Preventing permission escalation in chat apps utilizing role "permission weighting"

Mauro M.*

*Independent Researcher, hello@maurom.dev

Abstract

With the rise of online chat apps with a hierarchical role system, the likes of Discord and Teamspeak, since communities frequently have over one hundred thousand members, the need for automation protection systems for these communities has risen proportionally. One of the desired features in the aforementioned protection systems is the ability to prevent malicious actors from gaining more permissions or access than they had been initially granted.

Given the amount of access a member has is directly derived from the member's roles there is the need to quantify the amount of access a particular role grants a member. This paper puts forward an algorithm that numerically quantifies this access in a comparable manner by attributing each permission a numerical weight. This weight is based on the amount of damage a malicious actor could exert upon the community, furthermore, the percentage of channels the role grants a member access to is also taken into account. This permits protection system developers to assess the risk of a member having a given role and act accordingly.

1 Definitions

Member *or* User A user of a chat app, member is used to describe a user when they are in a server.

Server *or* Guild Represents a group of users in a chat-app. Contains channels, members and roles.

Channel Servers are organized in to named text channels where users can post messages, upload files, and share images for others to see at any time.

Role Roles can be assigned to members and have permissions and a name. These can be used to set up a hierarchical system where some users have more permissions than others, to, among other cases, moderate servers.

2 Weighing Permissions

The most straightforward way to weigh permissions is to arrange them in an array, ordering them in a hierarchical manner of ascending importance. Importance is frequently subjective but should always account for the amount of damage a potential malicious actor could do with that permission.

```
[Manage Members, Manage Roles, Manage
 Channels, Manage Server, Kick, Ban,
           Administrator]
```

Figure 1. Example of a ordered array with permissions

Consider the following example utilizing permissions from a Discord role:

Weight can simply be attributed by taking the permission's index and adding one (if index is i and the permission weight is P_w, $P_w = i + 1$), for the example in Figure 1.

Name	Index (i)	Weight (P_w)
Manage Members	0	1
Manage Roles	1	2
Manage Channels	2	3
Manage Server	3	4
Kick	4	5
Ban	5	6
Administrator	6	8

Table 1. Example of table with index-based weights

Given a certain permission name (as$permission_name$, and a list of permissions (akin to Figure 1 as $permission_list$) a permissions weight can be obtained as simply as:

```
def get_permission_weight(permission_name:
    str) -> int:
    try:
        return
            permission_list.index(permission_name)
            + 1
    except IndexError:
        return 0
```

Figure 2. Sample function in Python to obtain the weight of a permission

It is important to note that this method is necessarily $O(n)$. There are ways to improve the overall speed of the algorithm for larger data sets however this paper will not contemplate these as roles usually have no more than 40 permissions making the time difference effectively negligible.

3 Weighing Roles

Put simply, the role weight is the product between the mean of the role's permission's weight and the percentage of channels the member can access.

The role permission weight (R_w) can be equated as:

$$R_w = \frac{\sum P_w}{l} \cdot \left(\frac{C_a}{C_t} \cdot 100 \right) \tag{1}$$

Where:

- P_w Represents the weight of each permission
- l is the amount of permissions
- C_a is the amount of channels the member can access
- C_t is the total amount of channels in the guild
- $C_t, l, C_a > 0$

Given a certain role object, with a property containing permissions objects ($role.permissions$) that have a $name$ property, the total amount of channels in a guild as $total\_channels$ and the amount of channels the member can access as $accessible\_channels$ the weight of a role can be obtained as:

```
def get_role_weight(role, total_channels:
    int, accessible_channels: int) -> float:

   weights =
       [get_permission_weight(permission.name)
       for permission in role.permissions]

   return (sum(weights)/len(weights)) *
       ((total_channels /
       accessible_channels) * 100)
```

Figure 3. Sample function in Python to obtain the weight of a role.

It is important to note that the function is very time complex. The $accessible\_channels$ integer requires an iteration through all of a guild's roles and a check for permissions for the user's roles, furthermore the example function itself (Figure 3) requires an iteration through all of the the role's permissions ultimately making the function $O(n^2)$.

4 Implementation Notes

Due to the pertinent time complexity of the methodology, if aiming for maximum efficiency it is advised to subscribe to a role creation and modification event (if the API used provides them) and determine the weights upon role creation or modification and store them in a form of persistent memory.

The algorithm itself can be used to determine how much access a user gains with a certain role (ΔR_w) allowing guild protection systems to recognize and mitigate potential permission escalations by unauthorized members.

Additional Information and Declarations

Competing Interests

There are no competing interests