보안공학연구논문지 *Journal of Security Engineering*
Vol.14, No.2 (2017), pp.103-114
http://dx.doi.org/10.14257/jse.2017.04.02

시간기반 난수화를 이용한 토큰인증의 강화

이병천[1)]

Strengthening of Token Authentication Using Time-based Randomization

Byoungcheon Lee[1)]

요 약

패스워드 기반 인증[1]은 사용자의 기억에 의존하는 인증방식으로 시스템 구현이 간단하여 컴퓨터 산업의 태동시부터 널리 사용되어져 왔다. 토큰 인증[2][3]은 서버가 발급한 특정 토큰을 소유하고 있다는 것을 보임으로써 사용자를 인증하는 방식으로 오늘날의 대규모 웹서비스 환경에서 서버의 부담을 줄여줄 수 있다는 효용성이 있어서 자동로그인 방식으로 널리 사용되고 있다. 그러나 이러한 인증 방식은 고정된 패스워드나 토큰이 통신로를 통해 평문으로 반복 전달되기 때문에 도청공격 등으로 노출될 수 있는 위험이 있어서 반드시 암호화된 통신 환경에서 사용되어야 한다. 고정된 패스워드 인증의 취약성을 해결하기 위해 TOTP(Time-Based One-Time Password)[4] 방식이 사용될 수 있는데 이것은 매 로그인시마다 공유되고 있는 고정된 패스워드와 현재의 시간정보를 입력으로 하여 난수화된 로그인 정보를 생성하여 제출하게 하는 것으로 도청공격에 대응할 수 있게 하였다.

이 논문에서는 TOTP의 난수화 방식을 일부 응용하여 현재 널리 사용되고 있는 웹서비스에서의 고정된 토큰 인증을 시간정보를 이용하여 난수화할 수 있는 방법을 제시한다. 이를 이용하면 토큰인증 정보가 시간에 따라 달라져서 도청공격을 무력화시킬 수 있다. 이러한 방법론은 기존에 사용되던 JWT[3] 토큰 인증 시스템을 약간 수정하여 난수화된 자동로그인을 제공할 수 있다는 장점이 있다.

핵심어 : 패스워드 인증, 토큰 인증, 난수화, 자동로그인, 도청공격, TOTP

Abstract

Password-based authentication[1] depends on human's memory and is simple to implement that it had been widely used in the real world from the beginning of computer industry. Token-based authentication[2][3] is an authentication of user by showing that the user possesses a specific token that was issued by the server, and it is widely used to implement auto login in current massive web service environment since it can greatly reduce server's workload. But in these authentication protocols fixed information is sent to the server repeatedly over the communication channel that it is subject to eavesdropping attack, thus these authentication protocols have to be executed over a secure communication channel such as SSL/TLS. To solve these eavesdropping attack TOTP(Time-based One Time Password)[4] protocol can be used which produces randomized authentication information depending on shared password and current time.

In this paper we propose a new randomized token-based authentication protocol by using TOTP approach to modified token-based authentication. This protocol can thwart eavesdropping attack since the authentication information changes depending on time. This approach can provide randomized token-based authentication with simple modification of traditional JWT authentication.

Keywords : Password-based authentication, Token-based authentication, Randomization, Auto login, Eavesdropping, TOTP

Received(February 27, 2017), Review request(February 28, 2017), Review Result(1st: March 15, 2017)

Accepted(April 03, 2017), Published(April 30, 2017)

[1]Dept. of Information Security, Joongbu Univ., 305, Dongheon-ro, Goyang-si, Gyeonggi-do, 10279, Korea
email: sultan@joongbu.ac.kr

ISSN: 1738-7531 JSE

1. 서론

패스워드 기반 인증[1]은 사용자의 기억에 의존하는 인증방식으로 시스템 구현이 간단하여 컴퓨터 산업의 태동시부터 널리 이용되어 왔으며 인증의 중요성이 강조되고 있는 오늘날까지도 컴퓨터 시스템의 가장 기본적인 인증방식으로 여전히 이용되고 있다. 이 방식은 컴퓨터와 사용자(소유자)가 동일공간에 있는 경우 사용자의 접근권한을 확인하기 위한 로컬인증 뿐만 아니라 원격지 서버에 통신망을 통해 접속하는 사용자의 신분을 확인하기 위한 원격인증에도 사용되고 있다. 고정된 기밀정보에 의존하는 패스워드 인증은 사전공격(dictionary attack)에 취약하므로 추측하기 힘든 안전한 패스워드를 설정하여 사용해야 하는데 패스워드는 사용자의 기억에 의존하므로 복잡한 패스워드를 사용하기 어렵다는 근본적인 한계가 있다. 또한 안전한 패스워드를 사용하더라도 도청공격(eavesdropping attack)에 노출될 수 있으므로 반드시 HTTPS, SSL/TLS, SSH 등의 암호화된 통신채널을 이용해야 한다.

패스워드 인증 방식은 사용자가 매 접속시마다 패스워드를 입력해야 하고 서버는 사용자 계정DB에서 패스워드 정보를 읽어와서 비교해야 하는데, 오늘날의 모바일 환경에서는 패스워드 입력이 불편하다는 단점이 있고, 구글, 페이스북 등과 같이 사용자 수가 많아 고도의 분산된 서버환경에서 운영해야 하는 대규모 웹서비스 환경에서는 사용자 로그인시마다 서버측에서는 사용자의 계정DB에서 패스워드 정보를 읽어와 비교해봐야 하는 부담이 있다. 이런 이유로 사용자가 한번 패스워드로 로그인을 하면, 서버는 서명된 토큰을 발급하고, 클라이언트는 브라우저의 로컬스토리지에 토큰을 저장하게 되며, 사용자의 다음 접속 요청시마다 토큰을 자동 첨부하게 하고, 서버는 첨부된 토큰을 확인하여 해당 사용자로 인증해주는 토큰인증이 널리 사용되고 있다[2][3]. 이런 토큰인증은 서버가 사용자의 로그인시마다 사용자 계정DB를 확인할 필요가 없이 사용자가 제시하는 토큰만으로 인증을 수행하기 때문에 확장성에 큰 장점이 있어서 현재 대규모 웹서비스에서 자동로그인 방식으로 널리 사용되고 있다. 예를 들면 구글, 페이스북 등의 대규모 웹서비스에서는 처음 로그인시에만 패스워드 인증을 요구하고 토큰을 발급하며 토큰의 유효기간 동안에는 패스워드 인증을 요구하지 않고도 로그인 상태를 유지하는 자동로그인 방식을 사용하고 있다.

그런데 패스워드 인증 및 토큰인증 방식은 고정된 인증정보를 반복 전송하는 방식이므로 도청공격에 취약성이 있어서 공격자가 패스워드 또는 토큰을 획득하게 되면 사용자의 신분으로 손쉽게 위장할 수 있게 된다. 그러므로 이러한 인증 과정은 반드시 HTTPS, SSL/TLS, SSH 등의 암호화된 통신채널을 사용해야 한다. 패스워드 인증의 이러한 취약성을 해결하기 위한 방안으로 TOTP[4] 방식이 사용될 수 있는데 이것은 클라이언트와 서버간에 공유되고 있는 고정된 패스워드와 현재 시간정보를 함께 사용하여 인증정보를 계산하여 제출하는 방식으로 로그인시마다 인증정보가 달라지게 되어 도청공격이 쓸모없게 된다.

이 논문에서는 현재 널리 사용되고 있는 고정된 토큰을 반복 사용하는 토큰인증의 도청공격에

대한 취약점을 해결하기 위하여 TOTP의 방법론을 일부 적용하여 시간에 따라 인증정보가 변경되는 개선된 토큰인증 프로토콜을 제안하고 있다. 구체적으로는 사용자가 패스워드로 로그인하게 되면 서버가 두 개의 연관된 토큰(공개토큰, 비밀토큰)을 발행하고 사용자가 클라이언트에 이들 두 개의 토큰을 저장하되, 자동로그인시에는 두 개의 토큰과 현재 시간정보를 함께 이용하는 해쉬값을 계산하고 공개토큰과 함께 전송하는 방법을 사용한다. 서버는 사용자가 제출한 공개토큰 정보를 이용하여 사용자의 신분을 확인할 수 있고 비밀토큰을 쉽게 계산할 수 있으며 자동로그인의 해쉬값을 검증할 수 있다. 네트워크를 도청하고 있는 공격자는 공개토큰은 획득 가능하지만 비밀토큰을 획득할 수 없으므로 사용자 신분으로 자동로그인을 위장할 수 없게 된다.

2. 기존 인증 기술

2.1 패스워드 인증

사용자의 클라이언트 컴퓨터와 원격지의 서버 사이에 패스워드 기반의 인증을 사용한다고 가정하자. 현재 유닉스, 리눅스 등의 운영체제, 기타 웹서비스 구현에 널리 사용되고 있는 기술은 다음과 같다.

1) 등록단계: 사용자 등록단계에서 사용자는 자신의 신분을 증명할 수 있는 정보를 제공하고 사용할 아이디(ID)와 패스워드(pass)를 서버에 전송한다. 서버는 솔트(salt) 값을 난수로 생성하고 해쉬함수의 반복횟수 I를 선정하여 패스워드의 암호화 해쉬값을 계산하는데 표준 유닉스 서버에서는 crypt()함수를, 웹서비스에서는 pbkdf2[5], bcrypt[6] 등의 함수를 이용한다. 서버는 사용자 계정DB에 아이디와 패스워드의 암호화 해쉬값을 저장한다.
2) 로그인단계: 사용자는 로그인 입력창에 ID와 pass를 입력하고 이것을 서버에 전송한다. 서버는 사용자가 전송한 pass 값과 사용자 계정DB에 저장된 암호화 해쉬값이 일치하는지 확인하여 정보가 일치하는 경우 로그인을 허용한다.

등록단계 및 로그인단계 모두 사용자의 패스워드(pass)는 평문으로 서버에 전송되는데 이것은 네트워크 통신을 도청하고 있는 공격자에게 쉽게 노출되는 위험성이 있다. 그러므로 이러한 패스워드 기반의 인증 프로토콜은 HTTPS, SSL/TLS, SSH 등의 암호화 통신채널을 이용하여 수행되어야 한다. 또한 단순한 암호화 통신채널만을 이용하면 중간자 공격(man-in-the-middle, MITM)에 취약할 수 있으므로 엄밀하게 상호 신분확인을 거치도록 구현되어야 한다.

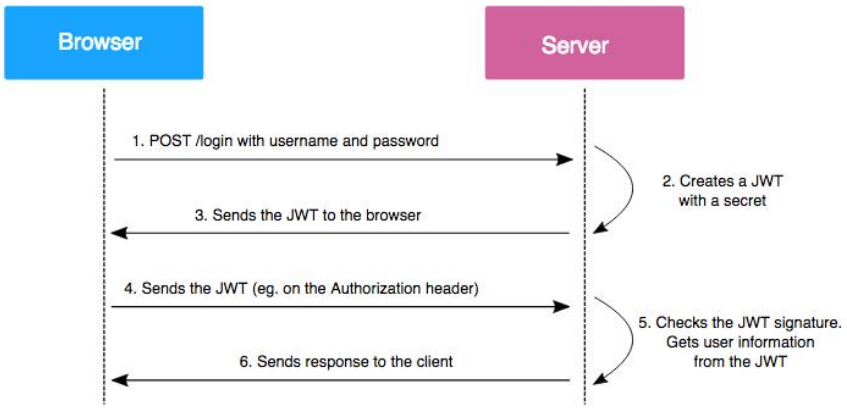
2.2 토큰 인증

전통적으로 패스워드 인증은 세션인증과 함께 사용되어 왔다. 즉, 서버에 한번 패스워드로 로그인을 하면 같은 세션 내에서는 더 이상 패스워드 인증을 요구하지 않고 서비스를 계속 사용할 수

있도록 하는 것으로 사용자 편의성 제공을 위해서는 반드시 필요하다. 이러한 세션인증 서비스를 제공하기 위해서는 서버에 사용자별 로그인 세션에 대한 정보를 저장하고 매 페이지 요청시마다 서버가 세션정보를 확인하는 방식을 사용한다. 그런데 사용자 수가 많아지고 대규모 분산 서비스를 필요로 하는 환경에서는 세션정보를 서버에 저장하고 관리하는 것이 부담이 되어 확장성에 문제가 있다. 또한 새로운 세션으로 접속하는 경우에는 여전히 패스워드 로그인이 필요하다. 이러한 문제점을 해결하기 위해 사용되는 방식이 토큰 인증[2][3]이다.

토큰 인증은 위의 패스워드 인증과 연결하여 사용자가 한번 성공적으로 로그인하여 신분이 확인되면 서버가 서버만이 보유하고 있는 비밀정보를 이용하여 서명된 토큰을 생성하여 이를 사용자에게 전송하고 사용자는 브라우저의 로컬스토리지에 토큰을 저장한다. 사용자 브라우저가 다음 페이지 요청시에는 저장된 토큰을 자동 첨부하게 되고 서버는 첨부된 토큰으로부터 사용자 정보를 획득할 수 있으며 토큰이 유효한지 확인하여 사용자의 신분을 확인할 수 있는데, 이 방법은 서버가 사용자 계정DB를 참조하지 않고도 전송되어온 토큰만으로 인증이 가능하다는 장점이 있다. 이러한 토큰 인증을 이용하면 한번만 패스워드 로그인을 하면 토큰의 유효기간 내에서는 항상 로그인되어 있는 것처럼 서비스를 이용할 수 있는 자동로그인 기능을 제공할 수 있게 된다.

다음의 [그림 1]은 JSON Web Token(JWT)[3] 방식에서의 로그인 및 토큰 발급/활용 과정을 보여주고 있다. 사용자가 브라우저를 이용하여 서버에 패스워드 로그인하면, 서버가 토큰을 발급하고, 이것이 사용자 브라우저에 저장되며, 브라우저가 다음 페이지 요청시 토큰을 첨부하면, 서버가 토큰의 서명을 검증하여 로그인을 승인하는 다음과 같은 과정을 보여준다.

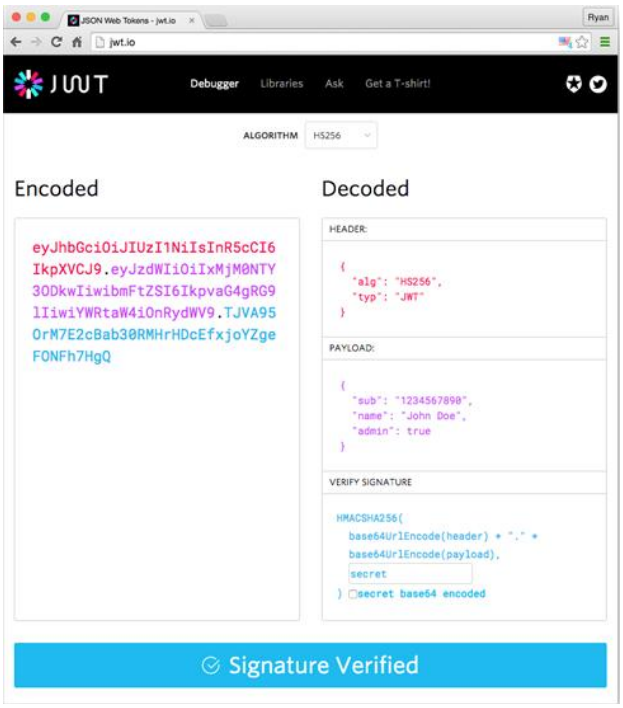


[그림 1] JWT의 발급 및 활용
[Fig. 1] Issuing and usage of JWT

1) 클라이언트는 서버에게 username, password를 전송하여 로그인
2) 서버는 로그인된 사용자에게 JWT를 생성

3) 서버는 JWT를 사용자에게 전송하고 클라이언트는 JWT를 브라우저의 로컬스토리지에 저장
4) 새로운 페이지 요청시 인증헤더에 JWT를 첨부하여 전송
5) 서버는 JWT의 서명을 검증하고 사용자 정보를 획득하여 자동로그인 처리
6) 서버는 클라이언트에게 요청 페이지를 전송

[그림 2]는 jwt.io 사이트에서 설명하는 JWT 토큰의 구조를 나타낸 것이다. 토큰은 서버가 생성하는 것으로 [헤더].[페이로드].[서명]의 세부분으로 구성되며 각각 base64 인코딩된 스트링이 . 으로 연결된 스트링이다. 여기에서 서명은 헤더와 페이로드 정보를 서버가 안전하게 보유하고 있는 비밀값 secret과 함께 HMACSHA256으로 인증코드를 생성한 것이다. 서명을 검증하기 위해서는 서버의 비밀값 secret이 필요하다. 그러므로 JWT 토큰은 서버만이 생성 가능하고 해당 서버만이 검증할 수 있어서 서버가 사용자의 신분을 인증하기 위해 사용된다. 토큰의 헤더와 페이로드는 base64 인코딩된 스트링이므로 base64 디코딩을 통해 정보를 읽어볼 수 있다. 통신을 도청하는 공격자는 토큰을 base64 디코딩하여 헤더와 페이로드에 포함된 정보를 읽어볼 수는 있으나 서명을 위조할 수는 없다.



[그림 2] JWT 토큰의 구조
[Fig. 2] Structure of JWT token

JWT 토큰 방식의 한 가지 단점은 토큰 자체를 도청하여 획득하고 재전송하면 타인이 사용자의 신분을 쉽게 위장할 수 있다는 것이다. 그러므로 JWT 토큰을 이용한 인증프로토콜은 HTTPS,

SSL/TLS, SSH 등 암호화된 채널을 통해 전송되어야 하는데 토큰이 모든 페이지 요청에 첨부되므로 사용자가 서버로 전송하는 모든 요청이 암호화되어야 한다는 것을 의미한다. 그러므로 현재 구글, 페이스북 등의 대규모 웹서비스들은 전체 통신채널을 암호화된 HTTPS로 운영하고 있는 실정이다. 아울러 브라우저에 저장되는 토큰은 클라이언트 해킹에 의해 누출될 수 있으므로 안전한 장소에 저장되어야 한다. 브라우저 프로그램에서 토큰 저장에 이용하는 로컬스토리지는 접속 사이트별로 독립된 샌드박스 모델로 운영되므로 악성코드 등의 타 프로그램에 의해 누출되기는 어렵지만 공격자가 사용자의 PC를 직접 손에 넣은 것과 같은 환경으로 해킹한다면 브라우저로부터 해당 토큰을 쉽게 복사해낼 수 있어서 사용자의 패스워드가 브라우저에 저장된 것과 같은 위험성을 가지고 있다. 그러므로 사용자는 토큰의 안전한 저장과 관리, 시스템의 접근제어 관리에 많은 신경을 써야 한다.

한편 토큰인증에 암호화된 통신채널을 사용한다고 해도 공격자가 중간자공격(Man-in-the-middle, MITM)으로 세션을 가로채서 정보를 획득할 수 있는 위험성이 있다. 예를 들면 공격자가 SSL 기능을 제공하는 프록시 프로그램을 이용하여 사용자 및 서버와 별개의 SSL 세션을 맺어 암호화 통신을 중개하고 사용자 로그인 정보를 획득할 수 있는데[7], 이를 방지하기 위해서는 클라이언트가 서버의 인증서를 엄밀하게 검증해야 한다.

2.3 TOTP

고정된 패스워드를 반복 사용하는 패스워드 인증은 클라이언트가 서버에 전송하는 값이 항상 일정하므로 공격자가 이것을 도청하여 획득하고 재전송하면 사용자의 신분을 쉽게 위장할 수 있다는 문제가 있다. 이런 문제를 해결하기 위해 제시된 것이 시간 기반 일회용 패스워드(Time-based One Time password)[4] 방식으로 클라이언트와 서버가 동일한 비밀키 K를 공유하고 있는 경우 현재의 시간정보와 함께 이용하여 난수화된 일회용 패스워드를 생성하고 이를 이용하여 인증하도록 할 수 있다. 일회용 패스워드는 다음과 같이 계산된다.

$$TOTP = HOTP(K, T)$$
$$T = floor(\frac{T_{curr} - T_0}{X})$$

여기서 X는 시간간격을 초단위로 나타낸 것이며 기본값은 30초이다. T_0는 유닉스 시간의 시점을, T_{curr}는 현재시간을 나타낸다. 그러므로 T는 현재 시간을 유닉스 시간단위로 나타낸 정수를 시간간격 X로 나누어 floor 함수를 처리한 정수를 나타낸다. K는 클라이언트와 서버가 공유하고 있는 비밀키를 나타낸다. HOTP 함수는 HMAC를 이용하여 K와 T를 입력으로 인증코드를 계산하는 함수이다.

이 방식에서는 클라이언트와 서버가 동일한 비밀키 K를 공유하고 있어서 현재시간 T를 이용하

여 동일한 TOTP 값을 계산할 수 있으므로 인증 가능하다. TOTP값은 시간간격 X가 지날때마다 새로운 값으로 바뀌게 되므로 통신을 도청하고 있는 공격자는 현재의 인증코드를 도청하여 획득하더라도 시간간격 X가 지나면 더이상 사용자의 신분을 위장할 수 없게 된다.

이러한 TOTP 방식은 클라이언트와 서버가 동일한 비밀정보를 공유하고 있는 환경에서 사용할 수 있다. 그러나 패스워드 인증 방식에서는 사용자가 입력하는 비밀번호(pass)가 서버에서는 솔트와 반복횟수 등의 추가정보를 이용하여 암호화 해쉬값으로 변경되어 저장되며 비밀번호 자체는 저장되지 않는다. 이것은 서버의 사용자 계정DB가 공격자에게 노출될 경우에도 사용자의 비밀번호 정보를 보호하기 위한 조치이다. 그러므로 클라이언트와 서버는 동일한 정보를 공유하고 있는 상황이 아니어서 TOTP 방식이 현재의 패스워드 인증에 직접 적용되지는 않고 있다.

3. 토큰인증의 난수화

본 연구의 목적은 JWT의 고정된 토큰을 사용하는 인증 방식이 도청공격에 취약하므로 이것을 TOTP에서와 같이 시간에 따라 인증정보가 난수화되는 방식으로 개선하고자 하는 것이다. 그런데 토큰 인증 방식에서는 서버가 사용자 계정DB를 검색하지 않고도 클라이언트가 서버에게 전송해주는 토큰을 이용하여 사용자 정보를 확인할 수 있어야 한다. 그런데 이를 위해 토큰을 그대로 전송하면 공격자가 토큰을 획득하여 재전송함으로써 사용자로 쉽게 위장할 수 있게 되는 상호 모순적인 상황에 직면하게 된다. 이 문제를 해결하기 위해 여기에서는 서버가 사용자에게 상호 연관성을 가진 두 개의 토큰(공개토큰, 비밀토큰)을 발급해주는 방식을 고안하였다. 공개토큰은 서버가 사용자 정보를 쉽게 확인할 수 있도록 서버에게 전송하는 토큰이며 비밀토큰은 인증정보를 계산하는데는 사용하지만 서버에게는 전송하지 않는 토큰을 말한다. 서버는 공개토큰으로부터 비밀토큰을 계산할 수 있어야 인증정보를 검증할 수 있다. 자세한 로그인 및 토큰 발급/활용 프로토콜은 다음과 같다.

3.1 사용자 등록

사용자는 브라우저에 ID, pass를 입력하고 이것을 서버에 전송하며 사용자 등록을 요청한다. 서버는 패스워드 해쉬값 $hpass = pbkdf2(pass, salt, I)$ 를 계산한다. 여기서 pbkdf2[5]는 패스워드기반 키생성함수이며 사용자의 입력 패스워드 pass와 난수로 선택하는 salt, 반복횟수 I를 입력으로 패스워드의 암호화된 해쉬값을 계산한다. 서버는 사용자 계정 DB에 <ID, hpass>를 저장한다.

3.2 사용자 로그인

사용자는 브라우저의 로그인창에 ID, pass를 입력하고 이것을 서버에 전송한다. 서버는 사용자

계정 DB에서 ID, hpass를 읽어오고 사용자가 전송한 pass가 일치하는지 확인한다. 사용자의 로그인이 확인되면 서버는 다음과 같이 토큰을 발급한다.

JWT 토큰은 [H].[P].[S]와 같이 헤더 H, 페이로드 P, 서명 S의 3부분으로 구성되는데 여기에서는 토큰의 서명을 t로 표시하고 이것의 생성과정을 함수 HMAC을 이용하여 편의상 다음과 같이 표기하기로 하자.

$$t = HMAC(H, P, \mathrm{secret})$$

여기서 함수 HMAC은 서명생성함수이며 공개가능한 H와 P를 입력으로 받아 공개되지 않는 서버의 비밀값 secret을 이용하여 토큰의 서명 t를 생성한다. 사용자 로그인이 확인되면 서버는 다음과 같이 연관된 두 개의 토큰(공개토큰, 비밀토큰)을 생성하여 브라우저에 전송하며 브라우저는 이들을 브라우저의 로컬스토리지에 저장한다.

$t_0 = HMAC(H, P_0, \mathrm{secret}), P_0 = [ID, M, T_s, 0]$: 공개토큰
$t_1 = HMAC(H, P_1, \mathrm{secret}), P_1 = [ID, M, T_s, 1]$: 비밀토큰

여기서 payload에 포함되는 정보는 ID(사용자의 ID), M(사용자의 기기정보), T_s(토큰의 유효기간) 등이며 마지막에 포함된 0/1은 단순한 플래그 정보이다. 즉 두 개의 토큰은 플래그 이외에 동일한 정보를 기반으로 생성된 것이다. 여기서 사용자의 기기정보 M을 추가한 것은 사용자가 여러 대의 기기들을 사용할 경우 서로 다른 토큰을 생성하여 이용하도록 하기 위한 구분정보이다. 두 개의 연관토큰을 생성하는 방식은 이 이외에도 여러 가지 변형된 방식이 사용될 수 있는데 서버가 제공된 공개토큰 정보로부터 비밀토큰을 자동으로 쉽게 계산할 수 있는 방식이면 된다.

3.3 자동로그인

사용자가 로그인된 상태에서 다른 페이지를 요청할 경우 브라우저는 두 개의 토큰 t_0, t_1과 현재시간 T을 이용하여 임시토큰 t_T를 다음과 같이 계산한다.

$$t_T = HMAC(t_0, t_1, T),$$
$$T = floor(\frac{T_{curr} - T_0}{X})$$

그리고 페이지 요청 패킷에 $< t_0, t_T, T >$를 덧붙여 서버에 전송한다. 그러면 서버는 전송된 공개토큰 t_0의 서명을 검증하고 페이로드를 디코딩하여 사용자의 정보를 확인하고 이를 이용하여 비밀토큰 t_1을 계산한다. 이후 임시토큰 t_T를 같은 수식을 이용하여 계산하고 사용자로부터 전송된 값과 같은지, T가 현재시간인지 확인하여 사용자의 유효성을 검증한다. 이러한 과정은 브라우저에 의해서 자동적으로 수행되므로 사용자는 자동로그인되어 있는 것처럼 통신을 이용할 수 있다. 이

러한 자동로그인 과정은 암호화되지 않은 일반 통신채널을 통해서 수행될 수 있다.

3.4 임시토큰을 이용한 메시지인증 및 암호화

이러한 방식으로 생성되는 임시토큰 t_T는 인증된 사용자의 브라우저와 서버 사이에 공유되는 난수화된 비밀정보로 생각할 수 있다. 비밀정보를 직접 전송함으로써 자동로그인에 이용할 수도 있지만, 이것을 전송하지 않고 메시지인증 및 암호화에 이용할 수도 있다. 예를 들어 클라이언트가 서버에게 보내는 메시지 M을 인증하기 위해서는 메시지인증코드를 $mac = HMAC(M,t_T)$와 같이 생성하여 $< t_0, T, mac, M >$을 전송할 수 있다. 이를 수신한 서버는 먼저 임시토큰 t_T을 계산한 후 mac의 유효성을 검증할 수 있다. 클라이언트가 서버에게 보내는 메시지 자체를 암호화할 필요가 있는 경우에는 임시토큰 t_T을 공유된 비밀키로 사용하여 메시지를 암호화하여 전송할 수 있고 서버는 임시토큰 t_T을 계산한 후 복호화할 수 있다. SSL/TLS를 이용한 암호화, 인증과 비교할 때 SSL/TLS방식은 클라이언트와 서버가 인증세션을 맺고 유지해야 하는 부담이 있으며 특히 서버 측면에서는 많은 사용자들과의 인증세션 정보를 관리해야 할 필요가 있다. 반면, 임시토큰을 이용한 방식은 평상시에 비암호화 통신을 하다가 필요한 경우에만 암호화, 메시지인증을 이용할 수 있고 서버는 세션정보를 유지할 필요가 없는 비접속형(stateless) 프로토콜로서의 장점이 있다.

4. 분석

사용자의 브라우저에는 서버가 생성하여 제공한 공개토큰 t_0와 비밀토큰 t_1이 저장되어 있으며 현재시간이 반영된 임시토큰 t_T을 계산할 수 있다. 서버는 사용자로부터 전송된 공개토큰 t_0를 이용하여 사용자의 정보를 확인할 수 있고 자신의 서명을 확인할 수 있다. 서버는 같은 사용자 정보를 이용하여 비밀토큰 t_1을 자동으로 계산할 수 있고 이것을 이용하여 임시토큰 t_T를 똑같은 수식을 이용하여 계산할 수 있게 된다.

비밀토큰 t_1은 서버의 비밀값 secret가 있어야 계산할 수 있으므로 공격자들은 공개토큰 t_0을 도청하더라도 이로부터 비밀토큰 t_1을 계산할 수 없고 임시토큰 t_T 값을 계산할 수 없다. 공격자가 현재의 임시토큰을 도청하여 획득하더라도 이것은 시간간격 X가 지나면 새로운 값으로 바뀌므로 사용자 신분으로의 자동로그인에 사용할 수 없게 되고 새로운 패스워드 로그인을 요청받게 될 것이다.

현재 대부분의 웹서비스들이 토큰을 이용한 자동로그인을 안전하게 사용하기 위해 암호화된 통신채널을 사용하고 있다. 이 논문에서 제시된 방법론을 이용하면 토큰인증의 인증정보가 시간에 따라 계속 바뀌게 되므로 토큰의 노출에 대한 위험성이 없어져서 암호화된 통신채널을 사용하지 않고도 자동로그인을 안전하게 사용할 수 있게 된다.

비밀토큰 t_1이 공격자에게 노출된다면 공격자가 사용자 신분으로 자동로그인을 위장할 수 있게 된다. 비밀토큰은 브라우저의 로컬스토리지에 안전하게 저장되어 네트워크 공격자에 대해서는 안전하다고 볼 수 있지만 사용자의 클라이언트 컴퓨터 자체가 공격당하여 공격자가 직접 조작할 수 있는 수준이 되면 비밀토큰을 복사해낼 수 있게 된다. 그러므로 사용자는 클라이언트 컴퓨터를 안전하게 관리하기 위한 노력을 기울여야 한다.

효율성 측면에서 비교해볼 때 HTTPS 방식의 암호화된 통신채널을 이용하는 것은 모든 패킷의 정보를 암복호화해야 한다는 계산량 측면의 부담과 함께 서버가 접속된 사용자별 SSL 세션을 계속 유지해야 한다는 세션관리 측면의 부담을 가지게 된다. 반면 제안된 자동로그인 방식은 토큰의 계산과 검증이라는 계산량을 부담해야 하지만 서버 측면에서는 SSL 세션을 관리할 필요가 없어서 HTTP의 비접속형(stateless) 프로토콜의 장점을 그대로 살릴 수 있다.

5. 결론

이 논문에서는 패스워드 인증과 JWT 토큰을 이용한 자동로그인이 고정된 인증정보를 반복 전송함으로 인해 도청공격에 취약하고, 이런 문제 때문에 항상 암호화된 통신채널을 이용해야 하는 제약이 있음을 보였다. 이를 개선하여 암호화된 통신채널을 사용하지 않고도 안전한 인증을 제공하기 위하여 시간정보를 이용한 난수화된 토큰인증 프로토콜을 제시하였다. 이 방법론을 이용하면 토큰인증의 인증정보가 시간에 따라 계속 바뀌게 되므로 도청을 통해 토큰을 획득하는 것이 쓸모없어져서 암호화된 통신채널을 사용하지 않고도 자동로그인을 안전하게 사용할 수 있게 된다. 현재 대부분의 대규모 웹서비스들이 토큰을 이용한 자동로그인을 암호화된 통신채널을 통해 사용하고 있는 현실에 비추어볼 때 이 방법론은 웹서비스에서 사용자 인증의 안전성과 효율성을 보장할 수 있는 획기적인 개선책을 제시한 것으로 볼 수 있다.

한편 패스워드를 이용한 초기 로그인에서는 여전히 패스워드를 서버로 평문으로 전송하는 방식을 사용하기 때문에 암호화된 통신채널을 사용해야 한다. 현재 사용자 인증의 안전성 향상을 위해 인증서 기반의 전자서명을 이용하는 사용자 인증이 널리 사용되고 있는데, 개인키의 안전한 저장을 위해 개인키를 패스워드로 암호화하여 저장하고 있고, 해킹 위협에 대항하기 위해 보안토큰 등 하드웨어 보안모듈에 저장할 것을 요구하고 있다. 그러나 개인키를 사용하기 위해서는 여전히 패스워드를 입력해야 하는 단점이 있다. 인증의 취약성이 사용자 기억에 의존하는 패스워드에 있으므로 패스워드 의존성을 줄이는 것이 근본적인 목표가 되고 있는데 FIDO[8]에서는 안전한 하드웨어에 개인키를 저장하면서도 생체인증을 이용한 로컬인증, 전자서명을 이용한 원격인증이 안전하게 결합된 방식으로 인증이 진화하고 있다. 즉, 생체인증을 이용하여 소유기기에 대한 로컬인증에 성공하면 소유기기가 전자서명을 이용한 원격인증까지 한번에 수행되도록 하는 것이다. FIDO를 이용하면 서버에 대한 사용자 초기 인증을 패스워드에 의존하지 않고 생체인증을 이용하여 안전하게 수행할 수 있다. 그런데 초기 인증 이후의 자동로그인은 웹서비스의 효율성 보장을 위해 여전

히 필요한 기술이며 이것의 안전성을 높이는 것이 중요하다. 본 논문에서 제안한 난수화된 자동로그인 방식은 암호화된 통신채널을 사용하지 않고도 자동로그인을 안전하게 수행할 수 있기 때문에 FIDO 기술과 함께 유용하게 적용될 수 있다고 생각된다.

References

[1] L. O'Gorman, Comparing Passwords, Tokens, and Biometrics for User Authentication, Proceedings of the IEEE. **(2003)**, Vol.91, No.12, pp.2019-2040.

[2] RFC7519, JSON Web Token, https://tools.ietf.org/html/rfc7519, Feb. 27 **(2017)**.

[3] JSON Web Token, https://jwt.io/, Feb. 27 **(2017)**.

[4] RFC6238, TOTP: Time-Based One-Time Password Algorithm, https://tools.ietf.org/html/rfc6238, Feb. 27 **(2017)**.

[5] PKCS #5: Password-Based Cryptography Specification Version 2.0, https://tools.ietf.org/html/rfc2898, Feb. 27 **(2017)**.

[6] bcrypt, https://www.npmjs.com/package/bcrypt, Feb. 27 **(2017)**.

[7] Woojoong Ji, Kyungmoon Lee, Byoungcheon Le, SSL Proxy MITM Attacks Against Insecure HTTPS Authentication System, Conference on Information Security and Cryptography 2016, KIISC **(2016)**.

[8] FIDO alliance, https://fidoalliance.org/, Feb. 27 **(2017)**.