Red Team Report: Attacker Perspective on Secure Messaging Client

BY: TUMUSIIME STEPHEN AND TUMWESIGE JONATHAN

Date: October 21, 2025

1 Executive Summary

Team: Red Team (Attack Simulation)
Target System: Secure Messaging Client (Python implementation using RSA-2048 for key exchange and AES-256-GCM for message encryption)
Objective: Identify and exploit vulnerabilities to compromise confidentiality, integrity, or availability of communications.

The red team simulated adversarial attacks on the messaging client. The system employs standard cryptographic primitives (RSA-OAEP for session key exchange and AES-GCM for encryption), but lacks robust authentication, forward secrecy, and secure key management. Key findings include successful man-in-the-middle (MITM) attacks via public key substitution, private key extraction from unencrypted files, and persistent session key exposure leading to decryption of all messages in a conversation. No direct cryptanalytic breaks were achieved against the primitives themselves, but implementation flaws enable practical breaches. Overall, the system is vulnerable to targeted attacks, especially in scenarios involving key sharing or file compromise.

2 Scope and Methodology

- **Attack Vectors Tested**: Network interception (MITM), file system access, side-channel leaks (e.g., debug output), session hijacking, and cryptanalytic attempts.
- **Tools Used**: Custom Python scripts to simulate key substitution and decryption; analysis of exported files; no advanced hardware for side-channels.
- **Assumptions**: Attacker has access to shared files (e.g., public keys or message packages) or can intercept exports; no initial network control but can exploit unauthenticated key exchanges.
- **Phases**: Reconnaissance (code review), exploitation (targeted attacks), and post-exploitation (data recovery).

3 Vulnerabilities Identified and Exploits

3.1 1. Lack of Public Key Authentication (MITM Vulnerability)

Description: Public keys are exported and imported without digital signatures or certificates. The `export_public_key` and `import_public_key` methods rely on unverified PEM files, allowing an attacker to impersonate users by substituting malicious keys.

Exploit Scenario:

- Attacker intercepts or replaces a public key file (e.g., `alice_public.json`) with their own key while preserving the username.
- Victim (e.g., Bob) imports the fake key, establishing a session with the attacker instead of Alice.
- Attacker decrypts messages intended for Alice and can relay/replay modified content.

Proof-of-Concept:

- Generate fake user: `fake_alice = User('FakeAlice')`.
- Export fake key with real username: Modify JSON to use 'Alice' as username but fake PEM.
- Import into victim's client: Session key exchanged with fake key, allowing attacker to decrypt all subsequent messages.

Impact: Full compromise of conversation confidentiality and integrity. Probability: High (if keys are shared via insecure channels like email).

3.2 2. Absence of Perfect Forward Secrecy (PFS)

Description: Session keys are generated once per peer pair and reused indefinitely (`sessions` dictionary). No ephemeral key exchange (e.g., Diffie-Hellman) is implemented, so compromising a private key exposes all past and future messages.

Exploit Scenario:

- Attacker obtains a user's private key (see Vulnerability 3).
- Decrypt stored session keys from message packages or intercepted exchanges.
- Use the session key to decrypt all messages in the conversation history.

Proof-of-Concept:

- Simulate compromise: Extract private key from unencrypted file.
- Decrypt session key: `compromised_key.decrypt(encrypted_session_key, ...)`.
- Decrypt messages: Replay AES-GCM decryption on stored ciphertexts.

Impact: Retroactive decryption of entire conversation histories. This is a common flaw in hybrid systems without PFS, as noted in cryptographic literature (e.g., weak links in RSA-AES hybrids where session keys persist).

3.3 3. Insecure Key Storage and Export

Description: Private keys are saved in plaintext JSON files (`save_keys_to_file`) without encryption or password protection. Message packages include encrypted session keys but are stored unencrypted overall.

Exploit Scenario:

- Attacker gains file system access (e.g., via malware or shared storage).
- Extract private keys directly: Parse JSON for `private_key_pem`.
- Use extracted key to decrypt session keys and messages.

Proof-of-Concept:

- Save keys: `user.save_keys_to_file('keys.json')`.
- Attacker reads file: Load PEM and use for decryption.

- For message packages: Extract `encrypted_session_key` and decrypt with stolen private key.

Impact: Total system compromise if any device is breached. High risk in multi-user or cloud environments.

3.4 4. Debug Output Leaks Sensitive Information

Description: Extensive print statements expose hex dumps of keys, IVs, ciphertexts, and tags (e.g., in `establish_session`, `encrypt_message`).

Exploit Scenario:

- Attacker accesses console output (e.g., via shoulder surfing, log files, or debug mode).
- Collect partial key hex (e.g., `session_key.hex()[:32]`) and brute-force the rest if truncated poorly.

Proof-of-Concept: Run in verbose mode; capture output showing key prefixes, enabling targeted attacks.

Impact: Partial key leakage accelerates brute-force or social engineering attacks.

3.5 5. No Message Signing or Integrity Beyond Encryption

Description: AES-GCM provides authentication, but there's no separate signing of messages or keys, allowing tampering if GCM tag is bypassed (unlikely but possible in weak implementations).

Exploit Scenario: Modify ciphertext and tag if attacker has session key; no non-repudiation.

Impact: Low for confidentiality but medium for integrity in replay attacks.

3.6 6. User Impersonation and Lack of Access Controls

Description: Anyone can create users with any username; no verification or uniqueness beyond local check.

Exploit Scenario: Create duplicate users to confuse sessions or intercept imports.

Impact: Facilitates social engineering.

4 Attack Success Metrics

- **Confidentiality Breach**: Achieved (decrypted messages via MITM and key compromise).
- **Integrity Breach**: Partial (message tampering possible post-compromise).
- **Availability**: Not targeted, but DoS via key flooding feasible.
- **Exploits Demonstrated**: 4 successful PoCs (MITM, key extraction, session decryption, leak capture).

5 Recommendations for Future Engagements

- Focus on network-level attacks if client is deployed over sockets.
- Test for side-channel vulnerabilities (e.g., timing in RSA decryption).

6 Conclusion

The system's core crypto is sound, but implementation gaps make it insecure for real-world use. Red team recommends treating all communications as potentially compromised in unauthenticated environments.