Journal of KIIT. Vol. 21, No. 12, pp. 179-187, Dec. 31, 2023. pISSN 1598-8619, eISSN 2093-7571
http://dx.doi.org/10.14801/jkiit.2023.21.12.179

블록체인 기반 ERC-721 Token을 사용한 Oauth 2.0 인증

최낙현*[1], 조동우*[2], 최선오**

Oauth 2.0 Authentication with Blockchain-based ERC-721 Tokens

Nakhyun Choi*[1], Dongwoo Joe*[2], and Sunoh Choi**

이 논문은 2023년도 교육부의 재원으로 한국연구재단 지자체-대학 협력기반 지역혁신 사업(2323RIS-008)과
한국연구재단 기초연구사업의 지원을 받아 수행된 연구임의 결과입니다(RS-2023-00237159)

요 약

Oauth 2.0은 인증을 위한 산업 표준 프로토콜이며, 다양한 이해 관계자들 간의 안전한 서비스 제공 및 상호 운용성을 용이하게 한다. 모든 Oauth 2.0 프로토콜의 흐름은 액세스 토큰의 생성으로 이어지며, 이 토큰은 사용자가 보호된 자원에 액세스를 요청할 때 사용된다. 그러나 Oauth 2.0 프로토콜은 액세스 토큰의 정의에 대해서는 투명하며, 어떤 특정한 토큰 형식을 지정하거나 토큰이 어떻게 생성되고 사용되는지는 명시하지 않는다. 대신, Oauth 2.0 사양은 이러한 모든 것들을 통합자의 설계 선택으로 남긴다. 따라서 본 논문에서는 분산 원장에 의해 지원되는 새로운 유형의 Oauth 2.0 인증 시스템을 제안한다. 본 논문에서 제안하는 시스템은 일반적인 Oauth 2.0 아키텍처를 고려하지만 기존 Oauth 2.0 프로토콜과 달리 블록체인을 사용하는 넓은 분야에서 확장이 용이하며 Oauth 2.0 프로토콜의 단점인 토큰의 위조나 임의 수정을 방지한다. 구체적으로 Ethereum 스마트 계약과 ERC-721 토큰 사양을 활용하여 Oauth 2.0 인증 시스템을 구성한다.

Abstract

Oauth 2.0 is an industry-standard protocol for authentication, facilitating secure service delivery and interoperability between various stakeholders. The flow of any Oauth 2.0 protocol leads to the generation of an access token, which is used when a user requests access to a protected resource. However, the Oauth 2.0 protocol is transparent about the definition of access tokens, and does not specify any particular token format or how tokens are generated and used. Instead, the Oauth 2.0 specification leaves all of these things as design choices for the integrator. In this paper, we propose a new type of Oauth 2.0 authentication system supported by a distributed ledger. Unlike the existing Oauth 2.0 protocol, the proposed system is scalable to a wide range of applications that use blockchains and prevents token forgery and arbitrary modification, which are shortcomings of the Oauth 2.0 protocol. Specifically, we utilize Ethereum smart contracts and the ERC-721 token specification to construct an Oauth 2.0 authentication system.

Keywords

Oauth 2.0, Ethereum, ERC-721 token, JWT

* 전북대학교 소프트웨어공학과 학사과정
- ORCID[1]: http://orcid.org/0009-0009-2885-4992,
- ORCID[2]: http://orcid.org/0009-0001-6739-3457
** 전북대학교 소프트웨어공학과 교수(교신저자)
- ORCID: http://orcid.org/0000-0002-0654-7109

· Received: Aug. 28, 2023, Revised: Nov. 20, 2023, Accepted: Nov. 23, 2023
· Corresponding Author: Sunoh Choi
Dept. of Software Engineering, Jeonbuk National University
Korea
Tel.: +82-63-270-4784, Email: suno7@jbnu.ac.kr

Ⅰ. 서 론

Oauth 2.0은 광범위한 채택을 받아 인증을 위한 산업 표준 프로토콜로 간주되고 있다[1]. Oauth 2.0은 위임과 상호 운용성을 가능하게 하며, 최종 사용자의 보안을 강화하고 액세스 제어 관리를 용이하게 한다. 이러한 흥미로운 특성으로 인하여, 초기에 고려한 것보다 더 높은 보안 요구사항이 있는 환경인 IoT 시스템, 오픈 뱅킹, eHealth, 전자 정부 및 전자 서명 등에서 사용된다[2]. 간단히 말하면, Oauth 2.0은 클라이언트가 인증 서버로부터 액세스 토큰을 획득하여 자원 서버에 저장된 보호된 자원에 액세스하는 프로토콜을 구성한다. 그러나 Oauth 2.0 사양은 액세스 토큰이 어떻게 생성되고 유효성을 검증하며 폐지되는지를 정의하지 않으며, 대신 Oauth 2.0 토큰의 수명 주기를 관리하도록 개방된 설계 선택으로 남긴다.

따라서 본 논문에서는 보유 증명 키로 보호되는 새로운 유형의 토큰을 제안한다. 이를 위해 분산원장 기술(DLT)에 기반을 둔다. 토큰의 구현은 Ethereum 블록체인과 ERC-721 토큰 사양을 기반으로 하며, 블록체인 기반의 토큰 관리 서비스를 구축하기 위해 Ethereum의 분산 앱 지원인 스마트 계약을 활용한다. 제안하는 토큰은 다음과 같은 장점을 가지고 있다.

첫째, 클라이언트는 토큰을 로컬에 저장할 필요가 없으며, 액세스 토큰과 관련된 비밀을 저장할 필요도 없다. 대신, 모든 토큰은 원장에서 검색될 수 있으므로 이동성이 좋고 여러 클라이언트 장치에서 쉽게 사용할 수 있다. 또한, 인기 있는 토큰 사양을 사용하므로 다양한 "지갑"과 라이브러리를 지원할 수 있다. 둘째, 토큰의 무결성과 신뢰성은 단순히 원장에서 조회함으로써 검증될 수 있다. 또한, 토큰 소유권은 인증 서버와의 상호작용 없이 안전하게 수정될 수 있다. 셋째, 모든 토큰은 원장에 불변하게 저장된다.

본 논문의 구성은 다음과 같다. 2장은 관련연구로 인증을 위한 산업 표준 프로토콜인 Oauth 2.0과 ERC-721 토큰, 관련된 기존 연구를 소개하며, 3장에서는 본 논문에서 제안하는 블록체인 기반 ERC-721 토큰을 사용한 Oauth 2.0 인증 시스템을 자세하게 기술한다. 4장에서는 결론 및 향후 연구에 관하여 기술한다.

Ⅱ. 관련 연구

2.1 Oauth 2.0

Oauth 2.0 시스템은 다음과 같은 엔티티로 구성된다. 자원 서버는 자원 소유자가 소유한 보호된 자원을 호스팅하는 역할을 한다. 클라이언트는 자원에 접근하려는 역할을 수행하며, 인증 서버는 액세스 토큰을 생성하는 역할을 담당한다. 액세스 토큰은 자원 소유자가 승인한 클라이언트에게 부여되며 클라이언트의 인증은 권한 요청을 사용하여 증명한다.

이러한 상호작용은 그림 1에 나타내어져 있다. 그림 1에서 볼 수 있듯이, 클라이언트는 먼저 자원 소유자에게 권한 부여를 요청하고, 받은 권한을 사용하여 인증 서버로부터 액세스 토큰을 얻는다. 마지막으로 자원 서버에 저장된 보호된 자원에 액세스 토큰을 사용하여 접근 후 자원을 획득한다. Oauth 2.0 프로토콜은 액세스 토큰과 권한 부여의 생성 및 유효성 검증 메커니즘을 투명하게 한다. 각 Oauth 2.0 배포는 사용할 토큰의 유형을 선택할 수 있다.

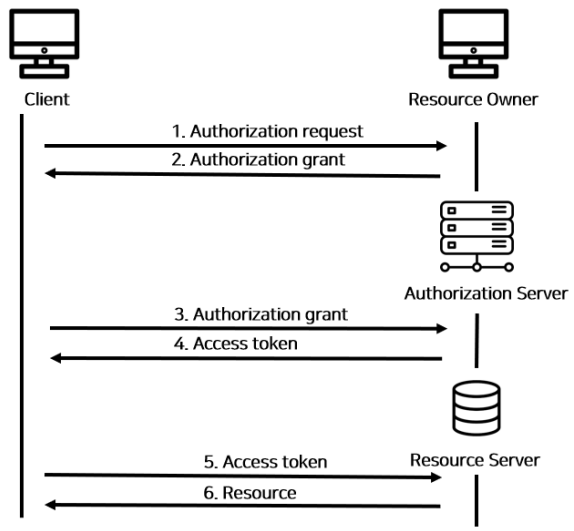

그림 1. Oauth 2.0 상호작용
Fig. 1. Oauth 2.0 interactions

가장 일반적으로 사용되는 토큰 유형은 무기명(Bearer) 토큰[3]으로 이를 소유한 모든 사용자가 사용할 수 있다. 추가적인 보안을 위해 토큰은 비밀키와 연결될 수 있으며, 비밀 키를 소유하고 있음을 증명할 수 있는 사용자만이 토큰을 사용할 수 있다. 후자 유형의 토큰은 더 높은 보안을 제공하며(소유권 확인을 위해 필요한 통신 오버헤드 비용이 있음) 본 논문이 제안하는 시스템에서 사용한다. 특히, 시스템의 구성은 블록체인 기반의 소유 증명 메커니즘을 갖춘 JSON Web Token(JWT)[4]를 기반으로 한 블록체인 기반 소유 증명 메커니즘을 사용한다. JWT는 "클레임(Claim)"을 나타내는 URL에 안전하고 간결한 표현 형식이다. 이는 0개 이상의 이름/값 쌍으로 구성되며, base64url로 인코딩되어 전송된다[5]. 표 1은 시스템에서 사용하는 JWT 클레임의 이름과 해당 의미를 포함한다.

표 1. JWT 클레임
Table 1. JWT claims

name	represent
iss	Token Issuer
aud	Token recipients
sub	Token owner
exp	Token expiration time
jti	Unique token identifier

2.2 Ethereum and ERC-721

Ethereum[6]은 분산 응용 프로그램인 "스마트 계약"을 지원하는 인기 있는 블록체인 시스템이다. 스마트 계약은 Ethereum 네트워크의 모든 "peer"에 의해 실행되며 합의에 따라 결과가 결정된다. 사용자는 사용자 각각의 키로 서명된 "트랜잭션"을 통해 스마트 계약과 상호 작용한다. 해당 키의 해시는 사용자 "주소"로 사용되며 블록체인에 저장된 정보와 사용자를 연결하는 데 사용한다. 또한 모든 트랜잭션은 불변하게 블록체인에 기록된다. 더 나아가 스마트 계약은 "이벤트"를 생성할 수 있다. 이벤트도 블록체인에 기록되며 블록체인과 최종 사용자 응용 프로그램은 특정 계약 이벤트를 "청취"하도록 구성할 수 있다.

Ethereum 커뮤니티는 스마트 계약을 위한 "Ethereum Request for Comments(ERC)"를 개발 중이다. ERC-721은 Ethereum 블록체인에서 "대체 불가능하거나 고유한 토큰"을 구축하는 방법을 설명하는 공개 표준[7]이다. 이 표준은 많은 면에서 가장 인기 있는 Ethereum 표준인 ERC-20 토큰과 매우 유사하며 사용자 정의 Ethereum 토큰을 생성하는 데 사용된다. 그러나 ERC-20 토큰과 달리 ERC-721 토큰은 "고유"한 토큰이며 다른 토큰과 교환할 수 없다(대체 불가능). Metamask3[8]와 같은 많은 Ethereum 지갑은 이러한 토큰을 처리할 수 있으며 모든 ERC-721 기반 토큰은 고유한 식별자(이를 "토큰 ID"라고 지칭한다)로 식별되며 오로지 한 명의 사용자만 소유할 수 있다.

ERC-721 기반 토큰은 Ethereum의 모든 다른 토큰과 마찬가지로 스마트 계약이 ERC-721 토큰을 생성하고 처리하기 위해 구현해야 할 일부 함수를 정의한다. 아래의 표 2는 우리 시스템에서 사용하는 ERC-721 토큰에 정의된 함수들을 설명한다.

표 2. ERC-721 토큰 함수
Table 2. ERC-721 Token Functions

name	Purpose
ownerOf(tokenId)	Return the token owner's address (for owner verification)
transferFrom(from, to, tokenId)	Transferring a token from the current owner (from) to a new owner (to)
getTransferredTokenIds()	Return a list of previously sent token IDs (used for authorization verification)
mintToken(to, metatdata)	Mint a new ERC-721 token and set up metadata
tokenURI(tokenId)	Set a metadata URI for a specific token
_setTokenURI(tokenId, _tokenURI)	Get the metadata URI of a specific token

2.3 관련된 기존 연구

Oauth 2.0 프로토콜에 블록체인을 결합한 기존 연구[8]에서는 ERC-721 토큰을 사용하여 새로운 유형의 분산 원장에 의해 뒷받침되는 새로운 유형의 Oauth 2.0 토큰을 제안한다.

또한 토큰 위임, 해지, 권한 부여와 자원 서버의 분리 등을 지원한다. 하지만 서명 검증이나 메타데이터의 암호화를 하지 않아 보안적인 취약점이 존재한다.

또한 DOauth 2.0: Oauth 2.0 기반 분산형 인가 프로토콜 연구[9]애서는 블록체인을 기술을 적용한 탈중앙화 인증 프로토콜인 DOauth 2.0을 제안한다. 기존 Oauth 2.0 프로토콜의 보안 요구 사항의 충족과 중앙화된 인증 서버에 의존하여 인증 서버가 손상되면 클라이언트 전체로 문제가 확산되는 문제를 해결한다. JWT와 ERC-721 Token을 사용한다는 점은 동일하지만 토큰의 정보가 노출되어 개인 정보 보호에 대한 보안적인 문제점이 존재한다.

III. 블록체인 기반 ERC-721 Token을 사용한 Oauth 2.0 인증

3.1 시스템 개요

본 논문에서 제안하는 시스템은 일반적인 Oauth 2.0 아키텍처를 고려한다. 따라서 시스템의 주요 구성 요소는 클라이언트, 인증 서버, 자원 서버이다. 시스템의 전체 작동은 다음과 같다. 또한, 클라이언트는 이미 자원 소유자로부터 권한 부여를 받았다고 가정한다.

첫째, 권한 부여를 받았다고 가정한 클라이언트의 ID와 Password를 사용하여 인증 서버로부터 액세스 토큰을 요청한다. 둘째, 인증 서버는 클라이언트의 권한 부여에 대해 유효성을 검사하고 액세스 토큰(JWT) 및 ERC-721 토큰을 생성한다. 다음으로 ERC-721 토큰을 이더리움 네트워크로 전송하고, 액세스 토큰(JWT)를 클라이언트에게 전달한다.

셋째, 클라이언트는 액세스 토큰(JWT)를 사용하여 자원 서버에 자원 접근을 요청한다. 자원 서버는 액세스 토큰(JWT)의 유효성과 소유권을 확인하기 위해 액세스 토큰에 포함된 정보를 사용하여 일치하는 ERC-721 토큰이 존재하는지 확인한다. 마지막으로, 모든 검증이 성공하면 자원 서버는 클라이언트에게 자원을 반환한다. 일련의 과정은 그림 2에 나타내어져 있다.

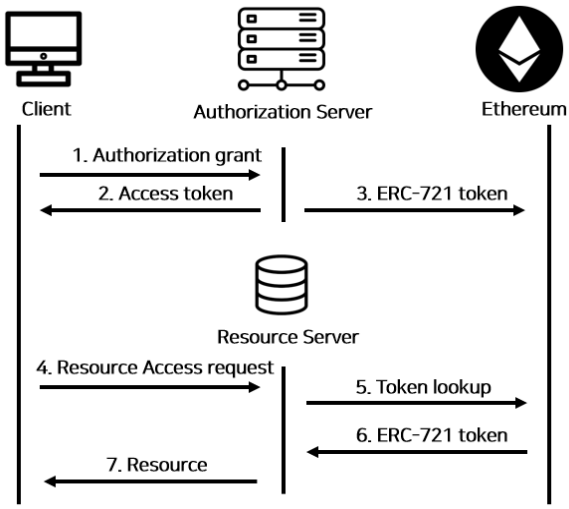

그림 2. 시스템 개요
Fig. 2. Overview of system

3.2 액세스 토큰 요청

액세스 토큰의 과정은 4가지 단계로 이루어진다. 액세스 토큰 요청 과정에 대한 의사코드는 그림 3에 나타내어져 있다. 첫째, 권한 부여를 받은 클라이언트는 인증 서버로부터 액세스 토큰을 요청한다. 인증 서버는 클라이언트의 소유권과 권한 부여의 유효성을 검증한다. 둘째, 검증이 성공하면 인증 서버는 표 1에 포함된 클레임을 포함하는 액세스 토큰(JWT)를 생성한다.

셋째, 인증 서버는 ERC-721 토큰을 생성한다. ERC-721 토큰의 tokenid는 JWT의 jti 클레임에 지정된 값과 일치하며 또한, 토큰의 메타데이터는 JWT의 base64url 인코딩과 동일하게 설정되며 암호화한다. 마지막으로, 인증 서버는 ERC-721 스마트 계약의 transferFrom 함수를 호출하여 생성된 토큰을 이더리움 네트워크로 전송하고, 생성된 JWT를 클라이언트에게 전송한다. 여기서 클라이언트는 JWT를 저장할 필요가 없다는 점에 유의해야 한다. 언제든지 ERC-721 컨트랙트에서 소유하는 모든 토큰을 검색하고 토큰의 메타데이터에서 해당 JWT를 추출할 수 있다. JWT와 ERC-721 Token에 대한 정보는 그림 4에 나타내어져 있다.

```
POST 요청 "/login"을 받음
1.요청 본문에서 "id"와 "pw" 값을 추출 후 사용자 인증을 수행하여 유효한 사용자인지 확인
2. 인증에 성공하면 해당 사용자의 ID를 반환받고, 실패 시 500 상태 코드를 응답으로 반환하고 종료
3. 성공적으로 인증한 사용자의 ID를 사용하여 액세스 토큰 생성 및 반환
    - 15분 후에 토큰이 만료되도록 설정
    - JWT 토큰을 사용하여 액세스 토큰을 생성
    - 이때 "iss" (발급자), "aud" (대상자), "sub" (주제), "exp" (만료 시간), 그리고 "jti" (토큰 ID)를 포함
4. 생성된 액세스 토큰을 사용하여 ERC-721 토큰발행
    - 액세스 토큰을 ERC-721 토큰의 ID로 사용
    - ERC-721 토큰의 현재 소유자 주소를 ownerId로 설정
    - JWT로 서명되고 암호화된 메타데이터를 생성하여 metadata 변수에 저장
5. ERC-721 토큰 발행에 성공하면 새로운 토큰 ID(newTokenId)를 반환받음
6. 액세스 토큰과 새로운 토큰 ID(newTokenId)를 클라이언트에 응답으로 반환
7. 오류 발생 시 500 상태 코드를 응답으로 반환
```

그림 3. 액세스 토큰 요청 의사코드
Fig. 3. Access token request pseudocode

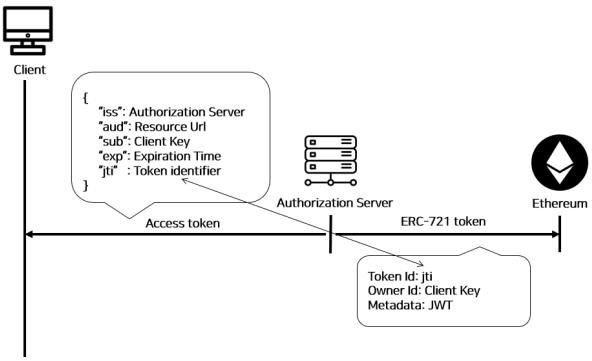

그림 4. 액세스 토큰 & ERC-721 token 정보
Fig. 4. Information of access tokens & ERC-721 tokens

3.3 자원 접근 요청

클라이언트가 보호된 자원에 액세스하기 위해 이전에 인증 서버로부터 받은 JWT를 포함하여 자원 서버에게 요청한다. 자원 서버는 다음 단계를 수행한다.

첫째, 액세스 토큰을 디코딩하여 JWT의 iss 클레임에 포함된 Authorization Server가 인증 서버와 일치하는지, 둘째, JWT의 aud 클레임에 포함된 Resource Url가 자원 서버의 URI와 일치하는지, 셋째, 토큰이 아직 유효한지(즉, 만료되지 않았는지) 확인한다. 넷째, JWT가 유효하면 자원 서버는 Contract의 getTransferredTokenIds()를 호출하여 이전에 전송한 ERC-721 토큰 목록을 가져온다. JWT의 jti에 포함된 tokenid와 일치하는 토큰이 있다면 Contract의 ownerOf()를 호출하여 해당 토큰이 클라이언트의 소유인지 확인한다. 클라이언트의 소유임이 확인되면 자원 서버는 클라이언트에게 자원을 반환한다. 일련의 과정은 그림 5에 나타내어져 있다. 또한 자원 접근 요청에 대한 의사코드는 그림 6에 나타내어져 있다.

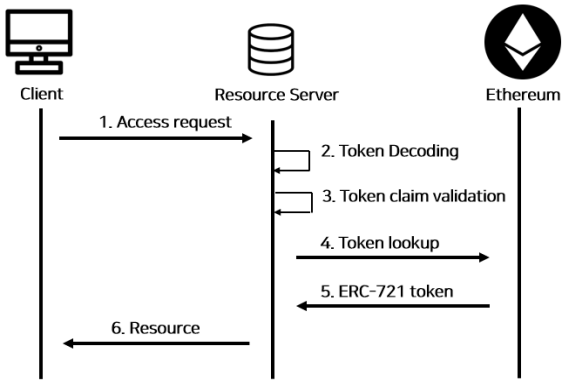

그림 5. 자원 접근 요청 과정
Fig. 5. Resource access request process

3.4 기존 연구와의 비교 분석

기존 연구[8]에서는 본 연구와 같이 ERC-721 토큰을 사용하여 새로운 유형의 분산 원장에 의해 뒷받침되는 새로운 유형의 Oauth 2.0 토큰을 제안한다. JWT와 ERC-721 Token을 사용한다는 점은 동일하지만 기존 연구에서는 서명 검증이나 메타데이터를 암호화하지 않는 보안적인 문제점이 존재한다. 서명 검증을 하지 않으면 위조 토큰을 생성하여 시스템에 접근할 수 있다는 문제가 있다. 또한 ERC-721 Token의 메타데이터는 불변하며 누구나 볼 수 있으므로 개인 정보에 위협이 될 수 있다. 본 연구는 기존 연구와 달리 서명 검증과 메타데이터의 암호화 및 복호화를 통하여 위조 토큰을 방지하고 개인 정보를 보호하며 데이터의 무결성을 보장한다. 구체적으로 서명 검증은 트랜잭션의 형식을 확인 후 Web3 라이브러리의 recoverTransaction 함수를 사용하여 서명된 주소를 복구한다.

"/resource" 엔드포인트로 GET 요청을 수신한다.
1. 요청에서 "Authorization" 헤더를 추출하며, 이 헤더는 액세스 토큰을 포함해야 한다.
2. "Authorization" 헤더가 누락되었거나 올바르게 형식이 지정되지 않았을 경우, 401 상태 코드 (권한 없음)로 응답하고 종료한다.
3. ACCESS_TOKEN_SECRET를 사용하여 액세스 토큰을 확인하고, 성공 시 디코딩하여 토큰을 해독한다.
4. 해독한 토큰의 "iss"가 인증 서버의 url과 일치 여부, "aud"가 자원 서버의 url과 일치 여부, 만료 여부를 확인한다.
5. 해독한 토큰에서 "jti" (토큰 ID)를 추출하여 "requestedTokenId"로 저장한다.
6. 계약(Contract)에서 "getTransferredTokenIds" 함수를 호출하여 전송된 토큰 ID 목록을 가져온다.
7. "requestedTokenId"가 전송된 토큰 ID 목록에 포함되어 있는지 확인한다.
- "requestedTokenId"가 목록에 포함되어 있다면, 다음 단계를 수행한다.
- 해독한 토큰에서 암호화된 메타데이터를 추출하고 "decryptMetadata" 함수를 사용하여 메타데이터를 복호화한다.
- 성공으로 응답하고 만약 "requestedTokenId"가 목록에 포함되어 있지 않다면, 실패로 응답한다.
8. 프로세스 중에 오류가 발생한 경우, 500 상태 코드 (내부 서버 오류)로 응답한다.

그림 6. 자원 접근 요청 의사코드
Fig. 6. Resource access request pseudocode

또한 Web3 라이브러리의 getTransaction 함수를 사용하여 트랜잭션의 발신자 주소를 얻는다. 서명된 주소와 발신자 주소를 비교하여 일치하면 서명이 유효하다고 판단한다. 암호화 & 복호화 알고리즘은 AES-256-CBC를 사용하여 메타데이터를 암호화하고 복호화한다.

표 3. 기존 연구와의 비교 분석
Table 3. Comparisons with existing research

	Existing Research	Ours
Signature Verification	X	O
Metadata Encryption & Decryption	X	O
JWT	O	O
ERC-721 Token	O	O

또한 DOauth 2.0을 제안한 연구[9]는 본 연구와 같이 Oauth 2.0의 중앙 집중화된 인증 서버의 문제를 보완하고 액세스 토큰을 도난당하더라도 공격자의 목적인 자원 획득에 실패하도록 한다. JWT와 ERC-721 Token을 사용한다는 점은 동일하지만 토큰의 정보가 노출되어 개인 정보 보호에 대한 보안적인 문제점이 존재한다.

Ⅳ. 시스템 구현 및 테스트

4.1 시스템 구성

위조 토큰 검증 테스트를 제외한 모든 테스트는 자바스크립트를 사용하여 개발한 웹 사이트를 통해 진행한다. 프론트엔드는 React, 백엔드는 Node.js로 개발하였다. 이더리움 네트워크(Ganache)와의 통신은 Web3 라이브러리를 사용한다. 또한, 테스트에서 인증 서버와 자원 서버는 동일하다고 가정한다. 시스템의 주요 구성 요소는 Smart Contract이다. Smart Contract는 Solidity8을 사용하여 개발하였으며 truffle[10]을 사용하여 배포한다. truffle은 이더리움 블록체인을 기반으로 하며 컴파일, 테스트, Smart Contract 배포를 통합하는 IDE이다. 테스트의 전반적인 과정은 아래 그림 7에 나타내어져 있다.

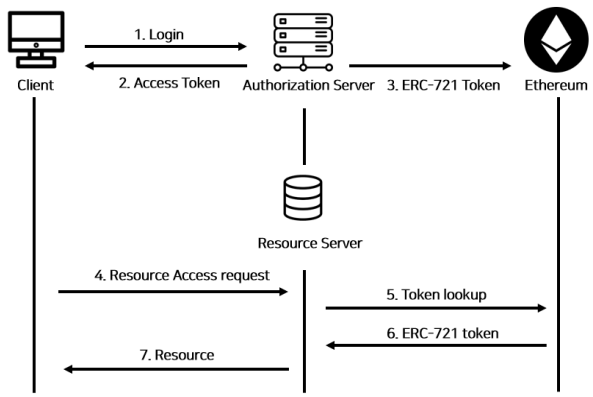

그림 7. 테스트 개요
Fig. 7. Overview of test

4.2 블록체인 구성 및 비용 평가

제안한 시스템을 Ganache[10]에서 테스트한다.

Ganache는 로컬에 배포된 블록체인 시뮬레이터로써 블록체인 네트워크를 시뮬레이션하고 테스트할 수 있는 그래픽 사용자 인터페이스이다. 또한 개인적인 이더리움 블록체인을 생성하여 테스트를 실행할 수 있으며, 명령을 실행하고 체인의 작동을 확인하여 상태를 검사할 수 있다. 시스템의 수행 과정에서 Smart Contract 기능을 호출하는 것은 약간의 계산 오버헤드를 발생시킨다. 가스는 이더리움의 연산 및 저장 자원을 측정하는 단위이다. 각 스마트 계약의 작업에는 고정된 양의 가스가 필요하며 가스 가격은 클라이언트가 가스 단위 당 지불하고자 하는 "Ether"의 양이다. 표 3은 블록체인 네트워크에 스마트 계약을 배포하는 비용과 가스 단위로 시스템에서 사용하는 비용을 나타낸다.

표 3. 블록체인 구성 비용
Table 3. Blockchain configuration costs

Operation	Cost measured in gas
Contraction deployment	2521387
create a token	61454
transfer a token	56286

4.3 액세스 토큰 발급

클라이언트는 자원 소유자에게 권한 부여를 받은 후 액세스 토큰 발급을 요청한다. 먼저 클라이언트의 ID와 Password를 사용하여 로그인(유효성 검사)을 진행한다. 로그인이 성공하면 메인화면으로 이동한다. 인증 서버는 액세스 토큰을 생성 후 ERC-721 토큰을 발행하여 각각 클라이언트와 이더리움 네트워크(Ganache)로 전송한다. 액세스 토큰에는 JWT Claims 정보가 포함된다. 또한, 액세스 토큰의 모든 정보는 ERC-721 토큰의 메타데이터에 포함된다. 테스트 결과는 아래 그림 8와 9에 나타내어져 있다.

그림 8. 로그인 화면
Fig. 8. Login screen

그림 9. 메인 화면(액세스 토큰)
Fig. 9. Main screen(access token)

4.4 자원 접근

액세스 토큰 즉, JWT를 발급받은 클라이언트는 자원 서버에게 JWT를 포함하여 자원 접근을 요청한다. 자원 서버는 3.3절에 기술된 4단계의 액세스 토큰 유효성 및 클라이언트 소유자 확인을 통해 검증을 수행한다. 검증이 성공하면 클라이언트에게 자원을 반환한다. 테스트에서는 Success라는 결과를 출력하는 것으로 대체한다. 자원 접근 요청의 결과는 아래 그림 10에 나타내어져 있다.

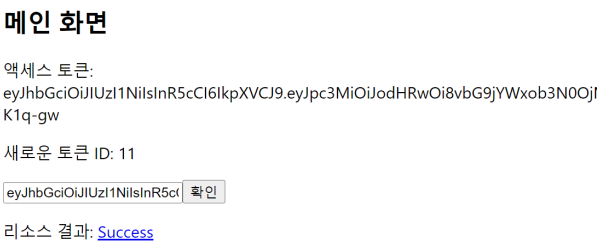

그림 10. 메인 화면(자원 접근)
Fig. 10. Main screen(resource access)

4.5 위조 토큰 검증

본 논문에서 제안하는 시스템은 기존 Oauth 2.0 프로토콜의 주요 취약점 중 하나인 토큰의 위조 및 임의 수정을 방지한다. 증명을 위해 위조 토큰 검증 테스트를 진행한다. 위조 토큰 검증 테스트는 Postman을 사용하여 진행하며 위조 토큰과 정상적인 액세스 토큰의 차이점은 JWT claims의 jti 부분에 있으며 위조 토큰은 jti부분을 null로 설정한다. 위조 토큰을 발급 후 자원을 요청하면 자원 획득에 실패한다. 3.3 절에 기술한 자원 서버의 검증 단계 4번째인 jti에 포함된 tokenid와 일치하는 ERC-721 Token이 존재하지 않기 때문이다. 테스트 결과는 아래 그림 11과 12에 나타내어져 있다.

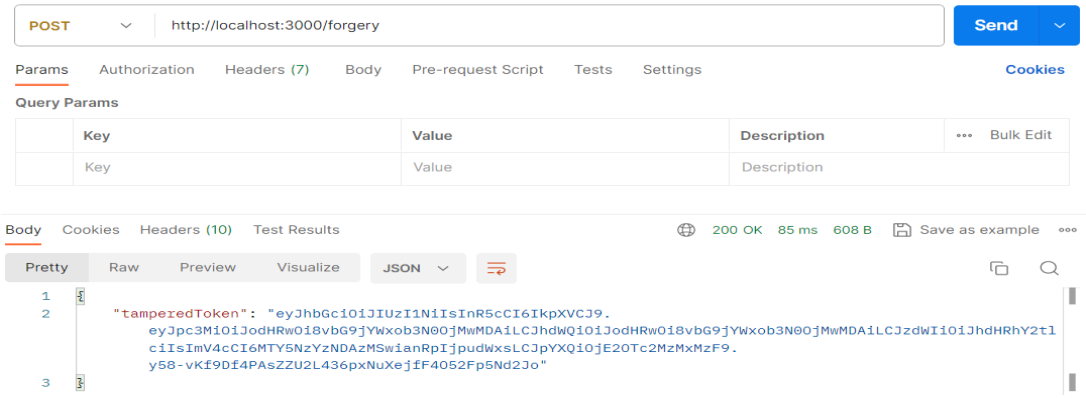

그림 11 위조 토큰 발급
Fig. 11. Issuing counterfeit tokens

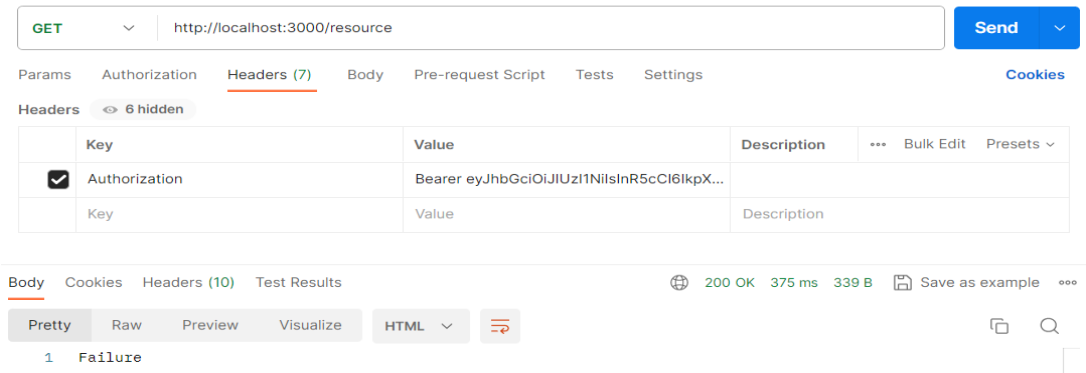

그림 12. 위조 토큰(자원 획득 실패)
Fig. 12. Counterfeit tokens (failed to acquire resources)

V. 결론 및 향후 과제

본 논문에서는 블록체인 기반 스마트 계약을 이용한 Oauth 2.0 인증 시스템과 새로운 형식의 토큰 설계와 구현을 제안한다. 해당 시스템은 Ethereum을 사용하여 기존에 존재하던 Oauth 2.0 인증 시스템과 달리 블록체인을 사용하는 넓은 분야에서 확장이 용이하며 기존 연구의 보안적인 취약점을 방지한다.

구체적으로 ERC-721 토큰을 활용하여 Oauth 2.0 프로토콜의 주요 취약점 중 하나인 토큰의 위조나 임의 수정을 방지한다. 향후 연구로는 토큰의 추적 등 토큰을 활용한 추가적인 연구를 진행할 계획이다.

References

[1] D. Hardt, "The Oauth 2.0 authorization framework", IETF, RFC6749, pp. 1-5, 2012.

[2] T. Lodderstedt, J. Bradley, A. Labunets, and d. Fett, "Oauth 2.0 security best current practice", IETF, draft-RFC, pp. 1-2, 2019.

[3] M. Jones and D. Hardt, "The Oauth 2.0 authorization framework:Bearer token usage", IETF, RFC 6750, pp. 2-3, 2012.

[4] M. Jones, J. Bradley, and N. Sakimura, "JSON web token (JWT)", RFC 7519, pp. 5-8, 2015.

[5] S. Josefsson, "The Base16, Base32, and Base64 data encodings", IETF, RFC 4648, pp. 5-7, 2006.

[6] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger", Ethereum Project Yellow Paper, Vol. 151, pp. 1-32, 2014.

[7] H. Kim, Y. Park, and H. Kim, "Introduction to Perpetual Contract Non-Fungible Token Technology", Journal of the Korean Communications Society (Information and Communication), Vol. 40, No. 4, pp. 60-65, Mar. 2023.

[8] N. Fotiou, L. Pittaras, V. A. Siris, S. Voulgaris, and G. C. Polyzos, "Oauth 2.0 autorization using blockchain-based tokens", NDSS Workshop on Decentralized IoT Systems and Security (DISS), Jan. 2020. https://doi.org/10.48550/arXiv.2001.10461.

[9] S. Hong, "DOauth 2.0: A Decentralized Authorization Protocol based on Oauth 2.0", D. Thesis, POSTECH, Graduate School of General Studies, pp. 13-18, Feb. 2021.

[10] C. N. Z. Latt, "A Data Provenance System for Myanmar Rice Cycle based on Ethereum Blockchain", Master's thesis, Pukyong National University, pp. 19-21, Feb. 2021.

저자소개

최 낙 현 (Nakhyun Choi)

2020년 3월 ~ 현재 : 전북대학교 소프트웨어공학과 학사과정
관심분야 : 블록체인/인공지능

조 동 우 (Dongwoo Joe)

2018년 3월~현재 : 전북대학교 소프트웨어공학과 학사과정
관심분야 : 인공지능/AI보안

최 선 오 (Sunoh Choi)

2005년 2월 : 고려대학교 컴퓨터학과(학사)
2008년 2월 : 고려대학교 컴퓨터학과(석사)
2014년 5월 : Purdue Univ ECE (박사)
2014년 ~ 2019년 : ETRI 정보보호연구본부 선임연구원
2019년 ~ 2021년 : 호남대학교 컴퓨터공학과 조교수
2021년 ~ 현재 : 전북대학교 소프트웨어공학과 부교수
관심분야 : 데이터보안, 네트워크보안, AI보안