# Defensive Programming ~ Maman 15 Threat Analysis

Ariel Cohen ~ ID 329599187

I have used the format provided as an example on the course web site:
https://opal.openu.ac.il/pluginfile.php/8577631/mod_resource/content/1/risks-table-example.pdf

## Weakness 1: Static Initialization Vector (IV) in AES-CBC

| | |
|---|---|
| **Threat** | Plaintext Information Leakage |
| **Affected Component** | Symmetric Message Encryption Protocol |
| **Module Details** | Client-side encryption logic for all messages encrypted with a symmetric key. |
| **Vulnerability Class** | Insecure Cryptographic Implementation / Information Disclosure |
| **Description** | The protocol specification explicitly mandates the use of a static, all-zero Initialization Vector (IV) for all AES-CBC encryption operations. In CBC mode, the IV must be unique and unpredictable for each message encrypted with the same key to ensure confidentiality. Using a fixed IV means that if two different messages start with the same block of plaintext, their first blocks of ciphertext will be identical. |
| **Result** | An attacker eavesdropping on the network can compare the first 16 bytes of different encrypted messages. If the blocks match, the attacker knows the first 16 bytes of the original plaintext are identical. This leaks significant information about message content without requiring decryption and undermines the semantic security of the encryption. |
| **Prerequisites** | An attacker must be able to monitor and capture network traffic between the clients and the server. |
| **Business Impact** | This flaw compromises user privacy and weakens the "end-to-end encryption" promise of the application. It allows an attacker to identify patterns in communication, which could be used to infer context or behavior. |

| | |
|---|---|
| **Proposed Remediation** | For each message, generate a new, cryptographically random 16-byte IV. Prepend this unique IV to the ciphertext before sending. The recipient will then read the first 16 bytes of the payload as the IV and use it for decryption. |
| **Risk** | **High.** The vulnerability is guaranteed to be present as it is part of the protocol specification. It is easily discoverable by anyone analyzing the protocol. |

## Weakness 2: Unauthenticated Public Key Exchange

| | |
|---|---|
| **Threat** | Man-in-the-Middle (MITM) Attack |
| **Affected Component** | Public Key Distribution and Key Exchange Protocol |
| **Module Details** | The protocol flow for "Request for public key" (Request Code 1102) and the subsequent use of that key to encrypt and send a symmetric key (Message Type 2). |
| **Vulnerability Class** | Broken Authentication |
| **Description** | When Client A requests Client B's public key, the server retrieves the key from its records and sends it to A. The protocol provides no mechanism for Client A to verify that the key it received actually belongs to Client B. The server is treated as an untrusted party for message content, but it is implicitly trusted to be an honest broker for public keys. |
| **Result** | A malicious server or an active attacker on the network can intercept the public key request and substitute their own public key. Client A will then unknowingly encrypt the new symmetric key with the attacker's public key. The attacker can decrypt this symmetric key, re-encrypt it with Client B's real public key, and forward it. The attacker now possesses the session key and can decrypt, read, modify, and inject messages between A and B, completely defeating the end-to-end encryption. |
| **Prerequisites** | An active attacker who can intercept and modify traffic between a client and the server, or a compromised/malicious server. |

| | |
|---|---|
| **Business Impact** | A total loss of confidentiality and integrity for all user communications. This is a critical failure of the security model that invalidates the primary promise of the application. |
| **Proposed Remediation** | Implement a key verification mechanism: the client application should cache a user's public key the first time it is retrieved. In all subsequent communications, if the public key received from the server does not match the cached key, the application must warn the user of a potential security risk. For higher security, applications often display key "fingerprints" that users can verify through a secondary, trusted channel (e.g., over the phone). |
| **Risk** | **Critical.** This vulnerability fundamentally breaks the entire security premise of the application and is a classic, well-known attack vector. |

### Weakness 3: Lack of Message Authentication and Integrity

| | |
|---|---|
| **Threat** | Active Message Tampering (Ciphertext Malleability) |
| **Affected Component** | Symmetric Message Encryption Protocol |
| **Module Details** | All encrypted messages sent between clients (Message Types 2, 3, and 4). |
| **Vulnerability Class** | Broken Integrity Protection / Cryptographic Malleability |
| **Description** | The protocol uses AES-CBC for confidentiality but fails to pair it with a Message Authentication Code (MAC), such as an HMAC, to ensure message integrity. Encryption by itself does not prevent an attacker from modifying the message. CBC mode is "malleable," meaning an attacker can make specific, targeted bit-flips in a block of ciphertext that will result in predictable bit-flips in the corresponding block of decrypted plaintext, all without knowing the encryption key. |
| **Result** | An active attacker can intercept an encrypted message in transit and alter its content without being detected by the recipient. For example, an attacker could change "I will pay you $100" to "I will pay you $900" by modifying the ciphertext. The recipient would decrypt the modified message and believe it was authentic. |

| | |
|---|---|
| **Prerequisites** | An active attacker who can intercept and modify traffic between the server and clients. |
| **Business Impact** | Loss of user trust and data integrity. The inability to guarantee that a received message is exactly what the sender wrote makes the application unreliable and dangerous for any form of sensitive communication. |
| **Proposed Remediation** | Implement an "Encrypt-then-MAC" scheme. After encrypting the plaintext, compute an HMAC (e.g., HMAC-SHA256) over the ciphertext (including the IV) and append it to the message. The recipient must verify the HMAC before attempting decryption. If the verification fails, the message must be discarded. |
| **Risk** | **Critical.** This allows an attacker to actively and silently manipulate conversations, which can have severe consequences depending on the content. |

### Weakness 4: Lack of Forward Secrecy

| | |
|---|---|
| **Threat** | Retrospective Decryption of Past Communications |
| **Affected Component** | Key Exchange Protocol |
| **Module Details** | The process of encrypting a symmetric session key using the recipient's long-term RSA public key (Message Type 2). |
| **Vulnerability Class** | Insecure Key Management |
| **Description** | The confidentiality of every conversation relies on a symmetric session key, which is itself protected only by the long-term RSA keys of the participants. If a user's long-term private key (from their `my.info` file) is ever compromised, an attacker who has been recording that user's past network traffic can use the stolen key to go back and decrypt every symmetric key exchange they have captured. |
| **Result** | A single compromise of a user's long-term private key leads to the compromise of all past conversations that have been recorded by an attacker. The security of historical data is not protected against future events. |

| | |
|---|---|
| **Prerequisites** | An attacker must first gain access to a user's long-term private key (e.g., by stealing their `my.info` file) **and** have previously recorded their encrypted network traffic. |
| **Business Impact** | A catastrophic and permanent loss of historical privacy for the affected user. This is a significant weakness for any application that claims to provide secure communication, as users expect past conversations to remain private. |
| **Proposed Remediation** | Implement a key agreement protocol that provides **Forward Secrecy**. Instead of using long-term keys to encrypt session keys, use them only to sign and authenticate an ephemeral key exchange. For each new session, clients generate a new, temporary (ephemeral) key pair. They perform a key agreement to derive a shared secret and then discard the temporary keys. This ensures that a future compromise of the long-term keys cannot reveal past session keys. |
| **Risk** | **High.** While exploiting it requires two steps (traffic recording and key theft), the impact is the total and irreversible compromise of all past data. |

### Weakness 5: Unauthenticated Resource Consumption (Denial of Service)

| | |
|---|---|
| **Threat** | Denial of Service (DoS) / Resource Exhaustion |
| **Affected Component** | Server-side Message Handling and Storage |
| **Module Details** | The server's logic for handling "Send Message" requests (Request Code 1103) and storing the associated payloads in memory or the database. |
| **Vulnerability Class** | Uncontrolled Resource Consumption |
| **Description** | The protocol does not specify any limits on the size of a message's payload (`Content Size` is a 4-byte integer, allowing for messages up to 4GB). Furthermore, there are no restrictions on the number of messages a single client can send to another. The server is obligated to accept and store any message from any registered client to any other registered client. |

| | |
|---|---|
| **Result** | A malicious or compromised client could intentionally flood the server with a high volume of messages or send a few extremely large messages (e.g., multi-gigabyte payloads). This would rapidly consume the server's available RAM (if storing in memory) or disk space (if using the database), causing the server to slow down, become unresponsive, or crash entirely. This would result in a Denial of Service for all legitimate users of the application. |
| **Prerequisites** | An attacker needs to have a single valid, registered client account. |
| **Business Impact** | Complete service unavailability for all users. A prolonged DoS attack could damage the application's reputation and reliability. |
| **Proposed Remediation** | Implement server-side validation and rate limiting: 1. **Size Limits:** Enforce a reasonable maximum `Content Size` for all incoming messages (e.g., 10 MB for files, 10 KB for text). Reject any request that exceeds this limit. 2. **Rate Limiting:** Implement a limit on the number of messages a single client can send within a given time window (e.g., no more than 100 messages per minute). 3. **Queue Quotas:** Limit the total number or aggregate size of pending messages that can be stored for any single offline user. |
| **Risk** | **High.** This is a straightforward and effective way for a low-level attacker to disrupt the entire service. |

## Weakness 6: Unencrypted Storage of Sensitive Credentials

| | |
|---|---|
| **Threat** | Credential Theft and Retrospective Decryption |
| **Affected Component** | Client-side Credential Storage |
| **Module Details** | The `my.info` file, which stores the user's username, UUID, and long-term private RSA key. |
| **Vulnerability Class** | Insecure Storage of Sensitive Data |
| **Description** | The protocol requires the client to store its long-term private RSA key in the `my.info` file in Base64 format. Base64 is an encoding scheme, not an encryption scheme. This means the private key, which is the ultimate root |

| | |
|---|---|
| | of trust for the user's identity and the security of their past conversations, is sitting unencrypted on the user's disk. |
| **Result** | If an attacker gains access to a user's computer (either physically or through malware), they can simply copy the `my.info` file. With this file, the attacker can: 1. **Impersonate the User:** The attacker can use the private key to authenticate as the user and decrypt any new incoming messages. 2. **Decrypt Past Conversations:** As described in the "Lack of Forward Secrecy" weakness, the attacker can use this stolen key to decrypt any previously recorded conversations. |
| **Prerequisites** | An attacker must gain local or remote file system access to the user's computer where the client application is stored. |
| **Business Impact** | A complete and permanent compromise of a user's identity, privacy, and data security. This undermines user trust and could have severe personal or legal consequences depending on the sensitivity of the compromised conversations. |
| **Proposed Remediation** | The private key must never be stored in plaintext. At a minimum, the `my.info` file should be encrypted with a key derived from a user-provided password. When the client application starts, it would ask the user for their password, use it to decrypt the private key into memory for the duration of the session, and clear it from memory when the application closes. This ensures that the key is only accessible when the user is actively using the application. |
| **Risk** | **Critical.** While it requires access to the user's machine, the impact is a total security failure for the affected user. Storing unencrypted private keys is a severe violation of security best practices. |

## Weakness 7: Replay Attacks

| | |
|---|---|
| **Threat** | Malicious Re-transmission of Valid Messages |
| **Affected Component** | Server-to-Client Message Delivery Protocol |
| **Module Details** | The protocol for pulling messages (Request Code 1104) and the structure of the messages returned to the client. |
| **Vulnerability Class** | Missing Protection Against Replay Attacks |

| | |
|---|---|
| **Description** | The messages sent between clients do not contain any unique, single-use identifier, such as a nonce or a timestamp, that would prevent them from being resent. While the server deletes messages after they are successfully pulled, an attacker on the network could capture a valid message payload and re-transmit it to the recipient at a later time. |
| **Result** | An attacker could capture a legitimate message and replay it. For example, if a user sends a message containing a symmetric key (Message Type 2), an attacker could capture it and replay it later, potentially forcing the recipient to revert to an old, possibly compromised, session key. Replaying a text or file message could be used to confuse users or cause unintended actions. |
| **Prerequisites** | An active attacker who can capture and inject traffic on the network. |
| **Business Impact** | This vulnerability can undermine the integrity of a conversation, cause confusion, and potentially be used as a stepping stone for more complex attacks (e.g., forcing the reuse of a compromised key). It reduces the overall robustness and trustworthiness of the communication channel. |
| **Proposed Remediation** | To prevent replay attacks, each message should include a value that makes it unique. Common solutions include: 1. **Message Counter:** Each client maintains a separate, monotonically increasing counter for every other user they communicate with. Each message includes the current counter value. The recipient rejects any message with a counter value less than or equal to the last one they received from that sender. 2. **Nonce:** Include a large, random, single-use number in each message. The recipient would need to keep a list of recently seen nonces to reject duplicates. |
| **Risk** | **Medium.** The direct impact is less severe than a full MITM attack, but it represents a significant gap in the protocol's integrity and could be exploited in targeted scenarios. |