10. Januar 2018

Nadja Stadelmann, Tatyana Merlo, Natalie Stalder

TBZ

Analyse Schlosslauf

M183 – Applikationssicherheit implementieren

Inhaltsverzeichnis

[Analyse Schlosslauf 1](#_Toc501521471)

[Einführung und Überblick 1](#_Toc501521472)

[Schlosslauf 1](#_Toc501521473)

[Softwareanalyse und gefundene Schwachstellen 1](#_Toc501521474)

[Sicherheitsrisiken 1](#_Toc501521475)

[Sicherheitsverbesserungen 1](#_Toc501521476)

[Resultate 1](#_Toc501521477)

[Quellen 1](#_Toc501521478)

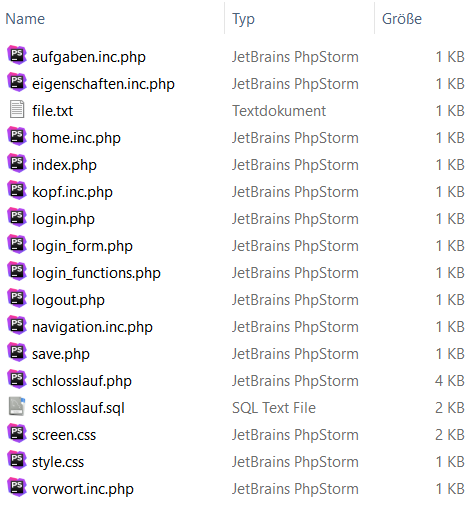
# Einführung und Überblick

# Schlosslauf

Beschrieb

Die Webapplikation «Schlosslauf» soll dazu dienen, dass sich Leute am Schlosslauf registrieren können. nach dem Login können sie ihre Daten anpassen, wenn gewünscht. Momentan kann nur jemand, der Zugriff zu einem bestehenden Konto hat, die Daten einfügen. Daher ist die Registrierung noch nicht unterstützt. Weiter soll die Applikation erweitert werden, sodass die User verschiedene Rollen haben, etwa einen Admin, der alles bearbeiten und ansehen kann, und ein Läufer, der nur die eigenen Daten begutachten kann.

# Softwareanalyse und gefundene Schwachstellen



## Architektur

Bei Betrachtung der Filestruktur fällt als erstes auf, dass bloss ein Ordner existiert. Dieser beinhaltet alle Files wild durcheinander. Das heisst, es besteht keine übersichtliche Trennung zwischen der Logik, den Daten und der Darstellung. Es existieren keine HTML-Dateien, das heisst, es besteht eine Vermischung zwischen den PHP- und den HTML-Dateien. Betreffend des Datenlayers ist ersichtlich, dass ein .txt-File und ein .sql-File existieren.

### Datenlayer

Bei genaueren Betrachtung enthält das File file.txt die Daten, welche in dem Webformular eingegeben wurden. Diese werden für jeden Eintrag untereinander aufgelistet. Die Datenbank (schlosslauf.sql) besteht aus einer einzigen Tabelle, die Logindaten speichert.

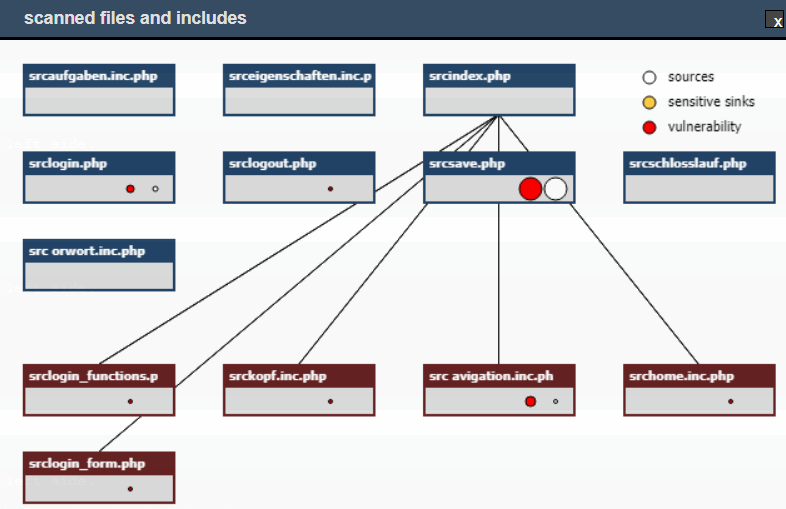
Generell macht es keinen Sinn Daten dieses Formats in ein .txt-File zu speichern, vor allem, wenn bereits eine Datenbank besteht. Wenn Daten aus dem File verändert werden sollten, ist dies eher umständlich zu lösen, während eine Datenbank bereits einfachere und sichere Mittel integriert hat.

Hier besteht sicher Handlungsbedarf.

### Businesslayer

Die Ordnerstruktur selbst gibt keine Auskunft über das Businesslayer. Um einen schnelleren Überblick zu erhalten, haben wir das Tool ‘Rips’ angewandt, welches unter anderem eine Graphik erstellt, aus der ersichtlich wird, welche Dateien wo eingebunden werden. Daraus ist ersichtlich, dass die Datei index.php der Eintrittspunkt der Applikation darstellt. Einen einzigen Eintrittspunkt zu haben ist von Vorteil, da so einige Sicherheitsprüfungen vereinfacht umgesetzt werden können. Dieser Eintrittspunkt sollte jedoch auch so umgesetzt werden, dass die anderen Files nicht einfach so aufgerufen werden können. Momentan kann ein nicht eingeloggter User alle Files öffnen, bzw. herunterladen, wenn er den zugehörigen Namen in der URL einfügt. Dies muss abgefangen werden.

Eine weitere Analyse der Datei index.php hat ergeben, dass für die Anzeige der Navigation und das Anzeigen der Inhaltsseiten überprüft wird, ob ein User eingeloggt ist. Dies ist grundsätzlich gut. Besser wäre es jedoch, 2 Templates zu erstellen, eines für die eingeloggte Ansicht und eines für den öffentlichen Bereich. So kann die doppelte Ausführung des Session-Checks umgangen werden.



### Presentationlayer

In der bereits angesprochenen Datei ‘index.php’ findet sich die Anzeigelogik der Website. Die darin eingebundenen .inc.php-Dateien stellen zusammen mit der im File integrierten HTML-Struktur die Webseite schlussendlich dar. Die meisten Dateien beinhalten reines HTML, Ausnahmen sind die Dateien navigation.inc.php und kopf.inc.php, welche Anzeige-Logik beinhalten. Die Vermischungen von verschiedenen Codearten sowie das unterschiedliche Handhaben von denselben Sachen sollte vermieden werden. Die anderen inc.php-Dateien könnten auch direkt als HTML-Files gespeichert werden, dies würde dem PHP-Interpreter einiges an Arbeit ersparen. Weiter sollten die Standardtags sich nicht in dem index.php-File befinden, sondern in einem HTML-Template besser von der Logik abgetrennt werden.

Die Formulare für das Login und für die Schlosslaufanmeldung werden unterschiedlich gehandhabt. Das Loginformular hat ein eigenes File, die Anmeldung ist mit der Logik im selben File eingebunden. Grundsätzlich sollten gleiche Dinge gleich gehandhabt werden, also sollte man sich für eine Art entscheiden.

Für das Styling der Daten wird die CSS-Datei ‘screen.css’ verwendet. Die zweite CSS-Datei ‘style.css’ wird unseres Erachtens nirgends verwendet.

## Sicherheitsrelevante Schwachstellen

### Daten

#### Login

- MD5

- Ordnerstruktur

Auf der Datenbank werden die Logindaten abgespeichert. In dem SQL-Script schlosslauf.sql ist neben dem CREATE TABLE Befehl auch ein INSERT mit Standardlogindaten zu finden. Dies ist ein grosses Sicherheitsrisiko. Nun kann man argumentieren, dass das Passwort ja verschlüsselt ist. Das Problem dabei ist jedoch, dass die Verschlüsselungsmethode MD5 angewandt wurde. Mittels einer Google-Suche («md5 <Hash aus Tabelle>») fanden wir bereits auf dem obersten Link das Passwort in Klartext und konnten uns erfolgreich einloggen.

### Logik

login: MD5 🡪 Rainbowtable

beliebig oft mit gleichem usernamen einloggen 🡪 Brute Force

noch keine Registrierung möglich, nur sekretärin mit schwachem login

## Konfiguration

Ordnerstruktur nicht anzeigen

File kann einfach angezeigt werden, datenbank kann downloaded werden 🡪 inklusive logindaten!!!

PHP-Version 5.6

## Weitere Mängel

Errorseite nicht einfach fail, sondern besser login mit usernamen

Falsche Zeichen / Rechtschreibfehler

login anzeige

# Bericht Sicherheitsrisiken

Zusammenfassung der Risiken

# Verbesserungen

## Architektur

Text mit Beschrieb, Bild neue Datenstruktur.

### Datenlayer

1. Logindaten aus schlosslauf.sql entfernen
2. Ordnerstruktur anpassen
   1. eigener Ordner
3. Daten aus file.txt in die Datenbank ‘schlosslauf’ integrieren
   1. Logik dazu nicht vergessen!
4. Andere Verschlüsselung als MD5 verwenden

### Businesslayer

1. Ordnerstruktur anpassen
2. Aufruf-Überprüfungen einbauen in allen Files

### Presentationlayer

1. Ordnerstruktur anpassen
2. HTML aus PHP-Dateien extrahieren / bzw. Dateien von PHP-Dateien in HTML-Dateien ändern
   1. inc.php-Dateien als HTML-Files speichern
      1. ausser navigation.inc.php, da diese dynamisch aufgebaut wird
   2. HTML-Template erstellen (aus index.php extrahieren)
3. style.css entfernen

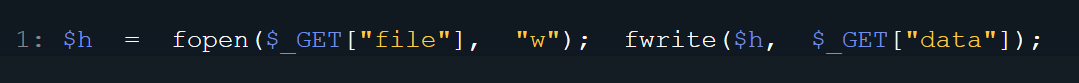
## Konfiguration

## Sicherheitsrelevante Schwachstellen

### File Manipulation

Ein Angreifer kann durch Eingabefelder versuchen ausführbaren Code einzubauen, der vielleicht in einem PHP-File gespeichert wird und beim potentiellen Ausführen Schaden anrichtet.

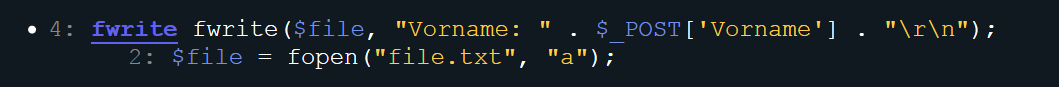
Ein Beispiel dafür ist:



#### Vorhandene Probleme

Bei der Untersuchung mittels RIPS sind 14 Meldungen zu File Manipulation aufgetreten - alle im File save.php. Diese Fehlermeldungen rühren daher, dass in der Applikation Schlosslauf die Usereingaben ohne Überprüfung ins File geschrieben werden.

Ein Beispiel:



#### Weiteres Vorgehen

Bei der Applikation ist deshalb sehr wichtig, dass man deshalb immer die Usereingaben auf allfällige Attacken untersucht (z.B. mit htmlspecialchars) und falls die einzugebenden Werte beschränkt sind, sogar eine Whitelist mit den zugelassenen Werten erstellt, so dass man den Userinput jeweils nur mit der Whitelist vergleichen muss.

### Cross Site Scripting

Ein Angreifer kann auf der Webseite ein JavaScript oder HTML Code ausführen lassen, wenn die Usereingaben nicht alle vor dem weiteren Verarbeitungsprozess validiert werden. Ist dies nicht der Fall und wird Userinput in den HTML Code eingebettet, kann der Angreifer durch seinen Code Scripts ausführen lassen, die ihm z.B. die Cookie-Daten oder die Session-ID verraten.

Ein Beispiel dafür ist:



#### Vorhandene Probleme

Während der Untersuchung durch RIPS wurden 5 Fälle von Cross Site Scripting gemeldet. Jedoch waren dies eher Fehlmeldungen, da in der Applikation Schlosslauf zwar echo-Befehle ausgeführt werden, diese jedoch keine User-Manipulierten Daten beinhalten. Deshalb ist Cross Site Scripting nicht wirklich ein Problem in dieser Applikation.

save.php, index.php , login.php

Ein Beispiel:



#### Weiteres Vorgehen

Da nach der Analyse klar ist, dass es keine Cross Site Scripting Probleme vorhanden sind, müssen wir diesbezüglich nicht viel unternehmen. Ein Teil von Cross Site Scripting wird auch durch die File Manipulation abgedeckt, bei welcher Script Code in Files abgespeichert werden könnte. Da das Vorgehen bei File Manipulation aber schon definiert ist, können wir diese Schritte hier weglassen.

### Encoding

Je nach dem wie ein File codiert ist, können einige Zeichen nicht richtig interpretiert werden. Ein grosses Problem sind immer wieder die Umlaute "ä", "ö" und "ü". Deshalb ist es wichtig gerade bei deutschen und internationalen Applikationen Encoding-Standards zu benutzen, welche die verschiedensten Zeichen anzeigen kann.

#### Vorhandene Probleme

Die Applikation kann momentan mit den Umlauten nicht umgehen und zeigt dort deshalb falsche Zeichen an in den Files aufgaben.inc.php und eigenschaften.in.php.

#### Weiteres Vorgehen

Die Files, welche falsch encoded (UTF-8) sind, müssen in konvertiert werden in ein anderes Code-Format. In dieser Applikation sind folgende Files betroffen:

* aufgaben.inc.php
* eigenschaften.in.php

### Penetrationtests

OWASP Zap durchführen

### Verschlüsselung

Die Verschlüsselung von sensitiven Daten in eine Applikation ist ein sehr wichtiger Punkt der Sicherheit. Ist diese Verschlüsselung z.B. bei Passwörtern nicht vorhanden, schwach oder veraltet, ist es sehr einfach ein Konto zu knacken. Deshalb ist es wichtig, dass man die sensitiven Daten verschlüsselt und eine passende Verschlüsselungstechnik wählt. Dabei ist nicht nur die Verschlüsselungstechnik wichtig, sondern auch der Gebrauch von Salt und Pepper. So werden die sensitiven Daten optimal geschützt.

#### Vorhandene Probleme

Die Passwörter in der Datenbank von Schlosslauf werden verschlüsselt abgespeichert. Jedoch haben wir das Problem, dass falls ein Angreifer Zugriff auf die Daten erhält, er die Passwörter sehr schnell geknackt hat, da die md5-Verschlüsselung sehr bekannt ist.   
Beim Eingeben des md5-Wertes im Browser wird einem direkt der entschlüsselte Wert geliefert. Dafür braucht man also nicht mal mehr ein Entschlüsselungsprogramm.

#### Weiteres Vorgehen

Damit die Passwörter der Applikation etwas sicherer sind, muss deshalb eine andere Verschlüsselungstechnik verwendet werden. Am besten eine eher neue, noch nicht so bekannte. So ist die Wahrscheinlichkeit, dass bereits viele Rainbow-Tables existieren, etwas kleiner. Ausserdem sollte zum Passwort ein Salt-Wert hinzugefügt werden und ein Pepper definiert sein.

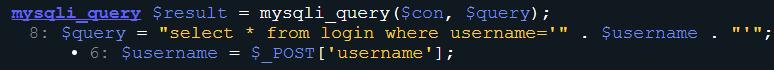
Deshalb der Vorschlag anstelle der md5-Funktion in PHP die password\_hash-Funktion zu verwenden. Sie ist sicherer als md5, jünger und bietet gleich noch die Möglichkeit einen Salt hinzuzufügen.

### SQL Injection

Kann ein Angreifer auf dem Server willkürlich SQL Code ausführen ist von SQL Injection die Rede. Hierbei ist die Überprüfung von Userinput nicht genügend umgesetzt worden, wodurch SQL Code direkt in die Query eingebunden werden kann.

Beispiel:  
mysql\_query("SELECT \* FROM users WHERE id = "  .  $\_GET["id"]);

#### Vorhandene Probleme

In dem Sourcecode wurde die Angriffsmöglichkeit 1x gefunden:  


Hier wird direkt ohne Überprüfung die Post Variabel „username“ abgefragt, wodurch SQL-Code eingefügt werden kann.

#### Weiteres Vorgehen

Es müssen Überprüfungen eingefügt werden im Code bevor die SQL-Abfrage durchgeführt wird. Hierzu sollte zuerst geprüft werden, ob die Variabel überhaupt vorhanden ist (isset), danach muss mit html\_specialchars der Input von möglichen SQL-Code bereinigt werden.

### File Inclusion

Dieses Sicherheitsproblem ergibt sich dadurch, dass Dateien dynamisch via „include“ oder „require\_once“ im Code eingebunden werden ohne diese auf ihre Gültigkeit zu überprüfen. Dadurch können Angreifer lokale und auch remote Dateien ausführen wodurch der Server kompromittiert werden kann.

Beispiel:  
include ("includes/"  .  $\_GET["file"]);

#### Vorhandene Probleme

In dem Sourcecode wurde 6x die „File Inclusion“ Angriffsmöglichkeiten gefunden.

Nach einer Analyse der betroffenen Codestellen fanden wir jedoch heraus, dass dies nur an einer Stelle ein wirkliches Problem besteht, da wir sonst die Dateien hardkodiert einbinden und nicht dynamisch.

Beispiel:  
include ($\_GET['inhalt\_mitte'])

#### Weiteres Vorgehen

Es sollte eine Whitelist erstellt werden um das bestehende Risiko zu beheben sowie auch eine bessere Absicherung für die Weiterentwicklung der App zu gewährleisten, falls in einem späteren Schritt weitere dynamische includes eingebunden werden.

### HTTP Response Splitting

Ein Angreifer kann Headers in den Response Header einfügen bei dieser Attacke. Dadurch kann er den Request auf eine externe Seite weiterleiten oder auch via SET-COOKIE die Cookies manipulieren.   
Weiterhin ist es möglich, Javascript einzufügen was zu Cross-Site-Scripting führt.

Dies wird durch dynamische Erstellung der Header ermöglicht:  
header("Location: " . $\_GET["url"]);

#### Vorhandene Probleme

In dem Sourcecode wurde 1 Mal „HTTP Response Splitting“ gefunden.

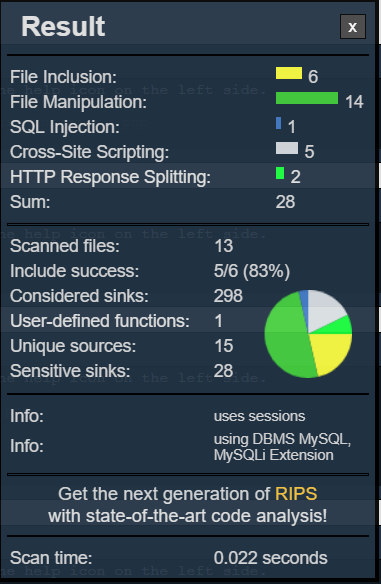
Nach einer Analyse der betroffenen Codestelle fanden wir jedoch heraus, dass dies bei uns kein Problem ist, da wir die Datei hardkodiert einbinden und nicht dynamisch:

#### Weiteres Vorgehen

Neuere PHP Versionen (5.4+) haben bereits einige Verbesserungen zur Verhinderung dieser Angriffsmöglichkeit eingebaut. Deswegen sollte auf eine möglichst neue PHP Version gewechselt werden falls in Zukunft die Seite mit dynamisch erstellten Headern erweitert wird. Weiterhin sollte auch hier eine Whitelist verwendet werden.

## Weitere Mängel

# Resultate



# Quellen

|  |  |
| --- | --- |
| Name | Link |
| Rips | https://sourceforge.net/projects/rips-scanner/?source=typ\_redirect |
|  |  |
|  |  |
|  |  |
|  |  |