VSDB

Verschlüsselung

Protokoll

Alexander Rieppel Dominik Backhausen

27. Februar 2014

5AHITT

Inhaltsverzeichnis

1 Aufgabenstellung

Kommunikation [12Pkt] Programmieren Sie eine Kommunikationsschnittstelle zwischen zwei Programmen (Sockets; Übertragung von Strings). Implementieren Sie dabei eine unsichere (plainText) und eine sichere (secure-connection) Übertragung.

Bei der secure-connection sollen Sie eine hybride Übertragung nachbilden. D.h. generieren Sie auf einer Seite einen privaten sowie einen öffentlichen Schlüssel, die zur Sessionkey Generierung verwendet werden. Übertragen Sie den öffentlichen Schlüssel auf die andere Seite, wo ein gemeinsamer Schlüssel für eine synchrone Verschlüsselung erzeugt wird. Der gemeinsame Schlüssel wird mit dem öffentlichen Schlüssel verschlüsselt und übertragen. Die andere Seite kann mit Hilfe des privaten Schlüssels die Nachricht entschlüsseln und erhält den gemeinsamen Schlüssel.

Sniffer [4Pkt] Schreiben Sie ein Sniffer-Programm (Bsp. mithilfe der jpcap-Library http://jpcap.sourceforge.net oder jNetPcap-Library http://jnetpcap.com/), welches die plainText-Übertragung abfangen und in einer Datei speichern kann. Versuchen Sie mit diesem Sniffer ebenfalls die secure-connection anzuzeigen.

2 Designüberlegungen

2.1 Verschlüsselung

Um die Kommunikation zwischen Client und Server zu gewährleisten wollen wir Object-Streams verwenden. Da diese es Ermöglichen sollten den PublicKey möglichst ohne Veränderung oder Anpassungen zu senden. Da der Symmetrische Schlüssel allerdings mit dem PublicKey verschlüsselt werden soll werden wir dafür ein eigenes Objekt schrieben welche die Verschlüsselten Bytes des Schlüssels beinhalten soll.

Wir haben wollen den Symmetrischen Schlüssel auf dem Server generieren lassen, und damit wollen wir den PublicKey am Gegenstück also dem Client erzeugen. Nun haben wir uns folgendes Verbindungsaufbaukonzept überlegt: Wenn sich ein Client zum Server verbindet sendet dieser seinen PublicKey. Der Server empfangt diesen und schickt den damit verschlüsselten Symmetrischen Schlüssel zurück. Am ende dieses Aufbaus sollten beide den Symmetrischen Schlüssel besitzen und auch verwenden können.

UM nun auch nicht verschlüsselte Nachrichten senden zu können wollen wir vor der unverschlüsselten Nachricht einfach einen eindeutigen Text setzten um diese nachricht auch identifizieren zu können.

2.2 Sniffer

Angedacht ist die Library jnetpcap zu verwenden um den Sniffer zu implemetieren. Diese Klasse stellt alle notwendigen Werkzeuge zur Verfügung um mit Netzwerkpaketen zu arbeiten. Demnach sollte die Implementierung so ablaufen, dass man sich die entsprechenden Methoden die für die Implementierung notwendig sind einfach aufgerufen werden müssen. Weitere Libraries werde nicht von Nöten sein um den Sniffer zu implementieren. Die Ausgabe bzw. auch das später mitgeloggt File sollen alle notwendigen Informationen, zumindest aber die mitgesniffte Nachricht enthalten.

3 Arbeitsaufteilung

Name	Arbeitssegment	Time Estimated	Time Spent
Dominik Backhausen	Verschlüsselung	6h	8h
Alexander Rieppel	Sniffer	5h	4h
Gesamt		11h	12h

4 Arbeitsdurchführung

4.1 Verschlüsselung

Um die verschlüsselte Kommunikation zu implementieren haben wir uns zuerst mit den Asymmetrischen und Symmetrischen Verschlüsselungsmöglichkeiten die Java bereitstellt auseinander gesetzt. Nachdem wir die Verschlüsselungsmethoden die Java bietet implementiert und getestet haben, wurde eine Cleint-Server Verbindung erstellt. Nun haben wir uns überlegt wie wir den PublicKey und den Symetrischen Key am besten übertragen. Als auch diese Entscheidung getroffen wurde, haben wir noch ein Konzept erstellt welches es uns ermöglicht verschlüsselte und unverschlüsselte Nachrichten gleichzeitig zu senden und auch auseinander zu halten. Somit wurde noch ein kleiner Chat hinzugefügt und das Programm finalisiert.

4.2 Sniffer

Der Sniffer wurde so wie in der Designüberlegung angegeben implementiert. Jedoch wird zusätzlich zu Library jnetPcap auch die Library log4j verwendet, für das mitloggen des Netzwerk-Traces verwendet. Dies wurde so entschieden, da das Arbeiten mit log4j sehr komfortabel und einfach ist und somit auch das File entsprechend schön erstellt wird. Zusätzlich kann man bei der Ausführung als Filter eine Source- und Destination-IP angeben um nur Pakete mit einer bestimmten IP im Header zu erhalten. Wie ebenfalls angegeben war es für die Implementierung des Sniffers lediglich notwendig die entsprechenden Methoden der besagten Library jnetPcap aufzurufen und die Daten auszugeben. Ausgegeben werden eine Kopfzeile für den Start einer neuen Sniffing-Session, mit dem gewählten Filter, Source- und Destination-IP des Paketes und die entsprechende Payload des Paketes als hexadzimaler Dump mit Klartext daneben. Dieser Klartext stellt den Inhalt des Pakets dar.

5 Testbericht

5.1 Verschlüsselung

5.2 Sniffer

Um den Sniffer auszuführen gibt es mehrere Möglichkeiten, wobei folgende Argumente notwendig sind:

- <s|d|sd>
- <SourceIP>
- <DestinationIP>

Beim ersten Argument ist zu beachten, dass das 's' nur für einen Source-IP Filter, das 'd' nur für einen Destination-IP Filter und 'sd' für einen Source- und Destination-IP Filter. Demnach ist bei 's' nur eine SourceIP-Adresse, bei 'd' nur eine DestinationIP-Adresse und bei 'sd' beide Typen von Adressen anzugeben.