# Labreport #3

Patrick Eickhoff, Alexander Timmermann

## 1 HTTP

### 1.1

Mit dem Befehl `telnet www.uni-hamburg.de 80` öffnen wir eine Verbindung zu der angegeben Adresse über Port 80 (Http Port). Mit der offenen Verbindung erwartet der Host nun unsere Request. Nach der üblichen Form für HTTP-Anfragen, fragen wir die *home.html* an:

```
GET /de/inst/ab/svs/home.html HTTP/1.1
Host:www.inf.uni-hamburg.de
```

Da wir eine HTTP/1.1 Anfrage stellen, müssen wir den Host angegeben, da HTTP/1.1 multiple Domains erlaubt.

Als Antwort erhalten wir jedoch, dass die gesuchte Seite verschoben wurde und nun unter *https://www.inf.uni-hamburg.de/de/inst/ab/svs/home.html* zu finden ist. Da `telnet` jedoch keine SSL-Verbindungen unterstützt, müssen wir per OpenSSL die HTML anfragen: `openssl s_client -connect www.inf.uni-hamburg.de:443` (Port 443 für ssl)

```
GET /de/isnt/ab/svs/home.html
Host: www.inf.uni-hamburg.de
```

Im Kopf der HTML können wir sehen, dass */assets/application-11e3b49e605ff8ba1f01d275bd36850edfdfc1fbbb8c22e55fae1baf643a00d0.css* der Stylesheet ist, den wir suchen. Da SSL jedoch eine sichere Verbindung ist, haben wir kaum Zeit unsere nächste Anfrage zu stellen:

```
GET /assets/application-11e3b49e605ff8ba1f01d275bd36850edfdfc1fbb
b8c22e55fae1baf643a00d0.css
Host:www.inf.uni-hamburg.de
```

## 2 SMTP(Mail-Spoofing)

### 2.1

Mittels `netcat mailhost.informatik.uni-hamburg.de 25` verbinden wir uns mit dem SMTP-Server des Informatikums.

```
HELO mailhost.informatik.uni-hamburg.de
MAIL FROM:<123Mustermann@informatik.uni-hamburg.de>
RCPT TO:<123opfer@informatik.uni-hamburg.de>
DATA
From: <123Mustermann@informatik.uni-hamburg.de>
To: <123opfer@informatik.uni-hamburg.de>
Date: Mon, 10 Apr 2016 10:00:00 -0400
Subject: Prank

Its just a prank.

.

QUIT
```

Wenn man nun den Quelltext unserer Fake-Mail und einer normalen Mail vergleicht sieht man einige Unterschiede:
Zum einem ist die Fake-Mail nicht im MIME-Format, wie normalerweise üblich. Sehr gut lässt sich auch erkennen, dass Nachrichten von authentifizierten Nutzern des RRZ auch als solche im Quelltext sichtbar sind: `(Authenticated sender: 123Mustermann)`. Dies sind nur einige der Unterschiede zwischen einer echten und unserer gefälschten Mail.

# 3 DNS-Spoofing

## 3.1

Nach einiger Interaktion mit dem Lizensserver, fällt uns auf, dass keine Authentifikation zwischen Klient und Server gefordert wird. Ausserdem ist die Bestätigung eienr Lizenz vom Server zum Klienten nur der String `SERIAL_VALID=1`. Dies lässt sich leicht fälschen, wenn wir einfach unseren eigenen Server mittels DNS-Spoofing als Lizenzserver ausgeben.

## 3.2

Um den Lizenclient auszutricksen, müssen wir zuerst sicherstellen, dass er sich mit unserem eigenem Server anstatt dem Lizenzserver verbindet. Obwohl die *LicenseClient.class* nicht einfach auslesbar ist, können wir mittels `strings LicenseClient.class` herausfinden, dass der Klient immer mit der selben Hostadresse *licenseserver* verbindet. Nun müssen wir nur noch in der *hosts*-Datei folgenden Eintrag hinzufügen: *127.0.1.2 licenseserver*, wobei 127.0.1.2 die IP-Adresse ist, auf der wir unseren eigenen Server laufen lassen.

Unseren Server haben wir in *Ruby* geschrieben
(Source:http://www.tutorialspoint.com/ruby/ruby_socket_programming.htm, sh. Appendix A). Wenn der Server angesprochen wird, tut dieser nichts anderes, als irgendeine Eingabe vom Klienten zu nehmen und mit `SERIAL_VALID=1` zu antworten.

### 3.3

# 4 License-Server(bruteforce-Angriff)

### 4.1

Den Bruteforce-Angriff haben wir als Python-Skript geschrieben (sh. Appendix B) basierend auf dem Passwort-Bruteforce. Als problematisch erwies sich jedoch, dass der LicenseServer bei zu vielen Anfragen bzw. Versuchen die Verbindung geschlossen und einen Neuaufbau für ein gewisses Zeitfenster abgelehnt hat. Aus diesem Grund haben wir sobald keine Antwort mehr vom Server zurückkommt, eine Wartezeit von 10s eingeführt, bis wir wieder eine Verbindung zum Server aufbauen. Dies funktioniert zwar, jedoch treibt es auch die Berechnungszeit enorm in die Höhe.

### 4.2

Eine Möglichkeit für den Betreiber sich gegen Bruteforceangriffe zu schützen, ist einen größeren Zeitabstand zwischen Anfragen zu fordern. Auf diese Weise würde ein Bruteforce-Angriff einen enormen Zeitaufwand haben. Eine weitere Möglichkeit ist, bei hoher Anzahl subsequenter Anfragen von der selben IP-Adresse, diese zu sperren (Blacklisting).