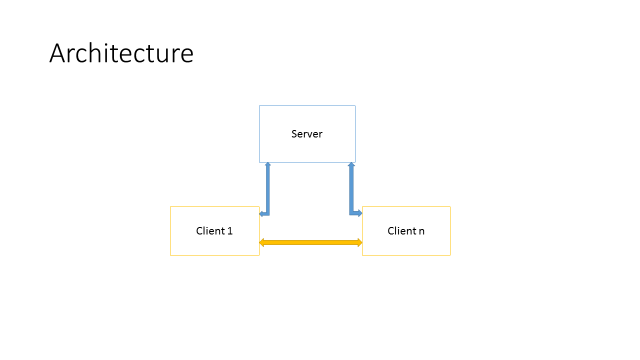
Network Security cs6740

Secure Instant Messaging Application Design

|  |  |  |
| --- | --- | --- |
| **Name** | **Email Id** | **NUID** |
| Aditya Kammardi Sathyanarayan | [kammardisathyanara.a@husky.neu.edu](mailto:kammardisathyanara.a@husky.neu.edu) | 001220609 |
| Rohit Patnaik | [patnaik.r@husky.neu.edu](mailto:patnaik.r@husky.neu.edu) | 001258956 |

Secure Instant Messaging Application

The architecture of the application involves a single server and multiple clients



**Types of communication between the entities**

Client to Server – Initial login, Request for list of online clients, Request for key to communicate with another client, Request for logout.

Client to Client – Request for communication, Session key establishment, sending and receiving secure messages.

**Assumptions**

Below are the assumptions made in this design

* The clients are pre-registered in the server with passwords (Secure Password)
* Secure Password – Combination of Uppercase lowercase and special characters and minimum length of 8 characters
* Each client will have the public key of the server
* Server is always online

**Cryptographic Algorithms and Key Sizes**

* Symmetric Encryption Algorithm – AES
* Symmetric Encryption Key size – 128 bit
* Symmetric Encryption Block Cipher mode - CTR
* Asymmetric Encryption Algorithm – RSA
* Asymmetric Encryption Key Size – 1024 bit
* Hashing Algorithm – Either SHA1 or SHA2

**Features**

* Authenticate a client
* List all online clients
* Session Key Establishment between clients
* Secure communication with authenticated clients (after session key establishment)
* Graceful logout

Notations for encryptions and signatures are as per the text book

* *Symmetric Key* {Message}
* [Message] *Private Key* - For signatures
* {Message} *Public Key* – Asymmetric Encryption with public key

**Authentication and Session Key Establishment (Client and Server)**

1. Involves sending receiving of cookies before authentication
2. C to S -> A // [REQ\_TO\_BE\_AUTHENTICATED]
3. S to C -> Hash (Client IP: Port, Secret, A) // [COOKIE\_SERVER]
4. C to S -> Hash (Client IP: Port, Secret, A) // [COOKIE\_CLIENT]
5. S to C -> [g p] *Private-Key S (*No Longer used)
6. C to S -> {Password, g^a mod p, A,Nonce} *Public-Key S* (The Nonce is an addition , it used for the CTR mode encryption treated as an IV) // [PASSWORD]
7. S to C -> [g^s mod p] *Private-Key S //* [DH-KEY-S]
8. SKC = g^cs mod p
9. Notify all online clients about the client which has come online now // [NEW\_CLIENT\_NOTIFICATION]

At step 8 both the client and server will have established a session key and future communication between the client and server for the session takes place using symmetric encryption of messages using SKC.

Message Integrity of the messages is ensured for messages 6 because if the password or A is changed then authentication fails, if DH-Key is altered then it will lead to incorrect SKC and the client will have a different key at step 8

Note: All symmetric encryptions will be accompanied by a HMAC for integrity protection (HMAC not implemented in the Login protocol between server to client)

**Listing all online clients**

When the user types list in the text interface

* Client 1 to Server -> *SKC1*{List All Online Clients} // [LIST]
* Server to Client 1 -> *SKC1*{Online clients List} // [LIST-ANSWER]

**Peer to Peer Communication**

Scenario – Client 1 needs to talk to Client 2

Steps involved are:

* Client 1 informs Client 2 that it wishes to communicate
* Client 2 sends a nonce (keeps track of the Ip address it sent the nonce to)
* Client 1 sends the nonce along with the request for key to communicate with Client 2
* Server sends back a key and a ticket to be given to client 2
* Client 1 and Client 2 establish a session key

Below are the exact steps:

Client 1 notifies Client 2

* Client 1 to Client 2 -> I want to talk to u // [REQ-TO-TALK]
* Client 2 to Client 1 -> N1 // [NONCE]

Obtain Key from Server

* Client 1 to Server -> SKC1{I want to talk to Client 2, N1} // [MESSAGE-REQ]
* Server to Client 1 -> SKC1{KC1C2, Nonce1} || SKC2{KC1C2, Client 1 N1, Nonce1} (*Server purges KC1C2 after sending the message*) (The Nonce is an addition used for CTR Mode encryption) // [SESSION-KEY-FROM-SERVER]

(*While implementing due to time constraints the verification of N1 at this step was missed out and this can lead to exploits*)

Session Key Establishment

* C1 to C2 -> SKC2{KC1C2, A N1} ||KC1C2{DH-KEY C1, Nonce2} // [SESSION-KEY-FROM-SERVER]
* C2 to C1 -> KC1C2{DH-KEY C2} // [DH-KEY-C]
* SC1C2 from DH-KEYs
* C1 to C2 -> SC1C2{N2} // [CLIENT-CHALLENGE]
* C2 to C1 -> SC1C2{N2, N3} //[CLIENT-CHALLENGE-RESPONSE]
* C1 to C2 -> SC1KC2{N3} // [CLIENT-RESPONSE]

**Graceful Logout**

User types logout in the terminal

* C to S-> SKC{LOGOUT}
* Server purges SKC, deletes C from list of online clients
* S to C1…. N -> SKC1{LOGOUT\_NOTIFICATION} …. SKC2{LOGOUT\_NOTIFICATION}
* All Clients purge Session Keys with the logged out client if any
* Note – The client does not send a nonce because it sends the logout message encrypted with its session key and even if an intruder stores this message and replays it next time, the session key will be invalid.

**Types of Messages**

We will be using a message type field in each message which is sent across (we wish to send across Json objects or some related objects which can store multiple fields in their contents)

Below are the types of messages we have identified so far

* [REQ\_TO\_BE\_AUTHENTICATED] – Client sends to Server before authentication
* [COOKIE\_SERVER] – Server sends a cookie to client
* [COOKIE\_CLIENT] – Client replies with a cookie
* [AUTH\_REQ\_PERM] – Server replies with g ,p used for DH-KEY exchange
* [PASSWORD] – Client sends encrypted password along with g^a mod p
* [REQ-TO-TALK]- Client sends request to talk to another client (peer)
* [NONCE] – Client sends back a nonce to a client who requested to talk
* [MESSAGE-REQ] – Client sends request to talk to another client (to server)
* [DH-KEY S] – Server’s part of DH-KEY sent from server
* [DH-KEY C] – Client’s part of DH-KEY sent from client
* [CLIENT-CHALLENGE] – Initiating client to the client with whom the initiator needs to talk
* [CLIENT-CHALLENGE-RESPONSE] – The client which receives CLIENT-CHALLENGE responds back with this message to the initiating client
* [CLIENT-RESPONSE] – The initiator sends back a response to the initiating client
* [SESSION-KEY\_FROM\_SERVER] - Session key
* [CONV] – Normal encrypted messages between clients
* [LOGOUT] – Logout message from Client to Server
* [LIST] – From Server to Clients
* [LIST\_ANSWER] – From Server to Clients
* [LOGOUT-NOTIFICATION] – From Server to Clients

**Protection against attacks**

* **Defence against DOS** (Client to Server) – Server does not allow authentication until the client responds back with the cookie.
* **Defence against DOS** (Client to Client) – When Client A wants to talk to Client B, B sends a nonce to Client A, which will send this to the server and the server then encrypts this nonce in the ticket to Client B. Which is proof enough for Client B to know that the Client A is who he/she claims to be. (*Note this protects against DOS and ensures B that A has spoken to the server*)
* **Forward Secrecy** – Each communication involves setting up a session key in which both the communicating entities participate and they forget their session key after log out.
* **Password Security** 
  + Dictionary attack – Ensure the passwords are a combination of upper and lowercase letters with special character
  + Salting and SHA2 for hashing
  + Online attack – Limit the number of failed attempts to 3 (Was not able to implement in time)
* **Protection from Server Snooping on messages** – Messages sent between the clients are encrypted by a session key which they establish
* **End Point Hiding** – An intruder will need to know the session keys of the clients with the server in order to know who is communicating with who. As the identities are encrypted using the session keys.

**Future Enhancements**

* Derive keys from passwords
* Improve Authentication to Server by not sending password and instead encrypting a value using the key derived from the client’s password
* Store the passwords hashes in an encrypted file
* Send notification to client when another client requests the server to verify the client’s certificate
* Allow anonymous communication as an option
* Handle non graceful logout (Check for live client)
* Delivery Notification
* Allow user to register

**Flaws in Design / Implementation**

1. When the user enters the password in the terminal, the input is not masked and can be seen in the terminal. An intruder standing behind can the actual user can snoop on the password
2. We have introduced message types and tried to implement some of sort of a sequence so that if a message arrives out of order the server / client will reject the message.

For example, when the client is authenticating to the server if the password message comes before the cookie message then the server knowing that it is not expecting a password message will reject it. Similarly, when the clients are authenticating each other and if a DH-KEY-C message comes out of order, the other client will reject it.

**Flaw** – Now the problem is we used a python list called next\_expected\_messages and we are appending the next expected message type in the list. What we missed is that we did not clear the list before adding a new message type into this list. And anyone who has access to our code can easily figure this out and the application is vulnerable to replay / DOS where the intruder can send messages which were previously legal.

1. Another assumption here is that the server is not compromised. If the server is compromised then the client to client communication is at risk.