# Einleitung

Für unser Projekt arbeiten wir an einem Bestehenden Projekt mit dem Namen Natework. Dies ist ein Framework, dass im Verlauf der Lehre entstanden ist und in Zukunft im Betrieb zum Einsatz kommen soll. Allerdings müssen einige Anforderungen erfüllt werden. Unter anderem muss die Sicherheit von Applikationen gesichert sein. Darum haben wir das Projekt nach OWASP Kriterien oprimiert.

# Security

## SQL Injection

*SQL Injektion ist leider immer noch eine Weitverbreitete Sicherheitslücke von Webseiten. Diese Lücke wird in den meisten Fällen durch fehlerhafte Programmierung ermöglicht. Mittels SQL Injektion kann es möglich sein sämtlich Benutzerdaten zu stehlen oder ganze Datenbanken zu löschen.*

Durch den integrierten Datenbank Controller können alle Datenbank Abfragen durch eine Klasse ausgeführt werden. Diese werden immer mit Prepared Statements ausgeführt. Dies soll verhindern, dass SQL code injiziert werden kann.

## Cross Site Scripting

*Cross Site Scripting ist wie SQL Injektion leider immer noch weitverbreitet und wird durch fehlerhafte, bzw. minimalistische Programmierung ermöglicht. Durch Cross Site Scripting können Benutzerdaten gestohlen werden, oder ganze Webseiten lahmgelegt werden.*

Cross Site Scripting zu verhindern ist nicht besonders schwierig. HTML Tags die in der Datenbank gespeichert werden, bzw. von Benutzereingaben kommen könnten, sollten immer als UNICODE String abgespeichert werden. Wird dieser Aspekt schon bei der Entwicklung beachtet, können sämtliche Sonderzeichen schon bei der Datenbank abfrage durch ihr UNICODE Zeichen ersetzt werden.

## Session Hijacking

*Session Hijacking zielt, wie der Name schon sagt, darauf ab die Sitzung eines Dritten zu stehlen. Dazu gibt es mehrere Möglichkeiten. Mittels Cross Site Scripting z.B. könnte ein Session Cookie gestohlen werden, wird nicht weiter überprüft, ob das Cookie auch zum richtigen Benutzer gehört, ist es möglich das Benutzerkonto eines anderen zu verwenden.*

Um Session Hijacking zu verhindern, muss Cross Site Scripting verhindert werden. Des weiteren ist es unerlässlich eine mittels SSL verschlüsselte Verbindung zu verwenden, dadurch werden „Man in the Middle“ Angriffe verhindert.

## Security Headers

*Security Headers sind neu in den OWASP TOP 10. Mit Security Headers kann das Verhalten von Webseiten eingestellt werden, z.B. kann verhindert werden, dass beim klick auf einen Link eine Referenz zur ursprünglichen Seite gesendet wird. Mit den Security Headers kann auch die Verwendung der Seite in einem iframe zu verwenden. Oder Cross Origin Policies können durchgesetzt werden.*

Securty headers zu implementieren ist kein grosser Aufwand. Es gibt mehrere Methoden dies zu erreichen. Entweder können die Header im Code, gesetzt werden. Die einfachere Lösung ist es in der „.htaccess“ Datei die entsprechenden Header anzugeben. Diese gelten dann global für die ganze Seite

## Passwort Sicherheit

Passwort Sicherheit zu implementieren, wird am besten schon ganz früh beachtet. Unterandrem sollte beachtet werden, dass Passwörter nicht in einer Reinform in die Datenbank geschrieben werden. Auch SQL Injektion stellt eine Gefahr für Passwortdiebstahl dar.

Die Passwörter werden mittels sha512 Hash Funktion unlesbar gemacht. Des Weiteren wird für jedes Passwort ein zufälliger Salt mit dem Passwort zusammen in die Hashfunktion mitgegeben. Zusätzlich wird für jede Applikation eine Pepper verwendet.

## Broken Access Controll

*Broken Access Controll ermöglicht es Benutzern Inhalte zu sehen, für die sie keine Berechtigung haben. Gerade in Applikationen mit mehreren Benutzerrollen (Kunde, Admin, Backoffice, etc.) kann Broken Access Controll ein grosses Problem Darstellen. Wird einem Kunden z.B. die gesamte Bestell- oder Kundenliste angezeigt, stellt dies ein grosses Datenschutzrisiko dar.*

Broken Access Controll zu verhindern ist auf Framework Seite leider nicht restlos möglich. In der Entwicklung muss stets beachtet werden, dass die Berechtigungen korrekt durchgesetzt werden. Dazu bietet das Framework Schlüsselwörter, welche dem Framework sagen was für welche Rolle angezeigt werden darf. Diese korrekt anzuwenden unterliegt den Entwicklern die das Framework verwenden.

## Vulnerable Components

*Vulnerable Components bezieht sich auf die Codebasis Selbst. Wird z.B. JQuery verwendet und eine Sicherheitslücke entdeckt, kann die von allen Ausgenutzt werden. Werden Sicherheitslücken in einem PHP-Framework gefunden kann dies sehr gefährlich werden. Das gilt auch für Code, der in öffentlichen Repositories zu finden ist.*

Vulnerable Components ist der Hauptgrund für die Entwicklung eines eigenen Frameworks. Durch den Privaten Code sind allfällige Sicherheitslücken nicht öffentlich bekannt. Auch das Suchen nach Lücken ist weniger interessant da die Verwendung sehr begrenzt ist.

# Accessability & Design

## MVC - View Parser

Model View Controller gehören heute zum guten Umgang. Für die View gibt es einen Parser. Dieser Parser ermöglicht es HTML wieder zu verwenden und strukturiert abzulegen. Damit das Funktioniert gibt es einige Schlüsselwörter die direkt in die HTML Datei geschrieben werden können. Diese werden vom Parser erfasst und dann die entsprechenden Funktionen ausgeführt. Schlüsselwörter können zum Beispiel folgende sein:

|  |  |
| --- | --- |
| {extends(„Datei“)} | Erweitert die Angegebene Datei mit dem Inhalt. |
| {section(„name“)} | Legt einen abschnitt fest, der immer wieder verwendet werden kann. So können mehrere Felder in einer Datei gruppiert werden. |
| {endsection} | Legt das ende eines Abschnitts fest. Für viele Schlüsselwörter gibt es ein „end“ Schlüsselwort. Dieses legt immer das Ende fest. |
| {auth} | Alles zwischen „{auth}“ und „{endauth}“ wird nur angezeigt, wenn der Benutzer angemeldet ist. |
| {role(„Rolle“)} | „{role(„“)}“ verhält sicht gleich wie „{auth}“, allerdings kann wird noch eine Rolle mitgegeben, z.B. Admin, so wird der Abschnitt nur Benutzern mit der Rolle Admin angezeigt. |
| {if(„bedingung“)} | Wird nur angezeigt, wenn die Bedingung erfüllt ist. Dazu gibt es auch noch „{elseif(„bedingung“)}“ und „{else}“. Der if- Block endet dann mit „{endif}“ |
| {for(„abbruch“)} | Mit der For schleife können Inhalte wiederholt angezeigt werden, bis die Abbruchbedingung erfüllt ist. Auch foreach und whlie Schleifen sind möglich. |
| {{ }} | Mit der doppelklammer „{{ $inhalt }}„ kann der Wert einer Variable ausgegeben werden. |

## Token basierte Authentifizierung

Eine Token basierte Authentifizierung ermöglicht gerade im B2B Bereich eine einheitliche Anmeldung. So können z.B. Admin Seiten von mehreren Applikationen und / oder Services mit denselben Anmeldedaten verwendet werden.

# Framework

Das neue Framework soll im Gegensatz zum alten Framework natürlich einige Verbesserungen bieten. Die Verbesserungen widerspiegeln sich hauptsächlich im View Parser wieder. Allerdings wird es auch möglich sein aus Model Dateien direkt ein SQL-Script zu generieren. Der MVC Aspekt im alten Framework hat sich gut bewährt und wird lediglich an die neue Umgebung angepasst und mit kleinen Optimierungen versehen. Folgend werden die wichtigsten Änderungen widerspiegelt.

## Alter View „Parser“

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungDer Alte View „Parser“ hat grundsätzlich gut funktioniert, allerdings war es nicht schön implementiert und die Fehlersuche war alles andere als einfach. Die Gründe liegen in der Implementierung. Denn eigentlich existiert kein richtiger Parser. Die Schlüsselwörter (beginnend mit @@) werden durch php code, oder den inhalt einer anderen Datei ersetzt.

Wurden alle Tags ersetzt wird der Zuvor zusammengesetzte Inhalt noch mit der «eval()» Funktion ausgeführt.

Tritt nun ein Fehler auf, wird die Zeile bzw. die Position des Fehlers im String angegeben, dass der String aber eigentlich aus mehreren Dateien besteht, macht die Fehlersuche schwierig. Dazu kommt noch, dass die «eval» Funktion in Sicherheitskreisen ein «no go» ist. Nach einigen Tests, entsteht zumindest in dieser Implementierung kein Sicherheitsrisiko. Möglicherweise schädlicher Code, wird erst während der Laufzeit der «eval» Funktion in die Datei geschrieben. Dem entsprechend wir der Code auch nicht ausgeführt. Allerdings dürfte es schwierig werden, die Security Teams der KastGroup zu überzeugen.

## Neuer View Parser

Ein Bild, das Text, Gedenktafel enthält.

Automatisch generierte BeschreibungDer Neue View Parser soll vor Allem durch eine verbesserte Fehlersuche bestechen. Der Parser soll auch mehr Befehle kennen. Dazu gibt es eine „.json“ Datei. Diese ist im Grunde eine Liste von Befehlen und dem „REGEX“ wert, mit dem der Befehlt in Dateien erkannt werden soll. Am Ende wird noch festgelegt, ob ein Schlusstag erwartet wird (true) oder nicht (false). Bei Schlusstags ist das Anfangstag angegeben.

Wird nun eine Seite geladen wird jede benötigte Datei einzeln indexiert. Statt, dass nun der gesamte Inhalt mit der „eval“ Funktion ausgeführt wird, werden nun die einzelnen Befehle gekapselt ausgeführt.

So viel zur Theorie. Einen Parser zu implementieren ist sehr zeitaufwändig und leider nicht in der kurzen Zeit zu realisieren. Es gibt auch viele Entscheide, die getroffen werden müssen. Diese sollten mit der KastGroup abgesprochen werden, damit das Tool auch für produktive Systeme verwendet werden kann.

## Weitere Änderungen

Das Framework soll das Bauen von APIs unterstützen. Dies war auch schon im alten Framework geplant. Wurde aber nie implementiert, da nie eine API nötig war.

Dateien, Klassen oder sonstige Ressourcen sollen erst geladen werden, wenn sie benötigt werden. Dies soll die Ladezeiten massiv verkürzen. Ladezeiten sind ein grosses Problem von Applikationen die in Zukunft mit dem neuen Framework realisiert werden sollen.

Mit der Hilfe von Caching sollen die Ladezeiten weiterverkürzt werden, die Indexierung und auch der Aufbau von kompletten Seiten kann mit der Hilfe die Caches gespeichert werden. Das spart Rechenleistung und kann die Performance positiv beeinflussen.