Free-Roaming 실행 환경에서 절단공격으로부터 이동에이전트의 안전한 실행 보장 기법

정창렬* · 이성근**

Secure Execution Assurance Mechanism of Mobile Agent from Truncation Attack in Free-Roaming Environments

Chang-Ryul Jung* · Sung-Keun Lee**

본 논문은 지식경제부 및 정보통신연구진흥원의 지원을 받아 수행된 연구결과임

요 약

Free-roaming 이동 에이전트의 데이터 보호는 이동성과 호스트 간 매핑으로 인해 보안에 대한 심각한 위협으로 완전히 해결되지 못한 문제이다. 특히 절단공격을 방어하는 측면에서 그렇다. 그러므로 사용자 중심의 응용 기술에 에이전트가 이용될 때 에이전트의 안전한 실행 보장은 필수적이다. 본 논문에서는 에이전트의 실행 중 악의적인 호스트에 의해 발생하는 보안 위협으로부터 안전한 실행을 보장한다. 그리고 공격자에 의해서 선의의 호스트가 악의적으로 남용되는 것으로부터 에이전트 실행을 보장하도록 하는 연쇄적으로 두 개의 호스트와 다음 두 개의 호스트 간에 체인관계 형성이 가능하기 때문에 안정성이 보장된다. 이는 안전한 이동에이전트 실행보장을 위한 실행 추적 프로토콜 메커니즘을 제안한다. 그리고 보안 분석을 통해 안전성을 분석한다.

ABSTRACT

The data security of free-roaming mobile agent is a problem which hasn't been resolved to the mobility and inter-host mapping. totally, especially in theaspect of keeping away truncation attacks. Therefore, when the agent is utilized for user oriented applied technique, the secure execution guarantee of agent is essential. In this paper, it guarantees safe execution from security threats generated by malicious host during the agent's execution. And the secure execution guarantee mechanism of agent is proposed from favorable host is maliciously abused by attacker. Thus, the execution trace protocol mechanism proposed as secure mobile agents execution guarantee. As security analysis of the safety analysis.

키워드

이동 대행, 공격, 악성 코드

Key word

mobile agent, attacker, truncation attack, malicious host

* 순천대학교 컴퓨터공학과

** 순천대학교 멀티미디어공학과(교신저자)

접수일자 : 2009. 10. 13

심사완료일자 : 2009. 10. 29

Ⅰ. 서 론

이동에이전트 기술은 분산 컴퓨팅환경에서의 새로운 패러다임으로 전자상거래, 개인 임무지원, 탐색과 검색, 모니터링 그리고 네트워크 관리등에 많이 응용되고 있다. 이들을 실현하기 위해서는 에이전트가 네트워크에서 자유로운 이동이 이루어짐으로써 악의적인 요소들로부터 프라이버시나 실행 결과에 대한 안전이 확보되어야 한다[1,2,8,11]. 이를 위해 다양한 측면에서 에이전트 보호를 위한 보안 프로토콜과 암호화 기법들이 적용되고 있으나, 에이전트를 실행할 때 공격자에 의해 정상적인 호스트가 악의적으로 남용되어 수행결과에 대한 혼란을 가져오고 있다[3,4,5]. 그러므로 에이전트는 주어진 임무를 마칠 때까지 무결성과 기밀성이 보장되어야 하고[6,7],

에이전트를 위한 많은 보안 프로토콜들은 자유롭게 이동하는 이동 에이전트를 보호하는데 초점을 맞추고 있다. 공격자는 이동 중인 에이전트를 공격하기 보다는 악의적인 호스트를 이용하는데 주력하고 있다.

본 논문에서는 에이전트가 실행되는 동안 안전한 이동과 에이전트 데이터의 안전을 위해 공격자로부터 에이전트의 안전한 실행 보장 기법을 제안한다. 이를 위해 에이전트의 자유롭게 이동하는 환경에서 공격자의 절단공격으로부터 이동 에이전트의 안전한 실행을 보장하는 메커니즘을 실행과정의 메시지 시퀀스를 통해 프로토콜의 안전성을 확인한다. 또한 에이전트는 이동경로가 결정되어 있지 않음으로 하나 이상의 호스트와 랜덤하게 매핑 한다. 이로 인해 악의적인 호스트들에 의해 에이전트의 실행 결과의 데이터를 위협받게 되어 프라이버시 침해가 이루어진다. 본 논문은 이러한 위협으로부터 에이전트의 안전한 실행이 보장될 수 있는 메커니즘을 제시한다. 본 논문은 2장에서는 관련 연구들을 살펴보고, 제3장에서는 절단공격으로부터 에이전트 실행 문제을 기술한다. 제 4장에서 에이전트의 안전한 실행과 절단공격을 방지하기 에이전트 실행 추적 메커니즘에 대해 제시한다. 제 5장에서는 제안한 메커니즘에 대한 보안 분석을 실시하고 한다. 마지막으로 결론 및 향후 연구에 대해 기술한다.

Ⅱ. 관련연구

에이전트는 자유롭게 이동하는 특성을 지니고 있어 에이전트의 실행 결과 데이터를 보호하기 위해 사용되는 메커니즘은 크게 두 가지의 범주에서 사용되고 있다. 이는 악의적인 요소들로부터 예방하는 메커니즘과 오류나 공격들을 탐지하는 메커니즘으로 나뉜다.

예방 메커니즘은 이동 에이전트가 데이터를 보호하기 위해서는 악의적인 호스트나 신뢰되지 않는 호스트로 이동하지 않아야 한다. 이를 위해서는 Java Card와 같은 특별한 소프트웨어를 통해서 신뢰감있는 컴퓨터 기술을 사용한다[10]. 또 다른 방법은 보안위협을 제한하기 위해 공격의 영향을 제한하기 위하여 보다 더 복잡한 접근 제어 체제를 가동하는 것이다. 이것은 모놀리식 샌드 박스 정책의 세심한 고안처럼 보다 작은 어플리케이션 전용 정책들을 조사한다. 최근에는 각기 다른 호스트들에 의해 생성되어진 각기 다른 데이터 사이의 이차원적 관계를 형성하기 위해 에이전트 복제를 사용하는 방법들이 제시되었다[7].

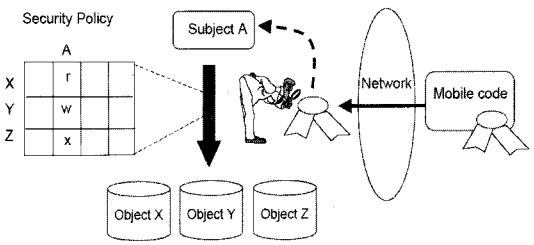

그림 1. 샌드박스 모델을 위한 보안정책
Fig. 1 security policy for sand box Model

Sun은 접근 통제의 정의에서 이동 코드를 실행하는 사용자의 동일성을 완성하는데 초점을 맞춘 JAAS(Java Authentication and Authorization Services)를 공개되기도 하였다. 이동 에이전트의 코드에 의해 수행되는 효과들은 동시에 정말로 유용한 소프트웨어를 실행하고 쓰는 것을 허용하는 것이다. 그러나 접근 체제의 시행은 런 타임(runtime)에 동적으로 수행되어지기 때문에 많은 비용을 필요로 하는 한계점을 지니고 있다.

에이전트 복제를 통한 악의적인 요인들이 에이전트

를 공격하는 것으로부터 예방하는 것은 에이전트가 자유로운 이동을 하기 때문에 이동하는 형식이나 폼에 의존하여 에이전트 데이터를 안전하게 보호하는 방법들은 악의적인 요소들의 절단공격과 같은 복잡한 공격에는 예방하는데 한계가 있다[11].

그렇기 때문에 탐지 메커니즘에 대한 많은 학자들의 연구가 활발히 이루어지고 있다. G. Karjoth[1,3], F. Hohl[2], A. Corradi et al[5], Karnik[7], A. Carzaniga[10]은 에이전트 데이터의 기밀성을 보호하기 위해 에이전트가 신뢰된 호스트에서 데이터를 이용할 수 있도록 하는 방법을 제안하였다. 송・수신자 간 자신의 비밀 키를 이용하여 평문(plaintext)을 해독 및 새로운 정보 추가 후 평문을 생성한다. 그런 다음에 체크 합을 통해 공격을 감지한다. 이는 여러 가지 형태의 공격에 취약할 뿐만 아니라 중복 등의 문제로 인해 체크 합의 검증을 입증이 어려운 요인을 지니게 되어 자신도 모른 사이 악의적인 호스트로 남용될 수 있다. 이러한 문제에 대해 A. Corradi et al[5]은 Multi-hop Protocol로서 이전의 홉(hop)의 결과들을 현재의 결과와 식별자 체인 관계를 형성함으로 체크 합을 입증하여 문제를 개선하였다[8]. 그러나 서명된 데이터는 수신자에게 기밀성을 보장하지 않는 문제점과 체인관계를 통해서 오류를 명확하게 감지하는데 한계가 있다. 또한 G. Karjoth[1,3]에서는 체인 관계를 랜덤 값과 서명을 이용하여 강화를 하였다.

이 방법은 서명자의 신분 추론이 가능함으로 공격자에 의해 서명자의 서명 체인 관계를 끊어 공격자의 새로운 서명 체인을 교묘히 삽입할 수 있는 문제점이 있다. 이는 공격자들에 의해 절단공격이 가능하여 악의적인 호스트로 남용되는 피해가 발생한다.

이러한 에이전트의 절단공격으로 에이전트의 실행 과정에서 발생하는 문제점에 대해 구체적으로 고찰한다.

Ⅲ. 절단공격으로부터 에이전트 실행 문제

에이전트의 활동이 안전하게 보장되어야 안전한 실행이 가능하다. 오픈 네트워크 환경에서 에이전트는 호스트에 기초한 전형적인 접근을 하기 때문에 악의적인 요소들에 의해 악의적으로 남용되는 공간으로 이용되기도 한다. 즉 암호화된 프로토콜의 악의적인 공격은 악성 데이터를 생성하고 에이전트를 제어하는 공격자들로부터 발생한다. 에이전트 서버(AS)에 에이전트가 이동되는 동안에 체인관계로 에이전트의 여정에 따라 이동하게 된다. 이는 그림 2에서와 같이 AS_{i+1}에서 AS_{n-1}사이에서 발생할 수 있음을 구체적으로 보여주고 있다.

$(1) AS_i \rightarrow AS_{i+1} (0 \le i \le n)$

$(2) AS_i \rightarrow AS_{i+1} \rightarrow AS_{i+2}$

$(3) AS_i \rightarrow AS_{i+1} \rightarrow AS_{i+2} \cdot \cdot \cdot \rightarrow AS_{n-1}$

$(4) AS_i \rightarrow AS_{i+1} \rightarrow AS_{i+2} \cdot \cdot \cdot \rightarrow AS_{n-1} \rightarrow AS_n \rightarrow AS_i$

그림 2. (2)와 (3)에서 절단공격 발생
Fig. 2 truncation attack in (2) and (3)

그림 2는 공격자의 공격으로 발생 가능 한 에이전트 서버들을 나타내고 있으며, (2)와 (3)에서는 공격자의 활동이 가능한 홉(hop)으로서 AS_{i+1}에서 AS_{n-1}사이에서 절단공격이 발생이 이루어진다. 공격자의 활동은 모든 에이전트 서버에서 발생하는 것이 아니고 특정한 홉에서만 발생한다.

첫 시작 AS(AS_i)는 에이전트 생성자임으로 불가능하고, 처음 소유자에게 되돌아오기 직전에는 AS_i와 상호 인증이 이루어지기 때문에 공격자의 공격활동이 노출됨으로 이루어질 수 없다.

공격자에 의한 악의적으로 활동이나 남용되는 것에 대한 방어적인 메커니즘에 대한 몇몇 연구들이 있다. [7]에서는 targeted state를 두어 에이전트의 기밀성을 보호하는 방법으로 제안되어 믿을 수 있는 호스트에서만 에이전트를 실행하게 하기 위한 방법이다. 이를 위해 각 각의 호스트들은 개인키와 해시 함수를 이용한 디지털 서명을 하고, 에이전트를 수신한 호스트들은 에이전트의 서명 확인을 위해 공개키를 이용하여 상호작용을 한다. 에이전트를 처음 생성하여 소유하고 있는 호스트로써 에이전트 서버를 갖고 전송되어 이동한 에이전트들을 자신의 복호키를 사용하여 암호화된 메시지를 복

호화 하여 에이전트의 활동 임무를 확인하여 실행을 허락한다하는 일을 하는 것은 에이전트 상태 구조의 소유자이다. 이러한 에이전트 상태 구조를 표현하면 $ENC_X(\{Agt_1\}k_i)$ $(1 \le i \le n)$와 같다.

구조에서 수신한 에이전트를 초기 상태와 키 $\{Agt\}k_Z$를 복사하고, 에이전트의 전송은 이동계획에 의해 $(ENC_Y\{m\}k_Y))$키와 함께 전송한다.

이 과정에서 공격자들의 공격을 검출되지 못하면 해당 호스트들은 악의적으로 남용되게 된다. 이를 위해 에이전트에 새로운 객체를 추가 할 때 마다 하나의 컨터이너에 포함시켜 수정과 도청으로부터 에이전트 데이터를 보호하고 체크 합(checksum)을 이용하여 컨테이너 훼손유무를 점검한다. 즉 초기의 체크 합이 C_0=checksum $= ENC^{Pub}_{X_0}\{object\}$라면, 체크 합은 에이전트가 처음 출발할 때 공개 키에 의해 시작한다.

에이전트의 이동은 $i_0, i_1, \cdots, i_n$로 표현한다면, i_0는 처음 출발하는 에이전트 서버 AS_i이고, i_n은 에이전트가 현재 에이전트가 있는 곳을 의미한다. 이때 $object$는 에이전트 소유자인 AS_i만 유일한 비밀 키로 암호화되어 에이전트의 생성자자에게 되돌아오게 되면 인증 프로토콜에서 비밀키를 사용하여 안전성을 체크한다. 또한, 체크합은 랜덤 한 $object$에 기초하여 에이전트 소유자에 의해서만 계산이 이루어진다.

즉, 컨테이너는 $(ENC_{i_1}\{Agt_1\}, \cdots, ENC_{i_n}\{Agt_n\}, C_n)$를 포함하고 있으며, 새롭게 추가되고자 할 때는 체크 합 $(C_{n+1} = ENC^{pub}_X\{C_n \| ENC_{i_{n+1}}(m_{n+1})\})$도 함께 업데이트 되어 에이전트가 AS_i로 되돌아오면 체크 합의 분석이 이루어진다. 에이전트의 소유자(AS_i)에 의해서 에이전트가 AS_{i+1}로 이동되면 AS_{i+1}는 C_j $(1 \le j \le n)$를 알 수 있다. AS_{i+1}는 현재의 체크 합(C_n)이 컨테이너에 보관되어 있기 때문에 언제든지 C_n을 알 수 있다. 또한 AS_{i+1}는 에이전트가 이전에 방문한 다른 에이전트 서버들과 공모하거나 에이전트가 이동되어 루프(loop) 되었다면 C_j $(j<n)$를 발견할 수 있다. 그렇기 때문에 AS_{i+1}는 컨테이너 실행을 최대 j번째에서 끝내고 에이전트를 원래의 형태로 되돌려서 위조 경로에서 활동하게 하거거나, 익명의 호스트로 가장하여 $l > j$인 오브젝트 m_l을 제거하거나 첨부한다. 이로써 $j+1$번째에 임으로 변경하고자 하는 데이터와 체크 합 C_j을 지닌 공격자의 컨터이너를 생성 한 후 AS_{i+1}의 에이전트를 AS_{i+2}나 AS_{i+n}에게 전송한다. 전송된 에이전트는 m_{j+1}에 목표된 상태를 삽입한 후 AS_{i-1}에게 다시 되돌려 보내는 구조는 다음과 같다.

$$AS_{i+1} \rightarrow AS_{i+2} : (\{m_{j+1}\}, C_j),$$
$$AS_{i+2} \rightarrow AS_{i-1} : (ENC_{AS_{i-1}}\{m_{j+1}\}, ENC^{pub}_{AS_{i-1}}\{C_j \| ENC_{AS_{i-1}}\{m_{j+1}\}\})$$

에이전트가 AS_{i+2}에서 AS_{i-1}로 되돌아오면 AS_{i-1}는 자신의 비밀 키를 이용하여 체크 합을 계산 후 AS_i의 공개키로 다시 암호화하면 하나의 새롭운 컨테이너가 생성된다. 구조는 다음과 같다.

$$C_{j+1} = ENC^{pub}_X\{ENC_Y\{ENC^{pub}_Y\{C_j \| \{m_{j+1}\}\}\}\}.$$

또한 새로운 형태의 컨테이너와 기존 컨테이너의 결합은 다음과 같다.

$$\{ENC_{i_1}\{m_1\}, \cdots, ENC_{i_j}\{m_j\}, ENC_Z\{m_{j+1}\}, C_{j+1}\}$$

이는 AS_{i+2}가 공격자에 의해 공격이 이루어질 경우 AS_{i+2}가 실행 후 결과들에 대해 AS_{i+2} 이전의 에이전트들의 인증서를 되돌릴 때 까지 체크 합이 계속해서 이루어진다. 하지만 AS_{n-1}에서는 에이전트 소유자의 공개키로 암호화되어 있음으로 C_j가 현재 상태에서는 공개적으로 입증되지 않는 문제가 있다.

때문에 공격자들에 의해 에이전트의 프라이버시는 계속해서 공격당할 것이다. 뿐만 아니라 검증 과정에서 컨테이너 안에 있는 C_j의 길이 이외에는 확인되지 않으므로 전체 오브젝트 수를 측정한 후 이상이 없다면 악의적으로 남용되어 의심없이 에이전트 실행이 이루어짐으로 문제가 발생가능성이 크다. 더욱이 체크 합은 채널을 통해 전달되는 고의가 아닌 우연히 발생하는 오류들을 방지하는 수동적인 암호화 체크기법으로 수신된 체크 합과 비교하여 오류를 탐지 하는 소극적인 무결성 체크 방법이다.

또한 G. Karjoth[1]는 free-roaming이 가능한 에이전트에 의해 획득된 에이전트 데이터의 기밀성을 보호하기 위한 또 다른 프로토콜의 형태인 The KAG Family of

Protocols을 제안하였다.

이는 쇼핑 에이전트의 시나리오에서 쉽게 찾아 볼 수 있다. 에이전트에 의해 많은 가게를 방문하여 가게들로부터 제공된 수집된 정보에 대한 에이전트 데이터의 기밀성을 보장하기 위해 디지털 서명 체인을 검증하는 기법이다. 공격자는 에이전트 초기 생성자 AS_i로부터 에이전트를 수신했다면 공격자는 $i_n(n>1)$번째가 될 것이다.

만약 AS_{i+2}가 공격자라면, AS_{i+2}는 j $(0<j<n)$인 새로운 i_{j+1}을 선택하고, $j-1$의 체인관계를 유지하기 위해서 j를 선택하고 i_j의 자료를 생성한다. 이러한 과정은 AS_{i+2}는 $AS_{i+2} \rightarrow i_{j-1}$에서 메시지 M_{j-1}를 제공하고, $i_j \rightarrow i_{j+1}$에서 M_j를 제공한다. 이때 공격자는 AS_{i+2}에서 임의의 자신의 새로운 서명 체인관계를 형성하여 위장한다.

그리고 i_{j+1}에서는 다시 공격자 AS_{i+2}에게 메시지 이동 과정($i_{j+1} \rightarrow AS_{i+2}$)를 거친다.

$$AS_{i+2} : i_{j+1} \rightarrow AS_{i+2}$$

이로써 공격자 AS_{i+2}에게 되돌아온 M_{j+1}을 버리고, j을 증가 시킨 후 새로운 i_{j+1}을 선택함으로써 공격자는 자신을 악의적인 호스트가 아닌 정상적인 호스트로 위장하는 과정을 마친다. 처리 과정은 다음과 같다.

$$AS_{i+2} \rightarrow i_j : \{M_0, \cdot\cdot, M_{j-1}\}$$

$$i_j \rightarrow i_{j+1} : \{M_0, \cdot\cdot, M_j\}\ i_{j+1} \rightarrow AS_{i+2} : \{M_0, \cdot\cdot, M_{j+1}\}$$

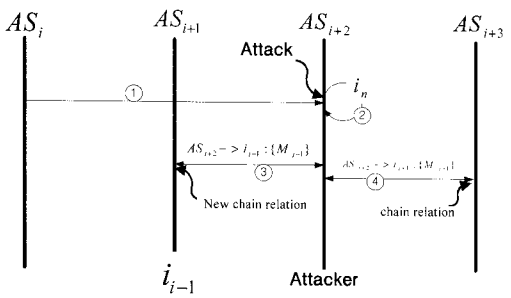

그림 3. 체인절단공격으로 새로운 체인 형성
Fig. 3 new chain relation for chain truncation attack

여기서 M_0가 에이전트 소유자의 서명을 가지고 있기 때문에 체인 관계와 캡슐화를 에이전트 소유자처럼 공격자가 정당하게 요구하게 되는 문제가 발생한다. 이로써 공격자 결정에 따라 에이전트가 이동하고, 서명 체인을 형성하면서 자유로운 이동을 하게 된다. 이 과정은 마지막(원래의 에이전트 소유자) 홉으로 이동되기 전까지 이루어진다. 그리고 에이전트가 이동할 때는 공격자의 서명체인이 아닌 AS_i가 처음 생성한 정상적인 서명 체인을 공격자에 의해 절단공격전에 복사해 두었던 정상적인 서명 체인을 AS_n으로 이동하여 에이전트에 연결한다. 이로써 공격자는 에이전트의 수행임무를 방해하고, 오류정보를 제공하게 만든다. 이를 방지하기 위해 서명 체인을 형성하게 된다. 그러나 서명 체인은 각기 다른 환경의 호스트와 메커니즘이 결정되어야 함으로 공모에 의한 절단공격으로 에이전트 데이터들을 포함하는 정보들에 대한 프라이버시 문제와 안전한 실행이 어려워진다. 에이전트의 안전한 실행 보장과 서명 체인의 절단 공격들로부터 안전한 실행 메커니즘이 필요하다.

Ⅳ. 안전한 에이전트 실행 메커니즘

이동 에이전트의 안전한 실행 메커니즘은 코드 보안 기술로써, 에이전트를 실행하는 호스트 플랫폼으로부터 실행오류를 검출하는 개념으로 최근 들어 추적자의 개념으로 기술되기도 한다. 본 논문에서 사용되는 이동 에이전트 및 암호화 표기는 [표1]과 같다.

검출 기술은 이동 에이전트 코드나 상태가 불법적으로 조작되는 것을 검출하는 것으로 데이터를 변질시킨 악의적인 공격자에게 악의적으로 남용되는 문제 해결을 의미한다. 이를 위해서는 에이전트의 오류 검출 시스템이 필요하다. 에이전트의 오류 검출은 실행 추적 메커니즘에서 실행 및 데이터 무결성을 위해 이루어진다.

표 1. 제안된 프로토콜에 사용된 표기법
[table. 1 notation of proposed protocol

$ENC_{AS_i}^{Pub}$	AS_i의 공개키
$ENC_{TTS_i}^{Pub}$	TTS_i의 공개키
ENC_{AS_i}	AS_i의 비밀키
ENC_{TTS_i}	TTS_i의 비밀키
K_i	임시키
Agt	에이전트
$Cert_{Pk_i}(Ret_i)$	Pk_i 에 의한 에이전트 데이터 인증
Pk_i	에이전트 인증을 위한 키
$Time-stamp$	타임스템프(처리시간을 나타냄)
$Time-out$	실행시간점검
i_A	에이전트가 가지고 있는 유일한 식별자
S_A	에이전트 A의 상태 정보
$T_{(C,Agt)}$	에이전트 실행 후 실행 정보 추적
ACC	에이전트 수락 및 거절 정보
$ENC_{AS_i}(data_i)$	에이전트 정보 데이터
AS_i	i 번째의 에이전트 서버
$ENC_x(Agt_1 k_i)$	에이전트 상태 구조
$C_i, \ldots C_n$	i, n번째의 체크합
C_j	j 번째의 컨테이너
i_j	공격자에 의해 생성된 체인관계
m_i	이동에이전트

본 논문에서는 에이전트의 이동경로에서 데이터의 변질을 막기 위해 생성한 체인을 양자 간 연쇄적인 체인관계로 유지하는 구조이며 기존의 면서 안전한 실행이 이루어질 수 있는 체인관계를 유지하고 있는 구조를 기본으로 기존의 체인관계의 문제점을 개선한 구조이다.

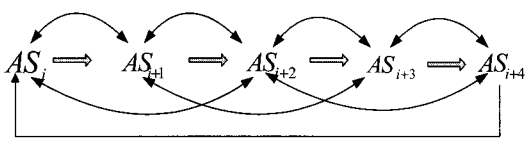

그림 4. 양자간 결연으로 연결된 체인관계
Fig. 4 Bilateral Chain Relationship

네트워크에서 발생하는 시스템 오버헤드와 네트워크의 트래픽을 고려한 안전한 에이전트 실행 제어 처리기(ECp:Execution Control process)가 호스트 내에서 에이전트 실행을 제어와 실행 추적자의 역할을 한다. 에이전트 실행을 인증, 신원확인, 부인봉쇄, 그리고 에이전트 신뢰성을 확보하기 위해 인증서 발행과 체인관계를 생성하여 에이전트 수행결과 데이터를 보호한다. 또한 에이전트의 데이터 암호 키를 생성하는 암호모듈과 인증 가능한 인증모듈을 두어 에이전트의 안전한 실행과 무결성을 보장한다. 그림 5는 에이전트의 안전한 실행과 악의적인 공격자들로부터 절단공격을 보호하기 위해 제안된 실행보장 메커니즘이다.

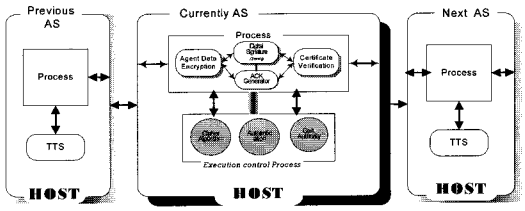

그림 5. 에이전트의 안전한 실행보장 메커니즘
fig. 5 an agents secure execution guarantee Mechanism

4.1 에이전트 서버(AS)와 실행제어처리기(ECp) 간 통신 프로토콜

에이전트의 실행제어(ECp)와 에이전트 서버(AS)간의 통신 프로토콜은 호스트 플랫폼 내에서 이루어진다. 실행과정은 그림 6과 같다.

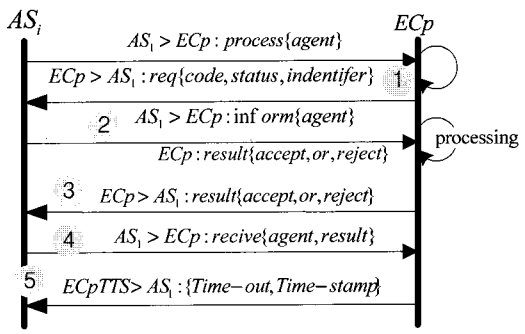

그림 6. 에이전트 서버와 실행제어처리기 간의 통신 프로토콜
Fig. 6 Communication Protocol between AS and ECp

$AS_i \rightarrow ECp : process\{agent\}$: 에이전트 서버(AS)의 호스트에 수신된 에이전트를 처리를 ECp에 처리를 요청한다.

$ECp \rightarrow AS_i : req\{code, statuse, dentifier\}$: ECpS에 전송된 에이전트를 처리하기 위해 에이전트 자원을 요청한다. 즉 에이전트 코드, 에이전트 상태 정보, 에이전트 확인자 등을 전송해 주길 요청한다. 요청을 받아들여지면 에이전트 서버(AS)는 ECp에게 에이전트 정보를 전송한다. 에이전트 정보를 수신한 ECp는 호스트의 에이전트 코드 검증을 통해서 정상적인 에이전트인 경우는 받아들여(accepts)져서 에이전트 임무 수행을 정상적으로 보장한다. 그렇지 않은 경우는 거절(rejects)한다. 이러한 수락과 거절을 결정하는 과정이다.

$$ECp : result\{accept, \text{or}, result\}$$

ECp는 에이전트 서버에게 처리 결과를 보낸다. 결과는 받아들일 것인가 아니면 거절하라는 정보이다.

$$ECp \rightarrow AS_i : result\{accept, \text{or}, reject\}$$

만약 ECp의 처리 결과를 에이전트 서버에게 거절로 전송하면 에이전트는 바로 이전의 호스트나 아니면 실행 가능한 다른 호스트를 찾아 이동해야 한다. 이때 호스트는 다른 에이전트를 수신할 준비를 하여 수신하거나 수신할 에이전트가 없으면 프로토콜 실행을 종료한다. 메시지를 안전하게 수신하였다면 ack를 ECp에 전송한다.

$$AS_i \rightarrow ECp : recive\{agent, result\}$$

에이전트를 수신한 후 악의적인 에이전트나 요소들에 의해 발생할 수 있는 서비스 거부 공격(DoS Attack) 시도를 대비하여, 서비스의 거부를 막기 위한 time-out과 에이전트 서버의 재실행 공격을 막기 위해 $Time-stamp$를 에이전트 서버에 전송한다.

$$ECp \rightarrow AS_i : \{Time-out, Time-stamp\}$$

즉 에이전트의 올바른 수행을 위한 ECp와 AS간에 통신하면서 공격자의 악의적인 행위를 검출(detection)한다. 이때 에이전트서버와 실행제어 처리기의 내부적인 수행 알고리즘은 그림 7, 그림 8과 같다.

```
// Migration to Agent server
  receive : m1 m3
   verification : Digital Signature(Cert_ECPb)
   agent code checking : ENC_Owner(c)
   security execution : (c)
   expiration time checker

// Agent Execution

  if secure agent
      C_i = ENC_AS(data_i)
      E_i = C_i + C_{i+1} + • • + C_n (i ≤ i ≤ n)
      ENC^Pub_ECPb(Cert_PBi-1(AS, Ret_i))
      Ret_i = ((code, E_i, Time-out), Time-stamp, i_A)
    if receive m1
      m1 execution
      Agt transfer to ECp
        ENC^Pub_ECPb(Cert_PBi-1(x), Ret_i)
    if receive m3
      signature verifier : Digital Signature(Cert_ECPb)
      if Acc = Accept
        Agent execution control : T(m_i, Agt)
      m4 response to TTS
      next host migration
    else
      protocol execution termination

   else
      protocol execution awaiting

 else
   m2 refusal
    protocol execution termination
    or waiting execution of ECp
```

그림 7. 에이전트 서버 실행 알고리즘
Fig. 7 Execution algorithm of Agent server

4.2 절단공격으로부터 안전한 에이전트 실행 프로토콜

에이전트는 이질적인 망에서 자신의 제어로 호스트를 옮겨다니면서 다른 호스트의 에이전트 서버와 상호작용이 지속적으로 이루어짐으로 공격자나 악의적인 요인들에 의한 절단공격이 발생할 수 있어 에이전트 실행 보호와 절단공격으로부터 보호되어야 한다. 그림 8과 9는 안전한 에이전트 실행을 위한 실행처리기와 전송 프로토콜이며, 에이전트 데이터를 보호하고 인증하기 위해서 암호 키 체인 관계를 형성하고 있다.

또한 인증서 체인은 악의적인 호스트에 의한 절단공격을 검출하고 에이전트 데이터를 보호하는데 적용된다. 뿐만 아니라 공격자가 절단공격에 실패하여 악의적인 호스트나 에이전트가 다른 악의적인 요인(호스트, 에이전트)들과 공모하여 절단공격을 시도할 수도 있다. 이를 위해 ECp는 인증서를 통해 상호 인증을 하며, ECp가 발행 및 검증이 가능함으로 악의적인 행위를 사전에 방어할 수 있다.

receiving message of Agt : (m_2)
waiting for sequence process

// Agent statues Authentication and Verification
execution ≠ last
Agt Authentication
verification : certificate chain of ECp
$C_i = ENC_{AS_i}(data_i)$
$E_i = C_i + C_{i+1} + \cdots + C_n (i \le i \le n)$
$ENC^{Pub}_{ECp_i}(Cert_{P_{K_{i-1}}}(x), Ret_i)$

// Digital Signature and chain relation
$h = hash(c,\ r)$: *one-way hash function*
$t = h^d \bmod n$: ***RSA signature of h***
$s(x) = x^d \bmod n$: ***RSA signature function***
$f(x) = h^x \bmod n$, $f_{signed}(x) = t^x \bmod n$:
digital signature function pair
$(s \circ f)(x) = s(f(x)) = s(h^x) = (h^x)^d = (h^d)^x = t^x = f_{signed}(x)$

// Certificate Creation
$Cert_{P_{K_i}} = Cert_{P_{K_i}}((s \circ f), Cert_{P_{K_{i-1}}})$
retain trace $T(c, Agt)$ as evidence
record $T(s, Agt) \neq T(Agt)$
Acc = Accept or reject
if Agt request = accept
Agent decision :m_3 transfer
response : Agt
reject →appending reject list

else
execution end and refusal Agt
record $Time-out(Agt)$

그림 8. 실행제어처리기의 수행 알고리즘
Fig. 8 Execution control process algorithm

특히 제안된 메커니즘은 공격자에 의해 임의의 호스트가 악의적으로 이용되거나 남용되는 것으로부터 에이전트를 보호하기 위해 전방위 인증체인 관계 구조를 지니고 있다. 호스트에 에이전트 서버와 ECp간에 신뢰성이 선행되어야 한다. 정상적인 검증과정이 이루어지면 절단공격의 위협으로 안전한 실행이 보장된다. 특히, 기밀성을 높이기 위해서 인증과 서명체인관계를 암호화 모듈을 통해 실행한다.

//Agent Server
$message_2 : AS > TCp : Agreegation\{Agt\}_{Acc}$
$ENC^{Pub}_{TTS_i}\{ENC_{AS_i}\{Agt(Ret_i), i_A, Acc\}\}$
$message_4$
$AS > TCp : encrypt\{Agt\}, chainrelationship$
$ENC^{Pub}_{ECp_i}\left\{\begin{matrix} Cert_{P_{K_i}}(AS_i),\ Time-out, \\ Time-stamp,\ K_i,\ K_i' \end{matrix}\right\}$
$message_5 : AS_i > AS_{i+1} : Migration\{Agt\}$
$ENC^{Pub}_{AS_i}\{Cert_{P_{K_i}}, Ret_i,\ T(c, Agt),\ Cert_{ECp_i}, Acc\}$
$message_6 : AS_i > AS_{i+1} : confirmed\ recive\{Agt\}$
$ENC^{Pub}_{AS_i}\{i_A, Acc\}$

//Execution Control Process
$message_1 : TCp > AS : Request\{Agt\}_{resource}$
$ENC^{Pub}_{AS_i}\{ENC_{ECp_i}\{i_A, ENC_{AS_i}(data_i), ENC_{owner}\{c\}\}, Cert_{ECp_i}\}$
$message_3 : TCp > AS : Result\{Agt\}_{pross}$
$S_i = ENC^{Pub}_{AS_i}\left\{\begin{matrix} Cert_{P_{K_i}}(AS_i),\ K_{i-1},\ Time-stamp,\ i_A, \\ S_A(c, ECp, T(c, ECp),\ Acc) \end{matrix}\right\}$
$ENC^{Pub}_{AS_0}\{Cert_{ECp_i}, S_i\}$

그림 9. 프로토콜의 메시지 전송 프로토콜
Fig. 9 Message transfer sequence of protocol

$message_1$: ECp에 의해 보내진 메시지는 에이전트 확인자 i_A, 에이전트 데이터($ENC_{AS_i}(data_i)$), 그리고 에이전트 코드c 등을 요청을 한다.

이때 ECp의 인증서도 함께 전송하여 상호 인증을 한다. 공격자의 절단공격으로 체인관계를 위장하였다면, ECp의 상호인증으로 공격여부를 판별하게 된다. 그리고 에이전트 데이터를 에이전트 서버의 키를 사용하여 암호화하며, 전체적으로 ECp의 키를 이용하여 통신함으로써 에이전트 서버간 안전한 통신을 한다.

$message_2$: 메시지 $message_1$을 수신 즉시 AS_i는 ECp에게 에이전트의 데이터 $Agt(Ret_i)$, i_A, 그리고 에이전트 수신동의(Acc)를 요구한다.

이때 $code$, E_i, $Time-out$, $Time-stamp$을 ECp의 공개키로 암호화하여 전송을 한다.

$message_3$: ECp는 수신된 에이전트를 수락이나 거절을 결정한다. 만약 TTS에서 accept하지 않으면 AS_i에 수신된 에이전트는 이동 경로에 의해 다음 호스트로 이동한다. 거절이 이루어지면 $Time-stamp$에 의해서 시간이 기록되어 다음에 이동되는 것을 체크한다. 이때 재전송공격등을 방지하며, 절단공격을 시도하여 실행 거절이 이루어진 악의적인 에이전트를 판별한다. 수신된 에이전트를 긍정적으로 수락(Acc=Accept)이 되면 정상적인 프로토콜 실행을 한다. 이때 이전 에이전트 서버(AS_{i-1})는 ECp의 대칭키 암호화 알고리즘에 의해서 생성된 키와 인증서를 가지고 다음 에이전트 서버(AS_{i+1})와 암호 키를 체인 관계를 형성한다.

때문에 악의적인 요인들에 의한 절단공격은 발생할 수 없다. 만약 공격이 발생하더라도 인증과정에서 $T(c, ECp)$을 체크하여 검출된다. 또한 ECp로부터 승인된 메시지를 AS가 수신 하였다면, 에이전트의 서비스 거부 공격(DoS Attack)과 지연 공격(delay attack) 시도를 방지하기 위해 $Time-stamp$ 와 $Time-out$ 정보를 ECp로부터 승인 메시지 정보에 포함한다.

인증된 에이전트는 정상적으로 수행이 이루어지면 에이전트 프로세스 검증과정에서 에이전트 이동경로를 추적하기 위해 각 호스트에서 에이전트의 코드 추적($T(c, Agt)$)을 에이전트에 기록하게 된다. 공격자에 의한 서비스 거부 공격을 방지하기 위해 ECp의 인증서 매개 변수 Pk를 사용한다. 특히 공격자의 절단공격을 시도하기 위해 에이전의 인증서를 절단한 후 공격자가 자신의 새로운 체인을 생성하여 에이전트 임무수행을 방해할 경우 상호인증을 통해 방지 할 수 있으며, $T(c, Agt)$에 의해 악의적인 호스트 체크가 가능하다.

뿐만 아니라 $ENC_{AS_i}^{Pub}\{Cert_{Pk_i}(AS_i)\}$를 이용하여 AS_i의 인증서는 임계값 Pk를 사용하여 디지털 서명을 하였기 때문에 공격자의 절단공격을 시도하려면 먼저 임계값 Pk를 알아야 한다. 그러나 임계값은 에이전트 생성자에 의해 생성되어 사용되기 때문에 에이전트 소유자가 악의적인 공격자가 아니고서는 알 수 있는 방법이 없다.

$message_4$: 에이전트 데이터 암호 키(K_i)를 생성하여 에이전트 데이터를 암호화한다. 그리고 암호 키는 키 체인을 형성한다. 에이전트 데이터는 에이전트의 코드와 유효 기간($Time-out$), 현재 시각 정보($Time-stamp$)를 포함한다.

그리고 현재 호스트의 ECp로부터 발행된 에이전트 서버의 인증서($Cert_{Pk_i}$)를 포함한 에이전트 데이터를 추가 후 에이전트 소유자의 공개키로 암호화한다.

$message_5$: 호스트 내에서 임무 수행을 완료하면 에이전트는 자신의 이동계획에 따라 다음 호스트를 찾아 이동한다. 이때 인증서와 에이전트 데이터의 암호 키 체인 관계와 상호 인증을 위해 인증서 $Cert_{Ecp_i}$를 함께 보낸다. 또한 수신한 에이전트를 받아들일 것에 대한 수락 여부의 메시지를 요구한다.

에이전트를 수신한 호스트(AS_{i+1})는 에이전트를 보내 준 호스트(AS_i)에 적절한 시간 내에 도달하는 것을 보증하기 위해 시각 정보를 사용하여 유효성을 체크한다. 호스트 간에 서로의 신뢰를 확인하기 위해 CA의 인증서를 통해 서로의 가용 능력을 상호 인증한다. 현재의 프로토콜과 에이전트의 확인자를 포함한 에이전트가 정상적으로 승인되었음을 AS_i에게 응답한다.

이로써 공격자에 의한 악의적인 남용되는 호스트를 방지하고 ECp에 의한 에이전트의 안전한 실행을 보장한다. 본 논문의 기여도는 다음과 같다. 첫째, 디지털 서명을 위해 임계값 매개 변수를 포함한 공개키 암호화 서명으로 견고성을 강화한다. 또한 인증서 체인을 형성하기 위해 해시함수를 사용하였다. 이로써 에이전트의 무결성과 절단공격으로부터 안전한 에이전트 실행을 보장한다. 둘째, 에이전트 데이터의 암호화 속도 향상을 위해 ECp의 대칭키 암호에 의한 키생성과 데이터 암호화하여 기밀성을 보장한다.

셋째, 에이전트의 실행을 추적할 수 있도록 하였으며, 절단공격의 발생을 방지하기 위해 인증체인관계를 구축하였다.

넷째, 에이전트 수행되는 프로토콜에서 발생 가능한 악의적인 호스트에 의해 선의의 에이전트가 악의적으로 남용되는 것을 방지하기 위해 자유로운 순회(free roaming)에서 한번 수행한 호스트를 다시 방문하지 않도록 한다.

이는 인증서 체인과 에이전트 데이터의 암호 키를 이

용하여 방문한 흔적이 있는 호스트는 다시 접근하지 않도록 하였다.

V. 안전성에 대한 분석

에이전트의 안전한 실행환경은 악의적인 호스트로부터 에이전트뿐만 아니라 호스트의 악의적 남용으로부터 보호할 수 있다. 이를 위해 본 논문에서는 인증서의 체인관계연결을 통해 인증서 체인 절단 공격을 검증하고, 에이전트 실행 코드를 추적하여 코드 분석을 통해 악의적인 공격자(호스트)를 찾아낼 수 있도록 제안하였다.

이때 인증서 체인은 인증서를 호스트의 H_{i-1}에서 전달된 인증서와 체인을 형성한다.

그러나 체인관계를 끊어 공격자가 자신이 생성한 체인관계를 새롭게 형성하여 변형된 체인으로 에이전트의 임무 수행을 방해하거나 악의적으로 목적수행을 위해 인증서 절단 공격이 발생할 수 있다. 이러한 공격을 방어하기 위해 인증서 $Cert_{Pk_{i-1}}$가 공격자에 의해 해독이 된다고 가정하였을 때 $Cert_{Pk_{i-1}}$는 정당한 호스트로 위장을 하기 위해서는 AS_{i-1}의 에이전트 서버가 있는 호스트의 인증서와 체인관계를 알기위해서는 서명생성에 사용한 AS_{i-1}의 개인키를 알아야 함으로 공모를 하지 않고서는 어렵다. 그러나 공모는 현실적으로 가능함으로 그림 10과 같이 연쇄적 키 체인을 형성함으로써 이러한 공모는 방지할 수 있다. 또한 암호화 모듈을 통해서 암호화가 이루어져 암호 해독이 선결되어야 함은 물론 공격자는 절단공격을 성공하기 위해서는 서명과 암호화를 위해 암호모듈의 $f^{-1}_{signed}(x)$를 알아야 한다.

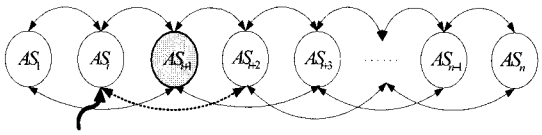

그림 10. 절단공격으로부터 안전성 분석
Fig. 10 security analysis from truncation attack

이를 위해서는 송신자의 비밀 키를 해독하기 위해 소수 p와 q를 알아야 한다. 그러나 소수 p와 q를 생성할 수 없다. 이는 송신자만이 p와 q알고 있기 때문에 소수 p와 q를 곱하여 비밀 키를 생성할 수 있기 때문이다. 그러므로 공격자에 의한 인증서 체인 공격은 성공적으로 이루어지지 못한다. 만약 이러한 악의적인 요소들이 공모를 하여 인증서 절단공격을 시도할 경우가 발생할 수 있다.

그러나 본 연구의 기본 메커니즘은 연쇄적으로 두 개의 이전 호스트와 다음 두 개의 호스트와 연쇄적인 체인관계 형성이 가능하기 때문에 안정성이 보장된다. 즉 AS_{i+1}과 AS_{i+3}이 악의적으로 공모를 하여 절단공격 시도를 하려고 한다면 AS_{i+2}에 만들어진 에이전트 실행 데이터를 버릴 수 가 없다. 실제적으로AS_i의 다음이 AS_{i+1}이며, 이를 부인할 수 없기 때문이다. AS_{i+2}가 아니라는 것은 AS_i에서 만들어진 데이터를 통해 알 수 있기 때문이다. 이로써 공격자에 의해 인증서 체인을 끊는 절단공격 수행은 실제적으로 어렵다. 인증서로 새로운 체인을 형성하여 정직한 호스트를 악의적인 호스트로 남용하는 것으로부.터 에이전트 실행을 안전하게 보호된다.

이 과정들은 정직한 호스트가 악의적인 호스트가 되어 자신의 의도와는 상관없이 악의적인 행위를 하게 되는 경우이다. 이러한 경우 또한 절단공격과 마찬가지로 기본 프로토콜이 연쇄적 상호 체인관계 메커니즘을 형성하고 있기 때문에 현실적으로 불가능하다. 또한 견고한 체인 관계 형성을 위해 암호화된 체인 관계를 임계값을 포함하는 디지털 서명을 하여 안전한 에이전트 수행을 보장하였다.

VI. 결 론

에이전트 기술의 응용은 사용자 위주의 응용 기술에서 많이 이용되고 있는 분산 컴퓨팅환경에서 새로운 패러다임이다. 그러나 에이전트가 안전하게 실행될 수 있도록 에이전트 보안과 에이전트 안전한 실행이 선행되어야 한다.

본 논문에서는 에이전트가 임무 수행을 위해 이동하는 동안 하나 이상의 호스트와 매핑을 하기 때문에 호스

트의 악의적인 공격자로 부터의 안전한 에이전트 실행 보장과 공격자의 절단공격으로부터 에이전트 안전한 실행을 보장하는 실행 메커니즘을 제안하였다. 뿐만 아니라 공격자에 의해서 선의의 호스트가 악의적으로 남용되는 행위를 방지할 수 있는 보안 메커니즘을 제안하였다. 그리고 보안 위협과 공격자들로 부터의 공모에 의한 절단공격을 방지하고, 에이전트의 안전한 임무수행을 위한 메커니즘을 통해 검증 시간의 단순성과 유용성을 보장하였다. 제안한 실행 추적 프로토콜 구조는 메시지 전송 시퀀스를 통해 분석하였고, 보안의 안전성에 대한 분석을 위해 절단공격 분석과 새로운 체인 생성공격에 대한 보안 분석을 하였다.

공격자의 절단공격이 다자간의 공모에 의해 이루어질 경우 에이전트가 안전하게 실행될 수 있는 보안 프로토콜 설계가 요구되며, 최적의 이동 경로로 공모에 의한 공격 회피가 이루어지는 보안 메커니즘 연구가 필요하다.

Acknowledgment

본 지식재산권은 지식경제부 및 정보통신연구진흥원의 지원을 받아 수행된 연구결과임
(09-기반, 산업원천기술개발사업)

참고문헌

[1] G. Karjoth, N. Asokan, and C. Gulcu, "Protecting the Computation Results of Free-Roaming Agents," K. Rothermel and F. Hohl(Eds.) in Proceeding of MA'98 Mobile Agents, LNCS 1477, Springer-verlag, pp.195-207, 1998.

[2] F. Hohl, "A Model of Attacks of Malicious Hosts Against Mobile Agents," in Proceeding of the ECCOP Workshop on Distributed Object Security and 4th Workshop on Mobile Object System : Secure Internet Mobile Computations, pp.105-120, 1998.

[3] G. Karjoth, "Secure Mobile Agent-based Merchant Brokering in Distributed Marketplaces," in Proceeding ASA/MA 2000 (D. Kotz and F. Mattern, eds.), vol. 1882 of Lecture Notes in Computer Science, pp. 44-56, Berlin Heidelberg: Springer-Verlag, 2000.

[4] C. David, G. BenjaMin, H. Colin, and L. David, "Itinerant Agents for Mobile Computing," in journal of IEEE Personal Communications, Vol. 2, No. 5, pp.34-49, October 1995.

[5] A. Corradi, R. Montanari, and C. Stefanelli, "Mobile Agents Protection in the Internet Environment," in The 23rd Annual International Computer Software and Applications Conference(COMPSAC '99), pp.80-85, 1999.

[6] V. Roth, and V. Conan, "Encrypting Java Archives and its Application to Mobile Agent Security," in Agent Mediated Electronic Commerce: A European Perspective, LNAI 1991, pp.232-244, Springer-Verlag, 2001.

[7] N. M. Karnik, and A. R. Tripathi, "Security in the Ajanta Mobile Agent System", Technical Report TR-5-99, University of Minnesota, Minneapolis, MN 55455, U.S.A., May 1999.

[8] A. C. Snoeren, C. Patridge, A. Sanchez, E. Jones, F. Tchakountio, S. T. Kent, and W. T. Strayer, "Hash-based IP Traceback," in Proceeding of SIGCOMM '2001, August 2001.

[9] A. Carzaniga, Gian P. Picco, and G. Vigna, "Is Code Still Moving Around? Looking Back at a Decade of Code Mobility," in Proceedings of 29th the International Conference on Software Engineering (ICSE '2007), may 2007.

[10] Xiang Tan, Yuqing Gu, Chongming Bao, "A method for mobile agent data protection," Journal of Software, China, vol.16, No.3, pp.477-484, 2005.

[11] Sreedevi R. N., Grrta U.N., U.P.Kuikarni, A.R.Yardi, "Enhancing Mobile Agent Application with Security and Fault Tolerant Capabilities," 2009 IEEE International Advance Computing Conference (IACC 2009), pp.992-996, India, March 2009.

저자소개

정창렬(Chang-Ryul Jung)

1999년 순천대학교 컴퓨터교육과 (석사)
2005년 순천대학교 컴퓨터과학과 (박사)

2005년~2007년 순천대학교대학원 컴퓨터과학과
※ 관심분야 : Security, RFID/USN, RFID Privacy

이성근(Sung-Keun Lee)

1987년 고려대학교 전자공학과 (석사)
1995년 고려대학교 전자공학과 (박사)

1997년~현재 순천대학교 멀티미디어공학과 교수
※ 관심분야 : 유비쿼터스 센서 네트워크, 멀티미디어 통신, 인터넷 QoS