HOCHSCHULE KONSTANZ TECHNIK, WIRTSCHAFT UND GESTALTUNG (HTWG)
Fakultät Informatik
Rechner- und Kommunikationsnetze
Prof. Dr. Dirk Staehle

Vorlesung Rechnernetze

Laborübung

Einstieg in die Socketprogrammierung

Prof. Dr. Dirk Staehle

Die Abgabe erfolgt durch Hochladen der bearbeiteten Word-Datei in Moodle.

Bearbeitung in Zweier-Teams

Team-Mitglied 1:

Team-Mitglied 2:

1 Einleitung

In dieser Übung lernen Sie die Programmierung mit Sockets kennen. Die Funktionen zum Umgang mit Sockets sind in jeder Programmiersprache ähnlich, da Sockets vom Betriebssystem angeboten werden. In der Laborübung wird als Programmiersprache Python verwendet. Python ist eine moderne Interpreter-Sprache, die als einfach zu erlernen gilt. Python hat sich in den letzten Jahren zu einer der populärsten Sprache für die Entwicklung von Netzwerk- und Webanwendungen entwickelt.

Dieser Laborversuch besteht aus drei Aufgaben, die in zwei Laborstunden durchzuführen sind. Der erste Versuch dient zum Einstieg in die Socket-Programmierung mit Python und greift die Skript-Beispiele aus der Vorlesung auf. Im zweiten Versuch lernen Sie „telnet“ kennen. Telnet ist ein Tool mit dem Sie über die Kommandozeile in einen Socket schreiben bzw. aus einem Socket lesen können. Als Beispiel für telnet dient die tägliche Mail, die Sie dann auch mit einem Python-Skript schreiben und lesen dürfen. Das dritte Beispiel ist das Erstellen eines Port Scanners, um den Umgang mit Sockets und Threads zu üben.

2 Vorbereitung

Wenn Sie mit Python noch nicht vertraut sind, arbeiten Sie sich in Python ein, indem Sie ein Python-Tutorial durchgehen. Python ist auf den Laborrechnern bereits installiert. Sie können beispielsweise Spyder oder IDLE als IDE nutzen.

Sockets werden in Python von der Bibliothek „socket“ unterstützt. Beschreibungen der Socket-Programmierung in Python finden Sie unter

1. https://docs.python.org/3/howto/sockets.html
2. https://docs.python.org/3/library/socket.html

3 ECHO-Anwendung

In dieser Aufgabe verwenden Sie einen ECHO-Client und einen ECHO-Server, wie sie in der Vorlesung beschrieben wurden. Nutzen Sie zur Kommunikation einmal ein Datagram-Socket(UDP) und einmal ein Stream-Socket (TCP). Der ECHO-Client sendet einen String an den ECHO-Server und der Server antwortet mit dem rückwärts gelesenen String.

Mit der Aufgabenstellung werden auch unkommentierte Skripte für UDP/TCP Client/Server hochgeladen.

3.1 Lokale Kommunikation

Testen Sie die Skripte zunächst lokal, indem Sie für Client und Server die IP-Adresse 127.0.0.1 (localhost) verwenden. Starten sie dazu erst das Server-Skript und dann das Client-Skript jeweils in einem Windows-Cmd-Fenster. Mit dem Tool rawcap (www.netresec.com/?page=RawCap) können Sie die Kommunikation auf dem localhost Interface aufzeichnen und mit Wireshark anschauen. Nutzen Sie außerdem das Tool CurrPorts (https://www.nirsoft.net/utils/cports.html), um die aktiven Sockets zu überwachen.

Erklären Sie den Zusammenhang von ausgetauschten Paketen und Python-Code, indem Sie

1. für jedes gesendete Paket bestimmen, welcher Befehl in welchem Skript (Client/Server) dafür verantwortlich ist, dass das Paket gesendet wird
2. für jeden blockierenden Befehl bestimmen, die Ankunft welches Pakets dafür verantwortlich ist, dass die Ausführung des Befehls vervollständigt wird

3.2 Netzwerk-Kommunikation

Suchen Sie sich ein Partner-Team. Konfigurieren Sie die IP-Adressen jetzt so, dass die Kommunikation im lokalen Netzwerk möglich ist.

Beantworten Sie die folgenden Fragen durch Experimente und unter Verwendung der Python-Hilfe:

1. Wie können Sie im Client Python-Skript die IP-Adresse und Port-Nummer des verwendeten lokalen Sockets bestimmen (im Sinne von herausfinden)?
2. Wann (in welcher Code-Zeile) und woher erhält ein Client seine IP-Adresse und Port-Nummer?
3. Wie können Sie im Client-Skript die IP-Adresse und Port-Nummer des Sockets setzen?
4. Warum müssen Sie Timeouts verwenden und wie funktioniert try … except? Mit welchem Befehl können Sie einen gemeinsamen Timeout für alle Sockets setzen?
5. Finden Sie experimentell heraus, ob Sie einen Server betreiben können, der ECHO-Anfragen auf dem gleichen Port für UDP und TCP beantwortet?

4 Mail

In diesem Versuch verstehen Sie am Beispiel eines Mail-Clients, wie Anwendungen über Sockets kommunizieren. Sie lernen das Tool „telnet“ kennen, mit dem ein Socket geöffnet werden kann, um über die Kommandozeile mit einem Server zu kommunizieren. Im ersten Versuch, bauen Sie eine Verbindung zu einem Mail-Server, um über SMTP eine Mail zu schreiben und über IMAP Mails zu lesen. In einem zweiten Versuch implementieren Sie einen Mail-Client in Python, um Mails zu versenden oder abzurufen.

4.1 Mail mit telnet

Mit telnet können Sie einen Socket zu einem Server auf einem bestimmten Port öffnen und dann über Kommandozeilen-Eingabe mit dem Server kommunizieren. In diesem Beispiel sollen Sie über telnet Mails verschicken und abrufen. Dazu verbinden Sie sich mit dem Mail-Server der Hochschule und benutzen die Protokolle SMTP (Simple Mail Transfer Protocol) und IMAP (Internet Message Access Protocol), um Mails zu versenden und abzurufen. Informationen zu SMTP und IMAP finden Sie in der Vorlesung oder auf Wikipedia.

4.1.1 SMTP

Hinweis: Zeichnen Sie den Nachrichtenaustausch mit WireShark auf.

Öffnen Sie mit telnet einen Socket zum SMTP-Port des Mail-Servers asmtp.htwg-konstanz.de. Melden Sie sich mit dem Account (Login: rnetin, Password: ntsmobil) an. Schreiben Sie eine Email an einen ihrer Mail-Accounts und prüfen Sie, ob die Mail angekommen ist. Senden sie von ihrem Mail-Account eine Antwort.

Hinweise:

1. Sie können „telnet“ entweder in der Windowskommandozeile oder über „Putty“ öffnen.
2. Gehen Sie vor wie beispielsweise auf Wikipedia beschrieben:

   https://de.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol
   https://de.wikipedia.org/wiki/SMTP-Auth

3. Login und Passwort müssen als Base64-kodierte Zeichenketten übertragen werden. Sie können die Konvertierung mit Hilfe eines Online-Tools durchführen oder die Python-Bibliothek base64 verwenden. Importieren Sie dazu die base64-Bibliothek in der Python Shell (`import base64`) und verwenden Sie folgende Befehle zum Konvertieren zwischen ASCII- und Base64-Zeichenketten:

   ```
   (base64.b64encode('ASCII-String'.encode('utf-8'))).decode('utf-8')

   (base64.b64decode('Base64-String')).decode('utf-8')
   ```

3. Die Antwort-Mail können alle Labor-Teilnehmer lesen, da alle den gleichen Mail-Account für die Laborübung nutzen.

4. Achten Sie bei der Eingabe von Text in telnet darauf, dass Sie sich nicht vertippen. Eine Korrektur ist nicht mehr möglich. Ein praktikabler Umgang mit diesem „Problem“ besteht darin, die Kommandos in einem Editor zu entwerfen und dann in das „telnet-Fenster“ zu kopieren. Das ist auch hilfreich für die nächste Aufgabe, wenn Sie den Mail-Client in Python implementieren.

4.1.2 IMAP

Öffnen Sie mit telnet einen Socket zum IMAP-Port des Mail Servers imap.htwg-konstanz.de. Melden Sie sich wiederum mit dem Account rnetin an. Bestimmen Sie, wie viele Nachrichten sich in der Inbox befinden, wie viele ungelesen sind und wie viele neu sind. Geben sie eine Nachricht ihrer Wahl aus. Zeichnen Sie den Nachrichtenaustausch mit WireShark auf.

Hinweise:

1. Gehen Sie auch hier vor wie bei Wikipedia beschrieben:

   https://de.wikipedia.org/wiki/Internet_Message_Access_Protocol

2. Eine Übersicht der grundlegenden IMAP-Befehle finden Sie im RFC 3501, Abschnitt 6 https://tools.ietf.org/html/rfc3501#section-6

   Sie benötigen minimal die Befehle `login`, `select`, `fetch` und `logout`, um den Versuch durchzuführen.

4.1.3 Fragen

Sie haben ihre Mail-Session mit WireShark aufgezeichnet. Überprüfen Sie, ob

1. Login und Nutzername in Klartext oder verschlüsselt übertragen wurden?
2. Ihre Email verschlüsselt oder in Klartext übertragen wurde.

Kopieren Sie aus Wireshark die Paketsequenz zur SMTP Session und zur IMAP Session.

4.2 Mail in Python

Implementieren sie einen Mail-Client in Python. Sie können sich frei entscheiden, ob der Client Mails senden, lesen oder beides können soll. Benutzen Sie nicht die in Python vorhandenen Mail-Bibliotheken (smtplib, imaplib) sondern programmieren Sie direkt auf Sockets. Sie können aber gerne zusätzlich einen Mail-Client mit den Bibliotheken implementieren. Ein Beispiel finden Sie am Ende der Bibliotheksbeschreibung (https://docs.python.org/3/library/smtplib.html, https://docs.python.org/3/library/imaplib.html)

Hinweise:

1. Verwenden Sie die Python Bibliothek base64, um Login und Passwort in Base64-Code zu konvertieren.
2. Alle Daten werden im UTF-8-Format übertragen.

3. Alle gesendeten Zeilen müssen mit "\r\n" enden, damit der Mail-Server das Zeilenende erkennt. Auch diese Zeichen müssen „UTF-8“-codiert gesendet werden.

5 Port Scan

5.1 Beschreibung

In diesem Versuch führen wir einen Port-Scan durch, um herauszufinden, welche Ports auf einem Server geöffnet sind. Wir scannen zum einen die standardisierten Ports von 1-50 nach offenen TCP Ports. Wenn wir einen offenen Port finden, versuchen wir, eine Nachricht an diesen Port zu schicken. Weiterhin vermuten wir, dass auf dem Server ECHO-Dienste für die Übertragung mit TCP und UDP laufen.

Ist ein TCP-Port auf einem Server geöffnet, dann wird ein Verbindungsaufbau auf diesem Port akzeptiert. Ist der TCP-Port nicht offen, so antwortet der Server entweder nicht (Windows Fehler-Code 10060) oder mit einem RST+ACK, in dem der Verbindungsaufbau zurückgewiesen wird (Windows Fehler-Code 10061).

Ist ein UDP Port auf einem Server geöffnet, so antwortet der Server entweder mit einer Nachricht oder überhaupt nicht. Die Reaktion hängt sowohl vom empfangenden Dienst als auch von der Nachricht selbst ab. Ist der UDP Port nicht geöffnet, so antwortet der Server entweder nicht oder mit einem ICMP Paket vom Typ 3, mit dem er mitteilt, dass das Ziel nicht erreichbar ist (Windows Fehler-Code 10054).

5.2 Versuch

5.2.1 TCP Port Scanner

Implementieren Sie ein Skript, das eine gegebene Anzahl von TCP-Ports auf einem gegebenen Server scannt und die offenen Ports zurückliefert. Führen Sie das Script für den Labor-Server 141.37.168.26 und Ports zwischen 1 und 50 durch. Zeichnen Sie die Kommunikation mit WireShark auf.

Starten Sie die einzelnen Port-Anfragen als Threads, um den Scan-Vorgang zu beschleunigen. Sie können einfach eine Funktion mit

```
t=Thread(target=<function>,args=(<arg>,))
```

starten. Achten Sie darauf, dass der Thread beendet werden kann. Einfach geht dies mittels eines global Flags `Continue`, das der Thread kontinuierlich abfragt und sich bei `Continue==False` beendet.

Hinweis: https://docs.python.org/3.4/library/threading.html

5.2.2 UDP Port Scanner

Erweitern Sie das Script, um auch UDP Ports zu scannen. Führen Sie das Script für den Labor-Server 141.37.168.26 und Ports zwischen 1 und 50 durch. Zeichnen Sie die Kommunikation mit

WireShark auf. Unterscheiden Sie Ports, auf denen Sie keine Antwort bekommen und Ports, auf denen Sie Fehlermeldung 10054 erhalten.

5.3 Fragen

1. Geben Sie die Liste der offenen TCP und UDP Ports an.
2. Wählen Sie für TCP und UDP jeweils einen offenen und einen geschlossenen Port und erklären Sie die entsprechende Paketsequenz, die Sie in WireShark aufgezeichnet haben.
3. Auf Port 7 des Servers läuft ein ECHO-Dienst. Testen Sie ihr Client-Script mit dem ECHO-Server. Versuchen Sie das TCP und das UDP Script.