iX DEVELOPER
Winter 2021/22
14,90 €
Österreich 16,40 €
Schweiz 27.90 CHF
Luxemburg 17,10 €
www.ix.de

Sichere Software entwickeln

Schwachstellen finden und vermeiden
OWASP Top Ten 2021
Statische Codeanalyse und Fuzzing
Einblick ins Pentesting

Programmiersprachen
Speicherfehler in C++ aufspüren
Die Security-Konzepte von Rust

Schwerpunkt DevSecOps
Methoden, Werkzeuge und Reifegradmodelle
Softwarelieferketten für das IoT absichern
Privacy by Design

Kryptografie
Grundlagen und Fallstricke
Post-Quanten-Kryptografie

Cloud-Security
Confidential Computing für sensible Daten
Identitätsmanagement mit Keycloak
Richtlinien definieren mit dem Open Policy Agent

Gekommen, um zu bleiben

Wer Software entwickelt, braucht nicht lange auf Hackerangriffe zu warten. Sicher ist nur eins: Jemand wird nach Schwachstellen suchen und sie – sofern vorhanden – ausnutzen. Und zwar nicht nur in öffentlich zugängliche Webapplikationen, sondern in nahezu jeder Anwendung, auch in vermeintlich sicheren Umgebungen. Selbst private Hobbyprojekte stehen unter Beschuss: Jüngere Typo-Squatting-Attacken haben versucht, Ransomware und Crypto-Miner über Paketmanager wie npm und PyPI in Projekte einzuschleusen.

Der Kampf ist unfair: Während eine einzige Schwachstelle für einen erfolgreichen Hack genügt, müssen Teams sowohl ihre komplette Anwendung als auch die Infrastruktur zum Entwickeln, Verteilen und dem Betrieb absichern. Und sie müssen mit dem Unerwarteten rechnen – von Spectre und Meltdown war vor vier Jahren auch sorgfältig geschriebener und getesteter Code betroffen.

Unsere Konferenz zu sicherer Softwareentwicklung heise devSec läuft seit 2017 unter dem Motto „Sichere Software beginnt vor der ersten Zeile Code“, denn die Gefahren lauern im gesamten Software-Lifecycle. Agile Prozesse und Continuous Delivery mit Releases, die im Halbtages- statt Halbjahresrhythmus erfolgen, erfordern eine sorgfältige Integration von Security in die Pipeline. Die Zeiten, in denen das Dev-Team eine Anwendung nach monatelanger Entwicklung dem QA-Team für die Kontrolle übergibt, sind lange vorbei.

Die Aufteilung von Monolithen in Microservices-Architekturen und der Einsatz von Serverless Computing bringen zahlreiche neue Schnittstellen und damit weitere Angriffsflächen mit sich. Das Absichern und Verwalten von Anwendungen für das Internet der Dinge birgt zusätzliche Tücken vom Warten der Firmware über Over-the-Air-Updates bis zum Schutz vor Attacken auf die IoT-Infrastruktur.

Ein zentrales und unverzichtbares Security-Thema ist die Kryptografie, aber die Auswahl der passenden Algorithmen und Tools überfordert viele Entwicklerinnen und Entwickler. Ein prominentes Beispiel sind die wenig durchdachten Konzepte der Luca-App. Dass manche Open-Source-Tools zwar gute Funktionen, aber eine schlechte Dokumentation mitbringen, erschwert den Einsatz. Vermutlich werden zudem Quantencomputer in Kürze die Karten neu mischen und derzeit noch als sicher geltende Algorithmen aufs Abstellgleis schicken.

Das Thema Sicherheit ist, um es mit der Band „Wir sind Helden“ auszudrücken, gekommen, um zu bleiben. Auf Superman und Wonder Woman, die Software ebenso schützen wie Metropolis oder die Paradiesinsel, warten wir vergeblich. Batman wäre vermutlich bereits mit dem Absichern seines Batmobils gegen DoS-Attacken überfordert.

Daher richtet sich dieses Heft an die Alltagshelden, die Software von Schwachstellen befreien: die Softwareentwickler, die Softwarearchitektinnen, die Product Owner und die Security Champions. *iX* und *heise Developer* möchte euch helfen, auch ohne Superkräfte sichere Software zu entwickeln.

RAINALD MENGE-SONNENTAG

Web-Application-Security

Das Open Web Application Security Project hat dieses Jahr die OWASP Top Ten 2021 veröffentlicht, die im Vergleich zu der Liste der Sicherheitsbedrohungen im Web von 2017 einige Änderungen mitbringen. Pentests zum Aufspüren von Schwachstellen in Webanwendungen sollten gut geplant und vorbereitet werden. OAuth ersetzt die Autorisierung mit Name und Passwort durch einen tokenbasiert Ansatz und soll in Kürze in Version 2.1 erscheinen – neun Jahre nach OAuth 2.0.

ab Seite 7

Programmiersprachen

Java-Versionen erscheinen inzwischen halbjährlich, und von Java 11 bis Java 17 hat es zahlreiche Veränderungen in der Sicherheitsarchitektur gegeben. Das performante C++ ist dank freiem Speicherzugriff anfällig für Schwachstellen durch Speicherfehler, die sich mit den richtigen Maßnahmen minimieren lassen. Eine automatisierte Codeanalyse hilft beim Absichern, aber der Einsatz erfordert grundlegendes Know-how. Rust bietet interessante Konzepte, mit denen die Programmiersprache ohne Performanceeinbußen sicherer als C oder C++ ist.

ab Seite 21

Cloud-Security

Der Balanceakt zwischen Security und schnellen Entwicklungszyklen ist in der Cloud besonders schwierig. Serverless-Computing ermöglicht eine flexible Skalierung, aber Hunderte verteilte Funktionen bieten eine große Angriffsfläche, die es abzusichern gilt. Keycloak ist eine Open-Source-Anwendung für Identitäts- und Zugriffsmanagement. Confidential Computing soll mit hardwarebasierter Verschlüsselung sensibler Daten das Vertrauen in die Public Cloud stärken. Der Einsatz von Containern erfordert dedizierte Schutzmaßnahmen. Zum Durchsetzen von Compliance in der Cloud hat sich der Open Policy Agent als Allzwecktool etabliert.

ab Seite 43

Kryptografie

Zum Absichern von Software und Systemen ist Kryptografie unverzichtbar, aber das Thema ist äußerst komplex. Vor allem lauern beim praktischen Einsatz zahlreiche Stolperfallen, die sich durch gute Planung umgehen lassen. In nicht allzu ferner Zukunft werden Quantencomputer vermutlich die Karten neu mischen, da bisher als sicher geltende Algorithmen angreifbar werden.

ab Seite 73

Qualitätssicherung

Neben den OWASP Top Ten hat die Organisation eine Liste der größten Gefahren für APIs veröffentlicht. Statische Codeanalyse hilft beim Aufspüren von Fehlern und Schwachstellen, kann aber Code-Reviews nicht ersetzen. Fuzzing füttert Programme mit zufälligen oder bewusst fehlerhaften Inhalten und hat sich als Testverfahren etabliert. Neben technischen Anforderungen gilt es, beim Entwickeln die juristischen Vorgaben zum Datenschutz im Blick zu halten. Security by Design und der Security Development Lifecycle lieferten bereits vor zwölf Jahren Richtlinien für sichere Softwareentwicklung. In der Embedded-Entwicklung sind Tests auf der Zielplattform unerlässlich.

ab Seite 91

DevSecOps

Damit Security bei immer kürzeren Entwicklungszyklen nicht zu kurz kommt, bietet DevSecOps Maßnahmen, jeden Schritt von der Entwicklung bis zum Betrieb abzusichern. Inzwischen existiert eine Fülle an Tools, die bei der Umsetzung helfen. OWASP SAMM bietet ein Framework für den Secure Development Lifecycle, und die Security Belts sollen helfen, die Kompetenzen in Teams bezüglich der Sicherheit zu stärken. Shift Left steht für den Ansatz, Sciherheitsaspekte möglichst früh in den Software-lebenszyklus einzubeziehen Eine sichere Software Supply Chain ist im IoT- Umfeld mit Over-the-Air-Updates eine besondere Herausforderung.

ab Seite 121

Goodbye Routinejobs

Hello Automatisierung!

Heft + PDF mit 29 % Rabatt

In diesem Sonderheft erfahren Sie u.a. mit welchen Tools und Arbeitsabläufen Sie **Routinejobs im Rechenzentrum automatisieren** können und noch mehr:

- GitOps: Modernes Infrastructure as Code
- Multi-Cloud-Betrieb effizient gestalten
- Das Netz mit Python steuern
- **Inkl. Tutorial:** Ansible an die eigenen Bedürfnisse anpassen
- Für Abonnenten portofrei

Heft für 14,90 € • PDF für 12,99 € • Bundle Heft + PDF 19,90 €

shop.heise.de/ix-komp-rechenzentrum21

Generell portofreie Lieferung für Heise Medien- oder Maker Media Zeitschriften-Abonnenten oder ab einem Einkaufswert von 20 €.
Nur solange der Vorrat reicht. Preisänderungen vorbehalten.

heise Shop

Web-Application-Security

Das Open Web Application Security Project hat dieses Jahr die OWASP Top Ten 2021 veröffentlicht, die im Vergleich zu der Liste der Sicherheitsbedrohungen im Web von 2017 einige Änderungen mitbringen. Pentests zum Aufspüren von Schwachstellen in Webanwendungen sollten gut geplant und vorbereitet werden. OAuth ersetzt die Autorisierung mit Name und Passwort durch einen Token-basierten Ansatz und soll in Kürze in Version 2.1 erscheinen – neun Jahre nach OAuth 2.0.

Die OWASP Top Ten 2021

Was lange währt

Christian Wenz

Die OWASP Top Ten 2021 aktualisiert die Liste der Sicherheitsbedrohungen im Web. Im Vergleich zu der Ausgabe von 2017 gibt es zahlreiche Verschiebungen. Wie üblich regt die Liste zu Diskussionen an.

Das Open Web Application Security Project (OWASP) verfolgt die Mission, die Sicherheit (webbasierter) Software zu verbessern. Es richtet Treffen und Veranstaltungen aus, veröffentlicht zahlreiche Dokumentationen und Checklisten zu diversen Aspekten der Websicherheit und hat auch Softwareprodukte unter seiner Obhut unter anderem den bekannten Sicherheitsscanner OWASP Zed Attack Proxy (ZED) (alle Links finden sich unter ix.de/zzq5).

Das bekannteste Teilprojekt des OWASP ist eine seit 2003 mehr oder weniger regelmäßig veröffentlichte Liste der größten Sicherheitsrisiken für Webanwendungen: die OWASP Top Ten, die erstmals 2003 und in zweiter Auflage 2004 erschien. Danach schaltete OWASP auf einen dreijährigen Rhythmus um: 2007, 2010 und 2013. Danach wurde es schwierig: Die für 2016 angedachte Liste stand unter einem schlechten Stern, verspätete sich, hatte großen internen Diskussionsbedarf und kumulierte schließlich in einem halb garen Release Candidate im Jahre 2017. Die finale Liste brachte im Vergleich dazu an einigen Stellen signifikante Änderungen und erschien einige Monate später noch im selben Jahr. heise Developer hat die Hintergründe seinerzeit ausführlicher zusammengefasst.

Blick auf die aktuelle Top Ten

2020 sollte alles anders werden, aber nicht nur die Pandemie machte einen Strich durch die Rechnung. Wer weiß, wie lange es bis zur finalen Liste gedauert hätte, wenn nicht der zwanzigste Jahrestag von OWASP am 9.9.2021 angestanden hätte. Das Jubiläum erzeugte offenbar genug Leidensdruck, um am 8.9.2021 eine Vorabversion der neuen Top Ten zu veröffentlichen. Trotz einiger Diskussionen, auch im offiziellen GitHub-Repository des Projekts, stellte Projektleiter Andrew van der Stock pünktlich in der Jubiläumskonferenz des OWASP am 24.9.2021 die neue Liste vor (siehe Tabelle).

Im Vergleich zu früheren Ausgaben der Top-Ten-Liste hat das Team – begleitet durch eine nur unauffällig auf der OWASP-Site verlinkte, projektspezifische Homepage – etwas mehr Einblicke in das Erstellen der Listenelemente und ihrer Reihenfolge gegeben, wenngleich die prinzipielle Herangehensweise ähnlich war wie bei den vorherigen Ausgaben. Zunächst hat die Organisation anonymisierte Daten aus Sicherheitsaudits von Webapplikationen eingesammelt. Basis war das CWE-Projekt (Common Weakness Enumeration), das Kategorien für Sicherheitslücken in Applikationen aufzählt (enumeriert). Die freiwilligen Datenspenden für die OWASP Top Ten bestanden im Wesentlichen aus CWEs und der Information darüber, wie häufig sie gefunden wurden. Abbildung 1 zeigt einen Ausschnitt aus den zusammengeführten Daten der Liste von 2017.

Im nächsten Schritt galt es, CWEs sinnvoll in Kategorien zu gruppieren und anschließend in eine Reihenfolge zu bringen. Damit ist die Top Ten zu 80 Prozent fertiggestellt, mit acht der zehn verfügbaren Positionen. Um frühzeitig Trends und Beobachtungen aus der Praxis aufgreifen zu können, die sich eventuell (noch) nicht in den nackten Zahlen widerspiegeln, fand gleichzeitig eine Community-Umfrage statt, welche CWEs zusätzlich einen Platz verdient hätten. Daraus ergeben sich die zwei fehlenden Listenelemente.

Obwohl dieser Ansatz gut dokumentiert ist, bietet er aufgrund der potenziell willkürlichen Gruppierung von CWEs

iX-TRACT

- Seit 2003 veröffentlicht das Open Web Application Security Project (OWASP) eine Liste der größten Sicherheitsrisiken für Webanwendungen: die OWASP Top Ten.
- Nachdem der Vorgänger 2017 erschienen war, sollte die aktuelle Ausgabe ursprünglich 2020 herauskommen, verzögerte sich aber bis September 2021.
- Acht Plätze der aktuellen Top Ten beruhen auf Auswertungen von Sicherheitsaudits und zwei auf den Ergebnissen einer Community-Umfrage.

sowie der Integration der Umfrageergebnisse Diskussionsbedarf allen voran: Gibt es Punkte, die auf der Liste eine völlig falsche Position haben oder gar komplett fehlen? In den folgenden Ausführungen zu den zehn Punkten der aktuellen Liste wagt der Autor dieses Artikels an der einen oder anderen Stelle ein subjektives Urteil.

OWASP Top Ten 2021	
Nr.	**Listeneintrag**
1	Broken Access Control
2	Cryptographic Failures
3	Injection
4	Insecure Design
5	Security Misconfiguration
6	Vulnerable and Outdated Components
7	Identification and Authentication Failures
8	Software and Data Integrity Failures
9	Security Logging and Monitoring Failures
10	Server-Side Request Forgery

1. Broken Access Control

Das größte Risiko ist laut der OWASP Top Ten 2021 eine defekte Zugriffskontrolle. Der Oberpunkt war in der Vorgängerliste auf Platz 5 vertreten und hat diesmal den Sprung nach oben geschafft. Beigetragen haben insgesamt 34 zu dem Listeneintrag passende CWEs.

Grundsätzlich geht es bei dem Punkt um fehlenden Zugriffsschutz, der aber in unterschiedlichen Ausprägungen vorkommen kann: Umgehen eines bestehenden Zugriffsschutzes durch Veränderung von URL-Parametern, fehlende Autorisierung bei anderen HTTP-Verben als den vorgesehenen ungesicherten Endpunkten oder Dateien sowie Replay-Angriffe mit JSON Web Tokens (JWT) – die Liste ist lang. Wie immer gilt es, Eingaben immer zu prüfen und zu jedem Zeitpunkt die entsprechende Autorisierung des Clients sicherzustellen.

Etwas überraschen mag, dass neuerdings Cross-Site Request Forgery (CSRF) unter diesem Dach ein neues Zuhause gefunden hat. Vor 2017 hatte der Angriff einen eigenen Platz in der Top Ten, und die 2017er-Ausgabe hat er knapp verpasst. Damals war der Autor auf der Basis seiner eigenen Funde im Rahmen von Audits und Codeanalysen der Meinung, er hätte in die Top Ten gehört. Inzwischen ist der Angriff voraussichtlich keine eigene Betrachtung mehr wert. Neben gut funktionierenden, teilweise auch standardmäßig aktiven Anti-CSRF-Maßnahmen in diversen Webframeworks schränken SameSite-Cookies die Angriffsoberfläche deutlich ein. Von einer großen Renaissance des Angriffs ist aktuell nicht auszugehen.

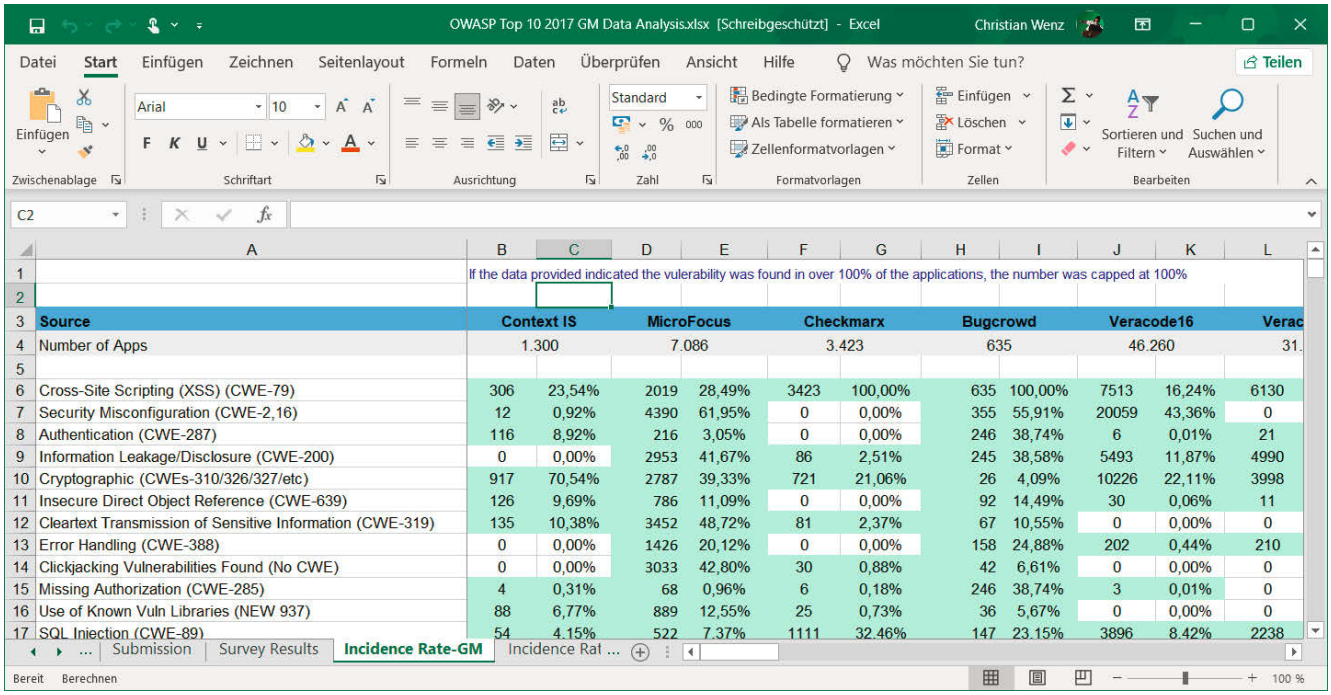
2. Cryptographic Failures

Der etwas furchterregende Begriff des kryptografischen Versagens sammelt respektable 29 CWEs unter seiner Regie. Viele der aufgeführten Punkte wie das Verwenden unsicherer Algorithmen, sind beim Einsatz eines halbwegs modernen Stacks kaum relevant. Kein zeitgemäßes Framework verwendet noch MD5 oder SHA1 zum Hashen eines Passworts – hoffentlich zumindest. Gerade das Passwort-Hashing können Entwicklerinnen und Entwickler frameworkseitig entweder automatisch nach Best Practices erledigen lassen (beispielsweise bei ASP.NET Core) oder auf eine einfach zu benutzende, aber sichere API zurückgreifen (z. B. bei PHP).

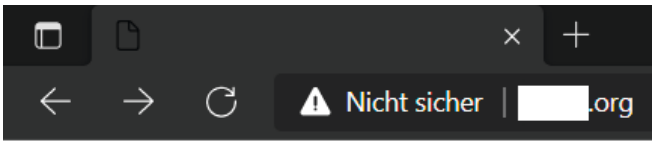
In der OWASP Top Ten von 2017 hieß der Punkt „Sensitive Data Exposure“. Damals war vermutlich das Hauptproblem der Verzicht auf Verschlüsselung, insbesondere auf dem Transportweg. Da die meisten Webbrowser inzwischen Sites ohne HTTPS geradezu drangsalieren (s. Abb. 2) ist von einem weiteren Rückgang auszugehen. Der Einsatz von HTTPS schließt zudem einige, wenn auch nicht alle Flanken von Session Hijacking. Das Erzwingen von HTTPS lässt sich durch HTTP Strict Transport Security (HSTS) zusätzlich verschärfen; das SecureFlag für Cookies verhindert, dass sie über ungesicherte Kommunikationskanäle transportiert werden.

3. Injection

Ein häufig vorgebrachter Kritikpunkt an der 2017er-Ausgabe der OWASP Top Ten – auch durch den Autor dieses Artikels –



Teile der für die OWASP Top Ten 2017 erhobenen Daten (Abb. 1).



Kein HTTPS, keine Liebe vom Browser (Abb. 2).

ist der Spitzenplatz von Injection. Gemeint ist damit primär SQL Injection, ein Angriff, gegen den die Verteidigung einfach ist: Parametrisierte Abfragen beziehungsweise Prepared Statements arbeiten mit Platzhaltern in SQL-Queries, die deutlich festlegen, was ein Datum ist und was ein Befehl. Es gibt keinen Grund (und, keine Ausrede) für das lockere Verketten von Strings. Beim Einsatz eines objektrelationalen Mappers wie Hibernate, Entity Framework oder Doctrine, ist der Code noch einen Schritt weiter von potenziell anfälligem SQL entfernt, da die Arbeit fast ausschließlich über eine API erfolgt, die SQL Injection nahezu unmöglich macht.

Die OWASP hat die Spitzenplatzierung von (SQL) Injection immer verteidigt, gerade im Vergleich mit Cross-Site Scripting (XSS). Dieser Angriff, bei dem es um das Einschleusen von schadhaftem JavaScript-Code oder HTML-Markup geht, hat von den absoluten Zahlen her deutlich höhere Werte als der Datenbankangriff, befand sich aber in der OWASP Top Ten 2017 nur auf Platz 7. Begründung: Der Auftritt eines XSS führt in Folge zu zig weiteren XSS-Funden und verfälscht dadurch die Statistik.

Vier Jahre später sieht es etwas anders aus. Injection ist auf Platz 3 abgefallen. Das offenbar weniger relevante XSS ist inzwischen allerdings Teil der Kategorie – das Einschleusen von JavaScript ist technisch ebenfalls eine Injektion. Das klingt subjektiv nach einem guten Kompromiss. OWASP wahrt das Gesicht (Injection immer noch in den Top 3), aber der Einfluss von XSS wird deutlich. Ohne die Hinzunahme von XSS wäre Injection laut der Live-Präsentation der 2021er-Liste an das Ende der Top Ten abgerutscht.

Dennoch ist dieser Zusammenschluss nicht ohne Kritik. Die Verteidigung gegen XSS hat deutlich mehr Facetten als der Schutz gegen SQL Injection. Neben einem ordentlichen Output-Escaping – und zwar kontextabhängig, da Sonderzeichenentwertung innerhalb von HTML anders funktioniert als innerhalb von JavaScript – helfen Defense-in-Depth-Mechanismen wie Content Security Policy (CSP), um bösartigem JavaScript und HTML-Markup den Garaus zu machen. Es bleibt zu hoffen, dass das Risiko „Injection" künftig noch weiter an Boden verliert.

4. Insecure Design

Die Kategorie „unsicheres Design" ist ein Neuzugang. Die Intention ist klar: Wichtige Sicherheitskonzepte wie Threat Modeling und der Einsatz von Referenzarchitekturen sind in vielen Webapplikationen unterrepräsentiert. Sicherheit fängt nicht erst beim Schreiben von Code an. Die Entscheidung sorgt für Diskussionen – so wie viele Allgemeinplätze in der OWASP Top Ten, gerade wenn sie neu sind. Im Vergleich zu anderen Punkten auf der Liste lassen sich nur beschränkt nachvollziehbare Aktionen ableiten. Die OWASP Top Ten ist eine Liste von Risiken, nicht von Angriffen, aber den meisten anderen Punkten lassen sich Angriffe besser zuordnen. Da ein sicheres Design wichtig ist, ist die Platzierung in der Liste durchaus berechtigt.

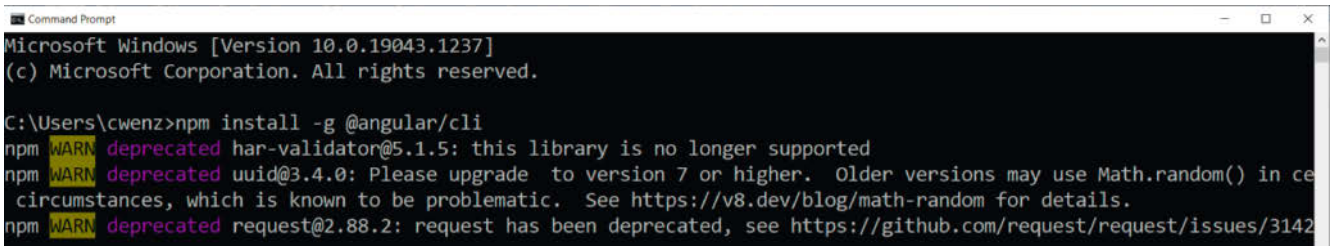
5. Security Misconfiguration

Ein verbreitetes Missverständnis lautet: Für die Konfiguration ist das Administrationsteam zuständig und die Programmierung hat damit nichts zu tun. Das mag in der Vergangenheit zeitweise gegolten haben, aber heutzutage sind viele Aspekte sicherer Einstellungen in den Händen derjenigen, die Codes schreiben. Beispiele, die in diese Kategorie fallen, gibt es reichlich: fehlende Sicherheits-HTTP-Header oder Cookie-Flags, nicht abgeschaltete detaillierte Fehlerseiten mit komplettem Stacktrace – oder Speichern von Passwörtern in einer Konfigurationsdatei. Etwas überraschend ist, dass XML External Entities (XXE) in der Liste von 2021 ein Zuhause unter der Oberkategorie der Fehlkonfigurationen gefunden haben. XXE waren ein Last-Minute-Neuzugang in der OWASP Top Ten 2017, der gleich auf Platz 4 eingestiegen ist. Begründet wurde das damit, dass XXE bei der Umfrage zu fehlenden Punkten äußerst häufig genannt wurden.

Nach Meinung des Autors ist die Gefahr von XXE allerdings stark abhängig von Stack und Technologie. Beispielsweise ist ASP.NET seit der 2014 veröffentlichten Version 4.5.2 standardmäßig vor dem Angriff abgesichert. In der Bibliothek `libxml`, die PHP nutzt, ist seit der 2012 erschienenen Version 2.9.0 standardmäßig das Feature deaktiviert, das den Angriff ermöglichte. Durch die Unterordnung von XXE in „Security Misconfiguration" hat OWASP einen der größten Kritikpunkte an der Vorgängerausgabe der Top Ten elegant umgangen.

6. Vulnerable and Outdated Components

Punkt 6 auf der OWASP Top Ten besteht eigentlich nur aus einem CWE-Eintrag: „CWE-1104 Use of Unmaintained Third Party Components". Zwar sind noch zwei weitere CWEs aufgeführt, die sich aber auf die OWASP Top Ten von 2013 und 2017 beziehen und somit als selbsterfüllende Prophezeiung zu sehen sind. Zeitgemäße Webanwendungen haben häufig viele externe Abhängigkeiten – insbesondere einige SPA-Anwendungen. Das soll nicht abwertend klingen: Bei einem großen Funktionsumfang ist das Setzen auf bestehende Methoden in Open-Source-Bibliotheken ein gangbarer und ökonomi-



Nicht alle verwendeten Komponenten sind taufrisch (Abb. 3).

scher Weg. Die Folge ist jedoch ein erhöhter Aufwand beim Nachverfolgen aller Dependencies.

Typisches (und wertfreies) Beispiel: Die Installation des Angular CLI richtet mehrere als deprecated (veraltet) markierte Bibliotheken ein (s. Abb. 3). Ein darauffolgender Aufruf von `ng new` erzeugt eine „leere" Angular-Anwendung. Das Zielverzeichnis ist anschließend über 300 MByte groß. Es aktuell zu halten ist trotz zur Verfügung stehender Unterstützungsmechanismen wie `npm audit` aufwendig. Daher ist der Aufstieg von Platz 9 auf der Vorgängerliste verständlich.

7. Identification and Authentication Failures

In der OWASP Top Ten von 2017 war ein Punkt namens „Broken Authentication" noch an zweiter Stelle aufgeführt. Unter neuem Namen findet er sich 2021 fünf Plätze weiter unten wieder. Enthalten sind CWEs zu diversen Session-Angriffen wie Session-Fixation oder zu langen Session-Timeouts. Darüber hinaus sind weitere Probleme im Bereich Authentifizierung enthalten. Dass dieses Risiko insgesamt zurückgegangen ist, mag darin begründet liegen, dass Frameworks beziehungsweise Systeme inzwischen Basisfunktionen rund um Authentifizierung in sicherheitstechnisch guter Qualität mitliefern. Vulgo: Es gibt bei einem Audit in dieser Hinsicht weniger zu finden als zu Zeiten, in denen ein größerer Anteil der Log-in-Masken selbst gestrickt waren. In dem Fall sorgt die Vereinheitlichung dafür, dass wichtige Security-Best-Practices auf Knopfdruck verfügbar sind.

8. Software and Data Integrity Failures

Der Eintrag zum Thema Software- und Datenintegrität ist neu in der OWASP Top Ten 2021. Pate stand womöglich der Angriff auf SolarWinds, bei dem ein schadhaftes Update eingespielt worden ist. Ganz allgemein geht es bei diesem Listeneintrag um eine fehlende Integritätsprüfung. In der Liste von 2017 war bereits der Sonderfall unsichere Deserialisierung als eigenständiger Punkt aufgeführt. Beim Entgegennehmen von Daten in serialisierter Form wie im JSON- oder XML-Format ist es wichtig, auf dem erwarteten Datentyp zu bestehen, statt blind die Daten zu deserialisieren. Beim Einsatz von Continuous Integration und Continuous Delivery (CI/CD) ist ebenfalls jeder Schritt des Prozesses vor fremdem Zugriff oder Manipulation abzusichern. Bei JavaScript-Code kann der Webbrowser helfen: Subresource Integrity gibt per Attribut des `<script>`-Tags einen Hash-Code vor, der zu der zu ladenden Skriptdatei passen muss. Das verhindert die Ausführung des schadhaften Codes etwa bei der Übernahme aus einem Content Delivery Network (CDN), weil der Inhalt nicht zum Prüfwert passt.

9. Security Logging and Monitoring Failures

Loggen und Monitoring aus Sicherheitssicht ist seit 2017 Teil der OWASP Top Ten; im Jahr 2021 stieg das Risiko um einen Platz nach oben. Das konkrete Herausarbeiten von Handlungsanweisungen fällt etwas schwer: Teams müssen loggen und sowohl die Applikation als auch die Logs überprüfen. Dennoch ist das etwas schwammiger als beispielsweise die konkreten Maßnahmen, die vor spezifischen Angriffen warnen, wie bei SQL Injection. Unter dem Oberpunkt versammeln sich dementsprechend nur vier CWEs, die sich alle auf Log-Dateien beziehen. Das Monitoring ist damit eine Art Bonusfunktion in Punkt 9 der OWASP Top Ten, aber nichtsdestotrotz eine wichtige ergänzende Maßnahme. Ein Log, das sich niemand anschaut, ist wie ein Backup, das nie für ein Restore dient.

10. Server-Side Request Forgery

Am Ende der OWASP Top Ten 2021 findet sich noch ein Neuzugang, der in der Community-Umfrage Platz 1 belegt hat. Die Rede ist von Server-Side Request Forgery, kurz SSRF. Die Namensähnlichkeit zu CSRF ist durchaus beabsichtigt. Während es bei Cross-Site Request Forgery meist darum geht, dass der Clientbrowser eine ungewollte HTTP-Anfrage erzeugt, ist es bei SSRF der Server. Der Angriff bringt somit den Webserver dazu, einen HTTP-Request zu erzeugen, der an ein System gerichtet sein kann, das von außen nicht zugänglich wäre. Auf diese Weise lassen sich Firewalls und andere Schutzmaßnahmen umgehen.

Bei den bereits in Punkt 5 thematisierten XML External Entities (XXE) könnte ein erfolgreicher Angriff eine solche Anfrage auslösen. Das OWASP weist explizit darauf hin, dass die erhobenen Daten keinen Platz in der Liste rechtfertigen würden, aber die Community-Umfrage eine Meinungslage für die Aufnahme von SSRF gezeigt habe.

Die subjektive Meinung des Artikelautors ist, dass für einen erfolgreichen SSRF-Angriff die Sterne extrem günstig stehen müssen: Nicht nur bedarf es eines Angriffsvektors wie XXE, der in den meisten modernen Stacks stark erschwert ist, sondern darüber hinaus eines optimalerweise über HTTP GET zugänglichen Endpunktes, der zudem entweder wertvolle Informationen liefert oder eine gefährliche Aktion auslöst, was bei GET unwahrscheinlich ist. 2017 gab es bei der kurzfristigen Aufnahme von XXE ähnliche Diskussionen. Möglicherweise verschwindet der Punkt in der nächsten Liste, die turnusmäßig 2024 (oder wieder verzögert 2025) ansteht. Alternativ könnte er in einem anderen wie „Broken Access Control" aufgehen.

Streitbare Grundlage

Über die OWASP Top Ten ließ sich immer trefflich streiten. Bei der aktuellen Liste ist das nicht anders. Dabei darf eins nicht vergessen werden: Es handelt sich um ein Awareness-Dokument. Ziel ist es nicht, eine Prioritätenreihenfolge auf wissenschaftlichem Niveau anzubieten, sondern relevante Risiken für Webapplikationen auf nachvollziehbare Art und Weise zu präsentieren, um kompakt und fokussiert in Form einer Referenz auf alles Wichtige und Wissenswerte hinweisen zu können. Das leistet die OWASP Top Ten seit mittlerweile 18 Jahren und OWASP selbst seit 20 Jahren. (rme@ix.de)

Quellen

Die Links zum OWASP und der Top Ten sowie zu den im Artikel erwähnten Referenzen finden sich unter ix.de/zzq5

Christian Wenz

Christian Wenz ist Autor, Trainer und Berater für Webentwicklung und -sicherheit. Er ist Gründer und Teilhaber von Arrabiata Solutions, MVP und Insider für ASP.NET.

Penetrationstests von Webanwendungen – Planung und Organisation

Simulierte Angriffe

Tobias Glemser

Penetrationtests tragen dazu bei Webanwendungen sicherer zu machen. Damit sinnvolle Prüfungen erfolgen und keine unvorhergesehenen Risiken entstehen, lohnt es sich Zeit in die Planung zu stecken.

Zu Beginn sei klargestellt: „Der" Penetrationstest existiert nicht. Seit Langem gibt es Versuche, Prüfungen zu standardisieren und zu beschreiben, was jedoch nur für einfache Prüfformen und zudem eingeschränkt funktioniert. Eine Basisprüfung auf der Infrastruktur ist ein Beispiel. Bei einem einfachen Port- und Schwachstellenscan ist die Beschreibungstiefe gering. Die Anforderungen lassen sich in einem Satz zusammenfassen: „Bitte alle von extern erreichbaren IP-Adressen auf offene Ports und bekannte Schwachstellen in den bereitgestellten Diensten prüfen!" Obwohl auch dabei beim Planen und Ausführen in der Praxis Stolpersteine lauern, sind kaum Absprachen notwendig und das resultierende Risiko für die geprüfte Infrastruktur strebt gegen null.

Penetrationstests sind zudem keine bezahlten Hackerangriffe. Hacking ist ein Mindset, Penetrationstests sind hingegen eine professionelle Dienstleistung ausgebildeter Expertinnen und Experten mit den passenden persönlichen Eigenschaften. Ein Pentester sollte nicht versuchen, einen realistischen Angriff abzubilden. Welcher sollte das auch sein? Wer definiert einen „echten Hacker"? In der Praxis warten beliebig viele Angreifende mit unterschiedlichen Fähigkeiten.

Das National Institute of Standardisation and Technology (NIST) beschreibt einen Penetrationstest in der Spezifikation SP 800 53r4 (der Link findet sich unter ix.de/z29z) als Resilienz-Prüfung. Er ermittelt, wie widerstandsfähig eine Anwendung gegenüber einem vorab definierten Szenario ist. Die Details ergeben sich aus der zur Verfügung stehenden Zeit, der Expertise des Prüfers oder der Prüferin, den Kenntnissen des Prüflings sowie der Testmethode.

Ebenfalls notwendig ist das Verständnis für die Testmechanik. Für folgende Quizfrage sollten alle bitte vor dem Weiterlesen kurz in sich gehen: Ein Penetrationstest besteht – neben Standardaufgaben wie Projektmanagement und Qualitätssicherung – aus dem eigentlichen Testen und der Dokumentation der Ergebnisse. Nehmen wir an, dass wir in einem Projekt zehn Tage Testzeit haben. Wie viel Anteil haben Test und Dokumentation in diesem Projekt?

1. 30 Prozent Test und 70 Prozent Dokumentation,
2. Fifty-fifty oder
3. 70 Prozent Test und 30 Prozent Dokumentation?

Zugegebenermaßen ist das eine Fangfrage: Jede Antwort kann zutreffen. In einem Projekt ergibt sich die zur Verfügung stehende Testzeit aus der Anzahl der gefundenen Schwachstellen. Testende müssen umso mehr dokumentieren, je mehr Schwachstellen sie finden. Da die Zahl der Funde im Voraus nicht feststeht, ist die Testzeit variabel. Das Ergebnis ist aber in jedem Fall valide: Finden Tester viele Schwachstellen in kurzer Zeit, müssen sie viel dokumentieren. Die geprüfte Anwendung ist offensichtlich nicht sonderlich widerstandsfähig. Stoßen die Prüferinnen und Prüfer dagegen auf wenige Probleme, können sie länger testen, da sie nur wenige Schwachstellen dokumentieren müssen. Die Anwendung zeigt sich resilient. Man spricht daher von Time-Boxing und bei äußerst pentestlastigen Zertifizierungsverfahren vom Fixed-Time-Ansatz. Die Methodik eines Penetrationstests ist daher nicht deterministisch.

No Risk, No Fun?

Es gilt, Voraussetzungen für Tests zu schaffen, die effizient und risikoarm sind. Hacker haben zwar keine Skrupel vor risikobehafteten Angriffen. Ein Vorteil von Pentests ist jedoch, dass man vorab Risiken identifizieren kann, sodass die Tests keinen Schaden anrichten können. Ein Denial-of-Service (DoS) einer Produktionsumgebung sollte bei einer Anwendungsprüfung im Regelfall kein Prüfziel darstellen.

Dennoch bleiben Prüfungen risikobehaftet. Viele Schwachstellen, darunter alle Formen sogenannter Injection-Angriffe wie SQL-Injection oder Cross-Site-Scripting (XSS), können Testende nur feststellen, indem sie diese ausnutzen. Jede Prü-

fung ist somit ein Exploit von Schwachstellen. Auch wenn Prüferinnen versuchen, mit handzahmen Angriffssignaturen (Payloads) vorzugehen, können sie eine Beeinträchtigung nie ausschließen.

Abschnallen bitte

Obwohl es auf den ersten Blick unsinnig erscheint, bei einem Autocrashtest aktive Sicherheitsmechanismen wie den Sicherheitsgurt und den Airbag zu deaktivieren, kann das Vorgehen durchaus notwendig sein, um ausschließlich die Schutzwirkung der passiven Sicherheitsmechanismen zu prüfen. Ähnlich verhält es sich beim Prüfen der Resilienz einer Anwendung. Dafür ist es empfehlenswert, Schutzmechanismen wie Web-Application-Firewalls (WAFs) oder Captchas als Schutz vor Anmelde-Bruteforcing für die Tests zunächst zu deaktivieren. Viele Schwachstellen lassen sich nur durch das Prüfen Tausender Payloads pro Variable aufspüren. Wenn die Mechanismen dafür sorgen, dass mit hoher Wahrscheinlichkeit nur wenige Payloads ihr Ziel erreichen, sinkt die Aussagequalität eines Tests massiv. Ein Angreifer hingegen hat dagegen theoretisch beliebig viel Zeit. Hinzu kommt, dass eine Schwachstelle beim Testen beispielsweise mit einer Payload an Position 1382 und bei einem anderen an erster Position gefunden werden könnte. Im ersten Fall würde die Payload aufgrund der Schutzmaßnahmen nicht zum Tragen kommen, im zweiten Fall dagegen funktionieren.

Man testet daher zunächst ohne zusätzliche Mechanismen und aktiviert sie nach dem Prüfen wieder. Anschließend prüfen die Tester die entdeckten Schwachstellen selektiv erneut. Damit stellen sie fest, ob die Sicherheitsmerkmale auch bei gezielten und wirksamen Angriffen funktionieren.

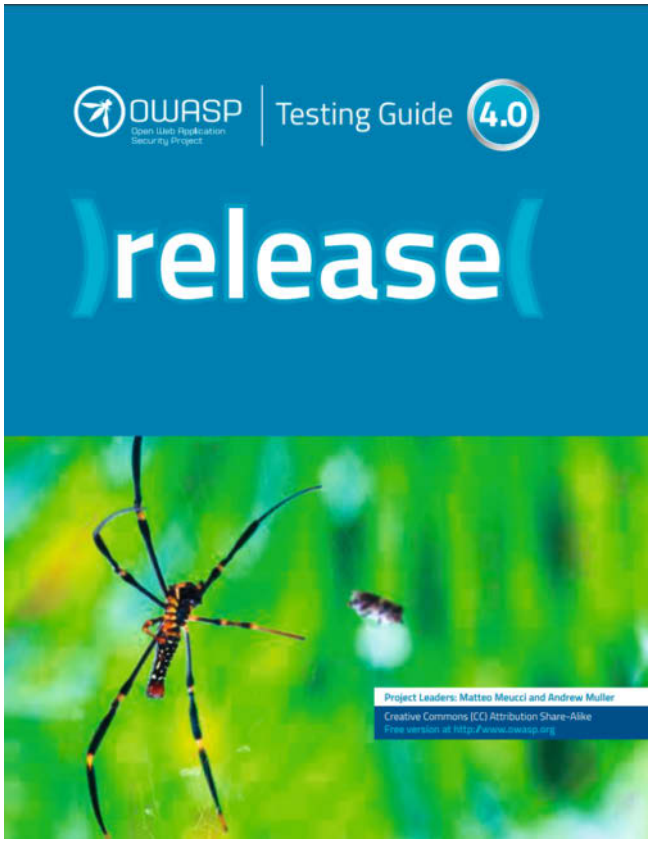
Der richtige Scope

Als Beispiel soll die Aufgabe dienen, einen B2B-Shop mit manuell verifizierten Usern zu prüfen. Der Shop kennt drei Nutzerrollen: Anonym, Kunde und Support. Tester könnten den Shop mit einem „explorativen Ansatz“ prüfen. Sie bekommen alle Zugangsdaten und legen einfach los. Alternativ können sie zunächst vorab den funktionalen Umfang der Anwendung anschauen und eine Einschätzung treffen, wie lange sie für eine Prüfung brauchen werden. Dabei unterscheidet man sinnvollerweise zwischen technischen Prüfungen und Tests auf das Rollen- und Rechtekonzept.

Der Umfang der technischen Prüfungen hängt von der Größe und Komplexität der Anwendung ab. Die Schwachstellen stecken meist in Funktionen der Applikation. Je funktional

iX-TRACT

- Penetrationstests prüfen Anwendungen auf Resilienz, indem sie das System von außen auf Schwachstellen untersuchen.
- Es gibt keinen standardmäßigen Pentest, da das Vorgehen von der zu testenden Anwendung abhängt.
- Gute Planung im Vorfeld und das Einbeziehen aller beteiligten Personen erhöhen die Effektivität und die Effizienz der Tests.



Quelle: Open Web Application Security Project

Der OWASP Testing Guide ist eine gute Grundlage für das Ausführen der geplanten Prüfungen.

umfangreicher diese ist, desto mehr Zeit ist zum Prüfen erforderlich. Deshalb lassen sich keine pauschalen Aussagen zum Prüfaufwand treffen, ohne den Funktionsumfang einer Anwendung zu kennen.

Beim Prüfen des Rollen- und Rechtekonzepts (Privilege Escalation) unterscheidet man zwischen horizontalen und vertikalen Ausbrüchen aus dem Konzept. Bei Ersteren greift jemand auf Daten anderer User mit derselben Berechtigung zu: Eine Kundin oder Kunde erlangt beispielsweise Zugriff auf die Daten eines anderen Kunden. Bei vertikalen Angriffen nutzt jemand Funktionen, die seiner Nutzerrolle gar nicht zur Verfügung stehen sollten.

Neben der explorativen und der Vollprüfung existiert eine weitere Variante, die sich in der Prüfpraxis seit vielen Jahren bewährt hat: Auch hier steht am Anfang die gedankliche Trennung der technischen Tests von den Prüfungen auf das Rollen- und Rechtekonzept. Für die technischen Tests fragt sich der Geprüfte vorab, aus welchen Rollen heraus technische Angriffe wahrscheinlich sind. In englischsprachigen Dokumenten ist vom Threat-Agent die Rede, auf Deutsch heißt es das Angreifermodell. Man stellt sich den Angreifer als reale Person vor und spricht von einem Pentest-Avatar. Die Analyse der Bedrohungslage ist Teil dieses risikoorientierten Vorgehens. Im Beispiel des B2B-Webshops könnte man in einem ersten Schritt nur aus Sicht des Pentest-Avatars ohne Anmeldedaten prüfen. Andere könnten bei der gleichen Risikobetrachtung desselben Shops aber einen Pentest-Avatar mit der Berechtigung „Kunde“ für sinnvoll erachten. Es gibt kein Richtig oder Falsch.

Das Prüfen des Rollen- und Rechtemodells kann – statt explorativ oder vollumfänglich – mit einer risikoorientierten Stichprobe erfolgen. Vorab werden zehn Testfälle wie das Risiko des Zugriffs eines Kunden A auf die Bestellhistorie von Kundin B definiert. Die Testfälle orientieren sich am Risiko des Geschäftszwecks der Anwendung. Eine vollumfängliche Prüfung ist häufig ineffizient, da derselbe Code, der für obigen Fall die Zugriffsrechte prüft, in vielen Szenarien mehrfach genutzt und geprüft würde.

Kultureller Wandel

Es hat mehrere Vorteile, wenn Prüfer sich vorab über die Bedrohungslage Gedanken machen und am Geschäftszweck ausgerichtete Prüfungen des Rollen- und Rechtemodells durchführen. Im Gegensatz zum explorativen Ansatz besteht dabei ein definierter Scope. Im Vergleich zur Vollprüfung ist die Planungsart deutlich effizienter. Gleichzeitig deckt die Prüfung alle aus Risikosicht relevanten Anwendungsanteile ab.

Wichtig ist, dass für die Risikoabschätzung stets die Stakeholder eingebunden sein müssen. Nicht Security-Managerinnen oder Entwickler können die Fragen der sinnvollen Pentest-Avatare und Testfälle beim Prüfen des Rollen- und Rechtemodells beantworten, sondern nur die fachliche Seite. Das Einbinden der verantwortlichen Kolleginnen und Kollegen in die Planung fördert den erforderlichen kulturellen Wandel in allen Organisationen, die Cybersecurity nachhaltig begreifen wollen: Es gilt, alle in die Prozesse zu holen, die für die Risiken verantwortlich sind. Im ersten Moment wird es erfahrungsgemäß Widerstände geben. Immer noch betrachten viele Cybersicherheit fälschlicherweise als Technikerthema. Analog dazu, dass keine Entwicklerin und kein Entwickler ohne Auftrag der Fachseite Anwendungen entwickelt, entsteht kein Risiko ohne Geschäftszweck. Wenn eine Organisationsleitung das begriffen hat, kann sie viele kommunikative Gräben überbrücken.

Umfassendes Konzept

Das Prüfen von Webanwendungen über Penetrationstest lässt sich nicht nach einem einheitlichen Schema erledigen. Teams sollten sich vorab darüber Gedanken machen, aus welcher Blickrichtung sie die Anwendungen prüfen lassen wollen und welche potenziellen Gefahren im Rollen- und Rechtesystem abzudecken sind. Auf die Weise können Testerinnen und Tester zielgerichtet und effizient Angriffsvektoren abklopfen.

Die technischen Prüfungen von denen auf das Rolle- und Rechtemodell zu separieren und risikoorientiert am Geschäftszweck entlang zu prüfen, erfordert vorab einen gewissen Planungsaufwand. Am Ende sind jedoch alle Stakeholder eingebunden und es gibt klar definierte Prüfziele sowie ein effizientes, risikoorientiertes Vorgehen. Der Ansatz schont im Vergleich zu anderen zusätzlich das Budget. (rme@ix.de)

Quellen

Die Links zu der NIST-Spezifikation und dem OWASP Testing Guide finden sich unter ix.de/z29z

Tobias Glemser

ist BSI-zertifizierter Penetrationstester und Geschäftsführer der secuvera GmbH.

OAuth 2.1 – das Sicherheitsupdate

Passwortfrei im Netz

Andreas Falk

Mit der anstehenden Version 2.1 wird OAuth ein Sicherheitsupdate erhalten, das es für heutige, ehemals nicht geplante Einsatzbereiche ausrüsten soll.

OAuth 2.0 ist in Internetdiensten und als Basis von OpenID Connect in Enterprise-Anwendungen weltweit vielfach im Einsatz. Das Standardprotokoll ist allerdings etwas in die Jahre gekommen und für viele heutige Einsatzgebiete ursprünglich nicht vorgesehen. OAuth 2.1 wird es daher auf den neuesten Stand bringen.

Klassische Anwendungsfälle von OAuth

Wer einen Webdienst oder Client eines Drittanbieters wie Stack Overflow einsetzt, kennt das Problem: Um den Dienst ernsthaft nutzen zu können, ist eine Anmeldung erforderlich. Nun möchte sich aber nicht jede Nutzerin oder jeder Nutzer für einen weiteren Service mit einem neuen Benutzernamen samt zugehörigem sicherem Passwort registrieren.

Das ist ein typischer Anwendungsfall für OAuth: Hier verwendet der Client nicht das Benutzerpasswort, sondern ein geheimes Token (Access-Token), das ein Autorisierungsserver etwa von Google, Facebook oder GitHub nur für ihn ausstellt. Nutzer kontrollieren, welche Rechte der Client erhält und welche nicht. Sie können sie ihm auch jederzeit wieder entziehen, ohne ihr Passwort zu ändern, indem sie das Token für ungültig erklären beziehungsweise die Autorisierung für diesen Client beim Autorisierungsserver, etwa GitHub, widerrufen.

OAuth ist dazu konzipiert, ohne ein Passwort einen kontrollierten Zugriff auf APIs mit fein abgestuften Berechtigungen zu ermöglichen.

Mit dem Autorisierungsserver führt OAuth 2.0 eine zentrale Instanz zur Anmeldung ein und separiert die Rolle des Clients von der des Ressourcenbesitzers (Benutzer). Der Client fordert über den Autorisierungsserver den Zugriff auf Ressourcen an, die der Ressourceneigentümer (Benutzer) kontrolliert und der Ressourcenserver (der API-Service) anbietet.

Dieses Modell ersetzt die zuvor gängige Praxis der Verteilung von Anmeldedaten an alle Beteiligten (Client, Server). Darüber hinaus bieten am Markt verfügbare Implementierungen von Autorisierungsservern standardmäßig die wesentlich stärkere Multi-Faktor-Authentifizierung an. Ohne OAuth 2.0 müsste jeder Client diese Funktion sonst selbst implementieren.

Für Anwendungsfälle, in denen im Zuge einer Authentifizierung die Identität des Benutzers zu überprüfen ist, existiert OpenID Connect (alle Links zum Artikel finden sich unter ix.de/z3dv), das seine Spezifikationen von der OpenID Foundation erhält und auf OAuth 2.0 aufbaut. OpenID Connect fügt hierfür zusätzlich zum Access-Token ein ID-Token im JSON-Web-Token-Format (JWT) zur Übertragung von Identitätsinformationen hinzu.

Wenn im weiteren Verlauf dieses Artikels die Rede von OAuth ist, dann treffen die beschriebenen Eigenschaften analog auch auf OpenID Connect zu. Gegebenenfalls davon abweichende Aspekte sind explizit genannt.

X-TRACT

- OAuth ist ein offenes Autorisierungsprotokoll, das in zahlreichen APIs und als Basis von OpenID Connect umfangreich zum Einsatz kommt.
- Version 2.1 soll das seit 2012 existierende OAuth 2.0 ablösen, das in vielen Fällen nicht mehr zeitgemäß ist.
- Die Kombination aus Name und Passwort weicht einem Token-basierten Ansatz, der eine höhere Sicherheit bieten soll.

Warum eine neue Version von OAuth?

Seit seiner Veröffentlichung in den Standards RFC 6749 und RFC 6750 im Jahr 2012 durch die Internet Engineering Task Force (IETF) hat sich OAuth 2.0 auf dem Markt weitgehend durchgesetzt (s. Abb. 1). Es wurde zum Standard für den