Secure Backup and Restore System

Overview

This document outlines the design decisions and assumptions made while developing a secure backup and restore system using Bash scripts. The solution modularizes functionalities into reusable functions and automates backups via cron jobs.

System Requirements

1. **Backup Functionality**:
    - Backup files and directories modified within a specified time frame (n days).
    - Encrypt backups using a user-provided encryption key.
    - Consolidate backups into a single encrypted archive.
    - Transfer the backup to a remote server for redundancy.
2. **Restore Functionality**:
    - Decrypt and extract backups.
    - Restore files to a specified directory.
3. **Automation**:
    - Automate daily backups using cron jobs.

Design Decisions

1. Modularization

The system separates the core logic into a library file (backup_restore_lib.sh), which contains:

- validate_backup_params: Validates input for the backup script.
- backup: Performs the backup process.
- validate_restore_params: Validates input for the restore script.
- restore: Handles the restore process.

2. Backup Script (backup.sh)

The script:

- Sources the library for validation and backup functions.
- Validates user input to prevent errors during execution.
- Creates encrypted backups of directories modified in the last n days.
- Consolidates individual backups into a single encrypted archive.
- Transfers the encrypted backup to a remote server using scp.

3. Restore Script (restore.sh)

The script:

- Sources the library for validation and restore functions.
- Validates user input to ensure accurate restoration.
- Decrypts and extracts backups to the specified directory.

4. Encryption

The **GPG (GNU Privacy Guard)** tool is used for:

- Encrypting tar archives during backup.
- Decrypting files during restoration.

Encryption ensures that backup data remains secure during transit and storage.

5. Automation with Cron

The backup script is scheduled to run daily at a predefined time using cron. This ensures backups are performed automatically without user intervention.

6. File Naming Convention

- File and directory names incorporate the current timestamp (YYYY-MM-DD_HH-MM-SS format) with colons and spaces replaced by underscores. This ensures unique and consistent naming.

7. Compression and Consolidation

- Individual backups are stored as tar.gz files.
- Consolidation of backups into a single archive uses tar with the append (--append) and gzip compression.

8. Error Handling

- Validations ensure all required inputs (e.g., directories, encryption key) are provided and valid. And reported to the user with clear messages

---

Assumptions

1. **System Setup**:
    - The system has required tools installed (tar, gpg, scp, gzip, sed).
    - SSH keys are configured for passwordless access to the remote server.
2. **Encryption Key**:
    - A valid encryption key is provided by the user during backup and restore.
3. **Directories**:
    - Source and backup directories are accessible and writable {Check the permissions, SELinux}.
4. **User Input**:
    - Parameters (source directory, backup directory, encryption key, and number of days) are correctly provided.
5. **Remote Server**:
    - The remote server is reachable, and the user has appropriate permissions to transfer files.

Backup Process Flow

1. Validate input parameters:
    - Check if the correct number of parameters is passed.
    - Ensure source and backup directories exist.
2. Generate a timestamp for naming directories and files.
3. Create a new directory within the backup directory based on the timestamp.
4. Loop through all subdirectories in the source directory:
    - Identify files modified within the last n days.
    - Create a tar.gz archive for each directory.
    - Encrypt the archive using the provided encryption key.
    - Delete the unencrypted tar file.
5. Consolidate all encrypted archives into a single tar.gz file:
    - Add files to the tar archive using the --append option.
    - Compress the tar archive and encrypt it.
    - Delete intermediate files.
6. Transfer the consolidated encrypted archive to a remote server.

Restore Process Flow

1. Validate input parameters:
    - Check if the correct number of parameters is passed.
    - Ensure the backup and restore directories exist.
2. Create a temporary directory in the restore directory.
3. Loop through all encrypted files in the backup directory:
    - Decrypt files using the provided decryption key.
    - Store decrypted files in the temporary directory.
4. Extract files from decrypted tar.gz archives to the restore directory.
5. Clean up temporary files.

Script Details

Library File: backup_restore_lib.sh

- Contains functions for validation, backup, and restore tasks.
- Promotes reusability across backup.sh and restore.sh.

Backup Script: backup.sh

- Sources backup_restore_lib.sh.
- Calls validate_backup_params and backup functions.

Restore Script: restore.sh

- Sources backup_restore_lib.sh.
- Calls validate_restore_params and restore functions.

Cron Configuration

To schedule the backup script:

1. Edit the crontab:

    ```
    crontab -e
    ```

2. Add the following entry to run the script daily at 2 AM:

    ```
    0 2 * * * /path/to/backup.sh /source/dir /backup/dir encryption_key 7
    ```