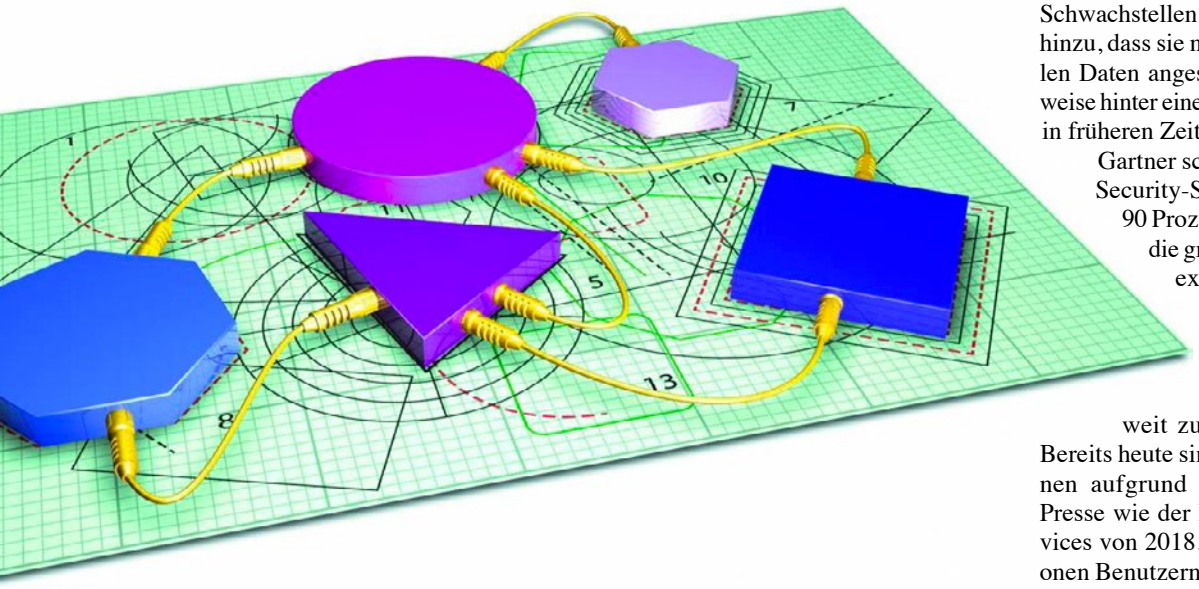
OWASP API Security Top 10

Gut behütet

Martin Burkhart



Zunehmend stehen APIs im Visier von Hackern. Ein Blick auf die neue OWASP-Liste zu den Schwachstellen zeigt, an welchen Stellen Entwickler gefordert sind.

Exponierte APIs entwickeln sich zur beliebtesten Angriffsfläche von Webanwendungen. OWASP (Open Web Application Security Project) reagiert darauf mit einer neuen und spezialisierten Top-Ten-Liste für API-Security.

Die ursprüngliche OWASP Top Ten ist eine weitverbreitete Liste der zehn wichtigsten Schwachstellen in Webapplikationen. Die Liste erschien erstmals 2003 und basiert auf den Daten hunderter Organisationen weltweit. Sie beschreibt jede Schwachstelle und potenzielle Gegenmaßnahmen im Detail. Über die Jahre hat OWASP zusätzlich Schwachstellen von APIs (Application Programming Interfaces) berücksichtigt, die in der Softwareentwicklung immer stärker Verbreitung finden. OWASP hat folglich nicht mehr einfach von Applikationen, sondern von „Applikationen oder APIs“ gesprochen. Ebenfalls hat die Organisation Schwachstellen aufgenommen, die API-spezifisch sind wie die „A4 – XML External Entities“ in der Ausgabe von 2017.

OWASP setzt nun ein Zeichen und hat im Dezember 2019 eine eigene Liste der Top-Ten-Schwachstellen für APIs veröffentlicht. Damit betont OWASP die zunehmende Wichtigkeit von API-Sicherheit für Unternehmen.

Beim Schutz der Programmierschnittstellen können dezidierte Sicherheitsprodukte wie Web Application Firewalls (WAF), API-Gateways und CIAM-Systeme (Customer Identity Access Management) einiges leisten. Sich einzig auf solche Produkte zu verlassen wäre allerdings fahrlässig. Der wichtigste Schutz geht von den Entwicklern der APIs aus, die Schwachstellen kennen und dadurch vermeiden können.

SPAs befeuern die Verbreitung von APIs

Anders als noch vor zehn Jahren sind Webapplikationen heute meist Single-Page Applications (SPA), die eine Vielzahl von APIs beispielsweise in Form von Microservices integrieren. Bei der Entwicklung der Nutzerschnittstelle kommen Webframeworks wie Angular oder React zum Einsatz. Die APIs kapseln einzelne Aspekte der Businesslogik. Die Zusammensetzung der Elemente auf der Nutzeroberfläche orientiert sich an Rich Clients.

Zeitgemäße APIs sind typischerweise als RESTful Webservices umgesetzt. Die SPAs rufen sie direkt aus dem Browser heraus auf, wodurch sie im gleichen Maße Angriffen ausgesetzt sind wie traditionelle Webanwendungen. Leider weisen solche APIs häufig ähnliche oder gar dieselben Schwachstellen auf. Erschwerend kommt hinzu, dass sie noch näher bei den sensiblen Daten angesiedelt sind als beispielsweise hinter einer Java-Enterprise-Fassade in früheren Zeiten.

Gartner schätzt in einer neuen API-Security-Studie, dass bis 2021 bei 90 Prozent der Webapplikationen die größte Angriffsfläche durch exponierte APIs entstehen wird. Deren Missbrauch wird demnach bis 2022 zum wichtigsten Angriffsvektor und weltweit zu Data Breaches führen. Bereits heute sind prominente Datenpannen aufgrund unsicherer APIs in der Presse wie der Hack des US Postal Services von 2018, der Daten von 60 Millionen Benutzern betraf. Grund für den erfolgreichen Angriff war das Fehlen von grundlegender Zugriffskontrolle auf Objekte.

Vergleich der Charts

Beim Vergleich der neuen Top-Ten-Liste für APIs mit der aktuellen für Web-Applikationen aus dem Jahr 2017 (A1-A10), fällt auf, dass etliche Schwachstellen gleich oder zumindest ähnlich sind:

- Broken Authentication (API2, A2)
- Security Misconfiguration (API7, A6)
- Injection (API8, A1)
- Insufficient Logging & Monitoring (API10, A10)

Ein Grund dafür dürfte sein, dass traditionelle Webanwendungen und SPAs mit APIs grundlegend dieselben Aufgaben erledigen. Beide Ansätze stellen eine Nutzerschnittstelle in Webbrowsern zur Verfügung und müssen Benutzer authentisieren. Sie nehmen Benutzerdaten entgegen und manipulieren Datensätze in Datenbanken. Beide laufen auf Servern oder in Containern und sind daher gegen Konfigurationsfehler anfällig. Administratoren überwachen den Betrieb und die Sicherheit gestern wie heute mit Monitorings von Log-Daten.

Die Listen haben nicht nur überlappende Themen, sondern auch eine ähnliche Priorität der Schwachstellen. Lediglich bei Injection fällt auf, dass sie bei

APIs auf Platz 8 stehen, während sie in der Webapplikationsliste den Spitzenplatz innehaben. Vielleicht liegt die Annahme zugrunde, dass moderne Webframeworks vermehrt Funktionen zur Validierung der Eingabedaten übernehmen und damit Clients weniger boshafte Strings an die APIs übergeben. Es gilt jedoch zu berücksichtigen, dass native Mobile Apps oder IoT-Clients ebenfalls die APIs nutzen. Dort fehlen entsprechende Frameworks. Zudem sollte man sich nie auf clientseitige Validierung verlassen, da Hacker an den Frameworks vorbei direkt die APIs angreifen können.

API-spezifische Schwachstellen

Interessanterweise ist „A4 XML External Entities (XEE)“ nicht in der API-Liste vertreten. Vermutlich ist das auf die abnehmende Bedeutung von SOAP-Webservices zurückzuführen. „A7 Cross-Site Scripting (XSS)“ fehlt ebenfalls auf der API-Liste. OWASP betrachtet XSS augenscheinlich als reines Browserproblem. APIs interpretieren freilich kein JavaScript und sind deshalb nicht direkt anfällig für XSS. Allerdings sollten API-Endpunkte ihre Eingabedaten dahingehend validieren, dass sie keine JavaScript-Befehle enthalten, die eine Anwendung dauerhaft speichern könnte. Je nach Client könnten die Befehle und damit XSS durchaus ein Problem darstellen.

Das generische Thema „A5 Broken Access Control“ ist in der API-Liste in unterschiedliche Aspekte aufgegliedert. Dabei trägt OWASP dem grundlegenden Aufbau von API-Aufrufen und verwendeten Formaten wie JSON Rechnung. Unautorisierte Zugriffe unterscheidet die Organisation danach, ob der Zugriff auf ganze Objekte (API1) oder einzelne Attribute von Objekten zugegriffen erfolgt. Bei den Attributen unterscheidet OWASP zudem zwischen Auslesen (API3) und Modifizieren (API6).

Im Vergleich zur OWASP Top 10 existieren zudem mit „API 4 Lack of Resources & Rate Limiting“ und „API9 Improper Assets Management“ zwei Neuzugänge. Das ist insofern nachvollziehbar, als API-Endpunkte näher an der effektiven Infrastruktur liegen als die URL eines Servlets, das Daten aus einem HTML-Formular verarbeitet.

Die passende Security-Infrastruktur

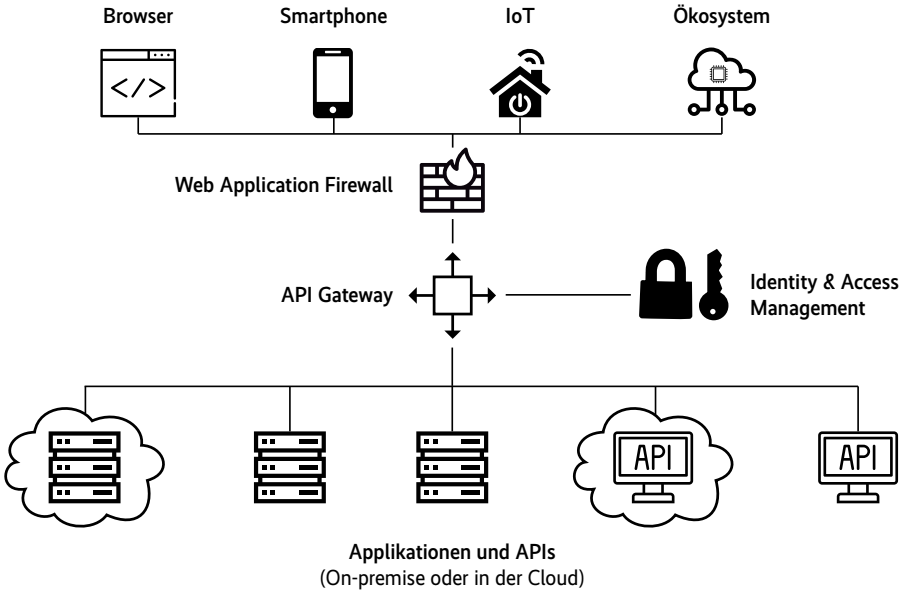
Nach dem aktuellen Stand der Technik sind Security-Services häufig vorgelagert aufgebaut, damit sie allen Applikationen und Schnittstellen zugutekommen. Die Services umfassen eine Kombination von Web Application Firewalls und API-Management, integriert mit Funktionen zur Zugriffsverwaltung (s. Abb. 1).

Je nach Produktpalette variieren die Funktionen einer solchen Architektur. Grundsätzlich bietet sie aber die Möglichkeit, neue APIs gezielt zu veröffentlichen (API9). Dass eine API intern zur Verfügung steht, bedeutet nicht automatisch, dass sie für den öffentlichen Zugriff freigegeben ist. API-Gateways können API-Keys ausstellen, die externen Entwicklern und Partnern ermöglichen, Clients auf Basis der öffentlichen APIs zu bauen.

Greift ein technischer Client mit einem gültigen Key zu, kommt die passende Usage Policy zur Anwendung. Dabei lassen sich Einschränkungen wie Throttling oder Quotas durchsetzen (API4). Liegen Spezifikationen für APIs vor – beispielsweise im OpenAPI-Format –, können die API-Gateways sie einlesen und sicherstellen, dass nur konforme Requests durchkommen. Das verhindert Explorations- oder Forceful-Browsing-Angriffe auf APIs aufgrund ungeschützter, undokumentierter Endpunkte oder durch Legacy-Endpoints beziehungsweise -Attribute (API9). Zudem lässt sich damit die korrekte Typisierung der Attribute auf dem Gateway

Übersicht: OWASP API Security Top 10

Nummer	Titel	Beschreibung
API1	Broken Object Level Authorization	API Endpoints nehmen Objekt IDs entgegen, ohne zu überprüfen, ob der User/Client für Zugriff auf diese Objekte berechtigt ist.
API2	Broken Authentication	Authentisierungslogik ist oft fehlerhaft implementiert. Dies erlaubt Angreifern, Authentisierungs-Tokens zu kompromittieren oder Fehler in der Authentisierung auszunutzen. Wenn die Identität des Clients bzw. des Users nicht sicher festgestellt werden kann, ist die Grundlage für API Security nicht gegeben.
API3	Excessive Data Exposure	Entwickler tendieren dazu, generell alle Objekteigenschaften auf Endpoints zu exponieren, auch die sensitiven. Man verlässt sich auf den Client und hofft, dass dieser die Daten passend filtert, bevor sie dem User angezeigt werden.
API4	Lack of Resources & Rate Limiting	APIs machen keine Limitierung bezüglich der Größe und der Anzahl Ressourcen, die ein Client anfragt. Dies kann zu schlechter Performance bis hin zu Denial of Service (DoS) führen. Des Weiteren können dadurch Brute-force Angriffe, z.B. auf Passwörter, ermöglicht werden.
API5	Broken Function Level Authorization	Komplexe Zugriffsregeln mit verschiedenen Hierarchien, Gruppen, Rollen und einer unklaren Trennung zwischen administrativen und regulären Funktionen führen zu Autorisierungsfehlern. Angreifer können so Zugriff auf Ressourcen erlangen, der ihnen nicht zusteht.
API6	Mass Assignment	Vom Client gelieferte Daten, z.B. im JSON Format, werden direkt und ohne Filterung ins Datenmodell abgefüllt. Angreifer können zusätzliche Attribute erraten, in der Dokumentation einsehen oder mittels Exploration herausfinden. Damit können sie Objekt Attribute modifizieren, zu denen sie keinen Zugriff haben sollten.
API7	Security Misconfiguration	Unsichere Konfigurationen resultieren meist aus unsicheren Default Settings, unvollständigen Konfigurationen, öffentlich verfügbaren Cloud Storages, falsch konfigurierten HTTPS Headers und Methoden, zu offenen CORS Regeln oder auch Fehlermeldungen, die zuviel preisgeben.
API8	Injection	Injection Schwachstellen, z.B. für SQL, NoSQL, LDAP oder OS Commands entstehen, wenn unsichere Input-Daten als Teil eines Kommandos an einen Interpreter geschickt werden. Ein Angreifer kann damit ggf. bösartige Aktionen auslösen und unautorisiert Daten auslesen oder verändern.
API9	Improper Assets Management	APIs exponieren tendenziell mehr Endpoints als traditionelle Web-Applikationen. Korrekte und aktuelle Dokumentation ist daher sehr wichtig. Ein Inventar über die Hosts und API Versionen verhindert z.B. die Veröffentlichung von nicht mehr unterstützten APIs und Debugging Endpoints.
API10	Insufficient Logging & Monitoring	Unzureichendes Logging und Monitoring, verbunden mit fehlender oder ungenügender Integration mit Incident Response, erlauben es Angreifern Systeme zu attackieren, sich einzunisten und weitere Ziele anzugreifen, mit dem Ziel Daten zu extrahieren oder zu modifizieren. Studien zeigen, dass Breaches oft erst nach 200 Tagen entdeckt werden, und das erst noch von externen Stellen und nicht durch interne Prozesse.



Web Application Firewall, API-Gateway und IAM arbeiten Hand in Hand für umfassenden API-Schutz (Abb. 1).

überprüfen und durchsetzen. Falls die Spezifikationen griffig und präzise sind, lassen sich damit Injection-Angriffe verhindern (API8).

CIAM-Systeme dienen dem Verwalten der Identität und Zugriffsrechte. Sie authentisieren die Benutzer (API2) und berechtigen sie durch Standards wie OAuth, OpenID Connect oder SAML für den Zugriff auf Applikationen und APIs (API5). Das ermöglicht zudem die Umsetzung eines übergreifenden Single Sign-Ons (SSO) und die Abwicklung von Standardaufgaben über Benutzer-Self-Services.

Web Application Firewalls bieten eine Vielzahl von Schutzmechanismen gegen bekannte Angriffe wie Injections, XSS oder CSRF (API8). Zudem verfügen sie über sichere Grundeinstellungen für HTTP Headers und TLS (API7) sowie Funktionen für das Zertifikatsmanagement unter anderem über Let's Encrypt. Für einen guten API-Schutz gilt es, darauf zu achten, dass die WAF effektiv JSON-Objekte analysiert und ihre Regeln auf einzelne Attribute anwenden kann. Sonst müsste ein passendes API-Security-Gateway die Arbeit übernehmen. Viele API-Gateways konzentrieren sich allerdings auf das API-Management und behandeln Security-Aspekte stiefmütterlich.

Eine dezidierte Security-Infrastruktur vereinfacht Monitoring, Troubleshooting und forensische Analysen (API10). Ergänzt durch SIEM-Systeme, lassen sich Informationen über verschiedene Komponenten einfach zusammenziehen und korrelieren. Ein Alerting bei Anomalien ermöglicht zudem, den reaktiven Modus zu verlassen und im Falle eines Eindringens wertvolle Zeit zu gewinnen.

Viele Security-Produkte setzen mittlerweile Machine Learning (ML) ein, um auf zuvor unbekannte Angriffe und Situationen reagieren zu können. Für einzelne Entwickler ist der Einsatz von ML schwierig, da die Methoden viele Daten erfordern und von einer serviceübergreifenden Perspektive profitieren.

Die Pflicht der Entwickler

Unabhängig von externen Gateways bleiben einige Aufgaben in jedem Fall bei den Entwicklern hängen. Generell gilt, dass sie Software so entwickeln sollten, dass sie ohne vorgelagerte Security-Services sicher ist. Sie sollten unter anderem Inputs validieren, unabhängig davon, ob eine WAF auf Injection-Angriffe prüft. WAFs müssen immer einen Kompromiss zwischen False Positives und False Negatives eingehen und haben daher keine hundertprozentige Erkennungsrate. Zudem können vom Entwickler unbemerkt Änderungen in der Infrastruktur geschehen. Es empfiehlt sich, Security by Design und nicht als Anhängsel („As an Afterthought") sicherzustellen. In die Build-Pipeline integrierte Vulnerability-Scanner können dabei helfen, bestehende Schwachstellen im fertigen Code aufzudecken.

Entwickler müssen sich zwingend um die fachliche Autorisierung von Businessobjekten kümmern. Ein API-Gateway kennt zwar die Endpunkte und kann zwischen GET- und UPDATE-Aufrufen unterscheiden. Aber die Businessobjekte und deren Attribute sind ihm fremd. Deshalb müssen Entwickler sicherstellen, dass Objekt-IDs, die der Client liefert, effektiv für den authentisierten Benutzer zugelassen sind (API1). Diese Prüfung ist ebenso beim Modifizieren einzelner Attribute notwendig (API6). Beim Minimieren der Datenauslieferung dürfen Entwickler sich nicht auf den Client verlassen, sondern müssen auf API-Seite heikle Attribute herausfiltern (API3).

Wichtig ist zudem der Umgang mit API-Keys. Sie sind explizit kein Mittel zur Authentisierung, sondern dienen lediglich der Identifikation von technischen Clients wie einer mobilen App oder einer SPA. Eine Anwendung darf die Zugriffe auf APIs nicht nur aufgrund von API-Keys freigeben, sondern muss solide Benutzer-Authentisierung und -Autorisierung sicherstellen. Selbstredend gehören API-Keys nicht in öffentliche Cloud Storages. Falls Clients hartkodierte API-Keys benötigten, müssen Teams in Client-Security investieren, um Angriffe wie Debugging und Dekompilierung des Clientcodes zu erschweren.

Angriffsfläche APIs

APIs entwickeln sich voraussichtlich in den nächsten Jahren zur Hauptangriffsfläche von Webanwendungen. OWASP reagiert darauf mit einer neuen und spezialisierten Top-Ten-Liste für API-Security. Einige Themen der neuen Liste wie Authentisierung und Injection-Angriffe dominieren die Sicherheit des Webs seit 2003, als OWASP die erste Top-Ten-Liste veröffentlicht hatte.

Die Schwachstellen gilt es erst recht im Kontext von APIs zu verhindern. Beim Schutz von APIs können dezidierte Sicherheitsprodukte wie Web Application Firewalls, API-Gateways und CIAM-Systeme einiges leisten. Entwickler dürfen sich jedoch nicht einzig auf Produkte verlassen. Gewisse Themen wie die fachliche Autorisierung von Businessobjekten und der sichere Umgang mit API-Keys bleiben in ihrer Verantwortung. (rme)

Dr. Martin Burkhart

ist Head of Product Management bei Airlock, einer Security-Innovation der Ergon Informatik AG. Anfangs leitete er IAM Integrationsprojekte und ist seit 2012 für das Produktmanagement des Airlock Secure Access Hubs zuständig.