웹사이트 구조와 사용패턴 분석을 통한 CSRF 공격 탐지★

최재영* · 이혁준* · 민병준**

요 약

CSRF(Cross Site Request Forgery)는 정상적인 자격을 부여받은 사용자의 요청을 변조하여 웹서버로 보내는 공격으로 정상적인 요청과 공격 요청을 구별하기 어렵다. CSRF 방지를 위해 비밀 토큰, 커스텀 헤더, 프록시, 정책 모델, CAPTCHA(Completely Automated Public Turing test to tell Computers and Humans Apart), 사용자 재인증 등이 연구되고 이용되고 있다. 그러나 무력화시킬 수 있는 공격 기법이 존재하며, CAPTCHA나 사용자 재인증의 남용은 웹사이트의 사용 편의성을 저하시킨다. 본 논문에서는 웹사이트의 구조와 웹사이트 사용 패턴을 분석하여 CSRF 공격을 탐지하는 방법을 제안한다. CSRF 공격 대상 후보를 선출하고 웹사이트 구조에 따른 패턴과 사이트 사용 로그를 분석한다. 정상적인 사용패턴을 추출하여, CSRF 공격을 탐지한다. 제안한 방법을 적용하여 CSRF 방지를 위해 정상 요청 시에도 사용자 개입을 요구하여 CAPTCHA를 입력하는 불편을 해소하고 이상 요청패턴을 탐지 시에만 사용자가 개입하도록 하여 CSRF를 방어하며 사용 편의성을 유지할 수 있음을 확인하였다.

Detecting CSRF through Analysis of Web Site Structure and Web Usage Patterns

Jae-Yeong Choi* · Hyuk-Jun Lee* · Byung-Jun Min**

ABSTRACT

It is difficult to identify attack requests from normal ones when those attacks are based on CSRF which enables an attacker transmit fabricated requests of a trusted user to the website. For the protection against the CSRF, there have been a lot of research efforts including secret token, custom header, proxy, policy model, CAPTCHA, and user reauthentication. There remains, however, incapacitating means and CAPTCHA and user reauthentication incur user inconvenience. In this paper, we propose a method to detect CSRF attacks by analyzing the structure of websites and the usage patterns. Potential victim candidates are selected and website usage patterns according to the structure and usage logs are analyzed. CSRF attacks can be detected by identifying normal usage patterns. Also, the proposed method does not damage users' convenience not like CAPTCHA by requiring user intervention only in case of detecting abnormal requests.

Key words : Detect CSRF, Usage Pattern, Web Security

접수일(2011년 12월 09일), 수정일(1차: 2011년 12월 14일), 게재확정일(2011년 12월 20일)

★ 본 논문은 2011년도 인천대학교 자체연구비 지원에 의한 연구결과임

* 인천대학교 컴퓨터공학과

** 인천대학교 컴퓨터공학과(교신저자)

1. 서 론

웹은 HTTP 표준에 근거하여 유저 에이전트와 웹 서버간 통신을 수행한다. HTTP는 무상태(stateless) 프로토콜로 웹서버는 클라이언트를 구분할 수 없다. 이에 클라이언트의 세션을 구별하기 위해 HTTP 요청에 문자열 형태의 세션 토큰을 사용한다. 클라이언트가 웹서버에 최초로 접근했을 때 서버는 세션 토큰을 발행하여 클라이언트와 공유하며 서버와 클라이언트가 요청시 항상 세션 토큰을 포함시켜 서로를 식별하고 세션 종료 시 토큰을 파기한다. 세션 토큰은 쿠키, URL 링크 파라미터로 자동 추가하는 URL re-writing 등을 이용한다.[1]

해커는 피해자의 세션 토큰을 가로채거나 유추하여 피해자의 자격을 취득할 수 있다. CSRF(Cross Site Request Forgery)는 웹서버로부터 정상적인 자격을 부여받은 사용자의 요청을 웹서버가 신뢰한다는 점을 이용하여 공격자가 변조된 요청을 정상적인 요청인 것처럼 위장, 삽입하여 웹서버로 보내는 공격 방법이다.[2] 세션ID나 쿠키에 의한 사용자의 자격 증명이 브라우저 내에서 발생하는 모든 요청에 자동으로 포함되기 때문에 공격자가 삽입한 요청에 자격을 부여하지 않아도 정상 요청으로 처리되어 공격이 작동하게 된다. 정상적인 요청과 공격 요청을 구별하기 어렵기 때문에 주요 요청 시 비밀 토큰, 커스텀 헤더, CAPTCHA, 사용자 재인증 등을 이용하여 CSRF를 방지하고 있다. 그러나 이러한 예방조치가 기술적으로 회피할 수 있으며, 일부는 웹사이트의 사용 편의성을 저하시키는 문제가 있다. 이에 본 논문에서는 기존 방어대책의 한계점을 기술하고 이를 해결하기 위한 방안을 제안한다.

본 논문의 구성은 다음과 같다. 2장은 CSRF의 개념과 동작과정을 기술하고, 3장에서 기존의 CSRF 방지 기법에 대한 소개와 기법의 문제점을 논한다. 4장에서 본 논문의 제안 방법인 웹사이트 구조와 사용패턴 분석을 통한 CSRF 방지 방안에 대해 기술하고 5장에서 결론을 맺는다.

2. CSRF의 개요

사용자가 웹사이트를 이용하면서 발생시키는 일반적인 요청은 주소창에 URL 입력, 하이퍼링크 클릭, 폼에 내용 기입 후 제출 시 발생한다. 그 외에 요청한 웹페이지 문서에 이미지 파일이나 서브페이지, 자바스크립트, CSS 파일, 미디어 파일 등을 참조했을 때 브라우저에 의해서 자동으로 요청이 생성된다.

웹페이지에 리소스 참조 태그를 삽입하여 사용자의 의도와 무관한 요청을 생성할 수 있다. 해커는 이미지(IMG) 태그나 IFRAME 태그를 삽입하여 사용자의 자격으로 요청을 자동 생성한다. 조금 더 지능적인 방법으로 자바스크립트를 이용하여 DOM 객체를 동적으로 생성하거나 XMLHttpRequest 객체의 setRequestHeader로 HTTP의 헤더까지 조작하여 요청을 생성할 수 있다.

CSRF는 웹사이트가 정상적으로 로그인한 사용자를 신뢰한다는 점을 이용하여, 로그인한 사용자의 인증정보를 악용하여 공격자가 변조된 HTTP 요청을 생성하며, 요청을 받은 취약한 웹어플리케이션이 해당 요청을 정당한 것으로 착각하여 공격자의 의도를 수행하게 만드는 공격이다. CSRF 공격은 다음과 같은 시나리오로 동작한다.

step1) 홍길동이 www.email.com 에 로그인(쿠키 값이 캐싱됨)하고 e-mail을 열람한다.

step2) 홍길동이 속임수, 사회공학적 방법, 숨겨진 이미지 링크를 통해 <SCRIPT src=http://www.email.com/index.html> 태그를 가진 evil.com을 방문

step3) 방문해서 받은 웹 페이지의 JavaScript가 실행, JavaScript 코드는 웹 브라우저 캐시에 저장된 로그인 인증정보를 첨부하여 인증된 사용자로 가장, 웹사이트 www.email.com 에 위조된 요청을 보낸다.

3. CSRF 방지 기법 및 문제점

본 장에서는 CSRF의 예방조치를 알아보고 각 기법의 문제점을 기술한다.

3.1 비밀 토큰

브라우저에서 자격 증명에 이용되는 기본적인 세션 토큰 이외의 별도의 예측 불가능한 비밀토큰(Nonce)을 링크나 폼에 삽입하여 서버로 들어오는 요청의 비밀토큰을 검증한다.[3][4] 공격자가 기본 자격증명 이외의 비밀토큰을 모를 경우 요청을 변조하지 못하도록 방지한다. 그러나 자바스크립트의 DOM(Document Object Model) 객체와 XmlHttpRequest를 이용하여 링크나 폼에 삽입된 비밀토큰을 읽어와 요청을 생성할 수 있다.

3.2 HTTP 리퍼러

HTTP의 리퍼러(Referer) 헤더는 요청의 이전 상태값으로 어느 링크를 통해 현재 페이지에 왔는지를 나타낸다. 리퍼러 검증을 통해 사용자가 어떤 페이지를 순차적으로 이용했는지 알 수 있지만 HTTP 헤더의 조작이 가능하며 변조여부를 알 수 없다. X-로 시작하는 커스텀 헤더는 크로스 도메인 요청 시 전송되지 않는다. 이를 이용해서 요청 변조를 검사하기도 한다.[5][6] Reflected CSRF 방지에는 이용할 수 있으나 동일 도메인에서 동작하는 Stored CSRF에서는 공격자가 커스텀 헤더의 조작이 가능하다. 또한 플래시나 실버라이트 같은 플러그인에서는 요청의 출처와 무관하게 헤더를 조작할 수 있기 때문에 커스텀 헤더를 이용한 방어를 무력화시킨다.[7]

3.3 Proxy

프락시는 보통 웹브라우저와 웹서버 사이에 데이터를 중계하는 별도의 서버를 지칭하며 웹컨텐츠 캐시, 방화벽, 보안 기능 등을 수행한다. CSRF 방지를 위해 서버와 브라우저 사이에 중계 역할을 하는 서버나 에이전트 등을 이용하여 웹트래픽을 감시하고 제어한다. 프락시는 위치에 따라 클라이언트 측 프락시(Client-side Proxy)와 서버 측 프락시(Server-side Proxy)로 분류할 수 있다.

RequestRodeo[8]는 클라이언트 측 프락시로 클라이언트에서 주고받는 요청과 응답을 모니터링하여 의심스러운 요청으로부터 쿠키, HTTP 세션 정보 등의 권한 정보를 제거하는 방안을 제시했다. 단일 웹사이트 정보를 이용하는 전통적인 형태의 웹사이트를 이용 시에는 어느 정도 효과가 있으나, 웹2.0 환경에서는 크로스 도메인간 정상적인 요청인지 공격인지 판단하기 어렵고 메쉬업 서비스 형태의 웹사이트는 민감한 정보의 제거로 서비스 이용이 불가능하게 된다.

NoForge[4]는 서버 측 프락시로 웹어플리케이션 수정 없이 HTML을 분석하여 비밀토큰을 삽입하고 클라이언트의 요청을 검증하는 방식을 제안하였다. 그러나 클라이언트에서 HTML이 동적으로 생성되는 경우 토큰을 자동으로 삽입할 수 없다. 프락시가 자동으로 비밀토큰을 삽입했듯이 자바스크립트를 통해 자동 삽입된 비밀토큰을 추출하여 가공하는 경우 방어가 힘들다.

3.4 정책 부여

[9]는 웹2.0 환경에서 클라이언트에서 다양한 서버를 상호 참조하는 요청에 대한 정책을 수립하여 요청에 대한 허용, 거부, 수정을 수행하는 모델을 제시했다. 그러나 이 연구방안은 서버와 클라이언트가 보안상 안전하다는 전제에서 서버에서 제공하는 모든 서비스에 대한 정책을 수립해야지 효과적으로 이용이 가능하다. 정책이 허용하는 범위 내에서 요청 변조가 발생 시 이를 방어하지 못하며, 클라이언트 측 정책 파일에 대한 변조의 위험이 있다. 또한 정책을 수립하지 않은 요청에 대한 방어가 어렵다.

3.5 CAPTCHA 및 재인증

CAPTCHA(Completely Automated Public Turing test to tell Computers and Humans Apart)[10]와 사용자 재인증은 사용자 데이터나 동작에 의해 피해를 일으키는 요청에 대해서 사용자가 개입하도록 하여 CSRF를 방지한다. CAPTCHA는 사람과 자동 스크립트를 구별하기 위해 OCR 프로그램이 인식하기 어려우나 인간이 쉽게 인지할 수 있는 왜곡된 이미지에 표시된 문자를 입력하도록 하는 방식이다. 사용자 재인증은 민감한 동작 요청 시 사용자가 개입하여 암호를 한 번 더 입력하게 요청하는 방식이다. 이는 주요한 페이지로의 접근 시 웹서버로부터 부여된 자격을 신뢰하지 않으며, 악성스크립트에 의해 자동으로 생성된 요청과 정상 요청을 구별해 낸다. CSRF 방지 효

과가 있으나 빈번한 사용자 개입은 웹사이트의 사용 편의성을 저하시키는 문제가 있다.

4. 제안 방안

기존 방어 기법들을 무력화 시킬 수 있는 신종 공격 기법이 발생하며, 웹어플리케이션의 논리적 허점이 존재할 수 있기 때문에 한가지 방법으로 공격을 완벽하게 차단하는 것은 불가능하다. 가능한 방어 기법들을 복합적으로 이용하여 안전성을 높여야할 것이다.

클라이언트로 보내진 세션 정보와 비밀토큰, 기타 보안 정보들은 공격자가 요청 변조를 통해 얼마든지 가공이 가능하다. 클라이언트와 서버간 공유되는 사용자 식별과 인증 정보는 보안을 위한 최소 정보가 된다. 클라이언트와 공유하지 않으면서 서버 내에서 사용자를 식별하고 인증할 수 있는 방안이 필요하다.

본장에서는 3장에서 기술한 기법들의 기술적 취약점으로, 비밀 토큰을 포함한 요청 생성, 리퍼러 헤더의 조작, 정책 부여 시 정책 허용 범위에서의 요청 변조 등의 기술적 한계점을 보완하고, CAPTCHA나 사용자 재인증 방식의 문제점인 사용 편의성을 저하시키지 않으면서 CSRF 방지 효과를 유지할 수 있는 방안을 제시한다. 서버 측에 웹사이트의 구조와 사용패턴을 분석한 데이터를 기초로 클라이언트의 웹사이트 사용 패턴을 모니터링하고 비정상적인 사용 패턴에 대응한다.

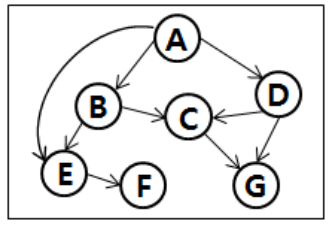
4.1 웹사이트 구조와 사용패턴 분석

CSRF 공격은 세션을 이용한 인증 메커니즘의 허점을 노린 공격으로 일반 이용자의 트래픽과 구별이 어렵다. 요청 변조의 발생 가능성이 있기 때문에 이용자의 세션을 무조건 신뢰할 수 없다.

본 논문에서는 웹사이트 구조 분석과 웹사용패턴을 분석[11][12]하여 정상적인 세션일지라도 변조 가능성이 있는 이상패턴을 탐지하여 CSRF 공격을 방지하는 방안을 제시한다.

웹사이트의 구조 분석은 웹페이지 간의 하이퍼링크를 통한 이동 가능 경로를 도출하여 웹사이트의 논리적인 구조를 파악한다. 분석결과는 (그림 1)과 같이 그래프 구조로 표현한다. 노드는 웹 페이지, 간선은 하이퍼링크를 의미한다. 정상적인 사용자의 요청은 웹사이트 구조 내에서 상호 참조에 의해 이루어진다. 웹사이트의 구조를 무시한 요청은 변조로 의심할 수 있다.

웹사용패턴은 웹서버 로그 파일을 이용하여 사용자의 웹사이트 사용 정보를 분석한다. 접속 IP와 브라우저 정보 등을 이용하여 사용자 세션을 분류하고 단일 세션, 즉 트랜잭션 별로 웹사이트를 방문한 클라이언트의 연속적인 요청 로그를 분석하여 페이지 요청 순서를 추출한다. 추출한 정보를 통해 페이지간 연관성을 찾을 수 있다.



(그림 1) 웹사이트 구조 그래프

본 논문에서는 사용자의 정상적인 웹사이트 이용 패턴을 벗어나는 이상 요청 패턴을 찾는데 목적이 있기 때문에 웹로그에서 분석에 불필요한 이미지, 멀티미디어, 라이브러리 소스 등의 요청 정보를 제거한다. 또한 해커의 주 공격 대상이 되는, 사용자 데이터나 동작에 의해 피해를 일으킬 수 있는 페이지를 선별한다. 이를 타겟 페이지라고 칭한다. 웹로그 분석을 통해 추출한 정보 중에서 타겟 페이지를 호출하는 트랜잭션만 추출하여 페이지 이동 경로를 분석한다. 타겟 페이지에 도달하기 위해 논리적으로 반드시 거쳐야할 페이지를 종속 페이지, 타겟 페이지 도달과 무관하거나 필수가 아닌 페이지를 보조 페이지라 명명한다. 사용자 인증이 필요없는 단순 조회 페이지, 타겟 페이지로의 종속성이 없는 보조 페이지는 호출 횟수만 계산하여 분석을 단순화시켜 분석 오버헤드를 줄인다.

웹로그 정보를 이용한 분석 시 프록시 서버나 클라이언트의 로컬 캐시 때문에 실제 사용자가 브라우저에서 요청하였으나 웹서버에 로그가 남지 않는 경우가 발생한다. 공격 대상이 되는 페이지는 민감한 정보

조회나 조작을 수행하는 페이지로 중간 과정에서 캐시로 남아 있으면 보안 문제가 발생하는 특징이 있기 때문에 타겟 페이지나 종속 페이지는 no-cache 옵션을 사용하는 동적 페이지로 구현되었다는 전제로 로그 분석을 수행한다.

(그림 2)의 로그 분석 데이터 (a)에서 타겟 페이지가 F일 때 F에 도달하는 트랜잭션 T1, T2, T4, T5를 추출하여 트랜잭션 내에서 각 페이지가 호출되는 가중치 (b)를 도출한다.

(a) 트랜잭션

	A	B	C	D	E	F	G
√T1	1	1	0	0	1	1	0
√T2	1	0	0	0	1	1	0
T3	1	1	1	0	0	0	1
√T4	1	0	0	0	1	1	0
√T5	1	0	0	0	1	1	0
T6	1	1	1	0	0	0	1

(b) 가중치

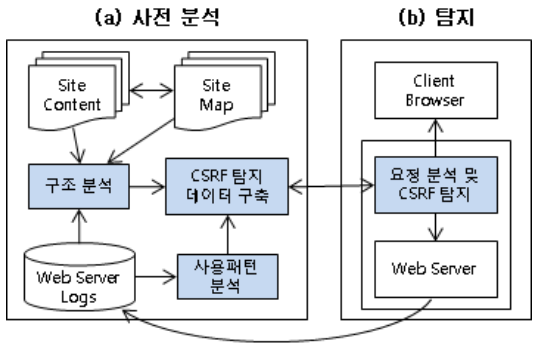
Page	Weight
A	1.0
B	0.25
E	1.0
F	1.0

(그림 2) (a) 트랜잭션 별 페이지 호출
(b) F에 도달하는 페이지의 호출 가중치

분석 결과, 페이지 A와 E는 F에 도달하는데 항상 호출되는 종속 페이지, B는 보조 페이지가 된다. F를 호출하는 트랜잭션에서 A나 E를 호출하지 않는 경우는 요청 변조로 의심하여 요청 처리를 거부하고 사용자 재인증 페이지로 강제 이동시킨다.

4.2 시스템 구성

(그림 3)은 웹사이트의 구조, 사용패턴 분석과 CSRF 탐지과정을 수행하는 제안 시스템의 구성도이다. 사전분석은 분석 작업과 분석결과 데이터 구축 작업으로 나눠진다. 구조분석은 사이트맵과 컨텐츠, 웹서버 로그를 활용한다. 페이지간 호출관계를 찾아서 웹사이트의 구조를 구축한다. 사용패턴분석은 로그파일을 분석하여 페이지 호출의 순차 패턴을 찾는다. CSRF 탐지 데이터 구축은 구조 분석과 사용패턴 분석 결과 데이터를 취합한다. 탐지는 클라이언트로부터 서버로 들어오는 요청을 수집, 분석하여 사전분석에서 구축한 패턴과의 일치 여부를 비교하여 불일치 시 CSRF 요청 변조로 판단한다.

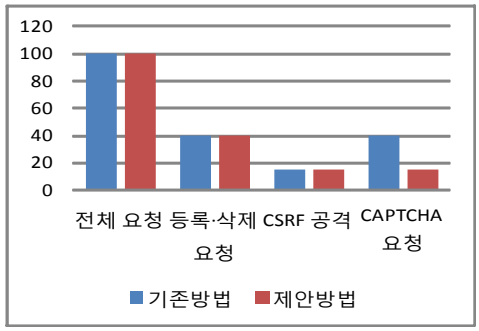


(그림 3) 제안 시스템 구성도

4.3 실험

제안한 방법의 효과를 확인하기 위해 게시판 형태의 실험 사이트를 구축하고 게시글에 CSRF 공격 코드를 삽입하였다. 게시글의 등록, 삭제에 모두 CAPTCHA를 입력하도록 하고 제안한 방법은 요청을 분석하여 CSRF 탐지 시에만 CAPTCHA를 입력하도록 하여 그 결과를 비교하였다.

(그림 4)와 같이 실험을 통해 CAPTCHA 도입은 정상적인 요청에도 입력을 요청하였으나 제안한 방법은 CSRF 공격에 대해서만 CAPTCHA 입력을 요구하여 동일한 안전성을 유지하며 사이트 이용의 편의성을 보장함을 확인하였다.



(그림 4) 실험 결과

5. 결 론

웹서버는 사용자를 식별하기 위해 브라우저에 세션 토큰을 할당한다. 세션 할당 후 동일한 브라우저에서 발생하는 모든 요청을 수용한다. CSRF 공격은 세션을 할당받아 정상적인 자격이 부여된 브라우저에서 공격자가 변조된 요청을 웹서버로 보내어 피해자의 자격으로 공격자의 의도를 수행한다.

본 논문에서는 웹사이트의 구조와 웹사이트 사용패턴을 분석하고, 실제 사이트 이용 요청과 비교하여 CSRF로 의심되는 이상 요청을 탐지하여 CSRF 공격을 방지하는 방법을 제시하였다. CSRF 공격대상이 되는 타겟 페이지를 선정하고 웹사이트 구조에 따른 패턴과 사이트 사용 로그를 분석하여 타겟 페이지로의 이동 경로를 도출하였다. 도출한 정상 사용패턴 결과와 사용자의 페이지 내비게이션을 비교하여 불일치 시 CSRF로 판단하여 사용자가 개입하도록 하였다. 실험을 통해 모든 타겟 페이지에 CAPTCHA 입력을 요구하여 발생하는 사용 편의성 저하를 최소화하면서 동일한 CSRF 방지 효과가 있음을 확인하였다.

향후 분석의 효율성과 사용패턴 비교의 정확성을 위한 알고리즘 연구가 요구된다.

참고문헌

[1] David Gourley and Brian Totty, “HTTP: The Definitive Guide”, O'Reilly Media, 2002.

[2] http://www.owasp.org/index.php/Cross-Site_Request_Forgery

[3] OWASP, CSRF Guard, http://www.owasp.org/index.php/CSRF_Guard

[4] N. Jovanovic, E. Kirda, and C. Kruegel, “Preventing Cross Site Request Forgery attacks”, In IEEE International Conference on Security and Privacy in Communication Networks (SecureComm), 2006.

[5] A. Barth, C. Jackson, and J. C. Mitchell, “Robust defenses for Cross-Site Request Forgery”, In Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS 2008), 2008.

[6] Mike Shema, "Seven Deadliest Web Application Attacks", Syngress, 2010.

[7] A. Klein. Forging, “HTTP request headers with Flash”, http://www.securityfocus.com/archive/1/441014, 2006.

[8] M. Johns and J. Winter, “RequestRodeo: Client side protection against session riding”, In In Proceedings of the OWASP Europe 2006 Conference, 2006

[9] W. Maes, T. Heyman, L. Desmet, and W. Joosen, "Browser Protection against Cross-Site Request Forgery", In Workshop on Secure Execution of Untrusted Code (SecuCode), 2009.

[10] www.captcha.net

[11] Bing Liu, "Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data", Springer, 2006.

[12] J. Srivastava, R. Cooley, M. Deshpande, P.-N. Tan, "Web usage mining: Discovery and applications of usage patterns from web data", ACM SIGKDD, 2000.

[저 자 소 개]

최 재 영 (Jae-Yeong Choi)

1999년 2월 인천대학교
전자계산학과 학사
2001년 8월 인천대학교
컴퓨터공학과 석사
2010년 3월 ~ 현재 인천대학교
컴퓨터공학과 박사과정

email : jero@incheon.ac.kr

이 혁 준 (Hyuk-Jun Lee)

2010년 3월 인천대학교
컴퓨터공학과 학사
2011년 9월 ~ 현재 인천대학교
컴퓨터공학과 석사과정

email : hjlee@incheon.ac.kr

민 병 준 (Byung-Jun Min)

1983년 2월 연세대학교
전자공학과 학사
1985년 2월 연세대학교
전자공학과 석사
1991년 12월 미국캘리포니아대학교
(UC Irvine)
전기및컴퓨터공학과 박사
1995년 ~ 현재 인천대학교
컴퓨터공학과 교수

email : bjmin@incheon.ac.kr