메소드 분산을 통한 자바 프로그램 난독화 기법

최석우 박희완 한태숙
한국과학기술원 전자전산학과 전산학전공
{swchoi, hwpark, han}@pllab.kaist.ac.kr

A obfuscating technique for Java program by distributing methods

Seok-Woo Choi Hee-Wan Park Taisook Han
Dept. of EECS, Korea Advanced Institute of Science and Technology

요 약

자바 프로그램은 자바 바이트코드로 컴파일되어 배포된다. 자바 바이트코드는 심볼릭 정보를 그대로 유지하고 있기 때문에 역컴파일(decompile) 도구에 의해서 쉽게 소스 파일이 노출될 수 있다는 취약성이 있다. 따라서 자바 역컴파일에 대한 위험을 방지 혹은 최소화 할 수 있는 방법에 대한 연구가 필요하다. 본 논문에서는 이러한 보안 취약성 및 문제점을 인식하고 거기에 대처할 수 있는 대응 기법에 대한 연구로서 메소드 분산을 이용한 프로그램 난독화 기법을 제시하려고 한다. 본 논문에 제안된 난독화 기법을 사용하면 역컴파일에 성공했다 하더라도 역컴파일된 소스 프로그램을 쉽게 이해하거나 재활용하기 어렵게 만들 수 있으며 다른 난독화 기법을 적용할 수 있는 범위를 넓혀 줄 수 있다.

1. 서 론[1)]

응용 프로그램 개발에 소요되는 시간과 비용을 절감하기 위해서 현존하는 경쟁 프로그램의 역공학 분석(reverse engineering analysis) 결과가 이용된다. 응용 프로그램의 역공학 분석 작업을 위해서 역어셈블러(disassembler)나 역컴파일러(decompiler)가 사용되는데 이러한 도구들을 통해서 프로그램의 자료 구조나 제어 구조를 분석할 수 있다. 이 분석작업은 숙련된 역공학자에 의해서 이루어질 수도 있지만 자동화된 분석 도구에 의해서 더욱 간단하고 이루어질 수도 있다.

특히 자바 바이트코드에서는 역공학 분석 작업의 효율성이 커진다. 자바 프로그램은 일반적으로 클래스 파일 형태로 배포되고, 이 클래스 파일은 하드웨어 독립적인 자바 가상 기계(Java Virtual Machine)에서 실행되어야 하기 때문에 원본 자바 소스 프로그램의 심볼릭 정보를 그대로 유지하고 있다. 따라서, 자바 클래스 파일은 기계어로 번역되는 다른 언어들과 비교하여 역컴파일이 용이하다. 게다가 자바는 방대한 표준 클래스 라이브러리를 제공하기 때문에 실제 자바 프로그램은 표준 라이브러리를 호출하는 단순한 형태로 작성될 수 있고, 결국 역공학적인 분석이 더욱 용이해진다[1,2].

자바로 만든 프로그램의 저작권과 지적 재산권 보호를 위해서 다양한 방법이 제시되었다. 자바 프로그램을 혼란 변환 시켜서 비록 역컴파일에 성공했더라도 프로그램을 쉽게 이해하거나 재활용하기 어렵게 만드는 방법이 있고, 더 진보된 방법으로는 역컴파일을 시도했을 경우 결과로 생성된 프로그램이 실행되지 않도록 혼란 변환하는 방법도 제기되었다[3].

근본적으로 역컴파일 시도를 막을 수 없다면 프로그램에 저작권에 대한 정보를 워터마크로 삽입하여 프로그램의 불법적인 도용을 막고자 하는 노력도 시도되었다. 워터마크 삽입은 정적으로 프로그램에 삽입하는 방식이 있고, 프로그램이 실행하는 도중에 동적으로 메모리에 생성되는 방식이 있다. 정적인 방식은 구현이 단순하다는 장점이 있지만 쉽게 제거될 수 있고, 동적인 방식은 워터마크를 없애기 어렵지만 프로그램 실행의 오버헤드로 작용할 수 있다[4].

2. 관련연구

그림 1[1]은 코드 난독화 기법의 분류에 대한 내용이다. 항목(a)에서와 같이 코드 난독화 기법은 크게 변환 대상에 따라서 4가지로 분류된다. 가장 단순한 변환 대상은 항목(c)의 소스 코드 포맷이나 변수의 이름 등을 대상으로 하는 레이아웃(layout)이다. 그리고 항목(b)의 프로그램에서 사용된 자료구조가 대상이 되는 데이터 난독화 기법이 있고 항목(c)의 프로그램의 제어구조를 바꾸는 제어(control) 난독화 기법이 있다. 항목(e)의 Preventive 변환 기법은 역컴파일러나 난독 해석기(deobfuscator)에 의한 해독 기법을 무력화하기 위한 변환을 의미한다.

- 본 연구는 첨단정보기술연구센터를 통하여 과학재단의 지원을 받았음

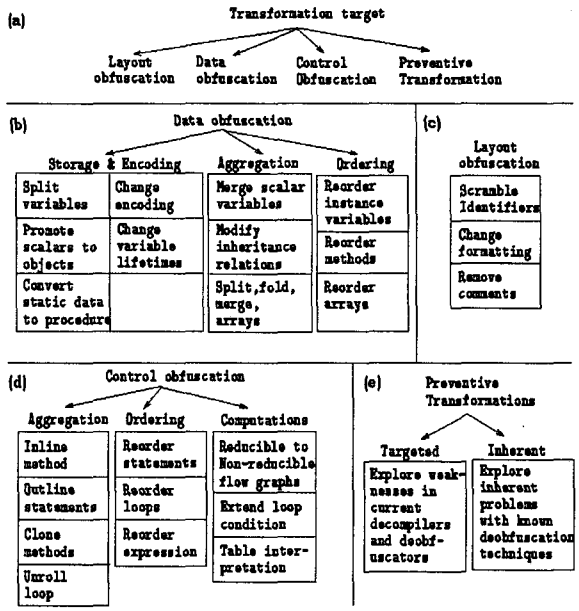

그림 1. 난독화 기법의 분류

3. 객체 지향성과 난독화

객체 지향 언어에서 가독성을 높여 주는 캡슐화(encapsulation)나 클래스의 독립성(independency)을 파괴하는 것을 통해서 난독화를 가능하게 할 수 있다[5].

3.1 캡슐화와 난독화

캡슐화(encapsulation)가 잘 되어 있는 프로그램은 은닉되어 있는 부분과 그렇지 않은 부분을 구분하는 것을 통해서 가독성을 증가시킨다. 따라서 은닉된 부분을 공개하는 것을 통해 가독성을 떨어뜨릴 수 있다. 자바에서는 접근 한정자(access modifier)를 통해서 클래스의 멤버를 은닉 한다. 접근 한정자들을 public으로 바꾸는 것을 통해 캡슐화를 깰 수 있다. 접근 한정자를 public으로 바꾸는 것은 실행에는 영향을 주지 않는다.

3.2 메소드의 분산과 난독화

클래스는 연관성이 있는 메소드들과 속성들의 집합이다. 메소드와 속성 사이의 연관성이 가독성을 높여 주기 때문에 메소드를 클래스에서 분리하여 다른 클래스로 옮긴다면 난독화가 일어난다.

메소드를 다른 클래스로 옮기게 되면 그 메소드를 호출하는 코드가 다 바뀌어야 한다. 또한 메소드 바디에서 옮겨지기 전에 속했던 클래스의 멤버에 직접 접근할 수 없기 때문에 클래스 멤버에 접근하는 코드가 바뀌어야 한다.

4. 난독화 알고리즘

이 논문에서는 캡슐화를 해체하고 메소드를 옮기는 한 가지 방법을 제안한다. 난독화는 크게 두 단계로 이루어진다. 첫째는 정적 데이터와 코드들을 한 클래스로 옮기는 과정이며, 둘째는 동적 코드 즉 인스턴스 메소드들을 한 클래스로 옮기는 과정이다. 셋째는 클래스의 상속 관계를 편평화(flattening)하는 과정이다. 이 과정들을 통해 모든 클래스들의 코드들과 스태틱 데이터들은 한 클래스로 옮겨지고 나머지 클래스들은 객체를 생성될 때 사용되는 멤버 변수만을 가지게 되며 상속 관계가 깨어지게 된다.

4.1 정적 변수를 포함한 정적 코드의 수집

프로그램의 모든 정적인 부분을 모으기 위한 클래스를 스태틱 루트 클래스(static root class)라고 하자. 모든 정적인 부분에는 클래스의 정적 변수와 인터페이스의 상수들 그리고 정적 메소드와 정적 초기화 코드(static initializer)가 포함된다. 여기에 인스턴스 메소드 중 오버라이딩 되지 않는 메소드가 추가될 수 있다. 이 메소드들은 정적 결합(static binding)이 될 수 있기 때문이다.

정적 변수들을 옮길 때는 그 정적 변수를 직접 참조하고 있던 메소드에서 그 변수를 참조하고 있던 부분에 스태틱 루트 클래스의 경로를 추가해 준다. 모든 정적 변수를 스태틱 루트 클래스에 옮기는 과정에서 정적 변수를 초기화하는 정적 초기화 코드도 함께 옮겨져야 한다.

정적 메소드를 옮길 때는 그 정적 메소드를 호출하고 있는 부분을 정적 메소드를 옮길 클래스를 참조하도록 경로를 추가해 주거나 수정해 준다.

정적 변수와 메소드를 옮기는 과정에서 같은 이름을 가지는 변수나 메소드가 있을 수 있다. 정적 변수와 메소드들은 정적 결합이 이루어지기 때문에 이름을 바꾸어주어도 상관이 없다. 따라서 변수와 메소드를 옮기는 과정에서 이름이 겹치지 않게 바꾸어주면 이름이 겹치는 문제를 해결할 수 있다. 또한 변수 이름이 오버라이딩 될 수 있는데 변수는 정적 결합이기 때문에 이름을 바꾸는 방법으로 이 문제를 해결할 수 있다.

인터페이스에 포함된 변수는 모두 상수이므로 이 과정을 통해 옮겨지며 인터페이스에 포함된 메소드 선언은 실행하는 데 영향을 주지 않기 때문에 생략 가능하다. 따라서 이 과정을 통해서 인터페이스에 해당하는 클래스를 제거할 수 있다.

4.2 동적 코드 수집

동적 코드에는 생성자와 인스턴스 메소드들이 포함된다. 인스턴스 메소드에서는 그 메소드가 속한 객체의 변수에

직접 접근할 수 있다. 인스턴스 메소드가 다른 클래스로 옮겨지기 위해서는 객체 변수에 접근하기 위한 방법이 제공되어야 하므로 메소드를 호출할 때 객체에 대한 참조자를 함께 넘겨 주어야 한다. 따라서 인스턴스 메소드의 선언부에 객체 참조자를 인자로 추가시켜 주고 메소드 바디에서 직접 참조하고 있던 변수들에 인자로 받은 객체에 대한 경로를 추가하여 준다. 변경된 메소드가 오버라이딩되지 않았다면 그 메소드는 정적 메소드로 변경 후 스태틱 루트 클래스로 옮길 수 있다.
오버라이딩 된 메소드는 실행 시간에 객체 타입에 따라서 다른 메소드가 불려지기 때문에 정적 메소드로 변화시키기 위해서 다음과 같이 타입 변수를 추가하여 동적 결합을 수동으로 해 준다.

```
class A {
  A_variable_declarations
  p(parameter_declarations) {Ap_body}
  method_definitions
}
class B extends A {
  B_variable_declarations
  p(parameter_declarations) {Bp_body}
  method_definitions
}
```

그림 2. 오버라이딩 메소드가 포함된 변환 전 코드

```
class A {int t = 0x123;A_variable_declarations}
class B extends A {int t = 0x124;B_variable_declarations}
class StaticRoot {
  AB_static_variables_declarations
  public static p1(Object o,params){Ap_body}
  public static p2(Object o,params){Bp_body}
  public static p(Object o,params){
    if (t == 0x123) p1(o,params);
    if (t == 0x124) p2(o,params);
  }
  method_definitions
}
```

그림 3. 메소드 수집 난독화로 변환된 코드

메소드 바디에서 super 키워드를 통해 상위 클래스의 메소드를 호출할 수 있게 하고 있다. 이 경우 호출하는 메소드가 어떤 메소드인지 실행 전에 구분할 수 있기 때문에 해당되는 클래스의 메소드를 호출하도록 하면 된다.
객체 선언과 동시에 초기화되는 명령들은 생성자가 실행되기 전에 실행되므로 생성자의 가장 위에 포함시킨다. 생성자는 객체 생성시 실행되는 특수한 메소드로 볼 수 있다. 따라서 생성자를 일반 메소드와 같은 방법으로 변환한 후 new 연산을 수행한 후 생성자가 변환된 메소드를 호출하는 명령을 추가해주면 된다.

4.3 클래스 구조 편평화

위의 변환 과정을 거치고 나면 인터페이스가 제거되고 생성자를 포함한 모든 동적 메소드가 제거된 상태이다. 따라서 상속할 때 상위 클래스에 있던 모든 변수를 포함하도록 변환시킬 수 있다. 변수는 정적으로 결합되므로 이름 변경을 통해 이름이 겹치는 것을 방지할 수 있다.

5 결론 및 향후 연구

본 논문에서는 캡슐화를 깨고 모든 정적 변수와 메소드를 한 클래스에 모으고 클래스 구조를 편평화하는 것을 통해서 자바 프로그램을 난독화하는 한 가지 방법을 제시하고 있다. 자바 메소드의 크기가 작기 때문에 다른 난독화 기법을 적용하기 어려웠던 점이 메소드 수집에 의해서 해결될 수 있다. 또한 코드에 워터마크(watermark)를 삽입할 때 이 방법을 함께 사용한다면 더 많은 방법을 찾아낼 수 있다는 장점이 있을 것이다.
향후 연구로는 이 방법과 함께 제어 난독화를 사용하는 것을 통해서 정적 분석(static analysis)에 의한 난독화 해제를 방지하는 방법에 대한 연구와 난독화 도구를 개발하여 코드 크기와 실행 속도 측면에서의 유용성을 확인하는 것을 들 수 있다.

6. 참고문헌

[1] Christian Collberg, Clark Thomborson, Douglas Low, *A Taxonomy of Obfuscating Transformation*, Thechnical Report #148, Department of Comuter Science, The University of Auckland, 1997

[2] Christian Collberg, Clark Thomborson, Douglas Low, *Manufacturing Cheap, Resilient, and Stealthy Opaque Constructs*, SIGPLAN-SIGACT POPL98, ACM Press , San Diego, CA, January 1998

[3] Jien-Tsai Chan and Wuu Yang, *Approaches to make the decompiled Java programs uncompilable*, Proceedings of the Eighth Workshop on Compiler Thecniques for High-Performance Computing, 102-111, 2002

[4] Christian Collberg, Clark Thomborson, *Watermarking, Tamper-Proofing, and Obfuscation Tools for Software Protection*, IEEE Transactions on Software Engineering, Vol28, No 6, June 2002

[5] Kazuhide FUKUSHIMA, Kouichi SAKURAI, *Obfuscating of Embedding Position of the Software Fingerprinting*. SCIS 2003 The 2003 Symposium on Cryptography and Information Security Hamamatsu,Japan, Jan.26-29,2003