Cibersegurança

Trabalho do módulo 3

Regime: Noturno

Grupo: G11

Alunos:

- Alexandre Silva 47192

- Diogo Cunha 47109

Índice

Introdução

Este documento tem como propósito apresentar o trabalho prático realizado para o terceiro módulo da unidade curricular de cibersegurança. Este trabalho consiste em modificar a aplicação ***ewallet*** com o objetivo de proteger a integridade e confidencialidade da informação sensível do utilizador da aplicação.

Será utilizado o ***Trusted Execution Environment*** (TEE) da Intel, **Intel SGX** [1], que permite a criação de *enclaves*, contentores isolados do ambiente onde a aplicação se encontra em execução, permitindo assim correr código sensível de forma protegida. Isto não significa que a aplicação passa a ser totalmente segura ou que esteja totalmente protegida de ataques, impede que a aplicação e as suas partes críticas consigam ser executadas em sistemas que possam ou estão comprometidos sem comprometer a aplicação.

Este documento tem a seguinte estrutura:

- Trabalho realizado: identificação da informação sensível, alterações ao código base e criação do enclave
- Funcionalidades da aplicação: apresentação das *features* da aplicação e do *flow* da sua execução
- Utilização da aplicação
- Testes realizados
- Conclusões e melhorias

1. Trabalho realizado

Como mencionado anteriormente, a aplicação *ewallet* será alterada para proteger a informação sensível da aplicação. Para tal, é necessário garantir os seguintes requisitos:

- A informação gerida e guardada pela aplicação de forma persistente deve ser protegida. Isto significa, por exemplo, que a informação não deve ser guardada em ***plain text*** no disco, *buffers* temporários devem ser limpos antes de libertar a memória, etc.
- Os métodos de encriptação e desencriptação devem ser executados no ambiente protegido (TEE). Para tal, serão usados *enclaves* para executar de forma segura estas e outras operações identificadas como críticas para a aplicação.

Com isto o trabalho a realizar pode ser divido nas seguintes etapas:

- Identificar a informação sensível a proteger;
- Identificar funções/operações vulneráveis da aplicação.
- Criar o contrato de comunicação (API – ficheiro ***enclave.edl***) entre a parte não segura e a segura da aplicação (parte segura é executa dentro do *enclave*).
- Criar as funções executadas dentro do ambiente seguro.
- Testar o código construído.

Podemos também representar o modelo da aplicação da seguinte forma:

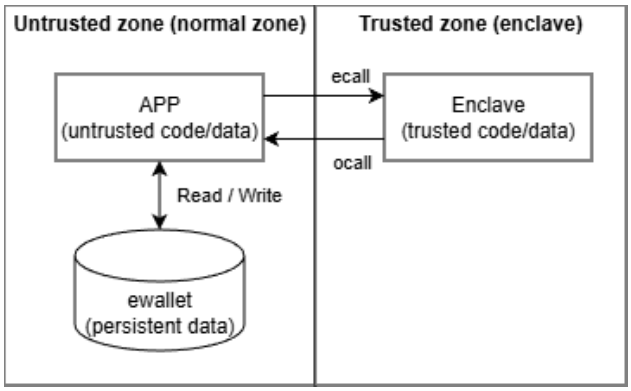

Figura 1 - Design de alto nível da aplicação

Na próxima secção, será justificado a representação da aplicação na figura 1 que apresenta, devido ao isolamento mencionado anteriormente, a comunicação entre ambientes feita através de uma API, com chamadas de funções *ecall* e *ocall*, e a escrita e leitura dos dados de forma persistente, feita pelo ambiente não seguro.

1.1. Informação e operações sensíveis

Como mencionado anteriormente, a informação sensível a proteger nesta aplicação é a informação guardada na carteira e gerida pela aplicação.

As operações sensíveis desta aplicação são:

- a geração de *passwords* com base na *master password*.
- A escrita e leitura da carteira em memória persistente. Isto inclui as seguintes ações:
  - Adicionar um item;
  - Remover um item;
  - Apresentar o conteúdo da carteira;
  - Alterar a *master password*.

1.2. Contrato de comunicação do *enclave*

Considerando a informação sensível e as operações críticas da aplicação, identificadas no capítulo anterior, foi criada a seguinte API (*enclave.edl)* entre o *enclave* e o ambiente não seguro com as seguintes funções:

- Do lado do ambiente não protegido (com prefixo *ocall*):
  - *ocall_write_to_wallet*
  - *ocall_print_wallet*
  - *ocall_print_string*
- Do lado do *enclave*, ambiente protegido (com prefixo *ecall*):
  - *ecall_generate_password*
  - *ecall_remove_item*
  - *ecall_add_item*
  - *ecall_create_wallet*
  - *ecall_change_master_password*
  - *ecall_show_wallet*

As funções *ocall* são funções chamadas pelo ambiente seguro que são executadas no ambiente normal (não seguro). Estas funções existem por diversas razões, neste caso, as funções forma criadas para:

- Escrever e ler a informação sensível, já encriptada, em memória persistente (*ocall_write_to_wallet*)
- Apresentar informação no terminal do sistema (*ocall_print_wallet* e *ocall_print_string*)

Não é possível executar estas operações dentro do *enclave* porque funções como ***printf***, ***read***, ***write***, etc; usam ***system calls*** que necessitam de alterar os **privilégios ao CPU** para usar recursos do *hardware*. Isto é um problema porque o objetivo do *enclave* é isolar uma zona de memória do resto do resto do ambiente, ou seja, memória, sistema operativo (OS), etc, pois o modelo de segurança assume que OS não é confiável, ou seja, pode estar ou ser comprometido. Ao usar uma *system call* num enclave estaríamos a expor o código e dados sensíveis porque para alterar os privilégios ao CPU é necessário interagir diretamente com o OS.

As funções *ecall* são funções chamadas pelo ambiente normal que são executadas dentro *enclave*. Estas funções existem por diversas razões, neste caso, as funções foram criadas para:

- Processo de geração de uma *password* derivada da *master_password* (*ecall_generate_password*)
- Encriptação e desencriptação da informação sensível (funções internas do *enclave*, mas usadas nos *ecalls* como *ecall_show_wallet* e *ecall_create_wallet*)
- Gestão da informação sensível (*ecall_remove_item*, *ecall_add_item*, *ecall_change_master_password*)

Estas funções são executadas dentro do *enclave* para esconder os processos de encriptação, desencriptação e geração das *passwords* derivadas, que são operações críticas do programa, e para evitar que a informação sensível, quando gerida pelo programa, seja exposta em memória do ambiente normal em *plain text*.

1.3. Processo de encriptação e desencriptação

O processo de encriptação e desencriptação da informação sensível é realizado através da funcionalidade de *sealing* do Intel SGX [1]. Esta funcionalidade utiliza o algoritmo AES-GCM de 128 bits para cifrar os dados, garantindo tanto a confidencialidade, a integridade e a autenticação da informação. A implementação foi realizada por meio de funções auxiliares internas ao enclave:

- *local_seal_data*: Esta função é responsável por cifrar dados em claro (*plaintext*). Utiliza a função *sgx_seal_data* fornecida pelo SDK do Intel SGX [1], que aplica automaticamente AES-GCM de 128 bits. Os parâmetros incluem os dados a serem cifrados, o tamanho dos dados e o buffer de destino onde os dados cifrados (*sealed data*) serão armazenados.
- *local_unseal_data*: Esta função é responsável por decifrar dados previamente selados. Utiliza a função *sgx_unseal_data* para reverter o processo de *sealing*, obtendo os dados originais em claro a partir dos dados cifrados.
- *local_get_sealed_data_size*: Função auxiliar que calcula o tamanho necessário do buffer para armazenar os dados cifrados, utilizando *sgx_calc_sealed_data_size*.

Adicionalmente, foi implementada uma camada de segurança através da verificação da *master password* em todas as operações sensíveis. Após desencriptar a carteira, o sistema compara a *master password* fornecida pelo utilizador com a armazenada na estrutura da carteira, impedindo acessos não autorizados mesmo que os dados sejam decifrados com sucesso.

Para garantir a segurança dos dados em memória, todos os buffers temporários que contêm informação sensível são limpos com memset_s antes de serem libertados, prevenindo que informação sensível permaneça acessível após o seu uso. O memset_s ao contrário do memset normal não é otimizado pelo compilador caso haja posteriormente uma chamada à função *free*.

1.4. Alterações às funções do ambiente não seguro

As funções do ambiente não seguro (*untrusted*) da aplicação foram alteradas para minimizar a exposição de informação sensível e delegar operações críticas ao enclave. As principais alterações incluem:

- Remoção de operações de encriptação/desencriptação: todas as operações criptográficas foram movidas para dentro do enclave. O código da aplicação no ambiente não seguro nunca tem acesso aos dados da carteira em texto claro.

- Gestão de dados cifrados: As funções do ambiente não seguro apenas manipulam dados já cifrados. Por exemplo, nas funções *add_item, remove_item, show_wallet e change_master_password*, a aplicação:
  1. Carrega os dados cifrados do ficheiro através da função *load_wallet.*
  2. Passa esses dados cifrados para o enclave através de *ecalls*.
  3. O enclave desencripta, processa e volta a encriptar os dados.
  4. Os dados cifrados são escritos de volta no ficheiro.

- Função *load_wallet*: Esta função foi modificada para apenas ler o *blob* de dados cifrados do ficheiro e passá-lo para o enclave, sem qualquer tentativa de interpretar ou processar o conteúdo.

- Limpeza de memória: Após cada operação, os *buffers* que contêm dados cifrados são limpos com *memset* antes de serem libertados, garantindo que mesmo dados cifrados não permaneçam em memória desnecessariamente.

- Delegação de validações: Operações sensíveis como a verificação da *master password* foram completamente delegadas ao enclave. O ambiente não seguro apenas verifica regras básicas (como tamanho de *passwords* e limites de índices) antes de chamar o enclave.

- Inicialização do enclave: No início da execução, o *enclave* é criado e um identificador global *(global_eid*) é mantido para todas as comunicações subsequentes com o enclave. No final da execução, o enclave é destruído

Estas alterações garantem que a *Trusted Computing Base* (TCB) da aplicação é minimizada, limitando-se apenas ao código executado dentro do enclave, reduzindo assim a superfície de ataque da aplicação.

2. Funcionalidades da aplicação

1. Inicialização

- O programa inicia criando o enclave SGX (*sgx_create_enclave*) e obtém um identificador global (*global_eid*) para comunicação.
- Os argumentos da linha de comandos são processados para determinar a operação a realizar.

2. Execução de Operações

Dependendo da operação solicitada, o fluxo varia:

a) Criar carteira (*create_wallet*):

- Ambiente não seguro: Verifica se a carteira já existe.
- Se existir chama a função *ecall_create_wallet.*
- Enclave: Valida a política de *password*, inicializa a estrutura *wallet_t.*
- Enclave: Cifra a carteira usando *local_seal_data*.
- Enclave: chama a função *ocall_write_to_wallet*.
- Ambiente não seguro: Escreve os dados cifrados no disco (memória persistente).

b) Adicionar item (*add_item)*:

- Ambiente não seguro: Valida os tamanhos dos campos do item, carrega os dados cifrados (*load_wallet*) e chama a função *ecall_add_item*.
- Enclave: Desencripta a carteira (*local_unseal_data*) e valida a *master password*.
- Enclave: Adiciona o item à carteira, cifra novamente (*local_seal_data*) e chama a função *ocall_write_to_wallet*.
- Ambiente não seguro: Escreve os dados atualizados e cifrados no disco.

c) Mostrar carteira *(show_wallet)*:

- Ambiente não seguro: Carrega os dados cifrados do ficheiro e chama a função *ecall_show_wallet*
- Enclave: Desencripta a carteira e valida a *master password*.
- Enclave: Chama a função *ocall_print_wallet*
- Ambiente não seguro: Imprime o conteúdo da carteira.
- Enclave: Limpa a memória sensível.

d) Remover item *(remove_item)*:

- Ambiente não seguro: Valida índice, carrega os dados cifrados e chama função *ecall_remove_item*.
- Enclave: Desencripta, verifica *master password* e remove o item reordenando o array.
- Enclave: Cifra a carteira atualizada e chama a função *ocall_write_to_wallet.*
- Ambiente não seguro: Escreve os dados atualizados e cifrados no disco.

e) Alterar a master password *(change_master_password*):

- Ambiente não seguro: Valida a política da nova *password*, carrega os dados cifrados e chama a função *ecall_change_master_password.*
- Enclave: Desencripta, verifica password antiga e atualiza password.
- Enclave: Cifra a carteira com nova password e chama a função *ocall_write_to_wallet.*
- Ambiente não seguro: Escreve os dados atualizados e cifrados no disco.

f) Gerar password (*generate_password*):

- Ambiente não seguro: chama a função *ecall_generate_password.*
- Enclave: Gera bytes aleatórios seguros (*sgx_read_rand*).
- Enclave: Converte bytes em *password* com caracteres válidos e chama a função *ocall_print_string.*

- Ambiente não seguro: Imprime a password gerada.
- Enclave: Limpa a memória da password.

3. Finalização

- Todos os *buffers* temporários são limpos *(memset/memset_s)*.
- Memória alocada é libertada.
- O enclave é destruído *(sgx_destroy_enclave)*.

Pontos-chave do fluxo de segurança:

- Os dados sensíveis apenas existem em texto claro dentro do enclave.
- Dados em trânsito entre ficheiro e enclave estão sempre cifrados.
- Todas as operações de gestão de memória garantem a limpeza dos dados sensíveis.
- A *master password* é validada pelo enclave antes de qualquer operação sensível.
- *ocalls* são utilizados apenas para operações que requerem elevação de privilégios ao CPU, como mencionado anteriormente, pois podem comprometer o enclave.

3. Utilização da aplicação

Em baixo encontram-se exemplos dos comandos das funcionalidades descritas no capítulo anterior:

- application/bin/ewallet.sh -p testpass123 -n → criar a carteira
- application/bin/ewallet.sh -p testpass123 -s → mostrar a carteira
- application/bin/ewallet.sh -g -l 12 → gerar uma nova *password*
- application/bin/ewallet.sh -p testpass123 -a -x example -y alice -z s3cr3t → adicionar um item à carteira
- application/bin/ewallet.sh -p testpass123 -r 0 → remover o item 0 da carteira
- application/bin/ewallet.sh -p testpass123 -c newpass456 → alterar a *master password*

Para correr estes comandos há duas maneiras. Uma é a normal, primeiro é necessário correr o comando "**source /opt/sgxsdk**" e depois correr os comandos anteriormente apresentados. Outra é usar o programa *bash* ***ewallet.sh*** na pasta "**application/bin**", para evitar que seja necessário correr o comando *source* cada vez que um terminal é inicializado.

4. Testes criados

Foram criados dois testes para testar a aplicação. Estes testes, que se encontram na pasta *scripts*, tem como objetivo:

- Testar o comportamento nominal da aplicação.
- Testar a capacidade da carteira.

Para correr os testes basta correr diretamente o programa *bash* na pasta *script* ou executar o comando "**make <test_name>**", para um teste específico, ou "**make all_test**" para correr todos os testes. Estes testes produzem um ficheiro de *log* na pasta *script* onde fica registado o resultado do teste.

4.1. Teste nominal (smoke_test)

Este teste nominal, como dito anteriormente, tem como objetivo testar o comportamento normal e esperado do programa.

Para tal são chamados os comandos possíveis com parâmetros que o programa esteja à espera na seguinte ordem:

- Criar a carteira
- Mostrar o conteúdo da carteira
- Gerar uma *password*
- Adicionar um item
- Mostrar o conteúdo da carteira
- Remover o item adicionado
- Mostrar o conteúdo da carteira
- Alterar a *master password*
- Verificar que a *master password* antiga não é aceite (falha esperada)

4.2. Teste à capacidade da carteira (size_test)

Este teste, como dito anteriormente, tem como objetivo testar a capacidade da carteira. O teste tem o seguinte formato:

- Criar a carteira
- Mostrar o conteúdo da carteira
- Adicionar 100 itens
- Mostrar o conteúdo da carteira
- Tentar adicionar o 101 item (falha esperada)

5. Conclusões e melhorias

Melhorias aos testes:

- Nos dois testes realizados, em vez de capturar a mensagem de erro da aplicação, poderia ser capturado o valor de retorno da aplicação, permitindo assim mais facilmente identificar um erro no programa. Consequentemente, tornaria mais fácil a criação de testes de injeção de erro, pois teríamos o código de erro associado ao erro injetado. Isto implica alterar o código para que o valor de erro seja corretamente retornado.
- Realizar mais testes.

Bibliografia

[1] Intel® Software Guard Extensions (Intel® SGX) SDK for Linux* OS - Developer Reference