情報保護學會論文誌
第17卷 第4號, 2007. 8

무선 센서 네트워크에서의 메모리 공간 삭제를 이용한 선행 코드-검증 기법

최 영 근,[1†] 강 전 일,[1] 양 대 헌,[1‡] 이 경 희[2]
[1]인하대학교 정보보호 연구실, [2]수원대학교 컴퓨터공학과

Proactive Code Verification Protocol Using Empty Memory Deletion in Wireless Sensor Network

Young-geun Choi[1†], Jeonil Kang[1], DaeHun Nyang,[1‡] KyungHee Lee[2]
[1]Information Security Research Laboratory, INHA University,
[2]Department of Electrical Engineering , The University of Suwon

요 약

일반적으로 무선센서네트워크에서의 인증은 엔티티의 인증을 의미하나, 센서 네트워크에서는 데이터 집약적 특징에 의해 센서 노드의 코드를 증명하는 것이 더욱 의미 있는 인증기법이라 할 수 있겠다. 가장 기본적인 검증 기법은 검증자(Verifier)가 자신이 기존에 가지고 있던 목표 노드의 메모리 내용을 현재 목표 노드가 가지고 있는 실제 메모리 내용과 비교하여 차이점을 발견할 시, 노드의 감염으로 인식하는 것이다. 본 논문에서 우리는 기존 기법의 약점을 서술하고 대안 기법을 제안할 것이다. 이 기법은 SWATT[1]와 달리 목표 노드의 전체 메모리 영역을 쉽게 검증할 수 있고 정확한 시간 정보에 의지하지 않고서도 악의적 코드가 자신을 은폐시킬 확률을 현저히 떨어뜨린다. 우리는 새로운 방식을 제안하고 여러 환경에서의 성능 평가 결과를 보여 줄 것이다.

ABSTRACT

The authentication in WSN(Wireless Sensor Network) usually means the entity authentication, but owing to the data centric nature of sensor network, much more importance must be put on the authentication(or attestation) for code of sensor nodes. The naive approach to the attestation is for the verifier to compare the previously known memory contents of the target node with the actual memory contents in the target node, but it has a significant drawback. In this paper, we show what the drawback is and propose a countermeasure. This scheme can verify the whole memory space of the target node and provides extremely low probability of malicious code's concealment without depending on accurate timing information unlike SWATT[1]. We provide two modes of this verification method: BS-to-node and node-to-node. The performance estimation in various environments is shown.

Keywords : *Wireless Sensor Network, Authentication, Code Attestation*

접수일: 2007년 3월 6일; 채택일: 2007년 6월 13일
* 본 연구는 한국과학재단 특정기초연구(R01-2006-000-10957-0)지원으로 수행되었음.
† 주저자, choizak@seclab.inha.ac.kr
‡ 교신저자, nyang@inha.ac.kr

Ⅰ. 서 론

무선 센서 네트워크에서 보안은 가장 중요한 이슈중

하나이다. 군사 감시 용도로 쓰이는 무선 센서 네트워크를 가정해보자. 노드가 적에 의해 포획당하여 공격 용도로 쓰이게 된다면 이런 노드들은 적의 침입을 감지해낼 수 없고, 심지어는 아군의 패배를 초래하는 거짓 정보를 보고하게 할 수도 있다. 이러한 센서 노드들의 비정상적 행위를 막기 위해, 의심스러운 노드를 증명하는 방법이 필요하다.

불행히도 노드에 대한 물리적 공격은 어떠한 방법으로도 막을 수 없다. 오직 노드 내부의 수정을 물리적으로 강제하는 하드웨어 모듈만이 포획 노드가 다른 노드에게 피해를 주는 악의적 노드로 변하는 것을 방지할 수 있을 뿐이다. 따라서 지금까지의 연구는 대부분 어떻게 오염된 노드를 발견할 수 있는지에 대해 초점을 두어왔다. 대부분의 이러한 연구에선 오직 오염된 노드가 비정상적 행위를 하고 난 뒤에서야 그 오염된 노드를 발견 할 수 있다. 따라서 노드의 비정상행위 전에 악의적 노드를 찾을 방법은 없었다. 이러한 이유 때문에 WSN 은 악의적 노드에 의한 피해에 취약하였다.

따라서 우리는 악의적 노드가 비정상적 행위를 수행하기 전에 코드의 이상여부를 알아낼 수 있는 사전 탐지 기법이 필요하다. SWATT(SoftWare-based Attestation for embedded device)[1]은 이러한 소프트웨어 기반 증명을 위해 고안되었다. SWATT 는 매우 간단하게 적용될 수 있지만 정확한 시간을 기반으로 동작하기 때문에 예상할 수 없는 네트워크 지연에 민감하고 하드웨어 플랫폼에 의존적이다. 이 기법은 후에 Mark Shaneck에 의해 시간 기반의 약점을 개선하여 새롭게 제안되었다.[2]

이 논문에서는 오염된 노드를 탐지하는 완전히 새로운 사전 코드-검증 기법을 제안한다. 본 기법은 목표노드의 전체 메모리 내용을 검증하고 SWATT[1]와는 달리 정확한 시간 정보에 의지하지 않고서도 악의적 코드가 자신을 은폐시킬 확률을 현저히 떨어뜨린다. 이 논문에서는 새로운 방식을 제안하고 여러 환경에서의 기능 평가 결과를 보여 줄 것이다.

Ⅱ. 가정, 공격 모델 및 필요사항

2.1. 가정

이 논문에서는 검증자가 목표노드의 메모리 내용의 사본을 가지고 있다고 가정한다. 각각의 엔터티는 서로간의 안전한 통신 채널을 구축하기위해 키 쌍을 분배한다. 이 키 쌍은 무작위 키 선-분배 기법 등에 의해 설정될 수 있다. 그리고 검증자가 원격으로 실행 가능한 증명코드가 목표노드의 메모리 안에 탑재되어 있다고 가정한다. 공격자는 포획한 노드의 메모리 접근에 대한 모든 권한을 가지고 있다. 하지만 그 외의 하드웨어 구조에 대한 수정 가능성은 공격자가 외부 자원을 이용할 수 있다는 의미가 됨으로 이 논문에선 논외로 한다. 우리는 프로그램 가능한 플래쉬 메모리는 특별한 목적이 없는 한, 어떠한 빈 공간도 가지고 있지 않는다고 가정한다. 또한 노드는 가상 메모리를 지원하지 않으며 데이터 스토리지에서는 직접적으로 코드를 실행시킬 수 없다고 가정한다.

2.2. 공격모델

검증 기법의 가장 원시적 기법은 검증자가 목표 노드의 실제 메모리 내용과 자신이 알고 있는 메모리 내용을 비교해보는 것이다. 이 기법에서 검증자는 먼저 목표노드의 증명을 요구하는 메시지를 보낸다. 그러면 목표노드는 자신의 메모리에 있는 증명코드를 실행한다. 이 증명코드는 노드의 메모리 내용에 접근하여 그 내용을 검증자에게 전송한다. 마지막으로, 검증자는 수신한 내용을 자신이 가지고 있는 복사본과 비교한 후 목표노드의 감염여부를 결정한다.

그러나 공격자는 다음과 같은 공격기법을 사용함으로서, 포획한 노드 내부에서 악의적으로 수정된 펌웨어 코드가 이러한 원시적인 검증 기법에 의해 발견 되는 것을 쉽게 피할 수 있다. 우선, 공격자는 원래 메모리 내용에 대한 유효한 정보를 가지고 있는 공격코드를 센서 노드의 빈 공간에 심어 놓는다. 그리고선 노드 내의 증명코드가 실제 메모리 내용에 접근하는 대신에 심어 놓은 유효정보에 접근하도록 수정한다. 그러면 증명코드는 악의적으로 수정된 펌웨어의 정보가 아닌 노드가 공격받기 이전의 유효한 메모리 정보 값을 전송할 것이다. 결과적으로, 검증자는 목표노드의 감염여부를 정확히 검증해내지 못한다.

Ⅲ. 제 안

이 장에서 우리는 소프트웨어 기반 검증 방법에 대한 새로운 기법을 제안할 것이다. 이 기법의 기본 동작 원

리는 검증자의 요청이 있을 때마다 목표노드의 메모리 안에 악의적 코드가 동작할 공간을 남기지 못하도록 만든 후 전체 메모리 내용의 값을 전달하는 것이다. WSN은 일반적으로 대규모의 센서들과 몇 개의 BS(Base Station)으로 구성되어 있다. BS 가 여러 업무를 동시에 처리할 수 있을 정도의 충분한 자원을 소유하고 있는 반면에 센서 노드는 작은 메모리 크기, 적은 배터리 용량, 좁은 라디오 범위 등 매우 제한된 자원을 가지고 있다. 본 단원에서는 BS가 검증자로서 센서 노드의 정상 동작 여부를 증명할 수 있는 방법을 논의한다.

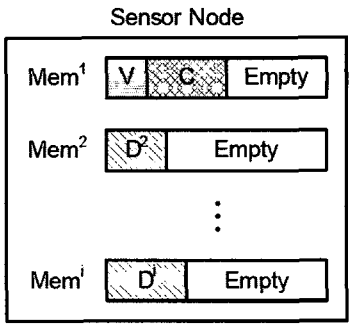
3.1. 표기법

본 논문에서 다음과 같은 표기를 사용한다.

$V \| D$	코드 V, D 간의 연결.
$D_i \|\|_{i=1}^{m}$	D_1부터 D_m까지의 연결.
$K_{A,B}$	노드 A, B 간에 분배된 키.
$E_{K_{A,B}}(m)$	메시지 m을 키 $K_{A,B}$로 암호화.
$D_{K_{A,B}}(c)$	암호문 c를 키 $K_{A,B}$로 복호화.

3.2. 코드 검증 기법

센서 노드는 여러 종류의 메모리를 가지고 있다. 예를 들어 UC Berkeley에 의해 개발된 MICA mote는 4kB의 고정 RAM, 128kB 프로그램 가능한 플래쉬 메모리와 4Mbit의 외부 플래쉬 메모리를 가지고 있다.[3] 대부분의 센서 노드 구조는 MICA mote와 유사하다. 우리는 마이크로 컨트롤러의 SRAM을 Mem^1로 표시할 것이다. 그 외 Mem^i는 플래쉬 프로그램 메모리나 외부 데이터 저장소를 나타낸다. 보통 오직 한 개의 프로그램 가능한 플래쉬 메모리가 센서노드의 마이크로 콘트롤러 안에 내장되고 외부의 데이터 저장소는 EEPROM 이다. 여기서는 외부 데이터 저장소를 이용한 공격은 염두해두지 않는다. 그 이유는 외부 데이터 저장소를 공격 목적으로 사용하기에는 접근시간이 너무 길어서 쉽게 발견할 수 있기 때문이다. 센서 노드가 배치된 후에, 증명코드 V와 펌웨어 코드 C는 노드가 동작하는 동안 Mem^1에 상주하게 된다. Mem^i는 각각의 데이터 파트 D^i를 가지고 있으므로, 하나의 노드가 가지고 있는 총 데이터는 $D=\{D^i\}_{i=1}^{\Lambda}$ 이다. 여기서 Λ는 하나의 센서 노드가 가지고 있는 모든 메모리 개수를 의미한다. [그림 1]은 이러한 메모리 맵을 나타낸다.



[그림 1] 센서노드의 메모리 내용. Mem^1은 마이크로 콘트롤러의 RAM을, $Mem^i(i>1)$은 프로그램 가능한 플래쉬 메모리를 나타낸다.

3.2.1. 요청단계

우리의 검증 메커니즘은 4가지 단계로 구성되어 있다. 검증자의 요청단계와, 목표노드의 연산단계, 응답단계, 그리고 다시 검증자의 검사단계가 있다. 먼저 목표노드의 메모리 내용을 검증하기 위한 요청 단계에서, 검증자인 BS는 요청 메시지헤더(Req)와 BS의 ID(ID_{BS}), 그리고 ID_{BS}와 nonce(n)를 BS와 노드 간의 분배된 안전키($K_{BS,node}$)로 암호화하여 이 세 가지를 함께 목표노드에게 전송한다. 목표노드가 이들 메시지를 수신하게 되면 노드는 $E_{K_{BS,node}}(n, ID_{BS})$를 복호화하여 ID_{BS}와 비교함으로써 메시지의 합법성 여부를 판단한다. 이를 통해 공격자가 분배 키 $K_{BS,node}$를 알지 못하는 한, 공격자는 BS로 위장할 수 없으므로 특정 노드에 대한 서비스 거부 공격(Denial of Service)을 방지할 수 있다.

$$BS \rightarrow node : Req, ID_{BS}, E_{K_{BS,node}}(n, ID_{BS})$$

3.2.2. 연산단계

검증자의 요청 단계에 이어 목표노드의 연산단계에서, 목표노드는 n을 시드로 하여 난수를 생성한다. 그리고 목표노드 내 메모리의 빈 영역은 이렇게 생성된 난수를 이용한 특정 알고리즘의 결과 값인 비트 열로 채워진다. Mem^i의 빈 공간의 크기인 S_e^i는 증명코드, 펌웨어 코드, 데이터 영역을 제외한 빈 메모리 영역의 크

기를 의미한다. 이는 다음처럼 계산할 수 있다.

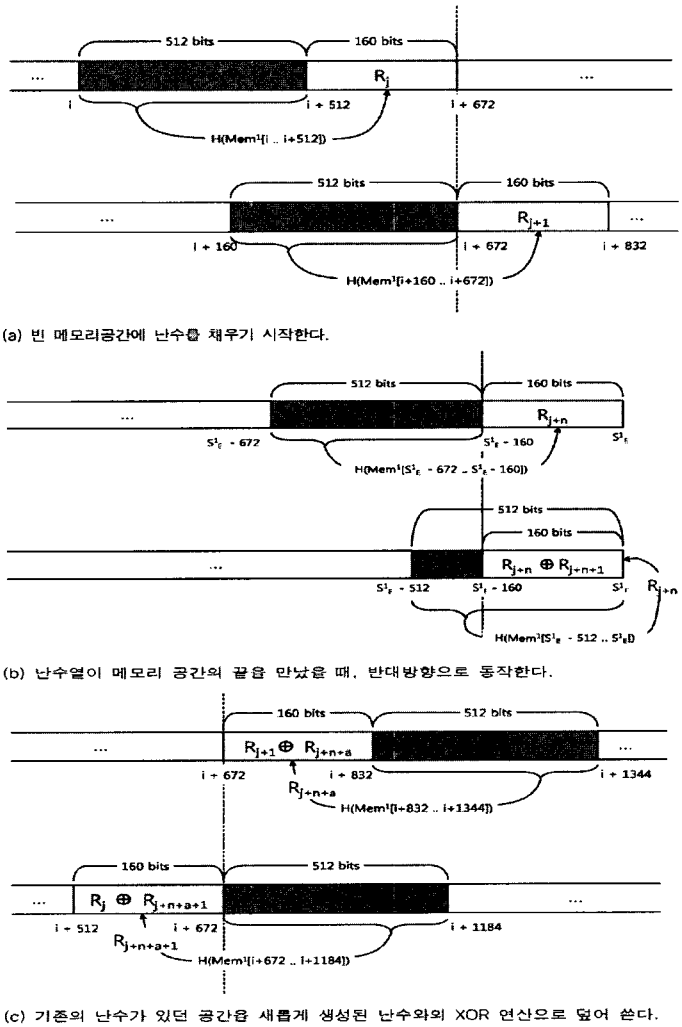
$$S_e^i = S^i - (S_V^i + S_C^i + S_D^i) \quad (1)$$

본 단계에서 사용하는 알고리즘은 메모리의 빈 공간을 두개의 난수 값의 XOR연산으로 채우는 기능을 수행하게 된다. 먼저, 난수는 S_e^i를 채운다. 그리고 이러한 난수열이 빈 메모리 공간을 다 채우게 되면 반대 방향으로 동작하면서 기존의 난수가 있던 공간을 새롭게 생성된 난수와의 XOR 연산으로 덮어쓰게 된다. 이 알고리즘에서 사용하는 해쉬 함수의 입력 값은 이전에 채워진 512-bits의 메모리 값을 사용한다. [그림 2]는 본 알고리즘의 동작과정 예를 보여준다.

그러나 이 알고리즘이 4번째 라운드까지 동작하기 이전에는 해시 입력을 위한 512-bits의 메모리 값이 존재하지 않으므로, 첫 번째 라운드에서의 난수 값은 단지 시드 n에서부터 생성되고 두 번째부터 네 번째 라운드까지의 연산은 메모리 값의 빈 부분을 0으로 채운 값을 입력으로 한다. 이 알고리즘을 사용하는 이유는 4.3절에서 다룰 것이다. 지금까지 설명한 증명단계의 알고리즘 Fill_Memory(n)은 다음과 같다.

3.2.3. 응답단계

그 다음, 목표노드의 응답단계에서 해당 노드는 자신



[그림 2] 연산 단계에서 두 난수의 XOR 연산 값으로 메모

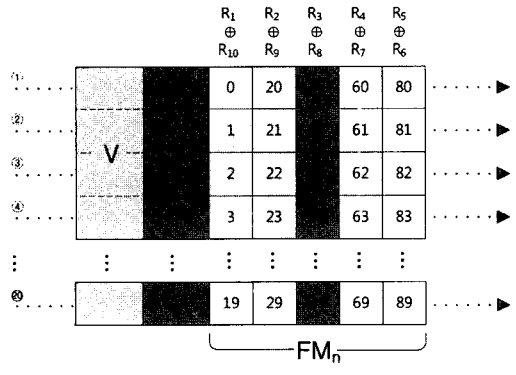
```
Algorithm Fill_Memory(n)
1  for i=1 to Λ
   // Filling the empty memory
   //  with random number
2    for j=1 to 4
3      n←h^512(n||padding 0)
4      Mem^i[j-1]←n
5    for j=5 to S_e^i
6      Mem^i[j-1]←h^512(previous 512-bit)
       // XOR operation in
       //  the opposite direction
7    for j=1 to S_e^i
8    Mem^i[S_e^i-j]←Mem^i[S_e^i-j]⊕h^512(previous 512-bit)
9    return
```

〔그림 3〕 Algorithm Fill_Memory( )

의 메모리 전체의 해쉬값을 응답으로 하여 검증자인 BS에게 보내게 된다. 이 해쉬값은 Mem^1부터 Mem^i까지의 h(MIXED_MEM ‖ D)이다. 여기서 MIXED_MEM은 $V \| C \| FM_n$을 특정 규칙에 의거하여 읽은 메모리 값의 열이다. MIXED_MEM으로 표현한 이런 혼합화는 공격자가 포획한 다른 노드를 이용하는 등의 방법으로 FM_n전체의 해쉬 값을 미리 사본으로 저장해 놓는 것을 방지한다. 〔그림 4〕는 위 설명의 간단한 예이다.

$$node \rightarrow BS: E_{K_{BS\text{-}node}}(h(MIXEDMEM \| D, D))$$

BS는 노드가 임시적으로 보관하고 있는 데이터 D를 모르기 때문에, 일반적으로 노드는 증명 단계 이전에 BS 에게 D를 보내야 한다. 그리고 목표 노드의 메모리

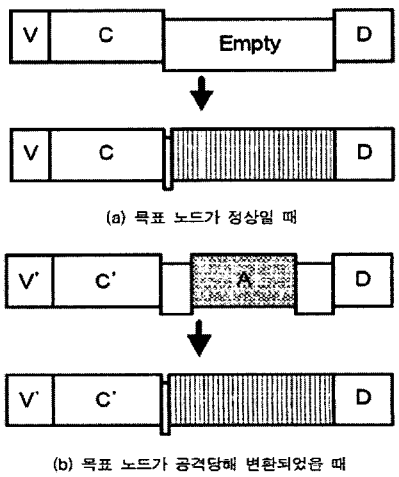


〔그림 4〕 MIXED_MEM(SHA-1 사용)

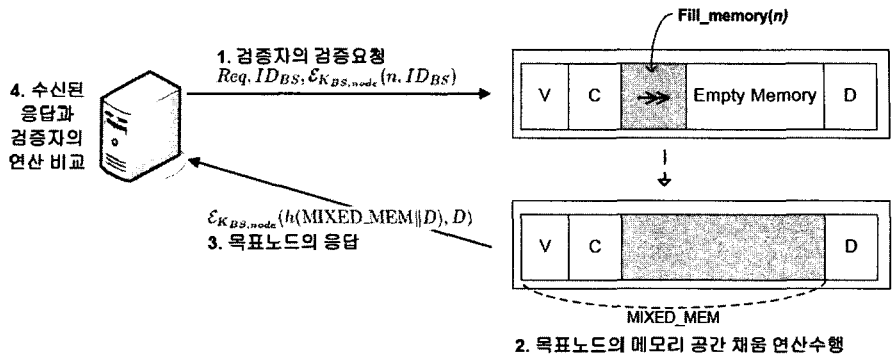
안의 V와 C를 제외한 모든 공간을 난수로 채운 뒤, 메모리의 해쉬값으로 응답한다. 응답 단계가 끝나면 BS는 자신이 보관하고 있던 D를 본래의 자리인 노드에게로 되돌려줘야 한다. 하지만 이 방식은 두 가지 이유로 부담이 되는데, 첫 번째 이유는 노드의 제한된 배터리 용량에 기인한다. 노드는 데이터 D를 송신, 수신해야하기 때문에 에너지를 두 번 소비하게 된다. 결과적으로 노드는 검증을 위하여 자신이 원래 가지고 있던 D를 지우고, 검증이 끝난 후엔 그것을 다시 얻기 위하여 불필요한 에너지 낭비를 하게 되는 것이다. 다른 이유는 불필요한 증명 시간의 증가이다.

3.2.4. 검사단계

이 논문이 제안하는 기법에서 사실상 노드는 D를 다시 수신 받아야할 필요가 없다. 응답 단계에서 노드는 단지 D를 해쉬값과 함께 BS에게 보내기만 하면 되는 것이다. 그러면 BS는 이미 자신이 알고 있는 목표노드의 V, C와 함께 D도 알게 된다. 참고로 노드가 D를 단지 임시로 사용하기 때문에 굳이 D를 저장할 필요가 없을 경우엔 노드는 단지 해쉬값 $E_{K_{BS\text{-}node}}(h$(MIXED_MEM))만 BS에게 전송하면 된다. 마지막으로 검증자의 검증 단계에서 BS는 목표노드로부터 수신한 해쉬값과 자신이 가지고 있는 목표노드의 메모리정보로 계산한 해쉬값을 비교한다. 만일 이 둘이 일치하지 않다면 BS는 목표노드가 공격자에 의해 오염되었거나 불법적으로 수정되었다고 최종적으로 판단하게 되고, 다른 노드들에



〔그림 5〕 Fill_Memory() 알고리즘 전후의 메모리 내용 변화



〔그림 6〕 검증기법의 4단계에 대한 전체적인 설명

게 이를 공지할 것이다.

〔그림 5〕는 본 기법의 연산단계 전후 상황을 간략하게 보여준다. 만일 목표노드가 오염되지 않았다면 노드는 전체 메모리 이미지를 해쉬한 유효한 값을 전송할 것이며, 목표노드가 오염되었고 공격자가 유효한 값을 응답하기 위한 공격코드를 심어놓았더라도 난수가 채워짐으로 인하여 공격코드를 덮어쓰게 된다. 결과적으로 오염된 노드는 검증자인 BS에게 유효한 값을 전송할 방법이 사라지게 된다. 〔그림 6〕은 지금까지 설명한 기법의 4단계에 대한 전체적인 그림이다.

Ⅳ. 보안 고려사항

4.1. 시간에 민감하지 않은 검증

WSN에서 SWATT는 악의적 코드를 검출해내는데 있어 연산 시간에 대한 면밀한 측정을 필요로 한다. 만약 노드가 더 강력한 마이크로프로세서를 사용한다면 연산 시간의 차이를 증폭시키기 위하여 검증의 더 많은 반복실행을 필요로 한다. 또한 이것은 노드가 많은 에너지를 소비하게 될 것이라는 것을 의미한다. 또 SWATT는 예상할 수 없는 시간 지연을 고려하기 힘들다. 만약 네트워크 지연시간 등을 고려하게 된다면, 공격자는 검증자가 공격에 필요한 연산 시간의 증가를 네트워크 지연시간으로 인식하게 만들 수 있다. 따라서 WSN에서의 SWATT는 매우 신중히 사용되어야한다.

반면에, 본 코드-검증 기법은 이러한 연산시간의 정확한 측정을 필요로 하지 않는다. 대신에 정확한 응답을 위한 메모리 공간을 필요로 하는데, 만약 센서 노드가 많은 메모리 공간을 가지고 있다면 빈 메모리 공간을 채우기 위한 시간을 필요로 하게 된다. 우리는 메모리의 크기에 비례하는 검증연산시간과 네트워크 지연시간을 합한 적절한 응답한계시간을 둠으로써 이후에 설명할 여러 가지 공격을 예방할 수 있다. 또한 본 기법에서 네트워크의 지연시간이 목표노드의 감염여부를 결정하는데 끼치는 영향은 미비하기 때문에 시간에 기반하지 않는 유연성을 제공한다.

4.2. 재전송 공격 방어

공격자에 의하여 오염된 노드는 재전송 공격을 할 수 있다. 이 노드는 이전에 다른 노드의 검증절차에 쓰였던 유효한 메시지를 도청할 수 있다. 그러나 본 기법은 기본적으로 challenge-and-response 방식이므로 설령 공격자가 다른 노드와 검증자간의 세션 키를 획득하게 되더라도 nonce의 특성상 재전송 공격은 불가능하다.

challenge-and-response 방식에서 만약 공격자가 두 엔티티간의 비밀을 알고 있다면 그는 이러한 비밀을 계산한 응답 메시지를 전송함으로써 쉽게 공격을 성공시킬 수 있다. 본 기법에서 비밀은 코드 그 자체이다. 사실상 코드는 공격자에게 쉽게 노출될 수 있기 때문에 본 프로토콜에 대한 공격은 쉬워 보인다. 그러나 우리의 기법에서 공격자는 악의적 코드와 정상 코드 모두를 유지할 메모리 공간을 가질 수 없다. 그러므로 오염된 노드에서 nonce를 사용한 유효한 응답을 도출해내는 것은 불가능하다.

4.3. 필요한 난수 블록을 직접 계산하는 공격과 그 방어

다른 공격 방법은 증명, 응답단계에서 악의적 코드가

해쉬값을 계산하는데, 필요한 시점에서 랜덤 시퀀스의 블록을 만들어내는 것이다. n을 빈 메모리 블록의 개수라고 본다면, n번째 블록은 $R_n \oplus R_{\lceil |S_e^i|/S_{ho} \rceil \times 2-n}$이기 때문에 시드로부터 $\lceil |S_e^i|/S_{ho} \rceil \times 2-n$번의 계산 값으로 볼 수 있다. 여기서 S_{ho}는 해쉬 함수의 출력 값의 크기이고 $|\ |$는 특정 메모리의 크기를 나타낸다.(해쉬 함수가 SHA-1이라면 S_{ho}는 160이 되고, MD5라면 128이 된다.) 빈 메모리 영역은 Fill_Memory(n)가 동작하는 동안 난수 값을 두 번 채우게 됨을 상기하라. 유사하게 $(n-1)$번째 블록 또한 생성할 수 있다.

만약 우리가 총 메모리의 길이가 공격코드에 비교하여 월등히 크다고 가정한다면, 공격 코드는 자신이 상주한 메모리 위치에 정상 코드가 검증받을 때와 비교하여 최소한 20배(SHA-1)에서 16배(MD5)의 해쉬 연산을 수행하게 된다. 그러나 총 메모리 길이 대 공격코드의 길이의 비율은 우리의 가정보다 크다. 그 때문에 우리는 보다 많은 시간이 소요됨을 기대할 수 있다.

사실 적절한 캐시를 이용하여 해쉬 체인 계산시간을 줄일 수 있는 거의 최적화된 방법이 존재한다[4]. 본 논문에서 우리는 두개의 512-bit길이 캐시, 512-bit의 시드 저장소 등을 사용하는 400byte(SHA-1)나 320byte (MD5) 길이의 공격코드가 이와 같은 최적화 된 방법을 사용했을 때에도 여전히 효과적임을 보여줄 것이다.

V. 증명기법과 공격모델의 분석

5.1. 증명메커니즘의 성능분석

본 기법에서 우리는 총 실행시간 T를 다음과 같이 계산할 수 있다.

〔표 1〕 센서노드의 하드웨어 플랫폼

Sensor	MICA MICA2Dot MICA2[6]	Telos[7]
마이크로프로세서	ATmega128	MSP430F1611
워드 크기	8-bit	16-bit
클락 주기	16MHz	8MHz
SRAM 크기	4kB	10kB
플래쉬 크기	128kB	48kB
MIPS	16	8

$$T = T_h + \sum_{i=1}^{\Lambda} (T_r + T_a^i) \times S_E^i \quad (2)$$

T_r는 난수 방생에 소요되는 시간이고 T_a^i는 Mem^i의 접근시간이다. 현재 대부분의 종류의 센서노드는 Mem^1, Mem^2, Mem^3 만을 사용하고 있는데 Mem^1은 SRAM, Mem^2는 프로그램 가능한 플래쉬 메모리, Mem^3는 외부 EEPROM 이다.

만약 우리가 앞서 가정한바와 같이 관리자가 이미 플래쉬 메모리의 빈 공간을 남겨두지 않았다면 EEPROM의 쓰기시간은 매우 느리기 때문에 본 기법을 이용할 때 오직 SRAM 만을 랜덤 값으로 채우면 될 것이다. 몇몇 종류의 센서노드에서는 외부 EEPROM의 크기가 몇 메가비트이다. 그리고 이러한 EEPROM을 채우려면 수천초가 걸릴 것이다.(예를 들어 EEPROM의 사이즈가 4Mbit이고 쓰기시간이 1비트 당 1ms가 걸릴 경우, 전체 쓰기 시간은 1시간 8분 16초가 소요된다.) 그에 반해 SRAM의 쓰기시간은 무시해도 좋을 만큼 매우 빠르다.

위에서 언급한 바와 같이, 본 기법의 실행시간은 오로지 해쉬 연산의 횟수로 계산할 수 있으므로 식2를 다음과 같이 다시 쓸 수 있다.

$$T' = T_h + T_r \times S_E^1 \quad (3)$$

U_f를 빈 메모리 공간을 채울 때의 해쉬 라운드 수로 보고 δ를 한 라운드의 실행 시간으로 보자. 그렇다면 $T_r \times S_E^1$은 $U_f \times \delta$로 다시 쓸 수 있다. 또한 U_f는 다음의 식으로 계산할 수 있다.

$$U_f = \lceil |SRAM|/S_{ho} \rceil \quad (4)$$

U_r을 응답 단계에서의 해쉬 라운드의 수로 보고

$$U_r = \lceil (|SRAM|+|Flash|)/S_{hi} \rceil + 1 \quad (5)$$

S_{hi}는 해쉬 함수의 한 라운드에서의 입력 값 크기이다. (보통 S_{hi}는 512이다. +1은 해쉬 함수에서 초기 패딩 연산이 있기 때문에 추가한다.) 따라서 T_h도 $U_r \times \delta$로 표현할 수 있다. 최종적으로 T'는 다음과 같다.

$$T' = U_r \times \delta + U_f \times \delta = (U_r + U_f) \times \delta \quad (6)$$

ATmega128에서 라운드의 실행시간 δ는 SHA-1에서 3636 μs, MD5에서 1473 μs임을 Alexander Dean의 논문 내 실험결과에서 언급하였다[9]. 각각의 MIPS를 비교해봄으로써 ATmega128의 값으로 ATmega163

〔표 2〕 라운드 시간과 필요한 해쉬 라운드 수(*는 추산 값이다.)

마이크로프로세서	라운드 시간 (δ)	라운드 수 (U_f)	라운드 수(U_r)
ATmega128			
(SHA-1)	3636 μs	205	2112(+1)
(MD5)	1473 μs	256	2112(+1)
MSP430F1611			
(SHA-1)	*7272 μs	512	928(+1)
(MD5)	*2946 μs	640	928(+1)

〔표 3〕 해쉬 연산에 필요한 시간

마이크로프로세서	시간($U_f \times \delta$)	시간($U_r \times \delta$)	총 시간(T')
ATmega128			
(SHA-1)	0.745s	7.683s	8.428s
(MD5)	0.377s	3.112s	3.489s
MSP430F1611			
(SHA-1)	3.723s	6.756s	10.479s
(MD5)	1.855s	2.737s	4.622s

과 MSP430F1611의 δ를 추산하였다.

〔표 3〕은 각각의 마이크로프로세서에서의 실행시간을 보여주고 있다. 이 표에서 나타나다시피 실행시간은 마이크로프로세서에 따라 약1초부터 10초에 이르기까지 큰 차이를 보이고 있다. 따라서 노드로부터의 응답 대기시간을 설정할 시, 네트워크 지연시간과 더불어 이러한 추정 실행시간을 충분히 반영하여야 할 것이다.

5.2. 공격 모델의 성능분석

이제 우리는 4.3 단원의 최적화된 공격모델에 대한 분석을 할 것이다. 공격자는 응답단계에서 해쉬 함수가 필요로 할 때, 난수 값으로 구성된 블록 값을 생성해야 함을 상기하라. 공격자가 자신의 코드를 숨기기 위해 수행해야할 총 연산시간은 다음과 같다.

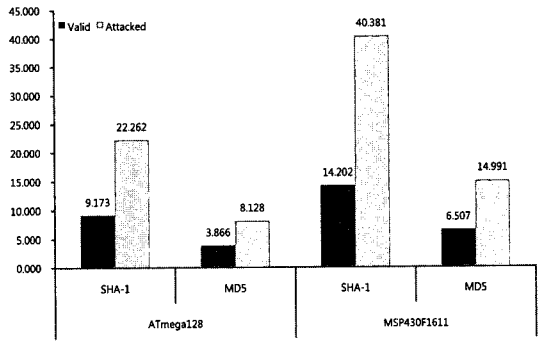
$$T_a = T' + U_a \times \delta \tag{7}$$

U_a는 공격자가 유효한 응답 값을 생성하기 위해 필요한 총 해쉬 라운드의 수이다. 우리는 본 공격 코드의 총 길이를 20-블록으로 가정하였다. 〔표 4〕는 공격 모델의 총 실행시간의 근사치이다.

〔그림 6〕는 정상적 검증기법의 총 연산시간과 공격코드의 총 연산시간의 격차를 보여준다. 이 그림에서 공

〔표 4〕 공격코드가 해쉬 연산에 필요한 시간

마이크로프로세서	라운드 (U_a)	시간 ($U_a \times \delta$)	공격코드의 총 시간(T')
ATmega128			
(SHA-1)	3600	13.089s	22.262s
(MD5)	2800	4.242s	8.128s
MSP430F1611			
(SHA-1)	3600	26.179s	40.381s
(MD5)	2800	8.484s	14.991s



〔그림 7〕 정상검증기법의 총 연산시간과 공격코드의 총 연산시간의 비교. 공격코드의 연산시간은 정상검증기법의 연산시간에 최소 2배이다.

격 코드의 시간은 정상의 그것에 최소한 두 배를 상회한다는 것을 알 수 있다. 이것은 검증자가 네트워크 트래픽등의 예상치 못한 시간 지연에도 불구하고 정상노드와 악의적 노드를 구분하기에 충분한 시간차를 얻을 수 있음을 의미한다.

VI. 결론 및 향후연구

이 논문에서 우리는 새로운 코드-증명 기법을 제안하였다. 이 기법은 매우 간단한 방식으로 동작하나 정확한 시간 정보에 의존하지 않고서도 센서노드 내의 악의적 코드를 발견해내는데 있어 효과적이다. 이 기법의 특성상 증명자와 목표노드간의 시도-응답 방식을 노드와 노드가 서로 증명자와 목표노드의 역할을 수행하는 방식으로 쉽게 변환할 수 있다. 우리는 이러한 노드-대-노드 증명기법을 연구하고 있으며, 특히 노드-대-노드 증명기법은 BS의 중계 없이 각각의 노드가 연계하여 포획된 노드를 네트워크 내에서 제외시키는 기능을 수행하도록 확장할 수 있다. 본 기법을 노드-대-노드 기법까지 확장

할 경우 BS가 없는 환경에서도 공격자에 의해 악의적으로 수정되었다고 의심되는 노드를 선행 검증하고 네트워크에 피해를 주는 행동을 하기 전에 적절히 제거할 수 있을 것이다. 또한 본 기법의 적절한 수정을 통하여 특정 그룹 내 노드들의 코드를 증명하는 그룹 코드-증명 기법을 고안할 수 있을 것이다.

참고문헌

[1] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla "SWATT:SoftWare-based ATTestation for Embedded Devices." *In Proceedings of the IEEE Symposium on Security and Privacy*, May 2004.

[2] M. Shaneck, K. Mahadevan, V. Kher, and Y. Kim "Remote Software-Based Attestation for Wireless Sensors." *In ESAS 2005*, LNCS 3813, pp. 27-41, 2005.

[3] Jason L. Hill, Dacid E. Culler "Mica: A Wireless Platform for Deeply Embedded Networks." In Nov-Dec 2002, IEEE MICRO

[4] Don Coppersmith and Markus Jakobsson. "Almost Optimal Hash Sequence Traversal." *In Proceedings of the Fifth Conference on Financial Cryptography* (FC '02), February 2002.

[5] Atmel 8-bit AVR Microcontroller with 16K Bytes In-System Programmable Flash. ATmega163 ATmega163L Summary Rev. 1142CS-09/01

[6] Atmel 8-bit AVR Microcontroller with 128K Bytes In-System Programmable Flash. ATmega128 ATmega128L Rev. 2467N.AVR. 03/06

[7] Telos Ultra low power IEEE 802.15.4 compliant wireless sensor module. PRELIMINARY Datasheet(12/5/2004)

[8] Atmel 4-megabit 2.5-volt or 2.7-volt DataFlash. AT45DB041B 3443C-DFLSH-5/05

[9] P. Ganesan, R. Venugopalan, P. Peddabachagari, A. Dean, F. Mueller, M. Sichitiu "Analyzing and Modeling Encryption Overhead for Sensor Network Nodes." *In WSNA'03*, September 19, 2003, San Diego, California, USA. ACM Press

〈著者紹介〉

최 영 근 (Young-geun Choi) 학생회원
2006년 2월 : 인하대학교 컴퓨터 공학과 졸업
2006년 3월~현재 : 인하대학교 정보통신대학원 석사
<관심분야> 무선 센서 네트워크 보안

강 전 일 (Jeonil Kang) 학생회원
2003년 2월 : 인하대학교 컴퓨터 공학과 졸업
2006년 2월 : 인하대학교 정보통신대학원 석사
2006년 3월~현재 : 인하대학교 정보통신공학과 박사 과정
<관심분야> 생체 인식 보안, 무선 센서 네트워크 보안, 무선 인터넷 보안

양 대 헌 (DaeHun Nyang) 정회원
1994년 2월 : 한국과학기술원 과학기술 대학 전기 및 전자 공학과 졸업
1996년 2월 : 연세대학교 컴퓨터 과학과 석사
2000년 8월 : 연세대학교 컴퓨터 과학과 박사
2000년 9월~2003년 2월 : 한국전자통신연구원 정보보호연구본부 선임연구원
2003년 2월~현재 : 인하대학교 정보통신대학원 조교수
<관심분야> 암호이론, 암호프로토콜, 인증프로토콜, 무선 인터넷 보안

이 경 희 (KyungHee Lee)
1989년 : 서울대학교 식품영양학과 학사
1993년 : 연세대학교 전산과학과 학사
1998년 : 연세대학교 컴퓨터과학과 석사
2004년 : 연세대학교 컴퓨터과학과 박사
1993년 1월~1996년 5월 : LG소프트(주) 연구원
2000년 12월~2005년 2월 : 한국전자통신연구원 선임연구원
2005년 3월~현재 : 수원대학교 조교수
<관심분야> 영상처리, 컴퓨터비전, 인공지능, 패턴인식, 생체인식, 얼굴인식, 다중생체인식