OWASP API Security Top 10 2019

Die 10 kritischsten Sicherheitsrisiken für APIs

Inhaltverzeichnis

Über das OWASP

Das Open Web Application Security Project (OWASP) ist eine offene Gemeinschaft, deren Ziel es ist Organisationen zu ermöglichen, Anwendungen und APIs zu entwickeln, zu erwerben und zu pflegen, denen man vertrauen kann.

Bei OWASP finden Sie freie und offene:

- Tools und Standards für die Anwendungssicherheit.
- Vollständige Bücher über Anwendungssicherheitstests, sichere Codeentwicklung und Sicherheitsüberprüfungen von Code.
- Präsentationen und Videos.
- Cheat sheets zu vielen gängigen Themen.
- Standard-Sicherheitskontrollen und Bibliotheken.
- Örtliche Chapter weltweit.
- Modernste Forschung.
- Umfangreiche Konferenzen weltweit.
- Mailinglisten.

Lerne mehr darüber auf: https://www.owasp.org.

Alle OWASP-Tools, -Dokumente, -Videos, -Präsentationen und -Kapitel sind kostenlos und für jeden zugänglich, der an der Verbesserung von Anwendungssicherheit interessiert ist.

Wir plädieren dafür, die Anwendungssicherheit als ein Mensch-, Prozess- und Technologieproblem zu betrachten, da die effektivsten Ansätze zur Anwendungssicherheit Verbesserungen in diesen Bereichen liegen.

Die OWASP ist eine neue Art von Organisation. Unsere Freiheit von kommerziellen Zwängen ermöglicht es uns, unvoreingenommene, praktische und kostengünstige Informationen über Anwendungssicherheit zuliefern.

OWASP ist nicht mit einem Technologieunternehmen verbunden, obwohl wir den Einsatz kommerzieller Sicherheitstechnologien unterstützen. Die OWASP erstellt viele Arten von Materialien in einer gemeinschaftlichen, transparenten und offenen Weise.

Die OWASP Foundation ist die gemeinnützige Einrichtung, die den langfristigen Erfolg des Projekts sicherstellt. Fast jeder, der mit OWASP zu tun hat, ist ein Freiwilliger, einschließlich des OWASP-Vorstands, der Chapter-Leiter, der Projektleiter und der Projektmitglieder. Wir unterstützen innovative Sicherheitsforschung mit Zuschüssen und Infrastruktur.

Tritt uns bei!

Vorwort

Ein fundamentales Element der Innovation in der heutigen App-getriebenen Welt ist die Anwendungsprogrammierschnittstelle (API). Von Bankenwesen, dem Einzelhandel bis hin zu IoT-Systemen, autonomen Fahrzeugen und intelligenten Städten - APIs sind ein wichtiger Bestandteil der modernen mobilen und Webanwendungen. APIs sind in alle modernen externen und internen Anwendungen zu finden.

APIs stellen von Natur aus Anwendungslogik und sensible Daten wie personenbezogene Daten externen Dritten zur Verfügung. Deshalb sind APIs zunehmend zu einem Ziel für Angreifer geworden. Ohne sichere APIs wäre eine schnelle Innovation unmöglich.

Obwohl eine umfassendere Top-10-Liste der Sicherheitsrisiken von Webanwendungen in Teilen anwendbar ist, ist aufgrund ihrer eigenen Besonderheiten eine API-spezifische Liste der Sicherheitsrisiken sinnvoll. Die OWASP Top 10 API Security Risks konzentriert sich auf die Vermittlung von Strategien zur Behebung der gängigsten Schwachstellen und Sicherheitsrisiken im Zusammenhang mit APIs.

Wenn Sie mit dem OWASP Top 10 Project vertraut sind, dann werden Sie die Ähnlichkeiten zwischen diesen beiden Dokumenten feststellen. Diese Ähnlichkeiten ermöglichen eine schnelle Einarbeitung in diese Thematik. Wenn Sie die OWASP Top 10-Reihe noch nicht kennen, sollten Sie zuerst die Kapitel Sicherheitsrisiken für APIs und Methodik und Daten lesen bevor Sie sich mit dieser Top-10-Liste befassen.

Sie können zu den OWASP API Security Top 10 mit Ihren Fragen, Kommentaren und Ideen in unserem GitHub-Projekt-Repository beitragen:

- https://github.com/OWASP/API-Security/issues
- https://github.com/OWASP/API-Security/blob/master/CONTRIBUTING.md

Sie können die OWASP API Security Top 10 hier finden:

- https://www.owasp.org/index.php/OWASP_API_Security_Project
- https://github.com/OWASP/API-Security

Wir danken allen Beteiligten, die dieses Projekt mit ihrem Engagement und ihren Beiträgen möglich gemacht haben. Sie sind alle im Abschnitt Danksagungen aufgeführt. Vielen Dank!

Willkommen bei der OWASP API Security Top 10 - 2019!

Willkommen zur ersten Ausgabe der OWASP API Security Top 10. Wenn Sie mit der OWASP Top 10-Serie vertraut sind, werden Sie die Ähnlichkeiten bemerken: Sie sind auf Lesbarkeit und Akzeptanz ausgelegt. Andernfalls sollten Sie einen Blick auf die OWASP API Security Project wiki page werfen, bevor Sie sich näher mit den wichtigsten API-Sicherheitsrisiken auseinandersetzen.

APIs spielen eine sehr wichtige Rolle in der Architektur moderner Anwendungen. Da Sicherheitsbewusstsein und Innovation unterschiedliche Geschwindigkeiten haben, ist es wichtig, sich auf allgemeine API-Schwächen zu konzentrieren.

Das primäre Ziel der OWASP API Security Top 10 ist es, die an der Entwicklung und Wartung von APIs beteiligt sind, zum Beispiel Entwickler, Designer Architekten, Manager oder Organisationen zu bilden.

Im Abschnitt Methodik und Daten können Sie mehr darüber lesen, wie diese erste Ausgabe erstellt wurde. In künftigen Versionen wollen wir die Sicherheitsbranche einbeziehen, mit einem öffentlichen Aufruf zur Datenerhebung. Für den Moment ermutigen wir jeden dazu sich mit Fragen, Kommentaren und Ideen an unser GitHub-Repository oder unsere Mailingliste zuwenden.

Erläuterungen zur Veröffentlichung

Dies ist die erste Ausgabe der OWASP API Security Top 10, die regelmäßig, alle drei oder vier Jahre, aktualisiert werden soll.

Im Gegensatz zu dieser Version wollen wir in zukünftigen Versionen einen öffentlichen Aufruf zur Datenerhebung veröffentlichen, und die Sicherheitsbranche in unsere Initiative miteinbeziehen. In dem Kapitel Methodik und Daten finden Sie weitere Einzelheiten darüber, wie diese Version erstellt wurde. Detaillierte Informationen zu den Sicherheitsrisiken finden Sie im Kapitel Sicherheitsrisiken für APIs.

Es ist wichtig zu verstehen, dass sich in den letzten Jahren die Architektur von Anwendungen grundsätzlich verändert hat. Derzeit spielen APIs eine sehr wichtige Rolle in dieser neuen Infrastruktur aus Microservices, Single Page Applications (SPAs), mobilen Anwendungen und IoT-Systemen.

Die OWASP API Security Top 10 war eine notwendige Maßnahme, um das Bewusstsein für moderne API-Sicherheitsprobleme zu schaffen. Sie war nur möglich durch den großen Einsatz von zahlreichen ehrenamtlichen Helfern, welche alle im Abschnitt Danksagungen aufgeführt sind. Vielen Dank!

OWASP Top 10 API Security Risks – 2019

Risko	Beschreibung
API1:2019 - Broken Object Level Authorization	APIs neigen dazu, Endpunkte offenzulegen, die Objektidentifikatoren verarbeiten, was eine breite Angriffsfläche auf Zugriffskontrolle auf Objektebene schafft. Berechtigungsprüfungen auf Objektebene sollten in jeder Funktion berücksichtigt werden, die auf eine Datenquelle zugreift, die eine Eingabe des Benutzers verwendet.
API2:2019 - Broken User Authentication	Authentifizierungsmechanismen sind oft falsch implementiert, sodass Angreifer Authentifizierungstoken kompromittieren oder Implementierungsfehler ausnutzen können, um vorübergehend oder dauerhaft die Identität anderer Benutzer anzunehmen. Durch die Beeinträchtigung der Fähigkeit des Systems, den Client / Benutzer zu identifizieren, wird die API-Sicherheit insgesamt gefährdet.
API3:2019 - Excessive Data Exposure	Mit Blick auf generische Implementierungen neigen Entwickler dazu, alle Objekteigenschaften freizugeben, ohne deren individuelle Empfindlichkeit zu berücksichtigen, und sich darauf zu verlassen, dass die Clients die Datenfilterung durchführen, bevor sie dem Benutzer angezeigt werden.
API4:2019 - Lack of Resources & Rate Limiting	Oftmals gibt es bei APIs keine Beschränkungen für die Größe oder Anzahl der Ressourcen, die vom Client/Nutzer angefordert werden können. Dies kann sich nicht nur auf die Leistung des API-Servers auswirken und zu Denial of Service (DoS) führen, sondern öffnet auch die Tür für Authentifizierungs-Angriffe wie Brute Force.
API5:2019 - Broken Function Level Authorization	Komplexe Zugriffskontrollrichtlinien mit unterschiedlichen Hierarchien, Gruppen und Rollen sowie eine unklare Trennung zwischen administrativen und regulären Funktionen führen häufig zu Sicherheitslücken in der Autorisierung. Unter Ausnutzung dieser Schwachstellen können Angreifer Zugriff auf die Ressourcen anderer Benutzer und/oder administrative Funktionen erlangen.
API6:2019 - Mass Assignment	Das Verbinden von vom Client bereitgestellten Daten (z.B. JSON) an Datenmodelle ohne ordnungsgemäße Filterung der Eigenschaften auf der Grundlage einer Whitelist führt in der Regel zu Mass Assignment. Das Erraten von Objekteigenschaften, das Erforschen anderer API-Endpunkte, das Lesen der Dokumentation oder die Bereitstellung zusätzlicher Objekteigenschaften in Daten von Anfragen ermöglicht es Angreifern, Objekteigenschaften zu verändern, die sie nicht verändern dürfen.
API7:2019 - Security Misconfigura	Sicherheitsfehlkonfigurationen sind häufig das Ergebnis unsicherer Standardkonfigurationen, unvollständiger oder Ad-hoc-Konfigurationen, offener Cloud-Speicher, falsch konfigurierter HTTP-Header, unnötiger HTTP-Methoden, permissiver Cross-Origin-Resource-Sharing (CORS) und ausführlicher

tion	Fehlermeldungen mit sensiblen Informationen.
API8:2019 - Injection	Injection-Fehler, wie SQL, NoSQL, Command Injection usw., treten auf, wenn nicht vertrauenswürdige Daten als Teil eines Befehls oder einer Abfrage an einen Interpreter gesendet werden. Die bösartigen Daten des Angreifers können den Interpreter dazu verleiten, unbeabsichtigte Befehle auszuführen oder auf Daten zuzugreifen, ohne dazu berechtigt zu sein.
API9:2019 - Improper Assets Management	APIs neigen dazu, mehr Endpunkte freizugeben als herkömmliche Webanwendungen, was eine ordnungsgemäße und aktualisierte Dokumentation sehr wichtig macht. Eine ordnungsgemäße Inventarisierung der Hosts und der bereitgestellten API-Versionen spielt ebenfalls eine wichtige Rolle, um Probleme wie veraltete API-Versionen und offengelegte Debug-Endpunkten zu entschärfen.
API10:2019 - Insufficient Logging & Monitoring	Unzureichende Protokollierung und Überwachung in Verbindung mit einer fehlenden oder unwirksamen Integration in die Reaktion auf Vorfälle (Incidents Response) ermöglichen es Angreifern, weitere Systeme anzugreifen, die Persistenz aufrechtzuerhalten und auf weitere Systeme überzugehen, um Daten zu manipulieren, zu extrahieren oder zu zerstören. Die meisten Studien zu Breaches (Cyber Einbruch) zeigen, dass die Zeit bis zur Entdeckung eines Breaches mehr als 200 Tage beträgt und in der Regel eher von externen Parteien als von internen Prozessen oder der internen Überwachung entdeckt werden.

API1:2019 Broken Object Level Authorization

Bedrohungsakteure/Angriffsvektoren	**Sicherheitslücken**	**Auswirkungen**
API-spezifisch : Ausnutzbarkeit **3**	Häufigkeit **3** : Erkennbarkeit **2**	Komplexität **3** : Unternehmensspezifisch
Angreifer können API-Endpunkte ausnutzen, die anfällig für eine fehlerhafte Autorisierung auf Objektebene sind, indem sie die ID eines Objekts manipulieren, die innerhalb des Requests gesendet wird. Dies kann zu einem unberechtigten Zugriff auf sensible Daten führen. Dieses Problem tritt bei API-basierten Anwendungen häufig auf, da die Serverkomponente in der Regel nicht den vollständigen Status des Clients verfolgt und stattdessen stärker auf Parameter wie Objekt-IDs angewiesen ist, die vom Client gesendet werden, um zu entscheiden, auf welche Objekte zugegriffen werden soll.	Dies ist der häufigste und folgenreichste Angriff auf APIs. Autorisierungs- und Zugriffskontrollmechanismen in modernen Anwendungen sind komplex und weit verbreitet. Selbst wenn die Anwendung eine angemessene Infrastruktur für Autorisierungsprüfungen implementiert, können Entwickler vergessen, diese Prüfungen vor dem Zugriff auf ein sensibles Objekt durchzuführen. Die Erkennung von Zugriffskontrollen ist in der Regel nicht durch automatisierte statische oder dynamische Tests möglich.	Unbefugter Zugriff kann zur Offenlegung von Daten an Unbefugte, zu Datenverlust oder Datenmanipulation führen. Unbefugter Zugriff auf Objekte kann auch zu einer vollständigen Übernahme des Kontos führen.

Ist die API angreifbar?

Die Autorisierung auf Objektebene ist ein Zugriffskontrollmechanismus, der in der Regel auf Codeebene implementiert wird, um sicherzustellen, dass ein Benutzer nur auf die Objekte zugreifen kann, auf die er Zugriff haben sollte.

Jeder API-Endpunkt, der die ID eines Objekts empfängt und irgendeine Art von Aktion mit dem Objekt ausführt, sollte Berechtigungsprüfungen auf Objektebene implementieren. Diese Prüfungen sollten validieren, dass der eingeloggte Benutzer die Berechtigung hat, die angeforderte Aktion an dem angeforderten Objekt durchzuführen.

Versagen dieses Mechanismus führt in der Regel zur unberechtigten Offenlegung, Änderung oder Zerstörung aller Daten.

Beispiele für Angriffe

Szenario #1

Ein E-Commerce-Plattform für Online-Shops bietet eine Auflistungsseite mit Umsatzdiagrammen für die von ihr gehosteten Shops. Durch Untersuchung der Browseranfragen kann ein Angreifer die API-Endpunkte identifizieren, die als Datenquelle für diese Diagramme dienen und deren Muster `/shops/{shopName}/revenue_data.json` ist. Über einen anderen API-Endpunkt kann der Angreifer eine Liste aller gehosteten Shopnamen abrufen. Mithilfe eines einfachen Skripts, das die Namen in der Liste manipuliert und `{shopName}` in der URL ersetzt, erhält der Angreifer Zugang zu den Verkaufsdaten von Tausenden von E-Commerce-Shops.

Szenario #2

Beim Überwachen des Netzwerkverkehrs eines Wearable fällt einem Angreifer eine HTTP PATCH-Anfrage auf, die einen benutzerdefinierten HTTP-Anfrageheader mit dem Namen X-User-Id: 54796 enthält. Durch Ersetzen des X-User-Id-Werts durch 54795 erhält der Angreifer eine erfolgreiche HTTP-Antwort und kann die Kontodaten anderer Benutzer ändern.

Vorbeugende Maßnahmen

- Implementierung eines angemessenen Autorisierungsmechanismus, der auf den Benutzerrichtlinien und der Benutzerhierarchie basiert.
- Verwenden Sie einen Autorisierungsmechanismus, um zu überprüfen, ob der eingeloggte Benutzer berechtigt ist, die angeforderte Aktion für den Datensatz in jeder Funktion auszuführen, die eine Eingabe vom Client verwendet, um auf einen Datensatz in der Datenbank zuzugreifen.
- Verwenden Sie zufällige und nicht vorhersehbare GUIDs als IDs für Datensätze, wo immer möglich.
- Schreiben Sie Tests, um den Autorisierungsmechanismus zu bewerten. Veröffentlichen Sie keine anfälligen Änderungen, die die Tests brechen.

Referenzen

OWASP

- CWE-284: Improper Access Control
- CWE-285: Improper Authorization
- CWE-639: Authorization Bypass Through User-Controlled Key

API2:2019 Broken User Authentication

Bedrohungsakteure/Angriffsvektoren	**Sicherheitslücken**	**Auswirkungen**
API-spezifisch : Ausnutzbarkeit **3**	Häufigkeit **2** : Erkennbarkeit **2**	Komplexität **3** : Unternehmensspezifisch
Die Authentifizierung in APIs ist ein komplexer und unübersichtlicher Prozess. Softwareentwickler haben möglicherweise falsche Vorstellungen darüber, wo die Einschränkungen der Authentifizierung liegen und wie man sie richtig implementiert. Darüber hinaus ist der Authentifizierungsmechanismus ein leichtes Ziel für Angreifer, da er für jeden zugänglich ist. Diese beiden Punkte machen die Authentifizierungskomponente anfällig für zahlreiche Angriffe.	Es gibt zwei Unterpunkte: 1. Fehlende Schutzmechanismen: API-Endpunkte, die für die Authentifizierung zuständig sind, müssen anders behandelt werden als normale Endpunkte und zusätzliche Schutzmechanismen implementieren. 2. Falsche Implementierung des Verfahrens: Der Prozess der Authentifizierung wird verwendet/implementiert, ohne die Angriffsvektoren zu berücksichtigen, oder es ist der falsche Anwendungsfall (z. B. ist ein Authentifizierungsmechanismus, der für IoT-Clients entwickelt wurde, möglicherweise nicht die richtige Wahl für Webanwendungen) implementiert worden.	Angreifer können die Kontrolle über die Konten anderer Benutzer im System übernehmen, ihre persönlichen Daten auslesen und sensiblen Aktionen in deren Namen ausführen, wie Geld transferieren oder persönlichen Nachrichten versenden.

Ist die API angreifbar?

Endpunkte für die Authentifizierungs sind sensible Funktionen, die geschützt werden müssen. "Passwort vergessen / zurücksetzen" sollte genauso behandelt werden wie Authentifizierungsmechanismen.

Eine API ist verwundbar, wenn sie folgendes zulässt:

- Wenn diese credential stuffing zulässt. Hierbei versucht sich in Angreifer mit einer Liste geklauter Zugangsdaten einzuloggen.
- Wenn diese es Angreifern erlaubt, einen Brute-Force-Angriff auf dasselbe Benutzerkonto durchzuführen, ohne ein Captcha-Mechanismus oder eine Sperrfunktion für das Konto implementieren.
- Wenn die API unsichere Passwörter erlaubt.
- Sensible Anmeldeinformationen, wie Auth-Tokens und Passwörter, in der URL sendet.
- Die Authentizität von Tokens nicht validiert.
- Unsignierte/schwach signierte JWT-Tokens ("alg":"none") akzeptiert oder deren Ablaufdatum nicht validiert.
- Passwörter im Klartext speichert oder schwache Hash-Funktion für Passwörter benutzt.
- Schwache Verschlüsselungsschlüssel verwendet.

Beispiele für Angriffe

Szenario #1

Credential stuffing (durch Verwendung von Listen mit bekannten Benutzernamen/Passwörtern) ist ein häufiger Angriff. Wenn eine Anwendung keine automatischen Schutzmaßnahmen gegen Bedrohungen oder Credential Stuffing implementiert, kann die Anwendung als Passwort-Oracle (Tester) verwendet werden, um zu bestimmen, ob die Anmeldeinformationen gültig sind.

Szenario #2

Ein Angreifer startet den Passwort-Wiederherstellungs-Prozess, indem er eine POST-Anfrage an `/api/system/verification-codes` sendet und den Benutzernamen im Anfrage-Body bereitstellt. Als nächstes wird ein SMS-Token mit 6 Ziffern an das Telefon des Opfers gesendet. Da die API keine Policy zur Begrenzung der Abfragerate implementiert, kann der Angreifer alle möglichen Kombinationen mit einem mehrthreadigen Skript gegen das Endpunkt `/api/system/verification-codes/{smsToken}` testen, um innerhalb weniger Minuten das richtige Token zu entdecken.

Vorbeugende Maßnahmen

- Es sollte sichergestellt werden, dass alle Methoden zur Authentifizierung einer API bekannt sind (Mobile/Web/Deep Links, die eine Ein-Klick-Authentifizierung).
- Es sollte mit den Software-Entwicklern abgestimmt werden ob es noch weitere Methoden zur Authentifizierung gibt.
- Informieren Sie sich die genutzen Authentifizierungsmechanismen. Stellen Sie sicher, dass Sie verstehen, was und wie diese verwendet werden. OAuth ist keine Authentifizierung und API-Schlüssel auch nicht.
- Erfinde Sie das Rad nicht neu in Bezug auf Authentifizierung, Token-Generierung und Passwort-Speicherung. Verwenden Sie die anerkannten Standards.
- Endpunkte für Passwort-Vergeßen Funktion sollten in Bezug auf Brute-Force Angriffe, Begrenzung der Abfragerate und Sperrmechanismen wie Login-Endpunkte behandelt werden.
- Benutzen Sie das OWASP Authentication Cheatsheet.
- Implementieren Sie Multi-Faktor-Authentifizierung, sofern dies möglich ist.
- Implementieren SIe Anti-Brute-Force-Mechanismen, um Credential Stuffing, Dictionary-Attacken und Brute-Force-Angriffe auf die Authentifizierungs-Endpunkte zu erschweren. Dieser Mechanismus sollte restriktiver sein als die reguläre Begrenzung der Abfragerate der API.
- Implementieren Sie Accountsperren oder Captchas um Brute-Force-Angriffe auf bestimmte Benutzer zu verhindern. Prüfen sie alle Passwörter auf schwache Passwörter.
- API-Schlüssel sollten nicht für die Benutzerauthentifizierung verwendet werden, sondern für client app/ project authentication.

Referenzen

OWASP

- OWASP Key Management Cheat Sheet
- OWASP Authentication Cheatsheet
- Credential Stuffing

Externe Quellen

- CWE-798: Use of Hard-coded Credentials

API3:2019 Excessive Data Exposure

Bedrohungsakteure/ Angriffsvektoren	**Sicherheitslücken**	**Auswirkungen**
API-spezifisch : Ausnutzbarkeit **3**	Häufigkeit **2** : Erkennbarkeit **2**	Komplexität **2** : Unternehmensspezifisch
Die Ausnutzung von übermäßiger Datenfreigabe ist einfach und erfolgt in der Regel durch Sniffing des Datenverkehrs, um die API-Antworten zu analysieren und nach sensiblen Daten zu suchen, die nicht an den Benutzer zurückgegeben werden sollten.	APIs verlassen sich darauf, dass Clients die Datenfilterung durchführen. Da APIs als Datenquellen verwendet werden, versuchen Entwickler manchmal, sie auf generische Weise zu implementieren, ohne an die Sensibilität der offengelegten Daten zu denken. Automatische Tools können diese Art von Schwachstellen in der Regel nicht erkennen, da es schwierig ist, zwischen legitimen Daten, die von der API zurückgegeben werden, und sensiblen Daten, die ohne tiefes Verständnis der Anwendung nicht zurückgegeben werden sollten, zu unterscheiden.	Eine übermäßige Datenexposition führt in der Regel zur Offenlegung sensibler Daten.

Ist die API angreifbar?

Die API gibt sensible Daten absichtlich an den Client zurück. Diese Daten werden normalerweise auf der Client-Seite gefiltert, bevor sie dem Benutzer präsentiert werden. Ein Angreifer kann den Datenverkehr leicht ausspähen und die sensiblen Daten sehen.

Beispiele für Angriffe

Szenario #1

Das Mobile-Team verwendet den Endpunkt `/api/articles/{articleId}/comments/{commentId}` in der Artikelansicht, um Metadaten zu Kommentaren zu rendern. Durch das Sniffing des Datenverkehrs der mobilen Anwendung findet ein Angreifer heraus, dass auch andere sensible Daten, die sich auf den Autor des Kommentars beziehen, zurückgegeben werden. Die Implementierung des Endpunkts verwendet eine generische `toJSON()`-Methode des User-Modells, das PII enthält, um das Objekt zu serialisieren.

Szenario #2

Ein IoT-basiertes Überwachungssystem ermöglicht es Administratoren, Benutzer mit unterschiedlichen Berechtigungen zu erstellen. Ein Administrator erstellt ein Benutzerkonto für einen neuen Sicherheitsmitarbeiter, der nur Zugang zu bestimmten Gebäuden auf dem Gelände haben sollte. Sobald der Sicherheitsmitarbeiter seine mobile App verwendet, wird ein API-Aufruf ausgelöst: /api/sites/111/cameras, um Daten über die verfügbaren Kameras zu erhalten und sie auf dem Dashboard anzuzeigen.

Die Antwort enthält eine Liste mit Details über Kameras im folgenden Format: `{"id": "xxx", "live_access_token": "xxxx-bbbbb", "building_id": "yyy"}`. Während die Client-GUI nur Kameras anzeigt, auf die der Sicherheitsmitarbeiter Zugriff haben soll, enthält die tatsächliche API-Antwort eine vollständige Liste aller Kameras auf dem Gelände.

Vorbeugende Maßnahmen

- Verlassen Sie sich niemals auf die Client-Seite, um sensible Daten zu filtern.
- Überprüfen Sie die Antworten der API, um sicherzustellen, dass sie nur legitime Daten enthalten.
- Backend-Entwickler sollten sich immer fragen "Wer ist der Nutzer der Daten?", bevor sie einen neuen API-Endpunkt freigeben.
- Vermeiden Sie die Verwendung generischer Methoden wie `to_json()` und `to_string()`. Wählen Sie stattdessen spezifische Eigenschaften aus, die Sie wirklich zurückgeben möchten.
- Klassifizieren Sie sensible und persönlich identifizierbare Informationen (PII), die Ihre Anwendung speichert und verarbeitet. Überprüfen Sie alle API-Aufrufe, die solche Informationen zurückgeben, um zu sehen, ob diese Antworten ein Sicherheitsproblem darstellen.
- Implementieren Sie einen schema-basierten Response-Validierungsmechanismus als zusätzliche Sicherheitsebene. Im Rahmen dieses Mechanismus definieren und erzwingen Sie die Daten, die von allen API-Methoden zurückgegeben werden, einschließlich Fehler.

Referenzen

OWASP

- CWE-213: Intentional Information Exposure

API4:2019 Lack of Resources & Rate Limiting

Bedrohungsakteure/Angriffsvektoren	**Sicherheitslücken**	**Auswirkungen**
API-spezifisch : Ausnutzbarkeit **2**	Häufigkeit **3** : Erkennbarkeit **3**	Komplexität **2** : Unternehmensspezifisch
Die Ausnutzung erfordert einfache API-Anfragen. Es ist keine Authentifizierung erforderlich. Mehrere gleichzeitige Anfragen können von einem einzigen lokalen Computer aus oder unter Verwendung von Cloud-Computing-Ressourcen durchgeführt werden.	Es ist üblich, APIs zu finden, die keine Rate-Limitierung implementieren oder APIs, bei denen die Limits nicht ordnungsgemäß gesetzt sind.	Eine Ausnutzung kann zu DoS führen, wodurch die API nicht mehr reagiert oder sogar nicht mehr verfügbar ist.

Ist die API angreifbar?

API-Anforderungen verbrauchen Ressourcen wie Netzwerk, CPU, Arbeitsspeicher und Speicherplatz. Die Menge an Ressourcen, die zur Beantwortung einer Anfrage benötigt wird, hängt stark von der Eingabe und der Geschäftslogik des Endpunkts ab. Berücksichtigen Sie auch die Tatsache, dass Anfragen von mehreren API-Clients um die Ressourcen konkurrieren. Eine API ist verwundbar, wenn mindestens eine der folgenden Grenzwerte fehlt oder unangemessen eingestellt ist (z. B. zu niedrig/hoch):

- Ausführungszeitüberschreitungen
- Maximal zuweisbarer Speicher
- Anzahl der Dateideskriptoren
- Anzahl der Prozesse
- Größe der Nutzlast von Anfragen (z. B. Uploads)
- Anzahl der Anfragen pro Client/Ressource
- Anzahl der Datensätze pro Seite, die in einer einzigen Anfrageantwort zurückgegeben werden sollen

Beispiele für Angriffe

Szenario #1

Ein Angreifer lädt ein großes Bild hoch, indem er eine POST-Anfrage an `/api/v1/images` stellt. Wenn der Upload abgeschlossen ist, erstellt die API mehrere Miniaturbilder mit unterschiedlichen Größen. Aufgrund der Größe des hochgeladenen Bildes ist der verfügbare Speicher während der Erstellung der Miniaturansichten erschöpft und die API reagiert nicht mehr.

Szenario #2

Gegeben ist eine Anwendung, die die Benutzerliste auf einer Benutzeroberfläche mit einer Begrenzung von 200 Benutzer pro Seite anzeigt. Die Benutzerliste wird mit folgender Abfrage vom Server abgerufen: `/api/users?page=1&size=200`. Ein Angreifer ändert den Parameter `size` Parameter auf `200.000`, was zu Leistungsproblemen in der Datenbank führt. In der Zwischenzeit

reagiert die API nicht mehr und ist nicht mehr in der Lage, weitere Anfragen von diesem oder anderen Clients zu beantworten (DoS).

Das gleiche Szenario kann verwendet werden, wenn ein Integer oder Buffer Overflow auftritt.

Vorbeugende Maßnahmen

- Docker macht es einfach Speicher, CPU, Anzahl der Neustarts, Dateideskriptoren und Prozesse zu beschränken.
- Implementieren Sie ein Limit, wie oft ein Client die API innerhalb eines bestimmten Zeitrahmen abrufen kann.
- Benachrichtigen Sie den Client, wenn das Limit überschritten wird, indem Sie die Limitnummer und den Zeitpunkt, zu dem das Limit zurückgesetzt wird bekanntgeben.
- Hinzufügen einer ordnungsgemäßen serverseitigen Validierung für Query String und Request Body Parameter, insbesondere denjenigen, die die Anzahl der in der Antwort zurückzugebenden Werte betimmen.
- Definieren und erzwingen Sie die maximale Größe von Daten für alle eingehenden Parameter und und Nutzdaten, wie z. B. die maximale Länge für Strings und die maximale Anzahl von Elementen in Arrays.

Referenzen

OWASP

- Blocking Brute Force Attacks
- Docker Cheat Sheet - Limit resources (memory, CPU, file descriptors, processes, restarts)
- REST Assessment Cheat Sheet

Extern

- CWE-307: Improper Restriction of Excessive Authentication Attempts
- CWE-770: Allocation of Resources Without Limits or Throttling
- “*Rate Limiting (Throttling)*” - Security Strategies for Microservices-based Application Systems, NIST

API5:2019 Broken Function Level Authorization

Bedrohungsakteure/Angriffsvektoren	**Sicherheitslücken**	**Auswirkungen**
API-spezifisch : Ausnutzbarkeit **3**	Häufigkeit **2** : Erkennbarkeit **1**	Komplexität **2** : Unternehmensspezifisch
Um eine Schwachstelle auszunutzen, muss der Angreifer legitime API-Aufrufe an einen Endpunkt senden, auf den er eigentlich keinen Zugriff haben sollte. Diese Endpunkte könnten für anonyme oder nicht privilegierte Benutzer zugänglich sein. Es ist einfacher, solche Schwachstellen in APIs zu entdecken, da sie besser strukturiert sind und der Weg, um auf bestimmte Funktionen zuzugreifen, vorhersehbarer ist (z. B. indem man die HTTP-Methode von GET auf PUT ändert oder den Teil `users` in der URL in `admins` ändert).	Berechtigungsprüfungen für eine Funktion oder Ressource werden in der Regel über die Konfiguration und manchmal auch auf Code-Ebene verwaltet. Die Implementierung geeigneter Prüfungen kann eine verwirrende Aufgabe sein, da moderne Anwendungen viele Arten von Rollen oder Gruppen und komplexe Benutzerhierarchien (z. B. Unterbenutzer, Benutzer mit mehr als einer Rolle) enthalten können.	Solche Schwachstellen ermöglichen es Angreifern, auf nicht autorisierte Funktionen zuzugreifen. Verwaltungsfunktionen sind wichtige Ziele für diese Art von Angriffen.

Ist die API angreifbar?

Die beste Möglichkeit, um Schwachstellen in der Funktionsautorisierung zu finden, besteht darin, eine tiefgreifende Analyse des Autorisierungsmechanismus durchzuführen, wobei die Benutzerhierarchie, verschiedene Rollen oder Gruppen in der Anwendung berücksichtigt werden und die folgenden Fragen gestellt werden:

- Kann ein normaler Benutzer auf administrative Endpunkte zugreifen?
- Kann ein Benutzer sensible Aktionen (z. B. Erstellung, Änderung oder Löschung) durchführen, auf die er keinen Zugriff haben sollte, indem er einfach die HTTP-Methode ändert (z. B. von `GET` auf `DELETE`)?
- Kann ein Benutzer aus Gruppe X auf eine Funktion zugreifen, die nur Benutzern aus Gruppe Y zugänglich sein sollte, indem er einfach die Endpunkt-URL und Parameter errät (z. B. `/api/v1/users/export_all`)?

Nehmen Sie nicht an, dass ein API-Endpunkt nur aufgrund des URL-Pfads ein regulärer oder administrativer Endpunkt ist.

Obwohl Entwickler dazu neigen, die meisten administrativen Endpunkte unter einem bestimmten relativen Pfad wie `api/admins` zu platzieren, ist es sehr häufig, dass diese administrativen Endpunkte zusammen mit regulären Endpunkten unter anderen relativen Pfaden wie `api/users` zu finden sind.

Beispiele für Angriffe

Szenario #1

Während des Registrierungsprozesses für eine Anwendung, die nur eingeladenen Nutzern die Teilnahme erlaubt beitreten können, löst die mobile Anwendung einen API-Aufruf aus an `GET /api/invites/{invite_guid}`. Die Antwort enthält ein JSON mit Details über die Einladung, einschließlich der Rolle des Benutzers und seiner E-Mail.

Ein Angreifer duplizierte die Anfrage und manipulierte die HTTP-Methode und den Endpunkt zu `POST /api/invites/new`. Dieser Endpunkt sollte nur von Administratoren Administratoren zugreifen, die die Verwaltungskonsole verwenden, die keine Funktions Berechtigungsprüfungen implementiert.

Der Angreifer nutzt das Problem aus und sendet sich selbst eine Einladung zur Erstellung eines Administratorkonto zu erstellen:

```
POST /api/invites/new {“email”:”hugo@malicious.com”,”role”:”admin”}
```

Szenario #2

Eine API enthält einen Endpunkt, der nur für Administratoren zugänglich sein sollte - `GET /api/admin/v1/users/all`. Dieser Endpunkt liefert die Details aller Benutzer der Anwendung zurück und führt keine Berechtigungsprüfungen auf Funktionsebene durch. Ein Angreifer, der die Struktur der API kennt, errät den Zugriff auf diesen Endpunkt und erlangt dadurch Zugang zu sensiblen Details der Anwendungsnutzer.

Vorbeugende Maßnahmen

Ihre Anwendung sollte ein konsistentes und leicht analysierbares Autorisierungsmodul haben, das von allen Geschäftsfunktionen aufgerufen wird. Häufig wird diese Sicherheit durch eine oder mehrere Komponenten außerhalb des Anwendungscodes bereitgestellt.

- Die Durchsetzungsmechanismen sollten standardmäßig jeden Zugriff verweigern und explizite Genehmigungen für bestimmte Rollen für den Zugriff auf jede Funktion erfordern.
- Überprüfen Sie Ihre API-Endpunkte auf Mängel bei der Autorisierung auf Funktionsebene und berücksichtigen Sie dabei die Geschäftslogik der Anwendung und die Hierarchie der Gruppen.
- Stellen Sie sicher, dass alle Ihre administrativen Controller von einem abstrakten administrativen Controller erben, der Berechtigungsprüfungen auf Grundlage der Gruppe/Rolle des Benutzers implementiert.
- Stellen Sie sicher, dass administrative Funktionen innerhalb eines regulären Controllers Berechtigungsprüfungen auf Grundlage der Gruppe und Rolle des Benutzers implementieren.

References

OWASP

- OWASP Article on Forced Browsing
- OWASP Top 10 2013-A7-Missing Function Level Access Control
- OWASP Development Guide: Chapter on Authorization

Externe Quellen

- CWE-285: Improper Authorization

API6:2019 - Mass Assignment

Bedrohungsakteure/Angriffsvektoren	**Sicherheitslücken**	**Auswirkungen**
API-spezifisch : Ausnutzbarkeit **2**	Häufigkeit **2** : Erkennbarkeit **2**	Komplexität **2** : Unternehmensspezifisch
Die Ausnutzung erfordert in der Regel ein Verständnis der Geschäftslogik, der Beziehungen zwischen den Objekten und der API-Struktur. Die Ausnutzung von "Mass Asignment" ist in APIs einfacher, da sie aufgrund ihrer Konzeption die zugrundeliegende Implementierung der Anwendung zusammen mit den Namen der Eigenschaften offenlegen.	Moderne Frameworks ermutigen Entwickler zur Verwendung von Funktionen, die automatisch Eingaben vom Client in Codevariablen und interne Objekte binden. Angreifer können diese Methode nutzen, um die Eigenschaften sensibler Objekte zu aktualisieren oder zu überschreiben, die die Entwickler nie offenlegen wollten.	Die Ausnutzung kann zu einer Ausweitung von Privilegien, Datenmanipulation, Umgehung von Sicherheitsmechanismen und vielem mehr führen.

Ist die API angreifbar?

Moderne Anwendungen können viele Objekteigenschaften enthalten. Einige dieser Eigenschaften sollten direkt vom Client aktualisiert werden (z. B. `user.first_name` oder `user.address`), während andere nicht aktualisiert werden sollten (z. B. `user.is_vip-Flag`).

Ein API-Endpunkt ist anfällig, wenn er automatisch Client-Parameter in interne Objekteigenschaften konvertiert, ohne dabei die Sensibilität und das Gefährdungspotential dieser Eigenschaften zu berücksichtigen. Dies könnte einem Angreifer ermöglichen, Objekteigenschaften zu aktualisieren, auf die er keinen Zugriff haben sollte.

Beispiele für sensible Eigenschaften:

- **Permission-related properties**: `user.is_admin`, `user.is_vip` sollte nur nur von Administratoren gesetzt werden.
- **Process-dependent properties**: `user.cash` sollte nur intern gesetzt werden nach der Zahlungsüberprüfung gesetzt werden.
- **Internal properties**: `article.created_time` sollte nur intern gesetzt werden durch die Anwendung gesetzt werden.

Beispiele für Angriffe

Szenario #1

Eine Anwendung für Mitfahrgelegenheiten bietet einem Nutzer die Möglichkeit, grundlegende Informationen für sein Profil zu bearbeiten. Während dieses Prozesses wird ein API-Aufruf gesendet an `PUT /api/v1/users/me` mit dem folgenden legitimen JSON-Objekt:

```
{"user_name":"inons","age":24}
```

Die Anfrage `GET /api/v1/users/me` enthält eine zusätzlich ein `credit_balance` Attribut:

```
{"user_name":"inons","age":24,"credit_balance":10}
```

Der Angreifer wiederholt die erste Anfrage mit der folgendem Payload:

```
{"user_name":"attacker","age":60,"credit_balance":99999}
```

Da der Endpunkt anfällig für "MAss Assignment" ist, erhält der Angreifer Credits, ohne zu bezahlen.

Szenario #2

Ein Portal zur gemeinsamen Nutzung von Videos ermöglicht es Nutzern, Inhalte hochzuladen und in verschiedenen Formaten herunterzuladen. Ein Angreifer, der die API erforscht, fand heraus, dass der Endpunkt `GET /api/v1/videos/{video_id}/meta_data` ein JSON-Objekt mit den Eigenschaften des Videos zurückgibt. Eine der Eigenschaften ist `"mp4_conversion_params":"-v codec h264"`, was darauf hinweist, dass die Anwendung einen Shell-Befehl zur Konvertierung des Videos verwendet.

Der Angreifer fand auch heraus, dass der Endpunkt `POST /api/v1/videos/new` für "Mass Asignment" anfällig ist und es dem Client ermöglicht, eine beliebige Eigenschaft des Videoobjekts festzulegen. Der Angreifer setzt einen bösartigen Wert wie folgt: `"mp4_conversion_params":"-v codec h264 && format C:/"`. Dieser Wert verursacht eine "Remote Code Execution", sobald der Angreifer das Video als MP4 herunterlädt.

Vorbeugende Maßnahmen

- Vermeiden Sie nach Möglichkeit die Verwendung von Funktionen, die automatisch die Eingaben eines Benutzers in Variablen oder interne Objekte speicher, ohne diese zu validieren.
- Setzen Sie nur die Eigenschaften auf die Whitelist, die vom Client aktualisiert werden sollen.
- Verwenden Sie integrierte Funktionen, um Eigenschaften, auf die der Client nicht zugreifen darf, zu blockieren.
- Definieren und Erzwingen Sie gegebenenfalls explizit Schemata für die Eingabedaten.

Referenzen

Externe Quellen

- CWE-915: Improperly Controlled Modification of Dynamically-Determined Object Attributes

API7:2019 Security Misconfiguration

Bedrohungsakteure/Angriffsvektoren	**Sicherheitslücken**	**Auswirkungen**
API-spezifisch : Ausnutzbarkeit **3**	Häufigkeit **3** : Erkennbarkeit **3**	Komplexität **2** : Unternehmensspezifisch
Angreifer versuchen oft, ungepatchte Schwachstellen, gemeinsame Endpunkte oder ungeschützte Dateien und Verzeichnisse zu finden, um sich unbefugten Zugriff auf das System zu verschaffen oder Kenntnisse über das System zu erlangen.	Eine Sicherheitskonfiguration kann auf jeder Ebene des API-Stacks unsicher sein, von der Netzwerkebene bis zur Anwendungsebene. Es gibt automatisierte Tools, die dazu dienen, Fehlkonfigurationen wie unnötige Dienste oder veraltete Optionen zu erkennen und auszunutzen.	Unsichere Sicherheitskonfigurationen können nicht nur sensible Benutzerdaten, sondern auch Systemdetails preisgeben, die zu einer vollständigen Kompromittierung des Servers führen können.

Ist die API angreifbar?

Die API könnte anfällig sein, wenn:

- Es fehlt eine angemessene Sicherheitshärtung, sei es auf einer beliebigen Ebene des Anwendungs-Stacks oder wenn die Berechtigungen für Cloud-Services falsch konfiguriert sind.
- Die neuesten Sicherheitspatches fehlen oder das System veraltet ist.
- Unnötige Funktionen aktiviert sind (z. B. HTTP-Methoden).
- Keine Transport Layer Security (TLS) eingesetzt wird.
- Sicherheitsrichtlinien werden den Clients nicht übermittelt (z. B. Security Headers).
- Eine Cross-Origin Resource Sharing (CORS)-Richtlinie fehlt oder falsch konfiguriert ist.
- Fehlermeldungen, Stack-Traces enthalten oder andere sensible Informationen offengelegt werden.

Beispiele für Angriffe

Szenario #1

Ein Angreifer findet die Datei `.bash_history` unter dem Stammverzeichnis des Servers, die Befehle enthält, die vom DevOps-Team für den Zugriff auf die API verwendet werden:

```
$ curl -X GET 'https://api.server/endpoint/' -H 'authorization: Basic Zm9vOmJhcg=='
```

Ein Angreifer könnte auch neue Endpunkte für die API identifizieren, die nur vom DevOps-Team verwendet werden und nicht dokumentiert sind.

Szenario #2

Um einen bestimmten Dienst ins Visier zu nehmen, verwendet ein Angreifer eine beliebte Suchmaschine, um nach Computern

zu suchen. Der Angreifer fand einen Host, auf dem ein beliebtes Datenbankmanagementsystem auf dem Standardport läuft. Der Host verwendete die Standardkonfiguration, bei der die Authentifizierung standardmäßig deaktiviert ist. Der Angreifer erlangte Zugriff auf Millionen von Datensätzen mit personenbezogenen Daten, persönlichen Präferenzen und Authentifizierungsdaten.

Szenario #3

Bei der Untersuchung des Datenverkehrs einer mobilen Anwendung stellt ein Angreifer fest, dass nicht der gesamte HTTP-Verkehr über ein sicheres Protokoll (z. B. TLS) abgewickelt wird. Der Angreifer findet dies insbesondere für den Download von Profilbildern heraus. Da die Daten binär kodiert sind, findet der Angreifer, ein Muster in der Größe der API-Antworten, das er nutzt, um die Präferenzen der Benutzer bezüglich des dargestellten Inhalts (z. B. Profilbilder) auszulesen.

Vorbeugende Maßnahmen

Der API-Lebenszyklus sollte Folgendes umfassen:

- Ein wiederholbarer Härtungsprozess, der zu einer schnellen und einfachen Bereitstellung einer ordnungsgemäß gehärteten Umgebung führt.
- Die Überprüfung und Aktualisierung von Konfigurationen für den gesamten API-Stack. Die Überprüfung sollte Folgendes umfassen: Orchestrierungsdateien, API-Komponenten und Cloud-Dienste (z. B. S3-Bucket-Berechtigungen).
- Ein sicherer Kommunikationskanal für alle API-Interaktionen, einschließlich des Zugriffs auf statische Assets (z. B. Bilder).
- Ein automatisierter Prozess zur kontinuierlichen Bewertung der Effektivität der Konfiguration und Einstellungen in allen Umgebungen.

Darüber hinaus:

- Um zu verhindern, dass Exception-Traces und andere wertvolle Informationen an Angreifer zurückgesendet werden, sollten, falls zutreffend, alle API-Antwort-Schemata definiert und erzwungen werden, einschließlich Fehlermeldungen.
- Stellen Sie sicher, dass auf die API nur mit den angegebenen HTTP-Verben zugegriffen werden kann. Alle anderen HTTP-Verben sollten deaktiviert werden (z. B. `HEAD`).
- APIs, auf die von browserbasierten Clients zugegriffen werden soll (z. B. WebApp Front-End), sollten eine angemessene Cross-Origin Resource Sharing (CORS)-Richtlinie implementieren.

Referenzen

OWASP

- OWASP Secure Headers Project
- OWASP Testing Guide: Configuration Management
- OWASP Testing Guide: Testing for Error Codes
- OWASP Testing Guide: Test Cross Origin Resource Sharing

Externe Quellen

- CWE-2: Environmental Security Flaws
- CWE-16: Configuration
- CWE-388: Error Handling
- Guide to General Server Security, NIST
- Let’s Encrypt: a free, automated, and open Certificate Authority

API8:2019 Injection

Bedrohungsakteure/Angriffsvektoren	Sicherheitslücken	Auswirkungen
API-spezifisch : Ausnutzbarkeit **3**	Häufigkeit **2** : Erkennbarkeit **3**	Komplexität **3** : Unternehmensspezifisch
Angreifer füttern die API mit bösartigen Daten über alle verfügbaren Injektionsvektoren (z. B. direkte Eingabe, Parameter, integrierte Dienste usw.) und erwarten, dass diese an einen Interpreter gesendet werden.	Injection-Fehler sind sehr häufig und werden oft in SQL-, LDAP- oder NoSQL-Abfragen, Betriebssystembefehlen, XML-Parsern und ORM gefunden. Diese Schwachstellen sind bei der Überprüfung des Quellcodes leicht zu entdecken. Angreifer können Scanner und Fuzzer verwenden.	Ein "Injection"-Angriff kann zur Offenlegung von Informationen und zum Datenverlust führen. Es kann auch zu DoS oder einer vollständigen Übernahme des Hosts kommen.

Ist die API angreifbar?

Die API ist anfällig für Injection-Angriffe, wenn:

- Die von den Clients gelieferten Daten nicht von der API validiert, gefiltert oder bereinigt werden.
- Die von den Clients gelieferten Daten direkt für SQL/NoSQL/LDAP-Abfragen, OS-Befehle, XML-Parser und Object Relational (ORM) / Object Document Mapping (ODM) verwendet oder konkateniert werden.
- Daten von externen Systemen (z.B. integrierten Systemen) nicht von der API validiert, gefiltert oder bereinigt werden.

Beispiele für Angriffe

Szenario #1

Die Firmware einer Kindersicherungseinrichtung stellt den Endpunkt `/api/CONFIG/restore` zur Verfügung, der die Übermittlung einer appId als Multipart Parameter erwartet. Mit Hilfe eines Decompilers findet ein Angreifer heraus, dass die appId direkt in einen Systemaufruf ohne jegliche Validierung übergeben wird:

```
snprintf(cmd, 128, "%srestore_backup.sh /tmp/postfile.bin %s %d",
        "/mnt/shares/usr/bin/scripts/", appid, 66);
system(cmd);
```

Mit dem folgenden Befehl kann der Angreifer jedes Gerät mit der gleichen verwundbaren Firmware ausschalten:

```
$ curl -k "https://${deviceIP}:4567/api/CONFIG/restore" -F 'appid=$(/etc/pod/power_down.sh)'
```

Szenario #2

Wir haben eine Anwendung mit grundlegenden CRUD-Funktionen für Operationen mit Buchungen. Ein Angreifer konnte

herausfinden, dass eine NoSQL-Injection möglich ist, welche durch den Query-String-Parameter `bookingId` in der Anfrage zum Löschen von Buchungen besteht. Die Anfrage sieht folgendermaßen aus: `DELETE /api/bookings?bookingId=678`.

Der API-Server verwendet die folgende Funktion zur Bearbeitung von Löschanfragen:

```
router.delete('/bookings', async function (req, res, next) {
  try {
    const deletedBooking = await Bookings.findOneAndRemove({'_id' : req.query.bookingId});
    res.status(200);
  } catch (err) {
   res.status(400).json({error: 'Unexpected error occured while processing a request'});
  }
});
```

Der Angreifer hat die Anfrage abgefangen und den Abfrageparameter `bookingId` wie unten gezeigt geändert. In diesem Fall gelang es dem Angreifer, die Buchung eines anderen Benutzers zu löschen:

```
DELETE /api/bookings?bookingId[$ne]=678
```

Vorbeugende Maßnahmen

Um Injection-Angriffe zu verhindern müssen Daten von Befehlen und Abfragen getrennt werden.

- Führen Sie eine Datenvalidierung mit einer einzigen, vertrauenswürdigen und aktiv gepflegten Bibliothek durch.
- Validieren, filtern und bereinigen Sie alle vom Client bereitgestellten Daten oder andere Daten, die von integrierten Systemen kommen.
- Sonderzeichen sollten unter Verwendung der spezifischen Syntax für den Zielinterpreter escaped werden.
- Bevorzugen Sie eine sichere API, die eine parameterisierte Schnittstelle bereitstellt.
- Begrenzen Sie immer die Anzahl der zurückgegebenen Datensätze, um einen Datenverlust im Falle von erfolgreichen Injection-Angriffen zu verhindern.
- Validieren Sie eingehende Daten mithilfe ausreichender Filter, um nur gültige Werte für jeden Eingabeparameter zuzulassen.
- Definieren Sie Datentypen und strenge Muster für alle String-Parameter.

Referenzen

OWASP

- OWASP Injection Flaws
- SQL Injection
- NoSQL Injection Fun with Objects and Arrays
- Command Injection

Externe Quellen

- CWE-77: Command Injection
- CWE-89: SQL Injection

API9:2019 Improper Assets Management

Bedrohungsakteure/Angriffsvektoren	Sicherheitslücken	Auswirkungen
API-spezifisch : Ausnutzbarkeit **3**	Häufigkeit **3** : Erkennbarkeit **2**	Komplexität **2** : Unternehmensspezifisch
Alte API-Versionen sind in der Regel nicht gepatcht und bieten daher eine einfache Möglichkeit, Systeme zu kompromittieren, ohne sich mit modernsten Sicherheitsmechanismen auseinandersetzen zu müssen, die zum Schutz der neuesten API-Versionen implementiert wurden.	Veraltete Dokumentation erschwert die Suche nach Schwachstellen und die Behebung dieser. Das Fehlen einer Übersicht aller Systeme und einer Abschalt-Strategie führt zu ungepatchten Systemen, was zu einer Offenlegung sensibler Daten führen kann. Es ist häufig anzutreffen, dass API-Hosts unnötigerweise exponiert werden, da moderne Konzepte wie Microservices eine einfache Bereitstellung und Unabhängigkeit von Anwendungen ermöglichen (z. B. Cloud Computing, k8s).	Angreifer können über alte, ungepatchte API-Versionen, die mit derselben Datenbank verbunden sind, Zugriff auf sensible Daten erlangen oder sogar den Server übernehmen.

Ist die API angreifbar?

Die API könnte verwundbar sein, wenn:

- Der Zweck eines API-Hosts unklar ist und es keine klaren Antworten auf folgende Fragen gibt:
- In welcher Umgebung wird die API ausgeführt (z. B. Produktion, Staging, Test, Entwicklung)?
- Wer sollte Netzwerkzugriff auf die API haben (z. B. öffentlich, intern, Partner)?
- Welche API-Version wird ausgeführt?
- Welche Daten werden von der API erfasst und verarbeitet (z. B. PII)?
- Wie verläuft der Datenfluss?
- Es gibt keine Dokumentation oder die vorhandene Dokumentation ist veraltet.
- Es gibt keinen Plan zur Ausmusterung jeder API-Version.
- Die Dienstübericht der Hosts fehlt oder ist veraltet.
- Die Übersicht der integrierten Dienste, sowohl von Erst- als auch von Drittanbietern, fehlt oder ist veraltet.
- Alte oder frühere API-Versionen werden ungepatcht ausgeführt.

Beispiele für Angriffe

Szenario #1

Nach der Neugestaltung ihrer Anwendungen ließ ein lokaler Suchdienst eine alte API-Version (`api.someservice.com/v1`) weiterlaufen, ohne Schutz und mit Zugriff auf die Benutzerdatenbank. Während eines Angriffs auf eine der neuesten

veröffentlichten Anwendungen fand ein Angreifer die API-Adresse (`api.someservice.com/v2`). Durch das Ersetzen von `v2` in der URL durch `v1` erhielt der Angreifer Zugriff auf die alte, ungeschützte API und konnte personenbezogene Daten (PII) von über 100 Millionen Nutzern offenlegen.

Szenario #2

Ein soziales Netzwerk hat einen Mechanismus zum Rate-Limiting eingeführt, der Angreifer daran hindert, mittels Brute-Force-Attacken zurückgesetzte Passwort-Token zu erraten. Dieser Mechanismus wurde nicht als Teil des API-Codes selbst implementiert, sondern in einer separaten Komponente zwischen dem Client und der offiziellen API (`www.socialnetwork.com`). Ein Forscher fand einen Beta-API-Host (`www.mbasic.beta.socialnetwork.com`), der dieselbe API ausführte, einschließlich des Mechanismus zum Zurücksetzen des Passworts, aber der Mechanismus zum Rate-Limiting war nicht implementiert. Der Forscher konnte das Passwort jedes Benutzers zurücksetzen, indem er mittels einer einfachen Brute-Force-Methode den 6-stelligen Token erraten hat.

Vorbeugende Maßnahmen

- Inventarisieren Sie alle API-Hosts und dokumentieren Sie wichtige Aspekte. Konzentrieren Sie sich dabei auf die API-Umgebung (z. B. Produktion, Staging, Test, Entwicklung), wer Netzwerkzugriff auf den Host haben sollte (z. B. öffentlich, intern, Partner) und die API-Version.
- Inventarisieren Sie integrierte Dienste und dokumentieren Sie wichtige Aspekte wie ihre Rolle im System, welche Daten ausgetauscht werden (Datenfluss) und ihre Sensibilität.
- Dokumentieren Sie alle Aspekte Ihrer API wie Authentifizierung, Fehler, Umleitungen, Ratenbegrenzung, Cross-Origin Resource Sharing (CORS)-Richtlinien und Endpunkte, einschließlich ihrer Parameter, Anfragen und Antworten.
- Generieren Sie die Dokumentation automatisch durch die Annahme von offenen Standards. Fügen Sie den Dokumentationsaufbau in Ihre CI/CD-Pipeline ein.
- Stellen Sie die API-Dokumentation nur denjenigen zur Verfügung, die berechtigt sind, die API zu nutzen.
- Verwenden Sie externe Schutzmaßnahmen wie API-Sicherheits-Firewalls für alle freigegebenen Versionen Ihrer APIs, nicht nur für die aktuelle Produktionsversion.
- Vermeiden Sie die Verwendung von Produktionsdaten bei nicht-produktionsbezogenen API-Bereitstellungen. Wenn dies unvermeidlich ist, sollten diese Endpunkte die gleiche Sicherheitsbehandlung wie die Produktionsendpunkte erhalten.
- Wenn neuere Versionen von APIs Sicherheitsverbesserungen enthalten, führen Sie eine Risikoanalyse durch, um die Entscheidung über die erforderlichen Maßnahmen zur Risikominderung für die ältere Version zu treffen: zum Beispiel, ob es möglich ist, die Verbesserungen rückwärtskompatibel zu machen, ohne die API-Kompatibilität zu beeinträchtigen, oder ob Sie schnell zur älteren Version wechseln und alle Clients zwingen müssen, zur neuesten Version zu wechseln.

Referenzen

Externe Quellen

- CWE-1059: Incomplete Documentation
- OpenAPI Initiative

API10:2019 Insufficient Logging & Monitoring

Bedrohungsakteure/Angriffsvektoren	**Sicherheitslücken**	**Auswirkungen**
API-spezifisch : Ausnutzbarkeit **2**	Häufigkeit **3** : Erkennbarkeit **1**	Komplexität **2** : Unternehmensspezifisch
Angreifer nutzen die fehlende Protokollierung und Überwachung aus, um Systeme unbemerkt zu missbrauchen.	Ohne Protokollierung und Überwachung oder mit unzureichender Protokollierung und Überwachung ist es fast unmöglich, verdächtige Aktivitäten zu verfolgen und rechtzeitig darauf zu reagieren.	Ohne Einblick in laufende bösartige Aktivitäten haben Angreifer genügend Zeit, um Systeme vollständig zu kompromittieren.

Ist die API angreifbar?

Die API ist angreifbar, wenn:

- Sie keine keine Protokolle erzeugt, die Protokollierungsstufe nicht korrekt eingestellt ist oder die Protokoll Nachrichten nicht genügend Details enthalten.
- Die Integrität der Protokolle nicht gewährleistet ist(z. B. Log Injection).
- Die Protokolle nicht kontinuierlich überwacht werden.
- Die API-Infrastruktur nicht kontinuierlich überwacht wird.

Beispiele für Angriffe

Szenario #1

Die Zugriffsschlüssel einer administrativen API wurden in einem öffentlichen Repository veröffentlicht. Der Repository-Eigentümer wurde per E-Mail über das mögliche Datenleck informiert, brauchte aber mehr als 48 Stunden, um auf den Vorfall zu reagieren. Die Veröffentlichung der Zugangsschlüssel hat eventuell Zugang zu sensiblen Daten ermöglicht. Aufgrund der unzureichenden Protokollierung kann das Unternehmen nicht beurteilen, auf welche Daten zugegriffen wurde.

Szenario #2

Eine Video-Sharing-Plattform wurde von einem "groß angelegten" Credential Stuffing-Angriff getroffen. Obwohl fehlgeschlagene Anmeldungen protokolliert wurden, wurde in der Zeitspanne des Angriffs kein Alarm ausgelöst. Als Reaktion auf Nutzerbeschwerden wurden die API-Protokolle analysiert und der Angriff wurde entdeckt. Das Unternehmen musste eine öffentliche Ankündigung machen, in der es die Nutzer aufforderte ihre Passwörter zurückzusetzen. Außerdem musste das Unternehmen den Vorfall den Aufsichtsbehörden melden.

Vorbeugende Maßnahmen

- Protokollieren Sie alle fehlgeschlagenen Authentifizierungsversuche, verweigerten Zugriff und

Eingabevalidierungsfehler.

- Die Protokolle sollten in einem Format geschrieben werden, das von einer Log-Management-Lösung verarbeitet werden kann, und sie sollten genügend Details enthalten, um den Angreifer zu identifizieren.
- Protokolle sollten als sensible Daten behandelt werden, und ihre Integrität sollte stets gewährleistet sein.
- Konfigurieren Sie ein Überwachungssystem zur kontinuierlichen Überwachung der Infrastruktur, des Netzwerks und der Funktionsweise der API.
- Verwenden Sie ein Security Information and Event Management (SIEM), um Protokolle aller Komponenten der API und der Hosts zu sammeln und zu verwalten.
- Konfigurieren Sie benutzerdefinierte Dashboards und Warnmeldungen, um verdächtige Aktivitäten zu erkennen und darauf reagieren zu können.

Referenzen

OWASP

- OWASP Logging Cheat Sheet
- OWASP Proactive Controls: Implement Logging and Intrusion Detection
- OWASP Application Security Verification Standard: V7: Error Handling and Logging Verification Requirements

Externe Quellen

- CWE-223: Omission of Security-relevant Information
- CWE-778: Insufficient Loggin

Was kommt als Nächstes für Entwickler?

Die Aufgabe, sichere Software zu erstellen und zu pflegen oder bestehende Software zu reparieren, kann schwierig sein. Bei APIs ist das nicht anders.

Wir glauben, dass Fortbildung und Bewusstsein die Schlüsselfaktoren für das Schreiben sicherer Software sind. Alles andere, was erforderlich ist, um das Ziel zu erreichen, hängt ab von **Einführung und Verwendung von wiederholbaren Sicherheitsprozessen und standardisierten Sicherheitskontrollen**.

OWASP hat seit Beginn des Projekts zahlreiche freie und offene Ressourcen zum Thema Sicherheit vorgestellt. Bitte besuchen Sie die OWASP-Projektseite für eine umfassende Liste der verfügbaren Projekte.

Bildung	Sie können je nach Beruf und Interesse mit dem Lesen von OWASP Education Project materials beginnen. Für praktisches Lernen haben wir **crAPI** - **C**ompletely **R**idiculous **API** auf unsere Roadmap gesetzt. In der Zwischenzeit können Sie WebAppSec mit dem OWASP DevSlop Pixi Module üben, einem verwundbaren WebApp- und API-Service, der Benutzern beibringen soll, wie man moderne Webanwendungen und APIs auf Sicherheitsprobleme testet und wie man in Zukunft sicherere APIs schreibt. Sie können auch an den Schulungssitzungen der OWASP AppSec Conference teilnehmen oder Ihrem lokalen Verband beitreten.
Sicherheitsanforderungen	Sicherheit sollte von Anfang an Teil eines jeden Projekts sein. Bei der Anforderungserhebung ist es wichtig zu definieren, was "sicher" für dieses Projekt bedeutet. OWASP empfiehlt, den OWASP Application Security Verification Standard (ASVS) als Leitfaden für die Festlegung der Sicherheitsanforderungen zu verwenden. Wenn Sie ein Projekt auslagern, sollten Sie den OWASP Secure Software Contract Annex verwenden, der entsprechend den lokalen Gesetzen und Vorschriften angepasst werden sollte.
Sicherheitsarchitektur	Sicherheit sollte während aller Projektphasen ein Thema bleiben. Die OWASP Prevention Cheat Sheets sind ein guter Ausgangspunkt für Anleitungen, wie man Sicherheit schon in der Architekturphase einplant. Unter vielen anderen finden Sie das REST Security Cheat Sheet und das REST Assessment Cheat Sheet.
Standard-Sicherheitskontrollen	Die Übernahme von Standard-Sicherheitskontrollen verringert das Risiko, beim Schreiben Ihrer eigenen Logik Sicherheitslücken einzuführen. Trotz der Tatsache, dass viele moderne Frameworks mittlerweile über eingebaute effektive Standardkontrollen verfügen, gibt OWASP Proactive Controls einen guten Überblick darüber, welche Sicherheitskontrollen Sie in Ihr Projekt einbauen sollten. OWASP stellt auch einige

	Bibliotheken und Tools zur Verfügung, die für Sie nützlich sein können, wie z.B. Validierungskontrollen.
Secure Software Development Life Cycle	Sie können das OWASP Software Assurance Maturity Model (SAMM) verwenden, um den Prozess bei der Erstellung von APIs zu verbessern. Mehrere andere OWASP-Projekte stehen zur Verfügung, um Sie in den verschiedenen Phasen der API-Entwicklung zu unterstützen, z. B. das OWASP Code Review Project.

Was kommt als Nächstes für DevSecOps?

Aufgrund ihrer Bedeutung in modernen Anwendungsarchitekturen ist der Aufbau sicherer APIs von entscheidender Bedeutung. Die Sicherheit darf nicht vernachlässigt werden und sollte Teil des gesamten Entwicklungslebenszyklus sein. Jährliches Scannen und Penetrationstests sind nicht länger ausreichend.

DevSecOps sollte die Entwicklung begleiten und kontinuierliche Sicherheitstests über den gesamten Lebenszyklus der Softwareentwicklung hinweg durchführen. Ihr Ziel ist es die Entwicklungspipeline durch Sicherheitsautomatisierung zu verbessern, ohne die Geschwindigkeit der Entwicklung zu beeinträchtigen.

Im Zweifelsfall sollten Sie auf dem Laufenden bleiben und das DevSecOps Manifesto lesen.

Verstehen Sie das Bedrohungsmodell	Testprioritäten ergeben sich aus einem Bedrohungsmodell. Wenn Sie keins haben, können Sie den OWASP Application Security Verification Standard (ASVS) und die OWASP Testing Guide als Grundlage verwenden. Die Einbeziehung des Entwicklungsteams kann helfen, das Sicherheitsbewusstsein zu stärken.
Verstehen Sie den SDLC	Nehmen Sie am Entwicklungsteam teil, um den Software Development Life Cycle besser zu verstehen. Ihr Beitrag zu kontinuierlichen Sicherheitstests sollte mit Menschen, Prozessen und Werkzeugen kompatibel sein. Jeder sollte mit dem Prozess einverstanden sein, so dass es keine unnötigen Reibungen oder Widerstände gibt.
Teststrategien	Da Ihre Arbeit die Entwicklungsgeschwindigkeit nicht beeinträchtigen sollte, sollten Sie mit Bedacht die beste (einfachste, schnellste, genaueste) Technik zur Überprüfung der Sicherheitsanforderungen wählen. Das OWASP Security Knowledge Framework und der OWASP Application Security Verification Standard können gute Quellen für funktionale und nichtfunktionale Sicherheitsanforderungen sein. Es gibt weitere hervorragende Quellen für Projekte und Tools, ähnlich wie die von der DevSecOps-Community angebotenen.
Erreichen von Abdeckung und Genauigkeit	Sie sind die Brücke zwischen den Entwicklern und den Betriebsteams. Um eine gute Abdeckung zu erreichen, sollten Sie sich nicht nur auf die Funktionalität, sondern auch auf die Orchestrierung konzentrieren. Arbeiten Sie von Anfang an eng mit den Entwicklungs- und Betriebsteams zusammen, damit Sie Ihre Zeit und Ihren Aufwand optimieren können. Sie sollten einen Zustand anstreben, in dem die wesentliche Sicherheit kontinuierlich überprüft wird.
Klare Kommunikation der gefundenen	Tragen Sie mit weniger oder gar keinen Reibungsverlusten zur Wertschöpfung bei. Stellen Sie die Ergebnisse zeitnah mit den von den Entwicklungsteams verwendeten Tools bereit (keine PDF-Dateien). Arbeiten Sie mit dem Entwicklungsteam zusammen, um die Schwachstellen zu

Schwachstellen	besprechen. Nutzen Sie die Gelegenheit, sie aufzuklären, indem Sie die Schwachstelle klar beschreiben und erläutern, wie sie missbraucht werden kann, einschließlich eines Angriffsszenarios, um es realistisch zu machen.

Methodik und Daten

Überblick

Die AppSec-Branche hatte sich bisher nicht speziell auf die neueste Architektur von Anwendungen, in denen APIs eine wichtige Rolle spielen fokusiert. Deshalb wäre eine Liste der zehn kritischsten API-Sicherheitsrisiken auf der Grundlage eines öffentlichen Datenaufrufs zu erstellen, eine schwierige Aufgabe gewesen. Obwohl es keinen öffentlichen Datenaufruf gab, basiert die Top-10-Liste auf öffentlich verfügbaren Daten, Beiträgen von Sicherheitsexperten und offene Diskussionen mit der Community.

Methodik

In der ersten Phase wurden öffentlich verfügbare Daten über Sicherheitsvorfälle mit APIs von einer Gruppe von Sicherheitsexperten gesammelt, geprüft und kategorisiert. Solche Daten wurden innerhalb eines Zeitraums von einem Jahr von Bug Bounty-Plattformen und Schwachstellendatenbanken gesammelt. Sie wurden für statistische Zwecke verwendet.

In der nächsten Phase wurden Sicherheitsexperten mit Erfahrung in Penetrationstests gebeten, ihre eigene Top-10-Liste zusammenzustellen.

Zur Durchführung der Risikoanalyse wurde die OWASP Risk Rating Methodology verwendet. Die Bewertungen wurden von den Sicherheitsexperten diskutiert und überprüft. Für Überlegungen zu diesen Themen finden Sie im Abschnitt API-Sicherheitsrisiken.

Der erste Entwurf der OWASP API Security Top 10 2019 entstand aus einem Konsens zwischen den statistischen Ergebnissen aus Phase eins und den Listen der Sicherheitsexperten. Dieser Entwurf wurde dann einer weiteren Gruppe von Sicherheitsexperten mit einschlägiger Erfahrung im Bereich der API-Sicherheit weitergeleitet.

Die OWASP API Security Top 10 2019 wurde erstmals auf der OWASP Global AppSec Tel Aviv Veranstaltung (Mai 2019) vorgestellt. Seitdem steht sie auf GitHub für öffentliche Diskussionen und Beiträge zur Verfügung.

Die Liste der Mitwirkenden ist im Abschnitt Danksagungen verfügbar.

Danksagungen

Danksagungen an einzelne Mitwirkende

Wir möchten uns bei den folgenden Personen bedanken, die öffentlich auf GitHub oder über andere Wege zu diesem Projekt beigetragen haben:

- 007divyachawla
- Abid Khan
- Adam Fisher
- anotherik
- bkimminich
- caseysoftware
- Chris Westphal
- dsopas
- DSotnikov
- emilva
- ErezYalon
- flascelles
- Guillaume Benats
- IgorSasovets
- Inonshk
- JonnySchnittger
- jmanico
- jmdx
- Keith Casey
- kozmic
- LauraRosePorter
- Matthieu Estrade
- nathanawmk
- PauloASilva
- pentagramz
- philippederyck
- pleothaud
- r00ter

- Raj kumar
- Sagar Popat
- Stephen Gates
- thomaskonrad
- xycloops123
- Nick Lorenz
- Moritz Gruber
- Tim Barsch (domai-tb)
- Steffen Thamm