KIPS Tr. Comp. and Comm. Sys.
Vol.6, No.8 pp.329~342 pISSN: 2287-5891

https://doi.org/10.3745/KTCCS.2017.6.8.329

Design and Implementation of CoAP Authorization Framework Based on OAuth 2.0

Kyoung-Han Kim† · Hyun-Kyo Lim†† · Joo-Seong Heo†† · Youn-Hee Han†††

ABSTRACT

Recently, interest and investment in the Internet of Things (IoT) have increased significantly, and security issues are constantly being raised. As a solution, the IETF ACE Working Group is establishing the ACE framework standard, which is a new security framework for various constrained IoT environments based on the existing OAuth 2.0. However, additional work is required to apply the ACE framework, which proposes a new lightweight security system, to the existing Internet environment, and this additional cost is a factor that hinders the application of OAuth 2.0 to the IOT environment. Therefore, we propose an IoT authentication framework based on OAuth 2.0's existing development motivation, and implement a proposal framework based on CoAPthon and analyze its performance.

Keywords : IoT, OAuth 2.0, CoAP, CoAPthon

OAuth 2.0 기반 CoAP 인증 프레임워크 설계 및 구현

김 경 한† · 임 현 교†† · 허 주 성†† · 한 연 희†††

요 약

최근 사물인터넷에 대한 관심과 투자가 크게 증가하고 있으며 보안 측면에서 문제가 지속적으로 제기되고 있다. 그 해결책으로서 IETF ACE 워킹 그룹이 기존의 OAuth 2.0 기반으로 여러 제약적인 사물인터넷 환경에 적합한 새로운 보안 프레임워크인 ACE 프레임워크 표준을 제정 중에 있다. 그러나 새로운 경량 보안 체계를 제안하는 ACE 프레임워크를 기존 인터넷 환경에 적용하기에는 추가적인 작업이 필요하며, 이러한 추가적 비용은 IoT 환경에 OAuth 2.0의 적용을 저해하는 요인이 된다. 따라서 본 논문에서는 기존 인터넷 환경에서 활용되고 있는 보안 프로토콜인 DTLS를 기반으로 하고, OAuth 2.0의 기존 개발 동기에 맞춘 사물인터넷 인증 프레임워크를 제안하며, python 기반 오픈 라이브러리인 CoAPthon을 기반으로 제안 프레임워크 구현하며 성능을 분석한다.

키워드 : IoT, OAuth 2.0, CoAP, CoAPthon

1. 서 론

최근 ICT 분야에서 가장 주목 받는 기술 키워드인 사물인터넷[1]은 사물이 인터넷을 통해 서로 연결되어, 다양한 서비스를 만들어 내는 기술을 일컫는다. 사물인터넷에 대한 관심과 투자는 지속적으로 증가하고 있지만 사물인터넷의 확산에 더욱 박차를 가하기 위해서 해결해야 할 가장 큰 문제 중 하나는 보안(Security) 관련 문제이다. 사물인터넷의 대표적인 분야에 스마트 홈, 스마트 카, 스마트 빌딩, 웨어러블 기기 등이 있기 때문에 기기로부터 전송되는 정보들은 민감한 개인 정보를 포함하는 경우가 많다.

이러한 사물인터넷의 보안 문제 한계성 때문에 IoT 서비스를 제공하고 있는 사업체들은 자체적으로 개발한 웹 기반 응용이나 스마트폰 응용만이 그들이 설치한 IoT자원에 접근 가능하게 하고 있으며, 서드파티(3rd Party) 응용에게는 접근을 허용하지 않는 추세이다. IoT 서비스 업체 측에서 운영하고 있는 자사의 IoT자원에 접근 가능한 새로운 응용들을 서드파티 업체에서 다양하게 개발될 수 있다면 결과적으로 IoT 서비스 업체에서 많은 투자를 통하여 운영하는 IoT 서비스에 대한 수요와 이윤도 증가할 것으로 예상된다.

이미 일반적인 웹서비스 분야에서는 이와 같은 서드파티

※ 이 논문은 2016년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업(No. NRF-2016R1D1A3B03933355)과 2016년도 한국기술교육대학교 교수 교육연구진흥과제 지원에 의하여 수행된 연구임.
† 준 회 원 : 한국기술교육대학교 컴퓨터공학부 석사
†† 준 회 원 : 한국기술교육대학교 컴퓨터공학부 석사과정
††† 종신회원 : 한국기술교육대학교 컴퓨터공학부 정교수
Manuscript Received : December 28, 2016
First Revision : April 7, 2017
Accepted : May 2, 2017
* Corresponding Author : Youn-Hee Han(yhhan@koreatech.ac.kr)

응용에 대한 자사의 자원 접근을 인가하기 위하여 OAuth 2.0 인증 프레임워크[2]가 활발히 이용되고 있다. 특히, 구글, 페이스북, 네이버, 다음카카오와 같이 많은 자원을 지닌 서비스 업체에서는 자신들의 자원에 접근하기를 원하는 서드파티 응용들에게 OAuth 2.0인증을 요구하고 있으며, 이를 기반으로 해당 자원들의 소유자 허락을 득하여 그 자원들을 개방하고 있다.

한편, IETF (Internet Engineering Task Force)에서는 2014년 1월에 ACE (Authentication and Authorization for Constrained Environments) 워킹 그룹[3]을 결성하여, 상대적으로 열악하고 각종 제약을 가진 소형기기의 통신에 OAuth 2.0을 적용한 보안 프레임워크 표준을 개발하기 시작하였다.

그러나 OAuth 2.0 프레임워크가 TLS에 의존적인 것과는 달리 ACE 프레임워크에서는 자체적으로 각 참여 주체가 보내는 메시지에 대한 서명을 포함한 인증 및 검증 방법, 메시지 암호화 방안 등을 모두 담고 있다. 즉, ACE 프레임워크는 OAuth 2.0의 기존 개발 동기를 고수하며 개발되기보다는, 다른 계층의 도움 없이 오로지 응용 계층으로만 해결이 가능한 새로운 경량 보안 체계를 제안하고 있다. 이러한 보안 체계는 새로운 응용 계층 기반 보안 체계임에 따라 확산되기에 시간 필요하며 ACE의 보안 요소 기술인 CBOR (Concise Binary Object Representation)은 아직 범용적으로 아직 활용되고 있지 않다.

현재 각종 제약을 지닌 소형기기에 적용 가능한 TLS/DTLS에 대한 연구[4, 5]도 활발하게 진행되고 있기 때문에, OAuth 2.0의 기존 개발 동기에 맞춘 사물인터넷 인증 프레임워크 설계는 계속하여 연구될 필요가 있다. 이에 본 논문에서는 TLS가 소형기기에 적용될 수 있다는 가정하에서 기존 OAuth 2.0 프레임워크의 개발 동기를 준수하는 OAuth 2.0 기반 CoAP 보안 프레임워크 설계를 제안하고 Python 기반 라이브러리를 통해 제안 프레임워크를 구현한다.

2. 관련 연구

2.1 CoAP

CoAP은 IETF (Internet Engineering Task Force)에서 표준화한 프로토콜로 상대적으로 적은 전력을 소모하고, 신뢰성 있는 통신을 제공함으로써, 사물인터넷 환경에서 다양한 서비스를 제공하기 위한 핵심 프로토콜이다[6].

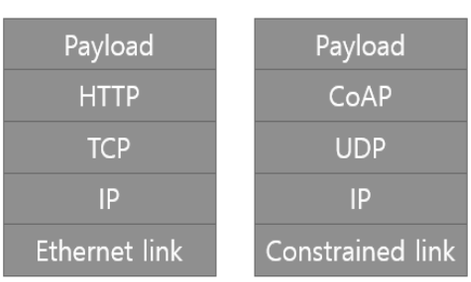


Fig. 1. HTTP and CoAP Structure

CoAP 프로토콜은 제약 네트워크 내의 소형 노드와 인터넷의 서버와 통신을 할 때 사용될 수 있으며, 소형 노드끼리 통신하는데 사용될 수도 있다. 또한 소형 노드는 프록시(Proxy)를 통하여 기존 HTTP 서비스와 연동될 수 있다. CoAP 프로토콜은 노드의 메모리 및 성능 제약, 네트워크의 제약 등 다양한 요구사항을 만족시킬 수 있도록 설계되었다. Fig. 1에서 알 수 있듯이 CoAP 프로토콜은 메모리 크기와 메시지 크기를 줄이기 위하여 HTTP가 전송 계층 프로토콜로 TCP를 사용하는 것과는 달리 UDP를 사용하고, 바이너리 인코딩을 사용하여, HTTP에 비해 10분의 1이하로 전송량을 줄였다. CoAP은 레스트풀(RESTful) 구조로, HTTP와 그 사상이 같기 때문에, HTTP 프록시 연동 및 기존 웹서비스와 쉽게 연계가 가능하다[7]. 또한, IETF CoRE 워킹그룹은 CoAP 기본 규격인 RFC 7252 이후로 다양한 확장 및 보안 규격을 논의하고 있다. 대표적인 확장 기능으로는 블록 전송(Block-wise Transfer) 기능[8], 자원 관찰 (Observing Resources) 기능이 있다.

2.2 OAuth 2.0

OAuth 2.0은 인터넷에 존재하는 자원에 대해 임의의 제3자 웹, 모바일 앱, 또는 어플리케이션이 접근하기 위한 권한 인증 절차를 표준화한 인증 프레임워크이다 [2].

OAuth의 시작은 2006년에 Twitter OpenID 개발자인 Blaine Cook과 소셜 북마크 서비스인 Ma.gnolia의 개발자가 OpenID를 지닌 사용자들로 하여금 권한을 인증해 주고자 하는 방식을 논의하던 때로 거슬러 올라간다. Twitter와 Ma.gnolia 두 회사의 개발자들은 API 접근 위임에 대한 표준이 필요하다는 사실을 알게되고, 2007년 4월 비공식적인 OAuth 논의 그룹을 만든 뒤 OAuth 드래프트 제안서를 만들어 공유했다. 이후, 2008년 73차 미네소타 IETF 회의에서 OAuth BoF (Bird of Feather)가 만들어졌고, 2010년에 OAuth 1.0 공식 표준안이 RFC 5849로 발표되었다[9].

OAuth 1.0이 Twitter를 주축으로 비공식적으로 논의되던 인증 및 권한 부여 프로토콜이 IETF라는 국제 표준으로 인정되었다면, 이후 IETF OAuth WG이 만들어진 이후에는 다양한 회사들이 적극적으로 참여하여 IETF에서 약 2년간의 공식적인 미팅을 통하여 좀더 성숙한 형태의 OAuth 2.0 표준인 RFC 6749 [2]를 발표하였다. OAuth 1.0은 그 자체로 Protocol을 정의하고 있지만 OAuth 2.0은 OAuth 1.0과의 호환성을 고려하지 않고 좀 더 인증 절차를 간단히 만든 프레임워크(Framework) 형태로 발표가 되었다. 즉, OAuth 2.0은 프레임워크이기 때문에, OAuth 2.0을 도입하는 회사에서 자체적으로 자신들에게 좀 더 맞는 형태로 구현을 하여 사용할 수 있는 융통성이 존재한다. 그러한 융통성 때문에 OAuth 2.0은 정식 RFC가 되기 전부터 Facebook, Google, Microsoft 등 주요 인터넷 서비스 업체에서 OAuth 2.0을 사용하기 시작하였다.

OAuth 2.0의 권한 인증 절차에 참여하는 주체들의 역할 구성 형태는 “3-legged OAuth”라고 불리지만, 자원 소유자

(Resource owner), 클라이언트(Client), 자원 서버(Resource server), 권한 서버(Authorization server), 이렇게 4가지 주체로 이루어져 있다. 자원 소유자는 보호 자원에 대한 접근 권한을 부여할 수 있는 개체로 일반적으로 이용자를 나타낸다. 클라이언트는 자원 서버에 보호 자원을 요청하고 관련 서비스를 제공하는(서드-파티) 어플리케이션이고, 자원 서버는 보호된 자원에 대한 서비스 API를 제공하는 서버이며, 권한 서버는 클라이언트가 보호된 자원에 대한 제한된 접근을 할 수 있도록 자원 접근 권한을 관리하는 서버이다.

OAuth 2.0를 이용하면 클라이언트는 자원 서버가 가지고 있는 특정 자원 소유자의 자원에 제한적인 접근 권한을 얻을 수 있다. 자원 소유자의 권한 승인 하에 권한 서버를 통해서 클라이언트에게 접근 토큰(Access Token)이 발급된다. 클라이언트는 접근 토큰을 사용해서 보호되고 있는 자원에 접근하는 대신, 특정 범위와 수명 등 접근에 제한이 있다.

2.3 IETF ACE

사물인터넷의 보안 문제를 해결하기 위해서 IETF (Internet Engineering Task Force)에서는 2014년 1월에 ACE (Authentication and Authorization for Constrained Environments) 워킹 그룹을 결성하여, 각종 제약을 가진 소형기기의 통신에 OAuth 2.0을 적용한 보안 프레임워크 표준을 개발하기 시작하여 2016년 10월에 draft-ietf-ace-oauth-authz-04을 발표하기에 이르렀고 현재에도 활발하게 표준화를 진행하고 있다[10].

ACE 프레임워크는 한정된 자원을 지닌 기기가 지닌 자원에 대한 접근에 대해 인증 및 권한 부여 보안 프레임워크이다. OAuth 2.0 프레임워크가 TLS에 의존적인 것과는 달리 ACE 프레임워크에서는 자체적으로 각 참여 주체가 보내는 메시지에 대한 서명을 포함한 인증 및 검증 방법, 메시지 암호화 방안 등을 모두 담고 있다. 즉, 하부 계층과는 독립적인 응용 계층의 보안 절차 및 방안을 자체적으로 제정하고 있다. 그러므로 TLS의 도움없이도 ACE 프레임워크만으로 암호화, 인증 및 권한 부여까지 가능한 모든 보안 이슈가 해결되도록 설계하고 있다.

ACE 프레임워크를 이루는 주요한 4가지 구성요소는 다음과 같다.

- OAuth 2.0
- CoAP (Constrained Application Protocol): 전송 지연이 높고, 패킷 손실률이 높은 제약이 있는 네트워크 환경 및 노드 환경을 고려한 HTTP 스타일의 응용 계층 데이터 전달 프로토콜[6]
- CBOR (Concise Binary Object Representation): 최대한 작은 코드 사이즈, 작은 메시지 크기 그리고 버전 협상이 필요 없는 확장성이 가능하도록 설계된 데이터 포맷[11]
- COSE (CBOR Object Signing and Encryption): CBOR 메시지를 기반으로 한 서명 및 암호화 포맷[12]

2.4 IoT 환경에서의 DTLS

TLS/DTLS는 전송 계층 보안 프로톨로서 클라이언트/서버 응용 프로그램이 네트워크로 통신을 하는 과정에서 도청, 간섭, 위조를 방지하기 위해서 설계되었다. 그리고 암호화를 해서 최종단의 인증, 통신 기밀성을 유지시켜준다. 따라서 현재 네트워크 환경에서 사용자의 인증, 데이터의 무결성 및 기밀성을 보장하기 위한 보안 시스템 개발 시 그 활용도가 매우 크다.

한편, IoT환경에서는 안전한 통신을 하기 위해서는 메시지 암호화와 인증을 함께 제공하는 암호화 알고리즘이 요구되고 있다. 그러나 IoT 환경은 경량화 및 저전력을 요구하는 디바이스의 특성상 DTLS를 바로 적용하기에는 무리가 있다.

왜냐하면, DTLS에서 사용되는 대표적인 암호 알고리즘은 AES 알고리즘이라고 할 수 있는데, 이 AES 알고리즘은 자원이 제한적인 IoT 디바이스에 사용하기에는 비용과 성능면에서는 부적합하다[4].

그렇기 때문에, IETF ACE 프레임워크에서는 전송 계층 보안의 대체로 응용 계층 보안인 CBOR 기반의 보안 메시지 형식인 COSE을 사용한다.

하지만, 기존의 네트워크 환경에서 많이 활용되고 있는 DTLS가 아닌 새로운 보안 체계를 제안하는 IETF ACE 프레임워크는 기존 인터넷 환경에 적용하기에는 추가적인 작업을 필요로 하며, 이러한 추가적 비용은 IoT 환경에 OAuth 2.0의 적용을 저해하는 요인이 된다. 뿐만 아니라 현재 각종 제약을 지닌 IoT 디바이스에 적용 가능한 TLS/DTLS에 대한 연구가 활발하게 진행되고 있기 때문에, IoT 환경의 경량 DTLS를 기반으로 한 권한 인증 프레임워크에 대한 연구도 진행해야 할 가치가 있다.

2.5 CoAPthon

현재 다양한 CoAP 라이브러리가 존재하며 대표적으로 Java를 기반으로 하는 Californuim[14] 및 Python을 기반으로 하는 CoAPthon[15]이 널리 알려져 있다. 본 논문에서는 Californium보다 개발이 더 용이한 CoAPthon을 활용하였다[16].

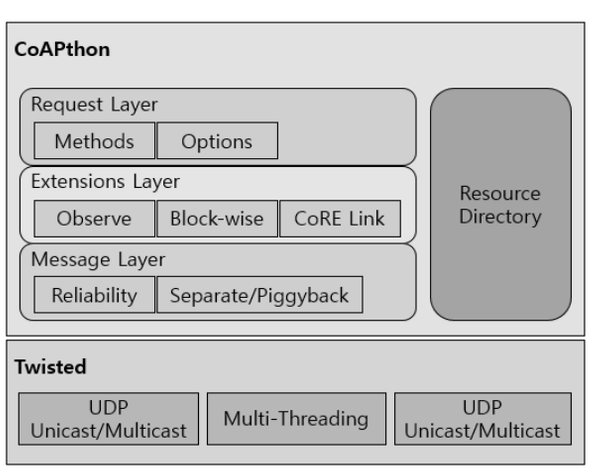


Fig. 2. CoAPthon architecture

Fig. 2는 CoAPthon 프레임워크 아키텍처를 나타내고 있으며 이 구조는 표준 문서인 RFC7252에 정의된 CoAP을 반영하였다. 또한 CoAPthon은 프로토콜 구현, 서버 프로그램 작성, 배포, 시스템 설치까지 거의 모든 과정을 실현가능한 Python 기반 네트워킹 엔진인 Twisted 프레임워크를 기반으로 하고 있다.

또한, Message Layer는 응답모드 및 옵션 관리, Request Layer는 CoAP 요청에 응답하기 위한 처리과정, Extension Layer는 확장 기능을 수행하는 역할을 맡고 있다.

3. 제안 프레임워크 설계

3.1 설계 고려사항

1) 자원서버와 권한 서버의 분리

현재의 일반적인 OAuth 2.0의 구성요소는 자원 소유자, 클라이언트, 자원 서버, 권한 서버이고, 자원 서버와 권한 서버는 구축 및 운영 방식에 따라 동일 또는 별도 서버로 존재 가능하다. 그러나 사물인터넷의 제약 환경에서는 기존의 OAuth 2.0과는 달리 권한 서버와 자원 서버가 동일 서버로 존재 하는 것은 불가능하다.

2) CoAP 추가옵션

제안 프레임워크에서는 사물인터넷의 제약 환경을 고려하여 CoAP을 기반으로 통신을 한다. 그러나 CoAP을 통해 클라이언트가 보호된 자원에 접근하기 위해, 혹은 기존 OAuth 2.0의 토큰 점검을 CoAP에 적용하기 위해서는 CoAP의 기존 옵션과는 별도로 새로운 옵션을 추가해야만 한다.

3) 자원서버의 토큰 점검 메시지 캐시

기존 OAuth 2.0에서는 네트워크의 트래픽 양과 노드의 부하를 줄이기 위해, 선택적으로 자원 서버에서 토큰 점검 메시지를 캐시(Cache)하여 추후 해당 토큰으로 보호된 자원에 접근 시 자원 서버가 자체적으로 토큰의 상태를 점검할 수 있게 한다[13]. 이러한 방식은 사물인터넷의 제약 환경에서는 매우 중요하므로 자원 서버의 토큰 점검 메시지 캐시는 제안 프레임워크에서는 필수사항이다.

4) 토큰 점검 메시지 캐시의 일관성 유지

토큰 점검 메시지 캐시 방식을 도입하면 자원 서버에서 캐시하고 있는 토큰 정보와 실제 토큰 정보를 관리하는 권한 서버에서의 토큰 정보의 일관성(Consistency)을 유지해야 하는 절차가 필요하다.

제안 프레임워크에서는 이러한 일관성 문제를 해결하기 위해 CoAP의 확장 기능인 자원 관찰(Observing Resources) 기능을 활용한다[18]. 자원 관찰 기능은 자원(예를 들어 토큰 자원)을 관리하는 측에서 자원의 상태가 변화할 때마다 해당 자원의 상태 변화에 관심이 있는 측으로 통지를 하는 기능이다.

예를 들어, 자원 소유자가 토큰을 철회를 요청하는 경우 권한 서버에서는 해당 토큰을 폐기시키고 해당 토큰에 관한 정보를 캐시하고 있는 자원 서버에게 갱신된 정보를 알려야만 한다.

이를 위해, 자원 서버는 토큰 점검 메시지를 권한 서버에게 요청함과 동시에 자원 관찰 등록(Registration)을 수행한다. 이후, 토큰 발급 시간 만료, 자원 소유자의 토큰 철회 등 토큰의 상태가 변화할 경우 권한 서버는 자원 서버에게 해당 토큰에 대한 상태 변화 통지 메시지를 전송한다. 이 메시지를 받은 자원 서버는 해당 통지 메시지를 해석하여 자신이 캐시한 토큰에 대해 적절한 작업을 수행한다. 이러한 방법에 의하여 자원 서버에 캐시된 토큰 정보는 늘 최신성이 유지되기 때문에 잘못된 토큰 정보로 인한 오류는 발생하지 않는다.

3.2 제안 프레임워크 구조

앞서 언급한 대로, ACE 프레임워크는 현재 표준화가 진행 중이며, 응용 계층만으로 모든 보안 이슈를 해결하기 위한 새로운 경량 보안 체계이다. 하지만, 제약사항이 있는 사물인터넷 노드에 적용 가능한 TLS에 대한 연구도 활발하게 진행되고 있기 때문에, 이러한 경량 TLS를 기반으로 보안 체계를 제안하는 기존 OAuth 2.0에 맞춘 사물인터넷 인증 프레임워크 설계도 계속하여 연구될 필요 있다. 따라서 본 논문에서는 앞 절에서 언급한 사항들을 고려하여, Fig. 3에서와 같은 구조를 가진 OAuth 2.0 기반 CoAP 인증 프레임워크를 제안하며 Table 1은 Fig. 3에 표시된 절차 (A)~(I)의 세부내용을 나타낸다.

제안 프레임워크에서는 자원 서버의 한정된 네트워크 및 계산 자원을 고려하여, 클라이언트와 자원 서버, 권한 서버와 자원 서버의 통신은 CoAP을 통해 이루어져야 하며, 그 통신 과정의 보안을 위해 DTLS를 사용한다. 그 외의 User-agent와

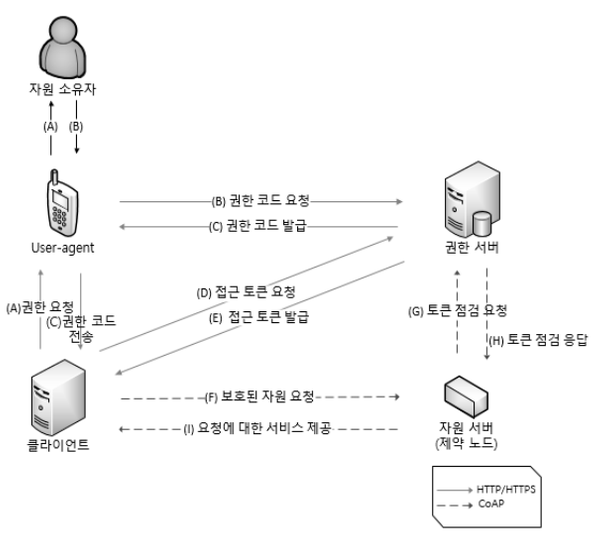


Fig. 3. Proposed Framework Architecture

Table 1. Explanation of the Proposed Framework Operation Flow

step	detail
(A) Authorization Request	The client requests authorization from the resource owner via the User-agent.
(B) Authorization Code Request	The resource owner requests the authorization code from the authorization server via User-agent.
(C) Authorization code Issue	If the authentication of the resource owner of the authorization server is valid, it generates an authorization code and sends the code to the User-agent.
(D) Access Token Request	The client that receives the authorization code through the User-agent requests the access token by transferring the client_id and client_secret to the authorization server based on the authorization code.
(E) Access Token Issue	When the authorization server determines that the client's request is valid, it generates an access token and sends it to the client.
(F) Protected Resources Request	The client requests authentication and resources to the resource server as an access token.
(G) Token Introspection Reques	The resource server sends a token check request message containing observing resource function option to the authorization server.
(H) Token Introspection Response	The authorization server verifies the validity of the token and responds with parameters [12], such as access scope, validity, and adds the resource server to the resource observation list for that token. Then, the resource server analyzes the parameters received from the authorization server and chooses whether to accept or reject the client's resource request.
(I) Service provision	The resource server sends a response message to the client with the appropriate response code.

클라이언트, User-agent와 권한 서버, 권한 서버와 클라이언트의 통신은 기존의 OAuth 2.0과 동일하게 CoAP이 아닌 HTTPS를 통해 이루어질 수 있다. 한편, 제안 프레임워크에서는 기존과 유사하게 토큰 전송 방식으로 “bearer”토큰을 사용한다[17].

또한 제안 프레임워크는 User-agent를 통해 접근 토큰이 노출되지 않게 하기 위해 기존 OAuth 2.0의 4가지 권한 승인 방식 중 권한 코드 승인(Authorization Code Grant)을 기반으로 한다. 따라서 제안 프레임워크의 클라이언트는 권한 코드를 사용하여 권한 서버에게 직접 접근 토큰을 요청하며, 재발급 토큰(Refresh Token)을 통해 장기적 접근이 가능하다는 장점이 있다[2].

Fig. 4는 자원 서버의 토큰 점검 메시지 캐시 후 제안 프레임워크 동작 과정을 나타내며 Table 2는 Fig. 4에 소개된 (A)~(C) 과정의 세부내용을 나타낸다.

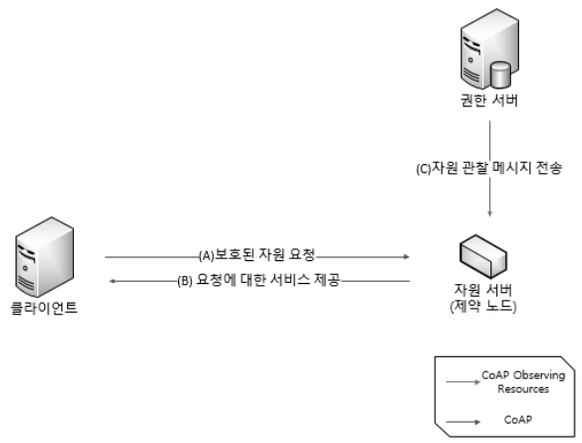


Fig. 4. Token Cache Consistency Management Through Resource Observation After the Resource Server's Token Cache

Table 2. Explanation of the Proposed Framework Operation Flow After Token Cache of Resource Server

step	detail
(A) Protected Resources Request	The client requests authentication and resources from the resource server as an access token.
(B) Service provision	The resource server checks the validity of the token through the cached token information and sends a response message to the client with the appropriate response code.
(C) Resource Observation message Transfer	When information on the token is updated, a notification message is transmitted to the resource server registered in the resource observation function in the token.

4. 제안 프레임워크 구현

4.1 개발 도구

본 논문에서의 제안 프레임워크는 사용자의 편의를 위하여 별도의 프로그램을 설치하지 않고도 사용할 수 있도록 웹 기반으로 개발하였다. 기본적인 개발 도구 및 구현 환경은 Table 3과 같다.

Table 3. Development Tools and Implementation Environment

Implementation environment	OS	Ubuntu 14.04.4 LTS
	IDE	PyCharm 2016.2.3
Development Tools	Programming Language	Python 2.7.6
	Open Library	CoAPthon, python-oauth2
	Database	MongoDB 2.6.12

구현 환경으로 운영체제는 Ubuntu 14.04.4 LTS, 통합 개발 환경(Integrated Development Environment, IDE)은 PyCharm 2016.2.3.을 사용하였다. 또한 개발도구로 프로그래

밍 언어는 python 2.7.6을 사용하였으며, 제안 프레임워크의 CoAP과 OAuth 2.0 기능은 기존에 오픈 라이브러리로 공개되어있는 CoAPthon, python-oauth2 [19]를 기반으로 하여 개발하였으며, 권한 서버의 토큰 관리 및 자원 소유자의 정보(회원 정보)를 관리하기위해 MongoDB 2.6.12를 사용하였다.

4.2 CoAP 추가옵션

앞서 언급한 대로, CoAP을 통해 클라이언트가 보호된 자원에 접근하기 위해, 혹은 기존 OAuth 2.0의 토큰 점검을 CoAP에 적용하기 위해서는 CoAP의 기존 옵션과는 별도로 Table 4, 5와 같이 새로운 옵션을 추가해야만 한다. 토큰 점검 요청 메시지와 오류 메시지를 위한 추가 옵션은 [20]에서 이미 정의된 옵션을 활용한다.

Table 4. Additional Options for CoAP Token Introspection Request Messages and Error Messages

Name	Format	Length	Default
Bearer	opaque	var	(none)
Token-type-hint	string	1-255	(none)
Error	uint	0-2	(none)

Table 5. Additional Options for CoAP Token Introspection Response Messages

Name	Format	Length	Default
Active	string	1-255	(none)
Scope	string	1-255	(none)
Token-Type	string	1-255	(none)
Exp	uint	0-4	(none)

Table 4는 토큰 점검 요청 메시지와 오류 메시지를 위해 CoAP에 추가해야 하는 옵션으로, 클라이언트는 “Bearer”옵션을 통해 토큰을 전송하고 “Token-type-hint”옵션을 통해 해당 토큰의 타입을 전송한다. “Error”옵션으로 사용되는 오류 코드는 [20]을 참고하여 invalid_request (0), invalid_token (1), insufficient_scope (2)와 같이 총 3가지로 구성된다. invalid_request (0)은 요청 메시지가 토큰 점검에 꼭 필요한 파라미터를 포함하고 있지 않거나 지원하고 있지 않는 파라미터를 포함 발생한다. 이 경우 자원 서버는 응답 코드로 4.00 (Bad Request)을 전송한다. invalid_token (1)은 만료, 훼손, 폐지 등 여러 이유로 접근 토큰이 유효하지 않을 경우 발생한다. 이 경우 자원 서버는 응답 코드로 4.01 (Unauthorized)을 전송한다. insufficient_scope (2)은 요청 메시지가 보호된 자원에 대해 접근 토큰에 의해 제공되는 권한 보다 더 많은 권한을 요구할 때 발생한다. 이 경우 자원 서버는 응답 코드로 4.03 (Forbidden)을 전송한다.

Table 5는 토큰 점검 응답 메시지를 위해 추가해야 하는 옵션이다. 기존 OAuth 2.0의 토큰 점검 응답 메시지에 대해 정의된 파라미터[13]와는 달리 제안 프레임워크에서는 제약노드인 자원 서버를 고려해 필수적으로 필요한 옵션만 사용한다. “Active”옵션은 접근 토큰의 상태를 알 수 있는 값이다. “Scope”옵션은 클라이언트가 보호된 자원에 접근 가능한 범위를 나타낸다. “Token-type”옵션은 접근 토큰의 타입을 나타내는 옵션으로 토큰을 검색할 때 사용한다. “Exp”옵션은 접근 토큰이 만료되는 시간을 나타내며 만료시간이 지난 토큰은 거절해야 한다.

본 논문에서는 해당 추가 옵션을 CoAPthon에 구현하기 위해서 CoAPthon의 옵션정의를 담당하는 모듈인 OptionRegistry 모듈에 Fig. 5와 같이 추가하였다.

```
TOKEN_TYPE = OptionItem(72, "Token_type", STRING, False, None)
SCOPE = OptionItem(76, "Scope", STRING, False, None)
BEARER = OptionItem(71, "Bearer", OPAQUE, False, None)
TOKEN_TYPE_HINT = OptionItem(73, "Token_type_hint", STRING, False, None)
ERROR = OptionItem(74, "Error", INTEGER, False, None)
ACTIVE = OptionItem(75, "Active", STRING, False, None)
EXP = OptionItem(77, "Exp", INTEGER, False, None)
```

Fig. 5. CoAPthon Additional Option Definition

또한 추가 옵션마다 관련 함수를 구현할 필요가 있다. 기본적으로 해당 옵션의 값을 호출하기위한 호출 함수, 해당 옵션의 값을 설정하기위한 설정 함수, 해당 옵션을 삭제하기 위한 삭제 함수가 필요하다. Figs. 6, 7, 8은 추가 옵션 중 error 옵션 관련 함수들이며 각각 호출, 설정, 삭제 함수이다.

```
@property
def error(self):
    for option in self.options:
        if option.number == defines.OptionRegistry.ERROR.number:
            return option.value
    return None
```

Fig. 6. Call Function for Error Option

```
@error.setter
def error(self, value):
    option = Option()
    option.number = defines.OptionRegistry.ERROR.number
    option.value = int(value)
    self.add_option(option)
```

Fig. 7. Setting Function for Error Option

```
@error.deleter
def error(self):
    self.del_option_by_number(defines.OptionRegistry.ERROR.number)
```

Fig. 8. Delete Function for Error Option

4.3 토큰 점검 메시지 요청 및 응답

제안 프레임워크는 CoAPthon과 python-oauth2를 기반으로 구현이 되어있다. 그러나 위의 오픈 라이브러리들에는 토큰 점검에 관련된 내용이 구현이 되어있지 않다. 따라서 본 논문의 제안 프레임워크의 구현을 위해서는 토큰 점검에 관련된 내용을 구현해야만 한다.

본 논문에서는 토큰 점검을 구현하기위해 Fig. 9와 같이 토큰 점검 메시지 요청에 해당하는 introspect 함수를 구현하였다. introspect 함수는 GET요청을 기반으로 하며 제안 프레임워크 설계에서 언급했던 추가옵션인 Bearer 옵션과 Token-type-hint 옵션을 포함하고 있으며 Token-type-hint 옵션에는 권한 코드 승인 방식을 나타내는 “code”라는 값을 입력한다. 또한 uri_path 옵션에는 권한 서버의 토큰 종점의 주소값을 입력한다.

```
def introspect(self, path, bearer, callback=None, access_path=None):
request = Request()
request.destination = self.server
request.code = defines.Codes.GET.number
request.uri_path = path
request.bearer = bearer
request.token_type_hint = "code"
request.scope = access_path
if callback is not None:
    thread = threading.Thread(target=self._thread_body, args=(request,
callback))
        thread.start()
else:
        self.protocol.send_message(request)
        response = self.queue.get(block=True)
        return response
```

Fig. 9. Token Introspection Message Request Function

한편, 토큰 점검 메시지 응답 또한 구현을 해야만 한다. 그러기 위해서 CoAPthon의 ResourceLayer라는 모듈에 해당 내용을 구현해야만 한다. ResourceLayer는 CoAP 서버가 가지고 있는 자원을 관리하는 모듈로써 대표적으로 자원을 수정하는 edit_resource 함수, 새로운 자원을 추가하는 add_resource 함수, 자원을 삭제하는 delete_resource 함수, 자원의 값을 전송하는 get_resource 함수 등이 있다.

본 논문에서는 제안 프레임워크의 토큰 점검 메시지 응답 구현을 위해, get_resource 함수에 해당 구문을 구현하였고, Fig. 10은 그 중 핵심 구문을 나타낸다.

권한 서버의 점검 종점은 introspection이라는 멤버 변수를 가지고 있으며 해당 변수는 데이터베이스(MongoDB)에서 가져온 토큰 정보를 저장하고 있다. Fig. 9에서 확인할 수 있듯이 해당 구문에서는 점검 종점의 introspection 변수를 기반으로 오류 메시지 처리와 해당 토큰의 유효성을 판단하고 있다.

```
if str(transaction.request.bearer) in resource.introspection:
    if resource.introspection[str(transaction.request.bearer)] is 1:
        error_option = Option()
        error_option.number = defines.OptionRegistry.ERROR.number
        error_option.value = 1
        transaction.response.add_option(error_option)
    elif resource.introspection[str(transaction.request.bearer)] is 2:
        error_option = Option()
        error_option.number = defines.OptionRegistry.ERROR.number
        error_option.value = 2
        transaction.response.add_option(error_option)
    else:
        active_option = Option()
        active_option.number = defines.OptionRegistry.ACTIVE.number
        if time.time() > resource.introspection[str(transaction.request.bearer)]["e
xpires_at"]:
                active_option.value = str(False)
        else:
            active_option.value = str(True)
            transaction.response.add_option(active_option)
else:
    active_option = Option()
    active_option.number = defines.OptionRegistry.ACTIVE.number
    active_option.value = str(False)
```

Fig. 10. Core Implementation Code for Token Introspection Message Response

4.4 토큰에 대한 자원 관찰 등록 기능

본 논문의 제안 프레임워크에서는 자원 서버가 권한 서버에게 토큰 점검 메시지 요청 시 자원 관찰 기능 등록을 함께 보내야 한다. 따라서 토큰 점검 요청에 자원 관찰 기능 등록을 포함하고 있는 새로운 함수 구현이 필요하다. 본 논문에서는 이러한 기능을 가진 introspect_observer를 Fig. 11에서와 같이 구현 하였다. 자원 관찰 기능 등록을 포함한 토큰 점검 요청을 구현하기 위해 앞서 언급했던 introspect 함수에 “request.observe = 0”이라는 구문을 추가하였고, 이 구문은 요청 메시지의 옵션에 자원 관찰 기능을 추가하는 것을 의미한다.

```
def introspect_observer(self, path, bearer, callback=None, access_path=None):
        request = Request()
        request.destination = self.server
        request.code = defines.Codes.GET.number
        request.uri_path = path
        request.bearer = bearer
        request.observe = 0
        request.token_type_hint = "code"
        request.scope = access_path
        if callback is not None:
            thread = threading.Thread(target=self._thread_body, args=(request, ca
llback))
            thread.start()
        else:
            self.protocol.send_message(request)
            response = self.queue.get(block=True)
            return response
```

Fig. 11. Token Introspection Message Request Function Including Registration of Resource Observation Function

4.5 토큰에 대한 자원 관찰 통지 기능

토큰에 대한 자원 관찰 통지 기능을 위해서는 토큰의 상태 값 변화를 감지하고 해당 토큰에 자원 관찰 기능을 등록한 자원 서버에게만 통지 메시지를 전송하는 기능을 구현해야만 한다. 이를 위해서 본 논문에서는 token_manager라는 모듈을 구현하였으며 token_manager 모듈의 구성요소로는 address_registry 함수, store_token_information 함수, compare_token 함수가 있다.

address_registry 함수는 권한 서버의 점검 종점이 자원 서버의 토큰 점검 요청 메시지를 수신하였을 경우, 점검 종점의 멤버 변수인 token_address에 해당 자원 서버의 주소를 저장하는 기능을 수행하는 함수이고 token_address 변수는 통지 메시지를 전송할 때 사용된다. Fig. 12는 address_registry 함수를 보여주고 있다.

```
def address_registry(Resource, request):
    try:
        for option in request.options:
            if option.number == defines.OptionRegistry.BEARER.number:
                if str(option.value) not in Resource.expired_token_list:
                    Resource.token_address[str(option.value)] = request.source
    except AttributeError:
        return
```

Fig. 12. Functions for Storing Resource Server Addresses

store_token_information 함수는 자원 서버의 토큰 점검 요청 메시지에 포함된 접근 토큰이 권한 서버가 발급한 토큰인지 확인하고, 해당 토큰이 권한 서버가 발급한 토큰일 경우 점검 종점의 introspection 변수에 해당 토큰의 정보를 저장하는 역할을 맡고 있다. 이런 역할을 수행하기 위해 해당 함수는 권한 서버가 발급한 접근 토큰 정보가 저장되어 있는 권한 서버의 데이터베이스에서 토큰 점검 요청 메시지에 포함된 접근 토큰 검색하고 유효성을 판단한다. 또한 store_token_information 함수는 자원 서버의 토큰 점검 요청 메시지에 포함된 접근 토큰이 유효한 접근 범위에 접근하고 있는지도 판단하고 있다. 해당 기능은 introspection 변수에 저장되어 있는 접근 범위를 기준으로 판단하며 접근 범위에서 벗어났을 경우 introspection 변수에 정수 2를 저장함으로써 추후에 오류 메시지를 반환할 때 판단 기준을 제공한다. 한편, Fig. 13은 store_token_information 함수를 보여주고 있다.

compare_token 함수는 데이터베이스에 저장되어 있는 접근 토큰 정보의 상태 변화를 감지하는 역할을 수행한다. compare_token 함수는 introspection 변수에 저장되어 있는 접근 토큰 정보와 데이터베이스에 저장되어 있는 접근 토큰 정보를 비교함으로써 접근 토큰 정보의 상태 변화를 감지한다. 한편, Fig. 14는 compare_token 함수를 보여주고 있다.

```
def store_token_information(Resource, request):
    db_host = 'localhost'
    db_port = 27017
    db = pymongo.MongoClient(db_host, db_port).test_database
    valid_scope = False
    token = None
    try:
        for option in request.options: #권한 서버가 발급한 토큰인지 판단
            if option.number == defines.OptionRegistry.BEARER.number:
                token = str(option.value)
                if db["access_tokens"].find_one({"token": str(option.value)}) is not None:
                    Resource.introspection[str(option.value)] = db["access_tokens"].find_one({"token": str(option.value)})

        for option in request.options: #접근 범위의 유효성 판단
            if option.number == defines.OptionRegistry.SCOPE.number:
                if token in Resource.introspection and token is not None:
                    if Resource.introspection[token] is not None:
                        for scope in Resource.introspection[token]["scopes"]:
                            if str(scope) == option.value:
                                valid_scope = True
                        if not valid_scope:
                            Resource.introspection[token] = 2
    except AttributeError:
        return
```

Fig. 13. Functions for Storing Token Information

```
def compare_token(Resource):
    db_host = 'localhost'
    db_port = 27017
    db = pymongo.MongoClient(db_host, db_port).test_database
    try:
        for token in Resource.token_address:
            if str(token) in Resource.introspection:
                if Resource.introspection[str(token)] != db["access_tokens"].find_one({"token": str(token)}):
                    Resource.observe_client_list.append(Resource.token_address[str(token)])
                    Resource.introspection[str(token)] = db["access_tokens"].find_one({"token": str(token)})
                elif Resource.introspection[str(token)]["expires_at"] < time.time():
                    if str(token) not in Resource.expired_token_list:
                        Resource.observe_client_list.append(Resource.token_address[str(token)])
                        Resource.expired_token_list.append(str(token))
                else:
                    print Resource.token_address[str(token)]
    except AttributeError:
        return
```

Fig. 14. Functions for Comparing Token Information

5. 제안 프레임워크 검증

5.1 검증 시스템 구성

본 논문의 제안 프레임워크의 구현을 검증하기 위해 Table 6과 같이 시스템을 구성하였다. 권한 서버와 클라이언트 서버는 Table 6의 사양을 갖춘 본 연구실의 서버에서 동작하고, 자원 서버는 제약 노드임을 가정해 Raspberry pi 2에서 동작한다. 또한 User-agent로는 Firefox 웹브라우저를 이용하였다.

Table 6. Verification System Specification

	Authorization Server	Client Server	Resource Server
CPU	Inter Xeon Processor E3-1280 v5	Inter Xeon Processor E3-1280 v5	900MHz BCM2836
RAM	32GB	32GB	1GB
Storage Volumes	4TB	4TB	32GB
OS version	Ubuntu 14.04.4 LTS	Ubuntu 14.04.4 LTS	Raspbian Jessie
IP address	218.150.181.116	218.150.181.110	218.150.181.117

5.2 제안 프레임워크 검증

본 논문의 제안 프레임워크의 검증을 위해 자원 서버의 저장되어 있는 온도 데이터와 습도 데이터를 이용해서 불쾌지수를 측정하는 서드파티 응용인 불쾌지수 측정 웹앱을 구성하여, 해당 웹앱이 자원 소유자의 권한 승인으로 인해 해당자원(온도, 습도)에 접근 가능한 시나리오를 구성하였다. 본 검증에서 불쾌지수 측정 웹앱은 사전에 권한 서버에 등록된 서드파티 응용으로, 서드파티 응용이 권한 서버에게 등록 시 발급되는 client_id와 client_secret 값을 이미 발급받은 상태로 가정한다.

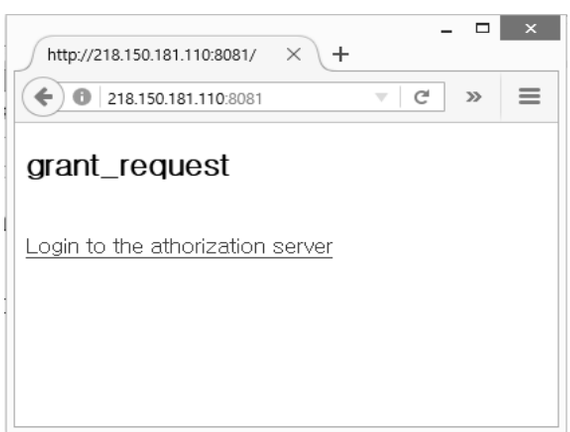


Fig. 15. Authorization Approval Request

Fig. 15는 불쾌지수 측정 웹앱이 자원 소유자에게 권한 승인을 요청하는 과정이다. 권한 승인 요청을 위해 불쾌지수 웹앱은 자원 소유자를 권한 서버의 자원 소유자 인증 페이지로 리다이렉션(redirection) 시킨다.

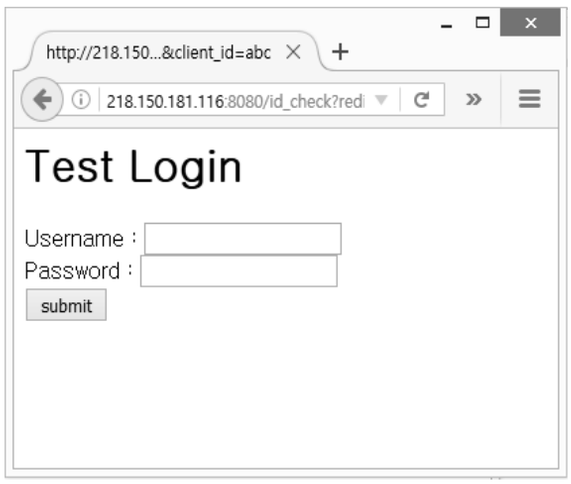


Fig. 16. Authorization Approval

Fig. 16은 불쾌지수 측정 웹앱에게 자원 소유자의 자원에 대한 접근 권한을 승인하기 위한 과정이다. 해당 그림에서 확인할 수 있듯이 불쾌지수 측정 웹앱(218.150.181.110)으로 인해 웹페이지가 권한 서버(218.150.181.116)에 리다이렉션 된 것을 알 수 있다.

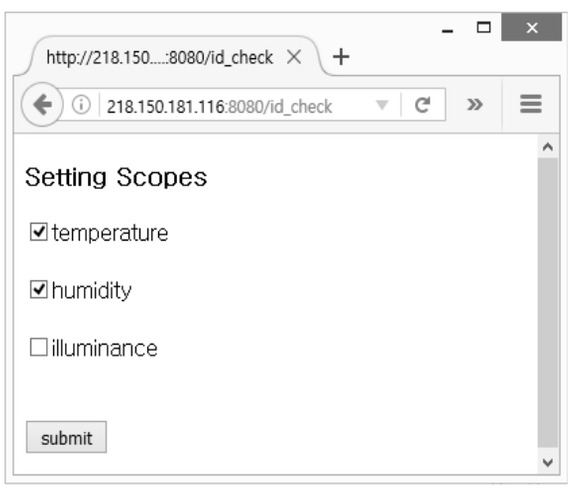


Fig. 17. Setting Resource Access Scope

Fig. 17은 불쾌지수 측정 웹앱의 자원 접근 범위를 설정하기 위한 페이지이다. 불쾌지수 측정을 위해서는 temperature 자원과 humidity 자원의 접근 허용이 필수적이다. 자원 소유자가 클라이언트에게 해당 자원의 접근을 허용하지 않을 시, 해당 웹앱은 불쾌지수 측정이 불가능하다.

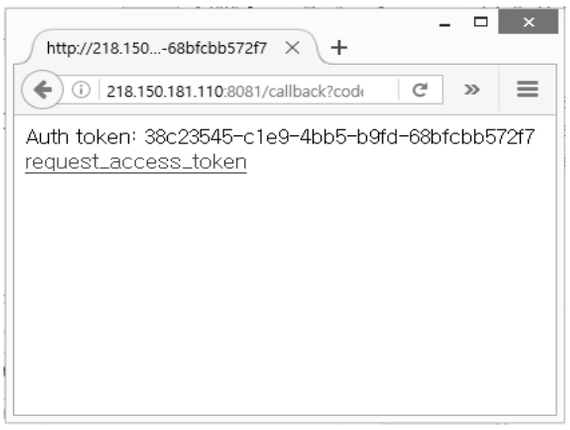


Fig. 18. Issue Authorization Code

Fig. 18은 권한 코드 발급이 정상적으로 이루어진 상황을 나타낸다. 권한 서버(218.150.181.116)는 웹 브라우저를 다시 불쾌지수 측정 웹앱(218.150.181.110)으로 리다이렉션 시키며, 이때 자원 소유자로부터 자원에 대한 접근 권한을 승인받았음을 나타내는 권한 코드를 전달한다.

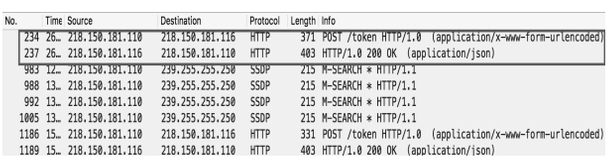


Fig. 19. Request and Issue an Access Token

Fig. 19는 접근 토큰 요청 및 발급 과정을 나타낸다. 클라이언트는 발급받은 권한 코드를 통해 권한 서버에게 접근 토큰을 요청할 수 있다. 그러나 이러한 접근 토큰 요청과 발급 과정은 User-agent(웹 브라우저)를 통해 이루어지지 않고 클라이언트와 권한 서버의 직접적인 통신으로 이루어지기 때문에 위의 과정은 패킷 분석 프로그램인 Wireshark를 통해 검증하였다. Fig. 20과 Fig. 21은 접근 토큰 요청 및 발급 메시지를 Wireshark를 통해 자세히 표현한 그림이다. Fig. 20에서 알 수

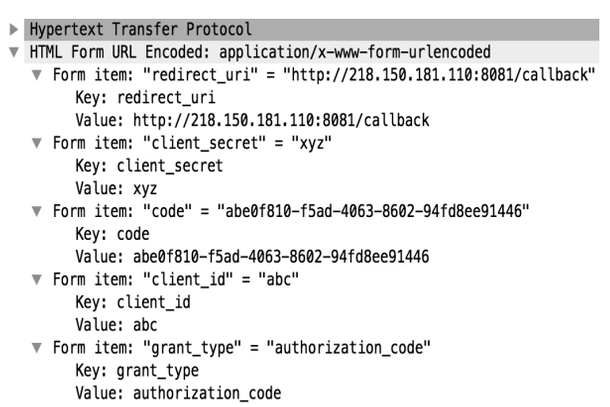


Fig. 20. Details of the Access Token Request Message

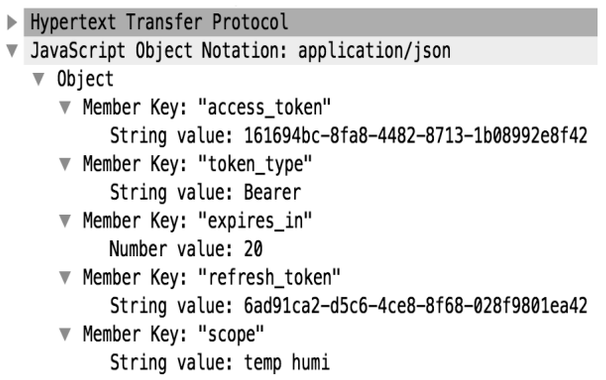


Fig. 21. Details of the Access Token Response Message

있듯이 접근 토큰 요청 메시지에는 redirect_uri, client_secret, code, client_id, grant_type을 포함하여 접근 토큰을 요청하는 것을 알 수 있다. 또한, Fig. 21에서는 접근 토큰 응답 메시지가 access_token, token_type, expires_in, refresh_token, scope를 포함하고 있는 것을 확인할 수 있다.

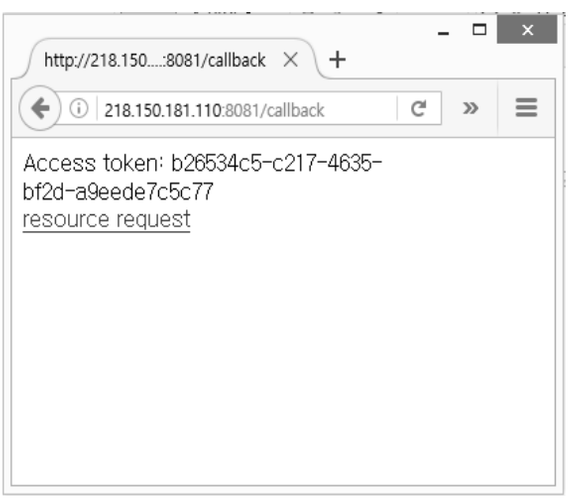


Fig. 22. Access Token Verification

Fig. 22에서는 클라이언트에게 접근 토큰이 제대로 발급되었는지 확인할 수 있다. 제안 프레임워크에서는 User-agent를 통해 접근 토큰이 공개되지 않으나, 본 검증의 정상적인 구동을 확인하기 위해 User-agent를 통해 접근 토큰을 보여줌으로써 발급 여부를 확인할 수 있게 하였다.

Fig. 23에서는 클라이언트에게 발급된 접근 토큰을 통해 불쾌지수 측정 웹앱의 서비스 제공이 정상적으로 동작하는 것을 검증한 그림이며, 클라이언트(218.150.181.110)가 접근 토큰을 통해 자원서버(218.150.181.117)의 자원에 접근 하는 과정은 User-agent를 통해서 확인할 수 없다. 따라서 해당 과정은 Fig. 24에서 확인할 수 있듯이 Wireshark를 통해 검증하였다.

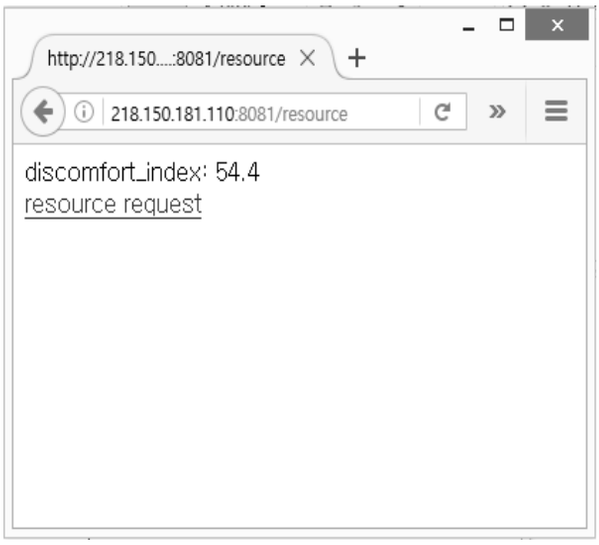


Fig. 23. Requesting Resources and Service Provision

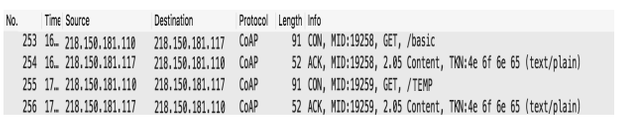


Fig. 24. Resource Access Through an Access Token

자원 요청 및 서비스 제공에 대한 각 메시지의 상세 내용은 Fig. 25, Fig. 26에서 확인할 수 있다. Fig. 25에서 확인할 수 있듯이 본 논문에서 새로 제안한 옵션들은 표준으로 정의된 옵션이 아닌 새로 정의된 옵션이기 때문에 Wireshark에서 제대로 분석하지 못한다. 그러나 Fig. 26을 통해 정상적으로 서비스가 제공되고 있는 것을 알 수 있다.

한편, 자원 서버(218.150.181.117)와 권한 서버(218.150.181.116)가 주고받는 토큰 점검 메시지 또한 Wireshark를 통해 검증하였고 Fig. 27은 해당 요청과 응답에 해당하는 CoAP 메시지이다.

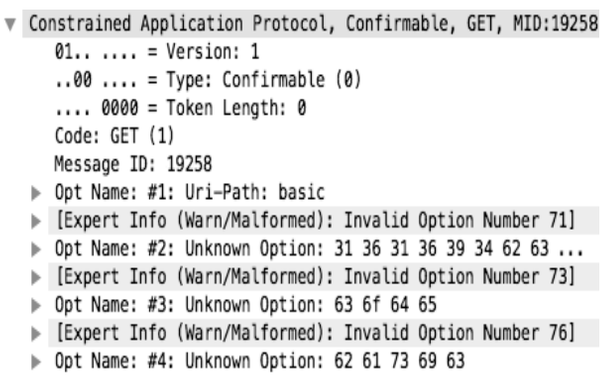


Fig. 25. Details of Resource Request Message

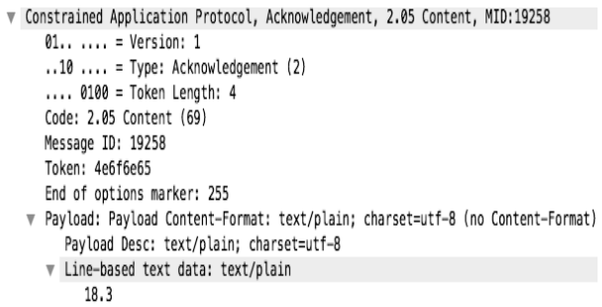


Fig. 26. Details of Service Provision Message

토큰 점검 메시지 요청 및 응답에 대한 각 메시지의 상세 내용은 Fig. 28, Fig. 29에서 확인할 수 있다. Fig. 28에서 확인할 수 있듯이 본 논문에서 새로 제안한 옵션들은 표준으로 정의된 옵션이 아닌 새로 정의된 옵션이기 때문에 Wireshark에서 제대로 분석하지 못한다. 그러나 토큰 정보, 토큰 타입 정보, 범위 옵션이 제대로 전송되는 것을 확인할 수 있으며, 또한 Fig. 29에서 토큰 점검 메시지 응답에 필요한 토큰 정보, 유효성 옵션이 제대로 전송되는 것을 확인할 수 있다.

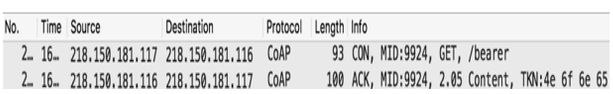


Fig. 27. Token introspection Request and Response Message Verification

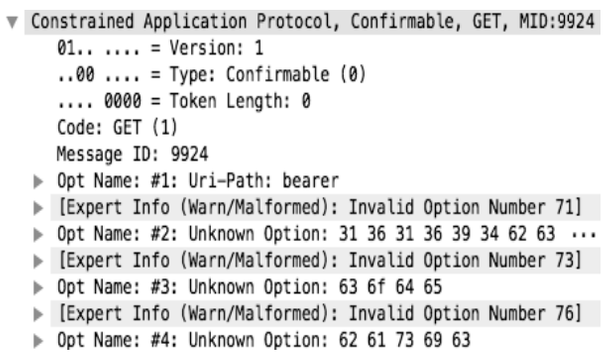


Fig. 28. Details of Token Introspection Request Message

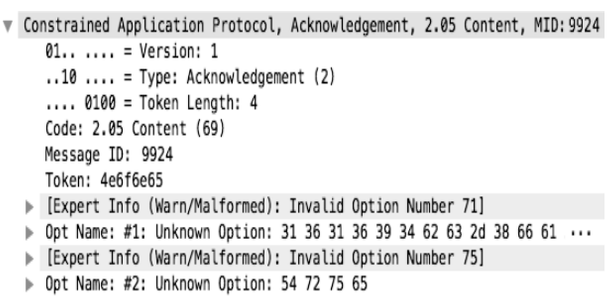


Fig. 29. Details of Token Introspection Response Message

6. 성능 분석

6.1 실험 내용

본 논문의 제안 프레임워크의 효율성을 파악하기 위해, 두 번의 실험을 진행하였다.

첫 번째 실험에서는 기존 HTTP를 기반으로 하는 OAuth 2.0과 CoAP을 기반으로 하는 OAuth 2.0의 효율성을 분석한다. 그러기 위해서 자원 서버의 통신 방법을 각각 자원서버의 통신 방법을 CoAP과 HTTP로 각각 설정하고, 자원 서버의 통신 방법에 따른 패킷 용량 누적 비교한다. 두 경우(HTTP, CoAP) 모두 토큰 점검 메시지를 캐시하지 않으며, 일정 시간 간격으로 클라이언트가 자원 서버에게 총 100번의 자원 접근 요청을 하고 매번 자원 서버는 토큰 점검 메시지 요청과 응답을 통해 토큰의 유효성을 판단한 후 적절한 응답을 한다.

두 번째 실험에서는 자원 서버가 토큰 점검 메시지를 캐시 한 경우와 하지 않은 경우를 비교 분석한다. 자원 서버가 토큰 점검 메시지를 캐시 한 경우에는 본 논문에서 제안하는 캐시 일관성 유지 기법인 CoAP의 자원 관찰 기능을 적용하였다. 그러나 토큰 점검 메시지를 캐시 하지 않은 경우에는 클라이언트의 자원 접근 요청 시 매번 토큰 점검 메시지를 권한 서버에게 전송한다. 해당 실험에서 일정 시간 간격으로 클라이언트가 자원 서버에게 10번, 20번, 30번의 자원 접근 요청을 하며, 토큰 점검 메시지를 캐시 하는 자원 서버는 캐시된 토큰 점검 메시지를 기반으로 토큰의 유효성을 판단한다. 그러나 토큰 점검 메시지를 캐시하지 않는 자원 서버는 토큰 점검 메시지 요청과 응답을 통해 토큰의 유효성을 판단하고 적절한 응답을 한다.

본 논문의 두 가지 실험은 본 논문의 구현 검증과 동일한 환경에서 진행하였다.

6.2 실험 결과

Fig. 30은 자원 서버의 통신 방법에 따른 자원 서버의 패킷 용량 누적 합계를 나타낸다. 본 실험을 통해 OAuth 2.0을 적용한 CoAP을 이용하면 제안 프레임워크의 자원 서버의 데이터 처리량이 현저히 줄어드는 것을 알 수 있다.

Fig. 31은 자원 서버의 캐시 관리 기법에 따른 자원 서버의 패킷 용량 누적 합계를 나타낸다. 본 실험에서는 접근 토큰의 만료시간을 20초로 설정하였다. 본 실험을 통해 본 논문에서 제안하는 자원 서버의 캐시 관리 기법은 자원 서버의 통신 횟수를 절반 가까이 줄임으로써 패킷 용량 누적 합계를 현저히 줄이는 것을 확인할 수 있다.

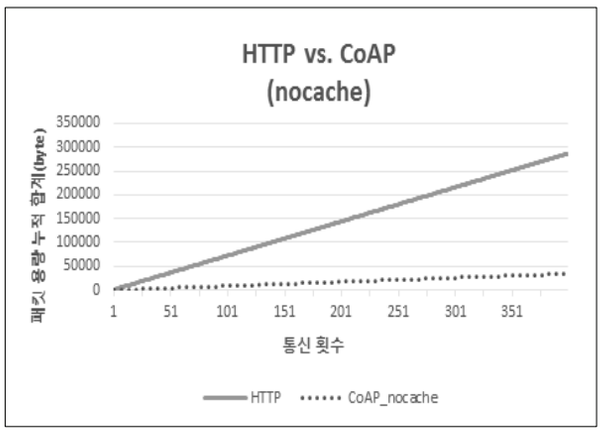


Fig. 30. Comparison of Total Packet Capacity According to Communication Method of Resource Server

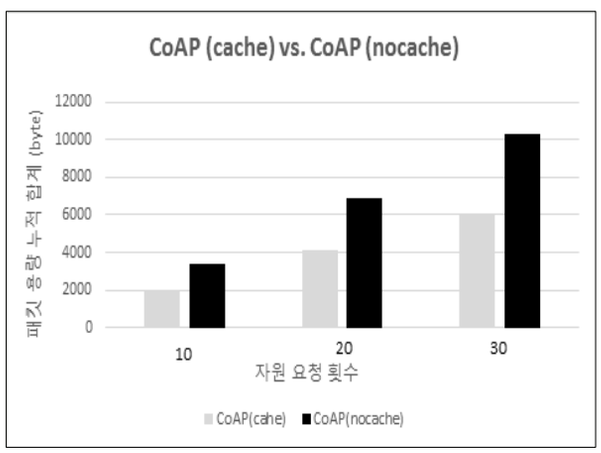


Fig. 31. Cache Efficiency Comparison of Resource Servers

7. 결 론

인터넷 프로토콜 표준을 제정해 왔던 IETF에서는 CORE WG을 중심으로 사물인터넷을 위한 표준인 CoAP 프로토콜을 제정하였으며, 최근에 결성된 ACE 워킹그룹은 사물인터넷 환경에서 기존 OAuth 2.0을 기반으로 응용 계층 기반의 새로운 ACE 보안 프레임워크 표준을 제정하고 있다.

그러나 ACE 프레임워크는 새로운 보안체계를 제안함으로써 기존 인터넷 환경에 적용하기에는 추가적인 작업이 필요하며, 이러한 추가적 비용은 IoT 환경에 OAuth 2.0의 적용을 저해하는 요인이 된다. 따라서 본 논문의 제안 프레임워크는 기존 인터넷 환경에서 사용하는 보안 프로토콜인 DTLS (Datagram Transport Layer Security)를 IoT 환경에 적용하기위한 연구가 활발히 진행되고 있기 때문에, 경량화된 DTLS를 기반으로 하여 기존 인터넷 환경에 쉽게 적용 가능한 범용성이 높은 새로운 OAuth 2.0 기반 CoAP 인증 프레임워크를 제안하였다.

특히, 제안 프레임워크에서 네트워크의 트래픽 양과 노드의 부하를 줄이기 위하여 자원 서버의 토큰 캐시 방안 및 자원 관찰 기능을 통한 캐시 일관성 유지 방안을 제안하였으며, 또한 제안 프레임워크를 Python기반 오픈소스인 CoAPthon과 python-oauth2를 기반으로 구현하였고 성능 분석을 통해 제안 프레임워크의 효율성을 입증하였다.

References

[1] Howon Kim, and Dong Kyue Kim, “IoT technology and security,” *Review of KIISC*, Vol.22, No.1, pp.7-31. 2012.

[2] D. Hart, The OAuth 2.0 Authorization Framework, IETF RFC 6749, Oct., 2012.

[3] IETF ACE WG [Internet], https://datatracker.ietf.org/wg/ace/

[4] SooHyun Ahn and Kwanghjo Kim, “A Method of lightweight DTLS protocol for IoT,” *KIISC CS-Conference Papers*, ISC-W'14, v.0. 2014.

[5] A. Capossele, V. Cervo, G. D. Cicco, and C. Petrioli, Security as a CoAP resource: An optimized DTLS implementation for the IoT, *IEEE International Conference on Communications (ICC)*, 2015.

[6] Z. Shelby, K. Hartke, and C. Bormann, The Constrained Application Protocol (CoAP), RFC 7252, Jun., 2014.

[7] SeokKap Ko, IETF CoAP Newest Standard Technology, *OSIA Standards & Technology Review*, Vol.28, No.4, pp.74-86, 2015.

[8] C. Bormann and Z. Shelby, “Block-Wise Transfers in the Constrained Application Protocol (CoAP),” IETF RFC 7252, Aug., 2016.

[9] E. Hammer-Lahav (Ed.), The OAuth 1.0 Protocol, IETF RFC 5849, Apr., 2010.

[10] L. Seitz, G. Selander, and E. Wahlstroem, Authentication and Authorization for Constrained Environments (ACE), draft-ietf-ace-oauth-authz-04, October 2016.

[11] C. Bormann, P. Hoffman, Concise Binary Object Representation (CBOR), IETF RFC7049, Oct., 2013.

[12] J. Schaad, CBOR Object Signing and Encryption (COSE), draft-ietf-cose-msg-23, Oct., 2016.

[13] J. Richer (Ed.), OAuth 2.0 Token Introspection, RFC 7662, Oct., 2015.

[14] Californium [Internet], https://github.com/eclipse/californium.git.

[15] CoAPthon [Internet], https://github.com/Tanganelli/CoAPthon.

[16] G. Tanganelli, C. Vallati, and E. Mingozzi, CoAPthon: Easy Development of CoAP-based IoT Applications with Python, IEEE 2nd World Forum on Internet of Things (WF-IoT), 2015.

[17] M. Jones, D. Hardt, The OAuth 2.0 Authorization Framework: Bearer Token Usage, RFC 6750, Oct., 2012.

[18] K. Hartke, Observing Resources in the Constrained Application Protocol (CoAP), IETF RFC 7641, Sep., 2015.

[19] python-oauth2 [Internet], https://github.com/wndhydrnt/python-oauth2.

[20] H. Tschofenig, The OAuth 2.0 Bearer Token Usage over the Constrained Application Protocol (CoAP), draft-tschofenig-ace-oauth-bt-00, Jul., 2014.

김 경 한

e-mail : goslim@koreatech.ac.kr

2015년 한국기술교육대학교 컴퓨터공학부 (학사)

2017년 한국기술교육대학교 컴퓨터공학부 (석사)

관심분야 : Internet of Things, OAuth 2.0

임 현 교

e-mail : glenn89@koreatech.ac.kr

2015년 한국기술교육대학교 컴퓨터공학부 (학사)

2015년～현 재 한국기술교육대학교 컴퓨터공학부 석사과정

관심분야 : Mobility Management, SDN, Machine Learning, Future Internet

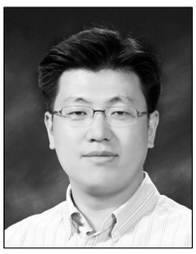
허 주 성

e-mail : chil1207@koreatech.ac.kr

2016년 한국기술교육대학교 컴퓨터공학부 (학사)

2016년～현 재 한국기술교육대학교 컴퓨터공학부 석사과정

관심분야 : Machine Learning, Social Network Analysis



한 연 희

e-mail : yhhan@koreatech.ac.kr
1998년 고려대학교 컴퓨터학과(석사)
2002년 고려대학교 컴퓨터학과(박사)
2002년 삼성종합기술원 전문연구원
2013년~2014년 미국 SUNY at Albany, Department of Computer Science 방문교수
2006년~현 재 한국기술교육대학교 컴퓨터공학부 정교수
관심분야 : Mobility Management, Internet of Things, Machine Learning, Social Networks, Future Internet