악성 이동 코드와 대응 기법

충북대학교 조은선*

1. 서 론

대표적인 악성 코드인 바이러스는 1980년대에 접어들어 플로피 디스크 내의 다른 파일에 기생, 전파되어왔다. 그러나 1990년대 중반 이후에는 기존의 전파 경로를 탈피하여 인터넷 등 네트워크를 주된 경로로 하여 빠른 속도로 확산될 수 있게 되었다. 이는 악성 행위를 하는 코드가 목표 시스템으로 직접 이동하여 파괴 등 허용되지 않은 작업을 하는 것을 의미한다. 이 때 이동되는 악성 코드를 악성 이동 코드(mobile malicious code)라고 부른다[1].

따라서, 넓은 의미로 악성 이동 코드는 최근 등장한 웜(worm)들을 포함하여 인터넷을 통해 이동한 후 피해를 입히는 각종 악성 코드들 모두를 포함하게 된다. 최근에 피해를 입힌 악성 이동 코드들이 전달되고 수행되는 것에 대해 개괄적으로 살펴보면 다음과 같다.

전달 악성 이동 코드는 메일, 공유 네트워크를 통해 유포되거나 웹 페이지를 통해 전달되어 왔다. 또 웹 서버나 웹브라우저, SQL 서버, 채팅 프로그램, P2P 등의 취약점을 통해 전파되기도 한다[2,3,4]. 최근에는 악성 메일과 취약성을 통한 침투 등 두 가지 이상의 방법을 동시에 이용하거나 결합하여 사용함으로써 단순하지 않은 경로로 이동되기 시작하였다[2,3].

수행 웹이나 메일 내에 동적 컨텐츠 (dynamic content) 형태로 포함되어 있는 자바 애플릿 코드나 자바스크립트, VB스크립트 등 스크립트 코드들은 수행 여부를 사용자에게 묻지 않고 곧바로 수행된다. 만일 이들 파일이 메일에 첨부되어 있다면 사용자가 파일을 열어봄과 동시에 수행된다.

MS Word, Powerpoint ShockWave 등의 문서 파일이나 멀티미디어 파일들은 각종 이벤트(파일 열기, 저장하기, 변경하기 등)의 발생 시점에서 수행될 유용한 작업을 프로그램 해 놓을 수 있다. 이러한 코드를 매크로라 부르며 여기에 악성 행위가 지시되는 경우 악성 매크로가 된다. 악성 매크로가 포함된 파일이 첨부 형태 대신 웹문서의 컨텐츠의 일부로 삽입되어 이동된다면, 사용자는 독립된 파일로 인식하지 못한 상태로 열람하게 된다. 따라서 악성 매크로에 대한 사전 검사 없이 파일을 열어보게 되는 것이므로 다소 위험하다.

행위 악성 이동 코드가 행하는 악성 행위들은 시스템 파괴, 자가 복제 및 파일 감염, 다른 시스템에 자동 배포, 정보 유출, 취약성 수집 등과 차후의 악성 행위를 모색하는 트로이 목마 프로그램을 설치하는 것 등으로 다양하다. 특히 사전 공격의 도움을 받아 보다 효율적이고 치밀한 방식으로 유포되거나 정보 유출을 하기도 하며, 역으로 공격에 도움을 주는 취약성 자료 등을 수집하거나 직접 공격에 가담하는 경우가 잦아졌다[5]. 그리고, 운영체제 위에서 수행되는 과거의 악성 코드들과는 달리, 인터넷 서비스를 위해 제공되는 가상 머신들 위에서 수행되는 것들이 많아지고 있는 것도 특징이다.

본 논문은 현재까지 알려진 악성 이동 코드들에 대하여 종류 별로 소개하고 이에 따른 대응 방안에 대해 고찰한다.

2. 악성 이동 코드의 종류

2.1 자바 애플릿

자바 애플릿(Java applet)은 HTML로 구성된 웹 페이지의 <APPLET>이라는 태그에 프로그램 파일

* 종신회원

의 이름을 명시함으로써 다운로드 되어 수행된다. 일반 자바 프로그램들과 달리 init(), start(), stop() 등의 메소드들이 애플릿의 생명 주기에 맞추어 수행되게되는데, 악성 자바 애플릿은 이러한 init()이나 start() 등의 코드 내에서 악성 행위를 수행하는 새로운 쓰레드를 만들게 된다[6].

그림 1은 전형적인 악성 자바 애플릿의 하나로 소개되어 온 NoisyApplet.java의 init(), start() 메소드 등의 일부이다. init()에서 생성된 객체 bark는 오디오 자료를 서버에서 가져오는 코드이다. start()에서는 새로운 쓰레드인 noisethread 를 생성, 가동시킨다. 여기서는 run() 메소드에 명시되어 있는데 해당 오디오를 무한 반복하는 것을 의미한다. "//... "으로 나타내어진 부분은 사용자에게 평범한 자바 애플릿으로 보이도록 화면상에 그림이나 글 등을 표시하는 코드이다.

```
public void init() {
    //...
       bark = getAudioClip(getCodeBase(),
       back.au  );
}

public void start() {
      //...
      noisethread = newThread(this);
      noisethread.start(0);
}
public void run() {
        if (bark != null) bark.loop();
}
```

그림 1 NoisyApplet.java의 일부[7]

악성 자바 애플릿은 크게 악의적인(malicious) 애플릿과 공격적인(attacking) 애플릿의 두 가지로 나뉜다[6]. 악의적인 애플릿은 위의 NoisyApplet.java와 같이 소리 등으로 사용자를 괴롭히거나 대상 컴퓨터를 서비스 불능상태로 만드는 일, 사용자나 시스템 개인 자료를 유출해가는 일 등을 하는 것이다. 악의적인 애플릿의 대부분은 LaDue[7]에 의해 연구되어 공개되었고, 자바 가상 머신에 이에 대응하는 방어 기능을 신속히 추가함으로써 현재는 악용될 소지는 없다.

공격적인 애플릿은 자바 언어 자체나 수행 환경 등의 취약성을 알아내어 시스템 자체를 공격하는 애플릿이다. 따라서 자바 수행 환경이 되는 소프트웨어와 시스템을 완벽하게 오류 없이 구성하지 않는 한 늘 공격당할 가능성이 있는 것이므로 주의가 요망된다. 현재 알려진 공격들은 자바의 타입 시스템이나 접근 제한 구조, 클래스 로더, 검증자(verifier)의 취약성과 자바 가상 머신을 탑재한 웹브라우저의 결함을 통해 접근하는 것들이 있다.

2.2 스크립트 언어

최근에는 Visual Basic Script(이하, VB스크립트), 자바스크립트(JavaScript) 등의 스크립트 언어로 작성된 코드가 컨텐트에 내포되어 함께 전송되어 오는 것이 보편화되어 있다. 대부분의 간단한 웹 페이지는 사용자 인터페이스 부분을 위해 애플릿보다 스크립트 코드를 더 많이 사용하고 있다. 이러한 스크립트 코드들은 브라우저에 의해 직접 수행되어 그 결과가 보여지게 된다는 단순성을 가진다. 자바스크립트와 VB스크립트는 각각 Netscape 사와 Micro soft 사에서 제작되었다. 자바스크립트는 Netscape와 Internet Explorer 모두에서 수행 가능하며 VB스크립트는 그 구조상의 이유로 Netscape에서 지원되지 않는다.

악성 행위를 하는 스크립트 코드의 일반적인 특성은 다음과 같다.

-웹브라우저에 포함된 가상 머신에 의해 수행되므로 사용자가 코드를 포함한 웹 페이지를 열람하기만 해도 사용자 모르게 수행이 된다. 메일 클라이언트 역시 웹 페이지 형식을 해독하여 보여주는 기능이 있으므로 메일에 포함된 경우에도 웹브라우저에서와 동일한 위험이 존재한다. 파일 형태로 메일에 첨부되어 온 경우에는 사용자가 열어 수행시킴으로써 악성 행위가 수행된다.

-특정 이벤트(클릭, 마우스 이동)의 핸들러 내에 악성 코드를 삽입하여 해당 이벤트가 발생되기 전 까지는 사용자가 전송된 페이지를 의심하지 않게 한다.

-사용되는 악성 행위는 스크립트나 열람 도구의 종류에 따라 몇 가지 시스템 객체를 사용하는 것이 일반적이다.

· VB스크립트 : 궁극적으로는 File 등 시스템 객체들에 접근하고 파괴한다. 웹 대신 메일을 통해 배포되는 웜(worm) 형태로 동작하는 경우가 더 흔한데, 확산을 위해 MS Windows의 메일 클라이언트 객체 Outlook.Application의 속성 객체인 CreateItem, Attachments, Copy, AddressLists, AddressEntries 등을 주로 사용한다. Microsoft의 웹/메일 도구들로 배포될 것을 가정하여 ActiveX 등 Microsoft 전용

컴포넌트를 사용하는 경우도 있다[1].

· 자바스크립트 : 웹브라우저 자체를 나타내는 window 객체나 그의 속성인 document, location 등의 시스템 객체와 각 속성들에 접근하거나 이들을 변경시킨다. 악성 행위로는 사용자의 시스템을 열람/파괴하는 것 외에도 다른 URL에서 사용했던 기밀 정보를 유출시키는 것이 포함된다.

-최근들어 Spida[3], Exploit.Messenger[4] 등 취약성을 이용한 전파에도 스크립트 코드들이 사용되기 시작했다.

그림 2의 예는 사용자의 컴퓨터에 다량의 윈도우를 생성하여 서비스 거부 공격을 유발하는 간단한 자바스크립트 코드이다.

```
<html>
<script language=JavaScript >
<!
        for (loop=0; loop <10000; loop++)
                window.open(about:<b><h1>
                Kill!);
-- > </script>
</html>
```

그림 2 서비스 거부공격을 유발할 수 있는 단순한 자바스크립트

2.3 악성 매크로

Microsoft의 Office 제품군이나 Shockwave 등의 멀티미디어 도구들은 사용자에게 데이터를 보여주는 동안 각종 이벤트 발생 시 각기 다른 매크로 코드를 수행시킨다. 일례로 Microsoft Excel에서는 열기/저장하기를 비롯 새 워크시트가 생성될 때, 현재 워크시트가 변경될 때, 윈도우 크기가 변경되거나 (비)/활성화 될 때 등의 각 이벤트들에 대한 핸들러 형식으로 매크로 코드를 정의할 수 있다. 그림 3의 예는 Microsoft Excel 내에서 제공되고 있는 매크로 관련 편집 창이다.

그간 알려진 보안 문제는 악성 행위를 하는 매크로가 포함된 파일을 다운로드 받아 열어볼 때 주로 발생하여 왔다. 그러나 근래에는 웹 페이지에 이러한 파일들이 웹 페이지에 삽입된 형태로 전달 될 수 있으므로 악성 매크로에 대한 보다 깊은 주의가 요구된다. 사용되는 대응 방법으로는 악성 스크립트 코드와 마찬가지로 매크로가 있는 경우 경고창을 띄우거나 매크로의 수행을 불가능하도록 설정하는 방법이 있다[1].

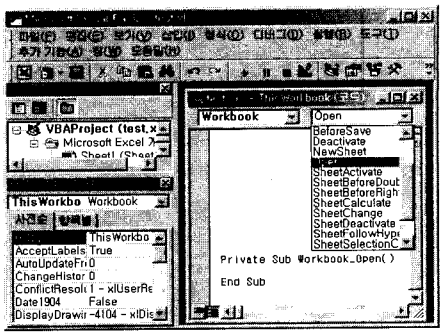

그림 3 Microsoft Excel에서 띄운 매크로 편집 창

2.4 수행 가능한 이진 코드

코드의 인터넷 이동성을 고려하게 되기 전의 악성 코드들은 대부분 수행 가능한 이진 코드 형태를 가진 바이러스들이었다. 이들의 악성 행위는 가상 머신을 사용하는 자바 및 스크립트 언어 코드와 비교할 때 사전 분석이 복잡하고, 주로 다운로드에 의해 전파되었으므로 이동성 자체에 대한 주목을 상대적으로 적게 받아 왔다. 대신, 트로이목마나 전통적인 바이러스에 관한 해법으로 접근하거나[1], 취약성을 통해 배포되는 경우의 피해사례가 많다는 점에서 침입과 관련하여 다루어지고 있다[5].

또한, 웹에 링크 되어 있는 코드 파일이 다운로드 되거나 메일에 첨부되어 이동되는 경우에는 사용자에 의한 명시적인 수행 개시 과정을 거치게 되어 수행 여부가 검사 및 사용자의 판단 후 결정될 수 있다. 단, ActiveX는 예외적으로 웹 페이지 내에 내포되어 있거나 매크로등에 포함되어 암묵적으로 수행될 수 있다. 현재는 웹브라우저에서 ActiveX의 배포자에 대한 인증과 사용자에게 수행 가부를 묻는, 정책에 기반하지 않은 단순한 접근 제어가 지원되고 있다.

3. 대응 방안 기법

3.1 사전 검사

메일을 통해 이동되는 코드들에 대해서는 서버에서 선별하여 받아들이는 방법이 널리 사용되고 있다. 일례로 sendmail 프로그램을 사용하는 경우 procmail을 사용함으로써 사용자가 sanitizer에 정한 일정한 규칙에 따라 미리 메일을 필터링 할 수 있게 된다[8]. 또 악성 첨부 파일에 보다 잘 대응하기 위해 최신 안티 바이러스 도구들을 연동시켜 사용할 수 있

도록 하는 메일 필터 도구들도 나와 있다[9,10,11]. 이러한 제품 중에는 안티 바이러스 제품 제조사에서 직접 제작되어 첨부 파일 형태 이외의 악성 코드도 발견/제거가 가능하도록 하는 경우도 있다[10]. 이러한 방식들은 대체로 송신처와 제목 등의 헤더 정보와 파일 이름, 그리고 기존의 안티 바이러스 제품에서 제공되는 악성 코드 대응책 등에 의존하게 된다.

현재의 안티 바이러스 제품은 알려진 악성 코드들의 고유 바이너리 패턴(시그니쳐)을 모아 데이터베이스로 가지고 있다. 만일 검사하고자 하는 코드가 기존의 악성 코드의 시그니쳐와 동일한 패턴을 포함하고 있으면 악성 코드로 판단하게 된다. 이 방식은 검사 속도가 빠르다는 장점을 가지고 있으나, 알려지지 않은 악성 코드에 대한 대응이 미흡하다는 단점을 가진다. 그 보완책으로 검사 속도에 큰 영향을 미치지 않는 한도에서 시그니쳐에 와일드 카드 등을 넣음으로써 알려지지 않은 악성 코드를 판별하도록 하는 방법이 있다[12].

#	BYTE SEQUENCE	ASSOCIATED BEHAVIOR
1.	B8 ?? 4C CD 21	Terminate program (permutation 1)
2.	B4 4C CD 21	Terminate program (permutation 2)
3.	B4 4C B0 ?? CD 21	Terminate program (permutation 3)
4.	B0 ?? B4 4C CD 21 0	Terminate program (permutation 4)
100.	B8 02 3D BA ?? ?? CD 21	Open file (permutation 1)
101.	BA ?? ?? B8 02 3D CD 21 0	Open file (permutation 2)

그림 4 와일드 카드 문자 '?'을 포함한 악성 코드 패턴[12]

반면 학계에서는 검사 속도에 관한 고려를 다소 보류하면서, 알려지지 않은 악성 코드에 대해서 대응할 수 있도록 예상되는 코드의 행위를 사전에 분석하는 것에 대한 연구가 이루어져 왔다[13,14,15,16]. 이 중에서 자바 바이트 코드 검증기(bytecode verifier)는 코드 로드 및 수행 속도의 저하에도 불구하고 실제 널리 사용되고 있다[6,16].

3.2 사전 검사의 보완

정적인 검사로 인하여 코드의 로드와 수행이 느려지는 것을 보완하기 위한 다양한 시도들이 있어 왔다.

첫째, 코드를 정적으로 스캔하되 판별을 수행 중으로 유보하는 기법이 있다. 의심스러운 부분에는 검사용 코드를 삽입하는 것이 있다[17]. 예를 들어 시스템 자원을 접근하는 부분 앞에, 정책 참고 및 제어를 하는 코드를 미리 삽입해 둠으로써, 수행 시 권한이 없는 행위인 경우 중단시킬 수 있도록 하고 있다. 이 방법은 동적 검사 기법에 비해서도 수행 오버헤드를 감소시킨다는 장점을 가진다. 특히 운영체제에 의한 동적 검사의 경우, 코드 수행(user) 모드와 검사를 위한 감시자(system) 모드간의 잦은 전환에 따른 오버헤드를 겪게 되는데, 여기서는 불필요해진다. 이미 이 방법을 이용하여 자바 애플릿에 대해 서버에서 일률적으로 검사 코드를 삽입하는 상업용 제품이 나와 있기도 하다[18].

둘째, 대상 호스트에서 적절한 정책을 수립하는 속도를 높이기 위해 정책 수립에 필요한, 코드에 관한 자료들을 코드와 함께 첨부되는 방법에 대해서도 연구된바 있다[19, 20, 21]. 이는 수행 중인 코드를 필요에 따라 이동시키는 이동 에이젼트(mobile agent)에서 수행 안전성(safety)을 보장하기 위해 사용되었다. 그러나 코드를 제공하는 측과 받는 측이 추가 자료에 대한 사전 협약이 필요하고 관련 미들웨어가 미리 배포되어 있는 것을 전제로 하므로 쉽게 실용화될 수 있는 방안을 함께 연구하는 것이 필요하다.

3.3 동적 검사

미리 코드를 스캔할 필요 없이 우선 수행시킨 후 별도로 감시 시스템이 프로그램의 행동을 모니터 하는 것도 가능하다. 자바의 가상 머신의 Security Manager나 AccessController와 같은 감시 시스템은 사용자가 미리 정한 정책을 기반으로 시스템 자원 접근에 관해 허용할 것인지를 판별하게 된다. 가상 머신 없이 운영체제에서 직접 수행되는 코드들을 제어하기 위해서는 운영체제 자체를 보안 운영체제(Secure OS)로 만들거나[22], 운영체제의 시스템 접근 부분을 가로채서(hooking) 원하는 보안 정책을 반영하기도 한다[23]. 가로채기의 위치는 시스템 호출과 관련된 부분이 일반적이다. 특히, Microsoft Windows 계열의 운영체제의 경우에는 COM 객체(Component Object Model)의 서비스를 받는 부분에서 가로채기를 수행할 수 있는 도구를 발표하기도 하였으며[24], 자바의 경우는 사용자가 클래스를 로드하는 로더에 보안 기능을 추가하여 새로 정의 할 수 있도록 한다[25].

또, 이동 코드를 대상 호스트가 아닌 다른 기계에서 수행시킨 후에 결과만을 열람하는 방식도 제안되어 있다[26]. 방화벽 밖의 물리적으로 독립된 공간에 존재하는 프록시 서버에서 이동 코드를 수행시킴으

로써 호스트의 안전을 보장받을 수 있다는 장점이 있으나, 프록시 서버와 호스트 간의 통신 오버헤드나 안전성에 대해 완전한 해답이 나오지 않은 상태이다.

3.4 정책 수립 모델

앞서 소개된 기법들은 해당 도구 제작/보급자 또는 사용자가 적절한 정책을 수립한다는 것을 전제로 한다. 예를 들어, 안티 바이러스 제품의 정책은 '기존의 악성 코드 시그니쳐 데이터베이스에 들어있는 패턴이 나타나면 악성 코드로 간주한다' 는 비교적 간단한 정책이다. 메일 필터링의 경우에는 연동되는 안티 바이러스 제품의 정책과 더불어 사용자가 직접 특정 파일 이름, 내용, 송신처, 제목 등이 일정 조건을 만족하는 지에 대한 정책을 세우게 된다. 코드 스캔 후 삽입 기법이나 동적인 감시 시스템에서의 정책 수립은 대부분 사용자의 부담이다. 따라서 사용자의 적절한 정책 수립을 유도 할 수 있는 우수한 정책 정의 모델 도입이 선행되어야 한다[27].

자바 보안 모델에서 기본적으로 제시하는 주체-행위의 모델은 단편적인 행위를 보고 판단할 수 있도록 하고 있다. 따라서, '파일 읽고 나서 네트워크 접속하기' 등 행위들의 연속을 정책 정의에 사용할 수가 없다. 이의 보완책이 될 수 있는 행위에 대한 모델 기법으로 Petri-Net, FSM (Finite State Machine) 등을 사용하는 것이 연구되고 있다[19,27].

주체에 관해서도 코드의 URL이나 첨부된 서명만으로 단순하게 인식하기 보다, 호출 스택을 분석하여 정의할 수 있도록 하는 것이 바람직하다. 자바의 경우 스택의 모든 주체가 권한을 가지고 있을 때 만 수행 가능한 것이 기본적이며, stack-inspection을 통해 보다 최근의 호출자의 권한을 기준으로 하는 정책 적용이 가능하다[6]. CORBA는 직전 호출자의 권한에 따르는 것이 기본 정책이나, delegation을 통해 연속된 호출에 따른 권한 위임을 자유로이 편집할 수 있도록 하므로 보다 일반적인 방식으로 주체를 결정할 수 있다[28].

하지만, 널리 사용되는 정책들은 주로 극히 단순한 방법들이다. ActiveX나 일반적인 Microsoft 제품군의 수행은 앞서 밝혔듯이 미리 수립된 정책이 없이 수행 직전 사용자에게 가부를 묻도록 하고 있다. 또 웹브라우저에서 자바스크립트를 수행 불가능하게 설정해 놓음으로써 피해를 막기도 한다.

3.5 침입과 관련된 배포에 대한 대응책

최근에는 메일이나 웹 등의 정상적인 경로 외에 불법 침입을 통해 악성 이동 코드가 수행되는 경우의 피해가 커졌다. 특히, Microsoft의 인터넷 익스플로러나 ftp 서버, TCP Wrapper 등 인터넷 이용을 위한 도구들과 보안 도구들은 침입에 필요한 기능 및 권한을 가지게되는 경우가 많으므로, 이들 프로그램의 취약성은 좋은 표적이 되고 있다. 이에 따라 각 제조자 측에서는 개선된 제품이나, 제품을 보완하는 패치(patch)를 끊임없이 발표하고 있다. 이러한 방식의 단점은, 알려지지 않은 잠재적인 취약성에 대해 방비할 수 없다는 것과 피해의 책임을 제품을 운용하는 사용자가 지게 된다는 것을 들 수 있다. 이를 극복하기 위하여 제품에 대한 취약성 검증을 미리 철저하게 하고 배포하는 기법들도 제안되고 있으나 각기 한계가 있으며 아직 널리 실용화 된 것은 아니다[19,29].

4. 결 론

인터넷의 발달에 따라 코드의 이동 수행이 보편화되면서 악성 이동 코드로 인한 피해 역시 갈수록 심각해지고 있다. 본 고에서는 악성 이동 코드의 특성과 이에 대한 대응 방안들을 소개하였다. 현재는 아직 모든 것을 해결할 수 있는 월등한 방안이 나온 것은 아니며, 한계를 극복하기 위해 각 상황을 고려하여 여러 가지 방법들을 복합적으로 적용하거나 다양한 접근을 시도하고 있다. 특히 기존의 컴퓨터 바이러스에 적용해오던 전통적인 보안 기법들을 발전시키는 작업 뿐 아니라 각종 컴퓨터 분야의 많은 이론과 기술들을 적용하고자 하고 있다. 그리고, 침입과 연동되어 발생하는 악성 이동 코드들에 의한 피해가 커짐에 따라 침입 탐지 등 보안의 다른 분야에서 제시되는 대응 방안과도 더불어 고려되어야 할 것으로 본다.

참고문헌

[1] R. Grimes, "Malicious Mobile Code," O'Reilly, 2001.

[2] 안철수 연구소, "바이러스 정보 보기-Win32-Nimda," Sep. 2001. http://home.ahnlab.com/smart2u/virus_detail_866.html

[3] 안철수 연구소, "바이러스 정보 보기 -JS/Spida",

May 2002. http://home.ahnlab.com/smart2u/virus_detail_981.html

[4] 안철수 연구소, "바이러스 정보 보기-JS/Exploit.Messenger", Feb. 2002. http://home.ahnlab.com/smart2u/virus_detail_927.html

[5] David Dittrich, "The 'stacheldraht' distributed denial of service attack tool," Dec. 1999. http://staff.washington.edu/dittrich/misc/stacheldraht.analysis

[6] G. McGraw and E. Felton, "Securing Java", Wiley, 1999

[7] M. LaDue, "A Collection of Increasingly Hostile Applets," 1996 http://www.cigital.com/ hostile-applets/

[8] J.. Hardin, "Enhancing E-Mail Security With Procmailthe E-mail Sanitizer," 2002 http://www.impsec.org/email-tools/procmail-security.html

[9] PLDaniels, "Infex-Unix Mail Server Scanner," 2001, http://www.pldaniels.com/inflex

[10] Trend Micro, "InterScan AppletTrap: Eliminating Malicious Mobile Code," white paper, 1999.

[11] Xamime, 2002, "Xamime : Content control - policy enforcement," 2002, "www.xamime.com"

[12] Symantec, "Understanding heuristics-symantec's bloodhound technology," Symantec White Paper Series Volume XXXIV, 2000.

[13] B. Chess, "Improving Computer Security using Extended Static Checking," in proc. of the IEEE Symposium on Security and Privacy, 2002.

[14] R. W. Lo, K. N. Levitt, R.A. Olsson, "MCF: a Malicious Code Filter," Computers & Security, 1995, Vol.14, No.6[outlook] F. Fernandeez, "Heuristic Engines," Virus Bulletin Conference Sep. 2001.

[15] M. Schultz et al. "Data Mining Method for Detection of New Malicious Executables," in proc. of the IEEE symposium on Security and Privacy, May 2001.

[16] X. Leroy, "Bytecode Verification on Java Smart Card," Software Practice and Experience, 2002, to appear.

[17] D. Evans, "Flexible Policy-Directed Code Safety," in proc. of the IEEE Symposium on Security and Privacy, 1999.

[18] Trend Micro, "Computer Network malicious code scanner," USA patent, Sep. 1999.

[19] R. Sekar, et. al, "Model-Carrying Code (MCC): A New Paradigm for Mobile-Code Security," in proc. of New Security Paradigms Workshop (NSPW'01), Sep. 2001.

[20] E.-S. Cho et. al "SKETHIC : Secure Kernel Extension against Trojan Horses with Information-carrying Codes," in proc. of the 6th ACISP Jul 2001.

[21] G.C. Necula and P. Lee, Safe Kernel Extensions Without Run-Time Checking, in the Proc. of the Second Symposium on Operating Systems Design and Implementation (OSDI96), 1996.

[22] T. Mitchem and R. Lu and R. OBrien, Using Kernel Hypervisors to Secure Applications, in the Proc. of the Annual Computer Security Application Conference (ACSAC97), 1997

[23] C. Ko, G. Fink, K. Levitt, "Automated Detection of Vulnerabilities in Privileged Programs by Execution Monitoring," Proc. of the 10th Annual Computer Security Applications Conference, Dec. 1994.

[24] Hunt, Galen, "Detours: Binary Interception of Win32 Functions," in proc. of the 3rd USENIX Windows NT Symposium, Jul. 1999.

[25] S. Oaks, Java Security, OReilly, 1998.

[26] D. Malkhi et. al. "Secure Execution of Java Applets using a Remote Playground," in proc. of the IEEE Symposium on Security and Privacy, May 1998.

[27] F. Schneider, Enforceable Security Policies, Techbnical Report, TR98-1664, Dept of Computer Science, Cornell University, 1998.

[28] B. Blakley, "Corba Security," Addison Wesley, 2000.

[29] J. Viega et. al, "ITS4: A static vulnerability

scanner for C and C++ code," in proc. of the 16th Annual Computer Security Applications Conference (ACSAC'00) Dec. 2000.

조 은 선

1991 서울대학교 계산통계학과 계산학 학사
1993 서울대학교 계산통계학과 전산학 석사
1998 서울대학교 전산과학과 박사
1999~2000 한국과학기술원 연구원
2000~2002 아주대학교 정보통신전문대학원 조교수 대우
2002~현재 충북대학교 전기전자 및 컴퓨터학부 조교수
E-mail : eschough@chungbuk.ac.kr