2.7 API-Authentifizierung – API-Key, Bearer-Token und OAuth 2.0

APIs dürfen häufig nicht ohne Zugangskontrolle verwendet werden.

Die API muss prüfen:

- wer die Anfrage sendet
- ob der Zugang gültig ist
- welche Aktionen erlaubt sind
- auf welche Daten zugegriffen werden darf

Dafür werden unter anderem API-Keys, Bearer-Tokens und OAuth 2.0 verwendet.

Lernziele

Nach dieser Seite solltest du erklären können:

- was Authentifizierung und Autorisierung unterscheiden
- wie ein API-Key funktioniert
- was ein Bearer-Token ist
- wie OAuth 2.0 grundsätzlich arbeitet
- was Access Token, Refresh Token und Scope bedeuten
- warum API-Zugangsdaten geschützt werden müssen
- wie typische Authentifizierungsfehler erkannt werden

Authentifizierung und Autorisierung

Authentifizierung beantwortet die Frage:

> Wer greift auf das System zu?

Autorisierung beantwortet die Frage:

> Was darf dieser Benutzer oder dieses Programm tun?

Begriff	Bedeutung
Authentifizierung	Identität überprüfen

Begriff	Bedeutung
Autorisierung	erlaubte Aktionen und Zugriffe prüfen

Beispiel:

Ein n8n-Workflow besitzt ein gültiges Token und ist damit authentifiziert.

Das Token darf Benutzer lesen, aber keine Benutzer löschen.

Der Workflow ist für das Lesen autorisiert, aber nicht für das Löschen.

API-Key

Ein API-Key ist ein geheimer Schlüssel, der einem Benutzer, Programm oder Projekt zugeordnet wird.

Beispiel:

```
X-API-Key: abc123
```

Der Server prüft, ob der Schlüssel gültig ist.

Möglicher Ablauf:

```
Client
  |
  | Request mit API-Key
  v
API
  |
  v
API-Key prüfen
  |
  v
Anfrage erlauben oder ablehnen
```

API-Keys werden häufig verwendet für:

- einfache Systemintegrationen
- interne Anwendungen
- technische Dienste
- automatisierte Workflows

- begrenzte API-Zugriffe

Übertragung eines API-Keys

Ein API-Key kann abhängig von der API unterschiedlich übertragen werden.

Als Header:

```
X-API-Key: abc123
```

Als Authorization-Header:

```
Authorization: ApiKey abc123
```

Manche APIs erlauben den Schlüssel als Query-Parameter:

```
https://api.example.com/users?api_key=abc123
```

Diese Variante sollte möglichst vermieden werden, da URLs beispielsweise in Logs oder Browser-Verläufen gespeichert werden können.

> Zugangsdaten sollten bevorzugt in einem geschützten HTTP-Header übertragen werden.

Vorteile und Nachteile von API-Keys

Vorteile	Nachteile
einfach einzurichten	Schlüssel kann kopiert werden
gut für technische Dienste	häufig keine Benutzeranmeldung
leicht in Automatisierungen nutzbar	teilweise grobe Berechtigungen
geringer technischer Aufwand	muss manuell ausgetauscht werden

Ein API-Key sollte nur die tatsächlich benötigten Rechte besitzen.

Bearer-Token

Ein Bearer-Token ist ein Zugangstoken, das im Authorization-Header übertragen wird.

Beispiel:

```
Authorization: Bearer eyJhbGciOi...
```

`Bearer` bedeutet sinngemäß:

> Der Besitzer dieses Tokens darf es verwenden.

Wer das Token besitzt, kann damit normalerweise die zugehörigen Berechtigungen nutzen.

Deshalb muss es wie ein Passwort geschützt werden.

Beispiel mit Bearer-Token

Request:

```
GET /users/15


Authorization: Bearer abc123
Accept: application/json
```

Der Server prüft:

1. Ist das Token vorhanden?
2. Ist das Token gültig?
3. Ist es noch nicht abgelaufen?
4. Besitzt es den benötigten Scope?
5. Darf es Benutzerinformationen lesen?

Token und API-Key im Vergleich

Merkmal	API-Key	Bearer-Token
Zweck	Anwendung oder Projekt identifizieren	Zugriff auf geschützte Ressourcen erlauben
Ablaufzeit	häufig langfristig	häufig zeitlich begrenzt
Berechtigungen	teilweise allgemein	häufig über Scopes begrenzt
Ausstellung	meist manuell	häufig über einen Anmeldeprozess
Verwendung	technische Integrationen	Benutzer- und Anwendungszugriffe

Die genaue Umsetzung hängt von der jeweiligen API ab.

OAuth 2.0

OAuth 2.0 ist ein Verfahren zur kontrollierten Zugriffsfreigabe.

Eine Anwendung kann damit Zugriff auf ein anderes System erhalten, ohne das Passwort des Benutzers dauerhaft zu speichern.

Beispiel:

> n8n darf auf ausgewählte Funktionen von Google Workspace zugreifen.

Dabei wird festgelegt:

- welche Anwendung zugreifen darf
- welcher Benutzer oder Administrator zustimmt
- welche Berechtigungen erlaubt sind
- wie lange der Zugriff gültig ist

Beteiligte Rollen bei OAuth 2.0

Rolle	Aufgabe
Resource Owner	besitzt die Daten oder erteilt die Freigabe
Client	Anwendung, die Zugriff erhalten möchte
Authorization Server	prüft Anmeldung und erteilt Tokens
Resource Server	stellt die geschützte API bereit

Beispiel:

OAuth-Rolle	Mögliches System
Resource Owner	Benutzer oder Administrator
Client	n8n
Authorization Server	Google-Anmeldedienst
Resource Server	Google-Workspace-API

Vereinfachter OAuth-Ablauf

```
n8n fordert Zugriff an
        |
        v
Benutzer meldet sich an
        |
        v
Berechtigungen werden angezeigt
        |
        v
Benutzer oder Administrator stimmt zu
        |
        v
Authorization Server erstellt Token
        |
        v
n8n verwendet Token für API-Anfragen
```

Das Passwort wird nicht direkt an n8n übergeben.

Access Token

Ein Access Token erlaubt den Zugriff auf eine API.

Beispiel:

```
Authorization: Bearer ACCESS_TOKEN
```

Access Tokens sind häufig nur begrenzte Zeit gültig.

Mögliche Gültigkeit:

- wenige Minuten
- eine Stunde
- mehrere Stunden

Nach Ablauf wird das Token abgelehnt.

Mögliche Response:

```
Status: 401 Unauthorized
```

Refresh Token

Ein Refresh Token kann verwendet werden, um ein neues Access Token anzufordern.

Ablauf:

```
Access Token abgelaufen
        |
        v
Refresh Token verwenden
        |
        v
neues Access Token erhalten
        |
        v
API-Anfrage erneut senden
```

Refresh Tokens sind besonders schützenswert, weil sie längerfristigen Zugang ermöglichen können.

> Access Tokens werden für API-Aufrufe verwendet. Refresh Tokens können neue Access Tokens anfordern.

Scopes

Ein Scope legt fest, welche Berechtigungen ein Token besitzt.

Beispiele:

Scope	Bedeutung
`users.read`	Benutzer lesen
`users.write`	Benutzer erstellen oder ändern
`projects.read`	Projekte lesen
`messages.send`	Nachrichten senden

Ein Token mit:

```
users.read
```

darf möglicherweise Benutzer abrufen, aber nicht löschen oder verändern.

Scopes unterstützen das Least-Privilege-Prinzip.

> Ein Token sollte nur die Scopes erhalten, die für seine Aufgabe erforderlich sind.

Client ID und Client Secret

Bei OAuth erhält eine Anwendung häufig:

- Client ID
- Client Secret

Client ID

Identifiziert die Anwendung.

Client Secret

Dient als geheimer Nachweis der Anwendung.

Das Client Secret darf nicht öffentlich gespeichert werden.

Beispiel:

```
Client ID: workflow-application-123
Client Secret: ***
```

OAuth ist keine Anmeldungsmethode allein

OAuth 2.0 regelt hauptsächlich die Zugriffsfreigabe.

Für eine vollständige Benutzeranmeldung wird häufig OpenID Connect zusätzlich verwendet.

Vereinfacht:

Verfahren	Hauptaufgabe
OAuth 2.0	Zugriff auf Ressourcen erlauben
OpenID Connect	Benutzeridentität für eine Anmeldung bestätigen

Für einfache API-Grundlagen reicht zunächst:

> OAuth 2.0 vergibt begrenzte Zugriffsrechte über Tokens.

Sichere Speicherung von Zugangsdaten

API-Keys und Tokens dürfen nicht direkt in öffentlich zugänglichen Dateien stehen.

Unsicher:

```
const token = "abc123";
```

Ebenfalls unsicher:

- öffentliches Git-Repository
- Wiki-Seite
- Chatnachricht
- Screenshot
- unverschlüsselte Textdatei
- Workflow-Beschreibung

Sicherer sind:

- Credential-Speicher von n8n
- Umgebungsvariablen
- Secret-Management-Systeme
- verschlüsselte Konfigurationen
- eingeschränkte Dateiberechtigungen

Zugangsdaten in n8n

n8n besitzt eine eigene Credential-Verwaltung.

Dort können beispielsweise gespeichert werden:

- API-Keys
- Bearer-Tokens
- OAuth-Verbindungen
- Benutzername und Passwort
- Client ID und Client Secret

Der Workflow verwendet die gespeicherten Credentials, ohne dass das Geheimnis direkt in jeder Node eingetragen werden muss.

Vorteile:

- zentrale Verwaltung
- einfacherer Austausch
- geringeres Risiko sichtbarer Tokens
- wiederverwendbare Verbindungen
- bessere Trennung von Logik und Zugangsdaten

Rotation

Rotation bedeutet:

> Ein Schlüssel oder Token wird regelmäßig durch einen neuen ersetzt.

Eine Rotation ist sinnvoll:

- in regelmäßigen Abständen
- nach einem Mitarbeiterwechsel
- bei Verdacht auf Missbrauch
- nach einer versehentlichen Veröffentlichung
- nach einem Sicherheitsvorfall
- wenn ein Dienst nicht mehr verwendet wird

Ablauf:

1. neuen Schlüssel erzeugen
2. Integration auf neuen Schlüssel umstellen
3. Funktion testen
4. alten Schlüssel deaktivieren
5. Änderung dokumentieren

Widerruf

Ein Token oder API-Key sollte widerrufen werden, wenn:

- er nicht mehr benötigt wird
- er möglicherweise bekannt geworden ist
- ein Mitarbeiter das Unternehmen verlässt
- eine Anwendung abgeschaltet wird
- verdächtige Zugriffe erkannt werden

Widerruf bedeutet:

> Die Zugangsdaten werden ungültig und dürfen nicht mehr verwendet werden.

Typische Fehler

Statuscode	Mögliche Ursache
400	Authentifizierungsdaten falsch aufgebaut
401	Token fehlt, ist ungültig oder abgelaufen
403	Token ist gültig, aber Berechtigung fehlt
429	zu viele API-Anfragen
500	Fehler im Zielsystem

---

401 Unauthorized

Mögliche Ursachen:

- Authorization-Header fehlt
- `Bearer` wurde falsch geschrieben
- Token ist abgelaufen
- API-Key ist ungültig
- falsche Zugangsdaten wurden verwendet

Beispiel:

```
Authorization: abc123
```

Möglicherweise erwartet die API:

```
Authorization: Bearer abc123
```

---

403 Forbidden

Mögliche Ursachen:

- Scope fehlt
- Rolle besitzt zu wenige Rechte
- Zugriff auf die Ressource ist nicht erlaubt
- Administratorfreigabe fehlt
- Anwendung wurde für diese Funktion gesperrt

Merksatz:

> 401 bedeutet fehlende oder ungültige Authentifizierung.
> 403 bedeutet gültige Identität, aber fehlende Berechtigung.

Praxisbeispiel: n8n und SaaS-API

Ein n8n-Workflow soll eine Nachricht versenden.

Ablauf:

1. n8n lädt die gespeicherten Credentials.
2. Das Access Token wird in den Request eingefügt.
3. Die API prüft das Token.
4. Die API kontrolliert den Scope `messages.send`.
5. Die Nachricht wird versendet.
6. Die Response wird ausgewertet.
7. Erfolg oder Fehler wird protokolliert.

Request:

```
POST /messages

Authorization: Bearer abc123
Content-Type: application/json

{
  "channel": "it-support",
  "text": "Der Workflow wurde erfolgreich ausgeführt."
}
```

Least Privilege

Ein Workflow sollte nicht automatisch Administratorrechte erhalten.

Beispiel:

Ein Workflow soll nur Nachrichten senden.

Benötigter Scope:

```
messages.send
```

Unnötige Scopes:

```
users.delete
billing.manage
administrators.write
```

Je weniger Rechte ein Zugang besitzt, desto geringer ist der mögliche Schaden bei Missbrauch.

Protokollierung

Bei API-Zugriffen sollten protokolliert werden:

- Zeitpunkt
- verwendete Anwendung
- Endpoint
- ausgeführte Aktion
- Ergebnis
- Statuscode
- betroffene Ressource

Nicht vollständig protokollieren:

- Access Token
- Refresh Token
- API-Key
- Client Secret
- Passwort

Geheimnisse können maskiert werden:

```
Authorization: Bearer ***
```

Systematische Fehlersuche

Wenn die API-Authentifizierung fehlschlägt:

1. Wird das richtige Authentifizierungsverfahren verwendet?
2. Ist der Authorization-Header korrekt aufgebaut?
3. Ist der API-Key oder das Token vorhanden?
4. Ist das Token noch gültig?

5. Wurde das Token widerrufen?
6. Ist der richtige Credential-Eintrag ausgewählt?
7. Besitzt das Token die benötigten Scopes?
8. Wurde eine Administratorfreigabe erteilt?
9. Stimmen Client ID und Client Secret?
10. Welcher Statuscode wird zurückgegeben?
11. Enthält die Response eine genauere Fehlermeldung?
12. Wurde das Geheimnis möglicherweise verändert oder offengelegt?

Wichtige Begriffe

Begriff	Bedeutung
Authentifizierung	Identität überprüfen
Autorisierung	Berechtigungen überprüfen
API-Key	geheimer Schlüssel für einen API-Zugang
Bearer-Token	Zugangstoken im Authorization-Header
OAuth 2.0	Verfahren zur kontrollierten Zugriffsfreigabe
Access Token	Token für API-Anfragen
Refresh Token	fordert ein neues Access Token an
Scope	begrenzt die erlaubten Aktionen
Client ID	identifiziert eine Anwendung
Client Secret	geheimer Nachweis einer Anwendung
Rotation	Zugangsdaten regelmäßig ersetzen
Widerruf	Zugangsdaten ungültig machen
Credential	gespeicherte Zugangsinformation

Gesamtmerksatz

> API-Keys und Bearer-Tokens weisen einen API-Zugang nach. OAuth 2.0 ermöglicht einer Anwendung einen kontrollierten Zugriff über zeitlich und funktional begrenzte Tokens. Zugangsdaten müssen sicher gespeichert, auf notwendige Rechte begrenzt und bei Bedarf ausgetauscht oder widerrufen werden.

Kontrollfragen

Was ist der Unterschied zwischen Authentifizierung und Autorisierung?

Authentifizierung prüft die Identität. Autorisierung prüft die erlaubten Aktionen.

Was ist ein API-Key?

Ein geheimer Schlüssel, der einen API-Zugang identifiziert.

Was ist ein Bearer-Token?

Ein Zugangstoken, das normalerweise im Authorization-Header übertragen wird.

Was ermöglicht OAuth 2.0?

Eine Anwendung erhält begrenzten Zugriff auf geschützte Ressourcen, ohne dauerhaft das Benutzerpasswort zu speichern.

Was ist ein Access Token?

Ein zeitlich begrenztes Token für API-Anfragen.

Was ist ein Refresh Token?

Ein Token, mit dem ein neues Access Token angefordert werden kann.

Was ist ein Scope?

Eine festgelegte Berechtigung eines Tokens.

Warum müssen Bearer-Tokens besonders geschützt werden?

Weil normalerweise jeder Besitzer des Tokens dessen Berechtigungen verwenden kann.

Was bedeutet Rotation?

Ein API-Key oder Token wird durch neue Zugangsdaten ersetzt.

Was ist der Unterschied zwischen 401 und 403?

401 weist auf fehlende oder ungültige Authentifizierung hin. Bei 403 fehlt die Berechtigung für die gewünschte Aktion.

Quellen

- IETF – RFC 6749: OAuth 2.0 Authorization Framework
- IETF – RFC 6750: Bearer Token Usage

- OAuth 2.0
- OpenID Connect
- n8n-Dokumentation – Credentials
- OWASP – Secrets Management
