자바 애플릿 보안을 위한 역컴파일 방지 기법에 관한 연구

손태식[U] 서정택 장준교 김동규

아 주 대 학 교 정보통신공학과
{ tsshon, sjtgood, visionjk, dkkim }@madang.ajou.ac.kr

A Study on the protection technique of a Applet decompilation in Java Environment

Tae-Shik Shon[U], Jung-Taek Seo, Jun-Kyo Jang, Dong-Kyoo Kim
Dept. of Information and Communication, Ajou University

요 약

Java는 간결하고 객체 지향적이며 플랫폼 독립적으로 실행 가능한 특성을 지니고 있어, 웹 상에서의 증대된 표현 능력과 유연성을 제공해 왔다. 하지만 이렇게 개방적이고 넓은 수용 가능성은 보안에 있어 많은 위협 취약점을 발생시키게 한다. 특히 현재 네트웍을 통해 널리 사용되는 애플릿은 그 자체가 사용자의 시스템에 전송되어 실행되므로, 만일 악의의 사용자가 그 애플릿 파일을, 역 컴파일러를 사용해 소스 코드를 얻어낸다면, 애플릿 서비스를 하는 시스템 및 네트웍에 대한 보안 문제 및 경제적 피해는 물론이고, 애플릿 개발자 및 다른 사용자들에게 있어 많은 문제를 일으킬 수 있다. 따라서 본 논문에서는 현재 자바 애플릿 역컴파일에 의한 보안 취약성 및 문제점을 진단하고, 거기에 대처할 수 있는 대응 기법에 대한 연구를 통해 보다 나은 자바 애플릿 역컴파일 방지 기법을 제시하려고 한다. 본 논문에 제안된 방법은 역컴파일 방지를 위해 클래스 파일에 혼란 코드를 삽입하는 방법으로서, 그 기법 및 사용 방법을 제시하고 역컴파일 방지 도구의 모델을 제안한다. 또한 앞으로의 연구는 여기서 제안된 자바 애플릿 역컴파일 방지 도구 모델의 세부적 구현으로 진행되어야 하겠다.

1. 서론

급속히 발전하는 인터넷 환경 속에서 기존의 클라이언트/서버 기반의 분산 컴퓨팅, WWW(World Wide Web) 환경에서의 많은 한계점을 극복하도록 해준 JAVA는 간결하고, 객체 지향적이고, 플랫폼에 독립적으로 실행되는 프로그래밍 언어라는 특징을 가지고 있다. 여기에서 주목해야 될 점은 플랫폼 독립적인 자바의 특징으로서, 곧 네트웍을 통해 애플릿이라는 실행 가능한(Executable Contents) 프로그램이 전송되어 어떤 사용자의 시스템에서도 동작 가능하다는 것이다.

이러한 범용성이 가능한 이유는 자바 소스 컴파일 과정에서 생성되는, 클래스 파일이 전송 받은 시스템의 프로세서 종류에 따라, 실행 가능한 특정 기계어(명령어) 코드를 생성하는 것이 아니라, 이미 컴파일 과정에서 자바 가상 머신이라는 공통의 플랫폼에 알맞게 실행되는 1byte의 명령어 코드를 생성하기 때문이다. 이 바이트 코드는 분기에 관련된 2개의 명령어 코드를 제외한 모든 명령어 코드가 바이트 단위로 구성되어 있으며, 대부분의 명령어는 오퍼랜드를 가지고 있지 않는, 오직 하나의 실행 코드만을 가지고 있다. 이처럼 경량화 된 자바 byte-code는 자바 가상 머신의 클래스 로더에 탑재되기 전에 클래스 파일의 크기를 줄일 수 있게 해주며, 또한 자바 가상 머신에 내장된 자바 표준 라이브러리의 사용 역시, 클래스 파일의 크기를 줄여주므로, 자바 가상 머신에서 바이트 코드의 최적화 된 수행을 가능하게 한다. 이러한 플랫폼 독립적인 자바 가상 머신 상에서의 바이트 코드의 실행 가능성은 역공학적(reverse engineering)측면에서 자바 클래스파일을 자바 소스 파일로 역컴파일 하는 것을 가능하게 할 수 있다는 취약성을 가지고 있다.

따라서 본 논문에서는 네트웍을 통해 전송되는 애플릿 파일이 역컴파일 됨으로써 야기되는 문제점과 이를 방지하기 위한 기존의 자바 역컴파일 대응 방안에 대한 분석과 함께, 자바 역컴파일을 방지 혹은 최소화 할 수 있는 방지 도구 모델을 제시하며, 근본적인 자바 역컴파일에 대한 대응책을 모색해본다.

2.기존의 자바 애플릿 역컴파일 방지 기법에 관한 연구

네트웍을 통해서 전송되는 애플릿 파일의 역컴파일 취약성에 의해서 발생되는 문제점으로는 첫번째로 프로그램에 대한 저작권 보호의 어려움, 두 번째로 애플릿을 통해 서비스하는 네트웍의 보안에 취약성을 야기, 세 번째로 애플릿 소스 분석을 통해 얻어낸 정보를 사용해 불법적인 서비스 이용 문제, 그리고 서비스를 제공하는 서버에 대한 해킹 등의 여러 문제점을 가지게 된다. 아래의 예는 간단한 자바 클래스 파일과 "mocha"라는 자바 역 컴파일러를 통해서 역컴파일이 실제로 어느 정도나 소스 파일을 노출시키는지 간단한 예를 들겠다. 우선 아래의 test.java 파일은 역컴파일 되기 전의 소스 코드이고, 아래의 test.mocha는 mocha라는 역 컴파일러를 통해 test.class 파일을 역컴파일 한 결과 파일이다. 즉, 아래의 결과에서도 볼 수 있듯이 역 컴파일러는 자바 클래스 파일에서 원래의 소스 파일을 거의 근접하게 복원하는 것이 가능함을 알 수 있다.

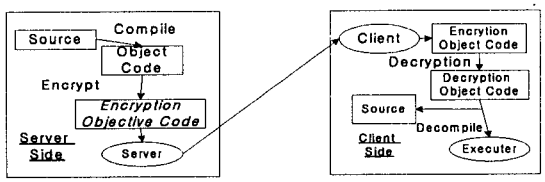
```
//Decompiler test java file
public class test {
        public static void main( String args[] ) {
        System.out.println("Hello!!"); }
}
                        [ test.java ]

/*Decompiled by Mocha from test.class */
import java.io.PrintStream;
public synchronized class test
{   public test() { }
    public static void main(String astring[]) {
        System.out.println("Hello!!");  }
}
                        [ test.mocha ]
```

따라서 자바 역컴파일을 방지하기 위해서 법적으로나 기술적으로 많은 대응책이 연구되고 있으며, 앞으로도 많은 연구가 진행될 것이다. 우선 법적인 경우의 자바 역컴파일 대책에 있어서는 이미 역컴파일로 인한 영향 및 피해가 발생한 후에 그 대응이 일어날 수 있게 되어, 문제의 성격에 따라 이미 그 대응이 시기를 놓치는 경우가 발생 하는

경우가 대부분이며, 일반적인 개발 업체의 경우에는 영세성과 비전문성(법적인)으로 인해 그 대응 및 해결이 쉽지 않은 것이 또한 사실이다. 결국 개발자 및 자바 연구자들에 의한 기술적인 자바 역컴파일 대응 방법의 강구가 가장 바람직한 대안이라고 할 수 있다. 그렇다면 현재 제안되고, 사용되고 있는 몇 가지의 자바 역컴파일 대응 방안에 대하여 살펴보겠다.

2.1 암호화 및 해쉬 함수의 사용

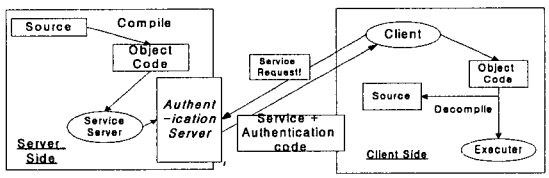


[그림 1] 암호화 기법 이용

암호화 기법은, 서버 측에서 클래스 파일의 생성 후에 암호화하여 클라이언트에게 전송하고, 역시 클라이언트는 사전에 정해진 키로 애플릿 파일을 복호화하여 사용한다. 여기에는 적용 환경에 따라 공개키나 관용키 암호화를 이용할 수 있다. 하지만 단순히 애플릿에 대한 해쉬 함수 값을 사용하여 애플릿을 전송하는 방법과 해쉬 함수 모듈을 애플릿에 삽입하여, 해쉬 값을 생성할 때 애플릿에 대한 해쉬 코드값과 서버측에서 애플릿에게 넘겨주는 특정 값을 같이 사용하여 해쉬 코드 값을 생성해내고, 이 생성된 해쉬값을 이용하여 애플릿의 변질을 확인하여 역컴파일을 통해 소스가 노출되었는지 알아내는 방법 등이 있다.

2.2 사용자 인증 기반의 역컴파일 방지 기법

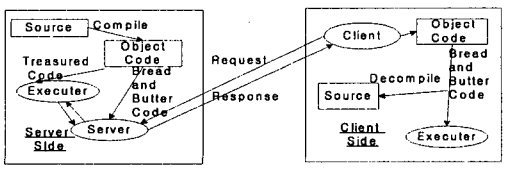
사용자 인증 기반의 역컴파일 방지 기법이란, 서버측에서 서비스를 제공받는 사용자들이 인증을 거쳐 서버측의 서비스에 접근 가능하도록 하며, 그 이후에도 사용자측에서 다시 돌아오는 정보에 대해서는 그 각각에 대해서 처음 사용자가 인증 될 때 부여한 고유 번호를 가지고 전송되게 하므로, 악의의 사용자가 애플릿을 역컴파일하여 소스를 알아내는 것을 방지 할 수 있다. 하지만 사용되는 인증 코드 위조나 변조의 문제점이 발생할 수 있기 때문에 위에서 살펴본 암호화기법과의 적절한 조화를 필요로 한다.



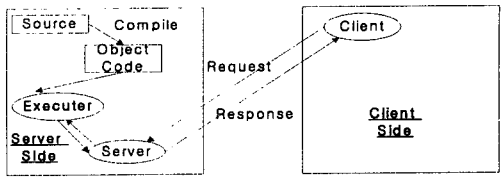
[그림 2] 사용자 인증 기법 이용

2.3 서버측 접근 제어 기법 이용

서버에서 서비스를 제공하는 경우에, 실행 파일을 서버 측에서만 실행하여, 단지 사용자에게는 서비스의 결과만을 제공하여 클래스 파일을 직접적으로 전송하지 않는 방법과(Server Side Execution), 또는 서버측에서 제공되는 서비스의 성격에 따라 역컴파일을 통한 소스파일 노출에도 보안 및 특별한 문제가 발생하지 않는 경우에는, 애플릿을 이용한 서비스와 서버측에서 실행한 결과를 부분적으로 제공하는 서비스를 병행하는 것(Partial Server Side Execution)도 하나의 자바 역컴파일 대응 방안으로 제시된다.



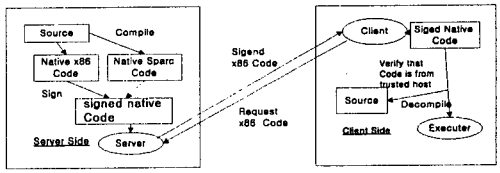
[그림 3-1] Server Side Execution



[그림 3-2] Patial Server Side Execution

2.4 네이티브 코드(Native Code)를 이용

서버에게 서비스를 요청하는 사용자의 시스템 아키텍쳐를 식별하여, 서버에서 실행코드를 생성할 때, 사용자의 아키텍쳐에 맞는 네이티브 코드를 생성하여 전송하는 방법이다. 비록 네이티브 코드를 생성하여 실행하는 경우라도 역컴파일의 가능성은 계속적으로 존재하지만, 자바



바이트 코드의 경우보다는 역컴파일이 더 어렵기 때문에 비교적 소스 코드의 노출에 있어서 안전하다고 할 수 있다.

[그림 4] native code 사용 기법

2.5 혼란(Obfuscation) 코드 삽입 기법 이용

혼란(obfuscation) 코드 삽입이란, 자바 클래스 파일에, 그 실행 결과에는 아무런 영향을 미치지 않는 코드를 삽입하는 것을 말한다. 비록 코드의 삽입이 프로그램의 크기나 메모리의 할당량 증가 및 연산 오버헤드의 증가 등을 가져오지만, 시스템이나 네트웍에 추가적 장비 없이 사용 될 수 있다. 그렇다면 혼란코드가 자바 클래스 파일에 어떻게 삽입되어 역컴파일을 방지하는지 알아보기로 하겠다.

```
public class test1 {
        public static void call_f1() {}
        public static void call_f2() {}
        public static void main( String args[] ) {
                call_f1(); call_f2(); }
}
```

[test1.java]

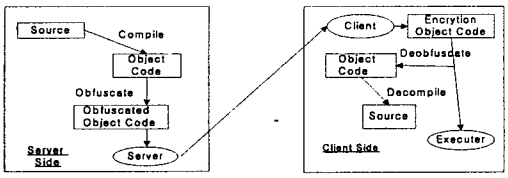
다음과 같은 자바 소스 파일이 있다고 하면, 우선 위의 소스 파일에 대한 클래스 파일을 생성한 후에, 소스 파일에 사용된 변수, 배열, 메소드, 상속 관계 및 호출 플로우등에 따라 적절한 혼란 코드가 클래스 파일에 삽입된다. 아래의 소스 파일은 test1.java를 통해 생성 된 자바 클래스 파일에 혼란코드가 삽입되었다고 가정한 후, 그 클래스 파일을 다시 역컴파일한 자바 소스 파일을 보여주고 있다.

```
public class test2 {
        public static void call_f1() {      }
        public static void call_f2() {      }
        public static void main( String args[] ) {
                call_f1();
                if ( 1 == 2 )
                { call_f2(); }
                if ( 1 > 2 )
                { call_f1(); }
                call_f2();
                System.out.println("Hello!!");      }
}
```

[test2.java]

test2.java 파일은 test1.java 파일의 클래스 파일에 어떤 경우에라도 항상 참이 되어 실제 프로그램에는 영향이 없는 혼란코드를 삽입한 후 역컴파일한 소스 파일이다. 위의 예제는 역컴파일을 막기 위한 혼란코드의 삽입을 보여주기 위해서 간단한 예를 든 것이며, 실제로는 역컴파일을 어렵게 하기 위해 혼란코드의 삽입 양 및 복잡도 등의 여러 역컴파일 방지 강도 요소의 조절이 필요하며, 또한 코드의 증가에 따른 컴파일 성능 오버헤드 역시 역컴파일 방지 필요성, 중요도에 따라 적

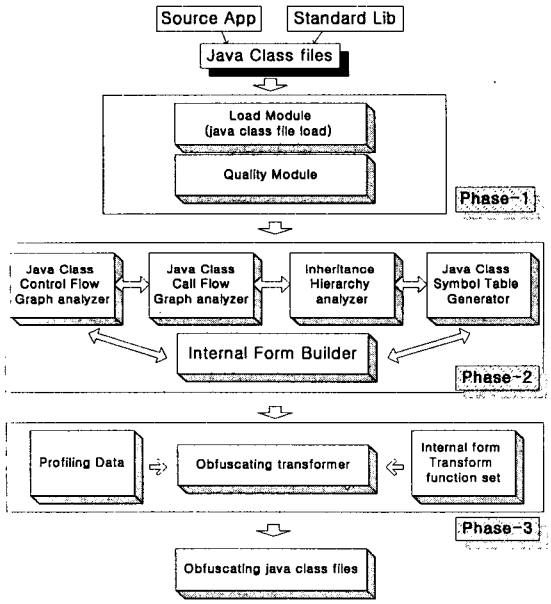
절한 조절이 필요하다. 다음 장에서는 혼란코드 삽입 및 변환을 통한 자바 역컴파일 방지 도구모델을 제안 하고자 한다.



[그림 4] 혼란 코드 삽입 기법 이용

4. 자바 애플릿 보안을 위한 역컴파일 방지 도구 모델

제시된 자바 애플릿 역컴파일 방지 도구 모델은 자바 클래스 파일을 컨트롤 플로우나 클래스 상속 관계 등을 이용하여 클래스 파일의 특성을 파악한 후, 혼란 코드를 삽입해 역컴파일이 실행 할 경우에도 적의를 가진 사용자가 원래 소스 파일을 알아내기 어렵게 만든다. 모델은 크게 세 단계의 절차를 통해 혼란 코드가 삽입된 역컴파일 방지 클래스 파일을 만든다. 본 논문에서는 역컴파일로 인해 발생할 수 있는 위험요소에 따른, 애플릿 보안 기법을 분석하고, 하나의 대안으로 역컴파일 방지 도구의 모델을 제시하고자 하며, 각 모듈의 세부 구현은 본 논문에서는 다루지 않는다.



Phase-1

첫 번째 단계에서는 클래스 파일을 읽어 들이고(load module), 사용자가 원하는 역컴파일 방지 강도를 설정(quality module)하게 한다. 여기서 역컴파일 방지 강도란, 보통 얼마만큼 역컴파일된 파일을 해석하기 어렵게, 복잡하게 할 것인지, 어느 정도의 양으로 덧없는 코드(dummy code)를 삽입할 것인지, 어느 수준의 실행 시간 및 자원이 필요한 역컴파일 방지 기법을 사용할 것인지 등으로 나타낼 수 있다.

Phase-2

두 번째 단계는 컨트롤 흐름 분석, 콜 흐름 분석, 상속 관계 분석 및 심볼 테이블 생성을 통해, 주어진 클래스 파일에 변환 함수와 프로파일링 데이터를 적용할 수 있는 중간 포맷으로 바꾼다.

Control Flow Graph Analyzer : 모든 메소드와 클래스에 대해서 각각의 메소드 및 클래스가 가지고 있는 변수 및 키워드를 구성하고, 메소드와 클래스의 사용 빈도를 카운트 한다.

Call Flow Graph Analyzer : 메소드와 클래스의 사이의 호출관계를 구성

Inheritance hierarchy Analyzer : 각 클래스들 사이의 상속 관계를 분석한다.

Symbol Tabler generator : 클래스 파일에 나타난 상수(constant)에 대해 분석하여 테이블로 구성한다.

위의 네 가지 분석을 수행한 후 실제로 혼란코드를 삽입하기 위한 변환에, 알맞은 중간 포맷을 생성한다.

Phase-3

세 번째 단계는 실질적으로 입력된 클래스 파일에 대한 변환 과정이 일어나는 단계로서 두 번째 단계에서 생성된 중간 포맷에, 변환 함수 집합(transformation function set)에서 알맞은 변환 함수를 적용하고 프로파일링 데이터를 사용한다.

Transformation function set : 역컴파일 방지 도구 모델에서 가장 중요한 부분으로서 둘째 단계에서 분석된 데이터에 의해 실제적 혼란(Obfuscating) 기법을 적용하게 해준다. 아래의 표는 혼란 기법을 분류해 놓은 표로서, 우선 4가지 data, layout, control, preventive의 기준에 따라 혼란 기법을 적용하고, 다시 세부적인 특성에 따라 둘째 단계에서 생성된 중간 포맷을 최종 혼란 코드로 변환한다. 본 논문에서는 각각 기법에 대한 세부적용 사항은 다루지 않는다 .

Data Obfuscation	Storage&Encoding	Spilt variables, Convert static data , Change variables lifetime etc.
	Aggregation	Modify inheritance hierarchy, Spilt/fold/merge array etc.
	Ordering	Reorder instance variable, Reorder method
Layout Obfuscation	Scrambling identifiers, Changing formating, Removing comments, etc..	
Control Obfuscation	Aggregation	inline method, outline statements, clone method, etc..
	Ordering	reorder statements, reorder loops, reorder expression, etc..
	Computation	Extend loop condition, etc..
Preventive Transformation	Targeted	Explore weaknesses in decompilers and deobfuscater, etc..
	Inherent	Explore inherent problems with known deobfuscation techniques.

이렇게 세 단계의 과정을 거쳐서 최종적으로 각 클래스 파일의 특성과 사용자의 역컴파일 강도 요구에 알맞은 혼란 코드가 적용된 역컴파일 방지 클래스 파일(obfuscating class files)이 생성된다.

4. 결론

본 논문에서는 네트웍을 통해 전송되는 자바 애플릿의 역컴파일에 의한 위협요소를 분석하고, 그 대응 방안에 관한 연구와 자바 애플릿 역컴파일 방지 도구에 관한 모델을 제시하였다. 여기서 제시된 모델에 대해 부가적으로 요구되는 연구 목표로는 역컴파일 방지 도구 모델의 각 모듈에 대한 더욱 세부적인 기능 분석 및 정의와 그에 대한 실제적인 세부 구현이 필요하리라 생각된다.

마지막으로 역컴파일을 통한 자바 소스의 노출은, 원천 소스 공개라는 점에서 개발자 및 그 프로그램 사용자에게 있어 물질적, 정신적으로 막대한 피해를 미칠 수 있고, 만약 애플릿이 사용되는 환경이 공개 네트웍에서의 서비스 제공 환경이라면, 네트웍 사용자 전체에게도 영향이 파급될 수 있는 큰 문제를 야기 시킬 수 있다. 따라서 자바 애플릿 역컴파일 방지를 많은 기법이 제시되고 있지만, 어떤 방법을 사용할지라도 완벽하게 역컴파일을 막을 수 없고, 또한 역컴파일을 방지하기 위해 시스템 혹은 네트웍 유지에 부가적인 많은 노력 및 그에 따른 역기능이 나타나게 된다. 따라서 근본적인 해결을 위해 자바 클래스 파일의 플랫폼 독립적인 특성에서 오는 역공학적 측면에 대한 보안 요구 사항이 자바 가상 머신(Java Virtual Machine) 자체에서 이루어져야 한다.

5. 참고문헌

[1] C. Collberg, Breaking Abstractions and Unstructuring Data Structures, IEEE International Conference on Computer Languages (ICCL'98), pp 56.70, May.1998

[2] Manufacturing Cheap, Resilient, and Stealthy Opaque Constructs, ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL98), pp 23-37, Jan.1998

[3] H.P.van.Vliet, Mocha Decompiler, http://www.brouhaha.com/~eric/computers/mocha.html,1996

[4] Sun systems, Java FAQ, http://forum.java.sun.com/,2000

[5] Frank Yellin, Low Level Security Java, http://www.javasoft.com/sfaq/verifer.html