Indian Institute of Technology, Gandhinagar

Security Guard IITGN

Computer and Network Security
PROJECT Proposal
4th sep, 2024

GROUP MEMBERS:

Karna Pardheev Sai	21110097
Rovin Singh	24290010
Sudharshan Kumar Bharadwaj	21110218
Moirangthem Suraj Singh	24290008
Ajith Boddu	21110045

Project Overview

The goal of this project is to develop a secure log system for an institute to track the state of guests and employees, including their movements within the campus. The system will consist of two main programs, logappend and logread, which will handle appending new information to the log and querying the log, respectively. Security is paramount, so the system will implement robust measures to ensure the confidentiality, integrity, and authenticity of the log data**.**

Objectives

Confidentiality: Protect the content of the log file from unauthorised access by using encryption.

Integrity: Ensure that any tampering with the log file can be detected using hashing techniques.

Authentication: Implement token-based authentication to control access to the log system.

Access Control: Restrict access to the log file and its operations to authorised users only.

Attack Resistance: Prevent log forgery, replay attacks, and unauthorised modifications

Key Components

1. OpenSSL library: Used for cryptographic operations
2. AES-256-GCM: The encryption algorithm used
3. PBKDF2: Key derivation function
4. Salt: Used in key derivation
5. Initialization Vector (IV): Used in encryption.

Key Derivation Process

1. Salt generation:

    - For new files, a random 16-byte salt is generated using RAND_bytes(salt, sizeof(salt)).
    - For existing files, the salt is read from the beginning of the file.
2. Key derivation:
    - The deriveKey() function uses PBKDF2 (Password-Based Key Derivation Function 2).
    - It takes the user's token as the password and the salt.
    - 200,000 iterations are used for key stretching.
    - SHA-256 is used as the hash function.
    - The result is a 32-byte (256-bit) key stored in the key member variable.

Encryption Process

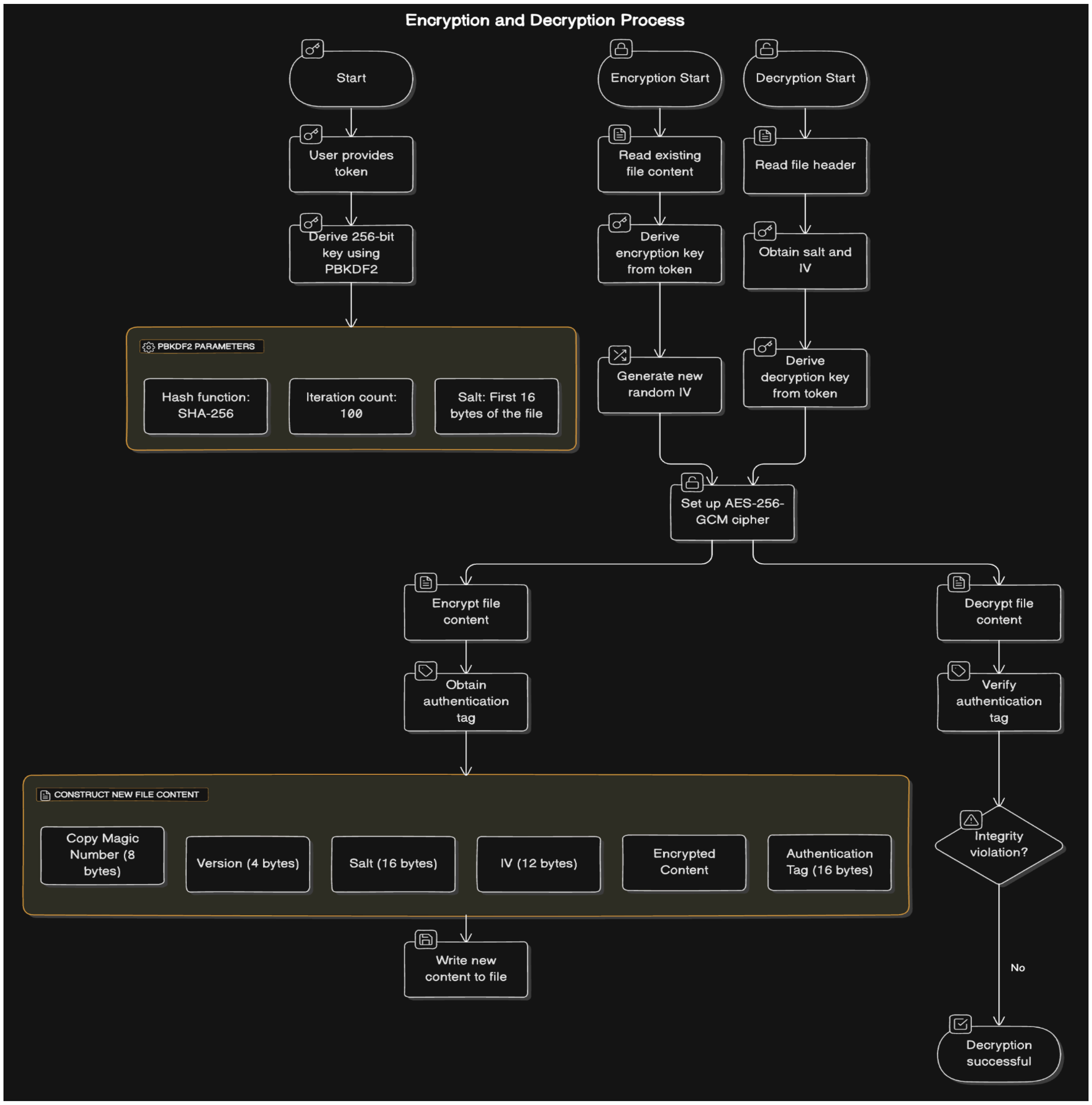
1. Prepare plaintext:
    - Concatenate all event data into a single string.
2. Generate IV:
    - A new 12-byte IV is generated for each write using RAND_bytes(iv, sizeof(iv)).
3. Set up encryption context:
    - Use EVP_aes_256_gcm() for AES-256 in GCM (Galois/Counter Mode).
4. Encrypt the data:
    - Use EVP_EncryptUpdate() and EVP_EncryptFinal_ex().
5. Get the authentication tag:
    - Retrieve the 16-byte GCM tag using EVP_CIPHER_CTX_ctrl().
6. Write to file:
    - File structure: Magic Number (8 bytes) | Version (4 bytes) | Salt (16 bytes) | IV (12 bytes) | Encrypted Content | Tag (16 bytes)

Decryption Process

1. Read file contents:
    - Read magic number, version, salt, IV, encrypted content, and authentication tag.
2. Set up decryption context:
    - Use EVP_aes_256_gcm() with the derived key and read IV.
3. Decrypt the content:
    - Use EVP_DecryptUpdate().
4. Verify the tag:
    - Set the expected tag value using EVP_CIPHER_CTX_ctrl().

    - Finalise decryption with EVP_DecryptFinal_ex(), which also verifies the tag.
5. Process decrypted data:
    - Parse the decrypted content and populate the events vector.

System Architecture:



Encryption and Decryption Process
Start
User provides token
Derive 256-bit key using PBKDF2
PBKDF2 PARAMETERS
Hash function: SHA-256
Iteration count: 100
Salt: First 16 bytes of the file
Encryption Start
Read existing file content
Derive encryption key from token
Generate new random IV
Decryption Start
Read file header
Obtain salt and IV
Derive decryption key from token
Set up AES-256-GCM cipher
Encrypt file content
Obtain authentication tag
CONSTRUCT NEW FILE CONTENT
Copy Magic Number (8 bytes)
Version (4 bytes)
Salt (16 bytes)
IV (12 bytes)
Encrypted Content
Authentication Tag (16 bytes)
Write new content to file
Decrypt file content
Verify authentication tag
Integrity violation?
No
Decryption successful

Modules:

Authentication and Decryption Module: Suraj Singh
Validate the authentication token against the one stored or expected for the log file.

1. Derive encryption key from the provided token using PBKDF2.
2. Read the file header to obtain the salt and IV.
3. Decrypt the file content using AES-256-GCM.
4. Verify the authentication tag.

Encryption Module: Saikarna
Encrypt sensitive log data before writing it to the log file (using AES).

1. Generate a new IV.
2. Encrypt the updated log content using AES-256-GCM with the new IV.
3. Construct the file content with header, encrypted data, and authentication tag.
4. Write the new content to the log file, overwriting the existing file if it exists.

Hashing Module : Ajith Boddu

We use SHA-256. SHA-256 (Secure Hash Algorithm 256-bit) is a cryptographic hash function that takes an input (or "message") and returns a fixed-length 256-bit (32-byte) hash value in hexadecimal.

- When a user or entity attempts to perform an operation (e.g., logappend or logread), they must provide a token using the -K flag.
- The provided token is hashed using SHA-256.
- The hash of the provided token is compared with the stored hash of the original token. If they match, the token is considered valid; otherwise, the operation is rejected with an error message.
- The logs are also stored using the hash functions.

Log Management Module: Rovin Singh
Create or open the log file, and append encrypted and hashed log entries.

Log Append:

1. Append Log Entries: Use logappend with -T, -K, and (-E | -G) with either -A (arrival) or -L (departure). Optionally specify -R for room-specific events.
2. Create Log: If the log file does not exist, create it.
3. Token Validation: Ensure the token matches the one previously used to create or append to the log.

4. Timestamp Validation: Ensure the new timestamp is greater than any previous timestamp in the log.
5. Batch Processing: If -B is provided, process each command line in the batch file sequentially.

Log Read:

1. Authenticate Token: Use -K to validate that the token matches the one used to create or append to the log.
2. Query Current State: Use -S to print employees and guests currently in the campus and their locations.
3. List Rooms Visited: Use -R with -E (employee name) or -G (guest name) to list all rooms visited in chronological order.
4. Total Time in Campus: Use -T with -E or -G to calculate the total time spent by an employee or guest in the campus.

Consistency Check Module: Sudharshan Kumar Bharadwaj

It is designed to verify that log entries conform to predefined rules and constraints, ensuring the accuracy and reliability of the log data that tracks the movements of guests and employees within the campus.

1. Log Append Consistency Check:
   - When a new log entry is appended using `logappend`, the module retrieves existing entries for the relevant person or room.
   - It checks that the new entry follows all predefined rules (e.g., arrivals before departures, room constraints).
   - If consistent, the entry is appended; if not, an error is generated, and the append operation is aborted.

2.Log Read Consistency Check:

- During a `logread` operation, the module checks the consistency of data being queried, ensuring it follows the correct sequence of events and that there are no logical discrepancies.
- This is crucial for accurate reporting, such as listing rooms visited or calculating the total time spent on campus.

Batch Processing Module: Rovin Singh

Read and execute commands from a batch file, handling each line individually and managing errors.

- Command Sequence Execution: The module reads and executes each command in the batch file sequentially, ensuring that all operations follow the intended order.

- Validation of Batch Commands: Each command is validated for correct syntax and parameters before execution, with invalid commands flagged and skipped without affecting subsequent commands.
- Efficient Bulk Operations: The module efficiently handles bulk operations, allowing for multiple log entries or queries to be processed in a single batch, improving overall processing speed.
- Integration with Other System Modules: The Batch Processing Module works with other system modules to ensure consistency and accuracy throughout the batch operations.

Error Handling Module: Group
Print error messages and exit with the correct status code when necessary.

Log Append:

1. Invalid Entries: Print "invalid" and exit with code 255 if:
   a. Arguments are missing, contradictory, or repeated incorrectly.
   b. Names, room IDs, or the log file path do not meet format constraints.
   c. Timestamp is not greater than the most recent timestamp.
   d. Token mismatch occurs.

2. Invalid Batch: If -B is used incorrectly (e.g., within a batch), print "invalid" and continue to the next line in the batch.

3. File Errors: Print "invalid" if log creation or access fails.

Log Read:

1. Invalid Token: Print "integrity violation" and exit with code 255 if the token does not match.
2. Invalid Command Line Arguments: Print "invalid" and exit with code 255 if:
   a. Arguments are missing, contradictory, or repeated incorrectly.
   b. Both -R and -T are used together.
3. Employee/Guest Not in Log.

4. Corrupted Log File: Print "integrity violation" and exit with code 255 if the log file appears corrupted or modified.

Security Considerations:

1. The token is never stored persistently and is cleared from memory after use.
2. All cryptographic operations use secure, standard algorithms (AES-256-GCM, PBKDF2, SHA-256).

3. Timing attacks are mitigated by using constant-time comparison for authentication tags.
4. Error messages are generic to avoid information leakage.

References:

1. https://en.wikipedia.org/wiki/PBKDF2#:~:text=PBKDF2%20applies%20a%20pseudorandom%20function,cryptographic%20key%20in%20subsequent%20operations.
2. PBKDF2 | Exploit Notes (hdks.org)