User Management Implementation in Django

This documentation outlines the implementation of a user management system using Django Admin and custom permission handling for pages.

Models Implementation

1. `Page` Model

The `Page` model defines the pages available in the application, each with a unique name.

```python
from django.db import models

class Page(models.Model):
    name = models.CharField(max_length=50, unique=True)

    def __str__(self):
        return self.name  # Display the Page name in the admin a
```

2. `PagePermission` Model

The `PagePermission` model manages the permissions for user groups on specific pages. It supports read and write permissions.

```python
from django.db import models
from django.contrib.auth.models import Group

class PagePermission(models.Model):
    group = models.ForeignKey(Group, on_delete=models.CASCADE)
    page = models.ForeignKey(Page, on_delete=models.CASCADE)
    can_read = models.BooleanField(default=False)
```

```
    can_write = models.BooleanField(default=False)

    def __str__(self):
        return f"{self.group.name} - {self.page.name} (Read:
{self.can_read}, Write: {self.can_write})"
```

Decorator for Permission Checks

To enforce page permissions, a custom decorator `check_page_permissions` has been implemented. This decorator validates whether the current user has the appropriate permissions for the specified page.

Directory Structure

```
Unset
utils/
|-- __init__.py
|-- permissions.py
```

Decorator Implementation

```
Python
from functools import wraps
from django.shortcuts import render, get_object_or_404
from home.models import Page, PagePermission

def check_page_permissions(page_name):
    def decorator(view_func):
        @wraps(view_func)
        def wrapper(request, *args, **kwargs):
            # Fetch the page and user's group
            page = get_object_or_404(Page, name=page_name)
            user_group = request.user.groups.first()
```

```
            permission =
PagePermission.objects.filter(group=user_group,
page=page).first()

            # No permission
            if not permission:
                return render(request, 'no_permission.html',
{'message': "You do not have permissions for this page."})

            # Read-only permission
            if permission.can_read and not permission.can_write:
                return render(request, 'less_priv.html', {
                    'message': "You do not have permission to
perform this action. Please contact the administrator."
                })

            # Allow access if write permission exists
            if permission.can_write:
                return view_func(request, *args, **kwargs)

            # Fallback
            return render(request, 'no_permission.html',
{'message': "You do not have permissions for this page."})
        return wrapper
    return decorator
```

Usage

To apply the decorator to a view, use the following syntax:

Python

```python
from utils.permissions import check_page_permissions

@check_page_permissions('page_name')
def my_view(request):
```

```
    # View logic here
    pass
```

Note: Replace `page_name` with the actual name of the page as registered in the admin panel.

Flow of Execution:

1. **User Requests a Page**:
    - A view function protected by the `check_page_permissions` decorator is triggered.
2. **Decorator Activates**:
    - The decorator fetches the page using `page_name` and the user's group.
    - It checks the `PagePermission` model for permissions.
3. **Decision Path**:
    - **No Permission**: Redirects to `no_permission.html`.
    - **Read-Only Permission**: Redirects to `less_priv.html` for restricted access feedback.
    - **Write Permission**: Allows access to the view function.
4. **Fallback**:
    - If no conditions are met, redirects to the `no_permission.html` template.

Component	Responsibility	Execution Flow
`Page` Model	Represents individual pages in the system. Each page is identified by a unique name.	Admin registers pages in the Django Admin panel.
`PagePermission` Model	Manages group-level permissions (read/write) for each page.	Admin assigns permissions (read/write) to groups for specific pages via the Django Admin panel.
Decorator (`check_page_permissions`)	Enforces page-level permissions for views dynamically.	1. Intercepts requests to views. 2. Fetches the page and user's group. 3. Validates permissions and decides access.

Permission Validation Logic	Provides fine-grained control over user actions based on their permissions.	Redirects to `no_permission.html` or `less_priv.html` for insufficient permissions, or grants access to the view.
`check_permission` **Utility**	Checks permissions programmatically for operations outside views.	1. Verifies if the user has the requested permission (read/write). 2. Returns `True` or `False` based on the result.
User Groups	Associates users with groups to manage permissions.	Users are assigned to groups via Django Admin. Group-level permissions dictate access.
Django Admin Panel	Provides an interface for managing pages and permissions.	Admins define pages and configure permissions for groups directly in the admin interface.

User Login

The `user_login` function handles user authentication for a web application. It verifies user credentials and manages login sessions using Django's authentication framework. Below is a detailed breakdown of its functionality:

Functionality

The `user_login` function allows users to log in to the application by providing a valid username and password. It checks the provided credentials against the database and manages the session upon successful authentication.

Core Components

Input Handling:

- **Fields:** Retrieves `username` and `password` from the POST request when the login form is submitted.
- **Validation:** Ensures both fields are populated before proceeding with authentication.

Authentication Process:

- **Django Authentication:**

    - Calls the `authenticate()` method to verify credentials.
    - If valid, returns a `user` object; otherwise, returns `None`.
- **Successful Authentication:**
    - Calls `login()` to create a session for the authenticated user.
    - Displays a console message indicating successful authentication.
    - Redirects the user to the `home` page.
- **Failed Authentication:**
    - Displays a console message indicating failed authentication.
    - Adds an error message using `messages.error()`.
    - Redirects the user back to the `login` page.

UI Rendering:

- Renders the `login.html` template if the request method is `GET` or if authentication fails.

Firewall Guide

Prerequisites

- **Vuurmuur version**: 0.8.1

Installation from Source

1. Download the Archive

Download the source archive, e.g., `vuurmuur-0.8.1.tar.gz`, and save it to your disk.

2. Unpack the Archive

Use the following command to unpack the archive:

```
Unset
1. gzip -cd vuurmuur-0.8.1.tar.gz | tar xvf -
```

3. Enter the Directory

Navigate to the extracted directory:

```
Unset
2. cd vuurmuur-0.8.1
```

4. Run the Installation Script

Execute the installer script with default settings:

```
Unset
3. ./installer/install.sh --install --defaults
```

This will configure:

- **Configuration files**: `/etc/vuurmuur`
- **Binaries**: `/usr/bin/`
- **Miscellaneous files**: `/usr/share/vuurmuur`
- **Logs**: `/var/log/vuurmuur`

Upon successful installation, a confirmation message will appear.

5. Enable Vuurmuur on Boot

To ensure Vuurmuur starts automatically during system boot, configure it using the `vuurmuur-initd.sh` script located at:

```
Unset
4. /usr/share/vuurmuur/scripts/
```

6. Set Up an Initial Configuration

Run the configuration wizard to generate an initial setup:

```
Unset
5. vuurmuur_conf --wizard
```

Vuurmuur Interface Overview

The Vuurmuur interface `vuurmuur_conf` includes various options and menus to manage and monitor the firewall. Here is a brief explanation of the main sections:

- **Rules (F9)**: View and manage firewall rules.
- **BlockList (b)**: Manage lists of blocked IPs or ranges.
- **Zones (F7)**: Define network zones for simplified rule management.
- **Interfaces**: Manage and configure network interfaces.
- **Services (F8)**: Manage predefined services for use in rules.
- **Vuurmuur Config (F6)**: Access and modify Vuurmuur configuration settings.
- **Logview**: View firewall logs to monitor activity.
- **Status (s)**: Check the current status of the firewall.
- **Connections (c)**: Monitor active connections passing through the firewall.
- **Traffic Volume (a)**: Analyze traffic volume data.

- **Vuurmuur_conf Settings**: Configure advanced settings using the Vuurmuur configuration tool.
- **Apply Changes (F11)**: Save and apply any configuration changes made.
- **About**: View information about Vuurmuur.
- **Quit**: Exit the Vuurmuur interface.

Rules

Here's a detailed explanation of the `rules` function, along with where it saves configurations and other key behaviors:

Purpose

The `rules` function handles the creation and management of firewall rules in a Django web application. It provides an interface for users to add rules to the Vuurmuur firewall system, based on their permissions.

Key Features and Flow

1. **Permission Handling**:
    - The function checks the permissions associated with the user's group (`user_group`) and the `rules` page.
    - If the user lacks permissions, they are redirected to the `no_permission.html` page with an appropriate message.
    - If the user has read-only permissions, they see an error on the `less_priv.html` page.
2. **Rule Submission**:
    - Users with write permissions can submit rules via a POST request.
    - Submitted data includes:
        - `selectedAction`: The action (e.g., Accept, Drop).
        - `selectedService`: The service for the rule.
        - `selectedSource` and `selectedDestination`: Source and destination zones.
        - `comment`, `in_max`, `out_max`, `in_min`, `out_min`: Additional rule details.
    - If all required fields (`selectedAction`, `selectedService`, `selectedSource`, `selectedDestination`) are provided:
        - A rule string is generated in the format:

```
Unset
RULE="{selected_action} service {selected_service} from
{selected_source} to {selected_destination} options
log,loglimit=\"30\",comment=\"good\""
```

        - The rule is appended to the `/etc/vuurmuur/rules/rules.conf` file.
3. **Configuration File Management**:
    - **File Path**: The rules are saved in `/etc/vuurmuur/rules/rules.conf`.
    - **Access**: The file is opened in append mode (`'a'`), ensuring new rules are added without overwriting existing ones.
4. **Dynamic Form Options**:
    - Available actions, services, and zones are dynamically loaded:
        - **Services**: Files from `/etc/vuurmuur/services/`.
        - **Zones**: Directories from `/etc/vuurmuur/zones/`.
    - These options are rendered on the `rules.html` page for the user to select.

Outputs

- **On Success**:
    - A success response (`JsonResponse`) with the message "Rule added successfully" is returned if the rule is valid and saved.
- **On Error**:
    - If required form fields are missing, an error response (`JsonResponse`) with the message "Invalid form data" is returned.
- **Frontend Rendering**:
    - Displays a form to create rules with dynamic options for actions, services, and zones.
    - Handles scenarios for insufficient permissions with appropriate messages

Blocklist

The `blocklist` function is used for blocking and unblocking IPs or host groups using a backend script (`vuurmuur_script`). Below is an explanation of the function,

Functionality

The `blocklist` function enables administrators to block or unblock specific IPs or host groups dynamically from a web interface. It interacts with the backend firewall script and processes user input to apply these changes.

Core Components

1. **Input Handling**:
    - **Fields**: The function retrieves `action` (block/unblock) and `ip_host_group` from the POST request.
    - **Validation**: Ensures these fields are provided by the user before proceeding.
2. **Blocking and Unblocking**:
    - If the action is `"block"`, the function runs a system command:

```
Unset
vuurmuur_script --block <ip_host_group>
```

    - If the action is `"unblock"`, the command changes to:

```
Unset
vuurmuur_script --unblock <ip_host_group>
```

    - Commands are executed using Python's `subprocess.check_output`.
3. **Error Handling**:
    - Success messages are shown if the command runs successfully.
    - If there is an error (e.g., the script fails or the command is invalid), it captures the error message and displays it to the user.
4. **UI Rendering**:

    - The function renders a template (`blocklist.html`) to provide feedback about the operation.

Key Files/Directories Used

- **`vuurmuur_script`**:
    - Located in the system's PATH, typically `/usr/bin` or a custom directory.
    - Handles blocking and unblocking logic.
- **Template File**:
    - `blocklist.html`: Used to display feedback to the user.
- **Firewall Configuration**:
    - If `vuurmuur_script` modifies persistent rules, it likely updates configuration files, e.g., `/etc/vuurmuur/rules/blocklist.conf`.

Example Workflow

1. **User Input**:
    - The administrator selects `"block"` and enters `192.168.1.100` as the `ip_host_group`.
2. **Processing**:
    - The function executes:

```
Unset
vuurmuur_script --block 192.168.1.100
```

Services

The `services` function is used to manage services dynamically using a backend script (`vuurmuur_script`). It supports creating, deleting, and renaming services through a web interface. Below is a detailed explanation of its functionality:

Functionality

The `services` function allows administrators to perform the following operations on services:

1. **Create a new service.**
2. **Delete an existing service.**
3. **Rename an existing service.**
   It processes user input and interacts with the backend firewall script to perform these changes.

Core Components

Input Handling:

- **Fields:** The function retrieves the following from the POST request:
  - `command`: The name of the new service to be created.
  - `delete`: The name of the service to be deleted.
  - `old_name` and `new_name`: The current and new names for renaming a service.
- **Validation:** Ensures valid input is provided before executing any operation.

Service Management:

- **Create a Service:**
  - If `command` is provided, the function executes:

```
Unset
vuurmuur_script -C --service <command>
```

  - On success, a success message is displayed.
  - On failure, it captures and displays the error.
- **Delete a Service:**
  - If `delete` is provided, the function executes:

```
Unset
vuurmuur_script -D --service <delete>
```

  - On success, a success message is displayed.
  - On failure, it captures and displays the error.
- **Rename a Service:**
  - If both `old_name` and `new_name` are provided, the function executes:

```
Unset
vuurmuur_script -R --service <old_name> -S <new_name>
```

    - On success, a success message is displayed.
    - On failure, it captures and displays the error.

Error Handling:

- Displays a success message if the command executes successfully.
- Captures and displays any errors from the subprocess (e.g., invalid input or script failure).

UI Rendering:

- Renders the template `services.html` to provide user feedback, such as success or error messages.

Key Files/Directories Used

vuurmuur_script:

- **Location:** Usually found in the system's PATH (e.g., `/usr/bin` or a custom directory).
- **Purpose:** Manages service creation, deletion, and renaming logic.

Template File:

- `services.html`: Displays feedback to the administrator about the operations performed.

Firewall Configuration:

- If `vuurmuur_script` modifies persistent rules, it likely updates configuration files such as `/etc/vuurmuur/rules/services.conf`.

Example Workflow

Create a Service:

1. The administrator provides `command` as `web_server`.
2. The function executes:

```
Unset
vuurmuur_script -C --service web_server
```

3. A success message is displayed:
   `Service web_server created!`

Delete a Service:

1. The administrator provides `delete` as `old_service`.
2. The function executes:

```
Unset
vuurmuur_script -D --service old_service
```

3. A success message is displayed:
   `Service old_service deleted!`

Rename a Service:

1. The administrator provides `old_name` as `http` and `new_name` as `https`.
2. The function executes:

```
Unset
vuurmuur_script -R --service http -S https
```

3. A success message is displayed:
   `Service http renamed to https!`

Zones

The `zones` function provides administrators with the ability to manage network zones dynamically through a web interface. It interacts with a backend script (`vuurmuur_script`) to perform various actions such as creating, deleting, renaming, commenting, and updating the status of zones.

Functionality

The function processes POST requests to execute commands related to network zones. Based on the action provided by the user, it invokes the appropriate backend command using `subprocess.check_output`. It ensures proper error handling and provides feedback to the user through success or error messages.

Core Components

Input Handling:

- Retrieves the following fields from the POST request:
  - **`command:`** The name of the new zone to create.
  - **`delete:`** The name of the zone to delete.
  - **`old_name:`** The current name of the zone to be renamed.
  - **`new_name:`** The new name for the renamed zone.
  - **`comment_zone:`** The zone to which a comment is to be added.
  - **`status_zone:`** The zone whose status is to be updated.
  - **`status_message:`** The message or value to be assigned to the zone's comment or status.

Zone Operations:

1. **Creating a Zone:**
   - **Command:** `vuurmuur_script -C --zone <command>`
   - **Action:** Creates a new zone with the specified name.
   - **Success Message:** `"Zone <command> created!"`
   - **Error Handling:** Captures and displays any errors during zone creation.
2. **Deleting a Zone:**
   - **Command:** `vuurmuur_script -D --zone <delete>`
   - **Action:** Deletes the specified zone.
   - **Success Message:** `"Zone <delete> deleted!"`
   - **Error Handling:** Captures and displays errors during zone deletion.

3. **Renaming a Zone:**
    - **Command:** `vuurmuur_script -R --zone <old_name> -S <new_name>`
    - **Action:** Renames a zone from `<old_name>` to `<new_name>`.
    - **Success Message:** `"Zone <old_name> renamed to <new_name>!"`
    - **Error Handling:** Handles errors during the renaming process.
4. **Adding a Comment to a Zone:**
    - **Command:** `vuurmuur_script -M --zone <comment_zone> -V COMMENT -S '<status_message>' --overwrite`
    - **Action:** Adds or overwrites a comment in the specified zone.
    - **Success Message:** `"Comment added to zone <comment_zone>: <status_message>"`
    - **Error Handling:** Handles errors during comment addition.
5. **Updating Zone Status:**
    - **Command:** `vuurmuur_script -M --zone <status_zone> -V Active -S '<status_message>' --overwrite`
    - **Action:** Updates the status of the specified zone to "Active" with a message.
    - **Success Message:** `"Status changed for zone <status_zone> to Active: <status_message>"`
    - **Error Handling:** Handles errors during status updates.
6. **Handling Status Message:**
    - **Command:** `vuurmuur_script -P --zone <status_message>`
    - **Action:** Processes a status-related operation on the zone based on the provided message.
    - **Success Message:** `"Zone <status_message> deleted!"`
    - **Error Handling:** Captures errors during status message processing.

Error Handling

- **`subprocess.CalledProcessError:`**
    - Captures command execution errors and displays a descriptive error message to the user.
    - Includes the command output to assist in troubleshooting.

UI Rendering

- **Template File:** `zones.html`
    - Displays the result of the zone operation, including success or error messages.

Key Files/Directories Used

1. **Backend Script:** `vuurmuur_script`
    - Handles the core logic for zone management.
    - Located in the system PATH.
2. **Template File:** `zones.html`
    - Displays feedback and results of the executed actions to the administrator.

Interfaces

The `interfaces` function enables administrators to dynamically manage network interfaces through a web interface. It integrates with a backend script (`vuurmuur_script`) to execute operations such as creating, deleting, and renaming network interfaces.

Functionality

The function processes `POST` requests and executes interface-related commands using `subprocess.check_output`. It provides success or error feedback to the user based on the command's execution result.

Core Components

Input Handling:

The function retrieves the following fields from the `POST` request:

- **`command`**: The name of the new interface to create.
- **`delete`**: The name of the interface to delete.
- **`old_name`**: The current name of the interface to rename.
- **`new_name`**: The new name for the renamed interface.

Interface Operations:

1. **Creating an Interface:**
    - **Command**: `vuurmuur_script -C -i <command>`
    - **Action**: Creates a new interface with the specified name.
    - **Success Message**: `"Interface <command> created!"`

    - **Error Handling**: Displays an error message if the creation fails, including command output.
2. **Deleting an Interface:**
    - **Command**: `vuurmuur_script -D -i <delete>`
    - **Action**: Deletes the specified interface.
    - **Success Message**: `"Interface <delete> deleted!"`
    - **Error Handling**: Displays an error message if the deletion fails, including command output.
3. **Renaming an Interface:**
    - **Command**: `vuurmuur_script -R -i <old_name> -S <new_name>`
    - **Action**: Renames an interface from `<old_name>` to `<new_name>`.
    - **Success Message**: `"Interface <old_name> renamed to <new_name>!"`
    - **Error Handling**: Displays an error message if renaming fails, including command output.

Error Handling

- **`subprocess.CalledProcessError`**:
    - Captures any errors that occur during command execution.
    - Provides a detailed error message, including the failed command's output, for troubleshooting.

UI Rendering

- **Template File**: `interfaces.html`
    - Displays the result of interface operations, including success or error messages, to the user.

Key Files/Directories Used

1. **Backend Script**: `vuurmuur_script`
    - Handles the core logic for interface management.
    - Located in the system `PATH`.
2. **Template File**: `interfaces.html`
    - Displays the feedback and results of the executed operations to the administrator.

Log Function

The `log` function provides administrators with the ability to view various types of logs dynamically via a web interface. It fetches logs from predefined file paths and displays the content to the user. Access to the function is protected with authentication and page-level permissions.

Functionality

The function processes `GET` requests and retrieves the requested log file's content based on the `log_type` query parameter. It uses the `subprocess.check_output` function to execute shell commands for reading the log files and provides the output or an error message based on the execution result.

Core Components

Input Handling:

- Retrieves the **`log_type`** parameter from the `GET` request.
- If the parameter is missing, an error message is displayed.

Log Operations:

1. **Traffic Log**:
    - **Command**: `cat /var/log/vuurmuur/traffic.log`
    - **Action**: Fetches and displays the traffic log.
    - **Log Type**: `"traffic"`
2. **Connection Log**:
    - **Command**: `cat /var/log/vuurmuur/connections.log`
    - **Action**: Fetches and displays the connection log.
    - **Log Type**: `"connection"`
3. **New Connection Log**:
    - **Command**: `cat /var/log/vuurmuur/connnew.log`
    - **Action**: Fetches and displays the new connection log.
    - **Log Type**: `"connNew"`
4. **Vuurmuur Log**:
    - **Command**: `cat /var/log/vuurmuur/vuurmuur.log`
    - **Action**: Fetches and displays the Vuurmuur log.
    - **Log Type**: `"vuurmuur"`
5. **Audit Log**:
    - **Command**: `cat /var/log/vuurmuur/audit.log`
    - **Action**: Fetches and displays the audit log.

    - **Log Type**: `"audit"`
6. **Error Log**:
    - **Command**: `cat /var/log/vuurmuur/error.log`
    - **Action**: Fetches and displays the error log.
    - **Log Type**: `"error"`
7. **Debug Log**:
    - **Command**: `cat /var/log/vuurmuur/debug.log`
    - **Action**: Fetches and displays the debug log.
    - **Log Type**: `"debug"`
8. **Invalid Log Type**:
    - If an unrecognized log type is provided, an error message is displayed.

Error Handling

- **Missing Log Type**:
    - If the `log_type` parameter is not specified, an error message is displayed: `"Log type not specified"`.
- **Invalid Log Type**:
    - If the `log_type` is not recognized, an error message is displayed: `"Invalid log type"`.
- **`subprocess.CalledProcessError`**:
    - Captures errors that occur while executing the shell commands.
    - Displays a descriptive error message: `"Error executing subprocess: <error_message>"`.
- **General Exception Handling**:
    - Captures unexpected errors and provides a message: `"Unexpected error: <error_message>"`.

UI Rendering

- **Template File**: `log.html`
    - Displays the content of the requested log file or error messages.
    - Passes the following variables to the template:
        - `output`: The content of the log file (if successful).
        - `error`: The error message (if any error occurs).
        - `log_type`: The requested log type for reference.

Key Files/Directories Used

1. **Log Files**:
    - Located in `/var/log/vuurmuur/`.

    - Includes `traffic.log`, `connections.log`, `connnew.log`, `vuurmuur.log`, `audit.log`, `error.log`, and `debug.log`.
2. **Template File**: `log.html`
    - Provides the user interface for displaying the log content or error messages.

Config Function

The `config` function allows users to modify specific settings in the `config.conf` file of the Vuurmuur firewall system through a web interface. This function supports updating configuration parameters dynamically via POST requests.

Functionality

The function processes both `GET` and `POST` requests, allowing users to view and update configuration settings:

1. **`GET` Request**:
    - Renders the `config.html` template, which displays the configuration update form.
2. **`POST` Request**:
    - Updates specific parameters in the `/etc/vuurmuur/config.conf` file based on the data provided in the request.

Core Components

Input Handling:

1. **`new_interval`**:
    - Extracted from the `POST` request using `request.POST.get('new_interval')`.
    - Used to update the `DYN_INT_INTERVAL` parameter in the configuration file.
2. **`log_no_syn`**:
    - (Commented Out) Intended to update the `LOG_NO_SYN` parameter in the configuration file.

File Update Logic:

- Uses the `subprocess.run` method to execute the `sed` command for in-place modification of the configuration file (`/etc/vuurmuur/config.conf`).

- The `sed` command searches for the target parameter (e.g., `DYN_INT_INTERVAL`) and replaces its value with the new value provided by the user.

Example:

- **`new_interval` Update**:

Command:
bash
CopyEdit
```
sudo sed -i
's/DYN_INT_INTERVAL="[0-9]*"/DYN_INT_INTERVAL="<new_value>"/g'
/etc/vuurmuur/config.conf
```

    -

Response:

- **Success**:
    - Returns a `JsonResponse` with a success message, e.g., `"Interval updated successfully"`.
- **Failure**:
    - Returns a `JsonResponse` with an error message, e.g., `"Invalid data"`.

Error Handling

- **Invalid Input**:
    - If neither `new_interval` nor `log_no_syn` is provided, the function returns an error response: `{'error': 'Invalid data'}`.

Security Features

1. **Use of `sudo`**:
    - Ensures that only authorized users can modify the configuration file.
2. **Controlled Parameters**:
    - Only specific parameters (`DYN_INT_INTERVAL`) are allowed to be modified, reducing the risk of unintended changes.

UI Rendering

- **Template File**: `config.html`
    - Provides a user interface for entering new configuration values.
    - Allows users to submit changes via a form.

Key Files Used

1. **Configuration File**:
   - Located at `/etc/vuurmuur/config.conf`.
   - Stores settings such as `DYN_INT_INTERVAL` and `LOG_NO_SYN`.
2. **Template File**:
   - **`config.html`**:
     - Displays the form for configuration updates.

Limitations

1. **Uncommented Logic**:
   - The `log_no_syn` functionality is commented out and currently inactive.
2. **Validation**:
   - The function does not validate the format or range of input values, which may cause issues if invalid data is provided.

Example Usage

Update `DYN_INT_INTERVAL`:

- Submit a POST request with `new_interval` in the form.
- The interval will be updated in the `config.conf` file.

Capabilities

Purpose:

The `capabilities` function is designed to render a static or dynamic web page (`capabilities.html`) showcasing the system's capabilities.

Implementation:

python

CopyEdit

```
def capabilities(request):

    return render(request, 'capabilities.html')
```

Explanation:

- **Input Handling**:
  - Does not process any user inputs or parameters.
  - Only handles `GET` requests.
- **Output**:
  - Renders the `capabilities.html` template to the user.
- **Use Case**:
  - Displays information about the system's capabilities, such as supported features or modules.

Plugins Function

The `plugins` function enables administrators to dynamically update specific backend configuration settings for the Vuurmuur firewall via a web interface. It modifies the `config.conf` file based on user input to update parameters like services, zones, interfaces, and rules. The function is secured with authentication and page-level permissions.

Functionality

The function processes **POST** requests to update backend configurations specified by the user. It dynamically applies changes using the `sed` command to modify key-value pairs in the configuration file. If the request is invalid, it renders the web interface without applying changes.

Core Components

Input Handling

- The function retrieves the following parameters from the **POST** request:
  - `service_backend`
  - `zones_backend`
  - `interfaces_backend`
  - `rules_backend`
- If a parameter is missing, no action is taken, and the web interface is displayed.

Backend Configuration Updates

Service Backend

- **Command**:

```
Unset
sudo sed -i
's/SERVICES_BACKEND="[^"]*"/SERVICES_BACKEND="<new_value>"/g'
/etc/vuurmuur/config.conf
```

- **Action**: Updates the `SERVICES_BACKEND` key in the configuration file with the user-provided value.
- **Parameter**: `service_backend`

Zones Backend

- **Command**:

```
Unset
sudo sed -i
's/ZONES_BACKEND="[^"]*"/ZONES_BACKEND="<new_value>"/g'
/etc/vuurmuur/config.conf
```

- **Action**: Updates the `ZONES_BACKEND` key in the configuration file with the user-provided value.
- **Parameter**: `zones_backend`

Interfaces Backend

- **Command**:

```
Unset
sudo sed -i
's/INTERFACES_BACKEND="[^"]*"/INTERFACES_BACKEND="<new_value>"/g'
/etc/vuurmuur/config.conf
```

- **Action**: Updates the `INTERFACES_BACKEND` key in the configuration file with the user-provided value.
- **Parameter**: `interfaces_backend`

Rules Backend

- **Command**:

```
Unset
sudo sed -i
's/RULES_BACKEND="[^"]*"/RULES_BACKEND="<new_value>"/g'
/etc/vuurmuur/config.conf
```

- **Action**: Updates the `RULES_BACKEND` key in the configuration file with the user-provided value.
- **Parameter**: `rules_backend`

Error Handling

- **Missing Parameters**:
  - If no valid parameter is found in the **POST** request, no updates are made, and the web interface is rendered.
- **File Modification Errors**:
  - Errors during file modification using `subprocess.run` are not explicitly handled, and exceptions will propagate as system-level errors.

UI Rendering

- **Template File**: `plugins.html`
  - Provides a user interface for submitting backend configuration updates.
- **Variables Passed to Template**:
  - None explicitly passed for **GET** requests.

Key Files/Directories Used

Configuration File

- **Path**: `/etc/vuurmuur/config.conf`
- **Description**: Stores configuration settings for the Vuurmuur firewall.

Template File

- **Path**: `plugins.html`
- **Description**: Provides the user interface for backend configuration management.

Security

- **Authentication**: The `@login_required` decorator ensures that only authenticated users can access the function.
- **Page-Level Permissions**: The `@check_page_permissions('plugins')` decorator ensures that the user has the necessary permissions to modify plugin configurations.

Modules Function

The modules function allows administrators to dynamically update module-related configurations in the Vuurmuur firewall via a web interface. It modifies the `config.conf` file based on user input to update parameters such as `LOAD_MODULES` and `MODULES_WAIT_TIME`. This function ensures controlled module loading behavior and system wait time adjustments before modules are initialized.

Functionality

The function processes POST requests to update module configurations specified by the user. It applies changes dynamically using the `sed` command to modify key-value pairs in the configuration file. If the request is invalid, the function renders the web interface without making modifications.

Core Components

Input Handling

The function retrieves the following parameters from the POST request:

- **load_modules** – Defines whether to load modules or not (yes or no).
- **waittime** – Specifies the wait time before loading modules.

If both parameters are missing, no action is taken, and the web interface is displayed.

Module Configuration Updates

Load Modules

Command:

```
Unset
sudo sed -i 's/LOAD_MODULES="[^"]*"/LOAD_MODULES="<new_value>"/g'
/etc/vuurmuur/config.conf
```

Action: Updates the `LOAD_MODULES` key in the configuration file with the user-provided value.
Parameter: `load_modules`

Modules Wait Time

Command:

```
Unset
sudo sed -i
's/MODULES_WAIT_TIME="[^"]*"/MODULES_WAIT_TIME="<new_value>"/g'
/etc/vuurmuur/config.conf
```

Action: Updates the `MODULES_WAIT_TIME` key in the configuration file with the user-provided value.
Parameter: `waittime`

Error Handling

- **Missing Parameters:** If no valid parameter is found in the POST request, no updates are made, and the web interface is rendered.
- **File Modification Errors:** Errors during file modification using `subprocess.run` are not explicitly handled, and exceptions will propagate as system-level errors.

UI Rendering

- **Template File:** `modules.html`
- **Description:** Provides a user interface for submitting module-related configuration updates.
- **Variables Passed to Template:** None explicitly passed for GET requests.

Key Files/Directories Used

- **Configuration File**
    - **Path:** `/etc/vuurmuur/config.conf`
    - **Description:** Stores configuration settings for the Vuurmuur firewall.

- **Template File**
    - **Path:** `modules.html`
    - **Description:** Provides the user interface for module configuration management.

Security

- **Authentication:** The function should be secured with `@login_required` to ensure that only authenticated users can access it.
- **Access Control:** Additional permission checks may be required for role-based access.

Logging Function

The logging function enables administrators to manage logging-related configurations for the Vuurmuur firewall through a web interface. It updates the `config.conf` file based on user input, modifying parameters related to logging policies, invalid packet logging, SYN logging, and network probes.

Functionality

This function processes POST requests to update logging configurations specified by the user. It modifies key-value pairs in the configuration file using the `sed` command. If the request is invalid, the function renders the web interface without making any modifications.

Core Components

Input Handling

The function retrieves the following parameters from the POST request:

- **netfilter** – Defines the network filtering group (`NFGRP`).
- **log_policy** – Determines the logging policy (`LOG_POLICY`).
- **log_policy_limit** – Sets the logging policy limit (`LOG_POLICY_LIMIT`).
- **log_blocklist** – Specifies whether blocklist logging is enabled (`LOG_BLOCKLIST`).
- **log_invalid** – Controls logging for invalid packets (`LOG_INVALID`).
- **log_no_syn** – Manages logging of TCP packets without SYN (`LOG_NO_SYN`).
- **log_probes** – Configures logging for network probes (`LOG_PROBES`).
- **log_frag** – Enables logging for fragmented packets (`LOG_FRAG`).

If no valid parameters are provided, no updates are made, and the web interface is displayed.

Logging Configuration Updates

Netfilter Group

Command:

```
Unset
sudo sed -i 's/NFGRP="[^"]*"/NFGRP="<new_value>"/g'
/etc/vuurmuur/config.conf
```

Action: Updates the `NFGRP` key in the configuration file.
Parameter: `netfilter`

Logging Policy

Command:

```
Unset
sudo sed -i 's/LOG_POLICY="[^"]*"/LOG_POLICY="<new_value>"/g'
/etc/vuurmuur/config.conf
```

Action: Updates the `LOG_POLICY` key.
Parameter: `log_policy`

Logging Policy Limit

Command:

```
Unset
sudo sed -i
's/LOG_POLICY_LIMIT="[0-9]*"/LOG_POLICY_LIMIT="<new_value>"/g'
/etc/vuurmuur/config.conf
```

Action: Updates the `LOG_POLICY_LIMIT` key.
Parameter: `log_policy_limit`

Blocklist Logging

Command:

```
Unset
sudo sed -i
's/LOG_BLOCKLIST="[^"]*"/LOG_BLOCKLIST="<new_value>"/g'
/etc/vuurmuur/config.conf
```

Action: Updates the `LOG_BLOCKLIST` key.
Parameter: `log_blocklist`

Invalid Packet Logging

Command:

```
Unset
sudo sed -i 's/LOG_INVALID="[^"]*"/LOG_INVALID="<new_value>"/g'
/etc/vuurmuur/config.conf
```

Action: Updates the `LOG_INVALID` key.
Parameter: `log_invalid`

SYN Packet Logging

Command:

```
Unset
sudo sed -i 's/LOG_NO_SYN="[^"]*"/LOG_NO_SYN="<new_value>"/g'
/etc/vuurmuur/config.conf
```

Action: Updates the `LOG_NO_SYN` key.
Parameter: `log_no_syn`

Probe Logging

Command:

```
Unset
sudo sed -i 's/LOG_PROBES="[^"]*"/LOG_PROBES="<new_value>"/g'
/etc/vuurmuur/config.conf
```

Action: Updates the `LOG_PROBES` key.
Parameter: `log_probes`

Fragment Logging

Command:

```
Unset
sudo sed -i 's/LOG_FRAG="[^"]*"/LOG_FRAG="<new_value>"/g'
/etc/vuurmuur/config.conf
```

Action: Updates the `LOG_FRAG` key.
Parameter: `log_frag`

Error Handling

- **Missing Parameters:** If no valid parameter is found in the POST request, no updates are made, and the web interface is rendered.
- **File Modification Errors:** Errors during file modification using `subprocess.run` are not explicitly handled, and exceptions will propagate as system-level errors.

UI Rendering

- **Template File:** `logging.html`
- **Description:** Provides a user interface for submitting logging configuration updates.
- **Variables Passed to Template:** None explicitly passed for GET requests.

Security

- **Authentication:** The `@login_required` decorator ensures that only authenticated users can access the function.
- **Access Control:** The `@check_page_permissions('logging')` decorator ensures that the user has the necessary permissions to modify logging configurations.

Conntrack

The `conntrack` function allows administrators to dynamically update connection tracking settings in the Vuurmuur firewall via a web interface. It modifies the `config.conf` file based on user input to update parameters such as `DROP_INVALID` and `CONNTRACK_ACCOUNTING`. This function ensures controlled handling of invalid connections and enables or disables connection tracking accounting.

Functionality

The function processes POST requests to update connection tracking configurations specified by the user. It applies changes dynamically using the `sed` command to modify key-value pairs in the configuration file. If the request is invalid, the function renders the web interface without making modifications.

Core Components

Input Handling

The function retrieves the following parameters from the POST request:

- **drop_invalid** – Determines whether to drop invalid connections (`yes` or `no`).
- **conntrack_accounting** – Enables or disables connection tracking accounting (`yes` or `no`).

If both parameters are missing, no action is taken, and the web interface is displayed.

Connection Tracking Configuration Updates

Drop Invalid Connections

- **Command:**

```
Unset
sudo sed -i 's/DROP_INVALID="[^"]*"/DROP_INVALID="<new_value>"/g'
/etc/vuurmuur/config.conf
```

- **Action:** Updates the `DROP_INVALID` key in the configuration file with the user-provided value.
- **Parameter:** `drop_invalid`

Connection Tracking Accounting

- **Command:**

```
Unset
sudo sed -i
's/CONNTRACK_ACCOUNTING="[^"]*"/CONNTRACK_ACCOUNTING="<new_value>
"/g' /etc/vuurmuur/config.conf
```

- **Action:** Updates the `CONNTRACK_ACCOUNTING` key in the configuration file with the user-provided value.
- **Parameter:** `conntrack_accounting`

Error Handling

- **Missing Parameters:** If no valid parameter is found in the POST request, no updates are made, and the web interface is rendered.
- **File Modification Errors:** Errors during file modification using `subprocess.run` are not explicitly handled, and exceptions will propagate as system-level errors.

UI Rendering

- **Template File:** `conntrack.html`
- **Description:** Provides a user interface for submitting connection tracking configuration updates.
- **Variables Passed to Template:** None explicitly passed for GET requests.

Key Files/Directories Used

Configuration File

- **Path:** `/etc/vuurmuur/config.conf`
- **Description:** Stores configuration settings for the Vuurmuur firewall.

Template File

- **Path:** `conntrack.html`
- **Description:** Provides the user interface for connection tracking configuration management.

System protection

The `system_protection` function allows administrators to dynamically update system protection settings in the Vuurmuur firewall via a web interface. It modifies the `config.conf` file based on user input to update parameters such as `PROTECT_SYNCOOKIE` and `PROTECT_ECHOBROADCAST`. This function helps enhance system security by enabling or disabling specific protection mechanisms.

Functionality

The function processes POST requests to update system protection configurations specified by the user. It applies changes dynamically using the `sed` command to modify key-value pairs in the configuration file. If the request is invalid, the function renders the web interface without making modifications.

Core Components

Input Handling

The function retrieves the following parameters from the POST request:

- **protect_syncookie** – Enables or disables SYN cookie protection (`yes` or `no`).
- **protect_echobroadcast** – Enables or disables echo broadcast protection (`yes` or `no`).

If both parameters are missing, no action is taken, and the web interface is displayed.

System Protection Configuration Updates

SYN Cookie Protection

- **Command:**

```
Unset
sudo sed -i
's/PROTECT_SYNCOOKIE="[^"]*"/PROTECT_SYNCOOKIE="<new_value>"/g'
/etc/vuurmuur/config.conf
```

- **Action:** Updates the `PROTECT_SYNCOOKIE` key in the configuration file with the user-provided value.
- **Parameter:** `protect_syncookie`

Echo Broadcast Protection

- **Command:**

```
Unset
sudo sed -i 's/PROTECT_ECHOBROADCAST="[^"]*"/PROTECT_ECHOBROADCAST="<new_value>"/g' /etc/vuurmuur/config.conf
```

- **Action:** Updates the `PROTECT_ECHOBROADCAST` key in the configuration file with the user-provided value.
- **Parameter:** `protect_echobroadcast`

Error Handling

- **Missing Parameters:** If no valid parameter is found in the POST request, no updates are made, and the web interface is rendered.
- **File Modification Errors:** Errors during file modification using `subprocess.run` are not explicitly handled, and exceptions will propagate as system-level errors.

UI Rendering

- **Template File:** `system_protection.html`
- **Description:** Provides a user interface for submitting system protection configuration updates.
- **Variables Passed to Template:** None explicitly passed for GET requests.

Key Files/Directories Used

Configuration File

- **Path:** `/etc/vuurmuur/config.conf`
- **Description:** Stores configuration settings for the Vuurmuur firewall.

Template File

- **Path:** `system_protection.html`
- **Description:** Provides the user interface for system protection configuration management.

Security

- **Authentication:** The function should be secured with `@login_required` to ensure that only authenticated users can access it.
- **Access Control:** Additional permission checks may be required for role-based access.

System Interface Management

The `interface2` function allows administrators to dynamically update system settings related to the interface configuration in the Vuurmuur firewall via a web interface. It modifies the `config.conf` file to update specific parameters such as `DYN_INT_CHECK` and `DYN_INT_INTERVAL`, based on user input. This function helps enhance system configuration by enabling dynamic interface checks and intervals.

Functionality

The function processes `POST` requests to update system interface configurations as specified by the user. It applies changes dynamically using the `sed` command to modify key-value pairs in the configuration file. If the request does not contain any valid parameters, the function renders the web interface without making modifications.

Core Components

Input Handling

The function retrieves the following parameters from the `POST` request:

- **`dyn_int_check`** – Enables or disables dynamic interface checks (value can be a string such as "yes" or "no").
- **`dyn_int_interval`** – Defines the interval for dynamic interface checks (value can be an integer).

If both parameters are missing, no action is taken, and the web interface is rendered.

System Interface Configuration Updates

- **Dynamic Interface Check**
    - Command:
      ```
      sudo sed -i 's/DYN_INT_CHECK="[^"]*"/DYN_INT_CHECK="<new_value>"/g' /etc/vuurmuur/config.conf
      ```
    - Action: Updates the `DYN_INT_CHECK` key in the configuration file with the user-provided value.
    - Parameter: `dyn_int_check`
- **Dynamic Interface Interval**
    - Command:
      ```
      sudo sed -i 's/DYN_INT_INTERVAL="[^"]*"/DYN_INT_INTERVAL="<new_value>"/g' /etc/vuurmuur/config.conf
      ```
    - Action: Updates the `DYN_INT_INTERVAL` key in the configuration file with the user-provided value.
    - Parameter: `dyn_int_interval`

Error Handling

- **Missing Parameters**: If no valid parameter is found in the `POST` request, no updates are made, and the web interface is displayed.
- **File Modification Errors**: Errors during file modification using `subprocess.run` are not explicitly handled, and exceptions will propagate as system-level errors.

UI Rendering

- **Template File**: `interface2.html`
    - Description: Provides a user interface for submitting system interface configuration updates.
    - Variables Passed to Template: None explicitly passed for `GET` requests.

Key Files/Directories Used

- **Configuration File**
    - **Path**: `/etc/vuurmuur/config.conf`
    - **Description**: Stores configuration settings for the Vuurmuur firewall, including dynamic interface configurations.

- **Template File**
    - **Path**: `interface2.html`
    - **Description**: Provides the user interface for system interface configuration management.

Security

- **Authentication**: The function should be secured with `@login_required` to ensure that only authenticated users can access it.
- **Access Control**: Additional permission checks may be required for role-based access control.

Connection Configuration Management

The `connections` function allows administrators to dynamically update connection-related settings in the Vuurmuur firewall via a web interface. It modifies the `config.conf` file to update specific parameters related to connection logging and rate limiting for SYN and UDP packets, based on user input. This function helps manage and control network traffic by applying these updates.

Functionality

The function processes `POST` requests to update connection configurations as specified by the user. It applies changes dynamically using the `sed` command to modify key-value pairs in the configuration file. If the request does not contain any valid parameters, the function renders the web interface without making modifications.

Core Components

Input Handling

The function retrieves the following parameters from the `POST` request:

- **`log_no_syn`** – Enables or disables logging for packets without SYN flags.
- **`use_syn_limit`** – Enables or disables SYN rate limiting.
- **`syn_limit`** – Defines the SYN rate limit (e.g., number of connections per second).
- **`syn_limit_burst`** – Defines the burst size for SYN rate limiting.
- **`use_udp_limit`** – Enables or disables UDP rate limiting.
- **`udp_limit`** – Defines the UDP rate limit (e.g., number of packets per second).
- **`udp_limit_burst`** – Defines the burst size for UDP rate limiting.

If none of these parameters are provided, no action is taken, and the web interface is rendered.

Connection Configuration Updates

- **Logging for Packets Without SYN Flags**
    - Command:
      ```
      sudo sed -i 's/LOG_NO_SYN="[^"]*"/LOG_NO_SYN="<new_value>"/g' /etc/vuurmuur/config.conf
      ```
    - Action: Updates the `LOG_NO_SYN` key in the configuration file with the user-provided value.
    - Parameter: `log_no_syn`
- **SYN Rate Limiting**
    - Command 1:
      ```
      sudo sed -i 's/USE_SYN_LIMIT="[^"]*"/USE_SYN_LIMIT="<new_value>"/g' /etc/vuurmuur/config.conf
      ```
    - Command 2:
      ```
      sudo sed -i 's/SYN_LIMIT="[^"]*"/SYN_LIMIT="<new_value>"/g' /etc/vuurmuur/config.conf
      ```
    - Action: Updates the `USE_SYN_LIMIT` and `SYN_LIMIT` keys in the configuration file with the user-provided values.
    - Parameters: `use_syn_limit`, `syn_limit`
- **UDP Rate Limiting**
    - Command 1:
      ```
      sudo sed -i 's/USE_UDP_LIMIT="[^"]*"/USE_UDP_LIMIT="<new_value>"/g' /etc/vuurmuur/config.conf
      ```
    - Command 2:
      ```
      sudo sed -i 's/UDP_LIMIT="[^"]*"/UDP_LIMIT="<new_value>"/g' /etc/vuurmuur/config.conf
      ```
    - Command 3:
      ```
      sudo sed -i 's/UDP_LIMIT_BURST="[^"]*"/UDP_LIMIT_BURST="<new_value>"/g' /etc/vuurmuur/config.conf
      ```
    - Action: Updates the `USE_UDP_LIMIT`, `UDP_LIMIT`, and `UDP_LIMIT_BURST` keys in the configuration file with the user-provided values.
    - Parameters: `use_udp_limit`, `udp_limit`, `udp_limit_burst`

Error Handling

- **Missing Parameters**: If no valid parameters are found in the `POST` request, no updates are made, and the web interface is displayed.
- **File Modification Errors**: Errors during file modification using `subprocess.run` are not explicitly handled, and exceptions will propagate as system-level errors.

UI Rendering

- **Template File**: `connections.html`
    - Description: Provides a user interface for submitting connection-related configuration updates.
    - Variables Passed to Template: None explicitly passed for `GET` requests.

Key Files/Directories Used

- **Configuration File**
    - **Path**: `/etc/vuurmuur/config.conf`
    - **Description**: Stores configuration settings for the Vuurmuur firewall, including connection-related configurations.
- **Template File**
    - **Path**: `connections.html`
    - **Description**: Provides the user interface for connection configuration management.

Security

- **Authentication**: The function should be secured with `@login_required` to ensure that only authenticated users can access it.
- **Access Control**: Additional permission checks may be required for role-based access control.

SNMP Configuration Management

The `update_snmp_config` and `start_snmp_service` functions allow administrators to update the SNMP configuration and restart the SNMP service, respectively. These functions help manage SNMP settings for monitoring and managing network devices.

Functionality

- **`update_snmp_config(location, contact)`**: This function updates the SNMP configuration file (`snmpd.conf`) by modifying the `sysLocation` and `sysContact` values based on the input parameters.

- `start_snmp_service()`: This function restarts the SNMP service to apply the updated configuration.

Core Components

`update_snmp_config(location, contact)`

This function processes the provided parameters and updates the corresponding values in the SNMP configuration file.

- **Input Parameters**:
    - `location` – The new value for `sysLocation` in the SNMP configuration.
    - `contact` – The new value for `sysContact` in the SNMP configuration.
- **Steps**:
    - **Read the Configuration File**: Opens the `/etc/snmp/snmpd.conf` file and reads its contents.
    - **Update Configuration Lines**: Iterates over the lines in the file and replaces the `sysLocation` and `sysContact` lines with the provided values.
    - **Write Back to File**: After making the necessary updates, writes the modified configuration back to the `snmpd.conf` file.
- **Command**:
    - Updates `sysLocation` and `sysContact` in `/etc/snmp/snmpd.conf` with the provided values.

`start_snmp_service()`

This function restarts the SNMP service to ensure the updated configuration is applied.

- **Steps**:
    - **Run the System Command**: Uses `systemctl restart snmpd.service` to restart the SNMP service.
    - **Error Handling**: If the service fails to restart, the error message is printed.
- **Command**:
    - Executes `systemctl restart snmpd.service` to restart the SNMP service.

Error Handling

- **File Handling Errors**: If there's an issue reading or writing to the SNMP configuration file (`/etc/snmp/snmpd.conf`), Python will raise an exception (e.g., `FileNotFoundError` or `PermissionError`).
- **Service Restart Failure**: If the service fails to restart, the error message is printed to the console.

UI Rendering

These functions do not render any user interface as they are intended to be used programmatically for SNMP configuration management.

Key Files/Directories Used

- **Configuration File**
  - **Path**: `/etc/snmp/snmpd.conf`
  - **Description**: Stores SNMP configuration settings, including `sysLocation` and `sysContact`.

Security

- **File Permissions**: The `/etc/snmp/snmpd.conf` file and the `systemctl restart` command may require elevated privileges. Ensure the functions are executed with sufficient permissions (e.g., as `root` or using `sudo`).
- **Service Control**: Restarting the SNMP service may require administrative privileges on the system.

SNMP Configuration Management Endpoint

The `snmp_config` function manages the configuration of SNMP services, allowing administrators to enable SNMP, set the SNMP location, and contact values via a web interface. This function processes both `POST` and `GET` requests to configure SNMP settings.

Functionality

- **`snmp_config(request)`**: This function handles both `GET` and `POST` requests to configure SNMP settings, including enabling the SNMP service and updating its location and contact information.

Core Components

Handling `POST` Requests

- **Input Parameters**:
    1. `enable_snmp` – A flag to enable or disable the SNMP service.
    2. `location` – The new `sysLocation` value to be updated in the SNMP configuration.
    3. `contact` – The new `sysContact` value to be updated in the SNMP configuration.
- **Steps**:
    1. **Enable SNMP**: If the `enable_snmp` flag is set, the function calls `start_snmp_service()` to restart the SNMP service.
    2. **Update Configuration**: The function updates the SNMP configuration using the `update_snmp_config(location, contact)` method.
    3. **Return Response**: After updating the configuration, the function re-renders the `snmp.html` page with the updated `location` and `contact` values as context.

Handling `GET` Requests

- When the request method is `GET`, the function simply renders the `snmp.html` template, allowing the user to view or submit the SNMP configuration form.

Error Handling

- **Unsupported Method**: If the request method is neither `POST` nor `GET`, the function returns a `405 Method Not Allowed` error.

Key Files/Directories Used

- **Template File**
    - **Path**: `snmp.html`
    - **Description**: Provides a user interface for submitting SNMP configuration updates and viewing the current settings.

Security

- **Authentication and Access Control**: The function should be secured to prevent unauthorized access. This may involve ensuring that only authenticated and authorized users can configure SNMP settings.
- **File Permissions**: The function requires sufficient privileges to modify the SNMP configuration file and restart the SNMP service. Ensure that the function runs with the necessary permissions (e.g., `root` or `sudo`).

DPI Configuration - `pie` Function

Overview:

The `pie` function allows for configuring and controlling the DPI (Deep Packet Inspection) service for network interfaces. It supports enabling and disabling the DPI service while validating inputs for IP addresses and ports when the service is enabled.

Purpose:

This view handles the process of starting, stopping, and managing the DPI service, while also providing the user interface for configuring network interfaces, IP addresses, and ports.

Functionality:

1. **POST Request**:
    - When a `POST` request is made, the function performs actions based on the user's choices:
        - **Enable DPI**:
            - Validate the IP address and port.
            - If valid, starts the DPI service using a subprocess that runs a bash script to read network traffic from selected interfaces.
        - **Disable DPI**:
            - If the DPI is currently running, it sends a signal to stop the service and terminates the subprocess.
2. **GET Request**:
    - A `GET` request is used to render the form for configuring the DPI service without any changes to the system. It displays the network interfaces and relevant configuration options.

Components:

- **Global Variables**:
    - `running_process`: Tracks the subprocess running the DPI service.

    - `exit_signal_file`: A temporary file used to terminate the DPI service.
- **Form Fields**:
    - `dpiService`: Specifies whether the DPI service should be enabled or disabled.
    - `interface`: List of network interfaces selected for DPI monitoring.
    - `ip`: IP address where the data should be sent.
    - `port`: Port number for the connection.

Request Handling:

- **POST Request**:
    - If `dpiService` is enabled, the function:
        1. Validates the IP address (must be in `xxx.xxx.xxx.xxx` format).
        2. Validates the port (must be in the range 1–65535).
        3. Constructs a bash script that captures network data from the selected interfaces using `ndpiReader`.
        4. The script sends the captured data to the specified IP and port via `curl` requests.
    - If `dpiService` is disabled, it attempts to stop the running DPI service by writing an exit signal and terminating the subprocess.
- **GET Request**:
    - Displays the form with network interfaces for configuration.

Example Flow:

1. **Enabling DPI**:
    - The user selects interfaces, enters an IP address, and a port number.
    - The function validates the inputs and, if valid, starts the DPI service.
    - The system then monitors the specified interfaces and sends the data to the provided IP and port.
2. **Disabling DPI**:
    - The user chooses to disable the DPI service.
    - The system terminates the running process and sends a termination signal.

Error Handling:

- The function checks for common errors such as invalid IP format or out-of-range port numbers and provides feedback to the user.
- If the DPI service is already running, it will notify the user that it cannot be started again.

- If there is no DPI service running, the function will inform the user that it cannot be stopped.

User Interface:

- **Message Types**:
  - Success: When the DPI service is started or stopped successfully.
  - Error: When there is an invalid input or if there is an issue starting or stopping the service.
- **Heading**:
  - The page is titled as "DPI Configuration" when the user accesses the `pie.html` template.

Security Considerations:

- Ensure that the IP and port values are sanitized to prevent injection attacks.
- Make sure that the subprocess is executed in a secure environment to prevent unauthorized access.

Template (HTML - `pie.html`):

The `pie.html` page includes form inputs for the user to select interfaces, enter an IP address, and specify a port number for the DPI service.

Get Network Interfaces - `get_network_interfaces` Function

Overview:

The `get_network_interfaces` function retrieves the available network interfaces on the system by listing the contents of the `/sys/class/net` directory. This directory contains the network interfaces currently available on the system.

Purpose:

This function is used to fetch and return a list of all network interfaces (such as `eth0`, `wlan0`, etc.) that the system is aware of. It provides a way for the user interface to dynamically display available interfaces for configuration in tasks like DPI configuration.

Function:

```python
import os

def get_network_interfaces():
    # Function to fetch available network interfaces from /sys/class/net
    return os.listdir('/sys/class/net')
```

Explanation:

- **Input**:
    - This function does not take any input parameters.
- **Output**:
    - The function returns a list of strings, each representing a network interface available on the system. This is derived from the contents of the `/sys/class/net` directory, which lists all network interfaces (both physical and virtual) available on the machine.

Example:

- If the system has the following network interfaces: `eth0`, `wlan0`, and `lo`, the function will return:

```python
['eth0', 'wlan0', 'lo']
```

Usage:

- This function can be used to dynamically display available network interfaces in a web form or configuration panel, where users can select interfaces for specific tasks (e.g., configuring DPI service or network monitoring).

Example Usage:

```python
# Example usage of the function to display available interfaces
interfaces = get_network_interfaces()
print("Available interfaces:", interfaces)
```

Security Considerations:

- Ensure that only the relevant network interfaces are displayed to the user. Depending on system security, some interfaces might not be accessible or should be filtered out.

Manage Firewall Rules - `rulesfig` Function

Overview:

The `rulesfig` function allows the management of firewall rules stored in a configuration file (e.g., `/etc/vuurmuur/rules/rules.conf`). The function reads, processes, and displays the rules to users. Users can delete and reorder firewall rules through the interface.

Purpose:

This function is designed to interact with a configuration file containing firewall rules, parse the rules into a usable format, and allow users to manage those rules through a web interface. It supports two main actions:

1. **Delete rules**: Users can delete specific firewall rules.
2. **Reorder rules**: Users can reorder the firewall rules.

Function:

Python

```
import re
from django.http import JsonResponse
from django.shortcuts import render

def rulesfig(request):
    file_path = '/etc/vuurmuur/rules/rules.conf'  # Update with
your file path

    # Read the file and extract information from each rule
    with open(file_path, 'r') as file:
        file_content = file.read()

    # Define regular expressions for each keyword
    action_pattern = re.compile(r'="(\w+)', re.IGNORECASE)
    service_pattern = re.compile(r'service (\w+)', re.IGNORECASE)
    source_pattern = re.compile(r'from (\w+)', re.IGNORECASE)
    destination_pattern = re.compile(r'to (\w+)', re.IGNORECASE)
    options_pattern = re.compile(r'options (.+)$', re.IGNORECASE)

    # Split content into individual rules
    rules = file_content.split('\n')

    all_rules_info = []
    for rule in rules:
        # Extract keywords from each rule
        action_match = action_pattern.search(rule)
        service_match = service_pattern.search(rule)
        source_match = source_pattern.search(rule)
        destination_match = destination_pattern.search(rule)
        options_match = options_pattern.search(rule)

        # Prepare data for the current rule
        rule_info = {
            'action': action_match.group(1) if action_match else
'',
```

```
            'service_name': service_match.group(1) if
service_match else '',
            'source': source_match.group(1) if source_match else
'',
            'destination': destination_match.group(1) if
destination_match else '',
            'options': options_match.group(1) if options_match
else '',
            'raw': rule  # Save the raw rule for easy reordering
        }

        # Exclude empty rules
        if any(rule_info.values()):
            all_rules_info.append(rule_info)

    # Add rule numbers to the rule information
    all_rules_info_with_numbers = [(index + 1, rule_info) for
index, rule_info in enumerate(all_rules_info)]

    if request.method == 'POST':
        delete_rule_numbers = request.POST.getlist('delete_rule')
        reordered_ids = request.POST.getlist('order[]')

        # Handle deletion
        if delete_rule_numbers:
            all_rules_info_with_numbers = [
                (index + 1, rule_info)
                for index, (number, rule_info) in
enumerate(all_rules_info_with_numbers)
                if str(number) not in delete_rule_numbers
            ]
            # Write the non-deleted lines back to the file
            with open(file_path, 'w') as file:
                for _, rule_info in all_rules_info_with_numbers:
                    file.write(rule_info['raw'] + '\n')
```

```
        # Handle reordering
        if reordered_ids:
            reordered_rules = []
            for rule_id in reordered_ids:
                for number, rule_info in
all_rules_info_with_numbers:
                    if str(number) == rule_id:
                        reordered_rules.append(rule_info['raw'])
                        break

            # Write the reordered rules back to the file
            with open(file_path, 'w') as file:
                for rule in reordered_rules:
                    file.write(rule + '\n')

        return JsonResponse({'status': 'success'})

    return render(request, 'rulesfig.html', {'rules':
all_rules_info_with_numbers})
```

Explanation:

- **Input**:
    - `request`: The HTTP request object, which may contain the `POST` data for deletion and reordering of rules.
    - The function processes the `/etc/vuurmuur/rules/rules.conf` file, parsing the rules using regular expressions.
- **Output**:
    - A JSON response indicating the success or failure of the operation when using the `POST` method.
    - Renders the HTML page (`rulesfig.html`) to display the rules and allow the user to delete or reorder them.

Core Functionality:

1. **Reading Rules**:
    - The file located at `/etc/vuurmuur/rules/rules.conf` is read.
    - The file content is parsed using regular expressions to extract key information such as `action`, `service_name`, `source`, `destination`, and `options`.
2. **Parsing Rules**:
    - Each rule is split into individual components using regex patterns and stored in a list of dictionaries (`all_rules_info`).
3. **Display and Manage Rules**:
    - The rules are displayed on the web interface with rule numbers for easy deletion or reordering.
    - The user can:
        - **Delete**: Select rules to delete. These rules will be removed from the configuration file.
        - **Reorder**: Reorder rules by adjusting their order in the configuration file.
4. **Handling POST Requests**:
    - The function handles `POST` requests to:
        - Delete selected rules.
        - Reorder the rules.
    - The updated rules are written back into the configuration file after deletion and reordering.
5. **Returning a Response**:
    - After a `POST` request, a `JsonResponse` with a status of `'success'` is returned.
    - For `GET` requests, the interface to manage the rules is displayed.

Example Usage:

- **GET Request**: Displays the list of firewall rules with options to delete or reorder them.
- **POST Request**:
    - Deletes specific rules based on user input.
    - Reorders the rules as per the new order provided.

Security Considerations:

- **File Access**: Ensure that the configuration file (`/etc/vuurmuur/rules/rules.conf`) is properly secured to prevent unauthorized access or modifications.
- **User Permissions**: Limit access to this functionality to authorized users only. Use proper role-based access controls (RBAC).

Example:

A sample rule in the `rules.conf` file might look like this:

```
Unset
action="allow" service "http" from "any" to "192.168.0.1" options "log"
```

The function will extract the `action`, `service`, `source`, `destination`, and `options` from this rule and display them in a user-friendly format for the user to modify.

Terminal Function - `terminal`

Overview:

The `terminal` function is used to start a terminal session on a web interface. It checks if the `ttyd` process (a web-based terminal) is already running and starts it if it's not. The terminal is then rendered in the `terminal.html` page for user interaction.

Purpose:

This function ensures that a terminal session (via `ttyd`) is available for users through a web interface. If the terminal process is not running, it starts the process with the `vtysh` command (commonly used for network devices like routers and switches).

Function:

```python
import subprocess
from django.shortcuts import render

def terminal(request):
    # Start the ttyd process if not already running
    if not subprocess.run(['pgrep', 'ttyd'],
capture_output=True).stdout:
        subprocess.Popen(['ttyd', '-W', 'vtysh'])

    return render(request, 'terminal.html')
```

Explanation:

- **Input**:
    - `request`: The HTTP request object from the client. This object is used for interacting with the web page.
- **Output**:
    - **Web Response**: It renders a `terminal.html` page where users can interact with the terminal.
    - **Process Handling**: It checks whether the `ttyd` process (which provides a web terminal interface) is already running.

Core Functionality:

1. **Process Check**:
    - The function uses `subprocess.run(['pgrep', 'ttyd'], capture_output=True)` to check if a `ttyd` process is currently running.

`pgrep` is a command-line utility that searches for processes based on their name.
    - If the process is not found (i.e., the `stdout` is empty), the function proceeds to start the `ttyd` process.
2. **Start `ttyd`**:
    - If the `ttyd` process is not running, it is started using `subprocess.Popen(['ttyd', '-W', 'vtysh'])`. This command launches the `ttyd` web terminal with the `vtysh` shell (commonly used for managing networking devices).
3. **Render Terminal Interface**:
    - Once the terminal is ready, the function renders the `terminal.html` page where the user can interact with the terminal interface in their browser.

Security Considerations:

- **Access Control**: Ensure that only authorized users can access the terminal to prevent unauthorized control over the system.
- **Process Handling**: Always verify that the terminal process does not conflict with other applications or cause resource consumption issues when started multiple times.

Example Usage:

- **GET Request**: When a user navigates to the page associated with this view, the function checks if the web terminal is running and starts it if necessary. Then, the terminal interface is displayed to the user.

Django LDAP Configuration

Functionality

The `configure_ldap` function in Django serves three main purposes: managing LDAP configuration using Django forms and models, establishing and validating LDAP connections, and caching LDAP configuration for improved performance. It retrieves and updates LDAP settings in the database, allowing users to configure the server details via a web form. Additionally, it attempts to connect to the LDAP server using provided credentials, handling

errors such as timeouts and authentication failures. The function also utilizes Django's caching mechanism to store LDAP details dynamically, ensuring efficient access while providing real-time feedback on connection status.

Code Breakdown

1. Basic LDAP Configuration Using Django Forms and Models

This section is commented out in the provided code but serves as a reference for using Django models and forms for LDAP configuration.

- The function `configure_ldap(request)` fetches the first LDAP configuration from the database.
- If a POST request is received, it updates the configuration using a Django form.
- If the form is valid, it saves the configuration and redirects to a success page.
- Otherwise, it renders a template (`configure_ldap.html`) containing the form.

Python

```python
from django.shortcuts import render, redirect

from .forms import LDAPConfigForm

from .models import LDAPConfig


def configure_ldap(request):

    ldap_config = LDAPConfig.objects.first()

    if request.method == "POST":

        form = LDAPConfigForm(request.POST, instance=ldap_config)

        if form.is_valid():

            form.save()

            return redirect("success_page")
```

```
    else:
        form = LDAPConfigForm(instance=ldap_config)
    return render(request, "configure_ldap.html", {"form": form})
```

2. Connecting to LDAP and Handling Connection Errors

The second section of the code handles real-time LDAP connection testing.

Functionality:

- Takes user input for LDAP server IP, Bind DN, and Bind Password via a form.
- Attempts to establish an LDAP connection using the provided credentials.
- Uses `ldap.set_option(ldap.OPT_NETWORK_TIMEOUT, 5)` to limit connection wait times.
- Uses `simple_bind_s()` for authentication.
- Captures and logs errors like connection failures and timeouts.
- Stores and displays connection status messages in `connection_results`.

Python

```python
import ldap
from django.shortcuts import render
from socket import timeout


# Keep track of connection results
connection_results = {}


def configure_ldap(request):
    global connection_results
    if request.method == 'POST':
```

```
        ldap_ip = request.POST.get('ldap_ip')
        bind_dn = request.POST.get('bind_dn')
        bind_password = request.POST.get('bind_password')

        # Debug logs
        print("LDAP IP:", ldap_ip)
        print("Bind DN:", bind_dn)
        print("Bind Password:", bind_password)

        status_message = "Not Connected"

        try:
            ldap.set_option(ldap.OPT_NETWORK_TIMEOUT, 5)  # Timeout in seconds
            ldap_connection = ldap.initialize(ldap_ip)
            ldap_connection.simple_bind_s(bind_dn, bind_password)
            status_message = "Connected Successfully"
        except ldap.LDAPError:
            status_message = "Connection Failed"
        except timeout:
            status_message = "Connection Timed Out"
        except Exception as e:
```

```
            status_message = f"Unexpected Error: {str(e)}"
        finally:
            if 'ldap_connection' in locals():
                ldap_connection.unbind()

        print("Connection Status for", ldap_ip, ":",
status_message)
        connection_results[ldap_ip] = status_message

    print("Connection Results:", connection_results)
    return render(request, 'ldap.html', {'connection_results':
connection_results})
```

3. Caching LDAP Configuration

The last section of the code focuses on storing LDAP configuration dynamically in Django's cache.

Functionality:

- Accepts LDAP server details through a POST request.
- Invalidates any previously cached LDAP settings.
- Saves the new configuration in Django's cache (`cache.set()`).
- Debug logs display stored cache values.
- Returns an HTTP response confirming successful configuration.

Python

```
from django.core.cache import cache

from django.http import HttpResponse
```

```
from django.shortcuts import render


def configure_ldap(request):
    if request.method == 'POST':
        ldap_ip = request.POST.get('ldap_ip')
        bind_dn = request.POST.get('bind_dn')
        bind_password = request.POST.get('bind_password')

        # Invalidate old cache values
        cache.delete('LDAP_SERVER_URI')
        cache.delete('LDAP_BIND_DN')
        cache.delete('LDAP_BIND_PASSWORD')

        # Store new LDAP configuration in cache
        cache.set('LDAP_SERVER_URI', f'ldap://{ldap_ip}',
timeout=60)
        cache.set('LDAP_BIND_DN', bind_dn, timeout=60)
        cache.set('LDAP_BIND_PASSWORD', bind_password,
timeout=60)

        # Debugging
        print("LDAP IP received:", ldap_ip)
```

```
        print("Bind DN received:", bind_dn)

        print("Bind Password received:", bind_password)

        print("Cache for LDAP_SERVER_URI:",
cache.get('LDAP_SERVER_URI'))

        print("Cache for LDAP_BIND_DN:",
cache.get('LDAP_BIND_DN'))

        print("Cache for LDAP_BIND_PASSWORD:",
cache.get('LDAP_BIND_PASSWORD'))



        return HttpResponse("LDAP configuration saved
successfully.")

    return render(request, 'ldap.html')
```