



KV Web Security SS 2011

durchgeführt am
Institut für Informationsverarbeitung und Mikroprozessortechnik (FIM)
Johannes Kepler Universität Linz



Leitung: Dr. Michael Sonntag

Insecure Cryptographic Storage - Using XOR encryption for password

Markus Jackstadt 911 1058025

20.06.2011

Inhaltsverzeichnis

1 Grundlagen der XOR-Verschlüsselung	3
2 Exploit einer fehlerhaften Webapplikation	4
3 Sichere Passwortverschlüsselung	6

1 Grundlagen der XOR-Verschlüsselung

Auf der untersten Ebene eines Computers bestehen Daten, seien es Werte oder Befehle, immer aus einer Folge von 0-en oder 1-en. Erst eine Interpretation macht daraus eine Anweisung, eine Ganzzahl, einen String etc. Die 0 und die 1 kann man aber auch als boolesche Werte falsch bzw. wahr interpretieren, auf die logische Operationen angewendet werden können. Eine solche logische Operation die zwei boolesche Werte zu einem neuen booleschen Wert verknüpft ist die Operation *XOR*. In diesem Abschnitt wollen wir uns nun anschauen, wie wir mit *XOR* verschlüsseln können.

Die Definition der logischen Operation *XOR* kann aus folgender Tabelle entnommen werden.

a	b	a XOR b
0	0	0
0	1	1
1	0	1
1	1	0

Zusammenfassend kann man also sagen, dass $a \text{ XOR } b$ immer genau dann wahr wird (den Wert 1 erhält) wenn sich die beiden Werte, also a und b , voneinander unterscheiden. Wie sieht nun eine Verschlüsselung mit dieser logischen Operation aus? Nehmen wir einfach eine Folge von 0-en und 1-en, die den *Originaltext* (eng. *Plaintext*) darstellen. Etwa die 8 Zeichen

01010101

Als *Schlüssel* (eng. *Key*) wählen wir nun eine gleichgroße Folge von 0-en und 1-en, etwa

11001100

Den *Geheimtext* (eng. *Ciphertext*) erhalten wir wenn wir dir Vorschrift $c_i = m_i \text{ XOR } k_i$ anwenden. Dabei bezeichnet m_i das i -te Zeichen im *Originaltext* und k_i das i -te Zeichen im *Schlüssel*. Sie liefern über *XOR* miteinander verknüpft c_i , das i -te Zeichen des *Geheimtextes*. In unserem Beispiel würden wir also den *Geheimtext*

10011001

erhalten. Interessant ist, dass man beim Verschlüsseln des so erhaltenen *Geheimtextes* mit der gleichen Methode und dem gleichen *Schlüssel* wieder den *Originaltext* erhält. Es gilt also

Originaltext XOR Schlüssel = Geheimtext

und draus folgend

Geheimtext XOR Schlüssel = Originaltext.

2 Exploit einer fehlerhaften Webapplikation

Im diesem Abschnitt wollen wir nun an Hand einer fehlerhaften Webapplikation zeigen, dass sich die XOR-Verschlüsselung **nicht** für eine sichere Passwortverwaltung eignet. Die einzelnen Schritte des Angriffes auf die Webapplikation werden dabei ausführlich erklärt.

Die Angreiferin Cindy findet auf dem Webserver eine geheime Passwortdatei. In dieser Datei werden sowohl die Benutzernamen als auch die dazugehörigen verschlüsselten Passwörter gespeichert. Gefunden werden kann diese Datei unter der URL `http://127.0.0.1:8080/A7_14/private/passwords.html?show=true`. (Es ist dabei zu beachten, dass hier der Port 8080 verwendet wird. Sollte das Aufrufen der angegebenen URL's Probleme bereiten dann verwenden Sie doch Bitte den Standardport 80.)

Hier befindet sich die streng geheime Passwort-Datei...

Benutzername	Verschlüsseltes Passwort
bob	VW[Q[
alice	@EVFAL

Wie in obiger Abbildung zu sehen ist, sind also die beiden Benutzer Alice und Bob bereits registriert. Die Angreiferin Cindy denkt sich nun, dass sie selbst auch in der Passwortdatei zu finden sein müsste, falls sie sich ebenfalls registrieren würde. Um sich ebenfalls zu registrieren surft Cindy auf die Startseite der Webapplikation. (zu finden unter `http://127.0.0.1:8080/A7_14/index.html`)

Willkommen! Bitte Loggen Sie sich ein.

Benutzername:
Passwort:

[Registrieren](#) [XOR-Tool](#)

Cindy klickt nun links unten auf „Registrieren“. Daraufhin gibt sie als Benutzernamen den Namen „cindy“ und als Passwort „yxcvbn“ an und klickt auf den Button „Registrieren“. (Es empfiehlt sich dabei, das hier vorgeschlagene Passwort zu verwenden!)

Hier können Sie sich registrieren!

Bitte geben Sie in den beiden Eingabefeldern ihren gewünschten Benutzernamen und Passwort an. Beachten Sie dabei, dass das Passwort **genau** 6 Stellen lang sein muss.



Benutzername: cindy
Passwort: yxcvbn

Klicken Sie [HIER](#) um zur Startseite zurückzukehren

Im folgenden wird Cindy nun mitgeteilt, dass sie sich erfolgreich registriert hat. Cindy betrachtet nun wieder die Passwortdatei (http://127.0.0.1:8080/A7_14/private/passwords.html?show=true) und stellt fest, dass ihr dort auch ein Eintrag gewidmet ist. (Gegebenenfalls muss die Seite neu geladen werden, damit Cindy auch tatsächlich in der Passwortdatei auftaucht!) Cindy notiert sich nun das verschlüsselte Passwort „@EVFAL“ von Alice.

Hier befindet sich die streng geheime Passwort-Datei...

Benutzername	Verschlüsseltes Passwort
bob	VW[Q[
cindy	HJPBWX
alice	@EVFAL

Nun versucht Cindy ihr verschlüsselte Passwort „HJPBWX“ mit ihrem gewählten Passwort „yxcvbn“ *XOR* zu verknüpfen. Glücklicherweise befindet sich ein *XOR*-Tool am Webserver mit dem man zwei Werte miteinander *XOR* verknüpfen kann. Gefunden werden kann es unter der URL http://127.0.0.1:8080/A7_14/xor.html. (Dieses Tool kann auch mit einem Klick auf „*XOR*-Tool“ auf der Startseite http://127.0.0.1:8080/A7_14/index.html erreicht werden.) Cindy gibt nun „HJPBWX“ und „yxcvbn“ ein und klickt auf „Berechne“.

Mit Hilfe dieses Tools können Sie zwei Werte miteinander XOR verknüpfen. Dabei ist zu beachten, dass die beiden Eingabewerte die gleiche Länge aufweisen müssen.



Wert 1: yxcvbn
Wert 2: HJPBWX

Klicken Sie [HIER](#) um zur Startseite zurückzukehren

Das Ergebnis dieser Berechnung lautet „123456“. Dabei handelt es sich um den geheimen Schlüssel mit dem alle Passwörter verschlüsselt werden. Verknüpft Cindy nun diesen geheimen Schlüssel mit dem verschlüsselten Passwort eines Benutzers *XOR* so erhält die

das Passwort des Benutzers. Also berechnet Cindy die *XOR*-Verknüpfung von „123456“ und „@EVFAL“. Bei letzterem Wert handelt es sich um das verschlüsselte Passwort von Alice.

Mit Hilfe dieses Tools können Sie zwei Werte miteinander XOR verknüpfen. Dabei ist zu beachten, dass die beiden Eingabewerte die gleiche Länge aufweisen müssen.



Klicken Sie [HIER](#) um zur Startseite zurückzukehren

Das Ergebnis dieser Berechnung lautet „qwertz“. Dabei handelt es sich also um das Passwort der Benutzerin Alice. Cindy kehrt daraufhin zur Startseite (http://127.0.0.1:8080/A7_14/index.html) zurück und loggt sich mit dem Benutzernamen „Alice“ und dem Passwort „qwertz“ erfolgreich als Benutzerin Alice ein.

3 Sichere Passwortverschlüsselung

In diesem Abschnitt wollen wir uns nun damit beschäftigen wie man Passwörter sicher abspeichern kann. Zum einen sollte es einem Angreifer nicht möglich sein Zugriff auf die verschlüsselten Passwörter zu erhalten. In unserer Beispielapplikation war dies für einen Angreifer recht einfach. In den meisten Fällen dürfte dieses Unterfangen jedoch eine größere Hürde darstellen. So könnte ein Angreifer beispielsweise mittels einer *SQL-Injection*¹ an die verschlüsselten Daten gelangen.

Zum anderen gilt es die Passwörter sicher zu verwalten. Um dieses Ziel zu erreichen werden in der Regel *kryptographische Hashfunktionen* eingesetzt. Dazu wird ein gesalzener kryptographischer Hashwert von dem eigentlichen Passwort berechnet. Durch die Eigenschaft der *Einwegfunktion*² der verwendeten Hashfunktion ist sichergestellt, dass eine direkte Wiederherstellung des ursprünglichen Passwortes aus der Kenntnis des Hashwertes nicht ohne erheblichen Aufwand möglich ist.

Sehen wir uns das obig beschriebene Verfahren nun im Detail an. In der Methode *addNewUser* wird zuerst ein zufälliger, *64 bit* langer *Salt* erstellt. Dabei ist zu beachten,

¹SQL-Injection bezeichnet das Ausnutzen einer Sicherheitslücke in Zusammenhang mit SQL-Datenbanken, die durch mangelnde Maskierung oder Überprüfung von Metazeichen in Benutzereingaben entsteht. Der Angreifer versucht dabei, über die Anwendung, die den Zugriff auf die Datenbank bereitstellt, eigene Datenbankbefehle einzuschleusen. Sein Ziel ist es, Daten auszuspähen, in seinem Sinne zu verändern, oder Kontrolle über den Server zu erhalten.

²Eine injektive Funktion $f: X \rightarrow Y$ erfüllt die Eigenschaften einer Einwegfunktion wenn für alle $x \in X$ der Funktionswert effektiv berechenbar ist und wenn es kein effizientes Verfahren gibt, um aus einem Bild $y = f(x)$ das Urbild x zu berechnen.

dass für jeden Benutzer ein unterschiedlicher Salt verwendet wird! Für die Berechnung des Hashwertes werden nun die folgenden drei Parameter benötigt:

- *ITERATION NUMBER* (= 1000): Repräsentiert die Anzahl der zu berechnenden Iterationen der Hashfunktion
- *password*: Das gewählte Passwort des Benutzers
- *bsalt*: Der zuvor erstellte Salt

Sowohl der berechnete Hashwert als auch der Salt werden daraufhin im *Base64*-Format kodiert und dann zusammen mit dem gewählten Benutzernamen abgespeichert.

```
//wird bei Programmstart initialisiert
private static HashMap<String, String[]> userMap;
private static final int ITERATION_NUMBER = 1000;

private static void addNewUser(String user, String password)
    throws NoSuchAlgorithmException, UnsupportedEncodingException {

    SecureRandom random = SecureRandom.getInstance("SHA1PRNG");
    byte[] bSalt = new byte[8];
    random.nextBytes(bSalt);
    byte[] bDigest = getHash(ITERATION_NUMBER, password, bSalt);

    String sSalt = new String(Base64Coder.encode(bSalt));
    String sDigest = new String(Base64Coder.encode(bDigest));

    String[] hashValues = new String[2];
    hashValues[0] = sSalt;
    hashValues[1] = sDigest;

    userMap.put(user, hashValues);
}
```

Im folgenden nehmen wir die Methode *getHash* genauer unter die Lupe. Hier wird einfach mit Hilfe der Hashfunktion *SHA512* ein gesalzener Hashwert berechnet. Die Berechnung des Hashwertes wird jedoch nicht einmal sondern *iterationNb*-mal durchgeführt. Der berechnete Hashwert hat also die Form `Hash(Hash(Hash(Hash(... Hash(password||salt))))))`.

```
private static byte[] getHash(int iterationNb, String password, byte[] salt)
    throws NoSuchAlgorithmException, UnsupportedEncodingException {

    MessageDigest digest = MessageDigest.getInstance("SHA-512");
    digest.reset();
    digest.update(salt);
```

```
byte[] input = digest.digest(password.getBytes("UTF-8"));

for (int i = 0; i < iterationNb; i++) {

    digest.reset();
    input = digest.digest(input);
}

return input;
}
```

Dieser Abschnitt beruht auf dem lesenswerten Artikel „Hashing Java“ des Open Web Application Security Projects (*OWASP*). Gefunden werden kann der Artikel unter der URL https://www.owasp.org/index.php/Hashing_Java.

Zu guter Letzt sollte jedoch noch unbedingt erwähnt werden, dass selbst die beste Passwortverwaltung nichts nützt wenn der Benutzername und vor allem das Passwort im Originaltext an die Webapplikation übertragen werden. Daher sollte unbedingt *SSL/TLS* verwendet werden!

Um die Sicherheit zusätzlich zu erhöhen sollte die Webapplikation beim Registrierungsvorgang schon überprüfen ob das vom Benutzer gewählte Passwort gewisse Anforderungen erfüllt. So sollte normalerweise jedes Passwort mindestens eine Länge von acht Zeichen besitzen, möglichst viele Zahlen und/oder Buchstaben enthalten und kein Wort sein, das in einem Wörterbuch nachschlagbar ist. Ferner sollte es kein Eigennamen oder gar der eigene Vor- oder Nachname sein. Zusätzlich sollte es ein Sonderzeichen enthalten und nicht aus einer Folge von Zeichen bestehen die auf der Tastatur unmittelbar nebeneinander liegen, wie zum Beispiel die Sequenz *qwertz*. Außerdem ist es empfehlenswert das Passwort in regelmäßigen Abständen zu verändern. Neben diesen Anforderungen an das Passwort sollte die Applikation nur eine geringe Anzahl von Fehlversuchen beim Login tolerieren (z.B. 3) und danach die Kennung sperren.

Aber auch der Benutzer selbst kann (bzw. muss!) zur Sicherheit beitragen. So sollte er nicht auf Nachfrage anderer Personen sein Passwort verraten und auch lieber auf einen vermeintlich gut versteckten „Spickzettel“, auf dem das Passwort niedergeschrieben wurde, verzichten.