Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.26, NO.5, Oct. 2016 ISSN 2288-2715(Online)
http://dx.doi.org/10.13089/JKIISC.2016.26.5.1211

저전력 블루투스를 통한 사물 인터넷 장치의 소프트웨어적인 코드 검증*

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

Software Code Attestation for IoT Devices by Bluetooth Low Energy*

GeunYoung Kim,[1†] Jeonil Kang,[1] DaeHun Nyang,[1] KyungHee Lee[2‡]
[1]INHA University, [2]The university of Suwon

요 약

사물 인터넷(Internet of Things) 환경에서의 장치들의 신뢰성 확보는 무엇보다 중요하다. 현재의 보안 위협은 대부분 정보의 노출과 조작, 또는 금전적인 이득을 위한 것이나, 사람의 삶이 사람을 둘러싼 장치들(Things)에 의해서 인터넷으로의 연결성의 강화되었을 때, 그 장치들로부터의 보안 위협은 직접적으로 사람을 겨냥할 가능성이 있다. 장치의 경우 인증은 인증 대상이 배타적으로 알고 있는 정보, 즉 비밀키를 검증함으로써 이루어진다. 하지만 공격자가 물리적으로 장치를 수정한다면 더 이상 비밀키를 알고 있다는 것은 신뢰의 증거가 될 수 없게 된다. 따라서 코드 검증(code attestation)과 같은 강력한 신뢰 확보 방법이 필요하다. 이 논문에서는 검증의 효율을 위해 비용이 적은 소프트웨어적인 코드 검증을 사용하였다. 원본 코드 복사를 통한 회피 방법에 대해 안전한 코드 검증 방법을 제시하고 이를 임베디드 디바이스에 적용하여 성능을 분석해 보인다.

ABSTRACT

In IoT environment, making sure of trust of IoT devices is the most important one than others. The security threats of nowadays almost stay at exposure or tampering of information. However, if human life is strongly connected to the Internet by IoT devices, the security threats will probably target human directly. In case of devices, authentication is verified using the device-known private key. However, if attacker can modify the device physically, knowing the private key cannot be the evidence of trust any more. Thus, we need stronger verification method like code attestation. In this paper, we use software-based code attestation for efficiency. We also suggest secure code attestation method against copy of original code and implement it on embedded device and analyze its performance.

Keywords: Software Code attestation, IoT device

I. 서 론

사물 인터넷(IoT, Internet of Things)을 구성하는 소프트웨어 내장 기기의 수가 광범위한 영역에 폭증하고 있다. 이와 함께 사물 인터넷 장치를 표적으로 한 해킹 공격이 증가 추세에 있다. 코드가 수정된 사물 인터넷 장치는 소극적으로는 시스템 내부의 데이터를 외부로 유출 시키는 역할을 수행할 수 있지만 보다 적극적으로는 시스템이 올바르게 동작하지 못하도록 할 가능성이 있다[1]. 나아가서 원격 수술 로봇과 같이 인간의 생명에 밀접하게 연결되어 있는 장치를 공격할 경우, 사용자에게 큰 피해를 입힐 가능성이 존재한다[2]. 따라서 반드시 수정된 장치를

Received(08. 03. 2016), Modified(09. 30. 2016), Accepted(09. 30. 2016)
* 이 논문은 2016년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(2014R1A1A2059852)
† 주저자, kimky@isrl.kr
‡ 교신저자, khlee@suwon.ac.kr(Corresponding author)

찾아 시스템에서 격리 시켜야만 한다. 수정된 장치를 찾는 방법으로 논문에서는 코드 검증을 사용할 것이다. 코드 검증은 대상이 가지고 있는 실행코드와 메모리 자체를 검증하는 방법으로 대상이 물리적으로 수정되지 않았는지를 알아내는 방법론이다. 코드 자체를 검증하기 때문에, 수정된 장치를 정확하게 찾아낼 수 있다. 대부분의 사물 인터넷 장치는 외부에 노출되어 있으므로 물리적 접근을 차단하기가 어렵기 때문에 주기적인 코드 검증의 수행으로 감염된 사물 인터넷 장치를 찾아야 한다.

장치에 내부에 존재하는 소프트웨어를 검증하기 위해서는 물리적으로 장치에 접근할 수도 있으나, 확인해야할 장치가 매우 많은 경우에 이는 검증 요청자에게 큰 부담을 주게 된다. 또한 하드웨어 모듈에 의한 코드 검증은 필연적으로 장치의 가격을 비싸게 만든다. 따라서 소프트웨어 기반의 코드 검증이 필요하며 검증하고자 하는 사물 인터넷 장치의 위치와 수에 크게 영향을 받지 않도록 무선 네트워크 환경에서의 코드 검증이 가능하게 해야 할 것이다.

이 논문에서는 사물 인터넷 환경의 장치들을 위하여, 향상된 소프트웨어 코드 검증 알고리즘을 제시하고 실제 저전력 블루투스 장치에 코드 검증 기능을 구현하였고 스마트 기기를 이용하여 원하는 사물 인터넷 장치에 대한 코드 검증을 수행하였다. 제안하는 기법이 바탕을 두고 있는 코드 검증 기법[7],[8]은 많은 이론적 분석에도 불구하고 사물 인터넷 장치에서 실제로 구현된 적은 없다. 그러나 기존에 분석된 최적 실행시간 및 하드웨어 성능을 바탕으로 예측된 코드 검증 성능은 임베디드 운영체제 및 네트워크 상황에 따라 차이를 보일 수 있다. 따라서 이 논문에서는 실제 사물 인터넷 장치에서 코드 검증 시 발생할 수 있는 편차를 검증하고 미세한 시간차를 이용한 접근 방법의 어려움을 보인다.

이 논문의 구성은 다음과 같다. Ⅱ장에서는 현재까지 연구된 소프트웨어적인 코드 검증 관련 연구를 소개하고 Ⅲ장에서는 논문에서 사용된 가정과 공격모델에 대해 설명할 것이다. Ⅳ장에서는 코드 검증 기법에 대하여 서술하고 Ⅴ장에서는 이 기법을 실제 임베디드 디바이스에 탑재하여 성능을 실증하고 그 보안성을 분석할 것이다. Ⅵ장에서 이 논문을 마무리한다.

Ⅱ. 관련 연구

2000년, D. Spinellis는 소프트웨어의 무결성을 확인하기 위해 reflection 기반의 코드 검증 기법을 소개하였다[3]. 이 기법은 기본적으로, 검증 요청자의 요청에 대해서 메모리 영역에 탑재된 프로그램 코드의 일부분을 암호학적 해시 함수를 이용하여 해시한 후에 전송하여 확인하도록 하는 것이다. 검증 요청자는 프로그램 코드 영역이 중복되도록 복수의 코드 검증을 수행한다. 하지만 이러한 방법으로는 원래 소스코드를 압축하여 다른 메모리 공간에 저장하는 회피 방법에 대응할 수 없기 때문에, 검증 수행자는 모든 외부 인터럽트를 끄고 프로세서의 상태를 특정한 값으로 초기화한 다음, 코드 검증을 수행한 뒤, 그 상태에서의 프로세서의 상태를 함께 전송하도록 하는 확장 방법을 제안하였다. 검증 수행자가 올바른 코드 검증 결과 값을 생성하기 위해서 추가적인 연산은 수행한다면 프로세서의 상태가 변하게 되므로 이를 찾아낼 수 있다. 이때, 프로세서의 상태는 예측 불가능한 것으로 알려져 있다.

2004년, A.Seshadri 등은 SWATT(SoftWare-based ATTestation)이라고 알려져 있는 임베디드 디바이스에 적합한 소프트웨어 기반의 코드 검증 기법을 제안하였다[4]. 이 기법에서는 요청된 코드 검증에 대해서 임베디드 디바이스가 자신의 메모리를 무작위로 재참조하며, 그에 대한 체크섬을 생성하여 응답하게 된다. 이때 만약 원래의 실행코드를 여분의 메모리 공간에 옮기고 이를 이용하기 위해서 검증 코드를 수정하면 수정된 임베디드 디바이스는 올바른 응답을 주게 된다. 이러한 회피를 찾아내기 위해서, 그들은 원래의 코드와 수정된 코드 사이에서 시간차가 발생함을 이용하였다. 만약 체크섬 생성을 위한 메모리 참조 횟수가 작다면 수정된 코드와 시간차가 크게 발생하지 않게 된다. 그러나 메모리 참조 횟수가 굉장히 많다면 이로써 증폭되는 시간차는 원래의 코드와 구별 가능한 수준이 된다는 것을 사용하였다.

2006년, A. Seshadri 등은 SWATT 기법을 바탕으로 WSN에 적용한 ICE(Indisputable Code Execution)를 만들었다[5]. ICE는 SWATT와 다르게 체크섬을 생성할 때 마이크로프로세서의 PC(program counter)와 상태 값을 이용한다. 하지만 모든 마이크로프로세서에서의 PC와 상태 값에 접근 가능한 것이 아니기 때문에, 폭넓은 적용에는 제한이 따른다.

2005년, M. Shaneck 등은 고정된 검증 코드를 센서 노드 안에 내장하지 않고, 원격으로 매번 다른 검증 코드를 전송하는 기법을 제안하였다[6]. 이 기

법은 매번 다른 검증 코드가 전송되기 때문에 전송된 검증 코드의 어느 위치에서 어떠한 메모리를 참조하는지 실시간으로 알아내는 것이 어려워 센서 노드 수준에서는 검증 코드를 정적 분석을 통하여 제시간 안에 분석하여 응답을 만들어내는 것이 불가능하다. 문제는 이렇게 안전한 검증 코드를 매번 다르게 생성하는 것 자체가 쉽지 않으며, 전송된 검증 코드를 저장하고 실행하기 이해서는 센서 노드의 프로그램 메모리를 일정부분 삭제하고 저장하여야 하는데, 이러한 과정이 느리며, 복잡하다는 것이다.

2007년, 최영근 등은 검증 요청자로부터 수신한 랜덤 시드를 이용하여 센서 노드 자신의 남아 있는 여분의 SRAM 메모리 공간을 랜덤비트로 가득 채운 후 전체 메모리(프로그램 및 SRAM 공간 포함)를 해시하여 응답을 생성하는 소프트웨어 기반의 코드 검증 기법을 제시하였다[7]. 남아 있는 모든 메모리 공간을 채움으로써, 수정된 노드가 원래 코드를 위한 공간을 채우지 않고 메모리가 필요할 때 실시간으로 필요한 랜덤비트를 생성하는 공격을 완화하기 위하여, XOR을 이용하여 랜덤비트를 반복적으로 덮어쓰는 방식을 이용하였으며, 전체 메모리에 대한 해시를 계산하기 위하여 메모리 순서에 수직 방향으로 해시를 계산하도록 하였다.

2011년, T. AbuHmed 등은 [7]의 기법에 대하여 시뮬레이션을 통하여 그 성능을 검증하였으며, 해당 기법을 바탕으로 BS(Base Station)의 도움 없이 노드 사이에 수행할 수 있는 그룹 코드 검증 기법을 제안하였다[8].

III. 가정 및 공격 모델

3.1 가정

이 논문에서 사용하는 가정은 다음과 같다.

가. 검증자의 코드 검증을 도와줄 서버는 목표 장치(검증하고자 하는 장치)의 메모리 내용의 사본을 가지고 있다.

나. 검증자가 원격으로 실행 가능한 코드 검증 루틴은 목표 장치의 메모리 안에 탑재되어 있으며 검증자의 스마트 기기와 미리 페어링이 되어 있어 향후에 연결이 가능하다.

다. 공격자는 포획한 사물 인터넷 장치의 메모리 접근에 대한 모든 권한을 가질 수 있으나, 그 외의 하드웨어에 대한 수정 권한은 가지고 있지 않다. 이는 공격자가 외부 자원을 이용할 수 있다는 의미가 되므로 이 논문에서는 논외로 한다.

라. 프로그램 가능한 플래시 메모리는 특별한 목적이 없는 한, 어떠한 빈 공간도 가지고 있지 않는다.

마. 장치는 가상 메모리를 지원하지 않으며, 데이터 저장소에서는 직접적으로 코드를 실행할 수 없다.

3.2 공격 모델

장치에 내재된 코드의 수정 여부를 검증하기 위한 가장 원시적인 기법은 검증자가 목표 장치의 실제 메모리 내용과 자신이 알고 있는 메모리 내용을 비교해 보는 것이다. 이 때 공격자는 장치의 내부에 악의적으로 수정된 악성 코드를 이용하여 검증 기법에 의해 발견되는 것을 피할 수 있다. 공격자는 원본 프로그램 코드를 메모리의 빈 공간에 보관하고, 검증자가 검증을 요청하게 되면 공격코드를 실행 시켜 미리 저장한 코드로 검증 과정을 넘겨줌으로써 장치가 변조되기 이전의 유효한 검증 값을 만들어 내도록 한다. 이 과정을 통해 검증자는 검증 대상 사물 인터넷 장치가 정상적인 장치라고 검증하게 된다.

3.3 저전력 블루투스

블루투스 무선 기술은 다양한 전자기기 간의 간편한 연동을 위한 글로벌 단거리 무선 표준이다. 현재까지 약 90억 대 이상의 블루투스 디바이스가 출하됐으며, 매년 20억 대의 디바이스가 생산되고 있다. 2010년 6월 30일에 채택된 블루투스 4.0 버전에는 Bluetooth Low Energy(BLE)라는 새로운 프로토콜이 추가되어 기존 버전들보다 전력을 적게 소모하게 되었다. 그 이후의 버전들은 BLE를 베이스로 한다. 또한 128 비트 AES데 데이터 암호화를 사용하여 업계에서 가장 높은 수준의 보안을 허용한다[10]. 논문에서 사용한 코드 검증에서는 많은 양의 데이터 통신을 필요로 하지 않기 때문에 사물 인터넷에 적합한 BLE를 통신에 사용하도록 하였다.

IV. 코드 검증 기법

이전의 코드 검증 기법은 충분한 시간이 있다면, 수정된 장치라고 하더라도 캐시를 사용하는 트릭 등을 이용하여 올바른 응답을 만들어 낼 수 있다. 이는 코드 검증 기법이 시간에 대하여 민감성을 가진다는 것을 의미한다. 무선 네트워크 환경의 불확실성을 고려하였을 때, 정상적인 장치와 수정된 장치에서의 코드 검증 시간의 차이는 이전의 코드 검증 기법들보다 훨씬 커야할 필요가 있다. 코드 검증 시 더 많은 메모리 영역에 대하여 횡단을 수행한다면 이러한 시간 차이를 크게 만들 수 있는데, 이 경우는 필연적으로 정상적인 장치의 코드 검증 시간도 길어지게 된다. 따라서 정상적인 장치에서는 비슷한 수준의 횡단 비용을 지불하면 체크섬을 만들 수 있지만 수정된 장치에서는 더 큰 비용을 지불하도록 해야 한다. 본 단락에서는 기존의 코드 검증 기법에서 캐시를 사용하는 공격자가 가지는 이득을 줄이기 위해, 무선 네트워크 환경에서 이루어지는 코드 검증의 과정과 더불어 개선된 메모리 채우기 기법을 설명한다.

4.1 코드 검증 단계

논문에서 제시할 소프트웨어 코드 검증 기법은 다음과 같은 순서로 이루어진다.

1) 요청 단계

검증자는 저전력 블루투스를 이용하여 검증하고자 하는 장치를 검색한다. 검증하고자 하는 장치들은 사용되기 이전에서로 주소, 이름, 프로파일 등이 교환되어 있으므로 검증자는 스마트 기기에서 원하는 장치를 선택함으로써 장치와 연결한다. 이후에 무작위의 시드 값을 입력한 후 암호화하여 장치와 서버로 송신한다.

2) 연산 단계

시드를 수신 받은 사물 인터넷 장치와 서버는 코드 검증 루틴을 시작한다. 시드를 시작으로 하여 4.2에서 설명할 특정한 규칙에 의하여 빈 메모리 공간을 채우게 된다. 메모리 공간을 모두 채운 후 전체 메모리에 대해서 체크섬을 계산하여 스마트 기기로 전송한다.

체크섬을 계산할 때는 메모리를 채울 때의 방향에 직교한 방향으로 체크섬을 계산하여 생성한다. 이는 기존의 코드 검증 기법[7],[8]과 동일하며, 공격자로 하여금 공격코드를 반복하여 복사하게 하므로 정상적인 장치에서와 수정된 장치에서의 체크섬 계산 시간 차이를 증가시키게 된다.

3) 검증 단계

검증자는 서버에서 받은 체크섬 값과 검증하고자 하는 사물 인터넷 장치가 생성한 체크섬의 값을 비교함으로써 사물 인터넷 장치의 수정 여부를 판별할 수 있다.

4.2 개선된 빈 메모리 채우기 기법

이전의 코드 검증 기법 [7],[8]에서 다소 복잡한 방법으로 빈 메모리 공간을 채우도록 하였는데, 이는 메모리 복사 공격을 수행하는 악성 코드에 대하여 메모리 횡단 시 더 큰 비용을 지불하도록 만들기 위함이었다. 하지만 이 방법은 빈 메모리를 채우는 위치가 예측 가능하기 때문에 효율적으로 캐시를 적용할 수 있는 위치를 찾아 이용할 경우 다소 취약할 수 있다. 따라서 빈 메모리를 채우는 위치의 무작위성이 보장되어야 하는데, 이 경우 쿠폰 수집가 문제로 인하여 채워지지 않는 부분이 매우 높은 확률로 발생할 수 있다. 공격자는 이렇게 채워지지 않은 공간을 이용할 수 있다. 따라서 이 문제를 해결하기 위해 아직

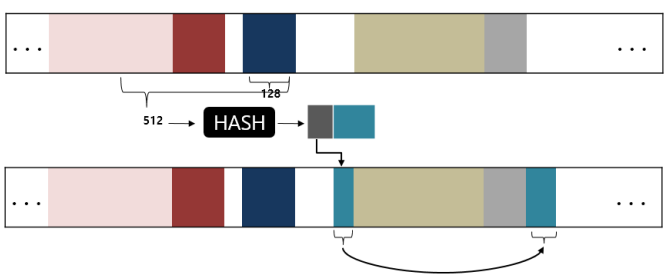

Fig. 1. If Collision occurs, find new position and write random numbers.

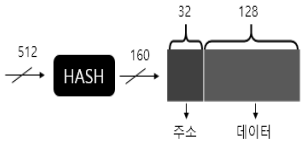

Fig. 2. The output of hash

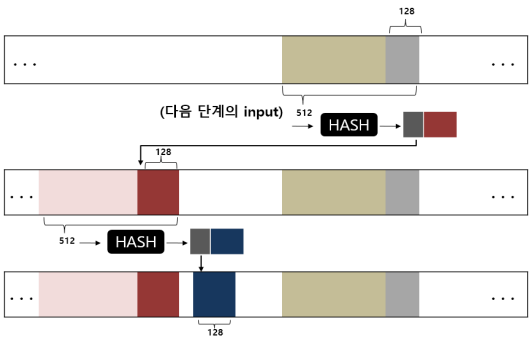

Fig 3. Filling memory space - Phase 1.

채워지지 않은 공간인지 아닌지를 확인하여, 동일한 공간에 무작위의 비트 값을 덮어쓰지 않도록 충돌을 감지하고 이를 회피하도록 하였다. 즉, 이미 채워진 공간에 다시 덮어 쓰려고 할 때, 이를 감지하고 다른 메모리 공간을 찾아 채우는 방식으로 빈 메모리 공간이 남는 것을 방지할 수 있다. 그림 1은 이를 도식화한 것이다.

메모리를 채우는 알고리즘은 두 단계로 이루어지며, 이 두 단계를 통해 메모리는 두 층을 이루는 난수로 채워지게 된다.

첫 번째 단계는 시드를 이용하여 메모리의 빈 공간을 난수로 채우는 단계이다. 사물 인터넷 장치는 스마트 기기에서 받은 시드와 프로그램 메모리 일부분을 합쳐 해시함수의 입력으로 이용한다. 그림 2는 해시함수의 결과 값과 그 용도를 나타낸다. 해시함수의 결과로 나온 160비트의 난수의 [0…31] 비트는 난수를 채울 메모리의 주소를 만드는데 이용하고 [32…159] 비트는 메모리에 쓴다. 마지막 비트부터 이전의 512비트를 다음 단계의 해시함수의 입력으로 사용한다. 모든 메모리가 채워질 때까지 같은 작업을 반복한다. 그림 3은 이에 대한 의사 코드, 그림 4는 이를 도식화 한 그림이다.

두 번째 단계는 이미 채워진 난수를 새로운 난수와 XOR 연산으로 덮어쓴다. 첫 번째 단계가 종료되면 장치는 해당 위치부터 다시 512비트의 메모리 공간을 읽어 해시함수의 입력 값으로 사용한다. 출력된

```
FOR ī=1 to n DO
  IF ī=1 THEN
    result = HASH^512(Programcode||SEED)
    next_address = result[0…31]
    filling_data = result[32…159]
    Mem[next_address] = filling_data
  ELSE
    result
        = HASH^512(previous416−bit||Mem[next_address])
    next_address = result[0…31]
    filling_data = result[32…159]
    isEmpty = false;
    index = next_addresss
    WHILE !isEmpty DO
      IF Mem[index] != 0 THEN
        index++
      ELSE
        isEmpty = true
      ENDIF
    ENDWHILE
    Mem[index] = filled_data
  ENDIF
ENDFOR
```

Fig. 4. Pseudo code of phase 1.

160비트의 난수를 이용하여 첫 번째 단계와 마찬가지로 앞의 32비트는 주소공간을 지정하기 위한 용도로 뒤의 128비트는 메모리를 채우는 용도로 사용한다. 이 때 첫 번째 단계에 의해 이미 메모리는 난수로 채워져 있으므로, 새로 생성한 난수를 기존의 난수와 XOR함으로써 덮어 쓴다. 두 번째 단계는 메모리 전체에 대해 동작하지 않고 전체 메모리 공간의 1/2의 공간을 덮어쓸 때까지 진행한다. 그림 5는 이를 도식화한 그림이고 그림 6은 이 과정에 대한 의사코드이다.

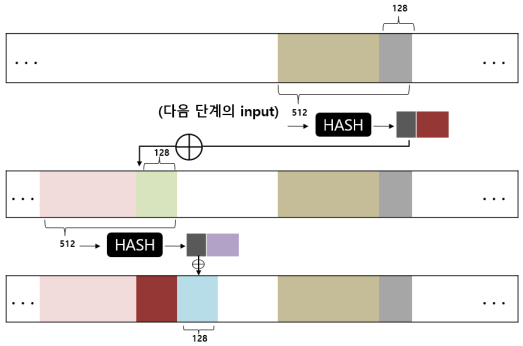

Fig. 5. Filling memory space - Phase 2.

```
FOR ī=1 to E DO
  IF ī=1 THEN
    result = HASH^512(last512-bit)
    next_address = result[0…31]
    filling_data = result[32…159]
    Mem[next_address] = filling_data
  ELSE
    result
    = HASH^512(previous416-bit||Mem[next_address])
    next_address = result[0…31]
    filling_data = result[32…159]
    Mem[index] ⊕ filled_data
  ENDIF
ENDFOR
```

Fig. 6. Pseudo code of phase 2.

V. 실험 및 보안성 분석

5.1 실험

코드 검증 실험은 코드 검증 루틴을 저전력 블루투스 실드가 장착된 아두이노 우노에 탑재하여 진행하였다. 총 1,000개의 길이가 16인 시드 값을 무작위로 선택하여 실시하였다.

5.1.1 실험 환경

코드 검증을 실험한 환경은 표 2와 같으며, 실제로 구현한 모습은 그림 7과 같다. 아두이노 우노(아래쪽 보드)에 저전력 블루투스 실드(위쪽 실드)를 장착한 모습이다.

아두이노는 마이크로프로세서와 입출력 모듈이 하나의 칩으로 만들어져 특정한 기능을 수행할 수 있는 작은 임베디드 장치이다. 오픈 소스에 기반을 두고 있으며, 다수의 센서나 스위치로부터 데이터를 수신하고, 보드에 연결된 장치들을 통제함으로써 물리적

Table 1. Specification of Arduino UNO.

MicroContoller	ATmega328P
Flash Memory	32KB of which 0.5KB used by bootloader
Clock Speed	16MHz
SRAM	2KB
EEPROM	1KB

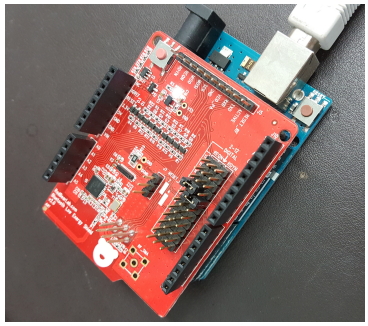

Fig. 7. Actual feature of Embedded Device which is used in code attestation test

Table 2. Implementation Environment of Code attestation

Bluetooth Shield	Nordic nRF8001 Bluetooth Low Energy IC	
Smart Device for Attester	Device	Galaxy Note5 (SM-N920L)
	CPU Speed	1.5GHz
	CPU	Octa-Core
Server	Intel(R) Core(TM) i5-4460 CPU @ 3.20GHz, RAM 16GB	

인 환경과 상호작용하는 기기를 비교적 쉽게 개발할 수 있다. 아두이노의 기본이 되는 모델은 아두이노 우노(Arduino UNO)이며, 성능에 영향을 주는 마이크로프로세서의 종류와 메모리의 크기, 입출력 핀의 개수에 따라 다양한 시리즈가 존재한다. USB 포트가 있기 때문에 USB 케이블을 이용해서 컴퓨터와 연결하여 프로그래밍이 가능하다. 대부분의 컴포넌트 및 쉴드는 아두이노 우노와 호환이 되어 사물 인터넷 분야에서 많이 사용되는 모델이다[11]. 사양은 표 1과 같다. 아두이노 우노의 메모리는 2KB SRAM, 32KB 플래시 메모리, 1KB의 EEPROM으로 구성되어 있다. 보통 오직 한 개의 프로그램 가능한 플래시 메모리가 마이크로컨트롤러에 내장되고 외부 데이터 저장소는 EEPROM이다[11]. 논문의 가정에서 외부 데이터 저장소를 이용한 공격은 염두 해두지 않는다고 하였는데, 그 이유는 외부 데이터 저장소를 공격 목적으로 사용하기에는 접근 시간이 너무 길어서 쉽게 발견할 수 있기 때문이다.

5.1.2 코드 검증 과정

스마트 기기에서 코드 검증을 실시했을 때의 모습은 그림 8~그림 11과 같다.

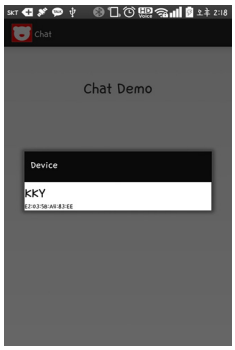

Fig. 8. Running code attestation program on smart device.

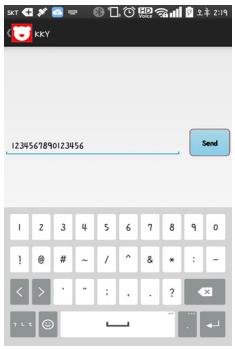

Fig. 9. Input seed to send IoT device and server.

Fig. 10. Result of test on non-compromised IoT device.

Fig. 11. Result of test on compromised IoT device.

검증자는 코드 검증 프로그램을 설치한 스마트 기기를 소지하고 검증을 하고자 하는 장치의 근처로 이동하여 블루투스를 활성화 한 후에 프로그램을 실행한다. 그러면 미리 페어링 되어 있는 블루투스 장치들의 이름이 나열된 그림 8과 같은 화면이 나타난다. 검증자는 검증하고자 하는 사물 인터넷 장치를 선택한다. 장치를 선택한 후에 그림 9와 같이 길이가 16인 시드를 입력하고 "Send" 버튼을 누르면, 입력한 시드 값이 서버와 검증하고자 하는 장치로 각각 전송된다. 시드 값을 받은 사물 인터넷 장치는 코드 검증 루틴을 거쳐 계산한 체크섬 값을 스마트 기기로 전송하고, 서버도 가지고 있던 사물 인터넷 장치의 메모리 사본을 이용하여 체크섬을 계산하여 스마트 기기로 전송한다. 이 때, 두 값이 같은 경우 그림 10과 같이 네트워크를 포함한 시간, 네트워크를 포함하지 않은 시간, 그리고 Not Compromised를 출력하게 된다. 두 값이 같지 않을 경우에는 그림 11과 같이 Compromised를 출력하게 된다.

5.1.3 실험 결과

총 1,000개의 길이가 16인 시드 값을 이용하여 코드 검증을 실시한 결과는 다음과 같다. 수행 시간을 50ms 단위로 나눈 실험 결과의 시간 분포와 그 평균 그리고 최대 편차는 표 3과 같다.

5.1.4 실험 결과 분석

공격자가 장치를 수정했을 경우, 정상적인 체크섬 값을 계산하기 위해서는 5.2.3에서 설명할 Memory-hard function의 원리에 따라, 정상적인 장치에서의 체크섬 계산 시간에 비해 월등히 증가된 계산 시간이 필요할 것이다. 코드 검증에 소요되는 시간이 실험에 따르면 시드 값에 따라 약간의 차이는 있지만 대략적인 시간을 유추할 수 있으므로, 최대 코드 검증 시간을 설정함으로써 코드의 수정 여부를 판별할 수 있다.

A. Seshadri 등의 연구[4]의 실험 결과로는 정상적인 장치와 수정된 장치에서의 코드 검증 시간의 차이가 0.25ms 로 나타났다. 그러나 이 논문에서 실험 결과 무선 네트워크 통신 시간에서만 평균적으로 52.449ms가 소요되고 시드 값에 따라 장치 내 검증의 경우 최대 61ms, 네트워크를 포함할 경우 최대 127ms의 편차가 발생함을 알 수 있다. 따라서 해당 기법은 무선 네트워크 환경에서의 코드 검증을 위해서는 두 장치에서의 코드 검증에 필요한 시간 차이를 더욱 증가시켜야 할 필요성이 있음을 확인하였다.

또한, 실험에 사용된 코드 검증 루틴이 SRAM에서 차지하고 있는 메모리는 1,203바이트로 이는

Table 3. Result Distribution

Time (unit:sec)	Attestation Time on IoT device	RTT on Smart device
0.00~0.60	0	0
0.60~0.65	527	0
0.65~0.70	473	450
0.70~0.75	0	531
0.75~0.80	0	19
Average	0.650	0.702
Maximum Deviation	0.061	0.127

2,048바이트 아두이노 우노 SRAM의 약 58%를 차지한다. 따라서 실험의 결과를 통해 같은 코드 검증 알고리즘을 사용하여 다른 사물 인터넷 장치에 대한 검증을 실행하였을 때 그 결과를 예측할 수 있다.

5.2 보안성 분석

5.2.1 Reply Attack

공격자는 장치를 수정하여, 이전의 장치의 검증에 사용 되었던 유효한 메시지를 재사용하는 방법 등으로 Reply Attack을 시도할 수 있다. 그러나 본 검증 기법은 challenge-and-response 방식이므로 검증자는 매번 다른 값을 송신하고 검증하고자 하는 장치로 하여금 매번 다른 값을 계산해 내도록 요구한다. 공격자는 충분하지 않은 메모리를 가지고 있으므로 수정된 장치에서 정상적인 응답을 도출해내는 것은 불가능하다.

5.2.2 Compression Attack

compression attack은 그림 12와 같이 원래의 코드를 압축한 다음, 빈 공간을 이용해서 악성코드를 삽입하는 공격이다. 이 논문에서 사용한 코드 검증 알고리즘에서는 빈 메모리 공간을 순차적이 아닌, 임의의 주소에 임의의 값을 채우도록 하였다. 공격자는 정상적인 값을 계산하기 위해서 압축된 원래의 코드를 on-the-fly 방식으로 압축 해제하여야 한다. 그러나 이 논문에서는 빈 공간에 임의의 값을 임의의 위치에 채워지도록 하였고, 이는 C. Castelluccia

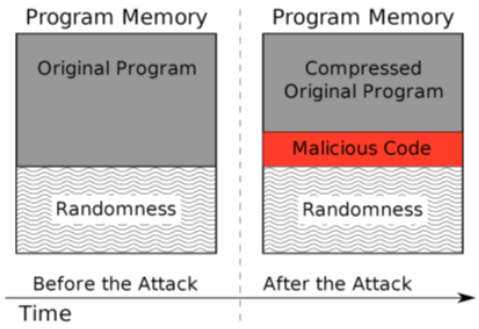

Fig. 12. compression attack[12].

등의 연구[12]에 의하면 순차적으로 접근하여 압축 해제하는 경우보다 월등하게 많은 시간을 필요로 하게 됨을 알 수 있다. 코드 검증에 소요되는 시간이 실험에 따르면 시드 값에 따라 약간의 차이는 있지만 대략의 시간을 유추할 수 있으므로, 최대 코드 검증 시간을 설정함으로써 코드의 수정 여부를 판별할 수 있다.

5.2.3 필요한 난수를 직접 계산하는 공격

우리는 빈 메모리 공간을 특정한 알고리즘으로 채우고, 공격자는 공격코드를 배치한 특정한 위치의 메모리의 값을 계산함으로써 검증을 통과하려고 한다. 이 때, 본 논문에서 제시한 빈 공간 채우기 알고리즘은 2단계로 이루어져 있고, 이때 1단계에서 해시함수의 입력-출력의 관계에 있었던 메모리들이 2단계에서 출력-입력의 관계가 된다면, 그림 13과 같은 사이클이 메모리 내에 생성되게 된다. 따라서 공격자는 기존의 메모리 값을 저장하지 않는 이상 바로 알

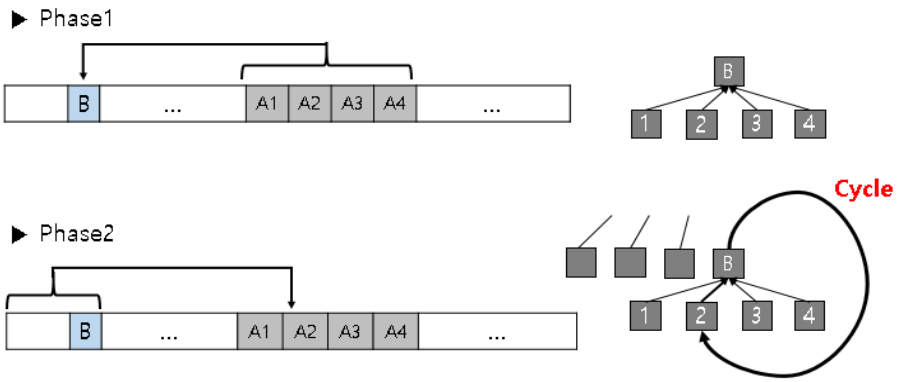

Fig 13. Cycle is generated in empty space through 2 phases of filling random bits.

수 없다. 게다가 메모리가 부족한 공격자는 기존의 값, 혹은 중간 값을 저장할 수 없고 정상적인 메모리의 값을 알기 위해서 공격자는 모든 단계를 다시 계산해야 한다. 계산에 필요한 메모리와 수행 시간이 trade off의 관계에 있을 때, Memory-hard function이라고 하며 이 때 계산하기 위하여 시간 T와 공간 S가 필요하다고 할 때, 시간과 공간은 T' ⟩⟩ T if S' ⟨ S의 관계에 있다. 그래프로 나타내면 그림 14와 같다[13]. 따라서 메모리가 제한될수록 계산 시간의 증가속도가 매우 빨라짐을 알 수 있다. 이와 같은 성질을 이용하면 제한된 메모리로 정상적인 값을 생성해 내어야 하는 공격자에게 큰 어려움을 주게 되며, 이는 메모리의 무결성을 확인하는 [Proofs of space]이 가능하다[14]. 이는 본 연구에서의 의도와 일맥상통한다. 공격자는 캐시를 이용하여 중간 값을 저장할 수 있지만, 이는 캐시를 위한 메모리가 추가로 필요하게 되고 충분하지 않은 메모리를 가지고 있으므로 공격이 어렵게 된다.

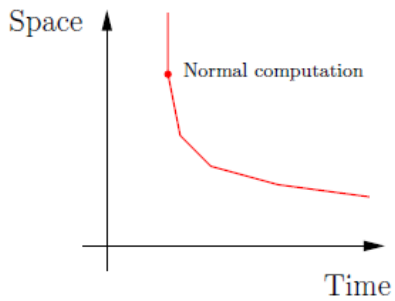

Fig 14. Relation between space and time in memory-hard function[13].

VI. 결 론

이 논문에서 제시한 코드 검증 기술은 목표로 하는 장치의 연산 능력이 지나치게 높은 경우(스마트폰, 개인 PC와 같은 경우) 오히려 보안성을 확보하기 힘든 측면이 있다. 연산 능력이 높으면 메모리 구조를 특정하기 힘들뿐만 아니라, 하드 디스크나 EEPROM과 같이 비대칭적인 쓰기와 읽기 속도를 가진 하드웨어의 경우 빈 메모리 공간을 채우는 것이 연산보다 더 느리기 때문에 메모리 복사를 수행하는 악성코드에게 큰 이득을 주게 된다. 장치의 연산 능력이나 메모리 용량이 지나치게 낮은 경우 (키보드와 같은 경우) 또한 몇몇 핵심적인 함수를 수행할 수 없어, 문제가 될 수 있는데 이러한 경우 보안성을 희생하더라도 동작 가능한 함수를 사용해야만 한다. 이러한 경우가 아니라면, 대부분의 장치에 대해서 코드 검증 기술은 적용 가능할 것으로 예상된다.

사물 인터넷 환경에서는 장치가 신뢰할 수 있는지 확인하는 것은 사물 인터넷으로 어떠한 서비스가 가능한가를 고민하기에 앞서 장치가 사람에게 위해를 가하지 않는가를 인증하는, 서비스 자체가 가능하기 위한 선결 조건으로써 무엇보다 중요하다고 할 수 있다. 따라서 궁극적으로는 장치를 위한 표준 코드 검증 기술이 필요하다. 이 연구는 이를 위한 기초가 될 수 있으며 장치 간 코드 검증 기술 및 표준 코드 검증 기술은 차후 연구 과제로 남겨둔다.

References

[1] SUNNYVALE, calif. - January 16, 2014. Proofpoint, Inc.

[2] Dieter Bohn. - June 13, 2013. The Verge, Vox Media, Inc.

[3] Diomidis Spinellis, "Reflection as a Mechanism for Software Integrity Verification," ACM Transactions on Information and System Security, Vol. 3, No. 1, pp. 51 - 62, February 2000.

[4] Seshadri, A., Perrig, A., Van Doorn, L., & Khosla, P. (2004, May). SWATT: Software-based attestation for embedded devices. In Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on (pp. 272-282). IEEE.

[5] Seshadri, A., Luk, M., Perrig, A., van Doorn, L., & Khosla, P. (2006, September). SCUBA: Secure code update by attestation in sensor networks. In Proceedings of the 5th ACM workshop on Wireless security (pp. 85-94). ACM.

[6] Shaneck, M., Mahadevan, K., Kher, V., & Kim, Y. (2005, July). Remote software-based attestation for wireless sensors. In European Workshop on Security in Ad-hoc and Sensor Networks (pp. 27-41). Springer Berlin Heidelberg.

〔7〕 Choi, Y. G., Kang, J., & Nyang, D. (2007, August). Proactive code verification protocol in wireless sensor network. In International Conference on Computational Science and Its Applications (pp. 1085-1096). Springer Berlin Heidelberg.

〔8〕 T. AbuHmed, J. Kang, D. Nyang, and KyungHee Lee, "A Software-Based Group Attestation for Wireless Sensor Networks," Adhoc & Sensor Wireless Networks Journal, Vol. 13, No. 1-2, pp. 121-154, Jan 2011.

〔9〕 Blom, Gunnar; Holst, Lars; Sandell, Dennis (1994), "7.5 Coupon collecting I, 7.6 Coupon collecting II, and 15.4 Coupon collecting III", Problems and Snapshots from the World of Probability, New York: Springer-Verlag, pp. 85–87, 191, ISBN 0-387-94161-4, MR 1265713.

〔10〕 Bluetooth Smar of Version 4.0+ of the Bluetooth specification, https://www.bluetooth.com/what-is-bluetooth-technology/bluetooth-technology-basics/low-energy

〔11〕 Arduino UNO, http://www.arduino.cc/en/Main/ArduinoBoardUno

〔12〕 Castelluccia, C., Francillon, A., Perito, D., & Soriente, C. (2009, November). On the difficulty of software-based attestation of embedded devices. In Proceedings of the 16th ACM conference on Computer and communications security (pp. 400-409). ACM.

〔13〕 Biryukov, Alex, Daniel Dinu, and Dmitry Khovratovich. "Fast and Tradeoff-Resilient Memory-Hard Functions for Cryptocurrencies and Password Hashing."*IACR Cryptology ePrint Archive* 2015 (2015): 430.

〔14〕 DZIEMBOWSKI, Stefan, et al. Proofs of space. In: Annual Cryptology Conference. Springer Berlin Heidelberg, 2015. p. 585-605.

〈저자소개〉

김 근 영 (GeunYoung Kim) 학생회원
2015년 2월: 인하대학교 컴퓨터정보공학과 학사
2015년 3월~현재: 인하대학교 컴퓨터정보공학과 석사과정
〈관심분야〉 정보보호, 사물 인터넷 보안, 네트워크 보안, 암호학

강 전 일 (Jeonil Kang) 정회원
2003년 2월: 인하대학교 전기전자컴퓨터공학과 학사
2006년 2월: 인하대학교 정보통신대학원 석사
2014년 8월: 인하대학교 정보통신공학과 박사
2014년 9월~현재: 인하대학교 인간중심컴퓨팅연구소 박사후연구원
〈관심분야〉 RFID 보안, 생체 인식 보안, 무선 센서 네트워크 보안, 무선 인터넷 보안, 웹 인증 보안

양 대 헌 (DaeHun Nyang) 종신회원
1994년 2월: 한국과학기술원 과학기술대학 전기 및 전자공학과 학사
1996년 2월: 연세대학교 컴퓨터과학과 석사
2000년 8월: 연세대학교 컴퓨터과학과 박사
2000년 9월~2003년 2월: 한국전자통신연구원 정보보호연구본부 선임연구원
2003년 2월~현재: 인하대학교 컴퓨터정보공학과 교수
〈관심분야〉 암호이론, 암호 프로토콜, 인증 프로토콜, 무선 인터넷 보안

이 경 희 (Kyung Hee) 정회원
1993년 2월: 연세대학교 컴퓨터과학과 학사
1998년 8월: 연세대학교 컴퓨터과학과 석사
2004년 2월: 연세대학교 컴퓨터과학과 박사
1993년 2월~1996년 5월: LG소프트(주) 연구원
2000년 12월~2005년 2월: 한국전자통신연구원 선임연구원
2005년 3월~현재: 수원대학교 전기공학과 부교수
〈관심분야〉 바이오인식, 정보보호, 컴퓨터비전, 인공지능, 패턴인식