Kevin Chen
CS 475
Assignment 4

Analysis

Overview
This document discusses various attacks and how this authentication protocol handles them.

Replay Attacks
The authentication protocol protects against replay attacks by incorporating a shared session key encrypted with the client's public key. Upon receiving an authentication request, the server's only concern is whether it can decrypt the encrypted message it has received. This means that an eavesdropper could just send the encrypted payload of an authorized user, which would be the include a valid client name and a valid signature. The server would decrypt this and find the client name without an issue, send its response, and start listening for commands from this client. The eavesdropper would only be able to gain access to this server by knowing the original client's private key to decrypt the server's response and get the shared session key. Without this, the server will refuse any incoming messages.

Replaying encrypted messages from the server to the client would not work as the client would be expecting messages from the server it is attempting to connect to.

Man-in-the-Middle Attacks
Similar to Replay Attacks, a man-in-the-middle attack would require the intercepting entity to know what decrypted messages being sent. Without knowing the server's private key, the intercepting entity would not be able to read the encrypted client name sent from the client. Without knowing the client's private key, the intercepting entity would not know the shared session key. This type of attack would need to know both of these keys, along with each public key, to successfully gain access to the server.

Going the other way, since the client would be trying to connect to the middle man thinking it is the actual server, so sending an encrypted response that the server would have sent would gain access to the client.

Imposters
Like the above two attacks, knowledge of private keys is necessary to do any damage to the server. An imposter, knowing the client name and server's public key, could start the protocol by sending $E_s(N_c)$, but would be stopped by the session key when the server responds with an encrypted value.

Conversely, impersonating the server would require knowing only the client's public key or a specific client name. The client will not establish a connection with the server unless the server can respond with the client name encrypted with the client's public key. If an impersonating server has the client's public key and the client's name, then it can encrypt those values, spoof a session key,

and make the client believe that the response was sent after reading its initial authentication request, which the imposter never cared about.

One way to protect against impersonating servers would be to add a randomly generated sequence number S in the protocol, so:

1. Client sends $E_s(N_c, \mathrm{S})$
2. Server responds with $E_c\big(D_s(N_c, K_s, \mathrm{S}+1)\big)$

This way, both steps of the protocol are encapsulated. Unless the impersonating server could guess the incremented sequence number, the client would reject the response without the correct value.