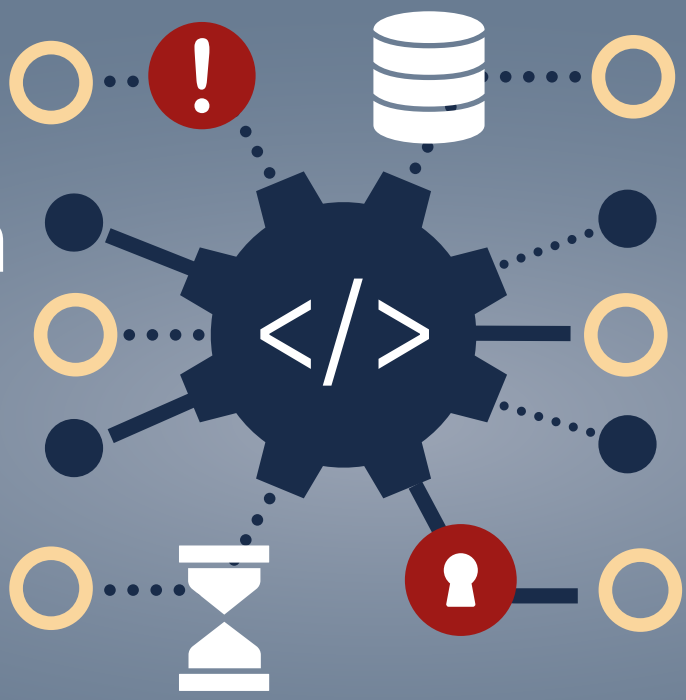
Schwachstellen in APIs aufspüren



Injections und Fehlkonfigurationen machen APIs angreifbar. Werkzeuge helfen, die Schwachstellen zu finden.

Von Frank Ully

Während sich der vorige Artikel mit fehlerhafter Authentifizierung und Autorisierung, falscher Inventarisierung und Massenzuweisung beschäftigte, geht es nun um die Gefahren, die von Injections, Fehlkonfigurationen und unzureichendem Überwachen und Protokollieren ausgehen. Passende Werkzeuge helfen, Sicherheitslücken bereits während der Entwicklung aufzuspüren.

Injektionsschwachstellen sind eine der bekanntesten Sicherheitslücken und liegen in den OWASP Web Top 10 regelmäßig auf den ersten Plätzen, bereits 1998 beschrieb ein Hackermagazin SQL Injections. Die Autoren der API Top 10 verwiesen Injections in ihrer Liste auf einen der hinteren Plätze, da diese bei Schnittstellen seltener auftreten. Designmuster wie objektrationale Abbildungen (Object Relational Mapping, ORM) und moderne Frameworks abstrahieren die Gefahr von den Entwicklern. Dennoch sollte ihnen die Grundursache von Injektionsschwachstellen bewusst sein: Übergeben sie Daten an einen Interpreter und der kann nicht klar zwischen Code und Daten unterscheiden, kann ein Angreifer dem Interpreter bösartige Daten so unterjubeln, dass er sie als Code interpretiert. Bei SQL Injections und Injections in NoSQL-Anfragen werden Befehle in die Datenbankabfrage eingeschleust, bei einer Befehlsinjektion Betriebssystemkommandos oder Code der jeweiligen Programmiersprache. Grundlage ist immer fehlender Kontext, der den jeweiligen Interpreter dazu verleitet, einen beabsichtigten Befehl und manipulierte Daten zu vermengen, die ebenfalls als Code bei ihm ankommen.

Was man Injections entgegensetzen kann

Wirksam gegen Injections ist Parametrisierung, die Befehle und Daten mit Platzhaltern trennt. Bei SQL werden meist vorbereitete Anweisungen (Prepared Statements) genutzt. Betriebssystemkommandos kann man ebenso aufrufen. Sämtlichen Eingaben – selbst von eigenen Systemen und internen Schnittstellen – sollten Entwickler grundsätzlich misstrauen und sie serverseitig validieren. Zu prüfen sind Header, Dateitypen, das HTTP-Verb, der Content Type und alle Parameter, ganz gleich, ob sie sich in einem JSON-Objekt oder der URL befinden. Die erwartete Methode definieren Entwickler pro Endpunkt und beschränken den akzeptierten Content Type bei JSON-Endpunkten auf `application/json`.

Gibt es ein API-Schema, validiert man eingehende Anfragen daraufhin und lehnt Unerwartetes ab. Oft wird Eingabevalidierung als vollständiger Schutz gegen Injections gesehen, was allerdings nicht stimmt: Eine nach RFC valide E-Mail-Adresse kann SQL-Befehle enthalten. So wichtig Eingabevalidierung ist, sollten sich Verteidiger nie allein auf sie verlassen.

Webschwachstellenscanner wie der kommerzielle Burp Scanner, OWASP ZAP und sqlmap finden solche Lücken. Das GitHub-Wiki von sqlmap beschreibt, wie man es nutzt. Schwieriger zu entdecken sind SQL Injections zweiten Grades. Sie treten auf, wenn Eingaben mit einem SQL-Injection-String am ursprünglichen Endpunkt keine Codeausführung verursacht haben, weil sie sicher verarbeitet wurden. Fragt ein Entwickler dieselben Daten an anderer Stelle seiner Anwendung aus der Datenbank ab und baut sie

iX-TRACT

- Sicherheitsrelevante Fehlkonfigurationen treten nicht nur bei APIs auf, sie zählen zu den häufigsten Lücken bei Web-Apps. Sie zu vermeiden ist knifflig, weil sie mehrere Teile des Anwendungslebenszyklus betreffen.
- Injection-Schwachstellen sind bei Schnittstellen anders als bei Webanwendungen nicht sehr häufig, Entwickler müssen sie trotzdem im Blick haben.
- Systematisches Loggen sicherheitsrelevanter Ereignisse hilft, Angriffe frühzeitig zu erkennen und zu unterbinden.
- Beim Auffinden und Beheben von Sicherheitslücken in APIs helfen zahlreiche kostenfreie Werkzeuge.

unsicher in eine SQL-Abfrage ein, weil er hier nicht mit bösartigen Benutzereingaben rechnet, besteht wieder eine Injection. Programmierer können trotz moderner Bibliotheken noch für Injections anfälligen Code schreiben, wenn sie nicht vertrauenswürdige Daten direkt an parametrisierte Abfragen hängen, statt die Funktionen wie vorgesehen zu verwenden.

Ein Werkzeug zur statischen Codeanalyse (Static Application Security Testing, SAST) wie das als Open Source bereitstehende Semgrep fahndet im Quelltext nach unsicheren Interpreteraufrufen, die zwischen Code und Daten nicht sauber trennen. Es deckt weitere unsichere Muster auf, darunter angreifbare Aufrufe von JWT-Bibliotheken. Entwickler und Auditoren ergänzen es um eigene Regeln. In der Installation in Listing 1 vereinfacht der Python-Helfer pipx das Verwalten unterschiedlicher Python-Umgebungen und erlaubt, Kommandozeilentools einfach aufzurufen.

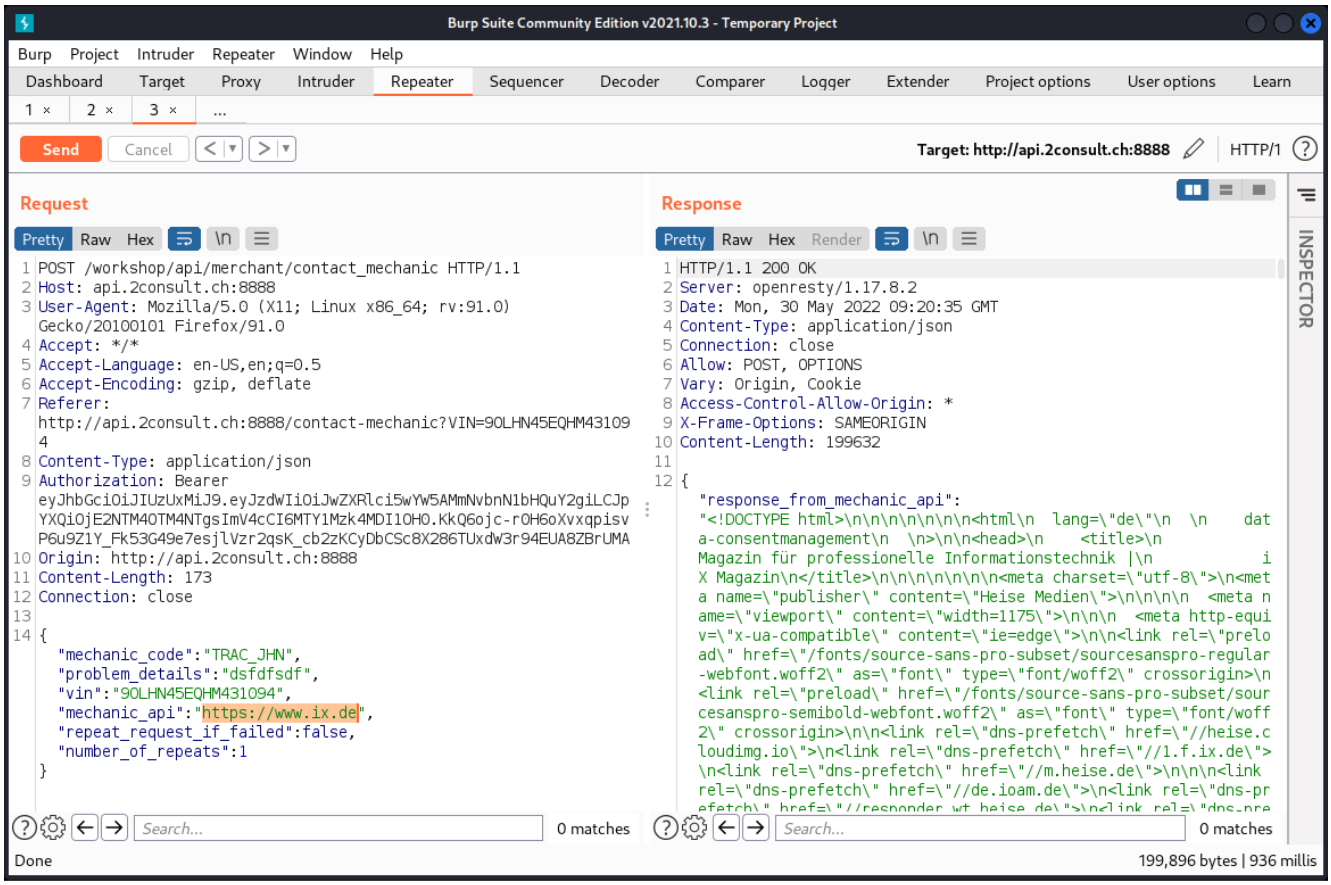
Listing 1: Semgrep findet Schwachstellen und Fehlkonfigurationen

```
$ sudo apt install -y pipx
$ pipx ensurepath && source ~/.profile
$ pipx install semgrep

$ cd ~/crAPI
$ semgrep --config=auto

Fetching rules from https://semgrep.dev/registry.

Scanning across multiple languages:
    <multilang> |  89 rules × 674 files
           java | 165 rules ×  72 files
             js | 302 rules ×  65 files
           yaml |  33 rules ×  58 files
         python | 312 rules ×  43 files
             go | 129 rules ×  16 files
           bash |   6 rules ×  13 files
           json |   7 rules ×   6 files
     dockerfile |   5 rules ×   5 files

Ran 1374 rules on 338 files: 86 findings.
```

Gefälschte Anfragen: Server-Side Request Forgery

Server-Side Request Forgery (SSRF) ist kein Punkt der API Top 10, doch Pentester entdecken die Schwachstelle häufig bei Kundenaudits. Weil sie recht verbreitet ist, steht sie in den aktuellen Web Top 10. ProxyShell und ProxyLogon, die kritischen Bugs in Microsofts Mailserver Exchange im Jahr 2021, hatten eine SSRF-Komponente (siehe ix.de/zb7g).

Die Ursache von SSRF ist, dass eine Anwendung eine Anfrage ausführt, die als Folge eine interne oder externe Ressource aufruft und deren Ziel der Benutzer ganz oder teilweise vorgibt. Das ist nichts Ungewöhnliches, viele APIs nehmen in der Anfrage direkt eine URL entgegen und tun etwas damit, prüfen etwa den angegebenen Link oder erzeugen aus dem Bildpfad eine Vorschau. In anderen Fällen leiten sie Teile der Anfrage zur Authentifizierung an Drittsysteme weiter, an ein Zahlungsgateway oder eine externe API.



Eine manipulierbare URL in einer Anfrage ermöglicht Server-Side Request Forgery (SSRF) und bringt die serverseitige Anwendung dazu, Anfragen an ein unbeabsichtigtes Ziel zu stellen (Abb. 1).

Dabei bestehen Vertrauensbeziehungen, weil Server sich selbst und anderen Systemen im selben Netzwerk vertrauen. Nutzen Angreifer dies aus, können sie Firewalls umgehen und aus dem Internet Dienste erreichen, die eigentlich nur auf dem lokalen System oder im internen Netz verfügbar sind, das entfernte Netzwerk scannen und sensible interne Daten auslesen. Auch reflektierte XSS-Angriffe und das Ausführen von Code auf einem Server sind möglich. Als kritisch erweist sich SSRF in einer Cloud und in Kubernetes-Clustern: Dort werden über Metadaten-URLs, die mit http://169.254.169.254/ beginnen, Zugangsdaten für die Cloud-Umgebung abgefragt.

Entdeckt wird SSRF zum Teil von automatisierten Scannern wie ZAP oder dem kostenpflichtigen Burp Scanner. Tester finden SSRF manuell, indem sie prüfen, ob ein Wert einer Anfrage ganz oder teilweise an andere Systeme weitergeleitet wird und dabei manipulierbar ist. Admins mit legitimem Zugriff auf die Anwendungsserver entdecken in ausgehenden Firewalllogs ungewöhnliche Ziele von Anfragen.

In der bewusst anfällig konstruierten crAPI der OWASP enthält der Contact-Mechanic-Endpunkt, aus dem vorigen Artikel bekannt wegen fehlender Ressourcenbegrenzung, eine URL für die Mechanic-Backend-API. Ersetzt man in Burp den ursprünglichen Wert von `mechanic_api` durch `https://www.ix.de`, hat man die serverseitige Schnittstelle zu einem Aufruf der iX-Website verleitet (siehe Abbildung 1).

Allow-Listen zum Prüfen nutzen

Einen nur lückenhaften Schutz bietet Denylisting, das Blockieren mutmaßlich bösartiger Eingaben, die etwa die Metadatenadresse 169.254.169.254 enthalten. Jedoch lassen sich IP-Adressen unterschiedlich darstellen: hexadezimal als 0xA9FEA9FE oder 2852039166 in Dezimalnotation. Eine vom Angreifer kontrollierte Domäne löst einfach auf die Metadaten-IP-Adresse auf. Verlassen sich Entwickler beim Filtern auf IP-Adressen, müssen sie die Adressen vor einer Sicherheitsentscheidung normalisieren; dazu gibt es für jede Plattform passende Bibliotheken. Anschließend prüfen sie die IP-Adresse am besten gegen eine Allow-Liste. Generell bietet Denylisting keinen Schutz gegen die meisten Angriffe.

Akzeptiert ein Endpunkt eine vollständige URL und ein anderer Ansatz (Vorauswahl oder Beschränkung auf den Domänennamen) ist nicht möglich, können Programmierer ihre Clients so schreiben, dass sie die URL in die einzelnen Komponenten Schema, Domäne, Pfad und Port splitten. Ein API-Endpunkt nimmt die Bestandteile entgegen und prüft sie.

Zur Prävention von SSRF validiert man die vom Benutzer oder Angreifer gelieferten Daten so, dass sie nur erwartete Werte akzeptieren. Idealerweise erfolgt das mit Allowlisting: Erlaubt ist nur der Zugriff auf eine Liste mit vorgegebenen Ressourcen, nicht nur bezogen auf Hostnamen, sondern auch auf das Protokoll (nur https ist erlaubt) und den Port. Der Webclient der Schnittstelle darf keinen Weiterleitungen folgen. Als weiterer Schutz dienen ausgehende Firewallregeln, die einschränken, mit welchen Servern und Ports die API kommunizieren darf.

Schließlich sollte man interne Dienste wie Monitoringtools, Adminoberflächen und nachgelagerte Schnittstellen immer mit Zugangsdaten schützen. Verteidiger beschränken durch eingehende Regeln in einer Firewall, von welchen Adressbereichen sie eine Kommunikation akzeptieren. Nicht von jedem internen System muss man Verbindungen zulassen.

Listing 2: Sicherheitsheader am Endpunkt setzen

```
Strict-Transport-Security: max-age 3153600; includeSubDomains
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
Content-Security-Policy: frame-ancestors 'none'; sandbox; default-src 'none'
```

Sicherheitsrelevante Fehlkonfigurationen – API7

Sicherheit hängt ebenfalls von der Konfiguration ab. Setzt man unsichere Standardeinstellungen und Komponenten ein, für die es bekannte Schwachstellen gibt, hilft es nicht, die Anwendungslogik sicher zu entwickeln. Diese Schwachstellenkategorie tritt nicht nur bei APIs auf, sie ist eine der häufigsten Lücken bei Web-Apps. Ihre Auswirkungen sind gravierend, wenn ein System komplett kompromittiert wird.

Wer unnötige Systeme (siehe API9 im vorigen Artikel) abschaltet, nicht benötigte Ports schließt, Zugriff auf administrative Funktionen beschränkt, nicht verwendete Funktionen in seinem Anwendungsserver deaktiviert sowie Betriebssysteme und darunterliegende Komponenten regelmäßig patcht, hat die Angriffsfläche seiner Lösung schon verringert. Je mehr die Anwendungsarchitektur einzelne Module voneinander abschottet, etwa die Authentifizierung von der Businesslogik, desto geringer wirken sich Lücken in einzelnen Brandabschnitten aus. Nirgendwo sollten noch Standardzugangsdaten (Passwort admin) gelten, mit denen Angreifer zur Verwaltungsoberfläche des Anwendungsservers vordringen oder sich bei administrativen Funktionen anmelden dürfen.

Anders als bei anderen Lücken ist es schwieriger, sicherheitsrelevante Fehlkonfigurationen zu vermeiden. Der Bereich bezieht sich auf mehr Abschnitte

Listing 3: TruffleHog fahndet in Quelltextrepositorys nach API-Schlüsseln und anderen Geheimnissen

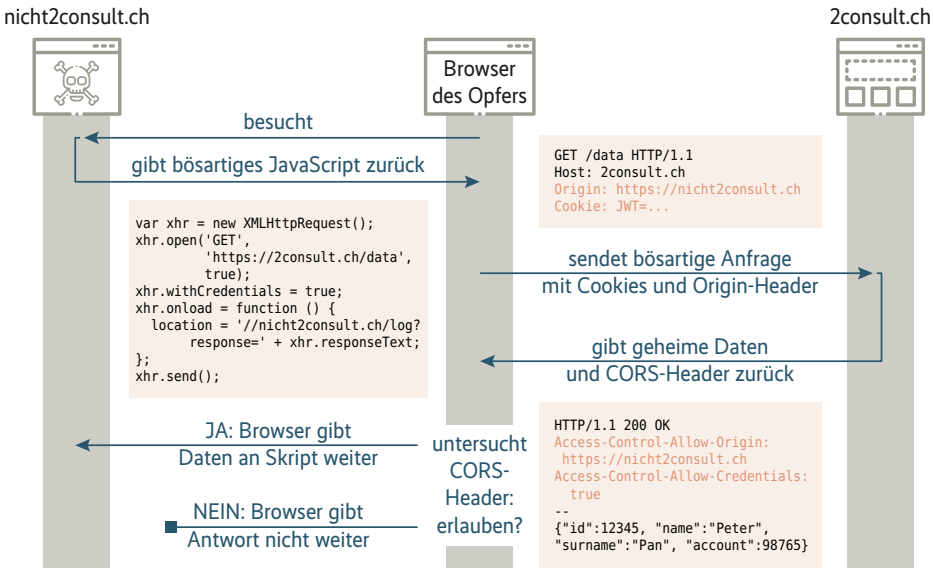
```
$ sudo docker run -it -v "$PWD:/pwd" trufflesecurity/trufflehog:latest github --repo https://github.com/trufflesecurity/test_keys

Found verified result
Detector Type: AWS
Raw result: sS/UtboQeUMjWfieHO41OqHMwPvfT+rkXl32Al70
Timestamp: 2022-02-15 15:17:18 -0800 -0800
Line: 8
Link: https://github.com/trufflesecurity/test_keys/blob/c0c91ecaa7661e70ee9d6d6cb0ba366e2dc215c3/keys
Repository: https://github.com/trufflesecurity/test_keys.git
Commit: c0c91ecaa7661e70ee9d6d6cb0ba366e2dc215c3
Email: dxa4481@rit.edu
File: keys
```

des Anwendungslebenszyklus und liegt in der Verantwortung mehrerer Rollen: Architektinnen, Entwickler und Admins sind gefordert.

Verteidiger fahren schnelle Gewinne ein, wenn sie frei verfügbare Werkzeuge gezielt nutzen. Bereits von klassischen Webanwendungen bekannte sicherheitsrelevante Header wie eine Content Security Policy (CSP) helfen bei APIs mit einem Browser als Client, dort bestimmte Schutzmechanismen scharf zu schalten. Sie sollten an jedem API-Endpunkt gesetzt sein (Listing 2).

Das Programm drHEADer mahnt fehlende Sicherheitsheader an und scannt auf Wunsch einzelne Endpunkte oder eine Liste:

```
$ pipx install drheader
$ drheader scan single ↵
        http://api.2consult.ch:8888/
```

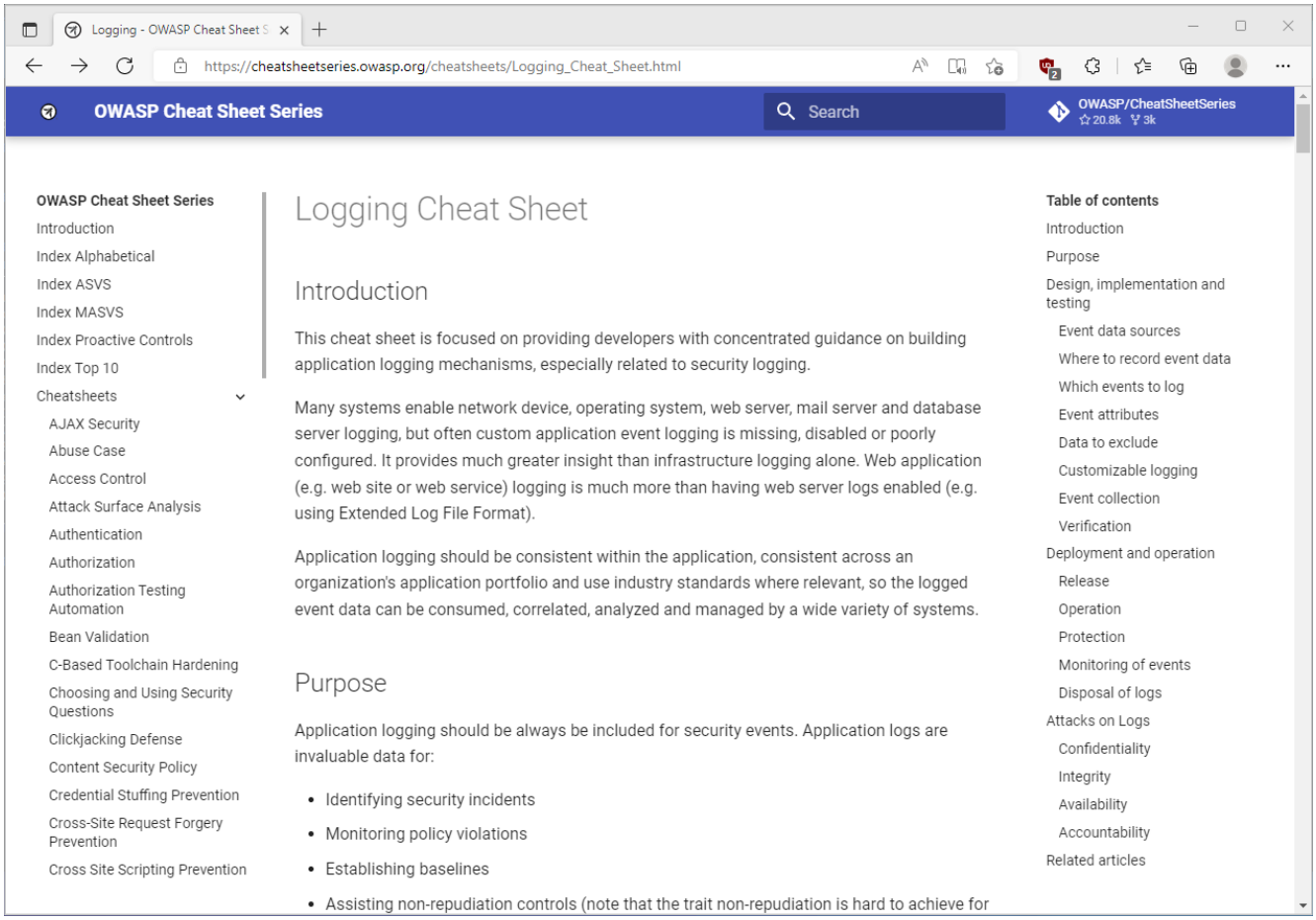


Bei einem Angriff über eine Fehlkonfiguration von Cross-Origin Resource Sharing liest JavaScript Daten aus (Abb. 2).

Fetch-Metadata-Header zur Entscheidungsfindung

Entwicklerinnen und Entwickler können die neueren Fetch-Metadata-Header, die viele Browser setzen, zusätzlich für Sicherheitsentscheidungen nutzen und unzulässige Anfragen verwerfen. So verhindern sie bestimmte Angriffe (siehe Beitrag „Protect your resources from web attacks with Fetch Metadata“ unter ix.de/zb7g).

Ein wahres Trüffelschwein ist TruffleHog, ein Open-Source-Tool zum Aufspüren von Zugangsdaten wie API-Schlüsseln, die in der Historie eines GitHub-Repositorys abgelegt oder in einer JavaScript-Datei vergessen wurden. Das frei



Die zahlreichen Cheat Sheets von OWASP – hier zum Thema Logging – sind eine Wissenssammlung für Verteidiger (Abb. 3).

Kostenfreie API-Prüfwerkzeuge

Name	Beschreibung	Typ	hilft gegen API-Schwachstelle	URL
Arjun	findet Abfrageparameter für Endpunkte	dynamisch	API6	https://github.com/s0md3v/Arjun
Burp Suite Community	kommerzieller Interception-Proxy in einer einschränkten kostenfreien Communityversion	dynamisch	API1-API9	https://portswigger.net/burp/communitydownload
ChopChop	scannt nach unbeabsichtigt erreichbaren Dateien und Ordnern	dynamisch	API7	https://github.com/michelin/ChopChop
CORScanner	prüft Endpunkte auf fehlerhaft gesetzte CORS-Header	dynamisch	API7	https://github.com/chenjj/CORScanner
drHEADer	prüft Endpunkte auf fehlende sicherheitsrelevante Header	dynamisch	API7	https://github.com/Santandersecurityresearch/DrHeader
Dredd	prüft, ob Endpunkte so auf Anfragen antworten wie in der API-Dokumentation beschrieben	dynamisch	API9	https://dredd.org/
Kali Linux	Linux-Distribution mit Programmen für Pentests und digitale Forensik	–	API1-API9	https://www.kali.org/
Kiterunner	für APIs angepasstes Brute-Forcing-Werkzeug, das unbeabsichtigt veröffentlichte Endpunkte findet	dynamisch	API9	https://github.com/assetnote/kiterunner
mitmproxy2swagger	erstellt aus mitgeschnittenem Webverkehr für eine Schnittstelle eine OpenAPI-Spezifikation	dynamisch	API9	https://github.com/alufers/mitmproxy2swagger
OpenVAS	allgemeiner Schwachstellenscanner, der bekannte Bugs und Fehlkonfigurationen in der Infrastruktur aufdecken kann	dynamisch	vor allem API7, auch andere	https://www.openvas.org/
OWASP APICheck	DevSecOps-Werkzeugsammlung für REST-APIs, mit der sich bestehende Auditwerkzeuge in einer Pipeline verbinden lassen	dynamisch und statisch	vor allem API7	https://owasp.org/www-project-apicheck/
OWASP Dependency-Check	findet eingebundene Drittkomponenten wie Bibliotheken mit bekannten Schwachstellen; Open Source	statisch	API7	https://owasp.org/www-project-dependency-check/
OWASP Zed Attack Proxy (ZAP)	Open-Source-Interception-Proxy mit für Webanwendungen und APIs optimiertem Schwachstellenscanner	dynamisch	API1-API9	https://www.zaproxy.org/
Postman	Werkzeug zum Entwerfen, Erstellen, Testen und Iterieren von APIs	dynamisch	API1-API9	https://www.postman.com/
RESTler	Fuzzing-Werkzeug, das REST-APIs zustandsbehaftet prüft	dynamisch	API1-API8	https://github.com/microsoft/restler-fuzzer
Semgrep	semantische Codeanalyse, die Schwachstellen findet und Codestandards durchsetzen kann	statisch	vor allem API7, API8, auch API1 und andere	https://semgrep.dev/
Snyk	findet eingebundene Drittkomponenten wie Bibliotheken mit bekannten Schwachstellen; proprietär, aber kostenlos	statisch	API7	https://snyk.io/
sqlmap	findet automatisiert SQL-Injection-Schwachstellen	dynamisch	API8	https://sqlmap.org/
Syft	erstellt eine Software Bill of Materials (SBOM) von Drittkomponenten	statisch	API7	https://github.com/anchore/syft
JSON Web Token Toolkit v2 (jwt_tool)	prüft, ob JWT ohne bekannte Schwachstellen und ohne schwache Geheimnisse verarbeitet werden; hilft gegen unsichere Authentisierung	dynamisch	API2, API7	https://github.com/ticarpi/jwt_tool
TruffleHog	fahndet in Quelltextrepositorys nach unbeabsichtigt eingecheckten Geheimnissen wie API-Schlüsseln	statisch	API7	https://github.com/trufflesecurity/trufflehog

verfügbare Werkzeug erkennt fast 700 Geheimnisarten und vermeidet so gut es geht falsch positive Funde. Im Beispiel in Listing 3 schnüffelt TruffleHog an einem Testrepo des Anbieters.

ChopChop ist ein DevSecOps-Werkzeug, das Ordner und Dateien im Wurzelverzeichnis einer Web-App findet, die nicht veröffentlicht werden sollten – etwa den .git-Ordner auf einem Produktionssystem. Entwickler untersuchen ihre Quelltextrepositorys mit dem freien OWASP Dependency Check oder mit dem kommerziellen kostenlosen Snyk auf veraltete Komponenten mit bekannten Schwachstellen und aktualisieren diese möglichst rasch. Eine nachhaltige Lösung dabei ist, mit Software Bill of Materials (SBOM) eine Stückliste eingesetzter Drittsoftware zu pflegen, um schnell zu beantworten, ob die eigene Anwendung beispielsweise von Log4Shell betroffen ist, der Ende 2021 ins Gespräch gekommenen Sicherheitslücke der Java-Logbibliothek Log4j. Beim Erzeugen der Liste nehmen Open-Source-Tools wie Syft Arbeit ab.

Die Werkzeuge lassen sich einfach installieren und nutzen, eine Anleitung findet sich in ihren GitHub-Repos. Neben den beschriebenen kostenlosen Helferlein ist ein Blick in den gut gefüllten Werkzeugkasten der kommerziellen DevSecOps-Tools empfehlenswert [1].

Schwachstellenscanner leisten gute Dienste

Es ist unbestreitbar, dass ein Schwachstellenscanner eines der lohnendsten Sicherheitsprodukte ist, die eine Organisation beschaffen kann. Neben kostenfreien Produkten wie OpenVAS gibt es preiswerte kommerzielle Scanner wie Nessus. Hat man ein solches Werkzeug und Admins und Sicherheitsverantwortliche im Haus, die es einsetzen können, oder beauftragt für den Scan einen Sicherheitsdienstleister, sollte man regelmäßig alle aus dem Internet erreichbaren Systeme prüfen.

Browser schützen Webanwendungen und Schnittstellen mit der Same-Origin Policy (SOP). Sie erlaubt clientseitigen Skriptsprachen wie JavaScript nur dann uneingeschränkten Zugriff auf Daten in einer zweiten Webseite, wenn beide vom selben Ursprung stammen. Ressourcen haben denselben Ursprung, wenn Protokoll, Domäne und Port genau übereinstimmen. Ein unter https://www.2consult.ch/verzeichnis1 eingebettetes JavaScript kann auf Elemente unter https://www.2consult.ch/verzeichnis2 zugreifen, nicht aber auf https://2consult.ch/; die Domäne ist nicht identisch.

Der Mechanismus Cross-Origin Resource Sharing (CORS) weicht die Same-Origin Policy gezielt auf und erlaubt, über den Browser zwischen Webanwendungen zu kommunizieren, die auf unterschiedlichen Origins laufen. CORS als Standard wurde notwendig, weil mit der Micro-

services-Architektur und externen APIs Clientzugriffe auf entfernte Ressourcen zunehmen, die sonst nur mit unschönen Abhilfen machbar sind.

CORS verwendet Header. Der Browser oder Client setzt bei Cross-Origin-Anfragen den Origin-Header. Die Schnittstelle antwortet mit `Access-Control-Allow`-Headern und signalisiert damit dem Browser, woher (`Access-Control-Allow-Origin`) welche HTTP-Methoden (`Access-Control-Allow-Methods`) erlaubt sind, welche Header in der Anfrage enthalten sein dürfen (`Access-Control-Allow-Headers`) und ob der Client vorhandene Zugangsdaten über Cookies oder über HTTP-Authentifizierung mitschicken soll (`Access-Control-Allow-Credentials`).

Anfällig wird eine API, wenn sie blindlings jeden empfangenen Origin-Anfrage-Header im Antwortheader `Access-Control-Allow-Origin` reflektiert und den Header

```
Access-Control-Allow-Credentials: ↵
                                  True
```

setzt. In diesem Fall kann ein bösartiges JavaScript von der Domäne eines Angreifers, die ein Opfer ansurft, auf einer verwundbaren API im Namen des dort angemeldeten Opfers Daten auslesen und Aktionen durchführen. In subtileren Fällen lauern ebenfalls Gefahren: wenn Entwickler das Origin `null` zulassen oder das Origin nur in Bruchstücken validieren und `nicht2consult.ch` als erlaubten Ursprung ansehen statt der Domäne `2consult.ch` (siehe Abbildung 2).

Anwendungen ohne `Access-Control-Allow-Credentials` sind anfällig, wenn sie als Origin die Wildcard * oder `null` erlauben oder es nur unzureichend prüfen und reflektieren, etwa um IP-Beschränkungen und clientseitige Angriffe zu umgehen.

Häufig sind Einstellungen in Frameworks fehlerhaft. Entwickler schaffen Abhilfe, indem sie Standardkonfigurationen prüfen und CORS nur aktivieren, wenn es wirklich benötigt wird. Dann sollten sie das Origin gegen eine Allowlist strikt validieren und nicht mit regulären Ausdrücken. Man findet CORS-Lücken, indem man mit einem Proxy Origin-Header in Anfragen einfügt und wie beschrieben verändert. Dem eiligen Admin hilft der CORScanner:

```
pipx install corscanner
corscanner -u http://api.2consult.↵
                                ch:8888
```

Wer mehr über die Angriffsfläche unsachgemäß gesetzter CORS-Header lesen will, schmökert im umfangreichen Whitepaper „The Complete Guide to CORS (In) Security“ (siehe ix.de/zb7g).

Unzureichende Protokollierung und Überwachung – API10

Das letzte Risiko der Top 10 beschreibt keine Schwachstelle, sondern fehlenden oder unzureichenden technischen und organisatorischen Schutz: Wenn es bei Protokollierung und Überwachung hapert, bleiben Angriffe unerkannt. Im Nachhinein stellen Verteidiger nur lückenhaft oder gar nicht fest, ob und welche Daten abgeflossen sind. Das ist der Grund, warum Organisationen oft erst durch Externe auf Sicherheitsvorfälle aufmerksam gemacht werden und es in der Praxis meist Monate dauert, bis sie einen Datenverlust bemerken.

Relevante Ereignisse sollte man in einem einheitlichen Format protokollieren,

Informationsmaterial für Angreifer und Verteidiger

Für Pentester und Verteidiger, die sich mit API-Angriffen im Detail beschäftigen wollen, ist das Fachbuch „Hacking APIs“ von Corey Ball lesenswert. Es beschreibt, wie man eine Übungsumgebung unter anderem mit der crAPI einrichtet, wie Angreifer Endpunkte aufspüren und Schwachstellen ausnutzen. Es geht auf die Top 10 ein, ergänzt um viele weiterführende Inhalte. Wer lieber Videos schaut: Auf der YouTube-Playlist „Everything API Hacking“ sammelt Katie Paxton-Fear mehrere Stunden angriffsorientiertes Material.

Als weitere Übungsumgebungen mit verwundbaren Schnittstellen eignen sich VAmPI und der OWASP Juice Shop, die sich lokal und in der Cloud einrichten lassen. Die in Python mit dem Webframework Flask geschriebene Vulnerable API orientiert sich wie crAPI an den Top 10. Weitaus mehr Angebote bietet der sprichwörtliche Saftladen: Die REST-API des Juice Shop ist serverseitig in Node.js mit Express geschrieben und wird von einer Angular-SPA abgefragt. Sein Entwickler bestückt den Shop regelmäßig mit neuen Schwachstellen. Die umfangreiche Dokumentation hilft, den Saftladen anzuzapfen und mit ausführlichen Lösungswegen Lücken zu finden.

Die Web Security Academy von PortSwigger, dem Anbieter von Burp, ist die wohl umfangreichste Onlinelernumgebung für Angriffe auf Webanwendungen, nicht nur auf Schnittstellen bezogen. Ihre Inhalte sind kostenfrei.

Der kostenpflichtige Videokurs „API Security Best Practices“ zeigt anhand von Grundlagen und Codebeispielen, wie man APIs und Microservices sicher entwirft. Der Autor Philippe De Ryck stellt auf seiner Website Vorträge zu einzelnen Themen bereit. Wer lieber liest, für den ist das Fachbuch „API Security in Action“ von Neil Madden die Referenz. Verteidiger, die sich interaktiv mit den API Top 10 auseinandersetzen wollen, lernen beim „Application Security Training“ von KONTRA, wie Angreifer Schwachstellen finden, welche Fehler zur Lücke geführt haben und wie man sie fachgerecht stopft. Textorientierte Lerner finden auf APIsecurity.io eine Übersicht zu jedem Punkt der Liste, die als dreiseitiges Cheat Sheet herunterladbar ist.

Die Top 10 und dieser Artikel beleuchten einzelne kritische und verbreitete Schwachstellen. Eine Veröffentlichung, die sich dem systematischen Testen und sicheren Entwickeln von Web-Apps und APIs widmet, ist der Application Security Verification Standard (ASVS) der OWASP. Das Framework stellt in mehreren Kategorien detaillierte Sicherheitsanforderungen auf, die helfen, eine Anwendung sicher zu entwickeln und zu testen. Sicherheitsdienstleister können eine Schnittstelle gegen den ASVS prüfen. Eine weitere Quelle ist der Web Security Testing Guide (WSTG), der beschreibt, wie man Webanwendungen und Schnittstellen auf Bugs abklopft. Die Mind-API schließlich ist eine gut strukturierte Mindmap für Sicherheitstester und interessierte Verteidiger (alle Links siehe ix.de/zb7g).

dazu zählen Fehler bei der Authentifizierung, Autorisierung und Eingabevalidierung. Zu klären ist dabei, wann und wo in der API etwas passiert ist und wer den Aufruf verursacht hat. Die Logeinträge müssen Daten zum aktuellen Anwendungskontext enthalten, damit verdächtige Benutzerkonten entdeckt werden. Allerdings sollten Entwickler nicht alles ins Protokoll schreiben, denn Zugangsdaten in Logdateien sind eine sicherheitsrelevante Fehlkonfiguration. Am besten belegen die Logs nicht nur Speicherplatz, sondern ein SIEM (Security Information and Event Management) überwacht sie regelbasiert auf verdächtige Ereignisse. Die Reaktion auf Sicherheitsvorfälle sollte in einem Incident-Response-Prozess festgeschrieben sein, den die Beteiligten geübt haben und auf den sie im Verdachtsfall zurückgreifen [2].

Logging und Monitoring sind leidige Themen. Mit dem richtigen Vorgehen ließen sich einige Artikel füllen. Jedenfalls sollten Entwickler so früh wie möglich Protokollierung in ihre APIs einbauen und anwendungsübergreifend genutzte, leichtgewichtige Funktionen dafür schreiben. Eine kompakte Einführung bietet das Logging Cheat Sheet der OWASP (siehe Abbildung 3).

Generell sind die knapp achtzig OWASP Cheat Sheets für Architekten, Entwickler und Admins, die sich in einzelne Themen vertiefen wollen, eine unschätzbare Sammlung von kompakt aufgeschriebenem Wissen. Sie enthalten Grundlagen wie Authentifizierung und Autorisierung, Absichern einzelner Techniken wie Node.js, PHP und JSON Web Token für Java, Vermeiden von Lücken wie Injection, CSRF, SSRF und Massenzuweisung sowie Sicherheitsmechanismen wie Content Security Policy, Eingabevalidierung und HTTP-Header.

Mit Werkzeugen Sicherheitslücken aufspüren

Kommerzielle Webscanner wie Burp Suite Professional und das kostenfreie OWASP ZAP finden bereits während der Entwicklung in einer laufenden Instanz der Anwendung Lücken. Mit APIs tun sie sich teilweise schwer, weil sie nicht mehr wie bei klassischen Webanwendungen einfach alle Seiten aufspüren können. Einige Scanner akzeptieren OpenAPI-Spezifikationen. Injection-Verwundbarkeiten und sicherheitsrelevante Fehlkonfiguration erkennen sie relativ verlässlich. Doch schon bei Massenzuweisung müssen automatisierte Tools oft passen. Und Fehler in der Zugriffskontrolle lassen sich nur schwierig ohne Handarbeit aufspüren.

Admins und Entwickler finden nicht nur Hilfe bei kommerziellen Anbietern, die sich mit Testwerkzeugen auf den API-Markt gestürzt haben. Sie können sich APICheck ansehen, eine DevSecOps-Sammlung, die bestehende Prüfwerkzeuge für dynamische Tests integriert. Das quelloffene Fuzzing-Tool RESTler von Microsoft, das eine Schnittstelle ausgehend von einer Spezifikation zustandsbehaftet testet, lässt sich früh in den Entwicklungszyklus integrieren. Eine Übersicht über die in diesem und im vorigen Artikel angesprochenen Werkzeuge bietet die Tabelle „Kostenfreie API-Prüfwerkzeuge“. (nb@ix.de)

Quellen

[1] Oliver Schonschek; Gut gefüllter Werkzeugkasten; Marktübersicht DevSecOps-Tools; iX 10/2021, S. 82

[2] Enno Ewers, Martin Junghans; Was tun, wenns brennt?; Erste Hilfe nach einem Hackereinbruch; iX 10/2019, S. 46

[3] Alle Links zu den Beiträgen: ix.de/zb7g

Frank Ully

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