クラスファイルの実行直前改変によるソフトウェア保護方法の検討

2005MT079 西隠居雄毅

指導教員 真野芳久

1. はじめに

今日、プログラムの盗用による技術の流出が大きな問題となっている。この問題を解決するための技術の総称をソフトウェアプロテクションと呼ぶ。プログラムの可読性を意図的に下げることで、プログラムを保護する技術である難読化はその技術の1つである。暗号化と呼ばれる技術もまた、その一部が発展しソフトウェアプロテクションの 1 つとして広く利用されている。

これらの技術の特性を生かした新しいソフトウェアプロテクションの技術が、神崎らにより提案されている[3]。この手法ではプログラム中の任意の命令を異なる命令で偽装し、プログラムの自己書き換え機構を用いて、実行時のある期間においてのみ本来の命令へと復元を行う。そうすることで攻撃者が偽装された命令を含む解析を試みたとしても、命令の書き換えを行うルーチンの存在に気づかない限りプログラムの本来の動作を正確に理解することが不可能となる。

神崎らの研究ではカムフラージュ前の状態において、行数が1490、命令数が947のプログラムにて、500の命令がカムフラージュされたとき、実行時間がオリジナルの 48 倍となるとされており、実行時間に大きな問題を抱えている

本研究では、プログラムの書き換え回数を 1 回に留め、実行時間を実用に耐えうる形で実現する手法を提案する。

2. ロード処理と変換の組合せ方法

アイディアとしては、偽装を施したプログラム（以下偽装プログラム）の実行直前にロード処理をインターセプトし、本来行うべき処理へと書き換えた後にプログラムをロードし、実行させるというものである。このように処理の書き換え回数を1回にとどめることで、実行時間の短縮、ファイルサイズの軽減を見込む。

この手法ではプログラムのロード処理のインターセプトを行うため、偽装プログラムとは別にそのプログラムのロードを行なう起動用プログラムが別途必要となることが考えられる。そこで次にプログラムのインターセプトを行うとともに復元能力を持った起動用プログラム(アプリケーションランナー、以下AR)を用いた2種類の手法について検討する。

1 つ目の手法は、起動用プログラムを偽装プログラムとは別に用意する手法である。この手法では1つの起動用プログラムが多数の擬装用プログラムと対応付けられるため、全体的なファイルサイズの軽減が見込める。

またもう一方の手法は、起動用プログラムが偽装プログラムを含み、見かけ上 1 つのプログラムで復元から実行までを行う。そのため、この手法ではソフトウェアプロテクションを施した事実を隠蔽することができる。

3. ロード処理と変換の組合せ方法実現

本研究では JVM(Java 仮想マシン)にクラスロードする処理をインターセプトし、プログラムが実際にロードされる前にクラス表現を変換することを利用しプログラムの保護を試みた。本来、クラスロードのインターセプトを行うためには、アプリケーションのクラス用のローダを独自に定義し使用する必要がある。ここでは Javassist[2]によってあらかじめ定義されているローダの機能を利用した。

3.1. クラスロード処理のインターセプト

Javassist でのクラスロード処理のインターセプトは、javassist.ClassPool クラスに基づいて行われる。このクラスはクラスパスを管理し、クラスファイルをディスク等から実際に読み込む作業を担当する。クラスをロード時に操作するために、ClassPool は Observer パターンを使用する。このObserberを利用することで、新しいクラスがClassPoolから要求されるたびに、Observer 内のメソッドが呼出され、ClassPoolによって実行される前に、クラス表現を修正することが可能となる [1]。

3.2. AR を別に用意する手法

処理の流れとしてはプログラムを起動し復元を行う AR を実行し、偽装クラスファイル（以下 A’）をロードする。このとき AR が用意したクラスローダにより A’はロードされるため、クラスファイルが読み込まれるごとにロード処理がインターセプトされる。A’内の情報を元に復元が行われる。A’が本来の処理を行うクラスファイル（以下 A）へと復元されると、インターセプトされていたクラスローダ処理が通常の処理に戻り、A が JVM に読み込まれ実行される。

このとき、A はメモリ内から読み込まれ、ファイルとして保存されることはないので、実行終了後のファイルは AR と復元前の A’のみとなる。処理の流れを図 1 に記す。

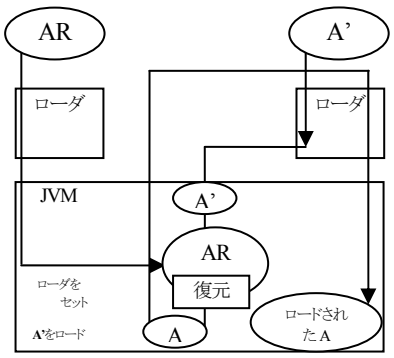

図1 AR を別に用意する手法の流れ

3.3. AR が偽装プログラムを含む手法の流れ

AR が A’を含む手法の流れは次のようになる。まず A'を含む AR をロードし、JVM に読み込まれた AR の main メソッドが実行される。JVM 上で AR は A’の main メソッドにあたる部分を検索し、クラスファイルの復元を実行する。復元された内容はメモリ内から直接クラスローダに渡され、再度別のクラスファイルとして JVM へと読み込まれ、復元された A が実行される。実行終了後に残るファイルは、復元前の A’を含む AR のみとなる。この流れを図 2 に記す。

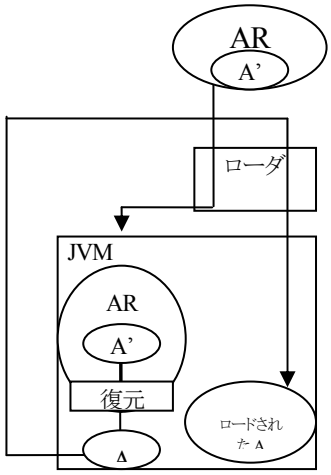

図 2 AR が偽装プログラムを含む手法の流れ

4. 提案手法の実現

実現にあたり、クラスローダの定義と復元を行うプログラム、および実験対象となる偽装プログラムを作成した。提案手法の中で重要となるインターセプトを行うローダの定義について述べる。図 3 のように記述することで、クラスファイルが呼ばれるごとに、クラスファイルの書き換えを行えるよう、インターセプトをかける Translator 機能をクラスローダに付与する。こうすることで通常のクラスローダにインターセプト機能を追加することができる。

```
public static void main(String[] args) {
    if (args.length >= 1) {
        try {
            // クラスローダに translator 機能を付与
            Translator xlat = new VerboseTranslator();
            ClassPool pool = ClassPool.getDefault(xlat);
            Loader loader = new Loader(pool);
            …
            // 指定したクラスのメインメソッド呼び出し
            loader.run(args[0], pargs);
…}}}
```

図 3 ローダの定義を行うプログラム

5. 実験とその結果

今回実験として、クラスファイルのインターセプト処理を行うプログラムを 10000 回繰り返し、オリジナルの実行時間を差し引き、提案手法によって生ずる 1 回当りの平均オーバーヘッド時間を求めた。

実験の結果として、AR を偽装プログラムと別々に用意する場合が約 11.36 ミリ秒となり、AR が偽装プログラムを含む場合が約 16.28 ミリ秒という結果が得られた。この結果から本研究の手法によってオーバーヘッドを十分小さい値に抑えられることがわかった。またそれぞれの手法の利用方法として、実行時間を優先する必要がある場合は AR を偽装クラスファイルと別々に用意する手法を利用し、攻撃へのより高い耐性を求める場合、AR が偽装クラスファイルを含む手法を利用するという使い分けが考えられる。

6. おわりに

今回クラスファイルの実行直前改変という新たな選択肢を示した。しかし実用までにクリアすべき課題は少なくない。今後の課題としては偽装プログラムを生成するシステムの構築、偽装復元ルールのデータベース化などがある。

参考文献

[1] D. Sosnoski: “Java プログラミングのダイナミックス”, http://www.ibm.com/developerworks/jp/java/library/j-dyn0203/

[2] 千葉滋: “Javassist — Java バイトコードを操作するクラスライブラリ — 入門”, Java Press Vol.35, pp.76-85 (March 2004).

[3] 神崎ら: “命令のカムフラージュによるソフトウェア保護方法”, 電子情報通信学会論文誌 A Vol.J-A, No.6, pp.755-767 (June 2004).