコード 変換されていることが検出されることは避けられるが，デジタル署名等に代表されるコード そのものの変化検出手法で検知されてしまうといったデメリットがある．一方で VM を用いて振る舞いを観測するという手法もあるが [9]，現状では VM 検知技術とのいたちごっこになっていることや，あくまで API の呼び出し等を観測する技術だけで振る舞いの変更についてはこれまで提案されていない．本研究で提案した動的なコード 変換が VM 上からでも行えるかについては，今後検討していく予定である．

7 関連研究

ソフトウェアの動作に制限を加えるアクセス制御に関してはこれまで様々な研究がなされている．プロセスレベルによるサンドボックスの例として，Java の `SecurityManager` クラスがある [6]．これはアプリケーション内でセキュリティーポリシーを実装することができる．しかし，Java のサンドボックスは Java のバイトコードのみに適用でき，それ以外のバイナリコードの実行に対しては，保護することができない．

一方，ネイティブコードに対するプロセスレベルサンドボックスを記述する言語として，Generic Software Wrapper[5] がある．これは，システムコールの呼び出しをフックするプログラムを高水準の言語で記述する方法を提案している．しかし，提案されている言語はあらかじめ決められた関数群を用いて記述する独自の仕様になっており，記述したコードをほかのプログラムで再利用することは難しい．

これらに対し，本研究ではネイティブコードに対する振る舞い解析用のコードを書く新しい手法を提案した．この手法を用いることで，プログラムの様々な振る舞いからジョインポイントを選択し，前後への処理の挿入や処理の置き換えを容易に記述することができる．さらに，AOP におけるモジュール化性能を引き継いでいるため，振る舞い解析用のコードを他のプログラムに容易に移植することができる．

8 おわりに

本稿では，OS 上でネイティブに動作するソフトウェアに対する振る舞い解析の必要性を述べ，その上で，ネイティブコードを実行しながらその挙動を外部から観測する動的振る舞い解析の既存手法を説明した．さらに，AOP の考え方を利用したプログラミング手法を与えるこ

とにより，ネイティブコードに対する振る舞い解析コードを容易にかつ高移植性を保ちながら記述することが可能になった．

今後は Windows OS を対象とした実装を与え，提案手法が実現可能であることを示す予定である．その後，実際にプログラムの振る舞いが解析できることを記述実験する予定である．また，今回は `call` ポイントカットのみを考え，DLL の関数呼び出しをジョインポイントとして抽出したが，今後は変数へのアクセスやコントロールフローを利用した解析等，DLL の関数呼び出し以外のタイミングでのポイントカットの与え方や解析手法についても検討する予定である．さらに節 6.2 で述べたように VM 上からのアプローチについて検討を行う．

謝辞

本研究を行うにあたって，第一筆者は文部科学省による九州大学グローバル COE プログラム「マス・フォア・インダストリ教育研究拠点」の支援を受けた．

参考文献

[1] 2009 CWE/SANS Top 25 Most Dangerous Programming Errors, http://cwe.mitre.org/top25/

[2] AspectJ, http://www.eclipse.org/aspectj/

[3] Breech, B., Pollock L., “A Framework for Testing Security Mechanism for Program-Based Attacks”,*Proc 2005 workshop on Software Engineering for Secure Systems (SESS’05)*, 2005.

[4] Elrad, T., Filman, R.E., and Bader A., “Aspect-oriented programming,” Communications of the ACM, vol.44, no.10, pp.29-32, 2001.

[5] Fraser, T., Badger, L. and Feldman, M., “Hardening COTS Software with Generic Software Wrappers”, *Proc. 1999 IEEE Symposium on Security and Privacy*, pp. 2-16, 1999.

[6] Gong, L., Mueller, M., Prafullchandra, H., Schemers, R., “Going Beyond the Sandbox: An Overview of the New Security Architecture in the Java Development Kit 1.2”, *Proc. USENIX Symposium on Internet Technologies and Systems*, pp.103-112, 1997.

[7] Kiczales, G., Lamping, J., Mendhekar A., Maeda, C., Lopes, C., Loingtier, J., and Irwin, J., “Aspect-Oriented Programming,” *Proc. 11th European Conference on Object-Oriented Programming (ECOOP ’97)*, pp.220-242, 1997.

[8] Kiczales, G., and Mezini, M., “Separation of Concerns with Procedures, Annotations, Advice and Pointcuts,” *Proc. 19th European Conference on Object-Oriented Programming (ECOOP 2005)*, pp.195-213, 2005.

[9] 安藤 類央, Quynh, N., 須崎 有康, “Windows OS のメモリ挙動モニタと Libvirt によるゼロデイ攻撃の検出システムの構築”, 2009 年暗号と情報セキュリティシンポジウム, 2009.

[10] 森崎 修司, 吉田 則裕, 肥後 芳樹, 楠本 真二, 井上 克郎, 佐々木 健介, 村上 浩二, 松井 恭, “コードクローン検索による類似不具合検出の実証的評価”, 電子情報通信学会論文誌 Vol.J91-D No.10 pp.2466-2477, 2008.

[11] 吉岡 克成, 松本 勉, “自動マルチパス解析によるマルウェア動的解析の提案”, 2009 年暗号と情報セキュリティシンポジウム, 2009.