Inhaltsverzeichnis

[Softwareanalyse und gefundene Schwachstellen 2](#_Toc503713479)

[~~Architektur~~ 2](#_Toc503713480)

[~~Businesslayer~~ 2](#_Toc503713481)

[Sicherheitsrelevante Schwachstellen 3](#_Toc503713482)

[Daten 3](#_Toc503713483)

[Logik 3](#_Toc503713484)

[File Manipulation 3](#_Toc503713485)

[Cross Site Scripting 4](#_Toc503713486)

[Penetrationtests 4](#_Toc503713487)

[Verschlüsselung 4](#_Toc503713488)

[SQL Injection 5](#_Toc503713489)

[File Inclusion 5](#_Toc503713490)

[HTTP Response Splitting 5](#_Toc503713491)

[Konfiguration 6](#_Toc503713492)

[PHP 6](#_Toc503713493)

[Weitere Mängel 6](#_Toc503713494)

[Encoding 7](#_Toc503713495)

[Verbesserungen 7](#_Toc503713496)

[Architektur 7](#_Toc503713497)

[Datenlayer 7](#_Toc503713498)

[Konfiguration 7](#_Toc503713499)

[Struktur 7](#_Toc503713500)

[Sicherheitsrelevante Mängel 7](#_Toc503713501)

[Weak Typing 7](#_Toc503713502)

[URL Routing 7](#_Toc503713503)

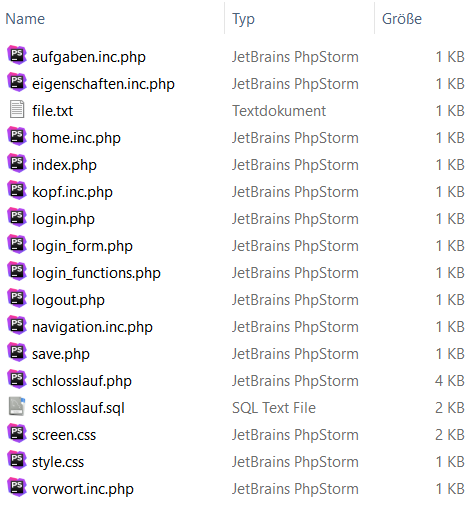
[HTML Fehler 8](#_Toc503713504)

[Code Qualitätsverbesserungen 8](#_Toc503713505)

[Signup Seite 9](#_Toc503713506)

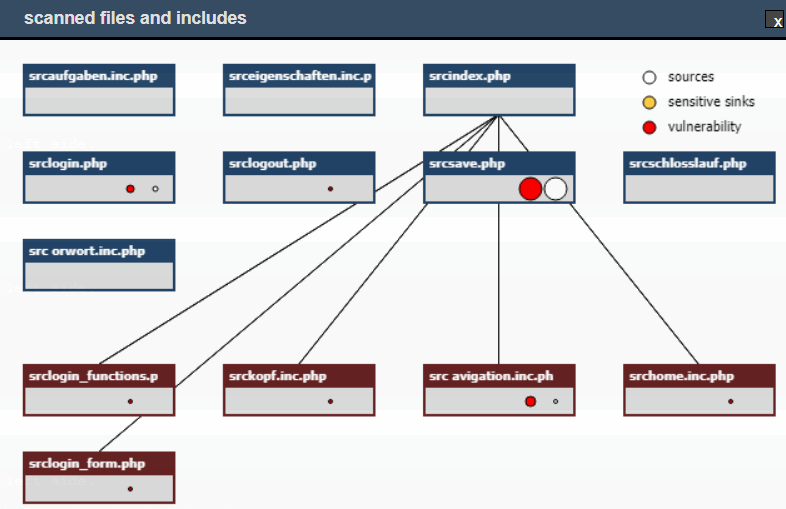
[~~Resultate~~ 9](#_Toc503713507)

# Softwareanalyse und gefundene Schwachstellen



## ~~Architektur~~

### ~~Businesslayer~~



## Sicherheitsrelevante Schwachstellen

Klassifizieren

### 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

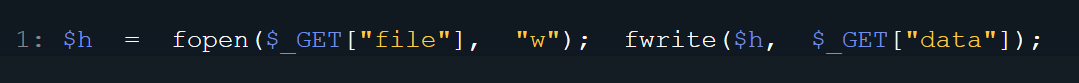
noch keine Registrierung möglich, nur sekretärin mit schwachem login

Ordnen nach OSWAB oder wie es heisst!

### 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.

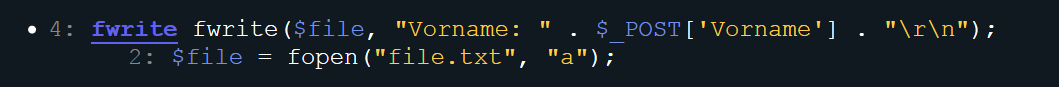
Ein Beispiel dafür ist:



#### 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 der Funktion htmlspecialchars). Falls die einzugebenden Werte beschränkt sein sollten, sollte sogar eine Whitelist mit den zugelassenen Werten erstellt werden, so dass man den Userinput jeweils nur mit der Whitelist vergleichen muss. Eine Whitelist für ein Bilderupload könnte beispielsweise aus ‘.jpg’, ‘.jpeg’ und ‘.png’ bestehen. So könnte bereits verhindert werden, dass ein PHP-File über den Upload auf unseren Server gelangt.

### 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.

### Penetrationtests

OWASP Zap durchführen und protokollieren

### Verschlüsselung

Die Verschlüsselung von sensitiven Daten in eine Applikation ist ein sehr wichtiger Sicherheitsaspekt. Ist diese Verschlüsselung z.B. bei Passwörtern nicht vorhanden, schwach oder veraltet, ist es sehr einfach einen Account zu knacken. Deshalb ist es wichtig, dass man die sensitiven Daten verschlüsselt und dazu auch 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 ein Angreifer, der sich Zugriff auf die Daten beschaffen kann, die Passwörter sehr schnell geknackt hat. Dies liegt daran, dass die verwendete md5-Verschlüsselung sehr bekannt ist.

Beim Eingeben des gespeicherten 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.

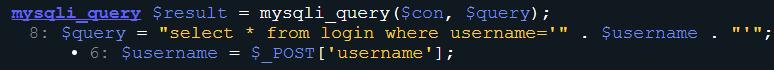
Deshalb der Vorschlag anstelle der md5-Funktion in PHP die password\_hash-Funktion zu verwenden. Sie ist sicherer als md5, jünger und bietet gleich noch die Möglichkeit einen Salt hinzuzufügen.

### 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 den Query eingebunden werden kann.

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

#### Vorhandene Probleme

In dem Sourcecode wurde die Angriffsmöglichkeit ein Mal gefunden:  


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

#### Weiteres Vorgehen

Es müssen Überprüfungen im Code eingefügt werden bevor die SQL-Abfrage durchgeführt wird. Hierzu sollte zuerst geprüft werden, ob die Variabel überhaupt vorhanden ist (isset), danach muss mit der Funktion 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 sechs Mal die „File Inclusion“ Angriffsmöglichkeiten gefunden.

Nach einer Analyse der betroffenen Codestellen fanden wir jedoch heraus, dass nur an einer Stelle ein wirkliches Problem besteht, da wir sonst die Dateien hartcodiert 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

Bei dieser Attacke fügt ein Angreifer weitere Header in den Responseheader ein. 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 ein 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 hartcodiert 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.

### Konfiguration

Ordnerstruktur nicht anzeigen

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

PHP-Version 5.6

### PHP

#### Weak Typing

PHP konvertiert Daten automatisch zu anderen Typen wodurch fehlerhafte Umwandlungen stattfinden können. In unserer Applikation werden Vergleiche immer mit „==“ durchgeführt, was zu einer Implizierten Typen Umwandlung führt. Um dies zu verhindern muss mit „===“ verglichen werden.

#### Error-handling

PHP benutzt ein im Vergleich zu anderen Sprachen schwaches Error Handling, wodurch bei Fehlern oft kein Abbruch stattfindet was zu unerwartetem Verhalten führen kann.

In unserer Applikation besteht zurzeit kaum Error Handling, deshalb sollte dies durch try&catch Blöcke umgesetzt werden. Auch sollte „error\_reporting“ auf eine möglichst hohe Stufe gesetzt werden damit alle Fehler erkennt werden.

#### Unbrauchbare Builtins

Gewisse Funktionen wie „mysql\_escape\_string“ sind zum Teil fehlerhaft oder ungenügend implementiert, wodurch eine falsche Sicherheit vermittelt wird.

Bei unserer Applikation muss vor der Verwendung der Funktionen eine Recherche über deren Aktualität durchgeführt werden um dieses Risiko zu vermindern.

#### URL Routing

PHP verwendet einen internen Routing Mechanismus welcher Dateien mit der Endung „.php“ verwendet. Dadurch werden Source Code sowie auch weitere relevante Dateien in öffentlich zugreifbaren Ordnern abgelegt. Durch eine Fehlkonfiguration können Angreifer nun auf diese Dateien zugreifen.

Um diesen Zugriff zu verhindern muss die Konfiguration unter „.htaccess“ angepasst werden.

#### Cross-Site Request Forgery

Ist in PHP ohne eingesetztes Framework, wie es bei uns der Fall ist, nicht standardmässig vorhanden und muss implementiert werden, um ungewollte Aktionen der authentifizierten Benutzer zu verhindern, wenn sie z.B auf einen Link eines Angreifers klicken welcher zur Löschung ihres Profils führen könnte. Bei unserer jetzigen Applikation sind noch keine gefährlichen Angriffe dieser Art möglich, jedoch könnte dies in Zukunft zu einem Problem werden weshalb bereits jetzt mit einem Token dieser Angriff zu verhindern ist.

## Weitere Mängel

Errorseite nicht einfach fail, sondern besser login mit usernamen

Rechtschreibfehler

login anzeige fluchtlinien

### 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

# Verbesserungen

## Architektur

### Datenlayer

1. Andere Verschlüsselung als MD5 verwenden

## Konfiguration

## Struktur

* Resources Ordner hinzugefügt um eine bessere Projektstruktur zu erreichen

## Sicherheitsrelevante Mängel

### Weak Typing

* Weak typing Schwäche von PHP umgangen durch die Verwendung von === anstatt ==, wodurch bei Vergleichen kein implizierter Cast durchgeführt wird.
* Yoda Expression verwendet, um bei Vergleichen zu verhindern, dass Variablen den Wert zugewiesen bekommen was immer als „true“ ausgewertet wird, wenn ein Programmierfehler vorhanden ist.

### URL Routing

Durch die nicht vorhandene .htaccess Konfiguration waren Source Code sowie auch weitere relevante Dateien in öffentlich zugreifbaren Ordnern abgelegt worauf Angreifer durch simples abändern der URL auf die Dateien zugreifen konnten.

Um dies zu verhindern haben wir eine .htaccess Datei hinzugefügt (<https://gist.github.com/sepehr/3370984>) welche diese Schwäche umgeht indem der Zugriff auf diese Dateien verhindert wird

### HTML Fehler

* Index.php: head Elemente hinzugefügt. „charset“ um UTF-8 Zeichen richtig in der Website anzuzeigen und „title“ um auf dem Tab den Namen anzuzeigen

### Code Qualitätsverbesserungen

Wir haben das PHPStorm Plugin „Php Inspections (EA Extended)“ verwendet um den Code zu überprüfen und allfällige PHP Fehler zu finden.

Diese Verbesserungen sind nicht oder nur leicht Sicherheitsrelevant, führen aber zu einem besseren, einfacher zu unterhaltenen Code wodurch die Weiterentwicklung vereinfacht wird

* Unnötige „?>“ Closing Tags entfernt.
* Unnötige Klammern bei „includes“ entfernt
* Single anstatt double Quotes verwendet, da diese sicherer sind da hier keine Evaluation von Variablen stattfindet
* CSS „px“ entfernt da dies redundant ist „screen.css“ hatte ein fehlendes Semikolon bei #kopf sowie auch einen Schreibfehler „absolut“ anstatt „absolute“. Auch hatte es einen Schreibfehler „hight“ anstatt „height“ bei #inhalt\_mitte
* Kopf.php: H1 onclick funktionierte nicht. Wurde durch einen Link ersetzt.

### Signup Seite

Es war bisher nicht möglich für neue Benutzer sich zu registrieren. Um dies zu ändern haben wir einen Link auf der Loginseite hinzugefügt welche auch die Registrationsseite verweist.

Die Registration weist folgende Eigenschaften auf:

* Clientseitige Password Validierung. Hilft dem Benutzer Schreibfehler zu vermeiden bei der Passwortsetzung. Muss Serverseitig nicht vorhanden sein da dies nur eine Benutzerhilfe ist.
* Serverseitige Benutzernamen Validierung. Benutzername ist Unique und muss daher überprüft werden ob er überhaupt noch frei ist.

# ~~Resultate~~

