

**Securing A Moral Distress Reporting and Analysis System With A Role-Based Access  
Control Approach**

A Technical Report Submitted to the Computer Science Department  
Presented to the Faculty of the School of Engineering and Applied Science  
University of Virginia, Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree of Bachelor of Science in Computer  
Science, School of Engineering

**Neha Krishnakumar**

Spring 2024

On my honor as a University Student, I have neither given nor received unauthorized aid on this  
assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Main Advisor Kevin Sullivan, Department of Computer Science  
Secondary Advisor Angela Orebaugh, Department of Computer Science

## **Abstract**

The University of Virginia's Computer Science Department, in conjunction with the University of Virginia School of Nursing, developed a system called the Moral Distress (MoD) system, designed to report, document, and remedy moral distress among healthcare providers. As a result, this system features sensitive data and needs proper security. This security comes in the form of the current state of development, integrating security and design through experimental concept-based software engineering, and also comes in the form of access control design and implementation. Access control measures were iteratively developed using role-based access control (RBAC). Along the way, these measures were tested, and the results were documented in this report. Finally, the system is evaluated through a series of ongoing studies.

## **Problem**

The central problem with the associated software system is that the system has critical data and thus needs to be secured. Multifarious methods can be involved in securing a software-based system. Some methods involved in securing a software-based system, through a technical approach, are using password policy, cipher suites, and secure network protocols. These methods will be outlined in this section, followed by what cloud security has to offer to remedy these problems. Finally, a portion of this section will be devoted to addressing the problems associated with cloud security, and specifically access control - of which the solution will form the basis of this report.

### *Password Policy*

The most common model of executing authentication is through a username and password [1]. Password policy can range from involving specific characters and length of a password, or can involve the duration of a password before it is changed and other factors

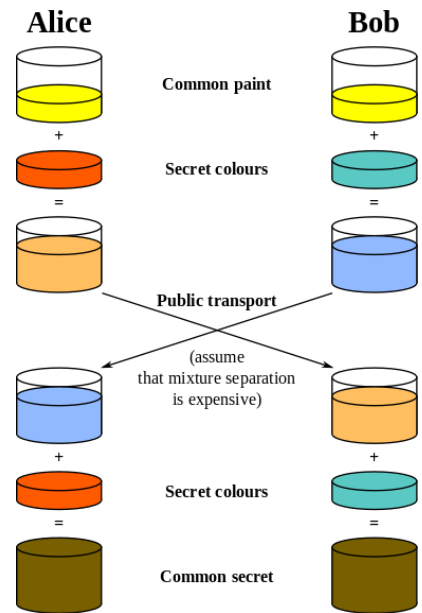
associated with not just its creation [1]. Policy creation is subjective, but formalized standards have been developed for password policy, such as that developed by Shay et. al from Purdue University [1]. This standard involves four stages - creation, memorization and storage, updating, and deletion, and Shay et. al incorporates formal logic to make assertions related to password management. Their work also featured an experiment that assessed the assertions mentioned earlier in creating strong passwords, and the results proved successful but variable based on the input.

Besides the use of formally verified standards for password policy, there also exists industrially viable standards. The National Institute of Standards and Technology (NIST) dictates that passwords must be at least eight characters long and that businesses creating random passwords must make them at least six characters long and can make these entirely composed of numbers [2]. Additionally, verifiers, or the information security apparatus designed to handle passwords, must not give hints to unauthenticated users and must compare passwords against commonly used passwords, repetitive character passwords, and other memorized secrets that are deemed unworthy because of weakness as a secret. Thus, in the realm of research and industry, there exists methods to develop password policy, and thus secure a system.

### *Cipher Suites*

Cipher suites are a group of methods that secure a network connection through the Secure Sockets Layer (SSL) / Transport Layer Security (TLS) [3]. Each cipher suite comprises four different types of algorithms: the key exchange algorithm, the authentication algorithm, the bulk encryption algorithm, and the Message Authentication Code (MAC) algorithm. Each algorithm will be described through an example in each subsequent section.

Key exchange algorithm: One example of a key exchange algorithm is the Diffie Hellman key exchange algorithm. Diffie and Hellman, along with earlier work by Merkle, made advancements in public key cryptosystems and key exchange systems, and the key exchange or key distribution system proposed by Diffie and Hellman involved using independently chosen random numbers as secrets [4]. These users would place their secrets in shared files with their name and address and then would use a formula involving logarithms and modular division to exchange secrets. Ultimately, while the algorithm is simple to develop, it is difficult to decipher due to the difficulty in computing the logarithm of the modular division of the selected prime number,  $q$ . The Diffie-Hellman algorithm, disregarding the mathematics, can be simply understood through the mixed-paint example, shown below in Fig. 1 [5].



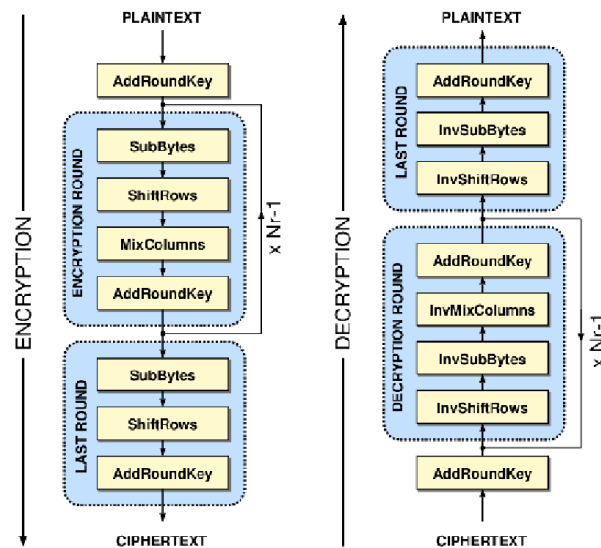
**Fig. 1: Mixed Paint Example Illustrating Diffie-Hellman Key-Exchange Algorithm**

Authentication algorithm: One example of an authentication algorithm is the Rivest-Shamir-Adelman algorithm (RSA). This algorithm allows for the fact that if the encryption key is revealed, the corresponding decryption key will not be revealed [3]. RSA is

based on the difficulty of factoring  $n$ , or the product of two large primes  $p$  and  $q$ , and is simple to implement, with a guide provided in the original paper by Rivest, Shamir, and Adelman.

Bulk encryption algorithm → One example of a bulk encryption algorithm is the Advanced Encryption Standard (AES).

The Federal Information Process Standard Publications (FIPS) 197 developed the standard of AES based on the Rijndael cipher, which is a cipher that translates secrets to cipher text, or text that is intended to be difficult to decipher [6]. AES is a symmetric block cipher, meaning that aspects of the message to become ciphertext are shifted based on certain permutations associated with a key. [6]. Below, in Fig. 2, is an illustration of how AES works [7].



**Fig. 2: AES with Operations Included for Encryption and Decryption**

MAC Algorithm → One example of a MAC algorithm is the Secure Hashing Algorithm- 512 (SHA-512). SHA-512 is part of the SHA-3 family and is specified by NIST in FIPS 180-4 [8].

Specified in the Secure Hash Standard (SHS), the SHA-3 family involves a set of algorithms that produce a message digest, a condensed representation given a message, which can be used to

verify the integrity of a message - or whether a message has been modified or not. Hashing algorithms like these can be used to verify a MAC and is thus a MAC algorithm.

*Secure Network Protocols*

There exists a wide variety of secure network protocols, and if used, can enhance the security of a system. Below, in Table I, is a list of some secure network protocols that are commonly used in industry and research along with information about each protocol [9]. Table II, below Table I, contains a list of some insecure network protocols and information about each protocol.

<b>Port</b>	<b>Application Layer Protocol</b>	<b>Transport Layer Protocol</b>
443	Hypertext Transfer Protocol Secure Sockets (HTTPS)	TCP
53	Domain Name Server/Service (DNS)	UDP and TCP
25	Simple Mail Transfer Protocol (SMTP)	TCP
22	Secure Shell Protocol (SSH)	TCP
135-139 and 445	Remote Procedure Call (RPC)	TCP and UDP
500	Internet Security Association and Key Management Protocol (ISAKMP), Secure Internet Protocol (IPSec), virtual private networks (VPNs)	UDP

**Table I: Secure Network Protocols**

Port	Application Layer Protocol	Transport Layer Protocol
80	Hyper Text Transfer Protocol (HTTP)	TCP
20 and 21	File Transfer Protocol (FTP)	TCP
23	Telnet	TCP

**Table II: Insecure Network Protocols**

*Cloud Security Remedies and Issues*

Because this system lies in the cloud, many of these factors can be easily dealt with. Any system developed on Amazon Web Services (AWS), specifically a system using AWS services like Amazon Cognito, will have access to an immediate password policy through user pools, which will be elaborated on later in this report. This system is no exception. Additionally, this system already utilizes cipher suites in its configuration, specifically TLS-AES-GCM-SHA-256. For proof, see Fig. 2 below, which is a Wireshark network protocol analysis that identifies the cipher suite involved in the system.

```

  ▾ Handshake Protocol: Server Hello
    Handshake Type: Server Hello (2)
    Length: 118
    Version: TLS 1.2 (0x0303)
    Random: 821b5142e9fc1c75584e21388eea8adb824a55be59bff5976180ff66a8c7eb2
    Session ID Length: 32
    Session ID: b0070522b501566ab55a14fa24a1d0c329b7a539ba962ff46642d58d83f96079
    Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)

```

**Fig. 2: Wireshark Network Protocol Analysis with Cipher Suite**

Finally, secure networking protocols are used by AWS, avoiding insecure protocols such as Telnet, for remote login, and File Transfer Protocol (FTP), for transferring files [9].

However, because this system lies in the cloud, there are a whole host of new security issues that must be addressed. According to the Organization of Web Application Security Professionals (OWASP), some of the most common issues include public data storage buckets, unencrypted or poorly encrypted secrets, API keys, and passwords, and vulnerable third-party

resource usage [10]. In the OWASP Cloud Native Application Security Top 10, these examples are grouped into ten categories. One category this report will seek to remedy for the case study system, is authentication and authorization, specifically those associated with Identity and Access Management (IAM) in AWS.

## **Domain**

### *Moral Distress Background*

The domain of this project involves an ethical phenomenon known as moral distress. Felt by many healthcare providers throughout the United States, moral distress involves a feeling that arises when one is faced with the inability to make a decision they feel is morally correct [11]. It involves initial and reactive distress, whereby initial distress happens in real-time, and reactive distress occurs after the fact, and may result in a feeling of moral inadequacy [11]. There is a clear need to identify and remedy this sense of inadequacy, which may cause those in the healthcare profession to leave their jobs, though this is different than stress, burnout, or fatigue [11].

### *Identifying and Remediating Moral Distress*

Many research-validated measures to identify and remedy moral distress among healthcare providers have been created, such as the Moral Distress Thermometer (MDT) by Wocial and Weaver [12], and the Moral Distress Consultancy Service (MDCS) by Hamric and Epstein [13]. These measures may be quantitative, like the MDT, or qualitative, like the MDCS. The MDT and MDCS are embedded into the Moral Distress software-based system that has been developed as part of this project.

## **Software Engineering and Computer Security Context**



The Moral Distress system was built based on a software engineering paradigm known as concept-based software engineering. Developed by Daniel Jackson at MIT, this paradigm features a concept or a unit of functionality with a specific purpose [14]. These concepts “have no visible form”, and exist as part of design and to be implemented in a software-based system [14]. The purpose of a concept is exceedingly important, as misinterpretations of the key concept of a piece of software can lead to disastrous results. For example, a misunderstanding of the use of the trash concept in Dropbox can lead one to delete files that are essential for another team member in a team project [14]. As concepts are also abstract and can thus have multiple implementations, Jackson outlines that developers need to design concepts to be simple and design a user interface that makes sense for each concept [14].

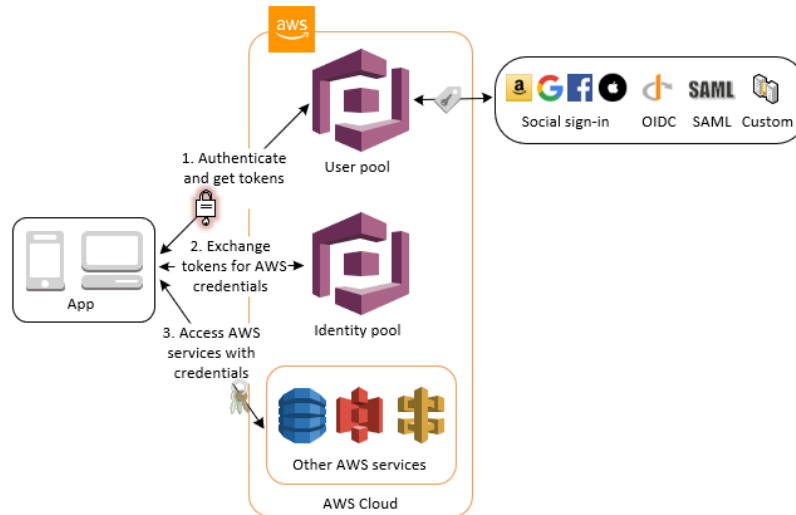
The MoD system utilized a concept-based design and implementation by including four concepts. These concepts are as follows: survey, which includes the reasons for feeling moral distress, thermometer, which includes the earlier described MDT, resiliency resources, which help after a moral distress reading has been submitted, and the auth concept, which is a modular authentication and authorization implementation.

#### *The Auth Concept: Authentication and Authorization*

Authentication, which refers to ensuring that principles can be identified, and authorization, which involves access and resource control, ensures that a system is secure by design [14]. Jackson stated that a system that is secure by design focuses more on working fully despite security holes since it is impossible to close all of them [14].

The authentication and authorization implementation for the MoD system is based on the use of Amazon Cognito, a service as a part of Amazon Web Services (AWS). Amazon Cognito divides the organization of authentication and authorization into user pools and identity pools

[15]. User pools, primarily used for authentication, return access tokens and ID tokens which can later be exchanged for AWS credentials by the identity pool [15]. Fig. 3, below, illustrates the relationship between user pools and identity pools [15].



**Fig. 3: The Relationship Between the User Pool and Identity Pool in Amazon Cognito**

These AWS credentials give non-users guest access, based on implementation, and give users authenticated access. Users are authenticated through a trust policy, while a permissions policy grants access to AWS resources, among other purposes [15]. Some AWS resources that the MoD application accesses include S3 buckets, of which S3 is a service that stores encrypted data, API Gateway, which allows for REST API operations, and ExecuteAPI which allows one to invoke their API.

### **Role-Based Access Control Approach**

According to Sandhu and Samaranti, access control can be defined as “constrain[ing] what a user can do directly, as well as what programs executing on behalf of the users are allowed to do” [16]. Access control, thus, exists to keep malicious actors outside of the system. Multiple types of access control can be utilized to improve security, both inside and outside of

Amazon Web Services. This report will consider the role-based access control approach, which has been implemented to secure the MoD application.

Role-Based Access Control (RBAC) was achieved through Amazon Cognito, as delineated in the Software Engineering Context section. An iterative approach was used in developing the RBAC approach, starting with one IAM role, and then progressing to creating two roles with different permissions policies, and then multiple roles that apply to not just the system administrators and MDCS team.

The first IAM role that was created was the role of the system administrator, or sysadmin as it was called. The sysadmin specifically had four permissions to write data to S3 buckets, one permission to list the objects of an S3 bucket, and one permission to read data from an S3 bucket. For the starter policy, I allowed permissions for all S3 buckets, denoted by the asterisk.

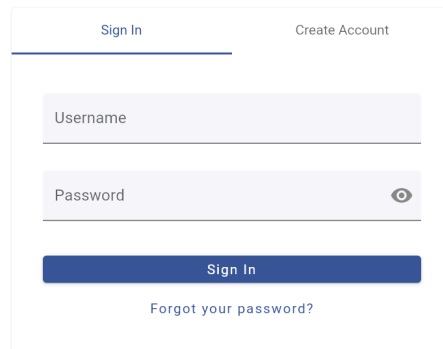
The second IAM role that was created was the role of the Moral Distress Consultancy Service, or modconsultservice, as it was called. The modconsultservice role did not have permission to read, write, or list data from S3 buckets. Both IAM roles had the permission to invoke the API, as part of the service ExecuteAPI.

#### Iteration One: Proof-of-Concept with Two Users

Having two roles was a proof-of-concept approach, to test that one role could have more permissions than the other. Proving that one role had more permissions than the other involved some development/debugging work, to find the ID and access tokens, and then using these in the AWS CloudShell, which is a command line interface for AWS, to use commands provided in the documentation to determine whether or not the users had permissions. For this test, I created the user ayy\_ayy and the user bee\_bee, along with two email accounts for the users so that they could become verified users and receive the necessary permissions for testing.

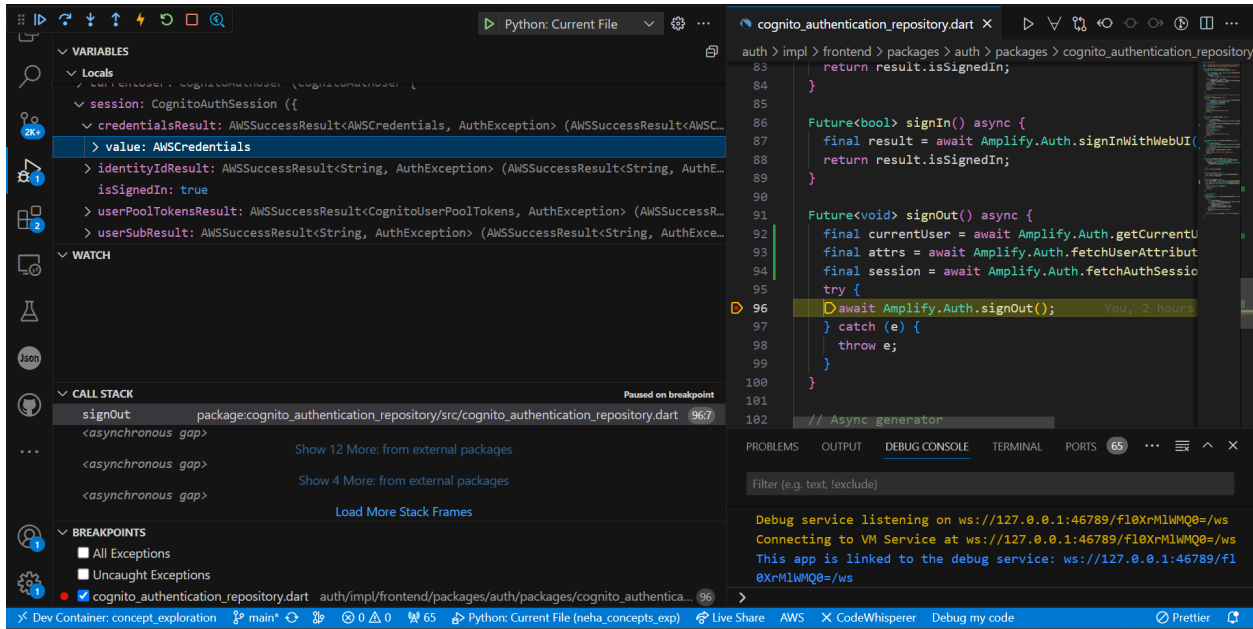
In testing the access control permissions, I solely used the auth concept. Below, in Fig. 4, is the graphical representation of the auth concept with full functionality.

---



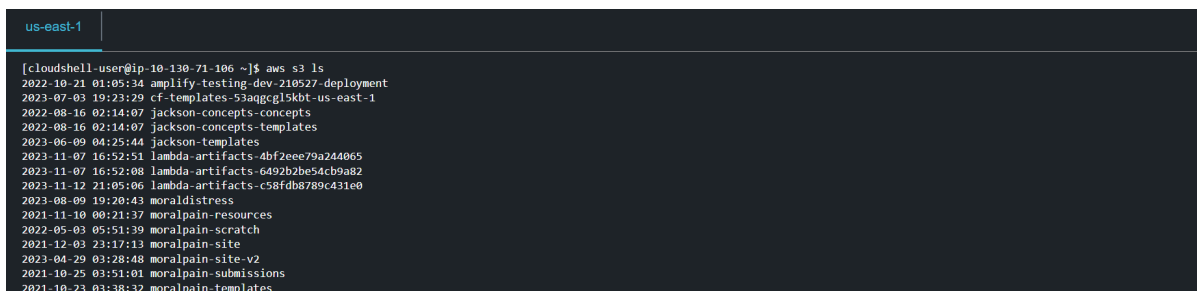
**Fig. 4: Graphical Representation of the Auth Concept**

Instead of creating my user through a specific command line interface, or the AWS SDK, I used the auth concept, with breakpoints inserted to find the `accessKey` and the `secretAccessKey` which I would later use in the AWS Command Line Interface (AWS CLI) to create a profile and test the permissions policies, specifically testing whether the user `ayy_ayy` had S3 bucket permissions, and, more importantly, whether `bee_bee` did not. Below, in Fig. 5, is a picture of the three lines I added to the Cognito authentication code, called `cognito_authentication_repository.dart`, to obtain the credentials required. Fig. 5 also includes the debugging stack, but not the actual values associated with the user `ayy_ayy`, for security purposes. However, as seen in the figure, an `AWSSuccessResult` was obtained, meaning that the identity pool was properly configured through an appropriate trust policy, and that credentials could be obtained.



**Fig. 5: Debugging Stack and Manipulated Code**

After obtaining the credentials for username *ayy\_ayy*, I went to the AWS CLI to determine if *ayy\_ayy* had permissions for listing all S3 buckets present, through the command **aws s3 ls**. Thankfully, the IAM role, with its permissions policy, appeared to work, as *ayy\_ayy* had this permission, shown below in Fig. 6. Not shown is the Access Key ID, Secret Access Key, and Session Token, which were all used to authenticate the user through the AWS CLI.



**Fig. 6: Results of aws s3 ls, for Testing Username ayy\_ayy**

Next, I had to test the user *bee\_bee*, to check if the IAM role *modconsultservice* had a permissions policy that did not allow access to S3 buckets. I tested this through the same method, and got the expected result, that the command **aws s3 ls** would not work, shown below in Fig. 7.

```
[cloudshell-user@ip-10-130-73-147 ~]$ aws s3 ls
An error occurred (AccessDenied) when calling the ListBuckets operation: Access Denied
[cloudshell-user@ip-10-130-73-147 ~]$
```

**Fig. 7: Results of aws s3 ls, for Testing Username bee\_bee**

With both results working, I could conclude that the proof-of-concept approach worked, and I would simply need to modify the permissions policies, create more users, and test in the same manner before evaluating my work in real-time.

### Iteration Two: Automation for Multiple Users

The next step would be to automate the entire process through CloudFormation templates and deploy the system so the nursing studies could continue. This would first require examining the existing CloudFormation templates, and then refactoring these templates to include the new roles that were created.

CloudFormation template management and variable parametrization were done through Sceptre [17]. Sceptre typically involves a combination of configuration files, each paired with a template [17]. In my configuration and template code, I automated the creation of two different stacks: **dev-nphair-auth-cognito-cognito-pools** and **dev-nphair-auth-cognito-identity-pool-roles**. As a stack is a collection of resources, these resources included user and identity pools and configurations for these as part of the cognito-pools stack, and roles and IdentityPoolRoleAttachments, which are used for mapping roles to users - the central aspect of my template code. Below, I included part of the core code driving the mapping of roles to users as part of the IdentityPoolRoleAttachment, in Fig. 8.

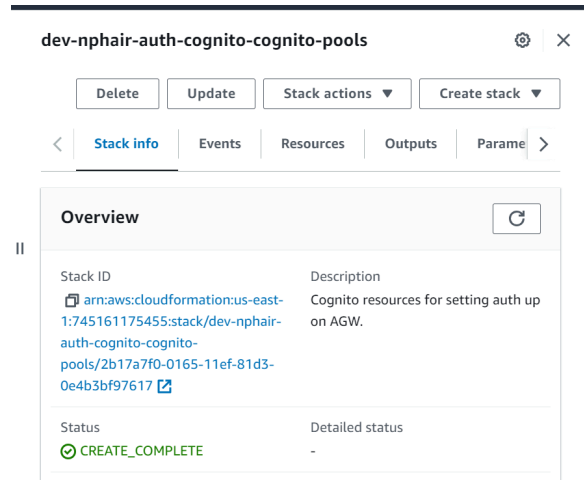
```

Resources:
  IdentityPoolRoleMapping:
    Type: AWS::Cognito::IdentityPoolRoleAttachment
    Properties:
      IdentityPoolId: !Ref IdentityPool
      Roles:
        authenticated: !GetAtt Nurses.Arn
        unauthenticated: !GetAtt UnauthRole.Arn
  RoleMappings:
    You, 48 seconds ago + update auth 4/23/2024
    "userpool1":
      IdentityProvider: !Sub "cognito-idp.${AWS::Region}.amazonaws.com/${UserPoolId}:${UserPoolClientId}"
      AmbiguousRoleResolution: AuthenticatedRole
      Type: Rules
      RulesConfiguration:
        Rules:
          - Claim: email
            MatchType: "Equals"
            RoleARN: !GetAtt ResearchTeam.Arn
            Value: "nphair@virginia.edu"
          - Claim: email
            MatchType: "Equals"
            RoleARN: !GetAtt ResearchTeam.Arn
            Value: "sullivan@virginia.edu"
          - Claim: email
            MatchType: "Equals"
            RoleARN: !GetAtt ResearchTeam.Arn
            Value: "nk7ry@virginia.edu"

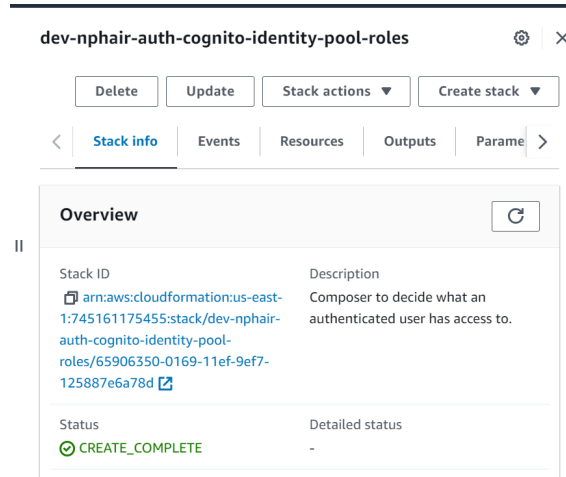
```

**Fig. 8: IdentityPoolRoleAttachment for RoleMapping**

This iteration proved to successfully deploy, before user testing which would be conducted in May 2024. I deployed this iteration of template code through Sceptre and Visual Studio Code's terminal. The stacks **dev-nphair-auth-cognito-cognito-pools** and **dev-nphair-auth-cognito-identity-pool-roles** were successfully created, as shown in Fig. 9 and Fig. 10.



**Fig. 9: Successful Creation of the dev-nphair-auth-cognito-cognito-pools Stack**



**Fig. 10: Successful Creation of the dev-nphair-auth-cognito-identity-pool-roles Stack**

With this iteration of the access control template code working, I could proceed to the evaluation of the system through a user study, which is forthcoming.

## Evaluation

This system will be evaluated through a multi-institutional study, occurring during May 2024. This is because the best way to assess the success of a role-based access control implementation is to test the system through users. All other testing methods before evaluation are conducted through the local deployments in the approach section.

## Related Work

### *Systems that Report Moral Distress*

There has been one software-based system to report moral distress [18]. This was developed as part of a study by the Computer Science Department at the University of Virginia as well as the University of Virginia School of Nursing. This application, which involved the causes of moral distress, an MDT reading, and resiliency resources, formed the basis of the system I worked on securing. However, this system needed to be developed with the methodology of concept-based software engineering.



### *Systems that Utilize Concept-Based Software Engineering*

Developed by Daniel Jackson and his team, there exist two software-based systems that use concept-based software engineering. One of these systems is Gitless, a method of version control that avoids common pitfalls associated with Git, and the other is Deja Vu.

Gitless, primarily developed by Santiago Perez de Rosso under Daniel Jackson's advice, was designed to be a new version of Git with concept-based design in mind, handling issues like staging changes and file path classifications [19]. Deja Vu, also developed by Perez de Rosso and Daniel Jackson, explored web application development with concept-based software engineering [20]. In Deja Vu, a large application is built from smaller web applications from a programming course at MIT [20]. It should be noted that neither Gitless nor Deja Vu were built in the healthcare domain, or with a specific client in mind where additional testing and evaluation could be conducted.

## References

- [1] Shay, R., Bhargav-Spantzel, A., & Bertino, E. (2007). Password policy simulation and analysis. *Proceedings of the 2007 ACM Workshop on Digital Identity Management*.  
  
<https://doi.org/10.1145/1314403.1314405>
- [2] *NIST Special Publication 800-63B*. pages.nist.gov. (n.d.).  
  
<https://pages.nist.gov/800-63-3/sp800-63b.html>
- [3] Villanueva, J. C. (n.d.). *An introduction to cipher suites*. JSCAPE.  
  
<https://www.jscape.com/blog/cipher-suites>
- [4] Diffie, W., & Hellman, M. E. (n.d.). *New Directions in Cryptography*.  
  
<https://www-ee.stanford.edu/~hellman/publications/24.pdf>
- [5] *Diffie-Hellman Key Exchange*. Information Security Stack Exchange. (n.d.).  
  
<https://security.stackexchange.com/questions/58658/diffie-hellman-key-exchange>
- [6] *Advanced encryption standard (AES)*. (n.d.).  
  
<https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197-upd1.pdf>
- [7] Figure 1. the basic AES-128 cryptographic architecture. (n.d.-b).

[https://www.researchgate.net/figure/The-basic-AES-128-cryptographic-architecture\\_fig1\\_230853805](https://www.researchgate.net/figure/The-basic-AES-128-cryptographic-architecture_fig1_230853805)

- [8] NIST FIPS 180-4. (n.d.-c). <https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.180-4.pdf>
- [9] Orebaugh, A., Ramirez, G., & Burke, J. (2007). *Wireshark & Ethereal Network Protocol Analyzer Toolkit*. Syngress Pub.
- [10] *Owasp Cloud-Native Application Security Top 10*. OWASP Cloud-Native Application Security Top 10 | OWASP Foundation. (n.d.).  
<https://owasp.org/www-project-cloud-native-application-security-top-10/>
- [11] Rushton, C. H., Caldwell, M., & Kurtz, M. (2016). CE: Moral Distress: A Catalyst in Building Moral Resilience. *The American journal of nursing*, *116*(7), 40–49.  
<https://doi.org/10.1097/01.NAJ.0000484933.40476.5b>
- [12] Wocial, L. D., & Weaver, M. T. (2013). Development and psychometric testing of a new tool for detecting moral distress: the Moral Distress Thermometer. *Journal of advanced nursing*, *69*(1), 167–174.  
<https://doi.org/10.1111/j.1365-2648.2012.06036.x>
- [13] Hamric, A. B., & Epstein, E. G. (2017). A Health System-wide Moral Distress Consultation Service: Development and Evaluation. *HEC forum : an interdisciplinary*

*journal on hospitals' ethical and legal issues*, 29(2), 127–143.

<https://doi.org/10.1007/s10730-016-9315-y>

- [14] Jackson, D. (2023). *Essence of software: Why concepts matter for great design*.

Princeton University Press.

- [15] Common Amazon Cognito scenarios - Amazon Cognito. (n.d.-b).

[https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-scenarios.htm](https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-scenarios.html)

l

- [16] Sandhu, R. S., & Samarati, P. (1994). Access control: Principle and practice. *IEEE*

*Communications Magazine*, 32(9), 40–48. <https://doi.org/10.1109/35.312842>

- [17] *About*. About - Sceptre 4.4.2 documentation. (n.d.). <https://docs.sceptre-project.org/latest/>

- [18] Amos, V., Phair, N., Sullivan, K., Wocial, L. D., & Epstein, B. (2023). A Novel

Web-Based and Mobile Application to Measure Real-Time Moral Distress:

An Initial Pilot and Feasibility Study. *Joint Commission journal on quality*

*and patient safety*, 49(9), 494–501.

<https://doi.org/10.1016/j.jcjq.2023.05.005>

- [19] A conceptual design analysis of Git - Santiago Perez De Rosso. (n.d.-a).

<https://spderosso.github.io/ms-thesis.pdf>

[20] Declarative assembly of web applications from predefined concepts. (n.d.-d).

<https://spderosso.github.io/phd-thesis.pdf>