Übungsblatt 5 – Dynamisches Routing & Application Layer

Viele Protokolle im Internet, insbesondere die älteren, sind Textbasiert, d.h. es werden einzelne lesbare Befehle zwischen den Rechnern hin- und her geschickt. Entsprechende Server können Sie demzufolge auch einfach „per Hand“ als Client bedienen, um deren Funktion zu testen bzw. auf deren Dienste zuzugreifen. Dazu reichen beispielsweise Programme wie *telnet* völlig aus.
Andererseits können Sie aber auch einfach Skripte mit beliebige Befehlsfolgen zusammenstellen und automatisiert an den Server senden. Sei es zum automatisieren auf der Kommandozeile oder als Fuzzy-Test mit beliebigen Datenmüll um zu prüfen, wie Fehlertolerant Ihr eigener Server ist.

Aufgabe A – Routing & Traceroute

Nachdem Sie recherchiert haben, wie *traceroute* arbeitet, welche Kritik an Traceroute geäußert wurde und wie diese mit dem Tool Paris-Traceroute abgestellt wurden, sollen beide Tools hier kurz erprobt werden.

1.) Überlegen Sie sich zunächst anhand Ihrer Recherche was *traceroute* in etwa ausgeben müsste, wenn Sie im Labor eine Route von einem Rechner A zu einem Rechner B verfolgen würden. Wobei beide Rechner zu unterschiedlichen Netzwerken gehören (d.h. unterschiedlichen Tischreihen).

2.) Nutzen Sie anschließend *traceroute* um sich die Router zwischen zwei Laborrechnern anzeigen zu lassen. Stimmen Ihre theoretische Überlegungen mit denen von *traceroute* überein? Falls nicht, sollten Sie analysieren woran dies liegen könnte.

3.) Vergleichen Sie die Ausgaben von *traceroute* und *paris-traceroute* für folgende IP-Adressen:

1.) 41.231.21.44

2.) 91.198.174.192

3.) 37.220.21.130

4.) 80.239.142.229

Hinweis: Für *paris-traceroute* sollten Sie den „exhaustive algorithm“ Nutzen (`-na exhaustive`)

4.) Analysieren Sie anschließend die Ausgabe beider Tools.

5.) Warum wurde Ihnen eine Liste von IP-Adressen genannt anstelle von Domainnamen? Nennen Sie mindestens zwei Gründe!

Aufgabe B – Domain Name System (DNS)

1.)

a.) Fragen Sie mit jedem der vier Tools auf der Kommandozeile jeweils einmal einen Hostnamen (bspw. `www.htw-berlin.de`), einen Domainnamen (htw-berlin.de) und eine IP-Adresse (141.45.5.100) ab.

b.) Schauen Sie sich die Ausgabe von *dig* bei der Abfrage der IP-Adresse genauer an – dort werden Sie in der „Question Section“ sehen, das nach dem A-Resource-Record mit dem Namen 141.45.5.100 gefragt wurde. Wenn Sie den Namen zu dieser IP-Adresse suchen – welchen Resource-Record müssen Sie dann anstelle des A-Records erfragen?

c.) In welcher Form müssen Sie dann die IP-Adresse angeben?
(Test mit dig -t <record-type> <richtiges-format-ip-adresse>).

d.) Denken Sie sich einen Domainnamen aus, den es wahrscheinlich geben könnte, aber den noch niemand aus dem Netzwerk der HTW-Berlin innerhalb der letzten Stunden angefragt hat (z.B. `www.uriminzokkiri.com` oder `www.northkoreatech.org`).

e.) Erfragen Sie diesen Namen zweimal kurz hintereinander via *dig* und vergleichen Sie die beiden Ausgaben. Worin unterscheiden sich beide Einträge? Begründen Sie diese Unterschiede!

f.) Erfragen Sie mit *host, dig und nslookup* den zuständigen Mail-Server für die Domain `htw-berlin.de`.

g.) Erzwingen Sie mit *host, dig* und *nslookup*, das die Namensauflösung nicht mit dem Standard-Nameserver des Betriebssystems, sondern mit einem öffentlichen Nameserver (bspw.: 9.9.9.9) erfolgt. Testen Sie am Besten zuerst mit *dig* oder *nslookup*, da diese Ihnen immer sagen, welche Nameserver sie genutzt haben. *host* liefert diese Information nur, wenn Sie explizit eigene Server angefordert haben.

2.) DNS-Resolver: Das Listing zeigt die „resolv.conf“eines Servers.

```
1 # Dynamic resolv.conf(5) file for glibc resolver(3) generated by resolvconf(8)
2 # DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN
3 nameserver 141.45.3.100
4 search f4.htw-berlin.de
```

Was bedeuten die Einträge mit den Schlüsselwörtern: „nameserver“ und „search“?

Aufgabe C – HTTP(S) & HTML

In der vorigen Übung haben Sie bereits einen Webserver aufgesetzt. Im wesentlichen sollten Sie verstanden haben, was die Aufgabe des Webservers ist – das ausliefern von Dateien über einen Socket.

- Auf jedem Raspberry Pi ist ein Apache Webserver vorinstalliert. Konfigurieren Sie diesen erneut, sodass Ihr Raspberry Pi die default Seite auf seiner Laboradresse ausliefert.
- Rufen Sie die Webseite eines anderen Raspberry im Browser auf (mit HTTP ohne TLS-Verschlüsselung).
- Zeichnen Sie diesen Aufruf parallel mit Wireshark auf und finden Sie heraus, welche *HTTP*-Befehle der Browser an den Server zum Abruf der HTML-Seite gesendet hat. Welcher Port wurde dazu verwendet?
- Verbinden Sie sich nun mit dem Kommandozeilen-Programm *telnet* mit dem selben Server und Port.
- Nachdem die Verbindung hergestellt wurde, tippen Sie die Befehle ein, die auch durch den Browser gesendet wurden. Können Sie beobachten, dass Sie als Antwort die Startseite des Webservers erhalten?
- Welche der vom Browser gesendeten Befehle müssen Sie mindestens eingeben, um die Webseite zu sehen?
- Wenn Sie zu einem Server eine Verbindung aufbauen, wird serverseitig ein Timeout gestartet, so das, wenn nicht innerhalb einer gewissen Zeitspanne eine Anfrage kommt, der Server die Verbindung beendet. Wenn Sie etwas umfangreichere Befehle an den Server senden müssen, oder das Ganze ohne manuellem Eintippen automatisieren wollen, können Sie das Tool *netcat* nutzen.

  a.) Schreiben Sie dieselben HTTP-Befehle zum Abruf der Webseite jetzt in eine lokale Textdatei (alle Zeilenumbrüche beachten!).

  b.) Lassen Sie sich den Inhalt der Datei auf der Kommandozeile nach Std-Out ausgeben (bspw.: durch *cat* oder *less*).

  c.) Leiten Sie diese Ausgabe mittels Pipe als Eingabe in den Befehl *netcat* um. Rufen Sie *netcat* dabei mit Parametern so auf, dass es eine Verbindung wieder zum gleichen Webserver und Port wie bisher aufbaut.
  Wenn Sie alles richtig gemacht haben, sehen Sie wieder die Startseite des Webservers.

- Damit bei Klartext-Protokollen keine Nutzerdaten durch Fremde mitgelesen werden können, werden von vielen Diensten die eigentlich originalen Protokolle in eine TLS-Verbindung verpackt, um die Daten für die Anwendung transparente zu verschlüsseln.
  Ein Programm um beliebige Verbindungen nachträglich mit SSL/TLS zu versehen ist Teil des *OpenSSL*-Toolkits. Mit dem Befehl *openssl s_server* können Sie Serveranwendungen, welche kein TLS unterstützen aber über Std-In Befehle entgegennehmen, darüber absichern. Mit dem Befehl `openssl s_client` wiederum können Sie Client-Verbindungen mit TLS-Unterstützung aufbauen oder auch manuell ausführen.

  a.) Zeichnen Sie alle Abrufe der Webseite mit mit Wireshark auf und prüfen Sie, was sie dort sehen können.

  b.) Bauen Sie noch einmal testweise eine HTTP-Verbindung mit *telnet* oder *netcat* zum Webserver des Rechenzentrums der HTW (`www.rz.htw-berlin.de`) auf und fragen Sie die Startseite an.

  Nutzen Sie nun anstelle von *telnet* das Programm `openssl s_client` um eine Verbindung zum gleichen Webserver. Dieses mal jedoch auf dem *HTTPS* Port. (Welcher Port wird für HTTPS genutzt?) Rufen Sie nach erfolgreichem Verbindungsaufbau wieder die Startseite ab.

  c.) Welche Informationen über den TLS-gesicherten Server bekommen Sie mit `openssl s_client`? Wo sehen Sie folgende Einträge?
  Gültigkeit des Zertifikates?
  Den Zeitraum der Gültigkeit?
  Wer hat das Zertifikat ausgestellt?

Aufgabe D – E-Mail mit POP3, IMAPv4 & SMTP

Zum Abruf von E-Mails gibt es die beiden Protokolle *POP3* und *IMAPv4*.

1.) Bauen Sie nun mit `openssl s_client` eine gesicherte Verbindung zum einem Ihrer Mail-Server auf. (bspw.: `mail.rz.htw-berlin.de`)
Loggen Sie sich auf Ihrem Account ein, um dann Ihre Mails abzurufen.
Achtung – bitte loggen Sie sich nicht ohne TLS aus dem Labor heraus auf einem Mailserver ein. Andere Studenten werden sicherlich parallel Wireshark laufen lassen und könnten dann Ihre Zugangsdaten sehen!

2.) Bauen Sie mit `openssl s_client` eine Verbindung zum POP3-SSL Port auf und loggen Sie sich mit Ihren Nutzerdaten ein. Anschließend rufen Sie erst die Liste aller Nachrichten und dann eine spezielle Nachricht ab, um sie zu lesen. (Eine beispielhafte POP3-Session mit den notwendigen Befehlen finden Sie leicht im Netz oder z.B. bei Wikipedia).

3.) Setzen Sie das Gleich mit *IMAP* um. **Hinweis**: Alle Mail-Protokolle unterstützen auch das *STARTTLS* Kommando. Damit kann eine nicht gesicherte Verbindung nachträglich noch mit TLS abgesichert werden. Sie bauen also im Klartext z.B. zum POP3-Server mit Standard-Port eine Verbindung auf und senden dann im Klartext das Kommando `STARTTLS`. Daraufhin wird auf diesem Port eine verschlüsselte Verbindung aufgebaut und alle nachfolgenden Befehle können nicht mehr von anderen mitgelesen werden.

4.) Starten Sie `openssl s_client` und mit dem Parameter `starttls` eine gesicherte Verbindung zum POP3-Standard-Port. Versuchen Sie sich dann mit falschen Login-Daten anzumelden. Beenden Sie die Verbindung.

5.) Zeichnen Sie den Verbindungsaufbau parallel mit Wireshark auf und prüfen Sie, was sie davon sehen können. **Hinweis**: Sollten Sie kein E-Mail-Programm griffbereit haben, können Sie das natürlich auch das per Hand erledigen. Das SMTP-Protokoll ist ebenfalls relativ einfach und text-basiert.

6.) Bauen Sie mit `openssl s_client` nacheinander eine Verbindung zu allen drei SMTP-Ports auf. Finden Sie heraus, welche Ports direkt mit SSL gesichert sind und welche Ports mit *STARTTLS* nachträglich gesichert werden müssen.

7.) Loggen Sie sich nun auf dem SSL-Port mit ppenSSL ein und versenden Sie eine E-Mail.
Hinweis: Viele Server nutzen inzwischen beim Versand zur Spambekämpfung *SMTP-AUTH* (SMTP-Authentication) um nur eigenen Nutzern zu erlauben Mails an fremde Server zu versenden.
An eigene E-Mail-Adressen des SMTP-Server können Sie aber immer senden (d.h. wenn Sie mit dem SMTP-Server der HTW verbunden sind, können sie immer eine E-Mail an eine Empfängeradresse „s0XXXXXX@htw-berlin.de“ senden. Wollen Sie eine E-Mail an z.B. „...@posteo.de“ senden, müssen Sie sich vorher authentifizieren.