Abhishek Velayudham & Varun Chitkara (Team: NoTime4)

Design Document

Appending:

First, we get all the input parameters, store them, and verify that the parameters are proper. Then, if it is the first time that the log is being appended to, the valid command is then stored in the log and, using authenticated encryption, we encrypt the log. If it is not the first time the log is being created and it is being appended to, then we decrypt the file first to read its contents and then check against the log if the command is valid. If the command is valid, then we append to the log and encrypt again.

Log Format:

All lines include the following info:
T: #ID #(A|L): #(E|G) \"#NAME" (R: #ROOMID)
Where you replace ID, NAME and ROOMID with the correct data and you have (A or L) or (E or G) depending on the given input and the room is optional.

Reading:

First, we get all the input parameters, store them, and verify that the parameters are proper. Then we apply decryption on the log file specified. During the decryption process, we can check if the token given was the correct one for the log file as the encryption/decryption scheme allows for that. After this, we pass the data to different methods depending on the action.

For S, I first get a list of the guests, employees, and rooms in the log. To get the people in each room, I then iterate through the list of rooms and I insert/remove into a list for that room based on the line's command.

For R, I iterate through all of the log file and check to see if the line has a room id and that the user specified is the user in the current line. If so, I then concatenate it into my result string. After the iterations are done, I simply return this string.

Attacks:

(Note: all were implemented)
Specific attacks that we considered were buffer overflow, ensuring confidentiality, integrity, authenticity, and integer overflow, We made sure that buffer overflows were prevented by reading with fgets and fread so that we know how many byte we are reading from. We also implemented the CIA triad in our code by using authenticated encryption. We ensure confidentiality by using a token provided by the user to encrypt the log. So the attacker cannot read the log unless they have the specific token. We ensure integrity since the attacker cannot modify the file without us knowing, since the decryption would produce invalid output. We ensure authenticity by ensuring that only people who know the token can access the results, which we do through our encryption/decryption process. We ensure that no integer overflows occur by carefully checking to make sure that none of the inputs are out of range.

Some code samples:
–- ‘ file_read = fread(encrypted, 1, sizeof(encrypted) - 1, log);’

```
--‘if(ret > 0){
   total_len += olen;
 }else{
   printf("integrity violation");
   return 255;
 }


-- ctx = EVP_CIPHER_CTX_new();
   EVP_EncryptInit_ex(ctx, EVP_aes_128_gcm(), NULL, NULL, NULL);

   EVP_EncryptInit_ex(ctx, NULL, NULL, token, iv);

   EVP_EncryptUpdate(ctx, encrypted, &olen, msg_str, strlen(msg_str));
   total_len = olen;
   EVP_EncryptFinal_ex(ctx, encrypted + olen, &olen);
   total_len += olen;
   encrypted[total_len] = '\0';

   EVP_CIPHER_CTX_ctrl(ctx, EVP_CTRL_GCM_GET_TAG, 16, tag);

   for(i = total_len; i < total_len + 16; i++){
     encrypted[i] = tag[i - total_len];
     //printf("%c", encrypted[i]);
   }

   //printf(tag);

   encrypted[total_len + 16] = '\0';

   cipher = fopen(logpath, "w");
   fwrite(encrypted, 1, total_len + 16, cipher);

   EVP_CIPHER_CTX_free(ctx);

_
```