https://doi.org/10.33778/kcsa.2024.24.4.003

JWT 토큰 서명 알고리즘 보안 성능에 대한 연구★

이 채 은*, 전 상 훈**

요 약

웹 환경이 급속히 확장되면서 사용자들은 다양한 웹 서비스를 이용하고 있다. 인터넷 이용자가 늘어나면서 사용자 ID와 비밀번호를 도용하는 등의 피싱 수법이 발생하고 있다. 안전한 환경을 제공하기 위해 보안 기술을 충분히 알고 적용해야 하며 이를 위해 안전한 인증 방안이 필요하다. 본 연구에서는 HMAC, RSA와 같은 토큰 서명 알고리즘 중 HS256과 RS256을 사용한 웹 페이지 보안을 비교 분석한다. HS256 알고리즘은 빠르고 효율적이지만, 대칭 키를 관리하는 어려움이 있다. 반면, RS256은 공개키와 개인키를 사용하여 더 높은 보안성을 제공하지만, 키의 길이가 길어질수록 키 생성 및 암호화에 소요되는 시간이 증가한다. 무차별 대입 공격에 대한 두 알고리즘의 보안 성능을 비교한 결과, RS256이 HS256에 비해 상대적으로 안전한 것으로 나타났다. 이 결과는 웹 보안을 위해 토큰 서명 알고리즘의 처리 성능과 보안 성능을 고려한 적절한 선택이 필요함을 보여준다. 이 연구가 웹 보안을 강화하고 사용자 인증 정보를 보호하기 위한 알고리즘 선택에 중요한 지침이 되기를 기대한다.

A Study on the Security Performance of JWT Token Signature Algorithms

Chaeeun Lee*, Sanghoon Jeon**

ABSTRACT

With the rapid expansion of the web environment, users are using various web services. As Internet users increase, phishing techniques such as stealing user IDs and passwords are occurring. In order to provide a safe environment, security technologies must be fully known and applied, and a secure authentication method is required for this. This study compares and analyzes web page security using token signing algorithms such as HMAC and RSA, specifically HS256 and RS256. The HS256 algorithm is fast and efficient but has difficulties in managing symmetric keys. In contrast, RS256 uses a public and private key, offering higher security, although longer key lengths increase the time required for key generation and encryption. Evaluation of the security performance of the two algorithms against brute force attacks show that RS256 is relatively safer than HS256. These results demonstrate the need for an appropriate selection of token signing algorithms, considering both processing performance and security performance, to ensure web security. We hope this study will be an important guideline for selecting algorithms to enhance web security and protect user authentication information.

Key words : JSON Web Token(JWT), Token Signature Algorithm, Web Security, Brute Force Attack

접수일(2024년 07월 22일), 수정일(1차: 2024년 08월 06일), 게재확정일(2024년 08월 13일)

★ 본 연구는 2021년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임 (No. 2021R1I1A1A0104094313)

* 수원대학교/정보보호학과(주저자)

** 수원대학교/정보보호학과(교신저자)

1. 서 론

웹 환경의 급속한 확장과 함께 사용자들은 다양한 웹 서비스를 이용하고 있다. 웹 서비스는 여러 운영 체제 및 디바이스에서 동작 가능하며, 사용자는 웹 브라우저를 통해 언제 어디서든 서비스를 손쉽게 이용할 수 있다. 이는 본인 인증 만으로도 접근이 가능하다. 흔히 볼 수 있는 로그인 기능은 ID와 패스워드이다. ID와 패스워드를 이용한 사용자 인증은 시스템 구현이 간단하여 컴퓨터 시스템의 가장 오래된 기본적 인증방식이다[1]. 하지만 인터넷 사용자가 증가함에 따라 사용자의 ID와 패스워드나 거래 정보를 탈취하여 악의적인 거래를 시도하는 피싱 기법이 더욱 교묘해지고 있다[2]. 이에 따른, 개인정보 해킹사고가 자주 발생하고 있다. 인터넷 사용률이 높은 환경에서도 사회 곳곳에 보안 사각지대가 존재하며, 웹 해킹에 대한 미흡한 대응이 보안사고 취약성의 주요 원인으로 나타나고 있다[3]. 사용자 편의성을 유지하면서도 안전한 환경을 제공하기 위해 개발자는 보안 방법과 보안 기술을 충분히 알고 적용해야 하며 이를 위해 더욱 안전한 인증 방안이 필요하다.

웹 애플리케이션에서는 사용자 인증과 권한 부여를 효과적으로 관리하기 위해 다양한 보안 기술을 사용한다. 보안 기술의 예로서, OAuth, JSON Web Token(JWT)가 있다. Google, Naver, Kakao 같은 인기 사이트들은 사용자 인증 및 권한 부여를 위해 OAuth 2.0과 JWT를 사용한다. 이들은 OAuth 2.0을 통해 API 인증과 권한 부여를 관리하며, JWT 형식의 액세스 토큰을 사용하여 사용자 정보를 포함하고 안전한 데이터 교환과 시간 기반 권한 부여를 제공한다. OAuth는 클라이언트 집합과 서버 집합 사이의 인증 방식을 표준화한 인증기법으로 여러 애플리케이션을 통합하여 사용하는 것이 가능하다[4]. 하지만 SNS 이용 증가에 따른 로그인 횟수를 줄이려는 OAuth 토큰의 연동 기능을 고려할 때 연동 서비스 인증 과정에서 받을 수 있는 CSRF(Cross Site Request Forgery) 공격에 대한 방어는 OAuth 토큰 자체로는 해결할 수 없어 서버에서 다양한 공격 패턴에 대응해야 하는 문제점이 있다[1].

JWT 기법은 표준 JSON 객체 데이터 형식으로, JSON 형태의 데이터 안에 많은 정보를 담아 전달할 수 있어 토큰의 만료 일자와 같은 정보를 사용할 수 있는 장점이 있다. 하지만 Base64형으로만 인코딩된 헤더는 공격자에게 쉽게 노출해 해당 웹 서비스 인증 시스템에서 어떤 알고리즘을 가지고 있는지 노출되기 쉽다[5].

JWT 토큰의 무결성을 확보하기 위한 알고리즘이 JWT 토큰 서명 알고리즘이다. JWT 토큰 서명 알고리즘에는 Hash-based Message Authentication Code (HMAC), and Elliptic Curve Digital Signature Algorithm(ECDSA)이 있다. 각 알고리즘은 고유한 특징과 보안 수준을 가진다. HMAC 알고리즘은 동일한 키를 사용하여 메시지를 생성하고 확인하는 대칭 키 알고리즘이다[6]. RSA 알고리즘은 공개키와 개인키를 사용하여 서명하는 비대칭 키 알고리즘으로, 확인이 쉽고 신뢰성이 높다[1]. ECDSA 알고리즘은 타원곡선 암호화 기반 암호화 기반의 비대칭 키 알고리즘으로, 공개키와 개인키를 이용해 서명과 검증을 수행한다[6].

본 논문에서는 HMAC와 RSA 토큰 서명 알고리즘을 비교하여 각각의 특징과 보안 수준을 분석하고, 이를 바탕으로 효과적인 토큰 서명 알고리즘 선택을 위한 지침을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 JWT를 활용한 관련 연구 사례를 다루고, 3장에서는 JWT 구조와 두 가지 토큰 서명 알고리즘을 설명하며 성능 및 보안 취약성 검사를 다룬다. 4장에서는 보안 성능 평가 결과를 다룬다. 마지막으로 5장에서 결론 및 향후 연구계획을 소개한다.

2. 관련 연구

이번 섹션에서는 JWT 인증을 활용하여, 기존의 인증 방식에 비해 어떻게 효율성을 높이고 보안성을 강화하는 실 예시를 살펴본다.

2.1 기존 로컬 클라우드 인증방식 개선 사례

기존 로컬 클라우드 인증방식은 주로 ID/PW 방식을 사용하며, Cloud-Agent나 다른 디바이스의 자원 접근 시 상호 간 인증된 키 값을 모바일 디바이스에

저장하여 접근 요청 시 인증 키를 비교하여 권한을 부여 한다. 이러한 방식은 하나의 모바일 디바이스에 다수의 접근 권한을 저장하는 경우, 다수의 키 저장 문제로 불편을 초래할 수 있다. 이를 해결하기 위해 한 연구에서는 JWT 인증 방식과 Zero Configuration을 활용한 접근 방식을 제안한다.

모바일 디바이스가 다른 디바이스 및 Cloud-Agent에 자원 접근을 요청하면, 기존의 ID/PW로 사용자 인증을 요청하고, Cloud-Server에서는 JWT 토큰 방식으로 요청한 아이디 및 권한을 토큰화하여 모바일 디바이스로 전송한다. JWT은 클레임 기반 인증 메커니즘으로, 사용자 정보를 JSON 객체로 포함하고 이를 암호화하여 토큰 형태로 전달하는 방식이다[5]. 이후 모바일 디바이스는 재접근 요청 시 토큰을 사용하여 인증을 진행한다. 접근 권한 확인 후, 로컬 네트워크 내에 Cloud Client가 설치된 네트워크를 탐색하고, Multicast Notification 메시지를 보낸다. 이를 통해 해당 모바일 디바이스를 검증된 사용자로 인식하여 자원 접근을 허용한다[7].

2.2 차량 매칭 서비스에서의 적용 사례

4차 산업혁명의 가속화로 인해 IT 기반 산업들은 제품 중심 사업에서 서비스 위주의 사업으로 변화하고 있다. 차량 공유 시스템을 서비스하는 기업들은 많은 사용자들이 실시간으로 서버에 접속하여 인증을 요청하는 환경을 맞이하며, 이러한 환경에서 서버 기반 인증 방식은 메모리에 부하가 걸리는 문제가 발생할 수 있다. 이를 해결하기 위해 차량 매칭 서비스에서 사용자 인증을 위한 JWT 활용 방식을 제안한다.

사용자가 웹 서비스에 로그인을 위해 핸드폰 번호를 입력하면, 서버에서 해당 핸드폰 번호로 인증번호를 전송하고, 사용자는 이를 웹사이트에 입력하여 로그인을 요청한다. 서버는 사용자의 인증번호와 서버에서 생성한 인증번호가 일치하는 경우 JWT를 생성하여 반환한다. 이후 사용자는 JWT를 통해 인증 정보를 서버에 전달하고, 서버는 매번 인증 정보를 확인하여 인증을 처리한다. 이 방식은 기존의 세션 기반 인증 방식보다 효율적이며, 서버의 메모리 부하를 줄일 수 있다[8].

3. 제안 방법

3.1 JSON Web Token 구조

JWT의 구조는 JSON Web Token(JWT)는 헤더(Header), 페이로드(Payload), 서명(Signature) 세 개로 나누어진다. 토큰은 각 영역을 마침표(.)로 구분하여 하나의 문자열을 이룬다[1]. 헤더는 일반적으로 해당 토큰 타입에 대한 'typ' 값과 해시 알고리즘에 대한 정보 'alg' 값을 담은 JOSE(JavaScript Object Signing and Encryption) 헤더 형태이다[6]. typ은 JWT 이고, alg는 HS256, RS256 등이 있다[9]. (그림 1)은 JWT의 구조를 나타낸다. 페이로드는 클레임(Claim)정보를 포함한다. 클레임은 등록된(Registered) 클레임, 공개 (Public) 클레임, 비공개(Private) 클레임 세 종류가 있다. 헤더와 페이로드는 Base64 Url로 인코딩하고 Signature은 Secret 키를 이용하여 헤더에 지정된 해시 알고리즘으로 서명한다.

(그림 1) JWT의 구조

3.2 토큰 서명 알고리즘

토큰 서명 알고리즘(Token Signature Algorithm)은 주로 토큰 기반의 인증 시스템에서 사용되며, 토큰의 무결성을 보장하고 위조 방지를 위해 서명을 생성하고 검증하는 데 사용된다. 토큰은 주로 JWT 형식으로 사용된다.

3.2.1 대칭 토큰 서명(Symmetric Token Signature)

동일한 키가 서명 및 검증에 사용되며, 주로 HMAC을 사용하여 서명이 생성된다[10]. 토큰을 생성할 때 페이로드와 Secret key를 이용하여 HMAC 함수를 적용하여 서명을 생성한다. 이 방식은 빠른 연산 속도와 간단한 구현이 가능하며 작은 키 크기로도 안전한 서명이 가능하다는 장점이 있다. 하지만 키의 유

출이 발생할 경우 보안이 취약해질 수 있다는 단점이 있다.

3.2.2 비대칭 토큰 서명(Asymmetric Token Signature)

서로 다른 공개키와 개인키 쌍이 사용되며, 주로 RSA 또는 ECDSA 알고리즘을 사용한다. 토큰을 생성할 때 개인키를 이용하여 서명을 생성한다. 공개키는 토큰의 검증 시 사용되며, 생성된 서명과 공개키를 이용하여 검증한다. 이 방식은 키 유출의 영향을 최소화할 수 있다는 장점이 있으며 연산 속도가 대칭 토큰 서명에 비해 상대적으로 느리다는 단점이 있다.

3.3 JWT 토큰 서명 알고리즘 성능 분석

HMAC와 RSA 각각의 서명 및 키 생성 연산에 걸리는 시간을 측정하여 알고리즘 간의 성능을 비교한다.

HMAC 서명 생성 작업을 수행한다. HMAC을 계산할 수 있는 라이브러리를 사용하여 서명을 생성한다. 현재 시간을 기록한 후, HMAC 생성 작업이 완료된 최종 시간을 기록하고 총 경과 시간을 계산한다.

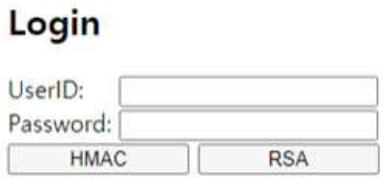
RSA 키 생성 작업을 한다. RSA 키를 생성할 수 있는 라이브러리로 키 쌍을 생성한다. 이를 위해, 2048비트와 1024 비트의 RSA 키를 각각 생성하여 어떤 비트에서 보안이 안전한지 비교한다. 모든 연구는 동일한 방법으로 현재 시간을 기록한 후, 키 생성 작업이 완료된 최종 시간을 기록하여 총 경과 시간을 계산한다.

마지막으로, 계산된 HMAC 서명 생성 경과 시간과 RSA 키 생성 경과 시간을 비교하여 각각의 알고리즘 성능을 평가한다.

3.4 JWT 보안 취약성 분석

3.4.1 인터넷 웹 페이지 구축

HMAC 알고리즘과 RSA 알고리즘의 보안 성능을 비교하기 위해 ID와 패스워드를 사용한 로그인 기능을 구현한 웹 페이지를 만든다. (그림 2)는 HMAC과 RSA를 사용하여 로그인하는 기능을 구현한 폼이다. 두 함수는 동일한 ID와 패스워드로 로그인을 시도한다.



(그림 2) 로그인

3.4.2 무차별 공격

웹 해킹 종류 중 무차별 대입 공격(Brute Force Attack)은 시행착오를 거쳐 비밀번호, 로그인 인증서, 암호화 키를 해킹하는 방법이다[11]. 이러한 공격을 통해 해커는 반복적으로 가능한 모든 조합을 시도하여 무차별적으로 인증을 시도한다.

HS256과 RS256 알고리즘을 적용한 로그인 웹 페이지에서, 사용자 ID와 패스워드를 반복적으로 입력하여 로그인 시도하는 과정을 통해 보안 성능을 비교 분석한다. HS256 알고리즘은 비밀키를 사용하여 토큰을 서명하며, RS256 알고리즘은 비대칭 키 방식으로, 공개키와 개인키를 사용하여 토큰의 서명 및 검증을 수행한다[10]. 비밀키나 개인키가 노출된 상태에서의 연구는 각 알고리즘의 취약점을 분석하고, 보안 성능을 비교한다.

본 연구에서는 사용자별로 토큰을 생성하고, 이 토큰과 함께 ID와 패스워드를 포함한 POST 요청을 서버에 보내 응답을 확인한다. 응답이 성공하면 공격을 성공한 것이며, 실패할 경우 공격을 실패한 것으로 간주한다. 이 과정을 통해 두 알고리즘의 무차별 공격에 대한 보안 취약성을 평가한다.

4. 성능 평가

4.1 알고리즘 생성 시간 측정 비교

생성 시간을 비교해 알고리즘 성능을 평가하기 위해 HMAC 서명과 RSA 키를 생성한 시간을 각각 측정했다. 특정 키와 메시지를 사용하여 HMAC 서명을 생성하는데 걸리는 경과 시간과 RSA 키 쌍을 생성하

는 데 소요되는 시간을 포함하여 측정했다.

4.1.1 HMAC 서명 생성 시간 측정

'HS256' 알고리즘을 사용하여 JWT 토큰을 생성하는 데 걸린 시간은 1044.75 μs로 나타난다. 이는 아주 빠른 속도로 토큰이 생성되었음을 알 수 있다. 'HS256' 알고리즘은 HMAC-SHA256을 사용하는 대칭 키 알고리즘으로 빠른 속도로 서명을 생성할 수 있는 장점을 가지고 있다. 이러한 결과는 시스템의 성능이 우수하거나 알고리즘이 효율적으로 구현되었다는 것을 나타낸다. (그림 3)와 (그림 4)은 HMAC 서명을 생성하는 식과 그 결과이다.

```python
def generate_hs256(secret, data):
    start_time = time.time()

    jwt_header = {"alg": "HS256", "typ": "JWT"}
    jwt_payload = {"sub": "1031", "name": "han kuang", "iat": 1516239022}

    jwt_token = generate_jwt(jwt_header, jwt_payload, secret)

    end_time = time.time()
    elapsed_time = (end_time - start_time) * 1e6

    return jwt_token, elapsed_time

# 시크릿 키, 메시지
secret_key = "test_key"
data = "hvbiparngilnsklfnklweopthklndlfk"
```

(그림 3) HMAC 측정

```
Generated HS256 JWT Token:
eyJhbGciOiAiSFMyNTYiLCAidHlwIjogIkpXVCJ9.eyJzdWIiOiAiMTAz
MSIsICJuYW1lIjogImhhbiBrdWFuZyIsICJpYXQiOiAxNTE2MjM5MDI
yfQ.O6UnAnL0VLHQYr6F4R4O-dpq5gGV_MKXWxD6Cu1r4n4

Elapsed Time (HS256): 1044.7502136230469 microseconds
```

(그림 4) HMAC 결과

4.1.2 RSA 키 생성 시간 측정

PyCryptodome 라이브러리를 사용하여 2048 비트 RSA 키를 생성한다. 실험 결과로, RSA 키 생성 시간은 925919.77 μs가 소요되었다. 생성된 RSA 키는 공개키와 개인키의 쌍으로 구성되며, 공개키는 'rsa_key.publickey()' 메서드를 통해 얻을 수 있다. 반면, 개인키는 'rsa_key' 자체에 포함된다. 1024 비트 RSA 키를 생성해 본 결과, RSA 키 생성 시간은 68254.71 μs가 소요되었다. 이러한 결과는 1024 비트 RSA가 2048 비트 RSA보다 보안 측면에서 취약하다는 것을 시사한다. 따라서 보안이 중요한 상황에서는 보다 큰 비트 수를 권장한다. 2048 비트 키의 생성은 1024 비트에 비해 더 많은 계산이 필요하다. (그림 5)와 (그림 6)은 RSA 키를 생성하는 식과 그 결과이다.

```python
def generate_rsa_key(bits):
    start_time = time.time()
    key = RSA.generate(bits)
    end_time = time.time()
    elapsed_time = end_time - start_time
    return key, elapsed_time

# RSA 키 생성 및 소요 시간 출력
bits = 2048
rsa_key_2048, elapsed_time_rsa_2048 = generate_rsa_key(bits)
print(f"Generated {bits}-bit RSA Key:", rsa_key_2048.export_key().decode('utf-8'))
print(f"RSA Elapsed Time ({bits} bits):", elapsed_time_rsa_2048, "microseconds")
```

(그림 5) RSA 측정

```
Generated RSA Key:
MIIEowIBAAKCAQEAnVYzh7Plj3pp9RxaEtBVNt7RDKHRmEXrmucemNyulLAyQBN0
C+p5qAKjX6Umw+JE+PkAR89AEp4Qf1YUsjssutgBephTDa/qlCLR5zQdvpmHpc1K
+o+XJhcj/Cm73co4NN6LA5Bac0nwrqKjfVQUyboDl1HnZMJTE7IKN5UHY2IssepQ
ErMgDYL7MvElfgSXoFP/naqhYf93+/fTxP7YFYXdW/mo2nv2tsl1nNnzkgMnyPmw
7dhngccA0VmoyTNT9t2Tuo+juoagHelTFqW2NEidPDj0MHv0Yi9peSdrC67voGNs
... LAcvC1jlUst/ADwb3sc0hdeGNwlXvHGqHEYFB9y10tzFIMBgNnlIYdHEVlqTzdS/
SDfhcgzqx4ivGenpzfi7bgGucdWrRRrYSpJceXhWzFYqZ9rpuK4d

RSA Elapsed Time (2048 bits): 925919.771194458 microseconds
```

(그림 6) RSA 결과

4.1.3 결과 비교

'HS256' 알고리즘을 사용한 JWT 토큰 생성은 매우 빠른 속도로 생성된다. 1044.75 μs로 나타난 이 결과는 HMAC-SHA256을 사용하는 대칭 키 알고리즘으로 인해, 효율적인 서명 생성이 가능하다.

반면, 2048 비트 RSA 키 생성은 925919.77 μs, 1024 비트 RSA 키 생성은 68254.7 μs가 소요되었다. 2048 비트 RSA 키 생성은 1024 비트 RSA 키 생성보다 약 13.56배 정도의 시간이 더 필요하다. 비교적 작은 키 길이로 인해 1024 비트 RSA 키 생성은 빠른 속도를 보이지만, 공격자가 해독할 가능성이 높다. 따라서 보안이 중요한 상황에서는 2048 비트 이상의 RSA 키를 사용하는 것을 권장한다.

정리하면, 'RS256'은 'HS256'에 비해 키 길이에 따른 보안성 측면에서 더 안전하다. 'HS256' 알고리즘은 빠르고 효율적이지만, 대칭 키 관리에 어려움이 있다. RSA 키는 길이가 길수록 보안이 강화되지만, 키 생성 및 암호화에 소요되는 시간도 증가한다. 따라서 보안 요구 사항과 성능 간의 균형을 고려하여 적절한 키 길이를 선택해야 한다. 보안이 중요한 상황에서는 2048 비트 이상의 RSA 키를 사용하는 것이 좋으며, 처리 성능이 우선되는 경우 'HS256'과 같은 대칭

키 알고리즘을 사용하는 것이 권장된다.

4.2 알고리즘 보안 성능 평가

본 절에서는 HS256 알고리즘과 RS256 알고리즘을 사용한 로그인 기능을 갖춘 웹페이지를 구현하고 각 알고리즘에 대한 무차별 공격에 대한 보안 성능을 평가한다.

HS256 알고리즘은 클라이언트가 서버로부터 받은 인증 정보를 안전하게 저장하고 요청을 보내는 방식으로 서버와 클라이언트 간의 통신을 보호한다. (그림 7)은 시크릿 키를 받은 토큰을 서명하고 HS256 알고리즘을 사용한 웹 사이트의 무차별 공격 코드의 일부이다. ID와 패스워드를 포함하여 로그인 페이지에 POST 요청을 서버로 보낸 후, 서버는 이를 받아 사용자의 로그인을 검증한다. 서버의 응답은 사용자의 로그인 성공 여부에 따라 다르며, 이를 통해 로그인 성공 여부를 확인한다.

```python
def generate_token(username):
    payload = {
        'username': username,
        'exp': datetime.datetime.utcnow() + datetime.timedelta(minutes=30)
    }
    token = jwt.encode(payload, SECRET_KEY, algorithm='HS256')
    return token


def verify_token(token):
    try:
        payload = jwt.decode(token, SECRET_KEY, algorithms=['HS256'])
        return payload['username'], token  # 토큰과 함께 사용자 이름도 반환
    except jwt.ExpiredSignatureError:
        return None, None
    except jwt.InvalidTokenError:
        return None, None
```

(그림 7) 보안 성능을 위한 HS256 토큰 생성 및 검증

RS256 알고리즘을 사용한 JWT는 토큰에 사용자의 정보와 만료 기간을 포함하기 때문에, 각 로그인 시도마다 새로운 토큰을 생성함으로써 로그인 시도 횟수의 제한을 우회할 수 있는 가능성을 고려하였다. 토큰은 발행된 시점으로부터 30분 후에 만료되도록 설정되어, 30분이 지나면 사용할 수 없게 된다. RS256 알고리즘은 공개키와 개인키를 사용하여 토큰의 서명 및 검증을 수행하기에, HS256 알고리즘과 다르게 개인키를 사용하여 토큰을 서명한다. (그림 8)은 RS256 알고리즘을 사용한 웹 사이트의 무차별 공격을 위해 만들어진 코드의 일부이다.

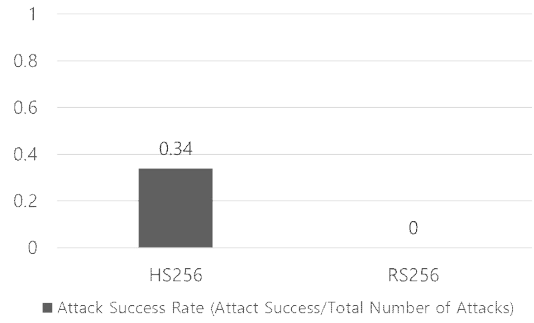
```python
def generate_token(username):
    payload = {
        'username': username,
        'exp': datetime.datetime.utcnow() + datetime.timedelta(minutes=30)
    }
    token = jwt.encode(payload, SECRET_KEY, algorithm='RS256')
    return token.decode('utf-8')
```

(그림 8) 보안 성능을 위한 RS256 토큰 생성 및 검증

4.3 무차별 공격 결과

HS256 알고리즘의 실험 결과는, 50번의 시도 중 33번은 사용자 정보가 옳지 않다는 이유로 실패하였고 17번은 무차별 공격이 성공하였다. 반면, RS256 알고리즘을 사용했을 때는 공개키와 개인키를 사용한 올바르지 않은 서명 때문에 토큰이 거부되었고, 50번의 시도 중 모두 공격이 성공하지 못했다. 이는 RS256이 비대칭 키 알고리즘으로, 개인키가 노출되지 않는 한 서명을 위조하기 어렵기 때문에 발생한다.

이에 따라 HS256 알고리즘을 사용한 웹 사이트에서의 무차별 공격 성공률은 0.34으로 나타났으며, RS256 알고리즘을 사용한 웹 사이트의 무차별 공격 성공률은 0이었다. (그림 9)은 HS256과 RS256 알고리즘의 무차별 공격 성공률을 차트로 나타낸 것이다.



(그림 9) HS256 및 RS256 공격 성공률

웹사이트에 대한 무차별 공격 실험에서, HS256 알고리즘은 토큰 검증 시 비밀키만을 사용하기 때문에 무차별 공격에 취약한 반면, RS256은 공개키와 개인키를 활용하여 토큰을 서명하고 검증함으로써 더 높은 보안성을 제공한다. 따라서, 보안 측면에서 RS256 알고리즘은 HS256 알고리즘에 비해 상대적으로 안전하며, 이는 사용자 인증 정보를 안전하게 보호할

수 있다.

본 연구는 HS256과 RS256알고리즘의 보안 성능을 비교하기 위해, 비밀키와 개인키가 노출된 상황에서 취약점을 분석하였다. 실제 환경에서는 보안성을 강화하기 위해 추가적인 인증절차를 도입하는 것이 중요하다[9]. 이러한 절차는 HS256 알고리즘의 무차별 대입 공격에 대한 취약점을 보완할 수 있으며, 알고리즘 선택과 더불어 추가적인 보안 조치를 함께 추가적인 보안 조치를 함께 고려하는 것이 보안성을 높이는 데 필수적이다.

5. 결론 및 향후 연구계획

본 연구에서는 웹 애플리케이션 보안 강화를 위해 토큰 서명 알고리즘인 HMAC와 RSA를 비교 분석한다. 각 알고리즘은 JWT 토큰의 무결성 보장을 위해 고유한 특징과 보안 수준을 제공한다.

HMAC 알고리즘은 대칭 키를 사용해 서명 생성과 검증을 간편하게 할 수 있는 반면, RSA 알고리즘은 공개키와 개인키를 사용해 높은 신뢰성을 보장한다. HS256과 RS256 알고리즘의 보안 성능을 비교함으로써, 선택된 알고리즘은 웹 애플리케이션의 보안 요구와 성능 우선순위에 따라 결정되어야 한다. 이는 서버에 대한 보호와 성능 측면에서 중요한 고려 사항이다. 이 연구 결과를 바탕으로 JWT 토큰을 사용할 때 RS256 알고리즘을 선택하여 사용자 인증 및 데이터 보호를 강화할 수 있다.

본 연구에서는 보안 강도와 성능 차이를 비교하기 위해 HS256과 RS256 알고리즘에 초점을 맞추어 분석하였다. 향후 연구에서는 다뤄지지 않은 다른 토큰 서명 알고리즘들에 대한 비교 연구를 수행하여, 알고리즘들의 성능을 최적화하고 서버 공격에 대비하는 효과적인 방법에 대한 후속 연구를 진행할 것이다.

참고문헌

[1] Kang, Hee-Bog, Haeng-Cheon Jang, and Chang-Soo Jang, "IUWT Based Token Authentication Technology", Journal of KIIT, Vol. 17, No. 2, pp. 143-150, Feb. 28, 2019.

[2] Sung-hoon Lee, Seung-hyun Kim, Eui-yeob Jeong, Dae-seon Choi, and Seung-hun Jin, "An Attack of Defeating Keyboard Encryption Module using Javascript Manipulation in Korean Internet Banking", Journal of The Korea Institute of Information Security & Cryptology, VOL.25, NO.4, Aug. 2015.

[3] SiChoon Noh, "A Study of Web Hacking Response Procedures Model based on Diagnosis Studies for Cross-Site Scripting (XSS)Process", Journal of Information and Security, Vol.13, NO.6, Dec. 2013.

[4] Sung-Tae Yu and Soo-Hyun Oh, "OAuth-based User Authentication Framework for Internet of Things", Journal of the Korea Academia-Industrial cooperation Society, Vol. 16, No. 11 pp. 8057-8063, 2015.

[5] JinKwang Son, "Vulnerability Analysis and Secure Coding Implementation Measures for Authentication Server Based on JWT", Proceedings of the Korean Information Science Society Conference, 2016.

[6] M. Jones, "RFC7519 - JSON Web Token (JWT)", IETF, 2024. https:/tools.ietf.org/html/rfc7519.

[7] Sang-Ho Jeon, Ki-Hyung Kim, Kang-seok Kim, "A Study for Accessing Cloud Agent using JWT and Zero Configuration", Dept. of Information Security Engineering, Ajou University, pp. 287-288, Nov. 2016.

[8] UnDong Kim, and YoHan Park, "Efficient Authentication Scheme using JWT for Ride Sharing Services", 2020 Korea Broadcasting and Media Engineering Society (KOSBE) Autumn Conference, pp. 332-334, Nov. 2020.

[9] Kim HyeongGyeom and Kim KiCheon, "A Study on FIDO UAF Federated Authentication Using

JWT Token in Various Devices", Journal of the Korea Society of Digital industry and Information Management, Vol. 16, No.4, Dec. 2020.

[10] Clara Do, Weronika Kolodziejak and Juliusz Chroboczek, "RFC8967 – MAC Authentication for the Babel Routing Protocol", IETF, 2024. https://tools.ietf.org/html/rfc7519.

[11] Fortinet, "What is a Brute Force Attack?", 2024. https://www.fortinet.com/resources/cyberglossary/brute-force-attack.

〔저 자 소 개〕

이 채 은 (Chaeeun Lee)
2022년 3월～현재 수원대학교 정보보호학과 학사과정
email : eundery@naver.com

전 상 훈 (Sanghoon Jeon)
2012년 2월: 경북대학교 IT대학 심화전자공학 공학사
2014년 2월: 대구경북과학기술원 정보통신융합공학전공 공학석사
2020년 8월: 대구경북과학기술원 정보통신융합전공 공학박사
2020년 3월～8월: 한양대학교 산학협력단 선임연구원
2020년 9월～2022 9월: 한양대학교 의과대학 응급의학과 포닥연구원
2022년 10월～2023 9월: 한양대학교 의과대학 응급의학과 연구조교수
2023년 10월～현재: 수원대학교 지능형SW융합대학 정보보호학과 조교수
email : shjeon@suwon.ac.kr