RBAC for Healthcare-Infrastructure and data storage

Ramesh Narasimman\textsuperscript{a}, Izzat Alsmadi\textsuperscript{b}

\textsuperscript{a}Syracuse University
\textsuperscript{b}Texas A&M, San Antonio

\textbf{ARTICLE INFO}

\textbf{Keywords:}
RBAC
Dynamic RBAC
Temporal RBAC
Healthcare Infrastructure
Alloy

\textbf{ABSTRACT}

Role based Access control (RBAC) is the cornerstone of security for any modern organization. In this report, we defined a health-care access control structure based on RBAC. We used Alloy formal logic modeling tool to model and validate system functions. We modeled system static and dynamic or temporal behaviours. We focused on evaluating properties such as integrity, conformance and progress.

1. Introduction

Data protection and safeguard are of paramount importance in all industries. In Healthcare, it is also a regulatory and compliance requirement since the data collected includes patient’s medical, financial and personal information. Exposure of this data will severely impact the patients including financial losses and potentially impact the brand reputation of the organization. Depending on the nature of the data and the size of the exposure, breaches could lead to criminal charges. In this day and age of computing, all these records are electronic and hence require much more sophisticated means of protection from a wide variety of threats. As of 2017, nearly 9 in 10 (86\%) of office-based physicians had adopted any Electronic Health Records (EHRs), and nearly 4 in 5 (80\%) had adopted a certified EHR. Since 2008, office-based physician adoption of any EHRs has more than doubled, from 42\% to 86\%\cite{11}. This data cannot be locked down completely in order to protect it. Doctors and nurses will need access to this data to perform their jobs. Not having access to the right data at the right time could potentially result in loss of life. Hence securing this data should also focus on making this data accessible to people with the right credentials. This necessitates the need for an access management system. As stated by Haraty and Naous \cite{9}, an access control system should:

- Prevent unauthorized modifications
- Maintain internal and external consistency
- Prevent authorized but improper modifications

Role-Based Access Control is a simple but powerful concept that helps with this. Usually referred to as RBAC, Role-Based Access control implements access management through Roles. The major components of RBAC are Resources, Actions and Subject. Each role is assigned a set of permissions that encompasses resource(s) and related action(s) that can be performed on that resource. These roles are then assigned to a subject which will allow the subject to perform these actions. The role essentially defines a logical boundary by explicitly defining the permitted actions and hence prevents a subject from being able to perform any actions that are not assigned. These roles can also be based on several factors including time and state of the system. A subject can hold multiple roles and the combined permissions will be the permissions for the subject. Separation of Duties (SOD) is implemented to prevent conflicting permissions from being assigned to the same subject. SOD ensures subjects cannot inherit conflicting permissions when roles are combined. RBAC roles and SOD definitions are based on the roles and responsibilities defined within an organization and are a direct representation of an organization’s security policy.

Designing a security policy is the first step. For this paper, We used a tool called Security Policy Tool (SPT)\cite{12} that provides users a way to represent the elements in a tree-like structure. There is a also a command line option, XACML. Security Policy Tool allows us to compose, test, and validate access control policies and to ensure there are no access control leaks when the policies are deployed in a system. With these tests, the access control policies can be effectively analyzed by a policy author to ensure policy works as designed and identify unintended permission assignment.
The crucial part is having the model tested to ensure it covers all scenarios and produces expected outcome despite user count, multiple roles or other factors. To ensure this, the policy needs to be validated using formal methods. A validated policy is authentic and foolproof. Formal methods are a trusted and efficient way to validate these policies. Formal methods enable reasoning from logical or mathematical specifications of the behaviors of computing devices or processes; they offer rigorous proofs that all system behaviors meet some desirable property [4].

For this report, we used Alloy to formally specify the policies and validate them. Alloy is a great tool for defining the policy specs, generating a model based on specs and validate the model for correctness. It is a modeling tool based on first order logic combined with relational Algebra. It also includes an analyzer that performs bounded analysis of model specifications and provides a visual representation. Alloy can also evaluate the models and provide counter examples if applicable. Alloy, being based loosely on object oriented language concepts, makes it easy to write code. Its structure includes objects called Signatures. It also includes Facts that can be used to constrain the models. Additional operations can be performed using Predicates and Functions. Assertions can be used to validate the constraints placed on the model. Alloy uses SAT solver to analyse the model and establish validity or provide counter examples.

There are different variations of the Alloy tool. Two are most notable: Sterling and Electrum. They both use Alloy as the base and add additional modules to enhance the functionality. Sterling combines Alloy with web-based visualizations, providing both basic Alloy visualization capabilities and a robust platform for the development of domain specific visualizations of Alloy instances. The Alloy Instances tool provides a styling language and sharing platform that is useful for development of visualizations once a model has been completed, but it lacks in utility during the iterative modeling process. Sterling aims to bridge this gap and further build out the visualization and sharing platforms. Sterling is an Open Source tool based on Alloy as well as Spark, Gradle Shadow Plugin and additional JavaScript libraries.

Using these tools, we have created three basic policies to demonstrate how these tools can be effectively applied in a Healthcare infrastructure. The policies are simple and cover only a finite resources since the scope of this paper is purely academic. However they do incorporate all the necessary attributes and constructs to demonstrate the various aspects of a typical RBAC policy. Also, since RBAC policies closely resemble an organization’s security policies, there is no single template that could be created that fits all.

2. Literature Review

We have reviewed literature that are related to Role Based Access Control, Separation of Duties, Temporal-RBAC as well as various formal languages for model analysis.

Chuck Easttom [7] proposes a modeling language for cyber security. He discusses the need for a modeling language in cyber security similar to SysML. Cybersecurity can be defined as system of systems. That allows SysML to be used as foundation for the proposed language. With SysML as a base, he proposes a new language that adds additional
diagrams that are designed specifically for Cyber security. This new language provides greater flexibility to the security engineers in designing the models and can be used as a generalized security modeling language.

In this next paper, Sommestad, Ekstedt and Holm [15] discuss about Cyber Security Modeling Language (CySeMoL) and its effectiveness in identifying potential attacks against a system. This language also includes a probabilistic inference engine designed based on compilation of research results on a number of security domains and a wide range of attacks and countermeasures. CySeMoL aims to fill one of the biggest gaps in designing a security system - lack of understanding of the vulnerabilities in different components that interact with the system as well as within the system. CySeMoL is based on logical relations, experimental research and covers a variety of attacks, including software exploits, flooding attacks, abuse of obtained privileges, and social-engineering attacks. It is easy to use and does not require security expertise. A Turing test indicates that the reasonableness and correctness of CySeMoL assessments compare to those of a security professional.

Johnson, Lagerström and Ekstedt [13] discuss the need for a reusable, domain independent attack logic that can be utilized to codify attack steps and their dependencies. The article proposes Meta Attack Language (MAL) as a potential solution and details the specification and discusses features to help with the challenges in identifying the data to collect, issues in collecting the data and analyze and the article recommends the use of attack simulations. These simulations are based on attack graphs. Since many systems are similar, the attack graphs can be replaced by identifying common factors and import them into a modeling language. This can then be reused across multiple systems without the need for recreating the graphs.

Marrone, et-al discuss the need for vulnerability analysis in critical infrastructure systems such as railway systems. This paper reviews the existing system and provides a specification language CIP-VAM and a model-driven framework, METRIP, based on UML for performing vulnerability analysis and protection modeling functionality. This model takes one station and models the security in that station identifying the gaps and pros.

The next report reviewed [14] that discusses the need for formal verification of security models for Industrial control systems and how the improvements in IoT is exposing these once-isolated systems to cybersecurity threats. Specific areas of concern are securing the data and the control flow. The proposed model in this paper consists of the control network layer and a PC that connects to the control network for data transfer and management. The PC is the component running the SCADA software that connects externally to the cloud. There are also mobile components that access the data. The paper defines the Security requirements based on the IEC-62443-3-3 standard and defines behavioral and security properties. The paper also provides the verification as well as the code that is written in the proposed model that includes all the specified requirements.

Chong et. and all [4] review and document the security issues in major verticals including Hardware Architecture, Operating Systems, Distributed Systems and Networks as well as privacy and how formal methods can help resolve these. The report also stresses the importance of formal methods and the benefits it can provide in designing a system that meets the standards, makes interfaces easy, accurate and eliminate potential threats. The report also discusses various problems including ensuring whole-system security, defining correct abstractions, standardizing tool and techniques, ensuring formal methods are supported throughout the life cycle of the product and integrating formal methods with the common industry process in a scalable and compatible way. The workshop provides several recommendations to improve the efficiency and to improve the available talent pool that can help with the increased implementations of formal methods. Those include teaching formal methods early in the education curriculum sooner than they are now and include much more complex problems. Additional recommendations include encouraging community-based development of tools and techniques including conferences, journals as well as repositories. The workshop recommends that focus should be placed on both scientific work and applied engineering to improve cybersecurity through formal methods.

The focus of the next report, a thesis by Nicole Emerentiana van Deursen [6], is a new method to assess socio-technical information security risks. The core of this is method is information sharing across healthcare organizations as this could lead to collective learning and problem solving when faced with similar issues. The aim is to reach a collective state of information security that creates trust in the information we collect and share and ensures its validity. This method relies on a central database shared by various organizations. Incidents reported to this database are analyzed, validated by experts and combined with their insights in dealing with similar issues. The results are then transformed into a risk map that contains actionable information and prevention methodologies that can be used by participating healthcare organizations. The map is constantly updated as new incidents are reported. This method differs from its closest competitors due to its indefinite scope and wider system context.

The next report reviewed is also a thesis by Shan Wu [16]. This thesis reviews the methods for the development of
information security policies at organizations. The author stresses that the goal of information security policies should be to protect the business than focusing on specific data sets. The author then also details the functions each security policy much adhere to including stated punishment and also lists the stakeholders that should be involved in the policy development life cycle.

The next report [1] is part of the document detailing the Proceedings of the 10th international Joint Conference On Biomedical Engineering Systems and Technologies and was authored by Raja Manzar Abbas, Noel Carroll, Ita Richardson and Sarah Beecham. This report explores what determines the trustworthiness of healthcare software solutions. It also explores the potential risk of stakeholders trusting the software without realizing the consequences and proposes a model to assess the trustworthiness. While medical devices are subject to strict regulations and constant monitoring by authorities, Healthcare software is not. This leads to questions about the trustworthiness of the applications and the need to evaluate them against a standard. Though there is no proposed model, the report recommends that Security, Efficiency, Safety, Functionality, Reliability, Regulation, Validity and Accuracy be the key factors in the standard to determine the trustworthiness of this software.

The next report deals with Eager formal methods. Funded by the NSA, this report [8] looks at developing a highly abstract model of SELinux access control, map the flow and use model checking to validate the design and establish this as a case study for implementing Eager formal methods. This report also demonstrates how a combination of the formal models and an appropriate algorithm like SELinux provides the necessary framework to improve security in the operating systems and a model that can be replicated easily across any similar systems.

The next report by Haraty and Naous [10] focuses on validating the RBAC aspect of commercial applications using formal specifications. They model the RBAC components using Alloy and check the model consistency. This paper uses an example of a procurement system to demonstrate the implementation of RBAC and the process of modeling and validating using Alloy.

GEO-RBAC paper summarizes the concept of spatial awareness for RBAC and expands it further to include geographic features/boundaries [5]. This model not only checks whether the subject has the necessary attributes for role assignment but also looks to ensure the subject is physically present in the location to be able to get the role. This can be enforced using GPS and other location-tracking mechanism. This model also applies the Separation of Duties on to the GEO-RBAC model further strengthening it. The model is designed to be dynamic enough to automatically add or remove roles based on locations.

The problem of multiple organizations existing in the same setup being forced to share access control mechanisms is discussed in [2]. A framework is proposed for these policies to be merged while preserving all permissions and SOD requirements. The framework also proposes methodologies to resolve conflicts as well as a way to identify and remove potentially conflicting permissions while maintaining the original requirements.

TrustBAC describes the challenges in implementing RBAC in open systems as well as the shortcomings of combining RBAC with Credential based Access control [3]. The authors propose a solution that introduces the trust level in this model. Users who are authenticated using the credentials will be assigned to a trust level based on their behavioural history and these trust level will be mapped to Roles which in turn are mapped to permissions. By introducing the trust levels, the model is much more dynamic and secure.

3. Policy Definitions

In this paper, we created three policies that implement basic RBAC concepts. All three policies are based on these objects:

- Resource
- Role
- Action

The roles were assigned action(s) for individual resources that are then applied to users. User with the roles will be able to perform the allowed actions. They will not be able to perform anything that is not included in the permissions list. In addition, to ensure compliance and to ensure conflicting permissions are not inherited, separation of duties was also implemented.

Policy 1 deals with access for three servers - DC1, DC2 and FS1. Traditional roles including System Engineer are defined. Permissions for specific actions on the resources are defined. Policy 2 deals with access for Applications,
related servers and database servers. Multiple roles are defined, including the Application Administrator. Permissions are defined based on specific actions on resources. Policy 3 is specifically about security logs and log aggregator devices.

| Table 1 |
|---|
| Policy Resource Summary |

| Policy | Resources                        |
|-------|----------------------------------|
| Policy 1 | Domain Controller 1, Domain Controller 2, File Server |
| Policy 2 | App Servers, DB servers, Application |
| Policy 3 | Servers, Network Devices, Log Aggregator |

Detailed policy definitions are provided below:

### 3.1. Policy 1: Infrastructure Server Access for Different Roles

This policy defines access levels for different roles in the company. Access levels defined are:

- Admin Access to Servers
- RDP Access to Servers
- Share access to Servers

Permissions are defined for the following roles:

- System Engineer
- Network Engineer
- DBA
- IT Manager
- End Users
- External Users

Requirements are defined as follows:

- External users are treated as unauthorized and should not have any access to any resources.
- IT Manager should not have Admin or RDP access to servers. This is for separation of duties.
- Network Engineers and DBA should not have ADMIN or RDP Access to any of the servers.
- Only System Engineers should have Admin as well as RDP Access to the Servers.
- All users with the exception of external users, should have access to the shares.

### 3.2. Policy 2: Application and Database Servers

This policy defines access rules for the application and database servers as well as PHI and De-Identified data in the servers. Access Levels defined are:

- ADMIN Access to Application Servers
- ADMIN Access to Database Servers
- Database ADMIN access
- Access to Production Application
• Access to Production Data
• Access to Test Application
• Access to Test/De-Identified Data

Permissions are defined for the following roles:

• Developer
• Tester
• Server Administrator
• DBA
• Application ADMIN Users
• Application Users
• Other Users

Requirements are defined as follows:

• Production Servers
  – Developers and Testers will not have any access
  – Server Admins will have full Admin access to both Application and Database Servers
  – DBA will have Admin access only on the Database servers
  – DBA will also have Database Access
  – None of the users will have any access to the servers.

• Test Servers
  – Developers and Testers will not have any access
  – Server Admins will have full Admin access to both Application and Database Servers
  – DBA will have Admin access only on the Database servers
  – DBA will also have Database Access
  – None of the users will have any access to the servers.

• Production Application
  – Developers and Testers will not have any access
  – Server Admins will not have any access
  – DBA will not have any access
  – Application Admin users will only have access to the Admin Screens
  – Application End users will only have access to Non-Admin screens

• Test Application
  – Developers and Testers will have full access
  – Server Admins will not have any access
  – DBA will not have any access
  – Application Admin users will only have access to the Admin Screens
  – Application End users will only have access to Non-Admin screens
• Prod PHI/PII Application Data
  – Only DBA and Application users will have access.

• De-Identified Application Data
  – Testers, Developers, DBA, and Application users will have access.
  – Server Admins and Other Users will not have access

3.3. Policy 3: Log Access
This policy defines access levels for Server and Network Security logs. Access levels defined are:

• Admin Access
• Log Review
• Log Access

Permissions are defined for the following roles:

• Server Engineer
• Network Engineer
• Security Engineer

Requirements are defined as follows:

• Server Access is restricted to Server Engineers. Both Server Engineer and Security Engineer will have log review access on servers
• Network Device access is restricted to Network Engineers
• Both Network Engineer and Security Engineer will have log review access on network devices
• Log Aggregator access is restricted to Security Engineer.

Table 2
Policy Role Summary

| Policy   | Resources                                      |
|----------|------------------------------------------------|
| Policy 1 | System Admin, Network Engineer, DBA, IT Manager, End Users, External Users |
| Policy 2 | Developer, Tester, Server Admins, DBA, App ADMIN, End Users, Other Users   |
| Policy 3 | Server Engineer, Network Engineer, Security Engineer                           |

Table 3
Policy Actions Summary

| Policy   | Actions                                      |
|----------|----------------------------------------------|
| Policy 1 | Admin Access, Share Access, RDP Access       |
| Policy 2 | Admin Access, DB Access, App Access, AppAdmin Access, Data Access |
| Policy 3 | Admin Access, Log Review                     |
4. Security Policy Definition - Using Security Policy Tool

Based on the policy definitions, policy structure was created using the Security Policy Tool. Subject, Resources and Actions were defined and combined to create Access control policies using Deny-overrides Algorithm and Deny Biased Policy Enforcement Algorithm. Verification cases were defined to check for the validity of the policies. The tool can be downloaded from https://www.securitypolicytool.com.

4.1. Policy 1: Infrastructure Server Access for Different Roles

As defined, this policy dictates and controls access to servers for various roles in the company and specifies separation of duties as well. Definitions are as shown in Figure 2. Based on these definitions, access control policies were defined.

![Figure 2: Policy 1 - Definitions](image)

4.2. Policy 2: Application and Database Servers

This policy defines access rules for the application and database servers as well as Protected Health Information (PHI) and De-Identified data in the servers. Definitions are as shown in Figure 3. Based on these definitions, access control policies were defined.

![Figure 3: Policy 2 - Definitions](image)

4.3. Policy 3: Log Access

This policy defines access levels for Server and Network Security logs and centralized Log Aggregator. Definitions are as shown in Figure 4. Based on these definitions, access control policies were defined as shown.

![Figure 4: Policy 3 - Definitions](image)
5. **Formal Specification of the models - Alloy**

With the policies defined in the security policy tool, the next step is to formally specify the models using Alloy. Each policy is modeled separately. The objects defined in the policy are created as Sigs in Alloy and the ACLs are implemented through predicates. Facts are also used to enforce constraints on the model.

5.1. **Policy 1 - Alloy specification**

Based on the policy definition, below are the Sigs defined for Policy 1.

```plaintext
abstract sig Role() // defines the roles
abstract sig Resource() // defines the Resources
abstract sig Action() // defines the Actions or Access Levels
```

In addition to the Sigs, we added time and state for adding the temporal aspects to the model. The state will determine permissions based on the current status of the system.

```plaintext
sig Time() // for time definitions
sig RState()
```

With the definitions, several facts were added to model the policy. These facts enforce that every user must have one predefined role and the External Users role cannot be combined with any other roles. To demonstrate the role assignments, we created users that are derived from the user Sig and will inherit the role(s). The roles are listed in Table 4.

```plaintext
fact noOtherRolesforEU {
    no u:User | ExtUsers in u.urole and some (SysAdmin + DBA + NetworkEngineer + ITManager +
    ⇐ EndUsers) & u.urole
}
//Fact 2 - All Users must have one of predefined role
fact {
    all u:User | u.urole in Role
}
```

Assertions are added to validate these facts.

```plaintext
assert rolesNeededforall { 
    all u:User | u.urole in Role
}
assert ExtUserAccess{ 
    no u:User | ExtUsers in u.urole and some (SysAdmin + DBA + NetworkEngineer + ITManager +
    ⇐ EndUsers) & u.urole
}
```

With the basic blocks defined, we added predicates to build the permissions. A predicate was defined for each role to define the permission set for that role that details all the actions that the role can perform on the resources. Below is one of the predicates.

```plaintext
pred saPerms[rs:RState] {
    all usa:SysAdmin,fs:FS | usa.permissions = rs + fs + AdminAccess and usa.permissions = rs + fs +
    ShareAccess and usa.permissions = rs + fs + RDPAccess
    all usa:SysAdmin,dc1:DC1 | usa.permissions = rs + dc1 + AdminAccess + rs and usa.permissions =
    rs + dc1 + ShareAccess + rs and usa.permissions = rs + dc1 + RDPAccess + rs
    all usa:SysAdmin,dc2:DC2 | usa.permissions = rs + dc2 + AdminAccess and usa.permissions = rs +
    dc2 + ShareAccess and usa.permissions = rs + dc2 + RDPAccess
}
```

With the predicates defined for assigning permissions, we added a trace function that defines different permissions for different states of the system. These permissions are assigned through Emergency role assignment and are valid only for a specific duration providing the temporal aspect of the model. Dedicated predicates for role assignment enforce SOD depending on the state.

```plaintext
//System Admin Role has all permissions
pred trace {
    init [T/first]
    let rs' = InService | roleassign[rs'] implies (exuPerms[rs'] and saPerms[rs'] and nePerms[rs']
    ⇐ and itmPerms[rs'] and dbaPerms[rs'] and euPerms[rs'])
    let rs'' = Troubleshooting | eroleassign[rs''] implies (exuPerms[rs'''] and saPerms[rs'''] and
    ⇐ nePerms[rs''] and itmPerms[rs'''] and dbaPerms[rs'''] and euPerms[rs'''])
}
```
With all the necessary components in place, the compilation was successful. Figure 5 shows the compilation results. Figure 6 shows the metamodel generated by this. The model generated is shown in Figure 7.

**Table 4**  
**Policy 1 - Users**

| User   | Default Role          |
|--------|-----------------------|
| userEx | External User         |
| userSA | System Administrator  |
| userNE | Network Engineer      |
| userDBA| Database Administrator|
| userITM| IT Manager            |
| userEU | End-User              |

5.2. Policy 2 - Alloy specification

Based on the policy definition, below are the Sigs defined for Policy 2. This policy also has an additional object Environment. The resources are classified in to Production and Test based on the environment.

```plaintext
abstract sig Role() //defines the roles
abstract sig Resource() //defines the Resources
abstract sig Action() //defines the Actions or Access Levels
abstract sig Env() //defines the environment
```

Code to add time and state for adding the temporal aspects to the model.

```plaintext
sig Time() //for time definitions
sig RState()
```

With the definitions, several facts were added to model the policy. These facts enforce that every user must have one predefined role and the External Users role cannot be combined with any other roles. To demonstrate the role
assignments, we created Users that are derived from the user Sig and will inherit the role(s). The roles are listed in Table 4.

```
fact {
  no u: User | OtherUsers in u. urole and some (Developer + Tester + DBA + ServerAdmin + AppAdminUsers + AppUsers) & u. urole
}

fact {
  all u: User | u. urole in Role
}
```

Assertions are added to validate these facts.

```
assert rolesNeededforAll {
  all u: User | u. urole in Role
}

assert OtherUserAccess {
  no u: User | OtherUsers in u. urole and some (Developer + Tester + DBA + ServerAdmin + AppAdminUsers + AppUsers) & u. urole
}
```

With the basic blocks defined, we added predicates to build the permissions. A predicate was defined for each role to define the permission set for that role that details all the actions that the role can perform on the resources. Eventually, we modified the code to include Environment as part of the permissions and rewrote all the permission predicates to include that. Below is one of the predicates.

```
pred devPerms [rs: RState] {
  // No Access to Prod App, Prod and Test App Server, Prod and Test DB Server & Data
  no ud: Developer, aps: AppSrvr, p: prod, t: test | ud. permissions = p + rs + aps + AdminAccess and
  ud. permissions = t + rs + aps + AdminAccess and
  ud. permissions = t + rs + aps + Access

  // Only Allowed access to Deid Data
  all Ud: Developer, td: DeidData | ud. permissions = rs + td + DataAccess

  // Only Allowed access to Test App
  all Ud: Developer, ta: App, p: prod | ud. permissions = p + rs + pa + AppAdminAccess and ud.
  ud. permissions = p + rs + pa + AppAccess

  // Only Allowed access to Prod App, Prod and Test App Server, Prod and Test DB Server & Data
}
```

With the predicates defined for assigning permissions, we added a trace function that defines different permissions for different states of the system. These permissions are assigned through Emergency role assignment and are valid only for a specific duration providing the temporal aspect of the model. Dedicated predicates for role assignment enforce SOD depending on the state.

```
pred trace {
  init [T/first]
```

Figure 7: Policy 1 - Model Output
Figure 8: Policy 2 - Alloy Output

![Policy 2 - Alloy Output]

Figure 9: Policy 2 - Metamodel Output

```plaintext
let rs' = InService | roleassign[rs'] implies (ouPerms[rs'] and devPerms[rs'] and testPerms[rs']
leftrightarrow saPerms[rs'] and dbaPerms[rs'] and appadminPerms[rs'] and appuserPerms[rs'])
let rs' = Troubleshooting | eroleassign[rs'] implies (ouPerms[rs'] and devPerms[rs'] and
testPerms[rs'] and saPerms[rs'] and dbaPerms[rs'] and appadminPerms[rs'] and appuserPerms[
leftrightarrow rs'])
```

With all the necessary components in place, the compilation was successful. Figure 8 shows the compilation results. Figure 9 shows the metamodel generated by this. The model generated is shown in Figure 10

Table 5
Policy 2 - Users

| User   | Default Role       |
|--------|--------------------|
| userDev| Developer          |
| userTest| Tester            |
| userSA | System Administrator |
| userDBA| Database Administrator |
| userAAU| Application Administrator |
| userAU | Application End-User |
| useOU  | Other Users       |

5.3. Policy 3 - Alloy specification

Based on the policy definition, below are the Sigs defined for Policy 3.

- `abstract sig Role() //defines the roles`
- `abstract sig Resource() //defines the Resources`
- `abstract sig Action() //defines the Actions or Access Levels`

Code to add time and state for adding the temporal aspects to the model.

- `sig Time() //for time definitions`
- `sig RState()`

The facts added to the model are listed below. The roles are listed in Table 6.
Figure 10: Policy 2 - Model Output

// Fact 1 - All Users must have one of predefined role
fact {
  all u:User | u.urole in Role
}

// Fact 2 - Users in this policy must have unique roles as this policy concerns log access
fact {
  no u:User | ServerEngineer in u.urole and some (NetworkEngineer + SecurityEngineer) & u.urole
}

Assertions are added to validate these facts.

// Assertion 1 - Each user must have one role
assert rolesNeededforAll {
  all u:User | u.urole in Role
}

// Assertion 3 - Users must have unique roles
assert uniqueNWrole {
  no u:User | NetworkEngineer in u.urole and some (ServerEngineer + SecurityEngineer) & u.urole
}

Predicates were added to assign permissions to the roles

// Permissions for Network Engineer Role
pred nwPerms [rs:RState] {
  // No Access to Servers and Logs
  no une:NetworkEngineer, dc:DC | une.permissions = rs + dc + AdminAccess and une.permissions = rs + dc + LogReview
  no une:NetworkEngineer, ap:AppSrvr | une.permissions = rs + ap + AdminAccess and une.permissions = rs + ap + LogReview
  // Full Access to Network Devices and Logs
  all une:NetworkEngineer, fw:FireWall | une.permissions = rs + fw + AdminAccess and une.permissions = rs + fw + LogReview
  all une:NetworkEngineer, sw:Switches | une.permissions = rs + sw + AdminAccess and une.permissions = rs + sw + LogReview
  // No Access to Log Aggregator
  no une:NetworkEngineer, lg:LogAggregator | une.permissions = rs + lg + AdminAccess and une.permissions = rs + lg + LogReview
}

With the predicates defined for assigning permissions, we added a trace function that defines different permissions for different states of the system. These permissions are assigned through Emergency role assignment and are valid only for a specific duration providing the temporal aspect of the model.

pred trace {
  init [T/first]
  let rs' = InService | roleassign[rs'] implies (sePerms[rs'] and saPerms[rs'] and nwPerms[rs'])
  let rs' = Troubleshooting | eroleassign[rs'] implies (sePerms[rs'] and saPerms[rs'] and nwPerms[rs'])
}

With all the necessary components in place, the compilation was successful. Figure 11 shows the compilation results. Figure 12 shows the metamodel generated by this. The model generated is shown in Figure 13.
Conclusion

In this paper, a formal access control model is proposed for a healthcare system. We selected based users, roles and functions that can be application in most health sector domain as well as other relevant domains. We adopted state of the art RBAC access control model due to its ability to capture fine-grained access control requirement. We used Alloy Analyzer as a formal logic modeling tool to create and verify our model. We focused on some properties related to access control and showed using Alloy how such models can be implemented and also checked. Alloy includes also the ability to check for assertions to find possible ways to violate the model based on the model entities and constraints.

References

[1] Abbas, R.M., Carroll, N., Richardson, I., Beecham, S., . The need for trustworthiness models in healthcare software solutions. Proceedings of the 10th international Joint Conference On Biomedical Engineering Systems and Technologies URL: https://www.academia.edu/download/51967997/VandenBroek_etal-BioSTEC2017-Health_Informatics.pdf#page=470.
[2] B. Shafiq, J. B. D. Joshi, E. B., Ghafoor, A., . Secure interoperation in a multidomain environment employing rbac policies. IEEE Transactions on Knowledge and Data Engineering, vol. 17, no. 11, pp. 1557-1577 URL: https://ieeexplore.ieee.org/abstract/document/1512040.

[3] Chakraborty, S., Ray, I., . Trustbac: Integrating trust relationships into the rbac model for access control in open systems. . URL: https://doi.org/10.1109/25.1133067.

[4] Chong, S., Guttman, J., Datta, A., Myers, A., Pierce, B., Schaumont, P., Sherwood, T., Zeldovich, N., . Reports on the nsf workshop on formal methods for security. . URL: https://arxiv.org/pdf/10.1145/1133058.1133067.

[5] Damiani, M.L., Bertino, E., Catania, B., Perlasci, P., . Geo-rbac: A spatially aware rbac. . URL: https://dl.acm.org/doi/10.1145/1210263.1210265.

[6] van Deursen, N.E., . Hi-risk: a socio-technical method for the identification and monitoring of healthcare information security risks in the information society. . URL: https://core.ac.uk/reader/4029638.

[7] Eastom, C., . Secmfl: A proposed modeling language for cybersecurity. . URL: https://ieeexplore.ieee.org/document/160800678.

[8] Guttman, J.D., Herzog, A.L., Ramsdell, J.D., . Information flow in operating systems: Eager formal methods. . URL: http://citeseer.ist.psu.edu/viewdoc/download?doi=10.1.1.473.8427&rep=rep1&type=pdf.

[9] Haraty, R.A., Naous, M., a. Role-based access control modeling and validation. . URL: https://www.researchgate.net/publication/271550189_Role-Based_Access_Control_modeling_and_validation.

[10] Haraty, R.A., Naous, M., b. Role-based access control modeling and validation. . URL: https://www.researchgate.net/publication/271550189_Role-Based_Access_Control_modeling_and_validation.

[11] of the National Coordinator for Health Information Technology, O., . Office-based physician electronic health record adoption, health it quick-stat #50. . URL: https://www.healthit.gov/quickstat/physicianchr-adoption-trends.php.

[12] InfoBeyondTechnology, . Security policy tool. . URL: https://securitypolicytool.com/

[13] Johnson, P., Lagerström, R., Ekstedt, M., . A meta language for threat modeling and attack simulations. . URL: https://dl.acm.org/doi/10.1145/3230833.3232799.

[14] Kulik, T., Tran-Jørgensen, P.W.V., Boudjadar, J., . Vulnerability modeling and analysis for critical infrastructure protection applications. . URL: https://www.researchgate.net/profile/Jabil_Boudjadar/publication/334633010_Compliance_verification_of_a_cyber_security_standard_for_Cloud-connected_SCADA/links/5db58db44585155e2707c44c/Compliance-verification-of-a-cyber-security-standard-for-Cloud-connected-SCADA.pdf.

[15] Sommestad, T., Ekstedt, M., Holm, H., . The cyber security modeling language: A tool for assessing the vulnerability of enterprise system architectures. . URL: https://www.diva-portal.org/smash/get/diva2:561243/FULLTEXT02.

[16] Wu, S., . Review of the methods for the development of information security policies at organizations. . URL: https://jyx.jyu.fi/bitstream/handle/123456789/51700/URN%3aNBN%3aati%3ajyu-201610274457.pdf?sequence=1&isAllowed=y.

Izzat Alsmadi is an Associate Professor in the department of computing and cyber security at Texas A&M, San Antonio. He has his master and PhD in Software Engineering from North Dakota State University in 2006 and 2008. He has more than 100 conference and journal publication. His research interests include: Cyber intelligence, Cyber security, Software security, software engineering, software testing, social networks and software defined networking. He is lead author, editor in several books including: Springer, The NICE Cyber Security Framework Cyber Security Intelligence and Analytics, 2019, Practical Information Security: A Competency-Based Education Course, 2018, Information Fusion for Cyber-Security Analytics (Studies in Computational Intelligence), 2016. The author is also a member of The National Initiative for Cybersecurity Education (NICE) group, which meets frequently to discuss enhancements on cyber security education at the national level.

I am currently pursuing my Masters in Cyber Security at Syracuse University. I received my bachelors degree in computer science from Annamalai University, Chidambaram, India. I started my professional career as System Administrator and held various roles including Developer and teacher. I hold two MCP certifications, MCSD certification as well as a WebSphere certification from IBM. In addition to my Masters, I am also working on advancing my MCSD certification.