**LI EL Code Files Explanation**

**1. Exploring Linux Namespaces**

**1. 1\_create\_and\_explore\_namespace.sh**

This Bash script demonstrates how to create and explore a Linux network namespace.

* **Creating a namespace:** sudo ip netns add mynamespace creates an isolated network namespace called mynamespace.
* **Listing namespaces:** ip netns list displays all existing network namespaces.
* **Setting up loopback:** sudo ip netns exec mynamespace ip link set lo up enables the loopback interface inside the namespace.
* **Launching a shell:** sudo ip netns exec mynamespace bash opens an interactive shell within mynamespace, allowing users to run commands (ip a, ping 127.0.0.1) in an isolated network.

This script helps in understanding how Linux network namespaces work, isolating network environments for testing or security purposes.

**2. 2\_demo\_namespace\_creation.c**

This C program uses the clone() system call to create a new network namespace and run a child process within it.

* **Namespace creation:** clone(child\_main, child\_stack + STACK\_SIZE, CLONE\_NEWNET | SIGCHLD, NULL) creates a child process with a separate network namespace.
* **Child process function (child\_main)** launches a bash shell inside the new namespace, where users can run networking commands (ip link, ip addr).
* **Parent process waits:** waitpid(pid, NULL, 0) ensures that the main process waits for the child process to complete.

**2. Demo Security Features**

By running sudo ./demo\_namespace\_creation, users can interact with an isolated network environment, reinforcing namespace concepts.

1. **1\_automate\_patch\_application.sh**
   * **Purpose**: Automatically checks for and applies system updates, ensuring that security patches and other critical updates are installed.
   * **Steps**:
     + **Step 1**: Runs sudo apt update to refresh the list of available package updates.
     + **Step 2**: Runs sudo apt upgrade -y to install available updates, including kernel patches.
     + **Completion**: Confirms successful update application.
2. **2\_policy\_management\_example.sh**
   * **Purpose**: Applies a security policy using SELinux to define and enforce access rules for web content.
   * **Steps**:
     + **Step 1**: Uses sudo semanage fcontext to set the SELinux context of /var/www/html as httpd\_sys\_content\_t (web content).
     + **Step 2**: Runs sudo restorecon -Rv /var/www/html to apply and enforce the SELinux policy.
     + **Completion**: Confirms that the policy has been applied.
   * **Error Handling**: If the SELinux rule is already defined, the script might show a ValueError, indicating that the policy is redundant.

These scripts are useful for automating security best practices on Linux systems, reducing manual effort while ensuring consistency in system updates and policy enforcement.

1. **Evaluating Security Mechanisms**

### ****1\_simulate\_privilege\_escalation\_attempt.sh****

* **Purpose**: Tests whether unauthorized access to sensitive system files (e.g., /etc/shadow) is blocked.
* **Steps**:
  1. **Step 1**: Attempts to read /etc/shadow, which contains hashed passwords.
     + Expected behavior: The operation should be denied due to file permissions and SELinux policies.
  2. **Step 2**: Confirms that the security mechanisms are working as intended.
* **Outcome**:
  1. If permissions are correctly set, the attempt is blocked (Permission denied).
  2. This validates that security policies prevent unauthorized privilege escalation.

### ****2\_stress\_test\_for\_namespaces.sh****

* **Purpose**: Tests the system’s ability to handle multiple network namespaces.
* **Steps**:
  1. **Step 1**: Creates 10 network namespaces (testns1 to testns10).
  2. **Step 2**: Lists all active namespaces to verify their creation.
  3. **Step 3**: Deletes the namespaces to clean up.
* **Outcome**:
  1. This simulates a stress test by rapidly creating and removing namespaces.
  2. If the system handles it well, it confirms that namespace isolation is functioning properly.

These scripts help assess security enforcement (privilege escalation protection) and system resilience (namespace management).

1. **Containerization Demo**

### ****1\_mini\_container.c****

* **Purpose**: Creates a simple container using Linux namespaces.
* **Key Features**:
  + Uses clone() to create a new process with **UTS, PID, and mount namespaces**.
  + Sets a custom **hostname** (mini-container).
  + Mounts the **proc filesystem** to support process-related commands.
  + Launches an interactive **bash shell** inside the container.
* **Expected Behavior**:
  + Inside the container, commands like hostname, ps aux, and ls /proc work independently of the host.
  + Running exit terminates the container.
* **Takeaway**: This is a lightweight, low-level way to create containers without Docker.

### ****2\_docker\_container\_demo.sh****

* **Purpose**: Automates the process of building, running, interacting with, and cleaning up a Docker container.
* **Key Features**:
  + Checks **Docker installation**.
  + Creates a **Dockerfile** using **Alpine Linux** as the base image.
  + Builds a **Docker image** and runs a **detached container**.
  + Executes a command inside the container to print the hostname and a **custom environment variable** (DEMO\_MSG).
  + Cleans up the **container, image, and temporary files**.
* **Expected Behavior**:
  + The script prints messages with **emoji-enhanced explanations** for clarity.
  + The Docker container echoes its hostname and a predefined message before sleeping indefinitely.
  + Cleanup removes all traces of the container after execution.
* **Takeaway**: This showcases a **high-level approach** to containerization, making it easier to package and deploy applications.