응용 소프트웨어 안전성 검증 시스템 설계 및 구현

소 우 영*

요 약

악성 소프트웨어로 인한 피해가 나날이 급증하면서, 컴퓨터 사용자가 보안상 안전하게 사용할 수 있는 환경이 필수적으로 요구되고 있다. 일반적인 백신 프로그램은 악성코드가 실행된 이후에 이를 탐지한다. 이러한 백신 프로그램은 알려진 악성코드에 대해서는 효율적인 결과를 보이지만, 실행 전 응용 소프트웨어에 포함되어 있는 악성코드의 검출에 대해서는 그 기능이 없거나 부족한 실정이다. 이에 본 논문에서는 응용 소프트웨어의 실행 전 악성코드의 유무를 판단하기 위해 응용소프트웨어의 안전성 검증 시스템을 제안한다. 제안하는 안전성 검증 시스템은 악성코드의 흐름 유형을 파악하여 소프트웨어가 실행되기 전 이를 탐지함으로서 악성코드로 인해 일어날 수 있는 피해를 줄일 수 있는 계기가 될 것으로 사료된다.

Design and Implementation of Safety Verification System for Application Software

Wooyoung Soh*

ABSTRACT

A safe computer environment is necessarily required for computer users, because of a damage is widely increased by a malicious software such as the warm, virus and trojan horse. A general vaccine program can detect after the malicious software intruded. This kinds of the vaccine program show good result against a malicious code which is well known, however, there is no function in the vaccine or not enough ability to detect an application software which a malicious code included. So, this paper proposes an application verification system to decide existence and nonexistence of a malicious code in the application software. The proposed application verification system with a mechanism that grasps the flow type of malicious code, can make a reduction of a damage for computer users before the application software executed.

Key words : Detection of Malicious Code, SVS(Safety Verification System)

---

* 이 논문은 2008년도 한남대학교 교비학술연구조성비 지원에 의하여 연구되었음.
** 한남대학교 컴퓨터공학과

1. 서 론

최근 보안 침해사고는 인터넷을 통한 바이러스, 웜, 트로이 목마 등의 악성코드인해 발생하나, 악성코드의 근본적인 근절이 불가능한 현실에서 이로 인한 피해를 최소화하기 위한 노력이 지속적으로 이루어지고 있고, 이에 대한 연구 또한 활발히 진행되고 있다.

대규모 네트워크를 운영하는 시스템에서는 악성코드로부터 자유로워지고자 안티바이러스나 전문 정보보호제품(Information Security System)을 함께 운영함으로써 악성코드에 대응하고 있으나, 일반 보안 전문 지식이 부족한 PC(Personal Computer) 사용자는 그에 대한 대비가 미비한 상태이다.

악의적인 목적을 가진 침입자는 위와 같은 취약점을 이용하여 PC에 침입, 개인 정보를 빼오거나, DOS(Denial of service)공격에 악용하기도 한다. 대게 개인 사용자는 V3나 알약과 같은 개인용 백신 프로그램을 이용하지만, 악성코드 제작 기술이 날로 늘어나고 그 침입형태가 매우 지능적이라는 점은 백신업체와 PC 사용자에게 상당한 노력과 비용의 문제를 야기하게 한다.

보통 백신 프로그램은 시그니처 방식과 휴리스틱 방식으로 개발되고 있다. 시그니처 방식은 알려진 악성코드 즉, 악의적인 목적을 가진 공격에 대해서는 그 효과가 좋으나 알려지지 않은 악성코드에 대해서는 효율성이 떨어지며, 반면 알려지지 않은 공격에 대한 대응책 중 하나인 휴리스틱 방식은 오용탐지가 많다는 단점이 있다.

본 논문에서는 보통 PC 사용자의 악성코드 감염 경로는 인터넷을 통한 이메일의 첨부파일이나 불법소프트웨어의 다운로드로 인해 발생한다는 점을 고려하여, 소프트웨어의 소스코드 레벨에서 악의적인 목적을 가진 악성코드의 존재여부를 파악할 수 있는 방법을 제시하고, 이를 바탕으로 한 소스코드 안전성 검증시스템 프로토타입을 설계 및 구현한다.

본 논문에서 제시한 응용 소프트웨어 안전성 검증 시스템(Safety Verification System)은 대부분의 악성코드와 일반적 시스템관련 프로그램이 C 소스코드 형태로 작성되어 있고, C 언어가 가지는 시스템 접근성이 용이하다는 점을 고려하여, C 언어를 기반으로 하며, 소프트웨어 취약성을 가장 범용적으로 만족하는 트로이 목마를 그 대상으로 한다.

본 논문에서는 실행 가능한 프로그램은 해당 프로그램의 유해 여부를 떠나 어떤 형식으로든 처리 흐름을 가지고 있다는 점을 주지하고, 해당 소프트웨어가 실행되기 전, 악성코드의 포함 여부를 탐지하기 위해 소스코드를 중간코드로 변환하고 중간코드에 포함된 정보를 기반으로 처리 흐름을 기술, 이를 악성코드와 비교함으로서 악성코드 포함 여부를 탐지한다.

2. 관련연구

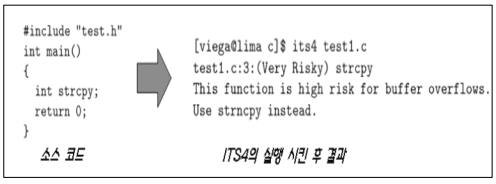
본 장에서는 기존의 소스코드 안전성 검증 도구들에 대해 비교 분석하고, 장단점을 돌출하고 이를 본 논문에서 제시하는 소스코드 안전성 검증 시스템에 적용한다.

2.1 ITS4(It's The Software, Stupid-Security Scanner)

프로그래머들이 보안에 대해서 신경쓰지 않고 개발할 수 있는 환경을 제공하기 위해 만들어진 ITS4는 C와 C++의 정적 취약점 점검 스캐너인 ITS4는 C와 C++ 소스코드의 보안 취약점을 찾아주는 커맨드 라인의 도구로 전체 소스코드를 스캔하여 잠재적인 위험성이 있는 코드를 찾아주는 도구이다[1].

ITS4의 동작은 입력으로 하나 이상의 C나 C++ 소스 파일을 입력으로 받아 각각을 토큰 스트림으로 자른 후, 이를 검사하여 'suspects' 데이터베이

스와 비교한다. 식별자 검사는 휴리스틱 방식을 사용하기 때문에 정확성에 있어 완벽하지 않으며 [2], (그림 1)과 같은 Security Neutral 식별자가 표시 될 수도 있다.



(그림 1) ITS4 테스트 결과

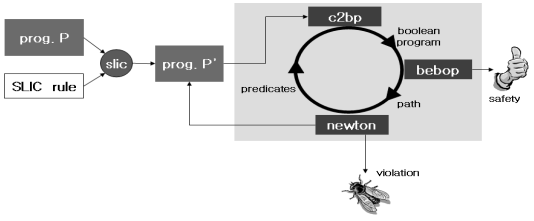
(그림 1)을 살펴보면 실제 파싱 없이 사전적으로 사용되는 모든 변수를 정확하게 결정하지 못함으로써 생기는 거짓 양성(False Positive) 문제이다.

2.2 SLAM(Specification Language for Interface Checking)

C언어로 설계된 소프트웨어를 그 대상으로 한는 SLAM은 MS사에서 자체 제작되는 소프트웨어의 정확성을 검증하기 위해 수행된 프로젝트이다. SLAM에서는 소프트웨어의 안정성 증명을 휘한 사용자의 개입이나 추상화 작업을 필요로 하지 않고 대신 C언어로 구현된 코드를 추항화 한 Boolean 프로그램을 자동으로 추출하고, 그 추상모델이 안전성을 만족하는지를 증명하는 방법이다[3, 4]. SLAM은 소프트웨어 안전성 검증을 위해 (그림 2)와 같이 다음의 세 가지 도구를 순차적으로 사용한다.

- C2BP : 소스코드를 분석 → Boolean 프로그램으로 변환
- Bebop : 변환된 boolean → 오류 상태 검출
- Newton : Bebop → 오류사항 제거

MS에서는 Windows XP를 발표하면서 Windows XP 내부에서 동작하는 몇 가지 Device driver들을 SLAM을 이용하여 안정성을 증명하였다. 앞으로도 많은 Windows 내부의 프로그램들을 대상으로 SLAM을 이용하여 안정성을 확보하기 위해 연구를 진행중이며 현재 SLAM을 공개 배포중이다[4-6].

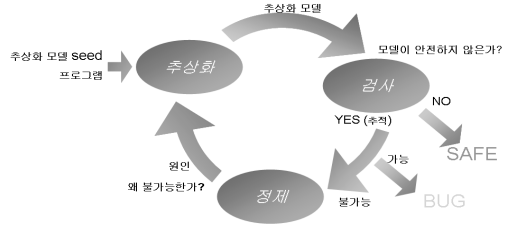


(그림 2) SLAM 개념도

2.3 BLAST(Berkely Lazy Abstraction Software Verification Tool)

C 프로그램에 대한 안전성을 검사하기 위한 검증시스템인 BLAST는(BLAST가 만들어내는 오류 레이블)에 도달할 어떠한 가능성에 대하여 검사하기 위해 '추상화-모델 검사-정제'의 루프를 수행한다. 그 추상화 모델은 프로그램 구조의 이해를 빠르게 하기위해 Predicate Abstraction을 사용하여 만들어졌다[7].

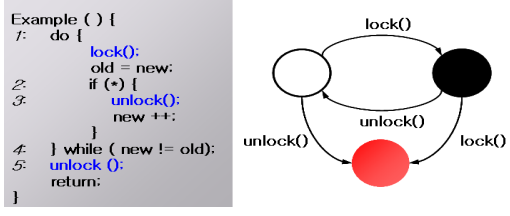
BLAST는 '추상화-검사-정제'의 루프를 수행하여 프로그램의 안전성을 검증한다. assert, lock과 unlock, setuid와 system 호출 등에서 발생할 수 있는 문제점들의 안전한 속성을 추상화 모델로 생성하여 이러한 속성들에 대해 프로그램이 안전한지를 검사한다. BLAST의 검사 모듈에 따른 루프는 (그림 3)과 같다.



(그림 3) BLAST의 개념도

BLAST에서는 장치 Device driver에 대한 안전성 여부에 가장 문제점을 두고 있어 lock과 unlock

에 따른 세마포어 자원할당 문제점에 가장 중점을 두고 있다. 다음 (그림 4)는 lock와 unlock에서 발생할 수 있는 문제점을 검사하기 위한 추상화 모델 seed 예와 그래프로, 교착 상태를 방지하기 위해 lock은 자원 할당 시에 호출되고 자원을 반납할 경우에는 unlock이 호출된다. 그러나 같은 자원에 대하여 lock이 호출된 상태에서 다시 lock이 호출되면 문제가 발생되고, 이와 반대로 unlock이 호출된 상태에서 다시 unlock이 호출되어도 같은 문제가 발생된다. 이와 같은 안전성에 대해 요구되어지는 사항을 추상화 모델 seed로 생성하여 검사 대상 소스코드와 함께 추상화 모델로 생성하여 안전성을 검사한다[7-9].



(그림 4) lock과 unlock에 대한 추상화 모델과 그래프

2.3 소스코드 안전성 검증 도구 분석 결과

본 장에서 분석한 여러 소스코드 안전성 검증 도구들의 특징을 요약하면 다음과 같다.

첫째, ITS4에서 나타낸 단순한 함수명 패턴 매칭의 가중치 기법은 아직까지 모든 검증도구에서 기본적으로 사용되어지고 있다. 그러나 이러한 단순 패턴 매칭에만 의존하는 기법은 오용탐지가 크기 때문에 실용성이 떨어진다.

둘째, SLAM과 BLAST는 소스코드 레벨에서 Device driver에 대한 안전성에 대해 중점을 두고 있기 때문에 본 연구와는 다른 방향의 검증 도구라 할 수 있다. 그러나 특정 안전성에 대한 속성들을 기술하여 이에 위반하는 사항들을 검증하는 기법은 악성 소스코드의 속성을 기술하여 악성코드를 탐지하는 기법으로 활용될 수 있을 것으로 사료된다.

3. 응용 소프트웨어의 안전성 검증 시스템 프로토타입 설계

3.1 설계 개요

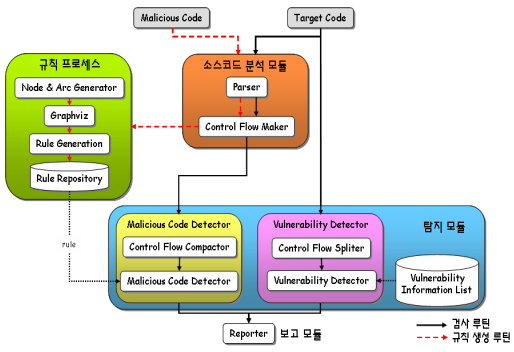
특정한 목적으로 개발된 응용프로그램의 경우 대게 기능이 정상적으로 실행되는지 또는 다른 기능과 충돌이 발생함으로서 야기되는 또 다른 문제의 유무는 해당 프로그램 검수자가 수행할 수 있는 사항이다. 그러나 방대한 개발 프로젝트일 경우에는 각각의 소스코드와 처리 흐름을 분석하지 못한다. 따라서 본 논문에서는 단순히 처리된 결과의 유효성을 기준으로 코드의 유효성을 검수를 그 목표로 한다.

또한 본 논문에서는 C언어를 기반으로 하며, 실행시간 이전에 악성코드를 탐지하는 것을 목적으로 하고, 탐지 대상으로는 트로이 목마와 포인터를 이용한 취약성인 버퍼오버플로(Buffer over flow)를 그 대상으로 한다.

3.2 설계 구성도

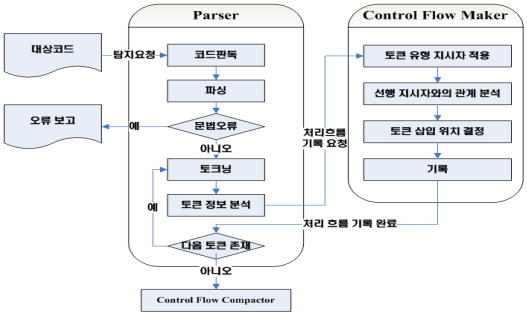
전체 시스템은 소스코드 분석 모듈과 이를 활용하여 규칙을 생성하는 규칙 프로세스, 소스코드 분석 모듈에서 생성된 정규형 언어로 표현된 중간코드를 통한 규칙과 취약성 함수 리스트를 비교하여 검증하는 검사 모듈로 구성한다.

Parser와 Control Flow Maker로 구성된 소스코드 분석 모듈은 검사 모듈에서 사용되는 푸쉬다운 오토마타 PDA 입력을 위해 정규형 언어로 표현된 중간코드를 생성하고, 규칙 프로세스는 악성코드를 판별할 수 있는 근거가 되는 규칙을 Graph Generator로 분석하여 FSA 정규형 언어로 생성하고 저장한다. 검사 모듈은 앞 단계에서 생성된 중간코드와 규칙을 이용하여 대상 코드 내에 악성코



(그림 5) 제안 시스템 전체 설계 구성도

드가 포함되어 있는지를 Malicious Code Detector를 통해 판별한다. 또한 검사 모듈에서는 중간코드와 취약성 리스트를 대조함으로써 프로그램 개발자의 부주의에 의해 생성되어진 소프트웨어 취약성을 Vulnerability Detector를 통해 탐지한다. 마지막으로 판별과 탐지에 따른 보고와 권고사항은 Reporter를 통해 출력된다.



(그림 7) Parser 모듈과 Control Flow Maker 모듈의 소스코드 처리도

4. 응용 소프트웨어 안전성 검증 시스템 프로토타입 구현

4.1 시스템 개발 환경

본 논문에서 구현하는 응용 소프트웨어 안전성 검증 시스템의 개발 환경은 다음과 같다.

〈표 1〉 시스템 개발 환경

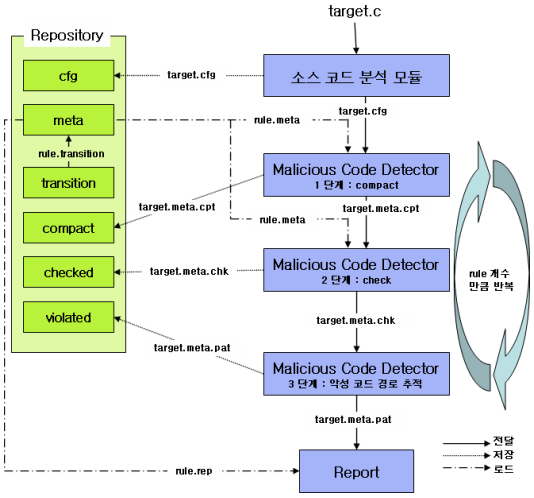
구 분	개발환경
운영체제	Red Hat Linux release 7.3, Windows XP
개발 도구	Borland JBuilder 2005
개발 언어	J2SE 1.4.1_07
참조 라이브러리	JBCL, RC

4.2 소스코드 안전성 검증 시스템 처리 흐름

본 논문에서 제시한 검증 시스템의 가장 중요한 역할을 수행하는 것은 Malicious Code Detector와 Vulnerability Detector 이다. 소프트웨어의 검증을 위한 오염인증 시스템의 처리 흐름은 두 가지로 기술할 수 있다. 첫 번째는 Malicious Code Detector에 의한 악성코드 탐지 흐름이고, 두 번째는 Vulnerability Detector에 의한 소프트웨어 취약성 탐지 흐름이다. 사용자 인터페이스에서 검사를 수행하면, 악성코드 및 취약성 탐지를 수행하고 그 결과를 보고 모듈이 종합하여 사용자에게 알려준다. (그림 7)는 Malicious Code Detector에 의한 악성코드 탐지 흐름을 기술하고 있다. Malicious Code Detector는 아래의 단계를 거쳐 악성코드를 탐지하게 된다.

단계 1 : 대상코드를 분석 → target.cfg 파일을 생성 → repository의 cfg에 저장 → Malicious Code Detector에게 전달
단계 2 : rule.meta 파일을 로드 → target.cfg를 이용하여 규칙과 연관된 코드 흐름을 추출 → target.meta.cpt 파일에 기록 → reposiory의 compact에 저장
단계 3 : target.meta.cpt 파일과 rule.meta 파일을 초기 상태, 종결 상태, 전이 형태의 내부 모델로 변환 → 흐름 탐지 → 악성코드의 흐름으로 의심이 되는 경로를 target.meta.chk 파일로 생성 → repository의 checked에 저장

단계 4 : 악성코드의 흐름으로 의심이 되는 경로를 대상코드와 비교 → target.meta.pat 파일에 기술 → repository의 violated에 저장
단계 5 : 대상코드 해당 라인표시 → rule.rep 파일을 로드 → 결과를 보고
단계 6 : 다음 규칙을 로드 → 단계 2부터 반복



(그림 7) 소스코드 안전성 검증 시스템의 처리 흐름도

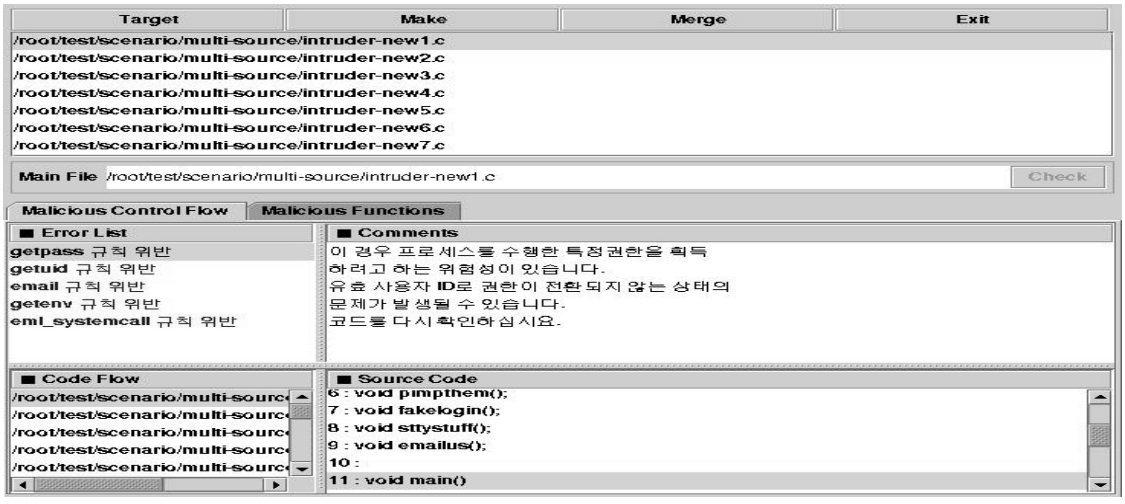
4.3 소스코드 안전성 검증 시스템 설치 및 테스트

본 논문에서 제시한 시스템은 리눅스환경에서 운영되며 기본적으로 GCC(gcc 2.96), JDK(j2sdk-1_4_1_07-fcs-linux-i586), MCD(Malicious Code Detector-mcd.tar.gz)을 운영하여 테스트한다.

악성 코드가 포함된 다중 소스코드를 대상으로 한 테스트 결과 (그림 8)과 같이 'getpass 규칙 위반', 'getuid 규칙 위반', 'email 규칙 위반', 'getenv 규칙 위반', 'emal_systemcall 규칙 위반' 목록이 표시되며, 각각의 목록을 클릭할 경우 해당되는 권고문과 악성코드를 추적하는 코드흐름이 'Comments' 창과 'Code Flow' 창에 나타난다. 'Source Code' 창에는 'intruder1.c'의 소스코드가 나타나며 'Code Flow' 창의 각 추적 라인을 클릭 시 해당되는 소스코드 라인을 따라가도록 구현하였다.

6. 결 론

악성코드가 포함되어 있는지 분석하는 방법에는 크게 두 가지가 있다. 직접 실행을 시켜 그 결



(그림 8) 악성코드를 포함한 다중 소스코드 테스트 결과 화면

과를 이용하여 분석하는 방법과 실행하지 않고 프로그램 코드나 실행을 위한 중간코드를 분석하는 방법이다. 첫 번째의 직접 실행을 통한 분석 방법은 피해 후 처리라는 단점을 가지고 있다. 이러한 단점을 해결할 수 있는 방법이 바로 후자의 중간코드나 프로그램 코드를 직접 분석하는 방법이다. 따라서 본 논문에서는 후자의 방법을 이용하여 프로그램의 안전성을 보장하고 악성코드를 탐지할 수 있는 방법론을 제시하였다. 실행 가능한 모든 프로그램은 처리 흐름을 가지고 있으며 이를 탐지하여 악성코드의 흐름과 비교 분석함으로써 본 논문에서 제시한 소스코드 안전성 검증 시스템에 적용하였다. 본 논문에서 제시한 시스템은 악성코드의 새로운 형태의 처리 흐름에 대한 해결책이 뒤따라야 할 것으로 보이며, 향 후 윈도우 NT를 대상으로 한 연구가 필요 할 것으로 사료된다.

참 고 문 헌

[1] John Viega, J. T. Bloch, Tadayoshi Kohno, Gary McGraw Reliable Software Technologies, "ITS4: A Static Vulnerability Scanner for C and C++ Code," http://www.rstcorp.com.

[2] InterPay, "I-pay product web site," http://www.ipay.com.

[3] Thomas Ball, Sriram K. Rajamani, "The SLAM Project : Debugging System Software via Static Analysis," POPL 2002, pp. 1-3, 2002.

[4] Thomas Ball, Sriram K. Rajamani, "The SLAM Toolkit," CAV 2001.

[5] Thomas Ball, Sriram K. Rajamani, "Automatically Validating Temporal Safety Properties of Interfaces," SPIN 2001, Workshop on Model Checking of Software, LNCS 2057, pp. 103-122, 2001.

[6] Thomas Ball and Sriram K. Rajamani, "Boolean Programs : A Model and Process for Software Analysis," MSR Technical Report 2000-14.

[7] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar and Gregoire Sutre, "Lazy Abstraction," In ACM SIGPLAN-SIGACT Conference on Principles of Programming Languages, pp. 58-70, 2002.

[8] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Gregoire Sutre, "Software Verification with Blast," Proceedings of the 10th SPIN Workshop on Model Checking Software (SPIN), Lecture Notes in Computer Science 2648, Springer-Verlag, pp. 235-239, 2003.

[9] See the Tutorial Introduction section in the "BLAST User's Manual," http://cad.eecs.berkeley.edu/~tah/blast/, 2004.

소 우 영

1979년 중앙대학교 전자계산학과 (이학사)

1981년 서울대학교 전자계산학과 (이학석사)

1990년 미국매릴랜드대학 전자계산학과(이학박사)

1996년 한국전자통신 초빙연구원

1991년~현재 한남대학교 컴퓨터공학과 정교수