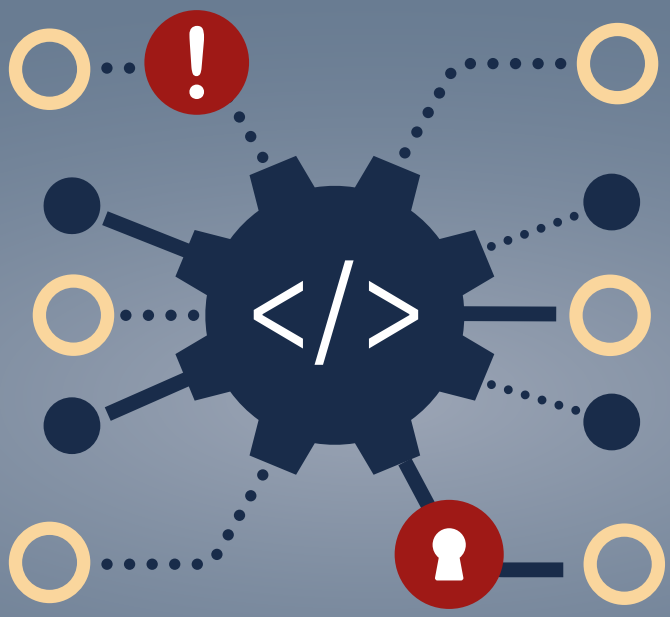
# APIs sicher entwickeln



APIs sind ähnlichen Bedrohungen wie klassische Webanwendungen ausgesetzt. Um sie gegen Schwachstellen zu schützen, orientieren sich Sicherheitsverantwortliche an den OWASP API Security Top 10.

**Von Frank Ully**

Das Open Web Application Security Project (OWASP) ist eine Non-Profit-Organisation, die sich auf die Fahnen geschrieben hat, die Sicherheit von Webanwendungen zu verbessern. Ihre wohl bekannteste Publikation sind die OWASP Top 10, eine Aufzählung der zehn kritischsten Sicherheitsrisiken in Web-Apps und deren Ursachen sowie Maßnahmen, die beim Entwickeln dagegen helfen [1].

Weniger geläufig sind die gleichfalls als Bestenliste angelegten API Security Top 10, veröffentlicht Ende 2019 von den Sicherheitsforschern Erez Yalon und Inon Shkedy. Die beiden griffen auf ihre Erfahrung als Penetrationstester zurück und werteten Statistiken von Bug-Bounty-Plattformen sowie öffentlich gewordene Sicherheitsvorfälle aus. Dieser Artikel stellt die API Security Top 10 vor – allerdings nicht in ursprünglicher Reihenfolge, sondern thematisch sortiert und um weitere Punkte ergänzt, die Pentestern im Rahmen von API-Audits bei Kunden häufig auffallen. Eine Übersicht findet sich in der Tabelle „Überblick über die OWASP API Security Top 10“.

## Schnittstellen sind überall

Warum war eine eigene Liste für API-Schwachstellen nötig? Den Analysten von Gartner zufolge werden Schnittstellen 2022 zum häufigsten Angriffsvektor auf Unternehmenswebanwendungen. Das liegt daran, dass bei Neuentwicklungen traditionelle serverbasierte Webanwendungen, geschrieben als Softwaremonolithen in klassischen Entwicklungssprachen wie PHP oder Java, auf dem Rückzug sind. Stattdessen wandert viel Anwendungslogik mit dynamischen Single-Page Applications (SPA) in den Browser, ermöglicht mit Frameworks wie React, Angular und Vue. Sie greifen auf leichtgewichtige Schnittstellen zu: mit FastAPI oder Flask unter Python geschrieben, unter serverseitigem JavaScript via Node.js mit Express oder in Golang entwickelt. Mit Spring Framework und Laravel spielen auch die Websprachendinosaurier Java und PHP mit. Frontend und Backend entkoppeln sich voneinander.

Mobile Apps auf Smartphones und Tablets greifen ebenso auf Schnittstellen zu. IoT-Geräte wie Beleuchtungssteuerungen, Alarmanlagen und Wettersensoren für das intelligente Zuhause funken über APIs mit der Hersteller-Cloud. Zunehmend beruht darauf der Informationsaustausch von Maschinen untereinander und mit Steuersystemen. SPAs und Apps fragen nicht nur die Schnittstellen des jeweiligen Anbieters oder eines Drittherstellers ab, sie greifen über ein riesiges Ökosystem an öffentlichen Schnittstellen auf Wettervorhersagen, Liedtexte und wortwörtlich Katzenbilder zu, laden über APIs Dateien hoch und lassen sie auf Schadcode prüfen oder eine Tonaufnahme als Text transkribieren. Allein das Projekt Public APIs auf der Codeplattform GitHub listet über tausend kostenfreie öffentliche Schnittstellen (siehe ix.de/zhxu). Kurz: APIs sind allgegenwärtig.

Je nachdem, welche Statistik man befragt, ist die Angriffsfläche über Schnittstellen beträchtlich und ihre Verfassung verheerend. Weil sie ein Teil vieler neu entwickelter Lösungen sind, geht die Zahl der geschätzten öffentlichen und privaten APIs in die 200 Millionen. APIs erzeugen den Großteil des Internetverkehrs, den

### iX-TRACT

- Moderne Single-Page-Webanwendungen und mobile Apps haben eine lose Kopplung zwischen Frontend und Backend und greifen im Hintergrund oft auf dieselben Schnittstellen zu.
- Durch die unterschiedliche Architektur und reifere Entwicklungswerkzeuge sind bei APIs einerseits klassische Schwachstellen wie SQL Injections weniger häufig, andererseits werden aber durch das Vertrauen in die Clientanwendungen oft zu viele Daten preisgegeben.
- Die API Security Top 10 des Open Web Application Security Project (OWASP) beschreiben zehn der kritischsten und am weitesten verbreiteten Sicherheitslücken in Schnittstellen.

## Überblick über die OWASP API Security Top 10

| | Titel | Definition des OWASP | Bereich |
|---|---|---|---|
| API1 | fehlerhafte Autorisierung auf Objektebene (Broken Object Level Authorization, BOLA) | APIs stellen Endpunkte bereit, die Objektbezeichner (Identifier, IDs) entgegennehmen. Wenn nicht bei jedem Zugriff geprüft wird, ob der Benutzer und der Client im aktuellen Kontext auf das Objekt mit der ID zugreifen darf, können Angreifer Daten anderer Benutzer einsehen. In jeder Funktion, die eine Datenquelle nutzt und eine Eingabe des Benutzers verwendet, müssen Entwickler Berechtigungen auf Objektebene prüfen. | Zugriffskontrolle |
| API2 | fehlerhafte Authentifizierung (Broken User Authentication) | Authentifizierungsmechanismen sind oft unzureichend implementiert. Dadurch können Angreifer Token kompromittieren oder Implementierungsfehler nutzen, um sich vorübergehend oder dauerhaft als ein anderer Benutzer auszugeben. Wenn die Schnittstelle Benutzer oder Clients nicht verlässlich identifizieren kann, ist ihre gesamte Sicherheit beeinträchtigt. | Zugriffskontrolle |
| API3 | Preisgabe sensibler Daten (Excessive Data Exposure) | Entwickler neigen dazu, alle Eigenschaften eines Objektes auf Endpunkten offenzulegen, ohne sensible Daten zu berücksichtigen. Wenn sie sich nur auf die API-Clients verlassen, um die Daten zu filtern, bevor sie dem Benutzer angezeigt werden, enthalten die Antworten der Schnittstelle potenziell vertrauliche Daten. | Zugriffskontrolle |
| API4 | fehlende Ressourcen- und Ratenbegrenzung (Lack of Resources & Rate Limiting) | APIs beschränken häufig nicht die Ressourcen, die ein Client oder Benutzer anfordern kann. Dies kann die Leistung des Servers bis hin zu Denial of Service (DoS) beeinträchtigen und ermöglicht Brute-Force-Angriffe etwa auf Anmeldefunktionen. | DevSecOps |
| API5 | fehlerhafte Autorisierung auf Funktionsebene (Broken Function Level Authorization, BFLA) | Komplexe Zugriffsregeln mit unterschiedlichen Hierarchien, Gruppen und Rollen sowie eine unklare Trennung zwischen administrativen und regulären Funktionen führen zu Autorisierungslücken. Angreifer erhalten Zugriff auf die Ressourcen anderer Benutzer oder administrative Funktionen. | Zugriffskontrolle |
| API6 | Massenzuweisung (Mass Assignment) | Bindet die Schnittstelle Daten, die sie vom Client im JSON-Format erhält, direkt an Datenmodelle, ohne erlaubte Eigenschaften mit einer Allowlist zu filtern, führt das zu einer Massenzuweisung. Denn Angreifer können in legitimen Anfragen nicht verwendete Eigenschaften von Objekten erraten, in anderen Endpunkten einsehen oder aus der Dokumentation entnehmen. Damit ändern sie Objekteigenschaften, auf die sie keinen Zugriff haben sollten. | Zugriffskontrolle |
| API7 | sicherheitsrelevante Fehlkonfiguration (Security Misconfiguration) | Sicherheitsrelevante Fehlkonfigurationen sind das Ergebnis von unsicheren Standardkonfigurationen, unvollständigen oder Ad-hoc-Konfigurationen, öffentlich zugreifbaren Cloud-Datenspeichern, falsch konfigurierten HTTP-Headern und unnötigen HTTP-Methoden; zudem von zu freimütigen Regeln bei Cross-Origin Resource Sharing (CORS) und ausführlichen Fehlermeldungen, die zu viele Daten preisgeben. | DevSecOps |
| API8 | Injection (Injection) | Injection-Schwachstellen wie bei SQL, NoSQL und Betriebssystembefehlen treten auf, wenn nicht vertrauenswürdige Daten als Teil einer Abfrage an einen Interpreter übergeben werden. Die bösartigen Daten des Angreifers können den Interpreter dazu bringen, vom Entwickler unbeabsichtigte Befehle auszuführen oder unberechtigt auf Daten zuzugreifen. | Validierung |
| API9 | unsachgemäße Inventarisierung (Improper Assets Management) | APIs exponieren in der Regel mehr Endpunkte als klassische Webanwendungen. Eine korrekte und aktuelle Dokumentation ist unerlässlich. Ebenso wichtig ist eine ordnungsgemäße Bestandsaufnahme der Systeme und der API-Versionen, um zu verhindern, dass auf veraltete APIs weiterhin zugegriffen werden kann oder dass Debug-Endpunkte öffentlich erreichbar sind. | DevSecOps |
| API10 | unzureichende Protokollierung und Überwachung (Insufficient Logging & Monitoring) | Unzureichende Protokollierung und Überwachung, verbunden mit fehlender oder unzureichender Integration in die Reaktion auf Sicherheitsvorfälle (Incident Response) ermöglichen es Eindringlingen, unbemerkt Systeme anzugreifen, sich einzunisten oder auf weitere Systeme auszubreiten, um Daten zu verändern, zu stehlen oder zu zerstören. Studien zeigen, dass Vorfälle oft erst nach mehreren Wochen oder Monaten entdeckt werden, häufig durch Dritte wie Sicherheitsforscher oder Strafverfolger und nicht durch interne Prozesse. | DevSecOps |

das Content-Delivery-Netzwerk Cloudflare durchleitet. Fast die Hälfte der vom Sicherheitsanbieter Radware befragten Unternehmen gab an, dass die Mehrheit ihrer Anwendungen über APIs mit dem Internet oder mit Diensten Dritter verbunden ist. Der Hersteller Salt schreibt in seinem State of API Security Report 2022, dass sich bei den befragten Kundenorganisationen die durchschnittliche Zahl der APIs in zwölf Monaten auf über hundert mehr als verdoppelte und 95 Prozent im vergangenen Jahr einen damit verbundenen kleineren oder größeren Sicherheitsvorfall erlitten. Das Whitepaper des Konkurrenten Wallarm beschreibt, dass allein im ersten Quartal 2022 Enterprise-Software wie Veeam Backup von Schwachstellen aus den API Top 10 betroffen war, ebenso Infrastrukturkomponenten wie Drupal und selbst Sicherheitssoftware von McAfee (alle Links zu den Reports unter ix.de/zhxu).

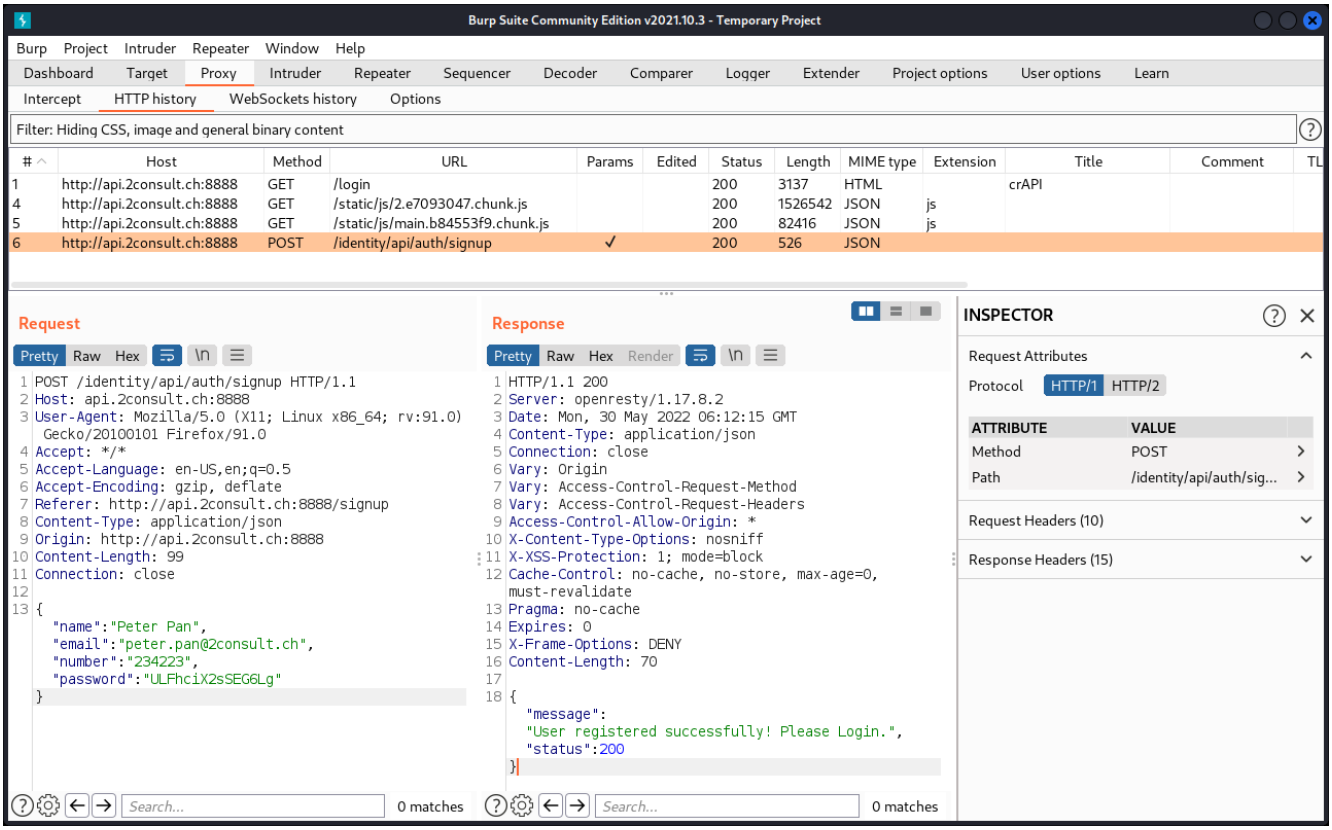
Die Autoren von APIsecurity.io veröffentlichen wöchentlich einen lesenswerten kompakten Newsletter, der Artikel zu API-Schwachstellen in konkreten Anwendungen und Webdiensten, neue Auditwerkzeuge sowie andere Neuigkeiten zu Schnittstellensicherheit verlinkt. Sie haben in den vergangenen drei Jahren Berichte über mehr als 350 Einbrüche durch API-Lücken gesammelt (siehe ix.de/zhxu). Nicht nur Hersteller, die ihre Produkte verkaufen wollen, schlagen Alarm: Das deutschsprachige Blog des Hackerkollektivs zerforschung dokumentiert unter den Schlagworten Security und API zahlreiche Lücken in Anwendungen von Coronatestzentren, Lieferdiensten und Lernanbietern, durch die Benutzerdaten abfließen konnten (siehe ix.de/zhxu).

## Schnittstellen sind Webanwendungen – und doch nicht

APIs bieten ähnliche, aber doch andere Angriffsflächen als monolithische Webanwendungen, weil die Daten und Informationen über den aktuellen Zustand auf den Client wandern, statt wie bei klassischen Web-Apps auf dem Server zu verbleiben. Anwendungslogik liegt im Client offen; nicht alle Sicherheitsmechanismen für Webanwendungen greifen mehr.

Schnittstellen entwickeln sich rasch weiter. Ein sichtbares Zeichen dafür ist die Versionsnummer in manchen Pfaden. Ihre URLs sind durch den Architekturstil REST (Representational State Transfer) zumindest innerhalb einer Schnittstelle ähnlich. Angreifer können so die Namen von Endpunkten erraten. Dabei sind vor allem REST-APIs in ihrem Blickpunkt. Dieser Artikel beschränkt sich bei den konkreten Beispielen darauf.

Ebenso bringt die von Facebook vorangetriebene Abfragesprache GraphQL, die an SQL erinnert, neue Angriffspunkte und Herausforderungen für Verteidiger mit. Gleiches gilt für das von Google entwickelte gRPC-Framework (Remote Procedure Call), auch wenn es für Angreifer wegen der geringeren Verbrei-




**Der Interception-Proxy Burp zeichnet den Verkehr zwischen SPA im Browser oder einer App auf dem Mobilgerät und der Schnittstelle im Klartext auf (Abb. 1).**

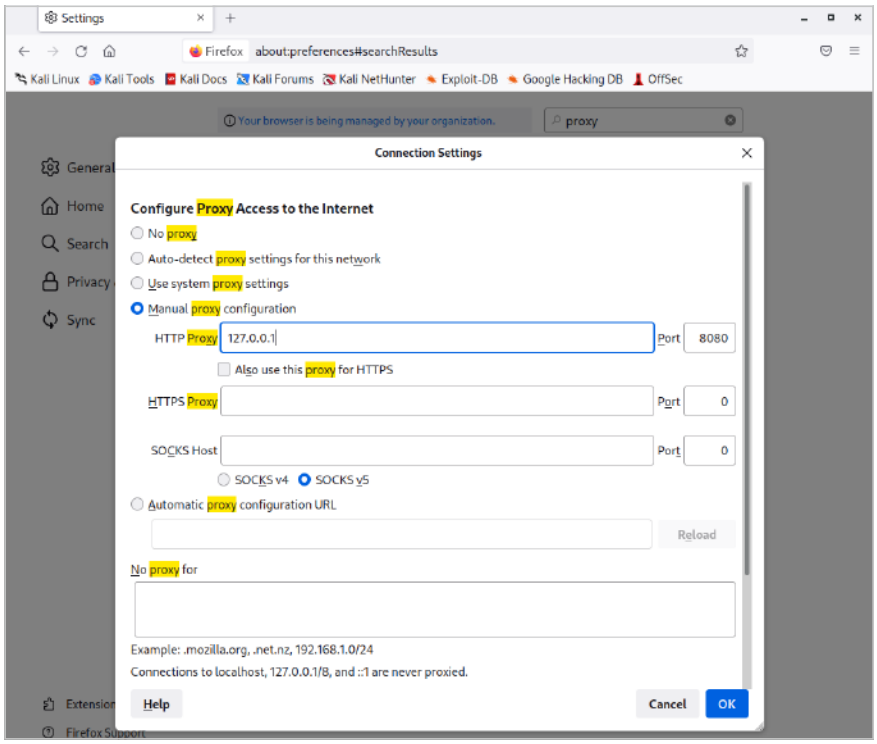
tung und fehlender Werkzeuge unbequemer auf Lücken abzuklopfen ist.

Es gibt gute Nachrichten: Schnittstellen sind weniger von klassischen Schwachstellen wie SQL Injections betroffen, weil Frameworks und objektrationale Mapper (ORM) Fehlerquellen von Programmierern wegabstrahieren. Mit Gefahren durch XML External Entity (XXE) Injection, die in einer älteren Version der Web Top 10 einen eigenen Punkt einnahmen, müssen sich nicht diejenigen herumschlagen, deren Schnittstellen kein XML entgegennehmen, sondern wie REST-üblich JavaScript Object Notation (JSON).

Entwickler brauchen nicht mehr Cross-Site Request Forgery (CSRF) zu fürchten, wenn ihre SPA Autorisierungsheader setzt, statt sich wie bei klassischen Webanwendungen darauf zu verlassen, dass der Browser Sitzungscookies automatisch mit jeder Anfrage mitsendet. Bei CSRF verleitet ein Angreifer über einen Phishinglink ein Opfer zu unbeabsichtigten Aktionen in der betroffenen Webanwendung.

## Das Männchen im Browser – Interception-Proxys

Das Untersuchen einer API beginnt für wohlmeinende Tester und für böswillige Angreifer damit, einen Interception-Proxy einzurichten. Der Proxy fungiert als Man in the Middle zwischen dem Browser oder einer mobilen App und der zu untersuchenden respektive angegriffe-




**Im Firefox-Browser in Kali konfiguriert man den Proxy auf den Localhost 127.0.0.1 und Port 8080, auf dem Burp im Standard lauscht (Abb. 2).**

nen Schnittstelle. Er zeichnet die gesamte HTTP-Kommunikation und den mit TLS verschlüsselten Verkehr im Klartext auf und erlaubt, Anfragen beliebig zu wiederholen oder zu manipulieren.

Der bekannteste Interception-Proxy ist die kommerzielle Burp Suite von PortSwigger, die als Java-Anwendung unter den meisten Betriebssystemen läuft. Die erschwingliche Software für Webtests – eine Lizenz kostet weniger als 500 Euro – gibt es in einer kostenlosen Community Edition mit eingeschränktem Funktionsumfang. Im Web finden sich zahlreiche Anleitungen, YouTube-Tutorials und Kurse zu Burp. Abbildung 1 zeigt die Historie aller aufgezeichneten Clientabfragen und Antworten des Webservers oder der API. Durch einen Klick in der Tabelle auf die Spalte mit dem Rautezeichen (#) sortiert man die Historie absteigend chronologisch, was große Listen übersichtlicher macht.

Der quelloffene OWASP Zed Attack Proxy (ZAP) ist eine kostenfreie Alternative, jedoch sperriger zu bedienen und mit weniger öffentlichem Informationsmaterial ausgestattet. Er bietet ebenso ein Ökosystem an Erweiterungen. Wie Entwickler mit ZAP zusammen mit Postman, dem Standardwerkzeug für Funktionstests von APIs, Schnittstellen auf Lücken prüfen, beschreibt ein Blogartikel bei The Test Therapist (siehe ix.de/zhxu).

Basis für Sicherheitstests ist am besten eine virtuelle Maschine mit Kali Linux, einer Debian-Distribution mit vorinstallierten Auditwerkzeugen. Viele Tools sind darauf ausgelegt. Auf der Website findet man betriebsfertige VMs für unterschiedliche Virtualisierer und eine englische Doku; Benutzername und Passwort lauten zunächst kali. Längere deutsche Anleitungen finden sich unter ix.de/zhxu. Weil manche Tools Speicher fressen, weist man der Kali-Maschine mindestens 6 GByte RAM zu. Falls die Tastaturbelegung nicht passt, gibt man im Kali-Menü „Keyboard“ ein und korrigiert sie in der Registerkarte „Layout“.

Die vorinstallierte Burp-Community-Version startet man aus dem Kali-Menü über den Eintrag „burpsuite“ und klickt im Startassistenten auf die Schaltflächen „Next“ und „Start Burp“. Im Firefox-Browser gelangt man über das Hamburger-Symbol in die Einstellungen, tippt im Suchfeld „Proxy“ ein, klickt im Ergebnis auf „Settings“, wählt „Manual proxy configuration“, gibt als HTTP-Proxy 127.0.0.1 auf Port 8080 ein und bestätigt die Eingaben (Abbildung 2). Auf demselben Weg, nur durch die Wahl von „No proxy“, nimmt man den Proxy wieder aus der Internetverbindung.

**Listing 1: Die absichtlich verwundbare crAPI unter Kali Linux**

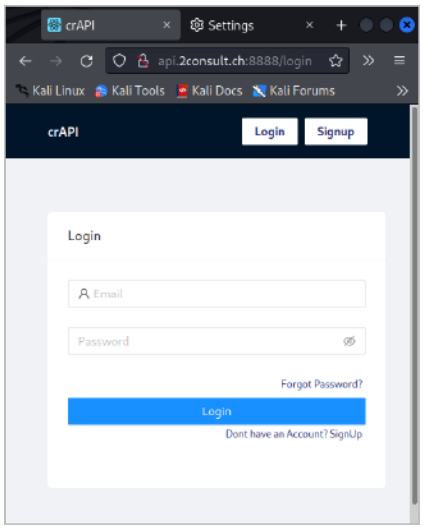
```
$ sudo apt update
$ sudo apt install -y docker.io docker-compose
$ sudo systemctl enable docker --now
$ echo "127.0.0.1 api.2consult.ch" | sudo tee -a /etc/hosts
$ git clone https://github.com/OWASP/crAPI.git && cd crAPI/deploy/docker
$ sudo docker-compose pull
$ sudo docker-compose --compatibility up -d
```

Manche Entwickler verlassen sich auf den vermeintlichen Schutz der Transportverschlüsselung. API-Nutzer haben volle Kontrolle über ihr Endgerät. Bei Einsatz des Proxys können sie den kompletten Verkehr zwischen Browser oder App und den Endpunkten der Schnittstelle einsehen und manipulieren.

Will man produktive Anwendungen testen, die über HTTPS verschlüsselt sind, muss man das Kontrollkästchen „Also use this proxy for HTTPS“ aktivieren und wie in der Online-Burp-Dokumentation beschrieben ein Proxyzertifikat im Browser installieren (siehe ix.de/zhxu). Zum Verfolgen dieses Artikels ist das nicht notwendig.

## Learning by doing: eine Übungsumgebung aufbauen

Am einfachsten lernt man durchs Machen. Um eine Übungsumgebung für API-Lücken einzurichten – ganz gleich, ob man Projektleiterin, Entwickler oder Pentester ist –, bietet sich eine Kombination von Interception-Proxy und absichtlich verwundbarer Software an: Ziel der Angriffe ist eine mit Vorsatz löchrige Schnittstelle. Davon gibt es inzwischen ein gutes Dutzend, die entweder direkt zu installieren sind oder als Docker-Image vorliegen (Übersicht siehe GitHub-Projekt „Awesome API Security“, ergänzt um viele Quellen zu Angriffen auf Schnittstellen und ihrer Verteidigung, unter ix.de/zhxu).




**Die absichtlich verwundbare Schnittstelle crAPI zeigt sich mit der SPA-Weboberfläche (Abb. 3).**

Ende 2021 veröffentlichten die Autoren der API Top 10 die completely ridiculous API (crAPI), die in verschiedenen Sprachen und Plattformen entwickelte Microservices vereint – allesamt auf Schwachstellen der Liste anfällig. Sicherheitstester und Interessierte üben an ihr das Aufspüren und Ausnutzen von Lücken, Programmierer lernen so typische Fehler kennen. In Kali Linux installiert und startet man die „völlig lächerliche“ API und legt für sie einen lokalen DNS-Eintrag an (Listing 1).

Das Hochfahren der Docker-Container dauert eine Weile, danach läuft in Kali unter http://api.2consult.ch:8888/ die crAPI-Web-App wie in Abbildung 3, bei der man zunächst ein Benutzerkonto anlegt. Wenn Burp bereits als Proxy eingerichtet ist, sieht es so aus, als würde die Verbindung hängen. Dann wechselt man in Burp auf die Registerkarte „Proxy“, klickt unter „Intercept“ auf die Schaltfläche „Intercept is on“ und schaltet so das direkte Abfangen aus.

## Unsachgemäße Inventarisierung – API9

Angriffsfläche einer Schnittstelle sind alle aus dem Internet erreichbaren Endpunkte. Für Angreifer ist es nur eine kleine Herausforderung, mit der Suche nach öffentlich verfügbaren Daten wie registrierten Domänen und Subdomänen die Staging-Umgebung einer API (staging.api.2consult.ch) oder vergessene und inzwischen veraltete Testsysteme aufzuspüren, die nicht so gut geschützt sind wie die Produktion. Ähnlich ist es mit alten API-Versionen oder Varianten, die über Pfade wie `/v1/` und `/v1.1/` oder `/1.0-deprecated/` und `/2.1-uat1/` erreichbar sind.

Statt die Anmeldung in der aktuellen Version `/v3/login` unter die Lupe zu

**Listing 2: Kiterunner und Wörterlisten installieren sowie nach API-Routen suchen**

```
$ sudo apt update && sudo apt install -y golang
$ cd && git clone https://github.com/assetnote/kiterunner.git && cd kiterunner
$ make build && sudo ln -s $(pwd)/dist/kr /usr/local/bin/kr & cd
$ mkdir wordlists && cd wordlists && wget -r --no-parent -R "index.html*" https://wordlists-cdn.assetnote.io/data/ -nH
$ cd ~/wordlists/data/kiterunner && tar xvzf routes-large.kite.tar.gz && cd
$ kr scan http://api.2consult.ch:8025 -w ~/wordlists/data/kiterunner/routes-large.kite
```

nehmen, klopft ein Angreifer `/v2/login` auf Anfälligkeiten ab. Manchmal pflegt eine Organisation mehrere Varianten einer API für SPA und App oder Entwickler schreiben für die Anmeldung von Mobilgeräten eigene Endpunkte. Dabei können sicherheitsrelevante Unterschiede bestehen.

## Eine richtige Inventarisierung mindert das Angriffsrisiko

Endpunkte von Nichtproduktivversionen einer Schnittstelle, die ein Betreiber wegen unsachgemäßer Inventarisierung nicht im Blick hat, sind für Eindringlinge oft ein erster Ansatzpunkt für weitere Attacken: Auf vergessenen Systemen und in veralteten Versionen suchen sie nach Lücken wie der Preisgabe sensibler Daten oder einer Injection. Bei fehlender Ratenbegrenzung starten sie Brute-Force-Angriffe auf Log-ins oder damit zusammenhängende Funktionen wie die Passwortwiederherstellung. Beispielsweise hätte ein böswilliger Angreifer das Passwort jedes Facebook-Benutzers über einen Betaendpunkt zurücksetzen können, wie der Bugfinder beschreibt (siehe ix.de/zhxu).

Ein neueres, für APIs optimiertes Open-Source-Werkzeug zum Finden spannender Endpunkte ist Kiterunner, das wie in Listing 2 installiert wird, um Wörterlisten von seinem Anbieter Assetnote ergänzt wird und im Mailserver der crAPI nach Endpunkten fahndet.

Admins und Entwickler sollten eine organisationsinterne Dokumentation pflegen, in der sie Systeme und API-Versionen inventarisieren. Alte, nicht mehr benötigte Versionen klemmen sie so bald wie möglich ab. Bleiben die Oldies aktiv, sind die in einer neueren Version gefundenen Lücken auch dort zu beheben und die Softwareabhängigkeiten weiterhin zu aktualisieren. Das gilt auch für nur als intern betrachtete APIs.

Ironischerweise ist es manchmal eine allen zugängliche OpenAPI-Dokumentation, in der Eindringlinge Hinweise finden, wenn sie in einer alten Version vorliegt oder automatisiert sämtliche Endpunkte beschreibt. OpenAPI, früher Swagger, hat sich als Standard zum Beschreiben von REST-APIs durchgesetzt. Ein Anbieter schildert in einem Artikel, welche Schwachstellen er in öffentlich zugreifbaren APIs fand, die ihre Swagger-Dokumentation veröffentlichten. In einem Blogpost berichtet ein Bug-Bounty-Jäger, wie er unbeabsichtigte Funktionen in von Web-Apps genutzten APIs aufspürt (beide Links unter ix.de/zhxu). Beispielsweise ist vielen Entwicklern nicht bewusst, dass die Haupt-JavaScript-Datei ihrer SPA, die jeder unangemeldete Benutzer im Proxy oder den Browserentwicklertools sieht, oft sämtliche API-Endpunkte enthält; so auch bei der crAPI.

Verteidiger sollten ihre Produktivumgebung von Test und Staging strikt trennen und keine Kundendaten auf Entwicklungssystemen nutzen. Falls doch, müssen in allen Umgebungen die gleichen Sicherheitsfunktionen konfiguriert sein und Prozesse wie Patchmanagement eingehalten werden. Richtige Inventarisierung ist nicht einfach, aber Werkzeuge helfen, zumindest ein unsachgemäßes Inventar aufzudecken. Ist eine OpenAPI-Spezifikation vorhanden, findet das Testframework Dredd Unterschiede zwischen Doku und Implementierung. Fehlt eine Schnittstellenbeschreibung, sollte man sie aus der Entwicklungsumgebung heraus erstellen. Zur Not greifen Entwickler und Tester auf mitmproxy2swagger zurück, das aus mitgeschnittenem Datenverkehr eine OpenAPI-Spezifikation generiert.

### Schutz vor geknackten Passwörtern

Wenigen Verteidigern ist bewusst, dass sie in dem populären Projekt Have I Been Pwned (HIBP) nicht nur suchen können, ob eine E-Mail-Adresse in einem Datenleck vorkommt, der Betreiber stellt auch herunterladbare Listen der kompromittierten Passwörter und eine API bereit (siehe ix.de/zhxu). Wenn in einer Anwendung ein Benutzer ein Passwort setzt, prüfen Entwickler mit Bibliotheken, ob es in der HIBP-Datenbank steht und deswegen als bekanntermaßen unsicher abgelehnt wird. Das ist ein guter Schutz gegen Credential Stuffing, bei dem Angreifer Kombinationen aus Benutzernamen und Kennwort ausprobieren, die in Datenlecks abgeflossen sind.

## Fehlerhafte Authentifizierung – API2

Die Mehrzahl der Lücken in den API Top 10 hat mit Fehlern bei der Zugriffskontrolle zu tun, die aus Authentifizierung und Autorisierung besteht. Wann immer ein Client mit einer Schnittstelle kommuniziert, ist sichere Authentifizierung wichtig: Der API-Endpunkt muss für jede Anfrage prüfen, zu welchem Benutzer sie gehört und ob sich der entsprechende Benutzer angemeldet hat. Besonders dabei sind Schnittstellen lückenhaft. Angreifer melden sich womöglich ohne Authentifizierung an, täuschen die Identität eines anderen Benutzers vor und erbeuten Passwörter oder Sitzungsinformationen.

Log-in-Funktionen müssen bei einer öffentlichen Anwendung notwendigerweise allgemein zugänglich sein. Bei APIs kommt hinzu, dass es aus historischen Gründen manchmal mehrere Anmeldeendpunkte gibt, etwa nicht nur `/api/v3/login`, sondern von einem Mobilgerät aus noch `/api/mobile/login`. Schützen Programmierer diese kritischen Endpunkte nicht besonders oder implementieren sie einen Mechanismus fehlerhaft, entstehen Lücken. Oft vergessen Verteidiger, dass nicht nur die eigentliche Anmeldung, sondern ebenso verwandte Funktionen wie Registrierung und Passwortänderung dazu zählen. Wichtig ist, den Passwort-per-E-Mail-zurücksetzen-Endpunkt gegen Brute-Force-Angriffe zu schützen, etwa durch Ratenbegrenzung wie später beschrieben. Zudem muss jede HTTP-Methode geschützt sein, sonst missbraucht ein Angreifer womöglich Funktionen mit einem weniger gebräuchlichen Verb wie `PATCH` unauthentisiert.

Programmierer sind sich oft nicht bewusst, dass API-Schlüssel ein schlechter Mechanismus sind, wollen sie Benutzer authentifizieren und nicht nur die Anwendung, von der aus eine API aufgerufen wird. Falls sie doch API-Schlüssel einsetzen, müssen sie sich Gedanken um

**Listing 3: JSON Web Token Toolkit installieren und prüfen**

```
$ sudo apt install python3-requests python3-pycryptodome python3-termcolor
$ pip install cprint
$ git clone https://github.com/ticarpi/jwt_tool.git
$ cd ~/jwt_tool
$ python3 jwt_tool.py -t http://api.2consult.ch:8888/identity/api/v2/user/dashboard -rh ↵
                                                    "Authorization: Bearer eyJhbGciOiJIUzUxMiJ9.[...]" -cv "role" -M at
```

**Listing 4: Schlüssel der crAPI-Zugriffstoken ausfindig machen**

```
$ sudo gzip -d /usr/share/wordlists/rockyou.txt.gz
$ echo -e "crAPI\ncrapi\ncrapi2022" | sudo tee -a /usr/share/wordlists/rockyou.txt
$ python3 jwt_tool.py eyJ[...] -C -d /usr/share/wordlists/rockyou.txt
[+] crapi is the CORRECT key!
```

**Listing 5: Anfrage mit Identifier**

```
GET /identity/api/v2/vehicle/ad7e0d91-6aa5-4540-a789-50f7bef47ee3/location HTTP/1.1
Host: api.2consult.ch:8888
```

die Rotation der Schlüssel machen. Generell sollten sensible Daten wie Schlüssel, Passwörter oder Zugriffstoken niemals als Abfrageparameter direkt in einer URL enthalten sein, weil sie sonst in der Browserhistorie und in Serverlogs landen und ein Referer-Header sie an andere Seiten übermittelt.

Ende 2021 hat die Sicherheitsforscherin Alissa Knight Schnittstellen und mobile Apps von fünfzig Banken untersucht (siehe ix.de/zhxu). Sie fand in allen APIs Lücken bei der Authentifizierung und konnte ohne ordnungsgemäße Anmeldung auf andere Konten zugreifen. Die Client-Apps enthielten in vielen Fällen hartcodierte API-Schlüssel und Zugriffstoken.

## In sich selbst ruhende Zugriffsschlüssel: JSON Web Token

Authentifizierung wird klassischerweise implementiert, indem der Server dem Benutzer nach erfolgreicher Anmeldung eine sitzungsspezifische Kennung ausstellt und serverseitig den Status der Sitzung verwaltet. Um den Anwendungsserver zu entlasten und in Szenarien mit mehreren Parteien ist mit einem kryptografisch signierten Token eine zustandslose Sitzungsverwaltung möglich: Alle Daten wie Ablaufzeit, Benutzer-ID und Berechtigungen sind in einem Token zusammengefasst und werden an die Schnittstelle gesendet. Sie prüft die Integrität des Tokens anhand seiner Signatur. Dadurch muss sie nicht mehr Sitzungszustände speichern, weil der Status im Token gekapselt ist und auf den Client wandert.

JSON Web Token (JWT) zusammen mit JSON Web Signature (JWS) sind etablierte Standards, zustandslose Sitzungsverwaltung auf Basis eigenständiger Token zu implementieren. Ein JWT hat die Form `header.payload.signature`, getrennt durch Punkte. Header und Payload sind Sätze von Schlüssel-Wert-Paaren in Base64-codiertem JSON. Ein JWT darf ebenso wenig wie eine klassische Sitzungskennung zu Dritten gelangen, etwa über einen Referer-Header. Im Token dürfen keine sensiblen Daten enthalten sein: Die Payload ist nur encodiert und nicht verschlüsselt und über Webtools wie JWT.io einfach einsehbar. Jede Partei, die das Token erhält, kann es decodieren und die Angaben im Klartext lesen. Falls notwendig lassen sich Inhalte mit dem Standard JSON Web Encryption (JWE) verlässlich verschlüsseln.

Sind JSON Web Token unzureichend kryptografisch gesichert oder ist die Implementierung respektive Konfiguration ihrer Prüfung fehlerhaft, kann ein Angreifer Daten für Authentifizierung und Autorisierung manipulieren und so seine Privilegien erweitern. Dabei ist zunächst wichtig, dass die Anwendung ein JWT nicht nur decodiert und dann den darin enthaltenen Angaben (Claims) blind vertraut, sondern prüft, ob das Token gültig ist: Ist eine Signatur vorhanden? Ist die Signatur gültig gemäß dem im Quelltext der Anwendung angegebenen Algorithmus? Ist das Token nicht abgelaufen? Dazu verwenden Entwicklerinnen und Entwickler am besten eine in der Praxis bewährte und aktuell gehaltene Bibliothek.

Bei der Signaturverifikation lauern ebenfalls Gefahren: Ist der Standard wortgetreu implementiert, verwendet die Anwendung den im Header angegebenen Algorithmus, der die Signatur prüft. Da der Signaturalgorithmus None spezifiziert ist, fälscht ein Angreifer die Token einfach mit einer leeren Signatur. Betroffen davon waren 2021 das Publish-Subscribe-Messaging-System Apache Pulsar und der E-Mail-Dienst Outlook von Microsoft (siehe ix.de/zhxu). Eine Schnittstelle muss den Algorithmus hartcodiert vorgeben und nicht dynamisch aus dem Token auslesen.

## Die richtige Verschlüsselung wählen

Für die Signatur müssen sich Entwickler zwischen symmetrischer und asymmetrischer Kryptografie entscheiden. Ein symmetrischer Algorithmus wie HS256 (HMAC mit SHA-256) hat nur einen von allen Komponenten und Parteien geteilten Schlüssel. Da man ihn zum Erzeugen und Validieren der Signatur verwendet, darf er nicht öffentlich werden. Ist der Schlüssel kompromittiert, kann ein Angreifer Zugriffstoken fälschen. Symmetrische Kryptografie sollte man nur dann verwenden, wenn eine einzige Partei alle Komponenten kontrolliert. In diesem Fall muss der gemeinsame Schlüssel stark genug sein, bei HS256 empfiehlt sich ein 256 Bit langes Geheimnis. Andernfalls lässt sich der JWT-Schlüssel mit einem Brute-Force-Angriff offline knacken.

Zum Testen von JWT dient das JSON Web Token Toolkit (jwt_tool), in das man ein aktuelles aus Burp kopiertes Token einfügt, und als cv-Parameter eine Zeichenkette, die die Schnittstelle bei erfolgreicher Anmeldung zurückgibt (Listing 3).

Zugriffstoken der crAPI sind durch keinen Implementierungsbug wie den None-Algorithmus verwundbar, aber mit einem schwachen Schlüssel signiert, der sich mit einer Wörterliste ermitteln und einfach raten lässt (Listing 4).

Angreifer stellen sich mit dem Geheimnis beliebige Zugriffstoken aus, etwa als Admin. Für Sicherheitstester und Interessierte ist neben dem umfangreichen GitHub-Wiki des jwt_tool der Blogpost „Hacking JSON Web Tokens“ ein guter Einstieg ins Thema (siehe ix.de/zhxu).

Ein asymmetrischer Algorithmus wie RS256 (RSA-Signatur mit SHA-256) und

der neuere PS256 nutzen einen privaten und einen öffentlichen Schlüssel. Da man öffentliche Schlüssel nicht sicher aufbewahren muss, können Identitätsanbieter wie Google und Facebook ihn über eine Metadaten-URL leicht zugänglich machen.

## Wie lange Token gültig sein sollten

Zugriffstoken sollten möglichst kurz gültig sein, eine Nutzungsdauer von 20 Minuten genügt. Aktualisierungstoken, die neue Zugriffstoken anfordern, sind je nach Anwendungsart unterschiedlich lange gültig: höchstens 30 Tage, bei APIs mit erhöhtem Schutzbedarf oder sensiblen Daten maximal 12 Stunden. Herkömmliche serverseitige Sitzungen laufen am besten nach einem Arbeitstag ab.

Wer JWT einsetzt, sollte die JSON Web Token Best Current Practices lesen oder einen Blick auf das einseitige Cheat Sheet bei Pragmatic Web Security werfen (siehe ix.de/zhxu). Bevor Entwickler voreilig auf eigenständige Zugriffstoken wie JWT umrüsten, sollten sie überlegen, ob für ihren Anwendungsfall nicht eine althergebrachte serverseitige Sitzung genügt. Das vermeidet das Problem, bei JWT Sitzungen nicht mehr jederzeit beenden zu können und zum Widerrufen einmal ausgestellter Token doch wieder Zustände auf einem Server zu verwalten.

Verwendet die Anwendung nur eine Domäne, können Entwickler zum Übermitteln der Sitzungskennung Cookies einsetzen und sich sparen, die Übertragung in einem Authorization-Header zu implementieren. Das hat den Vorteil, dass ein Browser bei jeder Anfrage von selbst Cookies mitsendet. Damit sind auf einfache Weise authentisierte Anfragen von eingebetteten Bildern und der Aufbau von WebSocket-Verbindungen möglich. Allerdings muss man sich dann um CSRF-Schutz kümmern. Die Art der Sitzungsverwaltung und der Übertragungsweg sind prinzipiell voneinander unabhängig: Klassische Sitzungskennungen sind in einem Authorization-Header und ein JWT als Cookie übertragbar.

Programmieren Entwickler eine SPA mit API, neigen sie dazu, die Gefahr von klassischem Cross-Site Scripting (XSS) zu unterschätzen, das sich womöglich trotz Schutzmaßnahmen in ihrem SPA-Framework in den Client eingeschlichen hat. Angreifer führen mit XSS beliebigen JavaScript-Code aus: Sie lesen nicht nur Zugriffstoken aus dem `localStorage`- oder `sessionStorage`-Objekt aus und übertragen sie zu sich, sondern lösen jede Anfrage aus der Clientanwendung so aus, dass sie für die Schnittstelle legitim aussieht. Bei besonders kritischen Anwendungen wie Gesundheit und Finanzen lohnt ein Blick auf das unbekanntere Entwicklungsmuster Backend for Frontend (BFF), das weitere Teile der Authentifizierung wieder auf den Server holt.

Eine Übersicht zu Mechanismen, Sitzungscookies und JWT, beiderseitige Authentifizierung über TLS (mutal TLS, mTLS) und was das für die Sicherheit bedeutet, gibt der Vortrag „Serving the right recipe for API authentication“ (siehe ix.de/zhxu). Entscheiden sich Architekten und Entwickler für JWT, sollten sie in den meisten Fällen auf asymmetrische Kryptografie setzen, die bei verteilten Systemen sicherer ist. Wie sehr die Sicherheit von JWT an kryptografischen Primitiven und gut entwickelten Bibliothek hängt, zeigte sich zuletzt im April 2022: Vom Psychic-Signatures-Bug in bestimmten Java-Versionen waren JWT-Signaturen mit Elliptic Curve Digital Signature Algorithm (ECDSA) betroffen, die gefälscht werden konnten.

## Elegant delegiert: OAuth 2.0 und OpenID Connect

JWTs haben als Mechanismus für clientseitigen Status andere Eigenschaften als serverseitige Sitzungen, in einer Anwendung verwendet man sie nicht einfach analog. Wenn sich Entwickler für JWT entscheiden, benötigen sie weitere Standards wie OAuth 2.0, um effektiv und sicher Zugriffsentscheidungen zu treffen.

Die Protokolle OAuth 2.0 für Autorisierung und Zugriffsdelegierung sowie das darauf aufsetzende OpenID Connect (OIDC) für Authentifizierung und Identitätsweitergabe sind der Unterbau hinter Single Sign-on (siehe Artikel „APIs mit Keycloak absichern“ in iX 8/2022). Ein Benutzer gestattet mit OAuth einem Dritten, in seinem Namen einen Dienst (Resource Server) zu verwenden, ohne dass er seine eigentlichen Zugangsdaten weitergeben muss. Er gewährt einem Client wie einer SPA oder App beim Autorisieren granularen Zugriff auf seine Daten. OpenID Connect als Schicht oberhalb von OAuth definiert, wie nach dem eigentlichen Log-in über eine REST-Schnittstelle Informationen über die Identität des Anwenders abgerufen werden.

Obwohl dazu gedacht, Autorisierung und Delegierung zu vereinfachen und sicherer zu machen, waren OAuth-Implementierungen im Lauf der Zeit von Schwachstellen geplagt. Wenn eines der beteiligten Systeme – Authorization Server oder Resource Server – nicht streng genug prüft, verwenden Angreifer die Zugangsdaten eines Opfers wieder, fangen sie ab oder verknüpfen eine von ihnen kontrollierte Ressource mit dem Konto des Opfers.

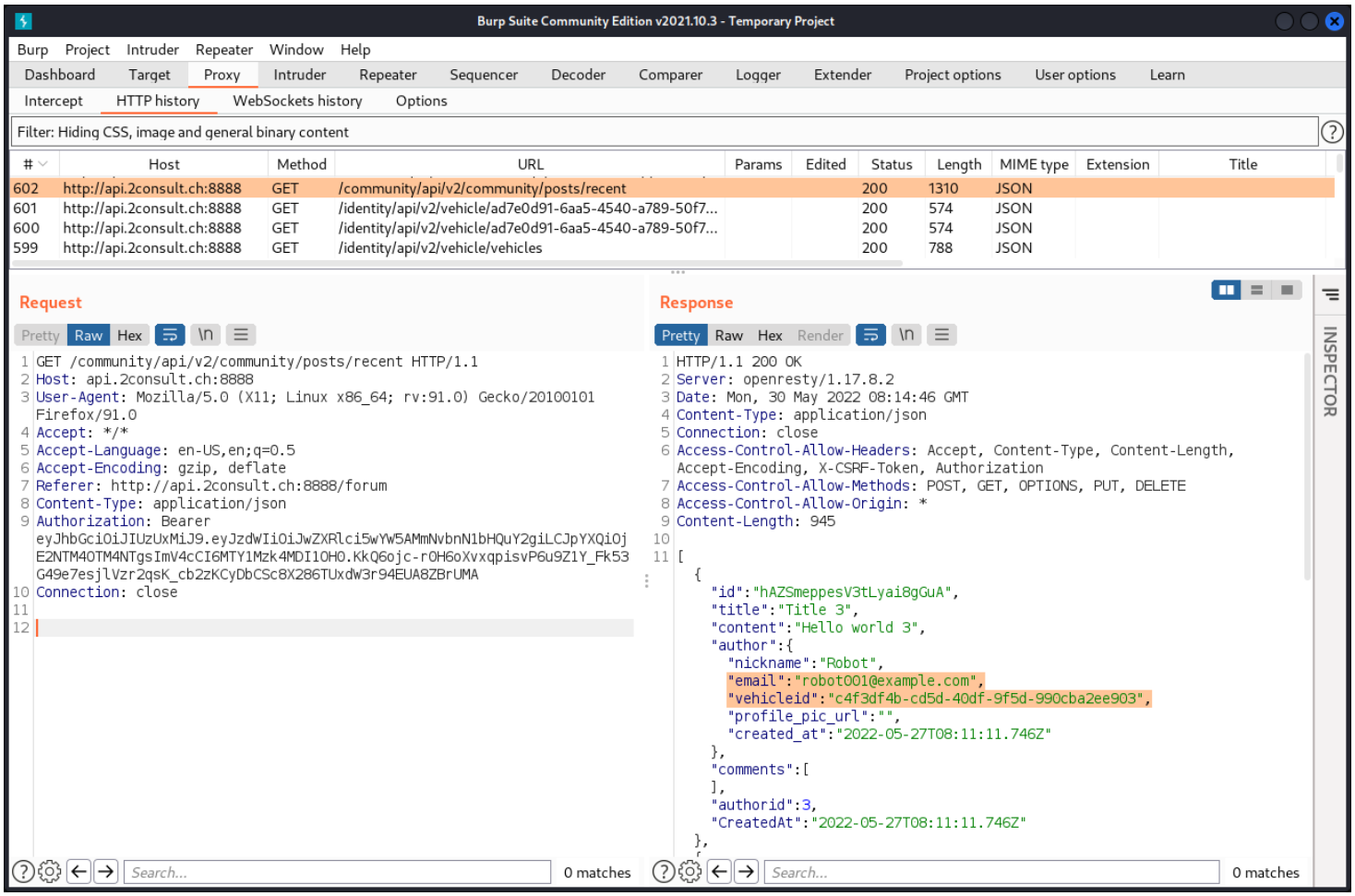
Hinzu kommt, dass die zugrunde liegenden Standards komplex sind und OAuth je nach Szenario – klassische Webanwendung auf einem Server, SPA im Browser, native App auf einem Smartphone – unterschiedliche Anmeldeflüsse bietet, die man Flows oder Grant Types nennt. Inzwischen gilt für die meisten Szenarien der Authorization Code Flow mit dem Zusatz Proof Key for Code Exchange (PKCE, gesprochen: Pixie) als Anmeldefluss der Wahl; er eignet sich für Clients wie SPAs und mobile Apps, die prinzipbedingt keine Geheimnisse wahren können.

Gegen OAuth-Angriffe schützen sich Entwickler, indem sie bei der Implementierung und Konfiguration alle Details zu den spezifizierten Schutzmaßnahmen beachten. Am besten setzen sie bewährte Lösungen für Authorization und Resource Server ein sowie vielgenutzte Bibliotheken für SPAs und Apps. Zudem beherzigen sie die OAuth 2.0 Security Best Current Practice. Wer nicht das gesamte Dokument durchsehen will, schaut sich das Cheat Sheet an (siehe ix.de/zhxu).

## Fehlerhafte Autorisierung auf Objektebene – API1

Autorisierung beantwortet die Frage: „Was darfst du?“, nachdem die Authentifizierung „Wer bist du?“ verlässlich festgestellt hat. Viele APIs prüfen die Berechtigungen von einmal authentifizierten Benutzern auf Objektebene nur unzureichend, daher der englische Name Broken Object Level Authorization (BOLA). Die Autoren der API Top 10 verzeichnen sie auf Platz 1, weil sie eine der verbreitetsten Lücken ist, ähnlich wie SQL Injections in den Anfangstagen des Webs. Vor allem wenn Autorisierungsmechanismen nicht zentral implementiert sind, wird bei Microservices leicht die Prüfung für eine nachgelagerte Komponente vergessen. Wird ein verwundbarer Endpunkt direkt statt von einem anderem Dienst mit eingebauter Zugriffsprüfung aufgerufen, entsteht eine Schwachstelle. Zudem ist die Angriffsfläche größer, weil bei REST-APIs Clients vermehrt Anfragen mit Identifier senden.

Ein Angreifer kann eine ID beliebig durch eine andere ersetzen und, falls die API die Autorisierung des Benutzers nicht prüft, Zugriff auf die Daten eines




**Die Antwort der API gibt sensible Daten preis: im legitimen Client nicht angezeigt, aber im Proxy sichtbar (Abb. 4).**

anderen Benutzers oder Mandanten erlangen. Diese Lücke wird manchmal als Insecure Direct Object Reference (IDOR) bezeichnet. Betroffen sind nicht nur IDs in Pfaden von Anfragen, sondern auch in Headern und JSON-Objekten.

In der crAPI unter Kali liest man im eingebauten Webmailserver unter http://api.2consult.ch:8025/ E-Mails, die die verhunzte API sendet. Mit den Angaben registriert man ein Fahrzeug und sieht es sich im Dashboard an. Über die Schaltfläche „Refresh Location“ fragt man den Ort ab; diese Anfrage enthält einen Identifier (Listing 5). Meldet man nun einen zweiten Benutzer an und stellt die Anfrage mit derselben ID, sieht der zweite Benutzer den Aufenthaltsort des ersten.

Damit eine solche Verwundbarkeit nicht entsteht, müssen Entwickler bei jedem Zugriff die Berechtigung serverseitig prüfen. Das Frontend spielt bei Autorisierungsprüfungen keine Rolle. Die API darf nicht der vom Client gesendeten ID vertrauen, sondern muss bei jedem Zugriff über die serverseitige Sitzung und bei JWT in dessen Claims feststellen, ob der authentifizierte Benutzer die gewünschte Aktion auf der gewünschten Ressource durchführen darf, etwa weil sie sich auf seine Benutzerkennung bezieht oder er der Eigentümer der Ressource ist. Auch als intern angesehene Endpunkte müssen jeden Zugriff selbst prüfen und dürfen sich nicht auf Prüfungen vorgelagerter Schnittstellen verlassen.

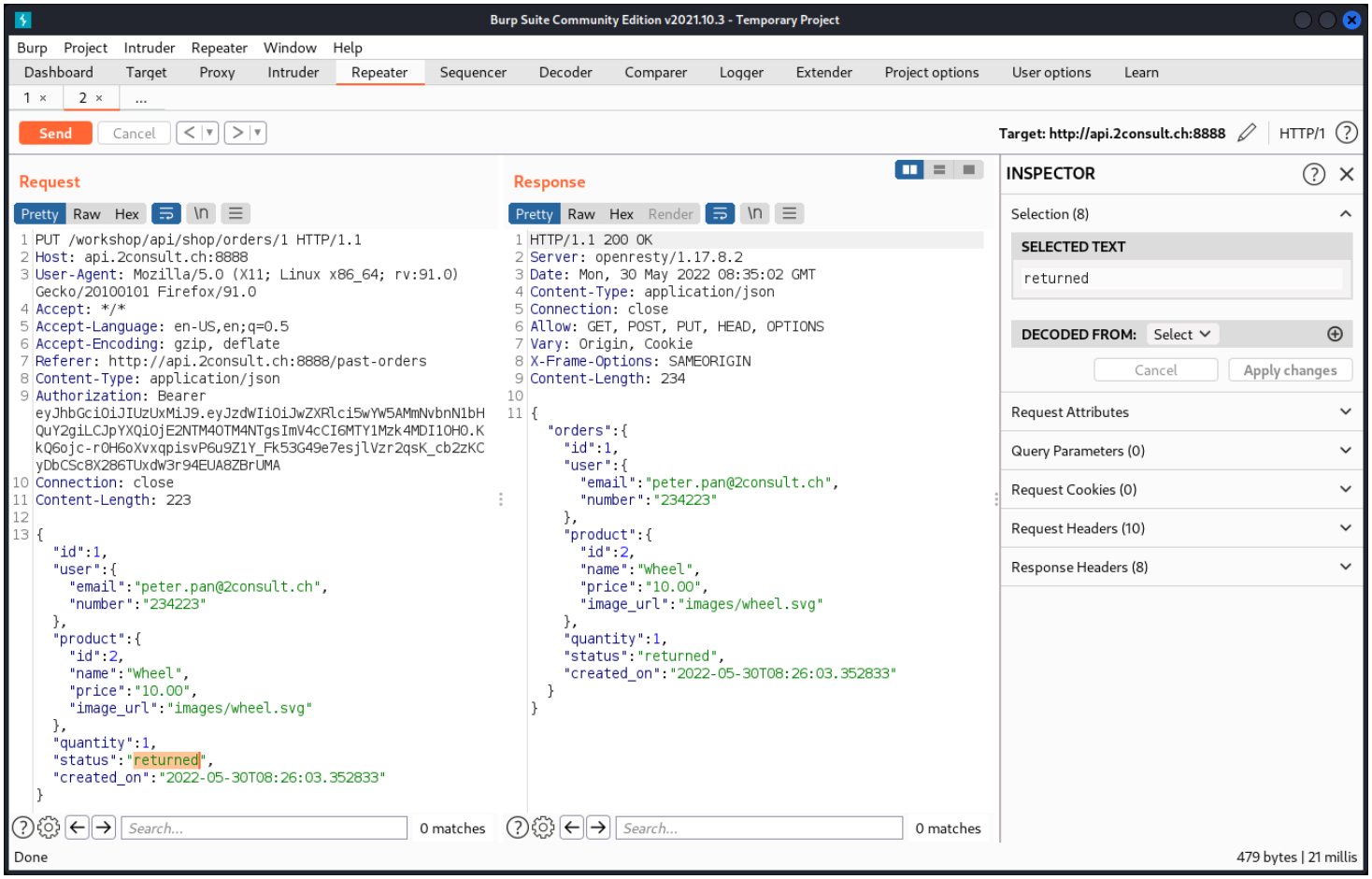
Manchmal setzen Programmierer wie in der crAPI statt auf einfache Nummern auf Universally Unique Identifier (UUID): zufällige, nicht erratbare Kennungen. Sie sind zwar eine gute Maßnahme zur Verteidigung in der Tiefe, verlassen darf man sich aber auf die vermeintliche Unerratbarkeit nicht. Angreifer finden UUIDs durch einen Informationsabfluss heraus, wie später beschrieben. APIs müssen in jeder Funktion jeden Zugriff prüfen, am besten mit einer zentralen Komponente für Zugriffsentscheidungen. Hier hilft eine Policy Engine wie Open Policy Agent (OPA). Entwickler prüfen eine zentrale Autorisierungsrichtlinie in Audits einfacher und verlässlicher selbst oder geben sie an einen Dritten, anstelle handgestrickter, auf verschiedenen Komponenten verteilter Prüfungen.

Abseits von Codereviews, die in einer Funktion das Fehlen einer zentralen Authentifizierungs- und Autorisierungskomponente aufdecken, sind Autorisierungsschwachstellen besonders durch manuelle Tests verlässlich zu finden. Tipps für Sicherheitstester: In einer Anfrage, die ein legitimer Client sendet (`GET /accounts/id/peterpan@2consult.ch`), lässt sich ein numerischer Wert einfügen, den manche Autorisierungsmechanismen ebenfalls akzeptieren: `GET /accounts/id/1953`. Zudem vergessen Entwickler bei Zugriffsprüfungen oft statische Ressourcen wie Bilder oder Dokumente, selbst wenn direkte API-Funktionen sicher sind (Beiträge zu BOLA-Lücken und wie man Autorisierungsfehler auf Objektebene aufdeckt siehe ix.de/zhxu).

## Fehlerhafte Autorisierung auf Funktionsebene – API5

Analog zur fehlerhaften Autorisierung auf Objektebene kommt es bei Funktionen zu Lücken, wenn die API Berechtigungen authentifizierter Benutzer unzureichend prüft. Ein Angreifer ändert den Pfad oder die Methode einer Anfrage, um einen privilegierten Endpunkt zu erreichen.

Schnittstellen sind wegen des REST-Paradigmas strukturierter aufgebaut als klassische Webanwendungen und der Zugriff auf bestimmte Pfade und Funktionen ist einfacher zu erraten: Wenn die Anfrage `GET /api/users/123` Informationen über einen Benutzer ausgibt, ist wahrscheinlich auch `PUT /api/users/123` implementiert, um ein Benutzerattribut zu




**Eine Massenzuweisung über einen erratbaren Endpunkt erlaubt, den Status einer Retoure auf eingeliefert zu setzen, um unberechtigt Geld zurückzubekommen (Abb. 5).**

ändern. Begünstigt wird diese Schwachstelle durch Clientanwendungen, die nur Links und Menüoptionen anzeigen, die der Anwender gerade ausführen darf, ohne dass die API serverseitig den Zugriff verhindert. Aber: Die Zugriffskontrolle auf Präsentationsebene funktioniert nicht.

Besonders interessant für Angreifer sind Funktionen, die sich mit sensiblen Daten befassen oder Ressourcen anderer Gruppen verwalten, und Administratives wie Benutzerkontenverwaltung. Finden sie Anfragen, mit denen sie ein Konto oder die Mitgliedschaft in einer Gruppe verändern, tragen sie sich als Mitglied der Buchhaltergruppe ein und greifen dann mit deren Rechten auf andere Ressourcen zu. Der einfachste Weg für Tester, die Lücke aufzuspüren, ist, die API-Dokumentation nach administrativen Funktionen zu durchforsten oder die Schnittstelle zunächst mit einem legitimen Adminkonto zu verwenden und anschließend als niedrig privilegierter Benutzer die gleichen Anfragen zu senden.

Entwickler schützen sich gegen fehlerhafte Autorisierung auf Funktions- und Objektebene: Das Backend prüft für jede Anfrage, ob sie erlaubt ist oder nicht, idealerweise mit einer zentralen Komponente. Ein Zugriff ist im Standard verboten und nur für berechtigte Gruppen erlaubt. Als weitere Maßnahmen teilt man normale und administrative Endpunkte in unterschiedliche Schnittstellen auf und härtet die Admin-API durch zusätzlichen Schutz wie eine beiderseitige Authentifizierung über TLS (mTLS) oder IP-Filter.

## Preisgabe sensibler Daten – API3

Backend-Architekten neigen dazu, in Schnittstellen alle Daten auszugeben, die ein Client womöglich braucht. Sie verlassen sich auf die Frontend-Entwickler, die Ergebnisse passend filtern und weiterverarbeiten. Das REST-Paradigma und API-Frameworks ermutigen dazu, generische Funktionen wie `toJSON()` einzusetzen, die Objekte mit allen Attributen ausspielen. Nutzt man legitime Clients wie SPA oder mobile Apps, ist diese Schwachstelle unsichtbar.

Wie beschrieben sehen böswillige Benutzer sich alle Rückmeldungen der Schnittstelle im Klartext an. So gelangen sie in einem sozialen Netzwerk an das Geburtsdatum statt nur an das Alter eines Benutzers, wenn das Datum auf dem Client weiterverarbeitet wird. In kritischeren Fällen wie einer Kundendienst-App enthalten API-Antworten für Servicemitarbeiter Hinweise auf den Umsatz, den das Unternehmen mit dem Kunden erzielt, oder gar Anmeldeinformationen wie gehashte Passwörter und Zugriffstoken. Davon war der Fahrdienst Uber betroffen, ein bösartiger Angreifer hätte sämtliche Benutzerkonten übernehmen können. Die API der Fitnessräder von Peloton gab zu ihren Nutzern Geschlecht, Alter und Wohnort preis (Links zu allen Storys siehe ix.de/zhxu).

Angreifer und Verteidiger finden Informationsabflüsse einfach, wenn sie sich mit geeigneten Werkzeugen die vollständigen Antworten der Schnittstelle anse-

hen und überlegen, ob jedes Datum im aktuellen Kontext angezeigt werden sollte. Navigiert man in der verwundbaren crAPI über „Community“ zum Forum, sieht man im parallel laufenden Burp nicht nur die Namen der anderen Benutzer, sondern auch ihre E-Mail-Adresse und die ID ihres Fahrzeugs (Abbildung 4).

Entwickler dürfen sich niemals auf clientseitige Filterung verlassen, sondern müssen sich bewusst sein, dass Angreifer vollen Einblick in Anfragen und Antworten aller API-Endpunkte haben. Eine sichere Schnittstelle gibt nur die Daten zurück, die der aktuelle Benutzer und Client wirklich benötigen.

Um die Herausforderung grundlegend anzugehen, sollten Architekten Daten in ihrer Anwendung von Anfang an klassifizieren. Vom Erstellen über das Verarbeiten bis zum Vernichten sollte man sich überlegen, welche Daten besonders schützenswert sind, wo und wie sie gespeichert und übertragen werden und wer darauf Zugriff hat. Besondere Vorsicht gilt bei Daten, auf die man unauthentisiert oder mit einer unprivilegierten Rolle zugreift.

Bei API-Audits sehen Pentester häufig Fehlermeldungen, die zu viel preisgeben, etwa die Version einer Komponente. Funktionen, für die ein Benutzer tendenziell vertrauliche Daten eingibt, sind ebenfalls interessant: Benutzernamen bei der Anmeldung, oft die E-Mail-Adresse, beim Zahlen individuelle Gutscheine und beim Registrieren eine Mitgliedsnummer. Weil die Endpunkte zum Beantworten solcher Anfragen korrekte Daten brauchen, reagieren sie im Fehlerfall oft mit der Nachricht, die eingegebenen Daten seien falsch. Ein Klassiker beim Log-in ist die Rückmeldung „Dieser Benutzer existiert nicht“. Angreifer erraten gültige Eingaben durch wiederholtes Ausprobieren und verwenden sie für weitere Attacken. Eine Liste vorhandener Benutzerkonten hilft beim Angreifen des Log-ins.

Endpunkte antworten mit einer generischen Fehlermeldung, aus der nicht hervorgeht, ob die Eingabe gültig war. Die Antwortzeit darf sich ebenso nicht messbar unterscheiden. Selbst bei gut gesicherten Log-ins dauert der Versuch für einen gültigen Benutzernamen, sich mit einem fehlerhaften Passwort anzumelden oder eine Kennwort-zurücksetzen-Mail anzufordern, oft länger als für einen ungültigen, weil das Backend die Zugangsdaten abgleicht oder synchron eine E-Mail sendet, während es bei einem nicht existierenden Konto gleich abbricht. Solche Routinen sollten im Fehlerfall gleich lange laufen wie im Erfolgsfall.

## Massenzuweisung – API6

Während das API3-Risiko zu viele Daten ausgibt, nimmt die Massenzuweisung zu viele Daten entgegen. Bösartige Nutzer der API packen mehr Parameter als vorgesehen in ihre Anfrage, die kein gutartiger Client senden würde. Spielt die Anwendung alle Eingaben ungeprüft ins Datenmodell zurück, können Angreifer Eigenschaften überschreiben, Daten manipulieren, Sicherheitsmechanismen umgehen und sich erhöhte Rechte verschaffen. Betroffen war die Cloud-Registry Harbor, bei der sich Benutzer zum Administrator machen konnten (siehe ix.de/zhxu). Diese Lücke ist schwieriger zu entdecken als die Preisgabe sensibler Daten, weil legitime Clients bei betroffenen Endpunkten keine Werte übergeben.

Doch die Namen solcher Parameter lassen sich herausfinden: Beim API3-Risiko enthält eine Antwort überflüssige Daten und verrät Parameternamen. Ein Sicherheitstester stöbert beim Lesen der API-Dokumentation interessante Parameter auf, die er bei einem anderen Endpunkt als dem laut Dokumentation vorgesehenen ausprobiert, etwa Rollenname oder Passwort. Das quelloffene Werkzeug Arjun findet automatisiert Abfrageparameter, die das Backend akzeptiert, aber kein Client sendet. Das Wiki des GitHub-Projekts beschreibt, wie man das Tool installiert und nutzt. Entwickler halten im Quelltext ihrer Anwendung nach Mustern wie `new User(req.body)` Ausschau, die eingehende Daten ohne Argwohn an interne Objekte binden.

Massenzuweisungen betrafen schon klassische Webanwendungen, sind in Schnittstellen aber wegen der offenliegenden Implementierung einfacher zu finden. Manche API-Frameworks wandeln JSON-Parameter in Anfragen automatisch in Objekteigenschaften um. Werden die JSON-Felder nicht beschränkt, können böswillige Benutzer das Feld für den Passwort-Hash und die Rolle an einem Benutzerobjekt oder gar den Preis für ein Produkt ändern. Auf der Suche nach weniger auffälligen Arten der Massenzuweisung ändern Tester die Encodierung der Payload, senden statt JSON URL-encodierte Schlüssel-Wert-Paare oder wechseln die HTTP-Methode von `POST` auf `PATCH` oder `PUT`.

Im crAPI-Shop kauft man durch den Klick auf Buy Autozubehör wie Reifen. Unter „Past Orders“ sieht man seine Bestellungen und löst über die Schaltfläche Return eine Retoure aus, bei der man einen QR-Code bei einem virtuellen Paketdienstleister aufgeben muss. Über eine Massenzuweisung erhält man unberechtigt Geld zurück, ohne das Produkt retourniert zu haben. In Burp klickt man unter „Proxy“ in „HTTP history“ mit rechts auf die Anfrage `GET /workshop/api/shop/orders/all` und wählt den Eintrag „Send to Repeater“. Dort bearbeitet man die Anfrage beliebig und schickt sie über „Send“ erneut ab. Ändert man das Verb auf `PUT` und die Anfrage wie in Abbildung 5, setzt man den Rücksendungsstatus selbst auf „returned“.

Zuverlässig vor Massenzuweisung schützen sich Entwickler, indem sie eingehende Daten nicht automatisch an Objekte binden

und erwartete Parameter definieren. Ein Ansatz dafür können Datentransferobjekte (DTO) sein. Idealerweise werden nicht nur die Parameter selbst beschränkt, sondern ihre Werte möglichst einer strengen Eingabevalidierung unterzogen: Was nicht im JSON-Schema spezifiziert ist, wird verworfen. Schon beim Entwurf der API können Architekten Schemata, Typen und Muster definieren, die ihre API in Anfragen akzeptiert und gegen die sie zur Laufzeit prüft. Wenn möglich sollte man nicht mehr zu verändernde Eigenschaften als nur lesbar definieren.

## Fehlende Ressourcen- und Ratenbegrenzung – API4

Jede API muss Mechanismen gegen Brute-Force-Angriffe enthalten, mindestens an kritischen Stellen wie dem Login. Andernfalls versuchen Angreifer, sich mit einem ihnen bekannten Benutzernamen oder naheliegenden Standardaccounts wie admin und Passwörtern aus einer Passwortliste so lange anzumelden, bis sie die richtige Kombination gefunden haben.

Ein Schutz ist Ratenbegrenzung (Rate Limiting): Wenn von einer einzelnen IP-Adresse, im Kontext eines Kontos oder innerhalb einer Sitzung in einer bestimmten Zeit zu viele Anfragen gestellt werden, ignoriert die Anwendung weitere Versuche und antwortet mit dem HTTP-Statuscode `429 Too Many Requests`. Meist senden Endpunkte mit Ratenbegrenzung Antwortheader wie `X-RateLimit-Limit` und `X-RateLimit-Remaining`, die über Grenzen und verbleibende Versuche informieren.

Oft haben APIs keine Ratenbegrenzung. Selbst wenn ein Rate-Limiting-Mechanismus implementiert ist, weil dafür eine Securitybibliothek genutzt wird, umgehen böswillige Benutzer ihn manchmal. Header wie X-Forwarded-For werden normalerweise von Infrastrukturkomponenten wie Reverse-Proxys gesetzt, die damit die eigentliche Quell-IP-Adresse einer Anfrage mitteilen. Nichts hindert Angreifer daran, einen solchen Header in einer Anfrage selbst einzufügen, etwa `X-Forwarded-For: 127.0.0.1` in einem Interception-Proxy. Vertraut die API diesen Angaben, lässt sich die Ratenbegrenzung umgehen. So ist die Anwendung durch Denial-of-Service- (DoS) und Brute-Force-Angriffe verwundbar.

Anfällig für Manipulationen dieser Art sind ebenfalls andere, auf die Quelle einer Anfrage bezogene Header wie `True-Client-IP`. Durch blindes Vertrauen in Header mit Bezug zum Ziel der Anfrage, etwa `X-Forwarded-Host`, entstehen Schwachstellen wie Web Cache Poisoning, die API-Antworten für nachfolgende Benutzer vergiften.

Fehlende Ratenbegrenzung findet man recht schnell, wenn die API trotz einer Flut von Anfragen weiterhin antwortet, statt den Missbrauch mit einem 429-Statuscode zu quittieren. Testen sollte man unterschiedliche Authentifizierungsfälle: als angemeldeter Benutzer, unangemeldet oder mit abgelaufenem Konto. Entdeckt werden Lücken mit umgehbarer Ratenbegrenzung durch automatisierte und manuelle Tests, bei denen man entsprechende Header gezielt einfügt und beobachtet, ob sich das Antwortverhalten ändert.

Entwickler sollten eine Ratenbegrenzung so eng wie möglich an die Applikationslogik und darin vorhandene Angriffspunkte knüpfen: Ein auf die IP-Adresse bezogenes Rate Limiting auf dem Anmeldeendpunkt ist gut, besser ist jedoch, wenn sich die Begrenzung auf die Benutzeridentität bezieht oder in anderem Kontext auf das Zugriffstoken.

Fehlende Limits bei Nutzdaten entwickeln sich zum Problem, wenn ein Endpunkt Bilder beliebiger Größe entgegennimmt und unbeirrt versucht, eine Gigabyte-große Abbildung in kleinere Vorschaubilder herunterzurechnen. Ist der Arbeitsspeicher erschöpft, ist die Schnittstelle lahmgelegt. Wenn ein Endpunkt einen Grenzwert als Parameter akzeptiert, führt allein die Anfrage `/api/nachrichten?limit=99999999` womöglich zu einem DoS auf Anwendungsebene. Betreiber einer API sollten die erlaubte Größe einer Anfrage und ihrer einzelnen Felder beschränken.

Im crAPI-Dashboard kann man über die Schaltfläche „Contact Mechanic“ seinen virtuellen Mechaniker kontaktieren. Dazu füllt man zunächst das Formular aus, sendet es ab, schickt aus der Burp-Proxy-Historie die Anfrage `POST /workshop/api/merchant/contact_mechanic` an den Repeater, verändert dort in der Anfrage folgende Werte, sendet sie und hat so DoS auf Anwendungsebene (Layer 7) ausgelöst:

```
"repeat_request_if_failed":true
"number_of_repeats":999999
```

Raten- und Ressourcenbegrenzungen richten Entwickler mit Bibliotheken oder korrekt konfigurierten Infrastrukturkomponenten ein. Nutzt man Container, lassen sich dort Arbeitsspeicherbedarf und Prozessorlast beschränken. Private Endpunkte, die nicht von jeder beliebigen IP-Adresse erreichbar sein müssen, schützen Verteidiger durch Allowlists. Generell gilt: An einem Endpunkt eintreffenden Daten sollten Entwickler an keiner Stelle blind vertrauen, auch nicht in Headern und Dateien.

## Fazit

APIs sind die Basis von mobilen Apps, Single-Page-Webanwendungen und automatisiertem Datenaustausch zwischen Maschinen. Sie bieten eine unterschätzte Angriffsfläche, deren unzureichende Absicherung zu schlagzeilenträchtigen Datenverlusten führt. Im Bewusstsein vieler Entscheider, Architektinnen und Entwickler ist noch nicht präsent, dass Schnittstellen von ähnlichen Sicherheitslücken wie klassische Webanwendungen betroffen sind.

Die API-Liste der OWASP ist nach eigener Zielsetzung kein Sicherheitsstandard. Die Top-10-Verantwortlichen sehen ihre Liste als Awareness-Dokument, das einem breiten Publikum von Sicherheitsverantwortlichen über Projektmanagerinnen und Entwickler bis zu Testern wesentliche Schwachstellen zugänglich macht. Deren Ursachen sollte man auf jeden Fall vermeiden, das Fehlen von Schutzmaßnahmen dagegen grenzt an Fahrlässigkeit.

Sicherheit muss möglichst früh im Lebenszyklus ansetzen, idealerweise von Anfang an berücksichtigt werden und sich auf die gesamte Anwendung beziehen. Klassische Produkte wie Web Application Firewalls (WAF), API-Gateways und spezifische für Schnittstellen vermarktete Produkte härten zusätzlich, sind aber kein alleiniger Schutz.

Über Injection, Fehlkonfiguration, unzureichendes Protokollieren und Überwachen, die Punkte API7, API8 und API10 der OWASP API Top 10, informiert der Artikel „Schwachstellen in APIs aufspüren“ ab Seite 64. (nb@ix.de)

### Quellen

[1] Tobias Glemser; Zwo, eins, Risiko ...; OWASP Top 10 in neuer Version; iX 2/2022, S. 86

[2] Alle Links zu weiterführenden Beiträgen: ix.de/zhxu

**Frank Ully**

ist Head of Research der Oneconsult Deutschland AG in München. Er beschäftigt sich mit aktuellen Themen der offensiven IT-Sicherheit.