# **Labøving til uke 4 - Bli kjent med TCP og TLS ved bruk av Wireshark**

Del 1 undersøker hvordan TCP etablerer en forbindelse med Three-way handshake (3WHS), kjent som Syn, SYN-ACK og ACK. Deretter undersøkes hvordan data overføres pålitelig ved å bruke sekvensnummer og kvitteringsnummer.

Del 2 undersøker hvordan HTTPS etableres, dvs at TLS setter opp en kryptert forbindelse for HTTP-kommunikasjon.

# **Øvingen – pakkefangst med Wireshark**

Start Wireshark og gjør et oppslag på Blackboard uten at man er pålogget fra før. Som regel dukker det opp så mange pakker i Wireshark at det er vanskelig å finne akkurat denne sesjonen vi ønsker å analysere. Vi er på jakt etter pakker som viser «ntnu-blackboard.com» i pakkelinjen

1. For å se hvilke TLS-sesjoner som har vært i bruk kan man først sette display-filer **tls**

Det kan fortsatt være mange pakker, PC-en er aktiv med masse online-aktiviteter.

1. Bruk et mer spesifikt display-filter: **tls.handshake.extensions\_server\_name=="ntnu.blackboard.com"**

Nå skal det vises bare en eller et fåtall «Client Hello» fra vår sesjon.

En nettleser kan sette opp parallelle sesjoner for å bedre responstiden. Vi ønsker å undersøke en særskilt strøm. Høyreklikk på en av pakkene med første Client Hello og velg **Follow / TCP-stream**

**Del 1: Undersøke TCP oppkopling og dataoverføring**

**A. TCP oppkopling (3WHS)**

Identifiser de tre oppstartspakkene fremst i listen som har flaggene SYN, SYN-ACK og ACK.   
Fyll inn tabeller:

*SYN-pakke 1 fra klient til tjener*

|  |  |  |
| --- | --- | --- |
| **Innhold** | **Verdi** | **Merknad** |
| Flagg |  | Oppstartspakke |
| Destination Port adr |  | Velkjent portnummer HTTPS |
| Source Port adr |  | Dynamisk/ kortlivde portnummer |
| Rel. seq. nr |  | ISN-nummer er 32 bit, her viser Wireshark et relativt 0 |
| Rel. ack. nr |  |  |
| Options |  | Eks. SACK=Selective Repeat v. retransmisjon |
| TCP-Pakkehode størrelse |  | Vanlige segmenter har 20 byte pakkehode |

*SYN-ACK pakke 2: fra Tjener til klient*

|  |  |
| --- | --- |
| **Innhold** | **Verdi** |
| Flagg |  |
| Destination Port adr |  |
| Source Port adr |  |
| Rel. seq. nr |  |
| Rel. ack. nr |  |
| Options |  |
| TCP-pakkehode størrelse |  |

*ACK pakke 3: fra Klient til tjener*

|  |  |
| --- | --- |
| **Innhold** | **Verdi** |
| Flagg |  |
| Destination Port adr |  |
| Source Port adr |  |
| Rel. seq. nr |  |
| Rel. ack. nr |  |
| Options |  |
| TCP-pakkehode størrelse |  |

**B. TCP dataoverføring**

Du skal undersøke hvordan data kvitteres slik at overføringen blir pålitelig. Dette kan egentlig startes hvor som helst i en TCP-assatrøm.   
*TIPS 1: Du kan fortsatt bruke «Follow /TCP-stream»* som filtrerer vekk alle pakker som ikke gjelder denne sesjonen.   
*Tips 2: Det vil gå en datastrøm i begge retninger og som kvitteres hver for seg.*

Fyll inn tabeller. Det benyttes relativ nummerering på sekvens- og kvitteringsnummer (det nummeret som Wireshark viser).

1. Første client hello data-pakke fra klient

|  |  |
| --- | --- |
| **Innhold** | **Verdi** |
| Sekv. nr |  |
| Ack. nr |  |
| Nyttelast (byte) |  |

2. Første svar fra tjener (dette kan være en pakke uten nyttelast, bare med ACK og kvitteringsnummer)

|  |  |
| --- | --- |
| **Innhold** | **verdi** |
| Sekv. nr |  |
| Ack. nr |  |
| Nyttelast (byte) |  |

*Undersøk videre pakkeflyt* og dann deg et bilde av hvordan sekvens- og kvitteringsnummer endres. Merk at kvitteringsnummer som returneres er lik mottatt <sekvensnummer + nyttelast>.

**Del 2: TLS meldingsutveksling**

Start med å lese den følgende teksten under. Deretter skal du bruke Wireshark til å se at TLS faktisk følger punktene 1-5 i praksis også.

Teksten er hentet fra [engelsk Wikipedia](https://en.wikipedia.org/wiki/Transport_Layer_Security).

1. The handshake begins when a client connects to a TLS-enabled server requesting a secure connection and the client presents a list of supported cipher suites (ciphers and hash functions).
2. From this list, the server picks a cipher and hash function that it also supports and notifies the client of the decision.
3. The server usually then provides identification in the form of a digital certificate. The certificate contains the server name, the trusted certificate authority (CA) that vouches for the authenticity of the certificate, and the server's public encryption key.
4. The client confirms the validity of the certificate before proceeding.
5. To generate the session keys used for the secure connection, the client either:
   1. encrypts a random number (PreMasterSecret) with the server's public key and sends the result to the server (which only the server should be able to decrypt with its private key); both parties then use the random number to generate a unique session key for subsequent encryption and decryption of data during the session
   2. uses Diffie–Hellman key exchange (or its variant elliptic-curve DH) to securely generate a random and unique session key for encryption and decryption that has the additional property of forward secrecy: if the server's private key is disclosed in future, it cannot be used to decrypt the current session, even if the session is intercepted and recorded by a third party.
6. This concludes the handshake and begins the secured connection, which is encrypted and decrypted with the session key until the connection closes. If any one of the above steps fails, then the TLS handshake fails, and the connection is not created.

When secured by TLS, connections between a client (e.g., a web browser) and a server (e.g., wikipedia.org) should have one or more of the following properties:

1. The connection is private (or has **confidentiality**) because a symmetric-key algorithm is used to encrypt the data transmitted.
2. The identity of the communicating parties can be **authenticated** using public-key cryptography. This authentication is required for the server and optional for the client.
3. The connection is reliable (or has **integrity**) because each message transmitted includes a message integrity check using a message authentication code to prevent undetected loss

**Praktisk oppgave**

Steg 1) Client Hello

Finn *cipher suite*-feltet og identifiser listen med suiter som støttes. Identifiser innholdet i Cipher Suite (støttede algoritmer) og verifiser at det inngår forkortelser som SHA, AES og RSA. Minn deg selv på hva disse forkortelsene betyr.

Steg 2) Server Hello

Finn cipher suite og noter hvilken suite som er valgt. Sjekk at denne suiten står i listen fra Client Hello

Steg 3) Certificate, server key exchange, server done

Observer at serveren her sender både sertifikater og hvilke mulighet som tilbys for oppretting av sesjonsnøkler. Dette (eller disse) sertifikatene er det klienten verifiserer. **OBS: Denne pakken sedens bare ved etablering av forbindelsen, ikke ved videre nedlastinger av objekter fra websiden.** Derfor valgte man det første client Hello i Wireshark

Steg 4 og 5) Client Key exchange/Change cipher Specs.

Etter dette benyttes kryptert overføring og videre HTTP-kommunikasjon er ikke lesbar for oss i pakkefangsten.