2000년 한국정보처리학회 추계 학술발표논문집 제7권 제2호

시간제한 블랙박스 보안을 이용한 자바 이동 에이전트의 보호

서준혁 이승우 방대욱
계명대학교 대학원 컴퓨터공학과
{jun,swlee}@jinri.kmu.ac.kr,dubang@kmucc.keimyung.ac.kr

A Java Mobile Agent Protection Using Time Limited BlackBox Security

Jun-Hyuk Seo Sung-Woo Lee Dae-Wook Bang
Dept. of Computer Engineering Graduate School, Keimyung University

요약

본 논문에서는 자바로 작성된 에이전트가 신뢰할 수 없는 호스트에서도 자신의 코드와 데이터를 노출시키지 않고 안전하게 수행할 수 있게 하는 3가지 시간제한 블랙박스 보안 방법 즉 변수 분해, 코드 어포스캐이팅 그리고 클래스 지연로딩을 제안한다. 변수 분해는 호스트가 에이전트 수행을 감시하여 에이전트 내의 데이터를 파악하는 것을 어렵게 하며, 코드 어포스캐이팅은 실행코드에 더미코드와 가상코드를 삽입하여 역컴파일 등과 같은 기법으로도 코드의 의미 분석을 힘들게 하여 에이전트를 보호한다. 그리고 클래스 지연로딩은 변수에 접근하는 함수들을 암호화하여 실행시간까지 그 접근 방법의 공개를 차단한다. 제안하는 블랙박스 보안방법들은 호스트가 에이전트 내부를 파악하는데 상당한 시간을 소요하게 하여 적어도 에이전트가 실행되는 동안은 에이전트가 보호되게 하는 보안 방법이다.

1. 서론

에이전트 보안의 연구는 기존의 암호화 기술에 의해 상당한 기술적 진전이 있어 왔다. 하지만 신뢰할 수 없는 호스트로부터 에이전트를 보호하는 보안은 에이전트를 수행시키는 주체인 호스트가 실행전이나 실행 중간에 다양한 방법을 통해서 에이전트를 해킹할 수 있기 때문에 기존 보안 메커니즘[1]의 적용만으로 완전하게 실현할 수가 없다.

신뢰할 수 없는 호스트가 외부에서 자신에게 이동해 온 에이전트를 공격하는 유형에는 다음 8가지가 있다.[3]

(1) 코드분석
(2) 데이터감시
(3) 실행흐름감시
(4) 코드조작
(5) 데이터조작
(6) 실행흐름조작
(7) 마스커라이딩(Masquerading)
(8) 수행거부

따라서 에이전트를 호스트로부터 보호하려면 공격유형별로 해결책이 필요하다.

(1),(3),(6)의 유형에서는 코드 전체를 대상으로 하는 경우와 코드 각 라인을 대상으로 하는 경우에 따라 에이전트 보호에 차이가 있다. 전자의 경우 에이전트 코드가 라이브러리 코드를 포함한다면 보호가 불가능하다. 후자의 경우 특정기능의 정확한 시작 위치가 알려지면 해킹코드의 삽입으로 공격이 가능하다. 따라서 라이브러리 코드를 사용하지 않고 에이전트 코드를 구성하고 실행흐름과 실행코드의 의미간의 관계를 파악할 수 없게 해야만 코드조작과 관련된 공격을 막을 수 가 있다.

(2),(5)의 유형에서는 해커가 데이터의 정확한 위치를 파악하고 공격하면 에이전트 보호가 불가능하다. 그러나 위치가 알려지더라도 해당 데이터의 정확한 의미를 파악할 수 없게 한다면 데이터를 변경하는 등의 공격은 피할 수가 있다.

(7)의 유형으로부터 에이전트를 보호하려면 한쪽에서 다른 쪽에 대한 인증을 하기 위해서 에이전트 스스로 자율성을 가지고 수행을 할 수 있어야 한다. 그러나 에이전트 특성상 호스트에 의해서 수행되기

때문에 자율성을 가지고 수행한다는 조건을 만족할 수가 없다. 그렇지만 코드와 실행흐름을 조작할 수 없게 하고 공용키 등의 정보를 보호할 수 있다면 마스커라이딩을 감지할 수 있다.

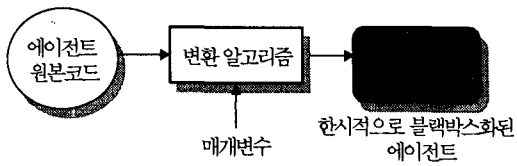
(8)의 유형에서 에이전트를 보호하려면 서비스를 거부한 호스트에 대해 그 이후로의 에이전트 이동에 대해 해당 호스트로 이동을 제외시키는 방법을 사용할 수 있다. 그러나 이것은 일반적인 방법이 될 수 없다. 다른 방법으로는 신뢰할 수 없는 호스트가 존재하는 환경에서 서비스 실행에 대한 기록을 에이전트 내부에 남기고 기록된 정보와 정보에 접근하는 실행흐름을 보호하는 방법이 있다.

이상과 같이 에이전트의 공격 유형을 분석해 본 결과, 호스트가 에이전트를 공격하려면 적어도 에이전트의 코드 사양과 데이터 사양을 파악해야 한다. 코드와 데이터 사양의 노출을 방지하기 위한 방법으로 이동 암호화[6]와 시간제한 블랙박스 보안[2,5]이 알려져 있다. 본 논문은 자바로 작성된 에이전트의 실행 코드의 각 행과 이에 대한 의미 사이의 관계를 이해할 수 없게 하고 데이터 각각의 의미를 추측할 수 없게 함으로써 신뢰할 수 없는 호스트로부터 에이전트를 보호하는 3가지 시간제한 블랙박스 보안 방법을 제안한다.

본 논문의 구성은 다음과 같다. 2절에서는 시간제한 블랙박스 보안 방법을 정의하고, 3절에서는 데이터 사양의 파악을 어렵게 하는 변수 분해 방식을 제안하고, 4절에서는 역컴파일 등과 같은 방법으로도 코드의 의미 분석을 힘들게 하기 위해 실행코드에 더미코드와 가상코드를 삽입하는 코드 어포스캐이팅 메커니즘을 제안한다. 그리고 5절에서는 분해된 변수에 접근하는 함수들을 암호화하여 실행 시간까지 그 접근 방법의 공개를 차단하는 클래스 지연로딩을 제안하고 기술한다.

2. 시간제한 블랙박스 보안

에이전트에 대한 블랙박스 보안은 영속적으로 에이전트의 코드와 데이터 명세에 대해 조작과 감시가 불가능하도록 하기 위해 에이전트 코드의 가독성을 떨어뜨리고 에이전트 내부 구조 명세를 얻으려는 호스트의 사전식 공격을 막기 위해 매개변수를 이용해 원본과 똑같은 명세를 가진 다양한 블랙박스화된 에이전트를 생성하는 기법이다. 그러나 이 기법의 실제구현은 에이전트를 해킹하려는 크랙커에게 충분한 시간이 주어진다면 에이전트의 내부 구조 명세가 노출될 수 있다. 이런 문제의 해결 방안으로 Hohl은 시간제한 블랙박스 보안 기법을 제안하였다. <그림1>은 시간제한 블랙박스 보안 기법을 나타낸



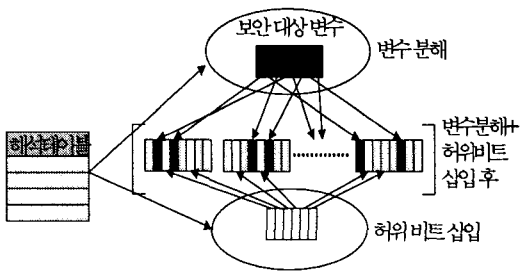
<그림 1> 시간제한 블랙박스 보안기법

것이다. 시간제한 블랙박스 보안[2,5]의 주된 개념은 블랙박스 보안에 시간적인 제한 요소를 추가해 에이전트가 실행되는 일정한 시간동안에만 블랙박스 보안의 유효성을 보장하려는 것이다. 이 기법은 변환 알고리즘을 이용하는데 이 알고리즘은 에이전트 원본 코드를 입력 후 서로 다른 블랙박스화된 에이전트를 생성하기 위해 매개변수를 이용한다.

본 논문은 이 개념을 자바환경에 구현한 시간제한 블랙박스 보안을 제안한다.

3. 변수의 분해

자바는 8개의 기본 자료형을 가지고 있다. 보안이 필요한 변수의 값이 노출되지 않도록 자료형에 의해 정해진 크기의 변수 기억장소를 분해하여 분산 저장하고 참조할 때는 재구성하도록 한다. 변수의 분해와 재구성에는 해석 테이블을 사용한다. <그림 2>는 해석테이블을 이용해 보안이 필요한 변수를 분해하는 그림이다. 보안변수를 분해할 때 해석 테이블 안에 임의의 개수만큼 새로운 변수를 생성하고, 보안변수 값을 비트 단위로 분해하여 새로운 변수들 내의 임의의 위치에 각각 저장하고 데이터 감시와 조작 공격에 대응하기 위해 허위 비트들도 같이 삽입한다.



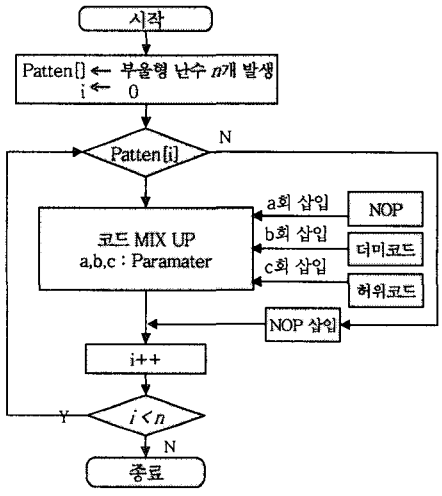
<그림 2>보안대상 변수 분해 형태

보안 변수의 분해에는 다음 두 가지 전략을 사용한다. 첫 번째는 보안이 유지되어야 하는 변수에 저장되는 값들은 동일한 형의 변수 집합에 저장시킨다. 그리고 변수집합에 대하여 원래의 변수 값은 분해하여 비트 단위로 임의의 순서와 위치에 저장시킨다. 비트 값들이 저장되는 공간을 사상공간이라 지칭한다면, 사상공간의 크기는 가장 큰 크기를 가지

는 기본형인 double형보다 크게 할당한다. 두 번째는 해커들의 사상공간 감시를 막기 위해서 사상 공간에 실제 값이 저장되는 위치 이외에도 무작위 선택 방법으로 사상공간의 다른 위치에 대해서도 허위 값에 대한 조작을 수행하는 허위 코드를 삽입시킨다.

4. 자바 코드 어포스캐이팅

일반적으로 자바 컴파일러[4]는 사람이 작성한 원시 코드를 일정한 패턴으로 유지한 코드군 형태로 생성한다. 따라서 역컴파일러를 이용하면 클래스 파일에 저장되어 있는 상수 풀을 참조하여 컴파일러에 의해 생성된 코드를 완전하게 분석할 수 있다. 역컴파일을 막기 위해 사용하는 일반적인 방법은 코드자체에 대한 변화를 가하지 않기 때문에 2절에 살펴본 에이전트 공격유형 중 코드명세를 이용한 공격을 막을 수 없다. 본 논문에서 제안하는 코드 어포스캐이팅 알고리즘은 <그림3>과 같이 자바 가상기계 상에서 처리되는 기계어 수준에 대한 코드 어포스캐이팅을 수행한다. 여기에 적용되는 전략은 다음 3가지이다.



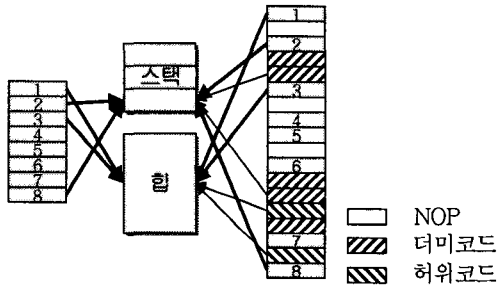
<그림 3> 코드 어포스캐이팅 알고리즘

첫 번째 전략은 코드의 임의의 위치들에 NOP명령을 삽입한다. 이것은 일정한 코드패턴 검색을 어렵게 하고 코드 길이를 변화시키는 효과가 있다.

두 번째 전략으로는 더미코드를 삽입한다. 이것은 실행의미 자체에는 변화가 없으나 스택을 기반으로 작동하는 자바 가상기계의 스택 상태를 일시적으로 변화시키는 효과가 있다. 따라서 스택 감시를 이용하는 공격을 막을 수 있다. 또한 원래의 수행코드와 더미 코드를 분리할 수 있는 것은 해당 코드의 수행 결과를 알 수 있는 실행시간이기 때문에 코드의 의미를 분석하는 것을 어렵게 만든다.

세 번째 전략은 허위코드를 삽입한다. 이 방법은 허위 비트에 대해 값을 할당하거나 읽는 방법이다. 코드의 의미를 알고 있어야하지만 삽입된 코드가 실행 흐름에 영향을 미치지 않는다.

이상의 전략들에 의해서 임의의 위치에 임의의 수의 NOP, 더미코드, 허위코드가 삽입될 경우에 <그림4>에서 처럼 실행 의미상의 코드와 허위코드 또는 더미코드 등이 동일하게 스택과 힙 영역에 접근하고 값을 변경하기 때문에 허위코드나 더미코드들과 실제코드를 분리하여 코드의 의미를 분석하는 것이 어려워지는 것을 알 수 있다.



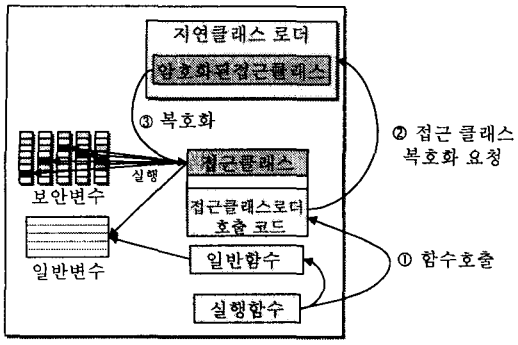
-원본 코드- -변형된 코드-

<그림 4> 원형 코드와 변형된 코드의 스택과 힙에 대한 접근

5. 자바 클래스의 지연 로딩

자바 클래스의 지연 로딩은 자바에서 사용되고 있는 클래스 로딩 메커니즘을 확장한 것이다. 클래스 지연 로딩이란 에이전트의 보안 대상 변수에 접근하는 메소드를 Inner 클래스[7] 형태로 정의하고, 접근 클래스의 코드 명세를 실행시간까지 감추기 위해서 Inner 클래스를 자바의 표준 클래스 로더를 사용해서 로딩하지 않고 실행 시간에 로딩할 수 있게 변형하고 재 정의된 클래스 로더 객체를 사용해서 실행시간에 로딩하는 메커니즘이다. 접근 클래스를 일반 클래스가 아니라 Inner 클래스로 정의하는 이유는 Inner 클래스는 정의된 속성에 대해서 클래스 멤버로서 권한을 가질 수 있기 때문이다.

<그림5>는 클래스 지연 로딩 메커니즘의 처리 순서를 나타낸다. 에이전트의 실행 함수 중에서 일반함수는 자바의 함수호출 절차를 따른다. 그러나 보안 대상 변수에 접근하는 함수는 다른 호출 방식에 따라 호출되게 한다. 보안 대상 변수에 접근하기 위해서는 접근 클래스를 로딩해서 접근 클래스형 객체를 생성하여 보안 대상 변수에 접근하게 한다. 이를 위해서 접근 클래스 호출코드 부분을 추가한다.



<그림 5> 클래스 지연 로딩

이 부분은 사용자 정의 클래스 로더에 접근 클래스의 적재를 요청하는 부분이다. 사용자 정의 클래스 로더를 이용해서 실제 접근 객체를 생성한 후 보안 대상 변수에 접근하게 한다. 사용자 정의 클래스 로더에 저장되어 있는 접근 클래스는 에이전트가 이동 후 해당 서버에 공격을 받을 수 있기 때문에 실행시간까지 안전하게 유지될 필요가 있다. 따라서 어포스캐이팅 수행 후 접근 클래스를 공개키 알고리즘인 RSA 암호화를 통해서 사용자 정의 클래스 로더에 저장하는 방식을 고려해 볼 수 있다.

클래스 지연 로딩 메커니즘에 사용되는 접근 클래스 구현 시에 주의할 사항은 호스트에 의해서 접근 클래스가 로딩되는 시점이 노출되기 때문에 보안 대상 변수에 직접적으로 값을 대입하는 형식보다는 값에 대한 정당성을 검사하는 부분이나 특수한 연산들을 거쳐서 저장하는 부분들이 추가적으로 포함되어야 한다.

6. 결론 및 향후 연구방향

본 논문에서는 신뢰할 수 없는 호스트에서 자바 이동 에이전트의 안전한 수행을 보장하기 위한 방법으로 변수분해와 어포스캐이팅 알고리즘을 이용해 에이전트의 코드 명세와 데이터 명세가 파악되지 않도록 데이터를 변형하여 수행되게 하고 분해된 변수를 접근하는 함수의 로딩을 지연시키는 시간제한 블랙박스 보안 방법을 제안하였다.

제안한 코드 어포스캐이팅 알고리즘은 자바 기계어 수준에서 NOP와 더미코드, 허위코드 삽입으로 코드 어포스캐이팅을 수행한다. NOP와 더미코드, 허위코드 삽입 횟수를 다르게 할 경우 에이전트 클래스 파일크기와 실행 속도 그리고 에이전트 클래스 암호화 시간과 에이전트 보안 수준의 정도를 결정할 수 있는 중요요소가 된다.

NOP는 아무런 일을 하지 않는 명령어이므로 삽입 횟수를 증가시켜도 더미코드와 허위코드보다 에이전트 클래스 파일 크기를 증가시키지 않는다. 또한 클래스 암호화 시간도 나머지 두 전략과 비교해 빠르나 실행 속도 측면에서는 명령어 수 증가로 감소할 것이다. 반대로 더미코드와 허위코드의 삽입 횟수를 증가시킬 경우 에이전트 클래스 파일크기는 증가되고 암호화 시간은 길어지지만 코드의 복잡성을 증가시키고 코드의 가독성을 떨어뜨리므로 에이전트 보안 수준의 정도를 높일 수 있으며 실행 속도를 증가시킬 수 있을 것이다. 따라서 NOP 명령어 삽입 횟수를 증가시키는 것보다는 더미코드와 허위코드에 대한 삽입 횟수를 증가시키는 것이 에이전트 보호를 위해서 좀 더 효율적인 전략이라 할 수 있다.

코드 어포스캐이팅 알고리즘의 안정성을 유지하기 위해서는 다양한 더미코드와 허위코드 패턴 개발이 앞으로 연구되어져야 하며 알고리즘의 안전성을 수학적으로 계산하기가 곤란하기 때문에 이를 보완하는 연구도 있어야 할 것이다. 또한 지연 로딩되는 클래스를 암호화하고 호출될 때만 동적으로 해독하고 실행할 수 있게 하는데 적절한 공개키 관리에 대한 연구도 필요하다.

참고문헌

[1] Li Gong, Java Security Architecture(JDK1.2) version 1.0, Java Soft, 1998.

[2] Fritz Hohl, Time Limited Blackbox Security: Protecting Mobile Agents From Malicious Hosts , Giovanni Vigna (Ed.): *Mobile Agents and Security*. pp 92-113. Springer-Verlag, 1998

[3] Fritz Hohl, A Model of Attacks of Malicious Hosts Against Mobile Agents , Proceedings of the ECOOP Workshop on Distributed Object Security and 4th Workshop on Mobile Object Systems: Secure Internet Mobile Computations, pp 105 - 120, INRIA, France, 1998.

[4] Jon Meyer, Troy Downing, *Java Virtural Machine*, OReilly & Associates Inc, 1997.

[5] K. Rothermel, F. Hohl, A Protocol Preventing Blackbox Tests of Mobile Agents , Tagungsband der ITG/VDE Fachtagung Kommunikation in Verteilten Systemen (KiVS'99). Springer-Verlag

[6] Tomas Sander , Christian F. Tshudin, Toward Mobile Cryptography , International Computer Science Institute, 1997.

[7] Java Inner Class, SUN Microsystems, 1998. http://java.sun.com/products/jdk/1.1/docs/guide/innerclasses/index.htm