Security on the TLS/SSL Handshake Protocol and Application Data Protocol

ECE-628 COMPUTER NETWORK SECURITY

PROJECT REPORT

Instructor: Professor G. Gong

Submitted by:

Mahalakshmi Selvarajan (21024263)

<u>**REPORT:**</u>

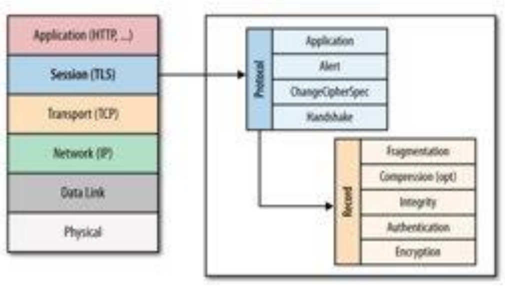
<u>TLS (Transport Layer Security):</u>

TLS (transport layer security) is a cryptographic protocol that is used for secure end-to-end Data exchange between applications over the Internet. It is a protocol between a client and a server to secure applications. It is used in various applications, including web browsing, email, and online banking. TLS is designed to provide confidentiality, integrity, and authentication of data transmitted between two endpoints.

In order for a user to connect securely to a server over the Internet, SSL was created. As a result, in both SSL and TLS, the two parties are referred to as client and server. Particularly when SSL was created, the goal was for a client to confirm the authenticity of the server it was connected to through the Internet. This is also the reason why server authentication in TLS is necessary but client authentication using public key certificates is optional.

TLS is normally implemented on top of TCP in order to encrypt Application Layer protocols such as HTTP, FTP, SMTP and IMAP, although it can also be implemented on UDP, DCCP and SCTP as well (e.g. for VPN and SIP-based application uses). This is known as Datagram Transport Layer Security (DTLS).

TLS is a layered protocol consisting of a record protocol and four client protocols as shown in the below figure. The record protocol is on top of the transport protocol, like the Transmission Control Protocol (TCP). The TLS record protocol encapsulates upper-layer protocols, that is, client protocols. The four TLS upper layer protocols are the handshake protocol, the change cipher spec protocol, the alert protocol, and the application data protocol. The record layer will take messages to be transmitted from the upper client, fragment to the manageable blocks, optionally compress, generate message authentication code, encrypt, and then transmit. The data will be received and then decrypted, verified, decompressed, rebuilt, and sent to the upper client.



A client and a server establish a secure connection using the TLS/SSL handshake protocol. There are various steps in the handshake:

1. ClientHello: A Client Hello message, includes the client's SSL/TLS version, supported cipher suites, and other pertinent information, which is sent by the client to the server to start a connection.
2. ServerHello: The server responds with a Server Hello message that contains the server's selected encryption suite, its highest supported SSL/TLS version, and other pertinent data. Along with this, the client and the server generate a random number and exchange them in the hello messages.
3. Certificate Exchange: The client receives the server's digital certificate, which includes the server's public key. In order to confirm that it is speaking with the intended server, the client can verify the certificate.

4. Key Exchange: The client and server exchange key material, through a process called asymmetric key exchange (in our code we implement RSA and DHE). From this key information, a shared secret, also called the session key, is derived.
5. Finished: Message Finished is always sent immediately after a ChangeCipherSpec message to conduct key confirmation, implicit authentication, and post-verification of the cipher suite negotiation. A finished message that is encrypted with the session key is sent by the client and the server simultaneously. This message verifies that the handshake has been successfully completed and that a secure connection has been established.
6. Secure communication: Following the handshake, the application data protocol can be used by the client and server to safely communicate data. The session key is used to encrypt the transmitted data, protecting its confidentiality and integrity.

DHE key exchange approach:

With the Diffie-Hellman Ephemeral (DHE) key exchange approach, a pre-master secret key is created during the SSL/TLS handshake process. The main advantage of using DHE is Perfect Forward Secrecy (PFS), which ensures that prior communication is safe even if a private key is compromised.

Here's an overview of the SSL/TLS handshake process when DHE is used for key exchange:

1. The "ClientHello" message is sent by the client to the server and it contains information such as the supported encryption suites, SSL/TLS versions, and other specifics.
2. The server defines the SSL/TLS version, cipher family, and other parameters in its "ServerHello" response.
3. The server then sends its digital certificate to the client, which contains its public key.
4. The client verifies the server's certificate using a trusted Certificate Authority (CA).
5. The Diffie-Hellman parameters (prime number, generator, and the server's transient public key) are sent by the server in a "ServerKeyExchange" message.
6. The client creates its own ephemeral private and public keys, and in a "ClientKeyExchange" message, it sends its public key to the server.
7. The shared secret (pre-master secret key) is independently computed by the client and server using both parties' public keys and private keys.
8. The master secret key is later used to generate symmetric encryption keys for data encryption and decryption after being derived from the pre-master secret key.
9. The "ChangeCipherSpec" messages between the client and server signal that they will now communicate using symmetric encryption keys.
10. The client and server exchange "Finished" messages to confirm the successful completion of the handshake process.

Following the handshake, the client and server can communicate data using symmetric encryption in a secure manner. Strictly speaking, symmetric algorithms like AES are used for data encryption. The DHE key exchange guarantees Perfect Forward Secrecy and assures that the pre-master secret key is unique for each session, increasing the security of the communication.

RSA encryption in key exchange:

During the SSL/TLS handshake, RSA encryption is used to create the pre-master secret key. A secure connection is established only after the SSL/TLS handshake between a client and a server. The creation of the pre-master secret key is one of several significant steps in the process.

Here's an overview of the process when RSA is used for key exchange:

1. The "ClientHello" message is sent by the client to the server and it contains information such as the supported encryption suites, SSL/TLS versions, and other specifics.
2. The server defines the SSL/TLS version, cipher family, and other parameters in its "ServerHello" response.
3. The server delivers the client its digital certificate, which is encrypted using RSA and contains the server's public key. Using a reliable Certificate Authority(CA)., the client checks the server's certificate. The entity authentication part is already considered to be done. so, we will not be implementing that in our code.
4. As part of the "ClientKeyExchange" message, the client creates a random pre-master secret key, encrypts it using the server's public key from the certificate, and sends it.
5. The server uses its private key to decrypt the pre-master secret key.
6. From the pre-master secret key, both the client and the server derive the master secret key, from which the symmetric encryption key for data encryption and decryption is derived.
7. The client and server exchange "ChangeCipherSpec" messages to let one another know they will now communicate using symmetric encryption keys.
8. To verify that the handshake procedure had successfully completed, the client and server exchange "Finished" messages.
9. After the handshake, the client and server can communicate data using symmetric encryption in a secure manner.

Note that RSA is used only for key exchange in this process, and the actual encryption of data uses symmetric algorithms like AES.

Security Analysis of TLS:

The Security Analysis of TLS mainly depends on the factors listed below.

- **Certificate Authentication**: To avoid man-in-the-middle attacks, it is essential to make sure the server's certificate is valid and trustworthy.
- **Protected Cipher Suites:** Using secure cipher suites with current up to date key exchange mechanisms and encryption algorithms helps defend against eavesdropping and other cryptographic attacks.
- The TLS (Transport Layer Security) protocol's key exchange procedure is essential because it creates a secure channel of communication between the client and the server. The importance of the key exchange process in TLS can be highlighted in the following ways:
  - **Confidentiality:** The key exchange procedure makes sure that the data being communicated between the client and the server is confidential. TLS makes sure

that all data communicated between the client and the server is encrypted and protected against eavesdropping or interception by other parties by generating a shared secret key that is only known to the client and the server.

- **Integrity:** The key exchange procedure makes sure that the data being sent between the client and the server is transferred reliably. TLS ensures that the data received by the client and the data sent by the server are consistent and have not been tampered with by confirming the server's identity and ensuring that the data was not altered during transmission.
- **Authentication:** The key exchange procedure enables client and server authentication of one another. TLS checks the identity of the server and gives the client confidence that it is dealing with a genuine server and not an imposter by using digital certificates and public key cryptography.

Code Algorithm:

SSLserver:

```
receive ClientHello record (client random, cxn)
if cxn ==RSA:
        send ServerHello record (server random, RSA)
        generate_rsa_keys (serverpublickey, serverprivatekey)
        send serverpublickey(pks)
        receive RSA_Enc_pks (premaster key)
        decrypt RSA_Dec_sks (RSA_Enc_pks (premaster key))
        session key = HKDF<SHA256>(premaster key, client random,  server random)
        Receive Client Finish
if cxn ==DHE:
        send ServerHello record (server random, DHE)
        receive ClientHello record (client random, DHE)
        generatekeypair (serverpublickey, serverprivatekey)
        send serverpublickey        // ServerKeyExchange record
        receive clientpublickey
        generate premaster key
        session key = HKDF<SHA256>( premaster key, client random,  server random)
        Receive Client Finish
```

SSLclient:

```
if cxn ==RSA:
        send ClientHello record (client random, RSA)
        receive ServerHello record (server random, RSA)
        receive serverpublickey(pks)
        generate premaster key
        send RSA_Enc_pks (premaster key)       // ClienKeyExchange record
        session key = HKDF<SHA256>( premaster key, client random,  server random)
        send ClientFinish

if cxn ==DHE:
        send ClientHello record (client random, DHE)
        receive ServerHello record (server random, DHE)
        generate p, g, q
        generate (clientpublickey, clientprivatekey)
        send clientpublickey              // ClienKeyExchange record
```

```
receive serverpublickey
generate premaster key
session key = HKDF<SHA256> (premaster key, client random, server random)
send ClientFinish
```

Code explanation:

For the project implementation, the TCP and most of the SSL code are already provided. Hence, our implementation is focused on completing two of the following functions: “connect()” in SslClient and “accept()” in SslServer to send and receive handshake records between client and server. In this project, we will be using DHE key exchange approach and RSA encryption for implementing key exchange in TLS.

As the first step of the TLS handshake, the SSL client sends a ClientHello record containing the “client random” and the key exchange protocol /algorithm (RSA or DHE) which is being used. The SSL server responds with a Server Hello record that contains a “server random” and that record also mentions the key exchange protocol that is going to be used. The entity authentication part is already considered to be done. so, we will not be implementing that in our code.

Hence, for **RSA encryption** in key exchange, the server will be directly sending the server’s public key to the client. The client then generates the premaster key and sends a ClientKeyExchange record containing the encrypted premaster key which is encrypted using the server’s public key. The server then decrypts the premaster key using its private key. Further, both the client and the server use the premaster key, the “client random”, and “server random” received in the handshake to derive the master secret. From the master secret, we derive the session key using the KDF (key derivation function). The main goal of our project here is to implement key generation. So, after generating the key, we directly set up the SSL/TLS connection to encrypt and decrypt the App data.

For **DHE key exchange protocol**, the client and server generate their own private and public key using the P, Q, and G value (DHE parameters) which is generated randomly every session. The server then directly sends the server’s public key to the client in ServerKeyExchange record and the client directly sends its public key to the server in ClientKeyExchange record. Both the client and the server then generate the premaster key using the server’s and client’s public key, which is then used to derive the master secret. From the master secret, we derive the session key using the KDF (key derivation function. Now that the keys are generated, we directly set up the SSL/TLS connection to encrypt and decrypt the App data.

A. Explain how entity authentication is conducted, for generating the pre-master key in DHE and RSA. Comment on the advantages and disadvantages of these two different key-sharing processes.

Entity Authentication:

Verifying the identity of a communicating entity, such as a person, machine, or system, is known as entity authentication. Entity authentication is a critical step in Generating the pre-master key for the Diffie-Hellman (DHE) and RSA key exchange protocols. This step confirms that both parties are whom they say they are.

Using the DHE key exchange protocol, two parties can create a shared secret across an unsecured channel without disclosing any information to eavesdroppers. In DHE, entity authentication is generally carried out using digital signatures, in which each party signs and sends the other party their public key once it has been signed. The other party can then confirm their identity and validate the signature using the sender's public key. After mutual authentication, the parties can exchange their public keys and use a mathematical formula to create the pre-master key.

RSA is another key exchange protocol that creates a shared secret key using public-key cryptography. Here, a certificate authority (CA), a dependable third party that confirms the identities of the communicating entities, is used in RSA entity authentication. Each entity possesses a set of public and private keys, with the public key being included in a digital certificate that has been signed by the CA. When two parties desire to create a shared key, they exchange digital certificates and confirm each other's identities by examining the certificate's signature. Once the identities have been verified, the parties can trade public keys and use the RSA technique to create the pre-master key.

DHE:

Advantages of DHE:

1. Perfect Forward Secrecy (PFS): DHE offers PFS, thus previous communication sessions are still safe even if a communication partner's private key is compromised in the future. In other words, it provides Dynamic key generation: DHE generates a unique shared secret key for every session, adding extra protection from attacks that depend on cracking a single key.

2. Agreement on a shared secret key: DHE enables two parties to arrive at a compromise on a shared secret key through an unsecured channel without actually exchanging the key. As a result, even if the communication is intercepted, the attacker will not be able to figure out the shared secret key without figuring out the underlying mathematical problem.

Disadvantages of DHE:

1. Man-in-the-middle (MITM) attack susceptibility: If the authentication process is not properly implemented, DHE is susceptible to MITM attacks. DHE is unique that it may produce keys on-the-fly and does not require a trusted third party like a certificate authority. The drawback is that it is susceptible to man-in-the-middle attacks, in which a hacker intercepts a communication and pretends to be one of the participants.

2. Performance overhead: DHE involves a number of processes, one of which is the creation of a fresh session key for every session, which may cause some performance costs. This is extremely important for servers that manage a lot of connections.

RSA:

Advantages of RSA:

1. The RSA method offers high levels of security for key exchange. It is built on the idea that huge prime numbers are challenging to factor, making it challenging for an attacker to separate the private key from the public key.

2. RSA is extensively supported by cryptographic tools and libraries, making it compatible with a variety of platforms and systems.

3. RSA is an effective solution for contexts with limited resources since it can be efficiently implemented in both hardware and software.

Disadvantages of RSA:

1. No PFS: As RSA does not offer PFS, previous communication sessions may also be compromised if a communication partner's private key is compromised.

2. Static key generation: Since RSA creates a unique fixed key for every session, it is more susceptible to attacks that depend on compromising a single key.

B. Explain how IVs are chosen in both CBC MAC and GCM in SSL/TLS application data protocol, attacks related to those settings in a real-world application, and their possible countermeasures.

Initialization vectors (IVs) are used in the SSL/TLS application data protocol's CBC-MAC (Cipher Block Chaining Message Authentication Code) and GCM (Galois/Counter Mode) encryption modes. IVs are used to make encryption more unpredictable and stop identical plaintext from producing an identical ciphertext.

CBC-MAC (Cipher Block Chaining Message Authentication Code)

The initialization vector (IV) value in CBC-MAC (Cipher Block Chaining Message Authentication Code) is frequently fixed at a value of all zeros. This constant value is frequently utilized because it has no impact on the MAC's security as long as the key is kept a secret.

Attacks: If the same key is used to generate the MACs for several messages, using a fixed IV could result in possible security problems. In this case, if an attacker gains access to the MACs created for each message, they can compute the XOR of the two messages, revealing information about the messages' contents.
Another attack that is related to IV is the BEAST (Browser Exploit Against SSL/TLS) Attack. This attack targets the SSL/TLS implementation of CBC mode. In this attack, an attacker can exploit the predictability of the IVs used in CBC mode to decrypt and obtain plaintext data.

Countermeasure: Utilise a distinct IV for each message as a countermeasure to reduce this vulnerability. In CBC-MAC, a common approach for generating a unique IV is to use a random number that has been chosen by the sender and receiver before the message exchange.

Another countermeasure is to Use TLS 1.1 or later, as these versions have built-in protection against BEAST by using explicit IVs, which are no longer predictable.

GCM (Galois/Counter Mode)

The IV should be different for each encryption operation carried out using the same encryption key for AES Galois Counter Mode (GCM) Cipher Suites for TLS.

The nonce format is used to ensure the IV's uniqueness. A fixed component and a counterpart exist for the nonce. The recommended length for the nonce is 12 octets, with the counterpart being 4 octets long. To provide uniqueness, the fixed component differs for each encryption device, while the counterpart begins at zero for the initial encryption and increases by one for each subsequent encryption.

Attacks: GCM Reuse Attack- This attack occurs when the same IV is used with the same key for GCM encryption, which can compromise the confidentiality and integrity of the encrypted data.

IV Prediction Attack-In this attack, the adversary can predict the IVs used in the encryption process, leading to the potential compromise of encrypted data. This attack can affect both CBC and GCM modes if the IV generation process is predictable.

Countermeasure: As advised in RFC 5288 and RFC 5116, make sure that distinct IVs are utilized for each encryption operation. Use an IV with a 4-octet counter and the suggested length of 12 octets, as per RFC 5116.To prevent key reuse, adopt effective key management practices.

In GCM, use a unique nonce for each encryption operation, as specified in RFC 5116.

C. Identify three different attacks on TLS/SSL in an application in that a user conducts his banking activities, and show the consequences, and countermeasures. You should include one attack which runs on HTTPS, i.e., HTTP over TLS.

1. **(BEAST) Attack:** The TLS/SSL protocols that use the CBC (Cipher Block Chaining) mode of encryption are vulnerable to the cryptographic attack known as the BEAST (Browser Exploit Against SSL/TLS) attack. This attack uses CBC mode's weakness to decrypt encrypted communications in their plaintext. The attacker injects packets into the TLS stream using MITM. They can then match the outcomes to those of the block they seek to decrypt, allowing them to guess the Initialization Vector (IV) used with the injected message.

   Consequence: A successful BEAST attack on a banking application could have disastrous results because it could provide the attacker access to sensitive data such as bank account credentials, personal information, and financial transaction information. Financial loss, identity theft, and other types of fraud could occur from this.

   Countermeasures: CBC mode can be disabled, and alternative encryption methods like GCM (Galois/Counter Mode), which are resistant to the BEAST attack, can be used as a substitute. Upgrading to safer protocols like TLS 1.3, which employs many encryption techniques and is not susceptible to the BEAST attack, is another defence.

2. **CRIME (Compression Ratio Info-leak Made Easy):** The compression feature in TLS/SSL protocols is the target of the cryptographic attack known as CRIME (Compression Ratio Info-leak Made Easy). Sensitive data sent between a user's web browser and a server in a banking application can be decrypted using this technique. The attacker injects specially crafted requests into the encrypted traffic, which allows them to infer the plaintext of the encrypted messages by observing the compression ratio.

   Consequence: In a banking application using TLS/SSL, an attacker may be able to intercept and decrypt sensitive data being transmitted between the user's web browser and the bank's server, such as bank account credentials, personal information, and financial transaction specifics, using the CRIME attack.

   Countermeasures: TLS/SSL implementations can disable the compression feature, which eliminates the vulnerability that the CRIME attack exploits. TLS 1.2 and later versions have implemented countermeasures to prevent the CRIME attack, such as using a different mechanism for compressing data.

3. **<u>(POODLE) Padding Oracle On Downgraded Legacy Encryption Attack (on HTTPS):</u>** This attack takes the use of a vulnerability in SSL 3.0, which is still supported by some websites. This is one attack that runs on HTTPS. It benefits from two elements. The first factor is that some servers and clients continue to offer SSL 3.0 in order to maintain compatibility and interoperability with older systems. The second element is an issue with block padding in SSL 3.0, which is a vulnerability. In this attack, the attacker injects malicious code and downgrades SSL/TLS to SSL 3.0 in order to steal sensitive data.

   Consequence: The attacker can steal sensitive information such as login credentials, session cookies, and other authentication tokens.

   Countermeasure: The bank needs to disable SSL 3.0 and only support TLS 1.2 or later in order to stop POODLE attacks. Furthermore, using HTTP Strict Transport Security (HSTS) can stop an attacker from lowering the SSL/TLS version. By instructing the browser to only communicate with websites via HTTPS, HSTS makes it more difficult for an attacker to degrade the connection.

D. Suppose Alice sends packets to Bob using TCP over IPsec. If the TCP acknowledgment from Bob is lost, then the TCP sender at Alice's side will assume the corresponding data packet was lost, and thus retransmit the packet. Will the retransmitted TCP packet be regarded as a replay packet by IPsec at Bob's side and be discarded? Explain your answer.

IPsec at Bob's end will not treat the retransmitted TCP packet as a replay packet and will not discard it. This is because of the anti-replay protection provided by IPsec. Under this strategy, IPsec uses a sliding window approach, where a range of sequence numbers is accepted as acceptable, and any packet with a sequence number within that range is not regarded as a replay packet. A packet's sequence number is verified when it is received to see if it falls within the range of acceptable sequence numbers. If it does, the packet is accepted as valid, and its sequence number is added to the sliding window.

IPsec merely maintains the integrity and confidentiality of the IP datagrams being carried, whereas TCP itself has its own system for handling packet loss and retransmission. The same TCP segment will therefore not be deemed a replay by IPsec even if it is retransmitted because it will have a separate IP datagram with a different sequence number.

E. In order to make TLS/SSL handshaking protocol more efficient, one of the approaches is to simplify TLS/SSL by restricting the key establishment to DHE. Provide a simplified handshaking protocol where certificate requests are always sent, and give an analysis of pros and cons for this approach. Find an example which exists in practical systems and comment their approach.

A simplified TLS/SSL handshaking protocol that restricts the key establishment to Diffie-Hellman Ephemeral (DHE) would have the following steps:

1. The client sends a ClientHello message, including a list of supported cipher suites that only includes DHE-based cipher suites.
2. The server responds with a ServerHello message, including the selected DHE-based cipher suite and the server's public key.
3. The server then sends its digital certificate to the client, which contains its public key.
4. The client verifies the server's certificate using a trusted Certificate Authority (CA).

5. The Diffie-Hellman parameters (prime number, generator, and the server's transient public key) are sent by the server in a "ServerKeyExchange" message.
6. The client creates its own ephemeral private and public keys, and in a "ClientKeyExchange" message, it sends its public key to the server.
7. The shared secret (pre-master secret key) is independently computed by the client and server using both parties' public keys and private keys.The master secret key is later used to generate symmetric encryption keys for data encryption and decryption after being derived from the pre-master secret key.
8. The "ChangeCipherSpec" messages between the client and server signal that they will now communicate using symmetric encryption keys.
9. The client and server exchange "Finished" messages to confirm the successful completion of the handshake process.

.

Analysis of Pros and Cons:

Pros:

1. Proven security: This strategy has undergone significant research and testing and is commonly utilized in secure communication protocols like SSL/TLS.
2. Offers strong authentication: By requesting the server's digital certificate, the communication partners are given strong authentication by knowing that the server is who they say it is.

Cons:

1. Needs trustworthy third parties: To issue and administer digital certificates, a trusted third party is needed, such as a Certificate Authority (CA). This raises the potential for vulnerability and adds another level of complication.
2. Subject to certificate authority compromise: Should the certificate authority get hacked, the protocol's entire level of security may be put at risk.

The Off-the-Record (OTR) protocol, a cryptographic protocol created for secure instant messaging, is illustration of a real-world system that employs a similar strategy. The Diffie-Hellman key exchange is used by the OTR protocol to create a shared secret between the communication participants. The keys are ephemeral, which means they are deleted at the end of each session.
The OTR protocol doesn't rely on digital certificates or other trusted third parties for authentication, similar to the simplified TLS/SSL handshaking protocol using DHE-only. To confirm the identity of the communication participants, the protocol instead makes use of an authentication technique based on shared secrets and a cryptographic signature scheme.

The OTR protocol's simplicity, which does away with the requirement for digital certificates and trusted third parties to establish a secure communication channel, is one of its benefits. However, the OTR protocol is susceptible to man-in-the-middle attacks if the authentication process is compromised, just like the DHE-only TLS/SSL approach.

To lessen this risk, the OTR protocol has features like forward secrecy, which makes sure that even if one communication session is compromised, subsequent sessions are still secure, and deniability, which enables communication partners to deny the existence of a conversation or the content of a message. Notwithstanding the lack of digital certificates and reliable third parties, these elements contribute to the protocol's increased security.

In summary, while a simplified handshaking protocol that always sends certificate requests provides strong authentication and confidentiality, it also comes with increased complexity and potential vulnerabilities. The trade-off between security and complexity should be carefully considered when designing a communication protocol.

REFERENCE:

1. **Communication System Security by Lidong Chen and Guang Gong**
2. **https://www.acunetix.com/blog/articles/tls-vulnerabilities-attacks-final-part/**
3. **https://docs.openstack.org/security-guide/secure-communication/introduction-to-ssl-and-tls.html**
4. **https://www.internetsociety.org/deploy360/tls/basics/#:~:text=TLS%20is%20a%20cryptographic%20protocol,a%20secure%20session%20is%20established.**
5. **https://www.cloudflare.com/learning/ssl/what-happens-in-a-tls-handshake/**
6. **https://www.expressvpn.com/blog/what-is-off-the-record-messaging-otr/**