Detection of Malicious URLs using ML with Role Base Access Control Policy

A CS814 Course Project Report

Under the guidance of
Dr. Mahendra Pratap Singh

Submitted By

HIMANSHU KASHYAP (202CS012)
PRANAB NANDY (202CS019)

Department of Computer Science and Engineering

National Institute of Technology Karnataka

P.O. Srinivasnagar, Surathkal, Mangalore-575025 Karnataka, India

January 2021

Table of Content

1. Introduction

The primitive usage of URL (Uniform Resource Locator) is to use as a Web Address. However, some URLs can also be used to host unsolicited content that can potentially result in cyber attacks. These URLs are called Malicious URLs. Malicious URLs host unsolicited content like spam, phishing, drive by downloads etc. and lure unsuspecting users to become victims of scams (monetary loss, theft of private information and malware installation) and cause losses of billions of dollars every year. Thus, it is imperative to detect and act on such threats in a timely manner.

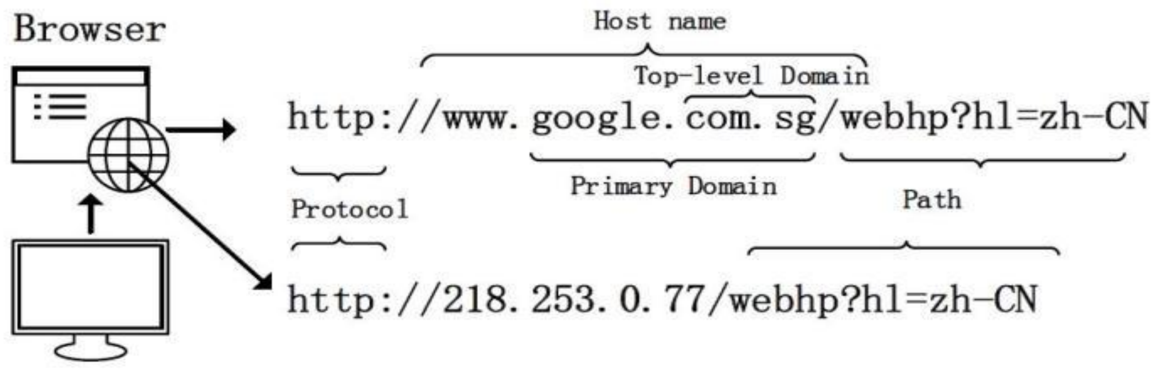

Browser
Host name
Top-level Domain
http://www.google.com.sg/webhp?hl=zh-CN
Protocol
Primary Domain
Path
http://218.253.0.77/webhp?hl=zh-CN

Figure 1: Example of a URL - “Uniform Resource Locator"

In this project, we have put a special emphasis on the detection of Malicious URLs. Detecting the URLs whether malicious or safe can be achieved by Machine Learning’s classification algorithms. Once it is classified that URL is malicious then it can be stopped to make HTTP Requests to the server.

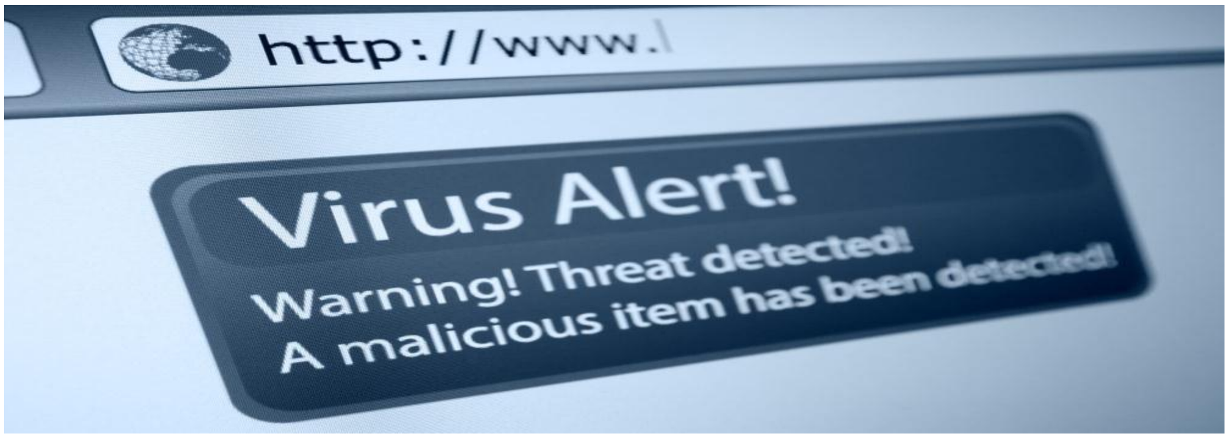

http://www.
Virus Alert!
Warning! Threat detected!
A malicious item has been detected!

Figure 2: Malicious URL

A machine learning model is a file that has been trained to recognize certain types of patterns. You train a model over a set of data, providing it an algorithm that it can use to reason over and learn from those data. Here we used 2 models to classify the label. Basically we used the Supervised Machine Learning Classification algorithm to train and predict the data. Those models are as follows,

- **Multinomial Naïve Bayes**
- **Logistic Regression**

Multinomial Naïve Bayes is based on Bayes' theorem, where the adjective Naïve says that features in the dataset are mutually independent. Occurrence of one feature does not affect the probability of occurrence of the other feature. For small sample sizes, Naïve Bayes can outperform the most powerful alternatives. Being relatively robust, easy to implement, fast, and accurate, it is used in many different fields.

$$P(\text{word}|\text{class}) = \frac{\text{Count}(\text{word},\text{class})+\alpha}{\text{Count}(\text{class})+\alpha\cdot\text{Count}(\text{unique words})}$$

Logistic regression is a supervised learning classification algorithm used to predict the probability of a target variable. The nature of target or dependent variable is dichotomous, which means there would be only two possible classes. In simple words, the dependent variable is binary in nature having data coded as either 1 (stands for success/yes) or 0 (stands for failure/no). Mathematically, a logistic regression model predicts P(Y=1) as a function of X.

It is one of the simplest ML algorithms that can be used for various classification problems such as spam detection, Diabetes prediction, cancer detection etc. Generally, logistic regression means binary logistic regression having binary target variables, but there can be two more categories of target variables that can be predicted by it. Based on those number of categories, Logistic regression can be divided into following types - Binary or Binomial and Multinomial.

We first prepared the gathered data through 2 kinds of vectorizer i.e count vectorizer and TFIDF vectorizer where we used defined tokenizer function. Then we train our model though train data and evaluate its performance against test data before deployment. In total we have 2 models with 2 kinds of vectorizer as follows,

- **Multinomial Naïve Bayes with count vectorizer**
- **Multinomial Naïve Bayes with TFIDF vectorizer**
- **Logistic Regression with count vectorizer**
- **Logistic Regression with TFIDF vectorizer**

Basically 2 kinds of actions we can perform with these model object which are as follows

- **Test whether a URL is malicious or not**
- **Train the model again by changing some Hypermeters**

In this project we have 2 hypermeters one is the alpha factor of Multinomial Naïve Bayesian and percentage of test data i.e it splits the dataset into 2 parts i.e training dataset and test dataset. In our system, we have a dataset of size 444321 having a good combination of Normal URLs and Malicious URLs.

After that we implement the Role Based Access Control policy to distinguish the roles among different set of users. As per RBAC policy, system must have set of users , set of Roles and set of Permissions. We created different set of roles and associated the permissions with corresponding roles. Here we also used the session concept by capturing some log files in the database. We have also put special emphasis on the scalability of the developed system. Here everything is managed by admin and nothing is hard coded. So flexibility with respect to addition and deletion of features as per future requirement is always there.

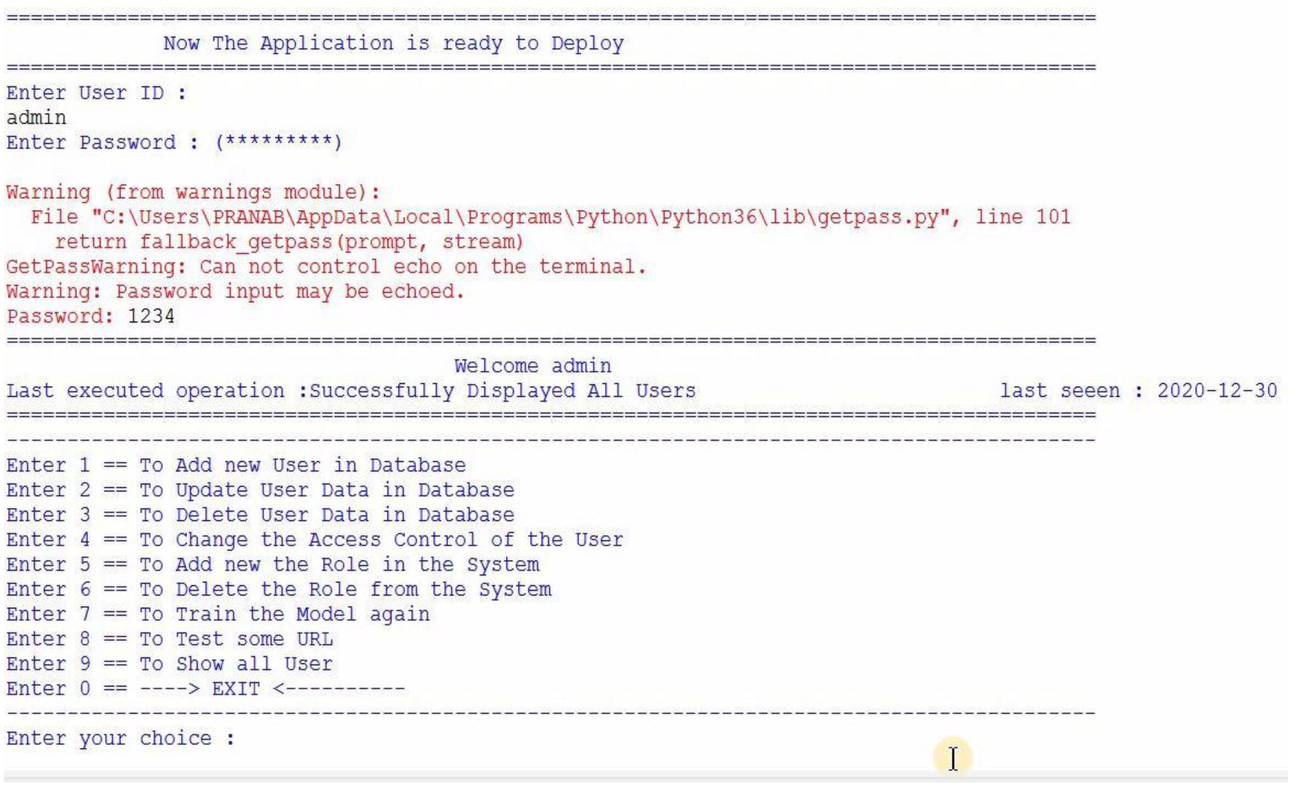
Entire project is developed in **Python language** with some set of available libraries and here SQLite3 is used as Database.

Used libraries : **pandas, numpy, matplotlib, random, re, sqlite3, datetime, getpass, datetime**

Here is dataset is present in our system. The dataset is directly loaded to the program with help of pandas library.

url_df = **pd.read_csv(**r'C:\Users\PRANAB\Downloads\URL2.csv'**)**

This project deals with CLI (Command Line Interface). So to visualize the data in the table one can Install SQLite3 application . This application takes approximately 2 minutes to train the models then it will be ready to deploy for user interaction. Here is the clips of the application for admin user just after login in the system.



```
==========================================================================================
             Now The Application is ready to Deploy
==========================================================================================
Enter User ID :
admin
Enter Password : (*********)

Warning (from warnings module):
  File "C:\Users\PRANAB\AppData\Local\Programs\Python\Python36\lib\getpass.py", line 101
    return fallback_getpass(prompt, stream)
GetPassWarning: Can not control echo on the terminal.
Warning: Password input may be echoed.
Password: 1234
==========================================================================================
                                     Welcome admin
Last executed operation :Successfully Displayed All Users                         last seeen : 2020-12-30
==========================================================================================
------------------------------------------------------------------------------------------
Enter 1 == To Add new User in Database
Enter 2 == To Update User Data in Database
Enter 3 == To Delete User Data in Database
Enter 4 == To Change the Access Control of the User
Enter 5 == To Add new the Role in the System
Enter 6 == To Delete the Role from the System
Enter 7 == To Train the Model again
Enter 8 == To Test some URL
Enter 9 == To Show all User
Enter 0 == ----> EXIT <----------
------------------------------------------------------------------------------------------
Enter your choice :
```

Fig 3 : Interface of admin user

2. Authorization

Role-based access control (RBAC) is a policy-neutral access-control mechanism defined around roles and privileges. The components of RBAC such as role-permissions, user-role and role-role relationships make it simple to perform user assignments. A study by NIST has demonstrated that RBAC addresses many needs of commercial and government organizations. RBAC can be used to facilitate administration of security in large organizations with hundreds of users and thousands of permissions. Although RBAC is different from MAC and DAC access control frameworks, it can enforce these policies without any complication.

Within an organization, roles are created for various job functions. The permissions to perform certain operations are assigned to specific roles. Members or staff (or other system users) are assigned particular roles, and through those role assignments acquire the permissions needed to perform particular system functions. Since users are not assigned permissions directly, but only acquire them through their role (or roles), management of individual user rights becomes a matter of simply assigning appropriate roles to the user's account , this simplifies common operations, such as adding a user, or changing a user's department.

Role based access control interference is a relatively new issue in security applications, where multiple user accounts with dynamic access levels may lead to encryption key instability, allowing an outside user to exploit the weakness for unauthorized access. Key sharing applications within dynamic virtualized environments have shown some success in addressing this problem.

Three primary rules are defined for **RBAC** :

- **Role assignment:** A subject can exercise a permission only if the subject has selected or been assigned a role.

- **Role authorization:** A subject's active role must be authorized for the subject. With rule 1 above, this rule ensures that users can take on only roles for which they are authorized.

- **Permission authorization:** A subject can exercise a permission only if the permission is authorized for the subject's active role. With rules 1 and 2, this rule ensures that users can exercise only permissions for which they are authorized.

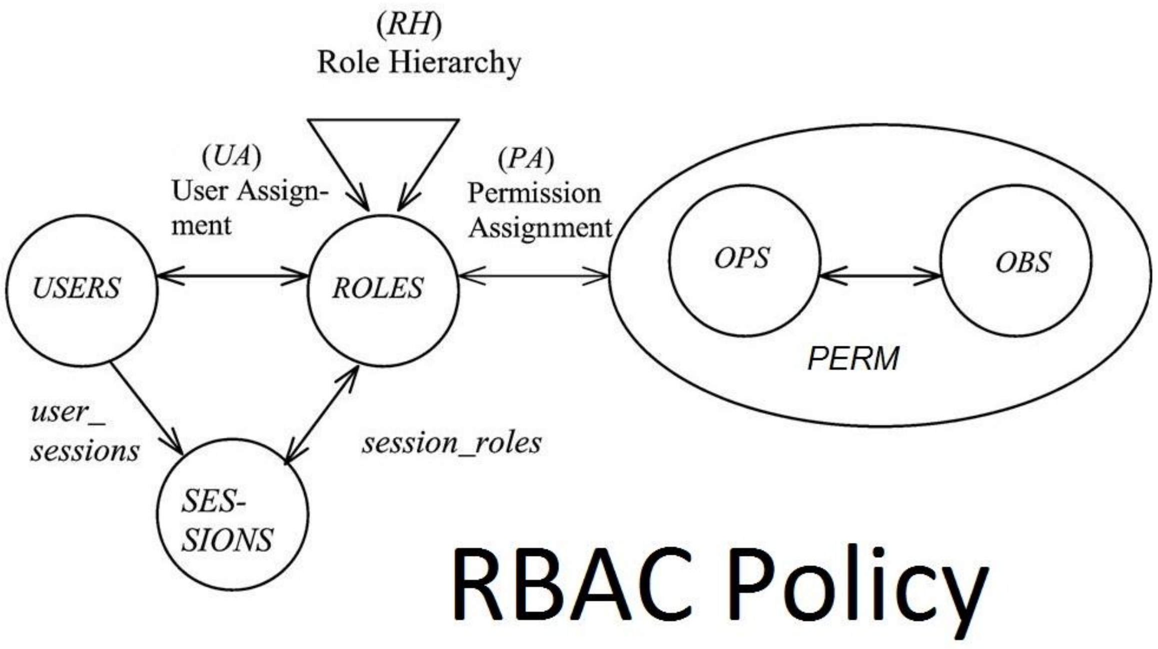


Figure 4: RBAC Policy

Additional constraints may be applied as well, and roles can be combined in a hierarchy where higher-level roles subsume permissions owned by sub-roles. With the concepts of role hierarchy and constraints, one can control RBAC to create or simulate lattice-based access control (LBAC). Thus RBAC can be considered to be a superset of LBAC.

When defining an RBAC model, the following conventions are useful:

- U = User or Subject = A person or automated agent
- R = Role = Job function or title which defines an authority level
- P = Permissions = An approval of a mode of access to a resource
- SE = Session = A mapping involving S, R and/or P'
- URA = User Role Assignment
- PRA = Permission Role Assignment
- RH = Partially ordered Role Hierarchy. RH can also be written: ≥ (The notation: x ≥ y means that x inherits the permissions of y.)

A constraint places a restrictive rule on the potential inheritance of permissions from opposing roles, thus it can be used to achieve appropriate separation of duties . For example, the same person should not be allowed to both create a login account and to authorize the account creation.

Thus, using set theory notation:

- $PA \subseteq P \times R$ and is a many to many permission to role assignment relation.
- $SA \subseteq S \times R$ and is a many to many subject to role assignment relation.
- $RH \subseteq R \times R$

In case of Database Implementation, we first analyze our requirement and try to come up with database schema. Here we have URA table where we store the user information along with its role. We could have created separate table for USER and URA but as URA table does not contains any redundant information we proceed with one table.

An user can have multiple role .But here I used the concept of Role Hierarchy. Like admin role can inherit all property of all other role as admin role is senior to all other role. So we don't need multiple role for a certain user. We can embedded all required permission to a single role for a particular set of user.

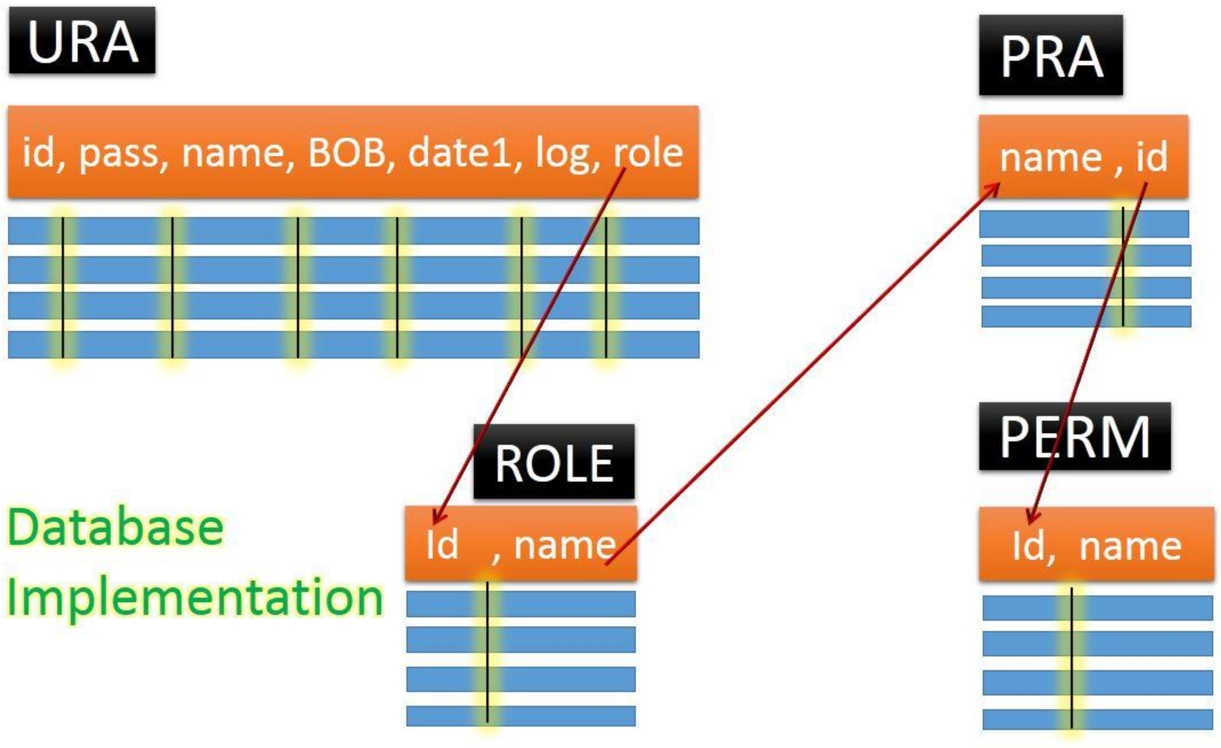
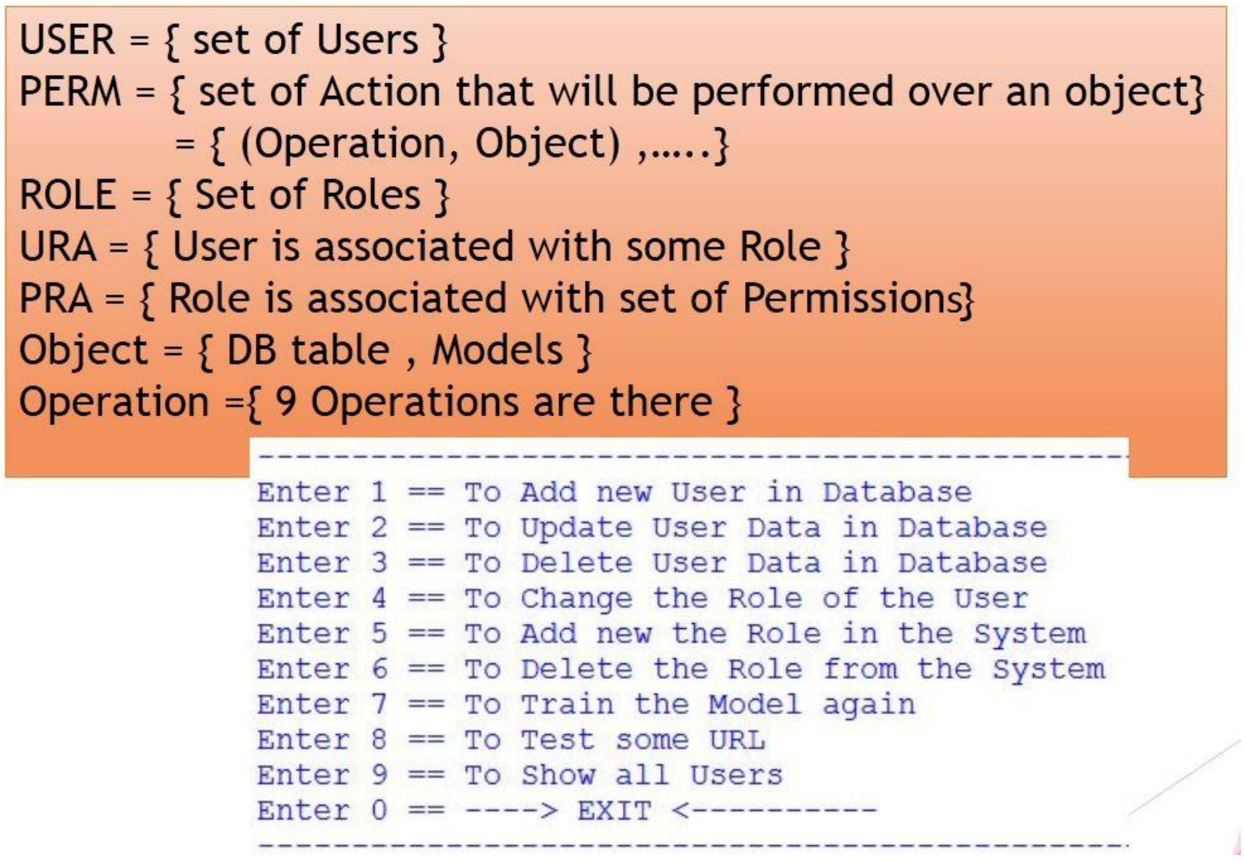


Figure 5: Database Implementation

We have ROLE table which contains role ID and role NAME .Here role id is the refered key and role value from URA table is the referring key. It creates an foreign key relationship with ON DELETE CASCADE property.

We have PRA table which basically means permission role assignment .Here it assign the permissions with the given role. Also here name attribute in PRA is the referenced attribute and id attribute in ROLE is the refering attribute. Here relationship has ON DELETE CASCADE property.

Lastly we have PERM table where all permissions are mentioned. Each permission has some unique id which is associated with it, Here id in PERM is referred by id in PRA table. So it creates a foreign key relationship with ON DELETE CASECADE property.



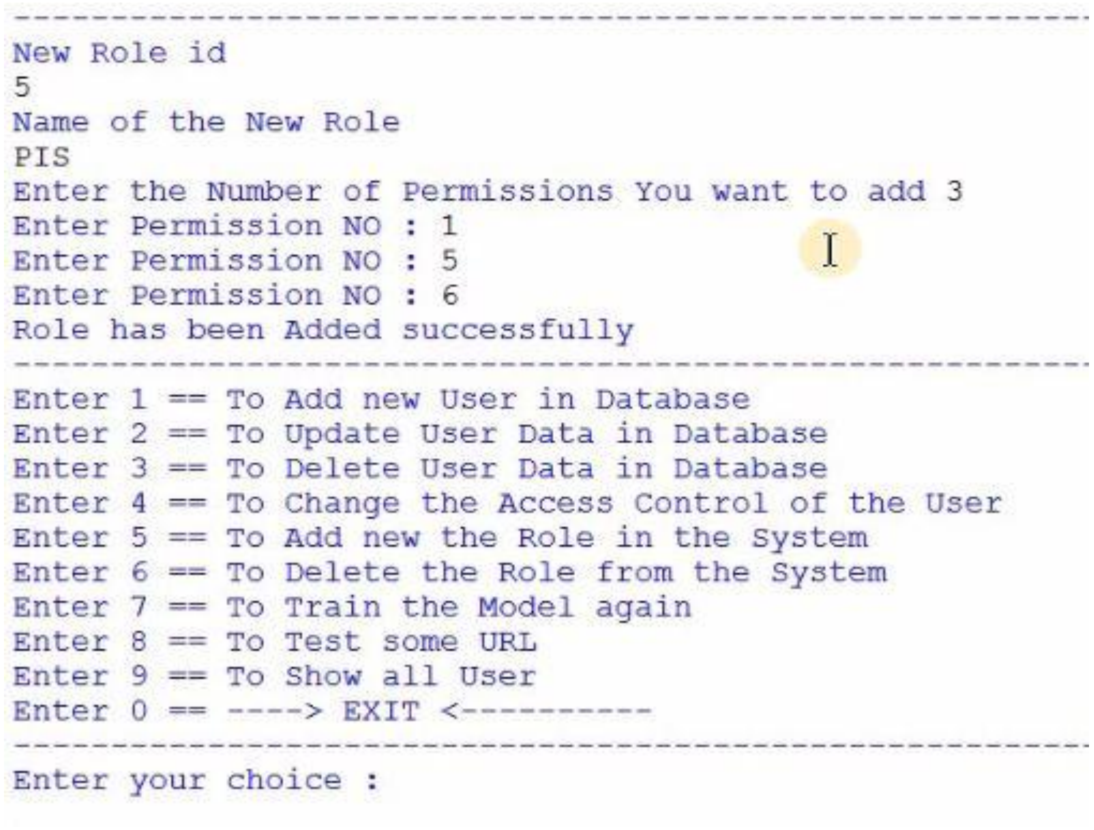
USER = { set of Users }
PERM = { set of Action that will be performed over an object}
= { (Operation, Object) ,…..}
ROLE = { Set of Roles }
URA = { User is associated with some Role }
PRA = { Role is associated with set of Permissions}
Object = { DB table , Models }
Operation ={ 9 Operations are there }

```
---------------------------------------------
Enter 1 == To Add new User in Database
Enter 2 == To Update User Data in Database
Enter 3 == To Delete User Data in Database
Enter 4 == To Change the Role of the User
Enter 5 == To Add new the Role in the System
Enter 6 == To Delete the Role from the System
Enter 7 == To Train the Model again
Enter 8 == To Test some URL
Enter 9 == To Show all Users
Enter 0 == ----> EXIT <----------
---------------------------------------------
```

Figure 6: Permissions of RBAC system in our Database

The following diagram implies how we implement the specification of Role Based Access Control Policy in our Database table. An user can Admin User , Premium User , Normal User etc. Role can be defined as Admin Role, Member Role , Normal User Role or any user defined role. Each role needs to associate some set of permission to perform certain task.

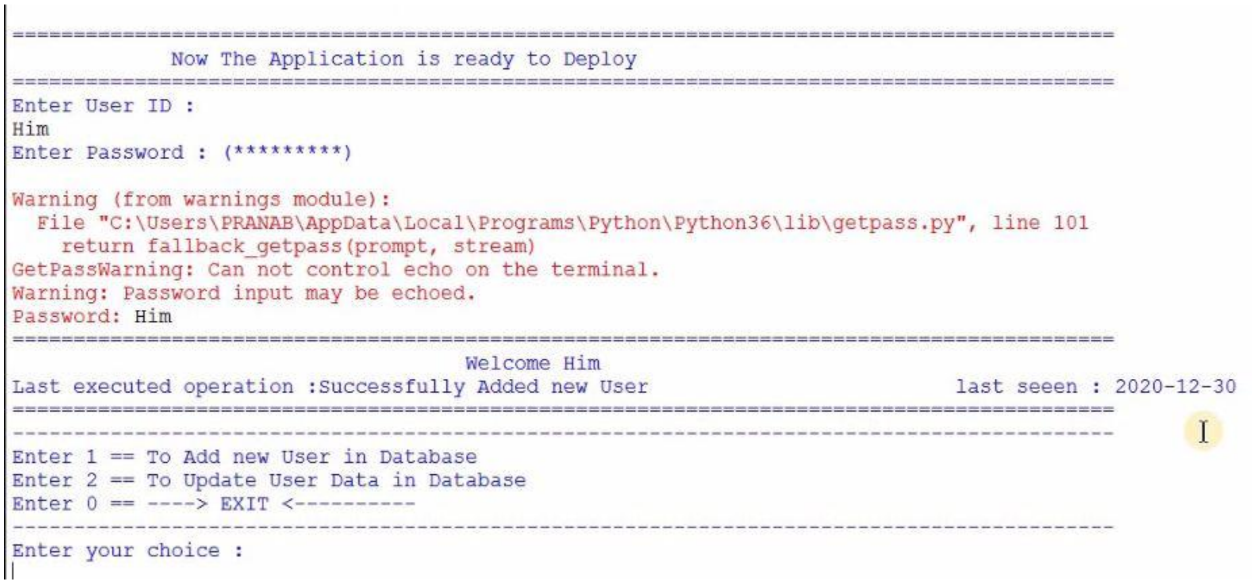
In our system user can create any kind of role through a good combination of 9 permissions. So set of Roles in our application is not fixed . Based on the requirement , Roles can be created and deleted but you can not the change the permission of the existing role. In that case, first you need to create the new Role then assign the required users to that newly created role and delete the old role.



```
--------------------------------------------------------
New Role id
5
Name of the New Role
PIS
Enter the Number of Permissions You want to add 3
Enter Permission NO : 1
Enter Permission NO : 5
Enter Permission NO : 6
Role has been Added successfully
--------------------------------------------------------
Enter 1 == To Add new User in Database
Enter 2 == To Update User Data in Database
Enter 3 == To Delete User Data in Database
Enter 4 == To Change the Access Control of the User
Enter 5 == To Add new the Role in the System
Enter 6 == To Delete the Role from the System
Enter 7 == To Train the Model again
Enter 8 == To Test some URL
Enter 9 == To Show all User
Enter 0 == ----> EXIT <----------
--------------------------------------------------------
Enter your choice :
```

Figure 7: Creation of a New Role in the System

The below pictures illustrates the interface a user for a Newly Created Role. The system has the capacity to capture the log file also. Like the user 'Him' performed his last operation which is Adding a new User and date of the last performed operation.



```
==========================================================================================
             Now The Application is ready to Deploy
==========================================================================================
Enter User ID :
Him
Enter Password : (*********)

Warning (from warnings module):
  File "C:\Users\PRANAB\AppData\Local\Programs\Python\Python36\lib\getpass.py", line 101
    return fallback_getpass(prompt, stream)
GetPassWarning: Can not control echo on the terminal.
Warning: Password input may be echoed.
Password: Him
==========================================================================================
                                     Welcome Him
Last executed operation :Successfully Added new User                         last seeen : 2020-12-30
==========================================================================================
------------------------------------------------------------------------------------------
Enter 1 == To Add new User in Database
Enter 2 == To Update User Data in Database
Enter 0 == ----> EXIT <----------
------------------------------------------------------------------------------------------
Enter your choice :
```

Fig 8 : Interface of an user based on Role

3. Conclusion

Despite many exciting advances over last decade for malicious URL detection using machine learning techniques and RBAC policy , there are still many open problems and challenges which are critical and imperative.

Most existing malicious URL detection approaches by machine learning are based on supervised learning techniques, which require labeled training data including both benign and malicious URL data. The labeled data can be obtained by either asking human experts to label or acquiring from blacklists/whitelists (which were often also labeled manually). Unfortunately the scale of such labeled data is tiny as compared to the size of all available URLs on the web .For example, one of the largest public available malicious URL training data sets in academia has only 2.4 million URLs.

Our System has predefined set of permission i.e we are having 9 set of permission which implies the maximum operation an admin can do with the system. But if anyone wants to add some new features then it should be done though database indirectly. This features is not present in our application. Either you can write separate program to do it or you can do with SQLite3 database manager.

Role Explosion happens when the level of granularity needed for your access control is too detailed. Role Explosion is difficult and costly to manage and makes access control confusing and complicated, reducing the access control effectiveness. Additionally, there are several other issues created that need to be monitored carefully when adding more roles to your access control deployment . One of these problems occurs when a user has too many roles assigned to them and then changes jobs or responsibilities within the company. IT system administrators either forget, or even make a conscious decision to leave old roles in place. The quantity of roles can lead to security holes that are often too difficult to find and close.

As a system administrator, it is important to understand the risks to your system. Conducting a security risk analysis with a proactive risk prevention plan is essential for RBAC deployment. RBAC is data focused; data is categorized relevant to the organizational structure and that leads to access control role definition. If your organization is reactive to security risks, RBAC may not be the optimal way of securing access to your network data . RBAC requires that you have intimate knowledge of the security layout of your company and of how permissions are being granted before deployment. Once deployed, it is hard to react to changing security threats and risks. So be careful and “measure twice, and cut once” with your RBAC policies.

4. Bibliography

1. A Astorino, A Chiarello, M Gaudioso, and A Piccolo. 2016. Malicious URL detection via spherical classification. Neural Computing and Applications (2016).

2. Anna L Buczak and Erhan Guven. 2016. A survey of data mining and machine learning methods for cyber security intrusion detection. IEEE Communications Surveys & Tutorials (2016).

3. S ANDHU , R. AND B HAMIDIPATI , V. 1997. The URA97 model for role-based administration of user-role assignment. In Database Security XI: Status and Prospect, T. Y. Lin and X. Qian, Eds. Elsevier North-Holland, Inc., Amsterdam, The Netherlands.

4. S ANDHU , R. 1997. Roles versus groups. In Proceedings of the 2nd ACM Workshop on Role-Based Access Control (Fairfax, VA, Nov. 6-7). ACM Press, New York, NY.

5. S ANDHU , R. S. 1992. The typed access matrix model. In Proceedings of the ACM Symposium on Research in Security and Privacy (Oakland, CA, May). 122–136.