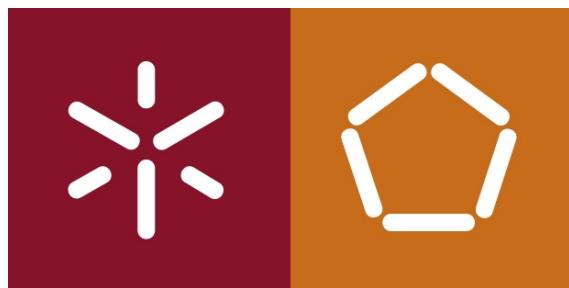


* Projeto 1 - Principles for software assurance
* Projeto 2 - Ferramentas e técnicas de code coverage



UNIVERSIDADE DO MINHO

MESTRADO EM ENGENHARIA INFORMÁTICA
CRIPTOGRAFIA E SEGURANÇA DA INFORMAÇÃO

ENGENHARIA DE SEGURANÇA

Projeto 3

André Gonçalves - A80368
Nelson Sousa - A82053
Pedro Freitas - A80975

6 de Julho de 2020

Resumo

No âmbito da Unidade Curricular de Engenharia de Segurança, enquadrada no perfil de especialização de Criptografia e Segurança da Informação, este projeto foi desenvolvido com o objetivo de abordar e explorar conceitos úteis e relevantes desta UC, sendo que se pretende desenvolver uma aplicação comando linha (CLI) que permita testar as operações do serviço SCMD (Signature CMD), fazendo reverse engineer da aplicação CMD-SOAP.

Conteúdo

1	Introdução	3
2	Arquitetura da Aplicação	4
3	Técnicas de desenvolvimento seguro de software	5
3.1	Threat modeling	5
3.2	Validação de inputs	6
3.3	Testes Automáticos	7
4	Ferramentas e indicadores de qualidade de software utilizados e/ou testes de software	8
5	Conclusão e Análise Crítica	9

1 Introdução

O presente relatório descreve a resolução do projeto final desta unidade curricular tendo como base o que foi aprendido como resultado do Projeto 1 e Projeto 2.

O objetivo deste trabalho consistiu em desenvolver na linguagem React uma aplicação comando linha (CLI) que permita testar as operações do serviço SCMD que consiste na Signature CMD (ou seja, na utilização da chave móvel digital para fazer a assinatura de um documento utilizando o web service em SOAP disponibilizado pelo estado português) fazendo reverse engineer a partir da aplicação CMD-SOAP desenvolvida na linguagem Python.

Assim sendo, neste relatório, vamos descrever de uma forma sucinta o processo de desenvolvimento da aplicação que levou até ao produto final. Inicialmente, os esforços do grupo centraram-se na estruturação da aplicação de forma a que a mesma seja funcional.

De seguida, começaram a ser testados os parâmetros relacionados com a segurança tais como técnicas de desenvolvimento seguro de software, ferramentas e indicadores de qualidade de software, entre outros.

Por fim, obtivemos como produto final a aplicação munida de grande parte dos requisitos estabelecidos e que serão justificados ao longo deste relatório.

2 Arquitetura da Aplicação

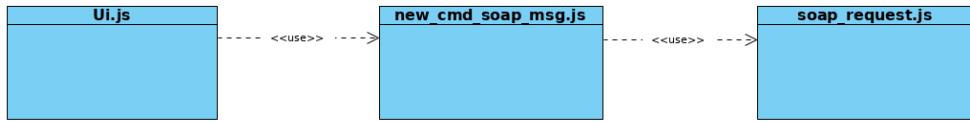


Figura 1: Arquitetura da aplicação

A arquitetura da aplicação é simples, no ficheiro *ui.js* possuímos a lógica da aplicação em *React*, aqui é fabricada as respostas aos comandos e são verificados os inputs passados pelo utilizador. Este ficheiro exporta o componente **App** que é nada mais do que a nossa aplicação comando de linha

No ficheiro *new_cmd_soap_msg.js* estão métodos usados pelo ficheiro *ui.js* para proceder aos diferentes comandos. Cada comando leva a um ou mais pedidos ao web service da chave móvel digital.

Para isso possuímos o ultimo ficheiro que tem a definição de um método responsável por fazer uma soap request, retornando o resultado. Para além disso também possui a definição dos diferentes envelopes SOAP necessários para todos os endpoints do web service CMD.

3 Técnicas de desenvolvimento seguro de software

Neste tópico, começamos por fazer a modelação de ameaças do sistema. A modelação de ameaças é uma atividade principal e prática fundamental no processo de construção de software confiável uma vez que uma análise prévia dos problemas que podem ocorrer no futuro permite solucioná-los mesmo antes da construção do sistema e se tal não for possível, permite delinear soluções de resposta imediata caso os mesmos se venham a verificar.

3.1 Threat modeling

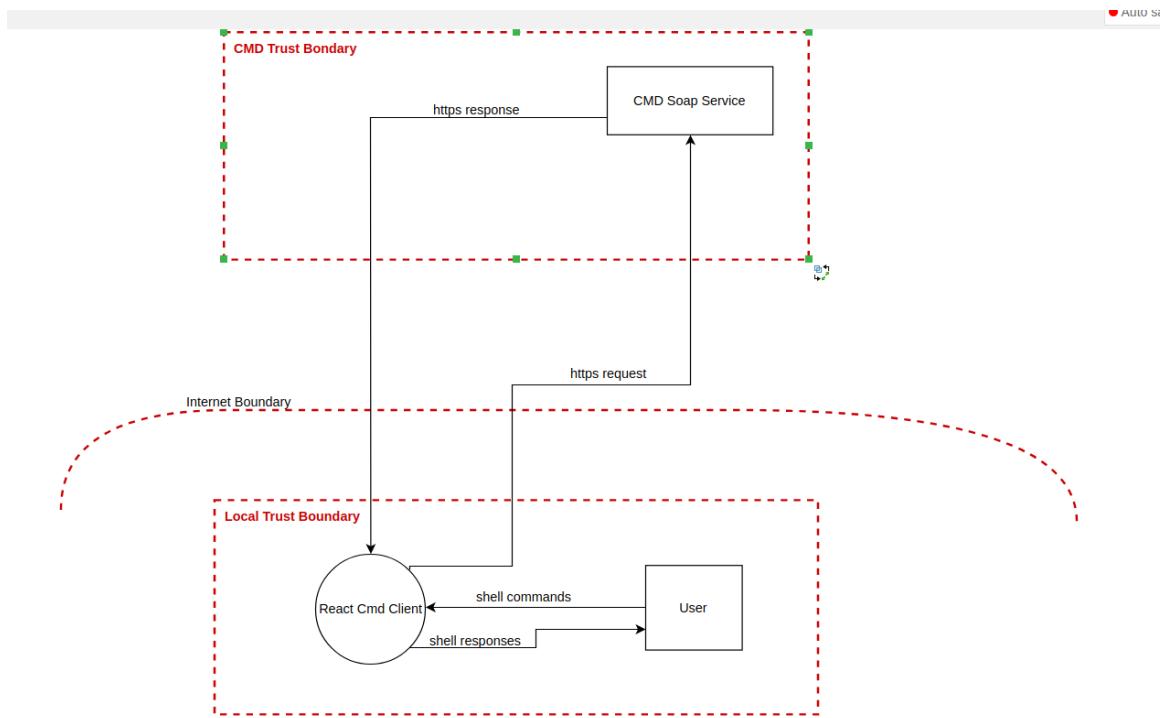


Figura 2: Modelo de Ameaças

Neste modelo de ameaças observamos que existem 3 intervenientes no sistema. O web service soap da chave movel digital, a nossa aplicação em react e o utilizador que vai inserir os comandos na shell.

As potenciais ameaças encontradas são a comunicação entre a aplicação e o web service, que caso não seja feita corretamente pode ser interceptada e alterada. E possíveis inputs malignos por parte do utilizador para tentar quebrar a aplicação ou tirar alguma vantagem da mesma.

A primeira ameaça fica solucionada usando o protocolo de comunicação *https* este protocolo de rede permite fazer pedidos a servidores remotos, sendo estes pedidos cifrados durante toda a sua trajectória.

Quanto aos inputs do utilizador, cabe-nos a nós desenvolvedores mitigar essa ameaças verificando os mesmos antes de correr as ações da aplicação.

3.2 Validação de inputs

Visto que a nossa aplicação é uma aplicação comando de linha (CLI) maior parte dos dados são recebidos como inputs do cliente. Durante este semestre foi-nos alertado do perigo que isto se pode tornar para o sistema pois um cliente poderá aceder a informação privilegiada, se tiver competências para tal!

Assim achamos de extrema importância implementar uma forma de validar os inputs recebidos do utilizador de forma a evitarmos *leaks* de informação ou até possíveis injeções de código.

Para tal definimos quatro métodos diferentes que nos irão ser muito úteis:

- **it.belongs_elem(elem, valid_elems)** - que recebe um elemento e uma lista de elementos válidos para a verificação
- **it.belongs_elems(inputs,valid_elems)** - que recebe uma lista de elementos a serem verificados e uma lista de elementos válidos para a verificação
- **it.belongs_elem(elem, valid_elems)** - que recebe um elemento e uma lista de elementos válidos para a verificação
- **validate_user_input(user_input,valid_elems)** - que recebe um input(mais precisamente o input correspondente ao utilizador) e uma lista de elementos válidos para a verificação

Através dos métodos em cima podemos verificar e validar os inputs recebidos do cliente, usando os métodos com duas abordagens diferentes.

Quando queremos verificar os dados do utilizador (o número de telemóvel) vamos invocar o método **validate_user_input(user_input,valid_elems)**, onde os **valid_elems** é um array composto por todos os dígitos de 0 a 9. Este método foi necessário visto que este input tem um formato muito específico.

A outra abordagem é quando verificam os restantes inputs e onde usamos os outros três métodos. Para esta abordagem vamos passar a estes métodos o input e um array com elementos indesejados (símbolos que podem ser potencialmente perigosos ou que não fazem parte do nosso dicionário) de forma a evitar qualquer tentativa de obtenção de dados.

validação de respostas Só API?

3.3 Testes Automáticos

Seguindo as indicações do documento *Principles for Software Assurance Assessment* da safeCode, analisado no projeto 1 desta mesma cadeira, desenvolvemos então um conjunto de testes automáticos para testar as funcionalidades da nossa aplicação. Tendo em conta o processo de maturação de desenvolvimento de software seguro do grupo e do tema do projeto, este era o passo indicado a fazer seguindo as ordens daquele documento.

Como a nossa aplicação é em *React* usamos a biblioteca *Jest* para integrar estes testas, em baixo podem se encontrar um pequeno exemplo de 2 testes automáticos. Sendo que os restantes encontram-se no código do projeto.

```
test("Test_GetCertificate_erro", () => {
  const componentApp = renderer
    .create(<App operation="GetCertificate" user="" />);
  let tree = componentApp.toJSON();
  expect(tree).toMatchSnapshot();
});

test("Test_CCMovelSign_erro", () => {
  const componentApp = renderer
    .create(<App operation="CCMovelSign" />);
  let tree = componentApp.toJSON();
  expect(tree).toMatchSnapshot();
});
```

Com este método está a ser criado um snapshot do output produzido em react da primeira vez que é executado o teste. Este snapshot pode ser editado caso algo não esteja como esperado.

Após a primeira execução dos testes, e a criação do snapshot, por cada execução dos testes automáticos vão ser comparados os snapshots produzidos com aqueles guardados e o teste tem sucesso caso os snapshots sejam iguais e não tem sucesso caso contrário.

Como é feito o teste?

* Como é feito o programa?
* Que libs/pacotes é necessário instalar?

O `readme.md` existe na diretória do projeto, leva à instalação de um programa que "não funciona".

4 ~~Ferramentas e indicadores de qualidade de software utilizados e/ou testes de software~~

Neste capítulo, a nossa primeira escolha foi o Code Coverage, uma vez que foi o indicador de qualidade de software analisado pelo nosso grupo no projeto anterior. Este permitiu-nos ter uma melhor visão sobre o código que desenvolvemos e que aspetos tinham de ser melhorados. Tentamos também criar o maior número de testes coverage de forma a cobrir todo o código-fonte:



Figura 3: Exemplo do coverage

5 Conclusão e Análise Crítica

Após a realização de mais um projeto podemos finalmente operar sobre os temas abordados ao longo do semestre e reter ensinamentos muito importantes e práticos para o mundo empresarial uma vez que a chave móvel digital é um instrumento bastante útil.

Este projeto ajudou-nos assim a perceber algumas técnicas e ferramentas utilizados e pudemos adquirir conhecimentos bastante práticos e valiosos.

Como lado menos positivo temos a apontar o facto de não termos conseguido explorar as funcionalidades do CMD completamente, pois não possuímos um leitor de cartões para ativar a conta. Contudo sabemos que os nossos pedidos ao web service estão a ser bem feitos uma vez que é retornado um código html 200, indicador de sucesso, e como mensagem é retornado utilizador não ativo, ou utilizador sem certificados.

Damos assim concluído o nosso trabalho prático, saindo do mesmo felizes e confiantes que os objetivos foram cumpridos com sucesso.

CMD pode ser
obtido no painel
de um leitor