Application Security Verification Standard 4.0.3

Finale Version

Oktober 2021

Table of Contents

Frontispiece

Über diesen Standard

Der Application Security Verification Standard ist eine Sammlung von Anforderungen und Tests an die Sicherheit von Anwendungen, die Architekten, Entwickler, Sicherheitsexperten, Toolhersteller und Verbraucher verwenden können, um sichere Anwendungen zu definieren, zu erstellen, zu testen und zu verifizieren.

Copyright and License

Version 4.0.3, Oktober 2021

Projektleitung

Andrew van der Stock	Daniel Cuthbert	Jim Manico
Josh C Grossman	Elar Lang	

Hauptautoren

Abhay Bhargav	Benedikt Bauer	Osama Elnaggar
Ralph Andalis	Ron Perris	Sjoerd Langkemper
Tonimir Kisasondi		

Weitere Autoren und Reviews von

Aaron Guzman	Alina Vasiljeva	Andreas Kurtz	Anthony Weems	Barbara Schachner
Christian Heinrich	Christopher Loessl	Clément Notin	Dan Cornell	Daniël Geerts
David Clarke	David Johansson	David Quisenberry	Elie Saad	Erlend Oftedal
Fatih Ersinadim	Filip van Laenen	Geoff Baskwill	Glenn ten Cate	Grant Ongers
hello7s	Isaac Lewis	Jacob Salassi	James Sulinski	Jason Axley
Jason Morrow	Javier Dominguez	Jet Anderson	jeurgen	Jim Newman
Jonathan Schnittger	Joseph Kerby	Kelby Ludwig	Lars Haulin	Lewis Ardern
Liam Smit	lyz-code	Marc Aubry	Marco Schnüriger	Mark Burnett
Philippe De Ryck	Ravi Balla	Rick Mitchell	Riotaro Okada	Robin Wood
Rogan Dawes	Ryan Goltry	Sajjad Pourali	Serg Belkommen	Siim Puustusmaa

Ståle Pettersen	Stuart Gunter	Tal Argoni	Tim Hemel	Tomasz Wrobel
Vincent De Schutter	Mike Jang			

Sollte ein Dank fehlen, eröffne bitte ein Ticket bei GitHub, so dass er in künftigen Updates berücksichtigt werden kann.

Der Application Security Verification Standard 4.0 wurde von denjenigen aufgebaut, die bei ASVS 1.0 im Jahre 2008 bis 3.0 im Jahre 2016 mitgewirkt haben. Viele der Struktur- und Prüfungselemente, die im heutigen ASVS immer noch vorhanden sind, wurden ursprünglich von Mike Boberski, Jeff Williams und Dave Wichers geschrieben Es gibt aber noch viele andere Mitwirkende. Dank all denen, die sich bisher eingebracht haben. Für die umfassende Liste aller, die Beiträge zu früheren Versionen geleistet haben, verweisen wir auf die jeweilige Vorversion.

Vorwort

Willkommen zum Application Security Verification Standard (ASVS) Version 4.0. Der ASVS ist eine von der Community gesteuerte Initiative, die Rahmenbedingungen für Sicherheitsanforderungen und -maßnahmen auf Anwendungsebene schaffen will. Ihr Schwerpunkt liegt auf der Definition der funktionalen und nicht funktionalen Sicherheitsmaßnahmen, die beim Entwerfen, Entwickeln und Testen moderner Webanwendungen und Webservices erforderlich sind. Version 4.0.3 ist das dritte kleine Update der Version 4.0. Es bereinigt Rechtschreibfehler und beschreibt Anforderungen deutlicher, ohne jedoch wesentliche Änderungen, wie das Ändern, Verschärfen oder Hinzufügen von Anforderungen, einzuführen. An einigen Stellen, an denen wir es für geboten hielten, haben wir Anforderungen abgeschwächt. Einige redundante Anforderungen sind entfallen, jedoch ohne neu zu nummerieren.

ASVS v4.0 ist das Ergebnis gemeinschaftlicher Anstrengungen und des Feedbacks der Branche der letzten zehn Jahre. Wir haben versucht, die Anwendung des ASVS für viele unterschiedlich geartete Anwendungsfälle und für die Dauer jedes sicheren Softwareentwicklungszyklus einfacher zu gestalten.

Risikoanalyse ist in gewissem Maße immer subjektiv. Wir gehen daher davon aus, dass es wohl nie eine 100%ige Einigung über den Inhalt eines Webanwendungsstandards, einschließlich des ASVS, geben wird. Es ist eine Herausforderung diese mit einen universellem und für alle gleichen Standard zu verallgemeinern. Wir hoffen jedoch, dass diese neue Version einen Schritt in die richtige Richtung darstellt und die Konzepte verbessert, die in diesem Standard eingeführt wurden.

Was ist neu in Version 4.0

Die wichtigste Änderung in dieser Version ist die Annahme der NIST 800-63-3-Richtlinien zur digitalen Identität, mit denen moderne, evidenzbasierte und erweiterte Authentifizierungsmaßnahmen eingeführt werden. Obwohl wir einen gewissen Widerstand bei der Anpassung an einen erweiterten Authentifizierungsstandard erwarten, halten wir es für wesentlich, dass Standards angepasst werden, vor allem wenn ein anderer angesehener Anwendungssicherheitsstandard evidenzbasiert ist.

Informationssicherheitsstandards sollten versuchen, die Anzahl der spezifischen Anforderungen zu minimieren, damit konforme Organisationen nicht über konkurrierende oder inkompatible Maßnahmen entscheiden müssen. Die OWASP Top 10 2017 und nun auch der OWASP Application Security Verification Standard sind jetzt in Bezug auf Authentifizierung und Sessionmanagement an NIST 800-63 angepasst worden. Wir ermutigen andere Normungsgremien, mit uns, NIST und anderen zusammenzuarbeiten, um allgemein anerkannte Maßnahmen der Anwendungssicherheit zu erarbeiten, wodurch die Sicherheit maximiert und Compliancekosten minimiert werden.

In ASVS 4.0 wurde ein neues Nummerierungsschema eingeführt. Dadurch konnten wir die Lücken weggefallener Kapitel schließen und längere Kapitel neu unterteilen. Weiterhin minimiert es die Anzahl der Maßnahmen, die vom Entwickler oder Team einzuhalten sind. Wenn eine Anwendung beispielsweise kein JWT verwendet, so gilt der gesamte Abschnitt zu JWT für deren Sessionmanagement nicht. Wir haben eine umfassende Zuordnung zur Common Weakness Enumeration (CWE), eine der am häufigsten gewünschten Funktionsanforderungen, die wir im letzten Jahrzehnt hatten, neu eingeführt. Mit der CWE-Zuordnung können Toolhersteller und Benutzer von Software zum Schwachstellenmanagement die Ergebnisse anderer Tools und früherer ASVS-Versionen abgleichen. Um Platz für den CWE-Eintrag zu schaffen, haben wir die Spalte „Seit“ entfernt. Sie ist in der neuen Nummerierung auch weniger sinnvoll. Nicht jeder Anforderung, z. B. sehr generischen, konnte eine CWE zugeordnet werden. Da CWE sehr vielfältig ist, haben wir versucht, die am meisten verwendete und nicht unbedingt die genaueste Übereinstimmung zu verwenden. Wir begrüßen die laufende Diskussion mit der CWE-Community, diese Lücke im Allgemeinen zu schließen.

Wir haben uns bemüht, die Anforderungen in Bezug auf die OWASP Top 10 2017 und die OWASP Proactive Controls 2018 umfassend zu erfüllen und zu übertreffen. Da die OWASP Top 10 2017 lediglich das Mindestmaß zur Vermeidung von Fahrlässigkeit darstellt, haben wir bewusst alle Anforderungen außer denen der spezifischen Logging Top 10 zu Maßnahmen der Stufe 1 gemacht. Dies vereinfacht die Anpassung an den derzeitigen Sicherheitsstandard für diejenigen, die OWASP Top 10 übernehmen.

ASVS 4.0 Stufe 1 soll die Anforderungen von PCI DSS 3.2.1, Abschnitt 6.5 für Anwendungsdesign, Programmierung, Tests, Überprüfungen von sicheren Codes und Penetrationstests vollständig abdecken.

Daher wurden der Pufferüberlauf sowie die unsicheren Speicheroperationen in V5 und unsichere speicherbezogene Compilierflags in V14 zusätzlich zu den bestehenden Anforderungen für die Verifizierung von Anwendungen und Webservices aufgenommen.

Wir haben die Umstellung des ASVS von monolithischen, ausschließlich serverseitigen Maßnahmen auf Sicherheitsmaßnahmen für alle modernen Anwendungen und APIs abgeschlossen. In Zeiten funktionaler Programmierung, serverloser API, Mobiltelefon, Cloud, Containern, CI / CD und DevSecOps, Föderation und mehr können wir die moderne Anwendungsarchitektur nicht länger ignorieren. Moderne Anwendungen werden ganz anders gestaltet als diejenigen, die zur Zeit der Veröffentlichung des ursprünglichen ASVS im Jahr 2009 erstellt wurden. Dar ASVS muss immer weit in die Zukunft schauen, damit wir unserer Hauptzielgruppe - den Entwicklern - fundierte Ratschläge geben können. Wir haben alle Anforderungen geändert oder gestrichen, die davon ausgehen, dass Anwendungen auf Systemen ausgeführt werden, die einer einzelnen Organisation gehören.

Aufgrund der Größe des ASVS 4.0 sowie unseres Ziels, den ASVS als Basis aller anderen ASVS-Ausarbeitungen zu benutzen, haben wir den mobilen Bereich zugunsten des Mobile Application Security Verification Standard (MASVS) eingestellt. Der Anhang zum Internet of Things (IoT) wird in einem künftigen ASVS IoT bei der Pflege des OWASP IoT-Projektes erscheinen. Wir haben eine frühe Vorschau des ASVS IoT in Anhang C aufgenommen. Wir danken sowohl dem OWASP Mobile Team als auch dem OWASP IoT Projektteam für ihre bei ASVS geleistete Unterstützung und freuen uns darauf, in Zukunft bei der Erstellung ergänzender Standards mit ihnen zusammenzuarbeiten.

Schließlich haben wir weniger wirksame Maßnahmen bereinigt. Im Laufe der Zeit ist der ASVS zu einer umfangreichen Ansammlung von Maßnahmen angewachsen, die aber nicht alle gleich gut sind. Die Bereinigung von Anforderungen mit nur geringer Auswirkung könnte künftig noch weiter gehen. In einer zukünftigen Ausgabe des ASVS wird das Common Weakness Scoring System (CWSS) dazu beitragen, die wirklich wichtigen Maßnahmen weiter zu priorisieren.

Ab Version 4.0 wird sich der ASVS ausschließlich darauf konzentrieren, der führende Webanwendungs- und Webservicestandard zu sein, welcher die traditionelle und moderne Anwendungsarchitektur sowie agile Sicherheitspraktiken und die DevSecOps-Kultur abdeckt.

Verwendung des ASVS

Der ASVS hat die zwei Hauptziele:

- Organisationen bei der Entwicklung und Pflege sicherer Anwendungen zu unterstützen.
- Anbietern von Sicherheitsdienstleistungen oder -werkzeugen und deren Kunden die Möglichkeit zu geben, ihre Anforderungen und Angebote aufeinander abzustimmen.

Stufen zur Verifizierung der Anwendungssicherheit

Der Application Security Verification Standard definiert drei Stufen der Sicherheitsverifikation, wobei jede Stufe an Tiefe zunimmt.

- Stufe 1 ist für geringe Sicherheitsanforderungen gedacht und lässt sich mittels Pentests prüfen.
- Stufe 2 ist für Anwendungen, die sensible Daten enthalten und diese schützen müssen. Das ist die empfohlene Stufe für die meisten Anwendungen.
- Stufe 3 ist für die kritischsten Anwendungen, z. B. solche, die hochwertige Transaktionen durchführen, sensible medizinische Daten enthalten oder aus anderen Gründen ein Höchstmaß an Vertrauen erfordern.

Jede Stufe des ASVS enthält eine Liste von Sicherheitsanforderungen. Jede dieser Anforderungen kann auch auf sicherheitsspezifische Merkmale und Fähigkeiten abgebildet werden, die von den Entwicklern in die Software eingebaut werden müssen.

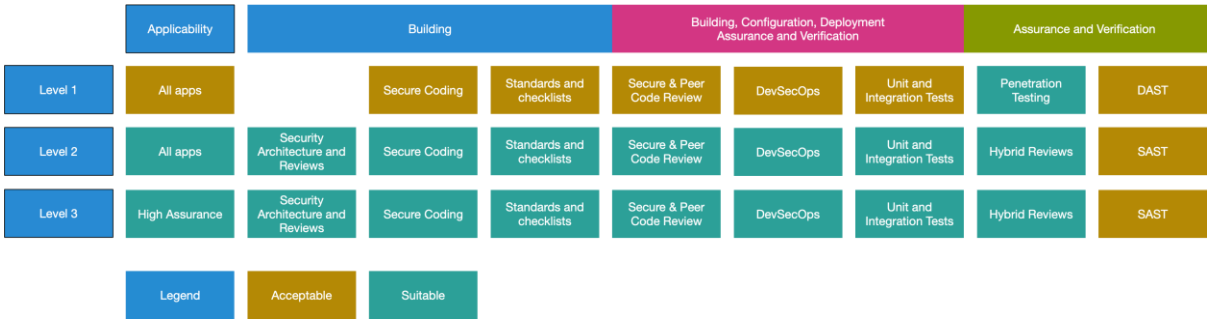

Figure 1 - Stufen des OWASP Application Security Verification Standard 4.0

Stufe 1 ist die einzige Stufe, die sich komplett mittels Pentests prüfen lässt. Alle anderen Stufen benötigen Zugang zu Dokumentation, Quellcode, Konfiguration und den am Entwicklungsprozess beteiligten Personen. Jedoch stellen Black Box Tests ohne Dokumentation und ohne Quellen keine wirksame Absicherung dar. Sie müssen langfristig ergänzt werden. Böswillige Angreifer haben sehr viel Zeit. Die meisten Penetrationstests hingegen sind innerhalb von wenigen Wochen abgeschlossen. Die Verteidiger müssen alle Sicherheitsmaßnahmen umsetzen, alle Schwachstellen finden und beheben sowie böswillige Akteure innerhalb einer angemessenen Zeit aufspüren und Gegenmaßnahmen ergreifen. Böswillige Akteure haben im Wesentlichen unendlich viel Zeit und benötigen nur eine einzige Lücke in der Verteidigung, eine einzige Schwachstelle, um erfolgreich zu sein. Black Box Tests, die oft am Ende der Entwicklung schnell oder gar nicht durchgeführt werden, sind völlig ungeeignet, dieser Asymmetrie gerecht zu werden.

Die letzten 30 Jahre haben gezeigt, dass Black Box Tests immer wieder kritische Sicherheitsprobleme übersehen, welche direkt zu weiteren, massiveren Problemen geführt haben. Wir befürworten ausdrücklich den Einsatz einer breiten Palette von Maßnahmen zur Gewährleistung und Verifizierung der Sicherheit, einschließlich des Ersatzes der Penetrationstests durch quellcodegeführte (hybride) Penetrationstests auf Stufe 1, mit vollem Zugang zu den Entwicklern und der Dokumentation während des gesamten Entwicklungsprozesses. Die Finanzaufsichtsbehörden dulden keine externen Finanzprüfungen ohne Zugang zu den Büchern, zu Stichproben von Geschäftsvorgängen oder zu den Personen, welche die Prüfungen durchführen. Derselbe Standard an Transparenz muss von der Industrie und den Regierungen auch im Bereich der Softwareentwicklung gefordert werden.

Wir befürworten ausdrücklich die Verwendung von Sicherheitstools innerhalb des Entwicklungsprozesses. DAST- und SAST-Tools können in die Buildpipeline eingebunden werden, um leicht zu findende Sicherheitsprobleme aufzuspüren, die niemals vorhanden sein dürfen. Automatisierte Tools und Onlinescans können höchsten nur noch die Hälfte des ASVS ohne menschliches Zutun erfassen. Ist eine umfassende Testautomatisierung für jeden Buildprozess erforderlich, wird eine Kombination aus benutzerdefinierten Unit- und Integrationstests zusammen mit den vom Buildprozess veranlassten Onlinescans verwendet. Mängel der Geschäftslogik und der Zugriffskontrolle lassen sich nur mit menschlicher Hilfe testen, am besten in Unit- und Integrationstests.

Verwendung dieses Standards

Eine der besten Möglichkeiten, den Application Security Verification Standard zu nutzen, ist seine Verwendung als Vorlage für die Erstellung einer Checkliste für Anforderungen zur sicheren Softwareentwicklung, die speziell auf Ihre Anwendung, Plattform oder Organisation zugeschnitten ist. In dem Sie die Anforderungen des ASVS auf Ihre Anwendungsfälle passend zuschneiden, setzen Sie den Schwerpunkt auf die Sicherheitsanforderungen, die für Ihre Projekte und Umgebungen am wichtigsten sind.

Zu den Sicherheitszielen aller Stufen gehören die Gewährleistung der Vertraulichkeit (z.B. Verschlüsselung), der Integrität (z.B. Transaktionen, Eingabevalidierung), der Verfügbarkeit (z.B. ordnungsgemäße Lastverteilung), der Authentifizierung (auch zwischen Systemen), der Nichtabstreitbarkeit, der Autorisierung und der Protokollierung.

Stufe 1 - Erste Schritte oder das absolute Minimum

Eine Anwendung erreicht ASVS Stufe 1, wenn sie einen ausreichenden Schutz gegen leicht zu entdeckende Schwachstellen bietet, die auch in den OWASP Top 10 und anderen ähnlichen Checklisten aufgeführt sind. Stufe 1 ist das für alle Anwendungen erforderliche absolute Minimum. Sie ist auch nützlich als erster Schritt eines mehrstufigen Prozesses oder für den Fall, wenn Anwendungen keine sensiblen Daten verarbeiten und daher nicht die strengeren Anforderungen der Stufen 2 oder 3 benötigen. Anforderungen der Stufe 1 können entweder automatisch durch Tools oder manuell ohne Zugriff auf den Quellcode überprüft werden.

Bedrohungen auf Stufe 1 gehen von Angreifern aus, die einfache und wenig aufwendige Techniken verwenden, um leicht zu findende und leicht ausnutzbare Schwachstellen zu ermitteln. Sie ist ungeeignet zur Abwehr entschlossener Angreifer, die die Anwendung gezielt ins Visier nehmen. Wenn Ihre Anwendung hochwertige Daten verarbeitet, werden Sie kaum mit einer Überprüfung der Stufe 1 zufrieden sein.

Stufe 2 - Die meisten Anwendungen

Eine Anwendung erreicht ASVS Level 2 (oder Standard), wenn sie die meisten Risiken, die heutzutage mit dem Einsatz von Software verbunden sind, angemessen abwehrt. Dabei wird sichergestellt, dass Sicherheitsmaßnahmen wirksam umgesetzt sind. Stufe 2 eignet sich in der Regel für Anwendungen, die wichtige Geschäftsvorgänge abwickeln, einschließlich solcher, die Informationen aus dem Gesundheitswesen verarbeiten, geschäftskritische oder sensible Funktionen umsetzen oder andere sensible Vermögenswerte verarbeiten. Sie eignet sich auch für Branchen, in denen Integrität ein kritischer Aspekt zum Schutz ihres Geschäfts ist, wie z. B. die Spieleindustrie, um Betrügern und Gamehacks entgegenzuwirken.

Maßnahmen zum Schutz der Anwendung auf Stufe 2 gehen in der Regel von geschickten und motivierten Angreifern aus, die sich auf bestimmte Ziele konzentrieren und Tools und Techniken einsetzen, die sehr versiert und wirksam Schwachstellen in Anwendungen entdecken und ausnutzen.

Stufe 3 - Hoher Wert, hohe Zuverlässigkeit oder hohe Sicherheit

ASVS Stufe 3 (oder Erweitert) ist die höchste Verifizierungsstufe innerhalb des ASVS. Diese Stufe ist in der Regel Anwendungen vorbehalten, die ein erhebliches Maß an geprüften Sicherheitsniveau erfordern. Diese sind z.B. im militärischen Bereich, dem Gesundheitssektor oder kritischen Infrastrukturen zu finden. Organisationen können ASVS Stufe 3 bei Anwendungen benötigen, die kritische Funktionen ausführen und bei denen ein Ausfall die Betriebsabläufe der Organisation oder sogar ihre Überlebensfähigkeit erheblich beeinträchtigen könnte. Eine Anwendung erreicht ASVS-Stufe 3, wenn sie angemessen vor fortgeschrittenen Schwachstellen geschützt ist und die Grundsätze eines guten Sicherheitsdesigns aufweist. Eine Prüfung auf ASVS-Stufe 3 erfordert eine tiefgründigere Analyse der Architektur, der Programmierung und der Testprozesse als alle anderen Ebenen. Eine sichere Anwendung wird auf sinnvolle Weise modularisiert. So können Resilienz,

Skalierbarkeit und vor allem Kapselung leichter umgesetzt werden, da sich jedes Modul selbst um seine eigenen Sicherheitsmaßnahmen kümmern kann (Defense in Depth). Die Umsetzung dieser Maßnahmen muss ordnungsgemäß dokumentiert werden.

Auswahl der ASVS-Stufen in der Praxis

Unterschiedliche Angreifer haben unterschiedliche Beweggründe. Einige Branchen verfügen über einzigartige Informations- und Technologiewerte sowie branchenspezifische gesetzliche Regulierungen. Wir raten Organisationen dringend dazu, ihre spezifischen Risikomerkmale auf der Grundlage ihres Geschäftsmodells eingehend zu prüfen. Auf dieser Basis wählen sie dann die geeignete ASVS-Stufe aus.

Referenzen auf Anforderungen des ASVS

Jede Anforderung des ASVS wird durch v<Version>-<Kapitel>.<Abschnitt>.<Laufende Nummer> identifiziert, z.B.: v4.0.3-1.11.3

- Die <Version> benennt die Version des ASVS. Im Laufe der Entwicklung des ASVS kann sich die Nummerierung zwischen den Versionen des Standards ändern. Das sich Referenzen ohne Versionsangabe immer auf die aktuelle Version des Standards beziehen, kann dies problematisch werden. Das „v“ wird dabei klein geschrieben.
- Das <Kapitel> gibt Anforderungen das Kapitel des Standards an, dem die Anforderung entstammt, z.B.: alle Anforderungen mit der Nummer 1.#.# sind an die Architektur.
- Der <Abschnitt> referenziert auf den Abschnitt, in dem die Anforderung beschrieben wird, z.B.: alle Anforderungen mit der Nummer 1.11.# beziehen sich auf die architekturellen Anforderungen an die Geschäftslogik.
- Innerhalb der Abschnitte gibt die <Laufende Nummer> schließlich die konkrete Anforderung an.

Die ASVS Anforderungen werden in CSV, JSON und anderen Formaten zur Verfügung gestellt.

Zertifizierung

ASVS-Zertifizierungen und Gütesiegel

OWASP ist eine herstellerneutrale, gemeinnützige Organisation. Sie zertifiziert derzeit keine Hersteller, Prüfstellen oder Software. Alle derartigen Versicherungen, Gütesiegel oder Zertifizierungen werden von OWASP nicht offiziell überprüft, registriert oder zertifiziert. Jede Organisation sollte in Bezug auf Aussagen eines Dritten, der behauptet, ASVS-zertifiziert zu sein, vorsichtig sein. Dies ist jedoch kein Verbot, solche Assurancedienstleistungen anzubieten, solange sie keine offizielle OWASP-Zertifizierung geltend machen.

Leitfaden für zertifizierende Organisationen

Der Application Security Verification Standard kann als Open Book Prüfung von Anwendungen verwendet werden, was den offenen und ungehinderten Zugang zu Schlüsselressourcen wie Architekten und Entwicklern, Projektdokumentation, Quellcode, authentifiziertem Zugang zu Testsystemen, einschließlich des Zugangs zu einem oder mehreren Benutzerkonten in jeder Rolle, insbesondere für L2- und L3-Verifikationen einschließt.

Klassischerweise werden Penetrationstests und Security Code Reviews so dokumentiert, dass nur fehlgeschlagene Tests im Abschlussbericht erscheinen. Eine zertifizierende Organisation muss jedoch in jedem Bericht den Umfang der Verifizierung angeben. Insbesondere, wenn eine Schlüsselkomponente außerhalb des Umfangs liegt, wie z.B. die SSO-Authentifizierung sowie eine Zusammenfassung der Verifizierungsergebnisse, einschließlich der bestandenen und fehlgeschlagenen Tests, mit klaren Angaben zur Korrektur der gefundenen Probleme.

Bestimmte Anforderungen des ASVS sind möglicherweise nicht auf die zu testende Anwendung anwendbar. Wenn Sie beispielsweise Ihren Kunden eine zustandslose API ohne eine Clientimplementierung zur Verfügung stellen, sind viele der Anforderungen im V3 Session Management nicht direkt anwendbar. In solchen Fällen kann eine zertifizierende Organisation immer noch die volle ASVS-Konformität bestätigen, muss aber im Bericht klar einen Grund für die Nichtanwendbarkeit der ausgeschlossenen Verifikationsanforderungen angeben.

Das Aufbewahren von detaillierten Arbeitspapieren, Screenshots, Filmen, Skripten zur zuverlässigen wiederholten Auswertung eines Tests sowie von elektronischen Testaufzeichnungen, wie z. B. Proxyprotokollen und zugehörigen Notizen, wie z. B. einer CleanUp-List, gilt als gängige Praxis. Diese können für kritische Entwickler als Beleg für die Ergebnisse hilfreich sein. Es reicht also nicht aus, einfach ein Tool laufen zu lassen und über die Fehler zu berichten: Dies liefert keinen ausreichenden Beweis dafür, dass alle Probleme auf einer Zertifizierungsebene gründlich getestet und geprüft worden sind. In Streitfällen sollte es ausreichende Nachweise geben, um zu belegen, dass jede einzelne verifizierte Anforderung tatsächlich getestet wurde.

Prüfmethode

Zertifizierende Organisationen können die geeigneten Prüfmethoden frei wählen, sollten diese aber in einem Bericht angeben. Je nach der zu testenden Anwendung und der Anforderung können unterschiedliche Testmethoden verwendet werden: Die Wirksamkeit der Eingabevalidierung einer Anwendung kann beispielsweise sowohl mit einem manuellen Penetrationstest als auch mit Hilfe von Quellcodeanalysen geprüft werden.

Die Rolle automatisierter Sicherheitstesttools

Der Einsatz von automatisierten Penetrationstests wird empfohlen, um eine möglichst hohe Abdeckung zu erreichen. Die ASVS-Verifizierung kann jedoch nicht ausschließlich mit automatisierten Penetrationstesttools durchgeführt werden. Während die meisten Anforderungen in L1 mit automatisierten Tests durchgeführt werden kann, ist die Mehrheit der Anforderungen der Stufen 2 und 3 nicht für automatisierte Penetrationstests geeignet. Die Grenzen zwischen automatisierten und manuellen Tests verschwimmen mit zunehmender Reife der Anwendungssicherheitsindustrie immer mehr. Automatisierte Tools werden häufig von Experten angepasst, und manuelle Tester nutzen oft eine Vielzahl von automatisierten Werkzeugen.

Die Rolle automatisierter Sicherheitstests

In Version 4.0 haben wir uns entschieden, L1 komplett penetrationstestfähig zu machen, ohne Zugriff auf Quellcode, Dokumentation oder Entwickler. Zwei Protokollierungselemente, die zur Einhaltung der OWASP

Top 10 2017 A10 erforderlich sind, erfordern Interviews, Screenshots oder eine andere Sammlung von Nachweisen, wie sie auch in der OWASP Top 10 2017 erforderlich sind. Das Testen ohne Zugang zu den notwendigen Informationen ist jedoch keine ideale Methode der Sicherheitsüberprüfung, da die Quelle nicht überprüft wird, Bedrohungen und fehlende Maßnahmen nicht identifiziert werden und ein weitaus gründlicherer Test in kürzerer Zeit nicht durchgeführt wird. Zur Durchführung einer Prüfung auf den Stufen 2- oder 3 ist der Zugang zu Entwicklern, der Dokumentation, dem Code sowie der Zugang zu einer Testanwendung mit Testdaten erforderlich. Penetrationstests, die auf diesen Ebenen durchgeführt werden, erfordern diese Zugriffsebene, die wir „hybride Überprüfungen“ oder „hybride Penetrationstests“ nennen.

Andere Verwendungszwecke für den ASVS

Neben der Verwendung zur Bewertung der Sicherheit einer Anwendung haben wir eine Reihe anderer möglicher Anwendungen für den ASVS identifiziert.

Detaillierte Anleitung zur Sicherheitsarchitektur

Eine der häufigeren Verwendungen des Application Security Verification Standards ist seine Verwendung als Ressource für Sicherheitsarchitekten. So fehlt z. B. in der Sherwood Applied Business Security Architecture (SABSA) eine Menge an Informationen, die für eine gründliche Überprüfung der Anwendungssicherheitsarchitektur erforderlich sind. Der ASVS kann verwendet werden, um diese Lücken zu füllen, indem Sicherheitsarchitekten bessere Maßnahmen für häufige Probleme des Datenschutzes oder der Eingabevalidierungsstrategien wählen können.

Ersatz für Standardchecklisten für sichere Softwareentwicklung

Viele Organisationen können von der Übernahme des ASVS profitieren, indem sie sich für eine der drei Stufen entscheiden, oder den ASVS domänenspezifisch anpassen. Wir befürworten diese Anpassungen, solange die Rückverfolgbarkeit gewährleistet ist. Wenn also eine Anwendung die Anforderung 4.1 bestanden hat, bedeutet dies dasselbe für die angepassten Varianten wie für den Standard aus dem sie sich entwickelt haben.

Als Leitfaden für automatisierte Unit- und Integrationstests

Der ASVS ist so konzipiert, dass er in hohem Maße testbar ist, mit Ausnahme der Anforderungen an die Architektur und den bösartigen Code. Durch die Erstellung von Unit- und Integrationstests, die spezifische und relevante Testfälle für Fuzz-Tests und Missbrauchsfälle prüfen, wird die Anwendung mit jedem Build nahezu selbstverifizierend. Beispielsweise können zusätzliche Tests für die Testsuite eines Logincontrollers erstellt werden, der die Parameter des Benutzernamens auf gängige Standardbenutzernamen, das Erraten von Benutzernamen, Brute Force Angriffe, LDAP- und SQL-Injektion und XSS testet. In ähnlicher Weise sollte ein Test des übermittelten Passwortes auf gängige Passwörter, die Passwortlänge, Null Byte Injection, Entfernen des Parameters, XSS, etc. umfassen.

Für Schulungen zur sicheren Softwareentwicklung

Der ASVS kann auch dazu verwendet werden, um Merkmale sicherer Software zu definieren. Viele Kurse zur sicheren Softwareentwicklung sind tatsächlich Kurse „Ethisches Hacking mit einem Hauch von Softwareentwicklung“. Sie orientieren sich gern an den OWASP Top 10 der Programmiersünden. Wissen über Hacking hilft den Entwicklern nicht unbedingt, einen sichereren Code zu schreiben. Anstelle dieses negativen Ansatzes sollten Kurse für sichere Softwareentwicklung den ASVS, mit dem Schwerpunkt auf seinen Sicherheitsmaßnahmen, verwenden.

Als Treiber für agile Anwendungssicherheit

Der ASVS kann in einem agilen Entwicklungsprozess als Rahmen verwendet werden, um spezifische Aufgaben zu definieren, die vom Team zum Erhalt eines sicheren Produkts implementiert werden müssen. Eine Vorgehensweise wäre: Man beginnt mit Stufe 1 und verifiziert die spezifische Anwendung oder das System gemäß Anforderungen des ASVS. Für fehlende Maßnahmen werden spezifische Tickets oder Aufgaben im Backlog generiert. Dies hilft bei der Priorisierung bestimmter Aufgaben (oder beim Grooming) und macht die Sicherheit im agilen Prozess sichtbar. Dies kann auch zur Priorisierung von Revisions- und Überprüfungsaufgaben in der Organisation verwendet werden. Bestimmte ASVS-Anforderungen können für ein bestimmtes Teammitglied Antrieb für eine Überprüfung, ein Refactoring oder eine Revision sein. Sie werden als „Offen“ im Backlog zur Abarbeitung aufgeführt.

Zur Beschaffung sicherer Software

Der ASVS ist ein großartiger Rahmen, der bei der Beschaffung sicherer Software oder der Beschaffung von Entwicklungsdienstleistungen hilfreich ist. Der Käufer stellt die Anforderung, dass die Software, auf ASVS-Ebene X entwickelt wird und verlangt vom Verkäufer den entsprechenden Nachweis. Dies funktioniert gut, wenn es mit dem OWASP-Vertragsanhang für sichere Software kombiniert wird.

V1 Architektur, Design und Threat Modeling

Ziel

In vielen Organisationen ist die Sicherheitsarchitektur fast in Vergessenheit geraten. Die Tage des Enterprise Architekten sind im Zeitalter von DevSecOps vorbei. Der Bereich der Anwendungssicherheit muss aufholen, agile Sicherheitsprinzipien übernehmen und gleichzeitig wichtige Grundsätze der Sicherheitsarchitektur wieder in die Softwareentwicklung einführen. Architektur ist keine Implementierung, sondern die Art und Weise, über ein Problem nachzudenken, das potenziell viele verschiedene aber keine einzig richtige Antwort hat. Nur allzu oft wird Sicherheit zu unflexibel betrachtet, und man verlangt von den Entwicklern, den Code auf eine bestimmte Art und Weise zu schreiben, obwohl diese vielleicht eine viel bessere Möglichkeit kennen, das Problem zu lösen. Es gibt keine einzig richtige, einfache Lösung für die Architektur. Jeder, der anderes behauptet, tut dem Softwareengineering keinen Gefallen.

Eine spezifische Implementierung einer Webanwendung wird wahrscheinlich während ihrer gesamten Lebensdauer kontinuierlich überarbeitet, ihre Architektur hingegen wird sich kaum ändern, sondern sich nur langsam entwickeln. Die Sicherheitsarchitektur hingegen ist identisch - wir brauchen Authentifizierung heute, wir brauchen Authentifizierung morgen, und wir werden sie in fünf Jahren brauchen. Treffen wir heute fundierte Entscheidungen, können wir viel Aufwand, Zeit und Geld sparen, wenn wir architekturkonforme Lösungen auswählen und wiederverwenden. So wurde beispielsweise vor einem Jahrzehnt Mehrfaktorauthentifizierung nur selten implementiert. Hätten Entwickler in einen einzigen, sicheren Identitätsprovider investiert, wie z.B. SAML Federated Identity, könnte der Identitätsprovider angepasst werden, um neue Anforderungen wie die Umsetzung des NIST 800-63 einzubeziehen, ohne die Schnittstellen der ursprünglichen Anwendung zu ändern. Wenn viele Anwendungen die gleiche Sicherheitsarchitektur und damit die gleiche Komponente nutzen, profitieren sie alle gleichzeitig von diesem Upgrade. SAML wird jedoch nicht immer die beste oder geeignetste Authentifizierungslösung bleiben - es muss möglicherweise gegen andere Lösungen ausgetauscht werden, wenn sich die Anforderungen ändern. Solche Änderungen sind entweder kompliziert und so kostspielig wie eine komplette Neuentwicklung oder ohne Sicherheitsarchitektur schlichtweg unmöglich.

In diesem Kapitel behandelt der ASVS die primären Aspekte jeder soliden Sicherheitsarchitektur: Verfügbarkeit, Vertraulichkeit, Integrität, Nichtabstreitbarkeit und Datenschutz. All diese Sicherheitsprinzipien müssen eingebaut sein und allen Anwendungen innewohnen. Es ist entscheidend, Sicherheit zeitig im Entwicklungsprozess zu integrieren. Das beginnt bei der Befähigung der Softwareentwickler mittels Checklisten für sichere Programmierung, Betreuung und Schulung. Es führt über Programmierung und Tests, dem Buildprozess, der Inbetriebnahme, der Konfiguration und dem Betrieb bis hin zu unabhängigen Tests, die sichern, dass alle Sicherheitsmaßnahmen vorhanden und funktionsfähig sind. Der letzte Schritt war früher alles, was wir als Branche getan haben, aber das reicht nicht mehr aus, wenn Entwickler mehrmals am Tag Änderungen in der Produktionsumgebung durchführen. Anwendungssicherheitsexperten müssen mit agilen Techniken Schritt halten, was bedeutet, dass sie Entwicklertools übernehmen, lernen zu programmieren und mit Entwicklern zusammenarbeiten müssen, anstatt das Projekt Monate später zu kritisieren, nachdem alle anderen bereits weitergemacht haben.

V1.1 Der sichere Softwareentwicklungszyklus

#	Beschreibung	L1	L2	L3	CWE
1.1.1	Prüfen Sie, dass der SDLC die Sicherheit in allen Entwicklungsphasen berücksichtigt. (C1)		✓	✓	
1.1.2	Prüfen Sie, dass für Designänderung oder Sprintplanung eine Bedrohungsanalyse stattfand, um Bedrohungen zu identifizieren, Gegenmaßnahmen zu planen und umzusetzen sowie passende Sicherheitstests zu planen.		✓	✓	1053

#	Beschreibung	L1	L2	L3	CWE
1.1.3	Prüfen Sie, dass alle Userstories und alle Merkmale funktionale Sicherheitsanforderungen enthalten, z.B. „Als Benutzer sollte ich mein Profil anzeigen und bearbeiten können. Ich sollte nicht in der Lage sein, das Profil eines anderen anzusehen oder zu bearbeiten“.		✓	✓	1110
1.1.4	Prüfen Sie die Dokumentation und Erläuterung aller Sicherheitsgrenzen, Komponenten und wichtigen Datenflüsse der Anwendung.		✓	✓	1059
1.1.5	Prüfen Sie die Definition und Sicherheitsanalyse der High Level Architektur der Anwendung und aller verbundenen Remoteservices. (C1)		✓	✓	1059
1.1.6	Prüfen Sie, dass Sicherheitsmaßnahmen zentralisiert, einfach, geprüft, sicher und wiederverwendbar implementiert worden sind. Dies vermeidet doppelte, fehlende, unwirksame oder unsichere Maßnahmen. (C10)		✓	✓	637
1.1.7	Prüfen Sie die Verfügbarkeit einer Checkliste für die sichere Programmierung, Sicherheitsanforderungen, eines Leitfadens oder Richtlinien für alle Entwickler und Tester.		✓	✓	637

V1.2 Architektur der Authentifizierung

Beim Entwurf der Authentifizierung spielt es keine Rolle, ob Sie über eine starke, hardwareunterstützte Mehrfaktorauthentifizierung verfügen, wenn ein Angreifer ein Konto zurücksetzen kann, indem er ein Callcenter anruft und auf allgemein bekannte Fragen antwortet. Beim Nachweis der Identität müssen alle Authentifizierungswege die gleiche Stärke haben.

#	Beschreibung	L1	L2	L3	CWE
1.2.1	Prüfen Sie die Nutzung spezifischer Betriebssystemkonten bzw. solcher mit minimalen Berechtigungen für alle Komponenten, Dienste und Server. (C3)		✓	✓	250
1.2.2	Prüfen Sie, dass die Kommunikation zwischen Anwendungskomponenten, einschließlich APIs, Middleware und Datenschichten, authentifiziert wird. Komponenten sollten die minimal notwendigen Berechtigungen haben. (C3)		✓	✓	306
1.2.3	Prüfen Sie, dass die Anwendung einen einzigen geprüften und sicheren Authentifizierungsmechanismus verwendet, der auf eine starke Authentifizierung erweitert werden kann und über ein ausreichendes Logging und Monitoring verfügt, um Einbrüche oder Missbrauch zu erkennen.		✓	✓	306
1.2.4	Prüfen Sie, dass alle Authentifizierungspfade und Identitätsmanagement-APIs eine einheitliche Stärke der Authentifizierung implementieren, so dass es keine schwächeren Alternativen pro Anwendungsrisiko gibt.		✓	✓	306

V1.3 Architektur des Sessionmanagements

Dies ist ein Platzhalter für zukünftige architektonische Anforderungen.

V1.4 Architektur der Zugriffskontrolle

#	Beschreibung	L1	L2	L3	CWE
1.4.1	Prüfen Sie, dass Zugriffskontrollen von vertrauenswürdigen Stellen, wie z.B. Accesscontrol Gateways, Servern oder serverlosen Funktionen, ausgeführt werden. Implementieren Sie Zugriffskontrollen niemals am Client.		✓	✓	602
1.4.2	[GELÖSCHT, NICHT UMSETZBAR]				

#	Beschreibung	L1	L2	L3	CWE
1.4.3	[GELÖSCHT, DUPLIKAT VON 4.1.3]				
1.4.4	Prüfen Sie, dass die Anwendung mit einer einzigen und gut erprobten Zugriffssteuerung auf geschützte Daten und Ressourcen zugreift. Alle Anfragen müssen diesen einen Weg nutzen, um Kopieren und Einfügen oder unsichere Alternativpfade zu vermeiden. (C7)		✓	✓	284
1.4.5	Prüfen Sie, dass eine attribut- oder merkmalsbasierte Zugriffskontrolle verwendet wird, die die Berechtigung des Benutzers zum Zugriff auf ein Merkmal oder Datenelement und nicht nur seine Rolle prüft. Die Berechtigungen sollten weiterhin über Rollen vergeben werden. (C7)		✓	✓	275

V1.5 Architektur des Ein- und Ausgabemanagements

Mit der Version 4.0 haben wir uns von dem Begriff „serverseitig“ als Synonym für Vertrauensgrenzen verabschiedet. Die Vertrauensgrenze spielt noch immer eine wichtige Rolle: Entscheidungen von nicht vertrauenswürdigen Clients können umgangen werden. Allerdings hat sich die Stelle, an der die Entscheidungen getroffen werden, bei heutigen Mainstreamarchitekturen dramatisch verändert. Wenn im ASVS der Begriff „vertrauenswürdige Serviceschicht“ verwendet wird, meinen wir daher jede vertrauenswürdige Stelle, unabhängig von ihrer Position, wie z.B. einen Mikroservice, eine serverlose API, serverseitig, eine vertrauenswürdige API auf einem Client, das über sichere Bootmechanismen verfügt, eine Partner- oder externe APIs und so weiter.

Die Bezeichnung „nicht vertrauenswürdiger Client“ bezieht sich auf Frontendtechnologien, wie z.B. der Darstellungsschicht. Der Begriff „Serialisierung“ soll sich nicht nur Daten, wie Felder in JSON-Strukturen, beziehen sondern ebenso auf komplexe Objekte, welche Programmlogik enthalten können.

#	Beschreibung	L1	L2	L3	CWE
1.5.1	Prüfen Sie, dass die Ein- und Ausgabeanforderungen klar definieren, wie die Daten auf der Grundlage des Typs, des Inhalts und der anwendbaren Gesetze, Vorschriften und anderen Richtlinien zu verarbeiten sind.		✓	✓	1029
1.5.2	Prüfen Sie, dass bei der Kommunikation mit nicht vertrauenswürdigen Clients keine Serialisierung verwendet wird. Ist dies nicht möglich, prüfen Sie, dass die Integrität geprüft und bei sensiblen Daten auch verschlüsselt wird, um Deserialisierungsangriffe oder Object Injection Angriffe zu verhindern.		✓	✓	502
1.5.3	Prüfen Sie, dass die Eingabevalidierung in einer vertrauenswürdigen Serviceschicht durchgesetzt wird. (C5)		✓	✓	602
1.5.4	Prüfen Sie, dass die Ausgabecodierung in der Nähe des oder durch den Interpreter erfolgt, für den sie bestimmt ist. (C4)		✓	✓	116

V1.6 Architektur kryptographischer Maßnahmen

Anwendungen müssen mit einer starken kryptographischen Architektur entworfen werden, um die Datenbestände gemäß ihrer Klassifizierung zu schützen. Alles zu verschlüsseln ist Verschwendung, nichts zu verschlüsseln ist fahrlässig. Es muss ein Gleichgewicht gefunden werden, in der Regel bei dem architektonischen oder High Level Design, den Designsprints oder den Architectural Spikes. Die sichere Implementierung kryptographischer Methoden, die nach und nach entworfen oder nachgerüstet werden, ist zwangsläufig viel teurer, als sie von Beginn an einzubauen.

Architektonische Anforderungen sind in der gesamten Codebasis integriert und lassen sich daher schwer in Unit- oder Integrationstests prüfen. Die architektonischen Anforderungen müssen in den Programmierstandards für die gesamten Programmierphase berücksichtigt werden und sollten während der Sicherheitsarchitektur, bei Peer- oder Code-Reviews oder bei Retrospektiven überprüft werden.

#	Beschreibung	L1	L2	L3	CWE
1.6.1	Prüfen Sie, dass es eine explizite Richtlinie für das Schlüsselmanagement gibt, und dass der Lebenszyklus eines kryptografischen Schlüssels konform zu einem Standard für das Schlüsselmanagement wie NIST SP 800-57 ist.		✓	✓	320
1.6.2	Prüfen Sie, dass Nutzer kryptografischer Dienste Schlüssel und andere Geheimnisse mit Hilfe von Schlüsseltresoren oder API-basierte Alternativen schützen.		✓	✓	320
1.6.3	Prüfen Sie, dass alle Schlüssel und Passwörter ersetzbar und Teil eines genau definierten Prozesses zur Neuverschlüsselung sensibler Daten sind.		✓	✓	320
1.6.4	Prüfen Sie, dass clientseitige Geheimnisse, wie symmetrische Schlüssel, Passwörter oder API-Token, architektonisch als unsicher betrachtet werden. Sie dürfen nicht zum Schutz sensibler Daten verwendet werden.		✓	✓	320

V1.7 Architektur von Fehlerbehandlung, Protokollierung und Audit

#	Beschreibung	L1	L2	L3	CWE
1.7.1	Prüfen Sie, dass im gesamten System Herangehensweise und Protokollformat einheitlich sind. (C9)		✓	✓	1009
1.7.2	Prüfen Sie, dass die Protokolle zur Analyse, Erkennung, Alarmierung und Eskalation sicher übertragen werden - vorzugsweise an ein eigenständiges System. (C9)		✓	✓	

V1.8 Architektonische Anforderungen zur Einhaltung des Datenschutzes

#	Beschreibung	L1	L2	L3	CWE
1.8.1	Prüfen Sie, dass alle sensiblen Daten identifiziert und klassifiziert werden.		✓	✓	
1.8.2	Prüfen Sie, dass für alle Schutzklassen entsprechende Anforderungen existieren, z. B. an die Vertraulichkeit, die Integrität, Aufbewahrung, Datenschutz etc. und dass diese in der Architektur angewendet werden.		✓	✓	

V1.9 Architektur der Kommunikationsverbindungen

#	Beschreibung	L1	L2	L3	CWE
1.9.1	Prüfen Sie, dass die Anwendung die Kommunikation zwischen Komponenten verschlüsselt, insbesondere wenn sich diese in verschiedenen Containern, Systemen, Standorten oder Cloudanbietern befinden. (C3)		✓	✓	319
1.9.2	Prüfen Sie, dass die Anwendungskomponenten die Authentizität beider Seiten einer Kommunikationsverbindung verifizieren, um Man-in-the-Middle-Angriffe zu verhindern. Beispielsweise sollten die Anwendungskomponenten TLS-Zertifikate und Zertifikatsketten verifizieren.		✓	✓	295

V1.10 Architektonische Anforderungen zum Schutz vor unbefugten Änderungen

#	Beschreibung	L1	L2	L3	CWE
1.10.1	Prüfen Sie, dass das Quellcodeverwaltungssystem sicherstellt, dass Check-Ins mit Issues oder Änderungstickets einhergehen. Das Quellcodeverwaltungssystem sollte über eine Zugriffskontrolle und identifizierbare Benutzer verfügen, um Änderungen nachverfolgen zu können.		✓	✓	284

V1.11 Architektur der Geschäftslogik

#	Beschreibung	L1	L2	L3	CWE
1.11.1	Prüfen Sie die Definition und Dokumentation aller Anwendungskomponenten auf die von ihnen bereitgestellten Fach- oder Sicherheitsfunktionen.		✓	✓	1059
1.11.2	Prüfen Sie, dass alle geschäftskritischen Abläufe, inkl. der Authentifizierung, des Sessionmanagements und der Zugriffssteuerung stets synchronisiert sind.		✓	✓	362
1.11.3	Prüfen Sie, dass alle geschäftskritischen Abläufe, einschließlich der Authentifizierung, des Sessionmanagements und der Zugriffssteuerung thread-sicher und sicher gegen TOCTOU Race Conditions sind.			✓	367

V1.12 Sicheres Datei Upload

#	Beschreibung	L1	L2	L3	CWE
1.12.1	[GELÖSCHT, DUPLIKAT VON 12.4.1]				
1.12.2	Prüfen Sie, dass vom Benutzer hochgeladene Dateien - sofern sie angezeigt oder von der Anwendung heruntergeladen werden müssen - entweder durch Oktett-Stream-Downloads oder von einer nicht verwandten Domäne, wie z.B. einem Cloud File Storage Bucket, bereitgestellt werden. Implementieren Sie geeignete Sicherheitsmaßnahmen für Dateiinhalte, um das Risiko von Angriffen mit Hilfe der hochgeladenen Datei zu reduzieren.		✓	✓	646

V1.13 API-Architektur

Dies ist ein Platzhalter für zukünftige architektonische Anforderungen.

V1.14 Architektonische Anforderungen an die Konfiguration

#	Beschreibung	L1	L2	L3	CWE
1.14.1	Prüfen Sie die Trennung von Komponenten unterschiedlicher Vertrauensstufen durch gut durchdachte Sicherheitsmaßnahmen, Firewallregeln, API-Gateways, Reverseproxies, cloudbasierte Sicherheitsgruppen o.ä.		✓	✓	923
1.14.2	Prüfen Sie, dass digitale Signaturen, vertrauenswürdige Verbindungen und vertrauenswürdige Downloadquellen verwendet werden, um Binärdaten auf Endgeräte zu verteilen.		✓	✓	494
1.14.3	Prüfen Sie, dass die Buildpipeline vor veralteten oder unsicheren Komponenten warnt und entsprechende Maßnahmen ergreift.		✓	✓	1104

#	Beschreibung	L1	L2	L3	CWE
1.14.4	Prüfen Sie, dass die Buildpipeline einen Schritt enthält, um die sichere Deploymentversion der Anwendung automatisch zu erstellen und zu verifizieren, insbesondere wenn die Anwendungsinfrastruktur softwarebasiert ist, wie z. B. Cloudumgebungen.		✓	✓	
1.14.5	Prüfen Sie, dass Anwendungen auf der Netzwerkebene voneinander separiert sind, z.B. per Sandbox oder Container, um Angreifer auszubremsen und davon abzuhalten, andere Anwendungen anzugreifen, insbesondere wenn sie sensible Aktionen wie eine Deserialisierung durchführen. (C5)		✓	✓	265
1.14.6	Prüfen Sie, dass die Anwendung keine nicht unterstützten, unsicheren oder veralteten clientseitigen Technologien wie NSAPI-Plugins, Flash, Shockwave, ActiveX, Silverlight, NACL oder clientseitige Java-Applets verwendet.		✓	✓	477

Referenzen

Weitere Informationen finden Sie unter:

- OWASP Threat Modeling Cheat Sheet
- OWASP Attack Surface Analysis Cheat Sheet
- OWASP Threat modeling
- OWASP Software Assurance Maturity Model Project
- Microsoft SDL
- NIST SP 800-57

V2: Authentifizierung

Ziel

Authentifizierung ist die Bestätigung einer Eigenschaft, wie z.B. der Identität einer Person oder eines Gerätes. Sie muss u.a. widerstandsfähig gegen Nachahmer sein und Passwörter sicher übertragen. Als der ASVS zum ersten Mal freigegeben wurde, waren Benutzername und Passwort die gebräuchlichste Form der Authentifizierung außerhalb von Hochsicherheitssystemen. Die Mehrfaktorauthentifizierung (MFA) wurde in Sicherheitskreisen allgemein akzeptiert, aber anderswo nur selten verlangt. Mit zunehmenden Passwortverletzungen wurde die Idee, dass Benutzernamen vertraulich und Passwörter unbekannt sind, als Sicherheitsmaßnahme unhaltbar. NIST 800-63 betrachtet beispielsweise Benutzernamen und wissensbasierte Authentifizierung als öffentliche Informationen, SMS- und E-Mail-Benachrichtigungen als „eingeschränkte" Authentifikatoren (restricted authenticators) und Passwörter als praktisch unsicher. Diese Realität macht wissensbasierte Authentifikatoren, SMS- und E-Mail-Wiederherstellung, Passwortverlauf, Komplexität und Passwortwechselstrategien nutzlos. Diese Maßnahmen waren schon immer wenig hilfreich und zwangen die Benutzer oft dazu, sich alle paar Monate schwache Passwörter auszudenken. Mit der Veröffentlichung von über 5 Milliarden Kombinationen von Benutzernamen und Passwörtern ist es an der Zeit, sich weiter zu entwickeln.

Daher haben sich die Kapitel „Authentifizierung" und „Session Management" am meisten verändert. Das Ausrichten an einem effektiven, evidenzbasierten Vorbild wird für viele eine Herausforderung sein, das ist völlig in Ordnung. Wir müssen jetzt den Übergang zu einer Zukunft ohne Passwort einleiten.

NIST 800-63 - Ein moderner, evidenzbasierter Authentifizierungsstandard

NIST 800-63b ist ein moderner, evidenzbasierter Standard. Der Standard ist für alle Organisationen auf der ganzen Welt hilfreich, besonders relevant jedoch für US-Behörden und Organisationen, die mit ihnen in Verbindung stehen.

Wenn man die Authentifizierung mit Benutzername und Passwort gewohnt ist, kann die Terminologie von NIST 800-63 kann anfangs etwas verwirrend sein. Fortschritte in der modernen Authentifizierung sind notwendig, daher müssen wir eine Terminologie einführen, die in Zukunft allgemein üblich sein wird. Wir verstehen aber auch die Verständnisschwierigkeiten, die sich ergeben, bis alle sich auf diese neuen Begriffe eingestellt haben. Wir haben viele Anforderungen des NIST umformuliert, um die Absicht der Anforderung und nicht die Buchstaben der Anforderung zu erfüllen. Zum Beispiel verwendet der ASVS den Begriff „Passwort", wenn das NIST von einem „gespeicherten Geheimnis" spricht.

Die Kapitel V2: Authentifizierung, V3: Session Management und in geringerem Maße auch V4: Zugriffskontrollen wurden so angepasst, dass sie eine standardkonforme Teilmenge der Maßnahmen gemäß NIST 800-63b sind, die sich auf allgemeine Bedrohungen und häufig ausgenutzte Authentifizierungsschwachstellen konzentrieren. Muss NIST 800-63 vollständig erfüllt werden, so nutzen Sie bitte NIST 800-63 direkt.

Auswahl einer geeigneten NIST AAL-Stufe

Bei der Erstellung des ASVS 4.0 haben wir ASVS L1 den Anforderungen von NIST AAL1, L2 zu AAL2 und L3 zu AAL3 zugeordnet. Dabei ist zu beachten, dass hier die ASVS-Stufe 1 nicht mehr die Voraussetzung für alle anderen Stufen ist: Soll die Anwendung beispielsweise NIST AAL3 erfüllen, so muss in den Abschnitten V2 und V3 Session Management die Stufe 3 - und nur die Stufe 3 - gewählt werden. Die Auswahl des NIST AAL sollte gemäß den Richtlinien von NIST 800-63b erfolgen, wie sie in NIST800-63b, Abschnitt 6.2 unter *Selecting AAL* beschrieben sind.

Legende

Anwendungen können immer die Anforderungen der aktuellen Stufe überschreiten, insbesondere wenn eine moderne Authentifizierung auf der Roadmap einer Anwendung steht. In den Vorgängerversionen erforderte der ASVS obligatorisch eine Mehrfaktorauthentifizierung (MFA). NIST tut dies nicht. Daher haben wir in diesem Kapitel eine Option eingeführt, die anzeigt, wenn der ASVS eine Eigenschaft empfiehlt. Dabei gilt:

Symbol	Beschreibung
	Nicht erforderlich
o	Empfohlen
✓	Erforderlich

V2.1 Passwortsicherheit

Passwörter, in NIST 800-63 als „gespeicherte Geheimnisse“ bezeichnet, umfassen Passwörter, PINs, Entsperrmuster, die Auswahl des richtigen Bildelements sowie Passphrasen. Sie werden im Allgemeinen als „Wissen“ betrachtet und oft als Einfaktor-Authentifikatoren verwendet. Die weitere Verwendung der Einfaktor-Authentifizierung steht vor erheblichen Herausforderungen, z.B.: Milliarden von öffentlich verfügbaren Kombinationen von Benutzername und Passwort, Standardpasswörtern, schwachen Passwörtern, Rainbowtables sowie Wörterbücher der häufigsten Passwörter.

Anwendungen sollten die Benutzer nachdrücklich dazu ermutigen, sich mittels Mehrfaktor-Authentifizierung anzumelden und ihnen erlauben, Token, wie z.B. FIDO- oder U2F-Token, die sie bereits besitzen, einzusetzen oder sich mit einem Credential Service Provider (CSP) zu verbinden, der Mehrfaktor-Authentifizierung anbietet. CSPs bieten ihren Benutzern eine föderierte Identität an. Benutzer verfügen oft über mehrere Identitäten bei mehreren CSPs, z. B. eine Unternehmensidentität bei Azure AD, Okta, Ping Identity oder Google und eine Verbraucheridentität bei Facebook, Twitter, Google oder WeChat, um nur einige gängige Alternativen zu nennen. Diese Liste soll keine Bestätigung dieser Unternehmen oder Dienste sein, sie soll lediglich Entwickler dazu anspornen, die Realität zu berücksichtigen, dass viele Benutzer viele etablierte Identitäten haben. Organisationen sollten entsprechend ihrem Risikoprofil die Einbindung der Benutzeridentität in Betracht ziehen, die sich aus der Stärke der Identitätsprüfung des CSP ergibt. So ist es beispielsweise unwahrscheinlich, dass eine Regierungsorganisation eine Social-Media-Identität als Login für sensible Systeme akzeptiert, da es leicht ist, falsche Identitäten zu erstellen. Ein Unternehmen für Handyspiele muss sich für die Erweiterung seiner aktiven Spielerbasis möglicherweise an eine große Social-Media-Plattformen anbinden.

#	Beschreibung	L1	L2	L3	CWE	NIST §
2.1.1	Prüfen Sie, dass Benutzerpasswörter mindestens 12 Zeichen lang sind, nachdem zusammenhängende Leerzeichen gekürzt wurden. (C6)	✓	✓	✓	521	5.1.1.2
2.1.2	Prüfen Sie, dass Passwörter mit 64 oder mehr Zeichen erlaubt sind, jedoch nicht mehr als 128 Zeichen. (C6)	✓	✓	✓	521	5.1.1.2
2.1.3	Prüfen Sie, dass Passwörter nicht gekürzt werden. Mehrere aufeinanderfolgende Leerzeichen können zu einem zusammengefasst werden. (C6)	✓	✓	✓	521	5.1.1.2
2.1.4	Prüfen Sie, ob alle druckbaren Unicode-Zeichen, auch Leerzeichen oder Emojis, in Passwörtern zulässig sind.	✓	✓	✓	521	5.1.1.2
2.1.5	Prüfen Sie, dass Benutzer ihr Passwort ändern können.	✓	✓	✓	620	5.1.1.2
2.1.6	Prüfen Sie, dass die Passwortänderungsfunktion das bisherige sowie das neue Kennwort des Benutzers erfordert.	✓	✓	✓	620	5.1.1.2

#	Beschreibung	L1	L2	L3	CWE	NIST §
2.1.7	Prüfen Sie, dass die bei der Kontoregistrierung, beim Login und bei der Passwortänderung übermittelten Passwörter mit einen Satz verletzter Passwörtern verglichen werden, und zwar entweder lokal (z. B. mit den 1.000 oder 10.000 häufigsten Passwörtern, die mit der Passwortrichtlinie des Systems übereinstimmen) oder mit Hilfe einer externen API. Bei Verwendung einer API muss sichergestellt werden, dass das Klartextpasswort nicht gesendet oder anderweitig offengelegt wird. Wird das Passwort offengelegt, muss die Anwendung den Benutzer auffordern, ein neues Passwort festzulegen. (C6)	✓	✓	✓	521	5.1.1.2
2.1.8	Prüfen Sie, dass ein Maß für die Passwortstärke bereitgestellt wird, damit Benutzer ein stärkeres Passwort erstellen können.	✓	✓	✓	521	5.1.1.2
2.1.9	Prüfen Sie, dass es keine Regeln für die Zusammenstellung der Passwörter gibt, welche die Art der zulässigen Zeichen einschränken. Die Verwendung bestimmter Zeichen, wie Groß- oder Kleinschreibung, Zahlen oder Sonderzeichen sollte nicht verlangt werden. (C6)	✓	✓	✓	521	5.1.1.2
2.1.10	Prüfen Sie, dass weder periodischer Passwortwechsel notwendig ist noch eine Passworthistorie gespeichert wird.	✓	✓	✓	263	5.1.1.2
2.1.11	Prüfen Sie, dass die „Einfügen“-Funktion, Passworthilfen der Browser und externe Passwortmanager zugelassen sind.	✓	✓	✓	521	5.1.1.2
2.1.12	Prüfen Sie, dass der Benutzer wählen kann, entweder das gesamte Passwort vorübergehend angezeigt zu bekommen oder das letzte eingetippte Zeichen des Passwortes angezeigt zu bekommen.	✓	✓	✓	521	5.1.1.2

Hinweis: Die Möglichkeit des Benutzers, sein Passwort angezeigt zu bekommen oder vorübergehend das letzte Zeichen zu sehen, soll die Eingabefreundlichkeit der Anmeldedaten verbessern, insbesondere bei der Verwendung längerer Passwörter, Passphrasen und Passwortmanager. Weiterhin sollen diese Anforderungen verhindern, Organisationen unnötigerweise dazu zwingen, dieses moderne benutzerfreundliche Sicherheitskonzept einiger Betriebssysteme zu deaktivieren.

V2.2 Allgemeine Sicherheitsanforderungen an den Authentifikator

Die Agilität des Authentifikators ist für zukunftssichere Anwendungen unerlässlich. Überarbeiten Sie die Application Verifier um zusätzliche Authentifikatoren nach Benutzerpräferenz zuzulassen. Entfernen Sie veraltete oder unsichere Authentifikatoren.

NIST betrachtet E-Mail und SMS als „eingeschränkt zur Authentifikation tauglich”, und es ist wahrscheinlich, dass sie in der Zukunft aus dem NIST 800-63 und damit dem ASVS entfernt werden. Anwendungen sollten so geplant werden, dass sie ohne E-Mail oder SMS auskommen.

#	Beschreibung	L1	L2	L3	CWE	NIST §
2.2.1	Prüfen Sie, dass Maßnahmen gegen automatische Angriffe, wie das Ausprobieren von Passworten oder das Aussperren von Benutzern, wirksam umgesetzt sind. Dazu gehören das Blockieren der am häufigsten verwendeten Passwörter, Soft-Lockouts, die Begrenzung der Anzahl von Anmeldungen, CAPTCHA, wachsende Verzögerungen zwischen den Fehlversuchen, IP-Adressbeschränkungen oder risikobasierte Einschränkungen wie Standort, erste Anmeldung auf einem Gerät, kürzliche Versuche, das Konto zu entsperren oder Ähnliches. Prüfen Sie, dass nicht mehr als 100 Fehlversuche pro Stunde bei einem einzelnen Konto möglich sind.	✓	✓	✓	307	5.2.2 / 5.1.1.2 / 5.1.4.2 / 5.1.5.2
2.2.2	Prüfen Sie, dass der Einsatz schwacher Authentifikationsmethoden, wie SMS und E-Mail, auf die sekundäre Verifizierung und Transaktionsgenehmigung beschränkt ist und nicht als Ersatz für sicherere Authentifizierungsmethoden dient. Prüfen Sie, dass stärkere Methoden vor schwachen Methoden eingesetzt werden, dass sich die Benutzer der Risiken bewusst sind oder dass geeignete Maßnahmen zur Begrenzung des Risikos getroffen werden.	✓	✓	✓	304	5.2.10
2.2.3	Prüfen Sie, dass die Benutzer sichere Benachrichtigungen nach Aktualisierungen der Authentifizierungsdetails, wie z. B. das Zurücksetzen von Anmeldedaten, E-Mail- oder Adressänderungen, Anmeldung von unbekannten oder risikobehafteten Orten erhalten. Die Verwendung von Push-Benachrichtigungen - anstelle von SMS oder E-Mail - ist vorzuziehen. Bei fehlenden Push-Benachrichtigungen sind SMS oder E-Mail jedoch akzeptabel, solange in der Benachrichtigung keine sensiblen Informationen offengelegt werden.	✓	✓	✓	620	
2.2.4	Prüfen Sie die Resistenz gegen Phishing durch Authentifizierung mittels Mehrfaktor-Authentifizierung, Public Key Kryptographie, Chipkarten und Push-Nachrichten, auf höheren AAL-Ebenen: clientseitige Zertifikate.			✓	308	5.2.5
2.2.5	Prüfen Sie, dass der CSP und die nutzende Anwendung über zweiseitig authentifiziertes TLS kommunizieren.			✓	319	5.2.6
2.2.6	Prüfen Sie, dass Authentifikationsdaten nicht wieder eingespielt werden können. Dies kann z.B. mit One Time Password (OTP) Generatoren, Chipkarten o.ä. verhindert werden.			✓	308	5.2.8
2.2.7	Prüfen Sie, dass eine Authentifikation nicht unbeabsichtigt stattfinden kann. Verlangen Sie die Eingabe eines OTP-Tokens oder eine vom Benutzer initiierte Aktion, wie z.B. einen Tastendruck auf einem FIDO-Hardwaretoken.			✓	308	5.2.9

V2.3 Lebenszyklus des Authentifikators

Authentifikatoren sind Passwörter, Softtoken, Hardwaretoken und biometrische Geräte. Ihr Lebenszyklus ist für die Sicherheit einer Anwendung entscheidend: Wenn sich jemand auf einem Konto ohne Identitätsnachweis selbst registrieren kann, ist die Identitätsbehauptung wenig vertrauenswürdig. Für Social-Media-Sites wie Reddit ist das völlig in Ordnung. Bei Bankinganwendungen hingegen ist ein größeres Augenmerk auf die Registrierung, die Anmeldedaten bzw. -geräten entscheidend für die Sicherheit der Anwendung.

Hinweis: Die NIST hat die Regulierung von Passwörtern geändert. Sie dürfen keine maximale Lebensdauer haben und sollten keinem routinemäßigen Wechsel mehr unterliegen. Passwörter müssen nur noch beim Verdacht auf Offenlegung ersetzt werden.

#	Beschreibung	L1	L2	L3	CWE	NIST §
2.3.1	Vom System generierte Anfangspasswörter oder Aktivierungscodes SOLLTEN sicher zufällig generiert werden. Sie SOLLTEN mindestens 6 Zeichen lang sein und KÖNNEN Buchstaben und Zahlen enthalten. Sie MÜSSEN nach einer kurzen Zeitspanne ablaufen. Diese Initialpasswörter dürfen nicht zum dauerhaften Passwort werden.	✓	✓	✓	330	5.1.1.2 / A.3
2.3.2	Prüfen Sie, dass die Registrierung und die Verwendung von vom Teilnehmer bereitgestellten Authentifizierungsgeräten unterstützt werden, wie z. B. U2F- oder FIDO-Token.		✓	✓	308	6.1.3
2.3.3	Prüfen Sie, dass die Aufforderung zur Erneuerung zeitgebundener Authentifikatoren rechtzeitig gesendet werden.		✓	✓	287	6.1.4

V2.4 Speicherung der Anmeldedaten

Architekten und Entwickler sollten sich bei der Erstellung oder dem Refactoring von Software an diesen Abschnitt halten. Dieser Abschnitt kann nur durch Codereview oder durch sichere Unit- oder Integrationstests vollständig verifiziert werden. Penetrationstests können diese Probleme nicht nachweisen, daher sind die Maßnahmen nicht mit L1 gekennzeichnet. Dieser Abschnitt ist jedoch von entscheidender Bedeutung für die Sicherheit von Anmeldedaten, falls diese gestohlen werden. Wenn Sie also den ASVS als Architektur- oder Programmierrichtlinie oder als Checkliste zum Codereview nutzen, setzen Sie diese Maßnahmen bitte wieder auf L1.

Die Liste der anerkannten Einwegfunktionen für die Schlüsselableitung wird in NIST 800-63 B, Abschnitt 5.1.1.2 und in BSI Kryptographische Verfahren: Empfehlungen und Schlüssellängen (2018) detailliert aufgeführt. Anstelle dieser Auswahlmöglichkeiten können die neuesten nationalen oder regionalen Algorithmen und Schlüssellängenstandards gewählt werden.

#	Beschreibung	L1	L2	L3	CWE	NIST §
2.4.1	Prüfen Sie, dass die Passwörter in einer Form gespeichert werden, die immun gegen Offlineangriffe ist. Passwörter MÜSSEN mit einem Salt versehen werden. Der Passworthash muss mit Hilfe einer sicheren Funktion zur Schlüsselberechnung oder einer Passwort-Hashfunktion berechnet werden. Die Funktionen zur Schlüsselberechnung und zum Passwort-Hashing nehmen ein Passwort, einen Salt und einen Kostenfaktor als Eingabewerte. (C6)		✓	✓	916	5.1.1.2
2.4.2	Prüfen Sie, dass das Salt mindestens 32 Bit lang ist und zufällig gewählt wird, um Saltwertkollisionen zwischen gespeicherten Hashes zu minimieren. Für jede Anmeldeinformation (Credential) MUSS ein eindeutiger Saltwert und der daraus resultierende Hash gespeichert werden. (C6)		✓	✓	916	5.1.1.2
2.4.3	Prüfen Sie, dass bei Verwendung von PBKDF2 der Iterationszähler so groß sein SOLLTE, wie es die Leistung des Verifikationsservers zulässt, normalerweise mindestens 100.000 Iterationen. (C6)		✓	✓	916	5.1.1.2
2.4.4	Prüfen Sie, dass bei Verwendung von bcrypt der Arbeitsfaktor so groß sein SOLLTE, wie es die Leistung des Verifikationsservers erlaubt, jedoch mindestens 10. (C6)		✓	✓	916	5.1.1.2

#	Beschreibung	L1	L2	L3	CWE	NIST §
2.4.5	Prüfen Sie, dass eine zusätzliche Iteration einer Funktion zur Schlüsselberechnung durchgeführt wird. Dabei ist ein Saltwert zu verwenden, der nur dem Verifizierer bekannt ist. Generieren Sie den Saltwert mit einem zugelassenen Zufallsgenerator [SP 800-90Ar1]. Stellen Sie die in der letzten Revision von SP 800-131A angegebene Mindestsicherheitsstärke sicher. Der geheime Saltwert MUSS getrennt von den gehashten Passwörtern gespeichert werden, z.B. in einem speziellen Gerät wie einem HSM.		✓	✓	916	5.1.1.2

Wo US-Normen erwähnt werden, kann bei Bedarf anstelle der US-Norm oder zusätzlich zu dieser eine lokale Norm verwendet werden. Ein Teil der Inhalte des SP 800-131A wird von der TR-02102-1 "Kryptographische Verfahren: Empfehlungen und Schlüssellängen" des BSI abgedeckt.

V2.5 Wiederherstellung von Anmeldedaten

#	Beschreibung	L1	L2	L3	CWE	NIST §
2.5.1	Prüfen Sie, dass ein vom System generiertes Initial- oder Wiederherstellungsgeheimnis nicht im Klartext an den Benutzer gesendet wird. (C6)	✓	✓	✓	640	5.1.1.2
2.5.2	Prüfen Sie, dass keine Hinweise auf Passwörter oder wissensbasierte Authentifizierung, z.B. „geheime Fragen“ vorliegen.	✓	✓	✓	640	5.1.1.2
2.5.3	Prüfen Sie, dass die Wiederherstellung von Anmeldedaten das aktuelle Kennwort nicht preisgibt. (C6)	✓	✓	✓	640	5.1.1.2
2.5.4	Prüfen Sie, dass Gemeinschafts- oder Standardkonten, z.B. „root“, „admin“, „Gast“ oder „sa“ deaktiviert oder gelöscht sind.	✓	✓	✓	16	5.1.1.2 / A.3
2.5.5	Prüfen Sie, dass der Benutzer informiert wird, wenn ein Authentifizierungsfaktor geändert oder ersetzt wird.	✓	✓	✓	304	6.1.2.3
2.5.6	Prüfen Sie, dass der Prozess zur Wiederherstellung, z.B. für vergessene Passwörter, einen sicheren Kanal, z. B. TOTP oder andere Softtoken, Mobile Push oder andere Offlinekanäle, verwendet. (C6)	✓	✓	✓	640	5.1.1.2
2.5.7	Prüfen Sie, dass der Identitätsnachweis bei Verlust von OTP- oder Mehrfaktor-Token auf derselben Ebene wie bei der Registrierung durchgeführt wird.		✓	✓	308	6.1.2.3

V2.6 Verifizierung von TAN-Listen

Vorgenerierte Listen von Geheimcodes, z.B. Transaktionsnummern (TAN), Wiederherstellungscodes für soziale Medien oder eine andere Reihe von Zufallswerten. Diese werden sicher an die Benutzer versandt. Diese Geheimcodes werden einmal verwendet. Wenn alle verwendet wurden, wird die Liste entsorgt. Diese Art von Authentifikator gilt als „etwas, das Sie haben“ bzw. Besitz.

#	Beschreibung	L1	L2	L3	CWE	NIST §
2.6.1	Prüfen Sie, dass die Geheimcodes nur einmal verwendet werden können.		✓	✓	308	5.1.2.2

#	Beschreibung	L1	L2	L3	CWE	NIST §
2.6.2	Prüfen Sie, ob die Geheimcodes eine ausreichende Zufälligkeit aufweisen (112 Bit Entropie). Falls weniger als 112 Bit Entropie vorhanden sind, sind ein einzigartiger und zufälliger 32 Bit Salt und eine zugelassene Hashfunktion zu nutzen.		✓	✓	330	5.1.2.2
2.6.3	Prüfen Sie, dass Geheimcodes gegen Offlineangriffe, wie z.B. vorhersehbare Werte, immun sind.		✓	✓	310	5.1.2.2

V2.7 Out-of-Band Verifizierer

In der Vergangenheit wäre ein üblicher Out-of-Band Verifizierer eine E-Mail oder SMS mit einem Link zum Zurücksetzen des Passworts gewesen. Angreifer nutzen diesen schwachen Mechanismus, um Konten, die sie noch nicht kontrollieren, zurückzusetzen, z. B. indem sie das E-Mail-Konto einer Person übernehmen und Passwörter neu anfordern. Es gibt bessere Möglichkeiten, die Out-of-Band Verifizierung zu handhaben.

Sichere Out-of-Band Authentifikatoren sind physische Geräte, die mit dem Verifizierer über einen sicheren Zweitkanal kommunizieren können. Das sind z.B. Push-Nachrichten an mobile Geräte. Diese Art von Authentifikator gilt als „etwas, das Sie haben“ bzw. Besitz. Will sich ein Benutzer authentifizieren, sendet die verifizierende Anwendung eine Nachricht an den Out-of-Band Authentifikator. Die Nachricht enthält einen Authentifizierungscode, z.B. eine zufällige sechsstellige Zahl oder eine Abfrage der Genehmigung. Die verifizierende Anwendung wartet auf den Empfang des Authentifizierungscodes über den Primärkanal und vergleicht den Hash des empfangenen Wertes mit dem Hash des ursprünglichen Authentifizierungscodes. Wenn sie übereinstimmen, kann der Out-of-Band Verifizierer davon ausgehen, dass der sich der Benutzer authentifiziert hat. Der Prozess kann direkt oder über einen Dienst Dritter ablaufen.

Der ASVS geht davon aus, dass nur wenige Entwickler neue Out-of-Band Authentifizierer wie Push-Nachrichten entwickeln werden, und daher gelten die folgenden Maßnahmen des ASVS für Verifizierer, wie die Authentifizierungs-API, Anwendungen und Single-Sign-On Implementierungen. Wenn Sie einen neuen Out-of-Band Authentifikator entwickeln, beachten Sie bitte NIST 800-63B § 5.1.3.1.

Unsichere Out-of-Band Authentifizierer wie E-Mail und VOIP sind nicht zulässig. PSTN- oder SMS-Authentifizierung ist derzeit durch NIST als eingeschränkt geeignet eingeschätzt und sollten durch Push-Nachrichten oder ähnliche ersetzt werden. Müssen Sie Telefon- oder SMS zur Out-of-Band Authentifizierung verwenden, lesen Sie bitte NIST 800-63B § 5.1.3.3.

#	Beschreibung	L1	L2	L3	CWE	NIST §
2.7.1	Prüfen Sie, dass Out-of-Band Authentifikatoren, die NIST „restricted“ sind, wie z.B. SMS, nicht standardmäßig angeboten werden und dass stärkere Alternativen wie Push-Nachrichten zuerst angeboten werden.	✓	✓	✓	287	5.1.3.2
2.7.2	Prüfen Sie, dass der Out-of-Band Verifizierer bei Out-of-Band Authentifizierungsanforderungen, -Codes oder -Tokens nach 10 Minuten abläuft.	✓	✓	✓	287	5.1.3.2
2.7.3	Prüfen Sie, dass Authentifizierungsanfragen, -Codes oder -Token an den Out-of-Band Verifizierer nur einmal und nur für die ursprüngliche Authentifizierungsanfrage verwendbar sind.	✓	✓	✓	287	5.1.3.2
2.7.4	Prüfen Sie, dass der Out-of-Band Authentifizierer und der Verifizierer über einen sicheren, unabhängigen Kanal kommunizieren.	✓	✓	✓	523	5.1.3.2
2.7.5	Prüfen Sie, dass der Out-of-Band Verifizierer nur eine gehashte Version des Authentifizierungscodes speichert.		✓	✓	256	5.1.3.2

#	Beschreibung	L1	L2	L3	CWE	NIST §
2.7.6	Prüfen Sie, dass der initiale Authentifizierungscode von einem sicheren Zufallszahlengenerator erzeugt wird, der mindestens 20 Bit Entropie enthält. Normalerweise ist eine sechsstellige Zufallszahl ausreichend.		✓	✓	310	5.1.3.2

V2.8 Ein- oder Mehrfaktor-Einwegverifizierer

Einmal-Passwörter (OTPs) sind physische oder Softtoken, die ein sich ständig veränderndes pseudozufälliges Passwort bereitstellen. Diese Geräte erschweren Imitiationsangriffe, wie Phishing, verhindern sie aber nicht. Diese Art von Authentifikator gilt als „etwas, das Sie haben“ bzw. Besitz. Mehrfaktor-Token sind ähnlich wie Einmalpasswörter erfordern jedoch zusätzlich einen gültigen PIN-Code, biometrische Entsperrung, USB-Stick oder NFC-Kontakt o.ä., der eingegeben werden muss, um den endgültigen OTP zu erstellen.

#	Beschreibung	L1	L2	L3	CWE	NIST §
2.8.1	Prüfen Sie, dass zeitbasierte OTPs eine definierte Lebensdauer haben, bevor sie ablaufen.	✓	✓	✓	613	5.1.4.2 / 5.1.5.2
2.8.2	Prüfen Sie, dass die symmetrischen Schlüssel, die zur Prüfung der eingegebenen OTPs verwendet werden, sicher geschützt sind, z.B. durch Verwendung eines HSM oder der sicheren Schlüsselspeicherung des Betriebssystems.		✓	✓	320	5.1.4.2 / 5.1.5.2
2.8.3	Prüfen Sie, dass anerkannte kryptografische Algorithmen bei der Generierung, dem Seeding und der Verifizierung verwendet werden.		✓	✓	326	5.1.4.2 / 5.1.5.2
2.8.4	Prüfen Sie, dass das zeitbasierte OTP nur einmal innerhalb des Gültigkeitszeitraums verwendet werden können.		✓	✓	287	5.1.4.2 / 5.1.5.2
2.8.5	Prüfen Sie, dass ein zeitbasiertes Mehrfaktor-OTP, das während der Gültigkeitsdauer wiederverwendet wird, protokolliert und mit sicheren Benachrichtigungen an den Inhaber des Geräts abgelehnt wird.		✓	✓	287	5.1.5.2
2.8.6	Prüfen Sie, ob physische OTP-Generatoren im Falle von Diebstahl oder Verlust gesperrt werden können. Stellen Sie sicher, dass der Widerruf sofort für alle eingeloggten Sitzungen, unabhängig vom Standort, wirksam ist.		✓	✓	613	5.2.1
2.8.7	Prüfen Sie, dass biometrische Authentifikatoren nur als sekundäre Faktoren in Verbindung mit etwas, das Sie haben oder etwas, das Sie wissen, verwendet werden dürfen.		o	✓	308	5.2.3

V2.9 Kryptografische Software und Geräte im Authentifizierungsprozess

Kryptographische Geräte sind Chipkarten oder USB-Sticks, die der Benutzer an den Computer anschließen muss, um die Authentifizierung abzuschließen. Der Verifizierer sendet einen Zufallswert (Nonce) an das kryptographische Gerät oder die Software. Dort wird eine Antwort auf der Grundlage eines sicher gespeicherten kryptographischen Schlüssels berechnet. Die Anforderungen für kryptographische Geräte und Software sind für Ein- und Mehrfaktorauthentifizierung gleich, da die Verifizierung der berechneten Antwort den Besitz des Authentisierungsfaktors nachweist.

#	Beschreibung	L1	L2	L3	CWE	NIST §
2.9.1	Prüfen Sie, dass die bei der Authentifizierung verwendeten kryptografischen Schlüssel sicher gespeichert und gegen Offenlegung geschützt sind, z. B. durch Verwendung eines Trusted Platform Module (TPM) oder eines Hardware Security Modules (HSM) oder eines Betriebssystemdienstes.		✓	✓	320	5.1.7.2
2.9.2	Prüfen Sie, dass der Zufallswert mindestens 64 Bit lang ist und statistisch einmalig oder für die Lebensdauer des kryptografischen Geräts einmalig ist.		✓	✓	330	5.1.7.2
2.9.3	Prüfen Sie, dass anerkannte kryptografische Algorithmen bei allen kryptographischen Operationen verwendet werden.		✓	✓	327	5.1.7.2

V2.10 Service-Authentifizierung

Dieser Abschnitt ist kann nicht mittels Penetrationstest getestet werden und hat daher also keine L1-Anforderungen. Wenn sie jedoch in einer Architektur-, Programmier- oder Codereview verwendet wird, gehen Sie bitte davon aus, dass Software, ebenso wie Java Key Store, die Mindestanforderung von L1 hat. Das Speichern von Geheimnissen im Klartext ist unter keinen Umständen zulässig.

#	Beschreibung	L1	L2	L3	CWE	NIST §
2.10.1	Prüfen Sie, dass Intra-Service-Geheimnisse nicht auf unveränderlichen Credentials, wie Passwörtern, API-Schlüsseln oder gemeinsam genutzten privilegierten Konten beruhen.		OS assisted	HSM	287	5.1.1.1
2.10.2	Prüfen Sie, dass Servicekonten, die zur Anmeldung genutzt werden, keine Standardpasswörter, wie root / root oder admin / admin, die häufig voreingestellt sind, nutzen.		OS assisted	HSM	255	5.1.1.1
2.10.3	Prüfen Sie, dass Passworthashwerte mit ausreichendem Schutz gespeichert werden, um Offlineangriffe, zu verhindern.		OS assisted	HSM	522	5.1.1.1
2.10.4	Prüfen Sie, dass Passwörter, Zugänge zu Datenbanken o.a. Systemen, Seeds, interne Geheimnisse sowie API-Schlüssel sicher verwaltet werden. Sie dürfen nicht in den Quellcode aufgenommen bzw. in Quellcoderepositories gespeichert werden. Eine solche Speicherung muss Offline-Angriffen widerstehen. Für die Passwortspeicherung wird die Verwendung eines sicheren Softwareschlüsselspeichers (L1), eines TPM oder eines HSM (L3) empfohlen.		OS assisted	HSM	798	

Zusätzliche Anforderungen der US-Behörden

US-Behörden müssen NIST 800-63 zwingend erfüllen. Der ASVS konzentriert sich auf die 80 % der Maßnahmen, die für fast alle Anwendungen gelten. In diesem Sinne ist der ASVS eine Teilmenge von NIST 800-63, besonders für die IAL1/2- und AAL1/2-Klassifikationen. Er ist jedoch nicht umfassend genug, insbesondere für die IAL3/AAL3-Klassifikationen. Wenn Sie Software für die US-Regierungsbehörden entwickeln, müssen Sie NIST 800-63 in seiner Gesamtheit überprüfen und umzusetzen.

Referenzen

Weitere Informationen finden Sie unter:

- NIST 800-63 - Digital Identity Guidelines

- NIST 800-63 A - Enrollment and Identity Proofing
- NIST 800-63 B - Authentication and Lifecycle Management
- NIST 800-63 C - Federation and Assertions
- NIST 800-63 FAQ
- OWASP Testing Guide 4.0: Testing for Authentication
- OWASP Cheat Sheet - Password storage
- OWASP Cheat Sheet - Forgot password
- OWASP Cheat Sheet - Choosing and using security questions

V3 Sessionmanagement

Ziel

Das Sessionmanagement ist die Kernkomponente jeder webbasierten Anwendung oder zustandsbezogenen API, mit dem sie den Zustand für einen Benutzer oder ein Gerät, das mit ihr kommuniziert, steuert und aufrechterhält. Es ist entscheidend für die Unterscheidung verschiedener Benutzer oder Geräte. Es ändert ein zustandsloses Protokoll in ein zustandsbehaftetes. Prüfen Sie, dass eine verifizierte Anwendung die folgenden Anforderungen an das High Level Session Management erfüllt:

- Sessions sind für jede Person einzigartig und können nicht erraten oder geteilt werden.
- Sessions werden ungültig, wenn sie nicht mehr benötigt werden.
- Sessions laufen ab, wenn sie eine bestimmte Zeit nicht aktiv sind.

Wie bereits erwähnt, wurden diese Anforderungen so angepasst, dass sie eine konforme Teilmenge ausgewählter NIST 800-63b-Maßnahmen darstellen, die sich auf gemeinsame Bedrohungen und häufig genutzte Authentifizierungsschwachstellen konzentrieren. Frühere Verifizierungsanforderungen wurden entfernt oder in den meisten Fällen so angepasst, dass sie sich stark an der Absicht der verbindlichen Anforderungen des NIST 800-63b orientieren.

V3.1 Grundlegende Sicherheit des Sessionmanagements

#	Beschreibung	L1	L2	L3	CWE	NIST §
3.1.1	Prüfen Sie, dass die Anwendung niemals Sessiontoken in URL-Parametern oder Fehlermeldungen offenbart.	✓	✓	✓	598	

V3.2 Session Binding

#	Beschreibung	L1	L2	L3	CWE	NIST §
3.2.1	Prüfen Sie, ob die Anwendung bei der Authentifizierung eines Benutzers ein neues Sessiontoken generiert. (C6)	✓	✓	✓	384	7.1
3.2.2	Prüfen Sie, dass Sessiontoken mindestens 64 Bit Entropie aufweisen. (C6)	✓	✓	✓	331	7.1
3.2.3	Prüfen Sie, dass die Anwendung Sessiontoken im Browser nur mit sicheren Methoden wie z.B. gesicherten Cookies (siehe Abschnitt 3.4) oder den HTML 5-Methoden speichert.	✓	✓	✓	539	7.1
3.2.4	Prüfen Sie, dass die Sessiontoken mit anerkannten kryptografischen Algorithmen generiert werden. (C6)		✓	✓	331	7.1

TLS oder ein anderer sicherer Transportkanal ist für das Sessionmanagement obligatorisch. Dies wird im Kapitel Kommunikationssicherheit behandelt.

V3.3 Beenden der Session

Die Session Timeouts wurden an NIST 800-63 angeglichen, was wesentlich längere Sitzungsauszeiten erlaubt, als die Sicherheitsstandards traditionell zulassen. Prüfen Sie die nachstehende Tabelle: Falls eine längere Zeit der Inaktivität mit dem Risiko des Geschäftsbetriebes vereinbar ist, kann sich die Obergrenze am oberen NIST-Grenzwert des Session-Timeouts orientieren.

L1 (Stufe 1) ist in diesem Zusammenhang IAL1/AAL1, L2 (Stufe 2) ist IAL2/AAL3, L3 (Stufe 3) ist IAL3/AAL3. Je kürzer die Leerlaufzeit für IAL2/AAL2 und IAL3/AAL3 ist, desto niedriger ist die Grenze der Leerlaufzeiten für das Ausloggen oder die erneute Authentifizierung zur Wiederaufnahme der Sitzung.

#	Beschreibung	L1	L2	L3	CWE	NIST §
3.3.1	Prüfen Sie, dass Abmeldung und Ablauf das Sessiontoken ungültig machen, so dass die Zurück-Taste oder eine nachgeschaltete Relying Party eine authentifizierte Sitzung auch nicht zwischen den Relying Parties wiederaufnimmt. (C6)	✓	✓	✓	613	7.1
3.3.2	Wenn die Benutzer eingeloggt bleiben, Prüfen Sie, dass eine erneute Authentifizierung in regelmäßigen Abständen sowohl bei aktiver Nutzung als auch nach einer Leerlaufphase erfolgt. (C6)	30 Tage	12 Stunden oder 30 Minuten Inaktivität, 2FA optional	12 Stunden oder 15 Minuten Inaktivität, mit 2FA	613	7.2
3.3.3	Prüfen Sie, dass es die Anwendung ermöglicht, alle anderen aktiven Sitzungen nach einer erfolgreichen Kennwortänderung zu beenden. Dies muss in der gesamten Anwendung, der föderierten Anmeldung (falls vorhanden) und bei allen Relying Parties wirksam sein.		✓	✓	613	
3.3.4	Prüfen Sie, dass die Benutzer jede oder alle derzeit aktiven Sitzungen und Geräte sehen und sich von ihnen abmelden können.		✓	✓	613	7.1

V3.4 Cookiebasiertes Session Management

#	Beschreibung	L1	L2	L3	CWE	NIST §
3.4.1	Prüfen Sie, dass bei Session-Cookies das Attribut „secure“ gesetzt ist. (C6)	✓	✓	✓	614	7.1.1
3.4.2	Prüfen Sie, dass bei Session-Cookies das Attribut „HttpOnly“ gesetzt ist. (C6)	✓	✓	✓	1004	7.1.1
3.4.3	Prüfen Sie, dass Session-Cookies das 'SameSite'-Attribut verwenden, um die Anfälligkeit für Cross Site Request Forgery zu begrenzen. (C6)	✓	✓	✓	1275	7.1.1
3.4.4	Prüfen Sie, dass Session-Cookies das Präfix „__Host-“ verwenden (siehe Referenzen), um die Vertraulichkeit von Session-Cookies zu gewährleisten.	✓	✓	✓	16	7.1.1
3.4.5	Falls die Anwendung unter einem Domänennamen zusammen mit anderen Anwendungen veröffentlicht wird, die Session-Cookies nutzen, welche die Sitzungscookies der geprüften Anwendung außer Kraft setzen oder offenlegen könnten, prüfen Sie, dass das Pfadattribut in den Cookies einen möglichst exakten Pfad erhält. (C6)	✓	✓	✓	16	7.1.1

V3.5 Tokenbasiertes Sessionmanagement

Tokenbasiertes Sessionmanagement umfasst z.B. JWT-, OAuth-, SAML- und API-Schlüssel. API-Schlüssel sind bekanntermaßen schwach und sollten im neuen Code nicht mehr verwendet werden.

#	Beschreibung	L1	L2	L3	CWE	NIST §
3.5.1	Prüfen Sie, dass die Anwendung Benutzern erlaubt, OAuth-Token, die Vertrauensbeziehungen zu verknüpften Anwendungen herstellen, zurückzuziehen.		✓	✓	290	7.1.2
3.5.2	Prüfen Sie, dass die Anwendung Sessiontoken anstatt statischer API-Schlüssel verwendet, außer bei Legacy-Implementierungen.		✓	✓	798	
3.5.3	Prüfen Sie, dass zustandslose Session-Token digitale Signaturen, Verschlüsselung und andere Gegenmaßnahmen zum Schutz vor Manipulation, Enveloping, Wiedergabe, Null-Chiffren und Schlüsselaustausch-Angriffen verwenden.		✓	✓	345	

V3.6 Erneute Authentisierung

Dieser Abschnitt bezieht sich auf Entwickler, die den Code für die Relying Party (RP) oder den Credential Service Provider (CSP) schreiben. Wenn Sie auf eine Software vertrauen, die diese Funktionen implementiert, Prüfen Sie, dass die folgenden Punkte korrekt behandelt werden.

#	Beschreibung	L1	L2	L3	CWE	NIST §
3.6.1	Prüfen Sie, dass die Relying Parties gegenüber den CSPs die maximale Authentifizierungszeit angeben, und dass die CSPs den Teilnehmer erneut authentifizieren, wenn sie innerhalb dieses Zeitraums keine Sitzung verwendet haben.			✓	613	7.2.1
3.6.2	Prüfen Sie, dass die CSPs die Relying Parties über das letzte Authentifizierungsereignis informieren, damit die RPs feststellen können, ob sie den Benutzer erneut authentifizieren müssen.			✓	613	7.2.1

V3.7 Verteidigung gegen Session Management Exploits

Es gibt eine kleine Anzahl von Angriffen auf das Sessionmanagement, von denen einige mit der Benutzerführung von Sessions zusammenhängen. Auf der Grundlage des ISO 27002 Standards haben Vorversionen des ASVS das Blockieren mehrerer gleichzeitiger Sitzungen gefordert. Das Blockieren gleichzeitiger Sitzungen ist jedoch nicht mehr zeitgemäß, nicht nur, weil die heutigen Benutzer viele Geräte haben oder die Anwendung eine API ohne Browsersession ist, sondern auch weil in den meisten dieser Implementierungen der letzte Authentifikator – meist der Angreifer - gewinnt. Dieser Abschnitt ist eine Anleitung zum Abschrecken, Verzögern und Erkennen von Angriffen auf das Sessionmanagement.

Beschreibung des halboffenen Angriffs

Anfang 2018 wurden mehrere Finanzinstitute durch sogenannte halboffene Angriffe kompromittiert. Die Angreifer griffen sowohl unterschiedliche Institute mit unterschiedlichen proprietären Codebasen an als auch scheinbar verschiedene Codebasen innerhalb derselben Institute. Der halboffene Angriff macht sich einen Fehler im Design vieler Authentifizierungs-, Sessionmanagement- und Zugangskontrollsysteme zu Nutze. Die Angreifer starten einen halboffenen Angriff, indem sie versuchen, Zugangsdaten zu sperren, zurückzusetzen oder wiederherzustellen. Ein beliebtes Designmuster für das Sessionmanagement verwendet Benutzerprofile in Sessionobjekten bzw. -modellen zwischen nicht authentifiziertem, halb-authentifiziertem, z.B. Kennwortrücksetzung, vergessener Benutzername, und vollständig authentifiziertem Code wieder. Dieses Designmuster pflegt ein gültiges Sessionobjekt oder Token mit dem Profil des Opfers, einschließlich des Passwort-Hashes und der Rollen ein. Wenn die Zugriffskontrollprüfungen nicht korrekt verifizieren, dass der Benutzer vollständig angemeldet ist, kann der Angreifer als der Benutzer handeln. Angriffe könnten das Benutzerkennwort auf einen bekannten Wert ändern, die E-Mail-Adresse zur Durchführung einer gültigen

Kennwortzurücksetzung aktualisieren, die Mehrfaktorauthentifizierung deaktivieren oder ein neues MFA-Gerät registrieren, API-Schlüssel offenlegen bzw. ändern und so weiter.

#	Beschreibung	L1	L2	L3	CWE	NIST §
3.7.1	Prüfen Sie, dass die Anwendung eine gültige Login Session gewährleistet oder eine erneute Authentifizierung oder eine sekundäre Verifizierung erfordert, bevor sensible Transaktionen oder Kontenänderungen zugelassen werden.	✓	✓	✓	306	

Referenzen

Weitere Informationen finden Sie unter:

- OWASP Testing Guide 4.0: Session Management Testing
- OWASP Session Management Cheat Sheet
- Set-Cookie __Host- prefix details

V4 Maßnahmen zur Zugriffssteuerung

Ziel

Autorisierung ist das Konzept, nur denjenigen den Zugriff auf Ressourcen zu gestatten, die diese auch nutzen dürfen. Prüfen Sie, dass eine verifizierte Anwendung die folgenden High Level Anforderungen erfüllt:

- Personen, die auf Ressourcen zugreifen, müssen dafür über gültige Berechtigungen verfügen.
- Die Benutzer sind mit einem genau definierten Satz von Rollen und Berechtigungen verbunden.
- Rollen- und Berechtigungsmetadaten sind vor Wiedereinspielen oder Manipulation geschützt.

V4.1 Design der Allgemeinen Zugriffssteuerung

#	Beschreibung	L1	L2	L3	CWE
4.1.1	Prüfen Sie, dass die Anwendung Regeln zur Steuerung der Zugriffe auf einer vertrauenswürdigen Serviceschicht durchsetzt, insbesondere wenn die clientseitige Zugriffssteuerung umgangen werden könnte.	✓	✓	✓	602
4.1.2	Prüfen Sie, dass alle Benutzer- und Datenattribute sowie Richtlinieninformationen, die von der Zugriffssteuerung verwendet werden, von den Endnutzern nicht manipuliert werden können, es sei denn, dies wird ausdrücklich genehmigt.	✓	✓	✓	639
4.1.3	Prüfen Sie, dass das Prinzip der minimalen Berechtigung gilt: Benutzer sollten nur auf die unbedingt notwendigen Funktionen, Dateien, URLs, Controller, Dienste und andere Ressourcen zugreifen können. Dies bedeutet Schutz vor Spoofing und Ausweitung der Berechtigungen. (C7)	✓	✓	✓	285
4.1.4	[GELÖSCHT, DUPLIKAT VON 4.1.3]				
4.1.5	Prüfen Sie, dass die Zugriffssteuerungsroutinen im Fehlerfall in einen sicheren Zustand fallen. (C10)	✓	✓	✓	285

V4.2 Operative Zugriffssteuerung

#	Beschreibung	L1	L2	L3	CWE
4.2.1	Prüfen Sie, dass sensible Daten und APIs vor direkten Objektangriffen geschützt sind, die auf das Erstellen, Lesen, Aktualisieren und Löschen von Datensätzen abzielen, z. B. das Erstellen oder Aktualisieren von Datensätzen einer anderen Person, das Anzeigen oder Löschen aller Datensätze.	✓	✓	✓	639
4.2.2	Prüfen Sie, dass die Anwendung oder das Framework einen starken Anti-CSRF-Mechanismus zum Schutz authentifizierter Funktionen durchsetzt, und dass eine effektive Anti-Automatisierung oder Anti-CSRF nicht authentifizierte Funktionen schützt.	✓	✓	✓	352

V4.3 Weitere Maßnahmen zur Zugriffssteuerung

#	Beschreibung	L1	L2	L3	CWE
4.3.1	Prüfen Sie, dass administrative Schnittstellen eine geeignete Mehrfaktorauthentifizierung verwenden, um unbefugte Nutzung zu verhindern.	✓	✓	✓	419

#	Beschreibung	L1	L2	L3	CWE
4.3.2	Prüfen Sie, dass das Durchsuchen von Verzeichnissen deaktiviert ist, es sei denn, dies ist absichtlich gewünscht. Ferner ist das Auffinden oder die Offenlegung von Datei- oder Verzeichnis-Metadaten, wie z.B. Thumbs.db, .DS_Store, .git oder .svn-Ordner, nicht zulässig.	✓	✓	✓	548
4.3.3	Prüfen Sie, dass die Anwendung über zusätzliche Berechtigungen (z. B. Step-Up oder adaptive Authentifizierung) für risikoarme Systeme und / oder Aufgabentrennung für brisante Anwendungen verfügt, um Betrugsbekämpfungsmaßnahmen entsprechend dem Anwendungsrisiko durchzusetzen.		✓	✓	732

Referenzen

Weitere Informationen finden Sie unter:

- OWASP Testing Guide 4.0: Authorization
- OWASP Cheat Sheet: Access Control
- OWASP CSRF Cheat Sheet
- OWASP REST Cheat Sheet

V5 Eingabeprüfung, die Bereinigung der Ausgaben und die Zeichencodierung

Ziel

Die häufigste Sicherheitsschwachstelle von Webanwendungen ist die mangelhafte Prüfung von Eingabedaten, die vom Client oder von der Umgebung kommen, bevor sie ohne Ausgabecodierung direkt weiterverwendet werden. Diese Schwachstelle führt zu fast allen signifikanten Schwachstellen in Webanwendungen, wie z.B. Cross Site Scripting (XSS), SQL-Injection, Interpreter-Injection, Angriffe auf Zeichensätze oder auf das Dateisystem und schließlich Pufferüberläufe.

Prüfen Sie, dass eine verifizierte Anwendung die folgenden High Level Anforderungen erfüllt:

- Die Architektur der Inputvalidierung und der Ausgabebereinigung haben eine abgestimmte Pipeline, um Injektionsangriffe zu verhindern.
- Die Eingabedaten werden stark typisiert, ihr Wertebereich bzw. ihre Länge wird überprüft, im Maximum werden sie bereinigt oder gefiltert.
- Die Ausgabedaten werden entsprechend dem Kontext der Daten so nah wie möglich am Interpreter codiert oder bereinigt.

In modernen Webanwendungen ist die Ausgabecodierung wichtiger denn je. In bestimmten Fällen ist es schwierig, eine robuste Eingabevalidierung umzusetzen. Daher ist die Verwendung sichererer Schnittstellen, wie parametrisierte Abfragen, Autoescape Template-Frameworks oder die sorgfältig ausgewählte Ausgabecodierung für die Sicherheit der Anwendung von entscheidender Bedeutung.

V5.1 Eingabeprüfung

Richtig implementierte Maßnahmen zur Eingabeprüfung, die eine positive Whitelist und eine starke Datentypisierung verwenden, können mehr als 90% aller Injektion-Angriffe eliminieren. Prüfungen der Länge und des Wertebereiches können weitere Angriffe verhindern. Während der Anwendungsarchitektur, der Designsprints, der Programmierung sowie der Unit- und Integrationstests muss eine sichere Eingabeprüfung eingebaut werden. Viele dieser Probleme können bei Penetrationstests nicht gefunden werden. Die Versäumnisse der Eingabeprüfung müssen dann in der Regel mit den Maßnahmen aus Abschnitt V5.3 Anforderungen an Ausgabecodierung und Injektionsverhinderung ausgebessert werden. Entwicklern und Code-Reviewern wird empfohlen, diesen Abschnitt so zu behandeln, als ob L1 für alle Elemente erforderlich wäre, um Injektions-Angriffe zu verhindern.

#	Beschreibung	L1	L2	L3	CWE
5.1.1	Prüfen Sie, dass die Anwendung über Abwehrmechanismen gegen Angriffe auf HTTP-Parameter verfügt, insbesondere dann, wenn das Anwendungsframework die Quelle der Anforderungsparameter (GET, POST, Cookies, Header oder Umgebungsvariablen) nicht unterscheidet.	✓	✓	✓	235
5.1.2	Prüfen Sie, dass Frameworks vor Angriffen durch massenhafte Parameterzuweisung schützen, oder dass die Anwendung über Gegenmaßnahmen zum Schutz vor unsicherer Parameterzuweisung verfügt, wie z.B. das Markieren von Feldern als privat oder ähnliches. (C5)	✓	✓	✓	915
5.1.3	Prüfen Sie, dass alle Eingaben (HTML-Formularfelder, REST-Anforderungen, URL-Parameter, HTTP-Header, Cookies, Batch-Dateien, RSS-Feeds usw.) mittels positiver Validierung (Whitelisting) validiert werden. (C5)	✓	✓	✓	20

#	Beschreibung	L1	L2	L3	CWE
5.1.4	Prüfen Sie, dass strukturierte Daten stark typisiert sind und gemäß einem definierten Schema validiert werden. Dazu gehören die erlaubten Zeichen, Länge und Muster (z. B. Kreditkarten- oder Telefonnummern, oder die Prüfung, dass zwei zusammenhängende Felder stimmig sind, z.B. Ort und Postleitzahl). (C5)	✓	✓	✓	20
5.1.5	Prüfen Sie, dass URL-Umleitungen und -Weiterleitungen nur Whitelist-Ziele zulassen, oder bei der Weiterleitung auf potenziell nicht vertrauenswürdige Inhalte einen Warnhinweis anzeigen.	✓	✓	✓	601

V5.2 Bereinigung und Sandboxing

#	Beschreibung	L1	L2	L3	CWE
5.2.1	Prüfen Sie, dass alle nicht vertrauenswürdigen HTML-Eingaben von WYSIWYG-Editoren o.ä. ordnungsgemäß mit einer HTML-Bereinigungsbibliothek oder einer Frameworkfunktion bereinigt werden. (C5)	✓	✓	✓	116
5.2.2	Prüfen Sie, dass unstrukturierte Daten bereinigt werden, um Sicherheitsmaßnahmen wie erlaubte Zeichen und Längenbegrenzung durchzusetzen.	✓	✓	✓	138
5.2.3	Prüfen Sie, dass die Anwendung zum Schutz vor SMTP- oder IMAP-Injektion Benutzereingaben bereinigt, bevor sie an Mailsysteme weitergeleitet werden.	✓	✓	✓	147
5.2.4	Prüfen Sie, dass die Anwendung kein eval() oder andere Funktionen zur dynamischen Ausführung von Code verwendet. Wenn es keine Alternative gibt, müssen alle Benutzereingaben, die einbezogen werden, vor der Ausführung des Programms gesäubert oder per Sandbox abgegrenzt werden.	✓	✓	✓	95
5.2.5	Prüfen Sie, dass die Anwendung vor Template-Injection-Angriffen schützt, indem Sie sicherstellen, dass alle Benutzereingaben, die aufgenommen werden, bereinigt oder per Sandbox abgegrenzt werden.	✓	✓	✓	94
5.2.6	Prüfen Sie, dass die Anwendung vor SSRF-Angriffen schützt, indem sie nicht vertrauenswürdige Daten oder HTTP-Dateimetadaten, wie z. B. Dateinamen und URL-Eingabefelder, validiert oder bereinigt. Verwenden Sie eine Whitelist von Protokollen, Domänen, Pfaden und Ports.	✓	✓	✓	918
5.2.7	Prüfen Sie, dass die Anwendung vom Benutzer bereitgestellte Scaleable Vector Graphics (SVG) von skriptfähigen Inhalten bereinigt, deaktiviert oder in Sandboxen abgrenzt, insbesondere in Bezug auf XSS, das aus Inline-Skripten und aus foreignObject resultiert.	✓	✓	✓	159
5.2.8	Prüfen Sie, dass die Anwendung vom Benutzer zur Verfügung gestellte skriptfähige Inhalte oder Inhalte von Expression Language Templates wie Markdown, CSS- oder XSL-Stylesheets, BBCode oder Ähnliches bereinigt, deaktiviert oder in Sandboxen abgrenzt.	✓	✓	✓	94

V5.3 Ausgabecodierung und Injektionsverhinderung

Für die Sicherheit der Anwendung ist es entscheidend, dass die Ausgabecodierung möglichst eng bei dem verwendeten Interpreter stattfindet. In der Regel wird die Ausgabecodierung nicht beibehalten, sondern dazu verwendet, die Ausgabe im entsprechenden Kontext für die sofortige Nutzung sicher zu machen. Unzureichende Ausgabecodierung führt direkt zu einer unsicheren Anwendung.

#	Beschreibung	L1	L2	L3	CWE
5.3.1	Prüfen Sie, dass die Ausgabecodierung für den Interpreter und den erforderlichen Kontext relevant ist. Verwenden Sie z. B. Codierer gezielt für HTML-Werte, HTML-Attribute, JavaScript, URL-Parameter, HTTP-Header, SMTP und andere, wie es der Kontext erfordert, insbesondere bei nicht vertrauenswürdigen Eingaben (z.B. Namen mit Unicode oder Apostroph, wie z.B. ねこ oder O'Hara). (C4)	✓	✓	✓	116
5.3.2	Prüfen Sie, dass die Ausgabecodierung den vom Benutzer gewählten Zeichensatz sowie die Spracheinstellung beibehält, so dass jeder Unicode-Zeichenpunkt gültig ist und sicher verarbeitet wird. (C4)	✓	✓	✓	176
5.3.3	Prüfen Sie, dass kontextabhängiges, vorzugsweise automatisches Output Escaping vor reflektierten, gespeicherten und DOM-basierten XSS schützt. (C4)	✓	✓	✓	79
5.3.4	Prüfen Sie, dass die Datenauswahl- oder Datenbankabfragen (z.B. SQL, HQL, ORM, NoSQL) parametrisierte Abfragen, ORMs, Entity Frameworks verwenden oder anderweitig vor Datenbank-Injektionsangriffen geschützt sind. (C3)	✓	✓	✓	89
5.3.5	Prüfen Sie, dass dort, wo keine parametrisierten oder sichereren Mechanismen vorhanden sind, eine kontextspezifische Ausgabecodierung, z. B. SQL-Escaping, zum Schutz vor Injektionsangriffen verwendet wird. (C3, C4)	✓	✓	✓	89
5.3.6	Prüfen Sie, dass die Anwendung vor Angriffen mittels JSON-Injektion, JSON-eval() und Evaluierung von JavaScript-Ausdrücken schützt. (C4)	✓	✓	✓	830
5.3.7	Prüfen Sie, dass die Anwendung vor LDAP-Injektionsschwachstellen schützt oder das spezifische Sicherheitsmaßnahmen zur Verhinderung der LDAP-Injektion implementiert wurden. (C4)	✓	✓	✓	90
5.3.8	Prüfen Sie, dass die Anwendung vor dem Einfügen von Betriebssystemkommandos schützt und dass Betriebssystemaufrufe parametrisierte Abfragen oder eine kontextbezogene Ausgabecodierung der Befehlszeile verwenden. (C4)	✓	✓	✓	78
5.3.9	Prüfen Sie, dass die Anwendung vor Local File Inclusion (LFI)- oder Remote File Inclusion (RFI)-Angriffen schützt.	✓	✓	✓	829
5.3.10	Prüfen Sie, dass die Anwendung gegen XPath Injection- oder XML-Injection-Angriffe schützt. (C4)	✓	✓	✓	643

Hinweis: Die Verwendung parametrisierter Abfragen oder des SQL-Escape ist nicht immer ausreichend: Tabellen- und Spaltennamen, ORDER BY usw. können nicht escaped werden. Die Aufnahme von escapten, vom Benutzer bereitgestellten Daten in diesen Feldern führt zu fehlgeschlagenen Abfragen oder SQL-Injektion.

Hinweis: Das SVG-Format erlaubt explizit ECMA-Skript in fast allen Kontexten, also ist es eventuell nicht möglich, alle SVG-XSS-Vektoren vollständig zu blockieren. Wenn ein SVG-Upload erforderlich ist, empfehlen wir dringend, diese hochgeladenen Dateien entweder als Text/Plain auszuliefern oder eine separate, vom Benutzer bereitgestellte, Inhaltsdomäne zu verwenden, um zu verhindern, dass ein erfolgreiches XSS die Anwendung übernimmt.

V5.4 Speicher, Strings und Unmanaged Code

Die folgenden Anforderungen gelten nur, wenn die Anwendung eine Systemprogrammiersprache oder unmanaged Code verwendet.

#	Beschreibung	L1	L2	L3	CWE
5.4.1	Prüfen Sie, dass die Anwendung speichersichere Zeichenfolgen, sicherere Speicherkopien und sichere Zeigerarithmetik verwendet, um Stapel-, Puffer- oder Heapüberläufe zu erkennen oder zu verhindern.		✓	✓	120
5.4.2	Prüfen Sie, dass Formatstrings keine potenziell feindliche Eingabe annehmen und konstant sind.		✓	✓	134
5.4.3	Prüfen Sie, dass Zeichen-, Bereichs- und Eingabeprüfungstechniken verwendet werden, um Ganzzahlüberläufe zu verhindern.		✓	✓	190

V5.5 Prävention von Deserialisierungsangriffen

#	Beschreibung	L1	L2	L3	CWE
5.5.1	Prüfen Sie, dass serialisierte Objekte Integritätsprüfungen verwenden oder verschlüsselt sind, um die Erstellung feindlicher Objekte oder die Manipulation von Daten zu verhindern. (C5)	✓	✓	✓	502
5.5.2	Prüfen Sie, dass XML-Parser nur die restriktivste Konfiguration verwenden und das unsichere Funktionen wie die Auflösung externer Entitäten deaktiviert sind, um XXE zu verhindern.	✓	✓	✓	611
5.5.3	Prüfen Sie, dass die Deserialisierung nicht vertrauenswürdiger Daten sowohl im benutzerdefinierten Code als auch in Bibliotheken von Drittanbietern (wie JSON-, XML- und YAML-Parser) entweder verhindert oder gesichert wird.	✓	✓	✓	502
5.5.4	Prüfen Sie, dass beim Parsen von JSON in Browsern oder JavaScript-basierten Backends JSON.parse zum Parsen des JSON-Dokuments verwendet wird. Verwenden Sie kein eval() zum Parsen von JSON.	✓	✓	✓	95

Referenzen

Weitere Informationen finden Sie unter:

- OWASP Testing Guide 4.0: Input Validation Testing
- OWASP Cheat Sheet: Input Validation
- OWASP Testing Guide 4.0: Testing for HTTP Parameter Pollution
- OWASP LDAP Injection Cheat Sheet
- OWASP Testing Guide 4.0: Client Side Testing
- OWASP Cross Site Scripting Prevention Cheat Sheet
- OWASP DOM Based Cross Site Scripting Prevention Cheat Sheet
- OWASP Java Encoding Project
- OWASP Mass Assignment Prevention Cheat Sheet
- DOMPurify - Client-side HTML Sanitization Library
- XML External Entity (XXE) Prevention Cheat Sheet

Weitere Informationen zum Auto-Escaping finden Sie unter:

- Reducing XSS by way of Automatic Context-Aware Escaping in Template Systems
- AngularJS Strict Contextual Escaping
- AngularJS ngBind

- Angular Sanitization
- Angular Security
- ReactJS Escaping
- Improperly Controlled Modification of Dynamically-Determined Object Attributes

Weitere Informationen zur Deserialisierung finden Sie unter:

- OWASP Deserialization Cheat Sheet
- OWASP Deserialization of Untrusted Data Guide

V6 kryptographische Komponenten

Ziel

Prüfen Sie, dass eine verifizierte Anwendung die folgenden High Level Anforderungen erfüllt:

- Alle kryptographischen Module fallen in einen sicheren Fehlerzustand und behandeln Fehler korrekt.
- Es wird ein geeigneter Zufallszahlengenerator verwendet.
- Der Zugriff auf Schlüssel wird sicher verwaltet.

V6.1 Datenklassifizierung

Das wichtigste Gut sind die Daten, die von einer Anwendung verarbeitet, gespeichert oder übertragen werden. Führen Sie immer eine Datenschutzfolgenabschätzung durch, um die Datenschutzbedürfnisse der gespeicherten Daten korrekt einzuschätzen.

#	Beschreibung	L1	L2	L3	CWE
6.1.1	Prüfen Sie, dass sensible personenbezogene Daten, oder Daten, die unter andere gesetzliche Regelungen zur Vertraulichkeit fallen, verschlüsselt gespeichert werden.		✓	✓	311
6.1.2	Prüfen Sie, dass Gesundheitsdaten, wie z. B. medizinische Aufzeichnungen, Details zu medizinischen Geräten oder deanonymisierte Forschungsaufzeichnungen, verschlüsselt gespeichert werden.		✓	✓	311
6.1.3	Prüfen Sie, dass Finanzdaten, wie z.B. Konten, Zahlungsausfälle oder Kredithistorie, Steuerunterlagen, Lohnhistorie, Begünstigte oder deanonymisierte Markt- oder Forschungsaufzeichnungen verschlüsselt gespeichert werden.		✓	✓	311

V6.2 Algorithmen

Fortschritte in der Kryptographie führen dazu, dass bisher sichere Algorithmen und Schlüssellängen nicht länger sicher genug sind, um Daten zu schützen. Daher muss es möglich sein, die Algorithmen zu ändern. Obwohl dieser Abschnitt nicht leicht mittels Pentests geprüft werden kann, sollten Entwickler diesen gesamten Abschnitt als obligatorisch betrachten, auch wenn L1 bei den meisten Punkten fehlt.

#	Beschreibung	L1	L2	L3	CWE
6.2.1	Prüfen Sie, dass alle kryptografischen Module in einen sicheren Fehlerzustand fallen und Fehler so behandelt werden, dass keine Padding-Orakel-Angriffe möglich sind.	✓	✓	✓	310
6.2.2	Prüfen Sie, dass allgemein anerkannte oder von der Regierung freigegebene kryptografische Algorithmen, Modi und Bibliotheken anstelle von Eigenentwicklungen verwendet werden. (C8)		✓	✓	327
6.2.3	Prüfen Sie, dass der Initialisierungsvektoren, die Chiffrierkonfiguration und die Blockmodi gemäß den neuesten Empfehlungen sicher konfiguriert werden.		✓	✓	326
6.2.4	Prüfen Sie, dass Zufallszahlengeneratoren, Verschlüsselungs- oder Hashalgorithmen, Schlüssellängen, Runden, Chiffren oder Modi jederzeit rekonfiguriert, aktualisiert oder ausgetauscht werden können. (C8)		✓	✓	326

#	Beschreibung	L1	L2	L3	CWE
6.2.5	Prüfen Sie, dass unsichere Blockmodi, wie ECB u.a., Padding-Modi, wie PKCS#1 v1.5 u.a., Algorithmen mit kleinen Blockgrößen, wie Triple-DES, Blowfish u.a., sowie schwache Hashalgorithmen, wie MD5, SHA1 u.a., nicht verwendet werden, es sei denn, dies ist aus Gründen der Rückwärtskompatibilität erforderlich.		✓	✓	326
6.2.6	Prüfen Sie, dass Nonces, Initialisierungsvektoren u. ä. nicht mehr als einmal mit einem bestimmten Verschlüsselungsschlüssel verwendet werden dürfen. Die Methode der Generierung muss für den verwendeten Algorithmus geeignet sein.		✓	✓	326
6.2.7	Prüfen Sie, dass verschlüsselte Daten mittels Signaturen, authentifizierte Chiffriermodi oder HMAC authentifiziert werden, um sicherzustellen, dass der Chiffriertext nicht von Unbefugten verändert wird.			✓	326
6.2.8	Prüfen Sie, dass alle kryptografischen Operationen zeitkonstant sind und keine „Kurzschluss“-Operationen bei Vergleichen, Berechnungen oder Rückgaben stattfinden, um Informationslecks zu vermeiden.			✓	385

V6.3 Zufallswerte

Das Erzeugen von guten Pseudo-Zufallszahlen (PRN) ist unglaublich schwer. Im Allgemeinen werden gute Entropiequellen innerhalb eines Systems schnell erschöpft, wenn sie übermäßig genutzt werden, Quellen mit weniger Entropie hingegen können zu vorhersagbaren Schlüsseln bzw. Geheimnissen führen.

#	Beschreibung	L1	L2	L3	CWE
6.3.1	Prüfen Sie, dass alle Zufallszahlen, zufälligen Dateinamen, zufälligen GUIDs und Zufallszeichenfolgen mit einem anerkannten kryptografisch sicheren Zufallszahlengenerator generiert werden, wenn diese Zufallswerte für einen Angreifer nicht zu erraten sein sollen.		✓	✓	338
6.3.2	Prüfen Sie, dass zufällige GUIDs mit dem GUID v4-Algorithmus und einem kryptografisch sicheren Zufallszahlengenerator erstellt werden. GUIDs, die mit anderen Zufallszahlengeneratoren erstellt wurden, können vorhersehbar sein.		✓	✓	338
6.3.3	Prüfen Sie, dass die Zufallszahlen mit der richtigen Entropie erzeugt werden, auch wenn die Anwendung unter starker Belastung steht, oder dass die Anwendung unter solchen Umständen angemessen reagiert.			✓	338

V6.4 Management von Schlüsseln und Geheimnissen

Obwohl dieser Abschnitt nicht leicht mittels Pentests getestet werden kann, sollten Entwickler diesen gesamten Abschnitt als obligatorisch betrachten, auch wenn L1 bei den meisten Punkten fehlt.

#	Beschreibung	L1	L2	L3	CWE
6.4.1	Prüfen Sie, dass eine Lösung für das Schlüsselmanagement, wie z.B. ein Schlüsseltresor, verwendet wird, um Geheimnisse sicher zu erstellen, zu speichern, die Nutzung zu kontrollieren und sie zu zerstören. (C8)		✓	✓	798
6.4.2	Prüfen Sie, dass das Schlüsselmaterial nicht in der Anwendung genutzt wird sondern ein Sicherheitsmodul kryptographische Operationen ausführt. (C8)		✓	✓	320

Referenzen

Weitere Informationen finden Sie unter:

- OWASP Testing Guide 4.0: Testing for weak Cryptography
- OWASP Cheat Sheet: Cryptographic Storage
- FIPS 140-2

V7 Fehlerbehandlung und Protokollierung

Ziel

Das primäre Ziel der Fehlerbehandlung und Protokollierung ist es, nützliche Informationen für den Benutzer, die Administratoren und die Incident Response Teams bereitzustellen. Das Ziel besteht nicht darin, große Mengen von Protokollen zu erstellen, sondern hochwertige Protokolle, die mehr Informationen als nutzloses Rauschen enthalten. Hochwertige Protokolle enthalten oft sensible Daten und müssen gemäß den lokalen Datenschutzgesetzen oder -richtlinien geschützt werden. Dies sollte beinhalten:

- Keine sensiblen Informationen zu protokollieren, es sei denn, dies ist ausdrücklich erforderlich.
- Sicherzustellen, dass alle protokollierten Informationen sicher gehandhabt und gemäß ihrer Datenklassifizierung geschützt werden.
- Sicherzustellen, dass die Protokolle nicht für immer gespeichert werden, sondern eine möglichst kurze Lebensdauer haben.

Wenn Protokolle personenbezogene oder sensible Daten enthalten, werden sie für Angreifer sehr attraktiv. Es ist ferner wichtig zu gewährleisten, dass die Anwendung in sichere Fehlerzustände fällt und dass Fehler keine unnötigen Informationen offenbaren.

V7.1 Protokollinhalt

Die Protokollierung sensibler Informationen ist gefährlich - die Protokolle werden selbst sensibel und müssen verschlüsselt werden. Sie unterliegen Aufbewahrungsrichtlinien und müssen bei Sicherheitsaudits offengelegt werden. Prüfen Sie, dass nur die notwendigen Informationen in den Protokollen aufbewahrt werden, jedoch auf keinen Fall Zahlungen, Anmeldedaten, einschließlich Sessiontoken, sensible oder personenbezogene Informationen. Dieser Abschnitt deckt OWASP Top 10 2017:A10 ab. Weil er nicht durch Penetrationstests verifizierbar ist, sollen:

- Entwickler die vollständige Einhaltung dieses Abschnitts sicherstellen, als ob alle Punkte mit L1 gekennzeichnet wären.
- Penetrationstester die vollständige Einhaltung aller Punkte durch Befragung, Screenshots oder Zusicherungen validieren.

#	Beschreibung	L1	L2	L3	CWE
7.1.1	Prüfen Sie, dass die Anwendung keine Anmeldeinformationen oder Zahlungsdetails protokolliert. Sessiontoken sollten nur in einer irreversiblen, gehashten Form in Protokollen gespeichert werden. (C9, C10)	✓	✓	✓	532
7.1.2	Prüfen Sie, dass die Anwendung keine sonstigen sensiblen Daten protokolliert, die z. B. gemäß Datenschutzgesetzen oder den einschlägigen Sicherheitsrichtlinien als solche definiert werden. (C9)	✓	✓	✓	532
7.1.3	Prüfen Sie, dass die Anwendung sicherheitsrelevante Ereignisse, einschließlich erfolgreicher und fehlgeschlagener Authentifizierungsereignisse, Fehler bei der Zugriffskontrolle, Deserialisierungsfehler und Fehler bei der Eingabeprüfung protokolliert. (C5, C7)		✓	✓	778
7.1.4	Prüfen Sie, dass jedes Protokollereignis die notwendigen Informationen enthält, um bei einem Vorfall eine detaillierte Untersuchung der Timeline zu ermöglichen. (C9)		✓	✓	778

V7.2 Protokollbearbeitung

Die rechtzeitige Protokollierung ist entscheidend für Auditereignisse, Triage und Eskalation. Prüfen Sie, dass die Anwendungsprotokolle klar sind und entweder lokal oder per Versand an ein Fernüberwachungssystem

leicht überwacht und analysiert werden können. Dieser Abschnitt deckt OWASP Top 10 2017:A10 ab. Weil er nicht durch Penetrationstests verifizierbar ist, sollen:

- Entwickler die vollständige Einhaltung dieses Abschnitts sicherstellen, als ob alle Punkte mit L1 gekennzeichnet wären.
- Penetrationstester die vollständige Einhaltung aller Punkte durch Befragung, Screenshots oder Zusicherungen validieren.

#	Beschreibung	L1	L2	L3	CWE
7.2.1	Prüfen Sie, dass alle Authentifizierungsentscheidungen protokolliert werden, ohne dass sensible Sitzungstoken oder Passwörter gespeichert werden. Dies sollte auch die relevanten Metadaten umfassen, die für Sicherheitsuntersuchungen benötigt werden.		✓	✓	778
7.2.2	Prüfen Sie, dass alle Authentifizierungen protokolliert werden können, und dass alle fehlgeschlagenen Versuche protokolliert werden. Dies sollte die relevanten Metadaten umfassen, die für Sicherheitsuntersuchungen benötigt werden.		✓	✓	285

V7.3 Schutz von Protokollen

Protokolle, die einfach geändert oder gelöscht werden können, sind für Sicherheitsuntersuchungen nutzlos. Die Offenlegung von Protokollen kann interne Details über die Anwendung oder die darin enthaltenen Daten enthüllen. Schützen Sie Protokoll sorgfältig vor unbefugter Offenlegung, Änderung oder Löschung.

#	Beschreibung	L1	L2	L3	CWE
7.3.1	Prüfen Sie, dass alle Komponenten Daten angemessen codieren, um Log-Injektions-Angriffe zu verhindern. (C9)		✓	✓	117
7.3.2	[GELÖSCHT, DUPLIKAT VON 7.3.1]				
7.3.3	Prüfen Sie, dass die Protokolle vor unbefugtem Zugriff und Änderungen geschützt werden. (C9)		✓	✓	200
7.3.4	Prüfen Sie, dass die Zeitquellen mit der richtigen Zeit und Zeitzone synchronisiert sind. Erwägen Sie ernsthaft die Protokollierung ausschließlich in UTC, wenn die Systeme global sind, damit die forensische Analyse nach dem Vorfall unterstützt wird. (C9)		✓	✓	

Hinweis: Das korrekte Codieren von Logeinträgen (7.3.1) ist mit automatisierten Tools und Penetrationstests schwer zu testen, aber Architekten, Entwickler und Quellcodeprüfer sollten sie dennoch als L1-Anforderung betrachten.

V7.4 Fehlerbehandlung

Der Zweck der Fehlerbehandlung besteht darin, dass die Anwendung sicherheitsrelevante Ereignisse zur Überwachung, Triage und Eskalation bereitstellen kann. Ihr Zweck ist nicht die Erstellung von Protokollen. Bei der Protokollierung sicherheitsrelevanter Ereignisse ist sicherzustellen, dass das Protokoll zielgerichtet ist und dass es durch SIEM- oder Analysesoftware ausgewertet werden kann.

#	Beschreibung	L1	L2	L3	CWE
7.4.1	Prüfen Sie, dass bei Auftreten eines unerwarteten oder sicherheitsrelevanten Fehlers eine generische Meldung angezeigt wird. Ggf. kann die Meldung eine ID enthalten, die dem Supportpersonal die Untersuchung erleichtert. (C10)	✓	✓	✓	210

#	Beschreibung	L1	L2	L3	CWE
7.4.2	Prüfen Sie, dass die Ausnahmebehandlung in der gesamten Codebasis verwendet wird, um erwartete und unerwartete Fehlerbedingungen zu berücksichtigen. (C10)		✓	✓	544
7.4.3	Prüfen Sie, dass ein Fehlerbehandlungsdienst der letzten Instanz definiert ist, der alle nicht behandelten Ausnahmen abfängt. (C10)		✓	✓	431

Hinweis: Bestimmte Sprachen, wie z.B. Swift und Go - und durch gängige Designpraxis - viele funktionale Sprachen, unterstützen keine Ausnahmen oder letztinstanzliche Fehlerbehandlung. In diesem Fall sollten Architekten und Entwickler anderweitig sicherstellen, dass Anwendungen außergewöhnliche, unerwartete oder sicherheitsrelevante Ereignisse sicher handhaben können.

Referenzen

Weitere Informationen finden Sie unter:

- OWASP Testing Guide 4.0 content: Testing for Error Handling
- OWASP Authentication Cheat Sheet section about error messages

V8 Schutz von Informationen

Ziel

Es gibt drei Schlüsselelemente für die Informationssicherheit: Vertraulichkeit, Integrität und Verfügbarkeit. Dieser Standard geht davon aus, dass die Informationssicherheit auf einem vertrauenswürdigen System, wie z.B. einem Server, durchgesetzt wird, das gehärtet wurde und ausreichend geschützt ist.

Anwendungen müssen davon ausgehen, dass alle Benutzergeräte in irgendeiner Weise gefährdet sind. Wenn eine Anwendung sensible Informationen auf unsicheren Geräten wie gemeinsam genutzten Computern, Telefonen und Tablets überträgt oder speichert, ist die Anwendung dafür verantwortlich, dass die auf diesen Geräten gespeicherten Daten verschlüsselt werden und nicht so einfach unrechtmäßig erlangt, verändert oder offengelegt werden können.

Prüfen Sie, dass eine verifizierte Anwendung die folgenden High Level Anforderungen erfüllt:

- Vertraulichkeit: Daten sollten sowohl während der Übertragung als auch bei der Speicherung vor unbefugter Beobachtung oder Offenlegung geschützt werden.
- Integrität: Daten sollten vor böswilliger Erstellung, Änderung oder Löschung durch unbefugte Angreifer geschützt werden.
- Verfügbarkeit: Die Daten sollten für autorisierte Benutzer nach Bedarf verfügbar sein.

V8.1 Allgemeines

#	Beschreibung	L1	L2	L3	CWE
8.1.1	Prüfen Sie, dass die Anwendung sensible Daten davor schützt, in Serverkomponenten wie Loadbalancern, Proxies u.ä. zwischengespeichert zu werden.		✓	✓	524
8.1.2	Prüfen Sie, dass alle serverseitigen temporären Kopien sensibler Daten vor unbefugtem Zugriff geschützt oder nach dem Zugriff des autorisierten Benutzers auf die sensiblen Daten bereinigt/invalidiert werden.		✓	✓	524
8.1.3	Prüfen Sie, dass die Anwendung die Anzahl der Parameter in einer Anfrage, wie z.B. versteckte Felder, Ajax-Variablen, Cookies und Header-Werte minimiert.		✓	✓	233
8.1.4	Prüfen Sie, dass die Anwendung eine abnormale Anzahl von Anfragen, z.B. nach IP, Benutzer, Gesamtzahl pro Stunde oder Tag o. ä., erkennt und Alarm auslöst.		✓	✓	770
8.1.5	Prüfen Sie, dass wichtige Daten regelmäßig gesichert werden und dass die Wiederherstellung regelmäßig geübt wird.			✓	19
8.1.6	Prüfen Sie, dass die Backups sicher aufbewahrt werden, um zu verhindern, dass Daten gestohlen oder verfälscht werden.			✓	19

V8.2 Clientseitiger Schutz

#	Beschreibung	L1	L2	L3	CWE
8.2.1	Prüfen Sie, dass die Anwendung Anti-Caching Header sendet, damit sensible Daten in modernen Browsern nicht zwischengespeichert werden.	✓	✓	✓	525
8.2.2	Prüfen Sie, dass die im clientseitigen Speicher (z. B. lokaler HTML5-Speicher, Sitzungsspeicher, IndexedDB oder Cookies) gespeicherten Daten keine sensiblen Daten enthalten.	✓	✓	✓	922

#	Beschreibung	L1	L2	L3	CWE
8.2.3	Prüfen Sie, dass authentifizierte Daten aus dem clientseitigem Speicher, z. B. dem Browser-DOM, gelöscht werden, nachdem der Client geschlossen oder die Sitzung beendet wurde.	✓	✓	✓	922

V8.3 Personenbezogene Daten

Dieser Abschnitt trägt dazu bei, sensible Daten vor unbefugtem Erstellen, Lesen, Ändern oder Löschen zu schützen. Voraussetzung ist die Einhaltung von V4-Zugriffskontrollmaßnahmen, insbesondere V4.2. So erfordert zum Beispiel der Schutz von personenbezogenen Daten vor unbefugten Veränderungen oder Offenlegung die Einhaltung von V4.2.1. Bitte halten Sie sich an diesen Abschnitt und an V4, um eine vollständige Abdeckung zu erreichen.

Hinweis: Datenschutzbestimmungen und -gesetze wie die EU-Datenschutzgrundverordnung oder die Australian Privacy Principles APP-11 wirken sich direkt darauf aus, wie die Speicherung, Nutzung und Übertragung von personenbezogenen Daten in der Anwendung umgesetzt werden muss. Dies reicht von einfachen Ratschlägen bis hin zu schweren Strafen. Bitte prüfen Sie Ihre lokalen Gesetze und Vorschriften und wenden Sie sich bei Bedarf an einen Datenschutzspezialisten.

#	Beschreibung	L1	L2	L3	CWE
8.3.1	Prüfen Sie, dass sensible Daten im HTTP-Textkörper oder in Headern an den Server gesendet werden, und dass die Query-String-Parameter aller HTTP-Requests keine sensiblen Daten enthalten.	✓	✓	✓	319
8.3.2	Prüfen Sie, dass die Benutzer ihre Daten bei Bedarf entfernen oder exportieren können.	✓	✓	✓	212
8.3.3	Prüfen Sie, dass die Benutzer in verständlicher Sprache über die Erfassung und Verwendung der bereitgestellten personenbezogenen Daten informiert werden und dass die Benutzer ihr Einverständnis zur Verwendung dieser Daten gegeben haben, bevor diese verwendet werden.	✓	✓	✓	285
8.3.4	Prüfen Sie, dass alle von der Anwendung erstellten und verarbeiteten personenbezogenen Daten identifiziert wurden und dass eine Regelung für den Umgang mit diesen Daten vorhanden ist. (C8)	✓	✓	✓	200
8.3.5	Prüfen Sie, dass bei Zugriff auf personenbezogene Daten geprüft wird - ohne die Daten selbst zu protokollieren - ob die Daten gemäß den einschlägigen Datenschutzrichtlinien erfasst werden oder ob eine Protokollierung des Zugriffs erforderlich ist.		✓	✓	532
8.3.6	Prüfen Sie, dass die im Speicher enthaltenen Informationen überschrieben werden, sobald sie nicht mehr benötigt werden, um Memory-Dump-Angriffe abzuschwächen.		✓	✓	226
8.3.7	Prüfen Sie, dass zu verschlüsselnde Informationen mit anerkannten Algorithmen verschlüsselt werden, die sowohl Vertraulichkeit als auch Integrität gewährleisten. (C8)		✓	✓	327
8.3.8	Prüfen Sie, dass personenbezogene Daten in Bezug auf die Datenspeicherung klassifiziert werden, so dass alte oder veraltete Daten automatisch, nach einem Zeitplan oder je nach Situation gelöscht werden können.		✓	✓	285

Wenn es um den Datenschutz geht, sollte man in erster Linie an die Massenextraktion oder -modifikation oder an eine übermäßige Nutzung denken. Viele Social-Media-Systeme erlauben beispielsweise nur das Hinzufügen von 100 neuen Freunden pro Tag, aber es ist nicht wichtig, aus welchem System diese Anfragen kommen. Eine Bankingplattform sollte vielleicht mehr als 5 Transaktionen pro Stunde blockieren, welche mehr als 1000 Euro

an externe Institute überweisen. Jedes System hat sehr unterschiedliche Anforderungen. Deshalb muss man bei der Entscheidung, was „abnormal“ ist, das Bedrohungsmodell und das Geschäftsrisiko berücksichtigen. Wichtige Kriterien sind hierbei die Fähigkeit, solche abnormalen Massenaktionen zu erkennen, abzuwenden oder vorzugsweise zu blockieren.

Referenzen

Weitere Informationen finden Sie unter:

- Nutzen Sie die Website securityheaders.com, um Ihre Site zu testen.
- OWASP Secure Headers project
- OWASP Privacy Risks Project
- OWASP User Privacy Protection Cheat Sheet
- Übersicht über die Datenschutz-Grundverordnung der Europäischen Union (DSGVO/ GDPR)
- European Union Data Protection Supervisor - Internet Privacy Engineering Network

V9 Kommunikation

Ziel

Eine verifizierte Anwendung erfüllt die folgenden High Level Anforderungen:

- Unabhängig von der Brisanz der Daten wird bei der Übertragung stets TLS oder eine starke Verschlüsselung verwendet.
- Es werden die aktuellsten Empfehlungen werden verwendet, ins Besondere:
  - Konfigurationen der Algorithmen
  - bevorzugt zu verwendende Algorithmen.
- schwache oder bald veraltende Algorithmen werden am besten gar nicht, höchstens aber mit geringster Priorität genutzt
- Veraltete oder unsichere Algorithmen werden deaktiviert.

Weiterhin sollten Sie stets:

- auf dem aktuellsten Stand bleiben, da sich die Empfehlungen zur sicheren TLS-Konfiguration häufig ändern. Dies kann auch auf Grund von katastrophalen Angriffen auf Algorithmen geschehen.
- die aktuellsten Versionen der TLS-Testtools zur Prüfung der Konfiguration nutzen.
- regelmäßig die Konfiguration testen, um sicherzustellen, dass die Sicherheit der Kommunikation jederzeit gewährleistet ist.

V9.1 Kommunikationssicherheit des Clients

Stellen Sie sicher, dass alle Nahrichten des Clients mit TLS 1.2 oder neuer verschlüsselt werden. Nutzen Sie regelmäßig die aktuellen Testwerkzeuge, um die Clientkonfiguration zu testen.

#	Beschreibung	L1	L2	L3	CWE
9.1.1	Prüfen Sie, dass der Client stets TLS-Verbindungen verwendet, das nicht auf unsichere oder unverschlüsselte Konfigurationen zurückfallen. (C8)	✓	✓	✓	319
9.1.2	Prüfen Sie mit aktuellen TLS-Testtools, dass nur starke Algorithmen und Protokolle genutzt werden. Dabei sind die stärksten Algorithmen und neuesten Protokollversionen zu bevorzugen.	✓	✓	✓	326
9.1.3	Prüfen Sie, dass nur die aktuell empfohlenen Versionen der TLS-Protokolle, also TLS 1.2 und TLS 1.3, genutzt werden. Die neueste Version ist dabei zu bevorzugen.	✓	✓	✓	326

V9.2 Sicherheit der Serverkommunikation

Serverkommunikation ist mehr als nur HTTP. Es müssen sichere Verbindungen zu und von anderen Systemen - wie Überwachungssysteme, Managementtools, Fernzugriff und ssh, Middleware, Datenbank, Mainframes, Partner- oder externe Quellsysteme eingerichtet werden. All diese müssen verschlüsselt werden, um zu verhindern, dass sie nach außen schwer aber nach innen kinderleicht abzufangen sind.

#	Beschreibung	L1	L2	L3	CWE
9.2.1	Prüfen Sie, dass Verbindungen zum und vom Server vertrauenswürdige TLS-Zertifikate verwenden. Werden intern generierte oder selbstsignierte Zertifikate verwendet, muss der Server so konfiguriert werden, dass er nur bestimmten internen CAs vertraut. Alle anderen müssen abgelehnt werden.		✓	✓	295

#	Beschreibung	L1	L2	L3	CWE
9.2.2	Prüfen Sie, dass eine verschlüsselte Kommunikation wie TLS für alle ein- und ausgehenden Verbindungen, einschließlich für Managementports, Überwachung, Authentifizierung, API- oder Webservicecalls, Datenbank-, Cloud-, serverlose, Mainframe-, externe und Partnerverbindungen verwendet wird. Der Server darf nicht auf unsichere oder unverschlüsselte Protokolle zurückgreifen.		✓	✓	319
9.2.3	Prüfen Sie, dass alle verschlüsselten Verbindungen zu externen Systemen, die sensible Informationen oder Funktionen beinhalten, authentifiziert sind.		✓	✓	287
9.2.4	Prüfen Sie, dass eine ordnungsgemäßer Zertifikatssperre wie z. B. das Online Certificate Status Protocol Stapling aktiviert und konfiguriert ist.		✓	✓	299
9.2.5	Prüfen Sie, dass TLS-Verbindungsfehler in das Backend protokolliert werden.			✓	544

Referenzen

Weitere Informationen finden Sie unter:

- OWASP – TLS Cheat Sheet
- OWASP - Pinning Guide
- Hinweise zu „Anerkannten TLS-Modi”:
  - In der Vergangenheit bezog sich der ASVS auf den US-Standard FIPS 140. Als globaler Standard kann die Anwendung von US-Standards allerdings schwierig, widersprüchlich oder verwirrend sein.
  - Eine bessere Methode, um die Einhaltung des Abschnittes 9.1 zu erreichen, wäre die Überprüfung von Leitfäden wie Mozillas serverseitigem TLS oder die Erstellung anerkannt sicherer Konfigurationen und die Verwendung bekannter TLS-Evaluierungs-Tools wie Sslyze, verschiedener Schwachstellenscanner oder vertrauenswürdiger TLS-Online-Assessment Services, um ein gewünschtes Sicherheitsniveau zu erreichen. In Sicherheitstests sehen wir die fehlende Konformität zu diesem Abschnitt durch die Verwendung veralteter oder unsicherer Algorithmen, dem Fehlen einer perfekten Forward Secrecy, veralteten oder unsicheren SSL-Protokollen, schwachen bevorzugten Algorithmen usw.
  - Für Deutschland veröffentlicht das BSI die Technische Richtlinie TR-02102 Kryptographische Verfahren: Empfehlungen und Schlüssellängen als Richtlinie für die Verwendung von TLS sowie zur Nutzung sicherer Algorithmen und Schlüssellängen.

V10 Bösartiger Code

Ziel

Stellen Sie sicher, dass der Code die folgenden High Level Anforderungen erfüllt:

- Böswillige Aktivitäten werden sicher und ordnungsgemäß behandelt, um den Rest der Anwendung nicht zu beeinträchtigen.
- Es gibt keine Zeitbomben oder andere zeitbasierte Angriffe.
- Es gibt kein „Phone Home“ zu böswilligen oder nicht autorisierten Zielen.
- Es gibt keine Hintertüren, Ostereier, Salamitaktik-Angriffe, Rootkits oder nicht autorisierte Codes, die von einem Angreifer kontrolliert werden können.

Das Feststellen der Abwesenheit bösartigen Codes ist der Negativbeweis, der unmöglich vollständig geführt werden kann. Es sollten alle Anstrengungen unternommen werden, um sicherzustellen, dass die Software keinen inhärenten bösartigen Code und keine unerwünschten Funktionen aufweist.

V10.1 Kontrollen der Code-Integrität

Die beste Verteidigung gegen bösartigen Code ist der Wahlspruch „Vertrauen ist gut, Kontrolle ist besser“. Das Einbringen eines nicht autorisierten oder bösartigen Codes in eine Software ist in vielen Rechtsordnungen ein kriminelles Vergehen. Die Richtlinien zur Softwareentwicklung sollten auf die Sanktionen für das Einbringen bösartigen Codes deutlich hinweisen. Leitende Entwickler sollten regelmäßig die Eincheckvorgänge für den Code überprüfen, insbesondere diejenigen, die auf Zeit-, Eingabe-, Ausgabe- oder Netzwerkfunktionen zugreifen können.

#	Beschreibung	L1	L2	L3	CWE
10.1.1	Prüfen Sie, dass ein Codeanalyse-Tool verwendet wird, das potenziell bösartigen Code, wie Zeitfunktionen, unsichere Dateioperationen und Netzwerkverbindungen erkennen kann.			✓	749

V10.2 Suche nach bösartigem Code

Bösartiger Code ist extrem selten und schwer zu erkennen. Eine manuelle, zeilenweise Überprüfung des Codes kann bei der Suche nach Logikbomben helfen, aber selbst die erfahrensten Codereviewer werden Schwierigkeiten haben, bösartigen Code zu finden, selbst wenn sie wissen, dass er existiert. Die Einhaltung dieses Abschnitts kann nicht ohne vollständigen Zugriff auf den Quellcode, einschließlich der Bibliotheken von Drittanbietern, geprüft werden.

#	Beschreibung	L1	L2	L3	CWE
10.2.1	Prüfen Sie, dass der Quellcode der Anwendung und die Bibliotheken von Drittanbietern keine Möglichkeiten zum Phone Home oder zur Datenerfassung enthalten. Wenn solche Funktionen vorhanden sind, holen Sie die Erlaubnis des Benutzers ein, bevor Daten gesammelt werden.		✓	✓	359
10.2.2	Prüfen Sie, dass die Anwendung keine unnötigen oder übermäßigen Genehmigungen für datenschutzrelevante Funktionen oder Sensoren wie Kontakte, Kameras, Mikrofone oder Standorte verlangt.		✓	✓	272
10.2.3	Prüfen Sie, dass der Quellcode der Anwendung und die Bibliotheken von Drittanbietern keine Hintertüren enthalten, wie z. B. hartcodierte oder zusätzliche undokumentierte Konten oder Schlüssel, Codeverschleierung, undokumentierte Binärblobs, Rootkits oder Anti-Debugging, unsichere Debuggingfunktionen oder andere veraltete, unsichere oder versteckte Funktionen, die bei Entdeckung böswillig verwendet werden könnten.			✓	507

#	Beschreibung	L1	L2	L3	CWE
10.2.4	Prüfen Sie, dass der Quellcode der Anwendung und die Bibliotheken von Drittanbietern keine Zeitbomben enthalten, wenn sie nach datums- und zeitbezogenen Funktionen suchen.			✓	511
10.2.5	Prüfen Sie, dass der Quellcode der Anwendung und die Bibliotheken von Drittanbietern keinen bösartigen Code wie Salamitaktik-Angriffe, logische Umgehungen oder Logikbomben enthalten.			✓	511
10.2.6	Prüfen Sie, dass der Quellcode der Anwendung und die Bibliotheken von Drittanbietern keine Ostereier oder andere unerwünschte Funktionen enthalten.			✓	507

V10.3 Integrität der Anwendung

Nach Bereitstellung einer Anwendung kann immer noch bösartiger Code eingefügt werden. Anwendungen müssen sich vor gängigen Angriffen wie die Ausführung unsignierten Codes aus nicht vertrauenswürdigen Quellen und die Übernahme von Subdomänen schützen. Die Einhaltung der Anforderungen dieses Abschnitts sind operativ und fortlaufend zu prüfen.

#	Beschreibung	L1	L2	L3	CWE
10.3.1	Prüfen Sie, dass Updates über sichere Kanäle bezogen und digital signiert werden müssen, wenn die Anwendung über eine automatische Client- oder Server Updatefunktion verfügt. Die digitale Signatur des Updates muss validiert werden, bevor das Update installiert wird.	✓	✓	✓	16
10.3.2	Prüfen Sie, dass die Anwendung einen Integritätsschutz, wie Code Signing oder Subresource Integrity verwendet. Die Anwendung darf keinen Code aus nicht vertrauenswürdigen Quellen laden oder ausführen, wie z. B. das Laden von Includes, Modulen, Plugins, Codes oder Bibliotheken aus nicht vertrauenswürdigen Quellen oder dem Internet.	✓	✓	✓	353
10.3.3	Prüfen Sie, ob die Anwendung Schutz vor der Übernahme von Subdomänen bietet, wenn die Anwendung auf DNS-Einträge oder DNS-Subdomänen angewiesen ist, z. B. abgelaufene Domänennamen, veraltete DNS-Pointer oder CNAMEs, abgelaufene Projekte in öffentlichen Quellcoderepositories oder vorübergehende Cloud-APIs, serverlose Funktionen oder Storage Buckets (autogen-bucket-id.cloud.example.com) oder Ähnliches. Die von den Anwendungen verwendeten DNS-Namen sind regelmäßig auf Ablauf oder Änderung zu überprüfen.	✓	✓	✓	350

Referenzen

Weitere Informationen finden Sie unter:

- Hostile Subdomain Takeover, Detectify Labs
- Hijacking of abandoned subdomains part 2, Detectify Labs

V11 Fachliche Funktionalität

Ziel

Stellen Sie sicher, dass eine verifizierte Anwendung die folgenden High Level Anforderungen erfüllt:

- Der Fluss der Geschäftslogik ist sequentiell: Er wird der Reihe nach verarbeitet und kann nicht umgangen werden.
- Die Geschäftslogik enthält Grenzen zur Erkennung und Verhinderung automatisierter Angriffe, wie z. B. kontinuierliche kleine Geldtransfers oder das Hinzufügen von einer Million Freunden, einer nach dem anderen, etc.
- Hochwertige Modelle der Geschäftslogik haben Missbrauchsfälle und böswillige Akteure in Betracht gezogen und sind gegen Spoofing, Manipulation, Abstreiten, Informationspreisgabe und die Erweiterung von Rechten immun.

V11.1 Sicherheit der fachlichen Funktionen

Die Sicherheit der Geschäftslogik ist für jede Anwendung zu individuell, als dass dafür jemals eine einzige Checkliste erstellt werden könnte. Die Sicherheit der Geschäftslogik muss so konzipiert sein, dass sie vor voraussichtlichen externen Bedrohungen Schutz bietet - sie kann nicht durch Webanwendungsfirewalls oder sichere Kommunikation hinzugefügt werden. Wir empfehlen die Verwendung von Threat Modeling während der Designsprints, z.B. mit dem OWASP Cornucopia oder ähnlichen Tools.

#	Beschreibung	L1	L2	L3	CWE
11.1.1	Prüfen Sie, dass die Anwendung nur Geschäftslogikflüsse für denselben Benutzer in sequentieller Schrittfolge und ohne das Überspringen von Schritten verarbeitet.	✓	✓	✓	841
11.1.2	Prüfen Sie, dass die Anwendung nur Abläufe der Geschäftslogik verarbeitet, wenn alle Schritte in realistischer menschlicher Zeit bearbeitet werden, d.h. die Transaktionen werden nicht zu schnell durch automatisierte Angreifer eingereicht.	✓	✓	✓	799
11.1.3	Prüfen Sie, dass die Anwendung über angemessene Grenzen für bestimmte Geschäftsaktionen oder Transaktionen verfügt, die für jeden Benutzer korrekt durchgesetzt werden.	✓	✓	✓	770
11.1.4	Prüfen Sie, dass die Anwendung über ausreichende Maßnahmen gegen automatische Nutzung verfügt, um Datenausleitung, übermäßige Anforderungen an die Geschäftslogik, übermäßige Dateiuploads oder DoS-Angriffe zu erkennen und sich dagegen zu schützen.	✓	✓	✓	770
11.1.5	Prüfen Sie, ob die Anwendung Grenzen der Geschäftslogik oder eine Validierung zum Schutz vor wahrscheinlichen Geschäftsrisiken oder Bedrohungen aufweist, die mit Hilfe von Threat Modeling oder ähnlichen Methoden ermittelt wurden.	✓	✓	✓	841
11.1.6	Prüfen Sie, dass die Anwendung nicht unter TOCTOU oder anderen Raceconditions für sensible Operationen leidet.		✓	✓	367
11.1.7	Prüfen Sie die Anwendungsmonitore auf ungewöhnliche Ereignisse oder Aktivitäten aus der Sicht der Geschäftslogik. Zum Beispiel auf Versuche, Aktionen durchzuführen, die außerhalb der Reihe sind, oder Aktionen, die ein normaler Benutzer niemals versuchen würde. (C9)		✓	✓	754

#	Beschreibung	L1	L2	L3	CWE
11.1.8	Prüfen Sie, dass die Anwendung über konfigurierbare Web Warnmeldungen verfügt, wenn automatisierte Angriffe oder ungewöhnliche Aktivitäten entdeckt werden.		✓	✓	390

Referenzen

Weitere Informationen finden Sie unter:

- OWASP Web Security Testing Guide 4.1: Business Logic Testing
- Schutz gegen automatische Angriffe kann auf vielfältige Weise erreicht werden, z.B.: OWASP AppSensor and OWASP Automated Threats to Web Applications
- Der OWASP AppSensor kann auch bei der Erkennung und der Bewältigung von Angriffen helfen.
- OWASP Cornucopia

V12 Dateien und andere Ressourcen

Ziel

Stellen Sie sicher, dass eine verifizierte Anwendung die folgenden High Level Anforderungen erfüllt:

- nicht vertrauenswürdige Datendateien müssen angepasst und auf sichere Weise behandelt werden.
- nicht vertrauenswürdige Datendateien, die aus nicht vertrauenswürdigen Quellen stammen, werden außerhalb des Webroots und mit eingeschränkten Berechtigungen gespeichert.

V12.1 Dateiupload

Obwohl Zip-Bomben hervorragend mit Penetrationstests getestet werden können, werden sie als L2 und höher eingestuft, um die Berücksichtigung von Design und Entwicklung durch sorgfältige manuelle Tests zu fördern und einen ungewollten DoS-Angriff durch automatische oder unqualifizierte manuelle Penetrationstests zu vermeiden.

#	Beschreibung	L1	L2	L3	CWE
12.1.1	Prüfen Sie, dass die Anwendung keine großen Dateien akzeptiert, die den Speicher füllen oder einen DoS-Angriff verursachen könnten.	✓	✓	✓	400
12.1.2	Prüfen Sie, dass die Anwendung gepackte Formate, wie z.B. zip, gz, docx oder odt vor dem Entpacken auf die maximal zulässige Filegröße und die maximale Anzahl Dateien überprüft.		✓	✓	409
12.1.3	Prüfen Sie, dass die Dateigröße und die maximale Anzahl von Dateien pro Benutzer limitiert wird, um sicherzustellen, dass ein einzelner Benutzer den Speicher nicht mit zu vielen oder übermäßig großen Dateien füllen kann.		✓	✓	770

V12.2 Dateiintegrität

#	Beschreibung	L1	L2	L3	CWE
12.2.1	Prüfen Sie, dass Dateien aus nicht vertrauenswürdigen Quellen sowohl auf der Grundlage des Dateiinhalts als auch des erwarteten Typs validiert werden.		✓	✓	434

V12.3 Ausführbare Dateien

#	Beschreibung	L1	L2	L3	CWE
12.3.1	Prüfen Sie, dass die vom Benutzer eingereichten Metadaten der Dateinamen nicht direkt vom Filesystem des Betriebssystems oder des Frameworks genutzt werden. Weiterhin ist eine URL-API zu verwenden, um vor Path Traversal zu schützen.	✓	✓	✓	22
12.3.2	Prüfen Sie, dass die vom Benutzer eingereichten Metadaten der Dateinamen validiert oder ignoriert werden, um die Offenlegung, Erstellung, Aktualisierung oder Entfernung lokaler Dateien zu verhindern.	✓	✓	✓	73
12.3.3	Prüfen Sie, dass die vom Benutzer eingereichten Metadaten der Dateinamen validiert oder ignoriert werden, um die Offenlegung oder Ausführung von serverseitigen Dateien via Remote File Inclusion (RFI) oder Serverside Request Forgery (SSRF) Angriffen zu verhindern.	✓	✓	✓	98

#	Beschreibung	L1	L2	L3	CWE
12.3.4	Prüfen Sie, dass die Anwendung vor Reflective File Download (RFD) schützt, indem sie die vom Benutzer eingereichten Dateinamen in einem JSON-, JSONP- oder URL-Parameter validiert oder ignoriert. Der Content-Type Header der Antwort muss auf text/plain gesetzt werden, und der Content-Disposition Header muss einen festen Dateinamen haben.	✓	✓	✓	641
12.3.5	Prüfen Sie, dass nicht vertrauenswürdige Dateimetadaten nicht direkt mit der System-API oder Bibliotheken verwendet werden, um vor OS Command Injection zu schützen.	✓	✓	✓	78
12.3.6	Prüfen Sie, dass die Anwendung keine Funktionen aus nicht vertrauenswürdigen Quellen, wie z. B. nicht verifizierte Inhaltsverteilungsnetzwerke, JavaScript-Bibliotheken, node npm-Bibliotheken oder serverseitige DLLs, enthält und ausführt.		✓	✓	829

V12.4 Speicherung von Dateien

#	Beschreibung	L1	L2	L3	CWE
12.4.1	Prüfen Sie, dass Dateien aus nicht vertrauenswürdigen Quellen mit eingeschränkten Berechtigungen und vorzugsweise mit starker Validierung außerhalb des Webroots gespeichert werden.	✓	✓	✓	552
12.4.2	Prüfen Sie, dass Dateien aus nicht vertrauenswürdigen Quellen von Antivirenscannern gescannt werden, um das Hochladen bekannter bösartiger Inhalte zu verhindern.	✓	✓	✓	509

V12.5 Download von Dateien

#	Beschreibung	L1	L2	L3	CWE
12.5.1	Prüfen Sie, dass die Webschicht so konfiguriert ist, dass nur Dateien mit bestimmten Dateierweiterungen bedient werden, um unbeabsichtigte Informations- und Quellcodelecks zu vermeiden. Beispielsweise sollten Sicherungsdateien (z.B. .bak), temporäre Arbeitsdateien (z.B. .swp), komprimierte Dateien (.zip, .tar, .gz, usw.) und andere von den Editoren üblicherweise verwendete Erweiterungen blockiert werden, sofern sie nicht erforderlich sind.	✓	✓	✓	552
12.5.2	Prüfen Sie, dass direkte Anfragen an hochgeladene Dateien niemals als HTML/JavaScript-Inhalt ausgeführt werden.	✓	✓	✓	434

V12.6 SSRF-Schutz

#	Beschreibung	L1	L2	L3	CWE
12.6.1	Prüfen Sie, dass der Web- oder Anwendungsserver mit einer Whitelist von Ressourcen oder Systemen konfiguriert ist, an die der Server Anfragen senden oder Daten/Dateien laden kann.	✓	✓	✓	918

Referenzen

Weitere Informationen finden Sie unter:

- File Extension Handling for Sensitive Information
- Reflective file download by Oren Hafif

- OWASP Third Party JavaScript Management Cheat Sheet

V13 API and Web Service

Ziel

Prüfen Sie, dass eine verifizierte Anwendung, die vertrauenswürdige Serviceschicht-APIs verwendet - üblicherweise unter Verwendung von JSON oder XML oder GraphQL - über Folgendes verfügt:

- Angemessene Authentifizierung, Session Management und Autorisierung aller Webdienste.
- Eingabeprüfung aller Parameter, die von einer niedrigeren auf eine höhere Vertrauensstufe übergehen.
- Effektive Sicherheitsmaßnahmen für alle API-Typen, einschließlich Cloud- und Serverlose API

Bitte lesen Sie dieses Kapitel in Kombination mit allen anderen Kapiteln derselben Stufe. Themen, wie z.B. Authentifizierung oder API-Session Management, werden hier nicht erneut aufgeführt.

V13.1 Allgemeine Sicherheit von Web Services

#	Beschreibung	L1	L2	L3	CWE
13.1.1	Prüfen Sie, dass alle Komponenten die gleichen Parser und (Zeichen-)Codierungen nutzen, um Angriffe auf Basis unterschiedlichen URI- oder File-Parsings, wie SSRF oder RFI, zu verhindern.	✓	✓	✓	116
13.1.2	[GELÖSCHT, DUPLIKAT VON 4.3.1]				
13.1.3	Prüfen Sie, dass API-URLs keine sensiblen Informationen wie den API-Schlüssel, Sessiontoken, etc. preisgeben.	✓	✓	✓	598
13.1.4	Prüfen Sie, dass Berechtigungsentscheidungen sowohl an der URI, umgesetzt durch programmatische oder deklarative Sicherheit am Controller oder Router, als auch auf der Ressourcenebene, umgesetzt durch modellbasierte Berechtigungen, getroffen werden.		✓	✓	285
13.1.5	Prüfen Sie, dass Anfragen mit unerwarteten oder fehlenden Inhaltstypen mit entsprechenden Headern zurückgewiesen werden (HTTP-Antwortstatus 406 oder 415).		✓	✓	434

V13.2 RESTful Web Services

Die Validierung des JSON-Schemas befindet sich im Entwurfsstadium der Standardisierung (siehe Referenzen). Wenn Sie die Verwendung der JSON-Schemavalidierung in Betracht ziehen, welche die Best Practice für RESTful-Webservices darstellt, sollten Sie die Verwendung dieser zusätzlichen Datenvalidierungsstrategien in Kombination mit der JSON-Schemavalidierung in Betracht ziehen:

- Parsing-Validierung des JSON-Objekts, z. B. ob es fehlende oder zusätzliche Elemente gibt.
- Validierung der Werte des JSON-Objekts mit Hilfe der üblichen Eingabeprüfungsmethoden, wie z. B. Prüfung auf Datentyp, Datenformat, Länge usw. und
- formale JSON-Schemavalidierung.

Sobald der JSON-Schemavalidierungsstandard formalisiert ist, wird der ASVS seine Empfehlungen in diesem Bereich aktualisieren. Alle verwendeten JSON-Schemavalidierungsbibliotheken müssen sorgfältig überwacht werden, da sie regelmäßig aktualisiert werden müssen, bis der Standard formalisiert ist und Fehler in den Referenzimplementierungen behoben sind.

#	Beschreibung	L1	L2	L3	CWE
13.2.1	Prüfen Sie, dass aktivierte RESTful-HTTP-Methoden eine gültige Wahl für den Benutzer oder eine Aktion sind, wie z.B. verhindern, dass normale Benutzer DELETE oder PUT auf geschützte API oder Ressourcen anwenden.	✓	✓	✓	650
13.2.2	Prüfen Sie, dass die JSON-Schemavalidierung vorhanden und verifiziert ist, bevor Sie eine Eingabe akzeptieren.	✓	✓	✓	20
13.2.3	Prüfen Sie, dass RESTful-Webdienste, die Cookies verwenden, durch die Verwendung von mindestens einem oder mehrerer der folgenden Verfahren vor Cross Site Request Forgery geschützt sind: Double Submit Cookie Pattern, CSRF-Nonces oder Prüfungen des Origin-Request Headers.	✓	✓	✓	352
13.2.4	[GELÖSCHT, DUPLIKAT VON 11.1.4]				
13.2.5	Prüfen Sie, dass die REST-Dienste explizit prüfen, ob der eingehende Contenttyp der erwartete ist, z.B. application/xml oder application/json.		✓	✓	436
13.2.6	Prüfen Sie, dass die Message Header und die Nutzdaten vertrauenswürdig sind und während der Übertragung nicht verändert werden. Die Anforderung einer starken Verschlüsselung für den Transport (nur TLS) kann in vielen Fällen ausreichend sein, da sie sowohl die Vertraulichkeit als auch den Schutz der Integrität gewährleistet. Digitale Signaturen pro Nachricht können bei Hochsicherheitsanwendungen für zusätzliche Sicherheit beim Transportschutz sorgen, bringen aber zusätzliche Komplexität und Risiken mit sich, die gegen die Vorteile abzuwägen sind.		✓	✓	345

V13.3 SOAP Web Service

#	Beschreibung	L1	L2	L3	CWE
13.3.1	Prüfen Sie, dass vor der Verarbeitung von Eingabedaten zuerst eine XSD-Schemavalidierung stattfindet, um ein korrekt geformtes XML-Dokument zu gewährleisten, gefolgt von der Validierung jedes Eingabefeldes.	✓	✓	✓	20
13.3.2	Prüfen Sie, dass die Nutzdaten der Nachricht mit WS-Security signiert sind, um einen zuverlässigen Transport zwischen Client und Service zu gewährleisten.		✓	✓	345

Hinweis: Aufgrund von Problemen mit XXE-Angriffen auf DTDs sollte die DTD-Validierung nicht verwendet und die DTD-Frameworkauswertung gemäß der im Abschnitt V14-Konfiguration festgelegten Anforderungen deaktiviert werden.

V13.4 GraphQL

#	Beschreibung	L1	L2	L3	CWE
13.4.1	Prüfen Sie, dass eine Query Whitelist oder eine Kombination von Begrenzung der Tiefe und Anzahl verwendet werden sollte, um einen DoS-Angriff von GraphQL oder Datenschichtausdrücken als Folge teurer, verschachtelter Abfragen zu verhindern. Für fortgeschrittenere Szenarien sollte die Abfragekostenanalyse verwendet werden.		✓	✓	770
13.4.2	Prüfen Sie, dass die Berechtigungen für die Datenschicht, z.B. GraphQL, in der Geschäftslogikschicht anstelle der Datenschicht umgesetzt ist.		✓	✓	285

Referenzen

Weitere Informationen finden Sie unter:

- OWASP Serverless Top 10
- OWASP Serverless Project
- OWASP Testing Guide 4.0: Configuration and Deployment Management Testing
- OWASP Cross-Site Request Forgery cheat sheet
- OWASP XML External Entity Prevention Cheat Sheet - General Guidance
- JSON Web Tokens (and Signing)
- REST Security Cheat Sheet
- JSON Schema
- XML DTD Entity Attacks
- Orange Tsai - A new era of SSRF Exploiting URL Parser In Trending Programming Languages

V14 Konfiguration

Ziel

Stellen Sie sicher, dass eine geprüfte Anwendung mit den folgenden Merkmalen ausgestattet ist:

- Eine sichere und automatisierbare Buildumgebung.
- Ein sicheres Management von Drittanbieterbibliotheken, Abhängigkeiten und Konfigurationen, so dass veraltete oder unsichere Komponenten nicht in die Anwendung integriert werden.

Die Konfiguration der Anwendung sollte „out of the box“ sicher für den jeweils geplanten Einsatzzweck sein.

V14.1 Build- und Deployprozess

Build Pipelines sind die Grundlage für wiederholbare Sicherheit. Jedes Mal, wenn etwas Unsicheres entdeckt wird, kann es im Quellcode, den Build- oder Deploymentskripten behoben und automatisch getestet werden. Wir befürworten nachdrücklich die Verwendung von Buildpipelines mit automatischen Sicherheits- und Abhängigkeitsprüfungen, die den Build ggf. unterbrechen, um zu verhindern, dass bekannte Sicherheitsprobleme in der Produktion angewendet werden. Unregelmäßig durchgeführte manuelle Schritte führen direkt zu vermeidbaren Sicherheitsfehlern.

Da die Branche zu einem DevSecOps-Modell übergeht, ist es wichtig, die kontinuierliche Verfügbarkeit und Integrität der Bereitstellung und Konfiguration zu gewährleisten, um einen bekannten funktionierenden (known good) Zustand zu erreichen. In der Vergangenheit dauerte es Tage bis Monate, wenn ein System gehackt wurde, um nachzuweisen, dass kein weiteres Eindringen stattgefunden hatte. Heute, mit dem Aufkommen der softwaredefinierten Infrastruktur, schnellen A/B-Deployment ohne Ausfallzeiten und automatisierten, containerisierten Builds, ist es möglich, automatisch und kontinuierlich einen „known good“-Ersatz für jedes kompromittierte System zu erstellen, zu härten und einzusetzen. Sind noch traditionelle Modelle vorhanden, müssen manuelle Schritte zur Härtung und Sicherung dieser Konfiguration erfolgen, damit die kompromittierten Systeme schnell durch hochintegrierte, sichere Systeme ersetzt werden können.

Um die Anforderungen dieses Abschnitts einzuhalten, ist ein automatisiertes Buildsystem und der Zugriff auf Build- und Deploymentskripte erforderlich.

#	Beschreibung	L1	L2	L3	CWE
14.1.1	Prüfen Sie, dass die Build- und Deploymentprozesse auf sichere und wiederholbare Weise durchgeführt werden, z. B. durch CI-/CD-Automatisierung, automatisiertes Konfigurationsmanagement und automatisierte Deploymentskripte.		✓	✓	
14.1.2	Prüfen Sie, dass die Compilerflags so konfiguriert sind, dass sie alle verfügbaren Pufferüberlaufschutzmechanismen und Warnungen aktivieren, einschließlich der Stackrandomisierung, der Verhinderung der Datenausführung und des Buildabbruchs, wenn ein(e) unsichere(r) Pointer, Speicher, Formatstring, Integer- oder Stringoperationen gefunden wird.		✓	✓	120
14.1.3	Prüfen Sie, dass die Serverkonfiguration gemäß den Empfehlungen des verwendeten Anwendungsservers und Frameworks gehärtet wird.		✓	✓	16
14.1.4	Prüfen Sie, dass die Anwendung, die Konfiguration und alle Abhängigkeiten mit Hilfe automatisierter Deploymentskripte wieder installiert werden können, indem sie innerhalb eines angemessenen Zeitraums aus einem dokumentierten und getesteten Runbook erstellt oder aus Backups zeitnah wiederhergestellt werden können.		✓	✓	
14.1.5	Prüfen Sie, dass autorisierte Administratoren die Integrität aller sicherheitsrelevanten Konfigurationen überprüfen können, um Manipulationen zu erkennen.			✓	

V14.2 Management von Abhängigkeiten

Die Verwaltung von Abhängigkeiten ist für den sicheren Betrieb jeder Art von Anwendung von entscheidender Bedeutung. Das Versäumnis, veraltete oder unsichere Abhängigkeiten auf dem aktuellen Stand zu halten, ist die eigentliche Ursache für die bisher größten und teuersten Angriffe.

Hinweis: Auf Stufe 1 bezieht sich die Einhaltung von 14.2.1 auf Beobachtungen oder Feststellungen von Client- und anderen Bibliotheken und Komponenten und nicht auf die genauere statische Codeanalyse oder Abhängigkeitsanalyse zur Buildzeit. Diese genaueren Techniken könnten bei Bedarf durch Befragung festgestellt werden.

#	Beschreibung	L1	L2	L3	CWE
14.2.1	Prüfen Sie, dass alle Komponenten auf dem neuesten Stand sind, am besten mit einem Abhängigkeitsprüfer zur Build- oder Kompilierzeit. (C2)	✓	✓	✓	1026
14.2.2	Prüfen Sie, dass alle nicht benötigten Funktionen, Dokumentationen, Beispiele und Konfigurationen entfernt werden.	✓	✓	✓	1002
14.2.3	Prüfen Sie, dass die Integrität externen Inhaltes durch Subresource Integrity (SRI) überprüft wird, wenn Anwendungsassets wie JavaScript-Bibliotheken, CSS oder Web-Fonts extern, z.B. bei einem Content Delivery Network oder bei einem externen Anbieter, gehostet werden.	✓	✓	✓	829
14.2.4	Prüfen Sie, dass Komponenten Dritter aus bekannten, vertrauenswürdigen und kontinuierlich gepflegten Repositories stammen. (C2)		✓	✓	829
14.2.5	Prüfen Sie, dass eine Softwarestückliste (Bill of Materials, SBOM) aller genutzten Bibliotheken von Drittanbietern geführt wird. (C2)		✓	✓	
14.2.6	Prüfen Sie, dass die Angriffsfläche durch Sandboxing oder Einkapselung von Bibliotheken von Drittanbietern reduziert wird, damit die Anwendung nur die erforderliche Funktionalität erhält. (C2)		✓	✓	265

V14.3 Offenlegung von Informationen

Konfigurationen für die Produktion sollten gehärtet werden. So schützen sie gegen gängige Angriffe wie Debugkonsolen, und legen die Messlatte für Cross Site Scripting (XSS) und Remote File Inclusion (RFI) Angriffe höher. Triviale Schwachstellen bei der Informationserfassung, die das unwillkommene Kennzeichen vieler Penetrationstestreports sind, werden so beseitigt. Viele dieser Probleme werden selten als signifikantes Risiko eingestuft, sondern sind mit anderen Schwachstellen verkettet. Wenn diese Probleme nicht standardmäßig vorhanden sind, wird die Messlatte für einen erfolgreichen Angriff deutlich höher gelegt.

#	Beschreibung	L1	L2	L3	CWE
14.3.1	[GELÖSCHT, DUPLIKAT VON 7.4.1]				
14.3.2	Prüfen Sie, dass die Debugmodi von Web- und Anwendungsserver sowie Anwendungsframework in der Produktion deaktiviert sind, um Sicherheitslücken durch Debugfunktionen oder Entwicklerkonsolen zu vermeiden.	✓	✓	✓	497
14.3.3	Prüfen Sie, dass die HTTP-Header und HTTP-Antworten keine detaillierten Versionsinformationen von Systemkomponenten enthalten.	✓	✓	✓	200

V14.4 HTTP Security Header

#	Beschreibung	L1	L2	L3	CWE
14.4.1	Prüfen Sie, dass jede HTTP-Antwort einen Content Type Header enthält. Für die Content types text/*, /+xml oder application/xml sollten ein sicherer Zeichensatz (z. B. UTF-8, ISO 8859-1) angeben sein. Der Inhalt muss zum angegebenen Content Type Header passen.	✓	✓	✓	173
14.4.2	Prüfen Sie, dass alle API-Antworten die Content-Disposition: attachment; filename=„api.json“ Header oder einen anderen geeigneten Dateinamen für den Inhaltstyp enthalten.	✓	✓	✓	116
14.4.3	Prüfen Sie, dass ein Content Security Policy (CSP) Response Header vorhanden ist, die dazu beiträgt, die Auswirkungen von XSS-Angriffen wie HTML-, DOM-, JSON- und JavaScript-Injektionsschwachstellen abzuschwächen.	✓	✓	✓	1021
14.4.4	Prüfen Sie, dass alle Antworten X-Content-Type-Optionen: nosniff Header enthalten.	✓	✓	✓	116
14.4.5	Prüfen Sie, dass ein HTTP Strict-Transport-Security Header in allen Antworten und für alle Unterdomänen enthalten ist, z. B. Strict-Transport-Security: max-age=15724800; includeSubdomains.	✓	✓	✓	523
14.4.6	Prüfen Sie, dass ein geeigneter Referrer-Policy Header enthalten ist, um das Veröffentlichen sensibler Informationen über den Referer Header zu vermeiden.	✓	✓	✓	116
14.4.7	Prüfen Sie, dass der Inhalt einer Webanwendung nicht standardmäßig in Seiten Dritter eingebunden werden kann. Das Einbinden der exakten Ressourcen ist nur erlaubt, wenn nötig. Dabei sind passende Content-Security-Policy: frame-ancestors und X-Frame-Options Response Header zu nutzen.	✓	✓	✓	1021

V14.5 Prüfung der HTTP Request Header

#	Beschreibung	L1	L2	L3	CWE
14.5.1	Prüfen Sie, dass der Anwendungsserver nur die von der Anwendung oder der API verwendeten HTTP-Methoden akzeptiert, einschließlich der Pre-Flight-OPTIONS. Alle ungültigen Request sollten ins Log geschrieben werden oder einen Alarm auslösen.	✓	✓	✓	749
14.5.2	Prüfen Sie, dass der bereitgestellte Origin Header nicht für Authentifizierungs- oder Zugriffskontrollentscheidungen verwendet wird, da der Origin Header von einem Angreifer leicht geändert werden kann.	✓	✓	✓	346
14.5.3	Prüfen Sie, dass der CORS-Access-Control-Allow-Origin Header eine strikte Whitelist mit vertrauenswürdigen Domains verwendet und den „Null“-Ursprung nicht unterstützt.	✓	✓	✓	346
14.5.4	Prüfen Sie, dass HTTP-Header, die von einem vertrauenswürdigen Proxy oder SSO-Geräten, wie z. B. einem Bearer-Token, hinzugefügt wurden, von der Anwendung authentifiziert werden.		✓	✓	306

Referenzen

Weitere Informationen finden Sie unter:

- OWASP Web Security Testing Guide 4.1: Testing for HTTP Verb Tampering
- Reflected File Download attacks zeigt, wie durch Hinzufügen der Content-Disposition und der "filename" Option in die API-Antwort viele Angriffe, die auf verschiedenen Interpretationen des MIME-Types durch Client und Server beruhen, unterbunden werden können.
- Content Security Policy Cheat Sheet
- Exploiting CORS misconfiguration for BitCoins and Bounties
- OWASP Web Security Testing Guide 4.1: Configuration and Deployment Management Testing
- Sandboxing third party components

Anhang A: Glossar

- **2FA: Zwei-Faktor-Authentifizierung** - fügt eine zweite Authentifizierungsebene zu einer Kontoanmeldung hinzu, z.B. neben einer PIN („Wissen“) auch eine Chipkarte („Besitz“).
- **Anwendungssicherheit** - Sicherheit auf Anwendungsebene konzentriert sich auf die Analyse von Komponenten, welche die Anwendungsschicht des Open Systems Interconnection Reference Model (OSI-Modell) bilden.
- **ASLR: Addressspace Layout Randomisation** - Eine Technik, welche das Ausnutzen von Fehlern der Speicherverwaltung erschwert.
- **Authentifizierung** - Der Nachweis der behaupteten Identität eines Anwendungsbenutzers.
- **Authentifikator** - Code, der ein Kennwort, ein Token, einen MFA, eine föderierte Behauptung usw. authentifiziert.
- **Automatisierte Verifizierung** - Die Verwendung von automatisierten Tools, die Schwachstellensignaturen verwenden, um Probleme zu finden.
- **Bericht zur Verifizierung der Anwendungssicherheit** - Ein Bericht, der die Gesamtergebnisse und die unterstützende Analyse dokumentiert, die der Prüfer für eine bestimmte Anwendung erstellt hat.
- **Black Box Test** - Es handelt sich um eine Softwaretestmethode, bei der die Funktion einer Anwendung untersucht wird, ohne in ihre internen Strukturen oder Abläufe zu blicken.
- **Bösartiger Code** - Code, der während der Entwicklung einer Anwendung ohne Wissen des Anwendungseigentümers in diese eingeführt wird, und der die beabsichtigte Sicherheitsrichtlinie der Anwendung umgeht. Dies ist nicht dasselbe wie Malware wie z. B. ein Virus oder Wurm!
- **BSI: Bundesamt für Sicherheit in der Informationstechnik** - Regierungsorganisation zur Verbesserung der Informationssicherheit in Deutschland
- **CI / CD: Continuous Integration / Continuous Deployment** - Techniken um von der Softwareentwicklung bis zur Produktionseinführung zu automatisieren
- **CORS: Cross Origin Resource Sharing** - erlaubt Webbrowsern Zugriffe auf andere Dienste außerhalb seiner eigenen Domain.
- **CSP:**
  - **Content Security Policy** - Sicherheitskonzept, das Einfügen von Daten in Webseiten verhindert.
  - **Credential Service Provider** - Ein Dienst zur Verwaltung von Identitäten.
- **CWE: Common Weakness Enumeration** - ist eine von der Community entwickelte Liste von allgemeinen Sicherheitsschwächen in Software. Sie dient als gemeinsame Sprache, als Messlatte für Softwaresicherheitstools und als Grundlage für die Identifizierung von Schwachstellen, deren Milderung und Prävention.
- **DAST: Dynamic Application Security Testing** - Dynamische Anwendungssicherheitstests sind so konzipiert, dass sie Bedingungen erkennen, die auf eine Sicherheitslücke in einer Anwendung im laufenden Betrieb hindeuten.
- **Design-Verifikation** - Die technische Bewertung der Sicherheitsarchitektur einer Anwendung.
- **DoS: Denial of Service** - Ein Überlastungsangriff mit dem Ziel, die Verfügbarkeit der Anwendung oder deren Komponenten einzuschränken.
- **Dynamische Verifikation** - Die Verwendung von automatisierten Tools, die Schwachstellensignaturen verwenden, um Probleme bei der Ausführung einer Anwendung zu finden.
- **Eingabeprüfung** - Die Kanonisierung und Validierung von nicht vertrauenswürdigen Benutzereingaben.

- **FIDO: Fast Identity Online** - Eine Gruppe von Standards zur Authentifikation, die verschiedene Authentifikationsverfahren, wie z.B. biometrische Verfahren, TPMs, USB-Token etc., definieren.
- **GUID: Globally Unique Identifier** - eine eindeutige Referenznummer, die als Identifikator in der Software verwendet wird.
- **Hardcodierte Schlüssel** - Kryptographische Schlüssel, die auf dem Dateisystem gespeichert werden, sei es in Form von Codes, Kommentaren oder Dateien.
- **HSM: Hardware Security Module** - Zusatzhardware zur Schlüsselspeicherung oder Berechnung kryptografischer Algorithmen.
- **HTTP: Hyper Text Transfer Protocol** - Ein Anwendungsprotokoll für verteilte, kollaborative, hypermediale Informationssysteme. Es ist die Grundlage der Datenkommunikation für das World Wide Web.
- **Hibernate Query Language** - Eine dem SQL ähnliche Abfragesprache, die von der Hibernate ORM Bibliothek genutzt wird.
- **Komponente** - eine in sich geschlossene Code-Einheit mit den zugehörigen Ein- und Ausgabeschnittstellen, die mit anderen Komponenten kommuniziert.
- **Kryptografisches Modul** - Hardware, Software und/oder Firmware, die kryptografische Algorithmen umsetzt und kapselt und/oder kryptografische Schlüssel generiert.
- **Malware** - Ausführbarer Code, der während der Laufzeit ohne Wissen des Anwendungsbenutzers oder -administrators in eine Anwendung eingeführt wird.
- **MFA: Multiple Factor Authenticator** - Mehrfaktorauthentifikator, der zwei oder mehr Einzelfaktoren beinhaltet.
- **ORM: Object Relational Mapping** - Eine Zuordnung von Daten in relationalen Datenbanken zu objektorientierten Programmcode.
- **OS: Operating System** - Betriebssystem
- **OTP: One Time Password** - Passwort zur einmaligen Verwendung
- **OWASP: Open Web Application Security Project** - Eine weltweite, freie und offene Gemeinschaft, die sich mit der Verbesserung der Sicherheit von Anwendungssoftware befasst. Unsere Mission ist es, die Anwendungssicherheit sichtbar zu machen, damit Menschen und Organisationen fundierte Entscheidungen in Bezug auf Anwendungssicherheitsrisiken treffen können. Siehe: https://www.owasp.org/
- **Padding Oracle Angriff** - Angriff auf Software zur Verschlüsselung, der mit Hilfe der aufgefüllten Blocks (Padding) Schlüssel errät.
- **PBKDF2: Password Based Key Derivation Function 2** - Ein Algorithmus, der aus einem Passwort und einem Zufallswert einen kryptografischen Schlüssel ableitet
- **Personenbezogene Daten** - sind Informationen, die allein oder zusammen mit anderen Informationen verwendet werden können, um eine einzelne Person zu identifizieren.
- **PIE: Position Idependent Executeable** - Positionsunabhängig ausführbarer Code ist der Körper eines Maschinencodes, der, wenn er irgendwo im Primärspeicher abgelegt wird, unabhängig von seiner absoluten Adresse korrekt ausgeführt wird.
- **PKI: Public Key Infrastructure** - verbindet öffentliche Schlüssel mit den jeweiligen Identitäten von Entitäten. Die Bindung wird durch einen Prozess der Registrierung und Ausgabe von Zertifikaten bei und durch eine Zertifizierungsstelle hergestellt.
- **Positivliste** - Eine Liste zulässiger Daten oder Operationen, z.B. eine Liste von Zeichen, die Eingabeprüfung akzeptiert.
- **PSTN: Public Switched Telephone Network** - Das ist das öffentliche Telefonnetz.

- **RFI: Remote File Inclusion** - Angriffstechnik auf scriptbasierende Webanwendungen, bei der weitere, nicht autorisierte, Scripte zur Ausführung gebracht werden
- **SAST: Static Appliction Security Testing** - Statische Anwendungssicherheitstests sind eine Reihe von Technologien zur Analyse von Anwendungsquellcode, Bytecode und Binärdateien in Bezug auf Codierungs- und Designbedingungen, die auf Sicherheitsschwachstellen hinweisen. SAST-Lösungen analysieren eine Anwendung von „innen nach außen“ in nicht laufenden Zustand.
- **SDLC: Software Development Life Cycle** - Der Lebenszyklus der Softwareentwicklung ist der Prozess, der die Entwicklung von Software von den ersten Anforderungen bis zur Betriebsphase beschreibt.
- **SBOM: Software Bill of Materials** - Softwarestücklisten deklarieren alle verwendeten Bibliotheken.
- **SFA: Single Factor Authentication** - Einfaktorauthentifikator, beruht auf nur einem Faktor z.B. etwas, das Sie kennen (gespeicherte Geheimnisse, Passwörter, Passphrasen, PINs), etwas, das Sie sind (biometrische Merkmale) oder etwas, das Sie besitzen (OTP-Token, Chipkarte)
- **Sicherheitsarchitektur** - Eine Abstraktion eines Anwendungsdesigns, die identifiziert und beschreibt, wo und wie Sicherheitsmaßnahmen verwendet werden, und auch den Ort und die Empfindlichkeit von Benutzer- und Anwendungsdaten identifiziert und beschreibt.
- **Sicherheitskonfiguration** - Die Laufzeitkonfiguration einer Anwendung, die sich auf die Verwendung von Sicherheitsmaßnahmen auswirkt.
- **Sicherheitsmaßnahme** - Eine Funktion oder Komponente, die eine Sicherheitsprüfung, z. B. eine Zugriffskontrollprüfung, durchführt, oder bei deren Aufruf ein Sicherheitseffekt, z. B. die Erstellung eines Auditdatensatzes, entsteht.
- **SQL-Injektion** - Eine Injektionstechnik, bei der SQL-Anweisungen in den Datenstrom eingefügt werden.
- **SSO-Authentifizierung** - Single Sign On - Ein Benutzer meldet sich bei einer Anwendung an und ist dann automatisch bei anderen Anwendungen angemeldet, ohne sich erneut authentifizieren zu müssen. Wenn Sie sich beispielsweise bei Google anmelden, werden Sie beim Zugriff auf andere Google-Dienste wie YouTube, Google Docs und Gmail automatisch angemeldet.
- **SSRF: Server-Side Request Forgery** - Nutzt den anfälligen Server als Proxy um Informationen aus dessen Umgebung zu erlangen.
- **Template Injection Angriff** - Webserver fügen per Template Engines Daten aus Datenquellen z.B. in Websites ein. Dazu nutzen sie Platzhalter u.a. Steuerelemente. Ungeprüft ausgeführte Nutzereingaben können diese Steuerelemente missbrauchen.
- **Threat Modeling** - Eine Technik, immer raffiniertere Sicherheitsarchitekturen zu entwickeln, um Bedrohungsagenten, Sicherheitszonen, Sicherheitsmaßnahmen und wichtige technische und geschäftliche Ressourcen zu identifizieren.
- **TLS: Transport Layer Security** - Kryptographische Protokolle, die Kommunikationssicherheit für eine Netzwerkverbindung bieten.
- **TOCTOU**: Time of Check - Time of Use Race Condition - Die Zeitdifferenz zwischen Prüfung einer Eigenschaft und Nutzung könnte groß genug sein, um Änderungen der Eigenschaft zu erreichen. So könnte eine Datei zwischen Prüfung durch den Virenscanner und dem darauf folgenden Zugriff mit Schadsoftware verseucht werden.
- **TOTP**: Time-based One-Time Password - Verfahren zum Erzeugen von Einmalpasswörten auf Basis der Uhrzeit
- **TPM**: Trusted Platform Module - Zusatzchip, der ein System um grundlegende Sicherheitsfunktionen erweitert.
- **U2F**: Universal 2nd Factor - Ein FIDO-Standard, der ein USB- oder NFC-Token als zweiten Authentisierungsfaktor einbindet.

- **URI/URL/URL-Fragmente** - Ein Uniform Resource Identifier ist eine Zeichenkette, die zur Identifizierung eines Namens oder einer Web-Ressource verwendet wird. Ein Uniform Resource Locator wird oft als Verweis auf eine Ressource verwendet.
- **Verifizierer**
  - NIST: Eine Einheit, welche die Identität des Nutzers durch die Überprüfung seines Besitzes, Wissens oder Eigenschaften mittels eines Authentifizierungsprotokolls verifiziert. Zu diesem Zweck muss der Verifizierer möglicherweise auch die Anmeldedaten validieren, die Authentifikatoren mit der Kennung des Nutzers verbinden und ihren Status überprüfen.
  - OWASP: Die Person oder das Team, die bzw. das eine Anwendung anhand der OWASP ASVS-Anforderungen überprüft.
- **Verifizierung der Anwendungssicherheit** - Die technische Bewertung einer Anwendung in Bezug auf den OWASP ASVS.
- **Whitelist** - zunehmend ungebräuchlicher Begriff, siehe **Positivliste**
- **WYSIWYG: What You See Is What You Get** - Editoren u.ä. Software, die das Ergebnis sofort im finalen Design anzeigen.
- **X.509-Zertifikat** - Ein X.509-Zertifikat ist ein digitales Zertifikat, das den allgemein anerkannten internationalen X.509 PKI Standard verwendet, um zu prüfen, dass ein öffentlicher Schlüssel zu der im Zertifikat enthaltenen Benutzer-, Computer- oder Service-Identität gehört.
- **XSS: Cross Site Scripting** - Eine Sicherheitslücke, die typischerweise in Webanwendungen zu finden ist, und die das Einfügen von clientseitigen Skripten in den Inhalt ermöglicht.
- **XXE: XML External Entity** - Ein Angriff auf XML Dokumente mit Hilfe externer Doctype-Definition.

Anhang B: Referenzen

Die folgenden OWASP-Projekte könnten für die Anwender des ASVS nützlich sein:

OWASP Kernprojekte

1. OWASP Top 10 Project: https://owasp.org/www-project-top-ten/
2. OWASP Web Security Testing Guide: https://owasp.org/www-project-web-security-testing-guide/
3. OWASP Proactive Controls: https://owasp.org/www-project-proactive-controls/
4. OWASP Security Knowledge Framework: https://owasp.org/www-project-security-knowledge-framework/
5. OWASP Software Assurance Maturity Model (SAMM): https://owasp.org/www-project-samm/

OWASP Cheat Sheet Series project

Das Cheat Sheet Projekt stellt für viele Bereiche eine Vielzahl von Hilfsmitteln bereit:

Die Übersicht befindet sich unter: https://cheatsheetseries.owasp.org/cheatsheets/IndexASVS.html

Mobile Security Projekte

1. OWASP Mobile Security Project: https://owasp.org/www-project-mobile-security/
2. OWASP Mobile Top 10 Risks: https://owasp.org/www-project-mobile-top-10/
3. OWASP Mobile Security Testing Guide and Mobile Application Security Verification Standard: https://owasp.org/www-project-mobile-security-testing-guide/

OWASP Internet of Things Projekte

1. OWASP Internet of Things Project: https://owasp.org/www-project-internet-of-things/

OWASP Serverless Projekte

1. OWASP Serverless Projekte: https://owasp.org/www-project-serverless-top-10/

Andere

Die folgenden Webseiten außerhalb der OWASP könnten für die Nutzer des ASVS interessant sein:

1. SecLists Github: https://github.com/danielmiessler/SecLists
2. MITRE Common Weakness Enumeration: https://cwe.mitre.org/
3. PCI Security Standards Council: https://www.pcisecuritystandards.org
4. PCI Data Security Standard (DSS) v3.2.1 Requirements and Security Assessment Procedures: https://www.pcisecuritystandards.org/documents/PCI_DSS_v3-2-1.pdf
5. PCI Software Security Framework - Secure Software Requirements and Assessment Procedures: https://www.pcisecuritystandards.org/documents/PCI-Secure-Software-Standard-v1_0.pdf
6. PCI Secure Software Lifecycle (Secure SLC) Requirements and Assessment Procedures: https://www.pcisecuritystandards.org/documents/PCI-Secure-SLC-Standard-v1_0.pdf

Appendix C: Internet of Things Verification Requirements

Dieser Abschnitt gehörte ursprünglich zum Hauptzweig, inzwischen hat das OWASP IoT-Team viel geleistet, so dass es keinen Sinn hat, zwei verschiedene Standards zu diesem Thema beizubehalten. Für die Version 4.0 verschieben wir diesen Abschnitt in den Anhang und empfehlen allen, lieber das OWASP IoT-Hauptprojekt zu verwenden.

Ziel

Eingebettete/IoT-Geräte sollten:

- das selbe Maß an Sicherheitsmaßnahmen innerhalb des Geräts haben, wie es im Server zu finden ist. Sicherheitsmaßnahmen sollen in einer vertrauenswürdigen Umgebung durchgesetzt werden.
- Sensible Daten, die auf dem Gerät gespeichert sind, sollten auf sichere Weise unter Verwendung von hardwaregestütztem Speicher gespeichert werden.
- Alle vom Gerät übertragenen sensiblen Daten sollten auf der Transportschicht gesichert werden.

Anforderungen an die Prüfung

#	Description	L1	L2	L3	Since
C.1	Prüfen Sie, dass die Debugging-Schnittstellen der Anwendungsschicht wie USB, UART und andere serielle Varianten deaktiviert oder durch ein komplexes Kennwort geschützt sind.	✓	✓	✓	4.0
C.2	Prüfen Sie, dass kryptographische Schlüssel und Zertifikate für jedes einzelne Gerät eindeutig sind.	✓	✓	✓	4.0
C.3	Prüfen Sie, dass Speicherschutzmaßnahmen wie ASLR und DEP durch das Betriebssystem aktiviert sind.	✓	✓	✓	4.0
C.4	Prüfen Sie, dass On-Chip-Debugging-Schnittstellen wie JTAG oder SWD deaktiviert sind, oder dass der verfügbare Schutzmechanismus aktiviert und entsprechend konfiguriert ist.	✓	✓	✓	4.0
C.5	Prüfen Sie, dass Trusted Execution implementiert und aktiviert ist, falls auf dem SoC oder der CPU des Geräts verfügbar.	✓	✓	✓	4.0
C.6	Prüfen Sie, dass sensible Daten, private Schlüssel und Zertifikate sicher in einem Secure Element, TPM, TEE (Trusted Execution Environment) gespeichert oder durch starke Kryptographie geschützt sind.	✓	✓	✓	4.0
C.7	Prüfen Sie, dass die Firmwareanwendungen Daten während der Übertragung auf der Transportschicht schützen.	✓	✓	✓	4.0
C.8	Prüfen Sie, dass die Firmwareanwendungen die digitale Signatur der Serververbindungen validieren.	✓	✓	✓	4.0
C.9	Prüfen Sie, dass drahtlose Kommunikationen gegenseitig authentifiziert sind.	✓	✓	✓	4.0
C.10	Prüfen Sie, dass die drahtlose Kommunikation über einen verschlüsselten Kanal gesendet wird.	✓	✓	✓	4.0
C.11	Prüfen Sie, dass alle verbotenen C-Funktionen durch die entsprechenden sicheren gleichwertigen Funktionen ersetzt wird.	✓	✓	✓	4.0
C.12	Prüfen Sie, dass jede Firmware ein automatisches Stücklistenmanagement führt, in der die Komponenten von Drittanbietern, die Versionierung und die veröffentlichten Schwachstellen katalogisiert sind.	✓	✓	✓	4.0

#	Description	L1	L2	L3	Since
C.13	Prüfen Sie, alle Codes, einschließlich der Binärdateien, Bibliotheken und Frameworks von Drittanbietern auf hartkodierte Anmeldedaten (Backdoors).	✓	✓	✓	4.0
C.14	Prüfen Sie, dass die Anwendungs- und Firmware-Komponenten nicht für OS Command Injection anfällig sind, indem sie Shell Command Wrapper oder Skripte aufrufen, oder dass Sicherheitsmaßnahmen OS Command Injection verhindern.	✓	✓	✓	4.0
C.15	Prüfen Sie, dass die Firmware-Anwendungen die digitale Signatur an einen oder mehrere vertrauenswürdige Server pinnt.		✓	✓	4.0
C.16	Prüfen Sie das Vorhandensein von Merkmalen zur Verhinderung oder Erkennung von Manipulationen am Gerät.		✓	✓	4.0
C.17	Prüfen Sie, dass alle verfügbaren Technologien zum Schutz des geistigen Eigentums, die vom Chiphersteller zur Verfügung gestellt werden, aktiviert sind.		✓	✓	4.0
C.18	Prüfen Sie, dass Sicherheitsmaßnahmen vorhanden sind, um ein Reverse Engineering der Firmware (z.B. Entfernen von ausführlichen Debugginginformationen) zu verhindern.		✓	✓	4.0
C.19	Prüfen Sie, dass das Gerät die Signatur des Bootimages vor dem Laden validiert.		✓	✓	4.0
C.20	Prüfen Sie, dass der Firmware-Aktualisierungsprozess nicht anfällig für TOCTOU Race Conditions ist.		✓	✓	4.0
C.21	Prüfen Sie, dass das Gerät Code Signing verwendet und Firmware-Upgrade-Dateien vor der Installation validiert.		✓	✓	4.0
C.22	Prüfen Sie, dass das Gerät nicht auf alte Versionen (Anti-Rollback) von gültiger Firmware zurückgestuft werden kann.		✓	✓	4.0
C.23	Prüfen Sie die Verwendung eines kryptographisch sicheren Zufallszahlengenerators auf einem eingebetteten Gerät, z.B. unter Verwendung von Hardware.		✓	✓	4.0
C.24	Prüfen Sie, ob die Firmware automatische Firmware-Updates nach einem vordefinierten Zeitplan durchführen kann.		✓	✓	4.0
C.25	Prüfen Sie, dass das Gerät Firmware und sensible Daten löscht, wenn eine Manipulation oder der Empfang einer ungültigen Nachricht festgestellt wird.			✓	4.0
C.26	Prüfen Sie, dass nur Mikrocontroller verwendet werden, die das Deaktivieren von Debugging-Schnittstellen (z.B. JTAG, SWD) unterstützen.			✓	4.0
C.27	Prüfen Sie, dass nur Mikrocontroller verwendet werden, die einen wesentlichen Schutz vor De-Capping- und Seitenkanal-Angriffen bieten.			✓	4.0
C.28	Prüfen Sie, dass Leiterbahnen zur Übertragung sensibler Daten nicht auf den äußeren Schichten der Leiterplatte ausgesetzt sind.			✓	4.0
C.29	Prüfen Sie, dass die Kommunikation zwischen den Chips, z. B. die Kommunikation von Haupt- und Tochterplatine, verschlüsselt ist.			✓	4.0
C.30	Prüfen Sie, dass das Gerät Code Signing verwendet und den Code vor der Ausführung validiert.			✓	4.0

#	Description	L1	L2	L3	Since
C.31	Prüfen Sie, dass sensible Informationen, die im Speicher gehalten werden, mit Nullen überschrieben werden, sobald sie nicht mehr benötigt werden.			✓	4.0
C.32	Prüfen Sie, dass die Firmware-Anwendungen Kernel-Container als Isolierung zwischen den Anwendungen verwenden.			✓	4.0
C.33	Prüfen Sie, dass sichere Compilerflags wie -fPIE, -fstack-protector-all, -Wl,-z,noexecstack, -Wl,-z,noexecheap für Firmware-Builds konfiguriert sind.			✓	4.0
C.34	Prüfen Sie, dass die Mikrocontroller mit Codeschutz konfiguriert sind (falls zutreffend).			✓	4.0

Referenzen

Weitere Informationen finden Sie unter:

- OWASP Internet of Things Top 10
- OWASP Embedded Application Security Project
- OWASP Internet of Things Project
- Trudy TCP Proxy Tool