사물인터넷 게이트웨이 보안을 위한 사용자 민감 데이터 분류

(User Sensitive Data Classification for IoT Gateway Security)

허만우*, 박기철*, 홍지만**
(Mhanwoo Heo, Kicheol Park, Jiman Hong)

요 약

IoT 기술이 산업 환경에서 널리 활용되면서, IoT 환경 보안 문제가 중요해지고 있다. 이러한 문제를 해결하기 위해 하드웨어 보안 기능을 활용한 연구가 활발히 진행되고 있다. 그러나, 기존 연구는 IoT 환경에서 하드웨어 보안 기능을 이용할 때 발생하는 성능저하에 대해 고려하지 않았다. IoT 환경에서 주로 활용되는 게이트웨이 기기는 자원이 제한적인 경우가 많다. 이러한 환경에서 하드웨어 보안 기능을 활용하는 것은 게이트웨이와 연결된 IoT 기기가 점점 증가함에 따라 심각한 성능 저하를 일으킬 수 있다. 따라서, 본 논문에서는 자원이 제한적인 환경에서 효율적으로 하드웨어 보안 기능을 활용하기 위한 데이터 분류 기법을 제안한다. ARM 트러스트존을 사용하여 제안한 기법이 적용된 플랫폼을 구현한다. 구현한 플랫폼상에서 실험을 통해 하드웨어 보안 기능으로 인한 성능 저하를 측정하고 제안한 기법을 적용하였을 시 성능과 비교 및 분석한다.

■ 중심어 : 사물인터넷 ; 트러스트존 ; 데이터분류 ; 플랫폼

Abstract

As IoT technology is widely used in industrial environments, its environmental security issues are becoming more important. In such a context, studies utilizing hardware security functions are being actively carried out. However, previous studies did not consider the performance degradation that occurs when using hardware security functions in IoT environment. Gateway devices that are mainly used in IoT environments are often resource-limited. Utilizing hardware security in such an environment can cause serious performance degradation as the number of IoT devices connected to the gateway increases. Therefore, in this paper, we propose a data classification scheme to efficiently utilize hardware security functions in resource limited environment. We implement a platform with the proposed technique using ARM Trustzone. Performance degradation due to the hardware security functions is measured through experiments on the implemented platform and compared with the performance as of when the proposed technique is applied.

■ keywords : IoT ; ARM Trustzone ; Data Classification ; Platform

Ⅰ. 서 론

IoT 기술이 발전함에 따라 스마트 팩토리와 스마트 홈과 같이 IoT 기술을 활용하여 서비스를 제공하는 사례가 증가하고 있다. 이러한 다양한 형태의 IoT 서비스를 제공하기 위해 새로운 기술과 프로토콜이 연구되고 있다[2]. 또한, 이러한 서비스 환경에서 많은 수의 IoT 기기로부터 생성된 데이터를 관리하는 플랫폼에 대한 연구가 활발히 진행되고 있다[6,8,9].

최근 개인 사용자를 위한 IoT 기기가 증가하고 있다. 스마트 홈 시스템에 사용되는 IoT 기기와 같이 개인 사용자를 위한 IoT 기기는 사용자의 실제 생활과 밀접한 데이터를 생성한다. 이러한 데이터를 통해 스마트 홈 환경의 실내 사용자의 유무, 위치, 생활 패턴 등과 같은 개인과 관련된 정보를 추출할 수 있다. 또한, 웨어러블 기기에서는 사용자의 신체 및 건강과 관련된 데이터를 생성한다.

IoT 기기에서 생성된 데이터는 직접 또는 게이트웨이를 통해 플랫폼으로 전송된다. 개인 사용자와 밀접한 IoT 기기의 수가

* 준회원, 숭실대학교 컴퓨터학과 대학원생
** 종신회원, 숭실대학교 컴퓨터학부 교수
이 논문은 2016년 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. NRF-2016R1D1A1B01016073, 가상화 기반 오픈 게이트웨이 플랫폼과 오케스트레이션 보안 서비스 프레임워크 설계 및 구현)

접수일자 : 2019년 08월 26일
게재확정일 : 2019년 11월 13일
교신저자 : 홍지만, e-mail : jiman@ssu.ac.kr

증가함에 따라, 게이트웨이를 통해 전달 및 관리되는 사용자 민감 데이터의 양 또한 증가하고 있다[3]. 개인 사용자와 밀접한 연관이 있는 데이터는 악의적인 유출 및 조작 시 개인에게 큰 손실을 발생시킬 수 있으며, 이를 막기 위해 게이트웨이 기기 보안 기술이 필요하다. 기존 연구에서는 ARM 트러스트존[10] 및 Intel Software Guard Extensions(SGX)[13]과 같은 프로세서 기반 하드웨어 보안 기능을 사용하여 IoT 환경 보안 기술을 구현한다. 두 기술의 기능 및 인터페이스는 다르지만, 안전한 실행공간 제공과 접근 제어를 통해 보안을 보장하는 것은 유사하다. 안전한 실행공간에서 연결된 IoT 기기와 데이터를 관리하는 것으로 보안성을 제공한다[1].

그러나 ARM 트러스트존과 Intel SGX와 같은 기술을 사용할 때 오버헤드가 발생한다. IoT 게이트웨이와 같이 제한된 자원을 가진 기기에서 이러한 오버헤드는 성능 저하를 발생시킬 수 있다. 모든 데이터에 대해 이러한 보안 기술을 사용한다면, 게이트웨이 기기와 연결된 IoT 기기가 많아질수록 오버헤드로 인한 성능 저하가 심각해진다. 개인이 사용하는 IoT 기기의 수가 크게 증가함에 따라 게이트웨이에 연결되는 기기의 수 또한 증가할 것이다. 따라서, 게이트웨이 기기에서 발생하는 성능 저하를 완화하기 위해 사용자에게 민감한 데이터만 분류하여 보안 기술을 적용할 필요가 있다.

본 논문에서는, 게이트웨이 기기에서 IoT 기기 연결 및 데이터 수집 시 ARM 트러스트존을 활용할 때 발생하는 오버헤드를 완화하기 위한 사용자 민감 데이터 분류 기법을 제안한다. 연결된 IoT 기기 특성과 기기로부터 수집한 데이터 특성을 분석하여 일반 실행공간에서 관리할지 안전한 실행공간에서 관리할 지를 결정한다. 또한 제안한 기법 적용 전과 후의 게이트웨이 기기 성능을 비교 분석한다. IoT 게이트웨이 환경으로 ARM 트러스트존을 활용하기 위해 BCM2837 64bit ARMv8 프로세서를 기반으로 한 Raspberry Pi 3 B Model을 사용하였다.

Ⅱ. 관련 연구

1. ARM 트러스트존

ARM 트러스트존은 ARMv7 이상의 Cortex-A 프로파일을 사용하는 ARM 아키텍처 기반 프로세서에서 제공하는 하드웨어 보안 기능이다. 하드웨어 및 소프트웨어 자원을 일반적인 실행환경을 의미하는 Normal world와 신뢰할 수 있는 실행환경을 의미하는 Secure world로 분할하여 격리된 실행공간을 제공한다. 두 실행공간 간의 안전한 전환을 위해 모니터 모드를 제공하고 있다[10].

Secure world 코드를 실행 중에는 모든 리소스에 접근이 가능하지만, Normal world 코드가 실행 중인 경우 Secure world의 리소스에 접근을 제한한다. 실행 중인 코드가 어느 영역의 코드인지 확인하기 위해 CP15 coprocessor의 SCR(Secure Configuration Register)을 확인한다. SCR의 NS bit를 확인하는 것으로 실행 중인 코드가 어느 영역에 속해 있는지 확인할 수 있다.

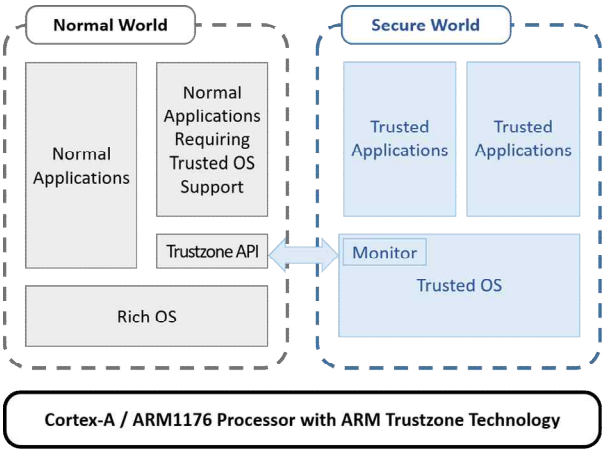

그림 1. ARM 트러스트존 구조

모니터 모드에서는 Secure world 메모리 영역에 접근이 가능하다. Secure world에서 사용하는 명령어 또한 사용 가능하기 때문에 Normal world와 Secure world간의 전환을 수행할 수 있다. 모니터 모드에서 수행 중인 world의 하드웨어 컨텍스트를 저장하고 전환할 world의 컨텍스트를 복원한다.

ARM 기반 리눅스 오픈소스를 지향하는 Linaro 단체는 ARM 트러스트존 기술을 활용하기 위한 OP-TEE 프로젝트를 공개했다[14]. OP-TEE는 부팅 과정부터 신뢰할 수 있는 실행환경을 구성한다. 신뢰할 수 있는 OS를 포함한 신뢰할 수 있는 실행환경을 먼저 구성한 뒤 일반 OS를 포함한 일반 실행환경을 구성한다. 신뢰할 수 있는 실행환경에서 다루는 데이터 및 명령어는 무결성과 기밀성을 보장한다. OP-TEE는 GlobalPlatform API 표준에 정의된 TEE Internal Core API Specification v1.1.2와 TEE Client API Specification v1.0을 지원한다. GlobalPlatform은 하드웨어 보안 기능을 활용하는 응용프로그램의 관리 및 배포를 용이하게 하기 위한 표준을 제시하고 있다[12].

2. IoT 데이터 보안 관련 연구

Christian 등은 IoT 플랫폼 보안을 위한 두 가지 하드웨어 기반 보안 기술을 비교하였다. ARM 트러스트존과 같은 프로세서 확장 기능에 기반한 보안 기술을 사용하는 경우 수행시간이 더 빠르고 개발 및 사용이 용이하다는 장점이 있으나 사이드 채널 어택과 같은 공격에 취약할 수 있다. 외부 보안 컨트롤러를 사용하는 경우 프로세서 확장 기능에 기반한 보안 기술보다 강력한 보안을 제공할 수 있지만, 별도의 디바이스 드라이버가 필요

하며, 개발 및 사용이 복잡하고 수행시간이 오래 걸린다는 단점이 있다[8].

Arijit 등은 [6]에서 IoT 환경 보안 요구사항을 정의하였다. 또한 이를 만족하기 위한 다양한 솔루션을 제안하였다. 제안한 솔루션 중의 하나로 트러스트존의 보안 부팅을 통하여 안전한 실행 환경을 제공하고, 악의적인 조작을 피할 수 있는 방법을 제시하였다.

Robert 등은 IoT 기기의 악의적인 조작 가능성을 제시하며 이를 해결하기 위한 플랫폼 구조를 제안하였다. 제안한 플랫폼은 데이터를 생성하는 IoT 기기와 생성된 데이터를 수집 및 관리하는 클라우드 서버로 구성되어 있다. ARM 프로세서를 사용하는 IoT 기기는 트러스트존을 활용하여 생성된 데이터의 보안성을 보장한다. 또한, Intel 프로세서 기반 클라우드 서버에서 SGX를 사용하여 전달받은 IoT 데이터가 악의적인 조작 위험성 없이 안전하게 관리할 수 있는 플랫폼을 제안하였다[9].

Chiara 등은 다양항 데이터가 유통되는 IoT 환경에서 일부 데이터가 악의적인 조작에 노출되어 있는 기기에서 발생할 수 있음을 지적하였다. 이러한 기기를 오염원으로 정의하고, 이를 추적하기 위해 IoT-Lysa를 사용한 제어 흐름 분석 기법을 제안하였다. 제안한 기법을 통해 변조된 센서 혹은 기기에서 발생한 데이터의 경로를 추적할 수 있다. 제어 흐름 분석을 통한 오염원 추적으로 보안 관련 문제가 있는 오염된 데이터를 분류할 수 있다[7].

Ⅱ. 사용자 민감 데이터 분류 플랫폼

1. 사용자 민감 데이터 분류 기법

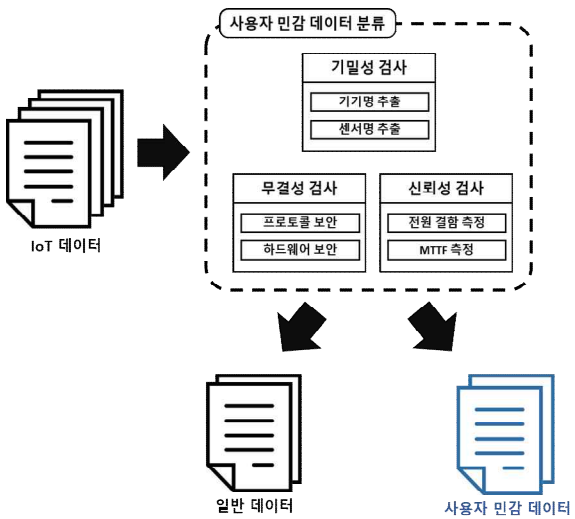

그림 2. 사용자 민감 데이터 분류 기법

사용자 민감 데이터를 분류하기 위해 세 가지 항목을 평가한다. 데이터를 생성하는 기기 특성과 데이터 특성을 분석하여 데이터 기밀성, 데이터 무결성, 데이터 신뢰성을 평가한다. 각 항목에 대해 평가한 값에 가중치를 설정하여 사용자 데이터 민감도를 계산한다.

가. 데이터 기밀성(Data Confidentiality)

데이터 기밀성은 수집한 데이터가 사용자가 공개하고 싶은 대상 혹은 인증된 대상에게만 공개되는지 평가하는 항목이다. 이는 해당 데이터 유출로 인해 사용자에게 발생하는 피해의 심각도를 반영한다. 개인의 신체 데이터, 금융 데이터, 계정 및 암호 등과 같이 무분별하게 유출될 경우 사용자에게 심각한 피해가 생길 수 있는 데이터를 기밀성이 높은 데이터로 평가한다.

제안하는 기법에서 데이터 기밀성을 평가하기 위해 개인이 특정 데이터를 기피하는 정도를 측정해야한다. [5]에서는 웹상에서 검색엔진 데이터를 기반으로 개인이 유출되는 것에 민감함을 나타내는 정보를 분석한 결과를 제시하고 있다. 제안하는 기법에서는 위 분석결과를 데이터 기밀성 평가 값으로 사용한다.

데이터 기밀성 평가 값을 추출하기 위해 데이터 특성과 데이터를 생성한 기기의 정보를 분석할 필요가 있다. 제안하는 기법을 적용한 플랫폼에서는 데이터 기밀성을 측정하기 위해 새로운 IoT 기기가 연결되었을 때 기기명과 연결된 기기에서 생성하는 센서명을 분석하여 데이터 기밀성을 평가한다.

나. 데이터 무결성(Data Integrity)

데이터 무결성은 데이터가 플랫폼으로 전달되는 과정에서 변조되지 않고 일관성 및 정확성이 보장되는지 평가하는 항목이다. 전달받은 데이터가 변조되었을 가능성이 높을 경우 낮은 평가 값을 가지게 된다. 데이터 무결성을 평가하기 위해 데이터 전달경로에 있는 기기들의 하드웨어 보안 제공 여부 및 프로토콜 보안성을 분석한다. 표 1은 제안하는 기법에서 데이터 무결성을 평가하기 위해 사용되는 표기 및 정의를 설명한다.

표 1. 데이터 무결성 평가 표기

표기	정의
SOURCE	데이터가 생성된 기기(센서 혹은 액추에이터)
SINK	데이터를 저장 혹은 사용하는 최종 기기
N	SOURCE부터 SINK까지 데이터가 경유하는 기기의 개수
$P(E_n)$	n번째 기기의 데이터 무결성 보장 확률
$P(I_n)$	SOURCE로부터 n번째 기기까지 데이터가 전달되었을 때, 데이터 무결성이 보장될 확률

하드웨어 보안 제공 여부를 분석하기 위해 데이터 관리시 신뢰할 수 있는 실행 환경을 제공하는지 확인한다. 이에 대한 정보는 하드웨어 정보 및 펌웨어 버전 분석을 통해 추출할 수 있다고 가정한다. 신뢰할 수 있는 실행환경을 통해 데이터를 관리할

경우 일반 실행환경에서 관리할 경우보다 높은 보안성을 제공할 수 있으므로 큰 값의 가중치를 설정한다.

프로토콜은 기존 IoT 환경에서 주로 사용하는 응용프로토콜을 대상으로 보안성을 분석하여 가중치를 설정한다. 예를 들어, CoAP(Constrained Application Protocol) 프로토콜은 DTLS(Datagram Transport Layer Security) 사용을 표준 보안 사항으로 명시하고 있다. CoAP에서 명시하고 있는 사양의 DTLS는 3072-bit RSA key를 사용한 것과 같은 정도의 보안을 제공할 수 있다[11]. 높은 보안성을 제공하는 CoAP에 큰 값의 가중치를 설정한다.

반면, MQTT(Message Queue Telemetry Transport) 프로토콜은 표준으로 명시하고 있는 보안 사항이 없다. MQTT 3.1.1 표준에서 MQTT의 보안 취약점을 제시했으나, 이를 해결하는 것은 개발자의 책임이라고 밝히고 있다[15]. 구현에 따라 보안성이 의존적이기 때문에 보안성이 높지 않은 것으로 평가하여 낮은 값의 가중치를 설정한다.

그 외의 프로토콜로 XMPP(eXtensible Messaging and Presence Protocol)는 SASL(Simple Authentication and Security Layer), TLS(Transport Layer Security)가 표준 보안 사항으로 명시되어 있다[16]. 하드웨어 보안 분석과 프로토콜 분석을 통해 IoT 기기의 무결성 보장 확률을 계산한다. 제안하는 기법에서 사용한 가중치 설정은 표 2와 같다.

표 2. $P(E_n)$계산을 위한 가중치 설정

하드웨어 보안 분석	가중치	프로토콜 분석	가중치
일반 실행 환경에서 관리	0.5	CoAP	1.0
		XMPP	0.7
신뢰할 수 있는 실행 환경 제공	1.0	MQTT	0.5

n번째 기기에서 데이터의 무결성 보장 확률 $P(I_n)$을 구하기 위해 (n-1)번째 기기까지 전달된 데이터의 무결성 보장 확률 $P(I_{n-1})$과 (n-1)번째 기기의 데이터 무결성 보장 확률 $P(E_{n-1})$이 필요하다. $P(I_n)$을 계산하는 수식은 다음과 같다.

$$P(I_n) = \begin{cases} P(E_0) & \text{if } n < 2 \\ P(E_{n-1})P(I_{n-1}) & otherwise \end{cases} \quad (1)$$

그림 3은 게이트웨이 기기를 통해 데이터가 전달되는 IoT 환경에서 데이터 무결성 평가의 예시이다. 평가를 수행하는 게이트웨이(n=N=2) 상에서 $P(I_2)$를 계산하여 데이터 무결성 평가 값으로 사용한다. $P(E_1)$을 하드웨어 분석 및 프로토콜 분석을 통하여 계산한다. $P(I_1)$은 엣지 게이트웨이(n=1)에서 전달받아 $P(I_2)$를 계산한다. 결과적으로 계산된 $P(I_2)$는 $P(E_1)P(E_0)$와 같다.

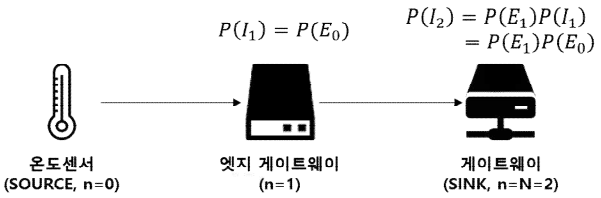

그림 3. 데이터 무결성 평가 예시

다. 데이터 신뢰성(Data Reliability)

하드웨어 결함을 측정하여 데이터를 생성하는 기기에 지속적인 접근을 보장하는지에 대한 평가한다. 하드웨어 결함이 발생하여 지속적으로 접근할 수 없는 경우 중요도가 낮은 기기로 간주하여 생성한 데이터가 사용자에게 민감하지 않은 것으로 평가한다.

데이터 신뢰성 평가에는 MTTF(Mean Time To Failure)를 측정하여 사용한다. MTTF는 정상적인 시스템이 결함이 발생하기까지의 평균 시간을 의미한다. 그림 4와 같은 전원 결함 측정 결과를 가진 기기에서 MTTF를 계산하는 방법은 수식(2)와 같다.

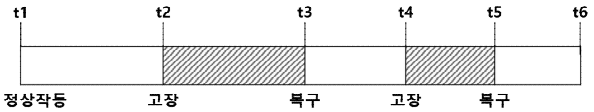

그림 4. MTTF(Mean Time To Failure) 측정

$$MTTF = \begin{cases} t_2 - t_1 & \text{if } t_2 \le t < t_4 \\ Average(t_2 - t_1, t_4 - t_3) & \text{if } t_4 \le t < t_6 \\ Average(t_2 - t_1, t_4 - t_3, t_6 - t_4) & \text{if } t_6 \le t \end{cases} \quad (2)$$

특정 시각 t에 수집한 데이터의 신뢰성 $R(t)$는 지수 분포를 사용하여 계산한다. λ는 고장률을 의미하며 MTTF 값의 역수를 취한 값이다. $f(t)$는 고장 확률 함수를 의미한다.

$$\lambda = \frac{1}{MTTF}$$
$$f(t) = \lambda e^{-\lambda t} \quad (3)$$

처음 플랫폼과 연결된 기기는 결함이 생길 때까지 MTTF를 구할 수 없다. 따라서 초기 MTTF 값을 설정해야한다. [4]에 따르면, 하드웨어적인 고장으로 인한 결함 MTTF는 평균 10년으로 측정하고 있다. 그러나, IoT 환경에서는 네트워크 연결망의 문제, 차폐물, 간섭 등 여러 요소로 인하여 결함이 발생할 수 있기 때문에 7일 미만의 MTTF를 가지는 것으로 측정하고 있다. 따라서, 게이트웨이와 연결된 IoT 기기의 초기 MTTF를 7일(=168시간)로 설정한다. 고장 확률 함수 $f(t)$를 통해 데이터 신뢰성 $R(t)$를 구하기 위한 수식은 다음과 같다.

$$R(t) = \int_t^{\infty} f(t)dt = e^{-\lambda t} \quad (4)$$

MTTF 값이 초기값인 168일 경우 t에 대한 $R(t)$ 값을 그린 도표는 다음과 같다. 결함 측정을 통해 MTTF 값이 변동될 경우 $R(t)$ 그래프 또한 변경되게 된다.

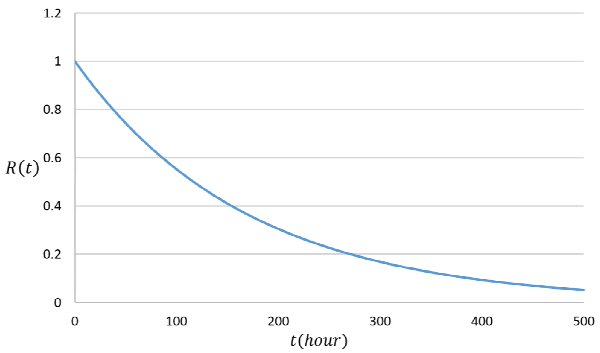

그림 5. 데이터 신뢰성 $R(t)$ (MTTF=168)

라. 사용자 데이터 민감도 계산

세 가지 평가를 수행한 뒤 사용자 데이터 민감도는 수식(5)와 같이 계산한다. 각 평가 값에 가중치를 곱한 값의 합으로 사용자 데이터 민감도를 계산한다. 이때, 세가지 평가 항목 중 어떠한 평가 지표 값을 더 비중 있게 반영할 것인지 가중치를 동해 결정한다. 예를 들어, 다양한 종류의 데이터가 유통되는 스마트 환경에서는 수집한 정보의 특성에 따라 민감도를 분류하기 위해 데이터 기밀성 가중치를 크게 설정할 필요가 있다.

$$Sensitivity = w_i * Integrity + w_r * Reliability + w_c * Confidentiality \quad (5)$$

세 가지 평가 지표가 균형 있게 민감도 계산에 반영될 수 있는 가중치 값을 설정하기 위한 시뮬레이션을 진행하였다. 실험 데이터 설정을 위해 2000개의 데이터에 데이터 기밀성 상위 40개를 골고루 분포하였다. 데이터 무결성은 시뮬레이션 환경에서 데이터가 여러 기기를 거치지 않는다는 가정하에 [0.25, 1.0] 사이 난수를 생성하였다. 마지막으로 데이터 신뢰성은 모든 기기의 MTTF를 초기 값인 168로 설정하고 t 값을 0시간에서 평균 수명인 168시간 사이로 설정하여 [0.3679. 1.0000] 사이의 난수를 생성하였다.

실험 데이터에 대하여 제안한 사용자 민감 데이터 분류 기법을 적용하였을 시 평가 지표당 가중치에 따른 영향도를 측정한다. 측정 방법은 2000개의 데이터 중 민감한 데이터로 분류된 1000개 데이터에 각 평가 지표 값 상위 1000개 데이터가 포함된 비율을 측정한다. 예를 들어, 민감한 데이터 1000개에 데이터 기밀성 상위 1000개 데이터 중 600개가 포함되어 있다면 영향도를 0.6으로 측정한다.

표 3. 사용자 민감 데이터 분류 평가 값 가중치별 영향도

w_i	w_r	w_c	무결성 영향도	신뢰성 영향도	기밀성 영향도
1.0	1.0	1.0	0.813	0.659	0.628
1.0	1.0	1.5	0.801	0.646	0.675
1.0	1.0	2.0	0.793	0.638	0.706
1.0	1.0	2.5	0.775	0.625	0.741
1.0	1.0	3.0	0.761	0.617	0.770
1.0	1.5	1.0	0.747	0.728	0.611
1.0	1.5	1.5	0.746	0.709	0.651
1.0	1.5	2.0	0.743	0.696	0.680
1.0	1.5	2.5	0.740	0.685	0.711
1.0	1.5	3.0	0.735	0.668	0.739
1.0	2.0	1.0	0.697	0.786	0.582
1.0	2.0	1.5	0.705	0.761	0.621
1.0	2.0	2.0	0.706	0.743	0.656
1.0	2.0	2.5	0.705	0.731	0.685
1.0	2.0	3.0	0.706	0.714	0.710
1.0	2.5	1.0	0.666	0.821	0.568
1.0	2.5	1.5	0.671	0.800	0.607
1.0	2.5	2.0	0.668	0.786	0.637
1.0	2.5	2.5	0.671	0.770	0.663
1.0	2.5	3.0	0.680	0.752	0.684

2. 사용자 민감 데이터 분류 플랫폼

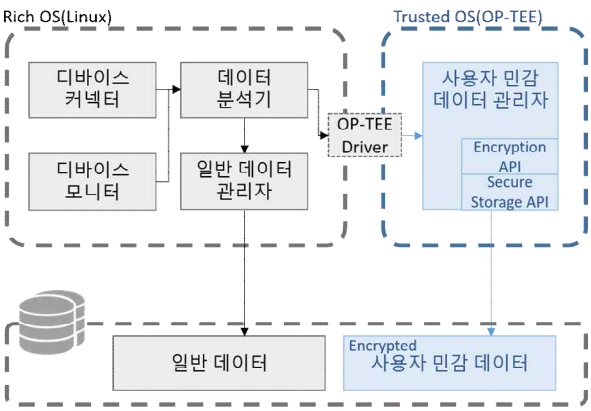

그림 6. 사용자 민감 데이터 분류 플랫폼

본 논문에서 제안하는 사용자 민감 데이터 분류 기법을 실험하기 위한 플랫폼의 구조는 그림 6과 같다. OP-TEE 오픈소스를 기반으로 트러스트존을 통해 신뢰할 수 있는 실행환경을 제공하는 플랫폼을 구성하였다.

Rich OS는 트러스트존을 사용하지 않는 일반 실행 공간에 위치한다. 디바이스 커넥터, 디바이스 모니터, 데이터 분석기, 일반 데이터 관리자 모듈은 일반 실행 공간에서 RichOS를 통해 실행된다.

디바이스 커넥터는 플랫폼과 연결된 IoT 기기와의 연결을 관리

하고 전달받은 메시지를 파싱한다. 등록에 관한 처리는 디바이스 커넥터에서 수행하며, 수집한 데이터에 관한 메시지는 데이터 분석기로 전달한다.

디바이스 커넥터를 통해 플랫폼과 연결된 디바이스는 디바이스 모니터를 통해 주기적으로 전원 결함 상태를 확인한다. 하트비트 메시지를 주기적으로 보내 전원 결함 상태를 확인하고 결함 데이터를 데이터 분석기로 전달한다.

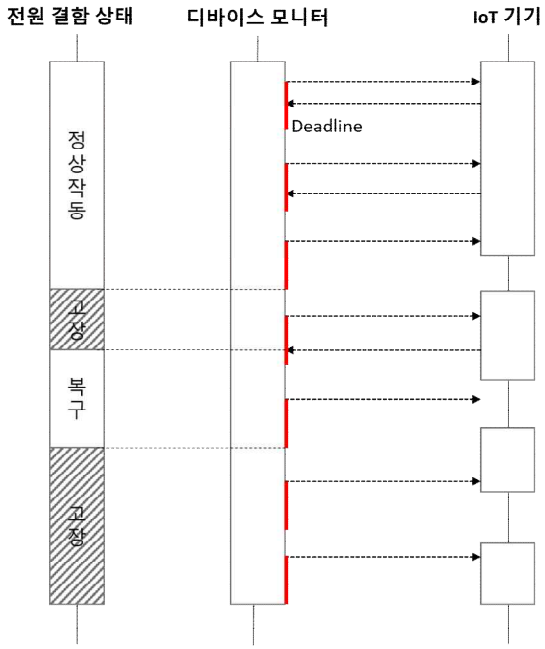

그림 7. 디바이스 모니터 전원 결함 검사

데이터 분석기는 디바이스 커넥터와 디바이스 모니터에서 전달받은 데이터를 기반으로 사용자 민감 데이터 분류를 수행한다. 데이터 분석기에서 민감하지 않은 데이터로 분류된 경우 일반 데이터 관리자로 전달하여 일반 실행 공간에서 데이터를 저장하고 관리한다.

```
    procedure DataAnalyzer(is_registration, recv)
1       if is_registration then
2           integrity <- 0
3           confidentiality <- 0
4           reliability <- 0
5       else
6           dev_info <- GetDevInfo(recv.dev_info)
7           integrity <- dev_info.integrity
8           confidentiality <- dev_info.confidentiality
9           goto GetReliability
10  GetIntegrity :
11      before_integrity <- recv.dev_info.integrity
12      if before_integrity = -1 then
13          before_integrity = 0.5
14      device_integrity = CalIntegrity(recv.dev_info)
15      integrity = before_integrity * device_integrity
16  GetConfidentiality :
17      confidentiality <- recv.dev_info.confidentiality
18      if confidentiality = -1 then
19          confidentiality <- CalConf(recv.dev_info.device_name
20                                     ,recv.dev_info.sensor_name)
21  GetReliability :
22      if is_registration then
23          mttf <- recv.dev_info.mttf
24          if mttf = -1 then
25              mttf = 168
26      else
27          mttf <- dev_info.mttf
28      reliability <- CalReliability(mttf)
29
30      SaveDB(integrity, confidentiality, reliability)
31      sensitivity = CalSensitivity(integrity,confidentiality,reliability)
32      if sensitivity < threshold then
33          SensitiveDataManager(recv)
34      else
35          NormalDataManager(recv)
36  end
```

그림 8. 데이터 분석기 의사코드

계산된 민감도가 설정된 threshold을 초과하는 데이터는 사용자 민감 데이터로 분류한다. 사용자 민감 데이터는 데이터 분석기에서 사용자 민감 데이터 관리자로 전달한다. TEE Client API를 통해 Trusted OS상의 사용자 민감 데이터 관리자와 통신한다. 사용자 민감 데이터 관리자에서는 민감한 데이터를 안전하게 관리하기 위해 암호화 기능을 제공한다. Secure Storage API를 사용하여 암호화에 사용된 키를 안전하게 관리한다.

Ⅲ. 실험 결과

본 논문에서 실험에 사용한 게이트웨이 플랫폼 환경은 표 4와 같다. 실험용 하드웨어는 Raspberry Pi 3 B Model을 사용한다.. 신뢰할 수 있는 실행 환경을 위한 Trusted OS는 OP-TEE v3.4.0 버전을 사용한다. 일반 실행 환경을 위한 Rich OS로 Linux v4.13 버전을 사용한다.

실험은 제안한 기법을 적용한 플랫폼에서 데이터를 처리하는 성능을 측정한다. 또한, 신뢰할 수 있는 실행 환경에서 트러스트존을 활용하여 모든 데이터를 처리했을 때 성능 및 일반 실행 환경에서 처리했을 때 성능 세 가지 성능을 구하여 비교한다. 마지막으로 사용자 민감 데이터 분류 기법에 의한 오버헤드를 측정한다.

표 4. 실험 환경

구분		설명
사용자 민감데이터 분류 플랫폼	하드웨어	Raspberry Pi 3 B Model
	CPU	Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
	메모리	1GB RAM
	저장공간	MicroSD 64GB
	Rich OS	Linux 4.13
	Trusted OS	OP-TEE 3.4.0

실험에 사용되는 데이터 패킷의 크기는 128bytes로 동일하다. 모든 데이터를 일반 실행 환경에서 별도의 암호화 작업 없이 처리한 경우, 연결된 IoT 기기의 개수가 늘어나도 크게 성능의 차이가 발생하지 않는다. 그러나, 모든 데이터를 신뢰할 수 있는 실행환경에서 암호화 작업을 거쳐 데이터를 처리할 경우, 연결된 IoT 기기의 개수가 늘어날수록 성능 저하가 더 심각해진다.

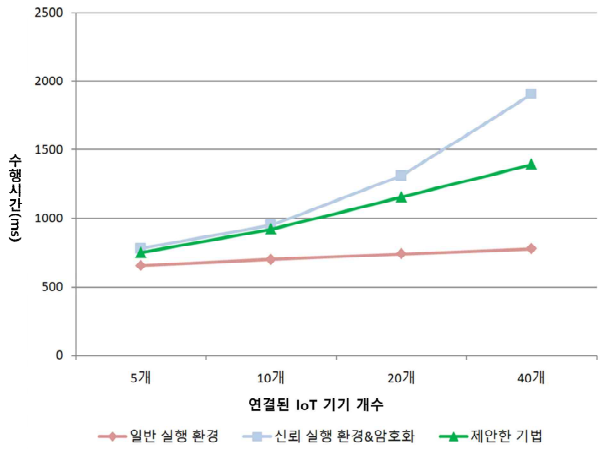

그림 9. 제안한 기법을 적용한 플랫폼 성능 비교

신뢰할 수 있는 실행 환경에서 트러스트존 기능을 사용하여 암호화할 때 일반 실행 환경에서 신뢰 실행 환경으로의 모드 전환이 발생한다. 이 때 전달할 수 있는 메시지의 크기가 512bytes로 한정되어 있기 때문에, 처리해야 하는 데이터가 많아질수록 성능 저하가 심각해진다. 이는 연결된 IoT 기기의 개수가 많아질수록, 데이터 크기가 커질수록 심각한 성능 저하를 발생시킬 위험이 있다.

위 실험에서는 제안한 기법을 통해 수집한 데이터의 절반을 사용자 민감 데이터로 관리하도록 threshold를 설정한다. 처리해야 하는 데이터가 많지 않은 경우, 제안한 기법은 트러스트존을 활용하여 모든 데이터를 암호화한 경우와 성능이 비슷하다. 그러나, 연결된 기기가 많아지고 처리해야 하는 데이터가 많아질수록 발생하는 성능 저하가 완화된 것을 확인할 수 있다.

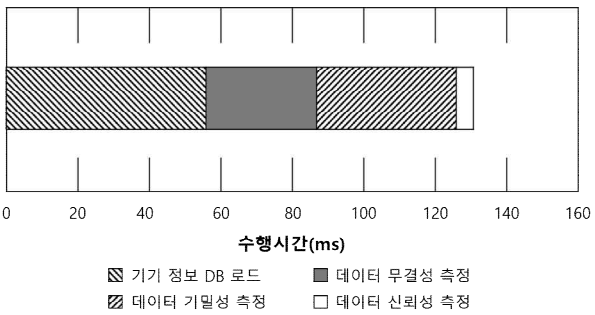

그림 10. 데이터 분석기 오버헤드 측정

제안한 플랫폼에서 사용자 민감 데이터 분류를 위한 데이터 오버헤드를 측정한다. 새로운 IoT 기기가 등록되었을 때 데이터 분석기가 수행되는데 134ms 정도가 소요된다. 가장 많은 시간이 걸린 부분은 기기 등록 시 중복 검사 및 갱신을 위한 기기 정보 DB 로드 시간이다. 제안한 기법을 적용하기 위해 발생하는 오버헤드는 기기 정보 DB 로드, 데이터 무결성 측정, 데이터 기밀성 측정의 영향이 큰 것을 확인할 수 있다. 따라서, 새로운 IoT 기기가 다수 연결될 때 성능 저하가 발생할 수 있으나 이미 등록된 기기에서 데이터 수집을 하는 경우 데이터 신뢰성 측정으로 인한 오버헤드만 발생하기 때문에 플랫폼 전체 성능에는 크게 영향을 주지 않는다.

Ⅳ. 결 론

본 논문에서는 IoT 환경에서 효율적인 트러스트존 활용을 위한 사용자 민감 데이터 분류 기법을 제안하였다. 제안한 기법을 통해 모든 데이터를 트러스트존을 통해 암호화하고 관리하는 것이 아니라 사용자에게 민감한 데이터만을 추출하여 안전하게 관리하였다. 제안한 기법이 적용된 플랫폼을 구현하여 실험을 진행하였다. 이전의 기존 연구와 같이 모든 데이터를 트러스트존을 통해 관리하였을 시 동시에 처리해야 하는 데이터가 많아질수록 성능 저하가 심각해지는 것을 확인하였으며, 제안한 기법을 적용하였을 시 트러스트존으로 전달해야 하는 데이터양을 효과적으로 줄이고 성능 저하를 완화하였음을 확인하였다.

REFERENCES

[1] 김정녀, "안전한 스마트 단말을 위한 가상화 기반 도메인 분리 보안 플랫폼 구현," *스마트미디어저널*, 제5권, 제4호, 116-123쪽, 2016년 12월

[2] 김진보, 김미선, 서재현, "사물인터넷 서비스 접근 제어를 위한 리소스 서비스 관리 모델 구현," *스마트미디어저널*, 제5권, 제3호, 9-16쪽, 2016년 9월

[3] 윤기하, 박성모, "128비트 LEA 암호화 블록 하드웨어 구현 연구," *스마트미디어저널*, 제4권, 제4호, 39-46쪽, 2015년 12월

[4] Ivanovitch Silva, Rafael Leandro, Daniel Macedo, Guedes,Affonso Luiz, "A dependability evaluation tool for the Internet of Things," *Computers and Electrical Engineering*, vol. 39, no. 7, pp. 2005-2018, Oct. 2013.

[5] Xiaofeng Lu, Zhaowei Qu, Qi Li, Pan Hui, "Privacy Information Security Classification for Internet of Things Based on Internet Data," *International Journal of Distributed Sensor Networks*, vol. 11, no. 8, pp. 1-8, Aug. 2015.

[6] Arijit Ukil, JaydipSen, Sripad Koilakonda, "Embedded Security for Internet of Things," *Proc. of the 2nd National Conference on Emerging Trends and Applications in Computer Science(NCETACS 2011)*, pp. 50-55, Shillong, Meghalaya, Mar. 2011.

[7] Chiara Bodei, Letterio Galletta, "Tracking sensitive and untrustworthy data in IoT," *Proc. of the First Italian Conference on Cybersecurity(ITASEC 2017)*, pp. 38-52, Venice, Italy, Jan. 2017.

[8] Christian Lesjak, Daniel Hein, Johanne Winter, "Hardware-Security Technologies for Industrial IoT:TrustZone and Security Controller," *Proc. of the 41st Annual Conference of the IEEE Industrial Electronics Society(IECON 2015)*, pp. 2589-2595, Yokohama, Japan, Nov. 2015.

[9] Robert Pettersen, Havard D Johansen, Dag Johansen, "Secure Edge Computing with ARM TrustZone," *Proc. of the 2nd International Conference on Internet of Things, Big Data and Security(IoTBDS 2017)*, pp. 102-109, Porto, Portugal, Apr. 2017.

[10] ARM, "ARM Trustzone." https://developer.arm.com/technologies/trustzone (accessed Dec., 24, 2019).

[11] CoAP, "CoAP Specifications." https://coap.technology/ (accessed Dec., 24, 2019).

[12] GlobalPlatform, "GlobalPlatform Specifications." https://globalplatform.org/ (accessed Dec., 24, 2019).

[13] Intel, "Intel SGX." https://software.intel.com/en-us/sgx (accessed Dec., 24, 2019).

[14] Linaro. "About OP-TEE." https://optee.readthedocs.io/general/about.html (accessed Dec., 24, 2019).

[15] OASIS, MQTT Version 3.1.1. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html (accessed Dec., 24, 2019).

[16] XMPP, "About XMPP." https://xmpp.org/ (accessed Dec., 24, 2019).

저 자 소 개

허만우(준회원)

2017년 숭실대학교 컴퓨터학부 학사 졸업.
2019년 숭실대학교 컴퓨터학과 석사 졸업

<주관심분야 : 시스템 소프트웨어, 운영체제>

박기철(준회원)

2019년 선문대학교 전자공학과 학사 졸업.
2019년~현재 숭실대학교 컴퓨터학과 석사 재학

<주관심분야 : 시스템 소프트웨어, 운영체제>

홍지만(종신회원)

2003년 서울대학교 컴퓨터공학과 박사 졸업
2004년~2007년 광운대학교 컴퓨터공학과 교수.
2007년~현재 숭실대학교 컴퓨터학부 교수

<주관심분야 : 시스템 소프트웨어, 운영체제>