TSS-WEB

TECHNISCHER SICHERHEITSSTANDARD FÜR WEBENTWICKLUNG

Vorlage

Ein Standard der Secodis GmbH

Version 2.0
1. November 2020

Über TSS-WEB

Dieses Dokument beschreibt **exemplarische Inhalte eines technischen (und organisatorischen) Sicherheitsstandards für Webanwendungen**, der als Grundlage oder Anregung für unternehmensinterne Standards verwendet werden darf. Konkrete Umsetzungshinweise (z.B. in Bezug auf zu setzende Response-Header oder Vorgaben an kryptographische Verfahren) sollten als Anhang oder separat beschrieben werden.

Die Inhalte wurden von der Secodis GmbH erstellt und werden kontinuierlich weiterentwickelt. Die aktuelle Version des Dokuments kann unter https://tss-web.secodis.com in **deutsch und englisch** heruntergeladen werden. Dort findet sich auch ein Mapping der Anforderungen aus diesem Dokument zur aktuellen **OWASP Top Ten** sowie eine Kontaktmöglichkeit für Fragen und Anmerkungen.

Bei den in diesem Dokument beschriebenen Vorgaben handelt es sich um Best Practices, die sich auf Unternehmen eines mittleren Bedrohungspotentials und durchschnittlichen Risikoappetits beziehen. Generell wird empfohlen, die einzelnen Anforderungen vor der Einführung dieses Standards hinsichtlich ihrer Angemessenheit für ein betrachtetes Unternehmen zu prüfen und ggf. anzupassen. Auch kann es empfehlenswert sein, diese Vorlage um konkrete Vorgaben im Hinblick auf die Verwendung vorhandener Infrastruktur oder architektonischer Patterns, die spezifisch zu einem betrachteten Unternehmen (z.B. eine WAF oder ein Access Gateway) oder lokale Umgebung sind, zu ergänzen.

Empfehlung: Verknüpfen Sie bei Bedarf eigene Checklisten, Guidelines, Bugbars und Richtlinien mit diesem Standard. Dokumentieren Sie Ausnahmen zu einzelnen Vorgaben in einem entsprechenden Register sowie im Sicherheitskonzept der entsprechenden Anwendung.

Erklärungen und Hintergrundwissen zu den Inhalten in diesem Dokument finden Sie u.a. in dem Buch „**Sicherheit von Webanwendungen in der Praxis“** (www.webappsecbuch.de).

Lizenz

Matthias Rohr
Secodis GmbH

Änderungshistorie

Version	Datum	Wichtige inhaltliche Änderungen
1.0	20.02.2015	Release (siehe Changelog für Anpassungen gegenüber Vorabversionen)
1.01	21.03.2015	– Neue Definition „Vertraulicher Programmcode“ und deren konsistente Verwendung im Standard.
1.02	1.7.2015	– Korrektur mehrere Typos und Formulierungen ohne inhaltliche Änderungen.
1.1	26.11.2015	– Zahlreiche Anpassungen, siehe Changelog
1.2	10.5.2016	– Einbau von Anforderungen für Rest Services (8.16.7) – Mehrere sprachliche Anpassungen – Änderung des Titels – Neues Layout
1.3	29.8.2016	– Vollständige Überarbeitung im Rahmen der englischen Übersetzung – Einführung neuer Schutzklassen – Anpassungen bzgl. Sicherheit in agilen Projekten
1.3.1	28.11.2016	– Anpassung an CSP-Statements – CVSS Scoring für 3rd-Party-Komponenten eingefügt
1.4	31.12.2016	– Neue Anforderungen für X.509-Zertifikate (8.17) – Anpassungen an Kapitel 5 in Bezug auf agile Teams – „Verwendung sicherer JavaScript-APIs“ aus 8.4 („Ausgabevalidierung“) nach 8.15 („Clientseitige Sicherheit“) verschoben. – Mehrere Änderungen an Kapitel 6 („Security Tests“). – Security Auditor in Security Officer umbenannt
1.5	15.8.2017	– Security Officer wurde in IT-Sicherheitsfunktion umbenannt – Überarbeitung von Kapitel 7 („Zulieferervorgaben“) – OWASP Top Ten Mapping wurde entfernt und wird zukünftig noch noch über die Webseite (tss-web.secodis.com) aktualisiert – Schutzklasse in Assuranceklasse umbenannt und diese angepasst – Anpassungen bzgl. Anforderungen an sicheren Dateiupload – Entfernung der Anforderung für HPKP und neue Anforderung für Referer Policy
1.6	19.3.2019	– Abschnitt 1.3 („Rollen“): Anpassung der Zuständigkeiten der Rolle „Security Champion“ und

		neue Rolle „Entwickler“ – Kapitel 3 („Sicherer Betrieb von Anwendungen“): Erweiterung der Anforderung bzgl. DMZ-Einschränkungen und Entfernen der Anforderung bzgl. Separierung in- und extern verwendeter Anwendungen und Integration von Security Monitoring – Überarbeitung Kapitel 5 („Sicherheit von Source- und Programmcode“): Neuer Punkt 2 bzgl. Zugriffe auf Entwicklungssysteme aus dem Internet. – Abschnitt 8.2 („Eingabevalidierung“) Neue Anforderung bzgl. unsichere Objekt- Deserialisierung hinzugefügt. – Abschnitt 8.5 („Authentifizierung & Registrierung von Benutzern“): Umstellung auf aktuellen NIST-Standard NIST SP 800-63B und entsprechende Level. – Überarbeitung von Abschnitt 8.8 („Authentifizierung am Backend“) – Überarbeitung der OAuth-Anforderungen in Abschnitt 8.11 (Access Controls) und Bewegen in Service Security mit CORS-Anforderungen – Überarbeitung von Abschnitt 8.14 (“Verwaltung kryptographischer Schlüssel“) – Abschnitt 8.16 („XML Parser Security“) im Sinne von API Security überarbeitet (u.a. Integration von OAuth, OpenID Connect) und in „Service Security“ umbenannt. – XML Parser Security nach Abschnitt 3.2 („Eingabevalidierung“) verschoben. – Anforderungen für Content Security Policy (CSP) in Anhang A (“Vorgaben für HTTP Security Header”) überarbeitet und Integration von SameSite-Cookies
1.7	24.5.2019	– Erweitern der Rolle „Security Champion“ – Umbenennen von 3rd-Party-Komponenten in 3rd-Party-Abhängigkeiten und Überarbeiten der relevanten Anforderungen in Kapitel 4 – Überarbeiten der Anforderungen in Kapitel 5 („Sicherheit im Softwareentwicklungsprozess“) – Überarbeiten der Anforderungen in Kapitel 6 („Security Tests“) – Aktualisierung von Anforderungen bzgl. Security Monitoring und Logging
1.8	1.7.2019	– Kapitel 3 („Sicherer Betrieb von Anwendungen“) im Hinblick auf Container- und Cloud-basierte Deployments überarbeitet. – Kapitel 5 („Sicherheit im Softwareentwicklungsprozess“) überarbeitet und neu strukturiert. – Kapitel 6 („Sicherheitstests“) komplett überarbeitet und

		neu strukturiert. – Name des Standards geändert von „Sicherheitsstandard für Webanwendungen“ in „Sicherheitsstandard in Webentwicklung“ – Mehrere kleinere Änderungen in verschiedenen Kapiteln.
2.0		Vollständige Überarbeitung, insbesondere Anpassungen an: – Kapitel 3 („Sicherer Betrieb“) – Kapitel 5 („Sicherheit im Softwareentwicklungsprozess“) überarbeitet und neu strukturiert. – Kapitel 6 („Security Tests“) – Abschnitt 8.11. („Data Security & Cryptography“) – Abschnitt 8.14 („Service & API Security”)

Sicherheitsstandard für Webentwicklung

Firma Muster AG

Version: 1.0
Klassifikation: INTERN

Inhalt

Dokumenteigenschaften

Ablageort	**tbd**
Owner	**tbd**
Typ	**Technischer Standard**
Klassifikation	**Nur für internen Gebrauch (INTERN)**
Nächster Review	**tbd**

Referenzierte Dokumente

Dokument	Ablageort
TSS-WEB Version 2.0 (Vorlage)	https://tss-web.secodis.com
Information Security Policy	tbd
Passwort Policy	tbd

Ansprechpartner für dieses Dokument

Name	E-Mail	Bereich

Änderungshistorie

Version	Datum	Änderungen	Geändert von
0.1	20.12.2020	Initiales Dokument auf Basis von TSS-WEB v2.0	Max Mustermann

1 Einführung

Dieser Standard beschreibt generelle Sicherheitsvorgaben (**Baseline**) für Webentwicklung, bzw. web-basierte Anwendungen, oder Anwendungskomponenten, die von oder für die Muster AG neu entwickelt werden. Dieses Dokument hat zum Ziel ein Basissicherheitsniveau zu definieren, welches im Bedarfsfall jedoch überschritten werden kann und auch sollte, sofern ein höherer Schutzbedarf (bzw. Gefährdungspotential) für eine Anwendung vorliegt.

1.1 Geltungsbereich

Dieser Standard gilt für alle ab dem **[DD].[MM].[JJJJ]** von der Muster AG neuentwickelten, in Auftrag gegebenen oder eingekauften und **produktiv eingesetzten webbasierten Anwendungen oder Anwendungskomponenten bzw. der darunterliegenden Infrastruktur**. Für alle übrigen Anwendungen besitzen die Vorgaben in diesem Dokument Empfehlungscharakter.

1.2 Arten von Vorgaben

In diesem Standard werden gemäß RFC2119, zwei grundsätzliche Arten von Vorgaben unterschieden:

- **Verbindliche Vorgaben**: Kennzeichnung durch Schlüsselworte „MUSS“, „MÜSSEN“, „SIND“, „IST“, „DARF NICHT“ etc.
- **Empfehlungen**: Kennzeichnung durch die Schlüsselworte „SOLLTE“, „KANN“, etc.

Von der Verwendung von Empfehlungen, darf im Fall von begründbaren Einwänden abgewichen bzw. abgesehen werden. Empfehlungen, die mit „KANN“ gekennzeichnet sind, beziehen sich vor allem auf Anwendungen, die ein höheres Schutzniveau anstreben.

Ausnahmen zu den Pflichten müssen begründet und durch die IT-Sicherheitsfunktion genehmigt werden.

1.3 Definitionen

Diesem Standard liegen die folgenden Begriffsdefinitionen zugrunde:

- **Anwendung:** Hier: Synonym für Webanwendung.
- **Change:** Jede Änderung an einer produktiven Anwendung außerhalb eines Releases.
- **Dependency Repository**: System, in welchem 3rd-Party-Abhängigkeiten (z.B. Bibliotheken, Maven Artefakte) abgelegt sind. Häufig stellt ein Dependency Repository einen Bestandteil eines allgemeinen Software Repositories wie Nexus oder Artifactory dar.
- **Externe Webanwendung**: Eine Webanwendung die von außerhalb des Unternehmens (z.B. über das Internet) aufrufbar ist.
- **Interne Webanwendung:** Eine Webanwendung die nur innerhalb des Unternehmens aufrufbar ist.
- **Interner Source- oder Programmcode**: Source- bzw. Programmcode, welcher nicht vertraulich oder öffentlich ist (Standard).
- **Kritikalität**: Hier: Synonym für Geschäftskritikalität.

- **Service:** Hier: Synonym für Web-basierten Dienst, z.B. Web Services oder REST Services / API.
- **Source Code Repository**: System in dem selbstentwickelter Sourcecode abgelegt wird (z.B. SVN, Git).
- **Vertrauliche Daten**: Daten, welche
  (1) vertrauliche Informationen enthalten bzw. enthalten könnte (z.B. Patente, sensible Rechenlogik, personenbezogene Daten, Passwörter),
  (2) explizit als solcher gekennzeichnet wurden oder
  (3) allgemein nur einem eingeschränkten Personenkreis zugänglich sind bzw. sein sollen.
- **Vertraulicher Source- oder Programmcode:** Source- bzw. Programmcode der -> *Vertrauliche Daten* darstellt.
- **Webanwendung**: Hier: Ein Softwareprogram, Service (z.B. Web API) oder eine Kombination aus diesen welches über HTTP(S) aufrufbar ist.
- **Web-basierte Anwendung**: Siehe Webanwendung.

1.4 Rollen

In diesem Standard werden die folgenden Rollen verwendet:

- **IT-Sicherheitsfunktion**: Organisatorische Einheit oder Person, die für die Erstellung von generellen IT-Sicherheits-Vorgaben und Kontrolle deren Einhaltung zuständig ist. Diese Rolle KANN auch durch einen Security Officer wahrgenommen werden, welcher für ein bestimmtes Projekt oder Team zuständig ist.
- **Security Champion:** Lokaler Koordinator bzw. Ansprechpartner für IT-Sicherheit innerhalb eines Teams (z.B. eines Entwicklungsteams). Zu den Aufgaben eines Security Champion zählt u.a.:
  (1) Ansprechpartner für IT-Sicherheit eines Teams.
  (2) Kenntnis relevanter Sicherheitsanforderungen, deren Umsetzung (bzw. Einhaltung) innerhalb des Teams einfordern, durchführen bzw. ggf. dies koordinieren.
  (3) Sicherheitsrisiken selbständig identifizieren und managen.
  (4) Prüft die korrekte Umsetzung von sicherheitsrelevanten Anforderungen.
  (5) Die Effektivität von Sicherheitsprüfungen und Maßnahmen laufend kontrollieren, wo möglich diese automatisieren (z.B. durch den Einsatz von Security Scannern) und die regelmäßige Auswertung identifizierter Security-Findings von eingesetzten Tools.
  (6) Nimmt an Security-Terminen teil (z.B. Security JF).
- **Entwickler:** An (Software-)Entwickler werden allgemein die folgenden Aufgaben in Bezug auf IT-Sicherheit gestellt:

(1) Besitzt eine allgemeine Kenntnis der relevanten Sicherheitsaspekte der von ihm / ihr verwendeten Technologien und hält sich hier laufend auf dem neuesten Stand.

(2) Ist in der Lage Schwachstellen selbständig zu identifizieren, zu fixen und zu vermeiden.

- **(Entwicklungs)Team:** Entwickelt Anwendung oder Anwendungskomponente. Ist für die Sicherheit der von ihm entwickelten Anwendung (umsetzungs-)verantwortlich, etwas durch die Umsetzung von Sicherheitsanforderungen und Korrektur von identifizierten Sicherheitsproblemen.

Referenzen zu diesen Rollen sind in diesem Standard *kursiv* gekennzeichnet.

1.5 Assuranceklassen

Einige Anforderungen in diesem Standard sind von bestimmten Aspekten einer Anwendung abhängig.

Zum einen betrifft diese deren Zugreifbarkeit, ob diese von extern (z.B. aus dem Internet) oder ausschließlich intern aufrufbar ist. Zum anderen deren Geschäftskritikalität:

		Zugreifbarkeit der Anwendung	
		Interne Anwendung (nicht aus dem Internet erreichbar)	*Externe Anwendung* (aus dem Internet erreichbar)
Geschäftskritikalität der Anwendung	*<= Mittel*	**Niedrig**	**Standard**
	Hoch	**Standard**	**Hoch**
	Sehr Hoch	**Hoch**	**Sehr Hoch**

Tabelle 1-1: Assuranceklassen von Anwendungen

2 Beheben von Schwachstellen in Anwendungen

Identifizierte Sicherheitsprobleme in Anwendungen SIND generell zeitnah und ursächlich zu beheben. Sofern die ursächliche Behebung einer Schwachstelle, von der ein signifikantes Risiko ausgeht, jedoch eine erhebliche Zeit erfordert, SOLLTEN zunächst schnell umsetzbare Maßnahmen ergriffen werden, um das von dieser Schwachstelle ausgehende Risiko möglichst weit zu reduzieren. Eine solche Maßnahme DARF immer nur als temporäre Zwischenlösung betrachtet und stets eine ursächliche Behebung der Schwachstelle angestrebt werden.

In Bezug auf externe Anwendungen, welche z.B. aus dem Internet erreichbar sind, gelten hierbei die folgenden Vorgaben bis zu welchem Zeitpunkt eine Schwachstelle spätestens korrigiert, bzw. deren Ausnutzbarkeit durch eine temporäre Maßnahme zu unterbinden sein MUSS:

		Bewertung der Schwachstelle		
		Als „kritisch“ bewertete Schwachstelle	Als „hoch“ bewertete Schwachstelle	Als „mittel“ bewertete Schwachstelle
Kritikalität[1] der Anwendung	>= *Hoch*	zum nächsten **Arbeitstag**	Innerhalb von **7 Tagen**[2]	Im Rahmen des nächsten Releases, jedoch spätestens nach 6 Monaten.
	<= *Mittel*	Innerhalb von **7 Tagen**	Innerhalb von **21 Tagen**	-

Tabelle 2-1: Vorgaben zur Behebung von Schwachstellen für externe Anwendungen

In Bezug auf Anwendungen, welche nicht aus dem Internet erreichbar sind (interne Anwendungen), MÜSSEN die folgenden Vorgaben beachtet werden:

		Bewertung der Schwachstelle		
		Als „kritisch“ bewertete Schwachstelle	Als „hoch“ bewertete Schwachstelle	Als „mittel“ bewertete Schwachstelle
Kritikalität der Anwendung	>= *Hoch*	Innerhalb von **7 Tagen**	Innerhalb von **30 Tagen**	Im Rahmen des nächsten Releases, jedoch spätestens nach 12 Monaten.
	<= *Mittel*	Innerhalb von **21 Tagen**	Innerhalb von **60 Tagen**	-

Tabelle 2-2: Vorgaben zur Behebung von Schwachstellen für interne Anwendungen

[1] Kritikalität = Geschäftskritikalität
[2] Tage = Kalendertage

3 Sicherer Betrieb

Die folgenden Vorgaben gelten für Systeme (infrastrukturelle Komponenten), auf denen produktive Anwendungen der Muster AG ausgeführt werden:

1. Systeme MÜSSEN strikt von Entwicklungs- und Testsystemen (d.h. infrastrukturell, applikationsseitig sowie datenseitig) voneinander getrennt werden.
   a) Produktions- und Entwicklungssysteme MÜSSEN voneinander separiert werden (etwa durch separate Cloud-Umgebungen).
   b) Verbindungen zwischen Umgebungen MÜSSEN wenn möglich verschlüsselt sein.
   c) Produktivdaten SOLLTEN auf Nicht-Produktivsystemen nicht verwendet werden (sehe Ausnahmen in Kapitel 6).
   d) Benutzer MÜSSEN dediziert für alle Umgebungen privilegiert werden.
   e) Access Server SOLLTEN separate Instanzen für alle Umgebungen mit separaten Reals verwenden.
2. Systeme (z.B. Webserver, Applikationsserver, Container-Plattformen, Content Management Systeme) MÜSSEN nach gängigen Best Practices gehärtet werden. Dies betrifft:
   a) Einem gehärteten Betriebssystem (z.B. durch Einsatz eines gehärteten Base Images)
   b) Abschaltung von nicht benötigten Diensten, Plugins und sonstigen Funktionen
   c) Einem gehärteten SSL/TLS Stack (siehe 8.11)
   d) Gesetzter Security Header (siehe Anhang A)
   e) Entfernter Default- und Demoinhalte
   f) Exponierte Netzwerkdienste (z.B. von Web- und Applikationsserver) MÜSSEN mit einer dedizierten Systemkennung mit minimalen Berechtigungen ausgeführt werden, welche von anderen Diensten abgeschottet sein SOLLTEN (z.B. durch Betrieb in dedizierten Container oder chroot-Umgebung)
   g) Binden von Netzwerkdiensten nur an Localhost, wenn diese von dem lokalen System erreicht werden müssen.
   h) Einschränken des Zugriffs auf Netzwerkdienste auf bestimmte IP-Adressen wenn möglich.
   i) Abschalten nicht erforderlicher Dateihandler (z.B. „.php“ bei einer Java-Anwendung)
   j) Abschalten unsicherer oder nicht benötigter HTTP-Methoden (insb. TRACE und TRACK).
   k) Web- und Applikationsserver dürfen keine Details (z.B. Versionsnummern) des serverseitigen Software-Stacks offenlegen. Entsprechende Response Header (z.B. „Server“ oder „X-Powered-By“) sind hierzu genauso zu minimieren bzw. deaktivieren wie in HTML-Code (oder anderer Stelle) eingetragene Informationen.
3. Docker Security:
   a. Docker Images MÜSSEN mittels vertrauenswürdiger Repositories gebaut werden.
   b. Docker Images MÜSSEN ein zugewiesenes Base Image verwenden.
   c. Im Rahmen des Builds von Docker Images MÜSSEN Betriebssystempaketen aktualisiert werden.
   d. Docker Images müssen gegen unsichere 3rd-Party-Abhängigkeiten und unsichere

Einstellungen gescannt werden.

e. Docker Images dürfen keine Remote Shell, SSH- oder Telnet-Dienste besitzen.
f. Docker Container MÜSSEN in ihrer maximalen Laufzeit begrenzt werden und danach die entsprechenden Images mit aktualisierten Betriebssystempaketen neugebaut werden.

4. Absicherung des Zugriffs auf Backend Ressourcen:
   a. Jeder Prozess DARF nur die notwendigen Berechtigungen besitzen, z.B. auf das Dateisystem oder die Datenbank zuzugreifen (Least Pricilege Principle)
   b. Zugriffe auf Backendsysteme MÜSSEN gemäß der Vorgaben aus Abschnitt 8.14 authentifiziert und autorisiert werden.
   c. Zugriffe auf Backendsysteme MÜSSEN über dedizierte Credentials erfolgen.
   d. Secrets MÜSSEN gemäß der Vorgaben aus Abschnitt 8.12 geschützt werden.
5. Isolierung von externen Systemen:
6. Administrativer Zugriff
7. System Maintanance:
8.
9. Systeme, welche direkt aus dem Internet zugreifbar sind MÜSSEN in einem abgeschotteten Netzwerbereich (DMZ oder Cloud VPC) betrieben werden. Für diesen gilt:
   a) Auf interne Netzwerkbereiche DARF NICHT von dort aus zugegriffen werden können, sofern hierfür keine genehmigten Ausnahmen existieren.
   b) Der ausgehende Zugriff (egress) auf das MUSS Internet eingeschränkt werden. Dies SOLLTE wenn möglich über Proxys (z.B. HTTP Proxy, SMTP Proxy) erfolgen.
10. Administrativer Zugriff MUSS soweit wie möglich beschränkt werden:
    a. Eingeschränkt auf limitierte Benutzergruppe mit dedizierten Accounts.
    b. Eingeschränkt auf das interne Netzwerk oder autorisierte IPs sofern möglich.
    c. Durch Einsatz eines zweiten Authentifizierungsfaktors (z.B. Hardware Tokens, Authenticator Apps, X.509-Client-Zertifikate) in Verbindung mit einem starken Passwort.
11. Anwendungen, die aus dem Internet erreichbar sind, KÖNNEN über eine Webanwendungsfirewall (WAF) zusätzlich geschützt werden. Solche Systeme lassen sich auch als IDS auf Applikationsebene und der Durchführung von Virtual Patching einsetzen. WAFs (oder vergleichbare Systeme) DÜRFEN jedoch NICHT im Rahmen der Entwicklung oder der Testdurchführung aktiv sein und keinesfalls als Ersatz für anwendungsinterne Sicherheitsmaßnahmen eingesetzt werden.
12. Anwendungen und deren Infrastruktur MÜSSEN periodisch und möglichst automatisiert auf mögliche Sicherheitsbedrohungen geprüft werden. Z.B.:
    a) Unsichere Konfiguration
    b) Schwachstellen in eingesetzten Standardkomponenten (bzw. fehlende Patches)
    c) Abgelaufene X.509-Zertifikate
    d) Exponierte Entwicklungsartefakte (z.B. SVN-Dateien)
    e) Potentielle Malware-Infektion
13. Potentielle Security Incidents SOLLTEN laufend über ein Security Monitoring identifiziert

werden. Beispiele:

a) Missbrauch von Accounts
b) Fehler in Security Controls und fehlgeschlagene Security Tests
c) Kritische Sicherheitsberechtigungen
d) Gültigkeit von X.509-Zertifikaten
e) Sicherheit von SSL/TLS-Konfigurationen
f) Potentielle Malware-Infektionen
g) DoS-Angriffe

4 Sicherheit von Source- und Programmcode

Die folgenden Vorgaben gelten für den Schutz des Source- und Programmcodes von produktiven Anwendungen und Diensten der Muster AG, sowie der Sicherheit dort eingebundener *3rd-Party-Komponenten*:

1. Alle Zugriffe auf *vertraulichen Source- oder Programmcode* MÜSSEN auf berechtigte Personengruppen beschränkt werden (Need-to-Know-Prinzip). Dies schließt die Authentifizierung und Autorisierung entsprechender Zugriffe an Cloud-Anbietern, in Code Repositories (z.B. SVN, Git), Build-Systeme (z.B. Jenkins, TFS), Wikis oder anderer Datenspeicher (z.B. das Dateisystem) ein.
2. Remote-Zugriffe auf Entwicklungssysteme aus dem Internet MÜSSEN über einen VPN-Tunnel erfolgen.
3. Der Austausch von eigenem *Source- oder Programmcode* außerhalb der Muster AG DARF NUR über sichere Kanäle (z.B. S/MIME oder per TLS) erfolgen.
4. Das Source-Code-Repository SOLLTE periodisch auf dort offengelegte Secrets (z.B. X.509 Private Keys oder API Keys) hin automatisiert geprüft werden.
5. *Source- oder Programmcode* DARF NUR nach expliziter Freigabe auf externen Plattformen (z.B. in Internet Foren) veröffentlicht oder Externen Personenkreisen auf sonstige Weise verfügbar gemacht werden.
6. Sicherheit von 3rd-Party-Abhängigkeiten:
   a) *3rd-Party-Abhängigkeiten* SOLLTEN nur über genehmigte Dependency Repositories bezogen werden.
   b) Bevor eine neue *3rd-Party-Abhängigkeit* in der Produktion (bzw. der Release Build Umgebung) eingebunden wird, MUSS diese vom das Architektur Board freigegeben werden. Neue Versionen (inkl. Major Releases) bereits freigegebener Komponenten sind hiervon nicht betroffen.
   c) *3rd-Party-Abhängigkeiten* SOLLTEN vor ihrer Verwendung in produktiven Anwendungen automatisiert mit aktuellen und hierfür geeigneten SCA-Tools (Software Composition Analysis) auf mögliche Schwachstellen (insb. Known Vulnerabilities) hin geprüft werden.
   d) Applikationen, welche *3rd-Party-Abhängigkeiten* mit verifizierten Schwachstellen mit einem CVSS v3 Score[3] >= 7.0 enthalten DÜRFEN NICHT in produktiven Anwendungen ohne entsprechende Freigabe (z.B. Risikoakzeptanz) verwendet werden.
      i. Teams DÜRFEN den CVSS Base Score um den CVSS Environment Score erweitern, um hierüber Aspekte wie Datenklassifikation oder Zugreifbarkeit zu berücksichtigen.
      ii. Wenn der CVSS Score geändert wurde, MUSS der entsprechende CVSS Vektor dokumentiert werden.
   e) Teams SOLLTEN ihre eigenen Applikationen innerhalb ihrer Build Pipeline auf Schwachstellen in verwendeten 3rd-Party-Abhängigkeiten scannen und alle Findings

[3] CVSS = Common Vulnerability Scoring System (CVSS) v3, https://www.first.org/cvss

mit CVSS v3 Score >= 60 analysieren. Im Falle eines entsprechenden Befundes SOLLTE der Build fehlschlagen oder auf „instable“ gesetzt werden.

f) Produktive Applikation, die nicht mehr weiterentwickelt werden, MÜSSEN ebenfalls regelmäßig auf neu bekanntgewordene Schwachstellen in verwendeten 3rd-Party-Abhängigkeiten analysiert werden (z.B. innerhalb des Software Repositories).

g) Management-Prozess etabliert werden (3rd-Party-Patch-Management).

5 Sicherheit im Softwareentwicklungsprozess

Die folgenden Vorgaben gelten für alle im Rahmen von Projekten sowie der Linie entwickelte Anwendungen innerhalb der Muster AG:

a. Generelle Anforderungen:

a. Für jedes Entwicklungsteam MUSS ein *Security Champion* benannt werden. Die gleiche Person kann diese Rolle für mehrere Teams gleichzeitig ausführen.

b. Sicherheit MUSS im Rahmen des gesamten Entwicklungsprozesses angemessen Betrachtung finden (Anforderungen, Architektur, Implementierung, Test, Deployment und Betrieb.

c. Alle sicherheitsrelevanten Entscheidungen und Annahmen MÜSSEN laufend auf mögliche Sicherheitsimplikationen hinterfragt werden.

d. Die Anwendungsentwicklung MUSS in Bezug auf gespeicherte Daten und Systemumgebungen in Test und Produktion separiert werden.

b. Eine **Bewertung aller Anforderungen und Changes** (z.B. User Storys) MUSS hinsichtlich ihrer Sicherheit erfolgen und dies dokumentiert werden, bevor diese für die Umsetzung (z.B. in einem Sprint, Release) eingeplant werden dürfen.

a. Agil arbeitende Teams SOLLTEN hierzu entsprechende Kriterien in ihre Definition of Ready (DoR) integrieren.

b. Für *Assuranceklasse* >= [HIGH] gilt eine Anforderung implizit als sicherheitsrelevant, wenn diese nicht explizit anders bewertet wurde.

c. **Sicherheitsfreigaben** (Security Gates):

a. Project Approval: Im Rahmen der Initiierung bzw. Genehmigung neuer Projekte MUSS eine Sicherheitsfreigabe durch die relevante *IT-Sicherheitsfunktion* erfolgen und durch diese bei Bedarf erforderliche Sicherheitsanforderungen für das Projekt festgelegt werden.

b. Architecture Approval: Für alle neu entwickelten Anwendungen mit *Assuranceklasse* >= [HIGH], oder falls dies durch die relevante *IT-Sicherheitsfunktion* explizit gefordert wurde, MUSS die Solution Architektur sowie relevante Teile des Sicherheitskonzepts (siehe unten) inklusive eines Bedrohungsmodells vor Beginn der Implementierung abgenommen und freigegeben werden. Hierfür können durch diese Kriterien festgelegt werden, für welche Arten von Änderungen an der Architektur eine erneute Freigabe erforderlich ist.

c. Go-Live Approval: Vor der Produktivnahme des ersten Releases einer Anwendung mit *Assuranceklasse* >= [HIGH] MUSS dieses durch die relevante *IT-Sicherheitsfunktion* freigegeben werden. Die relevante *IT-Sicherheitsfunktion* KANN festlegen, dass eine Abnahme auch für (bestimmte) nachfolgende Releases oder Anwendungen mit niedrigerer *Assuranceklasse* erforderlich ist.

d. Alle Freigaben, Risikoentscheidungen MÜSSEN dokumentiert werden.

d. Die **korrekte und vollständige der Umsetzung sicherheitsrelevanter Anforderungen** MUSS durch eine angemessene Sicherheitsprüfung verifiziert werden:

a. Im Fall von implementierten Sicherheitsfunktionen (z.B. Access Controls oder Krypto-APIs) SOLLTEN automatisiert durchgeführte Sicherheitstests erstellt

werden.

b. Für Anwendungen mit *Assuranceklasse* >= [HIGH], MÜSSEN alle commits auf Master Branches von einer zweiten Person gerewied werden (z.B. als Pull oder Merge Request).
c. *Security Champions* SOLLTEN die Umsetzung sicherheitsrelevanter Changes und User Storys reviewen.
d. Aktualisierung der Sicherheitsdokumentation (inkl. Threat Model) falls erforderlich.
e. Agil arbeitende Teams SOLLTEN hierzu entsprechende Kriterien in ihre Definition of Done (DoD) integrieren.

e. Die **Korrektur von sicherheitsrelevanten Mängeln** mit einer Bewertung >= [HOCH] MÜSSEN vor jeder Produktivnahme eines Releases erfolgen:
    a. Ist dies nicht möglich, MUSS eine Risikoübernahme durch die verantwortliche Managementfunktion (z.B. den Projektmanager) durchgeführt werden.
    b. In Ausnahmefällen (z.B. bei temporären Workarounds) DARF die relevante *IT-Sicherheitsfunktion* Freigaben unter Vorbehalt erlauben.
f. Alle Änderungen am Sourcecode MÜSSEN in ein *Source-Code-Repository* committed werden.
g. Zu jeder neu entwickelten Anwendung mit der *Assuranceklasse* >= [HOCH] MUSS ein **Sicherheitskonzept** erstellt und abgenommen werden, bevor die Entwicklung begonnen wird (relevante Aspekte) und eine Anwendung produktiv gehen darf (vollständige Dokumentation). In dieser Dokumentation SOLLTEN, sofern nicht anders mit der relevanten *IT-Sicherheitsfunktion* abgestimmt, die folgenden Aspekte zu dokumentiert werden:
    - Daten- und Anwendungsklassifikation (*Assuranceklasse*).
    - Systemübersicht (relevante Anwendungskomponenten, Schnittstellen intern/extern, verarbeitete Daten, wichtige Datenflüsse als Diagramm)
    - Zugrundeliegende Sicherheitsvorgaben (z.B. Sicherheitsstandards)
    - Ein Bedrohungsmodell, welches identifizierte Bedrohungen und relevante Gegenmaßnahme für deren Mitigierung beschreibt,
    - Rollen- und Berechtigungskonzept
    - Sicherheitsarchitektur, Sicherheitsmaßnahmen / -anforderungen und -Maßnahmen (fachlich und technisch)
    - Vorgaben für den sicheren Betrieb (erforderlich vor Produktivname)

6 Sicherheitstests

Die folgenden Vorgaben gelten in Bezug auf die Durchführung von Sicherheitstests von für den produktiven Einsatz bei der Muster AG entwickelten Anwendungen:

1. Die korrekte und vollständige Umsetzung jeder sicherheitsrelevanten Anforderung MUSS durch einen entsprechenden Sicherheitstest verifiziert werden.
2. Wo möglich SOLLTEN Sicherheitstests umfänglich, automatisiert, kontinuierlich (z.B. innerhalb der CI- und CD-Pipeline) und so früh wie möglich (Fail Fast Prinzip) durchgeführt werden.
3. Arten von Tests:
   a. Anwendungen MÜSSEN automatisiert mit Security-Code-Scanning-Tools (SAST oder IAST) gescannt werden, um Schwachstellen in Source- und Programcode zu identifizieren.
   b. Applikationen MÜSSEN automatisiert auf die Sicherheit der von ihr verwendeten *3rd-Party-Abhängigkeiten* hin analysiert werden (siehe Kapitel 4).
   c. Automatisiert ausgeführte funktionale Tests SOLLTEN ebenfalls funktionale technische Sicherheitsaspekte (z.B. Authentifizierung, oder Berechtigungen) oder sicherheitsrelevante Geschäftslogik abdecken.
   d. Anwendungen MÜSSEN durch einen Penetrationstest auf Basis der untenstehende Pentesting Policy geprüft und durch die relevante *IT-Sicherheitsfunktion* abgenommen werden.
   e. Penetrationstests SOLLTEN in einer produktionsnahen Testumgebung durchgeführt werden (z.B. in der Integrationsumgebung).
   f. Anwendungen SOLLTEN auch auf potentielle Sicherheitsprobleme hin analysiert werden, die nicht durch explizite Sicherheitsanforderungen abgedeckt sind (z.B. unsichere Geschäftslogik).

		Erreichbarkeit der Anwendung	
		Externe Anwendung (aus dem Internet erreichbar)	*Interne Anwendung* (nicht aus dem Internet erreichbar)
Kritikalität der Anwendung	>= *Hoch*	Vor initialer Produktivsetzung und min einmal **jährlich**[4]	Zeitnah zur initialen Produktivsetzung und min. alle **drei Jahre**
	<= *Medium*	Vor initialer Produktivsetzung und min alle **zwei Jahre**	-

Tabelle 6-1: Policy für die Durchführung von Pentests

4. Nach der Korrektur jeder Schwachstelle MUSS im Rahmen eines Retests die Wirksamkeit der Maßnahme verifiziert werden. Idealerweise SOLLTE dieser durch denselben Tester

[4] Sofern sichergestellt ist, dass in dieser Zeit keinerlei Änderungen an der betreffenden Anwendung durchgeführt wurden, DARF das Intervall um ein weiteres Jahr verlängert werden. Spätestens dann muss jedoch unabhängig von Änderungen eine neue Sicherheitsprüfung erfolgen, damit bis dahin neu bekanntgewordene Sicherheitsbedrohungen berücksichtigen werden.

erfolgen, der auch die Schwachstelle ursprünglich identifiziert hatte. Auf Entwicklungs- und Testsystemen DÜRFEN NUR synthetische oder anonymisierte Produktivdaten verwendet werden, die keinerlei Vertraulichkeit, bzw. einen indirekten oder direkten Personenbezug ermöglichen, besitzen.

5. Die Durchführung von Sicherheitstests DARF NICHT durch eine perimetrische Sicherheitskomponente (z.B. eine Webanwendungsfirewall) beeinflußt werden können.

7 Zulieferervorgaben

Die folgenden Vorgaben gelten zusätzlich zu den übrigen Vorgaben dieses Dokuments für Lieferanten, die im Auftrag der Muster AG Software entwickeln und sollten im Rahmen von vertraglichen Vereinbarungen aufzuführen. Der Auftragnehmer MUSS sich zu folgenden Punkten verpflichten:

1. Zur Einhaltung der allgemeinen Sorgfaltspflicht: Alle erforderlichen Maßnahmen innerhalb von Entwicklung, Betrieb und Qualitätssicherung werden umgesetzt, um das Auftreten von Sicherheitsmängeln zu vermeiden und den geltenden „Stand der Technik“ in Bezug auf Sicherheit umzusetzen.
2. Für Anwendungen der *Assuranceklasse* >= [HOCH]:
   a) Ein Nachweis[5], dass Sicherheit im gesamten Entwicklungsprozess angemessen berücksichtigt wurde.
   b) Ein, gemäß der Anforderungen aus Kapitel 5 dokumentiertes, Sicherheitskonzept als Bestandteil des Pflichtenheftes.
3. Zur Umsetzung erforderlicher Sicherheitsmaßnahmen, insbesondere der in diesem Dokument genannten Vorgaben an den sicheren Betrieb (Kapitel 3) sowie Implementierung (Kapitel 8).
4. Einen Ansprechpartner für Sicherheitsfragen in seinem Hause zu benennen, der konstruktiv bei der Bewertung und ggf. Behebung von Sicherheitsbefunden mitarbeitet.
5. Nur autorisierte und erforderliche (Need-to-Know-Prinzip) Personen Zugriff auf den im Auftrag erstellten Sourcecode besitzen.
6. Den im Auftrag erstellten Sourcecode bei Bedarf für die Durchführung einer Sicherheitsuntersuchung zur Verfügung zu stellen („Right to Audit“).
7. Auf eine kostenneutrale und zeitnahe Korrektur identifizierter Sicherheitsprobleme (vgl. Vorgaben in Kapitel 2). Was hierbei ein zu korrigierendes Sicherheitsproblem darstellt, wird einzig durch die Muster AG festgelegt.

[5] z.B. auf Basis von https://www.bsimm.com/about/bsimm-for-vendors

8 Implementierungsvorgaben

Die folgenden Vorgaben gelten für die Implementierung neuer webbasierter Anwendungen der Muster AG.

8.1 Allgemeine Grundsätze

1. *Minimierung der Angriffsfläche*: Nicht erforderliche oder benötigte Schnittstellen, Funktionen, Parameter, Dienste und Protokolle MÜSSEN auf extern erreichbaren Systemen und SOLLTEN auf allen sonstigen Systemen deaktiviert werden.
2. *Misstrauensprinzip*: Daten, die aus nicht vertrauenswürdiger Quelle stammen (z.B. von einem Browser) MÜSSEN stets misstraut und diese daher entsprechend validiert werden.
3. *Mehrschichtige Sicherheit* (Defense-in-Depth-Prinzip): Es SOLLTE stets eine mehrschichtige Sicherheit implementiert werden.
4. *Anpaßbarkeit von Sicherheitseinstellungen*: Sicherheit SOLLTE stets deklarativ (= mittels Konfigurationsanweisungen oder Annotationen) anstatt programmatisch (= mittels Programmcode) parametrisiert werden.
5. *Externalisierung von Sicherheitsfunktionen:* Wo dies möglich ist, SOLLTEN Sicherheitsfunktionen externalisiert werden (z.B. durch Nutzung vorhandener Authentifizierungssysteme).
6. *Konsistenz von Sicherheitsprüfungen:* Identische Sicherheitsprüfungen (z.B. einmal im Webfrontend und ein anderes Mal bei einer Rest-Schnittstelle) SOLLTEN stets über dieselben Sicherheitsfunktionen (= Programmcode) und Policies durchgeführt werden.
7. *Prinzip der ausgereiften Sicherheit:* Sicherheitsrelevanter Programmcode SOLLTE stets über ausgereifte Technologien, Algorithmen und Implementierungen (APIs, Frameworks, etc.) abgebildet werden.
8. *Testbarkeitsprinzip:* Vor dem Einsatz neuer Technologien (Protokollen, Frameworks, APIs, etc.) SOLLTE sichergestellt werden, dass sich diese auf mögliche Sicherheitsprobleme hin analysieren lassen (eingesetzte Tools etwa entsprechende Regeln besitzen).
9. *Verwende Secure Defaults:* Verwende stets sichere Defaults, um etwaige Fehler zu vermeiden.

8.2 Eingabevalidierung

1. Alle nicht-vertrauenswürdigen Eingaben (z.B. solche, die über externe Schnittstellen empfangen werden) MÜSSEN restriktiv validiert werden.
2. Eingabeprüfungen MÜSSEN stets serverseitig erfolgen, DÜRFEN jedoch entweder zusätzlich aus Usability-Gründen oder zur Abwehr client-seitiger Angriffe durchgeführt werden.
3. Eingabeprüfungen SOLLTEN (durch Einschränken von Datentyp, Länge und Wertebereich) möglichst restriktiv erfolgen, z.B.:
   a) Integer statt String-Datentyp,
   b) Min/Max-Einschränkung für numerische Datentypen oder
   c) eingeschränkte String-Typen (z.B. nur „a-z“ oder „A-Z“).

4. Eingabeprüfungen MÜSSEN auf sämtliche nicht-vertrauenswürdigen Parameter angewendet werden. Dies schließt neben Benutzerparametern auch Anwendungsparameter (z.B. mittels Hidden Form Fields oder HTTP Header wie z.B. Cookies) mit ein.
5. Dateipfade MÜSSEN stets normalisiert bzw. kanonisiert werden, bevor gegen diese eine Validierung durchgeführt werden. Hierbei werden Pfadtraversierungen („../../“) bereinigt.[6] Dies KANN durch Verwendung sicherer APIs erfolgen, welche entsprechende Traversierungen bereinigen.
6. Ein positives Validierungsmodell (Whitelisting) SOLLTE stets anstelle eines negativen (Blacklisting) verwendet werden.
7. Benutzereingaben SOLLTEN wenn möglich auch implizit mittels Data Binding (bzw. Type Casting) erfolgen.
8. Die Validierung von Anwendungsparametern SOLLTE, sofern möglich, implizit über Integritätsprüfungen oder Indirektionen erfolgen.
9. HTML-Eingaben MÜSSEN über eine ausgereiften HTML-Sanitizer-API restriktiv bereinigt werden.
10. Auch JSON- oder XML-Daten aus nicht-vertrauenswürdigen Quellen MÜSSEN mit restriktiv validiert werden. Dies KANN entweder direkt per XML-Schema oder indirekt (z.B. mittels Bean Validation) erfolgen.
11. XML-Parser, welche Daten aus nicht vertrauenswürdiger Quelle parsen, MÜSSEN wie folgt gehärtet werden, um gängige XML-basierte Angriffe zu unterbinden:
    a) Setzen restriktiver Limits (z.B. im Hinblick auf Verschachtelungstiefe oder Dokumentgröße).
    b) Deaktivierung externer XML Entitäten („External XML Entities“).
12. Zur Verhinderung unsicherer Deserialisierung von Objekten MÜSSEN eingelesene Objekte aus nicht vertrauenswürdigen Quellen vor Manipulation sensibler Attribute geschützt werden (z.B. durch Verzicht auf diese Attribute, mittels Whitelisting oder Integritätschecks).

8.3 Dateiuploads und -downloads

1. **Authentifizierung**: Jeder Upload einer Datei SOLLTE serverseitig authentifiziert werden und nur über den angemeldeten Bereich möglich sein. Dort wo dies nicht möglich ist, MÜSSEN ausreichende Anti-Automatisierungs-Mechanismen (z.B. mittels CAPTCHAs) implementiert werden, um DoS-Angriffe hinreichend zu unterbinden.
2. **Speicherung**:
    a) Hochgeladene Dateien SOLLTEN in einer zugriffsgeschützten Datenbank abgelegt oder
    b) Erfolgt die Ablage auf dem Dateisystem, so MÜSSEN dort die folgenden Vorgaben berücksichtigt werden:

[6] Häufig existieren hierfür Methoden wie `getCanonicalPath()`

1. Dateiuploads MÜSSEN so abgelegt werden, dass sie nicht von außen aufrufbar sind (z.B. außerhalb des Document bzw. Web Roots). Bei Anwendungen mit *Assuranceklasse* >= [HIGH] SOLLTEN Dateien in einem Benutzerspezifischen Verzeichnis abgelegt werden.
2. Dateiuploads MÜSSEN mit restriktiven Berechtigungen abgelegt werden (z.B. mittels des Unix-Kommandos „`chmod 0600`“).
3. Abgelegte Dateien DÜRFEN NICHT ausführbar sein.
4. Jeder Dateiupload SOLLTE die Erstellung einer neuen Datei und eindeutigen Dateinamen zur Folge haben.

3. **Limitationen**:
   a) Die Größe hochgeladener Dateien MUSS auf einen sinnvollen Wert (z.B. 5 MB) beschränkt werden.
   b) Die Anzahl hochgeladener Dateien SOLLTE auf einen sinnvollen Wert beschränkt werden (z.B. 8 Dateien pro Stunde und Benutzer).
4. **Validierung**:
   a) Dateitypen, die ausführbaren Code enthalten können (z.B. „`.html`“, „`.js`“, "`.exe`“ oder „`.bat`“) DÜRFEN NICHT hochgeladen werden können.
   b) Die Prüfung MUSS auf Basis von Whitelisting (nur erlaubte Dateitypen werden zugelassen) durchgeführt werden.
   c) Zusätzlich zur Dateierweiterung MUSS der angegebene MIME-Type einer Datei verifiziert werden.
   d) Bei Anwendungen mit *Assuranceklasse* >= [HIGH: Zusätzlich zum MIME Type SOLLTE der Dateityp auf Basis des tatsächlichen Dateiinhaltes mittels geeigneter APIs geprüft werden (z.B. mittels `getImageSize()`).
5. **Sanitization**: Hochgeladene Dateien von nicht-vertrauenswürdigen Quellen (z.B. dem Internet) mit potentiell ausführbaren Inhalten MÜSSEN
   a) im Hinblick auf ausführbaren Code geprüft (z.B. Makros) und dieser entfernt werden
   b) mittels einer Antiviren-Prüfung auf möglichen Schadcode hin geprüft und im Positivfall verweigert werden.[7]
6. **Download**:
   a) Dateidownloads SOLLTEN über eine separate Origin erfolgen (z.B. „`files.example.com`“), um dadurch die Auswirkungen von ausgeführten Skriptcode zu begrenzen.
   b) Beim Download von Dateien SOLLTEN entsprechende Security Header gesetzt werden, um damit z.B. MIME Type Sniffing im Browser zu deaktivieren (siehe Anhang).

8.4 Ausgabevalidierung (Enkodierung & Escaping)

1. Backend-Zugriffe (z.B. auf Datenbanken) MÜSSEN mittels Prepared Statements, ORM (Object-Relational Mapping, z.B. Hibernate) oder einem vergleichbaren Verfahren zur

[7] Diese Funktion lässt sich mit der EICAR-Testdatei testen (siehe www.eicar.org)

Parametrisierung[8] abgebildet werden, sofern für den verwendeten Interpreter eine entsprechende API existiert.

2. Unabhängig davon, ob internen oder externen Ursprungs, MÜSSEN sämtliche in Prepared Statements verwendete Variablen stets parametrisiert werden (= explizit als Parameter ausgewiesen werden müssen[9]). Eine Konkatenierung (Aneinanderhängen) von Parametern ist in Interpreter-Aufrufen generell zu vermeiden.
3. Sollte keine API zur Parametrisierung zur Verfügung stehen oder deren Einsatz nicht möglich sein, MÜSSEN Parameter mit einer geeigneten API enkodiert werden (z.B. SQL Encoding).
4. Bevor benutzerkontrollierte Parameter in Webseiten ausgegeben werden, MÜSSEN diese zuvor mit einer für den jeweiligen Ausgabekontext geeigneten API und Methoden enkodiert werden:
   a) HTML-Kontext: HTML Entity Encoding
   b) JavaScript-Kontext: JavaScript Escaping
   c) CSS-Kontext: CSS Escaping
5. Für die Durchführung der Ausgabevalidierung SOLLTEN ausschließlich ausgereifte APIs und Frameworks zum Einsatz kommen.
6. Hierzu SOLLTE stets eine implizite Validierung durchgeführt werden. Im Frontend wird eine solche etwa durch viele Webframeworks (bzw. Template-Technologien) zur Verfügung gestellt, im Backend z.B. durch den Einsatz eines ORM-Frameworks.

8.5 Authentifizierung & Registrierung von Benutzern

1. Im Rahmen einer Registrierung MUSS sich ein Benutzer mit einem für der *Assuranceklasse* der Anwendung geeigneten Verfahren identifizieren:
   a) <= [STANDARD]: E-Mail-Adressen DÜRFEN verwendet werden.
   b) [HOCH]: Ein zusätzlicher Faktor (z.B. Handy) SOLLTE zum Einsatz kommen.
   c) [SEHR HOCH]: Eine persönliche Identifikation MUSS erfolgen
2. Die Identifikation eines Benutzers MUSS abgeschlossen sein, bevor sich dieser an der Anwendung anmelden darf.
3. Bei der Registrierung und Authentifizierung von externen Benutzern MÜSSEN auf externen Anwendungen geeignete Verfahren zum Einsatz kommen, die automatisierte Angriffe (z.B. Brute Forcing) unterbinden. Beispiele hierfür sind eingebaute Delays, oder CAPTCHAs (Anti-Automatisierungs-Techniken).
4. Für die Authentifizierung von Benutzern MÜSSEN der jeweiligen *Assuranceklasse* angemessene Verfahren zum Einsatz kommen:
   a) <= [STANDARD]: Passwort-basierte Authentifizierung über sicheres Verfahren bzw. Protokoll. Passwörter müssen konform zur Passwort-Policy sein (NIST Authenticator

[8] Hierbei werden alle Parameter explizit als Parameter ausgewiesen (z.B. mittels einer API-Methode `setParam()`) oder implizit über ein Framework.
[9] z.B. mittels API-Methode `setParam()` oder ähnlichem

Assurance Level 1[10]).

b) [HOCH]: Wie beim normalen Schutzbedarf, jedoch in Verbindung mit einem zusätzlichen (Authentifizierungs-)Faktor. Der zusätzliche Faktor MUSS über einen anderen Kanal oder Kommunikationsschicht übertragen werden, darf sich jedoch auf demselben System befinden („Soft Crypto Token“). Beispiele hierfür sind: X.509-Zertifikate, zeitgesteuerte One Time Tokens (z.B. Google Authenticator App) oder Codes die per SMS oder E-Mail zugesendet werden (NIST Authenticator Assurance Level 2).

c) [SEHR HOCH]: Wie hoher Schutzbedarf, jedoch muss der zusätzliche Faktor über ein separates Gerät vom Benutzer generiert werden, welches speziell für diesen Zweck dient und freigegeben wurde („Hard Crypto Token“). Beispiele sind: RSA SecureID Tokens (NIST Authenticator Assurance Level 3).

5. HTTP Basic Auth SOLLTE nur als zusätzliche Zugriffskontrolle zum Einsatz kommen und DARF ausschließlich über HTTPS verwendet werden können.
6. Benutzernamen SOLLTEN frei wählbar und personen-spezifisch sein
7. Schlägt eine Anmeldung fehl, so MUSS eine neutrale Fehlermeldung ausgegeben werden, die keinen Aufschluss auf die Fehlerursache (ob Passwort oder Benutzername falsch) liefert. Positives Beispiel: „Benutzername oder Passwort fehlerhaft.“.
8. Die Autovervollständigung in Anmeldefeldern SOLLTE stets unterbunden werden. In den entsprechenden HTML-Input-Feldern ist hierzu das Attribut `autocomplete="off"` zu setzen.

8.6 Benutzerpasswörter I: Stärke und Behandlung

1. Benutzerpasswörter MÜSSEN der Passwort-Policy entsprechen. Dies MUSS sowohl bei der Registrierung von Benutzern als auch beim Neusetzen des Passwortes durch den Benutzer geprüft werden.
2. Sofern durch eine Passwort-Policy nicht anders vorgegeben, MÜSSEN Benutzerpasswörter
   a) mindestens 8 Zeichen Länge besitzen,
   b) sowohl aus Buchstaben, Zahlen und Sonderzeichen enthalten,
   c) nicht identisch mit dem Benutzernamen sein,
   d) maskiert mittels HTML-Passwort-Feldern eingegeben werden,
   e) nicht protokolliert oder zwischengespeichert werden,
   f) nur verschlüsselt über unsichere Kanäle (z.B. dem Internet) übertragen werden,
   g) nicht in URLs übertragen werden und
   h) nur mittels sicherer Verfahren, vorzugsweise als Salted Hash und mittels Key Stretching gespeichert werden, so dass diese auch dann noch ausreichend geschützt sind, wenn sie in die Hände Unbefugter gelangen sollten. Hierzu SOLLTE scrypt, bcrypt, PBKDF2 oder Argon2 verwendet werden.

[10] Siehe NIST SP 800-63b, NIST Special Publication 800-63B Digital Identity Guideline, Authenticator Assurance Levels: (https://pages.nist.sgov/800-63-3/sp800-63b.html#sec4)

3. Initialpasswörter MÜSSEN beim ersten Login durch den Benutzer geändert werden.
4. Standardpasswörter (= durch den Hersteller festgelegt) DÜRFEN NICHT verwendet werden und SIND durch individuelle Passwörter zu ersetzen.

8.7 Benutzerpasswörter II: Änderung und Zurücksetzung

1. **Passwort-Änderungs-Funktionen**:
   a) Benutzerpasswörter MÜSSEN durch den Benutzer geändert werden können.
   b) Bei Änderung eines Passwortes durch einen Benutzer SOLLTE diesem die aktuelle Stärke des eingegebenen Passwortes visuell angezeigt werden (Passwort-Stärke-Funktion).
   c) Beim Ändern seines Passwortes MUSS ein Benutzer sein altes Passwort zur Bestätigung eingeben.
   d) Benutzer SOLLTEN über die Änderung ihres Passwortes informiert werden (z.B. per E-Mail-Mitteilung).
2. **Passwort-Vergessen-Funktionen:**
   a) MÜSSEN das gleiche Sicherheitsniveau wie die Anmeldefunktion besitzen und genauso vor missbräuchlicher Verwendung geschützt werden.
   b) MÜSSEN über die E-Mail-Adresse (oder ein anderes Verfahren mit einem vergleichbaren oder höheren Sicherheitsniveau wie z.B. MFA) durch den Benutzer autorisiert werden.
   c) DÜRFEN KEINE Änderung des Profils eines Benutzers zur Folge haben (z.B. dessen Deaktivierung), solange diese Autorisierung nicht erfolgreich war.

8.8 Absicherung des Session Managements

Wird das Session Management über Standardkomponenten abgebildet? Dann hier entsprechend erwähnen.

1. Für die Abbildung des Session Managements MUSS eine Standardimplementierung (z.B. die des Application Servers oder Web Containers) eingesetzt werden.
2. **Session-IDs** MÜSSEN:
   a) Mindestens 120 Bit Länge besitzen,
   b) mit einem sicheren Cryptographically Secure Pseudo Random Number Generator (CSPRNGs) erstellt werden und vollständig zufällig sein,
   c) immer über sichere Kanäle (TLS/HTTPS)
   d) nach jeder Authentisierung durch den Benutzer (insbesondere beim Login) neu gesetzt werden,
   e) NICHT in der URL übertragen werden.
3. **Session Cookies** MÜSSEN in ihrer Gültigkeit weitestgehend eingeschränkt werden:
   a) Verwendung der Security-Attribute wie „`httpOnly`“, „`secure`“ und “`SameSite`“
   b) Vermeidung von persistenten Cookies (kein gesetztes Expire-Attribut) sowie
   c) über das Path-Attribut eingeschränkt werden, wenn mehre Anwendungen auf dem

gleichen Host ausgeführt werden.

4. **Authentifizierte serverseitige Sessions**
   a) MÜSSEN invalidiert werden, nachdem sich ein Benutzer abgemeldet hat,
   b) MÜSSEN spätestens nach 30 Minuten Inaktivität (Idle- oder Soft-Logout) invalidiert werden,
   c) SOLLTEN spätestens nach 24 Stunden invalidiert werden (Hartes Timeout, bzw. Session Lifetime) und
   d) SOLLTEN pro Benutzer nur einmal existieren. Meldet sich ein Benutzer erneut an der Anwendung an, SOLLTEN alle bestehenden Session-Objekte für diesen Benutzer invalidiert werden.
5. Bei ändernden Anfragen (Create, Update, Delete), die über eine authentifizierte Benutzersession durchgeführt werden, MUSS das Wiedereinspielen (Session Replay) sowie Unterschieben von Anfragen durch Dritte (Cross-Site Request Forgery, CSRF) verhindert werden.
   a) Hierfür lassen sich z.B. zufällige Anti-Replay-Tokens (bzw. Anti-CSRF-Tokens) verwenden, die als zusätzlicher Parameter (z.B. als zusätzliches Hidden-Field oder X-Header) eingebaut werden.
   b) Eine ändernde Anfrage DARF NUR in Verbindung mit einem gültigen Token zugelassen werden.
   c) Viele Webframeworks bieten bereits entsprechende Schutzfunktionen die bevorzugt statt der Implementierung eigener Funktionen verwendet werden SOLLTEN.
   d) Ändernde Anfragen DÜRFEN NICHT mit HTTP GET möglich sein.

8.9 Zugriffskontrollen (Access Controls)

Werden Standardkomponenten gegen die oder durch die Anwendungen Benutzer authentifizieren (Policy Decision Point) oder von denen Sie Rollen abrufen (Policy Information Point)? Dann hier entsprechend erwähnen.

1. Jeder sensible Objektzugriff (z.B. auf ein Datenbankobjekt) SOLLTE stets dediziert serverseitig autorisiert werden (Complete-Mediation-Prinzip).
2. Zugriffsprüfungen SOLLTEN sowohl auf URL-, also auch auf Datei-, Methoden- und Objekt-Ebene durchgeführt werden.
3. Zugriffsprüfungen MÜSSEN neben der erforderlichen Rolle einer Entität auch die erforderlichen Rechte prüfen, um auf ein bestimmtes Datenobjekt zuzugreifen.
4. Jeder Prozess und jede Rolle SOLLTE NUR die notwendigen Berechtigungen auf Ressourcen (Dateisystem, Datenbank, etc.) besitzen (Least-Privilege-Prinzip).

8.10 Fehlerbehandlung & Logging

1. Es MUSS sichergestellt werden, dass die Anwendung beim Auftreten eines Fehlers niemals in einen unsicheren Zustand fallen kann und Benutzern in einem solchen Fall

keine technischen Details (z.B. durch Ausgabe von Stack Traces) angezeigt werden.[11]

2. Im Fall von Sicherheitsfehlern SOLLTEN Security Exceptions geworfen werden.
3. Sicherheitsrelevante Zugriffe und Ereignisse (z.B. Anmeldung von Benutzern) MÜSSEN stets geloggt werden.
4. Anwendungen SOLLTEN Mechanismen besitzen um Missbrauch und Angriffe erkennten zu können (Application IDS).
5. Die folgenden Informationen SOLLTEN mit jedem Security Event geloggt werden:
   a) Security Tag („SEC“)
   b) Timestamp
   c) Beschreibung des Events
   d) Komponente
   e) Source IP
   f) Benutzername
   g) (keine weiteren personenbezogenen Informationen)
6. Technische Logs DÜRFEN keine personenbezogene daten enthalten

8.11 Datensicherheit & Kryptographie

Existieren bereits Kryptovorgaben im Unternehmen? Dann hier referenzieren.

1. Es DÜRFEN nur ausgereifte und standardisierte kryptographische Algorithmen, Betriebsmodi, Schlüsselstärken, SSL/TLS-Ciphers und Implementierungen eingesetzt werden.[12]
2. **Übertragung von Daten im Allgemeinen**:
   a) Generell SOLLTEN alle Zugriffe auf sensible Daten (intern oder extern) stets über HTTPS erfolgen.
   b) Erfordert eine Anwendung die Verwendung von HTTPS, MÜSSEN alle Zugriffe mittels http auf den entsprechenden HTTPS-Kontext weitergeleitet werden. Hierzu SOLLTE eine permanente Weiterleitung verwendet werden.[13]
   c) Ein HTTPS-Server MUSS aktuelle Cipher und Protokolle mit sehr hoher Sicherheit unterstützen. Unsichere SSL/TLS-Cipher (z.B. RC4-basierte) und -Protokolle (z.B. SSLv2, SSLv3) MÜSSEN deaktiviert und sichere unterstützt werden.
   d) Vertrauliche Daten im HTTP Request Body (z.B. mittels HTTP-POST-Methode) und nicht innerhalb der URL (Ausnahme: Objekt-IDs),
3. **Die Übertragungen über nicht-vertrauliche Netze** (z.B. dem Internet) MUSS
   a) stets verschlüsselt (mittels TLS/HTTPS) und mit gültigen Zertifikaten,
   b) mittels HSTS-Headern und

[11] Ein unsicherer Zustand ist dadurch gekennzeichnet, dass Sicherheitsfunktionen nicht mehr wie vorgesehen arbeiten und etwa einem nicht autorisierten Benutzer Zugriff auf sensible Daten gewährt wird.
[12] Siehe hierzu BSI Technische Richtlinie TR-02102-1, „Kryptographische Verfahren: Empfehlungen und Schlüssellängen“
[13] Siehe hierzu 8.13 zu HTTP Strict Transport Security (HSTS)

c) mittels gesetzter Anti-Caching-Header (siehe Anhang) erfolgen.

4. **Encryption at Rest:**
   a) Vertrauliche Daten MÜSSEN bei Speicherung in Cloud-Diensten oder auf dem Client stets verschlüsselt werden.
   b) Benutzerpasswörter MÜSSEN durch geeignete kryptographische Verfahren gespeichert werden (siehe 8.6).

8.12 Schutz von Secrets

The folgenden Anforderungen gelten für Secrets (z.B. Passwörter technischer User, API Keys, Credentials, Private Keys) die entweder in der Produktion verwendet oder für den Zugriff auf sensible Daten dienen:

1. Secrets MÜSSEN verschlüsselt werden, wenn diese zusammen mit dem Sourcecode abgelegt werden (andernfalls müssen diese vom Sourcecode separiert werden).
2. Der Zugriff auf Secrets MUSS vor unberechtigtem Zugriff geschützt werden (etwa durch restriktive Zugriffsrechte).
3. Für *Assuranceklasse* >= [HOCH]:
   a) Secrets MÜSSEN in einem abgesicherten Secret Store (Vault) bzw. Keystore verwaltet werden.
   b) Secrets MÜSSEN stets verschlüsselt werden.
   c) Secrets SOLLTEN mindestens einmal pro Jahr rotiert werden.

8.13 Clientseitige Sicherheit

1. Clientseitig sollten nur sichere JavaScript-APIs (kein ActiveX oder Java Applets) zum Einsatz kommen:
   a) JSON-Code DARF ausschließlich mit einer sicheren JavaScript-API wie `JSON.parse()`, nicht jedoch `eval()`, interpretiert werden.
   b) Anstelle der JavaScript-APIs mit „`.innerHTML`“ im Namen, SOLLTEN stets sichere APIs wie „`.innerText`“ und „`.textContent`“ genutzt werden, sofern nicht explizit HTML-Markup ausgegeben werden soll. Gleiches gilt auch für Webframeworks, die entsprechende APIs zur Verfügung stellen.
2. HTTP-Security-Header MÜSSEN entsprechend der Vorgaben aus Anhang A implementiert werden.
3. Clientseitig (z.B. im Browser) DARF KEINE sensible Geschäftslogik (z.B. Berechnungslogik von Tarifen) gespeichert werden.
4. User States oder andere Meta-Informationen, die clientseitig gespeichert wird, MUSS über einen entsprechenden Integritätsschutz (z.B. signierte JWT Tokens) für Manipulation geschützt werden.

8.14 Service und API Security

Existieren bereits Services (z.B. REST APIs, WebServices) im Unternehmen über welche bestimmte (perimetrische) Standardkomponenten wie XML-Firewalls / -Gateways abgebildet

werden? Dann hier entsprechend erwähnen.

Bitte beachten: Aus Gründen der Vereinfachung, wird in einigen der folgenden Anforderungen lediglich von „Services“ gesprochen. Hiermit werden jedoch sämtliche Implementierungen, inklusive XML Webservices oder REST APIs einbezogen.

1. Externe Services SOLLTEN ausschließlich über ein gehärtetes Service/API Gateway verfügbar gemacht werden.
2. **Service Authentifizierung**
   a) Die Stärke des verwendeten Authentifizierungsverfahren MUSS dem jeweiligen Schutzbedarf angemessen sein.
   b) Shared Secrets (z.B. API Keys oder OAuth 2.0 Client Secrets), die zur Authentifizierung von Services verwendet werden, MÜSSEN die folgenden Eigenschaften aufweisen:
      1. Länge min. 32 Zeichen (= 256 Bit)
      2. Kryptographische Zufälligkeit (mit einem sicheren Generator generiert)
      3. Speicherung gemäß der Vorgaben zu kryptographischen Schlüsseln (siehe 8.12)
      4. Übertragung außerhalb von URLs
   c) Services mit *Assuranceklasse* >= [HOCH] SOLLTEN asymmetrisch authentifizieren (z.B. signierte JWT Tokens oder X.509-Zertifikate).
   d) Authentifizierungs-Kennungen MÜSSEN pro Umgebung (Test, Produktion, etc.) eindeutig sein.
3. **Access Tokens**
   a) Zugriffe auf Services mit *Assuranceklasse* >= [HIGH] SOLLTEN mittels Access Tokens (OAuth 2.0 oder SAML) geschützt werden.
   b) An Access Tokens gelten die folgenden Anforderungen:
      1. Kurzlebig
      2. Eingeschränkter Scope
      3. Ausschließlich über HTTPS übertragen
      4. Ausgestellt von einem vertrauenswürdigen und gehärteten Server (z.B. OIDC Identity Server oder OAuth 2.0 Authorization Server)
      5. Erstellt und verifiziert bei ausgereiften APIs
4. Anforderungen für **OAuth 2.0/OIDC**
   a) Nur 3-Legged: Authorization Code Grant mit PKCE (oder eine alternative State-Binding-Technik wie z.B. OIDC Nounces) SOLLTE sowohl Public Clients (z.B. SPAs)[14]als auch Confidential Clients (z.B. serverseitige Webapps)[15] verwendet werden.
   b) Nur 3-Legged: Zum Schutz gegen CSRF-Angriffe MÜSSEN n “`state`“-Parameter oder PKCE oder OIDC Nounces verwenden.

[14] Siehe https://tools.ietf.org/html/draft-ietf-oauth-browser-based-apps-00
[15] Siehe https://tools.ietf.org/html/draft-ietf-oauth-security-topics-14

c) Nur 3-Legged: Clients MÜSSEN eine vollständige Redirect-URL gemäß RFC 6819 Section 5.2.3.5 registrieren und über diese validiert werden (kein Pattern Matching).

d) Service-to-Service-Zugriffe SOLLTEN mittels OAuth 2.0 Client Credential Grant geschützt werden.

e) Jede Kommunikation MUSS über HTTPS erfolgen.

f) Siehe obige Anforderungen an Access Tokens.

5. **Web UI APIs:**
   a) MÜSSEN dieselben Sicherheitsanforderungen umsetzen wie die entsprechende Web GUI.
   b) Enthält die Antwort eines Services Skriptcode (z.B. JSON bzw. JavaScript), so MUSS sichergestellt werden, daß dieser nicht direkt ausführbar ist.
   c) MÜSSEN einen CERF-Schutz implementieren wenn diese im User-Kontext agieren.
6. **Cross-Domain-Zugriffe:**
   a) MÜSSEN über sichere Verfahren (z.B. Cross-Origin Ressource Sharing, CORS) und minimaler Berechtigungen (einschränken der erlaubten Origins auf einzelne Hosts oder Domains) durchgeführt werden.
   b) MÜSSEN serverseitig im Hinblick auf ihre Herkunft autorisiert werden (z.B. durch Auswerten des Origin Headers).
7. Services SOLLTEN restriktive Limits (max. Anzahl pro Client pro Zeiteinheit) implementieren.
8. WebSockets, über die vertrauliche Daten übertragen werden können, MÜSSEN das wss://-Schema verwenden und eine Prüfung des Origin-Headers durchführen.

8.15 X.509-Zertifikate („SSL-Zertifikate“)

1. Alle externen HTTPS-Verbindungen MÜSSEN valide X.509-Zertifikate verwenden, die von einer vertrauenswürdigen Certificate Authority (CA) ausgestellt wurden.
2. Alle X.509-Zertifikate MÜSSEN eine Schlüssellänge von mindestens 2048 Bit bei RSA bzw. 256 Bit bei ECC besitzen.
3. Externe Kundenanwendungen (UIs) sollten EV-Zertifikate nutzten.

Anhang A: Vorgaben für HTTP Security Header

In modernen Browsern lassen sich verschiedene Schutzfunktionen über gesetzte HTTP Response Header aktivieren und dadurch das Sicherheitsniveau einer Webanwendung erhöhen. Im Folgenden sind entsprechende Vorgaben und Empfehlungen für **extern erreichbare produktive Webanwendungen** der Muster AG beschrieben.

Zentral zu setzende Header:

Response Header	Allgemeine Empfehlung	Wann?
Content-Type	`...; charset=utf-8`	Immer
Strict-Transport-Security[16]	`max-age=10886400; includeSubDomains; preload`	Für alle Webanwendungen, auf denen sich die Verwendung von HTTPS erzwingen lässt.
X-XSS-Protection[17]	`1; mode=block`	Immer
X-Frame-Options[18]	`SAMEORIGIN`	Immer
Referrer-Policy	`same-origin`	Immer
X-Content-Type-Options[19]	`nosniff`	Immer

Anwendungsspezifisch zusetzende Header und Attribute:

Response Header	Allgemeine Empfehlung	Wann?
Set-Cookie	`… ;httpOnly; secure; SameSite`	Übertragung sensibler Daten in Cookies (z.B. Session-IDs).
Cache-Control	`no-cache, no-store`	Bei Übertragung sensibler Daten.
Pragma	`no-cache`	
Expires	`-1`	
Content-Security-Policy[20]	`script-src 'self' [URL1] [URL2]; style-src 'self' unsafe-inline; object-src 'self';`	Allgemeine Empfehlung für neue Anwendungen

[16] HTTP Strict Transport Security (HSTS): Forciert, dass der Benutzer die Webseite zukünftig und entsprechend des angegebenen Zeitraums nur noch mittels HTTPS aufruft. Bei Verwendung dieses Headers muss sichergestellt sein, dass alle Zugriffe auf den Host (bzw. auch alle Subdomains bei Verwendung von „includeSubDomains“) per HTTPS möglich sind. Hiermit werden bestimmte „Man-in-the-Middle“-Angriffe abgewehrt.

[17] Additiver Schutz vor Cross-Site Scripting (nur für IE).

[18] Unterbindung von „Framing“ und Schutz vor Clickjacking

[19] Deaktiviert MIME-Sniffing im Browser, wodurch dieser versucht, den MIME-Type anhand des Inhaltes (und nicht des angegebenen MIME-Types) zu identifizieren.

[20] Content Security Policy (CSP): Additiver, aber sehr wirksamer Schutz vor Cross-Site Scripting und anderen clientseitigen Angriffen, der jedoch eine Kompatibilität des eingesetzten Frameworks erfordert und bereits frühzeitig innerhalb der Entwicklung berücksichtigt werden muss.

Response Header	Allgemeine Empfehlung	Wann?
	`script-src 'unsafe-inline' 'unsafe-eval' 'nonce-[NONCE]' 'strict-dynamic'` [URL1] [URL2]; `object-src 'none';`	Empfehlung für neue Webanwendungen, welche inline Script-Bereiche verwenden müssen. Diese Einstellung sollte nicht verwendet werden, wenn dies nicht erforderlich ist, da hierbei der CSP-Schutz für alte Browser deaktiviert wird.
Content-Disposition	`attachment; filename=<dateiname>`	Bei potentiell nicht vertrauenswürdigen Dateidownloads.
X-Download-Options	`noopen`	

<u>Achtung</u>: Das Setzen dieser Header kann Auswirkungen auf die korrekte Funktionsfähigkeit einer Webanwendung zur Folge haben. Daher SOLLTE die Verwendung eines neuen Headers stets nur in Verbindung mit ausführlichen funktionalen Tests erfolgen.