# Security Analysis of File Transfer Protocol – Question 2

## 1. Protocol Overview Vulnerabilities

### 1.1 Zero IV in AES-CBC Mode

**Threat**: Deterministic Encryption

**Affected Component**: Encryption Module (Client and Server)

**Module Details**: Client.cpp (encryptFileContent method), server.py (decrypt\_file function)

**Vulnerability Class**: Cryptographic Implementation

**Description**: The protocol uses a zero IV (Initialization Vector) for AES-CBC mode encryption. This practice makes the encryption deterministic, meaning identical plaintext blocks will produce identical ciphertext blocks, potentially leaking information about the file contents.

**Result**: An attacker can identify patterns in the encrypted data and potentially reconstruct parts of the file content.

**Prerequisites**: Access to multiple encrypted files or repeated content within the same file.

**Business Impact**: Confidentiality of transferred files could be compromised. **Proposed Remediation**:

- Generate a random IV for each encryption operation

- Include the IV in the encrypted message header

- Modify the protocol to accommodate IV transmission

**Risk Assessment**:

- Damage Potential: 7

- Reproducibility: 9

- Exploitability: 6

- Affected Users: 8

- Discoverability: 7

- Overall: 7.4

### 1.2 Missing Authentication in Reconnection

**Threat**: Session Hijacking

**Affected Component**: Client Authentication System

**Module Details**: server.py (handle\_reconnection function)

**Vulnerability Class**: Authentication Bypass

**Description**: The reconnection mechanism only verifies the client ID and name, without any additional authentication factors or session tokens.

**Result**: An attacker who captures a client ID can potentially impersonate the legitimate user during reconnection.

**Prerequisites**: Ability to intercept network traffic and obtain client ID.

**Business Impact**: Unauthorized access to user accounts and sensitive files. **Proposed Remediation**:

- Implement session tokens with limited lifetime

- Add challenge-response authentication during reconnection

- Include digital signatures in reconnection requests

**Risk Assessment**:

- Damage Potential: 8

- Reproducibility: 7

- Exploitability: 6

- Affected Users: 8

- Discoverability: 5

- Overall: 6.8

### 1.3 Weak RSA Key Size

**Threat**: Key Compromise

**Affected Component**: Key Exchange System

**Module Details**: Client.cpp (generateRSAKeys method)

**Vulnerability Class**: Cryptographic Implementation

**Description**: The protocol uses 1024-bit RSA keys, which are considered weak by modern standards.

**Result**: The RSA keys could potentially be broken through computational attacks. **Prerequisites**: Significant computational resources.

**Business Impact**: Compromise of encrypted communications and file content. **Proposed Remediation**:

- Increase RSA key size to minimum 2048 bits

- Implement regular key rotation

- Consider using modern alternatives like Ed25519

**Risk Assessment**:

- Damage Potential: 8

- Reproducibility: 4

- Exploitability: 3

- Affected Users: 9

- Discoverability: 7

- Overall: 6.2

### 1.4 Lack of File Integrity Verification Before Processing

**Threat**: Malicious File Upload

**Affected Component**: File Processing System

**Module Details**: server.py (handle\_file\_transfer function)

**Vulnerability Class**: Input Validation

**Description**: The server processes the entire file before verifying its integrity, potentially allowing resource exhaustion attacks.

**Result**: Attackers can upload malformed files that consume excessive server resources.

**Prerequisites**: Valid client credentials.

**Business Impact**: Potential denial of service and server resource exhaustion. **Proposed Remediation**:

- Implement file validation before full processing

- Add file size limits

- Stream processing with early validation

**Risk Assessment**:

- Damage Potential: 6

- Reproducibility: 9

- Exploitability: 8

- Affected Users: 7

- Discoverability: 6

- Overall: 7.2

### 1.5 Unprotected Client Information Storage

**Threat**: Information Disclosure

**Affected Component**: Client Storage System

**Module Details**: Client.cpp (storeClientInfo, storePrivateKey methods) **Vulnerability Class**: Information Exposure

**Description**: Client information and private keys are stored in plaintext files (me.info, priv.key) without encryption or access controls.

**Result**: Local attackers can easily access sensitive client credentials and keys. **Prerequisites**: Access to client’s file system.

**Business Impact**: Compromise of client credentials and encrypted communications. **Proposed Remediation**:

- Encrypt stored credentials and keys

- Implement secure storage using OS keychain/credential manager

- Add access controls to sensitive files

**Risk Assessment**:

- Damage Potential: 7

- Reproducibility: 8

- Exploitability: 6

- Affected Users: 7

- Discoverability: 8

- Overall: 7.2

### 1.6 Missing Rate Limiting

**Threat**: Brute Force Attacks

**Affected Component**: Authentication System

**Module Details**: server.py (handle\_client function)

**Vulnerability Class**: Authentication

**Description**: No rate limiting on connection attempts or authentication requests. **Result**: Attackers can perform unlimited connection attempts and authentication tries.

**Prerequisites**: Network access to server.

**Business Impact**: Server resource exhaustion and potential credential compromise. **Proposed Remediation**:

- Implement rate limiting per IP address

- Add exponential backoff for failed attempts

- Monitor and log suspicious activity

**Risk Assessment**:

- Damage Potential: 6

- Reproducibility: 9

- Exploitability: 8

- Affected Users: 6

- Discoverability: 7

- Overall: 7.2

## 2. Additional Security Concerns

1. **No Perfect Forward Secrecy**: The protocol doesn’t implement perfect forward secrecy, meaning if a private key is compromised, all previous communications could be decrypted.
2. **Lack of Certificate Authority**: The protocol doesn’t use any trusted third party to verify identities, making it vulnerable to man-in-the-middle attacks.
3. **No Message Authentication Code**: The protocol relies solely on CRC for integrity checking, which is not cryptographically secure and can be manipulated.

## 3. Recommendations for Improvement

1. **Implement Modern Cryptographic Practices**:
   * Use TLS 1.3 for transport security
   * Implement authenticated encryption (AES-GCM)
   * Add HMAC for message authentication
2. **Enhance Authentication**:
   * Implement mutual authentication
   * Add session management
   * Use secure token-based authentication
3. **Improve Security Controls**:
   * Add comprehensive logging
   * Implement intrusion detection
   * Add file type validation
   * Implement proper error handling without information disclosure
4. **Strengthen Protocol Design**:
   * Add version negotiation
   * Implement proper key rotation
   * Add support for perfect forward secrecy
   * Include protocol upgrade mechanisms

## Conclusion

The current protocol implementation has several significant security vulnerabilities that could lead to unauthorized access, data disclosure, and system compromise. While the basic cryptographic building blocks are present, the implementation details and security controls need substantial improvement to meet modern security standards.

The most critical issues are the use of a zero IV in CBC mode, weak RSA key size, and lack of proper authentication mechanisms. Implementing the suggested remediation measures would significantly improve the security posture of the system.