Research vulnerability report | Protocol used in the project

Name: Raz Dahan

Introduction

This report provides an in-depth analysis of five critical vulnerabilities identified in the implemented file transfer protocol. These vulnerabilities, if exploited, could compromise the security, integrity, and confidentiality of the system. The vulnerabilities discussed are:

1.Fixed IV in AES Encryption

2 .Lack of Perfect Forward Secrecy

3. Insufficient Key Size

4. Lack of Server Authentication

5. Vulnerability to Replay Attacks

Each vulnerability will be examined in detail, including its nature, potential impact, and recommended mitigation strategies.

Detailed Analysis of Vulnerabilities

A. Fixed IV in AES Encryption

The protocol uses a fixed, zeroed Initialization Vector (IV) for AES encryption in CBC mode. This practice severely undermines the security of the encryption process.

In the 'encrypt_aes' function of the Crypto class, the IV is set as follows:

std::vector<uint8_t> iv(CryptoPP::AES::BLOCKSIZE, 0);

This creates an IV filled with zeros for every encryption operation.

Potential Impacts

1. Pattern Recognition: Using a fixed IV allows patterns in the plaintext to become apparent in the ciphertext, especially when encrypting similar data multiple times.
2. Chosen-Plaintext Attacks: An attacker can exploit this to perform chosen-plaintext attacks, potentially decrypting portions of the ciphertext without knowing the key.
3. Information Leakage: It becomes possible to infer relationships between different encrypted messages, leading to information leakage.

Mitigation Strategies

1. Implement secure random IV generation for each encryption operation:

```
CryptoPP::AutoSeededRandomPool rng;

byte iv[CryptoPP::AES::BLOCKSIZE];

rng.GenerateBlock(iv, sizeof(iv));
```

2. Prepend the IV to the ciphertext and retrieve it during decryption.
3. Ensure the IV is generated using a cryptographically secure random number generator.

B. Lack of Forward Secrecy

The protocol uses long-term RSA keys for key exchange without implementing perfect forward secrecy (PFS).

Potential Attack: If an attacker obtains the private RSA key, they can decrypt all past communications.

Suggestion: Implement Diffie-Hellman key exchange or use Ephemeral RSA keys to provide perfect forward secrecy.

C. Insufficient Key Size

The RSA key size is set to 1024 bits, which is considered weak by modern standards.

In the 'generate_rsa_keys' function:

```
keys.private_key.Initialize(rng, 1024);
```

This generates a 1024-bit RSA key pair.

With advancements in computing power (like quantum computing), 1024-bit RSA keys are vulnerable to factorization attacks. If the key is factored, an attacker can decrypt all messages and impersonate either party. Even if not immediately practical, this becomes a more significant threat over time as computational power increases.

In order to make the breach harder, Increase the RSA key size to at least 2048 bits, preferably 4096 bits for long-term security.

D. Lack of Server Authentication

The protocol doesn't include a mechanism for the client to authenticate the server.

The current connection process doesn't verify the server's identity before establishing a secure connection:

```
tcp::resolver::results_type endpoints = resolver.resolve(ip, std::to_string(port));

boost::asio::connect(m_socket, endpoints);
```

Potential Impacts

1. Man-in-the-Middle Attacks: An attacker could impersonate the legitimate server, intercepting and potentially modifying all communications.

2. Data Confidentiality Breach: Sensitive information could be captured by connecting to a malicious server.

3. Malicious Payload Injection: An attacker could send malicious data or commands to the client.

Mitigation Strategies

1. Implement server authentication using digital certificates:

```
  SSL_CTX* ctx = SSL_CTX_new(TLS_client_method());

  SSL_CTX_set_verify(ctx, SSL_VERIFY_PEER, NULL);

  SSL_CTX_load_verify_locations(ctx, "path_to_CA_cert.pem", NULL);


  SSL* ssl = SSL_new(ctx);

  SSL_set_fd(ssl, socket_fd);


  if (SSL_connect(ssl) != 1)}
 //       Handle error - server certificate not verified
```

2. Implement certificate pinning on the client side to prevent acceptance of fraudulent certificates.

H. Vulnerability to Replay Attacks

The protocol doesn't include mechanisms to prevent replay attacks, such as timestamps or nonces.

The current message structure doesn't include any elements to ensure message freshness:

```
std::vector<uint8_t> Protocol::create_request(const std::vector<uint8_t>& client_id, uint8_t version, uint16_t code, const std::vector<uint8_t>& payload)}
 ... //     creating message without any timestamp or nonce
```

Potential Impacts

1. Unauthorized Actions: An attacker can capture and replay valid messages, potentially repeating sensitive operations.

2. Session Hijacking: Replay of authentication messages could lead to unauthorized access
3. Data Manipulation: Replaying file transfer commands could result in unintended file operations.

Mitigation Strategies

Implement nonces in each message:

```
  struct Message {
      std::vector<uint8_t> client_id;
      uint8_t version;
      uint16_t code;
      uint64_t nonce;
      std::vector<uint8_t> payload;
 ;}

  uint64_t generate_nonce () {
      static std::random_device rd;
      static std::mt19937_64 gen(rd());
      static std::uniform_int_distribution<uint64_t> dis;
      return dis(gen);
  ;}
```

Conclusion

The vulnerabilities identified in this report represent security risks to the implemented file transfer protocol. Addressing these issues is crucial to ensure the confidentiality, integrity, and authenticity of the data being transferred.

By implementing these changes that suggested throughout the article, the overall security posture of the protocol will be improved, providing stronger protection against various attack vectors and ensuring long-term data security

Threat Attacker may predict patterns in encrypted data or perform chosen-plaintext attacks.

Affected component Crypto class

Module details encrypt_aes function

Vulnerability class CWE-329: Not Using a Random IV with CBC Mode

Description The encrypt_aes function uses a fixed, zeroed Initialization Vector (IV) for AES encryption in CBC mode. This allows an attacker to identify patterns in encrypted data, especially when the same plaintext is encrypted multiple times. It can lead to information leakage and make the encryption vulnerable to chosen-plaintext attacks .

Result Compromised confidentiality of encrypted data. Potential for decryption of portions of the ciphertext or inference of plaintext patterns .

Prerequisites Attacker needs access to multiple encrypted messages or ability to perform chosen-plaintext attacks. No need for key knowledge .

Proposed remediation Implement secure random IV generation for each encryption operation. Prepend the IV to the ciphertext for use in decryption. Use a cryptographically secure random number generator for IV creation .

Risk Damage potential: 3

Reproducibility: 3

Exploitability: 2

Affected users: 3

Discoverability: 2

Overall: High .

Threat Attacker may impersonate the server, leading to man-in-the-middle attacks.

Affected component Client-server communication protocol

Module details Client connection and key exchange process

Vulnerability class CWE-287: Improper Authentication

Description The protocol doesn't include a mechanism for the client to authenticate the server. This omission allows an attacker to potentially impersonate the legitimate server. Without server authentication, a malicious actor could intercept the connection and act as a man-in-the-middle, potentially capturing sensitive data or injecting malicious content .

Result Potential for complete compromise of communication confidentiality and integrity. Attacker could capture sensitive data, modify transmissions, or inject malicious payloads .

Prerequisites Attacker needs to be in a position to intercept network traffic between client and server (e.g., on the same network or controlling a router on the communication path).

Proposed remediation Implement server authentication using digital certificates or a pre-shared server public key. Use a trusted Certificate Authority for server certificates. Implement certificate pinning on the client side to prevent acceptance of fraudulent certificates. Consider implementing mutual authentication where both client and server authenticate each other .

Risk Damage potential: 3

Reproducibility: 3

Exploitability: 2

Affected users: 3

Discoverability: 2

Overall: High