通信情報保護學會誌
第8卷 第1號, 1998. 3

이동 에이전트의 보호

Security in Mobile Agents

이 영 화*, 이 남 용*

요 약

이동 통신 기술의 발전과 인터넷의 급속한 확산은 기존의 클라이언트-서버 환경을 대체하는 이동 에이전트 시스템을 출현시켰다. 그러나, 이동 에이전트 시스템이 안전하지 않은 네트워크 상에서 운영됨에 따라 다양한 위협에 노출됨으로 인해 이에 대한 보호 대책 수립이 시급한 실정이다. 본 논문에서는 우선 이동 에이전트 시스템의 특징 및 구조를 살펴보고, 관련된 위협 요소를 분석하며, 이동 에이전트에 대한 보호 대책, 특히 최근 관심이 고조되고 있는 악의적인 호스트로부터 이동 에이전트를 보호하기 위한 대책을 제시한다.

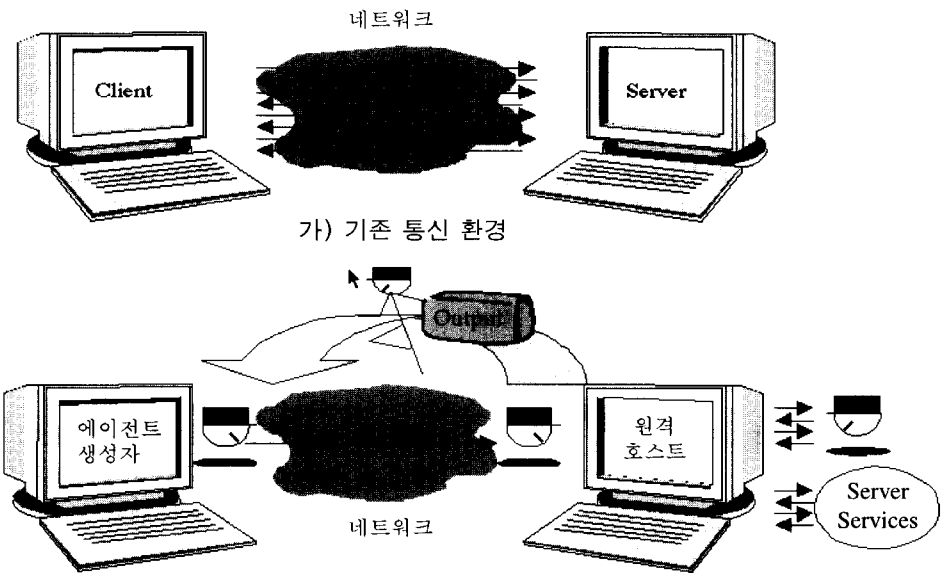
1. 서 론

에이전트 시스템이란 사용자를 대신하여 사용자가 원하는 작업을 자율적으로 해결해주는 프로그램이다. 에이전트 시스템의 개념은 1970년대 분산 인공지능 분야를 모체로 출발하였으며, 1980년대 후반에는 분산 협동 처리와 에이전트간 통신 개념이 대두되면서 새로운 분야로 독립하였고, 1990년대 중반부터는 인터넷 서비스의 폭발적인 증가, 개방형 분산 처리 기술의 급속한 발전, 이동 통신 기술의 접목 등으로 인해 미래의 통신 환경의 기반을 형성할 핵심 부분으로 발전하여 큰 주목을 받게 되었다.[3,5,16,19] 이 중 에이전트가 지능을 갖추고 인터넷과 같은 광역 통신망에서 이동하면서, 다양하며 이질적인 호스트들과 상호 작용하고, 자신의 소유자를 대신하여 정보를 수집·분석한 후, 설정된 임무를 수행하고 복귀하는 이동 에이전트 분야는 에이전트 시장 중 가장 큰 시장 점유율을 차지하며, 가장 빠른 상용화가 이루어지고 있는 분야이다. 특히 이동 에이전트는 최근 이동 컴퓨팅 기술과 연계되면서, 미래의 통신 환경의 기본 플랫폼으로 발전하고 있으며, 기존의 수동적인 통신 환경을 능동적인 작업 수행 환경으로 변경시킬 것으로 기대되고 있다.[1,2] 그러나, 이동 에이전트는 안전하지 않은 네트워크를 이동하면서 작업을 수행하는 과정에서 다양한 위협에 노출되게 되었다. 이동 에이전트의 위협은 크게 악의적인 에이전트로부터 네트워크 상에 존재하는 호스트들이 침해를 받을 위협과 악의적인 호스트에 의해 정당한 이동 에이전트가 침해, 조작

* 국방정보체계연구소

및 변경될 수 있는 위협으로 분류할 수 있다. 과거의 많은 연구에서는 악의적인 에이전트로부터 호스트들을 안전하게 보호하는 대책이 주로 연구되었으나, 최근 전자 상거래가 본격적으로 시작되면서 악의적인 호스트로부터 사용자를 대신하여 전자 상거래를 수행하는 이동 에이전트의 보호 방법에 많은 연구가 집중되고 있다.

본 논문에서는 우선 이동 에이전트의 구조에 대해 개략적으로 살펴보고 이를 토대로 이동 에이전트의 위협 영역과 위협 요소를 상세히 식별한 후, 악의적인 호스트로부터의 이동 에이전트 보호 대책을 중심으로 한 이동 에이전트 보호 대책을 소개한다.



[그림 1] 기존 통신 방식과 이동 에이전트 통신 방식의 비교

2. 이동 에이전트의 개요

이동 에이전트는 네트워크 에이전트, 또는 순회 에이전트(Itinerant Agent)라고도 불리며, 에이전트의 자율성(Autonomy), 협동성(Cooperation), 이동성(Mobility), 지능(Intelligence)의 특성 중 특히 이동성이 강조된 에이전트로서 에이전트 스스로의 판단에 의해 네트워크 상의 호스트들간을 이용하면서 사용자를 대신하여 작업을 수행한다. 이동 에이전트는 기존 통신 체계 하에서 발생하는 막대한 통신 비용의 절감, 인터넷을 통해 자유롭게 제품을 사고 파는 전자 상거래의 대두, 이동 컴퓨팅 및 분산 컴퓨팅 환경의 급속한 성장 등의 요구를 수용하는 해결책으로서 등장하게 되었다.[6, 14]

2.1 이동 에이전트의 특징

- 비동기식 컴퓨팅 수행을 통한 기존 통신 체계의 고비용 구조 해결

  기존의 RPC(Remote Procedure Calls) 환경의 네트워크에서 상호간에 데이터를 교환하기 위해서는 TCP/IP와 같은 통신 프로토콜을 이용하여 통신 주체들이 계속적으로 연결되어 있어야 한다. 그러나, 이동 에이전트 환경 에서는 [그림 1]에서 보듯이 계속적인 통신 연결이 불필요하게 된다.[6, 7, 16] 즉, 이동 에이전트를 이용하여 데이터를 교환할 때에는 서비스를 제공할 호스트와 연결하고 해당 서버로 이동 에이전트를 이동시킨 후 통신 연결을 끊어도 된다. 왜냐하면, 이동한 에이전트가 연결이 끊긴 상태에서도 서버에서 실행되어 작업을 수행하게 되며 작업을 끝낸 후 e-mail, 삐삐, 핸드폰, 팩스 등을 이용하여 사용자에게 통보할 수 있기 때문이다. 이 때 사용자가 다시 서버와 연결하여 완료된 작업 결과를 통보 받을 수 있게 된다. 이 경우 기존 통신 방식에 비해 메모리, 시간 등 많은 네트워크 자원을 절약할 수 있다.

- 이동 통신의 제약 해결

  개인 휴대용 단말(Personal Digital Assistants)와 같은 이동 디바이스는 낮은 대역폭을 제공하며, 제한된 저장 능력(예: HP 95/100/200 계열에서는 단지 2 MB의 메모리를 제공한다.)과 프로세싱 능력을 갖는다.[5] 그러나, 이동 에이전트는 이동 디바이스와 연결된 호스트의 대규모 메모리 및 고속의 프로세싱 능력을 이용하여 정보의 탐색 및 필터링 등의 작업을 수행하기 때문에 이동 디바이스가 갖는 메모리 및 프로세싱 능력의 제약을 극복할 수 있으며, 작업 수행 결과만을 요약하고, 압축 등의 방법을 이용하여 필요한 최소한의 자료만을 통신함으로써 낮은 대역폭으로 인한 통신 부하 문제를 해결할 수 있다.

- 이질적인 네트워크 환경에서 원활한 통신 서비스 제공

  이질적인 네트워크, 이기종 호스트상에서 다양한 서비스들을 원할하게 제공하는 것은 현재의 네트워크 상에서 매우 어려운 문제이다. 그러나, 이동 에이전트 시스템은 프로그램 및 데이터 교환을 위해 Script 언어를 사용함으로써, 플랫폼과 독립적인 프로그램 및 데이터 표현을 가능케 하였다. 즉, 네트워크 상의 모든 호스트들에 Script 언어를 지원하는 환경이 구축되어 있다면 이질적인 네트워크 환경에서도 원활한 의사 소통이 가능하게 된다.

- 의미에 의한 검색(Semantic Retrieval)에 기반한 원격 정보 탐색 및 필터링 지원

  현재 네트워크에 존재하는 대부분의 정보들은 구조화된 데이터베이스 형식이 아닌, 일반 문서 화일 형식으로 되어있으므로 원격 정보의 탐색 및 필터링을 위해서는 화일을 찾고, 열고, 필터링하고, 인덱스를 만드는 기능을 필요로 한다. 이동 에이전트 시스템에서는 의미에 의한 검색에 기반한 정보 검색 시스템을 이용하여, 질의를 생성하고, 이동 에이전트에 설치하여, 네트워크 상의 호스트들을 이동하면서 정보를 검색하고, 필터링한 후 그 결과를 사용자에게 전달할 수 있다.

- 복수 프로그램간 협력

  기존의 통신 환경은 단지 두 개의 프로그램만이 연결되어 작업을 수행하고 그 결과를 전송하는 단순한 구조였다. 그러

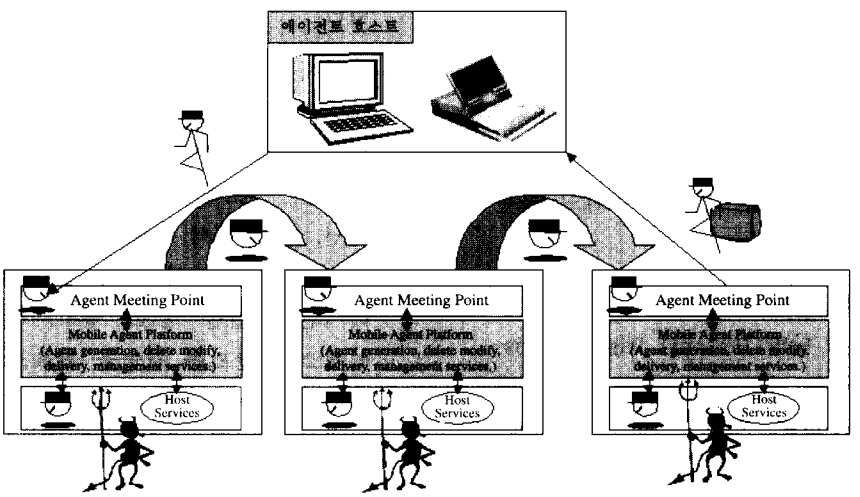
나, 이동 에이전트 시스템에서는 자신과 동일한 프로그램을 손쉽게 복제하여 분산 환경 하에서 병렬로 작업을 수행하고 그 결과를 네트워크 상에서 비교 한 후 종합하여 하나의 작업 결과를 도출할 수 있다. 즉, 분산 협동 기술을 접목하여 동일한 기능을 수행하는 여러 에이전트들이 이동하면서 서로 협력하며 작업을 수행함으로서 한 프로그램으로 할 수 없는 다양한 작업들을 처리할 수 있으며 사용자에게 신속하게 정보를 제공할 수 있다.

● 능동적인 작업 수행

기존의 통신 환경에서는 통신 주체가 네트워크 상에서 필요한 정보를 제공할 상대방을 직접 찾고, 연결하여, 정보의 제공을 요청한 후 상대방의 승낙 하에 정보를 전송받는 일련의 수동적인 과정을 수행한다. 그러나, 이동 에이전트 시스템에서는 사용자가 계속적으로 관여하지 않고 에이전트 스스로 네트워크를 이동하면서 정보를 제공할 상대방을 찾고, 필요한 정보를 획득하여 그 결과를 사용자에게 전달하는 능동적인 작업 환경을 제공한다.

● 오류 자동 복구

이동 에이전트는 스스로 학습, 판단 및 복제 및 이동할 수 있기 때문에 다수의 원격 및 개별적인 요구들을 손쉽게 조정·통제할 수 있고, 네트워크 상에서 발생하는 다양한 오류에 유연하고 능동적으로 대처할 수 있다.



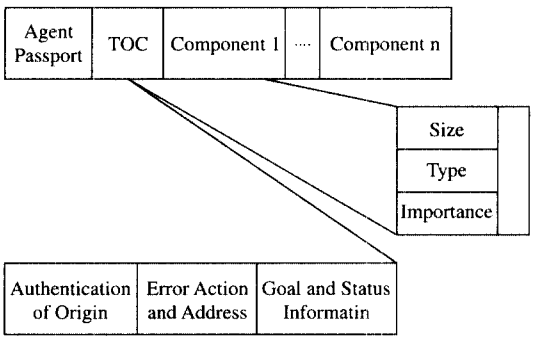
[그림 2] 이동 에이전트의 구조

2.2 이동 에이전트의 구조

이동 에이전트 시스템은 [그림 2]와 같은 구조를 갖으며, 크게 이동 에이전트, 이동 에이전트 플랫폼, 이동 에이전트 언어의 3 부분으로 구성된다.

2.2.1 이동 에이전트

이동 에이전트는 Script 언어로 작성되며 인터프리터 형식으로 수행된다. 이동 에이전트는 작업 수행을 위해 또는 다른 호스트로 이동을 위해서 이동 에이전트 플랫폼과 지속적인 대화를 수행할 수 있도록 설계되었다.



[그림 3] 이동 에이전트의 구성 요소

이동 에이전트는 [그림3]과 같이 크게 Agent Passport, TOC(Table of Contents) 및 Component로 구성된다.[5]

- Agent Passport
  Agent Passport는 이동 에이전트가 상이한 호스트간의 이동을 위해 요구되는 기본 정보들로 구성된다. Agent Passport는 크게 이동 에이전트의 생성자의 신원을 인증하는 생성자 인증 정보(Authentication of Origin), 이동 에이전트의 작업 수행 중 에러 발생 시 취해야할 행동을 제공하는 에러 행위 및 주소 정보(Error Action and Address), 그리고 이동 에이전트의 목적 및 상태 그리고 다른 에이전트와의 관계 등을 표현한 목적 및 상태 정보(Goal and Status Information)로 구성된다.

- TOC
  TOC는 이동 에이전트 구조에 대한 정보를 제공하는 부분으로 구성 요소의 크기, 유형 및 중요도 필드로 구성되어 있다. 크기 필드는 구성 요소의 크기를, 유형 필드는 구성 요소의 기능 수행을 위한 요구 사항을 간단하게 표현하며, 중요도 필드는 이동 에이전트 플랫폼상에서 수행될 해당 구성 요소의 우선 순위를 나타낸다.

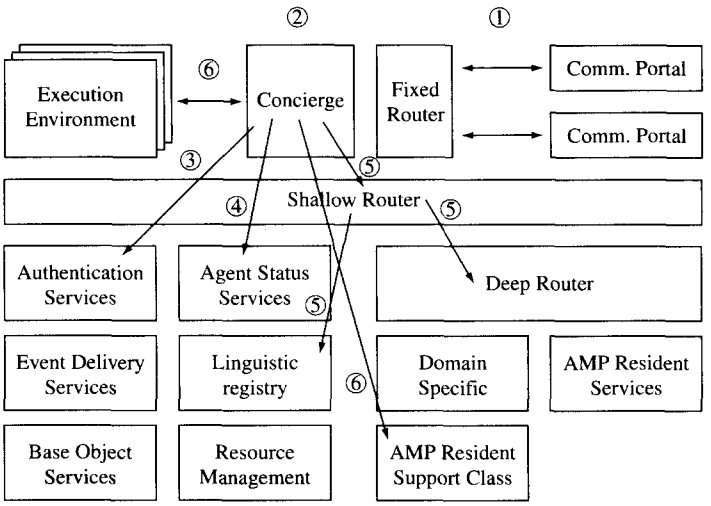
- Component
  이동 에이전트가 수행해야 할 작업을 기술한 목록이다.

2.2.2 이동 에이전트 플랫폼

이동 에이전트 플랫폼은 이동 에이전트를 생성, 관리, 해석, 수행, 종료하고, 필요시 다른 에이전트 플랫폼에 전송하는 일을 수행하는 이동 에이전트의 물리적 하부구조이다.[9] 이동 에이전트 플랫폼의 수행 기능을 세분하면 다음과 같다.

- 이동 에이전트 수행 : 이동 에이전트 언어 번역 및 수행, 메시지 전달, 에이전트 간 통신, 외부 프로그램과의 통신 및 에러 인지 및 처리 등
- 플랫폼 제어 : 이동 에이전트의 전송, 암호화 및 복호화, 권한 통제 및 관리, 서버 제공 서비스 및 자원 관리 등
- 기반 구조 제공 : 네트워크 기반 구조, Multitasking 및 Multithread 구조 제공

이 중 에이전트와 직접 연결되는 부분을 특히 AMP(Agent Meeting Point)라고 한다. 다음 [표 1]과 [그림 4]는 AMP의 구조 및 서비스 및 작업 수행 순서를 나타낸다.



[그림 4] AMP의 구조

2.2.3 이동 에이전트 언어

이동 에이전트 언어는 에이전트들이 상호 정보를 주고받기 위해 사용하는 메시지에 대한 구조 및 형식을 정의하는 언어로서 크게 에이전트 프로그램을 작성하는 프로그래밍 언어와 다양한 환경에서 적절하게 이동 에이전트의 목표, 작업, 선호도 및 단어 등을 표현하는 지식 표현 언어로 구성된다.

이동 에이전트 프로그래밍 언어는 에이전트 플랫폼에 독립적이며 다양한 언어를 지원하고 AMP의 실행 환경을 지원하여야 한다. 일반적으로 이동 에이전트 프로그래밍 언어가 갖추어야 할 특징은 다음과 같다.

- 접근 통제, 데이터 이동성 등에 객체 추상화(Object Abstraction)를 제공할 수 있는 객체 지향적인 언어이어야 한다.
- 이동 에이전트의 가장 큰 특징인 이동성을 지원할 수 있어야 한다.
- 이동 에이전트의 복제를 유연하게 지원하여야 한다.
- 이동 에이전트가 운영되는 분산 컴퓨팅 환경을 지원할 수 있어야 한다.

지식 표현 언어는 다양한 환경에서 적절하게 이동 에이전트의 목표, 작업, 선호도 및 단어 등을 표현하는 수단을 제공하는 언어로서 미 국방부의 KSE(Knowledge Sharing Effort) 하에서 연구된 에이전트간 의사소통 및 통신을 위한 언어인 KQML(Knowledge Query and Manipulation Language)이 대표적이다.

3. 이동 에이전트 침해 방법

이동 에이전트 시스템에서 발생하는 보호 문제는 크게 악의적인 에이전트로부터 네트워크 상의 호스트들이 침해를 받을 위협과 악의적인 호스트에 의한 이동 에이전트의 침해, 조작 및 변경 등의 위협으로 분류할 수 있다. 과거의 많은 연구에서는 악의적인 에이전트로부터 호스트들을 안전하게 보호하는 대책이 중점적으로 연구되었으나, 최근 전자 상거래가

[표 1] AMP의 구성요소 및 서비스

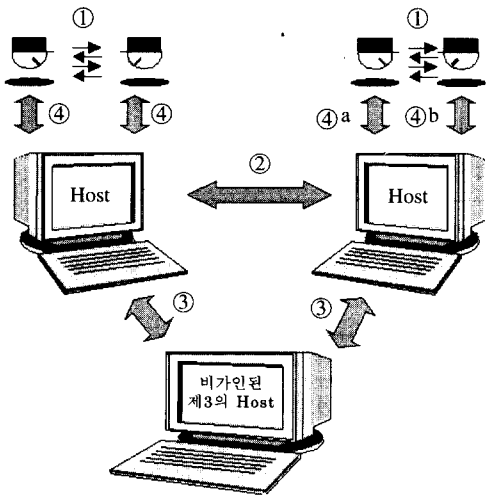
	세부 구성요소	수행 기능 및 활동
구성요소	Communications Portals	- 이동 에이전트의 도착 및 출발 관리 - AMP내에 상주 에이전트 및 서비스에 대한 메세지 라우팅 기능 - 프로토콜 관리자 지원
	Agent Concierge	- 이동 에이전트의 Passport와 TOC를 분석하여 에이전트가 요구하는 기능의 이용 자격을 분석하고 결과를 Shallow Router에 전달 - 다른 호스트 AMP로의 이동을 지원
	Shallow Router	- 에이전트와 AMP 구성 요소간에 인터페이스로서 기능을 수행 - 이동 에이전트로부터의 요구 사항을 수신하여 해당 서비스 및 자원에 할당하는 기능 담당
	Deep Router	- Shallow Router에서 처리 못한 상세하고 어려운 요구들을 처리
	Resource Manager	- AMP내의 이동 에이전트와 에이전트 자원의 등록부로서 역할 - 자원 사용의 관리자로서 기능 수행
	Agent Execution Environment	- Scripts 또는 에이전트에 대한 인터프리트 제공 - AMP의 기반 기능 및 서비스에 대한 접근 제공
	Linguistic Registry	- 에이전트 통신을 지원하는 데이터베이스 기능 제공
서비스	Authentication Services	- 도착한 이동 에이전트의 신원을 인증하는 서비스로 Agent Concierge에 의해 수행
	Base Object Services	- 객체 인터페이스의 통제, 원격 객체 호출, 객체의 정렬, 객체 기반 접근 통제 서비스, 객체 복제 및 에이전트에 대한 해석 제공
	Event Delivery Services	- AMP내의 서비스로부터 산출된 정보를 관련된 에이전트에 전달하는 기능 수행

본격적으로 시작되면서 악의적인 호스트로부터 사용자를 대신하여 전자 상거래를 담당하는 이동 에이전트를 안전하게 보호하는 방법에 많은 연구가 집중되고 있다. 본 절에서는 우선 이동 에이전트 시스템 침해 영역을 식별하고 각각에 대한 개략적인 보호 대책을 제시하며, 악의적인 호스트에 의한 이동 에이전트 침해 위협을 중점적으로 분석한다.

3.1 이동 에이전트 침해 영역

이동 에이전트는 네트워크 상에서 다른 호스트의 에이전트 또는 호스트 자원과 의사 소통할 수 있으며, 다른 호스트로 이동할 수도 있다. 의사 소통은 에이전트간 통신, 서비스의 요구 및 제공, 새로운 에이전트의 생성 및 기존 에이전트의 폐기 등으로 구성된다. 이와 같

이 이동 에이전트의 기능 수행 과정에서 여러 가지 침해 위협이 발생할 수 있다. 이동 에이전트의 침해 영역은 [그림 5]에서와 같이 크게 4가지 영역에서 분류할 수 있다.



[그림 5] 이동 에이전트의 침해 대상 영역

3.1.1 에이전트간 의사 소통시 침해 위협

두 에이전트간에서 발생하는 침해 위협으로 악의적인 에이전트에 의한 정상적인 에이전트 코드 및 데이터 영역의 조작, 거짓 식별자를 이용한 에이전트 위장(Masquerading), 속임수(Cheating) 및 서비스 부인(Denial of Service) 공격 등이 이에 해당한다. 전자 상거래를 수행할 때 위장은 거짓 식별자를 이용하여 다른 거래 당사자로 속여 거래에 참여하는 경우, 속임수는 대금을 지불하지 않고 특정한 서비스(예: 오락, 도서 구매 등)를 받는 경우, 서비스 부인은 메일 메시지 버퍼를 가득 채워 정상적인 메일 서비스를 수행하지 못하게 하는 경우에 해당한다. 이와 같은 위협 요소는 허용되지 않은 에이전트에 의한 물리적 접근을 막는 Java 등과 같은 에이전트 언어 사용, 디지털 서명 등을 이용한 에이전트의 인증 및 독립적인 서비스 계약 매카니즘의 활용 등을 통해 대처할 수 있다.

3.1.2 호스트간 의사 소통시 침해 위협

호스트간의 의사 소통 시에도 에이전트간의 침해와 마찬가지로 위장, 속임수, 서비스 부인 등의 침해 위협이 발생한다. 호스트간에 발생하는 침해 위협은 현재의 통신 환경에서 안전한 통신을 위해 사용되는 일반적인 보호 대책과 동일한 방법을 사용하여 보호할 수 있다. 즉, 일반적인 인증, 감리 증적, 상호작용 통제 매카니즘 등을 적용하여 호스트간에 발생하는 침해 문제를 해결할 수 있다.

3.1.3 호스트와 비인가된 제3의 호스트간의 의사 소통시 침해 위협

안전하지 않은 네트워크 상에서 두 개의 호스트간의 통신과 관련된 보호 문제로서 호스트간의 의사 소통 경우와 마찬가지로 일반적인 통신 보호 매카니즘을 적용하여 보호할 수 있다.

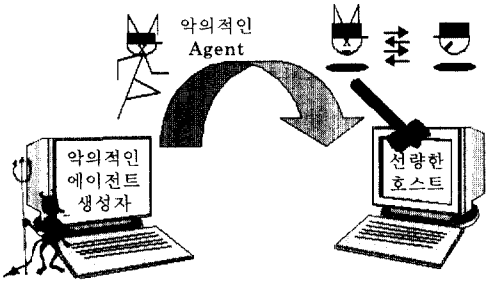
3.1.4 에이전트와 호스트간 의사 소통시 침해 위협

에이전트와 호스트간의 의사 소통시 침해 위협은 크게, 악의적인 에이전트로부터 호스트 침해 위협과 악의적인 호스트로부터 에이전트 침해 위협으로 분류할 수 있다.

- 악의적인 에이전트로부터 호스트 침해 위협

  악의적인 에이전트로부터의 침해 위협은 기본적으로 악의적인 에이전트로부터의 에이전트 침해 위협과 동일하다. 즉, 물리적 조작, 위장, 속임수 및 서비스 부인 등이 위협 요소이다. 그러나, 호스트가 에이전트의 실행을 통제할 수 있어서 필요시

언제든지 악의적인 에이전트의 실행을 중지시킬 수 있다는 점에서 에이전트간 공격과의 차이가 발생한다.



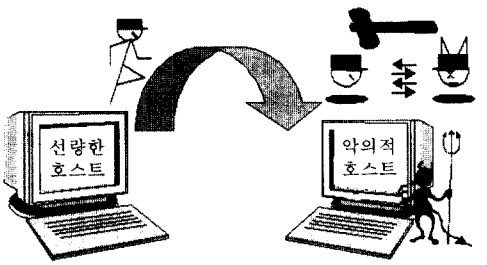
[그림 6] 악의적인 Agent로부터 Host공격

악의적인 호스트 공격에 대한 보호 방법은 다음과 같다.[11,21]

- 안전한 언어 또는 독립된 주소 공간 사용
- Agent Tcl 등에서 사용되는 암호 기법을 활용한 인증
- 책임 추적(Accounting) 및 계약 매카니즘의 사용
- 플랫폼에 대한 접근 권한 통제, 플랫폼간 상호 인증
- 계정 사용 및 실행 시간 제한 등 자원 사용 제한 매카니즘 사용
- 패킷 필터 등 운영체제 확장 매카니즘 이용

● 악의적인 호스트로부터 이동 에이전트 침해 위협

호스트가 악의적인 공격자가 되는 것으로서 호스트는 이동 에이전트가 수행하는 모든 행위를 관찰할 수 있고, 모든 코드 비트, 데이터 및 상태와 관련된 부분을 읽을 수 있으며, 이동 에이전트 코드에 대한 인터프리트를 통해 에이전트의 작업 수행 방식을 조정할 수도 있다. 악의적인 호스트로부터의 공격 유형을 세분하면 다음과 같다.



[그림 6] 악의적인 Host로부터 Agent공격

- 코드의 관찰, 조작 및 변경

악의적인 호스트는 자신의 이동 에이전트 플렛폼 상에서 이동 에이전트를 수행시킬 때 모든 코드를 볼 수 있다. 이 경우, 호스트는 코드에 대해 접근을 통해 이동 에이전트의 수행 전략, 메모리 상의 코드, 데이터의 정확한 물리적 구조 등에 대한 정보를 얻게 된다. 만일 호스트가 코드를 읽을 수 있고 동시에 코드 메모리에 접근할 수 있다면 이동 에이전트 프로그램을 수정할 수도 있다. 즉, 바이러스, 웜(Worm) 또는 트로이 목마 등을 이식함으로써 영구적으로 코드를 변경시킬 수 있고, 특정 호스트에서만 이동 에이전트코드를 변경하여 에이전트의 행동을 변경시킬 수도 있다. 특히 후자의 경우 이동 에이전트가 방문하게될 다음 호스트가 이동 에이전트 코드의 변경 여부를 탐색할 수 없기 때문에 더욱 큰 위험을 안게 된다.

- 데이터에 대한 관찰, 조작 및 변경
  비밀키, 전자 화폐 등과 같은 에이전트의 데이터 영역을 침해하는 경우로서 만일 호스트가 메모리 상의 데이터의 물리적 위치와 개별 데이터 구성 요소의 의미를 알 수 있다면 데이터를 조작할 수 있다. 예를 들어, 가장 저렴한 가격으로 꽃을 구매하는 이동 에이전트의 경우 호스트가 가격 부분의 코드를 조작하여 한 꽃집의 제시 가격을 가장 낮은 가격으로 조정한 후 나머지 꽃집 조사 대상 목록을 제거하는 경우가 이에 해당한다.
- 제어 흐름(Control Flow)의 관찰
  만일 호스트가 에이전트의 전체 코드 및 데이터를 알 수 있다면 언제라도 다음 수행 단계의 행동을 파악할 수 있다. 그러나, 전체 코드를 모르는 경우에도 제어 흐름 정보를 이용하여 이동 에이전트 상태 이상의 정보를 획득할 수 있다. 예를 들어, 실제 이동 에이전트 코드를 정확히 읽지 못한 경우에도 제어 흐름을 관찰함으로써 현재의 가격이 지금까지의 최적 가격보다 더 좋은지 또는 나쁜지에 대한 정보를 알 수 있다. 또한, if 선언문의 변경 등 제어 흐름 조작을 통해 자신의 가격을 가장 낮게 설정할 수 있다.
- 정당한 수신 호스트로 위장
  일반적으로 호스트는 수신 호스트의 신원을 검증하기 위해서 이동 에이전트를 전송한다. 이 때 악의적인 제3자가 전송 중인 이동 에이전트를 탈취 또는 복제하여 자신이 마치 올바른 수신 호스트인 것처럼 위장할 수 있다. 이 경우 이동 에이전트의 코드나 데이터를 침해하는 공격이 발생할 수 있다.
- 서비스 부인
  호스트가 이동 에이전트의 실행을 방해함으로서 에이전트의 기능을 수행하지 못하게 하는 것으로서 이동 에이전트가 호스트에 의해 실행되기 때문에 호스트는 손쉽게 에이전트의 실행을 중지시킬 수 있다.
- Black Box 시험 공격
  이동 에이전트에 임의의 자료를 계속적으로 입력한 후 출력되는 결과와의 상관관계를 분석함으로써 특별한 지식 없이도 이동 에이전트의 행위 패턴, 내부 구조 등을 분석할 수 있다.
- 동시 공격
  여러 공격을 동시에 수행하는 경우로서, 개별 공격보다 더 큰 위험을 발생시킨다. 예를 들어, 악의적인 호스트가 최대 가격뿐 아니라 제어 흐름까지 알고 있다면 해당 호스트가 지정하는 꽃집에서 최대 가격으로 꽃을 사도록 이동 에이전트를 조작할 수 있다.

4. 이동 에이전트 보호 방법

악의적인 에이전트로부터 호스트에 대한 공격은 전통적인 보호 매카니즘을 이용하여 효과적으로 보호할 수 있었다. 반면, 악의적인 호스트로부터의 이동 에이전트 공격 문제는 에이전트 개념이 전자 상거래에 본격적으로 도입되기 시작한 1990년 대 중반 이후에 제기되었다. 그러나, 에이전트의 공개적이고 이동하는 특성으로 인해 전통적 보호 매카니즘의 적용이 어렵다는 사실과 하드웨어가 부재한 상태에서 소프트웨어 단독으로 에이전트를 보호할 수 없다는 다양한 연구 결과 등으로 인해 효과적인 이동 에이전트 보호 방법의 개발에 많은 어려움이 있었다.

우선 악의적인 호스트로부터의 공격은 전통적인 보호 매카니즘을 활용하여 보호할 수는 없게 되었다. 왜냐하면, 악의적인 호스트 상에

서 이동 에이전트가 실행되므로 이동 에이전트의 코드, 데이터, 제어 흐름은 사실상 모두 위협에 노출될 수밖에 없기 때문이다. 즉, 전통적인 보호 매카니즘의 핵심인 비밀키의 안전성이 보장될 수 없게 되어, 이동 에이전트의 기밀성, 무결성, 가용성 등이 위협받게 되었다.

또한, 신뢰할 수 있는 하드웨어가 부재한 상태에서 소프트웨어만으로 이동 에이전트를 보호할 수 없다는 다양한 연구 결과가 제시되었다. 대표적으로 Farmer 등의 연구에서는 악의적인 호스트가 이동 에이전트의 코드, 데이터, 제어 흐름에 관한 모든 정보에 접근할 수 있기 때문에 사실상 호스트에 의한 공격으로부터 이동 에이전트를 보호하는 것은 불가능하며 따라서 이동 에이전트는 비밀 정보를 전달하지 않도록 하여야 한다고 주장한다.[8] 또한 Colin 등의 연구에서는 신뢰할 수 있는 하드웨어를 사용하지 않는 한 이동 에이전트를 보호하는 것은 불가능하다고 주장하였으며, Ordille 역시 호스트는 이동 에이전트의 데이터를 변경할 수 있고, 변경된 에이전트를 네트워크 상으로 전송할 수 있기 때문에 이동 중인 에이전트의 데이터를 보호하는 것은 불가능하다고 주장하였다.[5, 20]

그러나, 전자 상거래가 본격적으로 도입되면서 전자 화폐, 디지털 서명 정보 등 중요한 정보가 이동 에이전트 내에 포함되게 되었고 사용자를 대신하여 온라인으로 거래를 수행하는 다양한 이동 에이전트 시스템이 개발되면서 더 이상 이동 에이전트를 무방비 상태로 방치할 수는 없게 되었다. 따라서, 전통적 보호 매카니즘이 안고 있는 문제를 해결하고, 이동 에이전트의 특성을 반영한 새로운 보호 매카니즘 개발이 필요하게 되었다. 새로운 보호 매카니즘은 악의적인 호스트가 개별 코드와 코드의 의미와의 관계 그리고 메모리 비트와 데이터 구성 요소의 의미간의 관계를 분석할 수 없다면 이동 에이전트에 대한 공격이 어렵다는 점과 해킹을 위해서는 충분한 시간이 필요하다는 점에 착안하고 있다. 본 논문에서는 지금까지 수행되어온 이동 에이전트 보호 방법을 크게 전통적 보호 방법과 최신 보호 방법으로 분류하여 살펴본다.

4.1 전통적인 이동 에이전트 보호 방법

4.1.1 비 기술적 방법

- 법적인 보호 방법[4]

  법적 또는 계약에 의한 보호 방법은 네트워크에서 에이전트를 이용하게 될 호스트의 운영자가 자신의 호스트를 외부의 공격자로부터 안전하게 보호하고, 에이전트의 컴퓨팅과 관련된 비밀성 또는 무결성을 침해하지 않을 것을 법적 또는 계약 조건 등으로 동의하는 방법이다. 이 경우 법적 또는 계약 조건의 위배 여부를 탐색하는 매카니즘이 가장 중요하며, 적발된 호스트에 대해서는 관련 네트워크에 연결할 수 없게 하는 등의 법적 제재를 가하게 된다.

- 조직적 접근법[13]

  만일 호스트가 신뢰할 수 있는 권한 기관에 의해서 관리된다면 악의적인 호스트의 문제는 기관 내 문제로 축소시킬 수 있다. 즉, 신뢰할 수 있는 기관에 의해 해당 영역에 속한 모든 호스트들을 관리할 수 있으므로, 악의적인 호스트의 식별이 용이하게 된다. 조직적 접근법은 Telescript 기술을 사용하여 안전한 e-mail 서비스를 제공하는 AT&T의 PersonaLink에 적용되고 있다. 그러나, 이 방법은 분산 협동 환경 하에서 여러 서비스 및 정보 자원과 상호 연결을 통해 작업을 수행하는 이동 에이전트의 활동 영역을 제한하며 이동 에이전트의 가장

큰 장점인 "이동성"을 침해하므로 유용한 해결책이 아니다.

4.1.2 탐색 객체(Detection Object) 이용 방법[12,18]

탐색 객체 이용 방법은 데이터베이스의 악의적인 수정을 탐색하는 Naval Research Lab.의 연구를 응용하여, 합법적인 데이터베이스 이용자에 의해서도 수정될 수 없는 더미 데이터 또는 속성, 즉 탐색 객체를 사용하여 이동 에이전트에 대한 침입 여부를 탐색하는 방법이다. 이 방법은 에이전트의 생성자가 에이전트에 의해 생성된 데이터 및 에이전트와 에이전트 생성자 간에 통신했던 데이터 내용에 대한 침해 여부를 탐색하기 위해 탐색 객체를 사용할 수 있다는 점에 착안하였다. 즉, 에이전트 및 네트워크 상의 호스트들이 식별할 수 없는 탐색 객체를 생성하고, 탐색 객체의 수정 여부를 조사하여 수정이 발생하지 않는 경우에는 침해가 발생하지 않은 것으로 판정하는 것이다.

가장 저렴한 항공사를 탐색하여 비행기 표를 예약하는 항공 예약 에이전트의 예를 살펴보자. 우선, 에이전트 내에 더미 비행기를 탐색 객체로 설정하고 이를 에이전트 내의 탐색 목록에 포함하여 네트워크에 방출하였다고 가정해 보자. 그리고, 악의적인 호스트가 더미 비행기를 탐색 객체로 인식하지 못한 상태에서 탐색 목록 상에 있는 항공사로부터 수집한 정보를 삭제하고 자신의 비행기를 가장 저렴한 비행기로 등록한 후 에이전트의 생성자에게 그 결과를 전송하였다고 가정해 보자. 이때 에이전트 생성자는 더미 비행기와 관련된 정보가 존재하지 않는 것으로부터 에이전트에 대한 침해 사실을 탐색할 수 있다. 이 방법은 반복적으로 사용되는 일반적인 질의의 경우에는 매우 유용하지만, 다음과 같은 여러 가지 제약 사항을 갖고 있다.

- 이 기법은 어플리케이션별로 상이한 탐색객체를 사용되어야 한다. 왜냐하면, 항공 예약 에이전트에서 사용되었던 탐색 객체가 부동산 에이전트에서는 동일한 기능을 수행할 수 없기 때문이다.
- 더미 데이터로 활용되는 탐색 객체가 악의적인 호스트를 속일 수 있어야 한다.
- 악의적인 호스트가 에이전트의 반복적인 관찰을 통해 탐색 객체를 식별할 수 없도록 탐색 객체를 자주 교체하여야 한다.

4.1.3 결함 허용(Fault Tolerance) 이용 방법[10, 24]

결함 허용 이용 방법은 1996년 Schneider 등에 의해 제시된 방법으로 결함 허용 및 암호 분야에 이론적 근거를 두고 있다. 이 방법은 특정 영역 내에 단지 하나의 서버만이 존재하는 경우 또는 한 영역 내에 식별이 불가능한 여러 서버들이 존재하는 경우에 효과적으로 에이전트를 보호할 수 있는 방법으로 인정받고 있다. 항공 예약 에이전트를 이용하여 이 방법을 설명하기 위해 다음의 사항을 가정해 보자.

- 항공사 A는 하나의 서버(s_i)를 운영하고 있으며 악의적인 기능을 수행한다. 이 서버에서는 에이전트가 기존에 방문했던 항공사로부터 획득한 정보를 수정하여, 항공사 A를 가장 저렴한 항공사로 추천하도록 한다.
- 네트워크 상에 존재하는 항공사 서버들 간에는 안전한 통신 채널이 존재한다. 즉, 개별 서버는 공개키를 갖고 있고, 서버들은 이동 에이전트 생성자를 식별하기 위해 이 키를 사용한다.
- 침해가 발생하지 않은 서버에서는 에이전트 코드의 수정 여부를 판단할 수 있다.

- 에이전트 생성자는 에이전트가 방문하게 될 항공사 서버들의 순서를 정할 수 있다. 에이전트가 방문할 서버의 순서는 $S = s_1, s_2, \cdots, s_n$이고, 두 개의 에이전트 A_1과 A_2가 존재한다. A_1은 S를 따라서, A_2는 S^{-1}을 따라서 이동한다. 이 때 s_j는 에이전트가 정상적으로 작업을 수행할 때 선택하게될 가장 저렴한 항공사이다.

- 에이전트는 자신이 이전에 방문했던 서버를 식별할 수 있으며, 해당 서버와 정보를 교환할 수 있다.
  이 때 $j \leq i$일 경우를 살펴보면 A_1은 우선 s_j를 만나게 되고, 그 후 s_i를 만나게 된다. 이 때 A_1은 s_i에 의해 수정되어 s_i가 가장 저렴한 항공사라는 결과를 에이전트 생성자에게 보고하게 된다. 반면 A_2는 s_i를 먼저 만나 수정된 후, s_j를 만나게 되므로, 올바른 항공사인 s_j를 선택하여 그 결과를 에이전트 생성자에게 보고한다. 에이전트 생성자는 두 개의 에이전트로부터 제공되는 정보에 차이가 발생한다는 사실로부터 에이전트에 수정이 발생했다는 것을 알 수 있게 된다.
  그러나, 이와 같은 특수한 상황에서도 $j = i$ 인 경우에는 문제가 발생한다. 즉, 악의적인 호스트에서 제공하는 비행기의 가격이 가장 저렴할 경우 이 호스트에서는 최저 가격(P_i)을 P_i = (Second Best Price - ε)로 설정할 수 있기 때문이다. 이 밖에도 결함 허용 이용 방법은 복제된 에이전트들이 독립적으로 파괴될 수 있다는 비현실적 가정으로 인해 많은 비판을 받고 있다. 따라서 결함 허용 이용 방법은 일반적인 에이전트 보호 방법은 아니며 위와 같은 특수한 상황하에서 최적이 아닌 유용한 해결책으로 사용될 수 있다.

4.1.4 Probabilistically Checkable Proofs(PCP) 및 Computationally Sound Proofs(CSP) 이용 방법[4,22]

에이전트의 수행 결과가 침해되지 않은 에이전트에 의한 것인지를 분석하는 한 가지 방법은 전체 실행 기록(Execution Trace)을 에이전트 생성자에게 전송하여 정확성 여부를 판정하는 것이다. 그러나, 이 방법은 대량의 코드를 전송하기 위해 큰 대역폭을 필요로 하기 때문에 많은 비용을 발생시킨다. PCP와 CSP는 전체 실행 흔적 중 일부분을 이용하여 에이전트의 정확성을 조사하는 방법으로, 기존의 방법이 갖는 고비용 문제를 해결하였다. 에이전트의 프로그램 x와 이에 대한 실행 기록 y, 그리고 이들이 같은지 여부를 조사하는 $\rho(x, y)$가 존재한다고 가정해보자. 이전의 방법에서는 $\rho(x, y)$를 계산하기 위해 전체 실행 기록을 에이전트 생성자에게 전달했지만, PCP의 경우에는 정확성 판정을 위해 y의 일부분인 y'만을 전달하여 $\rho(x, y')$를 계산하면 된다. PCP에서는 에이전트 생성자가 y에 접근하여 y의 임의의 비트를 y'로 선택할 수 있도록 한다. 그러나, 이 경우 역시 상당량의 y'를 전송해야 하므로, 대역폭은 크게 절약되지 않으나, 검증 절차가 빨라진다는 장점을 갖는다. CSP 방법은 에이전트가 실행되고 있는 서버에서 y에 충돌 저항(Collision-resistant) 해쉬 함수 h를 사용하는 트리 해쉬 체계를 적용하고, 해쉬의 root만을 에이전트의 생성자에게 전달하여 에이전트의 정확성 여부를 조사하는 방법으로 root의 크기가 매우 작아 PCP 방법에 비해 많은 대역폭을 절약할 수 있다.

4.1.5 안전한 Coprocessor 이용 방법[4]

에이전트를 위한 안전하고 신뢰할 수 있는 실행 환경을 제공하기 위해 Coprocessor내에 에이전트의 실행 환경을 설정하는 방법이다. 대표적인 Bennet의 Sanctuary Project에서는

안전한 Coprocessor를 개발하여 Java기반의 에이전트가 안전하게 실행될 수 있는 에이전트 지원 API를 설치하였고, Java Interpreter사이를 안전하게 이동할 수 있는 기반 구조를 설정하였다.

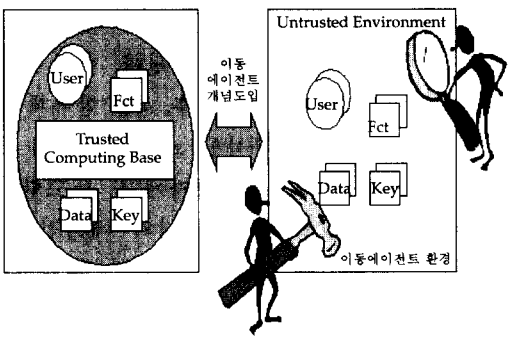
4.2 최신 이동 에이전트 보호 방법

최신 이동 에이전트 보호 방법 연구로는 Tomas Sander의 준동형 암호 체계(Homomorphic Encryption Scheme)와 함수 합성 기법(Function Composition Techniques) 등 이동 암호 체계를 이용한 에이전트 보호 방법과 Fritz Hohl의 코드 혼합 및 만기일 개념을 이용한 에이전트 보호 방법이 있다.

4.2.1 이동 암호 체계를 이용한 에이전트 보호 방법[23]

기존의 암호 체계를 이용하여 이동 에이전트를 보호하기 위해서는 비밀키를 안전하게 저장하고, 처리하며, 전송할 수 있는 TCB(Trusted Computing Base) 즉, 에이전트를 위한 안전한 실행 환경이 존재하여야 한다. 그러나, 기존의 암호 체계는 다음과 같은 문제점으로 인해 TCB를 제공할 수 없어 이동 에이전트의 보호 방법으로 사용될 수 없게 되었다.

- 기존 암호 체계를 적용하기 위해서는 호스트들간에 상호 신뢰할 수 있는 네트워크가 필수적이다. 그러나, 이 경우 네트워크 상의 호스트들의 기능을 통제하고 제약함으로써 분산 협동 환경 하에서 공개적으로 상호 협동하여 작업을 수행하는 이동 에이전트의 기본 철학을 침해하게 된다. 즉, 통제된 네트워크에 신규 가입이 있을 경우 기존의 모든 가입자들이 새로운 가입자를 신뢰할 수는 조치를 취해야 하며, 신규 가입자와 연결된 모든 호스트들에도 비밀키 등을 부여하는 막대한 노력이 필요로 된다.
- 기존 암호 체계를 적용할 경우 보호 서비스를 제공하기 위해 이동 에이전트와 에이전트 생성자 간에 지속적인 상호작용이 필수적이다. 그러나, Li Gong의 연구에 따르면 이동 에이전트 환경 하에서는 이동 에이전트에 속한 모든 데이터 및 코드가 악의적인 호스트에 의해 접근 가능하기 때문에 비밀키를 이용하는 기존의 암호 체계의 경우에는 이동 에이전트의 안전성을 보장할 수 없게 된다.[17] 따라서, 이동 에이전트와 에이전트 생성자 간에 최소한의 의사 소통만을 요구하는 암호 프로토콜이 필요하다.



[그림 8] 통신 환경의 변화

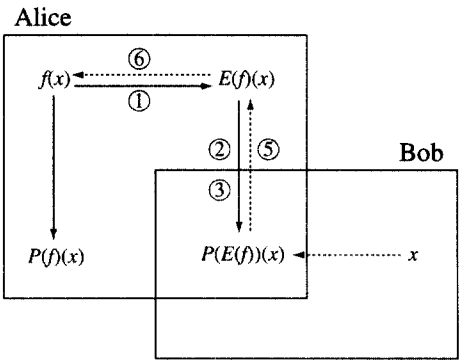
- 새로운 암호 체계에서는 TCB가 존재하지 않는 환경에서 이동 에이전트를 침해 위협으로부터 보호하기 위해 수학적으로 풀기 어려운 문제로 완벽히 증명될 수 있는 암호 프로토콜을 요구한다. 그러나, [그림 8]에서와 같이 기존 암호 체계에서는 TCB의 존재를 가정하고 있기 때문에 이동 에이전트 환경에서의 안전성을 객관적으로 증명할 수 없다.
- 기존 암호 체계에서는 이동 에이전트가 평문 코드 및 데이터로 구성되어 있다고

가정한다. 그러나, 실제 이동 에이전트가 반드시 평문 형태로 실행되어야 할 필요는 없다. 즉, 에이전트는 암호화된 형태로도 실행될 수 있다.

이와 같은 이유로 인해 기존 암호 체계로는 이동 에이전트를 안전하게 보호할 수 없게 되어 새로운 암호 체계에 대한 연구가 시작되었으며, 그 대표적인 것이 Sander와 Tschudin에 의한 "이동 암호 체계(Mobile Cryptography)" 연구이다. 이 연구에서는 준동형 암호 체계와 함수 합성 기법을 이용하여 에이전트 스스로 악의적인 호스트의 침입으로부터 자신을 보호할 수 있고, 비밀키를 노출시키지 않고 서명을 할 수 있으며, 실행 프로그램을 숨길 수 있는 암호 함수를 이용한 비 상호작용 컴퓨팅 기법(Non-interactive Computing with Encrypted Function)을 제안하였다. 비 상호작용 방식을 이용한 암호 함수는 TCB가 존재하지 않는 환경하에서 에이전트를 안전하게 보호할 수 있다.

- 암호 함수의 비 상호작용 컴퓨팅 기법

  암호 함수의 비 상호작용 컴퓨팅 기법은 실행 함수를 암호화하여 코드의 비밀성과 무결성을 보장하는 것이다. 이 기법을 설명하기 위해 다음과 같은 간단한 프로토콜을 살펴보자. 우선, 에이전트의 생성자인 Alice는 함수 f를 계산하기 위한 알고리즘을 갖고 있고, 원격 호스트인 Bob은 f에 대해 알지 못하지만 자신의 입력값 x를 적용하여 기꺼이 $f(x)$를 계산할 것이라고 가정해 보자. 이와 같은 상황하에서 적용될 암호 함수의 비 상호작용 프로토콜은 [그림 9]와 같으며, 다음과 같은 순서를 갖는다.



[그림 9] 암호 함수의 비 상호작용 프로토콜

(1) Alice는 f를 암호화한다.
(2) Alice는 $E(f)$를 포함하는 프로그램 $P(E(f))$를 생성한다.
(3) Alice는 $P(E(f))$를 Bob에게 전송한다.
(4) Bob은 x에 대해 $P(E(f))$를 계산한다.
(5) Bob은 $P(E(f))(x)$를 Alice에게 전송한다.
(6) Alice는 $P(E(f))(x)$를 해독하여 $f(x)$를 얻는다.

이 프로토콜에서 당사자들간에는 프로그램 및 마지막 결과 값만 교환하기 때문에 최소한의 상호작용만이 발생하게 되며, 상대방이 에이전트에 포함되어 있는 암호화된 함수를 분석할 수 없어 안전성이 보장되게 된다. 이와 같은 암호 함수는 준동형 암호 체계 및 함수 합성을 이용하여 작성할 수 있다.

- 준동형 암호 체계를 이용한 암호 함수

  준동형 암호 체계에 대한 연구는 1978년 Rivest가 덧셈에 대한 준동형(Additively Homomorphic) 특성을 갖는 암호 체계를 제안함으로써 시작되었으며, 1991년 Feigenbaum과 Merritt가 E(x)와 E(y)로부터 E(x+y)와 E(xy)를 쉽게 계산할 수 있는 공개키 함수의 존재에 대한 분석을

통해 구체화되었다.[15] 현재 입력 및 출력 값을 모른 채 암호화된 데이터 내의 다항식(Polynomial Expressions)을 평가할 수 있는 특성을 갖는 암호 함수는 쉽게 찾을 수 있으나, 암호화된 데이터에 대한 덧셈 및 곱셈을 모두 지원하는 암호 함수는 아직까지 발견되지 않고 있다. 그러나, Sander는 암호화된 다항식을 계산하기 위해서 암호 함수가 덧셈 및 곱셈 특성 모두를 만족할 필요는 없으며, 덧셈 및 혼합 곱셈(Additively and Mixed Multiplication) 특성을 만족할 수 있으면 암호 함수로써 이용될 수 있다는 사실을 증명하였다. 즉, R과 S가 환(Ring)이고, 암호 함수 E가 E : R → S 이고 x, y를 모른다는 가정 하에서 E(x), E(y)로부터 E(x+y)를 효율적으로 계산하는 알고리즘이 존재하는 경우를 덧셈에 관한 준동형(Additively Homomorphic)이라 하고, x를 모른 채 E(x)와 y로부터 E(xy)를 계산하는 효율적인 알고리즘이 존재하는 경우를 혼합 곱셈에 관한 준동형(Mixed Multiplicatively Homomorphic)이라고 하자. 만일 E: R → S가 덧셈 및 혼합 곱셈에 관한 준동형 암호 체계라면, E를 이용하여 다항식 $p \in R[X_1, X_2, \cdots, X_s]$에 대한 비 상호작용 암호 함수를 구현할 수 있으며, 특히 E: Z/NZ → R가 덧셈에 대한 준동형 암호 체계라면, E를 통해 다항식 $p \in Z/NZ[X_1, X_2, \ldots, X_s]$에 대한 비 상호작용 암호 함수를 구현할 수 있다.

- 함수 합성을 이용한 암호 함수

함수 합성 기법은 데이터와 함수를 결합하여 하나의 암호화된 함수를 생성함으로써 함수의 비밀성을 보장하는 방법으로, 준동형 암호 체계와 마찬가지로 비 상호 작용 컴퓨팅을 위한 암호 함수를 생성할 수 있는 기반을 제공한다. 하나의 함수 합성을 만들기 위해 우선 다음과 같은 사항을 가정해 보자.

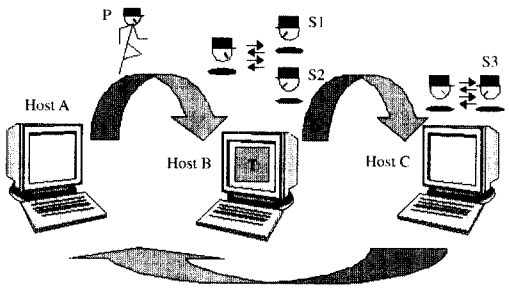
- Alice는 Bob의 컴퓨터 상에서 입력 값 x를 함수 A에 적용한 계산치 A(x)를 얻고자 한다.
- 이를 위해 Alice는 임의의 정칙 함수(Invertible Matrix) S를 선택하여 B: = S · A를 계산한 후 B를 Bob에게 전송한다.
- Bob은 y: = B(x)를 계산하여 y를 Alice에게 전송한다.
- Alice는 Bob에게 A를 노출시키지 않고 $S^{-1}y$를 계산하여 A(x)를 얻을 수 있다.

이때 만일 A와 S가 유리 함수(Rational Function)이고, B: = S · A라고 할 때, 이 방법의 안전성은 B로부터 A를 도출해내는 어려움에 근거하며, 현재까지 다변량 유리 함수를 Polynomial Time내에 분할하는 방법은 존재하지 않는다. 위와 같은 함수 합성 개념을 이용하여 함수 f와 서명 루틴 s를 결합하면 서명 함수를 노출시키지 않고 서명된 결과 값을 도출하는 디지털 서명 방식을 생성할 수 있다.

4.2.2 코드 혼합(Code Mess up) 및 만기일(Expiration Date)이용 방법[13]

1997년 Fritz Hohl에 의해 제시된 코드 혼합 및 만기일 이용 방법은 에이전트의 코드에 코드 혼합 방식을 적용하여 이해 및 분석하기 어려운 형태의 코드로 변형하고, 외부의 호스트와 교환되는 데이터와 코드 부분에는 시간 제약 정보, 즉 만기일을 첨부하여 공격자가 데이터를 읽고, 코드를 이해하고, 이를 조작하는데 소요되는 충분한 시간을 제공하지 않음으로서 에이전트를 안전하게 보호하는 방법이다. 코드 혼합 및 만기일 이용 방법을 설명하기

위해 [그림 10]과 같은 에이전트의 라이프사이클을 가정해 보자



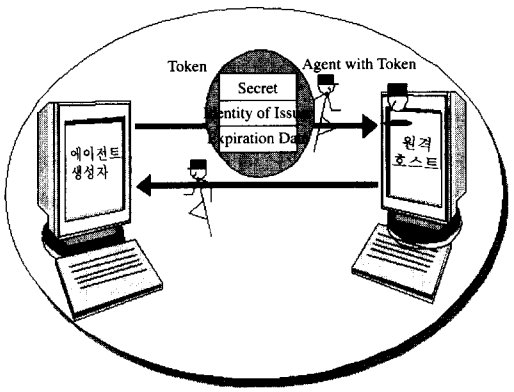
[그림 10] 코드 혼합 방법 적용 환경

- 호스트 A에서 출발한 이동 에이전트 P는 호스트 B로 이동한다. 이동 후 P는 꽃 서비스를 제공하는 에이전트 주소 목록을 제공하는 거래 중재 에이전트 T와 접촉한다.
- 호스트 B에는 꽃 서비스를 제공하는 두 개의 에이전트 S1과 S2가 있으며 P는 이들과 접촉하여 해당 꽃의 가격을 요청한다.
- P는 호스트 B상에서 호스트 C에 있는 S3에게 가격 정보를 요청한다. 이 때 S3는 가장 저렴한 가격을 제공한다고 가정한다.
- P는 S3로 이동한 후 자신의 전자 화폐를 이용하여 계약을 체결한 다음 꽃을 확보하고 호스트 A로 이동한다.

이와 같은 가정 하에서 코드 혼합 및 만기일을 이용하여 에이전트를 보호하기 위해서는 우선 호스트 A에 P에 코드 혼합 방법을 적용하여 이전과 동일한 기능을 수행하지만 전혀 다른 모습, 즉 다른 내부 구조를 갖는 새로운 형태의 코드로 된 이동 에이전트를 생성한다. 그리고, 코드를 변형하기 전에 만기일 정보를 외부 호스트와의 상호 작용에 사용되는 데이터 부분에 첨가한다. 이것은 전자 화폐와 비밀키 등의 중요 정보가 특정한 기간 동안만 유효하게 한다. 마지막으로 호스트 A는 에이전트 코드에 또 다른 만기일 정보를 이용하여 서명을 한 후 에이전트 P'를 호스트 B로 전송한다.

이와 같은 방법을 적용한 경우 에이전트 보호 효과는 다음과 같다.

- 만일 호스트 B와 C가 에이전트 P를 공격하는 악의적인 호스트라면, 호스트 B, C는 에이전트 P를 분석하는데 많은 시간을 소비한다. 왜냐하면, 호스트 B, C 입장에서는 이전에 보지 못한 새로운 에이전트 코드를 접하게 되기 때문이다.
- 만일 코드 및 데이터 요소에 대한 만기일이 단지 몇 초로 설정되어 있고, 호스트 B, C가 에이전트 P를 분석 및 조작하는데 이 시간을 초과하였다고 가정해보자. 그리고, 만기일이 지난 조작된 에이전트를 다른 호스트로 전송하였다고 가정해 보자. 이 경우 수신하는 호스트에서는 에이전트가 만기일을 초 과했기 때문에 이 에이전트를 유효하지 않은 것으로 판단하여 수신을 거부하게 된다.



[그림 11] 토큰을 활용한 만기일 이용 방법

[그림 11]은 에이전트의 데이터 중 일부분에 만기일 정보를 첨부하고, 이 부분을 디지털 서명 기법을 이용하여 보호하는 토큰 방식의 만기일 적용 매카니즘을 제시한다.

에이전트에 만기일 및 코드 혼합 개념을 사용하기 위해서는 다음과 같은 조건을 충족하여야 한다.

- 새로운 코드 및 데이터를 분석하는데 소요되는 최소한의 시간을 찾을 수 있어야 한다.
- 이 시간을 이용하여 데이터 요소에 대한 만기일을 설정한다.
- 이 시간 동안 에이전트는 충분한 작업을 수행할 수 있어야 한다.
- 주기적으로 새로운 코드를 생성할 수 있어야 한다. 이 때 새로운 코드는 자동화된 도구에 의해서 생성되며, 이전 코드와 동일한 기능을 수행할 수 있어야 하고, 도구 또는 사람에 의한 분석이 어려워야 한다.
- 코드 변환은 큰 공간(즉, 하나의 코드가 여러 개의 변수들로 변형될 수 있는 공간)에서 수행되어야 한다.
- 만기일을 갖는 데이터의 사용으로 인한 어플리케이션의 제약이 적어야 한다.

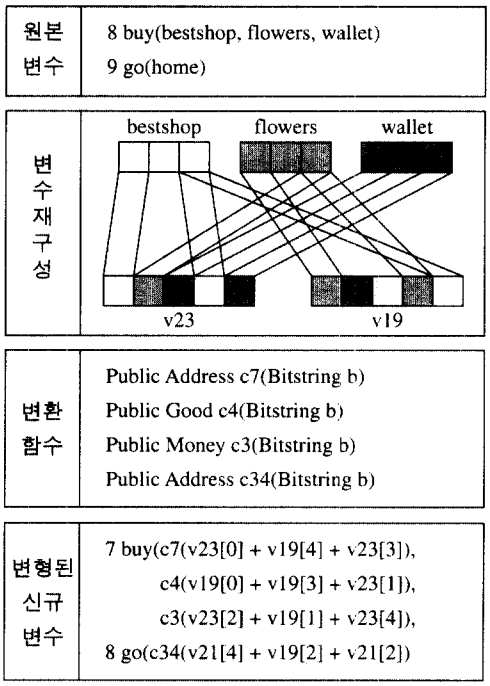
만일 위의 조건을 만족할 수 있다면, 만기일 전에 코드, 데이터 및 제어 흐름의 관찰 및 조작, 코드 오작동 등 호스트의 공격으로부터 보호될 수 있는 에이전트 시스템을 생성할 수 있다. 또한 에이전트의 데이터 및 코드가 특정한 시간 동안 안전성이 보장되므로 공개키 방식 등을 이용한 인증을 통해 악의적인 호스트의 사용자 위장 공격도 방어할 수 있다.

● 코드 혼합 매카니즘

코드 혼합 방법을 적용하기 위해서는 읽기 쉽고 이해가 용이한 코드를 가능한 한 읽기 어렵고 이해할 수 없는 형태의 코드로 변형하는 코드 혼합 매카니즘을 필요로 한다. 현재까지 코드를 체계화된 방식으로 혼합하는 매카니즘은 존재하지 않지만, 다음과 같은 여러 분야에서 코드 혼합 매카니즘에 대한 연구가 진행되고 있다.

- 소프트웨어 공학 분야

  소프트웨어 공학의 한 분야에서는 코드를 읽기 쉬운 형태로 만드는 방법을 연구하여 이에 대한 지침을 제시한다. 즉, 읽기 쉬운 코드를 만들기 위해서는 우선 의미를 갖는 변수 이름을 설정하고, 서브 루틴 등을 사용하여 코드를 모듈화하며, 프로그램을 간단하게 하는 데이터 표현 방법 등을 선택하도록 권고하고 있다. 소프트웨어 공학 분야에서의 코드 혼합 매카니즘은 변수 재결합방법(Variable Recomposition), 구조분해(Structure Dissolving) 및 제어 흐름 요소 변환 방법 등을 이용하여 위의 조건을 위반하는 코드를 생성한다.



[그림 12] 변수 재결합을 이용한 코드 혼합 매카니즘

① 변수 재결합 방법은 새로운 프로그램 변수들을 원래 변수의 일부 데이터와 혼합하여 생성하는 방법이다.
[그림 12]에서는 변수 재결합 방법을 적용하여 원래의 3개의 변수 bestshop, flowers, wallet으로부터 두 개의 새로운 변수 v23과 v19를 재구성하는 변수 재구성 체계와 새로운 변수들에 대한 인수 값들을 생성하는 코드 변환 함수를 사용하여 최종적인 새로운 변수 코드가 생성하는 예를 보여 준다. 그림에서 보듯이 변수들과 변환 함수간에는 직접적인 관계가 존재하지 않으며, 변수 명과 데이터 표현은 더욱 복잡한 형태를 갖게 된다.

② 구조 분해 방법은 블록 또는 프로시져와 같은 프로그램 구조를 제거하여 내부구조를 갖지 않는 프로그램을 생성하는 방법이다. 이를 위해서 지역 변수를 전역 변수로 변환하고, 프로시져 코드를 이용하여 프로시져 호출(call)을 대체하며 블록을 "goto"와 같은 선언문을 사용하여 대체하는 방법 등이 사용될 수 있다. 그러나, 구조 분해 방법은 코드의 외부 구조에만 적용될 수 있다는 한계를 갖는다.

③ if 또는 while 선언문과 같은 제어 흐름 요소는 제어 흐름이 혼합되는 컴화일 시간 동안에도 프로그래머의 분석 대상이 된다. 그러나, 만일 이러한 요소들을 변수들의 내용에 의존적인 형태로 변형할 수 있다면 제어 흐름은 이전처럼 쉽게 분석될 수 없을 것이다. 의존성은 변수 내용에 제한을 받는 switch 선언문 등을 사용하여 제공할 수 있으며 간단한 변수 대신 복잡한 변수 표현을 사용하여 분석을 더욱 어렵게 할 수 있다.

- 리엔지니어링 분야
리엔지니어링 분야에서는 순수(Bare) 코드를 이해하기 쉽고 체계화된 프로그램 형태로 변형하는 방법을 연구한다. 따라서, 이 연구 수행 결과로 표현되는 코드 변형에 어려움을 초래하는 요소들에 대한 정보는 코드 혼합 매카니즘 작성 시 중요한 자료로 사용될 수 있다.

- 자동화된 코드 분석
자동화된 코드 분석은 주로 프로그래밍 언어나 컴화일러 구조 분야에서 수행되는 연구로써 이전에 계산된 표현을 동일한 값을 계산하는 다른 표현으로 대체하는 "공통 표현 제거 방법(Deletion Method of Common Representation)"이 코드 혼합을 위해 적용될 수 있다. 이 방법은 원본 코드 보다 적은 양의 코드를 사용하기 때문에

처리 속도를 향상시키며, 원본 프로그램과의 의미적인 관계를 삭제함으로써 분석이 복잡한 프로그램을 만들 수 있다.

- 사용되지 않는 코드 탐색 방법
  이 방법은 프로그램 내에서 사용되지 않은 코드를 탐색하는 코드 최적화 매카니즘으로서 만일 이 매카니즘을 방해할 수 있는 기법을 찾는다면 코드를 더욱 이해하기 어렵게 할 수 있다.

5. 결 론

본 논문에서는 이동 에이전트와 관련된 침해 위협을 분석하고 악의적인 호스트로부터 이동 에이전트를 보호하기 위한 대책을 중심으로 이동 에이전트 보호 대책을 살펴보았다. 현재 선진국의 정부 기관, 연구소, 산업체 등 모든 분야에서는 미래의 통신 환경의 요구, 즉, 이질적인 통신 환경의 통합, 인터넷 및 이동 통신의 급속한 신장, 전자 상거래의 활성화 등을 실현할 핵심 매개체인 이동 에이전트에 대한 많은 연구를 수행하고 있다. 에이전트 분야의 국제 표준화 기구인 FIPA(Foundation of Intelligent Physical Agent)에서는 이질적인 통신 환경 하에서 이동 에이전트의 원활한 운영을 위한 연구를 중심으로 국제 표준안을 제정 중이며, 1997년 10월에는 에이전트 관리, 에이전트 언어, 에이전트와 비에이전트 소프트웨어간 상호 운용에 관한 표준안을 제정하였다.[9] 또한, IETF(Internet Engineering Task Force)에서는 1996년 12월에 Uniform Resource Agent에 대한 RFC 초안을 제안하였으며, OMG(Object Management Group)에서도 이동 에이전트 기능에 대한 규격서를 작성 중에 있다. 현재 상용화된 이동 에이전트 시스템으로는 General Magic의 Telescript, FTP Software의 Cyberagent, Guideware의 Clearlake Mobile Software Agents, 일본 IBM의 Aglets, NTT의 Webagent 등이 있으며, 국내에서도 ETRI 등을 중심으로 에이전트 시스템 개발 연구가 활발히 진행 중이다.[1,2,3]

그러나, 이동 에이전트의 실현에 가장 큰 걸림돌이 되는 보호 문제에 대해서는 아직까지 많은 연구가 진행되지 않고 있으며, 특히, 전자 상거래의 구현으로 인해 필수적인 이동 에이전트를 악의적인 호스트로부터 보호하는 방법에 대한 연구는 최근에 와서야 비로소 관심을 끌기 시작하였다. 안전한 이동 에이전트의 운영이 보장되지 않는 한 미래의 통신 환경에서 이동 에이전트의 활용 전망은 밝다고 할 수 없다. 따라서, 악의적인 호스트가 존재하는 네트워크 환경에서 스스로 지능을 갖추고 분산 협동 환경 하에서 자유롭게 이동하면서 사용자를 대신하여 작업을 수행하는 이동 에이전트를 안전하게 보호하기 위한 방법 연구에 많은 노력을 기울여야 할 것이다.

참 고 문 헌

[1] 송영기, 인소란, 김명준, "소프트웨어 에이전트 보호기술," 전자통신동향분석, 제12권 제6호, 1997.12

[2] 장명욱, 박상규, 이광로, 민병익, "이동 에이전트 기술 동향," 전자통신동향분석, 제12권 제1호, 1997.2

[3] 최중민, "에이전트의 개요와 연구방향," 정보과학회지, 제15권 제 3호, 1997.3

[4] Bennet S. Yee, "A Sanctuary for Mobile Agents," Technical Report CS97-537, Computer Science Department, Univ. of California in San Diego, Feb. 1997.

[5] Colin G. Harrison, David M.Chess, Aaron Kershenbaum, "Mobile Agent: Are they a good idea ?," IBM Research Division, March. 1995

[6] David Chess, Benjamin Grosof, Colin Harrison, "Itinerant Agents for Mobile Computing," IBM Research Report, March. 1995.

[7] David M. Chess, Security in Agent Systems, http://www.research.ibm.com/massive

[8] Farmer William, Guttmann Joshua, Swarup Vipin, "Security for Mobile agents:Issues and Requirements," Proceedings of the National Information Systems Security Conference,1996.

[9] FIPA, FIPA '97 Draft Specification Part 1 Agent Management Rev 2.0, FIPA Agent Management Technical Committee, June 1997.

[10] Fred B. Schneider, "Towards Fault-tolerant and Secure Agentry," 11th International Workshop on Distributed Algorithms, Sept. 1997.

[11] Gary Robert, "Agent Tcl: A Flexible and Secure Mobile-agent System," Proceedings of the 4th Annual Tcl/Tk Workshop, July 1996.

[12] Giovanni Vigna, "Protecting Mobile Agents through Tracing," Mobile Object Systems ECOOP Workshop' 97, 1997.

[13] Hohl, Fritz, "An Approach to solve the problem of malicious hosts," University of Stuttgart, 1997.3

[14] Hyacinth S. Nwana, Software Agent: An Overview, Cambridge University Press, 1996.

[15] J. Feigenbaum and M. Merritt, "Open Questions, Talk Abstracts, and Summary of Discussions," DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 1991.

[16] Jim White, Mobile Agents White Paper, General Magic, 1996. http://www.genmagic.com/agents/Whitepaper/whitepaper.html

[17] Li Gong, "Survivable mobile code is hard to build," DARPA Workshop on Foundations for Secure Mobile Code Workshop, March. 1997.

[18] Meadows Catherine, "Detecting Attacks on Mobile Agents," DARPA Workshop on Foundations for Secure Mobile Code Workshop, March. 1997.

[191] Michael R. Genesereth, Steven P. Ketchpel, "Software Agents," Communications of the ACM, Vol. 37, No. 7, July 1994.

[20] Ordille Joann, "When agents roam, who can you trust," Proc. of the First Conference on Emerging Technologies

and Applications in Communications, May. 1996.

[21] Robert Wahbe, Steve Lucco, T. E. Anderson, and Susan L.Graham, " Efficient Software-based Fault Isolation," In Proceedings of the ACM SIGCOMM 96 Symposium, 1996.

[22] Silvio Micali, "CS Proofs," In Proceedings of the 35th IEEE Symposium on Foundations of Computer Science, Nov. 1994

[23] Tomas Sander and Christian F. Tschudin, " Towards Mobile Cryptography," International Computer Science Institute, Nov. 1997.

[24] Yaron Minsky, Robbert van Renesse, Fred B. Scheider, and Scott D. Stoller, "Cryptographic Support for Fault-tolerant Distributed Computing," Technical Report TR96-1600, Department of Computer Science, Cornell University, July. 1996

□ 著者紹介

이 영 화

1991년 2월 고려대학교 경영학과 (학사)
1993년 2월 고려대학교 대학원 경영정보학 (석사)
1993년 ~현재 국방정보체계연구소 연구원

※ 주관심분야 : 컴퓨터/네트워크 보안, 에이전트 시스템, EC 등

이 남 용

1979년 숭실대학교 정보과학대학 전자계산학과(공학사)
1982년 고려대학교 경영대학원 경영정보학(석사)
1987년 정보처리기술사
1993년 Mississippi State Univ. 경영정보학 (박사)
1983년 ~현재 국방정보체계연구소 연구위원

※ 관심연구분야 : 컴퓨터/네트워크 보안, 정보시스템 감리 등