

# SVS Bachelor-Projekt Network Security

## Blatt 3: Datenkommunikation

Louis Kobras  
6658699

Utz Pöhlmann  
6663579

### 1 HTTP

#### 1.1 [local]

Funktioniert so nicht, da die Website über das HTTPS-Protokoll läuft, welches SSL erfordert, welches wiederum nicht von telnet unterstützt wird. Alternativ kann OpenSSL verwendet werden.

### 2 SMTP (Mail Spoofing)

#### 2.1 [local]

```
# zwei Sätze zum Topic
# Input Protokoll
# Kommentar erste Mail vs. echte Mail
# Modifikation bei zweiter Mail
```

### 3 License Server (DNS-Spoofing)

#### 3.1 [local]

Protokoll:

1. Key als User-Input
2. Übermitteln des Keys an den Server
3. Rückgabe vom Server, ob Key gültig oder nicht (`SERIAL_VALID=0` bzw. `SERIAL_VALID=1`)
- 4a Wenn gültig, Dank für Kauf
- 4b Wenn nicht gültig, FBI ist unterwegs

#### 3.2

- Verhindern der Kommunikation der Software mit dem echten Auth-Server
- Geschehen durch Erweitern des Hosts um `127.0.0.1 license-server.svslab` in `/etc/hosts`
- Herunterladen der Java-Klasse `TCPCClient.java`<sup>1</sup>
- Manipulieren des Servers: `ServerSocket` auf svslab-Port (1337) gesetzt
- Manipulieren des Servers: Rückgabe des Servers auf statisch `“SERIAL_VALID=1”` gesetzt
- $\implies$  alle Keys gültig, unabhängig von Eingabe

---

<sup>1</sup><https://systembash.com/a-simple-java-tcp-server-and-tcp-client/>

### 3.3

Es gibt zwei anmerkbare Mängel.

1. Es sollte nicht angegeben werden, ob die Serial-Länge korrekt ist.
2. Es könnte mithilfe einer eindeutigen Signatur o.Ä. eine Abfrage an den Server eingebunden werden, ob er "echt" ist (gehasht).

## 4 License Server (Brute-Force-Angriff)

### 4.1 [local]

Das Programm funktioniert an sich, wenn man aber an den Server sendet, kriegt man (scheinbar nach Zufall) entweder "invalid command" oder "invalid length" zurück, bei Eingabe von `serial=abcdefgh` ( $a, b, c, d, e, f, g, h \in \{0, 1, \dots, 9\}$ ).

Wir baten zwei Gruppen neben uns um Hilfe, jedoch konnten diese uns auch nicht weiterhelfen bzw. haben keinen Fehler in unserem Programm gefunden.

Als Ausgangspunkt wurde die Java-Klasse `TCPClient.java` von [todo:link] genommen.

Einige gültige Keys:

- |            |            |            |            |
|------------|------------|------------|------------|
| • 03133700 | • 18802200 | • 47005500 | • 87743600 |
| • 06264700 | • 21935900 | • 59540300 | • 90877300 |
| • 09401100 | • 25069600 | • 62674000 | • 94011000 |
| • 15668500 | • 31337000 | • 72075100 | • 97144700 |

### 4.2

Möglichkeiten, sich zu verteidigen, enthalten, sind jedoch nicht beschränkt auf:

- Sperren des Absenders der Auth-Anfrage nach  $n$  Fehlversuchen (Unterbrechen von Brute-Force-Attacken)<sup>2</sup>
- Prüfung der IP bzw. Prüfsumme, ob Empfänger und Absender korrekt sind (Zurechenbarkeit)
- Limitieren der Eingabe auf  $k$  pro Minute (Verlangsamen von Brute-Force-Attacken)

### 4.3 [local]

Alle gefundenen gültigen Schlüssel sind durch 100 teilbar, liegen also in der Form `xxxxxx00` vor. Ist ein Schlüssel außerdem durch 1000 teilbar, also in der Form `xxxxxx000`, so ist ebenso ein Schlüssel der Form `0xxxxxx00` gültig, der die gleiche Ziffernfolge anstelle der  $x$  enthält.

## 5 Implementieren eines TCP-Chats

### 5.1 [local]

Es werden zwei URLs in Fragmenten gesendet. Zusammengesetzt sehen sie folgendermaßen aus:

- [URL1] `http://www.oracle.com/technetwork/java/socket-140484.html`
- [URL2] `http://code.google.com/p/example-of-servlet/source/browse/trunk/src/javaLanguage/basic/?r=56#basic%2Fsocket`

---

<sup>2</sup>Je nach Art der Sperrung ist dies lediglich eine Bremse; wird z.B. nur die IP gesperrt, kann diese resettet werden, um wieder Zugang zu erlangen.

**5.2**

**5.3**

**5.4**

**5.5**