# Beveiliging van JSON Web Tokens in Web Authenticatie

Auteur: Lin Wenhao
Klas: BR2
Datum: 1 mei 2024

# Samenvatting


## Onderzoeksdoel
In dit onderzoek wil ik meer inzicht krijgen in hoe JWT (JSON Web Tokens) werkt om uiteindelijk de veiligheid van deze authenticatiemethode in het web authenticatie proces te evalueren op basis van de informatie die ik heb. Vervolgens zal ik de gebruiksscenario's samenvatten waarbij JWT wel en niet van toepassing is.

## Methodologie
Om een beter inzicht te krijgen in JWT (JSON Web Tokens), zal ik externe bronnen gebruiken en de veiligheid van deze authenticatiemethode voor web authenticatie evalueren op basis van de verzamelde informatie. De bronnen die in het onderzoek worden gebruikt, zullen worden gelabeld in de Bronnenlijst.

## Bevindingen

## Conclusies

# Inhoudsopgave

# 1. inleiding

In het vierde blok werd ik nieuwsgierig naar JSON Web Token, een authenticatietechniek die ik leerde kennen tijdens een schoolproject. Ik ga de afkorting JWT gebruiken, waar ik nieuwsgierig naar was omdat we in eerdere projecten bij het authenticeren van een gebruiker de ID van de gebruiker direct doorstuurden naar de API-server, maar met JWT gebruikten we een reeks willekeurige tekens zodat de gebruiker de authenticatie-informatie niet kon lezen. Ik was dus benieuwd hoe de API-server de JWT genereerde en parseerde, en hoe het authenticatieproces beveiligt.

Ik zal dus een hoofd vraag in dit onderzoek beantwoorden:

- Hoe beïnvloeden JWT-tokens de veiligheid van netwerkauthenticatie?

Voordat ik de hoofdvraag beantwoord, leid ik naar het antwoord op de hoofdvraag door de volgende 5 deel vragen te beantwoorden:

- Hoe worden JWT-tokens gegenereerd en opgeslagen?
- Wat is het verschil tussen JWT-token en traditionele authenticatiemethoden? (De validatiemethode die we vorig blok gebruikten)
- In welk geval is het gebruik van een JWT-token de meest geschikte keuze?
- Risico's en kwetsbaarheden in JWT-token.
- Omstandigheden waarin JWT-token niet geschikt is voor gebruik, beperkingen.

In het volgende zal ik deze vragen beantwoorden en mijn antwoorden zullen gebaseerd zijn op externe informatiebronnen. Om ervoor te zorgen dat de informatie betrouwbaar is, controleer ik waar mogelijk de betrouwbaarheid van de bronnen en beoordeel ik deze door gebruik te maken van verschillende, maar wederzijds bevestigende informatiebronnen. Ik zal ook mijn eigen begrip van en mening over deze informatie opnemen in mijn antwoorden op de analyse van specifieke vragen. Ten slotte zal ik alle informatiebronnen in een bronnenlijst opnemen.

De volgorde van mijn onderzoek zal beginnen met het verkennen van de deel vragen en eindigen met de hoofd vraag.

# 2. Generatie en Opslag van JWT

## 2.1 Generatieproces van JWT

Volgens jwt.io(z.d.), dat de structuur van JWT beschrijft, bestaat JWT uit drie delen: 'Header', 'Payload' en 'Signature'.

De header bestaat uit twee delen: het eerste deel bevat het type token en het andere deel is het gebruikte ondertekening algoritme, het specificeert het algoritme dat moet worden gebruikt om het token te genereren of te valideren.

De payload draagt daadwerkelijke informatie, ook wel 'claims' genoemd. Maar er zijn ook 3 soorten 'claims' : geregistreerd(registered), publieke(public), en privé(private) claims. Geregistreerd claims is een declaraties die bijvoorbeeld de geldigheidsperiode van het token aangeven. Publiek claims bevatten informatie die de gebruiker authentiseert, zoals een e-mailadres of gebruikersnaam. Privé claims kunnen alle informatie bevatten, bijvoorbeeld informatie over de instellingsvoorkeuren van de gebruiker.

De handtekening (signature) is het derde deel van JWT dat verantwoordelijk is voor het verifiëren dat er niet met het bericht is geknoeid tijdens de overdracht. Hij doorloopt de header en payload na Base-64 codering en genereert samen met de sleutel een hash waarde die deel uitmaakt van de handtekening.

Header, Payload en Signature worden base64 gecodeerd en het resultaat zijn drie strings die worden samengevoegd met een punt. Zo zie JWT er uiteindelijk uit: xxxxx.yyyyy.zzzzz.

## 2.2 opslag van JWT

Volgen Auth0 (z.d.) is het zo dat de JWT wordt gegenereerd op de API-server en vervolgens naar de client gestuurd, waarna de webpagina het token opslaat in de lokale opslag. De documentatie vermeldt de volgende risico's van het rechtstreeks opslaan van JWT in lokale opslag (Local Storage):

- Aanvallers kunnen lokaal opgeslagen gegevens verkrijgen via cross-site scripting (XSS).
- Omdat JWT lokaal in platte tekst wordt opgeslagen, is het token beschikbaar voor iedereen met toegang tot de browser.

Bij hoofdstuk 'Browser in-memory scenarios' wordt gesuggereerd om het token in het geheugen van de browser op te slaan, zoals in een web worker. Uit de documentatie van Mozilla.org (n.d.) blijkt dat de Worker-interface een thread op systeemniveau creëert en omdat de Worker-thread strak wordt beheerd vanuit andere threads, kunnen kwaadaardige scripts die lijken op XSS-aanvallen hun weg naar de gegevens in de Worker-thread niet vinden.

# 3. Verschillen in het verificatieproces tussen de JWT en plaintext authenticatie

## 3.1 Definitie van plaintext authenticatie

Plaintext authenticatie betekent dat gebruikersinformatie direct in het API-verzoek wordt gezet, wat ook betekent dat de gebruikersnaam of het wachtwoord van de gebruiker zichtbaar is.

## 3.2 Verificatieproces van plaintext authenticatie

Volgens de Apinizer documentatie (z.d.) stuurt de client een verzoek met een ‘plaintext’ naar een API-server, waarbij de authenticatie-informatie wordt ingevoerd in de ‘requestheader’. Bij ontvangst van het verzoek voert de API-server een authenticatieproces uit, dit stappen kan omvatten zoals het controleren of de gebruikersnaam geldig is, het wachtwoord correct is. Zodra de authenticatie is voltooid, verwerkt de API-server het verzoek en stuurt gegevens terug of voert de juiste actie uit. Als de authenticatie mislukt, stuurt de server een foutbericht terug om de client te informeren dat de toegang is geweigerd of dat er geldige authenticatiegegevens nodig zijn.

Deze methode is eenvoudig, maar er zijn risico's. Ten eerste kan iedereen de aanvraaginformatie wijzigen om toegang te krijgen tot andere gebruikers. Als gebruiker kan zijn identiteitsinformatie worden afgeluisterd en onderschept omdat deze in duidelijke tekst wordt verzonden, waardoor informatie uitlekt.

## 3.3 Verificatieproces van JWT

Na het genereren van JWT zijn er drie onderdelen, namelijk “header”, “payload” en “signature”, en wordt gescheiden door een punt (.) Scheid ze om een volledig JWT te vormen. Deze worden omgezet in een base64 URL-string en naar de client gestuurd en opgeslagen in de browser van de gebruiker. De gebruiker stuurt het verzoek met een JWT in de verzoekheader en de server parseert de JWT wanneer hij deze ontvangt. Maar hoe controleert de API de authenticiteit van de JWT?

Volgens het artikel van Koishigawa (2022) heeft het HS265-algoritme een privésleutel die niet alleen kan worden gebruikt om een JWT te genereren, maar ook om de authenticiteit van de JWT te verifiëren. Het gebruikt de “header” en “payload” van de input JWT, genereert dan een “signature” samen met de private sleutel en vergelijkt dan de gegenereerde “signature” met de handtekening van de input JWT. Als ze overeenkomen, is de JWT geldig.

## 3.4 samenvatting van verschillen

In de beveiligingsvergelijking tussen deze twee is JWT gebaseerd op ondertekende en versleutelde tokens. Het voorkomt dat anderen de tokeninformatie wijzigen tijdens de

overdracht. Plaintext authenticatie heeft geen handtekeningverificatie en kan daarom de authenticiteit van het bericht niet verifiëren. En door de transmissie van duidelijke tekst bestaat er ook een risico op het lekken van informatie.

# 4. Toepassingsgebieden van JWT

## 4.1 Verschil tussen Stateloos en Stateful

Volgens een blog (Brennenstuhl, 2023) is JWT een stateloze authenticatiemethode, wat betekent dat de server bij het uitvoeren van authenticatie de authenticiteit van een token niet hoeft te verifiëren via een netwerkverzoek bij een database. In plaats daarvan verifieert de server de authenticiteit via de handtekening van het token en wordt de informatie over de gebruiker opgeslagen in de payload van het JWT, die de server rechtstreeks uit het JWT kan halen zonder de gegevens via de database te hoeven opvragen.

Het tegenovergestelde van stateloosheid is stateful, waarbij de API-server het token en de identiteitsinformatie van de gebruiker die aan dat token is gekoppeld, opslaat in een database. Wanneer een gebruiker een verzoek verstuurt, vraagt API-server de database naar de geldigheid van het token en stuurt vervolgens de gebruikersinformatie terug die bij het token hoort.

## 4.2 Voordelen van JWT in scenario's met meerdere servers

Staatloosheid maakt het mogelijk om JWT te gebruiken in applicatiescenario's met meerdere servers. Elke server hoeft alleen de authenticiteit van de JWT te verifiëren zonder rekening te houden met de status van het token, wat betekent dat de API-servers zich allemaal onafhankelijk kunnen authentiseren zonder verzoeken naar de database te sturen voor query's, wat ook de druk op de database sterk vermindert.

# 5. Risico's en kwetsbaarheden van JWT

JWT is een stateloos token en stateloosheid zorgt voor eenvoudige en snelle informatie-uitwisseling, maar er zijn enkele problemen. Volgens een blog (Rao, 2022) is het zo, dat de JWT moet wachten tot de “expiratie” van het token is verlopen. Met andere woorden, zelfs als de gebruiker uitlogt en de JWT is verwijderd uit de lokale opslag, maakt de JWT nog steeds het mogelijk om toegang tot het systeem te krijgen.

Vanwege het vervalmechanisme van jwt worden er ook verschillende andere risicoscenario's genoemd in de blog. Stel dat een gebruiker het systeem misbruikt en jij als beheerder, wilt de rechten van die gebruiker beperken via jwt, maar dit is niet mogelijk. Gebruikers hebben nog steeds toegang tot de server met het token totdat het token verloopt.

De blog vermeldt een oplossing voor deze situatie door intrekkingstokens op te slaan in de database, die elke keer dat een gebruiker toegang krijgt tot de server worden gecontroleerd en het verzoek wordt geblokkeerd als het token bestaat in de database.

# 6. Scenario's waarop JWT niet van toepassing

## 6.1 Mogelijke oplossingen voor frequente identiteitscontroles

De beperkingen van JWT ontstaan in situaties waar frequente authenticatie nodig is. Dit komt door de stateloze aard van jwt, waarbij de server geen manier heeft om tokenpermissies in te trekken of te wijzigen.

Maar er is een manier om de tekortkomingen van JWT goed te maken. Volgens Copes (2023) is het oplossing zo, dat een verversingstoken kan worden toegevoegd tijdens het verificatieproces, en het verversingstoken kan worden gebruikt om opnieuw een nieuw toegangstoken aan te vragen bij de server nadat het toegangstoken is verlopen.

Op dit moment kan de server de verlooptijd van het toegangstoken instellen op een kortere periode, de tekst specificeert niet de exacte tijd, maar op basis van het validatiescenario, is de verlooptijd binnen een minuut.

Op deze manier kan de server wijzigingen aanbrengen aan de rechten van de gebruiker voordat het token verloopt.

## 6.2 De onderste regel van beveiliging

Stateless tokens zijn niet geschikt voor scenario's met een hoge beveiliging. Dit komt dat gebruikersinformatie in de JWT “payload” opgeslagen. En “payload” kan gestolen of geknoeid worden, hoewel JWT bij het valideren moet zoeken naar consistentie tussen de handtekening en de inhoud. Maar zodra de handtekening is geverifieerd, vertrouwt de server meestal alle informatie in de “payload”. Op dit punt is de aanvaller gelijkwaardig aan het verkrijgen van toegang tot de gebruiker en kan aanvaller informatie van de gebruiker op het systeem verkrijgen of wijzigen.

Vooral op gevoelige gebieden zoals financiën kan het uitlekken van persoonlijke gegevens van gebruikers ernstige gevolgen hebben, zoals fraude, diefstal van eigendommen, enz.

Daarom is het voor scenario's met hoge beveiligingseisen beter om bijvoorbeeld “stateful” tokens te kiezen. Deze hoeven namelijk geen gebruikersinformatie in de token te bevatten, waardoor het risico op informatielekken afneemt. En de server kan de levenscyclus van het token, toegangsrechten effectiever controleren.

# Bronnen


- Jwt.io (z.d.). Introduction to JSON Web Tokens. Geraadpleegd van JSON Web Token Introduction - jwt.io
- Auto0(z.d.). Token Storage. Geraadpleegd van Token Storage (auth0.com)
- Mozilla(z.d.). About Thread Safety. Geraadpleegd van Using Web Workers - Web APIs | MDN (mozilla.org)
- Apinizer(z.d.). Plain Text Authentication. Geraadpleegd van Plain-Text Authentication (apinizer.com)
- Koishigawa, K. (2022). How to Sign and Validate JSON Web Tokens-JWT Tutorial. Geraadpleegd van How to Sign and Validate JSON Web Tokens – JWT Tutorial (freecodecamp.org)
- Brennenstuhl, J. (2023). The Purpose of JWT: Stateless Authentication. Geraadpleegd van The Purpose of JWT: Stateless Authentication - jbspeakr.cc
- Rao, R. (2021). JSON Web Tokens (JWT) are dangerous for user sessions. Geraadpleegd van JSON Web Tokens (JWT) are Dangerous for User Sessions—Here's a Solution - Redis
- Copes, F. (2023). JWT authentication: Best practices and when to use it. Geraadpleegd van JWT authentication: Best practices and when to use it - LogRocket Blog