Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.28, NO.6, Dec. 2018 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2018.28.6.1343

OAuth 2.0 MAC 토큰인증의 효율성 개선을 위한 무상태 난수화토큰인증*

이 병 천†‡
중부대학교 정보보호학과

Stateless Randomized Token Authentication for Performance Improvement of OAuth 2.0 MAC Token Authentication*

Byoungcheon Lee†‡
Department of Information Security, Joongbu University

요 약

표준 인증유지기술로 널리 사용되고 있는 OAuth 2.0 Bearer 토큰[1,2], JWT(JSON Web Token)[3,4] 기술은 고정된 토큰을 반복 전송하는 방식을 이용하고 있어서 네트워크 공격자에 의한 도청공격에 취약하기 때문에 HTTPS[6] 보안통신 환경에서 사용해야 한다는 제약이 있다. 평문통신 환경에서도 사용할 수 있는 인증유지기술로 제안된 OAuth 2.0 MAC 토큰[5] 기술은 서버가 인증된 클라이언트에게 공유된 비밀키를 발급하고 인증요청시 이를 이용하여 MAC 값을 계산하여 제시하는 방식을 사용하는데 서버는 사용자별 공유비밀키를 관리해야 하므로 무상태(stateless) 인증을 제공할 수 없다는 단점이 있다. 이 논문에서는 서버측에서 사용자별 공유비밀키를 관리하지 않고도 무상태 MAC 토큰 인증을 수행할 수 있도록 개선된 난수화 토큰인증 프로토콜을 제시한다. 전체 인증과정에서 HTTPS를 사용할 필요가 없도록 하기 위하여 서버인증서를 이용한 초기인증보안, 인증서토큰을 이용한 전자서명 간편로그인 등의 기술을 난수화 토큰인증 기술과 결합하여 적용함으로써 완전한 무상태형 인증서비스를 제공할 수 있도록 설계하였으며 그 구현 사례를 제시한다.

ABSTRACT

OAuth 2.0 bearer token[1,2] and JWT(JSON web token)[3,4], current standard technologies for authentication and authorization, use the approach of sending fixed token repeatedly to server for authentication that they are subject to eavesdropping attack, thus they should be used in secure communication environment such as HTTPS[6]. In OAuth 2.0 MAC token[6] which was devised as an authentication scheme that can be used in non-secure communication environment, server issues shared secret key to authenticated client and the client uses it to compute MAC to prove the authenticity of request, but in this case server has to store and use the shared secret key to verify user's request. Therefore, it's hard to provide stateless authentication service. In this paper we present a randomized token authentication scheme which can provide stateless MAC token authentication without storing shared secret key in server side. To remove the use of HTTPS, we utilize secure communication using server certificate and simple signature-based login using client certificate together with the proposed randomized token authentication to achieve the fully stateless authentication service and we provide an implementation example.

Keywords: token authentication, JWT, OAuth, stateless authentication

Received(09. 11. 2018), Modified(11. 13. 2018), Accepted(11. 13. 2018)
* 이 논문은 2018년도 중부대학교 학술연구비 지원에 의하여 이루어진 것임

† 주저자, sultan@joongbu.ac.kr
‡ 교신저자, sultan@joongbu.ac.kr(Corresponding author)

I. 서 론

웹서비스의 활용범위가 크게 증가하고 웹서비스에 대한 다양한 공격사례들이 제시되면서 웹서비스의 보안이 매우 중요해지고 있다. 웹서비스의 보안을 위해 다양한 암호 및 보안기술들이 사용되는데 모든 보안의 출발점은 사용자인증이라고 생각할 수 있다. 이것은 웹서비스에 대한 각종 보안대책을 적용함에 있어서도 사용자를 정확하게 인증한 이후에야 의미가 있게 되기 때문이다. 현재 접속하고 있는 사용자가 정당한 사용자인지, 혹시 남의 신원을 도용하려고 시도하는 공격자는 아닌지 구별하는 것이 매우 중요하다.

웹서비스에서의 사용자인증은 초기인증과 인증유지의 두 가지로 나누어 생각해볼 수 있다. 초기인증(initial authentication)은 웹서비스에 처음 접근하는 사용자의 신원을 정확하게 확인하는 것을 의미하는데 이때 사용하는 인증기술은 패스워드, PIN 등의 지식기반 인증, 지문, 홍채 등의 생체기반 인증, 인증서, 스마트카드 등 소유기반 인증 등의 잘 알려진 인증기술들을 사용하게 된다. 초기인증시 서버는 사용자가 전송하는 인증정보를 자신이 가지고 있는 사용자계정 데이터베이스의 내용과 비교하여 사용자의 신원을 엄밀하게 확인하여야 하며 사용자가 전송하는 인증정보를 네트워크 공격자로부터 보호하기 위해 HTTPS 등의 안전한 보안통신채널을 적용해야 한다.

한편 인증유지(keeping authentication) 기술은 초기인증된 사용자의 인증된 상태를 유지시켜서 초기인증을 다시 요구하지 않고도 서비스를 오랜기간 효율적으로 제공하기 위한 기술로서 쿠키, 세션, 토큰 등의 기술을 사용한다. 인증유지 기술은 웹서버가 다수의 동시접속 사용자들에게 서비스를 효율적으로 제공할 수 있도록 하기 위해 매우 중요한 기술이며, 서비스 제공자의 효율성과 확장성을 위해 웹서버가 사용자 정보를 별도로 관리할 필요가 없이 사용자가 제시하는 정보만으로 인증을 즉시 확인할 수 있도록 하는 무상태(stateless) 인증을 제공하는 것을 목표로 하고 있다. 그 중에서도 토큰인증 기술은 초기로그인에 성공한 사용자에게 서버가 서명된 토큰을 발급하여 사용자 인증유지에 토큰을 사용하는 방식으로 OAuth 2.0 Bearer 토큰[1,2], JWT (JSON Web Token)[3,4] 등의 기술이 널리 사용되고 있다. 이러한 토큰인증을 이용한 인증유지 기술에서는 서비스 요청시마다 동일한 토큰을 서버에 반복적으로 전달하는 방식을 사용하는데 만일 도청공격에 의해 네트워크 공격자에게 토큰을 탈취당하면 인증세션을 탈취당하는 것과 동일하기 때문에 사용자가 서버에게 토큰을 안전하게 전달할 수 있도록 서버는 HTTPS 보안통신을 적용하여야 한다.

한편 보안통신을 사용하기 어려운 환경에서도 사용할 수 있는 인증유지기술로 제안된 OAuth 2.0 MAC 토큰[5] 기술은 서버가 인증된 클라이언트에게 공유된 비밀키를 발급하고 서비스 요청시 클라이언트는 공유된 비밀키를 이용하여 요청에 대한 MAC 값을 계산하여 서버에 제시하는 방식으로 매 요청시마다 인증값이 달라지므로 HTTPS 보안통신을 사용할 필요가 없다는 장점이 있다. 그러나 MAC 토큰인증 기법에서는 서버측에서 클라이언트의 요청을 검증할때마다 공유된 비밀키를 사용해야 하기 때문에 공유키를 관리하는 것이 큰 부담이 되고 무상태 인증을 제공할 수 없게 된다. MAC 토큰 기법은 2011년에 제안되었지만 널리 확산되지 못하고 아직도 Internet Draft 상태로 머물러 있다.

웹서비스에서 보안통신을 제공하는 HTTPS[6] 기술은 서버와 클라이언트 사이의 상호 인증과 통신세션 보안을 자동으로 제공할 수 있는 기술로 널리 사용되고 있는 표준 통신기술이다. 그러나 서버의 입장에서는 HTTPS 보안통신을 적용하는 것은 효율성, 확장성 측면에서 많은 부담이 된다. 서버는 현재 접속하고 있는 클라이언트의 보안통신 세션정보를 유지하고 관리하여야 하므로 인증유지기술이 추구하는 무상태 인증을 구현하기 어렵다. 즉 적용하는 인증유지기술이 Bearer 토큰 기술과 같이 무상태 인증을 제공하더라도 함께 사용해야 하는 HTTPS 때문에 최종적으로는 무상태 인증을 제공할 수 없게 된다. 특히 분산서버 환경에서 대규모 웹서비스를 제공하는 경우에는 세션정보를 유지해야 한다는 것이 큰 제약이 된다. 만일 HTTPS 보안통신을 사용하지 않고도 무상태 인증유지 기술을 안전하게 적용할 수 있다면 진정한 무상태 인증을 제공하여 웹서비스의 효율성과 확장성을 크게 향상시킬 수 있을 것으로 기대된다.

이 논문에서는 서버측에서 사용자별 공유비밀키를 관리하지 않고도 무상태 MAC 토큰인증을 수행할 수 있도록 개선된 난수화 토큰인증 프로토콜을 제시한다. 또한 초기인증 및 인증유지의 전체 인증과정에서 HTTPS를 사용하지 않도록 하기 위하여 초기 접속시 서버인증서 배포 및 이를 이용한 초기인증보안, 클라이언트에게 인증서토큰 발급 및 이를 이용한 전

자서명 간편로그인, 난수화 토큰인증을 이용한 인증유지기술을 결합하여 적용함으로써 무상태형 인증서비스를 제공할 수 있도록 설계하였으며 그 구현 사례를 제시한다. 이러한 개선된 인증방식은 웹서비스의 인증을 강화하면서도 웹서비스 사업자의 효율성과 확장성을 높이며 사용자 편의성을 크게 향상시킬 수 있을 것으로 기대된다.

이 논문의 구성은 다음과 같다. 2장에서는 초기인증기술, 인증유지기술, 보안통신기술 등 이 논문을 이해하는데 필요한 기반 기술들의 현황을 소개한다. 3장에서는 제안된 인증서비스를 구현하는데 필요한 요소기술로서 기존의 MAC 토큰인증 기술을 개선한 난수화 토큰인증, 서버인증서를 이용한 초기인증보호, 인증서토큰을 이용한 전자서명 간편로그인 등의 기술을 소개한다. 4장에서는 이들 요소기술들을 결합 이용하여 무상태 MAC 토큰인증 서비스를 구현하는 사례를 보인다. 5장에서는 구현된 서비스의 효율성과 보안성을 분석하고 6장에서 결론을 맺는다.

II. 관련 연구

2.1 상태형, 무상태형 서비스

웹서비스의 효율성을 논의하는데 있어서 상태형(stateful) 서비스인지, 무상태형(stateless) 서비스인지는 매우 중요하다. 서버와 클라이언트 사이에 통신이 연결되어 지속적으로 서비스되는 경우, 그리고 서버는 다수의 동시접속 클라이언트에게 서비스를 제공해야 하는 경우를 고려해보자. 서버가 클라이언트의 연결정보를 특별히 관리할 필요가 없는 경우는 무상태형 서비스라고 할 수 있다. 표준 웹프로토콜인 HTTP는 대표적인 무상태형 프로토콜로 서버는 많은 동시접속자들에게 효율성 높게 서비스를 제공할 수 있다. 반면 서버가 클라이언트와의 연결정보를 관리하고 지속적으로 이용해야 하는 경우는 상태형 서비스라고 한다. 표준 웹보안통신 프로토콜인 HTTPS는 서버와 클라이언트가 보안세션을 맺고 암호화통신을 위해 세션키를 관리하고 사용해야 하므로 상태형 서비스이다. 웹서비스에 HTTPS를 적용하면 서버는 암호화통신에 대한 부담과 함께 동시접속사용자들의 세션정보를 관리해야 하는데 분산형서버들로 운영되는 대규모 웹서비스에서는 이런 부담이 가중된다. 이 논문은 인증을 효율화하기 위한 무상태 인증서비스에 대한 것이다.

2.2 초기인증기술

초기인증(initial authentication)이란 사용자가 서버에 처음 접속하는 경우 서버가 사용자의 인증정보를 전달받아 사용자의 신원을 엄밀하게 인증하는 것을 말한다. 이때 사용하는 인증기술은 패스워드, PIN 등의 지식기반 인증, 지문, 홍채 등의 생체기반 인증, 인증서, 스마트카드 등 소유기반 인증 등의 잘 알려진 인증기술들을 사용하게 된다.

서버는 사용자 등록과정을 통해 사용자의 인증정보를 제공받아 사용자계정 데이터베이스에 저장하게 된다. 초기인증시에는 사용자가 제공하는 인증정보를 서버가 관리하고 있는 사용자계정 데이터베이스의 내용과 비교하여 신원을 확인하므로 상태형 인증이며 서버의 자원을 많이 사용하게 된다. 사용자 인증정보는 신원을 보증하는 중요한 정보로서 안전하게 보호해야 하며 만일 공격자가 도청공격을 통해 사용자 인증정보를 획득하게 되면 공격자가 사용자의 신원을 도용할 수 있게 된다. 그러므로 초기인증 과정을 보호하기 위해서는 사용자 등록시 및 초기인증시 HTTPS 등의 안전한 보안통신 기술이 적용되어야 한다.

2.3 인증유지기술

인증유지(keeping authentication) 기술은 초기인증된 사용자의 인증된 상태를 유지시켜서 초기인증을 다시 요구하지 않고도 서비스를 오랜기간 안전하게 제공하기 위한 기술이다. 인증유지 기술을 사용하지 않는다면 사용자는 서버로부터 지속적인 초기인증 요청을 받게 될 것이고 이것은 사용자 편의성을 크게 저해하게 된다. 또한 서버 측면에서도 초기인증을 처리하기 위해서는 서버의 자원을 많이 사용해야 하므로 부담이 커지게 된다. 현재 널리 사용되고 있는 대부분의 대규모 포털서비스들이 사용자에게 초기인증을 자주 요구하지 않는 이유는 사용자 편의성 향상과 서버의 효율성 향상을 위해 인증유지기술을 적극 활용하기 때문이다.

인증유지 기술로는 쿠키, 세션, 토큰 등의 기술이 있다. 인증유지 기술은 웹서버가 사용자 정보를 별도로 관리할 필요가 없이 사용자가 온라인으로 제시하는 정보만으로 인증을 즉시 확인할 수 있도록 하는 무상태 인증을 제공하는 것을 목표로 하고 있으며 이것은 웹서비스의 효율성과 확장성 측면에서 매우 중

요하다. 대규모 웹서비스를 운영하는 사업자의 입장에서는 서버의 성능 측면에서 볼 때 간헐적으로 발생하는 초기인증 처리의 부담보다도 현재의 동시접속자에 의해 지속적으로 발생하는 인증유지의 효율적인 처리가 더욱 중요하다.

쿠키(cookie)는 서버가 초기인증된 사용자에게 제공하는 작은 정보 파일로 클라이언트측(브라우저)에 저장하고 관리하는 정보이다. 클라이언트는 서버에 접속시 쿠키값을 함께 전송하고 서버는 쿠키값을 확인하여 인증을 유지하게 된다. 쿠키에는 사용자 인증이 유효한 시간을 명시할 수 있으며, 유효 시간이 정해지면 브라우저가 종료되어도 인증이 유지될 수 있다. 쿠키는 무상태 인증유지 방식으로 서버는 사용자의 정보를 유지할 필요가 없어서 효율적이다.

세션(session)은 사용자 정보를 브라우저에 저장하는 쿠키와 달리 서버측에서 관리하는 상태형 인증유지 방식이다. 서버는 클라이언트가 초기인증을 통과하면 로그인된 사용자의 세션정보를 서버의 메모리, 디스크, 데이터베이스 등에 관리하고 클라이언트를 구분하기 위한 세션 ID를 생성하여 클라이언트에게 발급한다. 클라이언트는 쿠키를 이용하여 세션 ID를 저장한다. 클라이언트는 서버에 다시 접속시 세션 ID 값을 서버에 전달하며 서버는 이 정보로부터 세션정보를 확인하여 클라이언트의 인증상태를 유지한다. 세션은 로그인된 사용자에 대한 정보를 서버에 두기 때문에 쿠키보다 보안에 좋지만, 사용자가 많아질수록 서버 메모리를 많이 차지하게 되며 동시접속자 수가 많은 웹사이트인 경우 서버에 과부하를 주게 되므로 성능 저하의 요인이 된다. 쿠키와 세션의 정보는 단순한 스트링 정보로서 보안성이 부족하지만 네트워크 공격자가 취득하게 되는 경우 인증을 탈취할 수 있게 되므로 이를 보호하기 위해서는 HTTPS 보안통신 환경에서 사용하여야 한다는 제약이 있다.

쿠키와 세션의 이런 장단점을 보완하기 위하여 토큰 기반의 인증방식이 제안되었다. 토큰은 초기로그인에 성공한 사용자에게 서버가 서명된 토큰을 발급하여 제공하는 것으로 클라이언트는 발급받은 토큰을 브라우저에 저장해두고 서버에 요청시마다 해당 토큰을 함께 서버에 전달한다. 토큰은 서버의 서명이 포함된 값으로 제3자가 위조할 수 없으며 해당 서버만이 생성하고 검증할 수 있다. 서버는 자신이 서명하여 발급한 토큰을 검증하고 사용자의 인증상태를 유지할 수 있는데 서버에는 로그인된 사용자정보를 관리하지 않아도 되는 무상태 인증유지 방식이다.

2.4 OAuth 2.0 표준

토큰인증 방식은 OAuth 2.0 Bearer 토큰[1,2], JWT(JSON Web Token)[3,4] 등의 기술로 표준화되어 현재 대규모 웹서비스들에서 널리 사용되고 있다. Fig. 1은 JWT 토큰의 구조를 나타내는데 서버는 헤더와 페이로드에 인증된 사용자 정보를 입력하고 서버의 비밀키를 이용하여 이들의 HMAC값을 계산하고 이것을 서명값으로 저장한다. JWT 토큰의 검증을 위해서는 서버의 비밀키를 이용하여 동일한 HMAC 값이 생성되는지 검증하는데 이것은 해당 서버만이 유효성을 검증할 수 있다. 서버의 비밀키는 토큰의 생성과 검증에만 사용되는 비밀값으로 외부에 노출될 필요가 없다.

Bearer 토큰을 이용한 인증유지 기술에서는 서비스 요청시마다 발급받은 토큰을 서버에 반복적으로 전달하는 방식을 사용하는데 이것은 네트워크 도청공격에 취약하다. 만일 네트워크 공격자가 도청공격에 의해 이런 정보를 획득하게 된다면 사용자의 인증을 탈취하는 것과 동일하기 때문에 사용자가 서버에게 토큰 정보를 안전하게 전달할 수 있도록 서버는 HTTPS 보안통신을 적용하여야 한다. 그러나 무상태 인증을 추구하는 인증유지 기술에 HTTPS 프로토콜을 함께 사용하게 되면 HTTPS 프로토콜 자체가 상태형 프로토콜이기 때문에 무상태 인증유지 기술을 사용하는 장점을 살릴 수 없게 된다는 단점이 있다.

OAuth 2.0 MAC 토큰[5] 기술은 보안통신을 사용하지 않고도 안전한 인증유지를 제공하기 위해 제안되었다. MAC 토큰 기술에서는 서버가 초기인증된 클라이언트에게 Fig. 2에 보인 바와 같은 토큰을 발급한다. 여기서 access_token은 JWT와 같은 bearer 토큰을 사용하며 token_type은 mac으로 선언되었다. 공유된 비밀키인 mac_key가 발급되었으며 kid는 mac_key를 검색하는데 사용하는 key

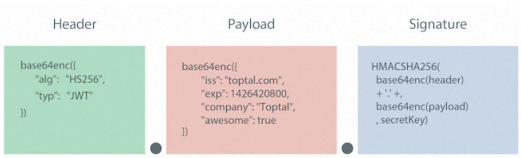

Fig. 1. JSON Web Token

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store

{
  "access_token":
"eyJhbGci0iJSU0ExXzUiLCJlbmMi0iJBMTI4QOJDK0hTMjU2In0.
pwaFh7yJPivLjjPkzC-GeAyHuy7AinGcS51AZ7TXnwkC8O0w1aW47kcT_UV54ubo
n0NbeArw0VuR7shveXnwPmucwrk_30CcHrCbE1HR-Jfme2mF_WR3zUMcwqmU0RIH
kwx9txo_sKRasjlXc8RYP-evLCmT1XRXKjtY5l44Gnh0A84hGvVfMxMfCWXh38hi
2h8JMjQHGQ3mivVui5lbf-zzb3qXXxN01ZYoWgs5tP1-T54QYc9Bi9wodFPWNPKB
kY-BgewG-Vmc59JqFeprk1008qhKQe0GCWc0WPC_n_LlpGWH6spRm7KGuYdgDMk0
bd4uuB0uPPLx_euVCdrVrA.
AxY8DCtDaGlsbGljb3RoZO.
7Ml2lRCaoyYx1HclVXkr8DhmDoikTm0p3ldEmm4qgBThFkqFq0s3ivXLJTku4M0f
laMAbGG_X6K8_B-0E-7ak-0lm_-_V03oBUUGTAc-F0A.
0wWNxnC-BMEie-GkFHzVWiNiaV3zUHf6fC0GTwbRckU",
    "token_type":"mac",
    "expires_in":3600,
    "refresh_token":"8xL0xBtZp8",
    "kid":"22BIjxU93h/IgwEb4zCRu5WF37s=",
    "mac_key":"adijq39jdlaska9asud",
    "mac_algorithm":"hmac-sha-256"
}
```

Fig. 2. MAC Token

ID이다. 클라이언트에 이러한 토큰이 발급되면 서버는 이후의 사용을 위해 kid와 mac_key를 DB에 저장하여야 한다.

클라이언트는 서비스 요청시 위의 mac_key를 이용하여 현재시간, nonce 및 요청내용에 대한 HMAC 값을 계산하여 서버에 전송한다. 서버는 클라이언트가 제시한 kid를 이용하여 mac_key를 찾아오고 HMAC 값의 유효성을 검증한다. 이와 같이 mac_key 값은 네트워크 통신에 노출되지 않으면서 인증을 위한 HMAC 값은 매번 달라지므로 HTTPS 보안통신을 사용할 필요가 없다는 장점이 있다. 그러나 MAC 토큰 기법에서는 서버측에서 클라이언트 요청의 검증시마다 공유된 mac_key를 찾아와서 사용해야 하기 때문에 무상태 인증을 제공할 수 없다. MAC 토큰 기법은 2011년에 제안되었지만 널리 사용되지 못하고 아직도 Internet Draft 상태에 머물러 있다.

2.5 HTTPS 보안통신

HTTPS[6] 프로토콜은 SSL/TLS 기술을 HTTP 웹통신 프로토콜에 적용한 웹보안통신 표준 기술이다. SSL/TLS 기술을 사용하기 때문에 인증서를 이용한 서버인증, 클라이언트인증을 적용할 수 있고 초기 접속시 합의된 세션키를 이용하여 보안통신 세션을 사용할 수 있게 된다. 무엇보다도 대부분의 웹브라우저에 HTTPS 기능이 구현되어 있기 때문에 서버를 운영하는 사업자나 사용자 모두 보안통신을 쉽게 이용할 수 있다는 장점이 있다. 그러나 HTTPS 프로토콜 자체가 상태형 프로토콜이기 때문에 서버는 세션이 연결된 동시접속사용자들의 세션정보를 서버의 메모리, 디스크, 데이터베이스 등에 관리해야 하며 이것은 웹서비스의 확장성과 효율성에 큰 제약이 된다.

III. 요소기술

이 논문에서는 HTTPS 보안통신을 사용하지 않고도 초기인증과 무상태 인증유지를 안전하게 수행할 수 있도록 하기 위해 난수화 토큰인증, 서버인증서를 이용한 초기인증 보호, 인증서토큰을 이용한 전자서명 간편로그인 등의 요소기술을 결합하여 사용한다.

3.1 난수화 토큰인증

JWT 방식의 토큰인증은 서버가 발급하는 토큰을 서비스 요청시마다 서버에 반복해서 전송하는 방식으로 도청공격에 취약하므로 HTTPS와 함께 사용되어야 한다는 제약이 있다. OAuth 2.0 MAC 토큰[5] 기술은 HTTPS를 사용하지 않고도 안전한 인증유지를 제공할 수 있는 기술로 개발되었는데 서버는 사용자와의 공유된 비밀키를 관리해야 하므로 무상태 인증을 제공할 수 없다는 단점이 있다.

우리는 OAuth 2.0 MAC 토큰 기술을 수정하여 서버가 공유된 비밀키를 관리할 필요가 없이 무상태 인증을 제공할 수 있도록 개선한 난수화 토큰인증(Randomized Token Authentication) 기술을 제안하는데 이것의 초기 아이디어는 [7]에서 제시된 바 있다. 핵심 아이디어는 서버가 발급하는 kid와 mac_key를 랜덤하게 생성하는 것이 아니라 서버만이 생성할 수 있는 특수한 관계를 가지도록 생성하는 것이다. MAC 토큰 방식에서는 mac_key를 랜덤하게 생성하기 때문에 서버도 이것을 관리해야 하는 부담이 발생한다. 제안된 방식에서는 JWT 형식의 두 개의 연관된 토큰(공개토큰, 비밀토큰)을 발급하는데 공개토큰은 kid의 역할로 사용하고 비밀토큰은 mac_key의 역할로 사용한다. 공개토큰이 주어지면 서버는 언제든지 비밀토큰을 계산할 수 있다는 특성을 가져야 한다. 자세한 프로토콜은 다음과 같다.

먼저 서버가 HMAC 기반의 토큰 발급에 사용하는 비밀키를 K라고 하자. 서버의 비밀키 K는 토큰 발급시 및 토큰 검증시 서버에 의해서만 사용되고 외부로는 전혀 노출될 필요가 없으므로 서버는 이것을 안전하게 관리할 수 있다. 편의상 토큰발급 과정과

토큰을 이용하여 서버에 인증유지를 요청하는 난수화 토큰인증 과정으로 나누어 설명한다.

3.1.1 토큰 발급

사용자의 정보를 A라 하자. 사용자가 서버에 초기인증에 성공하면 서버는 추후 인증된 사용자 정보를 쉽게 확인할 수 있도록 사용자 정보 I_A를 생성하고 사용자에게 공개토큰(public token)과 비밀토큰(secret token)의 두 개의 토큰을 발급한다. 먼저 I_A에 대한 서버의 HMAC 서명 t_p를 생성하고 이를 포함하는 JWT 토큰을 공개토큰 T_p로 생성한다. 이후 T_p에 대한 서버의 HMAC 서명 t_s를 생성하고 이를 포함한 JWT 토큰을 비밀토큰 T_s로 생성한다. 여기서 JWT 토큰을 발급한다는 것은 Fig. 1에 나타낸 바와 같이 필요한 정보를 Header와 Payload에 넣고 HMAC 서명을 Signature에 넣은 토큰을 생성하는 것이다.

$$t_p = HMAC(I_A, K)$$
$$t_s = HMAC(T_p, K)$$

공개토큰 T_p는 인증된 사용자임을 나타내기 위해 서버에 반복해서 전송하는 정보로서 kid(서명된 ID)와 같은 역할을 하며 비밀토큰 T_s는 외부로 노출되지 않고 난수화 인증정보 계산에만 사용하는 정보로서 mac_key(서명된 패스워드)와 같은 역할을 한다. 두 개의 토큰은 상호 연관되어 있어서 비밀키 K를 알고 있는 서버는 공개토큰이 주어지면 언제든지 비밀토큰을 계산할 수 있다는 특징이 있다. 서버는 클라이언트가 제시하는 공개토큰으로부터 언제든지 비밀토큰을 생성하여 사용할 수 있기 때문에 비밀토큰을 관리할 필요가 없게 된다.

클라이언트는 서버가 발급해준 두 개의 토큰을 브라우저의 저장소에 저장하고 사용한다. 브라우저에는 쿠키스토리지, 세션스토리지, 로컬스토리지 등의 저장소가 있는데 해당 서버와의 통신에 세션에 관계없이 오랜기간 저장하고 반복 사용할 수 있도록 하기 위해 로컬스토리지에 저장한다.

3.1.2 난수화 토큰인증

클라이언트가 서버에 초기인증 후 발급받은 공개토큰 T_p와 비밀토큰 T_s가 로컬스토리지에 저장되어 있다고 하자. 클라이언트는 서버에 보호된 페이지에 대한 서비스를 요청하려고 한다. 클라이언트는 현재시간 $time_c$와 자신의 비밀토큰 T_s를 이용하여 해시함수 계산을 통해 다음의 난수화 인증정보 $auth$를 계산한다.

$$auth = H(time_c, T_s)$$

클라이언트는 서버에 서비스를 요청하면서 자신이 이미 로그인된 상태임을 알리기 위하여 $< T_p, time_c, auth >$를 전송한다.

서버는 수신한 서비스 요청 정보를 검증하기 위해 다음과 같은 검증과정을 거친다.

1. 클라이언트의 공개토큰 T_p를 검증하여 자신이 발급한 토큰인지 확인하고 사용자 정보를 확인한다.
2. 서버의 비밀키 K를 사용하여 공개토큰 T_p로부터 비밀토큰 T_s을 계산한다.
3. 비밀토큰을 이용하여 $auth$를 동일하게 계산하여 일치하는지 확인한다.
4. 서버의 현재시간 $time_s$를 체크하고 클라이언트가 보내온 시간 $time_c$와의 차이 $T_{diff} = time_s - time_c$를 계산하여 시간이 정해진 오차범위 내에서 일치하는지 확인한다.

이러한 검증을 통과하면 서버는 클라이언트의 인증상태를 인정하고 서비스를 제공하게 된다. 시간차이를 검증하는 것은 도청공격자에 의한 재전송 공격을 방지하기 위한 것이다. 클라이언트가 전송하는 인증정보인 $auth$는 현재시간에 의존하는 정보이므로 매 서비스 요청시마다 달라지는 값이다. 이것은 도청공격자에 의해 도청되더라도 재사용될 수 없으므로 평문통신으로 전송하여도 무방하다.

공개토큰 T_p는 매 서비스 요청시마다 반복 전송되는 값이므로 사용자를 나타내는 서명된 ID와 같은 역할을 한다. 비밀토큰 T_s는 사용자 클라이언트와

서버가 공유하는 비밀값으로 인증값 계산에 사용하는 패스워드와 같은 역할을 하는데 서버는 공개토큰으로부터 언제든지 비밀토큰을 다시 계산할 수 있으므로 특별한 관리가 필요없다는 장점이 있다. 그러므로 서버는 사용자의 세션정보를 관리할 필요 없이 사용자의 요청정보를 언제든지 즉시 검증할 수 있는 무상태 인증유지가 가능하다.

3.2 서버인증서를 이용한 초기인증 보호

인증유지에 사용하는 토큰은 초기인증에 성공한 클라이언트에게 발급하게 되는데, 초기인증시 사용자의 인증정보를 서버에게 안전하게 전달하고 또 서버가 발급하는 토큰을 클라이언트에 안전하게 전달하기 위해서는 보안통신이 필요하다. 이러한 목적을 위해 일반적으로는 HTTPS 기술을 사용하지만 이것은 세션 기반의 상태형 보안통신 기술이므로 본 연구에서는 HTTPS의 사용을 배제하려고 한다.

여기에서는 초기인증시의 보안통신을 위해 서버는 초기 접속하는 사용자에게 자신의 인증서를 배포하고 이를 이용하여 사용자의 초기인증 정보를 보호한다. 사용자 브라우저는 획득한 서버인증서를 로컬스토리지에 저장한다. 초기인증을 위해 아이디, 패스워드 등 사용자 인증정보를 서버에 전송시 서버의 인증서에 포함된 공개키를 이용하여 암호화 전송한다.

3.3 인증서토큰을 이용한 전자서명 간편로그인

사용자가 초기인증을 수행하는 경우 중요한 인증정보를 서버에 전달하여야 하기 때문에 보안통신채널을 사용하여야 한다. 예를 들어 패스워드를 이용한 인증에서는 패스워드 자체를 서버에 전달하게 되는데 네트워크 공격자로부터 패스워드를 보호하기 위해서는 보안통신을 사용해야 한다. 또한 사용자는 매번 패스워드를 기억하고 입력해야 하는 불편이 있다.

보안통신을 사용하지 않고도 초기인증을 안전하고 편리하게 수행하기 위한 방법으로서 서버는 초기인증된 사용자에게 인증서토큰을 발급하며 이후에는 전자서명을 이용한 간편로그인을 이용하도록 할 수 있다. 이러한 전자서명 간편로그인은 보안통신채널로 보호되지 않아도 되며 서버는 사용자가 제시하는 인증서토큰을 이용하여 즉시 신분을 확인하고 서명을 검증할 수 있어서 무상태 인증이 가능하다는 장점이 있다.

인증서토큰은 서버의 개인키로 서명하여 발급하는 사설인증서로 해당 서버에 인증시에만 사용하는 특수 인증서이다. 인증서토큰은 로컬스토리지에 저장하여 해당 서버와의 로그인에 오랜 기간 반복 사용될 수 있도록 한다.

초기인증된 사용자 클라이언트에게 서버가 공개토큰과 비밀토큰을 발급하는 경우 이들을 안전하게 전달하여야 하는데 이러한 인증서토큰의 공개키를 이용하여 암호화하여 전달한다.

IV. 평문통신 환경의 무상태형 웹인증 구현

4.1 웹인증 시스템 설계

HTTPS 보안통신에 의존하지 않는 안전한 웹인증 시스템을 구현하기 위한 아이디어는 다음과 같다. 여기에서는 초기인증으로 패스워드를 사용하는 방식의 사례를 보인다.

1) 서버인증서 배포

서버는 서버인증서를 구비하고 처음 접속하는 사용자들에게 서버인증서를 배포한다. 사용자 브라우저는 서버인증서를 검증하고 로컬스토리지에 저장한다. 서버인증서는 클라이언트의 신뢰를 얻기 위하여 인증기관으로부터 발급받은 인증서를 사용할 수도 있는데 여기에서는 편의상 서버가 자체 발행한 자체인증서를 사용하였다.

2) 서버인증서를 이용한 초기인증 패스워드 보호

사용자가 처음 서버에 접속하는 경우 패스워드 기반의 초기인증을 수행해야 하는데 패스워드를 서버에게 안전하게 전달해야 한다. 이를 위해 패스워드 정보를 서버인증서에 포함된 서버의 공개키로 암호화하여 전달한다.

3) 초기인증된 클라이언트에게 인증서토큰 발급

패스워드 등의 방식으로 초기인증에 성공하면 서버는 클라이언트에게 인증서토큰을 발급한다. 인증서토큰 발급을 위해 클라이언트는 초기인증시 키쌍을 생성하고 패스워드와 함께 공개키를 서버에 전송하여야 한다. 클라이언트의 키쌍 소유를 증명하고 재전송 공격을 방지하기 위하여 개인키로 시간기반 난수화 서명된 값을 함께 전송한다. 서버가 발급하는 인증서토큰은 서버인증서로 서명된 사설인증서로 해당 서버

와의 통신에서만 사용한다. 사용자가 여러 컴퓨터로 동일한 서비스를 사용시 클라이언트마다 별도의 인증서토큰을 발급할 수 있다.

4) 인증서토큰을 이용한 전자서명 간편로그인

인증서토큰이 발급되면 클라이언트는 서버에 초기 인증시 패스워드를 전송할 필요 없이 전자서명을 이용하여 로그인할 수 있다. 이를 위해 클라이언트는 현재시간이 포함된 인증요청정보를 개인키로 서명한 서명값을 계산하고 인증서토큰과 함께 서버에 전송한다. 서버는 전송받은 인증서토큰을 검증하여 자신이 발급한 유효한 토큰인지 확인하고 여기에 포함된 공개키를 이용하여 사용자의 전자서명을 확인하여 유효한 경우 로그인을 허용한다. 전자서명 간편로그인은 평문통신채널을 통해 수행할 수 있다.

5) 난수화 토큰인증을 이용한 인증유지

클라이언트가 초기인증에 성공하면 서버는 공개토큰과 비밀토큰을 발급하며 이들은 인증서토큰의 공개키로 암호화하여 전달된다. 클라이언트는 이들을 로컬스토리지에 저장하고 사용한다. 클라이언트는 서버에게 보호된 페이지에 대한 서비스를 요청하고자 하는 경우 앞에서 설명한 난수화 인증정보 $auth$를 계산하고 $< T_p, time_c, auth >$를 서버에 전송하며 서버는 $auth$를 검증하여 유효한 경우 서비스를 제공한다.

6) 난수화 토큰인증을 이용한 암호화, 인증 통신

지금까지의 모든 인증과정이 HTTPS 필요 없이 평문통신채널을 사용하여도 안전하게 수행 가능한데, 실제로는 메시지 자체의 보안을 위해 암호화된 통신을 수행할 필요가 있을 수도 있다. 이 경우에는 난수화 인증정보 $auth$를 비밀 세션키로 이용하여 선택된 메시지에 대해 암호화 통신, 인증 통신을 수행할 수 있다.

4.2 웹인증 시스템 구현

위에서 제시된 설계 내용을 실제 웹서비스로 구현하였으며 구현 환경은 다음의 Table 1과 같다.

구현된 서버는 이 논문에서 제시된 기술을 검증하기 위해 특별하게 제작된 서비스이며 현재 다음의 주소에서 운영중이다.

Table 1. Implementation Environment

OS	Ubuntu 16.04
Server-side Framework	node.js, express
Database	MongoDB
Client-side Framework	Angular6
Crypto libraries	node-forge, bcrypt

http://isweb.joongbu.ac.kr:3000/

이하 서버의 동작을 기준으로 구현사례를 설명한다.

1) 초기화면 접속시 서버인증서 배포

사용자가 초기화면에 처음 접속하면 서버는 서버인증서를 클라이언트에 배포하고 이것은 클라이언트의 로컬스토리지에 자동 저장된다. 로컬스토리지의 정보는 크롬브라우저를 기준으로 할 때 F12키를 누르면 확인할 수 있다. 서버는 초기화면 하단에 다음 Fig. 3과 같이 서버인증서 정보와 서명된 일회용 인증정보를 표시하여 사용자가 서버의 신원을 확인할 수 있게 한다. 일회용 인증정보는 현재시간과 난수를 서버가 서명한 값이며 사용자가 페이지를 다시 로드하면 새로운 값으로 갱신된다.

Fig. 3. Server certification information in initial load of main page

2) 패스워드를 이용한 초기인증

사용자가 패스워드를 입력하고 초기인증하는 페이지는 HTTPS를 사용하지 않지만 실제로는 사용자의 패스워드가 서버의 인증서로 암호화하여 전달되므로 네트워크 공격자에게 노출되지 않는다. 서버가 사용자를 인증하게 되면 인증서토큰과 공개토큰, 비밀토큰이 발급되는데 이들은 인증서토큰의 공개키로 암호화되어 클라이언트로 전달되며 로컬스토리지에 자동 저장된다. Fig. 4는 로그인에 성공한 이후의 로컬스

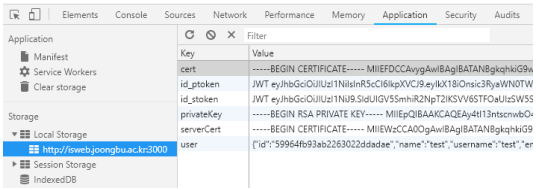

Fig. 4. Local storage information after successful login

토리지 정보를 표시하고 있는데 여기에 저장되는 정보들은 다음과 같다.

- cert(사용자의 인증서토큰)
- privateKey(사용자의 개인키)
- serverCert(서버인증서)
- id_ptoken(공개토큰)
- id_stoken(비밀토큰)
- user(사용자정보)

패스워드를 이용한 초기인증 단계에서는 인증서토큰 발급을 위하여 클라이언트는 키쌍을 생성하고 개인키는 브라우저에 저장하고 공개키를 서버에게 전송한다. 인증서토큰 발급 신청시 클라이언트는 개인키 소유증명과 재전송공격 방지를 위하여 시간기반 난수화 서명값을 계산하고 함께 전송한다. 인증서토큰은 X.509 방식의 인증서 형태로 발급하였으며 유효기간은 1년으로 설정하였다.

서버가 발급하는 공개토큰, 비밀토큰은 개인의 인증유지 상태를 증명하는데 사용되는 기밀정보이므로 서버가 클라이언트에게 안전하게 전달해야 하는데 이를 위해 클라이언트의 인증서토큰에 포함된 공개키를 이용하여 암호화하여 전달한다. 공개토큰, 비밀토큰은 1주일의 유효기간을 가지도록 발급하였으며 발급 이후 1주일 동안은 초기인증을 요구받지 않고 서버에의 인증상태를 유지할 수 있다. 토큰의 유효기간이 지나면 서버는 사용자에게 다시 초기인증을 요구하게 된다.

3) 인증서토큰을 이용한 전자서명 간편로그인

사용자가 한번의 패스워드 초기인증을 통해 인증서토큰을 발급받으면 사용자는 패스워드를 기억하고 입력할 필요없이 전자서명을 이용하여 간편하게 로그인할 수 있다. 클라이언트는 사용자 ID와 현재시간 정보를 인증서토큰으로 서명하고 이것을 인증서토큰과 함께 서버에 전송한다. 서버는 인증서토큰을 검증하고 사용자 정보를 확인할 수 있으며 전자서명을 검

Fig. 5. Simple signature-based login without password

증하여 사용자의 진위여부를 검증한다.

이러한 과정은 평문채널을 통해서도 안전하게 수행될 수 있는데 전자서명은 현재시간을 포함하여 시간기반으로 난수화된 서명이므로 공격자가 도청하더라도 재사용할 수 없기 때문이다.

4) 난수화 토큰인증을 이용한 인증유지

프로필(RTA) 페이지에서는 Fig. 6에서와 같이 난수화 토큰인증을 이용하는 인증유지 서비스의 사례를 보여준다. 이 페이지에 접속하기 위하여 클라이언트가 서버에 보내는 정보는 공개토큰, 현재시간, 난수화 인증정보 $< T_p, time_c, auth >$ 이다. 서버는 사용자의 공개토큰으로부터 사용자 정보를 확인한다. 공개토큰으로부터 비밀토큰을 계산하고 인증정보의 유효성과 현재시간의 유효성을 검증한다.

이 페이지를 갱신하면 현재시간이 바뀜에 따라 인증정보가 매번 바뀌는 것을 볼 수 있다. 이러한 인증정보는 도청공격자에 의해 도청되어도 재사용할 수 없기 때문에 인증유지 서비스는 평문채널을 통해서도 안전하게 운영할 수 있다. 이러한 인증유지 검증은 사용자가 해당 페이지를 요청할때마다 발생하는데 서버는 검증에 필요한 비밀토큰을 사용자가 제시하는 공개토큰으로부터 즉시 계산할 수 있으므로 별도의 사용자정보를 관리할 필요가 없어서 무상태 인증유지 서비스가 가능하다.

5) 난수화 토큰인증을 이용한 선택적 암호화 통신

구현된 전체 서비스는 평문채널을 통해 안전하게 인증이 유지된 상태로 서비스를 제공한다. 그런데 특정한 기밀 메시지를 암호화 전송할 필요가 있을 경우

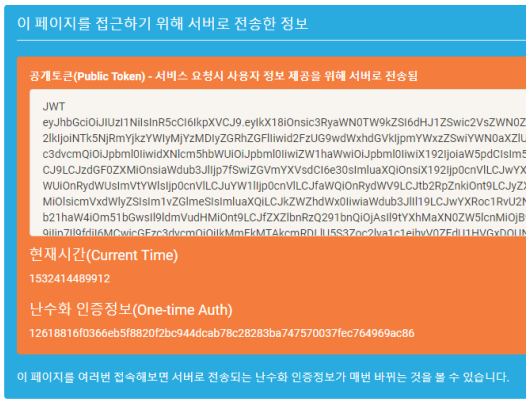

Fig. 6. Keeping authentication using randomized token authentication

에는 클라이언트와 서버가 공유하고 있는 비밀토큰을 이용하여 세션키를 생성하고 이것을 이용하여 메시지를 암호화하여 전송할 수 있으며 Fig. 7은 이런 사례를 보여준다.

사용자가 입력창에 입력하는 메시지가 생성된 일회용 비밀키를 이용하여 암호화 전송되고 서버는 클라이언트가 보내준 공개토큰과 현재시간 값으로부터 똑같은 일회용 비밀키를 생성할 수 있고 메시지를 복호화할 수 있게 된다. 일회용 비밀키는 현재시간정보를 이용하여 생성되므로 매번 달라짐을 알 수 있으며 네트워크를 도청하는 공격자는 메시지를 복구할 수

Fig. 7. Selected message encryption using randomized token authentication

없다. 이러한 기술은 전체 페이지를 암호화하는 것이 아니라 특정 HTML 태그요소만을 암호화 전달할 수 있도록 이용될 수 있다.

V. 분 석

5.1 효율성 분석

OAuth 2.0 Bearer 토큰, MAC 토큰 기법과 제안된 난수화 토큰인증(RTA) 기법의 효율성을 비교하여 Table 2에 나타내었다.

MAC 토큰인증 기법에서는 초기인증된 사용자에게 공유비밀키로 랜덤한 mac_key를 발급하므로 서버도 이것을 DB 등의 형태로 관리해야 한다. 클라이언트의 서비스 요청을 검증하는 경우 서버는 해당 클라이언트의 mac_key를 검색해서 찾아와야 하는데 이것은 상태형 인증으로서 효율성이 낮으며 동시 접속 사용자수가 많아지면 효율성이 크게 저하된다. 반면 RTA 기법에서는 비밀토큰이 mac_key와 동일한 역할을 하는데 비밀토큰은 클라이언트가 제공한 공개토큰으로부터 한번의 HMAC 연산으로 계산할 수 있다. 그러므로 무상태형 인증을 제공할 수 있어서 효율성이 높으며 동시 접속자수가 많은 대규모 웹 서비스에 적용할 경우 효율성을 크게 높일 수 있다.

초기인증 및 토큰 발급시 클라이언트는 인증정보를 서버에 안전하게 전달해야 하고 서버는 클라이언트에게 토큰을 안전하게 전달해야 한다. Bearer 및

Table 2. Comparison of performance

	Bearer token	MAC token	RTA
No. of tokens	1	1	2
server-side token management	X	Y	X
security in initial login	HTTPS	HTTPS	server certificate
security in keeping authentication	HTTPS	HTTP	HTTP
computations in auth req.	-	HMAC 1	HMAC 1
computations in auth verify	HMAC 1	HMAC 2	HMAC 2 Hash 1
stateful/less in keeping authentication	stateful	stateful	stateless

MAC 토큰 기법에서는 이를 위해 TLS 보안통신을 사용하는데 이것은 세션 기반 보안통신으로 서버와 클라이언트가 보안세션을 맺은 후에 공유된 비밀키로 암호화하여 전달하는 방식으로 보안통신 자체가 상태형 서비스를 필요로 한다. 반면 이 논문에서 제시된 RTA 인증유지의 구현사례에서는 세션을 맺지 않고 클라이언트가 서버에 인증정보 전달시 서버인증서로 암호화 전달하고 서버가 클라이언트에게 토큰을 전달시 인증서토큰에 등록된 클라이언트 공개키를 이용하여 암호화 전송하게 된다. 이것은 세션을 맺지 않는 무상태 보안통신으로 효율성이 높다.

이 논문에서 제시한 인증서토큰을 이용한 전자서명 간편로그인은 사용자가 패스워드를 기억할 필요가 없이 초기인증을 할 수 있게 하므로 사용자에게 높은 편의성을 제공해준다. 서버 입장에서도 사용자 DB를 검색할 필요 없이 클라이언트가 제시하는 인증서토큰을 이용하여 즉시 검증할 수 있으므로 무상태 초기인증을 제공할 수 있다.

초기인증은 매우 간헐적으로 발생하는 이벤트이고 인증유지는 서비스 이용기간동안 지속적으로 발생하는 이벤트이다. 제안된 난수화 토큰인증 기법을 이용하면 웹서비스는 평문통신채널을 이용하면서도 무상태형 인증유지 서비스를 안전하게 제공할 수 있다. 많은 동시접속 사용자를 대상으로 서비스를 제공해야 하는 대규모 웹서비스는 동일한 서버장비를 이용하여 더 많은 사용자에게 서비스를 제공할 수 있게 된다.

5.2 안전성 분석

1) 키관리, 토큰관리의 안전성

MAC 토큰 기법에서는 mac_key를 랜덤하게 생성하므로 서버도 이것을 DB 등에 저장해야 한다. 만일 공격자가 서버를 해킹하여 mac_key를 획득하게 되면 해당 사용자의 인증을 탈취할 수 있게 된다. 반면 RTA 기법에서는 비밀토큰을 서버에 저장하지 않으므로 이러한 해킹공격에 노출되지 않는다.

2) DOS 공격

공격자가 고의로 잘못된 접속 요청을 하는 경우 MAC 토큰에서는 제공된 kid로부터 mac_key를 찾아와서 검증해야 하므로 많은 DB 쿼리가 발생하게 된다.

RTA 기법의 경우 클라이언트의 인증요청을 즉시 계산하여 검증할 수 있으므로 잘못된 접속 요청을 빠르게 확인하고 서비스를 거부할 수 있어서 DOS 공격에 덜 치명적이다. 잘못된 요청이 발생하면 서버는 인증유지를 중지하고 초기인증을 다시 요구하게 될 것이다.

3) 중간자 공격

공격자가 중간자 공격으로 통신도청 및 변조 공격을 통해 사용자의 인증정보, 토큰의 탈취를 시도할 수 있을 것이다. 본 본문의 구현사례에서는 서버인증서, 클라이언트의 인증서토큰을 이용하여 서버와 클라이언트 사이의 통신을 보호하므로 이러한 공격을 방지할 수 있다.

현재의 상용 브라우저에는 이런 기능이 제공되지 않기 때문에 이 논문에서는 별도의 자바스크립트를 이용해 구현하였는데 이러한 기능이 HTTPS에서와 같이 브라우저의 자체 기능으로 지원된다면 더 안전하고 편리하게 사용할 수 있을 것으로 기대된다.

4) 서버의 비밀키에 대한 오프라인 공격

서버의 비밀키는 외부에 노출되지 않고 토큰의 발급 및 검증에만 사용하는 비밀정보이다. 그런데 공격자는 서버로부터 반복해서 토큰을 발급받을 수 있고 비밀토큰은 공개토큰으로부터 서버의 비밀키를 이용하여 HMAC 계산한 값이므로 공격자가 〈공개토큰, 비밀토큰〉의 데이터를 수집하여 서버의 비밀키를 찾으려고 시도할 수 있다. 이것의 안전성은 사용한 HMAC 알고리즘의 안전성에 의존하게 되며 이러한 공격은 bearer 토큰을 사용하는 모든 서비스에 공통적으로 해당되는 위협이다.

5) 클라이언트 측 토큰의 탈취 시도

클라이언트가 발급받은 토큰은 브라우저의 로컬스토리지에 저장되므로 공격자가 정상적인 사용자의 컴퓨터를 해킹공격하여 발급받은 토큰을 탈취하려고 시도할 수 있다. 발급받은 토큰을 탈취하게 되면 공격자는 해당 사용자로 로그인할 수 있게 된다. 이것은 치명적인 위협이지만 사용자의 컴퓨터에 정보를 저장하여 사용하는 모든 인증유지 기법에 모두 해당되는 취약성이다. 이런 위협에 대응하기 위해 사용자로부터 입력받은 패스워드를 이용하여 토큰을 암호화하여 로컬스토리지에 저장하는 기법이 제안된 바 있는데 [8] 사용자가 브라우저 사용시 별도의 패스워드를 입력해야 하므로 불편함이 있을 것이다.

VI. 결 론

OAuth 2.0 Bearer 토큰의 HTTPS 보안통신 요구에 따른 비효율성, MAC 토큰의 상태형 인증

특징으로 인한 비효율성을 극복하기 위하여 무상태 인증이 가능한 난수화 토큰인증 프로토콜을 제시하고 그 구현 사례를 제시하였다. 이것은 평문통신 환경에서도 안전하게 MAC 인증을 수행할 수 있게 해주며 HTTP 웹 통신프로토콜이 추구하는 무상태 서비스를 구현할 수 있게 한다.

초기인증과 토큰 발급 과정에서는 서버인증서와 인증서토큰를 이용한 무상태형 보안통신을 적용함으로써 HTTPS와 같은 상태형 보안통신 적용의 부담을 덜고자 하였다. 현재의 상용 브라우저에는 이런 기능이 제공되지 않기 때문에 이 논문에서는 별도의 자바스크립트를 이용해 복잡하게 구현하였는데 앞으로 이런 기능이 표준기술로 채택되고 브라우저의 자체기능으로 제공된다면 더 안전하고 편리하게 사용할 수 있을 것이다. 인증서토큰을 이용하면 사용자가 패스워드를 기억하고 입력할 필요 없이 전자서명 기반의 안전하고 편리한 초기인증을 수행할 수 있다.

이 논문에서 제시된 기술은 OAuth 2.0 MAC 토큰인증 기법의 효율성을 획기적으로 높이고 실용화할 수 있는 기술로 표준화를 위해 노력할 필요가 있으며 표준기술로 채택될 경우 대형 웹서비스들의 효율성을 높이고 비용을 줄이는데 큰 역할을 할 수 있을 것으로 기대된다.

References

[1] Dick Hardt, "The OAuth 2.0 authorization framework," RFC 6749, Oct. 2012.

[2] Michael B. Jones and Dick Hardt, "The OAuth 2.0 authorization framework: bearer token usage," RFC 6750, Oct. 2012.

[3] Michael B. Jones, John Bradley, and Nat Sakimura, "JSON web token (JWT)," RFC 7519, May 2015.

[4] JWT, https://jwt.io/

[5] Justin Richer, William Mills, Hannes Tschofenig, and Phil Hunt, "OAuth 2.0 message authentication code (MAC) tokens," Internet-Draft, Jan. 15, 2014. https://tools.ietf.org/id/draft-ietf-oauth-v2-http-mac-05.html

[6] Eric Rescorla, "HTTP over TLS," RFC 2818, May 2000.

[7] Byoungcheon Lee, "Strengthening of token authentication using time-based randomization," Journal of Security Engineering, vol. 14, no. 2, pp. 103-114, 2017.

[8] Hee Won Myeong, Jung Ha Paik, Dong Hoon Lee, "Study on implementation of secure HTML5 local storage," Journal of Korean Society for Internet Information, vol. 13, no. 4, pp. 83-93, 2012.

〈저자소개〉

이 병 천 (Byoungcheon Lee) 종신회원
1986년 2월: 서울대학교 물리학과 졸업
1988년 2월: 서울대학교 물리학과 석사
2002년 2월: KAIST 정보보호 박사
2002년 3월~현재: 중부대학교 정보보호학과 교수
〈관심분야〉 정보보호, 암호, 인증, 네트워크보안, 웹보안, IoT보안