

Introduction

The University of South Alabama and its School of Computing offers undergraduate degrees in Information Technology, Information Systems, and Computer Science. Graduates of these programs are required to participate in a senior capstone experience course (CIS-497) which gives opportunity for teams of 4-5 students to work on real-world projects proposed by industry, academic, and government partners to the school. The Center for Forensics, Information Technology, and Security (CFITS) coordinates the participation of various partners to propose projects that help students put into practice the systems development lifecycle methods they have learned as part of their curriculum.

Problem Domain

Pelz[1] is a tool that provides key wrapping and unwrapping services backed by trusted hardware. By using Intel's Software Guard Extensions (SGX), Pelz maintains key encryption keys (KEKs) within a secure enclave and provides key wrapping and unwrapping services to other processes running on a machine, without ever exposing these KEKs to system memory in plaintext. Pelz provides an example of this functionality as a plugin for Apache Accumulo, an open-source, NoSQL database.

Pelz is a prototype created by the NSA, and its features are being actively developed. Pelz currently provides its wrapping services as a Linux service, accepting JSON requests across a local socket and responding in the same manner. There is currently no access control mechanism built into Pelz. Currently, the prototype allows anyone with socket access to request wrapping and unwrapping of keys with any key-encrypting key held within a "trusted execution environment" (TEE).

Objective and Goals

The contribution of this project is to develop a solution for controlling the use and access of keys within the Pelz system. The access control mechanism should be simple and generic. This will require two of distinct steps:

- ❑ Pelz will need to provide an option to associate a KEK with an 'owner' via a public key.
- ❑ A signature must be added to the JSON object signing the wrapping/unwrapping request.
- ❑ Implementing and adding these features to the existing Pelz code base is the project goal.
- ❑ Pelz is in open-source, and the students are expected to contribute to the open-source repository.

Currently known constraints:

- ❑ Must support code submission via Pelz in the open-source GitHub repository
- ❑ Must be able to be integrated into the existing code base and address dependencies
- ❑ with Intel SGX, thus it is most likely that contributions will be coded in C
- ❑ Must be maintainable and well formatted code
- ❑ Must run within a Linux environment

Functional Requirements

1. The software should allow an authorized user to send a request for key wrapping/unwrapping services.
2. The request system should be able to use the user's provided private key to create a signature that is sent as part of a JSON request to the Pelz key management system.
3. The system should validate requests, using the user's public key to verify the signature included in the JSON request. This enables authentication of the requester.
4. The system should allow key-encrypting keys within a protected space to be assigned to particular users based on their public key. This will support authorization of a JSON request (i.e., key wrapping/unwrapping services using the requested KEK are authorized for the authenticated requestor).



Non-Functional Requirements

1. All code produced and pushed to the repository should be readable. Code formatting and commenting should follow accepted software development practices. "Readability" should be a primary consideration in how code is written and structured.
2. Source control practices must be employed and must support eventual submission as a pull request on the project's GitHub repository. To the largest extent possible, commits should be incremental. Multiple, smaller and more focused "pull requests" are typically preferred over a single, more comprehensive submission.
3. All code should be tested and checked for correctness and adherence to accepted coding standards. Ideally, code submissions will include unit tests run within the test suite.
4. Code must minimize any platform-specific dependencies. For example, generic path names should be employed.

Requirements Assessment

Prior to our efforts, Pelz included no mechanism to enforce only authorized users or applications to request key wrapping and/or unwrapping services. To meet the first functional requirement, the team added two new JSON request types:

1. Signed Key Wrap Request and
2. Signed Key Unwrap Request

We implemented additional Pelz functionality needed to parse the JSON messages representing these two new request types. Finally, we added new and modified some existing unit tests for the JSON request parsing features. The test suite now passes JSON requests in both unsigned and signed request format to Pelz, where they are correctly parsed and passed to the Pelz secure enclave where they can be processed with an appropriate response being returned to the requester.

The second functional requirement was met by augmenting the test suite, using features included in the JSON library to support sending signed Pelz requests. These signed requests include elements of the signature computed and its length. Figure #1 shows the Key Wrap Request cJSON object.

The third functional requirement was only partially completed due to time constraints, the complex architecture of the project, and some unexpected iteration required in the earlier phases of the project. Completion of requirements one and two were predecessors for the task.

Early in the project, it became apparent that the complexities of implementing the fourth functional requirement within an Intel SGX enclave was a very aggressive goal. Authoring code supporting the Intel SGX architecture has a somewhat steep learning curve. As part of the development environment setup, however, the team was able to successfully install the SGX SDK.

request_type [int] – 1 for key wrap, 2 for key unwrap
key_id [string] – URI for key location; Key identifier
key_id_len [int] – Length of 'key_id' URI
enc_data [string] – Base64 encode of key to be wrapped
enc_data_len [int] – Encode length
dec_data [string] - Base64 encode of key to be wrapped
dec_data_len [int] – Encode length
user_cert[string] – User supplied certificate
user_sig [string] – Signature made from above cJSON values and private key

cJSON Object

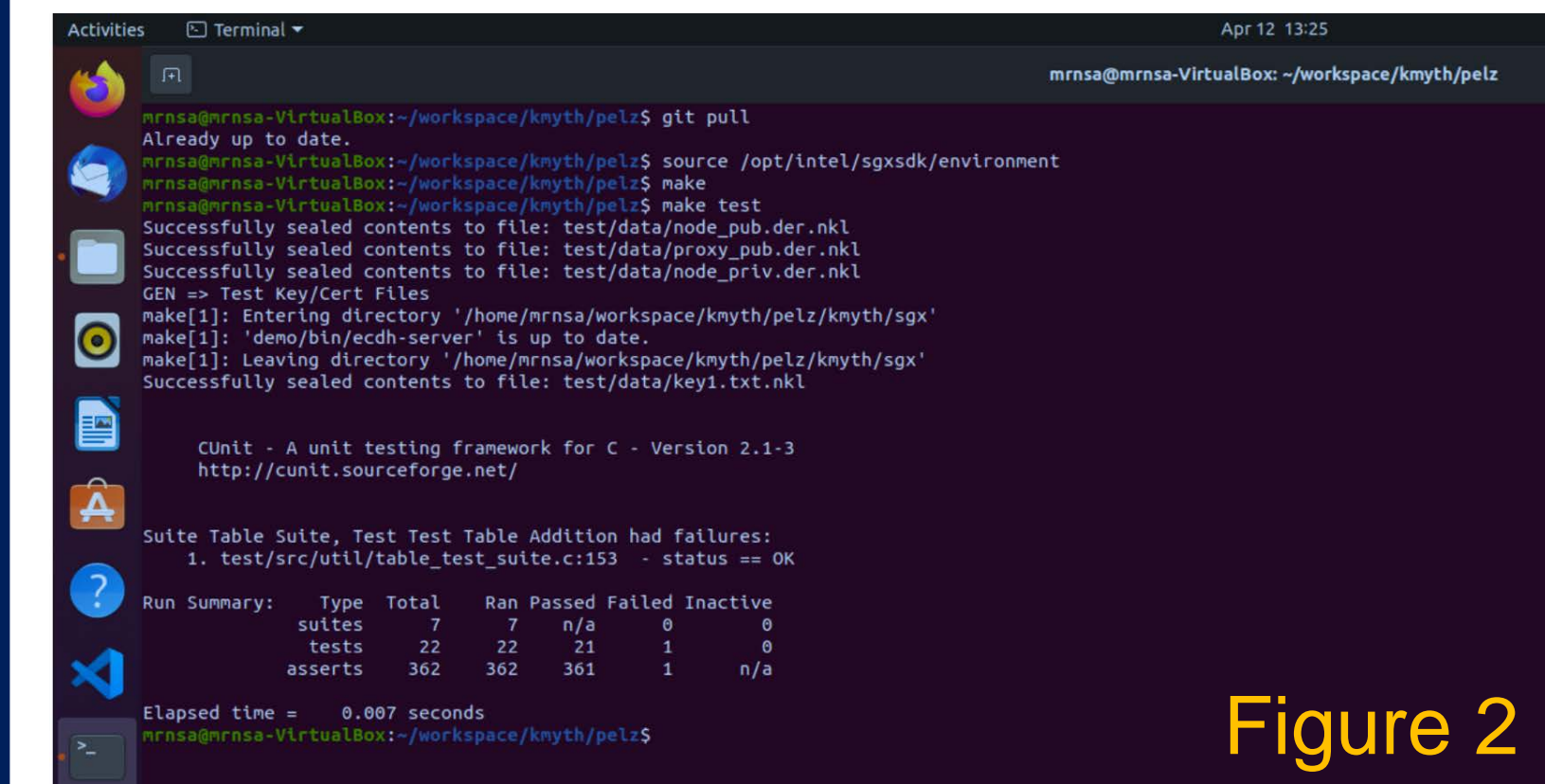
Figure 1

Test Results

Testing was done through the Pelz test suite hosted on an Ubuntu 22.04 virtual machine using Oracle VirtualBox.

The test environment was set up using a setup script provided by the client and altered by the team. The script included the installation and configuration for all Pelz prerequisites including the SGX SDK, cJSON, uriparser library, libkmp library, and TPM 2 simulator.

1. Public and private keys were passed successfully to notify Pelz to expect a signed request.
2. Request asserts were used to check values for their expected format before processing, rendering either a pass or fail.



```

mynsa@mynsa-VirtualBox:~/workspace/kmyth/pelz$ git pull
Already up to date.
mynsa@mynsa-VirtualBox:~/workspace/kmyth/pelz$ source /opt/intel/sgxdk/environment
mynsa@mynsa-VirtualBox:~/workspace/kmyth/pelz$ make
Successfully sealed contents to file: test/data/node_pub_der.nkl
Successfully sealed contents to file: test/data/proxy_pub_der.nkl
Successfully sealed contents to file: test/data/node_priv_der.nkl
GCM = test_key/Cert Files
make[1]: Entering directory '/home/mynsa/workspace/kmyth/pelz/kmyth/sgx'
make[1]: 'demo/bin/ecdh-server' is up to date.
make[1]: Leaving directory '/home/mynsa/workspace/kmyth/pelz/kmyth/sgx'
Successfully unsealed contents to file: test/data/keys.txt.nkl

Unit - A unit testing framework for C - Verion 2.1.3
http://cunit.sourceforge.net/

Suite Table Suite, Test Test Table Addition had failures:
1. test/src/utill/table_test_suite.c:153 - status == OK

Run Summary:
  Type      Total      Ran Passed Failed Inactive
  -----
  suites      7          7     0      0      0
  tests     22      22     21     1     0
  asserts   302     302    301     1     0

Elapsed time = 0.007 seconds
mynsa@mynsa-VirtualBox:~/workspace/kmyth/pelz$
  
```

Figure 2

Lessons Learned

The team utilized an AGILE methodology for development. The team focused on detailed planning and taking into consideration the project scope, activities, and risks. Project risks were mitigated by identifying potential risks early on and discussing risks often. Weekly meetings with the stakeholders were fundamental for the success of the project. The stakeholders were clear about the needs of the project which ensured the team was able to stay on track and deliver the functional requirements. However, due to time constraints some of the functional requirements fell out of the project scope.

Respect among team members and stakeholders was a crucial part of the team's success. When a team member encountered any difficulties, other team members were able to assist them. Throughout the course of the project the team maintained strong communication by staying in touch on a regular basis using messaging applications. Through frequent communication and general respect among members, the team maintained strong morale which in turn contributed to the success of the project.

Bibliography

1. Pelz. National Security Agency, 2022. <https://github.com/NationalSecurityAgency/>

QR
Code