

Sicherheitsuntersuchung der Sympa Mailinglist Manager Instanz der Landeshauptstadt München

07.06.2025, Version v1.0, Status: Final

Auftraggeber:
Jüren Bilberger, juergen.bilberger@muenchen.de
Landeshauptstadt München
Agnes-Pockels-Bogen 33, 80992 München

Autoren:
Reinhard Böhme, reinhard.boehme@mgm-sp.com
Mirko Richter, mirko.richter@mgm-sp.com

Inhalt vertraulich!

A. Zusammenfassung und Bewertung

Sprungtabelle

	Abschnitt	Seite
Inhaltsverzeichnis	A1	3
Management Summary	A2	6
Übersicht aller Findings	A3	8
Detailergebnisse nach Schwachstellen	C	26

Auf den folgenden Seiten sind die Ergebnisse der Sicherheitsanalyse zusammengefasst.



Alle Verweise in diesem Dokument sind aktive Links, d. h. können per Mausklick direkt angesprungen werden.

A1 Inhaltsverzeichnis

A.	Zusammenfassung und Bewertung	2
A1	Inhaltsverzeichnis	3
A2	Management Summary.....	6
A3	Übersicht aller Findings	8
A3.1	Übersicht toolbasierte Findings	8
A3.2	Übersicht Findings des Penetrationstests	9
A4	Assessment Details	11
A4.1	Getestete Applikation	11
A4.2	Testzeitraum	12
A4.3	Testbenutzer.....	12
A4.4	Einschränkende Rahmenbedingungen.....	12
A4.5	Toolgestützte Analyse.....	13
B.	Methodik und Bewertung	16
B1.1	Beschreibung der Methodik.....	16
B1.2	Bewertungsschema	19
B1.3	Aufbau der Testergebnisse.....	21
B1.4	Urheberrecht.....	24
C.	Testergebnisse im Detail	26
C1	Datenvalidierung	26
C1.1	Cross-Site-Scripting.....	27
C1.2	Open Redirect	28
C1.3	SQL-Injection	28
C1.4	Command-Injection.....	28
C1.5	LDAP-Injection.....	29
C1.6	XML-Injection.....	29
C1.7	Remote-File-Inclusion (RFI)	29
C1.8	Datenvalidierung auf dem Client.....	29
C1.9	Path-Traversal.....	30
C2	Authentisierung	31
C2.1	Enumeration von Benutzernamen oder Passwort.....	32
C2.2	Passwort-Cracking	32
C2.3	Passwort-Struktur, -Qualität.....	33
C3	Session-Management.....	34
C3.1	Transportmedium.....	35
C3.2	Gültigkeit des Session-ID-Cookies.....	36
C3.3	Session-ID im URL oder Referrer (Referrer-Leak)	38

C3.4	Logout-Funktionalität.....	38
C3.5	Logout durch Timeout bei Inaktivität.....	39
C3.6	Session-Fixation.....	43
C3.7	Cross-Site Request Forgery (CSRF).....	46
C3.8	Zufälligkeit der Session-ID.....	47
C4	Zugriffsschutz.....	49
C4.1	Privilegienerweiterung.....	50
C4.2	Entity-Enumeration.....	51
C5	Konfiguration und Deployment.....	53
C5.1	HTTP vs. HTTPS.....	54
C5.2	Benennung des Content-Type.....	54
C5.3	Directory-Indexing.....	54
C5.4	Beispieldateien.....	55
C5.5	Serverkonfiguration für HSTS (HTTP Strict Transport Security).....	55
C5.6	Serverkonfiguration für Cross-Origin Requests (CORS).....	56
C5.7	Serverkonfiguration für Content-Security-Policy (CSP).....	57
C5.8	Serverkonfiguration für MIME-Sniffing.....	58
C5.9	Serverkonfiguration für Anti-Clickjacking.....	59
C5.10	No-cache-Anweisung.....	63
C5.11	Known-Vulnerabilities.....	63
C6	Logik.....	64
C6.1	„Passwort vergessen“-Funktion.....	65
C6.2	„Passwort ändern“-Möglichkeit.....	65
C6.3	Reauthentisierung.....	65
C6.4	Denial-of-Service durch Benutzersperrung.....	66
C6.5	Ungenügende Anti-Automation.....	66
C6.6	Virus-Prüfung beim Datei-Upload.....	67
C6.7	Akzeptierte Dateien und Dateiformate der Upload-Funktion.....	67
C7	Informationsabfluss.....	68
C7.1	Bekanntgabe von unterschiedlichen Systeminformationen.....	69
C7.2	Fehlerseiten.....	72
C8	Kryptographie.....	75
C8.1	TLS/SSL-Verschlüsselung.....	76
C8.2	TLS/SSL-Zertifikat.....	76
C9	Testergebnisse automatische Werkzeuge.....	77
C9.1	SAST- und SECRET-Werkzeuge.....	77
C9.2	SCA-Werkzeuge.....	80
C10	Allgemeine negative Beobachtungen bzgl. Code-Sicherheit und –Qualität.....	82
C10.1	Fehlendes Abhängigkeiten Management für JavaScript.....	82
C10.2	SQLi-Gegenmaßnahmen teilweise riskant.....	83
C10.3	„Taint Mode“ nicht durchgängig robust umgesetzt.....	84
C10.4	Duplikate im Code.....	86
C11	Allgemeine positive Beobachtungen bzgl. Code-Sicherheit und –Qualität.....	87

Ergebnisbericht
Sicherheitsanalyse Sympa Mailinglistmanager der Landeshauptstadt München

5

C11.1	Projekt-Struktur	87
C11.2	XSS-Gegenmaßnahmen	87
C12	Sonstige Beobachtungen	88
C12.1	Password im Klartext.....	88

A2 Management Summary

Der vorliegende Bericht fasst die Ergebnisse der Sicherheitsuntersuchung der

Sympa Mailinglist Manager Instanz der Landeshauptstadt München

in der Version v6.2.76 auf Ebene der Web Application Security zusammen.

Die Anwendung wurde einer toolbasierten Quellcode-Analyse, sowie einer dynamischen Analyse in Form eines Penetrationstests unterzogen. Die Untersuchung erfolgte nach der Whitebox-Methode, bei welcher die Tester Zugriff auf den Source-Code und auf eine laufende vorkonfigurierte Instanz der Anwendung hatten.

Die Tests fanden im Zeitraum vom **12.05. – 07.06.2025** aus den Räumlichkeiten des Auftragnehmers statt und wurden über das Internet durchgeführt.

Die dynamischen Tests erfolgten auf einer speziell vom Auftraggeber zur Verfügung gestellten und individuell vorkonfigurierten Installation. Funktionalitäten, welche auf dieser Instanz nicht zur Verfügung standen, waren somit kein Bestandteil der Sicherheitsuntersuchung.

Ergebnis

Die Sympa-Instanz der Landeshauptstadt München weist zwei

Sicherheitslücken mittlerer Kritikalität

auf, mit denen ein Angreifer gezielt Benutzer kompromittieren kann.

Schwachstellenübersicht

Die Untersuchung ergab Schwächen im allgemeinen Umgang und der Verwaltung der Benutzersessions der Sympa-Anwendung.

Es konnte festgestellt werden, dass das Sitzungscookie nach dem Login nicht konsequent erneuert wird und so Session-Fixation-Angriffe ermöglicht werden (s. C3.6.1). Damit ist es einem Angreifer möglich, eine ihm bekannte Sitzung eines anderen Nutzers zu übernehmen, indem er dessen Browser diese aufzwingt. Da die Sympa-Anwendung weiterhin das Sitzungscookie nicht ausreichend schützt und ihm kein Ablaufdatum zuweist (s. C3.2.1), werden solche Angriffsszenarien erleichtert. Erschwerend kommt ein fehlender serverseitiger Timeout für Sitzungen hinzu (s. C3.5.1).

Ein Angreifer im Besitz eines fremden Sitzungscookies kann so den Account des legitimen Nutzers bis zu seinem nächsten aktiven Logout missbrauchen. Darüber hinaus könnte ein Angreifer das Sitzungscookie aufbewahren und nach dem nächsten Login desselben Nutzers Zugriff auf dessen Konto erhalten.

Bei der werkzeuggestützten Analyse sind zwei Stellen mit mittlerer Kritikalität aufgefallen, an denen der als schwach anzusehende Hash-Algorithmus MD5 im kryptografischen Kontext zum Einsatz kommt.

Darüber hinaus verwendet die Anwendung stark veraltete JavaScript-Abhängigkeiten. Dies führen wir auf ein fehlendes JavaScript Abhängigkeiten-Management zurück (s. C10.1).

Im Code wurden desweiteren potenziell gefährliche Implementierungen aufgefunden. So sind die Gegenmaßnahmen für SQL-Injection-Angriffen nicht konsequent umgesetzt (s. C10.2). Weiterhin konnte festgestellt werden, dass das „Untainting“ mit Perls „Taint Mode“ nicht durchgehend robust gestaltet wurde (s. C10.3).

Auffällig ist zudem die recht große Menge an Code-Duplizierungen (s. C10.4). Hier sollte ein Refactoring in Betracht gezogen werden, um auch zukünftig die generelle Erweiterbarkeit und insbesondere das effektive Handling von Schwachstellen (Identifikation und Härtung) zu gewährleisten.

A3 Übersicht aller Findings

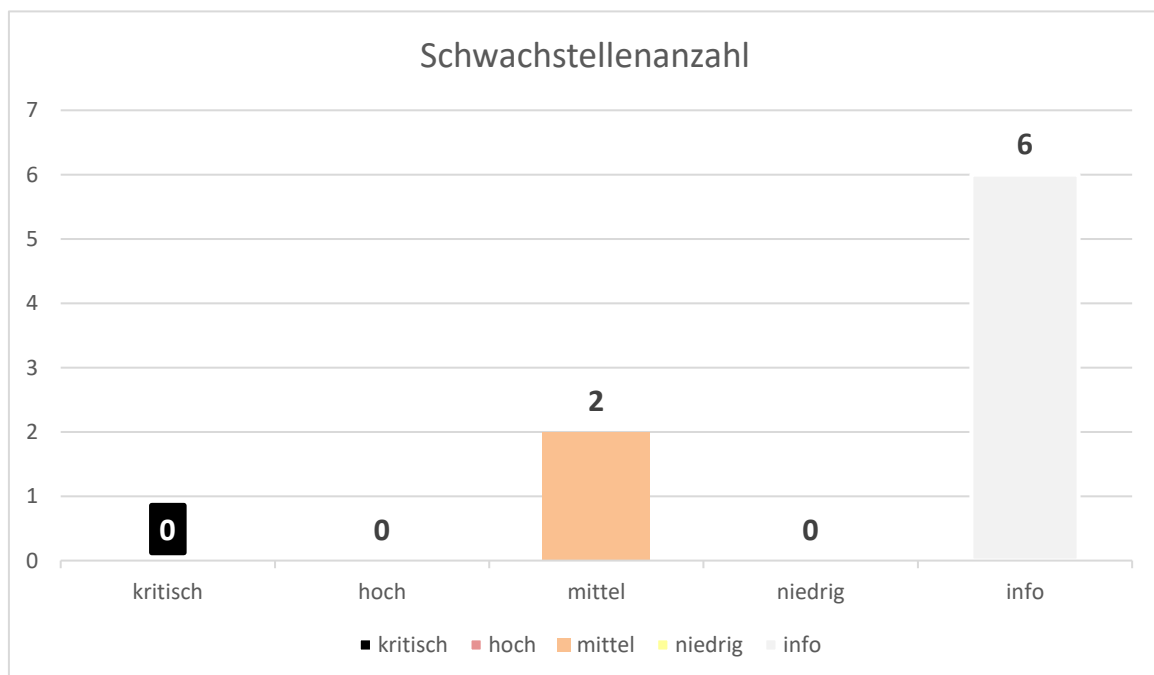
A3.1 Übersicht toolbasierte Findings

Dem in Kapitel B1.3.1 unten beschriebenen Vorgehen zur Bestimmung des Gefährdungspotentials folgend, werden in diesem Kapitel alle Ergebnisse automatischer Toolanalysen zusammengefasst dargestellt.

Hinweis: Die SARIF-Exporte können sich in der Menge von den Findings im Excel-Report unterscheiden, da unter Umständen bestimmte Findings von einer Bewertung ausgeschlossen wurden (Details dazu siehe Abschnitt C9 unten).

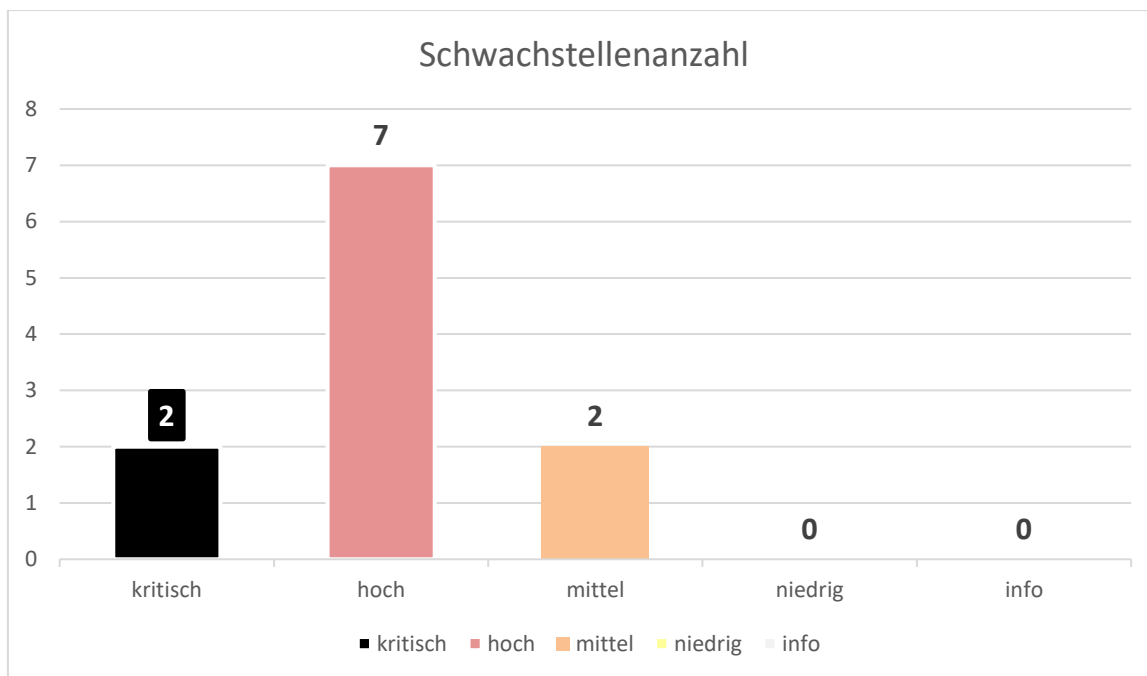
A3.1.1 SAST-Analysen

Auf Basis der durchgeführten semiautomatischen SAST- und SECRETS-Analysen (s. Abschnitte A4.5.1 und A4.5.2) ergeben sich, nach Abbildung auf unser Gefährdungspotential, folgende Mengen an Schwachstellen:



A3.1.2 SCA-Analyse

Auf Basis der automatischen SCA-Analyse (s. Abschnitt A4.5.3) ergeben sich, nach Abbildung auf unser Gefährdungspotential, folgende Mengen an anfälligen Abhängigkeiten:



A3.2 Übersicht Findings des Penetrationstests

In der folgenden Übersicht sind die Bedrohungen und Schwachstellen aufgeführt, welche durch manuelle DAST-Ansätze aufgefunden wurden.

In der folgenden Übersicht sind die Bedrohungen und Schwachstellen aufgeführt, die das Gefährdungspotenzial **niedrig**, **mittel**, oder **hoch** besitzen.

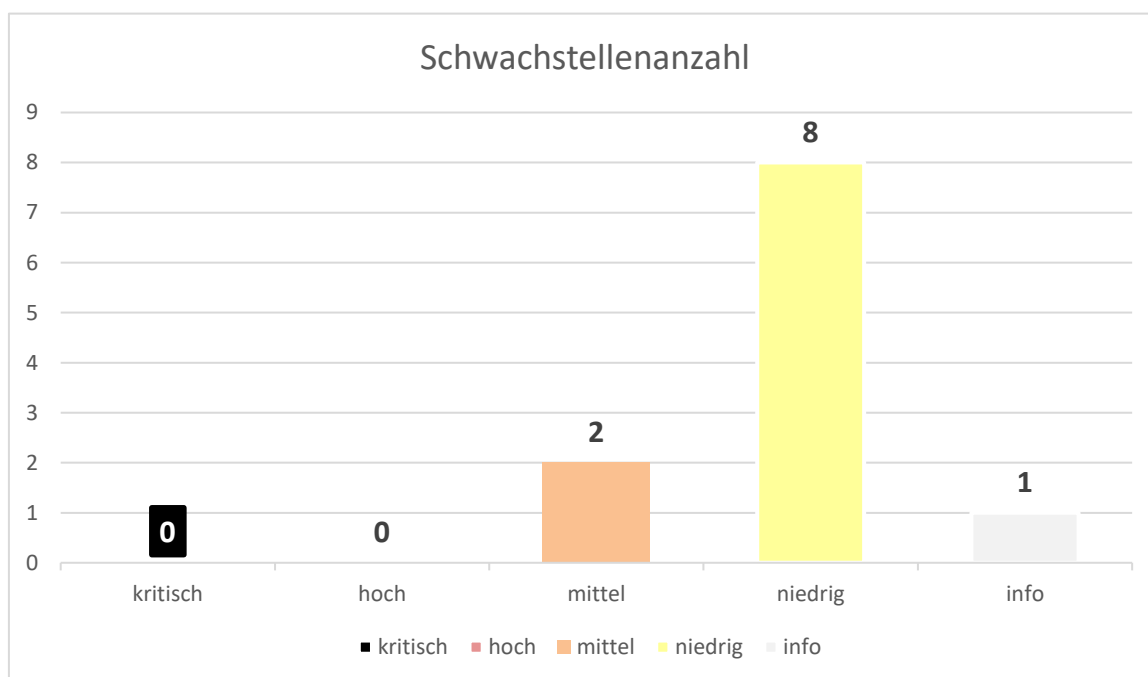
Schwachstelle	[niedrig]	[mittel]	[hoch]
Datenvalidierung			
-			
Authentisierung			
-			
Session-Management			
Schutz des Sitzungscookies mit Verbesserungspotenzial	C3.2.1		
Kein Logout bei Inaktivität		C3.5.1	
Session Cookie wird nach dem Login nicht aktualisiert		C3.6.1	
Zugriffsschutz			
Enumeration von Mailinglisten und -ownern möglich	C4.2.1		
Konfiguration und Deployment			
Keine Auslieferung des Content-Security-Policy-Headers	C5.7.1		

Schwachstelle	[niedrig]	[mittel]	[hoch]
Mime-Sniffing-Angriffe potenziell möglich	C5.8.1		
Einbetten der Anwendung in Fremdkontext möglich	C5.9.1		
Logik			
-			
Informationsabfluss			
Bekanntgabe von verwendeter Technologie und exakter Version	C7.1.1		
Bekanntgabe eines internen absoluten Pfades	C7.2.1		
Kryptographie			
-			

In der folgenden Übersicht sind die Informationen aufgeführt, die das Gefährdungspotenzial [info] besitzen.

[info]	Kapitel
Entropie des Sitzungscookies verbesserbar	C3.8.1

Es wurden insgesamt 10 Schwachstellen und eine Information gefunden.



A4 Assessment Details

A4.1 Getestete Applikation

Die dynamische Sicherheitsüberprüfung in Form eines Webapplikationspentests erfolgte auf der folgenden Webapplikation:

Akronym	URL	IP
Sympa Mailinglist Manager (v6.2.76)	https://pentest-extern.muenchen.de/sympa/	194.246.166.42

Für den dynamischen Analyseanteil wurde Sympa auf einer dedizierten Testumgebung der Landeshauptstadt München installiert. Sämtliche (Haupt-) Funktionalitäten, sofern nicht durch die Rahmenbedingungen verhindert, manuell getriggert und untersucht (siehe auch Methodik-Beschreibung in Abschnitt B), sowie alle mindestens als [hoch] eingestuft statischen Findings hinsichtlich ihrer tatsächlichen Ausnutzbarkeit nachgetestet.

Bei der SAST-Analyse wurde folgende Komponente auf Schwachstellen hin untersucht:

- Sympa, Mailing List Management Software (<https://github.com/sympa-community/sympa>) unter Verwendung des Tags 6.2.76 (<https://github.com/sympa-community/sympa/tree/6.2.76>)

Die zugrundeliegenden Mengengerüste der zum Einsatz kommenden Sprachen & Technologien (erstellt mit dem Werkzeug cloc - <https://github.com/AlDanial/cloc>) stellen sich folgendermaßen dar:

github.com/AlDanial/cloc v 2.04 T=12.81 s (33.9 files/s, 94349.0 lines/s)				
Language	files	blank	comment	code
PO File	129	187274	356387	505708
Perl	230	16994	20870	54926
JavaScript	16	7396	7839	29387
CSS	13	3012	339	11086
Markdown	10	731	23	1572
make	11	103	224	1363
Web Services Description	1	69	5	656
m4	1	54	0	620
C	6	72	157	337
Bourne Shell	2	50	51	301
YAML	8	31	37	239
Text	3	69	0	233
PHP	1	31	20	104
XML	2	2	0	102
JSON	1	0	0	74
SUM:	434	215888	385952	606708

A4.2 Testzeitraum

Die Tests wurden vom **12.05. – 07.06.2025** durchgeführt.

A4.3 Testbenutzer

Die folgenden Testbenutzer wurden vom Kunden bereitgestellt:

Benutzername	Rolle
[REDACTED]	Moderator
[REDACTED]	Moderator
[REDACTED]	Listmaster

Die Testbenutzer sollten nach dem Ende des Penetrationstestes wieder gelöscht oder deaktiviert werden. mgm-sp übernimmt keine Verantwortung für Schäden, die durch weiterhin vorhandene Testbenutzer entstehen.

A4.4 Einschränkende Rahmenbedingungen

Im Folgenden werden die für die Durchführung der Sicherheitsanalysen relevanten Einschränkung aufgeführt.

A4.4.1 Penetration Test

Anbindung des Sympa an den Single-Sign-On (SSO) der Landeshauptstadt München

Die Anbindung des Sympa an den Single-Sign-On (SSO) der Landeshauptstadt München ersetzt das eigene Loginsystem Sympas. Testfälle, welche den Login betreffen, wurden deshalb nicht ausgeführt. Dazu zählen:

- Nutzernamenenumeration
- Passwortcracking durch Brute-Force-Angriffe
- Denial-of-Service (DoS) durch Benutzersperrung
- Nutzernamenenumeration durch die „Passwort vergessen“ Funktion
- Ungenügende Anti-Automation im Login

Härtung der Einstellungen für das editieren von Mailinglisten

Durch die Landeshauptstadt München wurden Härtungsmaßnahmen für den Zugriff auf Verteilerlisten vorgenommen, welche von der Standardkonfiguration abweicht.

Funktionen, welche durch diese Maßnahmen nicht mehr zur Verfügung standen, wurden den Tests nicht unterzogen.

Einstellungen der SOAP-Schnittstelle

Da die SOAP-Schnittstelle der Sympa-Anwendung von der Landeshauptstadt München nicht aktiv genutzt wird, ist diese nicht konfiguriert. Die API-Endpunkte waren zwar

erreichbar, jedoch resultieren jegliche Zugriffe in einem Authentifizierungsfehler. Ein vollständiger Test konnte deswegen nicht durchgeführt werden.

Keine Untersuchung auf Infrastrukturebene

Die dynamische Sicherheitsuntersuchung erfolgte auf Webanwendungsebene, nicht aber auf Infrastrukture bzw. E-Mail-Infrastrukturebene. Testfälle, welche die Konfiguration des Mailservers oder des Hosts, auf welchem die Sympainstanz aufgesetzt wurde, waren nicht Bestandteil des Audits.

A4.4.2 Quellcodeanalyse

Festlegungen hinsichtlich der SAST-Analyse

- keine spezifische Untersuchung von Vagrant- / Docker-Releases und deren Konfiguration
- keine Untersuchung des Quellcodes von 3rd-Party-Abhängigkeiten
- keine Durchführung von Befragungen / Interviews der Projektbeteiligten (Leiter, Entwickler, etc.)
- keine Untersuchung von Demo- oder Test-Code bzw. Dokumentationen
- keine Bewertung von Findings in Entwicklungsskripten und Utilities, welche nicht für den produktiven Einsatz vorgesehen sind
- freie Auswahl der Analysewerkzeuge

A4.5 Toolgestützte Analyse

Die Anwendung würde einer eingehenden Sicherheitsuntersuchung unterzogen. Dafür kam eine Kombination folgender Herangehensweisen zum Einsatz:

- Automatische Quellcodeanalyse (SAST + SECRETS + SCA)
- Automatische und manuelle Penetrationstests (DAST)

Die konkret im jeweiligen Ansatz eingesetzten Werkzeuge werden in den folgenden Abschnitten aufgelistet.

Bei der Auswahl der Findings, welche im Rahmen dieser Analyse einem eingehenden manuellen Review unterzogen wurden, kam ein mit dem Auftraggeber abgestimmter "Best-Effort Ansatz" zum Einsatz. Die konkrete Auswahl erfolgte dabei u.a. auf Basis von Finding-Kategorien oder bestimmten Quellcode-Positionen, welche sich durch das konkrete Einsatzszenario beim Kunden ergeben. Für eine zusätzliche, effektivitätssteigernde Priorisierung kam zudem der LLM@SAST-Ansatz des Auftragnehmers zum Einsatz (weiterführende Informationen dazu z.B. unter <https://www.mgm-sp.com/neuer-glanz-fuer-sast>). Infolgedessen wurden zwar nicht alle Findings auditiert - die toolspezifischen Scan-Ergebnis-Ausgaben liegen diesem Report aber im SARIF-Format vollumfänglich bei.

Hinweis: Nicht alle Werkzeuge sind in der Lage, alle Sprachen und Frameworks umfassend zu analysieren. Die Werkzeugauswahl erfolgte durch den Auftragnehmer, jeweils passend zu den verwendeten Programmiersprachen und Frameworks.

A4.5.1 SAST: Eingesetzte Werkzeuge

Die zu analysierende Anwendung wurde mit folgenden SAST-Werkzeugen einem Scan unterzogen:

- ZARN (<https://github.com/htrgouvea/zarn>) - Standard-Regelsatz, Stand Mai 2025
 - Ergebnisse siehe Kapitel C9
- Snyk Code (<https://snyk.io/de/>) - Standard-Regelsatz, Stand Mai 2025
 - Ergebnisse siehe Kapitel C9
- CodeQL (<https://codeql.github.com/>) - Standard-Regelsatz, Stand Mai 2025
 - Ergebnisse siehe Kapitel C9

A4.5.2 SECRETS: Eingesetzte Werkzeuge

- GitLeaks (<https://gitleaks.io/>) – Standard-Regelsatz, Stand Mai 2025
 - Ergebnisse siehe Kapitel C9

A4.5.3 SCA: Eingesetzte Werkzeuge

- CPAN-Audit (<https://github.com/briandfoy/cpan-audit>) – Standard-Regelsatz, Stand Mai 2025
 - Ergebnisse siehe Kapitel C9

A4.5.4 DAST: Eingesetzte Werkzeuge

Für die dynamischen Analysen kam die Burp Suite Professional (<https://portswigger.net/burp>) in der Version v2025.5.1 zu Einsatz. Hierbei kam der „Burp Vulnerability Scanner“ (aktiv und passiv) umfassend zum Einsatz.

Alle darüber hinaus vom Tester als „aus Sicherheitssicht interessant“ eingestuften Requests wurden manuell (mit automatischer Werkzeug-Unterstützung) tiefgreifender analysiert. Dabei kamen (neben den eingebauten Hilfstools, wie Repeater oder Intruder) auch die folgenden Erweiterungen zum Einsatz:

- Logger++ v3.20.0
- SQLi Query Tampering v1.3
- ActiveScan++ v2.0.4
- Turbo Intruder v1.54
- Param Miner v1.5

— Autorize v1.8

Darüber hinaus wurden folgende Tools verwendet

— Feroxbuster v2.10.3 (<https://github.com/epi052/feroxbuster>)

— SoapUI v5.5.0 (<https://www.soapui.org/tools/soapui/>)

B. Methodik und Bewertung

B1.1 Beschreibung der Methodik

Der folgende Abschnitt befasst sich mit der den Testverfahren zu Grunde liegenden Methodik.

B1.1.1 Testverfahren (Whiteboxtest)

Die Anwendung wurde sowohl manuellen Tests, als auch automatischen Tests unterzogen. Zum Einsatz kamen verschiedene frei verfügbare Tools. Die Tests mit den Tools haben wir auf die Bereiche beschränkt, in denen die Stärken der Tools liegen. Die manuellen Tests erstrecken sich auf alle untersuchten Bereiche und erfassen insbesondere auch die Business-Logik der Anwendung.

B1.1.2 Art der Schwachstellenuntersuchung

Wir betrachten die im folgenden Bild gezeigten Ebenen (Aspekte) 2 bis 5 der Sicherheit der Webanwendungen:

	Ebene	Inhalt (Beispiele)
5	Semantik	Schutz vor Täuschung und Betrug - Informationen ermöglichen Social-Engineering-Angriffe - Gebrauch von Popups u. ä. erleichtern Phishing-Angriffe - Keine Absicherung für den Fall der Fälschung der Website
4	Logik	Absicherung von Prozessen und Workflows als Ganzes - Verwendung unsicherer E-Mail in einem ansonsten gesicherten Workflow - Angreifbarkeit des Passworts durch nachlässig gestaltete „Passwort vergessen“-Funktion - Die Verwendung sicherer Passwörter wird nicht erzwungen
3	Implementierung	Vermeiden von Programmierfehlern, die zu Schwachstellen führen - Cross-Site Scripting - SQL-Injection - Cross-Site Request Forgery (Session Riding)

2	Technologie	Richtige Wahl und sicherer Einsatz von Methoden - unverschlüsselte Übertragung sensibler Daten - Authentisierungsverfahren, die dem Schutzbedarf nicht angemessen sind - Ungenügende Entropie von Tokens
1	System	Absicherung der auf der Systemplattform eingesetzten Software - Fehler in der Konfiguration des Webservers - Bekannte Schwachstellen in den eingesetzten Softwareprodukten - Mangelnder Zugriffsschutz in der Datenbank
0	Netzwerk & Host	Absicherung von Host und Netzwerk

Abbildung B1.1.2: Ebenen der Web Application Security

Ebene 0 – Netzwerk und Host (nicht Gegenstand der Untersuchung)

Die Websicherheit steht auch in Abhängigkeit zur Sicherheit von Netzwerk, Hardware und Host. Wir betrachten die Berührungspunkte der beiden Ebenen, soweit dies erforderlich ist.

Ebene 1 – Systemebene (nicht Gegenstand der Untersuchung)

Ebene 1 beinhaltet all jene Software, die eine Webanwendung benötigt, um überhaupt laufen zu können. Dazu gehören der Webserver und der Applikationsserver, aber auch die Datenbank und beteiligte Backend-Systeme. Alle diese Komponenten müssen bei der Betrachtung der Sicherheit einer Webanwendung mit einbezogen werden. Eine Webanwendung A, die frei von Sicherheitsmängeln programmiert ist, ist trotzdem unsicher, wenn die von ihr verwendete Datenbank über einen anderen Kanal, etwa den nicht ausreichend abgesicherten und für ‚Innentäter‘ zugänglichen Direktzugriff per Database-Client manipulierbar ist und diese Manipulation im Web von einem Angreifer ausgenutzt werden kann.

Ein besonderes Feld sind die sog. Known-Vulnerabilities, die wir ebenfalls der Systemebene zuordnen.

Ebene 2 – Technik

In diesem Bereich geht es um die Auswahl der für den jeweiligen Zweck und Schutzbedarf richtigen Technik – und um deren richtigen Einsatz. Eine Webanwendung, die sensible Daten unverschlüsselt über das Internet transferiert, setzt nicht die richtige Technik ein. Und eine Webanwendung, die Passwörter zwar verschlüsselt, dies aber mit einem zu kurzen Schlüssel tut, setzt die richtige Technik falsch ein. Die erste Anwendung ist gegen Ausspähen auf dem Übertragungswege nicht geschützt, die andere nicht ausreichend gegen Passwort-Cracking. Beide sind also unsicher, auch wenn sie im Programmcode keine Sicherheitslücken enthalten.

Ebene 3 – Implementierung

Die Implementierungsebene ist die offensichtliche Ebene der Web Application Security. Dies ist der Bereich, in dem unbeabsichtigte Programmierfehler (Bugs) auftreten und zu Sicherheitsproblemen führen oder aber fehlerhafte Programmierung, wie nicht vorhandene oder ungenügende Datenvalidierung, stattfindet.

Logik (Ebene 4) und Semantik (Ebene 5)

Diese beiden Ebenen sind diejenigen, die gegenwärtig im Sicherheitskontext noch kaum beachtet werden. Dabei sind sie von großer Wichtigkeit – und werden es im Zeitalter von Phishing und Identitätsdiebstahl immer stärker werden – wenn man die Sicherheit von Webanwendungen nicht allein als den Schutz des Servers vor Eindringversuchen versteht, sondern sie umfassend begreift. Eine Webanwendung ist sicher, wenn sie selbst mitsamt dem sie beherbergenden System sicher ist, und wenn sie die Benutzer und deren vertrauenswürdige Daten vor Schaden bewahrt. Der Anbieter einer Webanwendung trägt also nicht nur die Verantwortung für sein eigenes System, sondern auch für alle an der Nutzung Beteiligten. Auf den Ebenen der Logik und der Semantik kommt dieser letzte Aspekt ganz besonders zum Tragen.

Ebene 4 – Logik

Diese Ebene betrifft die Logik der Abläufe in einer Anwendung – die Anwendungs- und Business-Logik – mit dem Benutzer. Ist diese zu ‚zweckorientiert‘ implementiert, d. h. ist die Möglichkeit, dass sie anders als beabsichtigt genutzt wird, zu wenig berücksichtigt, dann ist häufig eine Angreifbarkeit gegeben. Sind z. B. Mechanismen eingebaut, welche bei unsachgemäßer Bedienung (Zustände, die nur unter Umgehung der Browserfunktionalität möglich sind und damit eine missbräuchliche Verwendung eindeutig anzeigen) den Benutzer ausloggen oder andere Formen des Schutzes betreiben?

Ebene 5 – Semantik

Die semantische Ebene der Web Application Security umfasst inhaltliche und die Kommunikation betreffende Aspekte. Dies betrifft die Art der Informationen, die dem Benutzer gegeben werden, wie ihm Inhalte präsentiert werden und wie mit ihm umgegangen wird. Dieser Bereich kann in seiner Betrachtung selten auf eine einzelne Anwendung beschränkt bleiben. Er ist in der Regel vielmehr website- oder unternehmensübergreifend zu definieren und ähnlich, wie die Vorgabe einer CI von allen Kommunikationsmedien einzuhalten ist, von allen Anwendungen zu befolgen.

Ein fehlerhafter Umgang mit den semantischen Aspekten der Web Application Security erleichtert insbesondere Angriffe auf den Benutzer, was wiederum dem Vertrauen des Benutzers in die Anwendung und das Unternehmen sowie das Web insgesamt schadet. Zu derartigen Angriffen zählen Social-Engineering, Phishing, Identitätsdiebstahl, Täuschung, Fälschung, Betrug, sowie Aufbrechen des Datenschutzes und des Schutzes der Privatsphäre.

B1.2 Bewertungsschema

B1.2.1 Gefahrenpotenzial

Die durchgeführten Tests wurden jeweils mit einem Gefahrenpotenzial bewertet. Das Gefahrenpotenzial ist ein abstraktes Maß für die Klassifizierung einer Schwachstelle (Vulnerability) und der durch sie entstehenden Bedrohung (Threat).

Das Gefahrenpotenzial wird zu jeder getesteten Schwachstelle in der rechten Spalte genannt. Wir unterscheiden:

[OK] die genannte Schwachstelle liegt nicht vor

[kritisch] (siehe unten Gefährdungs-Matrix)

[hoch] (siehe unten Gefährdungs-Matrix)

[mittel] (siehe unten Gefährdungs-Matrix)

[niedrig] (siehe unten Gefährdungs-Matrix)

[info] hierbei handelt es sich um Informationen, nicht um eine Schwachstelle

Das Gefahrenpotenzial ist kein Maß für das Risiko, das eine solche Schwachstelle darstellt. Das Risiko (= Schadenshöhe * Eintrittswahrscheinlichkeit), welches dieses Gefahrenpotenzial darstellt, wird hier nicht bewertet, da zu dessen Beurteilung weitere Informationen (z. B. Schadenspotenzial) vom Auftraggeber nötig sind.

B1.2.2 Eintrittswahrscheinlichkeit

Die Eintrittswahrscheinlichkeit ist im Wesentlichen von folgenden Faktoren bestimmt:

- Wie einfach ist die Schwachstelle zu finden (Visibility)?
- Wie einfach ist die Schwachstelle auszunutzen (Exploitability)?
- Wie bekannt ist die Schwachstelle (Publicly Known)?
- Welche technischen Kenntnisse sind zum Ausnutzen nötig?
- Welche Zugriffsmöglichkeiten bestehen (remote vs. local)?
- Welchen Typs ist die Schwachstelle (Vulnerability Type)?

Vereinfacht kann man sagen, dass die Eintrittswahrscheinlichkeit von der Schwierigkeit des Angriffs abhängt.

Die Einschätzung der Eintrittswahrscheinlichkeit sollte nur mit großer Vorsicht dazu herangezogen werden, um über die Notwendigkeit der Beseitigung einer Schwachstelle zu entscheiden. Ein entsprechend motivierter Angreifer mit hinreichendem Kenntnisstand ist einem Pentester in der Regel überlegen, wenn er nur bereit ist, genügend Zeit zu investieren. Die Situation verschlechtert sich weiter, wenn man annehmen muss, dass eine Vielzahl von Angreifern nach Schwachstellen sucht.

B1.2.3 Schadenspotenzial

Das Schadenspotenzial ist vor allem die Folge, die sich aus der Ausnutzung der Schwachstelle ergeben kann:

- Verlust der Verfügbarkeit (Availability)
- Verlust der Vertraulichkeit (Confidentiality)
- Verlust der Vollständigkeit, bzw. Unversehrtheit (Integrity)
- Verlust von Schutzmechanismen
- Erweiterung der Zugriffsberechtigungen.

Das konkrete Schadenspotenzial kann i. A. nur vom Auftraggeber bestimmt werden.

B1.2.4 Risiko-/Gefährdungs-Matrix

Es ist allgemein üblich, das Risiko nach der Formel

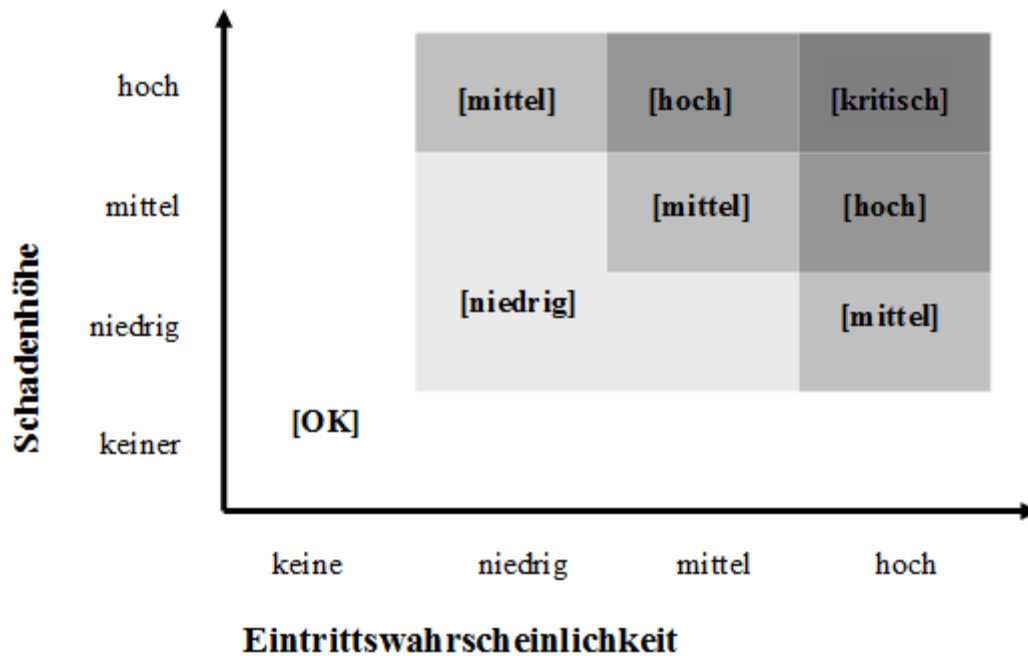
Risiko = Schadenspotenzial * Eintrittswahrscheinlichkeit

zu bestimmen. Aus den Beschreibungen zur Eintrittswahrscheinlichkeit und zum Schadenspotenzial wird ersichtlich, dass sich beide Faktoren ebenfalls aus mehreren Werten zusammensetzen. Z. B. hat eine Schwachstelle, die allgemein bekannt ist und für die bereits fertige Exploits existieren, eine hohe Eintrittswahrscheinlichkeit. Ist dadurch dann z. B. die Übernahme des Systems möglich oder können vertrauliche Daten eingesehen werden, dann ist auch das Schadenspotenzial hoch.

Um das Produkt aus Eintrittswahrscheinlichkeit und Schadenspotenzial mit ihren jeweils eigenen Faktoren zu berechnen, gibt es verschiedene Modelle (CVSS, DREAD, STRIDE, OCTAVE), die die einzelnen Faktoren unterschiedlich gewichten.

Um eine genaue Risikoabschätzung zu erhalten, müsste das Risiko mit der subjektiven Gewichtung des Auftraggebers nach den bekannten Modellen berechnet werden. Diese muss für jede in diesem Bericht aufgeführte Schwachstelle einzeln erfolgen.

Um die Schwachstellen einfach und schnell zu klassifizieren, wird für das Gefahrenpotenzial folgende vereinfachte Matrix verwendet:



Aus dieser Beurteilung kann auch eine Handlungspriorität abgeleitet werden:

[kritisch] – Es besteht sofortiger Handlungsbedarf; der Applikation, der Datenbank oder dem System kann schwerwiegender Schaden zugefügt werden; ggf. ist das System abzuschalten

[hoch] – Es besteht dringender Handlungsbedarf, die Webanwendung/das System und/oder die Daten sind bestandsgefährdet;

[mittel] – Es besteht Handlungsbedarf, die Webanwendung/das System muss z. B. durch einspielen von Sicherheitsupdates oder durch zusätzliche Sicherungsmaßnahmen (z. B. WAF) abgesichert werden.

[niedrig] – Der Handlungsbedarf liegt im Ermessen des Auftraggebers.

Hinweis

Obwohl wir bei der Bewertung objektive Maßstäbe bzw. als allgemeingültig anerkannte Grundsätze zugrunde gelegt haben und nach bestem Wissen und Gewissen versucht haben, die Vor- und Nachteile gegeneinander abzuwägen, kann eine solche Bewertung immer nur subjektiv sein. Uns nicht bekannte Details oder unterschiedliche Interpretation der Sicherheitsanforderungen können zu einer anderen als der hier jeweils abgegebenen Bewertung führen. Daher ist der Auftraggeber angehalten, anhand der gegebenen Informationen eine eigene Bewertung vorzunehmen, bzw. die Dringlichkeit der Behebung einzelner Mängel selbst zu beurteilen.

Gerne stehen wir dabei beratend zur Seite.

B1.3 Aufbau der Testergebnisse

Für jede Schwachstelle gibt es jeweils ein eigenes Unterkapitel, wobei deren Überschriften Auskunft über die Art der Schwachstelle geben.

Eine solche Überschrift sieht dann z. B. so aus:

- **n.m Cross-Site-Scripting**
- **n.m** ist die Unterkapitelnummer
- **Cross-Site-Scripting** ist die genaue Schwachstellenbezeichnung

Jedes dieser Unterkapitel ist dann gegliedert in:

- **Bedrohung**
Die Schwachstelle bzw. Angriffstechnik wird allgemeinverständlich erklärt. Ggf. mit Links zu weiterführenden Informationsquellen.
- **Beobachtung**
Ist die Anwendung gegen die beschriebene Bedrohung gefeit, wird dies hier begründet und belegt.

Ist die Anwendung gegenüber der beschriebenen Bedrohung anfällig, wird die Beobachtung in einem eigenen Unterkapitel wie folgt ausgeführt und belegt.

n.m.o Reflected-Cross-Site-Scripting stark verbreitet **[hoch]**

- **Reflected-Cross-Site-Scripting stark verbreitet**
ist eine knappe Zusammenfassung der gefundenen Schwachstelle
- **hoch**
ist die Bewertung des Gefahrenpotenzials

Die Gliederung sieht hier folgendermaßen aus:

- **Beobachtung**
Der Test und die Beobachtung der Reaktion der Anwendung werden beschrieben. Damit soll (1) der Leser in die Lage versetzt werden, den Angriff zu verstehen, um das Risiko selbst besser beurteilen zu können, und (2) der Angriff nachvollziehbar werden. Es ist nicht immer möglich, (2) zu gewährleisten, da für das Nachstellen u. U. eine umfangreichere Vorbereitung oder zeitliche Korrelation mit anderen Bedingungen erfüllt sein muss.
- **Ausnutzbarkeit**
Hier wird beschrieben, welche Voraussetzungen für einen Angriff gegeben sein müssen, in welcher Form ein Angreifer die Schwachstelle ausnutzen kann und mit welchen Konsequenzen und Risiken zu rechnen ist. Generell legen wir einen Worst-Case Ansatz zugrunde, d. h. schildern im Zweifel das bedrohlichere Szenario. Diese Schilderung ist unabhängig von der Eintrittswahrscheinlichkeit. Eine Bewertung der Eintrittswahrscheinlichkeit muss separat erfolgen.
- **Hinweis / Bemerkung (optional)**
Hier werden Besonderheiten und/oder bestimmte Bedingungen zur gefundenen Schwachstelle beschrieben, z. B. wenn diese nicht reproduzierbar auftrat oder nur mit bestimmten Browsern. Oft sind solche Anmerkungen zur Beurteilung der Bewertung (siehe oben) wichtig.

— **Maßnahme**

Es ist zwischen grundlegenden Maßnahmen und individuellen Maßnahmen zu unterscheiden. Grundlegende Maßnahmen sind meistens bedrohungsbezogen, wohingegen individuelle Maßnahmen eher schwachstellenbezogen sind. Grundlegende Maßnahmen lassen sich unabhängig von der jeweiligen Anwendung formulieren und haben universelle Gültigkeit.

Bei Webanwendungen, die ohne Berücksichtigung der grundlegenden Schutzmaßnahmen entwickelt worden sind, kann die nachträgliche Anwendung grundlegender Maßnahmen zu unvermeidbar hohem Aufwand führen. Hinzu kommt, dass eine Anwendung, die die beschriebene Maßnahme nicht umgesetzt hat, trotzdem nicht zwingend anfällig für einen Angriff sein muss, weil dies bewusst durch andere Maßnahmen sichergestellt worden ist, oder weil es sich eher zufällig so verhält. In diesem Fall wäre es nicht angebracht, die Umsetzung der genannten Maßnahme zu fordern.

Maßnahmen aus Sicht des Penetrationstests sind daher zumeist individuelle schwachstellenbezogene Maßnahmen, die erst bei Entdeckung einer Schwachstelle relevant werden. Dabei kommt zur Behebung der Schwachstelle häufig nicht genau eine Maßnahme in Betracht, sondern eine ganze Palette von möglichen Maßnahmen. Die Entscheidung über die richtige Maßnahme ist für jeden Anwendungsfall individuell zu treffen und abhängig von den softwaretechnischen Randbedingungen, der verwendeten Technik, der Art der Realisierung, den Realisierungskosten usw.

- Eine tatsächliche Bewertung und Entscheidung darüber, wie eine Schwachstelle zu beseitigen ist, muss der Auftraggeber treffen oder sie wird in einem abschließenden Workshop zusammen mit den Entwicklern bzw. Anwendungsverantwortlichen getroffen.

B1.3.1 Aufbau der Testergebnisse der toolbasierten Findings

Alle von den Werkzeugen gemeldeten Schwachstellen wurden bewertet und Ergebnisberichte (falls möglich bzw. notwendig) liegen im Anhang (s. jeweiliges Werkzeug in Abschnitt A4.5.1 bzw. grobe Übersicht in Abschnitt C9) bei.

Die tool-eigenen Kritikalitätseinstufungen wurden von uns folgendermaßen interpretiert:

- **Kritisch** (in den Tools: „Critical“)
- **Hoch** (in den Tools: „High“ oder „Error“)
- **Mittel** (in den Tools: „Medium“, „Moderate“ oder „Warning“)
- **Niedrig** (in den Tools: „Low“, „Note“ oder „Recommendation“)

Die eigentlichen Bewertungen erfolgten entweder in der vom Werkzeug bereitgestellten Oberfläche oder in von uns aus den RAW-Ergebnissen erstellten Excel-Dokumenten. Dabei musste bei der Bewertung teilweise auf tool-eigenes „Wording“ zurückgegriffen werden. Die Bewertungen in den Reports sind folgendermaßen zu interpretieren:

- **Bestätigt** (in den Bewertungen: „Exploitable“ oder „Confirmed“)

- Finding wurde bestätigt
- **Verdächtig** (in den Bewertungen: „Suspicious“)
 - Finding konnte nicht 100%ig bestätigt oder entkräftet werden (weder statisch noch durch DAST-Analysen), könnte aber ein (zukünftiges) Sicherheitsproblem darstellen, wessen sich das Projekt bewusst sein sollte. Hier muss die Bewertung und Behandlung von einem Projekt-Insider mit Sicherheitshintergrund entschieden bzw. abgeschlossen werden.
- **Schlechte Praxis** (in den Bewertungen: „Bad Practice“)
 - Finding wurde als Programmierfehler oder schlechte Code-Qualität eingestuft, welcher z.B. sicherheitsrelevante Konsequenzen oder logische Fehlinterpretationen zur Folge haben könnte. Die Wahrscheinlichkeit dafür wurde aber als sehr gering eingestuft.
- **Kein Problem** (in den Bewertungen: „Not an issue“)
 - Finding wurde als irrelevant oder False Positive eingestuft.

Auf Basis dieser beiden Einstufungen ergibt sich die folgende Interpretation zur in B1.2.4 definierten Risiko-/Gefährdungsmatrix (die Statistik-Übersicht in A3.1 basiert auf dieser Zuordnung):

[kritisch]

- Tooleinstufung: kritisch X Auditoreinstufung bestätigt

[hoch]

- Tooleinstufung: hoch X Auditoreinstufung bestätigt

[mittel]

- Tooleinstufung: kritisch X Auditoreinstufung verdächtig
- Tooleinstufung: mittel X Auditoreinstufung bestätigt

[niedrig]

- Tooleinstufung: kritisch X Auditoreinstufung schlechte Praxis
- Tooleinstufung: hoch X Auditoreinstufung verdächtig
- Tooleinstufung: niedrig X Auditoreinstufung bestätigt

[info]

- Tooleinstufung: hoch X Auditoreinstufung schlechte Praxis
- Tooleinstufung: mittel X Auditoreinstufung verdächtig
- Tooleinstufung: mittel X Auditoreinstufung schlechte Praxis
- Tooleinstufung: niedrig X Auditoreinstufung verdächtig
- Tooleinstufung: niedrig X Auditoreinstufung schlechte Praxis

B1.4 Urheberrecht

Die in diesem Dokument enthaltenen allgemeinen Beschreibungen von Schwachstellen, Angriffsszenarien, Maßnahmen und Bewertungsverfahren sowie die Berichtsstruktur sind das geistige Eigentum der mgm security partners GmbH. Das Dokument darf im Rahmen des vorgesehenen Zweckes vom Auftraggeber sowie direkt beteiligten

Personen verwendet werden. Eine Weitergabe als Ganzes oder in Teilen an sonstige Dritte ist untersagt. Alleiniger Inhaber der Urheber- und Verwertungsrechte dieser Texte ist die mgm security partners GmbH.

C. Testergebnisse im Detail

C1 Datenvalidierung

Die meisten Schwachstellen in Webanwendungen haben eine ungenügende Datenvalidierung (oft als Input-Datenvalidierung bezeichnet) als Ursache.

Unter Datenvalidierung sind alle Daten und zugehörigen Prüfungen zu verstehen, die die Daten bearbeiten, welche von außen in ein System kommen. Dazu gehören nicht nur die unmittelbaren Benutzereingaben, sondern auch die Daten, die von Drittsystemen (z. B. aus Datenbanken) kommen. Weiter muss jedes System sicherstellen, dass die Daten, die an Drittsysteme zur Verarbeitung weitergegeben werden, keinen schadhaften Code enthalten.

Korreakterweise muss daher zwischen Input-Validierung- und Output-Enkodierung unterschieden werden. Bei den folgenden einzelnen Schwachstellen und Bedrohungen wird ggf. auf diesen Unterschied aufmerksam gemacht.

C2 Authentisierung

C3 Session-Management

Session-Management beschreibt allgemein die technischen und logischen Voraussetzungen, die notwendig sind, um Sitzungen, Aktionen und Transaktionen in einer Umgebung mit zustandslosen Verbindungen zu betreiben.

C3.2 Gültigkeit des Session-ID-Cookies

Bedrohung

Cookies besitzen einen bestimmten Gültigkeitsbereich. Dieser wird beim Setzen des Cookies definiert und dem Browser in Form von Cookie-Parametern (`domain`, `path`, `expire`, `httponly`, `secure`) mitgeteilt. Der Browser entscheidet dann anhand dieser Parameter, ob das entsprechende Cookie an die Anwendung geschickt wird oder nicht. Die Anwendung hat keine Möglichkeit zu erkennen, unter welchen Bedingungen ein Cookie gesendet wurde.

C3.2.1 Schutz des Sitzungscookie mit Verbesserungspotenzial

[niedrig]

Beobachtung

Der folgende Screenshot zeigt die Cookie-Attribute des Sympa-Sitzungscookies:



Das Cookie wird ohne das `sameSite`-Attribut und mit `path=/` als Domain-Cookie gesetzt.

Ausnutzbarkeit

Die Anwendung schützt sensible Cookies nicht ausreichend. Wenn Sitzungs-Cookies oder andere sicherheitsrelevante Cookies durch einen Angriff ausgelesen werden können, ermöglicht dies in vielen Fällen die Übernahme von Benutzersitzungen und Benutzerkonten.

SameSite

Das `SameSite`-Attribut kann als Ergänzung zum CSRF-Token eingesetzt werden, diesen jedoch nicht komplett ersetzen. Das liegt daran, dass das Cookie-Flag vor Angriffen von Drittseiten schützt, jedoch keine Angriffe verhindern kann, die von der Originalseite durch anderweitige Schwachstellen, wie XSS, ausgelöst werden. Das Attribut verhindert das Senden von Cookies über Anfragen, die von Drittseiten ausgelöst wurden. Folgende Modi werden unterstützt:

- "none": Das Attribut ist nicht gesetzt.
- "lax": Verhindert Angriffe über unsichere HTTP-Methoden wie POST.

- "strict": Verhindert Angriffe für alle HTTP-Methoden. Für diesen Modus muss jedoch meistens das Session-Management angepasst werden.

Das Attribut wird seit 2018 von allen gängigen Browsern unterstützt, im Internet Explorer jedoch nur unter Windows 10 mit installiertem Sicherheitsupdate von Juni 2018. Seit August 2020 wird in den meisten gängigen Browsern das SameSite-Attribut automatisch auf den Wert „lax“ gesetzt.

Path

Ist das Path Attribut des Sitzungscookies nicht auf den Anwendungspfad eingeschränkt, wird der Browser das Cookie für alle Pfade verfügbar machen. Im Speziellen kann dies dazu führen, dass eine Anwendung unter dem selben Host, aber einem anderen Pfad, z.B. /devTestApp, ebenfalls Zugriff auf das Sitzungscookie erhält, welches nur für /sympa zugreifbar sein sollte.

Etwaige Sicherheitslücken in der /devTestApp, z.B. Cross-Site-Scripting (XSS), stellen dann ebenfalls eine Bedrohung für das Sitzungscookie dar. Darüber hinaus können jedwede legitime Funktionen der /devTestApp auf das Sitzungscookie zugreifen.

Maßnahme

Für die Übertragung von Cookies sollten die fehlenden Attribute aktiviert bzw. das Path-Attribut genauer eingeschränkt werden.

SameSite

- <https://owasp.org/www-community/SameSite>
- <https://tools.ietf.org/html/draft-west-first-party-cookies-07>
- <https://hacks.mozilla.org/2020/08/changes-to-samesite-cookie-behavior/>
- <https://www.chromestatus.com/feature/5088147346030592>

Path

Das Path-Flag sollte auf /sympa/ gesetzt werden. Hierbei ist der abschließende ' / ' zu beachten. Dadurch wird verhindert, dass das Session-Cookie an z. B. eine mögliche Anwendung unter /sympa/ vom Browser mitgesandt wird.

C3.5 Logout durch Timeout bei Inaktivität

Bedrohung

Wenn Anwendungen die Session bei Inaktivität nicht beenden, erhöht sich potenziell die Anfälligkeit für Angriffe auf die Session (Hijacking, Replay, Riding) und unbeabsichtigte Ausspähung der Daten am Client durch Dritte, da das Zeitfenster für einen Angriff nicht minimal gehalten wird.

C3.5.1 Kein Logout bei Inaktivität

[mittel]

Beobachtung

Die Anwendung hat keinen automatischen Logout nach einer hinreichend kurzen Zeit der Inaktivität implementiert. So war es möglich, nach über 24 Stunden Inaktivität die authentifizierte Session weiterhin zu verwenden.

Der folgende Screenshot zeigt den Aufruf der Home-Landingpage mit dem Sitzungscookie `sympa_session=03755156599741` am 02. Juni 2025 um 6:36 Uhr:

#	Host ^	Method	URL	Param	Edited	Status code
239	https://pentest-extern....	GET	/sympa			200
235	https://pentest-extern....	GET	/sympa/			200

Request

Pretty Raw Hex

```
1 GET /sympa/ HTTP/1.1
2 Host: pentest-extern.muenchen.de
3 Cookie: sympa_session=03755156599741; BIGSC=!jxPbq6gl/XhOevJV6Y09XTLauDl7H5f3VPL
```


Response

Pretty Raw Hex Render

```
1 HTTP/1.1 200 OK
2 Date: Mon, 02 Jun 2025 06:36:55 GMT
3 Cache-Control: no-store, max-age=0
4 Set-Cookie: sympa_session=03755156599741; path=/; secure; HttpOnly
5 Strict-Transport-Security: max-age=16070400; includeSubDomains
6 Upgrade: h2
7 Connection: Upgrade, Keep-Alive
8 Keep-Alive: timeout=5, max=100
9 Content-Type: text/html; charset=utf-8
10 Set-Cookie: TS0183cc68=
    01021d36f2a66c4079827dde095d95204d2d63ee4a5c04328ad7aea655feb85dc84762d26baa46a
    12575e108a6a2b504b824d20bad5c1f1971aa6b82903db934cada2b390e62110446572a0dba2d70
    Path=/; Domain=.pentest-extern.muenchen.de; Secure; HttpOnly;
11 Set-Cookie: TS383f4674027=
    087179dd52ab20008eb0e9a8538fe0db4b06908458a9b3ba56ed8dc32e4a4dff46ebe133843ad2
    34a9ba76b54b7e6b3f23f53530; Secure; Path=/
12 Content-Length: 10625
13
14 <!DOCTYPE html>
15 <html class="no-js" lang="en-US" xml:lang="en-US">
16   <!-- main.tt2 -->
17   <head>
18     <meta charset="UTF-8" />
19     <meta name="generator" content="Sympa 6.2.76" />
20     <meta name="viewport" content="width=device-width, initial-scale=1.0">
21     <title>
22       Mailing lists service (DEV) - home |
    </title>
```

Im Nachfolgenden Screenshot ist derselbe Request mit demselben Cookie zu sehen, welcher ebenfalls akzeptiert wird und zur Home-Landingpage führt. Dieser wurde jedoch zwei Tage später am 04. Juni 2025 durchgeführt:

#	Host ^	Method	URL	Param	Edited	Status code
239	https://pentest-extern....	GET	/sympa			200
235	https://pentest-extern....	GET	/sympa/			200

Request
Pretty Raw Hex
1 GET /sympa HTTP/1.1
2 Host: pentest-extern.muenchen.de
3 Cookie: sympa_session=03755156599741; BIGSC=!jxPbq6gl/XhOevJV6Y09XTLauDL7H5f3VP
? ⚙ ⬅ ➡ Search

Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Date: Wed, 04 Jun 2025 06:37:13 GMT
3 Cache-Control: no-store, max-age=0
4 Set-Cookie: sympa_session=03755156599741; path=/; secure; HttpOnly
5 Strict-Transport-Security: max-age=16070400; includeSubDomains
6 Upgrade: h2
7 Connection: Upgrade, Keep-Alive
8 Keep-Alive: timeout=5, max=100
9 Content-Type: text/html; charset=utf-8
10 Set-Cookie: TS0183cc68=01021d36f25602d820d9c70c546975ca0b125cef4ac7ae7b1c45bccbcecf8aa3a0c8b502a36ea0d8d85fe5b2ca33b5a534908d9c1e7e72555deb6d66ba27929c0bbbc85f4ca73ec43f8503d72f6dPath=/; Domain=.pentest-extern.muenchen.de; Secure; HttpOnly;
11 Set-Cookie: TS883f4674027=087179dd52ab200063a2951bd364d76f19a8287f48e3650761f9af578746b7ea9989c4868a407a0e5d14d6fc5a90361c63bb2eb4; Secure; Path=/
12 Content-Length: 10625
13
14 <!DOCTYPE html>
15 <html class="no-js" lang="en-US" xml:lang="en-US">
16 <!-- main.tt2 -->
17 <head>
18 <meta charset="UTF-8" />
19 <meta name="generator" content="Sympa 6.2.76" />
20 <meta name="viewport" content="width=device-width, initial-scale=1.0">
21 <title>Mailing lists service (DEV) - home</title>
22

Ausnutzbarkeit

Wenn ein Angreifer in den Besitz eines authentisierten Session-Cookies kommt, kann er dies über einen längeren Zeitraum nutzen und so Aktionen unter dem Namen eines anderen Benutzers durchführen. Solange der impersonierte Nutzer sich nicht ausgeloggt, erlischt die Session nicht.

Anmerkung: Da die Sicherheitsattribute des Cookies dieses nicht uneingeschränkt schützen, wird dieses Finding mit mittlerer Kritikalität bewertet.

Maßnahme

Die Sitzung ist nach einer hinreichend kurzen Zeit der Inaktivität (typische Werte liegen zwischen 30 und 60 Minuten) zu invalidieren.

Die Anwendung sollte ein Timeout für inaktive Sessions setzen. Bei Ablauf des Timeouts – also Zeit, in der die Session nicht benutzt wurde – ist die Session serverseitig vollständig zu löschen und der Benutzer beim nächsten Zugriff zum erneuten Login aufzufordern.

C3.6 Session-Fixation

Bedrohung

Session-Fixation ist eine Angriffstechnik, die dem Benutzer einer Anwendung eine bestehende und vom Angreifer initiierte Session vorgibt. Der Benutzer authentisiert dann diese vorgegebene Session und verschafft dem Angreifer damit Zugang zu seinem Benutzerkonto.

C3.6.1 Session Cookie wird nach dem Login nicht aktualisiert

[mittel]

Beobachtung

Wird beim Login ein Cookie mit dem Namen `sympa_session` versendet, welches in der Vergangenheit bereits einem Nutzer zugewiesen war, verzichtet die Sympaanwendung auf das Invalidieren und Neusetzen des Cookies. Dies geschieht auch dann, wenn der Nutzer zwischenzeitlich ausgelogged war. Der folgende Screenshot zeigt einen Login-Request, welchem ein in einer früheren Sitzung bereits verwendetes `sympa_session` Cookie vom Browser mitgesendet wird:

The screenshot displays the Network tab of a web browser's developer tools. A list of requests is shown, with the last one (index 234) highlighted in blue and a red box around it. This request is a GET to `/sympa/sso_login/oidc/init` with a status code of 302. Below the list, the 'Request' section is expanded, showing the raw HTTP request. A red box highlights the 'Cookie' header, which contains `sympa_session=03755156599741; BIG5C=1jxPbq6gl/Xh0evJV6Y09XTLAuDl7H5f3VPlfzt2Yqh01021d36f2a66c4079827dde095d95204d2d63ee4a5c04328ad7aea655feb85dc84762d26baa46a20e51e3236cf7c6ea04583c67dd13aae555578287b53deacabc9f624358e1f162890b1a; TS383f4674027=087179dd52ab2000804c753ed91f6dfbd84dee7cd56ff005cdfbc993f87d5732121778a66feb47cb08b8fb3`. A red arrow points from this cookie to the 'Response' section below. The 'Response' section shows a 302 'Moved' status. A red box highlights the 'Set-Cookie' header, which contains `sympa_session=03755156599741; path=/; secure; HttpOnly`, indicating that the same session ID is being reused.

#	Host	Method	URL	Params	Edited	Status code
242	https://pentest-extern.m...	GET	/sympa/edit_list_...			200
241	https://pentest-extern.m...	GET	/sympa/info/pent...			200
240	https://pentest-extern.m...	GET	/sympa/my			200
239	https://pentest-extern.m...	GET	/sympa			200
235	https://pentest-extern.m...	GET	/sympa/			200
234	https://pentest-extern.m...	GET	/sympa/sso_logi...			302

Request

Pretty Raw Hex

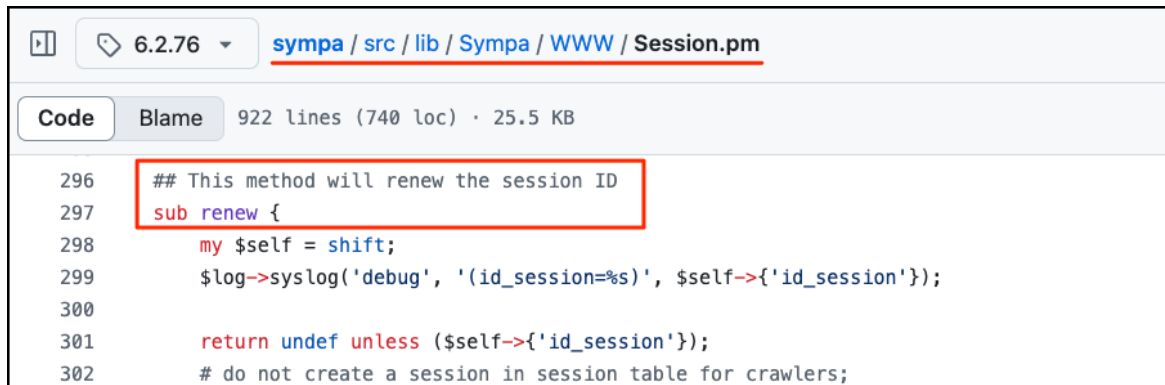
```
1 GET /sympa/sso_login/oidc/init HTTP/1.1
2 Host: pentest-extern.muenchen.de
3 Cookie: sympa_session=03755156599741; BIG5C=1jxPbq6gl/Xh0evJV6Y09XTLAuDl7H5f3VPlfzt2Yqh01021d36f2a66c4079827dde095d95204d2d63ee4a5c04328ad7aea655feb85dc84762d26baa46a20e51e3236cf7c6ea04583c67dd13aae555578287b53deacabc9f624358e1f162890b1a; TS383f4674027=087179dd52ab2000804c753ed91f6dfbd84dee7cd56ff005cdfbc993f87d5732121778a66feb47cb08b8fb3
4 User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 17_7_2 like Mac OS X) AppleWebKit/605.1.
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
```

Response

Pretty Raw Hex Render


```
1 HTTP/1.1 302 Moved
2 Date: Mon, 02 Jun 2025 06:36:52 GMT
3 Set-Cookie: sympa_session=03755156599741; path=/; secure; HttpOnly
4 Strict-Transport-Security: max-age=16070400; includeSubDomains
5 Location: https://pentest-extern.muenchen.de/sympa/
6 Content-Length: 0
7 Keep-Alive: timeout=5, max=99
8 Connection: Keep-Alive
9 Set-Cookie: TS0183cc68=01021d36f2a66c4079827dde095d95204d2d63ee4a5c04328ad7aea655feb85dc84762d26baa46a20e51e3236cf7c6ea04583c67dd13aae555578287b53deacabc9f624358e1f162890b1a; Path=/; Domain=.pentest
10 Set-Cookie: TS383f4674027=087179dd52ab2000ca71eec028673a2c813a9a8744dd18588d3f96279ec6e
```

Trotz der obigen Beobachtung, konnte im Code ein Mechanismus aufgefunden werden, welcher das Sitzungscookie verwaltet. Die Erneuerung des Sitzungscookies ist im Modul Session.pm vorgesehen:



```
6.2.76 ▾ sympa / src / lib / Sympa / WWW / Session.pm
Code Blame 922 lines (740 loc) · 25.5 KB
296  ## This method will renew the session ID
297  sub renew {
298      my $self = shift;
299      $log->syslog('debug', '(id_session=%s)', $self->{'id_session'});
300
301      return undef unless ($self->{'id_session'});
302      # do not create a session in session table for crawlers;
```

Jedoch wird das Cookie nur unter der Bedingung erneuert, dass die Verbindung des Clients nicht per SSL/TLS (HTTPS) erfolgt:



```
6.2.76 ▾ sympa / src / cgi / wwwsympa.fcgi.in
Code Blame 16781 lines (14790 loc) · 553 KB
1590  unless ($param->{'bypass'} eq 'extreme'
1591      or $maintenance_mode
1592      or $rss
1593      or $sajax) {
1594      $session->renew unless $param->{'use_ssl'};
1595  }
```

Ausnutzbarkeit

Ein Angreifer, welcher dem Browser des Clients ein Sitzungscookie aufzwingt, kann u.U. die Session des geschädigten Nutzers übernehmen. Hierbei muss jedoch der Wert des Sitzungscookies bereits vorher von dem Nutzer verwendet worden sein und die Zugriffe per HTTPS erfolgen.

Um einen gültigen Cookiewert zu erhalten, könnte ein Angreifer ein im Browser befindliches `sympa_session` Cookie stehlen (s. auch C3.2.1), oder aber dieses selbst generieren (s. auch C3.8.1).

Maßnahme

Nach dem Login des Benutzers ist die bereits bestehende Session-ID zu invalidieren. Dazu ist in der Regel folgendes Verfahren sinnvoll:

- Speichern der Session-Daten
- Vernichten des alten Session-Objektes
- Erzeugen eines neuen Session-Objektes und Kopieren der Session-Daten in dieses Objekt.

Auf diese Weise wird dem Benutzer nach der Authentisierung eine neue Session-ID zugewiesen und der Angreifer kann mit dem alten Cookie keinen Schaden mehr anrichten.

Vom User selbst gewählte Session-IDs sollten generell nicht akzeptiert werden.

C3.8 Zufälligkeit der Session-ID

Bedrohung

Wenn die Session-ID die einzige Information ist, anhand derer die Anwendung erkennt, mit welchem Benutzer sie es zu tun hat, dann unterliegt sie einem erhöhten Schutzbedarf. Insbesondere ist mit der Session-ID meist die Authentifizierungsinformation verknüpft, d. h. dass jeder, der in den Besitz der Session-ID kommt, diese nutzen kann, um eine Session fortzusetzen.

Daher muss die Session-ID eine dem Schutzbedarf der Anwendung entsprechende Güte besitzen. Session-IDs dürfen z. B. nicht einfach „hochgezählt“ werden, oder nach einem leicht zu durchschauenden Schema aufgebaut sein.

C3.8.1 Entropie des Sitzungscookies verbesserbar

[info]

Beobachtung

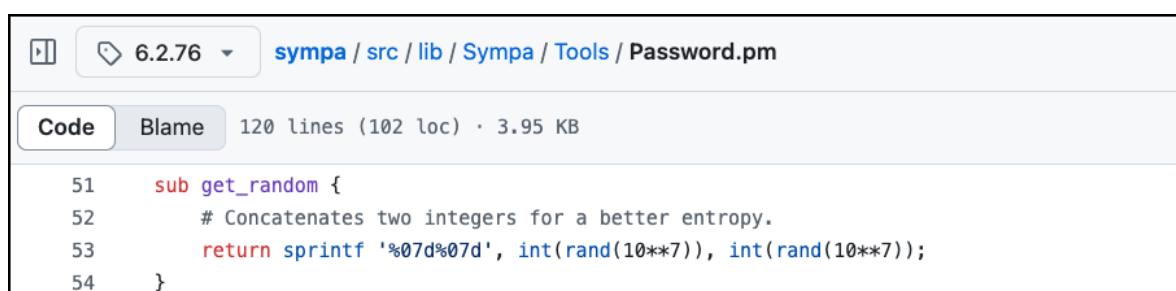
Die Entropie des Wertes des sympa_session Cookies könnte verbessert werden. Ein typischer Wert für diesen Sitzungscookie stellt sich folgendermaßen dar:

```
sympa_session=03755156599741
```

Dieser Wert ist ausschließlich numerischer Natur und besteht immer aus genau 14 Stellen. Im Code kann die Generierung dieses Wertes in den Modulen Session.pm und Passwort.pm nachvollzogen werden:



```
103      # create a new session context
104      $self->{'new_session'} =
105          1; ## Tag this session as new, ie no data in the DB exist
106      $self->{'id_session'} = Sympa::Tools::Password::get_random();
107      $self->{'email'} = 'nobody';
108      $self->{'remote_addr'} = $ENV{'REMOTE_ADDR'};
109      $self->{'date'} = $self->{'refresh_date'} = $self->{'start_date'} =
110          time;
```



```
51      sub get_random {
52          # Concatenates two integers for a better entropy.
53          return sprintf '%07d%07d', int(rand(10**7)), int(rand(10**7));
54      }
```

Die obigen Screenshots zeigen das Erzeugen eines Sitzungscookiewertes aus zwei zufällig gewählten 7-stelligen Zahlen.

Ausnutzbarkeit

Ein Angreifer könnte versuchen, gültige Session-IDs automatisiert zu enumerieren. Bei einer erfolgreichen Enumeration hätte er dann Zugriff auf die Session des Users.

Maßnahme

Ist das Sitzungscookie das einzige Authentifizierungsmerkmal nach dem Login, muss darauf geachtet werden, dass dieses ausreichend zufällig gewählt wird.

Nach der Empfehlung der OWASP, sollte die Entropie einer Session-ID mindestens 64 Bit betragen (https://owasp.org/www-community/vulnerabilities/Insufficient_Session-ID_Length)

C4 Zugriffsschutz

C4.2 Entity-Enumeration

Bedrohung

Wenn Funktionen einer Anwendung über Parameter der URL aufgerufen werden, dann ist es potenziell möglich, alle Werte dieses Parameters auszuprobieren. Man spricht dann von Entity-Enumeration.

C4.2.1 Enumeration von Mailinglisten und -ownern möglich

[niedrig]

Beobachtung

Zu jeder Mailingliste existiert eine Übersichtsseite, welche unter dem Pfad `/sympa/info/<Name Mailinglist>` erreicht werden kann. Sollte eine nichtexistierende Übersichtsseite aufgerufen worden sein, wird die Standard-Landingpage angezeigt. Existiert sie, gibt die Anwendung die konfigurierte Übersichtsseite aus.

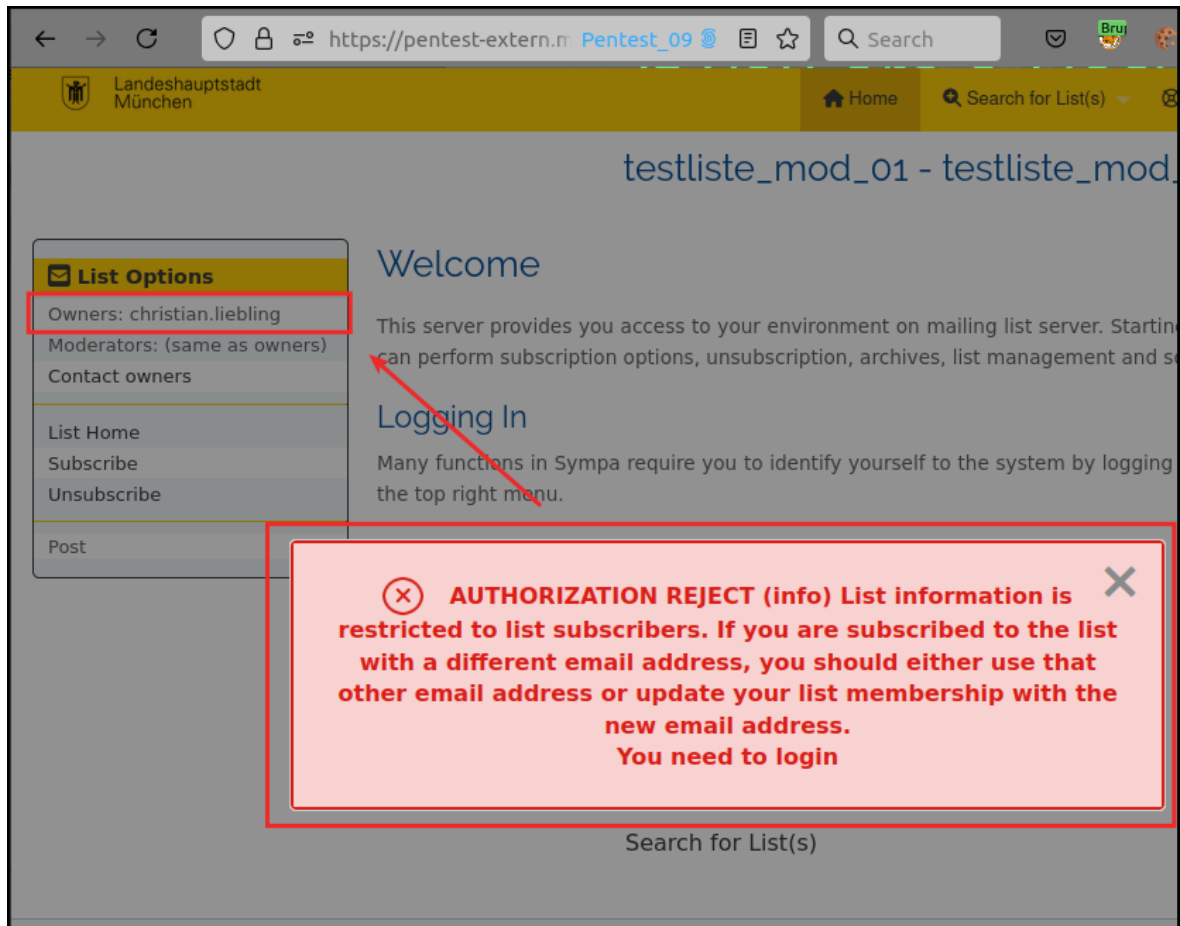
Sollte ein Nutzer keine Zugriffsberechtigung besitzen, wird ein Fehler mit der Aufforderung zur Authentifizierung angezeigt.

Da sich die Inhalte der Antworten unterscheiden, kann an der Antwortlänge erkannt werden, ob eine Mailingliste existiert.

Der folgende Screenshot zeigt mehrere Requests auf die Übersichtsseite für nicht-exstierende „nonexistent“ Mailinglisten. Darauffolgend werden drei existierende Mailinglisten (rot umrahmt) angegeben:

Request ^	Payload	Status code	Respons...	Error	Timeout	Length	Comme
0		200	184			10792	
1	nonexistant_02	200	728			10792	
2	nonexistent_03	200	172			11028	
3	nonexistent_04	200	226			11028	
4	nonexistent_05	200	255			11028	
5	pentest01	200	1370			12350	
6	pentest02	200	685			13071	
7	testliste_mod_01	200	431			13910	

Sollte eine Mailingliste aufgerufen worden sein, auf welche der aktuelle Nutzer keinen Zugriff hat, liefert die Anwendung neben der entsprechenden Fehlermeldung das Mailinglistmenü aus, aus welchem der Name des Mailinglistowners ersichtlich ist:



Ausnutzbarkeit

Ein Angreifer kann existierende Mailinglisten enumerieren und darüber hinaus an die Ownerinformation der Listen gelangen.

Diese Informationen sind sensibel und könnten zur Spezialisierung von zukünftigen Angriffen verwendet werden.

Maßnahme

Der Zugriff auf Dateien und Objekte sollte per nicht vorhersagbarer ID, z.B. Universal-Unique-Identifizier (UUID), geschehen.

Bei der Verwendung von URL-Parametern als Zugriffskontrolle auf Objekte oder Funktionen sollten ebenfalls Werte mit entsprechend großer Zufälligkeit (Entropie) genutzt werden.

C5 Konfiguration und Deployment

Unter Konfiguration und Deployment sind die Schwachstellen zusammengefasst, die ihre Ursache in den verwendeten Techniken und Methoden der Implementierung der Webanwendung haben.

C5.7 Serverkonfiguration für Content-Security-Policy (CSP)

Bedrohung

Die Content-Security-Policy (CSP) Direktive bietet unter anderem die Möglichkeit, in modernen Browsern das Cross-Site-Scripting-Risiko zu verringern. Diese Direktiven werden über HTTP-Header-Attribute gesetzt.

Die CSP nutzt einen Whitelisting-Ansatz, welcher explizit definiert, welche internen und externen Ressourcen innerhalb der Anwendung erlaubt sind, und welche der Browser ignorieren soll. Dieser Ansatz erlaubt feingranulare Policy-Direktiven, wie z. B. die zulässige Einbindung von Bildern, Schriftarten und JavaScript-Ressourcen.

C5.7.1 Keine Auslieferung des Content-Security-Policy-Headers

[niedrig]

Beobachtung

Der HTTP-Header Content-Security-Policy (CSP) wird nicht gesetzt, wie der folgende Screenshot beispielhaft zeigt:



```
1 HTTP/1.1 200 OK
2 Date: Tue, 27 May 2025 08:22:08 GMT
3 Cache-Control: no-store, max-age=0
4 Set-Cookie: sympasession=10437563926200; path=/; secure; HttpOnly
5 Strict-Transport-Security: max-age=16070400; includeSubDomains
6 Upgrade: h2
7 Connection: Upgrade, Keep-Alive
8 Keep-Alive: timeout=5, max=100
9 Content-Type: text/html; charset=utf-8
10 Set-Cookie: TS0183cc68=
01021d36f227047b327c3a6294015910d2c6c068dbe293ffdc7e19b3640e05ecccf1e0e1d52df91fc947eb8e39
2a2b5cc4555abe8a364886718e80fa5114969a3275db3598d4cdad038627099b78fa0cbe32aa72202d55e330b8
ff476187d076428c8c6352635afc1e35f93dfc6094242d77ffe924170766fafbcddf3cf91241a6f4e4fad9;
Path=/; Domain=.pentest-extern.muenchen.de; Secure; HttpOnly;
11 Set-Cookie: TS383f4674027=
087179dd52ab20003fe2b3168eeb9197eef0288ab6a3a468fd0db00bc15c76d0ee2ca6fdcb43d0a080ee29cf6
11300049546ddc85bacd18b4a8eb1eb510e6b69d9dfc1990a1f785ed31b4e8ec9bbedda0a0f6a0d0cfb269983
8cb2eaaeca5d; Secure; Path=/
12 Content-Length: 18845
```

Ausnutzbarkeit

Mithilfe einer CSP kann die Ausnutzbarkeit von clientseitigen Schwachstellen stark eingeschränkt werden, indem vertrauenswürdige Datenquellen explizit spezifiziert werden.

Fehlt eine solche CSP, wird der Browser JavaScript-Code aus beliebigen Quellen ausführen, auch sogenannten inline-JavaScript-Code, welcher meist bei der dynamischen Erzeugung von HTML-Code vorkommt.

Cross-Site-Scripting-Angriffe zielen darauf ab, genau solche dynamischen HTML-Code-Erzeugungen auszunutzen. Sollten hierbei Anwendereingaben verarbeitet werden, kann beliebiger Script-Code zur Ausführung im Browser kommen.

Neben Cross-Site-Scripting-Schwachstellen kann eine CSP vor weiteren Gefahren, wie DOM-based Cross-Site-Scripting, Cross-Site-Leaking, Clickjacking oder unverschlüsselten Verbindungen, schützen.

Maßnahme

Es sollte eine CSP mit mindestens den Direktiven `default-src`, `script-src` und `object-src` definiert werden, um einen zusätzlichen Schutzmechanismus gegen Cross-Site-Scripting-Angriffen in die Anwendung einzubringen. Die Attribute `'unsafe-inline'` und `'unsafe-eval'` sollte vermieden werden, da dies keine Trennung von HTML- und Skript-Code erzwingt bzw. aufhebt und somit die Sicherheit drastisch reduzieren.

Es ist zu beachten, dass der Einsatz einer starken CSP nicht trivial ist. Vor dem Ausrollen können Policies mit Reporting-Direktiven getestet werden.

Hinweis: Im Testgegenstand werden aktuell Inline-Skripte eingesetzt, was für den Einsatz einer starken CSP Code-Änderungen notwendig macht.

Referenzen:

- https://cheatsheetseries.owasp.org/cheatsheets/HTTP-Headers_Cheat_Sheet.html
- <https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP>

C5.8 Serverkonfiguration für MIME-Sniffing

Bedrohung

Der `X-Content-Type-Options` HTTP-Header stellt sicher, dass der vom Webserver ausgelieferte Content-Type auch im Browser als solcher umgesetzt wird.

Der Microsoft Internet Explorer implementiert eine Technologie, welche es dem Browser ermöglicht, den MIME-Type der aufgerufenen Ressource zu bestimmen. Diese Technologie ist als MIME-Sniffing bekannt und kann von einem Angreifer als „drive-by download“ Möglichkeit ausgenutzt werden. Hierbei nutzt der Angreifer eine Datei-Upload-Funktionalität in der Anwendung, um beispielsweise eine dynamische HTML-Datei hochzuladen, anstelle der vom Webserver erwarteten Bilddatei. Dies kann er z. B. durch eine Veränderung der Dateieindung erreichen. Wird nun diese Datei von der Anwendung an den Browser ausgeliefert, so kann es in einigen Browser-Versionen vorkommen, dass durch MIME-Sniffing der Inhalt der Bilddatei als HTML gerendert wird. Dies hat zur Folge, dass der dynamische HTML-Inhalt im Browser des Anwenders zur Ausführung kommt.

Ist der `X-Content-Type-Options` HTTP-Header gesetzt, so wird der Browser angewiesen, den im HTTP-Header gesetzten Content-Type als MIME-TYPE zu verwenden.

C5.8.1 Mime-Sniffing Angriffe potenziell möglich

[niedrig]

Beobachtung

Der X-Content-Type-Options-Header wird in den Server-Responses nicht gesetzt.



```
Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Date: Tue, 27 May 2025 08:22:08 GMT
3 Cache-Control: no-store, max-age=0
4 Set-Cookie: sympa_session=10437563926200; path=/; secure; HttpOnly
5 Strict-Transport-Security: max-age=16070400; includeSubDomains
6 Upgrade: h2
7 Connection: Upgrade, Keep-Alive
8 Keep-Alive: timeout=5, max=100
9 Content-Type: text/html; charset=utf-8
10 Set-Cookie: TS0183cc68=
01021d36f227047b327c3a6294015910d2c6c068dbe293ffdc7e19b3640e05eccc1e0e1d52df91fc947eb8e39
2a2b5cc4555abe8a364886718e80fa5114969a3275db3598d4cdad038627099b78fa0cbe32aa72202d55e330b8
ff476187d076428c8c6352635afc1e35f93dfc6094242d77ffe924170766fafbcddf3cf91241a6f4e4fad9;
Path=/; Domain=.pentest-extern.muenchen.de; Secure; HttpOnly;
11 Set-Cookie: TS383f4674027=
087179dd52ab20003fe2b3168eeb9197eef0288ab6a3a468fd0db00bc15c76d0ee2ca6fdcbc43d0a080ee29cf6
11300049546ddc85bacd18b4a8eb1eb510e6b69d9dfc1990a1f785ed31b4e8ec9bbdda0a0f6a0d0cfb269983
8cb2eaaeca5d; Secure; Path=/
12 Content-Length: 18845
```

Ausnutzbarkeit

Durch MIME-Sniffing ist es möglich, den Content-Type der angeforderten Ressource, z. B. einer Bilddatei oder einer ZIP-Datei so zu beeinflussen, dass deren Inhalt im Browser des Nutzers interpretiert wird.

Dadurch kann das Ausführen aktiver Inhalte im Browser forciert und damit Zugriff sowohl auf Websiteinhalte als auch den Browser-Speicher ermöglicht werden.

Maßnahme

Durch das explizite Setzen von X-Content-Type-Options: nosniff, kann der Internet Explorer und Google Chrome vor MIME-Sniffing Schwachstellen geschützt werden.

C5.9 Serverkonfiguration für Anti-Clickjacking

Bedrohung

Clickjacking beschreibt einen Angriffsvektor, in welchem der eigentliche Inhalt einer Anwendung durch eine legitim erscheinende andere Webseite visuell überdeckt wird. Dafür wird die Anwendung transparent (in einem Frame) vor die vom Angreifer kontrollierte und plausibel gestaltete Webseite gelegt. Sämtliche Eingaben werden dann nicht, wie vom Nutzer beabsichtigt, an die Webseite des Angreifers gesandt, sondern an die transparent davor gelegte Anwendung. So kann ein Nutzer unter Umständen dazu bewegt werden, Aktionen in der angegriffenen Anwendung in seinem Nutzerkontext auszuführen – im Glauben, er würde nur mit der harmlosen, vom Angreifer präparierten Webseite interagieren.

Clickjacking kann per Konfiguration unterbunden werden, entweder per CSP-Direktive `frame-ancestors` oder durch den HTTP-Header `X-Frame-Options`. Der Browser verhindert dann entsprechend, dass die Anwendung in fremde Frames eingebettet wird.

Eine komplexere Content Security Policy zum Verhindern von Clickjacking sieht beispielsweise wie folgt aus:

```
Content-Security-Policy: frame-ancestors 'self' '*.*mgm-sp.com'
                        'https://sectest.team';
```

Hierdurch wird das Einbetten der Anwendung in andere Seiten auf der gleichen Domain (`'self'`), sowie in Seiten unter `mgm-sp.com` (beliebige Subdomains, egal ob `http` oder `https`), und in die Seite `https://sectest.team` erlaubt. In anderen Seiten unterbindet der Browser dies. Gerade im betrieblichen Umfeld ist oft ein solches Whitelisting nötig. Der einfachste Fall, das Einbetten in Frames komplett zu verbieten, lässt sich via `frame-ancestors 'none'` realisieren.

Falls eine signifikante Anzahl der Nutzer der Anwendung ältere Browser verwenden, die den CSP-Standard noch nicht implementiert haben, muss auf den HTTP-Header `X-Frame-Options` zurückgegriffen werden. Das Einbetten in Frames kann hier entweder via `DENY` komplett unterbunden, oder aber via `SAMEORIGIN` auf den Origin der Anwendung beschränkt werden. Whitelisting ist über den Zusatz `ALLOW-FROM` möglich, was jedoch nicht von allen wichtigen Browsern unterstützt wird, generell deutlich weniger flexibel ist als per CSP-Direktive und nur mit einem Origin funktioniert.

Der CSP-Standard verlangt, dass beim Vorhandensein von CSP und `X-Frame-Options`-Header die CSP-Direktive Vorrang haben muss, was einige Browser jedoch nicht respektieren.

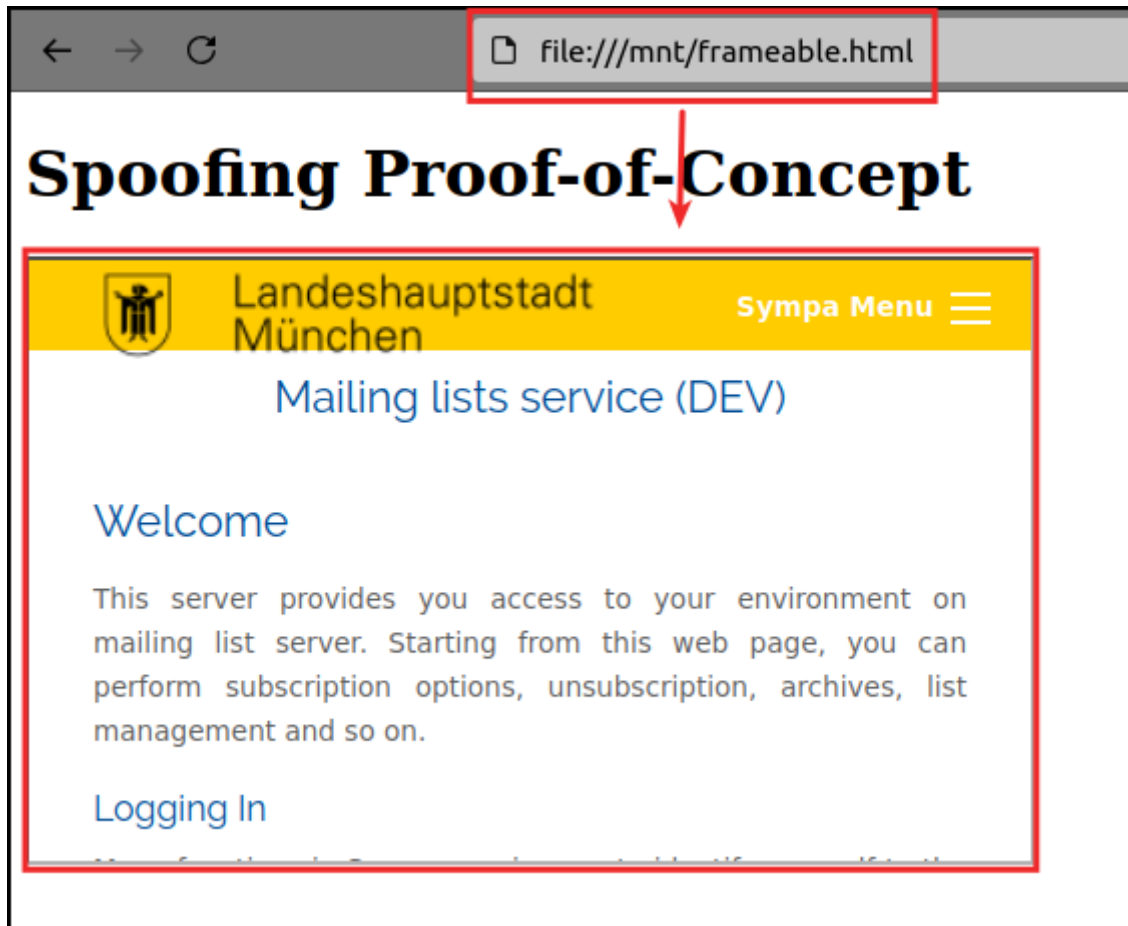
C5.9.1 Einbetten der Anwendung in Fremdkontext möglich [niedrig]

Beobachtung

Der `X-Frame-Options`-Header wird in den Server-Responses nicht gesetzt.

Ebenso ist keine CSP-Direktive konfiguriert, die das Einbetten der Seite in fremde Frames verhindert (siehe auch Kapitel C5.7.1).

Ein Beispiel für Clickjacking ist nachfolgend dargestellt. Im folgenden Screenshot ist eine vom Angreifer erstellte Webseite abgebildet, die ein `IFRAME`-Element einbettet:



Ausnutzbarkeit

Die Anwendung schützt Nutzer nicht vor Clickjacking-Angriffen. Ein Angreifer kann eine eigene Webanwendung aufsetzen, in welcher er die Sympa-Anwendung in einem iframe Element einbettet. Das iframe Element kann in der Folge mit beliebigem Inhalt überlagert werden. Dadurch ist es dem Angreifer möglich den Nutzer unwissentlich dazu zu bringen mit der Anwendung zu interagieren.

Maßnahme

Der Webserver sollte verhindern, dass die Anwendung sich im Browser in andere, nicht vertrauenswürdigen Anwendungen laden lässt. Dies kann durch Setzen des HTTP-Headers `X-Frame-Options` oder besser durch die CSP-Direktive `frame-ancestors` erreicht werden. Der jeweilige Header sollte auf allen ausgelieferten Seiten gesetzt werden.

Um jede Art von Framing zu verhindern, kann der folgende Header gesetzt werden:

```
Content-Security-Policy: frame-ancestors 'none';
```

Um ausschließlich Framing von Seiten mit dem gleichen Origin und `https://www.example.org` zu gestatten:

```
Content-Security-Policy: frame-ancestors 'self' https://www.example.org;
```


Der CSP-Standard verlangt, dass beim Vorhandensein von CSP und X-Frame-Options-Header die CSP-Direktive Vorrang haben muss, was einige Browser jedoch nicht respektieren.

C6 Logik

C7 Informationsabfluss

C7.1 Bekanntgabe von unterschiedlichen Systeminformationen

Bedrohung

Als Information-Disclosure bezeichnet man die Tatsache, dass eine Anwendung Informationen preisgibt, die für die Bedienung nicht erforderlich sind, aber Rückschlüsse auf die Arbeitsweise oder die verwendeten Techniken zulassen.

Wenn solche Informationen nicht für den Zugriff von außen bestimmt sind, dann kann dies auch ein Datensicherheitsproblem darstellen. Alle Informationen, die für den Betrieb der Anwendung durch den Benutzer nicht notwendig sind, können einem Angreifer auch helfen in ein System einzudringen.

Das Vorhandensein einer einzigen solcher Schwachstelle ist dabei häufig nicht als Problem anzusehen. Kommen jedoch mehrere solcher Schwachstellen zusammen, dann lässt sich daraus ein Gesamtbild konstruieren, das eine Bedrohung darstellt.

Informationen wie Serverbanner, Footprints von Modulen oder Software-Komponenten und sogar Hinweise auf Autoren einer Software ermöglichen einem Angreifer gezielt nach Schwachstellen zu suchen. Eine solche Suche umfasst z. B. die Known-Vulnerabilities (siehe C5.11). Ein Angreifer kann dann weitere Stellen in der Website suchen, die die gleichen Eigenschaften und damit Angriffspunkte haben.

Mögliche Maßnahme

Zur Einschränkung von Information-Disclosure sollte ein gewisser Aufwand getrieben werden. Einige Ansatzpunkte sind:

- Kommentare aus HTML-Seiten entfernen
- Server-Banner entfernen oder neutral formulieren (für Apache in der `httpd.conf` `ServerToken prod` setzen)
- Fingerprints, Kommentare und andere Module entfernen
- typische Merkmale wie `META`-Tags, Formvariablen usw. neutral umbenennen
- Fehlermeldungen abfangen und dafür neutrale Fehlerseiten liefern
- Serverkonfiguration überprüfen (z. B. Indexing abschalten, siehe C5.3)
- Enumeration verhindern

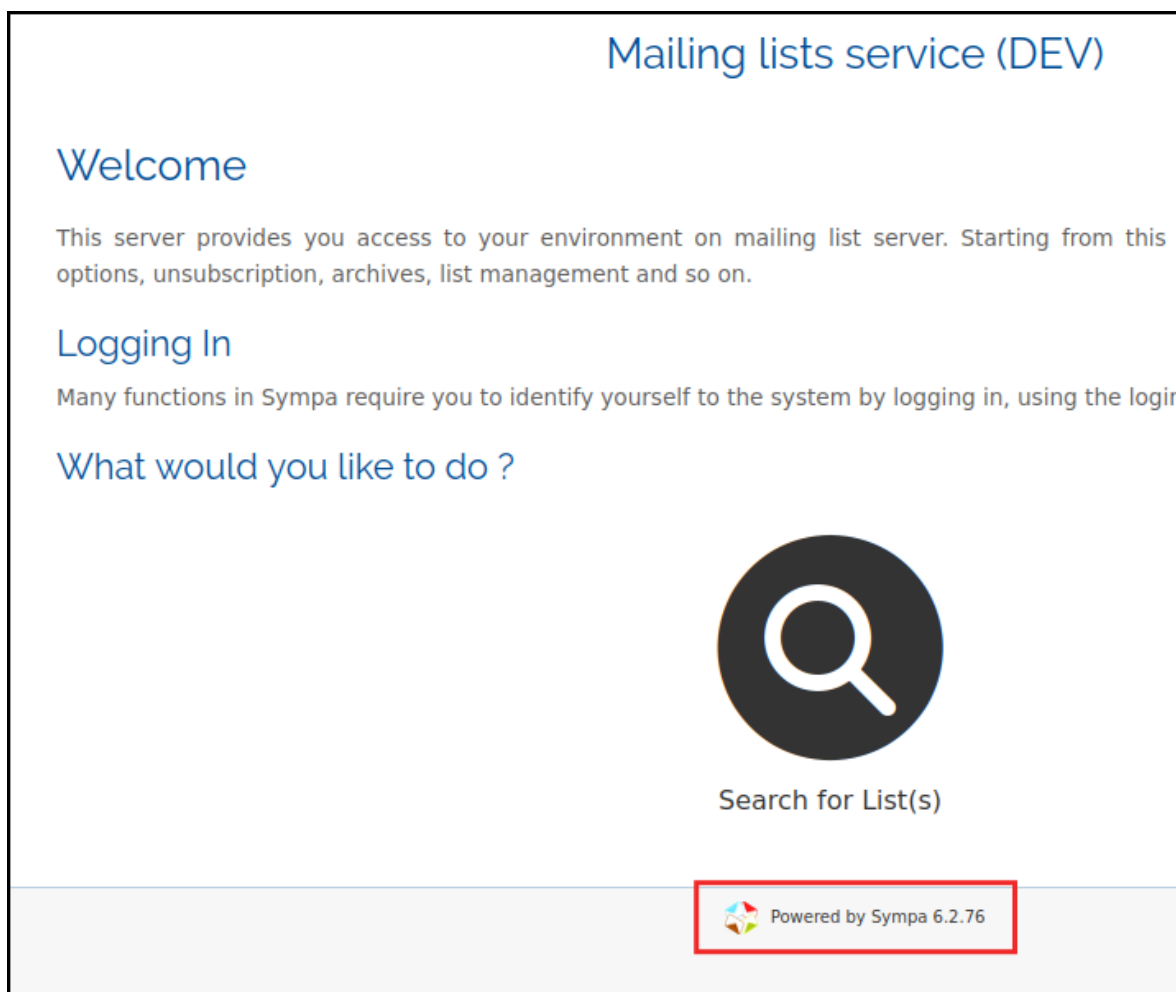
Referenzen:

<https://cwe.mitre.org/data/definitions/497.html>

C7.1.1 Bekanntgabe von verwendeter Technologie und exakter Version **[niedrig]**

Beobachtung

Die Landingpage der Sympa-Webanwendung verzeichnet die verwendete Sympa-Version, v6.2.76, im Footer der HTML-Seite:



In den HTTP-Antworten der Sympa-Anwendung werden verschiedene JavaScript-Biblotheken per HTML-Tag eingebunden. In den Adressen werden die Namen und exakten Versionsinformationen ausgegeben:

```
<script src="/static-sympa/js/jquery.js?v3.6.0">
</script>
<script src="/static-sympa/js/jquery-migrate.js?v1.4.1">
</script>
<script src="/static-sympa/js/jquery-ui/jquery-ui.js?v1.13.2">
</script>
<script src="/static-sympa/js/jqplot/jquery.jqplot.min.js?v1.0.8">
</script>
<script src=
"/static-sympa/js/jqplot/jqplot.categoryAxisRenderer.min.js?v1.0.8">
</script>
<script src="/static-sympa/js/jqplot/jqplot.barRenderer.min.js?v1.0.8">
</script>
<script src=
"/static-sympa/js/jqplot/jqplot.canvasAxisTickRenderer.min.js?v1.0.8">
</script>
<script src="/static-sympa/js/jqplot/jqplot.canvasTextRenderer.min.js?v1.0.8
">
</script>
<script src=
"/static-sympa/js/jquery-minicolors/jquery.minicolors.min.js?v2.3.6">
</script>
<script src="/static-sympa/js/sympa.js?v6.2.76">
</script>
```

- jquery.js v3.6.0
- jquery-migrate.js v1.4.1
- jquery-ui.js v1.13.2
- jqplot.min.js v1.0.8
- jqplot.categoryAxisRenderer.min.js v1.0.8
- jqplot.barRenderer.min.js v1.0.8
- jqplot.canvasAxisTickRenderer.min.js v1.0.8
- jqplot.canvasTextRenderer.min.js v1.0.8
- respond.min.js v1.4.2
- jquery.minicolors.min.js v2.3.6
- sympas.js v6.2.76
- what-input.js v4.2.0
- foundation.min.js v6.4.2

Des Weiteren liefert der Endpunkt /sympassoap die Version der verwendeten Perl SOAP-Bibliothek aus.

```
Response
Pretty  Raw  Hex  Render
1 HTTP/1.1 500 Internal Server Error
2 Date: Wed, 28 May 2025 15:21:14 GMT
3 Set-Cookie2: sympa_session=51086606999808; domain=pentest-ex
4 SOAPServer: SOAP::Lite/Perl/1.27
5 Strict-Transport-Security: max-age=16070400; includeSubDomain
6 Upgrade: h2
7 Connection: Upgrade, close
8 Content-Type: text/xml; charset=utf-8
9 Set-Cookie: BIGSC=!yRr4MA0Aa0ogLi4ErCh4Gj1n/EtHk8GxnQ/ur4kOz
10 Set-Cookie: TS0183cc68=
    01021d36f286f46d72a0250dcfe5db1bac2a2e441e6ffb40dfb07f8d2ecf
    25647072; Path=/; Domain=.pentest-extern.muenchen.de; Secure
11 Content-Length: 536
12
```

Ausnutzbarkeit

Name und Versionsnummer einer Anwendungskomponente, können dazu verwendet werden, öffentlich bekannte Schwachstellen dieser Komponente ausfindig zu machen.

Für einen Angreifer sind solche öffentlich bekannten Schwachstellen wertvolle Ansatzpunkte, da oftmals bereits fertige Exploits existieren, welche direkt zur Ausnutzung der Sicherheitslücke verwendet werden können.

Maßnahme

Die Offenlegung interner Informationen, welche nicht für den korrekten Betrieb der Anwendung benötigt werden, sollte auf allen Ebenen unterbunden werden.

C7.2 Fehlerseiten

Bedrohung

Fehlerseiten enthalten oft interne Information, z. B. Statusmeldung der Datenbank, Java-Stacktraces oder Serverbanner, die dazu benutzt werden können nach weiteren Schwachstellen zu suchen. Eine weitere häufig anzutreffende Schwachstelle ist, dass die Benutzereingaben ungefiltert in die Fehlerseite übernommen werden, was dann zu allen Formen von Cross-Site-Scripting führen kann.

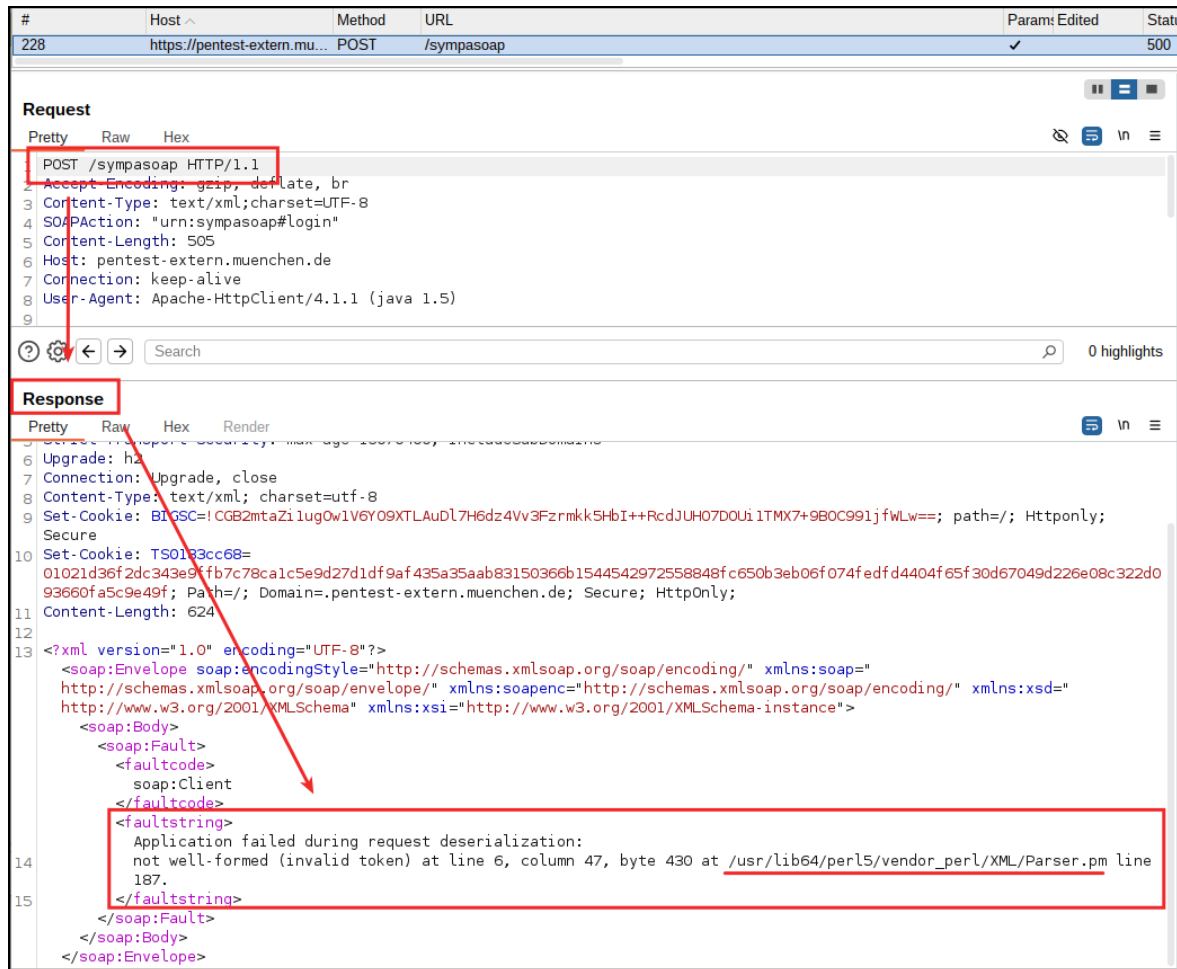
Die in den Seiten enthaltenen Informationen können einem Angreifer helfen oder ihn ermutigen gezielt nach weiteren Schwachstellen zu suchen.

C7.2.1 Bekanntgabe eines absoluten Dateipfades in Fehlerseite

[niedrig]

Beobachtung

Wir dein 500 internal Server Error angezeigt, beispielsweise bei der Eingabe unerwarteter Zeichen, so enthält die Fehlermeldung einen absoluten internen Dateipfad zur Perl-Installation.



Ausnutzbarkeit

Bestimmte Sicherheitslücken erfordern, dass der Angreifer den vollständigen Pfad zu der Datei kennt, die er einsehen möchte (z.B. Remote-File-Inclusion). Angreifer könnten das Wissen in Kombination einer solchen Schwachstelle dazu nutzen, um zum Beispiel Konfigurationsdateien der Webanwendung oder andere Dateien, die sich auf dem Betriebssystem befinden, auszulesen.

Maßnahme

Fehlermeldungen, die Informationen über interne Strukturen, wie z.B. Dateipfade, enthalten, sollten nicht an den Nutzer zurückgegeben werden.

Es sollte generell eine Standard-Fehlerseite an den Client geschickt werden, ohne dabei interne Informationen preiszugeben.

Sollte es notwendig sein, dem Endanwender einen Fehler anzuzeigen (z.B. für den First Level Support), so sollte dem Endanwender eine generische ID angezeigt werden. Unter dieser kann der Fehler anschließend bei Anruf beim Support nachgefragt werden.

C8 Kryptographie

C9 Testergebnisse automatische Werkzeuge

Im Folgenden werden die Untersuchungsergebnisse, unterteilt nach dem automatischen Analysewerkzeug, zusammengefasst dargestellt. Detaillierte Ergebnisdetails können den Ergebnisberichten und -dokumenten in der Anlage entnommen werden.

Nach der Durchführung des Scans wurden Findings in folgenden Ordnern von Review und Bewertung ausgenommen:

- .github/ (Github-Arbeitsdateien)
- www/js/ (Externe Abhängigkeiten - durch SCA abgedeckt)
 - jquery*.js
 - foundation/
 - html5shiv/
 - jqplot/
 - jquery-minicolors/
 - jquery-ui/
 - respondjs/
- spec/ (Tests und Fixtures)

Hinweis: Im Finding-Rohmaterial (SARIF- oder JSON-Dateien) sind möglicherweise auch Ergebnisse für die o.g. Ordner zu finden. Es wurden im folgenden Audit-Schritt zudem nicht alle Findings gereviewed (Details s. Kapitel A4.5)

C9.1 SAST- und SECRET-Werkzeuge

Alle bewerteten und durch uns kommentierten Findings sind im folgenden Excel-Ergebnisdokument ersichtlich:

- symp_audited.xlsx

Vom Review ausgeschlossene, sowie als Duplikat erkannte Findings, sind in folgender separaten Excel-Datei dokumentiert (in nachfolgenden Beschreibungen werden betroffene Findings allgemein als „von der Bewertung ausgenommen“ bezeichnet):

- symp_unaudited.xlsx

C9.1.1 ZARN

Diese Scans wurden mit den Standard-Regeln (Stand Mai 2025) durchgeführt. Die Scanner-Engine kam dabei in der Version 0.1.1 zum Einsatz.

Da das Werkzeug selbst keine Kritikalitätseinstufung zu den einzelnen Findings liefert, wurde für alle Findings eine Grundbewertung von warning angenommen. Alle nicht

durch uns ausgeschlossenen Findings wurden für die Auditierung in das zentrale Excel-Ergebnisdokument überführt, in welchem die Bewertung erfolgte.

Im Folgenden werden die aus dem Scan gewonnenen Erkenntnisse zusammenfassend kurz dargestellt.

Mitgelieferte Report-Artefakte

Die vom Werkzeug gelieferten (unauditierten) Roh-Daten können den folgenden Dateien entnommen werden:

- sympa-zarn-default-raw.sarif
- sympa-zarn-prototype-raw.sarif

Grobübersicht über das rohe Scan-Ergebnis des Werkzeugs

Im Folgenden wird die Anzahl zu bewertender Findings (pro Standard-Finding-Einstufung), sowie in Klammern die tatsächliche Anzahl gefundener, respektive der Anzahl ausgenommener Findings aufgelistet:

- 1.488x warning (1.673 gefunden und 185 von der Bewertung ausgenommen)

Übersicht über Bewertung durch den Auditor

Zusammenfassung der Bewertung durch den Auditor (Details siehe Excel-Ergebnisdokument im Anhang):

- 1.488 bewertete "warning"-Findings resultierten in folgenden Einstufungen:
 - 2 wurden als mittel eingestuft
 - 4 wurden als info eingestuft
 - 1482 wurden als kein Problem eingestuft

C9.1.2 Snyk Code

Diese Scans wurden mit den Standard-Regeln (Stand Mai 2025) durchgeführt. Die Scanner-Engine kam dabei in der SaaS-Version vom 23.05.2025 zum Einsatz.

Die initiale Finding-Einstufung erfolgte auf Basis der Snyk Code Standard-Bewertung (error, warning, note). Die vom Werkzeug gelieferten, nicht durch uns ausgeschlossenen Findings wurden für die Auditierung in das zentrale Excel-Ergebnisdokument überführt, in welchem die Bewertung erfolgte.

Im Folgenden werden die aus dem Scan gewonnenen Erkenntnisse zusammenfassend kurz dargestellt.

Mitgelieferte Report-Artefakte

Die vom Werkzeug gelieferten (unauditierten) Roh-Daten können der folgenden Datei entnommen werden:

— sympa-snyk-raw.sarif

Grobübersicht über das rohe Scan-Ergebnis des Werkzeugs

Im Folgenden wird die Anzahl zu bewertender Findings (pro Standard-Finding-Einstufung), sowie in Klammern die tatsächliche Anzahl gefundener, respektive der Anzahl ausgenommener Findings aufgelistet:

- 0x error (9 gefunden und 9 von der Bewertung ausgenommen)
- 1x warning (11 gefunden und 10 von der Bewertung ausgenommen)
- 0x note (9 gefunden und 9 von der Bewertung ausgenommen)

Übersicht über Bewertung durch den Auditor

Zusammenfassung der Bewertung durch den Auditor (Details siehe Excel-Ergebnisdokument im Anhang):

- 1 bewertes "warning"-Finding resultierten in folgender Einstufung:
 - 1 wurde als info eingestuft

C9.1.3 CodeQL

Diese Scans wurden mit den Standard-Regeln (Stand Mai 2025) durchgeführt. Die Scanner-Engine kam dabei in der Version 2.21.3 zum Einsatz.

Die initiale Finding-Einstufung erfolgte auf Basis der CodeQL Standard-Bewertung (error, warning, note). Die vom Werkzeug gelieferten, nicht durch uns ausgeschlossenen Findings wurden für die Auditierung in das zentrale Excel-Ergebnisdokument überführt, in welchem die Bewertung erfolgte.

Im Folgenden werden die aus dem Scan gewonnenen Erkenntnisse zusammenfassend kurz dargestellt.

Mitgelieferte Report-Artefakte

Die vom Werkzeug gelieferten (unauditerten) Roh-Daten können der folgenden Datei entnommen werden:

— sympa-codeql-js-raw.sarif

Grobübersicht über das rohe Scan-Ergebnis des Werkzeugs

Im Folgenden wird die Anzahl zu bewertender Findings (pro Standard-Finding-Einstufung), sowie in Klammern die tatsächliche Anzahl gefundener, respektive der Anzahl ausgenommener Findings aufgelistet:

- 0x warning (41 gefunden und 41 von der Bewertung ausgenommen)
- 0x error (2 gefunden und 2 von der Bewertung ausgenommen)

Übersicht über Bewertung durch den Auditor

Es waren keine Findings zu reviewen, da alle von der Bewertung ausgenommen wurden.

C9.1.4 GitLeaks

Diese Scans wurden mit den Standard-Regeln (Stand Mai 2025) durchgeführt. Die Scanner-Engine kam dabei in der Version 8.19.1 zum Einsatz.

Da das Werkzeug selber keine Kritikalitätseinstufung zu den einzelnen Findings liefert, wurde für alle Findings eine Grundbewertung von warning angenommen. Alle nicht durch uns ausgeschlossenen Findings wurden für die Auditierung in das zentrale Excel-Ergebnisdokument überführt, in welchem die Bewertung erfolgte.

Im Folgenden werden die aus dem Scan gewonnenen Erkenntnisse zusammenfassend kurz dargestellt.

Mitgelieferte Report-Artefakte

Die vom Werkzeug gelieferten (unauditerten) Roh-Daten können der folgenden Datei entnommen werden:

— sympa-gitleaks-raw.sarif

Grobübersicht über das rohe Scan-Ergebnis des Werkzeugs

Im Folgenden wird die Anzahl zu bewertender Findings (pro Standard-Finding-Einstufung), sowie in Klammern die tatsächliche Anzahl gefundener, respektive der Anzahl ausgenommener Findings aufgelistet:

— 5x warning (13 gefunden und 8 von der Bewertung ausgenommen)

Übersicht über Bewertung durch den Auditor

Zusammenfassung der Bewertung durch den Auditor (Details siehe Excel-Ergebnisdokument im Anhang):

- 5 bewerte "warning"-Findings resultierten in folgender Einstufung:
 - 1 wurde als info eingestuft
 - 4 wurden als kein Problem eingestuft

C9.2 SCA-Werkzeuge

Alle relevanten Findings sind in folgendem Excel-Ergebnisdokument ersichtlich:

— sympa-sca.xlsx

Hinweis: Da die von den Tools gewonnen Informationen auf Basis bekannter und verbreiteter Datenbanken beruhen, nehmen wir alle Findings als „bestätigt“ an. Der Absatz „Übersicht über Bewertung durch den Auditor“ entfällt daher für diese Werkzeugkategorie.

Eine feingranulare Bewertung (z.B. durch Nichtverwendung vulnerabler Bibliotheksanteile) kann aus unserer Erfahrung nur durch Projekt-Insider erfolgen.

C9.2.1 CPAN-Audit

Diese Scans wurden basierend auf der Known-Vulnerability Database Version 20250506.001 unter Verwendung der Scanner-Version 20250115.001 ausgeführt.

Die initiale Finding-Einstufung erfolgte ohne Standard-Bewertung.

Hinweis: Als "N/A" eingestufte Findings (3) wurden für die Gesamtauswertung mit low angenommen.

Im Folgenden werden die aus dem Scan gewonnenen Erkenntnisse zusammenfassend kurz dargestellt.

Mitgelieferte Report-Artefakte

Die vom Werkzeug gelieferten (unauditierten) Roh-Daten können der folgenden Datei entnommen werden:

— sympacpan-audit-raw.json

Grobübersicht über das rohe Scan-Ergebnis des Werkzeugs

Im Folgenden wird die Anzahl der Findings (pro Standard-Finding-Einstufung) aufgelistet:

- 2 Abhängigkeiten mit Known Vulnerabilities bis zu einer maximalen Werkzeugeinstufung von "critical"
- 7 Abhängigkeiten mit Known Vulnerabilities bis zu einer maximalen Werkzeugeinstufung von "high"
- 2 Abhängigkeiten mit Known Vulnerabilities bis zu einer maximalen Werkzeugeinstufung von "medium"

C10 Allgemeine negative Beobachtungen bzgl. Code-Sicherheit und – Qualität

Der folgende Abschnitt beschreibt negative Beobachtungen, welche bei der Arbeit mit dem Quellcode gemacht wurden.

C10.1 Fehlendes Abhängigkeiten Management für JavaScript

Beobachtung

Sympa verwaltet seine JavaScript-Abhängigkeiten nicht mit einem modernen Paketmanager. Stattdessen werden Builds von JavaScript-Bibliotheken, wie jQuery und FoundationJS, im eigenen Repository als Quellcode mitgeführt und verwaltet. Dies führt, u.a. hinsichtlich von Known Vulnerabilities, zu einer fehlenden Kontrolle und Sichtbarkeit, da eine begleitende SCA so nicht mehr möglich ist. Der folgende Screenshot zeigt zudem, dass eine Vielzahl von Abhängigkeiten das letzte Mal vor über 7 Jahren aktualisiert wurden:

Name	Last commit message	Last commit date
..		
foundation	Reference to map file causes HTTP 4...	7 years ago
html5shiv	Remove exec bit on some javascript ...	7 years ago
jqplot	Move all javascript libs to js directory	7 years ago
jquery-minicolors	[CVE-2021-4243] Potential XSS in jq...	3 years ago
jquery-ui	WWSympa: Update jquery-ui from 1...	2 years ago
respondjs	Remove exec bit on some javascript ...	7 years ago
jquery-migrate.js	Move all javascript libs to js directory	7 years ago
jquery.js	WWSympa: Update jQuery to 3.6.0	4 years ago
sympa.js	Merge pull request #1782 from iked...	last year

Ausnutzbarkeit

Für Angreifer sind veraltete Softwarebestandteile ein interessanter Angriffvektor, da sie möglicherweise bekannte Schwachstellen enthalten, für welche es bereits fertig entwickelte Exploit-Scripts gibt.

Ohne ein Abhängigkeitenmanagement können diese bekannten Verwundbarkeiten nur schwer identifiziert und Softwarebibliotheken nicht verlässlich aktualisiert werden.

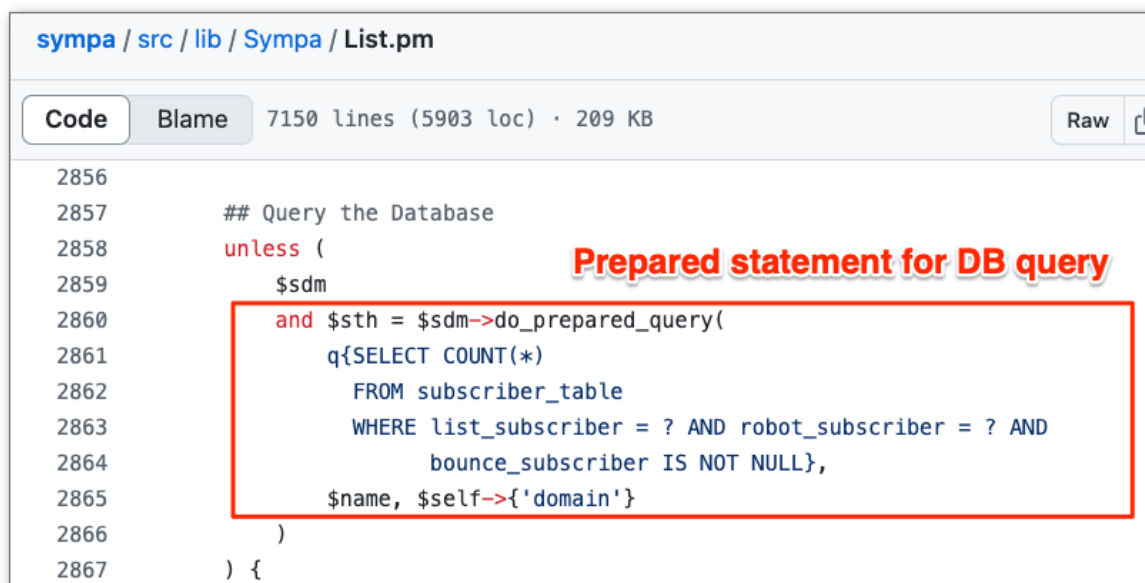
Maßnahme

Es sollte ein Abhängigkeitenmanagement genutzt werden, welches die JavaScript-Bibliotheken inventarisiert und verwaltet.

C10.2 SQLi-Gegenmaßnahmen teilweise riskant

Beobachtung

Sympa speichert seine Daten in einer relationalen Datenbank und unterstützt Treiber für PostgreSQL, Oracle und SQLite. Zum Schutz vor SQL-Injection-Angriffen verwendet Sympa vorrangig vorbereitete SQL-Anweisungen über die Subroutine `do_prepared_query`. Ein Beispiel dafür ist im untenstehenden Screenshot zu sehen. Unsere Analyse der Implementierung dieser Subroutine ergab keine Sicherheitsprobleme.



```
sympa / src / lib / Sympa / List.pm

Code Blame 7150 lines (5903 loc) · 209 KB Raw

2856
2857     ## Query the Database
2858     unless (
2859         $sdm
2860         and $sth = $sdm->do_prepared_query(
2861             q{SELECT COUNT(*)
2862                FROM subscriber_table
2863                WHERE list_subscriber = ? AND robot_subscriber = ? AND
2864                   bounce_subscriber IS NOT NULL},
2865             $name, $self->{'domain'}
2866         )
2867     ) {
```

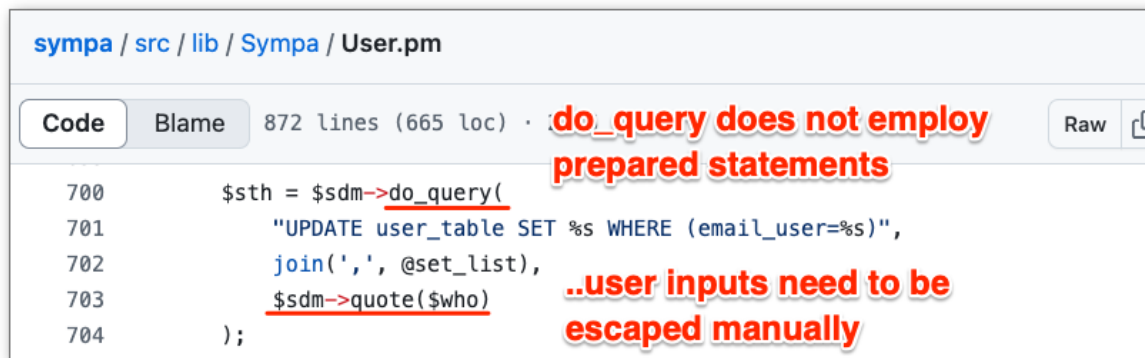
Prepared statement for DB query

Auch, wenn keine SQL-Injection Schwachstelle nachgewiesen werden konnte, sei darauf hingewiesen, dass Sympa neben der Verwendung von Prepared SQL-Anweisungen auch ungeschützte Methoden zur Datenbankabfrage einsetzt. So wird im unten dargestellten Fall die Subroutine `do_query` anstelle von `do_prepared_query` verwendet.

Im Gegensatz zu `do_prepared_query`, welches vorbereitete Anweisungen nutzt, erstellt `do_query` SQL-Abfragen, indem Benutzereingaben direkt in Zeichenkettenvorlagen interpoliert werden – ohne automatische Maskierung der Eingaben.

Dadurch ist diese Methode grundsätzlich anfällig für SQL-Injection-Angriffe und es sollte generell in Betracht gezogen werden, vollständig auf `do_prepared_query` umzusteigen.

Die getestete Anwendung mindert das Risiko, indem die Benutzereingaben entweder mit regulären Ausdrücken validiert oder mit der Subroutine `bind->quote()` maskiert werden. Es konnte im Test keine direkte Ausnutzbarkeit identifiziert werden.



The screenshot shows a code editor window for the file `sympa / src / lib / Sympa / User.pm`. It displays lines 700 to 704 of a Perl script. The code is as follows:

```
700     $sth = $sdm->do_query(  
701         "UPDATE user_table SET %s WHERE (email_user=%s)",  
702         join(',', @set_list),  
703         $sdm->quote($who)  
704     );
```

Two red annotations are present on the right side of the code:

- A red box highlights the `do_query` method call, with the text **do_query does not employ prepared statements**.
- A red box highlights the `$sdm->quote($who)` call, with the text **..user inputs need to be escaped manually**.

Ausnutzbarkeit

Auch, wenn aktuell keine direkte Ausnutzbarkeit besteht, kann ein kleiner Programmierfehler, z.B. das Vergessen des Aufrufs von `bind->quote()` bei einer zukünftigen Erweiterung unter Verwendung von nicht vertrauenswürdigen Daten, leicht zu einer SQL-Injection-Schwachstelle führen.

Ein Angreifer, der diese Schwachstelle dann erfolgreich ausnutzt, könnte alle in der Datenbank gespeicherten Informationen auslesen, verändern oder löschen. Abhängig von der Datenbankkonfiguration, kann auch ein Zugriff über die Datenbank hinaus auf das Hostsystem möglich sein.

Maßnahme

SQL-Abfragen sollten ausschließlich per Prepared Statement und nie dynamisch unter Verkettung mit Nutzereingaben erstellt werden. Eine strikte Trennung der Daten von der Anfrage-Logik, gilt hier als Best-Practice.

C10.3 „Taint Mode“ nicht durchgängig robust umgesetzt

Beobachtung

Die getestete Anwendung kann mit dem "Taint-Modus" von Perl (s. z.B. <https://www.linux-praxis.de/der-perl-taint-mode>) ausgeführt werden – einem Sicherheitsmechanismus des Perl-Interpreters, der sicherstellt, dass CGI-Parameter, die aus Benutzereingaben stammen, vor ihrer Verwendung in sicherheitskritischen Operationen wie Dateizugriffen oder Systemaufrufen validiert werden. Dieser Validierungsprozess, auch „Untainting“ genannt, besteht darin, Eingaben anhand vordefinierter sicherer regulärer Ausdrücke zu prüfen, um injection-artige Schwachstellen zu verhindern.

Die Implementierung dieses Mechanismus ist jedoch hier nicht durchgängig robust. Die regulären Ausdrücke zur Entschärfung („untainting“) sind in der Datei `wwwsympa.fcgi.in` definiert. Neben strikten Mustern finden sich dort auch großzügige Wildcard-Muster (siehe Screenshot unten), die die Schutzwirkung des Taint-Modus erheblich abschwächen und somit die Gesamtsicherheit reduzieren.

```
sympa / src / cgi / wwwsympa.fcgi.in

Code Blame 16714 lines (14726 loc) · 550 KB Raw

772  ## Regexp applied on incoming parameters (%in)
773  ## The aim is not a strict definition of parameter format
774  ## but rather a security check
775  our %in_regexp = (
776      ## Default regexp
777      '*' => '[\w\-\.\.]+',
778
779      ## List config parameters
780      'single_param' => '.*',
781      'multiple_param' => '.*',
782      'deleted_param' => '.*',
783
784      ## Textarea content
785      'template_content' => '.*',
786      'content' => '.*',
787      'body' => '.*',
788      'info' => '.*',
789      'new_scenario_content' => '.*',
790      'blacklist' => '.*',
791      'blocklist' => '.*',
792
793      ## Integer
794      'page' => '\d+|owner|editor',
795      'size' => '\d+',
796
797      ## Free data
798      'subject' => '.*',
799      'gecos' => '[^<>\\*\\$\\n]+',
800      'fromname' => '[^<>\\*\\$\\n]+',
801      'additional_field' => '[^<>\\*\\$\\n]+',
802      'dump' => '[^<>\\*\\$\\n]+', # contents email + gecoss
803
```



Ausnutzbarkeit

Zu lose definierte reguläre Ausdrücke bergen die Gefahr, eine unerwartete Eingabe im Code weiter verarbeiten zu müssen. Sollten dann im Weiteren die Eingaben nicht nochmals explizit und vollständig verifiziert werden, können daraus schwerwiegende Sicherheitslücken erwachsen.

Eine unzureichende Input-Validierung erleichtert insbesondere Deserialisierungsschwachstellen, kann aber auch die Ausnutzbarkeit von Injection-Schwachstellen erleichtern.

Maßnahme

Wildcard-Ausdrücke, im Speziellen für beliebige Zeichen, sollten in regulären Ausdrücken zur Absicherung gegen injection-artige Angriffe vermieden werden. Best-Practice wäre hier: je enger definiert, umso besser.

C10.4 Duplikate im Code

Beobachtung

Der Quellcode weist einige Stellen auf, die darauf hinweisen, dass größere Codeblöcke kopiert und an anderer Stelle wieder vollständig oder mindestens weitestgehend identisch eingefügt werden.

Für das Finden von Code Duplikaten wurde das Modul CPD (copy-paste detection) des Tools PMD (https://docs.pmd-code.org/latest/pmd_userdocs_cpd.html) mit aktivierter Programmiersprache `perl` verwendet. Gesucht wurde nach Codevorkommen, die in 100 oder mehr Stellen (Tokens) identisch sind (detaillierte Ergebnisse siehe `sympa-cpd-pm.txt`).

Die größten Funde weisen folgende Übereinstimmungen auf (alles in .pm-Dateien):

- 97 Zeilen (513 Tokens)
- 103 Zeilen (509 Tokens)
- 92 Zeilen (469 Tokens)

(insgesamt über 100 Duplikationen größer 100 Token in Perl-Dateien)

Ausnutzbarkeit

Code Duplikate weisen darauf hin, dass älterer Code kopiert und für neuere Funktionen verwendet wird, um identische oder ähnliche Logik abzubilden. Dieses Vorgehen ist jedoch aus vielfältigen Gründen nicht empfehlenswert: Zum einen kann der originale Code Fehler beinhalten (funktional oder/und aus Sicherheitssicht), die sich auch auf die neue Funktion auswirken. Zudem ist die Wartbarkeit des Codes beeinträchtigt, da nun sichergestellt werden muss, dass Änderungen (insbesondere die Korrektur von Schwachstellen) der einen Codestelle auch immer an der anderen Stelle durchgeführt werden müssen.

Maßnahme

Das Kopieren von Code ist ein sogenannter „Code-Smell“ und ist grundsätzlich zu vermeiden. Sollen gleiche Funktionalitäten an mehreren Stellen zum Einsatz kommen, bieten alle gängigen Programmiersprachen entsprechende Mechanismen, diese auszulagern, um sie an verschiedenen Stellen wiederzuverwenden (z.B. über spezialisierte Funktionen oder Klassen).

C11 Allgemeine positive Beobachtungen bzgl. Code-Sicherheit und – Qualität

Der folgende Abschnitt beschreibt positive Beobachtungen, welche bei der Arbeit mit dem Quellcode gemacht wurden.

C11.1 Projekt-Struktur

Der Quellcode der Sympa-Anwendung zeigt (abgesehen von den erwähnten Code-Duplikaten) ein hohes Maß an Qualität und Wartungsfreundlichkeit.

Die Verzeichnisstruktur des Projekts ist gut organisiert, was die Navigation durch die verschiedenen Komponenten erleichtert und ein klares Verständnis der Aufgabenverteilung ermöglicht.

Der Quellcode selbst ist übersichtlich geschrieben und umfassend kommentiert; jede Klasse ist mit ausführlichen Anleitungen und Dokumentationen versehen, was die Einarbeitung und das Verständnis erheblich erleichtert.

Darüber hinaus ist die Testsuite umfassend und deckt mit ihren Skripten die wichtigsten Funktionen und Verhaltensweisen der Anwendung effektiv ab, was die Zuverlässigkeit gewährleistet und die zukünftige Weiterentwicklung unterstützt.

C11.2 XSS-Gegenmaßnahmen

Sympa verfolgt einen systematischen Ansatz zur Vermeidung von Cross-Site-Scripting-(XSS)-Schwachstellen: bevor Daten von der FastCGI-Schicht an die View-Templates übergeben werden, werden die Inhalte mithilfe der Subroutine `Sympa::Tools::Text::encode_html` kodiert.

In Fällen, in denen benutzerdefiniertes HTML erforderlich ist – beispielsweise für individuelle Inhalte auf der Startseite einer Liste – nutzt Sympa die Bibliothek `HTML::StripScripts::Parser`, um potenziell gefährliche Tags und Attribute zu entfernen.

Wir betrachten dies als einen fundierten und effektiven Ansatz zur Minimierung des XSS-Risikos, wenn er auch in zukünftigen Erweiterungen konsequent eingehalten wird.

C12 Sonstige Beobachtungen

Im folgenden Abschnitt sind Beobachtungen festgehalten, welche im Zuge der Sicherheitsuntersuchung aufgefallen sind, jedoch in der modifizierten Instanz des Mailing List Managers der Landeshauptstadt München nicht aufgefunden werden konnten.

C12.1 Password im Klartext

Bedrohung

HTML bietet mit dem Attribut `type=password` die Möglichkeit, den Wert eines Eingabefeldes (INPUT-Tag) in der Browseransicht zu verschleiern. Wird ein solches Feld mit einem Wert (Attribute `value=`) vorbelegt, dann steht im Quelltext der Seite der Wert jedoch im Klartext und kann von einem Dritten, der physikalischen Zugang zum Browser des Benutzers hat, über die „Quelltext ansehen“-Funktion des Browsers ausgelesen werden.

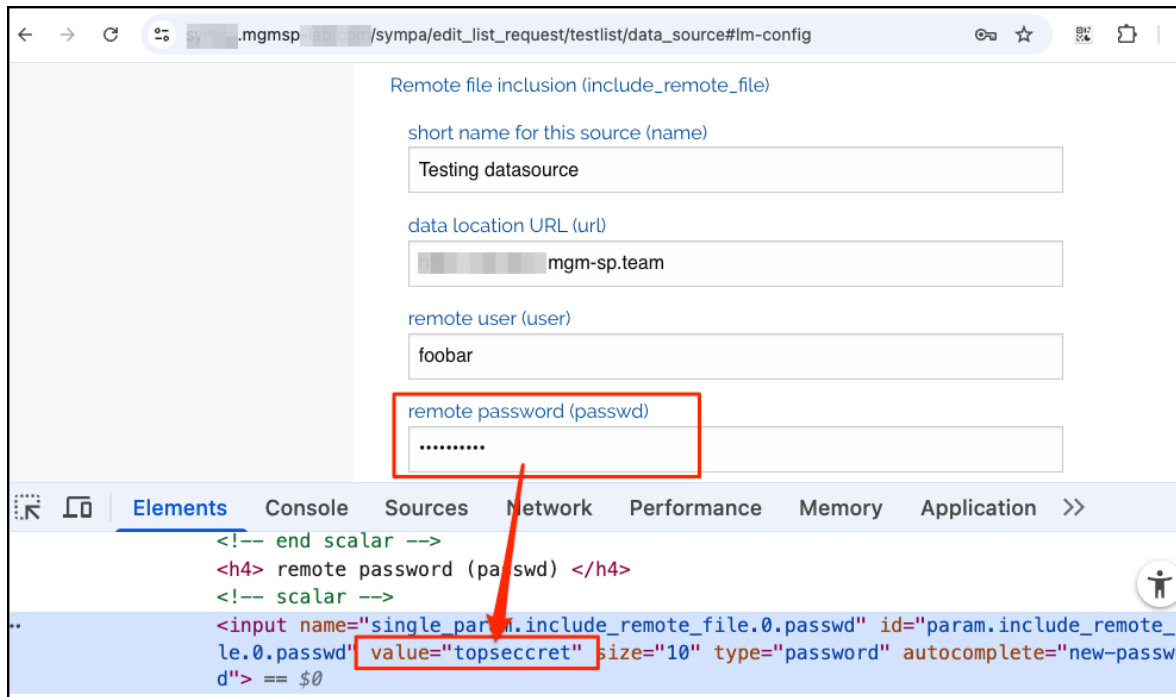
C12.1.1 Klartextpassword in HTML-Seite

[mittel]

Beobachtung

Für den Owner einer Liste ist es möglich, die Datasources seiner Liste festzulegen. Im Zuge dessen können Authentifizierungsinformationen für die Datenquelle (z.B. Datenbank) hinterlegt werden. Wird die Konfigurationsseite unter `/sympa/edit_list_request/<Mailinglistname>/data_source#lm-config` aufgerufen, können diese Informationen bearbeitet werden. Das Password wird durch Punkte im Input-Feld repräsentiert.

Das Password wird jedoch tatsächlich vom Server im Klartext ausgeliefert und im HTML-File abgelegt:



Um das Passwort im Klartext an den Client auszuliefern, muss das Passwort im Klartext im Backend gehalten werden.

Anmerkung: Diese Schwachstelle wird seit 2022 im Issue-Tracker des Sympa-GitHub-Repositories aufgeführt (<https://github.com/sympa-community/sympa/issues/1522>).

Für den aktuellen Testgegenstand, die Sympa-Instanz der Landeshauptstadt München, ist diese Schwachstelle nicht zutreffend, da die entsprechende Funktion durch die Härungsmaßnahmen den List-Ownern nicht zur Verfügung steht.

Ausnutzbarkeit

Liegen Passwörter im Klartext in der Datenbank, könnte ein Angreifer mit entsprechendem Zugriff auf die Datenbank, direkt auf diese zugreifen. Damit können die betroffenen Accounts vollständig kompromittiert werden.

Werden Passwörter im Klartext darüber hinaus an den Client ausgeliefert, erweitert dies die Angriffsoberfläche signifikant. So könnte durch clevere Clickjacking-Angriffe (siehe auch C5.9.1) oder Social Engineering bzw. „Shoulder Surfing“ das Passwort in die Hände eines Angreifers gelangen.

Maßnahme

Passwörter sollten nie im Klartext an den Client ausgegeben werden. Um trotzdem die Userexpericne im Sinne der Usability zu wahren, können stattdessen Platzhalter verwendet werden, um darzustellen, dass bereits ein Passwort für die entsprechende Datasource-Konfiguration gesetzt wurde.