Neuauflage der OWASP Top 10 – 2017

Mehr Bewusstsein für Security

Im Entwickleralltag fristen Securityaspekte allzu oft ein Schattendasein. 2003 ging ein Projekt an den Start, das sich zum Ziel gesetzt hatte, dem etwas entgegenzusetzen: die OWASP Top 10, eine Liste der zehn kritischsten Sicherheitsrisiken für Webanwendungen. Diese Sicherheitsrisiken werden seither vom Open Web Application Security Project (OWASP) in regelmäßigen Abständen zusammengetragen und veröffentlicht. Zum Jahresende 2017 war es wieder so weit. Die neueste Version enthält zahlreiche Änderungen im Vergleich zur letzten Auflage von 2013. Neue Risiken sind hinzugekommen, alte sind ausgeschieden, und auch in der Rangfolge bestehender Schwachstellen gab es einige Bewegung.

von Dr. Marius Hofmeister

Wer auf die Neuauflage der OWASP Top 10 gewartet hatte, musste zuletzt etwas Geduld mitbringen. Denn während die Veröffentlichung ursprünglich für Sommer 2017 geplant war, verzögerte sich die Fertigstellung dann doch bis zum Ende des Jahres. Grund hierfür war eine kontroverse Diskussion der OWASP-Community über die veröffentlichte Vorabversion der Top 10. Dieser erste Release Candidate wurde mehrheitlich abgelehnt. Doch beginnen wir am Anfang der Geschichte.

Grundlage der OWASP Top 10, der weit über Entwicklerkreise hinaus bekannten Liste der zehn kritischsten Sicherheitsrisiken für Webanwendungen, ist ein umfangreicher Datenpool von Unternehmen, die auf Anwendungssicherheit spezialisiert sind. Um aktuelle Daten für die Neuauflage zu sammeln, wurde im Mai 2016 ein initialer Data Call auf der OWASP-Website veröffentlicht. Im Frühjahr 2017 folgte die Fertigstellung eines ersten Release Candidates auf Basis der eingereichten Daten. Darin gab es einige Änderungen zur Version von 2013: Zwei Risiken wurden

zusammengefasst, zwei neu hinzugefügt und eins entfernt. Auf dem OWASP Summit 2017 in London kam es dann zu intensiven Diskussionen. Insbesondere die Forward-looking Items, die neu hinzugefügten Risiken, die sich nicht auf die Datenlage, sondern auf Einschätzungen für in der Zukunft relevante Sicherheitslücken beziehen, wurden von den Teilnehmern mehrheitlich abgelehnt. Bemängelt wurde auch die insgesamt als unzureichend eingestufte Datenbasis. Daher wurde beschlossen, einen neuen Data Call auszurufen, der die bestehenden Daten anreichern sollte, und die Forward-looking Items über eine Umfrage unter Industrievertretern zu ermitteln.

Und so kam es dann letztendlich auch. Auf mehr als vierzig Einreichungen basiert die neue Version der Top 10, die Schwachstellen von mehreren hundert Organisationen und mehr als 100 000 Applikationen und APIs umfasst [1]. Die Rohdaten sind erstmals frei verfügbar und lassen sich online einsehen. Der Fragebogen für die Forward-looking Items, in dem die Teilnehmer die wichtigsten Schwachstellen für eine Aufnahme in die Top 10 bewerten sollten, wurde mehr als 500-mal ausgefüllt. Um die finale Rangfolge der Risiken festzulegen, wurden jeweils die Ausnutzbarkeit des Angriffs, die Verbreitung, die Auffindbarkeit sowie die technischen Auswirkungen datengestützt und auf Erfahrungen basierend ermittelt (Tabelle 1). Die Gesamtbewertung ergibt sich dann über eine Multiplikation der gemittelten Wahrscheinlichkeitsfaktoren mit der technischen Auswirkung. Besonderen Wert legt das OWASP darauf, dass unternehmens- und anwendungsspezifische Bedrohungsquellen sowie geschäftsspezifische Auswirkungen eines erfolgreichen Angriffs von den Nutzern der Top 10 selbst miteinbezogen werden. So können sich beispielsweise Risiken, die in den Top 10 auf den vorderen Plätzen zu finden sind, als im Unternehmenskontext zweitrangig herausstellen, wenn Bedrohungsquellen nicht oder nur eingeschränkt vorliegen oder Auswirkungen nicht in dem Maße eintreten können, wie es allgemein zu erwarten wäre.

Wie sehr die Diskussion über den ersten Release Candidate die finale Version der Top 10 beeinflusst hat, wird beim Blick auf diese schnell deutlich. Der zweite Release Candidate, der letztendlich ohne größere Beanstandungen in die finale Version umgewandelt wurde, bringt gegenüber dem ersten zahlreiche Neuerungen und Änderungen mit sich. Zudem wird eine neue Handschrift sichtbar, die durch einen personellen Wechsel im Projekt Einzug gehalten hat. So haben sich Dave Wichers und Jeff Williams, zwei der Gründerväter der Top 10, von der Spitze des Projekts zurückgezogen. Sie haben das Zepter an Andrew van der Stock übergeben, der das Projekt nun mit drei Stellvertretern leitet.

Mit der bewussten Neuformulierung der Top 10 wird den aktuellen technischen Entwicklungen Rechnung getragen. Zu nennen ist in diesem Zusammenhang der Einzug von Microservices, die traditionelle monolithische Applikationen verdrängen und neue Herausforderungen für die Security mit sich bringen. Die wachsende Anzahl von Web-APIs erhöht die potenzielle Angriffsfläche beträchtlich. Auch der Trend, dass Anwendungslogik im Rahmen von Single-Page-Applikationen zunehmend in Richtung Client wandert, stellt neue Ansprüche an die Sicherheit von Anwendungen. Moderne JavaScript-Frameworks wie Angular und React sowie die serverseitige Variante Node.js stehen im Fokus aktueller Entwicklungen. Ein weiteres wichtiges Merkmal der neuen Top 10 ist Transparenz: Neben der Tatsache, dass alle eingereichten Daten online frei verfügbar sind, fanden auch große Teile der Projektkommunikation auf GitHub statt [2]. Das macht die Top 10 für Interessierte noch nachvollziehbarer.

OWASP Top 10 - 2013		OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	U	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	U	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	☒	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	☒	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

Abb. 1: Neuerungen der OWASP Top 10 (Quelle: OWASP)

Drei Neuzugänge, zwei ausgeschiedene Risiken und eine Zusammenführung

Werfen wir nun einen Blick auf die Risiken in den Top 10 von 2017. Das Spannendste vorweg: Drei Punkte haben erstmals den Weg in die Top 10 gefunden (**Abb. 1**). Datengestützt wurden „XML External Entities“ (XXE) erhoben. Aus der Industrieumfrage hervorgegangen sind ferner „Insecure Deserialization“ und „Insufficient Logging & Monitoring“. Getreu dem Motto „Was zusammengehört, wird zusammengefasst“ wurden die Risiken „Missing Function Level Access Control“ und „Insecure Direct Object References“ vereint. Zumal die Aufsplittung der Punkte erst 2007 erfolgte, um Problemen auf der Zugriffsebene, die Daten oder Funktionalität betreffen, mehr Gewichtung zu verleihen.

Ganz aus den Top 10 verschwunden sind hingegen unsere alten Bekannten „Cross-Site Request Forgery“ (CSRF) und „Unvalidated Redirects and Forwards“. Mittlerweile sind CSRF-Abwehrmechanismen in zahlreiche Frameworks integriert. Das hat zur Folge, dass in lediglich fünf Prozent der untersuchten Applikationen eine CSRF-Schwachstelle festgestellt wurde. Das zweite genannte Risiko, Weiterleitungen auf unsichere Webseiten, wurde zwar noch in acht Prozent der untersuchten Applikationen entdeckt. Der neue Punkt „XML External Entities (XXE)“ war letztlich aber häufiger vertreten und führte zur Verdrängung des ersteren.

Wenig Bewegung unter den Top 3

Der erste Blick auf die OWASP Top 10 fällt natürlich immer auf die obersten Plätze. Dort hat sich wie in den Versionen zuvor auch 2017 verhältnismäßig wenig getan. Ein Dauerbrenner, der OWASP bereits seit seiner Gründung begleitet, sind Injection-Angriffe. Ihr immer noch so häufiges Vorkommen ist vermutlich auf die Existenz von Legacy-Anwendungen zurückzuführen. Schließlich kennt heute nahezu jeder Entwickler diese Gefährdungsart und besitzt entsprechende Kenntnisse, sie zu vermeiden. Ob SQL, LDAP, OS oder XPath Injections: Überall, wo vom Nutzer eingegebene Daten ungefiltert einen Interpreter erreichen, kann es zu unberechtigtem Zugriff und Manipulation von Daten kommen. Immerhin, und das ist vermutlich auch der einzige Trost, lassen sich diese Schwachstellen verhältnismäßig leicht durch Codeanalysen entdecken und beheben.

Das zweite Risiko „Broken Authentication" geht auf kritische Anwendungsschwachstellen im Bereich der Authentifizierung und des Sessionmanagements ein. Im Gegensatz zu Injection-Angriffen handelt es sich hierbei um eine Schwachstelle, deren Behebung eine Vielzahl unterschiedlicher Maßnahmen erforderlich machen kann. Dazu gehören so vielfältige Punkte wie die frühzeitige Planung und Umsetzung von Authentifizierungs- und Autorisierungsfunktionalitäten in Entwicklungsprojekten, das Einfordern sicherer Passwörter von den Nutzern sowie der korrekte Umgang zur Erzeugung und zum Verwerfen von Session-IDs. Ziel ist es auch, die Entführung fremder Sessions (Session Hijacking) sowie das Unterschieben bestehender Sessions (Session Fixation) zu unterbinden.

Kommen wir zum letzten Risiko unter den Top 3, Sensitive Data Exposure. Dieses Risiko hat deutlich an Bedeutung gewonnen und rückte daher von Platz sechs auf den dritten Rang. Hier entstehen Probleme, wenn Daten durch Anwendungen nicht ausreichend geschützt werden und ungehindert ausgelesen oder modifiziert werden können. Erster Schritt ist die Analyse der Schutzbedürftigkeit der verwendeten Daten, seien es Passwörter, personen- oder geschäftsbezogene Inhalte. Der korrekte Einsatz von sicheren Hashes ist dabei im Einzelfall ebenso zu prüfen wie die Notwendigkeit der grundsätzlichen Speicherung der Daten überhaupt und deren Übertragung mittels sicherer Protokolle. Ein guter Ansatzpunkt zur Überprüfung der Vertraulichkeit sensibler Daten in eigenen Anwendungen ist der OWASP-Standard ASVS (Application Security Verification Standard). Er stellt eine Metrik zur Einschätzung der Vertrauenswürdigkeit der eigenen Webanwendung sowie eine Anleitung zum Einbau von Sicherheitsmechanismen zur Verfügung.

Die neuen Risiken im Detail

Werfen wir nun einen detaillierteren Blick auf die drei Newcomer in den Top 10. Das Risiko „XML External Entities (XXE)" wurde datengestützt erhoben und landete auf Platz vier. Betroffen sind insbesondere Web Services, die XML verarbeiten. Der prominente Rang verwundert auf den ersten Blick, befinden wir uns doch in Zeiten, in denen JSON XML zunehmend den Rang abläuft. Dass XXE trotzdem datengestützt in die Top 10 eingezogen ist, belehrt uns in Hinblick auf Securityaspekte jedoch eines Besseren. XXE-Angriffe kommen zustande, weil der XML-Standard das Konzept einer extern gespeicherten Entität als Referenz vorsieht. Das führt insbesondere bei älteren oder nachteilig konfigurierten XML-Prozessoren leicht zu Problemen. Möglich ist unter anderem ein unautorisierter Zugriff auf Daten bis hin zu Denial-of-Service-(DoS-)Attacken. Mit dem in Listing 1 angegebenen XML-Code wäre unerwünschterweise ein Zugriff auf eine lokal gespeicherte Datei möglich.

Zu zweifelhaftem Ruhm hat es ferner die Billion-Laughs-Attacke gebracht. Mittels des in Listing 2 angegebenen XML ließe sich ein DoS-Angriff ausführen. Beim Parsen des Dokuments würde nämlich das Wurzelelement *lolz* ausgewertet. *Lolz* würde die Entität *&lol9;* einbinden, die wiederum einen String mittels zehn *&lol8;*-Elementen einschließt. Auf diese Weise käme es zu einer Milliarde *lol*-Elemente, die letztendlich etwa 3 GB beanspruchen würden [3] – gegebenenfalls ein Vielfaches des dem Prozess zur Verfügung stehenden Speicherplatzes.

Als Fazit rät OWASP in seinem XML External Entity (XXE) Prevention Cheat Sheet explizit dazu, bei Java XML-Parsern XXE generell zu deaktivieren [4]. Dort ist auch Schritt für Schritt beschrieben, wie dies im Einzelfall erfolgen kann.

Kommen wir nun zum Punkt „Insecure Deserialization", der zu den beiden Risiken gehört, die communitygestützt über die erwähnte Industrieumfrage erhoben wurden. Insbesondere Java-Entwicklern dürfte diese Sicherheitslücke aus den letzten Jahren bekannt sein, sorgte sie doch für einige Furore. Als Serialisierung wird gemeinhin die Abbildung strukturierter Daten auf eine sequenzielle Form zur Speicherung und Übertragung verstanden. Im konkreten Fall wird ein Endpunkt zur Deserialisierung zum Angriffspunkt. Das allein ist jedoch nicht ausreichend. Notwendig sind ferner sogenannte Gadgets, ausnutzbare Klassen auf dem Klassenpfad

Bedrohungsquellen	Angriffsvektoren	Schwachstelle Verbreitung	Schwachstelle Auffindbarkeit	Technische Auswirkungen	Auswirkungen auf das Unternehmen
Anwendungsspezifisch	Einfach	Sehr häufig	Einfach	Schwerwiegend	Anwendungs-/geschäftsspezifisch
	Durchschnittlich	Häufig	Durchschnittlich	Mittel	
	Schwierig	Selten	Schwierig	Gering	

Tabelle 1: OWASP Top 10 – Bewertungsschema für Risiken [1]

der angegriffenen Anwendung. Betroffen sind hierbei zahlreiche Bibliotheken und Produkte, die Java-Serialisierung als Teil ihrer Remote-Kommunikation nutzen, z. B. HTTP-Invoker, Java Message Service (JMS) oder Java Remote Method Invocation (RMI). Der jeweilige Endpunkt kann dann zur Remote Code Execution missbraucht werden. In Eigenentwicklungen wird daher empfohlen, auf Java-Serialisierung bei potenziell nicht vertrauenswürdigen Daten komplett zu verzichten. Wenn Fremdbibliotheken verwendet werden müssen, ist für das regelmäßige Einspielen von Patches zu sorgen [5].

Das dritte der neuen Risiken, Insufficient Logging & Monitoring, wurde ebenfalls communitygestützt erhoben. Positiv ausgedrückt: Wenn für bestimmte Ereignisse wie fehlerhafte Log-ins, Zugriffsfehler und serverseitige Inputvalidierungsfehler ein konsequentes Logging stattfindet, das überwacht wird und dann bei Grenzwertüberschreitung eine Alarmierung auslöst, können viele Angriffe zeitnah verhindert werden. Hintergrund ist die erschreckende Erkenntnis, dass es im Jahr 2016 durchschnittlich 191 Tage dauerte, eine ausgenutzte Einbruchstelle zu entdecken [6]. Vorherrschende Sicherheitslücken werden also in der Regel nicht zeitnah erkannt und bleiben zu lange geöffnet. Viel zu viel Zeit, die von Angreifern in aller Ruhe genutzt werden kann, tiefgreifende Schäden anzurichten. An dieser

Listing 1: XXE – Zugriff auf externe Datei [1]

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <!DOCTYPE foo [
 <!ELEMENT foo ANY >
 <!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
 <foo>&xxe;</foo>
```

Listing 2: Billion-Laughs-Attacke

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
<!ENTITY lol "lol">
<!ENTITY lol2 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
<!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
<!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
<!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
<!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
<!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
<!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
<!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

Stelle hilft der Einbau von Detection Points zur Protokollierung von Fehlern bei Zugriffsberechtigungen oder Inputvalidierung, die Einrichtung von Thresholds zur Reaktion auf Events sowie die automatische Auslösung von Responses, z. B. mittels Ausloggen oder Sperren von Benutzern [7]. Diese Vorgehensweise ist insbesondere für exponierte oder kritische Anwendungen sinnvoll. Mit dem Stopfen von Sicherheitslöchern allein ist es demnach heutzutage immer häufiger nicht getan: Die Intelligenz unserer Anwendungen hilft proaktiv dabei, auf Angriffe adäquat zu reagieren.

Die beiden neuen, durch die Industrieumfrage bestimmten Risiken belegen anschaulich den Mehrwert der Forward-looking Items: Aktuelle Securitythemen finden den Weg in die Top 10 und erlauben es, neue Akzente zu setzen, die eine rein datengestützte Vorgehensweise nicht vermocht hätte.

Und sonst?

Kommen wir zu den letzten, bislang noch nicht besprochenen Risiken, die in der Rangliste aufgeführt werden. Der neu zusammengefasste Punkt „Broken Access Control" bezieht sich auf das Risiko, das entsteht, wenn Zugriffsbeschränkungen für Nutzer nicht korrekt gesetzt sind. Angreifer können Schwachstellen nutzen, um Zugriff auf Daten und/oder Funktionalität zu erhalten. Sensible Daten können entwendet und Funktionalität kann missbraucht werden. Von großer Bedeutung ist hier beispielsweise, Zugriffsrechte nicht nur im (unsicheren) Client, sondern stets auch im Backend zu überprüfen.

Nur leichte Veränderungen in der Rangliste sind hingegen beim Risiko „Security Misconfiguration" zu beobachten. Im Fokus steht hier die fehlerhafte Konfiguration von Anwendungen, Frameworks, Applikations-, Web- und Datenbankservern sowie deren Plattformen. Fakt ist, dass Sicherheitseinstellungen im Einzelfall definiert, umgesetzt und gewartet werden müssen, weil die Voreinstellungen oft unsicher sind. Zudem gilt es, regelmäßig für das Einspielen von Updates zu sorgen. Ein Sicherheitsrisiko kann auch entstehen, wenn nicht verwendete Features unnötigerweise aktiviert oder installiert werden.

Klar an Bedeutung verloren hat glücklicherweise der allbekannte Cross-Site-Scripting-(XSS-)-Angriff, obwohl er immer noch das Risiko mit der zweithäufigsten Verbreitung in den Top 10 ist. Diese Schwachstelle wurde in rund zwei Dritteln der Applikationen gefunden. Ähnlich zu Injection-Angriffen liegt die Ursache eines erfolgreichen Angriffs hier darin, dass die Anwendung Nutzerdaten ohne Validierung aufnimmt. Zusätzlich werden diese auch noch ohne Prüfung ausgegeben. So kann der Browser der Websitebesucher attackiert und beispielsweise JavaScript-Schadcode auf ihm ausgeführt werden. Benutzerdaten können damit ausgelesen und Sessions übernommen werden. Abhilfe schaffen hier eine serverseitige Eingabevalidierung mittels Positivlisten, Output-Escaping nicht vertrauenswürdiger Daten sowie der explizite Schutz von Sessiondaten im Client.

Am unteren Ende der Top 10 konstant geblieben ist auf Platz neun „Using Components with known Vulnerabilities". Hier geht es darum, dass die Ausnutzung einer verwundbaren Komponente zu schwerwiegendem Datenverlust bis hin zur Serverübernahme führen kann. Meist werden Bibliotheken, Frameworks oder andere Softwaremodule mit vollen Berechtigungen ausgeführt. Applikationen, die Komponenten mit bekannten Schwachstellen einsetzen, können daher Schutzmaßnahmen unterlaufen und Angriffe ermöglichen. Hier hilft es, nicht verwendete Abhängigkeiten zu entfernen und kontinuierliches Monitoring zu betreiben, beispielsweise mittels OWASP Dependency Check.

Auch die Beachtung der Top 10 bietet keinen vollständigen Schutz

Die Aufmerksamkeit, die die neue Top 10 mit sich bringt, kann über eine Tatsache nicht hinwegtäuschen: Auch die vollständige Adressierung der Themen der Rangliste garantiert niemals einen vollständigen Schutz. Zu viele Sicherheitsrisiken existieren darüber hinaus, die auch aus dem jeweiligen unternehmensspezifischen und technischen Kontext entstehen. An einer individuellen Analyse der vorliegenden Applikation führt daher kein Weg vorbei. Nichtsdestotrotz sollten die Top 10 aktiv dafür genutzt werden, wofür sie stets gedacht waren: als mächtiges Werkzeug, um Bewusstsein für Security in Webanwendungen zu schaffen. Die pointierte und strukturierte Aufmachung hilft dabei, auch nicht technikaffine Personen von der Notwendigkeit der Adressierung von Security-Issues zu überzeugen und sie bei deren sukzessiven Abarbeitung an die Hand zu nehmen.

Dr. Marius Hofmeister ist als Senior Consultant und Entwicklungsleiter bei OPITZ CONSULTING Deutschland beschäftigt und widmet sich schwerpunktmäßig dem Entwurf und der Implementierung von Webanwendungen in Back- und Frontend. Sein besonderes Interesse gilt dabei den Securityaspekten der verwendeten Technologien.

Links & Literatur

[1] OWASP Top 10 – Version 2017: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

[2] OWASP Top 10 GitHub Issues: https://github.com/OWASP/Top10/issues

[3] Sullivan, Bryan: „XML Denial of Service Attacks and Defenses": https://msdn.microsoft.com/en-us/magazine/ee335713.aspx

[4] OWASP XML External Entity (XXE) Prevention Cheat Sheet: https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Prevention_Cheat_Sheet#Java

[5] Schneider, Christian: „Angriff auf die Java-Deserialisierung", in: Java Magazin 10.2016

[6] Cost of Data Breach Study: https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=SEL03130WWEN&

[7] Schadow, Dominik: „Die Webanwendung schlägt zurück", in: Java Magazin 10.2016