Distributed Password Cracker

Sambit Mishra 720002013
Ankita Girish Wagh 621003584

24th Feb 2013

Architecture and Function Description

LSP Protocol

Server-

A server is designed to handle multiple client requests for cracking the password and handling it to the workers distributing the load equally. The worker cracks the password and handles it back to the server which sends it back to the client. Our server consists of a client list where it maintains the clients with their connection IDs. As soon as a client request comes and a connection is established, the client is added to the client queue. Our function lsp_server_create() creates the server , spawns a server epoch thread and initializes the input queue and output queue. The input queue will store all the incoming packets and output queue will store the outgoing packets. Our function lsp_server_read() dequeues the messages from the input queue and the function lsp_server_write() populates the output queue with the messages which are ready to be sent.
Server Epoch-
Our function server_epoch serves two purpose-
When the select() unblocks due to timeout the return value will be 0. At that time the server epoch sends an ACK for the most recently received data message by checking the input queue or if the data message has been sent but not yet acknowledged then resends the data message by checking the output queue.
When the select() unblocks due to data , the return value is greater than 0. At this time server_epoch uses recvfrom() to receive data, checks whether the payload is empty or contains some message. If it is an ACK then deletes the corresponding message from the output queue and if it has payload then populates it in the input queue. For handling duplicates , we keep a track of last message sent and next message expected. Thus on getting an ACK we compare the sequence number with sequence number of last message sent and on getting payload we check its sequence number with next message expected. It uses sendto() to send an ACK for the sucessful data message received and send message from output queue.

Client-

Our client is designed to work as a request client as well as worker for the application layer protocol. The lsp_client_create() functions creates a client. It takes care of resending a connection request if original connection request has not yet been received. It then spawns a client epoch thread and returns the client. The lsp_client_read() reads the incoming messages and empties the input queue and

lsp_client_write() populates the output queue with the messages which are ready to be sent.
Client Epoch-
Our client epoch also serves two purpose-
When the select() unblocks due to timeout then it sends an ACK for the most recently received data message by checking the input queue or an ACK with sequence number 0 if no data message have been received.If the data message has been sent but not yet acknowledged then resends the data message by checking the output queue.
When the select() unblocks due to data , the return value is greater than 0. At this time client_epoch uses recvfrom() to receive data, checks whether the payload is empty or contains some message. If it is an ACK then deletes the corresponding message from the output queue and if it has payload then populates it in the input queue. For handling duplicates , we keep a track of last message sent and next message expected. Thus on getting an ACK we compare the sequence number with sequence number of last message sent and on getting payload we check its sequence number with next message expected.It uses sendto() to send an ACK for the sucessful data message received and send message from output queue.

ReAssigning Worker
If a worker is killed in between of cracking the password. We will take the request from the worker and queue it. Once any new worker joins we will assign this reques to the new worker. The number of workers which are present are also controlled through a queue. **Password Cracker**
Our password cracker is a running on LSP protocol and it takes the 40 byte Hash key and takes the length of the string , finds the hash of all the strings and compare it with the given hash. When it finds the same hash it gives the corresponding string.

Execution
In order to execute the code we will need to clean the binaries and "make" it again. Run the following command :

```
make clean
make.
```
Now we can start to run the Password Cracker Program.

Example:
```
./server 8989
./worker 127.0.0.1:8989
./request 127.0.0.1:8989 9d989e8d27dc9e0ec3389fc855f142c3d40f0c50 3
```

Expected Output
```
Found: cat Client Exiting.
```

Once the password is found we exit the client.