Universidade de Aveiro

Project of Security of Information and Organizations

Isaac David Delgado Santiago(104327)

Luís Barros Ferreira De Sousa(108583)

Nuno Matos Carvalho(97783)

Departamento de Eletrónica, Telecomunicações e Informática

Licenciatura em Engenharia de Computadores e Informática

Table of Contents

1. Introduction

This report outlines the development and security analysis of a secure document repository system, focusing on the **Access Control Verification Requirements (V4)** as defined in the OWASP ASVS (Application Security Verification Standard). Access control is a critical aspect of secure system design, ensuring that sensitive resources are only accessible to authorized users under appropriate conditions.

The project aimed to create a robust repository for storing, sharing, and managing documents securely within organizations. A key challenge addressed during implementation was the design and enforcement of access control mechanisms, including role-based permissions, Access Control Lists (ACLs), and session-based authorization.

The report is divided into sections detailing the methodologies applied, the functionalities developed, and the security verification of the implementation under the scope of ASVS V4. For each control in the V4 chapter, an assessment is provided to determine applicability, supported by justifications and evidence from the implemented system. Where applicable, code snippets and testing results are included to substantiate compliance.

1.1. Problem Overview

Organizations face increasing demands for secure and efficient systems to manage sensitive documents. Without robust access control, unauthorized access, privilege escalation, or data leakage can compromise critical information. Addressing these challenges requires implementing mechanisms to:

1. Enforce least privilege and separation of duties.
2. Manage permissions dynamically through roles and ACLs.
3. Validate all access attempts against defined policies.

This project emphasizes these principles by incorporating role-based access control (RBAC) and ACLs into a document repository system. These mechanisms ensure that users only access resources they are authorized for, and that permissions can be easily adjusted as organizational needs evolve.

2. Methodology

2.1. Project Phases

The project was developed in three phases, aligned with the specified deliveries:

1. **Delivery 1 - Core Repository Functionality (Nov 20th)**
   Focused on implementing basic operations, including secure file storage with AES-GCM encryption, metadata management, and a simple CLI for file upload and retrieval. Testing scripts were provided to validate the functionality.
2. **Delivery 2 - Authentication and Access Control (Dec 20th)**
   Introduced session-based authentication, Role-Based Access Control (RBAC), and Access Control Lists (ACLs) to enforce permissions. Enhanced the CLI to support session management, role handling, and access controls. Comprehensive tests validated compliance with project requirements.
3. **Delivery 3 - Security Analysis and Final Refinements (Jan 1st)**
   Refined the repository and conducted a detailed security analysis under OWASP ASVS V4 (Access Control). Documented the implemented features, decisions made, and results of the analysis. Prepared the system and report for defense scheduled in January.

2.2. Tools and Technologies

- **Programming Language**: Python.
- **Cryptography Libraries**: Python cryptography for encryption and hashing.
- **Version Control**: Git for collaborative development

2.3. Security Practices

- **Granular Permission Management:** Implemented RBAC and ACLs.
- **Session Security:** Ensured secure storage and validation of session tokens.
- **Access Validation:** Implemented centralized mechanisms for consistent enforcement.

3. Implementation

3.1. First Delivery

In this delivery the following commands were implemented, and are functioning as asked in https://sweet.ua.pt/jpbarraca/course/sio/project/ :

- rep_subject_credentials
- rep_decrypt_file
- rep_decrypt_file
- rep_create_org
- rep_list_orgs
- rep_create_session
- rep_get_file
- rep_list_subjects
- rep_list_docs
- rep_add_subject
- rep_suspend_subject
- rep_activate_subject
- rep_add_doc
- rep_get_doc_metadata
- rep_get_doc_file
- rep_delete_doc

Every command was created with an approach that simplifies its use with a few steps ensuring the confidentiality and integrity of the repository along the way. Many precautions were made to ensure the mentioned above like role checking in the server side (repository.py) and key management for a correct and safe use of the repository resources for example.

In addition, two commands were developed to test some interactions:

- rep_delete_doc_from_rep
- rep_compare_files

a. Document Encryption and Storage:

- Implemented AES-GCM encryption to ensure document confidentiality, ensuring a fast and reliable use of symmetric cryptography for secure data storage and transmission to prevent unauthorized access.
- Encrypted files stored securely with metadata to manage encryption keys and document details.

Key Highlights:

- Documents were encrypted using a randomly generated key and IV.
- Metadata was managed in a centralized JSON structure to associate encryption details with each document.

Snippet Example:

```
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms,
modes

def rep_decrypt_file(encrypted_file, encryption_metadata):
    with open(encryption_metadata, 'r') as f:
        metadata = json.load(f)

    key = urlsafe_b64decode(metadata['key'])
    iv = urlsafe_b64decode(metadata['iv'])

    cipher = Cipher(algorithms.AES(key), modes.CFB(iv),
backend=default_backend())
    decryptor = cipher.decryptor()
    decrypted_data = decryptor.update(encrypted_file) +
decryptor.finalize()
    print("Decrypted data:", decrypted_data.decode())
```

b. Metadata Management:

- Metadata stores important details like file handle, creator, and encryption keys for ensuring the secure and efficient management of documents.
- Example of metadata handling by the repository:

```
@app.route("/document/add", methods=["POST"])
def add_document():
    ....
    documents[document_handle] = {
        "name": document_name,
        "create_date": create_date,
        "creator": creator,
        "file_handle": file_handle,
        "acl": {},
        "deleter": None,
        "alg": "AES-GCM",
        "key": base64.urlsafe_b64encode(encryption_key).decode("utf-8"),
        "iv": base64.urlsafe_b64encode(iv).decode("utf-8"),
        "auth_tag": base64.urlsafe_b64encode(auth_tag).decode("utf-8")
    }
    save_documents()
```

c. Command-Line Interface (CLI):

- Developed Python scripts for basic interactions, such as uploading and retrieving documents for an easier use of the repository by anyone that uses it.
- Example command to add a document:

```
def rep_add_doc(session_file, document_name, file_path):
    try:
        with open(session_file, 'r') as f:
            session_data = json.load(f)
        ...
        response = requests.post(endpoint, data=data, files=files)
        if response.status_code == 201:
            print(f"Document '{document_name}' added successfully.")
    except Exception as e:
        print(f"Error: {e}")
```

3.2. Second Delivery

In this delivery the following commands were implemented, and are functioning as asked in https://sweet.ua.pt/jpbarraca/course/sio/project/ :

- rep_assume_role
- rep_drop_role
- rep_list_roles
- rep_list_role_subjects
- rep_list_subject_roles
- rep_list_role_permissions
- rep_list_permission_roles
- rep_add_role
- rep_rep_suspend_role
- rep_reactivate_role
- rep_add_permission
- rep_remove_permission
- rep_add_permission
- rep_remove_permission
- rep_acl_doc

Every command was created with an approach that simplifies its use with a few steps ensuring the confidentiality and integrity of the repository along the way. The role checking was really important on this delivery to ensure that no one besides the authorized roles could affect the repository integrity. Extra checks were also made to ensure that a person with the right role can only work within its own organizations in order to block any attempt to affect organizations where the subject does not participate, even in the possibility of him being an administrator of a certain organization.

a. Role-Based Access Control (RBAC)

- Enhanced the explanation of RBAC roles to specify their usage and permissions.
- Included a corrected and simplified example showing how roles are added dynamically.

Code Example:

```
def rep_add_role(session_file, new_role):
    try:
        with open(session_file, 'r') as f:
            session_data = json.load(f)

        payload = {"session": session_data, "role": new_role}
        response = requests.post(f"{url}/role/add", json=payload)

        if response.status_code == 201:
            print(f"Role '{new_role}' added successfully.")
        else:
            print(f"Failed to add role: {response.text}")
    except Exception as e:
        print(f"Error: {e}")
```

b. Access Control Lists (ACLs):

- The example focused on how ACLs are used for managing granular permissions for specific roles.

Code Example:

```
def rep_acl_doc(session_file, doc_name, operation, role, permission):
    try:
        with open(session_file, 'r') as f:
            session_data = json.load(f)

        payload = {
            "session": session_data,
            "document_name": doc_name,
            "operation": operation,
            "role": role,
            "permission": permission
        }
        response = requests.post(f"{url}/document/acl", json=payload)

        if response.status_code == 200:
            print(f"Permission '{permission}' updated for role '{role}' on
document '{doc_name}'.")
        else:
            print(f"Failed to update ACL: {response.text}")
    except Exception as e:
        print(f"Error: {e}")
```

c. Session Management:

- The example illustrated how secure sessions are created using a challenge-response mechanism.

Code Example:

```
def rep_create_session(organization, username, password, credentials_file,
session_file):
    try:
        with open(credentials_file, "r") as f:
            credentials = json.load(f)
        salt = base64.b64decode(credentials["salt"])
        derived_key = derive_key(password, salt)

        private_key = serialization.load_pem_private_key(
            credentials["private_key"].encode(),
            password=derived_key,
            backend=default_backend()
        )
        ...

        payload = {
            "organization": organization,
            "username": username,
            "challenge": challenge,
            "signature": signature_b64
        }

        response = requests.post(f"{url}/session/create", json=payload)

        if response.status_code == 201:
            with open(session_file, "w") as f:
                json.dump(response.json(), f, indent=4)
            print(f"Session created successfully and saved to
{session_file}.")
        else:
            print(f"Failed to create session:
{response.json().get('error', 'Unknown error')}")
    except Exception as e:
        print(f"Error: {e}")
```

4. Security Analysis(V4):

This section evaluates the implementation against the V4 requirements from the OWASP ASVS.

4.1. Control V4.1: Verify the use of unique user identities.

- **Applicability:** Yes.
- **Justification:** Each user is authenticated using a unique username and password. Sessions are tied to these identities, ensuring accountability and traceability. For instance, every session creation request requires the `username` and a secure signature.
- **Evidence:**

```
def rep_create_session(organization, username, password, credentials_file,
session_file):
    ...
    payload = {
        "organization": organization,
        "username": username,
        "challenge": challenge,
        "signature": signature_b64
    }
    ...
    response = requests.post(endpoint, json=payload)
```

4.2. Control V4.2: Verify access to sensitive operations is restricted by roles.

- **Applicability:** Yes.
- **Justification:** Role-based permissions enforce access control for sensitive operations such as document deletion. For example, only users with a specific role (e.g., "Editor" or "Admin") can modify or delete a document.
- **Evidence:**

```
def rep_acl_doc(session_file, doc_name, operation, role, permission):
    ...
    payload = {
        "session": session_data,
        "document_name": doc_name,
        "operation": operation,
        "role": role,
        "permission": permission
    }
    response = requests.post(endpoint, json=payload)
```

4.3. Control V4.3: Verify access policies are configurable.

- **Applicability:** Yes.
- **Justification:** Administrators can dynamically configure roles and ACLs using dedicated endpoints. This ensures that policies adapt to organizational needs without requiring changes to the core system.
- **Evidence:**

```
def rep_add_role(session_file, new_role):
    ...
    payload = {"session": session_data, "role": new_role}
    response = requests.post(endpoint, json=payload)
```

4.4. Control V4.4: Verify that access control decisions are enforced consistently.

- **Applicability:** Yes.
- **Justification:** Centralized mechanisms ensure uniform enforcement of access control checks. For example, session data is validated before granting access to any document.
- **Evidence:**

```
def rep_get_doc_file(session_file, document_name, output_file=None):
    ...
    with open(session_file, 'r') as f:
        session_data = json.load(f)
    ...
    params = {"session": json.dumps(session_data), "document_name":
document_name}
    response = requests.get(metadata_url, params=params)
```

4.5. Control V4.5: Verify that default access settings are secure.

- **Applicability:** Yes.
- **Justification:** Default roles and ACLs are configured with the principle of least privilege, denying access unless explicitly granted. For example, users and roles start with no permissions until these are explicitly assigned by an administrator.
- **Evidence:** System behavior ensures no permissions are granted to users or roles by default, requiring explicit configuration by an administrator.

5. Conclusion

The secure document repository successfully implemented advanced security mechanisms, including RBAC, ACLs, and session management. Compliance with ASVS V4 requirements was verified, demonstrating the system's robustness against access control vulnerabilities. Future improvements include further automation of security testing and scalability enhancements.

Each member of the group contributed equally in the elaboration of the project.

6. References

- OWASP Application Security Verification Standard (ASVS) V4
- Python Cryptography Documentation
- Flask Framework Documentation

7. Improvements for supplementary season

After revision of the project multiple problems were noticed, some flagged in the presentation, others during personal tests. Those problems posed a serious danger for the repository.

Some of the problems flagged were:

. File Duplication to delete former content

. Impersonation, some subjects could pose as a manager (using permissions that only the managers had access to inside the Organization)

. Change the ACL list of documents from another Organizations, something that shouldn’t be possible in any scenario

. Eavesdropping

. Suspended subjects could still use the repository as if active

. Suspended roles still maintained the permissions granted to the subjects

. Organizations could be used from other ones

Every problem encountered was fixed, either minor or major.

An important step of this improvement phase was enforcing the right permissions to the subjects of a specific organization, now the subject needs to have the right permissions or the ACL rights associated with the role that it has inside the Organization.

Another important fix was the suspension of subjects and/or roles. From that point on any suspended person or role would lose its permissions denying further organization permission actions like manipulating acl's, reading documents and deleting it's file handle.

Duplication or repetition of creation or deletion of permissions, acl's, documents, roles are no longer possible.

Many checks were made in order to guarantee the right roles and/or permissions for further Organization actions to prevent abusive behaviour, making abuse obsolete.

An essential step of the repository creation was overlooked due to the time available on regular season that was the session file expiration. Now after a certain period of inactivity thesession file will be deleted obliging a new session file to be created (in our case the time chosen was one hour but tests with only ten seconds were made to ensure the deletion of the session files, a message will appear on the server side warning that the session file has expired and so revoked)

Regarding the eavesdropping problem, a simple method was developed in order to protect some key information like passwords, credentials and keys from being caught by malicious eavesdroppers.

In comparison from the regular season, the repository now offers more security and integrity, allowing a more smooth transition if this project were to be used for further development and use in some kind of system.