2001년 한국정보처리학회 춘계 학술발표논문집 제8권 제1호

샌드박스의 동적 클래스 계층구조를 통한 악성코드 탐지 기법의 설계[1]

김철민* 임영환 홍만표 예홍진 조은선
*아 주 대 학 교 정 보 통 신 전 문 대 학 원
{ily, najirong, mphong, hjyeh, eschough}@ajou.ac.kr
이철원** 박현동
**한 국 전 자 통 신 연 구 소
{choelee, hdpark}@etri.re.kr

Design Mechanism for Malicious Code Detection with Sandboxes in Dynamic Class Hierarchies

Cholmin Kim, Younghwan Lim, Manpyo hong, Hong-jin Yeh, Eun-sun Cho
Dept. of Computer and Information Engineering in Ajou University
Cholwon Lee, Hyundong Park
Electronics and Telecommunications Research Institute

요 약

알려지지 않은 악성 코드의 수행을 막는 방법으로 프로그램의 실행 환경을 제한하는 '샌드박스' 기법이 많이 쓰여져 왔다. 코드의 비정상 행위를 탐지하는 이 방식은 얼마나 다양한 샌드박스들을 두는가에 따라 적용성(configurability)과 편리성(ease of use) 간의 양면성 (trade-off)을 가진다. 기존의 MAPbox는 이 두 가지를 동시에 만족 시키기 위해 프로그램의 종류별로 샌드박스를 두는 클래스별 샌드박스 적용 기법을 사용한다[3]. 그러나, 이 방법은 정적으로 클래스들이 결정되므로 적용성에 한계가 있다. 본 논문에서는 MAPbox의 개념에 동적 클래스 생성 기능을 추가함으로써 적용성을 높이는 기법을 소개하고 실제로 구현한다. MAPbox에 비해 적용성이 높아진 예로 MAPbox에서는 정상행위이지만 비정상행위로 판단되는 경우가 제안된 기법을 통해 올바르게 판단됨을 보인다.

1. 서론

샌드박스란 환경의 제한을 통해 악성 프로그램을 검출하는 방법이다[1,2]. 즉 정의된 정상 행위를 가지고 프로그램을 실행하다 권한 이상의 것을 수행하려 할 때, 그것을 악성이라고 판단하는 것이다. 샌드박스는 아직 행위가 밝혀지지 않은 프로그램이 실행되어서 악성 행위를 보인다 하더라도 실제 시스템에는 직접적인 영향을 주지 않게 된다.

샌드박스는 구현상 적용성(configurability)과 편리성(ease of use)의 양면성(trade - off)을 가진다. 적용성은 얼마나 더 정확하게 다양한 프로그램이 악성인지 아닌지를 판별하는 정도이고, 편리성은 환경을 제한하는데 사용자의 개입이 얼마나 적은가를 나타내는 정도이다. 예를 들어 에디터 와 게임처럼 종류가 다른 프로그램은 서로 다른 정상 행위를 가질 것이다. 만약 적용성을 높이려 한다면 에디터를 위한 샌드박스와 게임을 위한 샌드박스를 개별적으로 만들면 된다. 그러나 이 경우 실행되는 프로그램이 에디터에 해당되는지, 게임에 해당되는지는 사용자가 결정해야 한다. 따라서 샌드박스를 세분화 할수록 편리성은 떨어지고 적용성은 높아진다.

MAPbox는 앞서 설명한 샌드박스 적용기법을 한층 발전시킨 것으로 기존의 단일 샌드박스 시스템이 가지는 단점인 낮은 적용성을 보완한 것이다[3]. 프로그램을 에디터, 컴파일러, 게임등의 클래스로 나누고 각 클래스는 해당 정상행위만을 가진다. 이후 프로그램이 실행 될 때 사용자가 프로그램이 어떤 클래스에 속하는지 결정해야 한다. 클래스의 분류로 인해 MAPbox가 샌드박스보다 편리성이 떨어지기는 하지만, 여기서의 클래스들은 앞서 말한 것과 같이 비교적 직관적으로 알 수 있도록 구분하였으므로 그리 큰 문제가 되지 않는다. 즉 MAPbox는 적용성과 편리성을 동시에 만족 시킨다.

[1] 본 연구는 한국전자통신연구원 부설 국가보안기술연구소의 과제로 수행되었음.

```
1   procedure Main ( current_class, lst )
2   class를 통해서 프로그램 실행 시작
3   while ( 프로그램 실행 )
4           if ( lst가 가지는 권한을 벗어남 )
5               if ( class의 자식 클래스 중 해당 권한을 가진 클래스가 있음 )
6                       call Main (current_class -> child , current_class -> child.lst)
7               else if ( 사용자가 새로운 권한을 허용 )
7                       lst = lst + 새로운 권한
8                       call Make_new_class( current_class, lst )
9               else
11                      악성 프로그램으로 판정
12                      프로그램 실행 종료
13  정상 프로그램으로 판정
14  종료
```

[그림 1] 제안된 기법의 알고리즘

그러나 MAPbox의 각 클래스는 프로그램의 정상행위를 정적으로 정의하고 있고, 이것은 업데이트를 통해 새로운 기능을 가지는 프로그램이 등장하였을 경우, 정상적인 프로그램이지만 기존의 정상행위이외의 권한을 요구할 수 있으므로 문제가 된다. 즉, MAPbox는 새로운 프로그램을 악성이라 판단할 것이기 때문이다. 예를 들어 html파일을 작성할 수 있고 그것을 웹 서버에 업로드 시킬 수 있는 에디터를 가정한다면, MAPbox가 정의하고 있는 에디터 클래스는 네트워크의 엑세스 권한을 갖지 않는다. 그러나 이 새로운 에디터는 네트워크의 접근을 필요로 할 것이고 MAPbox를 통해 실행 된다면 악성 코드로 인식 된다.

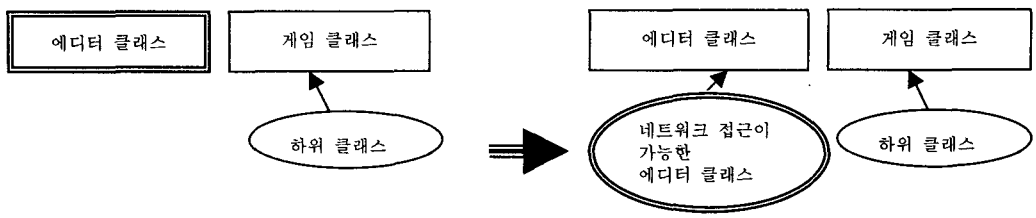
2. 제안된 기법

제안된 기법은 이러한 MAPbox의 한계를 보완하는 것이다. 즉 MAPbox가 정의하고 있는 각 클래스의 정상행위 외의 행위가 보이더라도 바로 악성코드로 판단하지 않는다. 만약 프로그램이 요구하는 새로운 권한을 사용자가 허용할 경우, 이러한 권한을 정상행위로 가지는 클래스를 동적으로 추가한다. 이를 위해 일단 MAPbox에서 정의된 클래스들을 이용하고, 새롭게 생성되는 클래스들을 기존 클래스의 하위 클래스로 만든다. 이를 통해 MAPbox의 클래스들을 루트 노드로 하는 클래스 트리의 집합이 생성된다. 제안된 기법을 이용하면 MAPbox를 이용할 때보다 정상인 프로그램을 악성이라 판단하는(false positive) 오류를 줄일 수 있다.

프로그램 실행 도중 클래스의 정의를 벗어나는 권한을 프로그램이 요구할 경우 제안된 기법에서는 일단 이 새로운 권한을 사용자에게 보여준다[그림 1, 줄번호 17]. 만약 사용자가 이 권한을 해당 프로그램에게 허용한다면 이 것은 기록되고 프로그램의 실행은 계속된다[그림 1, 줄번호 18, 19]. 한편 이 권한을 사용자가 허용하지 않는다면 프로그램은 악성으로 인식되고 새로운 클래스는 생성되지 않는다[그림 1, 줄 번호 19 ~ 21]. 그러나 프로그램 종료 시까지의 새로운 권한들을 사용자가 모두 허용한다면 이 권한들을 정상행위로 가지는 새로운 클래스가 생성되고 이것은 처음 프로그램이 실행되었던 클래스의 하위 클래스로 지정된다 [그림 1, 줄 번호 23 - 25], [그림 2]. 생성된 클래스에 해당되는 다른 프로그램이 실행될 경우 이 프로그램은 일단 생성된 클래스가 속하는 트리의 루트 노드 클래스에 의해 실행 되기 시작한다. 새로운 권한을 요구할 때마다 현재 노드의 하위 노드에 속하는 클래스중 해당 권한을 가지는 클래스로 프로그램이 이동되어 실행되고 결국 앞서 생성된 클래스에 와서 종료된다면 정상적인 프로그램으로 판단될 것이다. 만약 트리의 어떤 노드 상에서 이 프로그램이 요구하는 권한을 지원하는 하위 노드도 없고, 사용자가 이 권한을 허용하지도 않는다면 그 프로그램은 악성으로 판정

```
15  procedure Make_new_class ( current_class, lst )
16  while ( 프로그램 실행 )
17      if ( lst가 가지는 권한을 벗어남 )
18              if ( 사용자가 새로운 권한을 허용 )
19                      lst = lst + 새로운 권한
20              else
21                      악성 프로그램으로 판정
22                      테스트 종료
23  class를 호출 클래스의 자식 클래스로 만듦
24  정상 프로그램으로 판정
25  실행 종료
```

[그림 1] 제안된 기법의 알고리즘(계속)



[그림 2] 제안된 기법에 의한 클래스의 생성

된다.

앞서 MAPbox를 통해 실행 될 경우 정상적인 프로그램이지만 악성프로그램인 것으로 판단되는 경우의 예를 들었었다. 만일 제안된 기법을 통해서 이 프로그램이 실행된다면, 우선 사용자는 이 프로그램을 루트 클래스중 하나인 에디터 클래스로 실행 하려 할 것이다. MAPbox와 동일하게 이 클래스에 의한 실행도중 프로그램이 네트워크 엑세스를 요구하지만, 에디터 클래스는 이것을 허용하지 않는다. 에디터 클래스의 하위 클래스가 아직 생성되지 않았다면 제안된 기법은 사용자에게 네트워크 엑세스를 허용할지를 물어본다. 이때 사용자는 이 프로그램이 html의 업로드를 위해 네트워크 엑세스를 한다는 추가 정보를 얻을 수 있고 이를 통해 엑세스 권한을 허용한다. 이 프로그램의 실행이 끝나면 에디터 클래스의 하위 클래스로 네트워크 엑세스 기능의 에디터를 위한 클래스가 새로 생성되며 앞으로 이 클래스를 이용한 보다 정확한 악성과 정상 프로그램간의 판별이 가능할 것이다[그림 2].

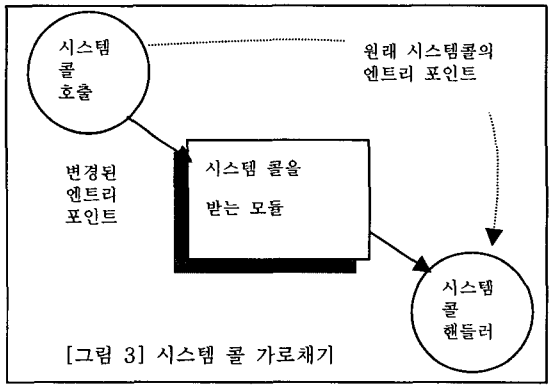
3. 구현

제안된 기법을 실제로 구현하기 위해 본 논문에서는 각 샌드박스의 클래스들을 객체 지향 언어를 이용해 정의 하였다. 한편 프로그램의 검사를 위해 MAPbox는 실제 프로그램의 실행중에 그 프로그램의 시스템 콜을 가로채는 것을 이용한 구현 이었지만 본 논문에서는 시스템 콜에 대한 프로그램의 실행시간 로그 데이터를 만든 다음 그 로그 데이터를 멤버값으로 갖는 클래스를 정의 하였다. 실행 환경은 리눅스이고 시스템 콜을 가로채어 로그를 남기는 프로그램은 C를 이용하여 리눅스 의존적으로 작성하였다[5, 6]. 한편 이러한 로그를 분석하여 실제 이 논문에서 제안하고 있는 기법을 실험하는 프로그램은 C++를 이용하되 ANSI C의 라이브러리들만을 이용하여 작성하였다[7].

먼저 시스템 콜을 가로채어 로그를 남기도록 하기위해 시스템 콜의 엔트리 포인트를 바꾸는 모듈형 프로그램을 작성하여 이것을 커널에 삽입한다[그림 3].

모듈이 삽입된 후에는 프로그램이 시스템 콜을 호출 하였을 경우 삽입된 모듈이 작업을 먼저 수행하고 이후에 실제 시스템 콜을 처리하는 부분을 수행한다. 위의 모듈은 로그 데이터를 남기는 작업만을 수행한다. 즉 엑세스한 파일의 경로와 그 파일이 갖는 권한을 텍스트 형식으로 남기게 된다[그림 4].

이러한 형식으로 로그 파일이 만들어 진 후 이 로그 파일을 멤버값으로 갖는 에플리케이션 클래스를 생성하였다. 즉 로그 파일이 가지고 있는 다른 파일에 대한 권한들을 MAPbox에서의 각 에플리케이션이 요구하는 인자와 같은 것으로 해석한다. 이를 위해 각 파일의 경로와 권한은 스트링과 불린형 변수로 저장된다. 로그 데이터로부터 인자를 읽어 들이는 데는 Add_Parameters() 메소드를 이용한다. [그림 5]. 한편 이러한 에플리케이션 클래스의 인스턴스는 실제 샌드박스 클래스의 인스턴스에 의해 실행된다. 샌드박스 클래스는 루트 노드에 해당되는 것만이 객체로 선언된다. 루트 노드가 아닌 샌드박스들은 실행도중에 동적으로 생성된다. 샌드박스 객체들은 그림 1 알고리즘의 Main에 해당하는 메소드인 Run_App()를 가진다. 각각의 샌드박스들은 하위의 샌드박스를 가르키는 Child 포인터를 가지고 있다. 오버로드된 두개의 Add_Policy() 메소드는 처음에 샌드박스가 생성될 때 전체 권한을 주기위한 것과 프로그램 실행도중 자식 클래스가 생성되어 하나의 권한을 더하는데 사용되도록 하였다. Make_New_Class() 메소드는 위의 알고리즘에서 제시한 것과 같은 것이다[그림 6].



[그림 3] 시스템 콜 가로채기

```
/root/file0          r w x
/dev/fd0             r w
/dev/tty0            r w
/etc/file1               x
```

[그림 4] 로그 데이터의 형식

4. 제안된 기법의 분석

```
class Application
{
private :
        char Parameters[MAX_PARAM_NUM][20];
        bool Permissions[MAX_PARAM_NUM][3];

public :
        void Add_Parameters();
};
```

[그림 5] 에플리케이션 클래스

```
class Sandbox
{
private :
        char TypeOfRoot[20];
        char PolicyList[MAX_POLICY_NUM][20];
        bool PermissionList[MAX_POLICY_NUM][3];
        class Sandbox *Child[MAX_CHILD_NUM];

public :
        void Add_Policy(void);
        void Add_Policy(char *Policy, bool *Permission);
        void Run_App(Application App);
        void Make_New_Class(Application App, char
SelectedParam[], bool *Permission);
};
```

[그림 6] 샌드박스 클래스

제안된 기법은 이후에 실행 하게 될 모든 프로그램을 MAPbox의 클래스들중 하나로 분류할 수 있다고 가정한다. 즉 루트 노드에 해당되는 클래스는 더 이상 생성시키지 않고, 루트 노드의 하위 노드만 생성시킨다. 또한 이 기법을 이용하는 사용자는 프로그램이 요구하는 권한의 허용 여부를 판단할 수 있을 정도의 지식을 갖춘 사용자라고 가정한다. 최종적으로 새로운 클래스를 생성하는 것은 결국 사용자의 결정이기 때문이다. 생성된 클래스는 상위 클래스가 가지는 모든 권한을 상속 받은 것과 같다. 실행 중인 프로그램이 생성된 클래스에 전달되기까지 상위 클래스가 가지는 모든 권한의 범위내에서 이미 실행 되었기 때문이다.

제안된 기법은 MAPbox에 비해 악성과 정상 프로그램을 구별할 때 적용성이 더 높다는 장점을 가진다. 반면 사용자가 프로그램이 새롭게 요구하는 권한이 정상인지 비정상인지를 판단해야 하므로 MAPbox에 비해 편리성이 떨어진다. 그러나, 만일 사용자가 MAPbox와 동일한 편리성을 가지기 위해 모든 판단을 일괄적으로 비정상으로 해버린다 해도 MAPbox와 동일한 적용성과 정확성을 보장받는다. 그리고, 일반적인 샌드박스들이 사용자의 권한 설정에 크게 의존하는 것에 비하면 이것은 그리 큰 문제가 아니다.

제안된 기법은 학습의 기능을 가지고 정상행위를 통해 악성 행위를 탐지하므로 기존의 학습형 비정상행위(anomaly) 탐지 기법과 유사점을 가진다[4]. 그러나 이러한 방법은 학습을 하더라도 대상 프로그램이 악성프로그램이다, 아니다의 판정이 더 정확해졌을 뿐이지 새롭게 추가된 권한에 대한 분석을 얻을 수는 없다. 이에 반해 제안된 기법은 미확인 프로그램이 요구하는 권한들이 새롭게 생성되는 클래스에 의해 밝혀진다는 장점이 있다.

5. 결론

이 논문에서는 MAPbox를 확장하여 정상적인 프로그램을 악성이라 판단 하던 오류를 극복 할 수 있는 기법을 살펴 보고 구현하였다. 동적인 클래스 생성을 통하여 좀더 구체적인 권한의 적용이 가능하므로 악성프로그램을 검출할 때의 적용성을 높일 수 있다. 한편 이 기법에 의해서 생성되는 클래스 트리를 재구성하는 것에 대한 연구가 계속된다면, 시간적, 공간적 효율이 더욱 향상될 것이다. 마지막으로 이 기법을 이용하면 미확인 프로그램의 권한 특성이 밝혀지므로 이를 이용한 차후 연구가 가능할 것이다.

참고 문헌

[1] Scott Oaks, "JAVA Security," O'REILLY, 1999.

[2] Ian Goldberg, David Wagner, Randi Thomas and Eric A. Brewer, "A Secure Environment for Untrusted Helper Applications," In *Proceedings of the 1996 USENIX Security Symposium*, 1996.

[3] Anurag Acharya and Mandar Raje, "MAPbox : Using Parametererized Behavior Classes to Confine Aplications," In *Computer Science Technical Report TRCS99-15*, Department of Computer Science University of California, Santa Barbara, 1999.

[4] Roland Biischkes, Mark Borning and Dogan Kesdogan, "Transaction-based Anomaly Detection," In *Proceedings of the Workshop on Intrusion Detection and Network Monitoring*, April 9-12, 1999.

[5] W. Richard Stevens, "Advanced Programming in the UNIX Environment", Addison Wesley 1998.

[6] M. Beck, H Bohme, M Dziadzka, U Kunitz, R Magnus, D Verworner, "LINUX KERNEL INTERNALS", Addison Wesley, 1999

[7] Stephen Prata, "C++ Primer Plus", the White Group, 1995.