10. Januar 2018

Nadja Stadelmann, Tatyana Merlo, Natalie Stalder

TBZ

Analyse Schlosslauf

M183 – Applikationssicherheit implementieren

Inhaltsverzeichnis

[Einführung und Überblick 3](#_Toc503890495)

[Schlosslauf 3](#_Toc503890496)

[Softwareanalyse und vorhandene Schwachstellen 4](#_Toc503890497)

[Architektur 4](#_Toc503890498)

[Datalayer 4](#_Toc503890499)

[Businesslayer 4](#_Toc503890500)

[Presentationlayer 5](#_Toc503890501)

[Sicherheitsrisiken 6](#_Toc503890502)

[File Inclusion 6](#_Toc503890503)

[File Manipulation 7](#_Toc503890504)

[Cross Site Scripting 7](#_Toc503890505)

[SQL Injection 8](#_Toc503890506)

[HTTP Response Splitting 8](#_Toc503890507)

[Verschlüsselung 8](#_Toc503890508)

[Brute Forcing 9](#_Toc503890509)

[Schwaches Login 9](#_Toc503890510)

[PHP 10](#_Toc503890511)

[Weitere Mängel 11](#_Toc503890512)

[Signup 11](#_Toc503890513)

[Errorhandling Login 11](#_Toc503890514)

[Code Qualität 11](#_Toc503890515)

[Encoding 12](#_Toc503890516)

[Rechtschreibfehler 12](#_Toc503890517)

[Bericht / Zusammenfassung 13](#_Toc503890518)

[Resultat nach Bereinigung der Lücken 14](#_Toc503890519)

[Architektur 14](#_Toc503890520)

[Datalayer 14](#_Toc503890521)

[Businesslayer 14](#_Toc503890522)

[Presentationlayer 14](#_Toc503890523)

[Sicherheitsrisiken 14](#_Toc503890524)

[File Inclusion 14](#_Toc503890525)

[File Manipulation 15](#_Toc503890526)

[SQL Injection 15](#_Toc503890527)

[HTTP Response Splitting 15](#_Toc503890528)

[Verschlüsselung 15](#_Toc503890529)

[Brute Forcing 15](#_Toc503890530)

[Schwaches Login 15](#_Toc503890531)

[PHP 15](#_Toc503890532)

[Weitere Mängel 16](#_Toc503890533)

[Signup 16](#_Toc503890534)

[Errorhandling Login 16](#_Toc503890535)

[Code Qualität 16](#_Toc503890536)

[Encoding 16](#_Toc503890537)

[Test 17](#_Toc503890538)

[Testsetup 17](#_Toc503890539)

[Testfallmatrix 17](#_Toc503890540)

[Quellen 17](#_Toc503890541)

# Einführung und Überblick

Security ist in der IT ein sehr wichtiges Thema. Gerade Webapplikationen, welche Personendaten beinhalten, sind ein beliebtes Ziel für Cyber-Attacken. Dies kommt daher, dass Daten immer wichtiger werden und mit ihnen auch einen grossen Schaden für Firmen und auch Einzelpersonen angerichtet werden können. Deshalb ist es umso wichtiger, seine Applikationen vor Angriffen zu schützen.

Im Rahme des Moduls 183 gehen wir genauer auf die Angriffsmöglichkeiten und Gefahren ein, damit wir fähig sind, unsere Webapplikation sicher zu halten. Um praktische Erfahrung im Lösen von Sicherheitsproblemen zu sammeln, analysieren und verbessern wir das Projekt Schlosslauf. Darin sind einige Mängel vorhanden, die Sicherheitsrisiken bilden. Diese gilt es zu erkennen, zu beschreiben und zu beheben. Weiter sind darin auch einige Logik- und Best-Practice-Fehler enthalten, die wir benennen werden. Der Fokus liegt jedoch nicht auf ihnen, weshalb wir nicht alle davon umsetzen werden.

## Schlosslauf

Die Webapplikation «Schlosslauf» soll dazu dienen, dass sich Leute an einem Lauf durch ein Schloss registrieren können. nach dem Login können sie ihre Gruppe 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 alle Anmeldungen ansehen kann, und ein Läufer, der nur die eigene Gruppe anpassen kann.

# Softwareanalyse und vorhandene Schwachstellen

## Architektur

Einführung überarbeiten

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.

### Datalayer

Das angesprochene Textfile «file.txt» enthält diejenigen 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 Textfile 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 dafür integriert hat. Hier besteht also Handlungsbedarf.

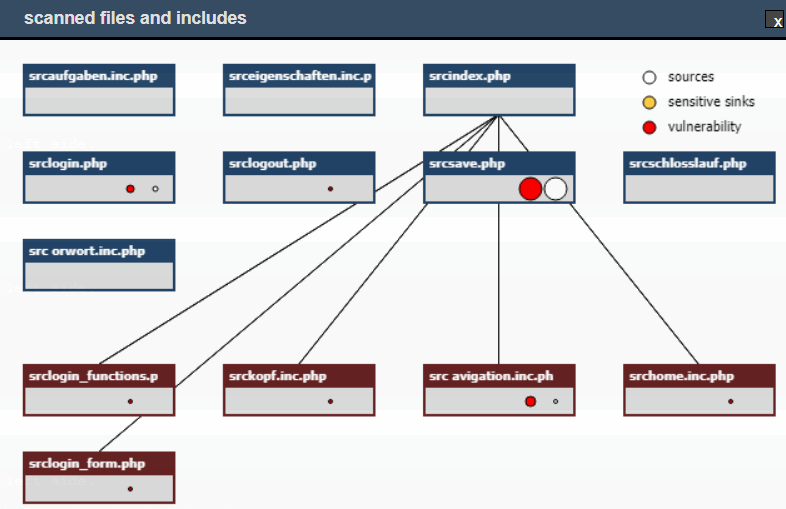
In dem Script «schlosslauf.sql» sind Standardlogindaten gespeichert. Dies kann zu grossen Sicherheitsproblemen führen, sollte der Script in falsche Hände gelangen. Der Angreifer könnte sich mit den gefundenen Daten einloggen und alles ausführen, was der entsprechende User kann. Je nach dem, über welche Berechtigungen dieser verfügt, kann so ein grosser Schaden angerichtet werden.

#### Weiteres Vorgehen

Die Datenbank wird neu strukturiert und das Textfile entfernt. Dabei werden die Daten aus dem Textfile begutachtet und ein entsprechendes Datenmodell erstellt. Dadurch entsteht ein neues Script, in dem auf keinen Fall Standardlogindaten eingefügt werden. Diese Änderungen ziehen auch einige Logikänderungen mit sich, die gesamte Datenspeicherung muss vom Beschreiben eines Files in eine Datenbankanbindung umgestellt werden.

### Businesslayer

Die Ordnerstruktur selbst gibt keine Auskunft über den logischen Aufbau der Applikation. Um einen Überblick zu erhalten, haben wir daher 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 (Single Point of Entry) 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 oder herunterladen, wenn er den zugehörigen Namen in der URL einfügt. Dies muss abgefangen werden.



Eine weitere Analyse der Indexdatei 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, zwei Templates zu erstellen, eines für die eingeloggte Ansicht und eines für den öffentlichen Bereich. So kann die doppelte Ausführung des Session-Checks umgangen werden, um Fehler zu vermeiden und Ressourcen zu schonen.

#### Weiteres Vorgehen

Die Ordnerstruktur wird überarbeitet um die Übersicht über die Webapplikation zu gewährleisten. Dies reduziert auch sicherheitsrelevante Fehler, die ansonsten aufgrund fehlender Übersicht nicht entdeckt werden würden.

Der Single Point of Entry wird durch Login-Überprüfungen in allen relevanten Files durchgesetzt. Ist kein User eingeloggt, so wird immer die Login- bzw. die Registrierungsseite des Schlosslaufs angezeigt. Dies reduziert das Risiko, dass ein Nicht-Autorisierter auf Inhalte zugreifen kann, die nicht für ihn bestimmt sind.

Um die zwei Templates umzusetzen, müsste die HTML-Struktur aus dem Indexfile ausgelagert werden. Da dies jedoch nicht sicherheitsrelevant und damit nicht erste Priorität dieser Arbeit ist, verzichten wir vorerst auf diese Verbesserung.

### Presentationlayer

In der bereits angesprochenen Datei «index.php» findet sich auch die Anzeigelogik der Website. Die darin eingebundenen «.inc.php»-Dateien stellen zusammen mit der im File integrierten HTML-Struktur die Webseite 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 derselben Art Logik sollte vermieden werden. Die anderen «inc.php»-Dateien könnten direkt als HTML-Files gespeichert werden, dies würde dem PHP-Interpreter einiges an Arbeit ersparen. Weiter sollten die Standardtags sich nicht in dem Indexfile befinden, sondern in einem HTML-Template besser von der Logik abgetrennt werden.

Die Formulare für das Login und für die Schlosslaufanmeldung werden unterschiedlich gehandhabt. Die Anzeige des Loginformulars wird in einem eigenen File gespeichert, das Anmeldeformular ist mit der Logik im selben File eingebunden. Grundsätzlich sollten gleiche Dinge gleich gehandhabt werden, also sollte man sich für eine Bearbeitungsart entscheiden. Eine Trennung der Ansicht und der Logik ist für die Wartung von Vorteil.

Der Zweck der Webapplikation «Schlosslauf» (siehe Kapitel Einführung, Schlosslauf) sieht ein Signup vor. Diese wurde bisher jedoch noch nicht umgesetzt.

Für das Styling der Schlosslaufwebseite wird die CSS-Datei «screen.css» verwendet. Die zweite CSS-Datei «style.css» wird unseres Erachtens nirgends verwendet und kann daher entfernt werden.

#### Weiteres Vorgehen

Im Rahmen der Änderung der Ordnerstruktur werden auch die Dateien mit der Benennung «.inc.php» in einen eigenen Ordner verlagert. Deshalb können sie zur Vereinfachung und besseren Lesbarkeit umbenennt werden. Wie bereits im vorhergehenden Kapitel erwähnt verzichten wir im Rahmen dieses Moduls aus Zeitgründen auf die Trennung von HTML und PHP – und bilden somit kein HTML-Template. Um dennoch die Übersicht zu verstärken, werden die reinen HTML-Files neu auch als datei.html gespeichert. So ist schneller erkennbar, in welchen dieser Files sich Logik verbirgt.

Das Registrierungsformular für den Schlosslauf wird aus dem Registrationscontroller ausgelagert, um die Formulare einheitlich zu handeln.

Das Signup muss umgesetzt werden, damit der Zweck der Applikation erfüllt werden kann.

Die unnötige Datei «style.css» birgt zwar keine Sicherheitsrisiken, dennoch wird sie aus dem Projekt entfernt.

## Sicherheitsrisiken

Im Web tummeln sich diverse Angreifer, welche jede sich durch Sicherheitslücken bietende Möglichkeit nutzen. Um dies zu verhindern muss man die Risiken erkennen und verhindern oder eindämmen. Um Probleme im Code zu finden, führten wir einen Scan mit RIPS durch, welches viele der beschriebenen Risiken fand. Einige Sicherheitslücken, wie beispielsweise das Brute Forcing oder ein schwaches Login kann von solchen Scannern jedoch nicht gefunden werden. Einige Risiken fanden wir daher auch durch Ausprobieren von Schwachstellen, welche wir in diesem Modul kennengelernt haben.

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 als Angriffsmöglichkeiten gefunden.

Nach einer Analyse der betroffenen Codestellen fanden wir jedoch heraus, dass nur an einer Stelle ein wirkliches Problem besteht, da wir an den anderen Stellen die Dateien nicht dynamisch sondern hartcodiert einbinden.

Beispiel:

include ($\_GET['inhalt\_mitte'])

Weiteres Vorgehen

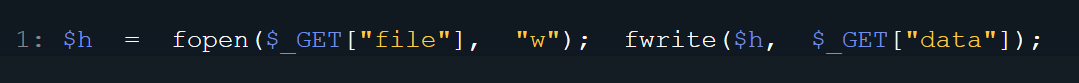
Es sollte eine Whitelist erstellt werden um das bestehende Risiko zu beheben sowie auch eine bessere Absicherung für die Weiterentwicklung der Applikation zu gewährleisten, falls in einem späteren Schritt weitere dynamische Includes eingebunden werden.

Kategorisierung: nur ein Problem, dies muss jedoch behoben werden

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.

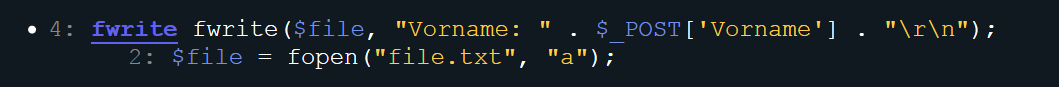
Ein Beispiel dafür ist:



Vorhandene Probleme

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

Ein Beispiel:



Weiteres Vorgehen

Bei einer Webapplikation ist deshalb sehr wichtig, dass man die Usereingaben immer 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.

Kategorisierung: Änderung erforderlich im Rahmen des Files 🡪 wird in db geändert und verschwindet sowieso – Whitelist wird im Rahmen der File Inclusion auch umgesetzt; htmlspecialchars verwenden

Cross Site Scripting

Ein Angreifer kann auf einer Webseite JavaScript- oder HTML-Code ausführen lassen, wenn nicht alle Usereingaben vor dem weiteren Verarbeitungsprozess validiert werden. Wird diese Validierung nicht durchgeführt und durch den Userinput neuen 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 Falschmeldungen. In der Applikation Schlosslauf werden zwar echo-Befehle ausgeführt (Dateien «save.php», «index.php» und «login.php»), diese beinhalten jedoch keine user-manipulierte Daten. Deshalb ist Cross Site Scripting kein bestehendes Problem in dieser Applikation.

Ein Beispiel:



Weiteres Vorgehen

Da nach der Analyse klar ist, dass 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.

Kategorisierung: keine Änderungen erforderlich

### SQL Injection

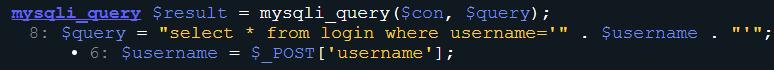
Kann ein Angreifer auf dem Server willkürlich SQL-Code ausführen, ist von SQL Injection die Rede. Hierbei ist die Überprüfung von Userinput nicht genügend umgesetzt worden, wodurch SQL-Code direkt in den Query eingebunden werden kann.

Beispiel:

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

#### Vorhandene Probleme

Im Sourcecode wurde die Angriffsmöglichkeit ein Mal gefunden:



Hier wird direkt ohne Überprüfung die POST-Variable «username» in den SQL-Query eingefügt, wodurch SQL-Code eingefügt werden kann, welcher dann auch tatsächlich auf der Datenbank ausgeführt wird.

#### Weiteres Vorgehen

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

Kategorisierung: Änderung erforderlich, ganze Tabellen und somit die gesamte Anmeldungen könnte durch diese Sicherheitslücke gelöscht 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. Weiter 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, für den Fall, dass in Zukunft die Seite mit dynamisch erstellten Headern erweitert wird. Weiter sollte auch hier eine Whitelist verwendet werden.

Kategorisierung: Änderung PHP Version, ansonsten kein Problem

### Verschlüsselung

Die Verschlüsselung von sensitiven Daten in einer 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 von mehreren Webseiten 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 für die verwendete Methode 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.

Kategorisierung: extrem wichtig, unbedingt anpassen

### Brute Forcing

Automatiertes Ausprobieren wird Brute Forcing genannt. Besonders zum Knacken von Zugangsdaten wird diese Angriffsmöglichkeit gerne verwendet. Ein allfälliger Username wird mit unendlich vielen Passwortkombinationen versucht zu knacken. Ist dem Angreifer ein Username bekannt, ist es für ihn umso einfacher, das Login zu knacken – er muss nur einen Script laufen lassen, welcher diverse Passwörter ausprobiert.

Um dies zu verhindern, kann beispielsweise der User nach einer bestimmten Anzahl Versuche für eine gewisse Zeit blockiert werden. Diese Angaben sollten jeweils an den Applikationszweck und die Art der verwendeten Daten angepasst werden.

Wenn man Brute Forcing über diesen Weg unterbindet taucht ein neues Problem auf. Der Angreifer kann durch Eingabe eines falschen Logins den User blockieren, so dass dieser nicht mehr auf das System zugreifen kann. Je nach dem kann auch so grosser Schaden angerichtet werden. Deshalb sollte man dem User die Möglichkeit bieten, sein Passwort zurückzusetzen und damit eine Passwortsperre aufzuheben.

#### Vorhandene Probleme

Momentan kann derselbe User x Mal nacheinander sein Passwort falsch eingeben. Dies sollte unbedingt unterbunden werden. Bei der Applikation Schlosslauf ist es nicht so relevant, wenn der User für beispielsweise eine Stunde gesperrt ist, dennoch wäre auch hier eine Funktion zum Passwort zurücksetzen von Vorteil. Die Mailadresse wird in der Applikation sowieso gespeichert, diese kann dazu benutzt werden.

#### Weiteres Vorgehen

Diese Sicherheitslücke wird schnell vergessen, da andere Probleme wie beispielsweise die SQL Injection viel prominenter sind. Dennoch darf sie nicht vergessen werden. Wenn wir genügend Zeit haben, werden wir mittels einem Counter und einem Timestamp die Session nach einigen Versuchen sperren.

Noch machen

Kategorisierung: wichtig, wird tendentiell vergessen, jedoch nicht das wichtigste für uns

### Schwaches Login

Mit dem Brute Forcing einhergehend ist ein schwach gewähltes Login ein grosses Problem. Dies ist natürlich ein Problem auf der Anwenderseite. Jedoch trägt auch der Programmierer Verantwortung für dieses allbekannte Risiko. Er kann bei der Passworterstellung den User zwingen, gewisse Zeichenarten zu verwenden und oft benutzte Buchstaben-, Zahlenkombinationen und Wörter verbieten. Ausserdem ist er zuständig, dass Testlogins entfernt werden und allfällige vordefinierte Administratorlogins keine Standardlogindaten enthalten.

#### Vorhandene Probleme

Der einzige bestehende User der Schlosslaufapplikation ist eine Sekretärin mit Adminrechten. Ihr Login besteht aus dem Usernamen «test» und dem Passwort «123». Beides sind Angaben, welche oft genutzt werden und es so einem Angreifer sehr leicht macht, an Administratorrechte zu gelangen.

Dazu kommt, dass das Login im Datenbankscript abgespeichert ist. Der Angreifer muss also nur dieses File herunterladen und schon kann er sich einloggen. 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.

#### Weiteres Vorgehen

Da wir uns in der Testumgebung bewegen wird das Login noch nicht angepasst. Sobald die Applikation jedoch in die Produktion gelangt, wird das Testlogin entfernt und durch eine sicherere Variante ersetzt.

Die Passwortüberprüfung auf bestimmte Zeicheninhalte wird eingeführt, wenn andere sicherheitsrelevanten Lücken geschlossen sind. Bis dahin hat der User die Verantwortung dafür.

Das Datenbankscript darf nicht mehr zum Download freigegeben werden. Am besten wird dies gar nicht erst auf dem Server in dieser Filestruktur gespeichert.

Kategorisierung: sehr wichtig

### PHP

Die falsche Anwendung einer Programmiersprache kann ebenso zu Fehlern und Sicherheitslücken führen. Darauf gehen wir in diesem Kapitel genauer ein.

#### Version

Wie bereits angesprochen benutzt die Applikation eine zu niedrige PHP-Version (Version 5.6). Diese sollte auf einen neueren Standard gehoben werden, um allfällige wichtige Neuerungen zu gewährleisten (version ergänzen!).

#### Weak Typing

PHP konvertiert Daten automatisch in andere Datentypen wodurch fehlerhafte Umwandlungen stattfinden können. In unserer Applikation werden Vergleiche immer mit «==» durchgeführt, was zu einer implizierten Typenumwandlung führt. So kann es unter Umständen zu falschen Programmflüssen kommen, was wiederum zu einem Sicherheitsrisiko werden kann. Um dies zu verhindern muss mit «===» verglichen werden.

Kategorisierung: wichtig, wird oft vergessen

#### Errorhandling

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

In unserer Applikation besteht zurzeit kaum Errorhandling, 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. Was ist das error reporting?

Kategorisierung: für korrektes Verhalten wichtig

#### Unbrauchbare Builtins (Buildins?)

Gewisse Funktionen wie «mysql\_escape\_string» sind zum Teil fehlerhaft oder ungenügend implementiert, wodurch eine falsche Sicherheit vermittelt wird. Was machen die Funktionen?

Bei unserer Applikation muss vor der Verwendung der Funktionen eine Recherche über deren Aktualität durchgeführt werden, um dieses Risiko zu vermindern. Die ungenügenden Implementationen müssen überarbeitet werden.

#### 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 und auch auf die Verzeichnisse zugreifen.

Um diesen Zugriff zu verhindern muss die Konfiguration unter «.htaccess» angepasst werden.

#### Cross-Site Request Forgery

Dies ist normalerweise in PHP integriert. Verwendet man jedoch PHP ohne eingesetztes Framework, wie es bei uns der Fall ist, so ist es nicht standardmässig vorhanden. Das Cross-Site Request Forgery muss implementiert werden, um ungewollte Aktionen der authentifizierten Benutzer zu verhindern. Wenn ein User etwa auf einen Link eines Angreifers klickt, welcher zur Löschung ihres Profils führen könnte, wird dies unterdrückt. Bei unserer jetzigen Applikation sind noch keine gefährlichen Angriffe dieser Art möglich, jedoch könnte dies in Zukunft zu einem Problem werden. Deshalb gilt es bereits jetzt, mit einem Token dieser Angriff zu verhindern. Irgendwie unklar was das genau ist

## Weitere Mängel

Neben den Sicherheitsrelevanten Mängel sind uns noch einige weitere Probleme aufgefallen, auf die wir in diesem Kapitel eingehen. Dies beinhaltet unter anderem auch Logikfehler.

### Signup

Der Sinn der behandelten Webapplikation ist, dass sich diverse Benutzer für einen Schlosslauf registrieren können und nach dem Signup diverse Infos erhalten. Bisher existiert jedoch noch gar keine Registrationsseite für die User. Dies muss behoben werden, damit der Zweck der Applikation auch erfüllt wird.

#### Weiteres Vorgehen

Signup erstellen. ergänzen

### Errorhandling Login

Momentan zeigt nach einem missglückten Login eine unformatierte Seite mit dem Wort «fail» an. Dies ist nicht sehr vorteilhaft, da der User keine Ahnung hat, was schiefgelaufen ist und so nicht weiss, was das Problem ist. Das Errorhandling sollte so umgesetzt werden, dass der User neben dem Loginformular eine klare Message erhält, warum das Login schiefgelaufen ist. So kann er direkt eine Neueingabe machen, die hoffentlich dann erfolgreich ist.

#### Weiteres Vorgehen

Abfrage verbessern, Message anzeigen

### Code Qualität

Die Qualität des Codes ist nicht wirklich sicherheitsrelevant. Dennoch ist es wichtig, einige Standards einzuhalten. Dies erleichtert nicht nur die Zusammenarbeit mit anderen Programmierern sondern sichern auch eine einfache Weiterentwicklung durch gut erhaltenen Code. Weiter können Fehler einfacher in einem Code gefunden werden, welcher strukturiert ist, als in einem Gewirr von diversen Codestyles und unnötigen Dingen wie nicht benötigte Klammern, etc.

#### Weiteres Vorgehen

Plugin installieren, welches hilft den Code zu überprüfen und entsprechende Änderungen vornehmen.

### Encoding

Je nachdem wie ein File codiert ist, können einige Zeichen von den Browsern nicht richtig interpretiert werden. Ein grosses Problem sind immer wieder die Umlaute «ä», «ö» und «ü», die mit der falschen Codierung nicht korrekt angezeigt werden. Deshalb ist es wichtig gerade bei deutschen und internationalen Applikationen Encoding-Standards zu benutzen, welche die verschiedensten Zeichen anzeigen können.

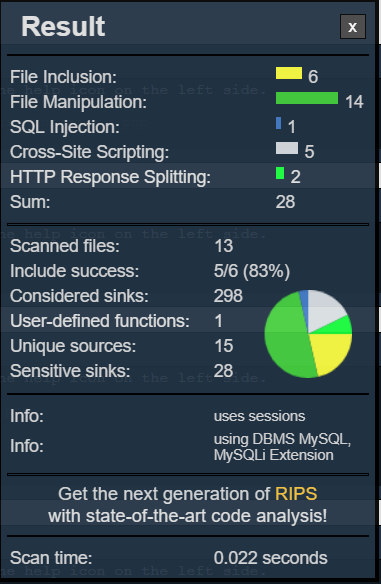
#### Weiteres Vorgehen

Die Applikation kann momentan mit den Umlauten nicht umgehen und zeigt deshalb falsche Zeichen in den Files «aufgaben.inc.php» und «eigenschaften.in.php» an. Die benannten Files, welche falsch encoded sind, müssen in UTF-8 konvertiert werden. Sind sie erst einmal in diesem Code-Format gespeichert, wird die Anzeige im Browser ohne Probleme möglich sein.

### Rechtschreibfehler

Auf einer Webseite muss die Rechtschreibung möglichst korrekt sein, denn oft geht es um einen Eindruck bei Kunden. Auch auf einigen Seiten der untersuchten Webapplikation haben sich Rechtschreibfehler eingeschlichen. Dies sollte behoben werden, jedoch ist es für die Sicherheit überhaupt nicht relevant und auch die Anzeige leidet nicht stark darunter. Deshalb verzichten wir vorerst auf diese Verbesserung.

# Bericht / Zusammenfassung

ist das überhaupt nötig?

🡪 hier klassifizierung vornehmen!!

- Datenbankanpassungen

# Resultat nach Bereinigung der Lücken

## Architektur

Ordnerstruktur beschreiben - Resources Ordner hinzugefügt um eine bessere Projektstruktur zu erreichen

### Datalayer

Neues Design einbinden

Die Datenbank sieht nun etwas anders aus. Wie in dem ERD zu erkennen ist, wurde die anfänglich existierende Logintabelle zusammen mit den Daten aus dem Textfile zu einer Usertabelle umgewandelt. Nach der Normalisierung sind ausserdem einige weitere Tabellen hinzugekommen, wie zum Beispiel die Gruppentabelle. Die Attribute sind einfachheitshalber alle auf «NOT NULL» eingestellt. Dies müsste in einem späteren Schritt nochmals überarbeitet werden.

Um die Sicherheit des Logins zu erhöhen, speichern wir ausserdem einen Salt ab, mit welchem das Passwort vor dem Hashen verschmolzen wird. Ausserdem ist kein Standardlogin mehr in dem Script vorhanden, welches von einem Angreifer benutzt werden könnte.

Weiter speichern wir ausserdem ab, ob der User Administratorrechte besitzt oder nicht. Standardmässig wird der User ohne Administratorrechte erstellt. Dies ist für die Logik der gesamten Webseite nötig (siehe Kapitel Einführung, Schlosslauf). Zusätzlich ist dies eine Erhöhung der Sicherheit, da so nur wenige Benutzer alle Daten einsehen können. Das reduziert das Risiko eines Angriffes enorm, da so spezifische Useraccounts gehackt werden müssen, um die Kontrolle über die gesamte Applikation zu erhalten.

### Businesslayer

Die Ordnerstruktur wurde erneuert, sodass man sich einfacher in der gesamten Umgebung zurechtfindet. Daraus wird wertvolle Zeit gewonnen – statt einer unendlich langen Suche in allen Files kann mehr Zeit in das Bereinigen von Schwachstellen investiert werden.

Neue Struktur einbinden

Der Single Point of Entry ist so umgesetzt, dass alle Zugriffe von nicht-eingeloggten Personen automatisch auf die Loginseite verweisen (ausgenommen der Registraturseite, diese muss natürlich ersichtlich bleiben).

### Presentationlayer

Im neuen Ordner «view» sind alle ehemaligen «.inc.php»-Dateien vorhanden. Nun sind die Webseiteninhalte viel schneller zu finden und somit einfacher wartbar als zuvor. Kostbare Zeit kann gespart werden. Sofort ist erkennbar, welche Dateien Anzeigelogik enthalten und welche nicht (HTML-Dateien).

Nochmals überprüfen und neue Files .html benennen, wenn nötig

Alle Formulare sind nun in eigenen Dateien zu finden, so auch das Anmeldeformular und das neue Signup-Formular. Die Logik dahinter ist im Ordner «controller» zu finden. Das Signup wurde so umgesetzt, wie es der Applikationszweck vorsieht.

## Sicherheitsrisiken

Nicht alle untersuchten Sicherheitslücken sind aufgetreten. Diejenigen, welche eine Änderungen erforderten sind hier nochmals mit dem jeweiligen Resultat aufgelistet.

### File Inclusion

Für den Include des Webseiteninhalts wurde eine Whitelist erstellt, so dass nur diejenigen Files eingebunden werden können, welche für die Webseite auch wirklich vorgesehen waren. Alle anderen Files werden ignoriert. Neue Files müssen jeweils zu der Whitelist hinzugefügt werden, sodass sie korrekt eingebunden werden.

Überprüfen: existiert whitelist?

### File Manipulation

Das Problem, dass ausführbarer Code in ein potentielles PHP-File geschrieben werden kann, ist erfolgreich eliminiert worden. Es besteht kein File mehr, welches dynamisch beschrieben wird, alles wird in die neue Datenbank abgespeichert. Bevor dies geschieht, wird der Input jeweils mittels der Funktion htmlspecialchars überprüft, sodass auch keine Codeteile in die Datenbank gelangen können.

Überprüfen: wurde txt file gelöscht?

### SQL Injection

Ein Fall von SQL Injection konnte bisher im Schlosslauf auftreten, das Feld «username» konnte entsprechend manipuliert werden. Da diese Lücke sich auf die Existenz unserer gesamten Datenbank auswirken könnte, haben wir hier natürlich sofort Initiative ergriffen. Alle Inputfelder werden nun mit der Funktion htmlspecialchars von allfälligen Codestücken bereinigt, sodass keine SQL-Ausführung mehr möglich ist.

Überprüfen: korrekte Methode?

### HTTP Response Splitting

Die PHP Version haben wir erneuert, die Whitelist haben wir jedoch noch nicht umgesetzt, da momentan kein Risiko besteht. So konnten wir uns akuteren Problemen widmen. Später würde dies natürlich noch umgesetzt werden.

Überprüfen: wirklich richtige Methode?

### Verschlüsselung

Um die Verschlüsselung der Logindaten zu gewährleisten haben wir die PHP-Funktion password\_hash verwendet. Dadurch kann der Passworthash – wenn überhaupt – nur schwer in Klartext gewandelt werden. Weiter wurde dem Passwort einen Salt- und einen Pepper-Wert hinzugefügt. Dies erschwert das Passwortknacken nochmals erheblich.

Überprüfen: Verschlüsselung da? Salt und Pepper auch hinzugefügt?

### Brute Forcing

Durch einen Counter für die Eingaben und einem Timestamp um die Session zu sperren wird das Brute Forcing nun entsprechend versperrt. Da die Daten der Applikation nicht sehr zeitrelevant sind, macht es nichts, wenn der User kurz ausgesperrt wird.

Überprüfen: ist dies wirklich umgesetzt?

### Schwaches Login

Das Login der Sekretärin wird nach der Testphase durch ein sichereres ersetzt. Die Zeichenüberprüfung bei der Registrierung wurde ebenfalls noch nicht umgesetzt, da andere Lücken bisher wichtiger waren. Die Überprüfung wird später noch eingefügt.

Das Datenbankscript ist nicht mehr innerhalb der Filestruktur zu finden und somit nicht downloadbar. So kann sich darin auch kein Login einschleichen und ein Angreifer kann sich nicht so einfach einen Überblick über die Datenbank beschaffen.

Überprüfen: DB Script aus dem source-folder entfernt?

### PHP

* Version Anpassung auf Version Nr
* Week Typing entfernt
  + 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.
* Errorreporting auf momentan höchste verfügbare Version gesetzt
* Errorhandling erweitert
* Cross-Site Request Forgery implementiert

#### 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

Überprüfen: Wurde das alles in der PHP section gemacht?

## Weitere Mängel

### Signup

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

Die Registration weist folgende Eigenschaften auf:

* Clientseitige Password Validierung. Hilft dem Benutzer Tippfehler 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.

Text Registratur aus Tatyanas Mail

### Errorhandling Login

beschreiben

### Code Qualität

In diesem Kapitel Text überarbeiten…

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

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

#### 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

### Encoding

Das Encoding-Problem mit den Umlauten sind wir durch die Umkonvertierung mittels Notepad++ relativ schnell losgeworden. Der Aufwand war sehr klein, die Auswirkung jedoch sehr gross – die Webseite mit richtigen Umlauten zu betrachten ist doch sehr viel entspannter als durch die speziellen Zeichen abgelenkt zu werden.

# Test

## Testsetup

## Testfallmatrix

# Quellen

|  |  |
| --- | --- |
| Name | Link |
| Rips | https://sourceforge.net/projects/rips-scanner/?source=typ\_redirect |
| PHP Risiken | https://www.owasp.org/index.php/PHP\_Security\_Cheat\_Sheet |
|  |  |
|  |  |
|  |  |

Reflexion

Ordnen nach OWASP oder wie es heisst!

Penetrationtests

OWASP Zap durchführen und protokollieren