액티브 네트워크의 보안 취약성에 관한 연구

양진석*, 장범환*, 김현구*, 한영주*, 정태명**
*성균관대학교 컴퓨터공학과
e-mail:*{jsyang, bhchang, hkkim, yjhan}@rtlab.skku.ac.kr
**tmchung@ece.skku.ac.kr

A Study on Security Vulnerability of Active Network

Jin-Seok Yang*, Beom-Hwan Chang*, Hyoun-Ku Kim*,
Young-Ju Han* and Tai M. Chung**

Dept. of Computer Engineering, SungKyunKwan University

요 약

액티브 네트워크는 중간 노드에 "처리" 능력을 부여함으로써 새로운 서비스의 빠른 전개가 가능하도록 하고 네트워크 기반 구조에 유연성을 부여할 수 있는 새로운 접근 방법이다. 중간 노드에서 프로그램의 실행은 장점도 있지만 성능과 보안에 있어서 단점을 수반하고 있으며 이에 대한 연구가 활발히 진행되고 있다. 액티브 네트워크 보안은 액티브 네트워크 구성요소 즉, 노드운영체제, 실행환경 등의 보안과 공격의 탐지 및 대응 그리고 기반 구조의 보안에 대한 연구는 활발히 진행되고 있지만 존재하는 취약성에 대한 연구는 전무한 상태이다. 본 논문은 액티브 네트워크에 암시적으로 존재할 수 있는 취약성을 기존의 취약성과 연관지어 분석한다.

1. 서론

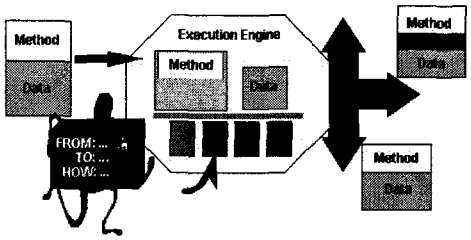
액티브 네트워크는 동적으로 새로운 서비스를 빠르게 제공하기 위해 중간 노드에 처리 능력을 부여한 네트워크 기술의 새로운 접근 방법이다. 중간 노드는 새로운 구성 요소—노드운영체제, 실행환경, 액티브 코드—의 추가로 사용자의 프로그램을 실행할 수 있다[1, 2, 3]. 그러나, 새로운 구성 요소의 추가는 기존의 취약성과 더불어 잠재적으로 새로운 취약성을 발생시킨다.

액티브 네트워크 구성 요소에서 발생하는 취약성은 능동적인 코드 전파에 따라 파급효과는 전체 네트워크로 확대될 수도 있다. 이러한 점에 있어서 액티브 네트워크 구성 요소가 갖는 취약성은 위험의 정도가 낮은 취약성이라 할지라도 보안을 신중히 고려해야할 필요가 있다.

본 논문에서는 2장에서는 액티브 네트워크 개념에 대해서 기술하고 3장에서는 액티브 네트워크 보안 취약성 분석을 위한 접근 방법을 기술하며 4장에서는 3장에서의 접근 방법을 기반으로 하여 액티브 네트워크 구성 요소에 대한 예외 상황과 취약성을 조사 분석할 것이다.

2. 관련 연구

사용자의 요구가 다양해지고 그에 따른 서비스 또한 다양하게 발전되었지만, 이에 비해 컴퓨팅 환경의 발전은 상당히 더디다. 이는 기존의 네트워크 기반 구조가 수동적이고 유연하지 못하기 때문이다[10]. 이러한 이유로 액티브 네트워크의 개념이 나오게 되었다.



[그림 1] 액티브 네트워크의 개념

액티브 네트워크는 중간 노드를 실행 가능하게 만들어 현재 중간노드들의 "저장-전달"의 기능에서 "저장-처리-전달"의 기능을 갖게 하며 이로 인해 동적인 구조를 갖게 한다[1, 2, 3].

[그림 1]은 액티브 네트워크의 개념을 나타낸다. 액티브 노드는 들어오는 패킷을 수신하여 실행 여부를 결정하고 실행 시에는 실행환경으로 액티브 코드를 전달하여 실행한다. 액티브 노드는 실행 결과에 따라 그대로 보내거나 새로운 코드를 가진 패킷을 생성하여 다음 노드로 전달한다. 중간 노드에서 이러한 처리 과정은 좀 더 유연하고 동적인 구조를 만들어 준다.

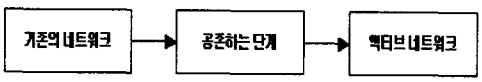
이러한 동적인 구조는 코드 실행 방식에 따라 분리방식, 캡슐방식, 혼합방식으로 나뉜다. 분리 방식은 패킷에 프로그램 참조 코드를 저장하여 노드에서 해당 코드를 실행시키는 방법으로 현재의 패킷 형식을 최대한 유지하거나 최소한의 변형만으로 액티브 네트워킹을 가능하게 하도록 하는 방식이고, 캡슐 방식은 전송되는 패킷에 프로그램과 데이터를 동시에 적재하여 보내게 하는 방식이다. 혼합 방식은 두 가지 방식을 혼합한 형태로써 규모가 큰 프로그램은 노드에 적재하고 규모가 작은 프로그램은 캡슐형태로 패킷에 포함시키는 방식이다.

노드운영체제는 노드의 통신, 메모리, 노드에서 진행하는 각종 패킷의 흐름 사이에 자원을 관리하는 등의 역할을 수행한다. 노드운영체제의 원활한 수행을 위해 도메인, 쓰레드 풀, 메모리 풀, 채널, 파일에 대한 추상적 개념을 정의하였다[1, 2].

실행환경은 액티브 어플리케이션을 관리 및 지원하기 위한 프로그래밍 모델을 정의한다. 액티브 네트워크 백본(ABone)에서 현재 운용중인 실행환경은 ASP(Active Signaling Protocol), ANTS(Active Node Transfer System), PLAN(Packet Language for Active Network)이다[4, 7, 9]. 현재 연구중인 실행환경은 실행환경 자체를 보호하기 위한 기술들을 제안하고 있지만 구현에 대한 부분은 거의 이루어지지 않고 있다[3].

3. 접근 방법

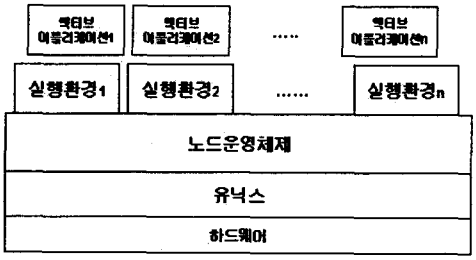
현재 액티브 네트워크 기술은 연구에서 개발로 전환하는 단계이고 기반 구조(infrastructure)는 [그림 2]에서 보는 바와 같이 액티브 네트워크 환경과 기존의 네트워크 환경이 공존하는 단계이다.



[그림 2] 액티브 네트워크로의 진화 과정

현재 구현된 액티브 노드는 [그림 3]에서 보는 바와 같이 PC(Personal Computer)기반의 액티브 노드로써 하위 레이어(Layer)에 전통적인 운영체제(유닉스)가 위치하고 차례대로 노드운영체제, 실행환경, 액티브 어플리케이션 위치하는 형태이다[1, 8].

액티브 노드가 전통적인 운영체제를 구성요소로 포함하고 있다는 것은 기존의 운영체제에 대한 취약성을 포함하고 있는 것과 같다.



[그림 3] 액티브 노드 구조

대표적인 액티브 네트워크 프로젝트에서 사용되는 전통적인 운영체제는 [표 1]에서 보는 바와 같다.

[표 1] 프로젝트별 전통적인 운영체제

프로젝트	전통적인 운영체제
ANTS	Linux
Smartpacket	FreeBSD
SENCOMM	FreeBSD 또는 Linux
JANE	Linux
FAIN	Solaris
CANES	Solaris 또는 Linux
Darwin	NetBSD 또는 FreeBSD
Liquid	Linux

운영체제뿐만 아니라 실행환경도 기존의 기술인 자바, Caml 등의 프로그래밍 언어를 이용하여 구현되었다. 액티브 노드의 구성 요소는 여러 부분에 있어서 기존 기술과 연동하여 연구 개발되고 있다[4, 5, 8]. 액티브 네트워크에서 사용하는 기존의 기술이 취약성을 내포하고 있다면 이를 사용한 액티브 네트워크 구성요소도 암시적으로 취약성을 내포하고 있다.

상기 기술한 내용을 미루어 볼 때 액티브 네트워크 취약성 분석은 기존 취약성에 대한 조사·분석이 선행되고 이를 기반으로 확장되어야한다. 이에 본 논문에서는 기존의 취약성 조사·분석을 토대로 액티브 네트워크 구성요소에 영향을 미칠 수 있는

취약성 인자를 추출하여 액티브 네트워크 구성요소가 발생시킬 수 있는 예외 상황을 분석할 것이다.

4. 취약성

본 장에서는 액티브 네트워크 워킹 그룹에서 제안한 레이어 별로 취약성 분석을 기술하겠다.

4.1 시스템 레벨 취약성

노드운영체제는 다음과 같은 예외 상황이 있을 수 있다.

■ 도메인 간섭 예외 상황
도메인은 서로의 영역에 간섭받지 않아야 하지만 도메인간의 간섭으로 예외 상황이 발생할 수 있다.
■ 도메인 종료 예외 상황
도메인은 상위 도메인 혹은 도메인 자신이 종료시킬 수 있어야 하지만, 종료시키지 못하는 예외 상황이 발생할 수 있다.
■ 도메인 경계 예외 상황
도메인은 자신의 메모리 영역과 자신의 이름공간에서 실행되어야 하지만 할당된 영역 이외에서 실행되는 예외 상황이 발생할 수 있다.
■ 부 도메인 예외 상황
도메인은 어느 때나 부 도메인을 생성할 수 있다. 한계 설정을 무시하고 도메인을 생성하는 예외 상황을 발생할 수 있다.
■ 채널 핸들링 예외 상황
incoming/cut-through/outcoming 채널 예외 발생으로 인해 처리 흐름이 틀려질 수 있다.
■ 디바이스 핸들링 예외 상황
운영체제는 각종 디바이스 드라이버를 가지고 있는데 이 때 핸들링 예외 상황이 있을 수 있다.
■ 바인딩(Binding) 예외상황
운영체제 가상머신은 노드운영체제 바인딩을 통해서 노드운영체제 인터페이스에 접근한다. 바인딩 과정에서 예외상황이 있을 수 있다.
■ 가상머신 경계(Write-barriers) 예외 상황
노드운영체제에서 동작하는 가상 머신은 구별된 메모리 영역과 이름 공간을 가지는데 잘못된 메모리 영역과 이름공간에서 쓰여질 수 있는 예외 상황이 발생할 수 있다.
■ 실행환경 인스톨 예외 상황
노드운영체제는 다중 실행환경을 지원한다. 다른 실행환경을 인스톨할 때 예외상황이 발생할 수 있다.
■ 사용자 코드 종료 예외 상황
사용자 코드는 강제로 종료될 수 있어야한다. 이 때 종료되지 않거나 메모리 반환을 못하는 예외 상황이 발생할 수 있다.
■ 공유 메모리 예외 상황
운영 체제는 공유 메모리를 제공한다. 이 때 공유 영역에 대한 예외상황이 발생할 수 있다.
■ 액티브 어플리케이션 호출 취소 예외상황
노드운영체제는 액티브 어플리케이션 호출을 취소할 수 있다. 이 때 호출을 취소하지 못하는 예외 상황을 발생시킬 수 있다.
■ Incoming 패킷 예외 상황
노드운영체제는 도메인이 자신의 패킷이라고 결정하기 전까지 메모리에 incoming 패킷들을 받아야만 한다. 이 때 버퍼 오버플로우 등의 예외 상황이 발생할 수 있다.
■ 패킷 버퍼 예외 상황
패킷 버퍼들의 사이즈를 고정시킴으로써 버퍼 오버플로우 등의 예외 상황이 발생할 수 있다.

상기 기술한 예외 상황이외에도 많은 암시적인 예외 상황이 있을 수 있다.

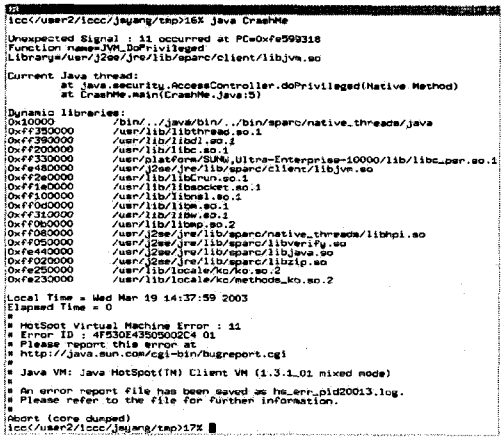
위와 같은 예외 상황은 노드운영체제의 업데이트(update), 보안패치(patch)등을 통해서 제거될 수 있다.

4.2 프로그래밍 레벨 취약성

실행환경은 실행환경을 구현한 프로그래밍 언어에 취약성에 의존하기 때문에 프로그래밍 언어가 갖고 있는 취약성을 암시적으로 갖고 있다. 프로그래밍 레벨에서는 다음과 같은 예외 상황이 있을 수 있다.

■ 실행환경 충돌 예외 상황
액티브 노드는 다중 실행환경을 지원하는데 실행중 실행환경간의 예외 상황이 있을 수 있다.
■ 코드 검증 예외 상황
실행환경에서 제공하는 코드 검증에 대한 처리를 하지 못했을 때 예외 상황이 발생할 수 있다.
■ 액티브 어플리케이션 도메인 예외 상황
다른 사용자의 어플리케이션을 접근하는 예외 상황이 발생할 수 있다.
■ 코드 요청 예외 상황
실행환경이 코드 로딩을 위해 계속적인 코드 요청을 하는 예외 상황이 있을 수 있다.
■ 예외 상황 핸들링 예외 상황
어떠한 예외 상황이 발생했을 때 그 예외 상황에 대한 조치 및 종료를 하지 못할 수 있다.

액티브 네트워크에서 프로그래밍 언어의 조건은 동적으로 프로그램이 실행된다는 면에 있어서 이동성 등의 특성을 제공해야 하지만 언어의 취약성을 극복하기 위해서는 엄격한 형 검사(strongly typed check), 무상태(stateless), 포인터 안전성(pointer safe) 등의 특성을 가지고 있어야 한다.



[그림 4] 악의적인 코드를 이용한 서비스 거부 공격

자바 프로그래밍 언어의 경우 보안은 악의적인 프로그램에 대한 안전성, 개인 정보 보호, 사용자·개발자 인증, 암호화, 감사추적 등의 보안을 제공한다[11]. 이처럼 프로그래밍 언어는 악의적인 프로그램으로부터 시스템 자체를 보호할 수 있는 보안 기술을 제공해야 한다.

4.3 어플리케이션 레벨 취약성

액티브 코드가 내포하고 있는 취약성은 코드 실행 전·후 취약성으로 나눌 수 있다. 실행 전은 코드 자체의 취약성이고, 실행 후는 액티브 어플리케이션의 취약성이다. 코드 자체의 취약성은 코드 자체를 보호하는 측면에서 액티브 어플리케이션 취약성은 자원에 대한 측면에서 고려되어질 수 있으며 다음과 같은 예외 상황이 발생할 수 있다.

■ 자원 제한 예외 상황
제한된 자원에 상관없이 실행되는 예외 상황이 있을 수 있다.

■ 증명 전달 코드 예외 상황
증명 코드의 잘못된 증명으로 인한 예외 상황이 있을 수 있다.

코드 자체 검증을 위한 방법은 핑거프린트(fingerprint) 검사, 증명 전달 코드(proof carrying code) 등의 방법이 쓰일 수 있다. 실행 후 취약성은 [그림 4]에서 보는 바와 같이 시스템 내의 제한 룰을 무시하고 자원을 소진시킬 수 있는 서비스 거부 공격과 인증되지 않은 코드의 실행으로 다른 임의의 명령어를 계속 실행시키는 보안 우회의 공격이 있을 수 있다[12].

5. 결론 및 향후 과제

본 논문에서는 액티브 노드의 구성 요소 자체에 대한 보안 취약성을 분석하였다. 분석한 취약성은 그 원인에 따라 자료 유출 및 변조삭제, 불법 자원 사용, 시스템 오류, 서비스 거부, 시스템 파괴 등의 여러 가지 치명적인 결과를 가져올 수 있다.

본 논문의 취약성 분석은 기존의 취약성을 기초로 액티브 네트워크 구성요소의 취약성 분석이다. 현재, 액티브 네트워크로의 진화 단계상 이러한 취약성 분석은 합당하나 궁극적으로 모든 기반 구조가 액티브 네트워크 환경으로 변화됐을 때의 환경을 고려하여 취약성 분석을 수행할 것이다. 또한, 그 자료를 기반으로 액티브 네트워크 환경에서 일반적인 취약성 분류에 대한 연구를 진행할 것이다.

참고문헌

[1] AN NodeOS Working Group, NodeOS Interface Specification, November, 2001.

[2] AN NodeOS Working Group, Architectural Framework for Active Networks , July, 1999.

[3] AN Security Working Group, Security Architecture for Active Nets, November, 2001.

[4] D. Wetherall et al., ANTS:A Toolkit for Building and Dynamically Deploying Network Protocols, IEEE OPENARCH' 98 Proc., San Francisco, April.1998.

[5] Danny Raz, Yuval Shavitt, Active Networks for Efficient Distributed Network Management, IEEE Communications Magazine, Vol. 38, No. 3, March 2000.

[6] Konstantinos Psounis Stanford University, Active Networks: Applications, Security, Safety, and Architectures, IEEE Communications Surveys & tutorials, Vol. 2, No. 1, First Quarter, 1999.

[7] Michael Hicks, Pankaj Kakkar, Jonathan T. Moore, Carl A. Gunter and Scott Nettles, PLAN:A Packet Language for Active Networks , ICFP, 1998.

[8] Patrick Tullmann, Mike Hibler, and Jay Lepreau, Janos : A Java-Oriented OS for Active Network Nodes, IEEE JOURNAL ON SELECTED AREAS IN COMMUNICAT IONS, Vol. 19, No. 3, March, 2001.

[9] Robert Braden, Bob Lindell, Steven Berson,Ted Faber, The ASP EE : An Active Network Execution Environment, IEEE, 2002.

[10] 이중수, 이승헌, 이명희, Arctive Network 구조:문제점 및 접근 방법, Sigcomm Review Vol. 1, No. 1, December. 2000.

[11] Scott Oaks, Java Security, O'REILLY, June, 2001.

[12] Security Focus Home Page, http://www.security focus.com/bid/3992