Labreport #6

Patrick Eickhoff, Alexander Timmermann

Aufgabe 1. Absicherung des TCP-Chats mit SSL

Um unseren ursprünglichen TCP-Chat mit SSL zu sichern, bedienen wir uns des *javax.net.ssl* Packetes. Dieses beinhaltet `SSLSockets` und `SSLServerSockets`, die wir benutzen um, eine sichere Verbindung aufzubauen. Die Sockets werden mittels `SSLSocketFactory` erstellt. Alle SSLSockets greifen können auf einen *keyStore* und einen *trustStore* zugreifen. Diese Java-spezifischen Datenbanken beinhalten Zertifikate und private Schlüssel für den Aufbau der SSL-Verbindung. Der *keyStore* beinhaltet einen privaten Schlüssel und die Zertifikate, die beim SSL-Handshake mitgeschickt werden. Im *trustStore* sind, dann die Zertifikate gespeichert, denen vertraut wird (trusted Certificate). Den *keyStore* des Clienten, haben wir mit *keytool* wie folgt initiiert:

```
$ keytool -keystore clienttest.keystore -keyalg RSA -genkey -alias client
```

Dieser Befehl erzeugt ein neues Schlüsselpaar mittels RSA, sowie ein Zertifikat mit dem Alias "client". Beim Erzeugen des Zertifikats fragt uns *keytool* nach den üblichen Parametern, wie z.B. Name, Länderkürzel, Organisation, etc. Für unsere Anwendung ist nur wichtig, dass der Client einen Namen in sein Zertifikat einträgt. Für den Server erstellen wir respektive einen *keyStore*:

```
$ keytool -keystore servertest.keystore -keyalg RSA -genkey -alias server
```

Beim SSL-Handshake werden die Zertifikate aus den Keystores dem Verbindungspartner präsentiert. Also präsentiert z.B. der Client sein selbstsigniertes Zertifikat aus *clienttest.keystore* dem Server. Der Server präsentiert wiederum sein Zertifikat dem Client. Die präsentierten Zertifikate werden dann gegen die Zertifikate im *trustStore* abgeglichen. Also müsssen wir die Zertifikate jeweils in die *trustStores* des Verbindungspartners importieren:

```
$ keytool -importkeystore -srckeystore servertest.keystore -destkeystore clienttest.truststore
$ keytool -importkeystore -srckeystore clienttest.keystore -destkeystore servertest.truststore
```

Jetzt akzeptieren Server und Client die präsentierten Zertifikate und eine sichere SSL-Verbindung wird aufgebaut.

Nun wollten wir unsere Nutzername-Password-Authentifikation durch die Zertifikatsauthentifikation der SSL-Verbindung ersetzen. Wie zuvor erwähnt, muss jedes Clienten-Zertifikat mindestens einen Namen enthalten. So können wir ein vertrautes Zertifikat genau einem Nutzer zuordnen. Der Anzeige-Name im Chat ist dann dem Namen im Zertifikat identisch.

Bei der vorherigen Implementation unseres Chats konnte man sich einfach mittels *netcat* verbinden. Deshalb war kein Java-Client von Nöten. Da *netcat* aber kein SSL-Protokol unterstützt, benötigt man zur Kommunikation nun einen Java-Chat-Clienten. Für die Implementation siehe Anhang A.

Aufgabe 2. CAs und Webserver-Zertifikate

2. Um ein selbstsigniertes Zertifikat zu erzeugen, sind im wesentlichen drei OpenSSL-Befehle notwendig:

```
$ openssl genrsa -out server.key 2048
$ openssl req -new -key server.key -out server.csr
$ openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt
```

Mit dem ersten Befehl erzeugen wir einen neuen, 2048-bit langen privaten Schlüssel. Wir erstellen dann mit dem zweiten Befehl die sog. *certificate signing request*, in der die Informationen enthalten sind, die das Zertifikat später umfassen soll. Diese werden interaktiv abgefragt. Als Common Name verwenden wir `vmsrv11.svslab`.

Im dritten Befehl verwenden wir schließlich den erstellten Private Key, um aus der CSR ein Zertifikat zu generieren, das 365 Tage gültig ist.

Um dieses Zertifikat für den Apache-Server einzusetzen, ersetzen wir in der Konfigurationsdatei die Pfade hinter `SSLCertificateFile` und `SSLCertificateKeyFile` mit den Pfaden zum Zertifikat und dem private key.

Dass selbst-signierte Zertifikate als unsicher angesehen werden liegt auf der Hand. Jeder kann sich ein Zertifikat erstellen, was bspw. für google.com gültig ist. Würde diesen Zertifikaten vertraut, ließe sich eine Man-in-the-Middle-Attacke zu google.com sehr leicht implementieren, da der User keinen Unterschied bemerkt, und ihm durch das Zertifikat weitere Sicherheit suggeriert wird.

Die Aufgabe einer CA ist es, diese Angaben zu validieren und im Zweifelsfall die Ausstellung eines Zertifikats zu verweigern. CAs sind deshalb Firmen oder Organisationen, denen vertraut wird, diese Macht nicht zu missbrauchen.

3. Um mit Hilfe von *mod_rewrite* eine Weiterleitung zu HTTPS zu realisieren, fügen wir folgende Optionen in die Apache-Config ein:[1]

```
1 RewriteEngine On
2 RewriteCond %{HTTPS} !=on
3 RewriteRule ^/?(.*) https://%{SERVER_NAME}/$1 [R,L]
```

Mit dem ersten Befehl wird die Bearbeitung der Rewrite-Regeln eingeschaltet. Danach wird geprüft, ob nicht bereits HTTPS verwendet wird, da damit die Weiterleitung hinfällig wird. Schließlich wird die Rewrite-Regel definiert, die dann weiterleitet. Die Optionen am Ende sind die Flags. `L` gibt an, dass die Regel die letzte ist, die ausgeführt wird. `R` spezifiziert, dass ein HTTP Redirect passieren soll.

Die Regel selbst ist ein regulärer Ausdruck, bei dem alles nach einem optionalen "/" matched und in die HTTPS-URL eingesetzt wird.

4. Über die verschlüsselte Verbindung können wir die Zugangsdaten selbstverständlich nicht mitlesen. Verwenden wir *sslstrip* als Proxy, so wird die Umleitung auf die verschlüsselte Verbindung verhindert und wir können die Zugangsdaten recht komfortabel im Log-File nachlesen.

[1] Quelle: `https://wiki.apache.org/httpd/RewriteHTTPToHTTPS`

Die Sicherheit der Weiterleitung ist damit maximal als "gut gemeinte" Maßnahme für Endanwender gedacht, taugt aber nicht, ernsthafte Angriffe abzuwehren. Hierzu sind weitere Maßnahmen nötig, wie beispielsweise HSTS.

HTTP Strict Transport Security (HSTS) ist ein in RFC 6797[2] beschriebener Standard, der einen solchen Angriff verhindern kann. Wird HSTS aktiviert, so können Seiten unter einer Domain, die HSTS setzt, nicht über eine unverschlüsselte Verbindung abgerufen weden. HSTS wird über einen Header aktiviert, der bei der Antwort mitgesendet wird und etwa so aussieht:

```
1 Strict-Transport-Security: max-age=15768000
```

Damit wird HSTS im Browser strikt forciert und lässt sich auch für den Endnutzer nicht wieder abschalten. Mit dem `max-age`-Parameter wird eine Art "Haltbarkeitsdatum" in Sekunden angegeben. 15768000 Sekunden entsprechen etwa einem halben Jahr, danach wird der Browser eine unverschlüsselte Verbindung prinzipiell wieder erlauben, wenn nicht erneut ein HSTS-Header mitgesendet wird.

Aufgabe 3. Unsichere selbstentwickelte Verschlüsselungsalgorithmen

1. Das Problem mit der BaziCrypt-Verschlüsselung ist, das die Paddingbytes (`0x00`) durch die XOR-Operation den Schlüssel direkt in den Ciphertext abbilden. Durch einfache Analyse der sich wiederholenden Bytes am Ende des Ciphertextes und ein paar Versuchen lässt sich sehr leicht der Schlüssel rekonstruieren. Mithilfe unseres Skripts B haben wir folgende Nachrichten entschlüsselt:

n01.txt.enc Hallo Peter. Endlich koennen wir geheim kommunizieren! Bis bald, Max

n02.txt.enc Hi Max! Super, Sicherheitsbewusstsein ist ja extrem wichtig! Schoene Gruesse, Peter.

n03.txt.enc Hi Peter, hast du einen Geheimtipp fuer ein gutes Buch fuer mich? Gruss, Max

2/3. Das PKCS7-Padding-Verfahren füllt die zu verschlüsselnde Nachricht nicht mehr mti Nullbytes, sondern mit Bytes die äquivalent zur Länge des Paddings sind.

Beispiel: Padding der Länge 6: `0x06 0x06 0x06 0x06 0x06 0x06`

Durch diese Methode lässt sich der Schlüssel nicht mehr einfach aus dem Ciphertext ablesen. PKCS7 unterstützt jedoch nur Blöcke bis zu 256-Byte und 256 mögliche Paddingbytes lassen sich in kurzer Zeit per Bruteforce durchprobieren. Unser Skript (B) probiert einfach nur alle Paddingbytes von 0 bis 100 durch (100 da die Dateien nur 100 Byte groß sind) und versucht die Nachricht zu entschlüsseln. Dann suchen wir einfach die Entschlüsselung aus, die korrekt ist. Mithilfe dieses Skripts haben wir folgende Nachrichten entschlüsselt:

n04.txt.enc Hi Max, natuerlich: Kryptologie von A. Beutelspacher ist super. Gruss Peter

n05.txt.enc Hi Peter, worum geht es in dem Buch? Ciao, Max.

n06.txt.enc Hi Max, das ist ein super Buch, das viele Krypto-Themen abdeckt. Gruss Peter

[2] `https://tools.ietf.org/html/rfc6797`

Aufgabe 4. EasyAES

Anhang A TCP-Chat

Listing 1: ChatServer.java

```
1  //SourceCode partially from:
2  // http://www.dreamincode.net/forums/topic/259777-a-simple-chat-program-with-clientserver-gui-optional/
3  // http://www.oracle.com/technetwork/java/socket-140484.html
4  // http://www.java2s.com/Tutorial/Java/0490__Security/SSLServerSession.htm
5  package tcp_chat;
6
7  import java.io.IOException;
8  import java.util.LinkedList;
9  import java.util.List;
10
11 import javax.net.ssl.SSLServerSocketFactory;
12 import javax.net.ssl.SSLSocket;
13 import javax.net.ssl.SSLServerSocket;
14
15 import passwd_save_java.Useradmin;
16
17 public class ChatServer {
18
19         public static void main(String[] args) {
20                 System.setProperty("javax.net.ssl.trustStore", "/home/patrick/workspace2/project_netsec/src/tcp_chat/se
21                 System.setProperty("javax.net.ssl.keyStore", "/home/patrick/workspace2/project_netsec/src/tcp_chat/serv
22                 System.setProperty("javax.trustStorePassword", "123456");
23                 System.setProperty("javax.net.ssl.keyStorePassword", "123456");
24
25                 ChatServer chatserver = new ChatServer();
26                 chatserver.listenSocket();
27         }
28
29         private SSLServerSocketFactory sslf;
30         private SSLServerSocket server;
31         private List<ClientThread> connections;
32         private List<String> loggedInUsers;
33         private Useradmin useradmin;
34
35         public ChatServer() {
36                 this.sslf = (SSLServerSocketFactory) SSLServerSocketFactory.getDefault();
37                 this.connections = new LinkedList<ClientThread>();
38                 this.useradmin = new Useradmin();
39                 this.loggedInUsers = new LinkedList<String>();
40         }
41
42         public void listenSocket() {
```

```
43              try {
44                      server = (SSLServerSocket) sslf.createServerSocket(4444);
45                      server.setNeedClientAuth(true);
46                      //server.setEnableSessionCreation(true);
47              } catch (IOException e) {
48                      System.out.println("Could␣not␣listen␣on␣port␣4444");
49                      System.exit(-1);
50              }
51              while (true) {
52                      try {
53                              ClientThread t = new ClientThread((SSLSocket)server.accept(),this);
54                              System.out.println("new␣Connection");
55                              t.start();
56                              connections.add(t);
57                      } catch (IOException e) {
58                              System.out.println("Accept␣failed:␣4444");
59                      }
60              }
61      }
62
63      protected void finalize() {
64              try {
65                      server.close();
66              } catch (IOException e) {
67                      System.out.println("Could␣not␣close␣socket");
68                      System.exit(-1);
69              }
70      }
71
72      public void notify(String message, int ID, String User) {
73              for (ClientThread t : connections) {
74                      if (t.ID != ID) {
75                              t.print(User + ":" + message);
76                      }
77              }
78      }
79
80      public void disconnect(int ID) {
81              for (ClientThread t : connections) {
82                      if (t.ID == ID) {
83                              loggedInUsers.remove(t.getUser());
84                              connections.remove(t);
85                              System.out.println("disconnected");
86                      }
87              }
88      }
89
90      public boolean authenticate(String username, char[] password) {
```

```
91                  if (loggedInUsers.contains(username)) {
92                          return false;
93                  }
94                  if (useradmin.checkUser(username, password)) {
95                          loggedInUsers.add(username);
96                          return true;
97                  }
98                  return false;
99          }
100 }
```

Listing 2: ChatClient.java

```
1  /*
2          SourceCode partially from: https://stackoverflow.com/questions/7872846/how-to-read-from-standard-input-no
3  */
4  package tcp_chat;
5
6  import java.io.BufferedReader;
7  import java.io.InputStreamReader;
8  import java.io.PrintWriter;
9  import java.util.concurrent.BlockingQueue;
10 import java.util.concurrent.LinkedBlockingQueue;
11 import java.util.concurrent.TimeUnit;
12
13 import javax.net.SocketFactory;
14 import javax.net.ssl.SSLSocketFactory;
15 import javax.net.ssl.SSLSocket;
16
17 public class ChatClient {
18
19          private static final BlockingQueue<String> socketlines = new LinkedBlockingQueue<String>();
20          private static final BlockingQueue<String> systemlines = new LinkedBlockingQueue<String>();
21
22    public static void main(String[] args) throws Exception {
23            BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
24            System.setProperty("javax.net.ssl.trustStore", "/home/patrick/workspace2/project_netsec/src/tcp_chat/clienttest.
25            System.setProperty("javax.net.ssl.keyStore", "/home/patrick/workspace2/project_netsec/src/tcp_chat/clienttest.k
26            System.setProperty("javax.trustStorePassword", "123456");
27            System.setProperty("javax.net.ssl.keyStorePassword", "123456");
28
29      SocketFactory sf = SSLSocketFactory.getDefault();
30      SSLSocket s = (SSLSocket) sf.createSocket("localhost", 4444);
31
32      BufferedReader sin = new BufferedReader(new InputStreamReader(s.getInputStream()));
33      PrintWriter sout = new PrintWriter(s.getOutputStream(),true);
34
35      Thread socketReaderThread = null;
36      socketReaderThread = new Thread(new Runnable() {
```

```
37          @Override
38                  public void run() {
39                          try {
40                                  while (!Thread.interrupted()) {
41                                          String line = sin.readLine();
42                                          socketlines.add(line);
43                                  }
44                          } catch (Exception e) {
45                                  // TODO: handle exception
46                          }
47                  }
48      });
49      socketReaderThread.setDaemon(true);
50      socketReaderThread.start();
51
52      Thread systemReaderThread = null;
53      systemReaderThread = new Thread(new Runnable() {
54          @Override
55                  public void run() {
56                          try {
57                                  while (!Thread.interrupted()) {
58                                          String line = in.readLine();
59                                          systemlines.add(line);
60                                  }
61                          } catch (Exception e) {
62                                  // TODO: handle exception
63                          }
64                  }
65      });
66      systemReaderThread.setDaemon(true);
67      systemReaderThread.start();
68
69      String line;
70      while (!s.isClosed()) {
71          if ((line = socketlines.poll(10L, TimeUnit.MILLISECONDS))!= null){
72                  System.out.println(line);
73          }
74          if ((line = systemlines.poll(10L, TimeUnit.MILLISECONDS))!= null){
75                  sout.println(line);
76                  if (line.equals("quit")) {
77                          s.close();
78                  }
79          }
80      }
81      socketReaderThread.interrupt();
82      systemReaderThread.interrupt();
83    }
84  }
```

Listing 3: ClientThread.java

```
1  //SourceCode partially from: http://www.dreamincode.net/forums/topic/259777-a-simple-chat-program-with-clientserv
2  //      http://www.oracle.com/technetwork/java/socket-140484.html
3
4  package tcp_chat;
5
6  import java.io.BufferedReader;
7  import java.io.IOException;
8  import java.io.InputStreamReader;
9  import java.io.PrintWriter;
10 import java.security.cert.Certificate;
11 import java.security.cert.X509Certificate;
12 import java.util.regex.Matcher;
13 import java.util.regex.Pattern;
14
15 import javax.net.ssl.SSLHandshakeException;
16 import javax.net.ssl.SSLSocket;
17
18 class ClientThread extends Thread implements Runnable {
19         private SSLSocket client;
20         private ChatServer observer;
21         public final int ID;
22         private String User;
23
24         ClientThread(SSLSocket client, ChatServer observer) {
25                 this.client = client;
26                 ID = this.hashCode();
27                 this.observer = observer;
28                 User = "";
29         }
30
31         public void run() {
32                 String line;
33                 BufferedReader in = null;
34                 try {
35                         PrintWriter out = new PrintWriter(client.getOutputStream(), true);
36                         in = new BufferedReader(new InputStreamReader(
37                                         client.getInputStream()));
38                         Certificate[] certs = client.getSession().getPeerCertificates();
39                         for (Certificate cert : certs) {
40                                 X509Certificate xcert = (X509Certificate)cert;
41                                 String cert_name = xcert.getSubjectDN().getName();
42                                 System.out.println(cert_name);
43
44                                 Pattern name = Pattern.compile("CN=((?:[\\pL\\p{Nd}_]{1,20}\\s*)*),.*");
45
46                                 Matcher mname = name.matcher(cert_name);
47                                 if (mname.matches()) {
```

```
48                                      User=mname.group(1);
49                              }
50                      }
51              }
52              catch (SSLHandshakeException e) {
53                      System.err.println("SSLHandshake failed");
54                      this.interrupt();
55              }
56               catch (IOException e) {
57                      System.err.println("in or out failed");
58                      this.interrupt();
59              }
60
61              while (!this.interrupted()) {
62                      try {
63                              line = in.readLine();
64                              if (line.equals("quit")) {
65                                      observer.disconnect(ID);
66                                      client.close();
67                                      client = null;
68                                      return;
69                              }
70                              observer.notify(line, ID, User);
71                      } catch (IOException e) {
72                              System.err.println("Read failed");
73                              this.interrupt();
74                      }
75              }
76      }
77
78      public synchronized void print(String message) {
79              try {
80                      PrintWriter out = null;
81                      out = new PrintWriter(client.getOutputStream(), true);
82                      out.println(message);
83              } catch (IOException e) {
84                      System.err.println("out failed");
85              }
86      }
87
88      public String getUser() {
89              return User;
90      }
91 }
```

Anhang B Decrypt

Listing 4: bazi_decrypt.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# First Argument: File-Destination
# Second Argument: Tolerance Level

import sys


def translateMessage(key, msg):
    translated = [] # stores the encrypted/decrypted message string

    keyIndex = 0
    for c in msg: # loop through each character in message
        num = ord(c)
        num = num ^ ord(key[keyIndex]) # add if encrypting
        translated.append(chr(num))
        keyIndex += 1 # move to the next letter in the key
        if keyIndex == len(key):
            keyIndex = 0
    return ''.join(translated)

key = []
message = []
tolerance = int(sys.argv[2])

f = open(sys.argv[1], "rb")
byte = f.read(1)
while byte != "":
    message.append(byte)
    byte = f.read(1)
print(message)
print(len(message))

i = 1
while not key[-tolerance:] == message[len(message)-len(key)-tolerance:len(message)-len(key)]:
    key.insert(0,message[-i])
    i+=1
    print(key)

f.close()
decrypted = translateMessage(key, message)
print(decrypted)
```

Listing 5: advazi_decrypt.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# First Argument: File-Destination
```

```
# Second Argument: Tolerance-Level

import sys

def translateMessage(key, msg):
    translated = []

    keyIndex = 0
    for c in msg:
        num = ord(c)
        num = num ^ ord(key[keyIndex])
        translated.append(chr(num))
        keyIndex += 1
        if keyIndex == len(key):
            keyIndex = 0
    return ''.join(translated)

message = []
tolerance = int(sys.argv[2]) #The Amount of Chars that have to loop in the message to accept as key

f = open(sys.argv[1], "rb")
byte = f.read(1)
while byte != "":
    message.append(byte)
    byte = f.read(1)
print(message)
print(len(message))

for padding_length in range(100):
    i = 1
    key = []
    while not key[-tolerance:] == message[len(message)-len(key)-tolerance:len(message)-len(key)]:
        key.insert(0,message[-i])
        i+=1
    key = map(lambda x: chr(padding_length ^ ord(x)), key)
    f.close()
    decrypted = translateMessage(key, message)
    words = decrypted.split("␣")
    print(decrypted)
```