Willkommen bei Authentifizierung und Autorisierung in Node.js

Geschätzte benötigte Zeit: **20 Minuten**

Ziele

In diesem Lesen werden Sie in der Lage sein:

- Authentifizierung zu definieren
- Sessions-basierte, token-basierte und passwortlose Authentifizierung zu erklären
- Verschiedene Arten von Authentifizierungen zu vergleichen und gegenüberzustellen, einschließlich sessions-basierter, token-basierter und passwortloser Authentifizierung

Authentifizierung

Der Authentifizierungsprozess bestätigt die Identität eines Benutzers mithilfe von Anmeldeinformationen, indem überprüft wird, wer er vorgibt zu sein. Die Authentifizierung gewährleistet die Sicherheit einer Anwendung, indem sichergestellt wird, dass nur diejenigen mit gültigen Anmeldeinformationen auf das System zugreifen können. Die Authentifizierung liegt in der Verantwortung des Backends einer Anwendung.

Drei gängige Authentifizierungsmethoden in Node.js sind:

1. Sessions-basiert
2. Token-basiert
3. Passwortlos

Lasst uns ein wenig über jede dieser Methoden erklären und sie vergleichen.

Sitzungsbasiert

Sitzungsbasierte Authentifizierung ist die älteste Form der Authentifizierungstechnologie. Typischerweise verläuft der Ablauf einer Sitzung wie folgt.

1. Der Benutzer meldet sich mit seinen Anmeldedaten an.
2. Die Anmeldedaten werden mit den Daten in einer Datenbank überprüft. Die Datenbank ist dafür verantwortlich, welche Ressourcen basierend auf der Sitzungs-ID zugegriffen werden können.
3. Der Server erstellt eine Sitzung mit einer Sitzungs-ID, die eine eindeutige verschlüsselte Zeichenfolge ist. Die Sitzungs-ID wird in der Datenbank gespeichert.
4. Die Sitzungs-ID wird auch im Browser als Cookie gespeichert.
5. Wenn der Benutzer sich abmeldet oder eine bestimmte Zeit vergangen ist, wird die Sitzungs-ID sowohl im Browser als auch in der Datenbank gelöscht.

Unten ist ein Code-Snippet, das die sitzungsbasierte Authentifizierung in einer Express-Anwendung demonstriert:

```
const express = require('express');
const session = require('express-session');
const app = express();
// Middleware to set up session management
app.use(session({
  secret: 'secret-key',      // Replace with a strong secret key
  resave: false,             // Whether to save the session data if there were no modifications
  saveUninitialized: true,   // Whether to save new but not modified sessions
  cookie: { secure: false }  // Set to true in production with HTTPS
}));
// POST endpoint for handling login
app.post('/login', (req, res) => {
  const { username, password } = req.body;
  // Simulated user authentication (replace with actual logic)
  if (username === 'user' && password === 'password') {
    req.session.user = username;  // Store user information in session
    res.send('Logged in successfully');
  } else {
    res.send('Invalid credentials');
  }
});
// GET endpoint for accessing dashboard
app.get('/dashboard', (req, res) => {
  if (req.session.user) {
    res.send(`Welcome ${req.session.user}`);  // Display welcome message with user's name
  } else {
    res.send('Please log in first');
  }
});
// Start the server on port 3000
app.listen(3000, () => console.log('Server running on port 3000'));
```

Erklärung:

- **Express Setup**: Dieser Code richtet eine Express-Anwendung ein und konfiguriert das Sitzungsmanagement mit express-session.
- **Sitzungskonfiguration**: Die Middleware express-session wird mit einem geheimen Schlüssel (secret: 'secret-key') zur Verschlüsselung der Sitzungsdaten und anderen Optionen wie resave und saveUninitialized konfiguriert.
- **Login-Endpunkt (/login)**: Bearbeitet POST-Anfragen für die Benutzeranmeldung. Wenn der angegebene Benutzername und das Passwort übereinstimmen, wird der Benutzername (req.session.user) in der Sitzung gespeichert.
- **Dashboard-Endpunkt (/dashboard)**: Überprüft, ob der Benutzer authentifiziert ist (req.session.user existiert). Wenn authentifiziert, begrüßt es den Benutzer; andernfalls wird er aufgefordert, sich anzumelden.

Token-basiert

Token-basierte Sicherheit umfasst zwei Teile: Authentifizierung und Autorisierung. Authentifizierung ist der Prozess, bei dem Anmeldeinformationen bereitgestellt und ein Token erhalten wird, das die Anmeldeinformationen des Benutzers nachweist. Autorisierung bezieht sich auf den Prozess, bei dem dieses Token verwendet wird, damit der Ressourcenserver weiß, auf welche Ressourcen der Benutzer Zugriff haben sollte.

Token-basierte Authentifizierung

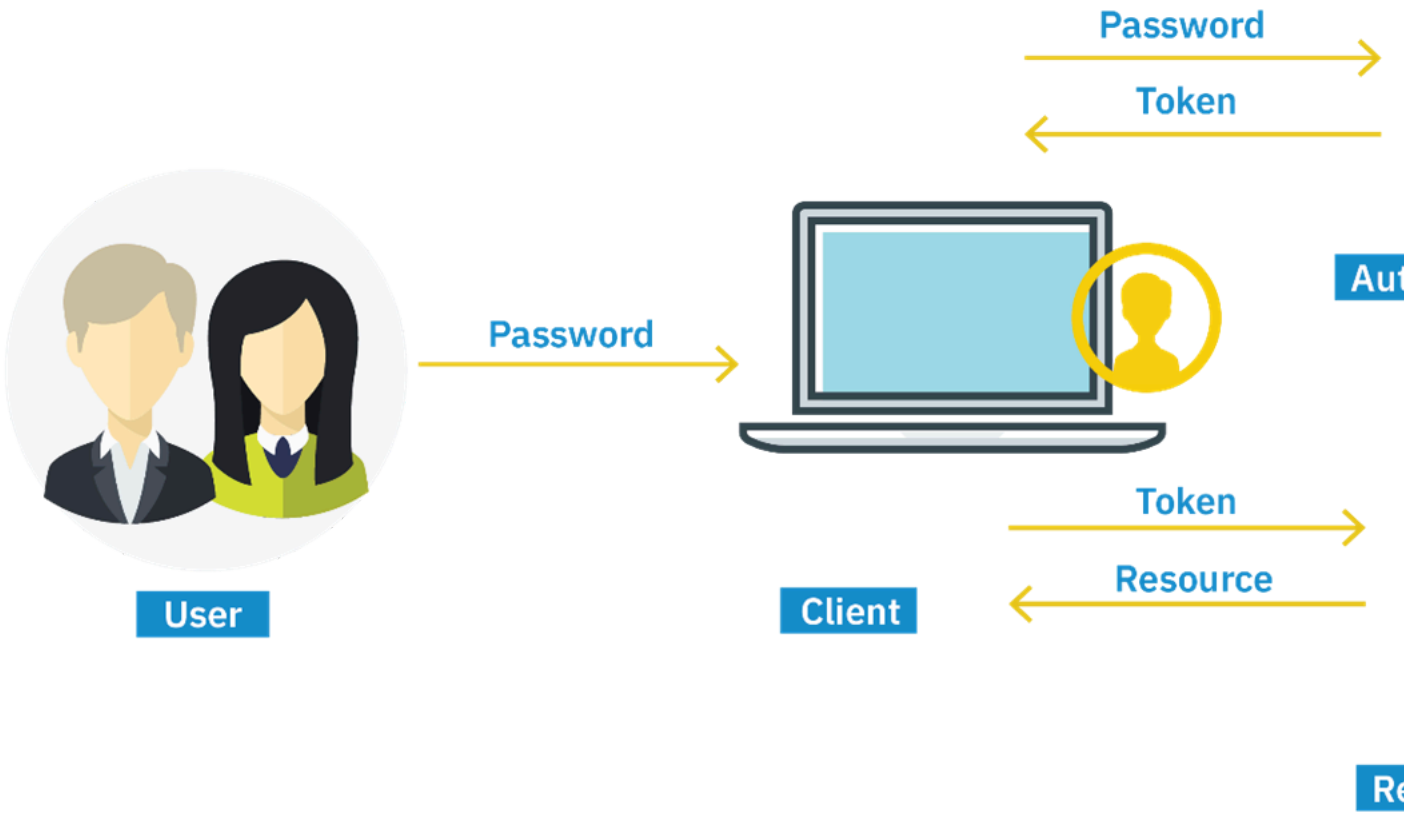
Die token-basierte Authentifizierung verwendet Zugriffstoken zur Validierung von Benutzern. Ein Zugriffstoken ist ein kleines Stück Code, das Informationen über den Benutzer, seine Berechtigungen, Gruppen und Ablauffristen enthält, die von einem Server an den Client übermittelt werden. Ein ID-Token ist ein Artefakt, das beweist, dass der Benutzer authentifiziert wurde.

Das Token besteht aus drei Teilen: dem Header, dem Payload und der Signatur. Der Header enthält Informationen über die Art des Tokens und den Algorithmus, der zu seiner Erstellung verwendet wurde. Der Payload enthält Benutzerattribute, die als Claims bezeichnet werden, wie Berechtigungen, Gruppen und Ablauffristen. Die Signatur überprüft die Integrität des Tokens, was bedeutet, dass das Token während der Übertragung nicht verändert wurde. Ein JSON-Web-Token, ausgesprochen "jot", aber als JWT geschrieben, ist ein Internetstandard zur Erstellung von verschlüsselten Payload-Daten im JSON-Format.

Der Browser eines Benutzers stellt eine Anfrage an einen Authentifizierungsserver und erhält Zugriff auf eine Webanwendung. Der Authentifizierungsserver gibt dann ein ID-Token zurück, das vom Client als verschlüsseltes Cookie gespeichert wird. Das ID-Token wird dann als Nachweis, dass der Benutzer authentifiziert wurde, an die App auf dem Webserver übermittelt.

Token-basierte Autorisierung

Dieses Flussdiagramm zeigt den Workflow eines Tokens im Autorisierungsprozess.



Der Autorisierungsprozess wird ausgeführt, wenn die Webanwendung auf eine Ressource zugreifen möchte, zum Beispiel auf eine API, die vor unbefugtem Zugriff geschützt ist. Der Benutzer authentifiziert sich beim Autorisierungsserver. Der Autorisierungsserver erstellt ein Zugriffstoken (beachten Sie, dass das ID-Token und das Zugriffstoken zwei separate Objekte sind) und sendet das Zugriffstoken zurück an den Client, wo das Zugriffstoken gespeichert wird. Wenn der Benutzer dann Anfragen oder Ressourcen stellt, wird das Token an die Ressource, auch API-Server genannt, übergeben. Das Token wird mit jeder HTTP-Anfrage übergeben. Das Token enthält eingebettete Informationen über die Berechtigungen des Benutzers, ohne dass diese Berechtigungen vom Autorisierungsserver abgerufen werden müssen. Selbst wenn das Token gestohlen wird, hat der Hacker keinen Zugriff auf die Anmeldeinformationen des Benutzers, da das Token verschlüsselt ist.

Im Folgenden finden Sie einen Codeausschnitt, der die tokenbasierte Authentifizierung in einer Express-Anwendung demonstriert:

```
const express = require('express');
const jwt = require('jsonwebtoken');
const bodyParser = require('body-parser');
const app = express();
app.use(bodyParser.json());
const secretKey = 'your-secret-key'; // Replace with a strong secret key
// POST endpoint for user login and JWT generation
app.post('/login', (req, res) => {
  const { username, password } = req.body;
  // Simulated user authentication
  if (username === 'user' && password === 'password') {
    // Generate JWT with username payload
    const token = jwt.sign({ username }, secretKey, { expiresIn: '1h' });
    res.json({ token }); // Send token as JSON response
  } else {
```

```
      res.send('Invalid credentials');
    }
  });
  // GET endpoint to access protected resource (dashboard)
  app.get('/dashboard', (req, res) => {
    // Get token from Authorization header
    const token = req.headers['authorization'];
    if (token) {
      // Verify JWT token
      jwt.verify(token, secretKey, (err, decoded) => {
        if (err) {
          res.send('Invalid token');
        } else {
          // Token is valid, send welcome message with username
          res.send(`Welcome ${decoded.username}`);
        }
      });
    } else {
      res.send('Token missing');
    }
  });
  // Start server
  app.listen(3000, () => console.log('Server running on port 3000'));
```

Erklärung:

- **Express Setup**: Konfiguriert eine Express-Anwendung mit Middleware wie body-parser, um JSON-Anfragen zu parsen.
- **JWT-Generierung (/login)**: Behandelt POST-Anfragen für die Benutzeranmeldung. Wenn die Anmeldedaten gültig sind, wird ein JWT (Token) generiert, das den Benutzernamen enthält (jwt.sign({ username }, secretKey, { expiresIn: '1h' })).
- **JWT-Überprüfung (/dashboard)**: Überprüft, ob ein JWT im Authorization-Header eingehender Anfragen vorhanden ist (const token = req.headers['authorization']). Wenn vorhanden, wird das Token überprüft (jwt.verify(token, secretKey)) und der Benutzername (decoded.username) extrahiert, um den Zugang zu gewähren.

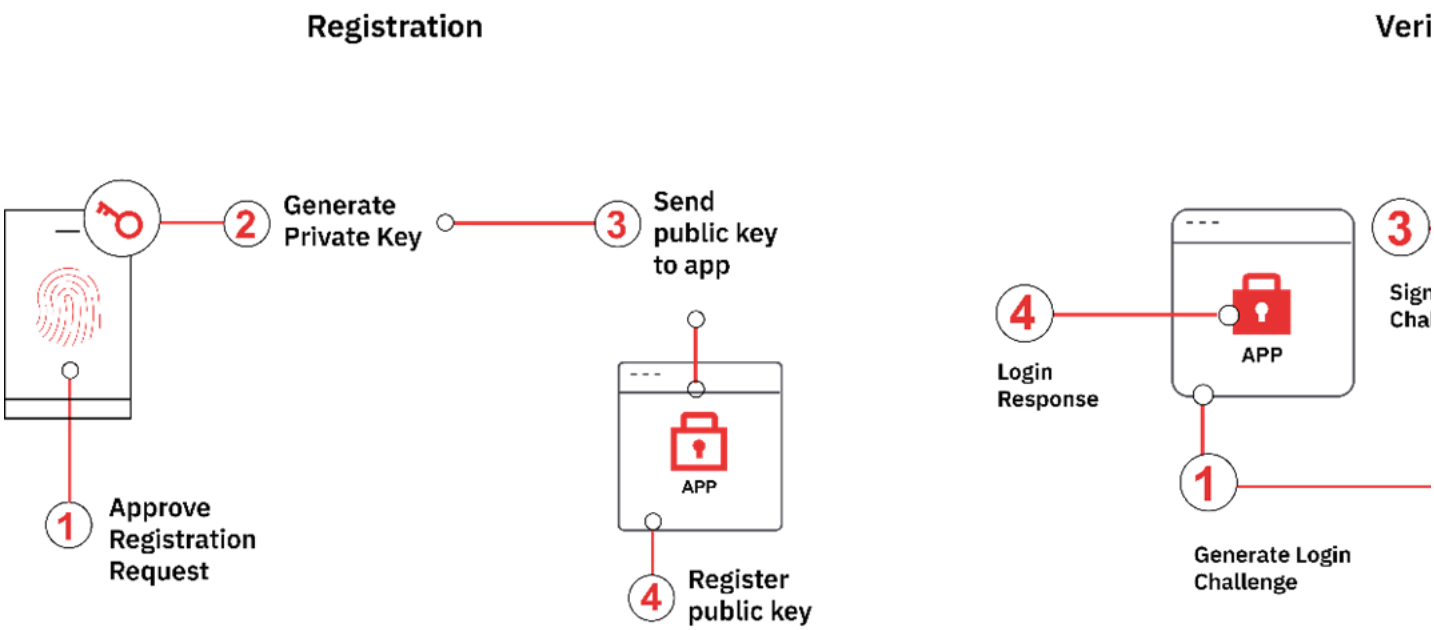
Passwortlos

Bei der passwortlosen Authentifizierung benötigt der Benutzer keine Anmeldedaten, sondern erhält Zugang zum System, indem er nachweist, dass er über einen Faktor verfügt, der seine Identität beweist. Zu den gängigen Faktoren gehören Biometrie wie ein Fingerabdruck, ein "magischer Link", der an seine E-Mail-Adresse gesendet wird, oder ein einmaliger Passcode, der an ein mobiles Gerät gesendet wird. Passwortwiederherstellungssysteme verwenden jetzt häufig passwortlose Authentifizierung.

Die passwortlose Authentifizierung wird durch die Verwendung von Public Key und Private Key Verschlüsselung erreicht. In diesem Verfahren generiert das Gerät des Benutzers bei der Registrierung für die App ein Paar aus privatem Schlüssel und öffentlichem Schlüssel, das einen Faktor nutzt, der seine Identität beweist, wie oben erwähnt.

Der öffentliche Schlüssel wird verwendet, um Nachrichten zu verschlüsseln, und der private Schlüssel wird verwendet, um sie zu entschlüsseln. Der private Schlüssel wird auf dem Gerät des Benutzers gespeichert, und der öffentliche Schlüssel wird mit der Anwendung gespeichert und bei einem Registrierungsdienst registriert.

Jeder kann auf den öffentlichen Schlüssel zugreifen, aber der private Schlüssel ist nur dem Client bekannt. Wenn der Benutzer sich in die Anwendung einloggt, generiert die Anwendung eine Anmeldeherausforderung, wie z. B. die Anforderung von Biometrie, das Senden eines "magischen Links" oder das Senden eines speziellen Codes per SMS, und verschlüsselt diese mit dem öffentlichen Schlüssel. Der private Schlüssel ermöglicht die Entschlüsselung der Nachricht. Die App überprüft dann die Anmeldeherausforderung und akzeptiert die Antwort, um den Benutzer zu autorisieren.



Im Folgenden finden Sie einen Codeausschnitt, der die passwortlose Authentifizierung in einer Express-Anwendung demonstriert:

```
const express = require('express');
const bodyParser = require('body-parser');
const nodemailer = require('nodemailer');
const app = express();
app.use(bodyParser.json());
const users = {}; // In-memory storage for demo purposes
// Endpoint to request access and send verification code via email
app.post('/request-access', (req, res) => {
  const { email } = req.body;
  // Generate a 6-digit verification code
  const code = Math.floor(100000 + Math.random() * 900000).toString();

  // Store the code in memory (users object)
  users[email] = code;
  // Simulated email sending (for demonstration)
  console.log(`Sending code ${code} to ${email}`);
  res.send('Code sent to your email');
});
// Endpoint to verify the received code
app.post('/verify-code', (req, res) => {
  const { email, code } = req.body;
  // Compare the received code with stored code for the email
  if (users[email] === code) {
    // Code matches, access granted
    res.send('Access granted');
  } else {
    // Code does not match, access denied
    res.send('Invalid code');
  }
});
// Start the Express server
app.listen(3000, () => console.log('Server running on port 3000'));
```

Erklärung:

- **Express Setup**: Richtet eine Express-Anwendung mit Middleware ein, um JSON-Anfragen (body-parser) zu parsen.
- **Anforderungszugriff (/request-access)**: Behandelt POST-Anfragen, bei denen Benutzer ihre E-Mail angeben, um Zugriff zu beantragen. Generiert einen 6-stelligen Verifizierungscode (code) und speichert ihn in einem speicherinternen Objekt (users[email] = code).
- **Code verifizieren (/verify-code)**: Behandelt POST-Anfragen, um den erhaltenen Code mit dem gespeicherten Code zu überprüfen (if (users[email] === code)). Bei Übereinstimmung wird der Zugang gewährt; andernfalls wird der Zugang verweigert.

Zusammenfassung

In diesem Text haben Sie gelernt, dass:

- Authentifizierung der Prozess ist, die Identität eines Benutzers mithilfe von Anmeldeinformationen zu bestätigen, indem überprüft wird, wer sie vorgeben zu sein.
- Sitzungsbasierte Authentifizierung verwendet Anmeldeinformationen, um eine Sitzungs-ID zu erstellen, die in einer Datenbank und im Browser des Clients gespeichert wird. Wenn der Benutzer sich abmeldet, wird die Sitzungs-ID zerstört.
- Tokenbasierte Authentifizierung verwendet Zugriffstoken, oft JWTs, die zwischen Server und Client mit den Daten, die zwischen den beiden ausgetauscht werden, übergeben werden.
- Passwortlose Authentifizierung verwendet öffentliche/private Schlüsselpaare, um Daten zu verschlüsseln und zu entschlüsseln, die zwischen Client und Server ohne die Notwendigkeit eines Passworts ausgetauscht werden.

Author(s)

Rajashree Patil
Sapthashree K S


Skills Network