RBAC Attack Exposure Auditor. Tracking User Risk Exposure per Role-Based Access Control Permissions

Adelaide Damrau

RBAC Attack Exposure Auditor.

Tracking User Risk Exposure per Role-Based Access Control Permissions

by Adelaide L. Damrau

An Undergraduate Thesis Submitted in Partial Fulfillment
of the Requirements for the
University Honors-in-Discipline Program
Honors College
and the
Honors-in Computing Program
College of Business and Technology
East Tennessee State University

Adelaide Damrau 4/6/2023
Adelaide Damrau Date

[illegible] 04/14/2023
Dr. Mohammad Khan, Mentor Date

Matt Harrison 4/7/23
Mr. Matthew Harrison, Reader Date

Declaration by student

I, Adelaide Damrau, hereby declare that the work presented herein is original work done by me and has not been published or submitted elsewhere for the requirement of a degree program. Any literature date or work done by others and cited within this thesis has been given due acknowledgement and is listed in the reference section.

Acknowledgements

Thank you to my family and friends, Ben, Janine, Hannah, and Tina, that encouraged and supported me through this process, whether through code reviews, rants, presentation practice, etc. I appreciate all of you. Shoutout to Mr. Matthew Harrison for making time to review my work and putting up with me over the last few years. Thank you to Dr. Ghaith Husari for guiding the direction of my thesis.

Certificate

Certified that the thesis titled “RBAC Attack Exposure Auditor. Tracking User Risk Exposure per Role-Based Access Control Permissions” submitted by Ms. Adelaide Damrau towards partial fulfilment for the Bachelor’s Degree in Computing (Honors-In-Discipline Scholars Program) is based on the investigation carried out under our guidance. The thesis part therefore has not been submitted for the academic award of any other university or institution.

Mr. Mohammad Khan

(Supervisor/Advisor)

Lecturer of Department of Computing

Mr. Matthew Harrison

(External Examiner)

Lecturer of Department of Computing

Abstract

Access control models and implementation guidelines for determining, provisioning, and de-provisioning user permissions are challenging due to the differing approaches, unique for each organization, the lack of information provided by case studies concerning the organization's security policies, and no standard means of implementation procedures or best practices. Although there are multiple access control models, one stands out, role-based access control (RBAC). RBAC simplifies maintenance by enabling administrators to group users with similar permissions. This approach to managing user permissions supports the principle of least privilege and separation of duties, which are needed to ensure an organization maintains acceptable user access security requirements.

However, if not properly maintained, RBAC produces the problem of role explosion. What happens when security administrations cannot maintain the increasing number of roles and their assigned permissions provisioned to the organization users?

This paper attempts to solve this problem by implementing a scalable RBAC system and assigning each permission a risk value score determined by the severity of risk it would expose the organization to if someone had unauthorized access to that permission. Using RBAC's role and permission design, each user will be assigned a risk value score determined by the summation of their roles' risk based on permission values. This method allows security administrators to view the users and roles with the highest level of risk, therefore prioritizing the highest risk users and roles when maintaining user roles and permissions.

Keywords: Role-Based Access Control, Principle of Least Privilege, Separation of Duties, Role, Permission, Risk, Access Control, Role Explosion

Table of Contents

I. Introduction

A. Overview and Motivation

B. The Present Study

II. Literature Review

A. Access Control Systems

B. Separation of Duties in Access Control

C. Role-Based Access Control Systems

D. Extended RBAC

E. RBAC Implementations

F. Role Explosion

G. Risk Scoring

III. Methods

A. Data Creation

B. Mapping Objects

C. Database Creation

D. Third-Party RBAC System

E. System Alterations

F. RBAC Audit

G. Risk Score Calculations

IV. Results

V. Discussion

VI. Future Work

VII. Conclusion

REFERENCES

Appendix A. Definitions

Appendix D. Permissions

I. INTRODUCTION

A. Overview and Motivation

Role-Based Access Control (RBAC) simplifies maintenance by enabling administrators to group users with similar permissions. In practice, a system's roles can become troublesome to manage due to changes to a system's resources and users' job responsibilities. RBAC can enable organizations to limit what users can access, as a way of protecting sensitive IT assets. It can support the principle of least privilege (PoLP), which minimizes user access to systems resources, limiting them to those permissions needed to do their jobs. It can also enforce separation of duties (SoD) by restricting the permissions that a given entity can acquire, ensuring that multiple entities share responsibility for conducting a potentially sensitive process. Authors Coyne and Weil in [1] state, "RBAC permits simplified auditing of the resources available to a given user as well as the users who have access to a given resource".

However, if not frequently maintained, an increase in unmanaged roles leads to role explosion, leading to an excessive number of permissions for users. An increase in unnecessary roles and permissions for users removes PoLP and SoD protections, placing the organization at greater risk if a user was compromised.

B. The Present Study

One solution to role explosion is to assign each permission a risk score determined by its risk to the organization if compromised. With each permission scored, each role can have a risk score determined by the summation of its included permissions' risk scores. Additionally, because each user can have more than one role assigned to their job function, the user's risk score is the summation of their corresponding roles' risk scores.

Once all permissions, roles, and users have their assigned risk score, the organization can easily view which roles or users have the highest risk for the organization and reevaluate if all the user’s roles are necessary, limiting role explosion by the highest risk items first.

This study explored one aspect of role explosion solution, how assigning each permission an individual risk score could support an organization facing role explosion by exposing which roles and users risk scores are above a designated safe score. A database was filled with fake user data to demonstrate the functionality of the RBAC system. The database contained data for users, permissions, roles, and resources. The database also maintained connections between the permissions to roles and roles to users. The database filled the RBAC system objects and mapped the corresponding permissions to roles and roles to users.

Once the security administrator fills the RBAC system with values from the database, the system allows the administrator a selection of commands, allowing viewing or editing the inserted data. Additionally, the RBAC system’s customized functions allows the administrator to view the risk score of an individual user, role, or permission, allowing in-depth analysis of the RBAC structure.

II. LITERATURE REVIEW

Access control is a form of data security, dictating who is allowed to access and use an organization’s information and resources. Access control enforces separation of duties and the principle of least privilege by limiting conflicts of interest and ensuring each user only has the least number of permissions needed to do their job. One access control model, known as role-based access control, is popular because of its simplicity organizing similar permissions into roles and provisioning roles to users. Many forms of access control have stemmed from RBAC, such as extended role-based access control, that provides context-based role filtering to dynamically suppress or provision access to roles to users given point in an organization’s operation. Several implementations of RBAC in varying businesses and environments demonstrate the strengthens and weaknesses of the access control model. One such weakness of RBAC is the risk of role explosion, which occurs when the RBAC system lacks frequent maintenance and retains an increased number of provisioned roles to users, breaking down separation of duties constrains and the principle of least privilege and opening the organization to risk. Risk scoring allows the organization to score and rank objects depending on the likelihood they will occur and the impact it would have on the organization.

A. Access Control Systems

In [2], Hobson discusses the various forms of access control usually implemented by organizations, Mandatory Access Control (MAC), Discretionary Access Control (DAC), Role-Based Access Control (RBAC), and Attribute-Based Access Control (ABAC). These varieties of access control systems integrate with the business applications and security programs.

MAC is an approach to access control that requires administrators to manually define all possible combinations of access permissions and rules that any user might require at any point in an organization’s operations. Generally found in government or military facilities with varying classifications and clearance levels, MAC is typically considered the most secure access control although requires more maintenance and manual work. DAC is an approach to access control that requires users to manage their own permissions on the resources that they curate. For example, a user making a social media post can edit who can view the content, private, public, or specific people. DAC is flexible and customizable per need, but because each user is the owner and responsible for their content, it is less centralized. ABAC systems is an adaptation of access control that provisions or revokes access based on user attributes, with respect to the requested object, the type of access requested, and the development environment hosting the task. ABAC systems update policies and rules as the conditions change.

One strength of these access control systems are their reporting features. Access control systems’ real-time reporting provides important analytics and data tracking, critical for centralized security auditing, system evaluation, performance measuring. Real-time monitoring provides the organization with instant notifications or alerts when suspicious or unprecedented activity occurs. Many offered access control systems, for example, if a user begins logging in to an application an unnecessary or significantly increased number of times, the access control systems regulating that user or application would alert the directed security administrator. Also, the access control system can provide logs of events, placing a timestamp on activities for future analysis.

In [3], Colombo and Ferrari discuss access control technologies for big data management systems. The increasing use of data-driven predictive strategies in decision making has created a demand for large volumes of data—typically schema-less, heterogeneous, and unstructured content—from sources such as IoT devices, e-mail, social media, other web postings, and wearable devices. This combination of volume and variety, together with the velocity (i.e., speed) with which data is generated, has complicated efforts to ensure this data’s security and privacy.

The predictive value of the data being gathered makes it a high-value target for exfiltration. For example, considering the online fitness industry and the increased use of wearable health-tracking devices, big data systems' abilities could potentially profile users and infer their lifestyle based on the databases containing user data on weight, exercise, and heart rate. However, Colombo and Ferrari state the majority of these big data systems lack access security and data protection [3].

One reason for this lack of protection is the lack of a standard model for control. Big data management systems are relatively new and continue to be under development. Current security standards for traditional databases cannot readily be adapted to these systems. The volume of data in big data systems requires the scope of access control security to broaden while still restricting access to what users need to do their jobs. The presence of unstructured data makes it harder to define constraints in an access control model. The speed at which data is collected requires the implementation of efficient mechanisms for integrating new data into an access control framework. Additionally, big data platforms do not usually have a consistent manipulation language, making it more challenging to create a cross-platform model.

Colombo and Ferrari [3] identify three requirements that address the security needs of big data management systems. The first requirement, fine-grained access control (FGAC), entails assigning access control rules to data at the finest granularity levels possible by the organization. By assigning each data item its own access rules based on its content, FGAC becomes an effective method of protection for sensitive data and user privacy concerns. Due to FGAC’s fine-grained nature, organizations must create their own access rules and tailor them to their systems' content. A second requirement, context management, entails support for context-based, data-type-specific constraints that can implement an organization's access control policies. For example, these constraints could restrict data access to specified time intervals, locations, or job shifts. The last requirement, efficiency, states that access control rules should not hinder the speed with which data can be collected and queried.

B. Separation of Duties in Access Control

In [4], Ferroni determines separation of duties (SoD) can be conceptualized in terms of duties, actors, risk, and conflicts. Duties are a process's units of work, also called tasks. Actors are the individuals or groups responsible for duties. SoD constraints concerning actors can be separated into three sections based on conflicts of interest stemming from the individual, function, or company. SoD by individual is the traditional SoD implementation in which different individuals perform different duties to uphold SoD. For example, a manager signs off on paychecks from the accountant before they can be distributed. The SoD by function or organizational unit relies on different departments to perform different duties to accomplish SoD. For example, the sales department makes an offer, and the accounting department confirms the process. The SoD by company level requires the duties to be performed by different organizations. For example, audits may require a third-party entity to eliminate any conflict of interest within the company.

Risk analysis after an initial SoD implementation encourages organizations to identify and mitigate potential vulnerabilities. When a risk scenario is deemed too dangerous to leave alone, SoD administrators should adapt SoD control policies to safeguard against that security fault. SoD systems with a lack of administration will likely cause inconsistencies among user permissions and have the potential to allow an actor to perform conflicting duties, leading to fraudulence.

According to Ferroni [4], SoD, ideally, should ensure that each actor has different tasks to enforce a balance of power, although exceptions could be integrated on a case-by-case basis. Exceptions should be considered if removing the conflict is not viable or too expensive and after a thorough risk analysis has deemed it acceptable. Conflicts occur more often when the initial design of the access control system is inadequate. Conflicts detected by the SoD system or defined by administrators can be mitigated by adding or changing processes, such as dividing functions into a greater number of tasks. By further separating functions into different tasks, it encourages a division of labor among users, although it increases in complexity.

C. Role-Based Access Control Systems

In [5], Bertino states, "Role-based access control (RBAC) is a technology that has been proposed as an alternative approach to traditional access control mechanisms both to simplify the task of access control administration and to directly support function-based access control". RBAC is an access control based on roles given to users or groups given their job requirements and position within an organization. Roles are a collection of permissions to authorize a user. Because RBAC can be used to implement the principle of least privilege, its use reduces potential damage from insider threats. Also, RBAC's structure allows users to easily shift among job roles as their position within the organization changes. Thus, it is "more scaleable than user-based security specifications and greatly reduces the cost and administrative overhead associated with fine-grained security administration at the level of in dividual users, objects, or permissions" [6].

One potential difficulty with RBAC is confirming that each user has only the required permissions needed at all times. In [7], Jin states “the proliferation of RBAC extensions might be unified by adding appropriate attributes within a uniform framework, solving many of these shortcomings of core RBAC”. When applying roles within an organization, RBAC identity and access administrators must balance manageability and specificity.

D. Extended RBAC

In [8], Liu et al. describe extended RBAC, an enhancement of RBAC that provides context-based role filtering to dynamically suppress access to roles that users will not need to access at a given point in an organization’s operation. Liu propose extended RBAC as an alternative to two other models that also provide finer-grained control over an organization’s permissions. One, mandatory access control (MAC), requires administrators to manually define all possible combinations of access permissions and rules that any user might require at any point in an organization’s operations. Enforcement of these rules is then delegated to that organization’s operating system. While the MAC model can provide organizations with a high degree of security, the model is difficult to maintain, nearly impossible to scale in a substantial organization, and not user-friendly. The other competing model, discretionary access control (DAC), requires users to manage their own permissions on the resources that they curate. This model is implemented by most online social networks; networks commonly enable users to choose who can access the various data items in their accounts. Similarly, in a business environment that implemented DAC, a database’s owner would maintain other users’ permissions for accessing that database’s content. While the DAC model is flexible, it requires every custodian of an organization’s IT resources to be trained in data security. It also has no provisions for enforcing standardization.

Extended RBAC supports the principle of least privilege by dynamically restricting a user's roles based on that user's identity, job responsibilities, permissions, existing roles, and available resources. Liu et al. [8] recommend using extended RBAC in situations that would, in normal RBAC, require an excessive number of roles or frequent changes in users' permissions management.

Liu et al. [8] tested the effectiveness of extended RBAC in a simulated environment that allocated time-and place-specific permissions and roles to users dynamically, based on working contexts. Their simulation updated these contexts at the start of each run; in practice, this would most likely be the start of every shift. The simulation altered the users' roles and permissions according to these changes. It assumed that all potential combinations of access permissions that users would need to carry out tasks would fit into a simple list of general contexts.

The authors' [8] simulation showed that extended RBAC could be used to reduce the number of roles that a comparable RBAC-based system of permissions would need to enforce policy. The authors also noted that they assumed that their simulation could account for all specialized contexts; in reality, determining contexts is complicated and most likely requires a lot of oversight, especially when first implementing extended RBAC.

E. Case Studies of RBAC Implementations

In [9], Carvalho and Bandiera-Paiva discuss strategies for implementing RBAC-based access in health systems environments. Their starting point for this discussion is a three-step model for granting access to a system-based resource. Initially, a user who requests access to a resource initially submits proof of identity. The most used means of authentication are usernames and passwords, although digital certificates such as smart cards and biometrics such as fingerprints are growing in popularity because of their increased security. The system then authenticates the user’s identity and authorizes their request to manipulate the resource. The last phase, authorization, uses a predefined list of permissions, which associates objects with users who can access them and their permitted means of access.

According to Carvalho and Bandiera-Paiva [9], additional support for SoD is essential for RBAC’s use in health systems. In health system environments, policies and professional standards impose a complex set of restrictions on access to massive amounts of electronic health records and other sensitive items. A strict infrastructure that provides for SoD helps maintain control of and ensure compliance with health industry needs and regulations.

Separation of duties (SoD) can be applied in two ways, static separation of duties (SSoD) or dynamic separation of duties (DSoD). SSoD states that no role should ever have any combination of permissions that could possibly give rise to a conflict of interest. Strembeck and Neumann [10] define SSoD as a constraint on the system “which specify that two mutual exclusive roles must never be assigned to the same subject simultaneously”. SSoD, for example, might disallow roles that entitle users to submit and authorize purchase orders, since this could enable those users to authorize their own purchase requests. DSoD, however, allows exceptions to be attached to roles that permit users to exercise potentially conflicting roles in allowable ways. Further defined by Strembeck and Neumann [10] as “two mutual exclusive roles must never be activated simultaneously within the same user session, or time constraints which restrict role activation to a specific time interval”. DSoD, for example, might allow a user to submit and authorize purchase orders, so long as a superior or another employee also authorized the purchase.

Case studies of RBAC implementations are rare due to concerns about what those studies reveal about an organization’s security policies. In [11], Schaad et al. present one such study, involving a large European bank, Dresdner Bank. Dresdner bank developed and implemented a custom implementation of RBAC called Funktionale Berechtigung (FUB). FUB can administer roles and permissions after viewing daily updates of the system applications and the users' contexts, such as job responsibilities and access to their current resources. Figure 1 shows the basic configuration of the FUB architecture. At the time of the bank's RBAC evaluation, FUB maintained an average of 2,000 roles for roughly 51,000 employees.

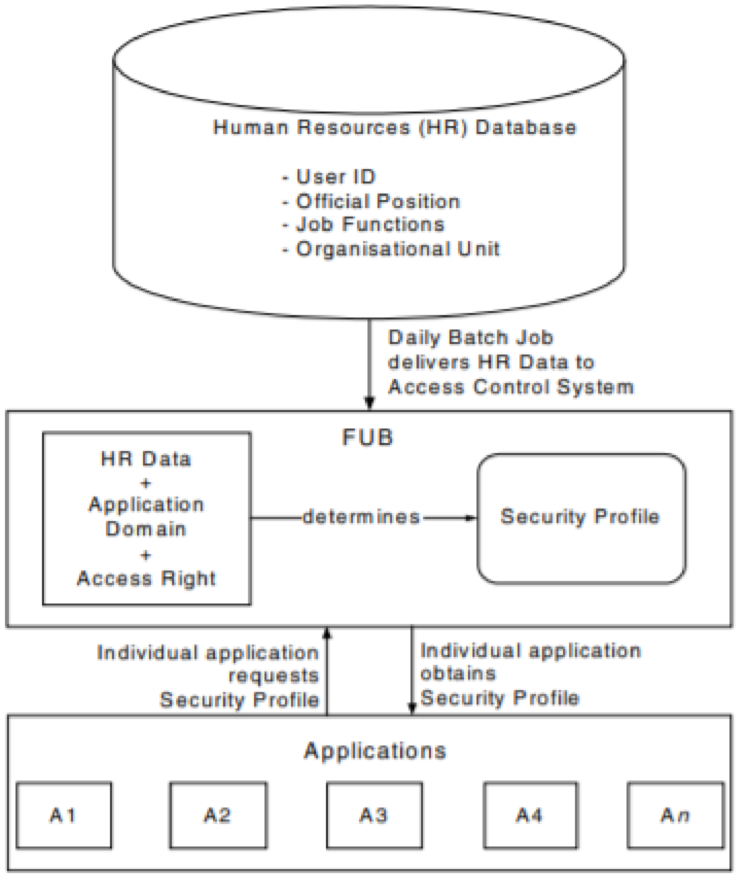

Human Resources (HR) Database
- User ID
- Official Position
- Job Functions
- Organisational Unit
Daily Batch Job delivers HR Data to Access Control System
FUB
HR Data + Application Domain + Access Right
determines
Security Profile
Individual application requests Security Profile
Individual application obtains Security Profile
Applications
A1
A2
A3
A4
An

Figure 1: The basic structure of the FUB

Theoretically, all of a bank employees' roles are the product of their job function and corporate position. The case study exposed a need for Desdner Bank to manage roles for temporary employees such as contractors and consults. The case study also demonstrated the need to assign temporary permissions for employees who were doing other employees' jobs and the need for grouping employees' positions and roles.

F. *Role Explosion*

The anonymous authors in [12] discuss common issues that businesses face when implementing RBAC. One issue is the extent to which RBAC will be used to manage access to business resources. While using RBAC to manage all access can work for smaller businesses, a practice known as role absolutism, this will likely create a role explosion in larger corporations. Role explosions occur when a model defines more roles than RBAC administrators can maintain. In response to role absolutism, professionals recommend limiting the use of RBAC to situations that do not require excessive roles changes. For example, where multiple users who continually change roles need to access sensitive data, an organization might consider an alternative to RBAC.

A second issue involves the need to define roles based on applications or user job descriptions. While RBAC vendors can try to define roles based on analyses of user data, ultimately determining permissions for roles requires manual evaluation. To ease the manual analysis, an organization should establish a fundamental role model specific to their needs that clearly defines user access requirements. Creating such a model can involve much design and revision, due to the need to tailor role configurations for business needs and processes. Potential risks in RBAC development include defining an inappropriate or inflexible model and defining a model that fails to anticipate likely changes; all of these errors can waste funds and resources [12]. Data accuracy is critical to safeguard against any potentially disastrous consequences of role definition: e.g., overlooking a need to protect sensitive information from widespread access.

Best practices in RBAC management include avoiding the creation of redundant roles and conducting periodic reviews and updates to user roles to limit system neglect. Administrators need to be cautious with existing system applications to ensure they can handle integrating with RBAC. One frequent challenge of RBAC implementation occurs when users receive access by requesting exceptions for extra permissions to their current role instead of moving to another role that fits their new access requirements [12].

Most important to ensuring successful RBAC implementation is establishing a reliable foundation of policies, processes, and a management team. Implementing policies for standardizing data throughout the organization extends system lifespan and usability.

G. Risk Scoring

In [13], Davis specifies the formula for risk value scores as

probability of occurrence x impact on organization

whereas the formula for security risk,

risk = (threat x vulnerability x probability of occurrence x impact)/controls in place

Risk scoring allows the organization to assign a risk value score in a standard way, either by level of risk such as low, medium, high or the existent cost of the risk, whether in budget, time, or the organization's reputation. Following NIST guidelines, one can derive the probability of occurrence for a vulnerability using three factors, threat source motivation and capability, nature of the vulnerability, and existence of effectiveness of current controls. The threat source is defined by NIST [14] as "the intent and method targeted at the intentional exploitation of a vulnerability or a situation and method that may accidentally trigger a vulnerability". This factor evaluates how liable the threat source is, what its abilities may be, what can the threat source gain from it, etc. Vulnerability is defined by NIST [14] as a "weakness in an information system,

system security procedures, internal controls, or implementation that could be exploited or triggered by a threat source”. The nature of the vulnerability considers what the cause or attributes of the vulnerability may be, where its weaknesses are, when is it active, is it already being exploited elsewhere, etc. The third factor, existence and effectiveness of current controls reflects on the security mechanisms already in place that could detect the threat or prevent it. The security administrator must consider past exploits and similar events in outside organizations.

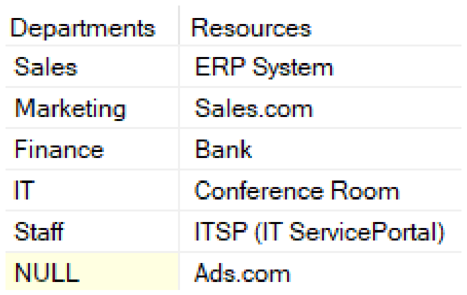
III. METHODS

A. Data Creation

Before system initialization, fake sample data was created to simulate that of a generic business, including objects:

- Department
- Resources
- Permissions
- Roles
- Users

Figure 2 shows the departments, resources, and roles objects created for the RBAC system meant to mimic real-life applications and business departments.



Departments	Resources
Sales	ERP System
Marketing	Sales.com
Finance	Bank
IT	Conference Room
Staff	ITSP (IT ServicePortal)
NULL	Ads.com

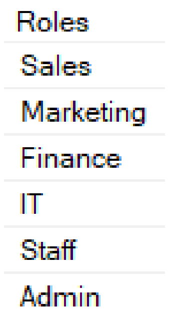
Figure 2: RBAC department, resources, and roles objects

Permissions were created based off potential actions a user could take on a resource or in an assigned department. Examples of permissions used in the data consist of:

- Edit Marketing Information
- View Billing Information
- Edit Balance Sheet
- View Account Information
- Edit Sales Report

Each permission had an assigned risk score value tied to it. The risk the permission posed to the organization if that permission were compromised by a malicious source determined the risk score value. For example, view type permissions do not allow changing data, so they are scored lower than the permissions that allow editing. Each permission was analyzed and evaluated, determining the attack exposure it would create if susceptible. Appendix B displays a complete list of permissions used in the RBAC system.

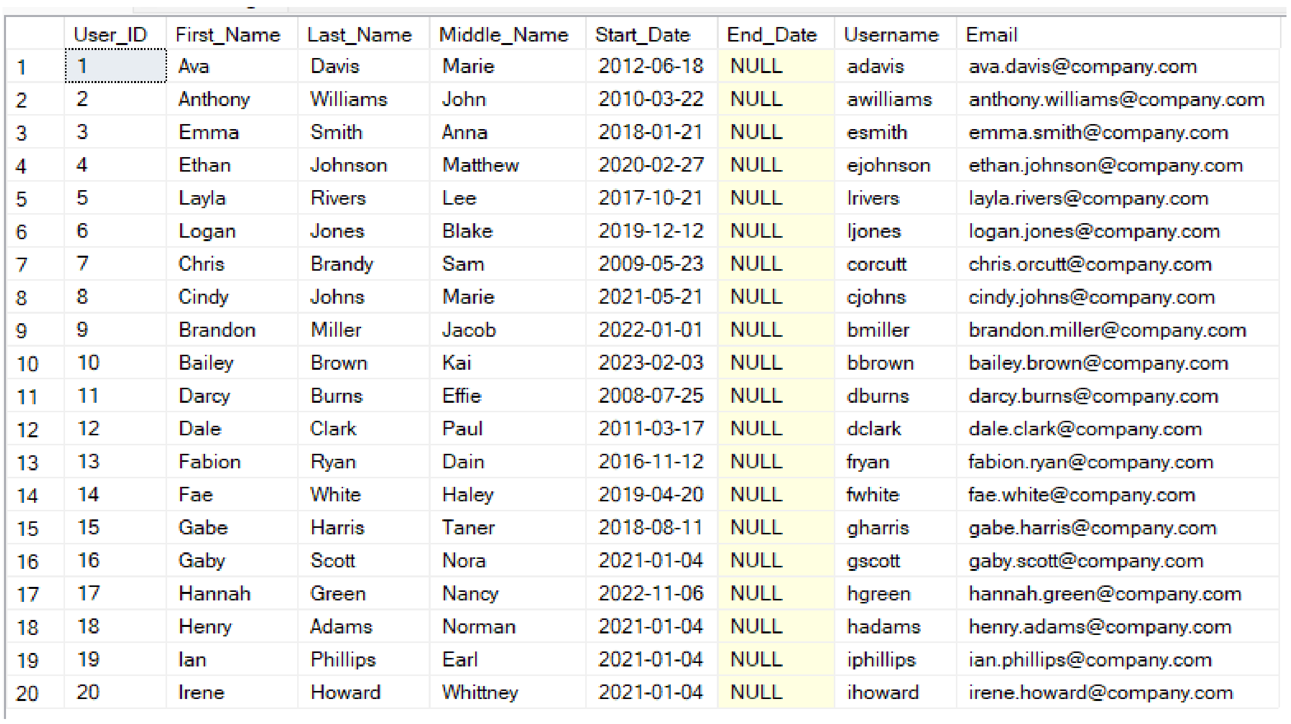
Roles were created from grouping similar permissions together for a user's job role in a department. For example, permissions allowing editing and viewing marketing reports directly corresponds to the marketing department where a user works, creating the marketing role. For simplicity and demonstration, roles created in the data correspond to their department. The only exception being the Admin role because it pertains to permissions in various departments. Once finished creating all roles, the corresponding permissions were mapped to them. Figure 3 displays the roles created.



Roles
Sales
Marketing
Finance
IT
Staff
Admin

Figure 3: Roles created based on grouping similar permissions

For this study, only 20 sample users were created, acting as a sample population of the larger generic business. The number of roles varied among users to provide a diverse range of user-maintained permissions and roles usually found in an organization facing role explosion. For this RBAC system demonstration it was unnecessary to create personal data for each user. Seen in Figure 4 is the complete user data.



	User_ID	First_Name	Last_Name	Middle_Name	Start_Date	End_Date	Username	Email
1	1	Ava	Davis	Marie	2012-06-18	NULL	adavis	ava.davis@company.com
2	2	Anthony	Williams	John	2010-03-22	NULL	awilliams	anthony.williams@company.com
3	3	Emma	Smith	Anna	2018-01-21	NULL	esmith	emma.smith@company.com
4	4	Ethan	Johnson	Matthew	2020-02-27	NULL	ejohnson	ethan.johnson@company.com
5	5	Layla	Rivers	Lee	2017-10-21	NULL	lrivers	layla.rivers@company.com
6	6	Logan	Jones	Blake	2019-12-12	NULL	ljones	logan.jones@company.com
7	7	Chris	Brandy	Sam	2009-05-23	NULL	corcutt	chris.orcutt@company.com
8	8	Cindy	Johns	Marie	2021-05-21	NULL	cjohns	cindy.johns@company.com
9	9	Brandon	Miller	Jacob	2022-01-01	NULL	bmiller	brandon.miller@company.com
10	10	Bailey	Brown	Kai	2023-02-03	NULL	bbrown	bailey.brown@company.com
11	11	Darcy	Burns	Effie	2008-07-25	NULL	dburns	darcy.burns@company.com
12	12	Dale	Clark	Paul	2011-03-17	NULL	dclark	dale.clark@company.com
13	13	Fabion	Ryan	Dain	2016-11-12	NULL	fryan	fabion.ryan@company.com
14	14	Fae	White	Haley	2019-04-20	NULL	fwhite	fae.white@company.com
15	15	Gabe	Harris	Taner	2018-08-11	NULL	gharris	gabe.harris@company.com
16	16	Gaby	Scott	Nora	2021-01-04	NULL	gscott	gaby.scott@company.com
17	17	Hannah	Green	Nancy	2022-11-06	NULL	hgreen	hannah.green@company.com
18	18	Henry	Adams	Norman	2021-01-04	NULL	hadams	henry.adams@company.com
19	19	Ian	Phillips	Earl	2021-01-04	NULL	iphillips	ian.phillips@company.com
20	20	Irene	Howard	Whittney	2021-01-04	NULL	ihoward	irene.howard@company.com

Figure 4: Complete RBAC system user sample data

B. *Mapping Objects*

The last requirement of data creation was mapping the permissions to roles and roles to users. Mapping permissions to roles and roles to users can be many-to-many as seen in Figure 5 by Ferraiolo [15]. The many-to-many relationships indicate that multiple permissions can be assigned to one role and in return multiple roles can be assigned to a permission. The same goes for roles and users, a user can have multiple roles and many roles can be assigned to a user. A collection of permissions assigned to a role is defined as a set. Authors Sahani, et al. in [16] describe, "suppose R1, R2, R3 and R4 are the roles defined and{P1; P2; P3}, {P4; P5}, {P3; P6}, {P7; P8; P9} are the permission sets assigned to them respectively. If roles R1 and R3 are assigned to the user, then permissions acquired by him would be {P1; P2; P3; P6}". Therefore, all the permissions provisioned to a user is the union of all the user's assigned roles.

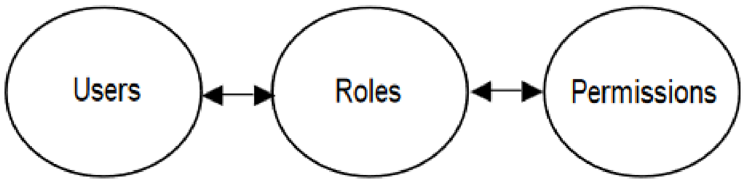


Figure 5: Generalized RBAC Model

Departments were assigned to each user, thus assigning them a role. To better mimic role explosion, some users were given multiple roles for job functions they might still have or job functions they moved from and their old roles were never removed. Figure 6 shows the distribution of roles among users.

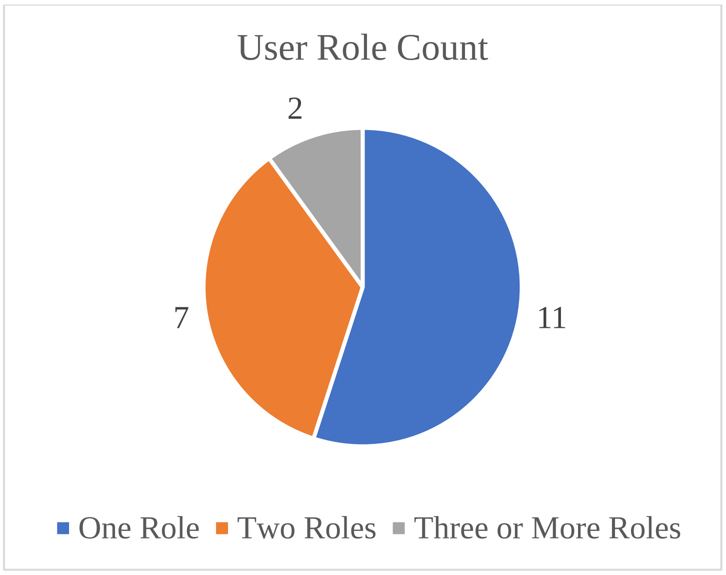


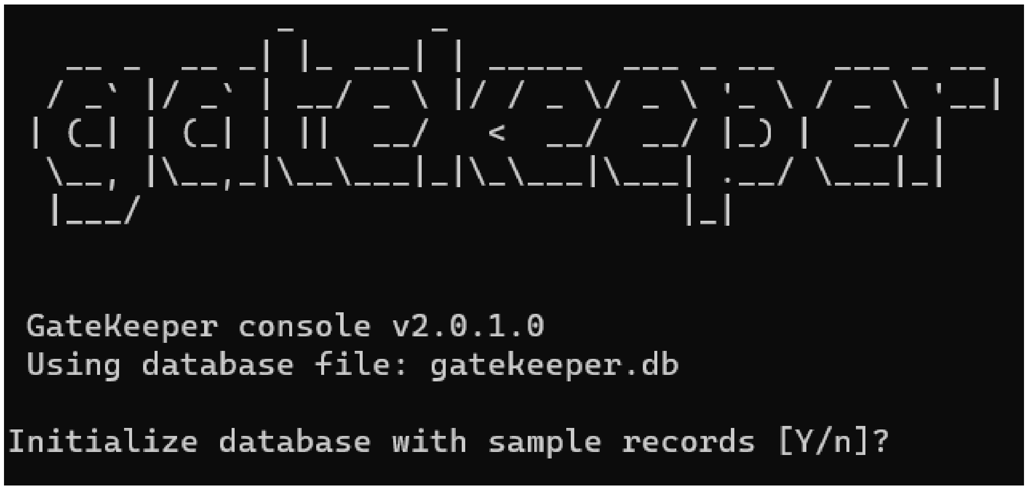
Figure 6: Distribution of user role count in sample data

C. *Database Creation*

After finishing data creation, a new database was created in Microsoft SQL Server Management Studio. The database was implemented and the RBAC objects and mappings were inserted into the database. Tables to maintain the permissions, roles, resources, and users were created and filled, followed by two additional tables to preserve the permission to role mappings and roles to user mappings.

D. Third-Party RBAC System

The initial RBAC system used to maintain roles, resources, permissions, and users was obtained from an open-source platform on GitHub called GateKeeper: a Roles-Based Access Control Library in C# created by Christner [17]. Christner states, “with GateKeeper, you can define users, roles, and permissions, then authorize access attempts to resources (by resource name and operation)”. Gatekeeper utilizes a third-party database system, WatsonORM, a “lightweight and easy to use object-relational mapper (ORM) in C# for .NET Core built on top of DatabaseWrapper” [18]. Also, Gatekeeper runs in the console, there is no graphical user interface (GUI), as shown in Figure 7.



```
 GateKeeper console v2.0.1.0
 Using database file: gatekeeper.db

Initialize database with sample records [Y/n]?
```

Figure 7: GateKeeper Console command line

E. System Alterations

Although the GateKeeper system had functionality to prepopulate RBAC sample values, they were vague and inconsistent with the sample data created to demonstrate RBAC auditing. Because the sample data for this study was maintained in Microsoft SQL Server Management

Studio, adjustments had to be made to the GateKeeper console code, adding in a database connection string and queries and removing the few prepopulated objects. GateKeeper also provided the functionality for creating the objects and mapping them, however, its process for mapping permissions to roles and roles to users was one to many, meaning that only one permission could be assigned to a role at a time and only one role could be assigned to a user. To overcome this obstacle, GateKeeper was altered, adding functions to logically link roles to permissions and users to roles through the database foreign keys. For example, the user Hannah Green has three roles: sales, marketing, and finance. The user-to-role table maintained in the SQL database has an entry for user and their different roles, so there would be three entries for Hannah Green in the role-to-user table, one for sales, marketing, and finance.

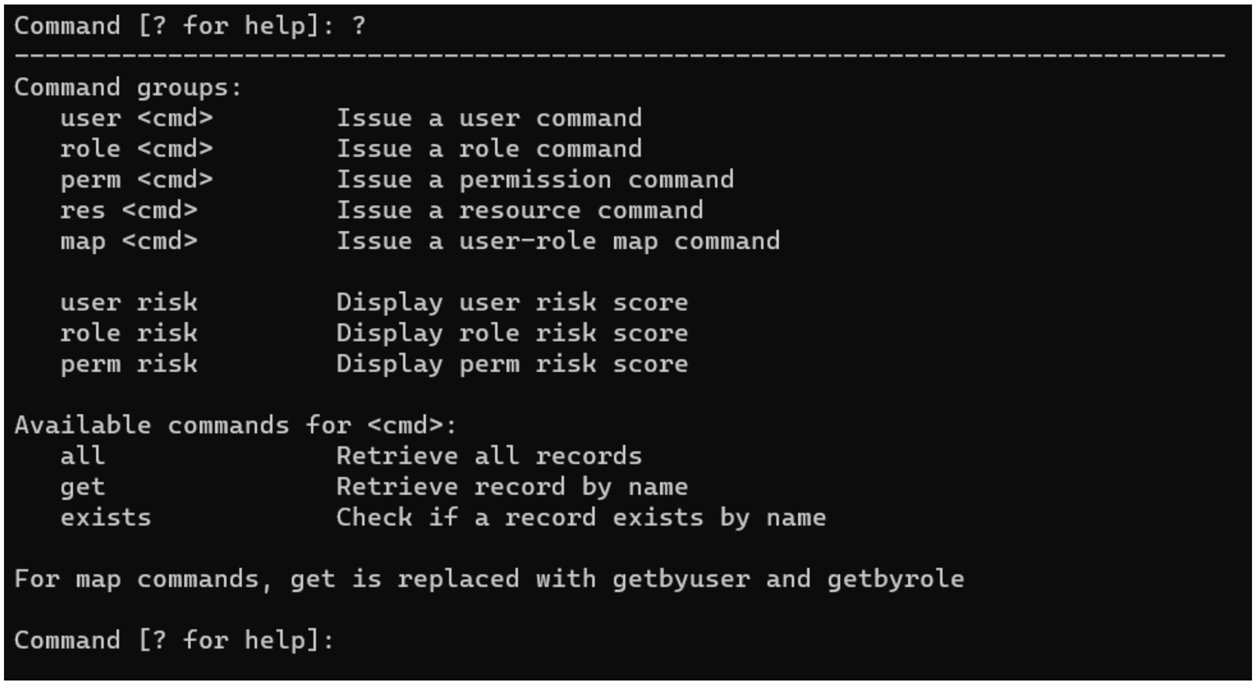
In addition, because Gatekeeper implements WatsonORM, the database structure could not be edited, such as adding and removing tables, rows, and columns. Therefore, to include the permission risk score value, the permission table columns were altered to accommodate the risk scores. The permission table contains columns:

- ID
- GUID
- Name
- RoleGUID
- ResourceGUID
- Operation
- Allow

The event authorization feature in GateKeeper was not used, so Allow was set to null in this study. The sample data created for permissions contained an ID, Name (which was the permission operation), and risk score value. In the RBAC system, to accommodate for the sample data structure, GateKeeper Name stored the risk score value and Gatekeeper Operation stored the sample data permission title because it was already named as its operation.

F. *RBAC Audit*

To build in the risk score auditing, three menu functions were created to allow a security administrator to view the individual risk of a user, role, or permission. These menu options were known as "user risk", "role risk", and "perm risk" as seen in Figure 8.



```
Command [? for help]: ?
--------------------------------------------------------------------------------
Command groups:
   user <cmd>         Issue a user command
   role <cmd>         Issue a role command
   perm <cmd>         Issue a permission command
   res <cmd>          Issue a resource command
   map <cmd>          Issue a user-role map command

   user risk          Display user risk score
   role risk          Display role risk score
   perm risk          Display perm risk score

Available commands for <cmd>:
   all                Retrieve all records
   get                Retrieve record by name
   exists             Check if a record exists by name

For map commands, get is replaced with getbyuser and getbyrole

Command [? for help]:
```

Figure 8: Updated Menu Display for user, role, and permission risk score values

Returning a permission risk score is the simplest of the three functions because it requires no computation; instead return the variable that contains the value determined by the user entered permission ID. However, because a role may have multiple permissions, the system must iterate through all the role's permissions contained in the role object and add their corresponding risk score values together to get the role's risk score. The user risk function gathers a list of all the user roles, creates a list of all those role's permissions, removes any duplicate permissions, then adds the permission risk score together for the total user risk score.

G. *Risk Score Calculations*

Below are the expressions to represent the set of user roles, the set of role permissions, and those used to calculate the risk score for a role and user.

$$User\ Roles = S = \{r_1, r_2, r_3, \ldots, r_n\}$$

$$Role\ Permissions = R = \{p_1, p_2, p_3, \ldots, p_n\}$$

$$Role\ Risk\ Score = \sum_{i=1}^{n} p_i$$

$$User\ Risk\ Score = \bigcup_{i=1}^{n} S_i$$

For example, user Bob has three roles assigned to him, $\{r_1, r_2, r_3\}$. This set of roles belonging to Bob is known as $bobS = \{r_1, r_2, r_3\}$. The permissions for each role:

$$r_1 = \{p_1, p_2, p_3, p_4\} = \{200, 100, 300, 200\}$$

$$r_2 = \{p_5, p_6\} = \{100, 400\}$$

$$r_3 = \{p_2, p_4, p_8\} = \{100, 200, 100\}$$

Following the expression stated for user risk,

$$User\ Risk = \bigcup_{i=1}^{n} S_i$$

$$Bob\ User\ Risk = \bigcup_{i=1}^{n} bobS_i$$

$$\bigcup_{i=1}^{n} bobS_i = \{p_1, p_2, p_3, p_4, p_5, p_6, p_8\}$$

$$\bigcup_{i=1}^{n} bobS_i = 200 + 100 + 300 + 200 + 100 + 400 + 100$$

$$\bigcup_{i=1}^{n} bobS_i = 1{,}400$$

Although simple summations get the roles' risk score, a summation will not accurately get a user's risk score because it would add duplicate permissions among the roles, such as r_1 and r_3 having both permissions p_2 and p_4. Instead, to accurately get the user's risk score, the set of all the roles the user maintains, known as *S*, let *S* be the union of the user permissions to prevent duplicate permissions from affecting the score.

IV. RESULTS

Using the user risk function of the altered GateKeeper system, each user's risk score value could easily be accessible. By calling the function for each user and logging the results, the security administrator has an overview of which users are more of a risk to the organization and should be reevaluated for role explosion. Figure 9 reveals the overview of user risk score values for the 20 sample users created in the RBAC system.

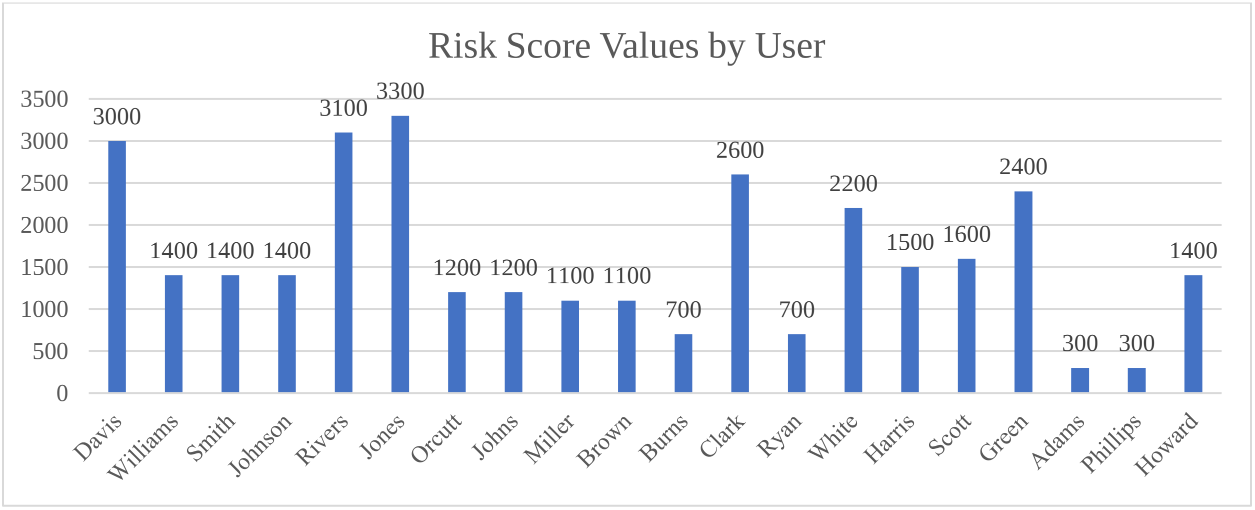


Figure 9: Overview of Risk Score Values by User

From the overview of user risk score values provided in Figure 9, the security administrator can easily determine that users Jones, Rivers, and Davis have the highest risk score values at 3300, 3100, and 3000, respectively. With this information, the security administrator can evaluate the roles maintained by users Jones, Rivers, and Davis, removing unnecessary roles and enforcing the PoLP.

Similar to the user risk function, using the role risk function of the altered GateKeeper system allows the security administrator to view the risk score value for each role. Calling the role risk function and logging all the risk score values provides the security administrator with an overview of which roles are more of a risk to the organization and should be given to users sparingly. Figure 10 shows the overview of role risk score values for the six sample roles created in the RBAC system.

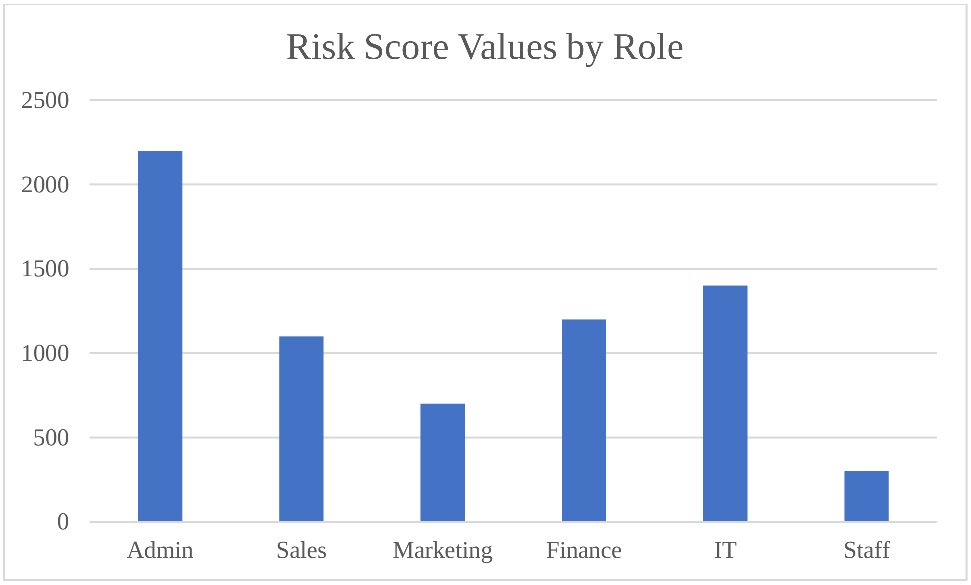


Figure 10: Overview of Risk Score Values by Role

From the overview of role risk score values provided in Figure 10, the security administrator can determine which roles maintain a higher level of permissions and therefore risk to the organization. With this information, the security administrator can evaluate the permissions in the role and remove any unnecessary permissions, further enforcing the PoLP. Also, the security administrator can use this information to help determine which roles should be provisioned to users more sparingly so as to not open the organization up to unnecessary risk.

V. DISCUSSION

The RBAC system requires frequent maintenance to preserve, and if not carefully updated, unmanaged roles provisioned to users leads to role explosion and generates excessive risk to the organization if that user or role were to become jeopardized. The overview of the user and role risk score values show the discrepancy of access among users and roles, allowing the security administrator to easily assess and edit the RBAC system permission provisioning. The results of comparing the user risk score values conclude that Jones, Rivers, and Davis have an excessive risk to the organization and be reevaluated for roles or permissions they may no longer use, reducing role explosion and enforcing PoLP. The results of comparing the roles risk score values determine the Admin role, followed by the IT role maintain the highest amount of risk to the organization and therefore the security administrator should limit the number of users that regularly use that role and cautiously provision it to more users.

This RBAC auditing system is significant because it attempts to ease maintenance on security administrators implementing RBAC while limiting the danger of role explosion in a growing organization. Also, this RBAC system provides an outline of valuable data concerning the organization’s provisioned permissions.

The results cannot detail, however, if a user is appropriately participating in Separation of Duties (SoD). This would require mapping of conflicting permissions and either stop the system from provisioning one of the conflicting permissions or provide an exception process, similar to one found in Dynamic Separation of Duties (DSoD). Also, the system relies entirely on the security administrator to perform the audit actions. If the security administrator takes eight months to check the user and role risk score values, the organization may have been at risk for any amount of time during those eight months.

VI. FUTURE WORK

One fault of the RBAC system is it requires the security administrator to individually lookup the system objects' risk scores. In the future system, it would be beneficial to implement a get all function to simplify work for the security administrator. Also, because the system requires manual lookup, it lacks real-time statistics. By implementing logs and a schedule into the RBAC system, the security administrator could have a concise email sent every morning containing any risks or notification in the system to resolve. The security administrator could take it further by executing a notification service into the RBAC system that sends alerts when users are abusing SoD constraints or risk becomes too high.

Much of the literature focuses on the importance of SoD to prevent users from maintaining too much power over a sensitive process. However, the present RBAC system does not include functionality for this feature. A future iteration of this RBAC system with built in SoD constraints could provide the security administrator with updated risk scores and a better understanding of the permissions, roles, and organization processes. A list of conflicting SoD permissions and process mapped into the RBAC system would limit specific users from accessing too much, limiting risk and further applying PoLP.

Future work should also map the roles to their corresponding resources in the database and RBAC system to ensure the security administrator can verify the involved resources risk scores and reevaluate role and permission provisioning to that resource if needed. Similar to how the users to roles and roles to permissions are mapped. Most roles will have a specific resource to connect to. The RBAC system could even map the resources to permissions to create a more fine-grained system.

Future work could further the user-interface for security administrators by developing a GUI to select commands. Implementing a GUI is decorative and mostly unnecessary, but it may increase the user-friendliness of the RBAC system and ease the job of the security administrator.

VII. CONCLUSION

The RBAC system presented in the study attempted to limit role explosion by providing the security administrator of the system an overview of the users and roles with the hist level of risk to the organization. Each permission maintained an assigned risk score value based on the risk it has to the organization if compromised. The permissions mapped to roles were added to get the total risk score value for each role. Each user risk score was determined by getting the union of all the permissions in all the roles maintained by that user and adding the permission risk scores. Once all the risk score values for the users and roles were logged by the security administrator, charting the scores provided an overview of the highest risk objects for the security administrator to assess.

In the future work, the RBAC system could implement real-time statistics and notification to alert the security administrator of any issues or logs. The system should further implement SoD constraints for users to ensure no user is abusing their power. The system may also map the role or permission to the resource it acts on, allowing the security administrator to retrieve and assess the risk score value of resources as well. The future RBAC system could also update its command console into a GUI to clarify user input options and increase user-friendliness.

The RBAC system implemented and investigated in this study demonstrate the usefulness of risk scoring in an access control environment. By assigning risk score values to permissions, the organization is better prepared to reduce role explosion and review their permission architecture.

REFERENCES

[1] Coyne, Ed, and Tim Weil. “ABAC and RBAC: Scalable, Flexible, and Auditable Access Management.” *IT Professional,* vol. 15, no. 3, IEEE Computer Society, May 2013, pp. 14–16, doi:10.1109/mitp.2013.37.

[2] Kisi, Kait Hobson. “Overview of Access Control Systems.” Security Industry Association, 4 Dec. 2019, www.securityindustry.org/2019/10/08/overview-of-access-control-systems.

[3] Colombo, Pietro, and Elena Ferrari. “Access Control Technologies for Big Data Management Systems: Literature Review and Future Trends.” Cybersecurity, vol. 2, no. 1, Springer Nature, Dec. 2019, doi:10.1186/s42400-018-0020-9.

[4] Ferroni, Stefano. “Implementing Segregation of Duties: A Practical Experience Based on Best Practices.” ISICA. 19 May 2016, https://www.isaca.org/resources/isaca-journal/issues/2016/volume-3/implementing-segregation-of-duties-a-practical-experience-based-on-best-practices.

[5] Bertino, Elisa. “RBAC Models — Concepts and Trends.” Computers & Security, vol. 22, no. 6, Sept. 2003, pp. 511–14, doi: https://doi.org/10.1016/s0167-4048(03)00609-6.

[6] Tolone, William J., et al. “Access Control in Collaborative Systems.” *ACM Computing Surveys*, vol. 37, no. 1, Association for Computing Machinery, Mar. 2005, pp. 29–41, doi:10.1145/1057977.1057979.

[7] Jin, Xin, et al. “A Unified Attribute-Based Access Control Model Covering DAC, MAC and RBAC.” *Springer eBooks,* Springer Nature, July 2012, pp. 41–55, doi:10.1007/978-3-642-31540-4_4.

[8] Liu, Gang; Zhang, Runnan; Wan, Bo; Ji, Shaomin; and Tian, Yumin. “Extended Role-Based Access Control with Context-Based Role Filtering.” *KSII Transactions on Internet and Information Systems*. 31 Mar. 2020, itiis.org/digital-library/23398.

[9] De Carvalho, Marcelo R., and Paulo Bandiera Paiva. “Health Information System Role-Based Access Control Current Security Trends and Challenges.” *Journal of Healthcare Engineering*, vol. 2018, Hindawi Publishing Corporation, Feb. 2018, pp. 1–8, doi:10.1155/2018/6510249.

[10] Strembeck, Mark, and Gustaf Neumann. “An Integrated Approach to Engineer and Enforce Context Constraints in RBAC Environments.” *ACM Transactions on Information and System Security*, vol. 7, no. 3, Association for Computing Machinery, Aug. 2004, pp. 392–427, doi:10.1145/1015040.1015043.

[11] Schaad, Andreas; Moffett, Jonathan; Jacob, Jeremy. “The Role-based Access Control System of a European Bank.” Symposium on Access Control Models and Technologies, May 2001, doi:10.1145/373256.373257.

[12] Idenhaus Consulting. “6 Common Role Based Access Control (RBAC) Implementation Pitfalls.” <https://www.idenhaus.com/rbac-implementation-pitfalls/>. 15 July 2020, www.idenhaus.com/rbac-implementation-pitfalls.

[13] Davis, John. “How to Calculate Cyber Security Risk Value and Cyber Security Risk.” MSI :: State of Security, 10 Jan. 2022, stateofsecurity.com/how-to-calculate-cyber-security-risk-value-and-cyber-security-risk/.

[14] “Glossary | CSRC.” Nist.gov, 2020, csrc.nist.gov/glossary.

[15] Ferraiolo, David & Kuhn, D. “Role-Based Access Control.” *Elsevier eBooks*, Elsevier BV, Jan. 2002, pp. 215–56, doi:10.1016/b978-193183650-0/50027-7.

[16] Sahani, Gurucharansingh, et al. “Scalable RBAC Model for Large-scale Applications With Automatic User-role Assignment.” *International Journal of Communication Networks and Distributed Systems*, vol. 1, no. 1, Inderscience Publishers, Jan. 2022, p. 1, doi:10.1504/ijcnds.2022.10041526.

[17] Jchristn. “Gatekeeper/Program.cs at Master · Jchristn/Gatekeeper.” GitHub, github.com/jchristn/Gatekeeper/blob/master/GateKeeperConsole/Program.cs.

[18] Jchristn. “GitHub - Jchristn/WatsonORM: WatsonORM Is a Lightweight and Easy to Use Object-relational Mapper (ORM) in C# for .NET Core.” *GitHub*, github.com/jchristn/watsonorm.

APPENDIX A. DEFINITIONS

Attribute-based access control (ABAC): an adaptation of access control that provisions or revokes access based on user attributes, with respect to the requested object, the type of access requested, and the development environment hosting the task.

Discretionary Access Control (DAC): an approach to access control that requires users to manage their own permissions on the resources that they curate.

Dynamic Separation of Duties (DSoD): a method of SoD implementation that allows exceptions to be attached to roles that permit users to exercise potentially conflicting roles in allowable ways.

Extended RBAC: an enhancement of RBAC that provides context-based role filtering to dynamically suppress access to roles that users will not need to access at a given point in an organization’s operation.

Funktionale Berechtigung (FUB): a custom implementation of RBAC developed and implemented by Dresdner bank.

Mandatory Access Control (MAC): an approach to access control that requires administrators to manually define all possible combinations of access permissions and rules that any user might require at any point in an organization’s operations.

Principle of Least Privilege: a concept that limits user access to content required just for the work required of them, thereby reducing the potential damage from insider threats.

Role: a collection of permissions to authorize a user.

Role Based Access Control (RBAC): access control based on roles given to users or groups given their job requirements and position within an organization.

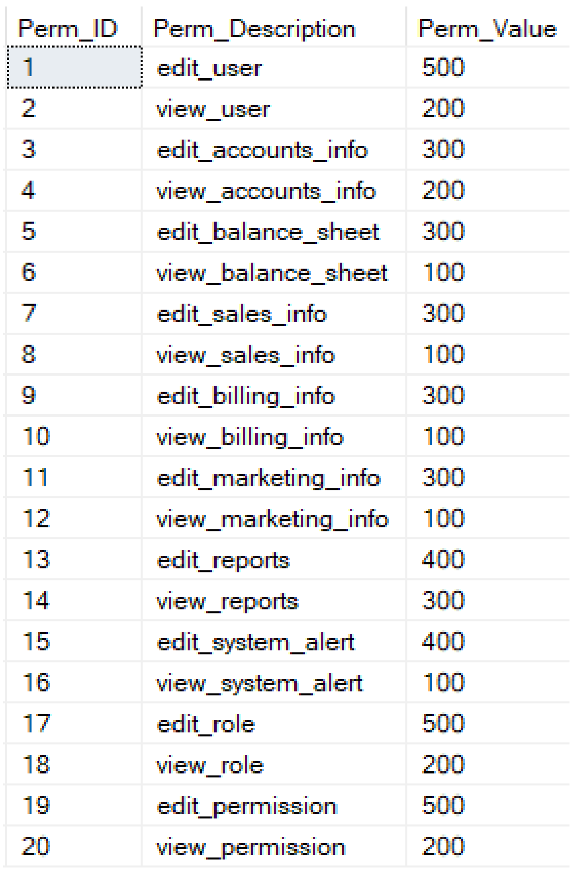
Separation of Duties (SoD): restricting the permissions that a given entity can acquire, ensuring that multiple entities share responsibility for carrying out a potentially sensitive process.

Static Separation of Duties (SSoD): a method of SoD implementation so no role should ever have any combination of permissions that could possibly give rise to a conflict of interest based on conflicting roles.

Threat Source: the intent and method targeted at the intentional exploitation of a vulnerability or a situation and method that may accidentally trigger a vulnerability.

Vulnerability: weakness in an information system, system security procedures, internal controls, or implementation that could be exploited or triggered by a threat source.

APPENDIX A. PERMISSIONS



Perm_ID	Perm_Description	Perm_Value
1	edit_user	500
2	view_user	200
3	edit_accounts_info	300
4	view_accounts_info	200
5	edit_balance_sheet	300
6	view_balance_sheet	100
7	edit_sales_info	300
8	view_sales_info	100
9	edit_billing_info	300
10	view_billing_info	100
11	edit_marketing_info	300
12	view_marketing_info	100
13	edit_reports	400
14	view_reports	300
15	edit_system_alert	400
16	view_system_alert	100
17	edit_role	500
18	view_role	200
19	edit_permission	500
20	view_permission	200

Permission data created for RBAC system