Group W6 Report – Development of AAA System

Introduction

Abstract

The system being developed is a secure Authentication, Authorisation and Accountability service around the operations and data of a medical service provider. In the medical service field, much sensitive information is stored about both clients, staff and resources, so it is of the utmost importance that this information can only be accessed by authorised personnel, and that those who need access and have authorisation are able to use resources without significant disruption.

In order to satisfy the service's needs, we will develop a system which ensures that personnel will: be properly authenticated by the system with minimal possibility of being impersonated, be given appropriate authorisation to access resources according to their role and only those resources, and be held accountable for all the actions they take and resources they access.

This document outlines the development of the system described so far, beginning with the conceptual design of the system, listing the intended features and planned implementations, before going into more detail on the implementation of this design as well as the specific decisions made in a chronological order. The document then lists our conclusion on the project, indicating the success of our system and our thoughts on it, before a final section briefly outlining each member's individual contributions to the project.

Table of Contents

Table of Contents

System Design

System Requirements

Authentication

Functional

- The system shall check the strength of a user's password when attempting to register to the system.
- The system shall prevent users from using a common password.
- The system shall authenticate the user using multiple methods.
- The system shall hash and encrypt passwords before storage.
- Communication between the client and the system server shall be encrypted.
- The system should verify trusted IP addresses.
- The system shall require two factors of authentication for a user to register or log in.

Non-Functional

- The user shall be able to log in to the system within 5 minutes.
- A message sent by the user to the system should be handled within 10 seconds.

Authorisation

Functional

- Within the system there shall be roles for Staff, Nurses, Doctors, Hospital Administrators, Regulators and IT Administrators.
- A user shall be able to have one or multiple roles.
- The system shall have a database containing Patient Health Records, Patient Personal Records, Appointment Bookings, Staff Data, Regulation Data and Log Data.
- Staff shall be able to create, edit, delete and view Appointment Booking records and view Patient Personal Records.
- Nurses shall have the same permissions as Staff, as well as be able to view Patient Health Records.
- Doctors shall have the same permissions as Staff, as well as be able to create, edit, delete and view Patient Health Records.

- Hospital Administrators shall be able to view Patient Health Records and Patient Personal Records and Log Data.
- Hospital Administrators shall be able to create, edit, delete and view Staff Data and Regulation Data.
- Regulators shall be able to view Regulation Data.
- IT Administrators shall be able to view Log Data.
- IT Administrators shall be able to authorise registration of new users, block or revoke users and edit users' assigned roles.
- The system shall provide different UIs to different users based on their role to prevent access to unauthorised actions.
- The system shall present a menu based on the commands that are allowed by a user's role. This will also dynamically change in the case where a user is assigned multiple roles.
- The system should validate user roles on both the client and the server before carrying out operations

Non-Functional

- Changing of a user's roles shall take less than one hour to perform.
- Interacting with a file shall take less than a minute to perform.

Accountability

Functional

- The system shall automatically log key events as they take place in the system, including information about the actor, the time, the activity performed and the result of the event.
- The system shall implement an active method of detecting suspicious activity occurring in real time.
- The system shall block user accounts which have been detected acting suspiciously.
- The system shall allow for access to be revoked and restored.
- The system shall provide notifications whenever suspicious activity takes place.
- The log shall be read only to all users.

Non-Functional

- It shall take less than a second for a logged activity to be added to the log.

Planned Authentication Techniques

Username and password.

- A password checker will be developed to ensure a secure password is being created by the user. E.g., checking for length, special characters, numbers.
- Passwords will be stored by using a combination of a hash for the password and unique salt. Advantages of this include 1. acting as a bottleneck to a Rainbow table attack and 2. Slowing down brute-force and dictionary attacks.
- Two-factor authentication will be implemented with a one-time password for the login session – this could be in the form of a 6-digit pin as commonly seen nowadays.

Public/private key encryption and signatures.

- A combination of Asymmetric encryption and Symmetric encryption can be used to ensure secure communication takes place between the client and server.
- Servers and clients will generate self-signed certificates.
- An implementation like that of SSL/TLS could be used for secure exchange of credentials whilst also being efficient.
    1. Initial handshake with the use of Asymmetric keys and digital signatures will take place where the secrets will be exchanged.
    2. Secrets will be used to generate a master secret on the server side and client side independently.
    3. New keys are generated for each session based on the master secret using a Pseudo random function.
- In order to prevent man in the middle attacks, a more modern encryption algorithm will be used – this will be further explored as the project takes place.
- Digital Signatures– (In the initial multiple secret transfers between client and server)
    1. Public and private key is generated for a user.
    2. To send a message a digitally signed document is made which is a combination of the hashed data encrypted with a private key and the original data.
    3. The receiver obtains the two items: encrypted hash and original data.
    4. Receiver generates hash from original data and decrypts the encrypted hash using the sender's public key.
    5. Hash from decryption and hash from hash generation are compared, if equal then user is authenticated.
    - Key Advantages: Message authentication, data integrity and non-repudiation.

Message Authentication

- MACs will be generated for the client and server messages before they are sent to the receiving party so that they can be verified. This will ensure that the message, if received from an external source, is guaranteed to be other party.

- The MAC keys will be generated as a result of the secrets transferred in the initial TLS handshake process. The use of secrets makes it more difficult for a middle-man to obtain the key.

User Authorisation Levels and System Data Types

System Data

Within this system, there are several types of data, each with distinct levels of access. The system data includes:

- Patient Health Records – This consists of medical information as well as treatment and condition history, as well as any unique requirements or considerations.
- Patient Personal Records – This includes contact information and emergency contact information as well as any demographic information that isn't directly related to a patient's health.
- Appointment Bookings – These outline a patient's booking, listing the information about the time and date, reason for appointment and any staff, if applicable, who are assigned.
- Staff Data – Information about staff's addresses, contracts and salaries which is stored for use by the administrators of the medical service.
- Regulation Data – Information compiled from other data sources by administrators for use by external regulators when assessing the medical service's performance.
- Log Data – Information on accesses and operations performed by users.
- System User Data – Information regarding a user of the medical system, consisting of username, password and other data required by the system.

User Roles

In order to keep this data secure within this system, different access levels are required. These access levels relate to the role a user performs, and thus they can access the relevant information only. These roles include:

- Staff – This role may create and access Appointment Bookings, as well as access Patient Personal Records.
- Nurse – This role has all permissions of Staff but may also access Patient Health Records in the event of emergencies or to assist other activities.
- Doctor – This role has all permissions of Staff but may also access and update Patient Health Records to perform their activities and fill in changes to a patient's situation.
- Hospital Administrator – This role may access Patient Health Records and Patient Personal Records for assessment purposes, access and update Staff Data for business

administration, access Log Data, as well as access and create Regulation Data to be assessed by the regulating body.

- Regulator – This role exists to allow the regulating body access exclusively to the Regulation Data for regulation and assessment purposes.
- IT Administrator – This role can access Log Data as well as manage System User Data and Access Control Systems.

With regards to the activities that can be performed by a given role, it may be necessary to explain further exactly what is meant by certain language. Thus, the language used is defined:

- Access – If a role can access a set of data, then they will be given permission to both view the data in its entirety, as well as search through the data for specific information
- Update – If a role can update a set of data, they have permission to add new records, edit fields in existing records, as well as delete records entirely. It logically follows that any role with permission to update also has permission to access, otherwise they would be modifying data that is unknown to them, which is not likely to be secure.

Many Roles for One User

By defining these roles, the aim is to strictly limit users' activities only to those which they have permission to perform, however there are often exceptions and it is impossible to predict what exceptions to the above plan may appear. With this in mind, it was decided that users must be able to have multiple roles assigned, their overall permitted actions being the union of all these roles' sets of permissions.

Allowing users to have multiple roles allows the flexibility of a member of staff who needs access to some specific data that they would not normally be able to access, without adding layers of complexity to or compromising the structure of the roles themselves. Keeping the roles atomic allows us to keep the authorisation system more secure, therefore avoiding the risk of potential attacks through this role assignment scheme.

Role Checking

It is important to check a user's roles properly, otherwise they might be able to pretend to have a higher authorisation level than they do. To satisfy this requirement, two checks were planned: checking a user's roles on the client side occurs, but the main goal of this check is to use the information to only show them the actions that their roles allow them to take; checking a user's role also occurs on the server side, matching a user's ID sent in their message against the roles stored for that user, which then allows for determining whether the action they are trying to perform is legal. This separation of checks ensures that the server is the only authority which can conclusively determine what someone's role is, preventing a user from lying or fabricating additional permissions.

Planned User Accountability Measures

To ensure accountability, the system should have an append-only log that is automatically added to whenever any operation happens on the system. These operations include system logins, accessing records, modifying records, etc. The system should log the operation, the user who carried out the operation, the time that the operation took place, and whether the operation succeeded or failed as failed operations could be indicative of an attempted attack (such as trying to brute force login credentials).

Preferably the system would have some mechanisms for detecting suspicious activity and generating alerts instead of relying on someone regularly reviewing the logs. Suspicious activity could include attempted access from non-whitelisted IP addresses, unusually large amounts of activity, attempted access to records above the user's access level, etc. These logs should be backed up automatically to ensure they are accessible in the event of an attack like ransomware.

The logs should not be accessible to every user, only administrators. Viewing the log is a log-able activity as it could be viewed as suspicious behaviour especially if a user who isn't authorised to view the logs attempts to. The logs should be sufficient to analyse and trace a cyberattack retrospectively, for example understanding whose account was used to make changes, which computers accessed the system at the time of the attack, which operations were carried out, etc.

System Attack Mitigation and Prevention

Prevention

- Two-way challenge-response authentication (see section 3.a) can be used to prevent man-in-the-middle attacks.
- Having a strong password policy, a delay between multiple wrong password attempts, two-factor authentication (see section 3.a) can be used to prevent password-guessing attacks
- The threat of insiders can be minimised by having clearly defined authorisation levels for each system user (see section 3.b) as well as having a set of policies that each system user with access to sensitive data must follow.
- Important data should be stored in encrypted form and all data should be encrypted when in transit.
- Damage from denial-of-service attacks can be minimised by having multiple server replicas in different data centres to avoid presenting an attacker with a single target. Different servers could also be used for staff and patients, where the staff server can only accept traffic from a pre-set lit of IP addresses, thus keeping the most important part of the system in operation during an attack. Staff working from home could use a VPN to access the system.

- Storing servers in known and secure datacentres or controlling access to physical server hardware to prevent physical attacks (ex. inserting USB and extracting data from the server)

Detection and mitigation

- The system should detect possible ongoing attacks by monitoring and detecting suspicious user behaviour (see section 3.c). This could be done by having a risk points system, where every suspicious behaviour (A doctor accessing an unusually high number of patients' data, logging in only after 2 unsuccessful tries, etc.) increases the risk score for a user and reaching a decision (blocking/informing the system administrators) once the number of points passes a pre-set threshold.
- Denial-of-service attacks can be detected and mitigated by monitoring incoming traffic. If the traffic is unusually high, suspicious IPs can be blocked. In case of a larger scale attack, traffic from all IP addresses could be blocked except for already known IP addresses of medical institutions or staff.

Project Implementation

Initial System Structure

Client and Server

For the implementation of our system, we opted for a client-server approach, with most of the user interface implemented on the client side (meaning very limited activity could be initiated on the server) and most of the operations of the system (preventing users from manipulating data significantly from the client).

This client-server connection was implemented using Java RMI due to Java being our language of choice. RMI has some inherent security risks, however we minimised these using only sendMessage and encrypted alternative as the methods run over it, allowing for a Message object to transmit the actual information required for the server. Encrypting the Message objects (as detailed later) transmitted over the connection thus allowed our communications to be secure between client and server.

Limited UI Freedom

The UI implemented for the user focussed on selecting functionality from lists, as well as clearly providing the structures necessary for the activities they wish to perform. This meant that there was a reduced risk of user errors, but selecting from lists also prevented a level of freedom that would help a malicious user in the manipulation of the system.

Message Class Structure

The message class used was structured to facilitate all the communication required, and so it had a defined structure as follows:

[Sender Info- IP and ID][Command ID][Message Content]

This structure allowed us to keep track of who was sending messages so that they could be held accountable, as well as understand what the message was indicating – thanks to the Command ID – before the message content was accessed, allowing the Message object to be multi-purpose.

Storage System

For our storage on the server, we utilised a bespoke DataManager class, allowing us to include the exact functionality we required, such as encryption of files implemented later, whilst also having the benefits of a structure unknown to attackers, increasing security.

Initially we considered SQL, however opting for our own bespoke solution allowed us to keep things simple and avoid known attacks against SQL, like SQL injection.

Authentication Scheme

At the beginning of this project, a key topic of implementation was the authentication of users as they access the system. Multiple authentication techniques were implemented to ensure that the client was who they said they were and to ensure that the server was the correct server that the client intended to communicate with.

How does the authentication scheme work?

Our approach is a hybrid approach that uses both asymmetric encryption and symmetric encryption. The approach is based on the TLS standard and is described as follows:

1. Both the server and client create a self-signed X509 certificate which certifies the asymmetric key pair and provides the public key to the recipient of the certificate.
2. Server and client exchange their certificates to emulate the TLS type approach. Unfortunately, as this is not an actual online system, these certificates cannot be authenticated by an external certificate authority, nevertheless, the generation of these certificates indicate that if the system were to be put into real use, we could enable external authentication by a CA and create a certificate chain.

3. A digital signature is then created by both the client and server to exchange the random secret created by both parties. This works by initially generating a symmetric key, encrypting the digital signature and original data with this key and then encrypting the symmetric key with the other party's public key for secure transport. This provides full security as the symmetric key can only be accessed by the other party because it is encrypted with their public key and the object which is in transport can only be decrypted when the symmetric key is accessed.
4. The client then generates a pre-master secret and transports this again using the encryption of the combination of a digital signature and original data via a symmetric key which had previously been securely transported.
5. Using a combination of the client secret, server secret and pre-master secret a master secret is generated using a hash function which uses the method SHA-512. This generates a 64-byte hash in which the first 48 bytes are used to create a session key. This occurs on both the client side and the server side independently so that they both now have access to a symmetric key that can be used to encrypt and decrypt objects/messages.
6. The last 48 bytes of the hash generated in this method are used to generate a MAC key which is separate from the session key. Once again, the same MAC key is generated independently on both the client and server side. This provides an extra layer of security which promotes integrity and authentication.
7. After this, the client and server now use a MAC key to generate a MAC and a session key to encrypt the whole message. This is then decrypted using the session key on the recipient side and the MAC is verified on the recipient side using the MAC key.

Why Digital Signatures?

The use of a digital signature to exchange secrets in this approach provides all three essential aspects of a secure communication: integrity, authentication and non-repudiation. The generation of a symmetric key after the secret transfers is required as symmetric encryption is faster but also allows for more data to be encrypted. The use of a session key allows the prevention of replay attacks. Given that these keys are temporary for the time that the client is communicating with the server, attackers cannot hold onto previous messages after interception and send them later, the client/server would simply refuse the message. Also, another advantage of a digital signature is that it can prevent a man in the middle attack, this is because it is very difficult for an attacker to replicate a digital certificate and have its authenticity confirmed by a certificate authority.

Why MACs?

This implementation is secure as we can ensure that only the intended client and server generate the MAC key and session key through the digital signature. Furthermore, the MAC allows us to ensure that any incoming messages could only be created by the other party and no tampering with the message has taken place. However, MACs do not provide non-repudiation due to both the client and server holding the MAC key. Therefore, in order to combat the future potential event of repudiation by either party, a detailed append-only log will be generated to provide further proof that indeed the other party did send the message. The opposing party would then in this event struggle to prove that they did not send the message.

Keys Used and their encryption types

Asymmetric Keys use the RSA algorithm with key size 2048. RSA's implementation is straightforward and has the advantage of scalability in terms of length to prevent brute force attacks.

Symmetric Keys use the AES algorithm – very secure against brute force attacks due to the long key length.

To ensure that only authorised users can register on the system, we implemented a registration code system. To register on the system, a system administrator generates a unique code consisting of one letter and six numbers, and the user enters this code when they register which is then validated by the server. The letter represents the role that the new account will be given on the system (e.g., N for nurse) which prevents new users from accidentally or deliberately choosing the wrong role. Once the code has been entered, it's removed from the active codes on the server so it will no longer be valid (i.e., it can't be reused) or alternatively, it will expire after 30 minutes if it's not used. Both policies protect against unauthorised registrations by limiting the number of active codes to the amount of people that currently need to sign up and limiting the window of time for new registrations. From the admin client, the system administrator can view the currently active codes as well as their expiry dates which helps keep track of potential entry points for attackers.

Password Generation

Another important aspect of user security is multi-factor authentication (MFA) where the user must present two or more of the following:

- Something the user knows (e.g., password or secret question)
- Something that the user has (e.g., specific mobile phone to generate one-time password)
- Something that the user is (e.g., fingerprint or face recognition)

A password is one factor that we used in our system (something the user knows) and it's the first line of defence to preventing unauthorised access to user accounts so it's vital that the users' passwords are strong and that they are stored/transmitted securely. To ensure passwords are secure, they must meet the following criteria:

1. Not in the top 1000 most common passwords[1]
2. At least 8 characters long
3. At least 1 number
4. At least 1 symbol
5. At least 1 uppercase letter
6. At least 1 lowercase letter

By enforcing these criteria, passwords are more protected against attacks like a dictionary attack. Users will often pick a password based around a dictionary word but by adding in symbols, numbers, etc. the number of variations that an attacker would have to try on average increases decreasing the chance of the attack being successful. A long minimum password length requirement decreases the chance of a letter-by-letter brute force attack by, again, increasing the variations that an attacker would have to try. All these checks are carried out on in the user client, so the plaintext of the password doesn't leave the user's computer.

Password Hashing and Storing

If the password meets the requirements, it's salted and hashed before being sent to the server for storage. This doesn't make much difference to the security of the system because the connection between the client and server should be secure anyway so theoretically it should be fine to send the password in plaintext (in fact, it's common practice to send plaintext passwords over HTTPS) and then hash it on the server. If the password was somehow intercepted (e.g., MITM attack or server being controlled by a bad actor), it wouldn't really matter if the password was in plaintext or hashed as the attacker could just send the username and the hashed password to the server and be authorised. The only security advantage to hashing the passwords on the users' machine would be that stolen plaintext credentials could easily be tried on other systems (e.g., email account). However, there is a big performance advantage to moving the hashing to the users' system as the password hashing algorithm that we used (PBKDF2) is designed to be computationally expensive so by delegating this to the client, it reduces the load on the server.

The passwords are hashed along with a 16-byte salt which is securely and randomly generated for each password. Using a salt defends against attacks using pre-computed hashes (e.g., rainbow attacks) by making it not practical for an attack to pre-compute hashes for common passwords as they would need to compute it for every salt combination too. In the unlikely situation where the database is stolen, and the attacker manages to crack one hash, a salt means that the attacker won't be able to find multiple people who have the same password as a unique salt causes the same passwords to be have different hashes. This salt is stored alongside the password in the database which is safe because having the salt doesn't help reverse the hash.

1 https://github.com/danielmiessler/SecLists/blob/master/Passwords/Common-Credentials/10-million-password-list-top-1000.txt

To prevent against brute force attacks, the server restricts login attempts for a user ID after multiple failed attempts. For the time that logins are restricted, the user cannot login even if they supply the correct password. The user gets three attempts between each restriction and the restriction time grows exponentially (2^x e.g., 1, 2, 4, 8, etc.) which means any serious brute forcing attempts would be significantly slowed down to the point where it would be impractical.

As mentioned before, the hashing algorithm used to encrypt the passwords was the PBKDF2 algorithm. This algorithm was made with password salting in mind, in fact, in order to generate a hash value, a salt must be provided. The key advantage of this method of encryption is the fact that it is slow. This helps to act as a bottleneck to brute-force attacks therefore forcing attackers to use more resources or otherwise give up due to the increased length of time. A longer time to crack passwords also means that in the case of a successful attack, minimal information will be accessed.

Time-Based One-Time Password

Another Multi-factor authentication method that was used in our system is one-time passwords. We used the TOTP (Time-based One-time Password) algorithm to implement this feature, as it is secure, widely adopted and can be used with authenticator applications on various devices. The authenticator device in this case represents the second factor used in our system (something the user has).

During registration, a secret is generated on the server which is then provided to the user in the form of a QR code. The user then uses the secret with an authenticator application to generate single use codes. The generated code is then sent to the server by the client, where it is compared to a code generated on the server. Unlike with Event-based One-time Passwords, new TOTP passwords are generated periodically (every 30s in our system), and only one password is valid at a time, which provides additional security to the system.

Encrypting the Database

Symmetric Encryption

Once the system was up and running and users were able to take actions that could affect records stored in files, it was time to implement encrypting of the data files stored on the server. Since these files were frequently accessed by users and access time was an important consideration, we chose to implement a symmetric encryption algorithm to allow for faster encryption and decryption times compared to asymmetric. The algorithm chosen was AES due to its reputability and our familiarity with it, which we implemented with a key and an initialisation vector.

Key Handling

A problem now posed was the storage of the key and vector, which was crucial in the securing of the files. To be as secure as possible, it was decided that although the system would hold the key in RAM in order to operate files, there would be no storing of the key and vector on the system. Even if the server is reliably secure, it is safest to store the key offline. This meant it was the IT administrator's job to ensure that the key is recorded, either on paper or a whiteboard, or saved on a pen drive. This was to keep the information needed stored for restoring the system in the event of a crash, without leaving it connected to the internet and vulnerable.

As is always the case with keys, there must be a level of compromise. The compromise in this case is that the key is written on a physical medium which could be stolen from the workplace in an unforeseen circumstance. Therefore, it is necessary to keep updating the key regularly, so that a stolen key loses its function. To do this, we had the key randomly generated again at midnight – when the disruption to people's work should be minimal – and the encryption on all the files updated.

Checking Encryption on Reboot

When rebooting the system, perhaps from a crash, the key and vector are required, and they need to be checked against something to ensure that they are correct. The criteria by which they are to be checked needs to be something known to the code but, storing any meta-data could betray file structure if the system code is searched for plaintext, increasing the feasibility of breaking the encryption. Therefore, the poem "The Highwayman" by Alfred Noyes was used, providing two advantages: its content says nothing about the system we've developed; it is long enough to guarantee that decryption has been performed successfully.

System Backups

Shortly before all the files have their encryption updated, a backup routine should be performed. This means that a day's information is available using that day's key, and data is secure if files are lost or corrupted. It is also the plan that as all files are backed up by this routine, the system logs (once they are implemented) are reset after being backed up – fresh without any of their previous data – ensuring that IT Administrators can access them easily and find information efficiently, despite the large quantity of data that would be added to them in a day.

Enforcing Access Control

Access Control Scheme

In order to prevent users from performing unauthorized actions, we implemented a role-based access control system (RBAC). Each system is assigned at least one clearly defined role and may only perform actions assigned for that role. The user roles and their permissions are outlined in the “User Authorisation Levels and System Data Types” section earlier.

Our system has 7 clearly defined roles. Each role only provides the minimum level of access needed to perform the assigned tasks. For example, the default role (no role) has least privilege and only provides access to methods necessary in order to login or register.

Access Control Checking

For increased security, access is checked both on the client and server sides of the system. User roles are initially assigned based on the registration code that the system administrator provides the user when they register but roles can also be added, removed or access completely revoked at any time by the system administrator.

The first reason for doing this is practicality, in an emergency (which are not uncommon in hospitals), a staff member may need temporarily escalated privileges if the correct person is not available. For example, a nurse could temporarily be escalated to a doctor role if they need to update patient records while the doctor is busy, before their permission is once again returned to only the permissions they need for their job. The second reason is security, if an account was compromised, the administrator could revoke access to limit the damage caused.

Securing the IT Admin Privileges

One likely target of an attack is the IT Administrator's account as it essentially has access to everything on the system and it can perform any available operations. To protect against a breach in this account, we implemented a separate admin client to perform operations such as generating new user registration codes, viewing logs, or changing user roles. The code for this admin client would only be present on the server which means even if an attacker did get a system administrators credentials (e.g., phishing attack) and somehow passed the multi-factor authentication, they would also need to breach the server to do any damage (e.g., physical access to the server or connect via SSH). Despite the admin client only being run on the server, it still must connect to the server via RMI to ensure it follows all the access control and logging requirements.

Another method to prevent unauthorised access to the system was the implementation of a dynamic user interface. Every role in the system has different options to select from the UI menu. These options can also change in the case where a user has been assigned other privileges e.g., a nurse temporarily being given doctor privileges in an emergency. This prevents attackers from accessing commands from basic user accounts, therefore preventing a loophole being found in the system to exploit.

System Logs

Logs are an essential part of an AAA system, as they provide a way to record all notable events that take place on the system. The logs can be used to detect unusual behaviour and mitigate any incidents that may take place in the system by providing data on the cause of such events.

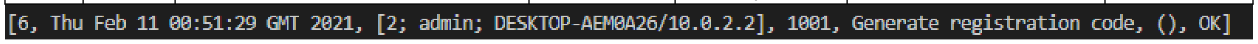
Our system saves and stores every message event (Whenever data is sent from client to server and back). The storing of a log is automated, and logs can in no way be added by a user of any role. Additionally they are stored in an encrypted file, alongside the rest of the database, so the file cannot be directly tampered with either.

Log Structure

Each column in the log file contains crucial information that is necessary for quality monitoring of system events. The date column contains the precise date and time of when an action has been performed. User data (user id, username, IP address) provides information on who was responsible for the performed action, so they can be held accountable. Command id, description, parameters and response from server provide detailed information on the command that was performed, so we know what has happened in the event of issues.

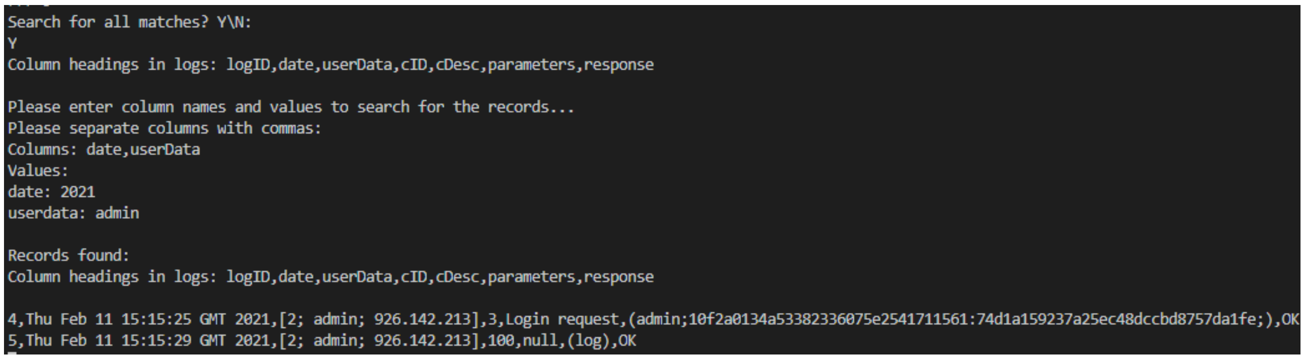
The logs are structured in a way that makes it easy for system administrators to understand the recorded information:

Log id	Date	User data[uid; Username; ip]	Command id	Command description	Command Parameters[p1,p2,..]	Response



```
[6, Thu Feb 11 00:51:29 GMT 2021, [2; admin; DESKTOP-AEM0A26/10.0.2.2], 1001, Generate registration code, (), OK]
```

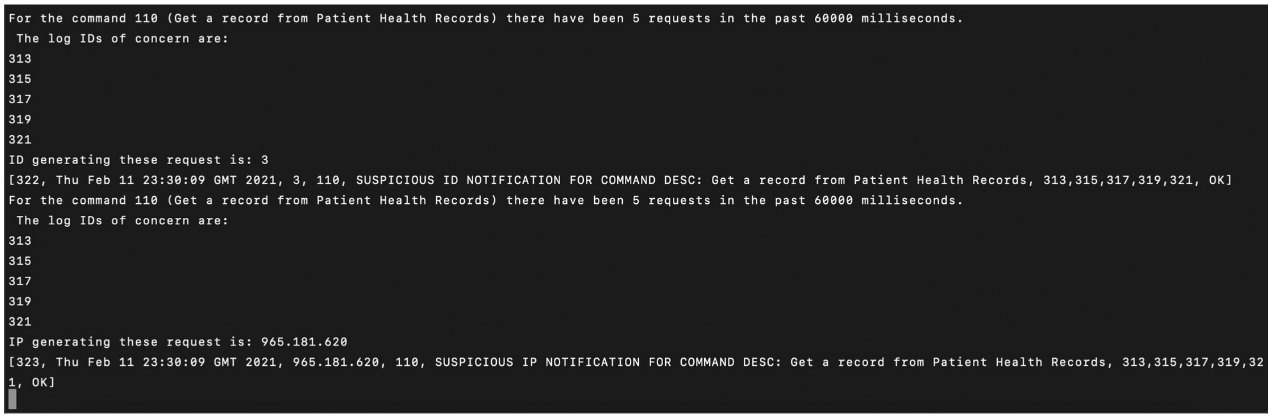
We also implemented filtered search where an admin can find matching log records according to the parameters specified (ex. list all records from year 2020 where username is admin). This feature should make log analysis easier:



```
Search for all matches? Y\N:
Y
Column headings in logs: logID,date,userData,cID,cDesc,parameters,response

Please enter column names and values to search for the records...
Please separate columns with commas:
Columns: date,userData
Values:
date: 2021
userdata: admin

Records found:
Column headings in logs: logID,date,userData,cID,cDesc,parameters,response

4,Thu Feb 11 15:15:25 GMT 2021,[2; admin; 926.142.213],3,Login request,(admin;10f2a0134a53382336075e2541711561:74d1a159237a25ec48dccbd8757da1fe;),OK
5,Thu Feb 11 15:15:29 GMT 2021,[2; admin; 926.142.213],100,null,(log),OK
```

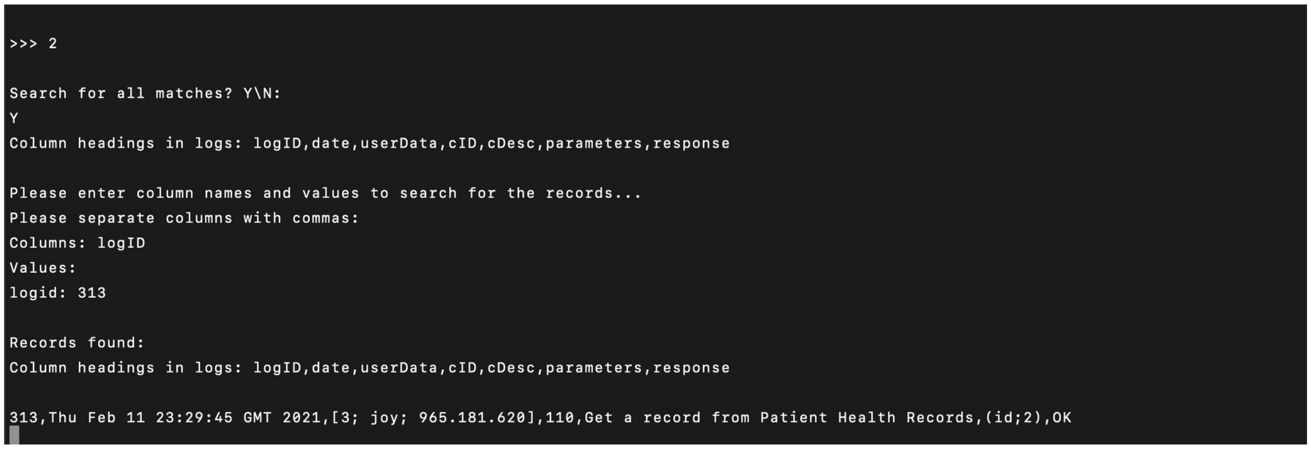
Notifications

A key feature in our system is the notification system. The notification system allows for suspicious behaviour to be detected and therefore indicates to the admin users that some action should be taken. The notifications generated are either IP specific or ID specific and contains the amount of suspicious log entries, the log IDs themselves and the command which is being repeatedly executed. The significance of this is to prevent attacks such as DDOS attacks. Once the system automatically detects that a DDOS attack is taking place, the IP/ID is blocked from accessing the system. This prevents the system from being overloaded and allows for system continuation. The specific notifications of ID and IP are also salient if accountability wants to be promoted in the system. Since we already have the use of digital signatures and message authentication codes, suspicious activity can be linked specifically to an individual, or at least in the case of IP, to a device on a network at a certain location.



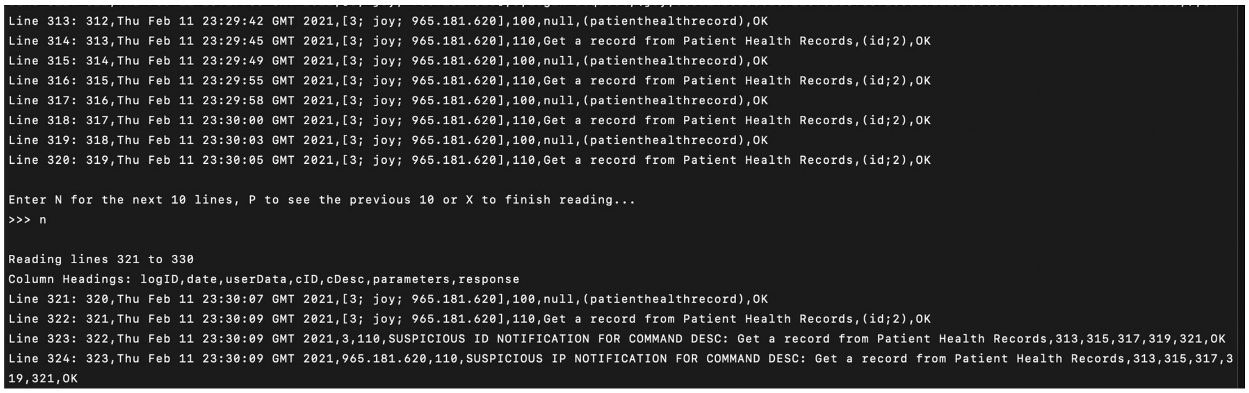
```
For the command 110 (Get a record from Patient Health Records) there have been 5 requests in the past 60000 milliseconds.
 The log IDs of concern are:
313
315
317
319
321
ID generating these request is: 3
[322, Thu Feb 11 23:30:09 GMT 2021, 3, 110, SUSPICIOUS ID NOTIFICATION FOR COMMAND DESC: Get a record from Patient Health Records, 313,315,317,319,321, OK]
For the command 110 (Get a record from Patient Health Records) there have been 5 requests in the past 60000 milliseconds.
 The log IDs of concern are:
313
315
317
319
321
IP generating these request is: 965.181.620
[323, Thu Feb 11 23:30:09 GMT 2021, 965.181.620, 110, SUSPICIOUS IP NOTIFICATION FOR COMMAND DESC: Get a record from Patient Health Records, 313,315,317,319,32
1, OK]
```

When a notification has been produced it is also written into the log. This allows for a thorough review of user action based on the IP or the ID which caused the notification to be made. Using the search function as described before, the admin can search for a specific log ID within the logs and a result like the one below will be shown.



```
>>> 2

Search for all matches? Y\N:
Y
Column headings in logs: logID,date,userData,cID,cDesc,parameters,response

Please enter column names and values to search for the records...
Please separate columns with commas:
Columns: logID
Values:
logid: 313

Records found:
Column headings in logs: logID,date,userData,cID,cDesc,parameters,response

313,Thu Feb 11 23:29:45 GMT 2021,[3; joy; 965.181.620],110,Get a record from Patient Health Records,(id;2),OK
```

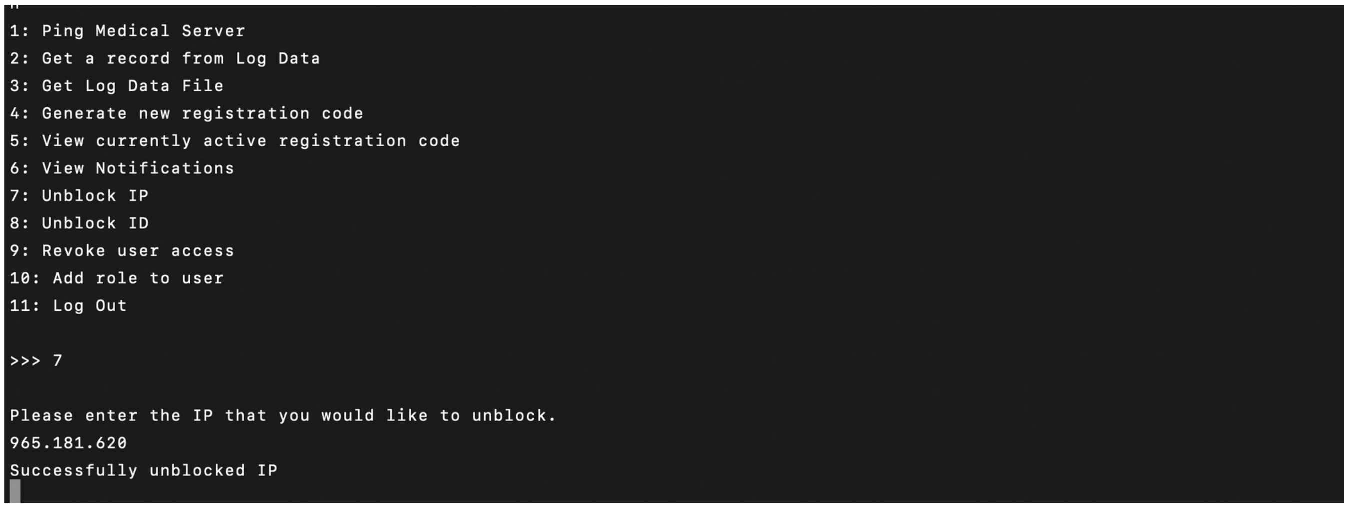
The admin can also look in the log file for the series of logs that are associated with the suspicious activity:



```
Line 313: 312,Thu Feb 11 23:29:42 GMT 2021,[3; joy; 965.181.620],100,null,(patienthealthrecord),OK
Line 314: 313,Thu Feb 11 23:29:45 GMT 2021,[3; joy; 965.181.620],110,Get a record from Patient Health Records,(id;2),OK
Line 315: 314,Thu Feb 11 23:29:49 GMT 2021,[3; joy; 965.181.620],100,null,(patienthealthrecord),OK
Line 316: 315,Thu Feb 11 23:29:55 GMT 2021,[3; joy; 965.181.620],110,Get a record from Patient Health Records,(id;2),OK
Line 317: 316,Thu Feb 11 23:29:58 GMT 2021,[3; joy; 965.181.620],100,null,(patienthealthrecord),OK
Line 318: 317,Thu Feb 11 23:30:00 GMT 2021,[3; joy; 965.181.620],110,Get a record from Patient Health Records,(id;2),OK
Line 319: 318,Thu Feb 11 23:30:03 GMT 2021,[3; joy; 965.181.620],100,null,(patienthealthrecord),OK
Line 320: 319,Thu Feb 11 23:30:05 GMT 2021,[3; joy; 965.181.620],110,Get a record from Patient Health Records,(id;2),OK

Enter N for the next 10 lines, P to see the previous 10 or X to finish reading...
>>> n

Reading lines 321 to 330
Column Headings: logID,date,userData,cID,cDesc,parameters,response
Line 321: 320,Thu Feb 11 23:30:07 GMT 2021,[3; joy; 965.181.620],100,null,(patienthealthrecord),OK
Line 322: 321,Thu Feb 11 23:30:09 GMT 2021,[3; joy; 965.181.620],110,Get a record from Patient Health Records,(id;2),OK
Line 323: 322,Thu Feb 11 23:30:09 GMT 2021,3,110,SUSPICIOUS ID NOTIFICATION FOR COMMAND DESC: Get a record from Patient Health Records,313,315,317,319,321,OK
Line 324: 323,Thu Feb 11 23:30:09 GMT 2021,965.181.620,110,SUSPICIOUS IP NOTIFICATION FOR COMMAND DESC: Get a record from Patient Health Records,313,315,317,3
19,321,OK
```

We can see in the screenshot above that the log for suspicious activity was created directly after the last 110 command execution – the user accessed the patient health record five times within one minute.

If the admin decides that the actions are nothing to be worried about or has received confirmation that this was normal behavior, then the IP/ID can be unblocked. The screenshot below shows the IP being unblocked for the examples shown above.



```
1: Ping Medical Server
2: Get a record from Log Data
3: Get Log Data File
4: Generate new registration code
5: View currently active registration code
6: View Notifications
7: Unblock IP
8: Unblock ID
9: Revoke user access
10: Add role to user
11: Log Out

>>> 7

Please enter the IP that you would like to unblock.
965.181.620
Successfully unblocked IP
```

Conclusion

In the end, we developed a system with which we were quite happy. The system is secure, we've worked well as a team and pushed ourselves to find some more ambitious solutions to problems than we might have normally.

Of course, as with all things, there are areas for improvement, and were we to continue working on the project there are a few things we'd hope to achieve. Firstly, whilst the code functions well when used as intended, it can be a little bit awkward if the incorrect data is entered, so adding code to account for that would be desirable, but more significantly we would've liked to have ensured that the IT Administrator got notifications of suspicious activity straight to their client, rather than when they check the server, as well as implementing detection for users using VPNS. Furthermore, it would be a help to the ease of recovery of the system if we had made use of replicas in the server, however we did not feel that was in scope, so we did not pursue it for this.

Despite the things we wished we had done extra, it is with great satisfaction that we conclude this project, and we hope that all of the lessons we learned will be useful in the future.

Members' Contributions

REMOVED

Week 11 – Writing the Abstract/Introduction and fleshing out the Authorisation section (system users, system data).

Week 12 – Setting up a Client and Server system and defining and creating the message structure. Also doing some document tidying and writing up the work done on the system.

Week 13 – Encrypting Storage and Extending Storage Operations to allow for deleting and editing data. In the document, writing up explanation of storage encryption method used and why.

Week 14 – Implementing user interactions with storage from client side, merging file encryption into project and implementing backup functionality as well as encryption key changing. Writing up all these changes and decisions into the document.

Week 15 – Writing readme file and conclusion plus submitting to moodle.

REMOVED

Week 11 –

Document Contributions:

- Initial Functional/ Non-functional requirements.
- Planned the authentication section with research into the best methods to secure communication. Suggested the use of TLS and digital signatures.

Week 12 –

Code Contributions:

- Initial securing of communication through the generation of session keys.
- Implementation of digital signatures
- TLS emulation (generation of secrets to generate symmetric key and MAC key).
- Hash function implementation for password salting.
- MAC generation and verification on client side and server side.

Week 13 –

Code Contributions:

- Limited the possible operations on UI for each role - creation of a dynamic UI for each role based on the operations they can conduct.
- Completed previous week 12 code on authentication.

Document Contributions:

- Authentication Scheme: How does the authentication scheme work, Why digital signatures? Why MACs? Keys used and encryption types
- Contribution towards password hashing and storing.

Week 14/15-

Code Contributions:

- Warning and alert detection with notifications based on command and message frequency, specific to each command and able to set a threshold based on the number of times the command is requested within a time range as a parameter.
- IP and ID blocking based on notifications that are suspicious.
- Allowing the admin to access and review suspicious activity notifications and unblock users based on the IP or ID.

Document Contributions:

- Notifications
- Contribution towards the securing IT admin privileges section

REMOVED

Week 11 – Completed the planned accountability measures section of the report

Week 12 – Enhanced the login and registration capabilities by checking strength of passwords, checking usernames weren't already in use, storing hashed and salted passwords securely, and logging in securely without exposing the user's password

Week 13 – Improved system architecture by moving system administrator operations off the server and onto a new admin client

Week 14/15 – Restricted consecutive failed login attempts to prevent brute force attacks, added functionality to the user panel for the system administrator to add/revoke roles, and wrote in the report about the part of the system I worked on

Martynas Dabravalskis

Week 11 – Protection against attacks section.

Week 12 – Multifactor Authentication. Implementing TOTP on the server side as well as providing UI functionality (registration/login).

Week 13 – Access Control. A class for role storage and role-based access control. Implementing authorization on the server side.

Week 14/15 – Storing logs. Saving all message events in a logfile. Log formatting to make logs human readable. Viewing/filtering multiple records. Tidying code (Add constants class). Report writing, bug fixes. Added Windows batch scripts for compiling and running the system.