Identity and Access Management Combining Role-Based Access Control and Attribute-Based Access Control

anonymous
Department of Computer Science
Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, NY 14586
anonymous@rit.edu

***Abstract*—Online services and cloud computing allow users to store and access data with their online accounts. This introduces cybersecurity threats such as account hijacking, unauthorized access and insider attacks, which steal valuable data. A secure identity and access management system, which is also called authentication and authorization, is one of the most effective ways to protect data. This project implemented a secure authentication module using Bcrypt algorithm and an efficient and flexible access control module by combining role-based access control and attribute-based access control. The result shows that it takes a similar amount of effort to implement a secure and insecure authentication module and thus the existence of insecure authentication module is due to lack of awareness instead of technical difficulty. Also, the composite access control module combines the advantages of role-based access control and attribute-based access control. The implemented access control model also shows advantages of easy to understand and flexible to configure compared to other models.**

***Index Terms*—IAM; RBAC; ABAC; Access Control; Authentication; Authorization; Cybersecurity**

I. Introduction

With the fast development of information technology, a lot of companies are moving their data from physical media into digital media. This digital transition allows valuable information to be quickly found and processed and makes information sharing a lot easier, which provides a lot of value to many industries, such as improving healthcare quality and lowering costs in the healthcare industry [1]. Although the digital transition greatly improved the productivity, it also brings cybersecurity concerns that valuable digital information can be accessed by users who should not access the information. Cybersecurity risk is becoming more important with the fast development of cloud computing. The advantage of cloud computing is that computer resources, especially data storage and computing power, are accessed on demand, and therefore, cloud computing saves cost. The cost-saving is based on the fact that hardware resources on the same physical machine is shared between untrusted tenants. This introduces the likelihood that a tenant may accidentally access data belonging to other tenants. Another advantage of cloud computing is that it allows people to connect from anywhere around the world with their username and password. However, this also increases the chances of cloud resources being misused by illegal users [2].

The type of cybersecurity threats can be mainly divided into the following types: account hijacking, unauthorized access, and insider attack [3]. Account hijacking is the process of stealing a customer's account and then using the account to access resources. One common reason for account hijacking is password leakage [4]. Unauthorized access is the situation where the flaw in applications could allow attackers to access other users' data. Insider attacks are illegitimate accesses to resources by persons who have some legitimate access to other information of an organization [5]. The access can be unintentional by the insiders who don't have a malicious intent or intentional by insiders who know they shouldn't access the information by the policy. These cybersecurity threats not only put the value data into risk, but also may put the organizations into legal responsibility. There are many laws that require organizations to securely protect their data [6]. For example, the Health Insurance Portability and Accountability of 1996 (HIPAA) requires that access control systems must be implemented to protected health information.

Identity and Access Management (IAM) system is one of the most important methods to defend against the cybersecurity threats discussed above [7]. Identity management, which is also frequently called authentication, is the process to verify that the user is trustworthy. The most common method for authentication is online login with username and password. The system checks the password to make sure it matches the password with the username and then believes the action is performed by the user. Access management, which is also often called access control, is the process of determining whether to allow or deny access to a resource. A properly implemented IAM system should be able to reduce the chance of account hijacking by identity management and reduce the chance of unauthorized access and insider risk by access management.

Although there have been many studies about storing password securely, many online services still store passwords as cleartext or hash without salt, which causes frequent password

breaches [8]. Another challenge is that the widely studied traditional access control models, such as Discretionary Access Control (DAC), Mandatory Access Control (MAC), Role-Based Access Control (RBAC), and Attribute-Based Access Control (ABAC), do not meet the requirements for access controls in today's environment [9]. There are studies about advanced or hybrid access control models, such as Bi-Layer Access Control [10], Hybrid Role and Attribute Based Access Control (HRABAC) [11] etc. Although these advanced models have many advantages, they are complex and need a lot of knowledge to fully understand them.

The hypothesis of this project was that it was possible to implement an efficient and flexible access control module to combine RBAC and ABAC. To support the hypothesis, the project implemented a salted hash password store system and compared the effort taken with an unsafe system. The project also implemented an example that used the flexible access control system mixing different access control models and then evaluated the results to establish that the hypothesis was valid.

In section 2 of the paper, I present the background and related works for this project. In section 3, I describe the design ideas of this project. Section 4 and section 5 are the implementation and results. Section 6 and 7 concludes the project and presents future works.

II. Background

Although there are many methods for authentication, such as multifactor authentication by text message, biometrics authentication and so on [7], the username and password are still the most used method. Al-Aboosi, Broner *et al.* [8] summarized the techniques used to store usernames and passwords and the attacks to steal passwords. Passwords are usually stored together with the usernames in databases such as MySQL etc. There are mainly three types of password storage: cleartext, hashed without salt, or hashed with salt. Store the password as cleartext is the most basic and least secure password storage and it has a big security risk that the software engineer with access to the database can directly read the users' passwords [8]. A more secure way to store password is using hashing algorithms to transform the password into data that cannot be converted back. The authentication process is to hash the user provided password using the same algorithm and compare it with the stored hash values. This way, people with access to the database cannot reverse the hash value into the original password and therefore they cannot directly get the original password. However, because passwords are usually between 5 to 12 characters [4], hackers usually precompute the hash values for common passwords and store them in a table, which is usually called "rainbow table", then the hackers can use reverse lookup to find out the original password [8]. The modern best security practice to store password requires adding some salt rounds when hashing a password to add entropy [8]. The hashing with salt method prevents attacks from "rainbow table" because the salt are large random numbers, and it makes it hard to precompute the hashes for all possible salts. Passwords hashed with salt still have the risk of brute force attacks where the hackers try to compute the hash with brute force methods. Ideally, an attacker should take a lot of time to crack a hashed password. There are many hashing algorithms such as MD5, SHA1, bcrypt, and Argon2. Bcrypt is recommended because it is an adaptive method where the hash time doubles when the cost parameter is increased by 1 [4].

In access control, the most basic access control models are Discretionary Access Control (DAC), Mandatory Access Control (MAC), Role-Based Access Control (RBAC), and Attribute-Based Access Control (ABAC). There are related works the explains the advantages and disadvantages [9], [2]. In DAC, the access permissions are directly assigned to specific users. This makes it too hard to use when a company has too many resources and too many users. In MAC, a security level is assigned to each subject and each object, and the subject can access an object if the subject's security level is higher than the object's security level. The MAC model doesn't support duty separation and RBAC is made of three parts, a set of users, a set of roles, and a set of permissions. Users are assigned into roles, and a role have some permissions that allows some actions. Hence, in RBAC, users get permission for their action by joining the roles with permission. RBAC has the advantage of simpler access administration and user permission review. It also has several drawbacks. RBAC requires an expensive process to define roles. RBAC usually grants more permissions than the role requires. RBAC also is static and inflexible in dynamically changing environments [10]. In ABAC, the permission to perform an action is determined by access policies that use the attributes from subjects (such as city of user), the attributes from objects (such as size of file), and attributes from the environment (such as time of day). ABAC has the advantage of its fine-grained access. ABAC has a disadvantage that is it very complicated because of the large number of attributes and policies. Overall, DAC and MAC are not well suited to digital businesses today and RBAC and ABAC are the main access control models used.

There are studies trying to combine the advantages of RBAC and ABAC and avoid their disadvantages. Bi-Layer Access Control (BLAC) was developed by Suhair Alshehri and Rajendra K. Raj [10] to combine RBAC and ABAC with fewer drawbacks. BLAC uses pseudorole as the first layer and attributed based policy as the second layer and access is allowed only if both checks work. Hybrid Role and Attribute Based Access Control (HRABAC) was proposed by Maria Penelova [11] as an easy configurable, fine-grained system that supports role hierarchy systems. HRABAC uses active subject or active role to filter access and then it allows access when one of the RBAC or ABAC policy matches. The Role-Attribute Assignment Based Access Control (RAAB-AC) model [2] provides both the flexibility and dynamicity as ABAC and the simplicity and security as RBAC. In the RAAB-AC model, subject attributes and object attributes are assigned to roles and roles are assigned with permissions. During the evaluation of the access, the subject need to fulfill all user attributes to

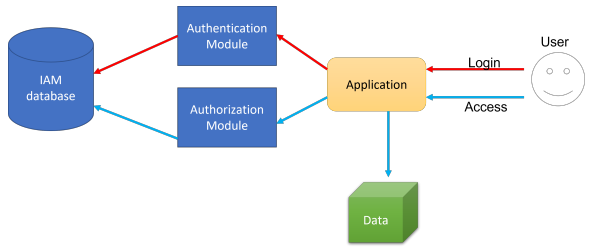

Fig. 1. Structure of the application.

be in the role and the object also need to fulfill all the object attributes to be actually in the role and then the role needs to have the permission so that the subject can access the object.

III. Design

A. Identity Management / Authentication

For the implementation of authentication part, I developed an authentication module using Bcrypt as the hashing algorithm to store passwords securely.

1) Bcrypt algorithm cost selection: Bcrypt algorithm has two advantages. The first one is that it uses a secure random number as the hash salt. The second advantage is that it has an adjustable cost which can be used to promote key strength and slow down calculation to prevent brute force attacks. Brute force attack is a brute force trials of hashing the potential passwords to match the hashed value that the hacker acquired. By increasing the cost and slowing down the calculation speed, the hacker needs more time to try potential passwords before finding a match. This adaptability is what allows us to compensate for increasing computer power with hardware improvement. In this process, I ran an experiment to encrypt password at different costs and then chose a cost that took the encryption process around 1 seconds.

2) Authentication database table: I created a database table with two columns. One is for usernames and the other one is for passwords which are hashed with salt by Bcrypt algorithms. When creating a user, the username and the hashed password are written into the table as a row. When a user logs in, we hash the password the user provided the same way when the user was created and then we check if the hash matches the stored hash value. This way, we don't store the user's password directly and can still verify the password.

B. Access Control / Authorization

For the authorization part, I created a database table with 3 columns. The first column is action. The second column is role which is used for RBAC. The third column is policy which is a text input in XML format that represents ABAC's policy. Both the second and the third column can be null. If the policy is null, then the model is a RBAC. If the role is null, then it's ABAC. If role and policy are both not null, then it's hybrid where it checks the role first and then checks ABAC's policy by evaluating the attribute relationship.

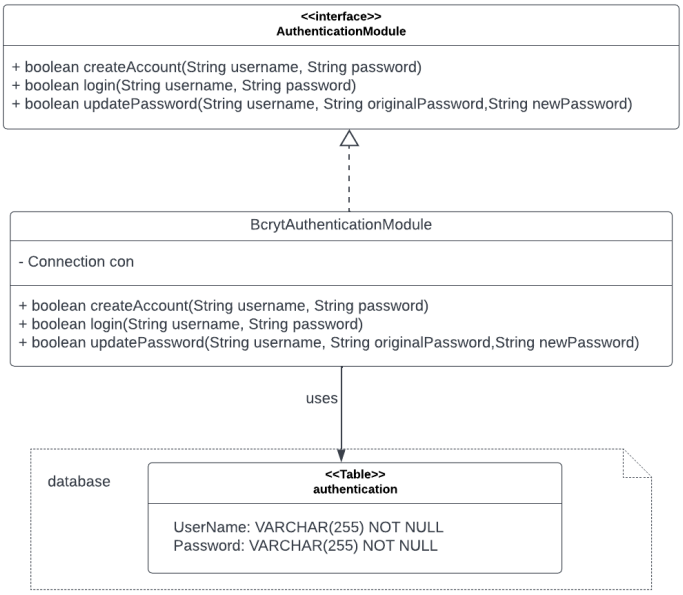

Fig. 2. The implementation of the authentication module including class methods and the database table.

C. Application

In this part, I created an application to show how the IAM system worked. Figure 1 is a diagram of the relationship of the application and the authentication and authorization module and the IAM database. The application is a healthcare system where there are managers, doctors, and patients. Managers can view all patient records and use RBAC. Doctors can only view patient records that they are the doctors and should use hybrid AC. Patients can view anything that are for them and uses ABAC.

IV. Implementation

A. Hardware and software used

In the implementation and test, the hardware used is an HP ENVY x360 laptop with Intel(R) Core(TM) i7-1065G7 CPU @ 1.30 GHz, 4 core/8 threads, 8 MB Cache. The machine's RAM has 12 GB DDR3@3200MHz. The code are mainly written in Java (temurin-18.0.1) with IntelliJ Community Version 2021.1 as the IDE. In addition to the libraries provided by Java, JBcrypt library from mindrot is used as the bcrypt algorithm implementation [12]. The database used to store the authentication and authorization tables is MySQL Community Server (version 8.0). The database setup are performed with the MySQL 8.0 Command Line client.

B. Authentication

2 shows the implementation of the authentication module. The database table named authentication has two fields: UserName and Password. The BcryptAuthenticationModule implements three methods: createAccount, login, and updatePassword. The createAccount method hashes the user's

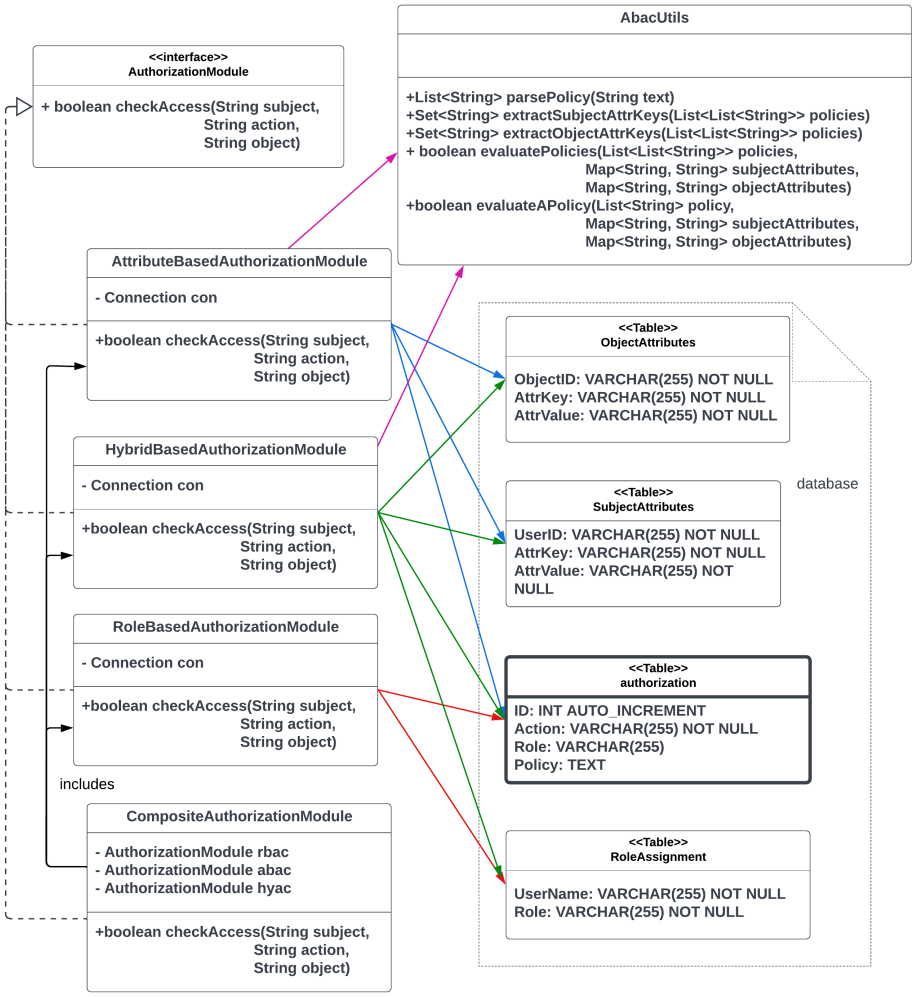

Fig. 3. The implementation of the authorization modules including class methods and the database tables.

password using Bcrypt algorithm and writes the hashed password into the database. The login method takes the hashed password from the database, and checks to see if the provided password can be hashed into the same hashed password. The updatePassword method first checks if the originalPassword can be hashed into the hashed password in the database, and then hashes the new password using Bcrypt and then updates the database with the new hashed password.

Two similar authentication modules are also implemented with one using SHA256 and another one using clear text.

C. Authorization

Figure 3 shows the implementation of the authorization module. The most important part of the implementation is database table name authorization. The authorization table has 3 important fields: Action, Role, and Policy. Action is a string that represents the action the subject wants to do, and it must be provided. Role is a string used to say which role has access to perform the action if provided. Policy is a text representation of the ABAC's policy if provided. The table can work in 3 modes: If only Role is provided and Policy is NULL, then it is RBAC; if only Policy is provided and Role is NULL, then it is ABAC; if both Role and Policy are provided, then it is hybrid access control.

The Policy text should be an XML text and using the format like

<policy><rule>
subject.name = object.ownerName
</rule></policy>

Basically, a policy is made up of rules and each rule is a boolean equation about attributes.

An interface AuthorizationModule is defined to abstract the access control. It contains only one method checkAccess with 3 arguments: subject, action, and object, meaning which subject wants to do what action to which object. For RoleBasedAuthorizationModule, it joins the RoleAssignment table that tells which user has which role, with the authorization table to check if a user has access or not. For AttributeBasedAuthorizationModule, it uses the subjectAttributes and objectAttributes table, together with the authorization table to

check access based on attributes. The subjectAttributes and objectAttributes tables store each user's or object's attributes in the form of key value pairs where the key is the attribute name, and the value is the attribute value. This makes it very easy to get the attribute value based on the attribute name. During the evaluation, the Policy texts are read from the authorization table, then the rules of ABAC are parsed from the text, then the attribute values are obtained from the subjectAttributes and objectAttributes tables, and finally the attribute values are used to check each policy to see if the user has the action on the object or not. An AbacUtils class was created to provide functions to make the ABAC check easier. Without the AbacUtils class, the code looked very confusing when they were all in one function. The HybridAuthorizationModule first uses the RoleAssignment table to take out the Policy from the authorization, and then the policies are checked. Finally, a CompositeAuthorizationModule is implemented using the three modules. The access check first checks RBAC, then checks hybrid AC, and finally checks ABAC because ABAC needs to check policy one by one which takes time, so it is the last one to check.

V. Analysis

A. Bcrypt hashed value

Fig. 4. The password hash format from Bcrypt algorithm.

Figure 4 shows the hash of the password "A1b2C3d4E5" hashed using BCrypt. As shown, the result contains the version, the cost, the salt, and the hash of the hashed password. BCrypt is safe against rainbow table attacks because the algorithm hashes the password with salts and we can see that the online rainbow table cannot crack the password protected by BCrypt, shown in the figure 5. However, for password hashed using MD5 and SHA256 without salt, the online rainbow table can easily figure out the hash algorithm used and the original password.

Table I shows that when using Bcrypt, the same password can be hashed into totally different values, which is another advantage of hashing with salt. But using SHA256 without salt, the hashes are the same, making it easy to guess that the original passwords are also the same. When using clear text, the password is directly shown, which makes it the least secure password storage method.

B. Bcrypt cost selection

Another advantage of Bcrypt is that its cost can be changed so that it can defend against brute force attacks. Typically, the longer it takes to encrypt a password, the longer it takes a hacker to hack a password using brute force methods. Table II shows the amount of time it takes to hash a password using Bcrypt with different costs. As the table shows, the time taken to hash a password increases exponentially with the increase of cost. On the hardware used in this project, the encryption takes 5 minutes at a cost of 22 and the cost of Bcrypt can be at most 30. Therefore, Bcrypt can still work against a significant computing power increase in the next following years. Because the longer time means higher security, but I don't want customers to wait too long for login, therefore I can pick a time at around 1 second. As a result, I picked the cost of 13 as the cost to use for Bcrypt encryption.

C. Comparison of secure and insecure authentication implementation

Table III shows a comparison of the implementation of the authentication module using Bcrypt, SHA256 and clear text. The implementation of a secure authentication module using Bcrypt uses 83 lines of code, while the implementation of the least secure authentication module using clear text takes 80 lines of code. Moreover, the number of lines of code (7 for SHA256 and 6 for clear text) that are different is also very small compared to the total number of lines. The major difference is the function call of the Bcrypt method to hash the password. Therefore, these data support the hypothesis that implementation of a secure authentication system takes a similar amount of effort compared to an insecure system. The insecure systems that lead to password breaches are mainly due to the lack of awareness instead of the efforts needed to implement a secure system.

D. Role-based authorization

The role-based authorization module works by looking up the role assignment through the RoleAssignment table and then using the assigned role to gain access to the action. In an example role assignment where user1 is assigned to role1, and role1 is assigned to have read action without any policy assigned, the result that user1 can get access to perform the read action shows the RBAC implementation works correct.

E. Attribute-based authorization

In attribute-based authorization, the permission to perform the action is checked by the policies. In the example where the policy is

<policy><rule>
subject.name = object.ownerName
</rule></policy>

the subjectAttributes table stores user1:name:Alice and the objectAttributes table stores object1:ownername:Alice, user1 can correctly gain access to read object1. The workflow is that when the policy is parsed, I need to check if the subject's name is the same as object's ownername. Because the subject is user1 and the object is object1, I can get the subject's name as Alice and the object's ownername is also Alice. Because they are the same and therefore user1 can read object1.

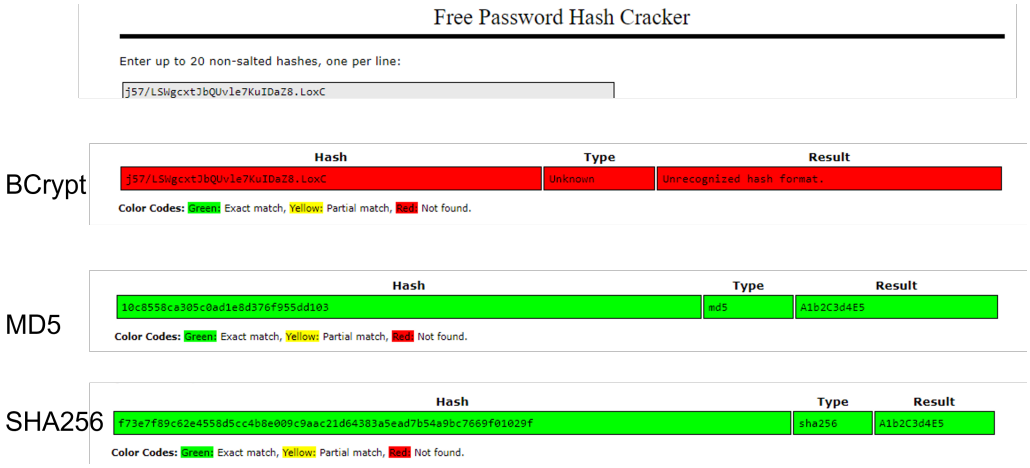

Fig. 5. The rainbow table attack of password hash by different algorithms.

TABLE I
PASSWORD STORAGE COMPARISON BETWEEN AUTHENTICATION MODULES

Password	Bcrypt	SHA256	Cleartext
11111	$2a$13$FkEt9UHHxzekan/3w/TH4eoD3uF DTZnQwx8idBdabyNaItNFBuJ6a	d17f25ecfbcc7857f7bebea469308be0b25809 43e96d13a3ad98a13675c4bfc22	11111
11111	$2a$13$eDmdbk49pPeN6smG6f7JrumzqIR O6vdyEsPfNfFoqKbMYaRVnhfNm	d17f25ecfbcc7857f7bebea469308be0b25809 43e96d13a3ad98a13675c4bfc22	11111
11111	$2a$13$pnyqCqPf2wTOS1Siz56.detaGaG p6fDha4NsWy1tH1Hzv1867XDOG	d17f25ecfbcc7857f7bebea469308be0b25809 43e96d13a3ad98a13675c4bfc22	11111

TABLE II
ENCRYPTION TIME CHANGE WITH THE CHANGE OF COST FOR BCRYPT ALGORITHM

cost	5	6	7	8	9	10	11	12	13
Time(ms)	7.4	10.4	18.0	33.4	61.2	117.8	218.9	522.8	722.6
cost	14	15	16	17	18	19	20	21	22
Time(ms)	1401.9	2734.4	5619.4	11379.3	22069.8	43822.4	83721.3	163661	325505

TABLE III
COMPARISON OF AUTHENTICATION MODULE IMPLEMENTATION DIFFERENCES.

method	lines of code	different lines of code
Bcrypt	83	0
SHA256	82	7
Cleartext	80	6

F. Hybrid authorization

In the hybrid authorization model, the permission to perform the action needs to meet both the role assignment and the ABAC's policy. An example is that the read permission is allowed if the Role is role1 and the policy is

<policy><rule>
subject.name = object.ownerName
</rule></policy>

The role assignments are user1:role1, user2:role1, user3:role2 and the subject attributes are user1:name:Alice, user2:name:Bob, user3:name:Alice. The objectAttributes table stores object1:ownername:Alice for the object1. Here user1 has access to object1 through the hybrid access control module, but user2 and user3 don't have access. user1 has access because user1 is in role1 and user1's name Alice matches object1's ownername. user2 has no access because its name Bob doesn't match object1's ownername Alice. user3 has no access because user3 is not in role1.

TABLE IV
ACCESS CONTROL INFORMATION USED IN THE HOSPITAL DATA.

id	action	role	policy
1	read	manager	NULL
2	read	doctor	$subject.ID = object.DoctorID$
3	read	NULL	$subject.ID = object.PatientID$

TABLE V
PATIENT VISIT INFORMATION USED IN THE HOSPITAL DATA.

VisitID	PatientID	DoctorID	Date	Description
visit1	patient1	doctor1	10/25/2022	cough
visit2	patient1	doctor2	10/29/2022	flu
visit3	patient2	doctor1	10/27/2022	flu

G. Hospital data management application

I built a simulated hospital data management system to combine the authentication and authorization modules. In the

TABLE VI
USER'S ACCESS OF HOSPITAL DATA.

user	records can read
manager1	visit1, visit2, visit3
doctor1	visit1, visit3
doctor2	visit2
patient1	visit1, visit2
patient2	visit3

application, a user needs to first put in his/her username and password in order to login. After logging in successfully, he/she can see a list of patient visit records. Different users can see different records based on the RBAC, ABAC and Hybrid AC models.

In this simple application, all three access control models are used to control the access of patient visit information. Table IV shows the 3 access control records used. The first record is a RBAC model that any manager can read all information. The second record is a Hybrid AC model that any doctor can only read objects where they are marked as the doctor. The third record uses an ABAC model that someone can read any patient visit information with him/her as the patient.

Table V shows three visit records including information such as DoctorID, PatientID, Date and Description for the visit. Table VI shows five users of the hospital system and the records that they have access to. manager1 has access to all records based on authorization rule 1 which uses RBAC only that any manager has read permission. doctor1 can access visit1 and visit3 and doctor2 can access visit2. The access is allowed by authorization rule 2 which uses Hybrid AC such that doctors can only access visits that they are listed as doctors. The authorization rule 3 which is ABAC only allows patients to read visit records about them so patient1 can access visit1 and visit2 and patient2 can access visit3.

In this application, a central authorization database table is used to store a combination of RBAC, ABAC and Hybrid AC authorization rules and these authorization rules can work together so that the most appropriate authorization rules can be used for the selected scenarios. The results show that the composite authorization model achieves high flexibility and configurability and reduces the complexity of the authorization rules. Moreover, the model also provides a smooth transition that RBAC and ABAC can be easily changed into Hybrid AC, therefore, this composite model is friendly to users who have less experience in access control and meets the needs of access control experts who need granular access control.

H. Comparison of access control models

In this project, in addition to the implementation of the common RBAC and ABAC, the main innovation is the model of the Hybrid AC and the Composite AC (which uses RBAC, ABAC and Hybrid AC). This innovation is built based on research efforts which studied similar models such as the BLAC model by Suhair Alshehri and Rajendra K. Raj [10] and the RAAB-AC model by Sara Alayda et al [2]. Table VII shows a comparison of the key characteristics. Overall, the Hybrid AC (HyAC) and Composite AC (CompAC), like the BLAC and RAAB-AC models, combine the advantages of RBAC and ABAC in permission granularity and configurability. The major difference comes from the knowledge requirement to understand the model and flexibility of the models. Most of the advanced models published in peer reviewed paper, such as BLAC and RAAB-AC, although with high security, are usually complicated and require a lot of knowledge for understanding. On the other hand, the model from this project only requires knowledge about RBAC and ABAC. Also, because the models usually define many required concepts, such as the pseudorole in BLAC and the attribute-role assignment in RAAB-AC, the flexibility of those models are reduced. For example, some ABAC only access may need additional dummy roles created in the RAAB-AC model to simulate ABAC. The composite AC model implemented in this project allows a simple rule of RBAC only (or ABAC only) to simplify the configuration.

VI. CONCLUSION

In this work, I implemented a secure authentication module using Bcrypt encryption algorithm and compared the efforts between secure and insecure authentication modules. I also implemented a composite access control module which uses RBAC, ABAC and a hybrid AC module to validate the hypothesis that an efficient and flexible access control module can be implemented.

The current status of the project is that all the scheduled work is finished, and the hypotheses are validated. Specifically, the composite access control module which uses RBAC, ABAC and a hybrid AC module is fully implemented using Java and MySQL database. A demo application to simulate hospital data shows that the access control module is very flexible and the different AC models can be used together to improve efficiency and flexibility. In addition, the implementation of the authentication module shows that it takes similar amount of effort to implement a secure and insecure authentication system and therefore the existence of insecure authentication system are due to the lack awareness instead of technical difficulty.

VII. FURTURE WORK

During the implementation, the most challenging work is how to implement the ABAC policy evaluation code. The challenge comes from the fact that the policy is defined as strings but the attribute values for different subjects and different objects are different. In this project's implementation, a database table is used to store the key value pairs for attributes, which makes the efficiency for policy evaluation great. But it only works for the evaluation of policy like $A = B$. It does not work for greater than, less than, comparisons, and does not work for environment attributes such as time of day. Future work can be done to make the policy check library more complete and work for different types of policy rules.

ACKNOWLEDGMENT

I would like to thank Prof. Rajendra K. Raj for his help throughout the project. His excellent courses in cybersecurity

TABLE VII
COMPARISON OF CHARACTERITICS OF ACCESS CONTROL MODELS.

Characteristics	RBAC	ABAC	**HyAC**	**CompAC**	BLAC [10]	RAAB-AC [2]
Granularity	Low	High	High	High	High	High
Configurability	High	Low	Medium	High	Medium	Medium
Understanding	Easy	Easy	Easy	Easy	Hard	Hard
Flexibility	Low	Medium	Medium	High	Medium	Medium

and big data analytics helped a lot in inspiring the original idea of the project. His expertise in these fields guided the direction of this project and ensured the finish of this project. I would also like to thank Prof. Carlos R. Rivero for his feedback and Prof. Hans-Peter Bischof for the guidelines for the project.

REFERENCES

[1] S. Alshehri, S. Mishra, and R. K. Raj, "Using access control to mitigate insider threats to healthcare systems," in *2016 IEEE International Conference on Healthcare Informatics (ICHI)*, 2016, pp. 55–60.

[2] N. Almowaysher, M. Humayun, and N. Zaman, "A novel hybrid approach for access control in cloud computing," *International Journal of Engineering Research and Technology*, vol. 13, pp. 3404–3414, 01 2020.

[3] R. A. Nafea and M. Amin Almaiah, "Cyber security threats in cloud: Literature review," in *2021 International Conference on Information Technology (ICIT)*, 2021, pp. 779–786.

[4] N. Katrandzhiev, D. Hristozov, and B. Milenkov, "A comparison of password protection methods for web-based platforms implemented with php and mysql," *International Journal on Information Technologies & Security*, vol. 11, no. 2, 2019.

[5] I. Homoliak, F. Toffalini, J. Guarnizo, Y. Elovici, and M. Ochoa, "Insight into insiders and it: A survey of insider threat taxonomies, analysis, modeling, and countermeasures," *ACM Computing Surveys (CSUR)*, vol. 52, no. 2, pp. 1–40, 2019. [Online]. Available: https://doi.org/10.1145/3303771

[6] S. P. Mulligan, W. C. Freeman, and C. D. Linebaugh, "Data protection law: An overview," *Congressional Research Service (March 2019)*, 2019. [Online]. Available: https://crsreports.congress.gov/product/pdf/R/R45631

[7] I. Indu, P. R. Anand, and V. Bhaskar, "Identity and access management in cloud environment: Mechanisms and challenges," *Engineering science and technology, an international journal*, vol. 21, no. 4, pp. 574–588, 2018.

[8] A. F. Al-Aboosi, M. Broner, and F. Y. Al-Aboosi, "Bingo: A semi-centralized password storage system," *Journal of Cybersecurity and Privacy*, vol. 2, no. 3, pp. 444–465, 2022. [Online]. Available: https://www.mdpi.com/2624-800X/2/3/23

[9] Z. N. Mohammad, F. Farha, A. O. Abuassba, S. Yang, and F. Zhou, "Access control and authorization in smart homes: A survey," *Tsinghua Science and Technology*, vol. 26, no. 6, pp. 906–917, 2021.

[10] S. Alshehri and R. K. Raj, "Secure access control for health information sharing systems," in *2013 IEEE International Conference on Healthcare Informatics*, 2013, pp. 277–286.

[11] M. Penelova, "Hybrid role and attribute based access control applied in information systems," *Cybernetics and Information Technologies*, vol. 21, no. 3, pp. 85–96, 2021.

[12] org.mindrot. org.mindrot>>jbcrypt>>0.4. [Online]. Available: https://mvnrepository.com/artifact/org.mindrot/jbcrypt/0.4