TU München, Fakultät für Informatik
Lehrstuhl III: Datenbanksysteme
Prof. Alfons Kemper, Ph.D.

Übung zur Vorlesung
***Einsatz und Realisierung von Datenbanksystemen* im SoSe14**
Moritz Kaufmann (moritz.kaufmann@tum.de)
http://www-db.in.tum.de/teaching/ss14/impldb/

Blatt Nr. 3

Aufgabe 1

Skizzieren Sie die Funktionsweise von SSL. Erläutern Sie hierzu, wie der einfache TLS Handshake funktioniert. Eine Lösungsmöglichkeit wäre das Zeichnen eines passenden Message Sequence Charts.

Alles ohne Resumed Handshake:

- **Client:** ClientHello, höchste Version, Zufallszahl1, Ciphers

  D.h. der Client übermittelt seine Parameter für die Verbindung, beispielsweise, was er für Techniken unterstützt etc.

- **Server:** ServerHello, genutzte Version, Zufallszahl2, genutztes Cipher

  Hierbei sollte gelten:

  Cipher $\in$ Ciphers, genutzte Version=$max(VersionenClient, VersionenServer)$, genutztes Cipher=$max(CipherClient, CipherServer)$

  Der Server wählt hier also bereits die geltenden Parameter für die Verbindung aus. Ein gängiges Problem ist, dass der Server sich aufgrund der vom Client übermittelten Informationen nicht auf zu unsichere Protokolle festlegen darf und potentiell Verbindungen abweisen muss, um effektiven Schutz vor Angriffen zu bieten.

- **Server:** Certificate

  Im Allgemeinen übermittelt der Server ein Zertifikat an den Client, welches es dem Client erlaubt, die Identität des Servers zu verifizieren. Im Internet, d.h. bei https Verbindungen, dienen hierzu typischerweise Zertifikate, die von einer Autorität signiert wurden, der der Client vertraut. Diese sog. root Zertifikate werden vom Browser Hersteller mit dem Browser an den Client ausgeliefert und typischerweise nicht durch den Anwender geprüft.

- **Server:** ServerHelloDone

  Handshake beendet, d.h. alle Informationen für die Einleitung der Verschlüsselung sind ausgetauscht.

- **Client:** ClientKeyExchange, PreMasterSecret, evtl. Public Key

  PreMastersecret wird nun i.A. mit dem Public Key des Servers verschlüsselt. Hierdurch ist diese dem Client bekannt, kann vom Server entschlüsselt werden, jedoch niemand, der die Verbindung abhört kann es einfach rekonstruieren.

- **Client, Server** Berechnen jeweils basierend auf den bisher ausgetauschten Werten $Zufallszahl1$, $Zufallszahl2, PremasterSecret$ das $MasterSecret$ der für die Session benutzt wird

- **Client:** ChangeCipherSpec: Ab hier verschlüsselt, Finished: Verschlüsselte Nachricht mit Hashes über alles vorherige.

- **Server** ChangeCipherSpec: Ab hier verschlüsselt, Finished siehe oben.

- Ab jetzt wird die Verbindung von der Applikation verwendet.

Alles streng nach `http://en.wikipedia.org/wiki/Transport_Layer_Security`

Aufgabe 2

Erläutern Sie Probleme, die bei der naiven Implementierung von SSL auftreten können. Lesen Sie hierzu beispielsweise `http://www.ietf.org/mail-archive/web/tls/current/`

msg07553.html und skizzieren Sie sowohl den traditionellen Angriff mittels eines Botnets sowie den dort neu vorgestellten Angriff mittels Renegotiation. Wie können derartige Angriffe verhindert werden?

Viele DoS Attacken basieren auf der Tatsache, dass durch den Client eine Operation auf dem Server bewirkt werden kann, deren Bearbeitung den Server mehr kostet als es den Client kostet, die Operation auszulösen.
Im Falle eine DDoS Attacke, d.h. eines verteilten Angriffs ist dies nicht zwingend nötig. Vielmehr muss eine große Zahl von Clients zur Verfügung stehen, die eine kostspielige aber nicht notwendigerweise unproportional teure Operation auf dem Server auslösen. Dies lässt sich beispielsweise durch ein Botnetz[a] realisieren, welches eine Webseite gleichzeitig von tausenden verschiedenen Computern ausruft.
Bei SSL bietet es sich an, eine SSL Verbindung von jedem Rechner des Botnets aufzubauen. Hierdurch werden auf dem Server tausende Handshake-Berechnungen durchgeführt, auf jedem Client jedoch nur eine. Die Last auf dem Server steigt potentiell, bis dieser keine neuen Verbindungen mehr annehmen kann, wodurch eine Webseite effektiv nicht mehr erreichbar ist.
Die vorgeschlagene Renegotiation Attacke verzichtet auf den Multiplikator mittels Botnets indem hier ein Client den Server anweist, die Verschlüsselung neu auszuhandeln. Durch die auf Serverseite höheren Kosten dieser Operation[b] kann ein einzelner Client mittels Renegotiation-Request sehr hohe Kosten auf Serverseite verursachen und so das gleiche Resultat erzielen, wie beim zuvor genannten Angriff. Besonders effektiv ist dieser Angriff auch da er im Gegensatz zu einem klassischen DoS Angriff nicht so einfach auf der IP Ebene erkannt werden kann (z.B. über anzahl der neuen Verbindungen pro Sekunde), da die Requests über eine bestehende Verbindung gemacht werden können. Der Angriff muss entweder in der Serveranwendung direkt abgewehrt werden, z.B. in dem die Anzahl der Renegotiation Requests pro Sekunde limitiert werden, oder von einer Firewall die das SSL Protokoll versteht und Pakete die Renegotiation Requests enthalten erkennen kann.

[a] http://en.wikipedia.org/wiki/Botnet
[b] http://www.gossamer-threads.com/lists/apache/dev/398033?do=post_view_threaded#398033

Aufgabe 3

Implementieren Sie den RSA - beispielsweise in Java unter Nutzung der BigInteger Klassen. Verdeutlichen Sie sich die Funktionsweise des RSA an einem geeigneten Beispiel, d.h. verschlüsseln Sie eine kleine Nachricht mit einem (nicht zu großen) Schlüssel und bringen Sie das Beispiel inklusive Verschlüsselung und Entschlüsselung mit in die Übung.

Siehe Übungsbuch oder (sehr gut gemacht!) aus Wikipedia unter http://de.wikipedia.org/wiki/RSA-Kryptosystem.

Aufgabe 4

Eine statistische Datenbank ist eine Datenbank, die sensitive Einträge enthält, die aber nicht einzeln betrachtet werden dürfen, sondern nur über statistische Operationen. Legale Operationen sind beispielsweise Summe, Durchschnitt von Spalten und Anzahl der Tupel in einem Ergebnis (**count**, **sum**, **avg**, ...). Ein Beispiel wäre eine Volkszählungsdatenbank. Für diese Art von Systemen existiert das in der Einleitung erwähnte *Inferenzproblem*.

Nehmen wir an, Sie haben die Erlaubnis, im **select**-Teil einer Anfrage ausschließlich die Operationen **sum** und **count** zu verwenden. Weiterhin werden alle Anfragen, die nur ein

Tupel oder alle Tupel einer Relation betreffen, abgewiesen. Sie möchten nun das Gehalt eines bestimmten Professors herausfinden, von dem Sie wissen, dass sein Rang „C4“ ist und er den höchsten Verdienst aller C4-Professoren hat. Beschreiben Sie Ihre Vorgehensweise.

Siehe Übungsbuch.

Aufgabe 5

Wolfgang S.[1] hat sich eine Internetseite programmiert und dabei folgenden Pseudocode verwendet:

```
string id = GET_PARAMS['id'];

string query = "select␣title,content␣from␣news␣where␣";
query+=        "␣id='" + id + "'␣and␣visible='1';";
Process proc("sqlite3␣wolfgang.sqlite3");
proc.write(query);
string result = proc.read();

display(result);
```

Die Seite wird etwa wie folgt aufgerufen: `http://wolfgangs.de/news?id=15`. Der `GET`-Parameter `id` wird hierbei in eine Variable gespeichert und dann in das gezeigte SQL Statement eingefügt. Anschließend wird die so entstandene Anfrage an einen SQLite Client übergeben, der diese ausführt und das Resultat dem Benutzer angezeigt.

- Geben Sie den genauen Code an, wie Sie sich hier Zugriff auf die Tabelle “users” verschaffen können.
- Wie kann eine solche SQL Injection verhindert werden?
- Injection: `.../news?id='; select * from users; --` + urlencode
- Verhindern: Escapen oder Typcasten.

Aufgabe 6

Sie haben Wolfgangs Users-Tabelle ausgelesen, jedoch scheint sein Passwort uncharakteristisch kompliziert zu sein. Das von Ihnen erhaltene Resultat ist das Folgende:

id	name	password
1	wolfgang	4d75e8db6a4b6205d0a95854d634c27a

- Was könnte der Grund für dieses hexadezimale, 32 Stellen lange Passwort sein?
- Können Sie trotzdem den Klartext finden?
- Wie können Sie das Passwort sicherer Speichern?
- Wie können Sie für diese Art von Passwortspeicherung Bruteforce-Attacken erschweren?

[1] Die Wahl des Namens und der weitere Text stellen keinen politischen Kommentar der Übungsleitung dar. Vielmehr wird Bezug darauf genommen, dass die Webseite der betroffenen Person mindestens ein Mal auf diese Weise gehackt wurde.

- Das Passwort wurde MD5 gehasht.
- Google nach dem Hash, es gibt sog. Rainbow-Tables in denen zahlreiche MD5 Hashes vorberechnet sind.
- MD5 mehrfach anwenden, besser: Einen Salt verwenden, beispielsweise das Passwort zusammen mit dem Erstellungsdatum des Accounts oder dem Accountnamen hashen.
- MD5 10000 anwenden. Hierdurch muss der Client sehr viel rechnen, um ein Passwort auszuprobieren. Eine bessere Hashfunktion verwenden (MD5 und SHA1 werden nicht mehr empfohlen). Genauso wichtig, den User zwingen komplexere Passwörter zu benutzen damit Wörterbuchangriffe ineffizient werden.

Aufgabe 7

Nehmen Sie an, dass es in der Universitätswelt einige Fakultäten gibt, denen die Professoren zugeordnet sind. Lese- und Schreibrechte auf Vorlesungen sollen nun nach Fakultät vergeben werden, z.B. gibt es eine Benutzergruppe, die nur Vorlesungen der Fakultät für Physik ändern darf. Definieren Sie ein Schema mit Sichten so, dass die Benutzergruppen Änderungen durchführen können, aber die dritte Normalform der Relationen nicht verletzt wird.

Hinweise:

- Überlegen Sie, welches Subset der bekannten Universitätsdatenbank modelliert werden muss.
- Skizzieren Sie Ihre Modellierung, bestimmen Sie die vorliegenden FDs und weisen Sie nach, dass sich Ihre Modellierung in 3. Normalform befindet.
- Erstellen Sie eine passend beschränkte Sicht. Wann ist eine Sicht updatebar?
- Weisen Sie den Mitgliedern der Gruppe ‘PProfessoren“ Rechte auf die zuvor definierte Sicht zu.

Siehe Übungsbuch.