Lernziele

Nach dieser Lektion wissen Sie:



- Was eine SQL Injektion ist.
- Warum Passwörter verschlüsselt in einer DB gespeichert werden.
- Was eine Einweg Hash Funktion ist.
- Was es bedeutet, Passwörter zu salzen (Salt) und zu pfeffern (Pepper).
- Was ein BCrypt Hash ist und wie diese Funktion im groben Funktioniert.

Einführung

Wenn Sie sich bei einem Onlineshop registrieren, werden sie immer nach einem Benutzernamen und einem Passwort gefragt. Diese Passwörter werden danach zusammen mit ihrem Benutzernamen in der Datenbank gespeichert. Das gespeicherte Passwort wird meistens verschlüsselt abgespeichert.
Aber warum eigentlich? Auf die Datenbank hat ja niemand Zugriff? Oder doch?

Was ist SQL Injektion

Leider sind Webseiten, welche mit einer Datenbank verbunden sind oft fehleranfällig. Ein Entwickler muss besonders darauf achten, dass nicht durch eine sogenannte SQL-Injektion heikle Daten preisgegeben werden.
Aber was ganz genau ist eine SQL-Injektion?

- Eine SQL-Injection wird normalerweise mittels Eingabeformular einer Webseite ausgeführt.
- Der Angreifer braucht keine Kontrolle über den Server oder über die Datenbank.
- Mit SQL-Injection können beliebige Daten ausgelesen werden, welche die Webseite normalerweise nicht ausgeben würde.

Weil SQL-Injection einfach auszuführen sind, werden sie auch oft eingesetzt.

Auftrag 1: SQL Injektion

Starten Sie die Webseite http://192.168.220.11/webshop und probieren Sie diese SQL-Injektion aus:

Was macht der SQL Befehl „**UNION**“? Erinnern Sie sich noch an das Modul 100? Sonst Google Fragen!

Nun wissen wir: Passwörter im Klartext in einer Datenbank zu speichern ist keine Lösung. Was können wir dann sonst tun?

One-Way-Hash

Eine Hash-Funktion berechnet aus einer beliebigen Eingabe (z.B. Passwort oder Datei, Kontonummer) einen Wert mit einer vorgegebenen Länge. Aus dem Alltag kennen wir die Prüfziffern bei ISBN oder EAN Strichcodes sowie bei Bankkontonummern die IBAN.

Es gibt ganz unterschiedliche Verfahren, wie diese Prüfziffern generiert werden. Hier ein Beispiel für eine ISBN.



Abbildung 1: Quelle: http://de.wikipedia.org/

Aus diesem Beispiel ist ersichtlich, dass aus der gleichen ISBN jeweils die gleiche Prüfziffer entsteht, aber aus einer Prüfziffer kann nie die ganze ISBN hergeleitet werden. Dies nennt sich Einweg Funktion. Genau so funktionieren auch Berechnungen der Hash-Werte in der Kryptografie, die wir zur Speicherung der Passwörter verwenden. In der Praxis wird oft die MD5 Hash-funktion verwendet.

- Eine Hash-Funktion wird schon beim Design als **Einweg-Funktion** ausgelegt, darum kann aus dem berechneten Wert nicht mehr die ursprüngliche Eingabe ermittelt werden.
- Verschiedene Eingaben können zu demselben berechneten Wert (**Kollision**) führen. Gute Hash-Funktionen generieren wenige Kollisionen.
- Ähnliche Passwörter ergeben nicht ähnliche Hash Werte, so sind **keine Rückschlüsse über die Ähnlichkeit** zweier Passwörter möglich.

Beispiel: Der Satz: **„Mein Passwort ist unglaublich sicher da es lang ist“** mit diversen Hashalgorithmen.

Hash-Funktion	Hash-Wert
MD5	0c19df92eaefbb9842537109973ecdc2
RIPEMD128	6a4ba8a1ad547f8741d61bcfebf93bdf
SHA1	6900d9200398af7180839f72d71334b835c9184b
SHA256	e651c8cb0ea984e2a4c1579f9dbe553938727bc20a62addbb2ea442c0bec5415

Eigene Beispiele können Sie auf der Webseite http://192.168.220.11/webshop **unter Hash selber generieren.**

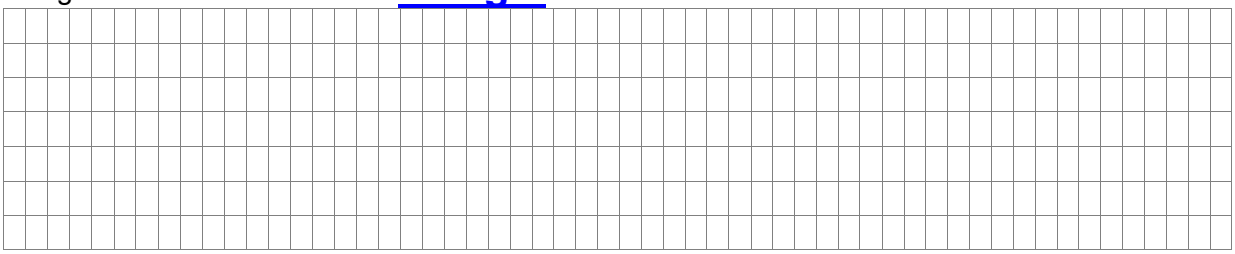
Nun haben wir eine Lösung für unser Problem. Statt das Passwort zu speichern, berechnen wir einfach dessen Hash-Wert und speichern ihn in der Datenbank. Wenn sich der Benutzer einloggen will, so berechnen wir wieder den Hash-Wert der Benutzereingabe, und vergleichen diesen mit dem Wert in der Datenbank.

Wir speichern nun zum Beispiel den Wert...

39b7ed33ff1d275ac2dc8f772e86c757

Da es ein Ein-Weg-Hash ist, können wir das ursprüngliche Passwort nicht herausfinden, oder weiss jemand das Passwort dazu?

Fragen Sie doch einfach mal **Google...**



Lookup-Tables

Lookup-Tables sind riesige Sammlungen vorberechneter Hash Werte. Im Beispiel vorhin, haben wir Google als Lookup-Table missbraucht, um das dazugehörende Passwort zu finden.

Ein Beispiel einer Lookup-Table könnte etwa wie folgt aussehen:

Passwort	MD5-Hash
...	...
schatzinsel	39b7ed33ff1d275ac2dc8f772e86c757
schatzkarte	032e49ff68095f86258109269de15e39
schatztruhe	e4ca741a4dc2f86271be4fe7b13e65d3
...	...

Stellen wir uns vor, wir erstellen eine Lookup-Table, die alle Wörter aus einem deutschen Duden (≈150'000), alle Namen aus einem Telefonbuch (≈5'000'000), und alle Zeichenkombinationen inklusive druckbare Sonderzeichen (70 Zeichen) bis zu 6 Stellen (70^6) = (117‘649‘000‘000) enthält. Wir erhalten eine Tabelle mit 117‘654‘150‘000 (117 Miliarden) vorberechneten Hash Werten. Heutzutage werden oft Grafikkarten (GPUs) zur Suche von Hashwerten eingesetzt, da diese extrem schnell parallele Berechnungen erledigen können. Eine handelsübliche GPU bewältigt etwa 8.5 Milliarden (8‘500‘000‘000) MD5 Werte pro Sekunde (Stand 2014, Quelle: http://hashcat.net/oclhashcat/#performance). Um unsere Beispiel Lookup-Table zu erstellen, braucht man also nur 13.8 Sekunden

Also kann dies auch nicht wirklich eine Lösung sein. Es sei denn, wir bringen Benutzer dazu längere Passwörter zu verwenden. Aber wie lang müssten diese sein?

Auftrag 2: Dauer bei 8 Stellen

Wie lange würde es gehen, wenn wir nach obigem Beispiel eine Lookup-Table mit Passwörter mit bis zu 8 Stellen erstellen wollen?

Solche Lookup-Tables sind im Internet unter dem Begriff Rainbowtables für die gängigsten Hashfunktionen zu finden. Rainbowtables nutzen im Gegensatz zu Lookup-Tables ein spezielles Verfahren um Speicherplatz zu sparen. Der Sinn ist aber der gleiche.

Salzen der Passwörter

Um sichere Passwörter zu haben, müssen diese also sehr lang sein. Es wird sehr schwer, Benutzer dazu zu bringen, so lange Passwörter zu verwenden. Darum benutzen wir einen Trick, der uns dies ermöglicht.
Dieser Trick nennt sich salzen von Passwörtern. Unter salzen versteht man das Hinzufügen einer zufällig gewählten Zeichenfolge zum Passwort, bevor man den Hash-Wert berechnet. Ein Salz (Salt) sollte mindestens 20 Zeichen lang sein.

$hash = MD5 ($passwort + "k32du7m01vVsQ2lB8g0s1");

So würde unser Passwort „Schatzinsel“ folgendermassen geändert.

Passwort	MD5-Hash
	...
schatzinsel	39b7ed33ff1d275ac2dc8f772e86c757
schatzinselk32du7m01vVsQ2lB8g0s1	e0ea9fe5d83aa79a1af13fe5bf94a877

Probieren Sie es auf der Webseite http://192.168.220.11/webshop unter Hash-> Hash Werte berechnen aus:
Können Sie es noch Googeln?

Am besten ist es, wenn das Salt beim ersten Mal, wenn sich der User registriert zufällig generiert wird.

- Ein Salt verhindert, dass der Hashwert einfach gegoogelt werden kann. Eine Lookup-Table müsste erst für genau dieses Salt erstellt werden.
- Jedes Passwort sollte sein **eigenes, möglichst einzigartiges** Salt bekommen. Dadurch muss ein Angreifer eine Lookup-Table pro Passwort erstellen, und kann nicht mit einer einzelnen Lookup-Table alle Passwörter ermitteln. Niemand erstellt eine Lookup-Table für ein einzelnes Passwort. Nach dem Finden einer Übereinstimmung macht das Weiterfahren keinen Sinn.
- Ein eigenes Salt pro Passwort bewirkt auch, dass Benutzer mit demselben Passwort unterschiedliche Hash-Werte erhalten.

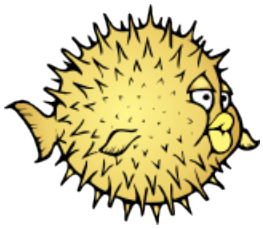
Natürlich muss nun das Salt ebenfalls in der Datenbank gespeichert werden, damit wir beim Login wieder dasselbe Salt zum Prüfen verwenden können. Es wird nur das erste Mal zufällig generiert. Danach muss immer wieder dasselbe Salt verwendet werden. Das Salt muss aber **nicht geheim** sein, es kann als Klartext gespeichert werden.

Bcrypt (Blowfish-Hash-Funktion)

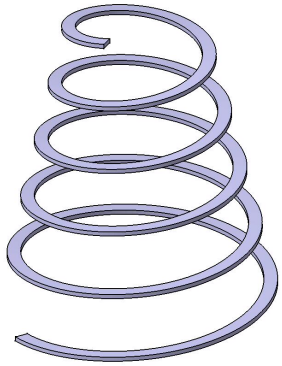
Das Salzen macht es definitiv schwieriger, an die Passwörter ranzukommen, da sie nicht mehr gegoogelt werden können. Es gibt keine vorgefertigten Lookup-Table. Wir müssten uns nun pro Salt jeweils eine eigene Lookup-Table erstellen. Aber es ist nicht unmöglich. Wir nehmen unser Lookup-Table aus dem ersten Beispiel und ergänzen die 117‘654‘150‘000 Einträge so, dass wir jeweils hinten noch das Salt anfügen.

Passwort	MD5-Hash
ak32du7m01vVsQ2lB8g0s1	a9b15157b49c8980f8700b217d281953fa6593e3
bk32du7m01vVsQ2lB8g0s1	9009d940fe45f07c382b79b7d9a8f7cb8346b243
ck32du7m01vVsQ2lB8g0s1	7dd93e6c47f3dfd94bb52e4fdb671d2e3fb967dd
Usw…	

Wenn nun diese Liste erstellt ist, braucht es wiederum ca. 13.8 Sekunden und wir haben das Passwort.



Aus genau diesem Grund wurde der **Bcrypt** Hash erfunden. Dieser stammt vom Verschlüsselungs-Algorithmus Blowfish (Kugelfisch) ab, ist aber eine echte Hash-Funktion. (Sprich nicht umkehrbar)
Bcrypt benötigt **viel Rechenzeit**, dadurch wird ein Brute-Force Angriff oder das Erstellen einer Lookup-Table unpraktikabel.



Die Bcrypt Funktion müssen sie sich so vorstellen, dass aus einem Passwort ein Hashwert generiert wird. Aus diesem Hashwert wird wiederum einen Hashwert generiert, aus diesem Hashwert wird wiederum…. So werden viele Runden durchlaufen und es wird dabei viel Rechenzeit in Anspruch genommen. Wie viele Runden durchlaufen werden, kann mit dem Wert „Kostenfaktor“ beeinflusst werden.

Durch erhöhen des Kostenfaktors, kann die benötigte Zeit für spätere und schnellere Computersysteme **angepasst** werden.

Zum Vergleich: Schon bei 1 ms Rechenzeit pro Hash-Wert, benötigt die Erstellung unserer Beispiel Lookup-Table bereits 3.7 Jahre statt 13.8 Sekunden.

Der Kostenfaktor (Load)



Der Kostenfaktor bestimmt, wie oft hintereinander die Bcrypt Funktion aufgerufen wird. Der Kostenfaktor wird mit Basis 2 potenziert, das heisst, wenn wir den Kostenfaktor um 1 erhöhen, so verdoppelt sich die Berechnungszeit.

```
$anzahlRunden = 2 ^ $kostenfaktor
```

Wenn eine Runde etwa 1 ms dauert, dann dauern 31 Runden ca. 74 Minuten. Genug Luft nach oben also. Brauchbare Werte liegen derzeit bei etwa 08 bis 12 Runden, je nach eingesetzter Hardware und Geduld.

Speichern in der Datenbank

Natürlich muss nun der Kostenfaktor ebenfalls, wie das Salt, in der Datenbank gespeichert werden. Dafür können wir aber bei jedem Login den Kostenfaktor anpassen, und einen neuen Hash Wert speichern. Das Format, in dem ein solches Passwort in einer Datenbank gespeichert wird ist folgendermassen genormt: Erst ein $ gefolgt von dem verwendeten Algorithmus, (2y steht für Bcrypt) ein $ gefolgt vom Kostenfaktor, ein $ gefolgt vom Salt und den Hashwert in einen 60 Zeichen langen String.

$2y$12$k32du7m01vVsQ2lB8g0s1.jdjGmlWQySpmZ0hxXugFjUZG.6J3yiC

Auftrag 3: Ausprobieren

Auf der Webseite http://192.168.220.11/webshop können Sie unter Passwörter -> Benutzer anlegen nun ein Login mit verschiedenen Verschlüsselungsarten anlegen. Probieren Sie es aus. Können Sie sich danach auch im Admin Bereich einloggen?

Verständnissfragen:

Frage: Warum können wir nur bei einem Login-Vorgang den Kostenfaktor oder das Salt ändern und nicht einfach nach Belieben?

Schützt das Salzen und der Blowfisch Algorithmus davor, wenn Benutzer schwache Passwörter (z.B.123456) gebrauchen?

Geben wir noch etwas Pfeffer dazu

Trotzdem ist es via SQL-Injektion immer noch möglich, an das Passwort zu kommen, da wir den Hashwert und das Salt auslesen können.

Aus diesem Grund kommt noch eine weitere Möglichkeit die sich pfeffern (pepper) nennt dazu.

Mit dem Pfeffern meint man das Kombinieren einer geheimen Zeichenfolge mit dem Passwort, bevor man den Hash-Wert berechnet.

```
$scharfesPasswort = $passwort + "p8empspher3ocm1yUwiq";
$hash = bcrypt($scharfesPasswort, $salt);
```

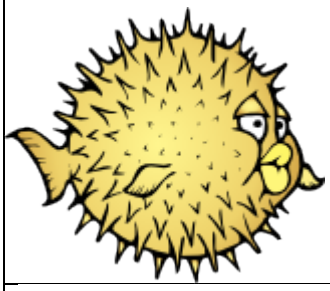
Der Pfeffer (Pepper) ist geheim und wird, im Gegensatz zum Salt, **nicht** in der Datenbank gespeichert. Stattdessen wird er an einem möglichst sicheren Ort hinterlegt, der gleiche Pepper gilt normalerweise für alle Passwörter.

Dadurch wird erreicht, dass sich das Passwort aus dem Hashwert und dem Salt aus der Datenbank nicht mehr rekonstruieren lässt. Denn der Pfeffer fehlt.

- Hat der Angreifer nur Zugriff auf die Datenbank (SQL-Injection / Backups), so kennt er zwar die Hash-Werte. Diese stammen aber nicht mehr von schwachen Passwörtern. Es sind Hash-Werte von langen Kombinationen von Passwort und einem starken Pepper. Kein Wörterbuch enthält je solche Passwörter, ein Wörterbuchangriff ist darum sinnlos.

- Hat der Angreifer Kontrolle über den Server, so kennt er auch den Pepper. Dieser bringt aber **keinerlei Vorteile** mehr. Rechte auf einem Server zu erlangen ist aber wesentlich schwieriger, als nur Lesezugriff auf die Daten zu erhalten.

Zum Schluss das ganze Menu mit allen Gängen

	Zum Speichern von Passwörtern, sollte eine Ein-Weg Hash-Funktion verwendet werden.
	Jedes Passwort wird mit einem eigenen Salt versehen, um den Einsatz von Rainbowtables unpraktikabel zu machen.
	Wir benützen einen langsamen, anpassbaren Hash-Algorithmus wie Bcrypt, um Brute-Force Angriffe auszubremsen.
	Durch Zugabe eines starken Pfeffers erreichen wir, dass sich das Passwort aus dem Hashwert und dem Salt aus der Datenbank nicht mehr rekonstruieren lässt. Denn der Pfeffer fehlt. Dieser wird nicht in der Datenbank sondern an einem sicheren Ort auf dem Server gespeichert.

Selbstkontrolle:



- ☐ Ich weiss, was eine SQL-injektion ist und kann dies in eigenen Worten erklären.
- ☐ Ich kann den Begriff One-Way-Hash in eigenen Worten erklären.
- ☐ Ich kann 3 One-Way-Hash Algorithmen benennen und kann diese mit Hilfe der Beispielwebseite anwenden.
- ☐ Ich weiss, was man unter Lookup-Tabels versteht.
- ☐ Ich kann das Salzen von Passwörtern in eigenen Worten erklären.
- ☐ Ich kann den Bcrypt Algorithmus in eigenen Worten Erklären.
- ☐ Ich weiss, warum Bcrypt auch für zukünftige Hardware noch sicher ist.
- ☐ Die Begriffe Salt (Salz), Load (Kostenfaktor), Pepper (Pfeffer) kann ich im Zusammenhang mit Bcrypt beschreiben.

Zusatz für Schnelle



- Auf der Webseite http://192.168.220.11/webshop -> Hash gibt es noch einen Punkt: „MD5 Wert einer Datei berechnen.“ Setzen Sie sich mit diesem Thema auseinander. Sie sollten danach den Begriff „Kollision“ erklären können.

- Beschäftigen Sie sich mit der Frage, was Rainbowtables genau sind.
  z.B. hier: http://www.zdnet.de/41544658/rainbow-tables-windows-passwoerter-nicht-mehr-sicher/

- Welche Hashalgorithmen gibt es? Welche gelten noch als sicher und welche nicht?

Quelle:
http://www.martinstoeckli.ch/hash/de/index.php