Manuelles durchführen von HTTP-Requests

Ein HTTP-Request besteht aus `Header` sowie `Body`. Im Header werden Daten übertragen die das Protokoll zum arbeiten braucht.

Der Aufbau eines HTTP-Requests besteht aus 3 Teilen im groben. Hier eine Beispielhafte Anfrage:

```
GET / HTTP/1.0
Host: www.spengergasse.at
```

Wichtig hierbei ist das eine neue Zeile nicht nur aus einem `Newline \n` bzw. Linefeed besteht, sondern zusätzlich auch ein `Carriage-Return \r` dabei haben. Streng genommen sieht das obere Paket also so aus:

```
GET / HTTP/1.0\r\n
Host: www.spengergasse.at\r\n
\r\n
```

Exkurs Steuerzeichen

`\n` bzw. `\r` ist eine Darstellung von Steuerzeichen in plain text. In ASCII sind die ersten 32 Zeichen steuerzeichen. `\n` ist das 13. und `\r` das 10. Zeichen. Sie sind dafür Zuständig dass beim drücken der Enter-Taste auch eine neue Zeile erzeugt wird. Es gibt noch viele andere Zeichen `\0` ist das Ende eines Strings, das letzte Zeichen ist zum löschen vorheriger Zeichen usw. `\r\n` bedeutet also das eine spezielle Art von Enter notwendig ist in HTTP. `\n` alleine wäre auch möglich, ist aber nicht HTTP-Konform. Texteditoren unter Windows benutzen für Enter automatisch `\r\n`. Unter Linux nur `\n`.

ASCII-Zeichentabelle, hexadezimale Nummerierung

Code	…0	…1	…2	…3	…4	…5	…6	…7	…8	…9	…A	…B	…C	…D	…E	…F
0…	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1…	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2…	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3…	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4…	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5…	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6…	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7…	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

HTTP Request selbst erstellen

Auf Linux gibt es das Programm *netcat* bereits vorinstalliert. Ausführbar mit dem Befehl `nc`. Um es auszuführen kannst du dir WSL2 herunterladen oder dir die für Windows kompilierte Version hier herunterladen.

Das Programm läuft in der Konsole und nimmt als Argument einen Host sowie einen Port. Darauf öffnet es dann einen Socket. Das Programm nimmt nun Eingaben entgegen die es wenn eine neue Zeile kommt an den Server schickt. Probier mal das Programm mit folgenden Argumenten zu starten: `nc www.google.at 80`.

Dann baut *netcat* eine Socket-Verbindung zu den Google-Servern auf Port 80 (dem Standard HTTP-Port) auf. In der Eingabemaske kannst du jetzt einen GET-Request absetzen indem du `GET / HTTP/1.0` schreibst. Die Google-Server sind hier sehr tolerant in dem was für Anfragen sie annhemen. `GET /` reicht auch schon.

Request auf spengergasse.at

Der Server unserer Schule (und so gut wie alle anderen) ist hier weit weniger tolerant. Das selbst zusammengebaute Paket muss exakt den HTTP-Spezifikationen entsprechen.

Setze dafür zunächst einen HTTP-Request mit deinem Browser ab und speichere die Anfrage als Textdatei. Was steht sonst noch in dem Header aus GET ...?

> Liste die Einstellungen auf und erkläre wofür sie gut sind.

Den Dateiinhalt kannst du nun über *netcat* an den Server schicken. Wenn alles passt kommt der Response-Code 200 zurück. Ist beim Request was falsch kommt 400 zurück.

Die Request-Kommandos sehen ca. so aus:

Linux:

```
cat request.txt | nc www.spengergasse.at 80
```

Windows:

```
type request.txt | .\nc.exe www.spengergasse.at 80
```

Die Befehle `cat` bzw. `type` geben den Dateiinhalt auf `stdout` aus. Statt das wir ihn auf der Konsole anzeigen übergeben wir ihn jedoch als `stdin` an ein weiteres Programm -natcat. Dies Funktioniert mit dem Pipe-Operator `|`. Dort rufen wir dann netcat mit Host, Port, und bereits bestehenden Eingaben auf.

Besonderheiten bei HTTPS

HTTPS (offizieller Name: HTTP Over TLS) hat zwischen TCP und HTTP noch TLS dazwischen. Das ermöglicht es zum einen dass die Daten verschlüsselt übertragen werden, jedoch kann man so nicht mehr ohne weiteres rein textbasierte Daten an einen HTTPS Server schicken. Man muss sich mit dem darunterliegenden TLS-Protokoll einen Schlüssel ausmachen mit dem alle weiteren Daten verschlüsselt werden.

Eine TLS-Verbindung mit einem beliebigen Port kann man mit folgendem Befehl aufmachen:

```
openssl s_client -connect <someserver>:443
```

Nach dem Austausch der Zertifikaten und dem Schlüsselaufbau wartet das Programm auf rein textbasierte Eingaben. Gibt man nun `GET /` ein während man mit den Google-Servern verbunden ist bekommt man (über TLS) HTTP-Daten zurück.

Auf Ubuntu (bzw. Linux generell) kann man auch mit `sudo apt install ncat -y` einen Nachfolger von `nc` installieren welcher `TLS` unterstützt.

Einen einfachen HTTPS/1.1 Request kann man dann ausführen mit:

```
printf 'GET / HTTP/1.1\r\nHost: google.com\r\n\r\n' | ncat --ssl google.com 443
```

Fragen

> Finde herraus wo du in deinem Browser dir anschauen kannst welche Requests an die HTTP-Server geschickt werden. Tipp: Es ist auch dort wo sich die Entwicklerkonsole befindet.

> Wofür stehen die Response Codes 200, 201, 400, 404, 500?

> Welche Request-Methoden neben `GET` gibt es noch? Wie müsste man die Requestdatei umbauen um einen Post o.ä mit Daten abzusetzen?

> Wie sieht deine Textdatei aus die zu einem erfolgreichen Request geführt hat? Worauf musstest du achten? Gibt es Einstellungen die du noch weglassen kannst?

> Führe einen Request auf http://httpbin.org aus. Wie müssen header und body aussehen damit du im body ein JSON mit einem POST-Request übertragen kannst? Wie sieht die Response aus?

> Kann man HTTP über UDP ausführen?

Jeder der mir diese Fragen am Ende der Stunden beantworten kann bekommt ein Plus. Allerdings sind die Fragen auch unabhängig davon wichtig. Bei der nächsten PLF werden die Fragen sehr ähnlich zu den hier gestellten sein.

Quellen:

> Carriage Return

> HTTP RFC2616

> HTTPS RFC2818