Dakota Fisher
Gil Popilski
December 8, 2019

CS6349 Programming Project Report

This document describes the final implementation of the programming project. It is similar to the milestone, but the protocol has been minimized and simplified wherever possible to remove redundancy, to allow the client to control the protocol in its entirety and to simplify message parsing. Further, the challenges have been augmented with a salt sent by the client and the minimum length limit removed.

In addition, a bug in the software was noted to have interesting security implications for the protocol (see “Exceptional Session Messages”). The protocol has been modified to make the security issues that result from these implications impossible to manifest in correct programs.

Encryption

Encryption is per-message CTR and is denoted as E(K,CTR,M) where K is the session key, CTR is the message counter and M is the message content. M must have a length of at most 64 bytes (256 bits). The formula for encryption is:

$E(K,CTR,M) = SHA(K|CTR) \oplus M$

Unused bytes are discarded.

Session management

Unless specified otherwise, session management messages are unencrypted.

Requesting server key

- To request the server key, the client sends ASCII “RKEY“ (requesting the key is in general optional, but for the test program implemented is required)
- To send the server key, the server sends “SKEY” followed by the key and then terminated with “SKEY.END”.

Session initiation

- To request a server session, the client sends a message consisting of:
    - ASCII “INIT”

    - a randomly-generated 16-byte (256-bit) session key K encrypted with the server's public key.
- To confirm the session, the server sends a message consisting of:
    - ASCII "ENCR"
    - An 8-byte random number CTR
    - The encrypted message E(K,CTR,"CONF")

CTR in the confirmation represents the initial counter variable.

SECURITY ADVISORY: Exceptional session messages

In the initial milestone, there were some exceptional session messages for synchronizing CTR. During implementation, a bug in this code was noticed that would allow for a sophisticated attacker to perform session hi-jacking. If the counter is reset to the value of CTR on every packet and the attacker can guess the plaintext. It was also noted that programs which had the incorrect behavior were significantly easier to implement and could be implemented without ever knowing the correct behavior. Therefore, such programs would be very likely to appear in-the-wild if this were a real protocol.

For these security reasons, it was decided to remove this particular class of message.

As an alternative, this is an advisory to the same effect:

ADVISORY: IT IS AN ERROR TO CONTINUE THE SESSION AFTER RECEIVING AN INCORRECT CTR. THE CORRECT WAY TO CONTINUE AFTER THE CTR IS DESYNCHRONIZED IS TO END THE CURRENT SESSION AND START A NEW ONE. FAILURE TO FOLLOW THIS ADVISORY MAY LEAD TO SESSION HI-JACKING

For completeness, struck through is what we would implement:

- ~~The client can request a "reset" consisting of:~~
    - ~~ASCII "RRST"~~
    - ~~The current value of CTR as understood by the sender of the reset request~~
- ~~After receiving a "reset" request, the server must sync its CTR as specified above then reply with a "sync" message of encrypted form:~~
    - ~~ASCII "SYNC"~~
    - ~~The old value of CTR as understood by the receiver of the message~~

Terminating the session

- The client can end the session by sending a message consisting of:
    - ASCII message "QUIT"
    - An optional (plaintext) reason for closure of the connection, such as "SESSION TIMEOUT"
- The server should respond to this with just the ASCII message "QUIT"

Session messages

All messages below shall follow the format for session messages.

Base format for session messages

- Once a session is established, all messages that do not perform session management will consist of:
    - ASCII “ENCR”
    - 8 bytes containing the counter CTR for the session
    - 8 bytes for the length of the remainder of the message (i.e. the encrypted portion) in bytes
    - The encrypted message E(K,CTR,M), where M is the intended sub-message

Note that the session confirmation message has this format, it is considered to be the first message for the sake of the counter. After session initiation, CTR should be 1 plus the initial value of CTR.

The value of CTR used represents the number of encrypted blocks sent. If a message has length Length=0, it does not increment CTR. Otherwise, CTR is incremented by $\lceil Length \div 64 \rceil$, this value is cumulative except when set by a SYNC session control command.

From now on, sub-formats will be described as having or consisting of “an encrypted form”, which refers to the contents of the sub-message M, so that a message of this format is received, which is decrypted to reveal another message of the described form.

Error responses

- The server can respond to any encrypted requests with a response having encrypted form of:
    - ASCII “FAIL”
    - An optional plaintext reason for failure
- Otherwise, the server must respond appropriately

Format for uploading files

- A request to upload a file has the encrypted form:
    - ASCII “UPLD”
    - The length of the file name (8 bytes)
    - The file name
    - The length of the file
    - The file data
- The server response has an encrypted form of just ASCII “SUCC”

Format for downloading files

- A request to download a file has encrypted form:
    - ASCII “DWNL”
    - The file name
- The server responds with a message having encrypted form:
    - ASCII “DATA”
    - The file length (8 bytes)
    - The entire file

File metadata

- File statistics requests have the encrypted form:
    - ASCII “STTR” (4 bytes)
    - The filename
- File statistics responses have the encrypted form:
    - ASCII “STAT” (4 bytes)
    - The length of the file (8 bytes)
    - The SHA hash of the file (32 bytes)

File integrity check

- File integrity challenges have the encrypted form:
    - ASCII “CHAL” (4 bytes)
    - Offset into the file (8 bytes)
    - Length of file to hash (8 bytes) (it is unspecified what should occur if the length exceeds that of the file)
    - Salt to hash with (32 bytes)
    - The filename
- File integrity answers have the encrypted form:
    - ASCII “ANSR” (4 bytes)
    - The result of hashing the portion of the file specified prepended with the salt

Security Analysis

Threat model

- The attacker knows the protocol
- The attacker has total control over the network, including the ability to read all TCP messages as well as to send and forge arbitrary TCP messages (but not, automatically, to forge the encryption).

- The attacker does not know the server's RSA public key and has no control or knowledge of random number generation.
- For purposes of integrity, the attacker is permitted to know the plaintext of any and all files as they are sent (but not to know any issued integrity challenges).

Security guarantees

Three security properties are guaranteed:

Authentication

The attacker cannot pretend to be the server. This is guaranteed by encrypting the session key with the server's public key. Since the session key is only visible over the network as an encrypted message, knowledge of the session key proves the identity of the server.

Integrity

The attacker cannot fabricate or corrupt file data that contain data without detection of errors in those files. To achieve this goal, users can request information that verify the integrity of files. While individual messages can be corrupted, integrity is only violated if files are erroneously accepted (i.e. without a challenge).

Confidentiality

The attacker can, at most, determine the length of messages sent, the protocol used, the session control messages and the value of CTR. Confidentiality is provided by and as strong as the encryption used.