Universidade do Minho
Escola de Engenharia

Mestrado Engenharia Informática

PROJETO 3

CMD-SOAP

GRUPO 13

Bruno Rodrigues pg41066

Carlos Alves pg41840

CONTEÚDO

INTRODUÇÃO

Este trabalho prático sugerido na Unidade Curricular – Engenharia de Segurança teve como objetivo desenvolver numa linguagem de programação atribuída aleatoriamente a cada grupo, no nosso caso foi atribuída a linguagem de programação D, uma aplicação Interface de Linhas de Comando que permita testar as operações do serviço SCMD. Para isto foi nos fornecido uma aplicação CMD-SOAP desenvolvida em Python que serviria para fazer reverse engineer da mesma. Este serviço SCMD consiste essencialmente na assinatura CMD ou chave móvel digital para realizar assinaturas de documentos, fazendo uso do serviço dum sistema facultado pelo Estado Português.[1]

De modo a demonstrar o que foi aprendido durante o semestre, especificaremos as técnicas de desenvolvimento seguro de software que utilizamos, e ainda usaremos técnicas e ferramentas de qualidade de software de modo a verificar os resultados atribuídos pelas mesmas, as ferramentas a usar serão as do projeto anterior.

Como já referido foi-nos atribuído a linguagem de programação D, que é uma linguagem de uso geral, baseada principalmente em C++ sendo que esta consegue ser obter performance iguais ou superiores ao C++, além que o código ser mais curto. Porém como é uma linguagem que nunca tivemos contacto, este trabalho tornou-se desafiante, principalmente na busca das bibliotecas necessárias para desenvolver esta aplicação – nomeadamente um cliente SOAP – e na busca de exemplos visto que não é muito utilizada.

CMD-SOAP

CONTEXTUALIZAÇÃO

Como requisito mínimo para começar o desenvolvimento deste projeto achamos por bem rever os nossos projetos anteriores e ainda os guiões das aulas. Deste modo, e como o tema do nosso primeiro projeto abordou o tema de desenvolvimento de software seguro usando a framework SSDF (Secure Software Development Framework), decidimos adaptar algumas das práticas aí descritas para nos guiarem e ajudarem no desenvolvimento seguro deste pequeno programa.

Primeiramente foi estudado o programa fornecido, analisando a forma como está estruturado, o seu funcionamento, entre outros aspetos. Depois de compreender o programa em questão, decidimos então colocar em prática algumas das tarefas que proferimos no primeiro trabalho. Começamos por preparar o grupo, realizando uma pequena reunião inicial na qual discutimos e definimos os requerimentos de segurança que iriamos implementar e ter em conta no desenvolvimento deste projeto. Visto que a linguagem de programação que nos foi atribuída era nos desconhecida, foi necessário que realizássemos alguns exercícios básicos, de modo a ambientar-nos á linguagem. Ainda neste momento, foi realizada uma buscar pelas bibliotecas que supostamente iriamos utilizar, mas foi reparado desde cedo que algumas das bibliotecas não existiam nesta linguagem, portanto foi necessário fazer “reverse engineering” de algumas, por exemplo a de SOAP -Java- que nos foi fornecida pelo docente. Logo nesta fase inicial resumiu-se principalmente na contextualização da linguagem, na investigação das bibliotecas e na preparação do grupo a nível de requisitos de segurança e por fim na distribuição de tarefas/funções que cada membro iria desempenhar.

Em relação ao protocolo usado como já referido foi o SOAP (Simple Object Access Protocol) que é um protocolo para troca de informações estruturadas numa plataforma descentralizada e distribuída. Baseia-se na Linguagem de Marcação Extensível (XML) para seu formato de mensagem, e normalmente baseia-se noutros protocolos da camada de aplicação, mais notavelmente em chamada de procedimento remoto (RPC) e Protocolo de transferência de hipertexto (HTTP), para negociação e transmissão de mensagens.

Também foi necessário rever o OpenSSL utilizado em várias componentes deste curso, que resume-se como sendo uma implementação dos protocolos TLS e SSL, sendo que esta implementação ainda nos fornece uma biblioteca genérica criptográfica, e visto que estão disponíveis *wrappers* que permitem o uso desta biblioteca em diferentes linguagens, tornando-

se mais fácil de incorporar nos projetos. Além disso o OpenSSL tem como objetivo ser utilizado para gerar certificados de autenticação de servers.

Portanto, o programa facultado pode ser caracterizado pelo conjunto de operações de SCMD/Signature CMD, que terá como finalidade assinar um documento ou ficheiro escolhido pelo utilizador.

REVERSE ENGINEERING DOS SCRIPTS FORNECIDOS

Nesta fase, começamos por reunir todas as bibliotecas necessárias e as bibliotecas que não existiam ou não foram encontradas, foram "traduzidas" para a linguagem D. As bibliotecas e métodos utilizados foram:

- **OpenSSL (x509,pem,asn1,evp,rsa)** – Trata da parte dos certificados, da validação entre outros elementos.
- **std.stdio** - Funções de E / S padrão que estendem o core.stdc.stdio
- **Std.digest.sha** - Calcula os hashes SHA1 e SHA2 de dados arbitrários. Os hashes SHA são quantidades de 20 a 64 bytes (dependendo do algoritmo SHA) que são como uma soma de verificação ou CRC, mas são mais robustas.
- **Std.net.curl** - Funcionalidade do cliente de rede, conforme fornecido pela libcurl.
- **Std.getopt** - Processamento de opções de linha de comando.
- **Std.base64** - Suporte para codificação e decodificação Base64.
- **Std.process** - Funções para iniciar e interagir com outros processos e para trabalhar com o ambiente de execução do processo atual.
- **Std.Array** - Funções e tipos que manipulam matrizes internas e matrizes associativas.
- **Std.utf** - Codifique e decodifique cadeias UTF-8, UTF-16 e UTF-32.

MÉTODOS – SCMD

Uma vez que já possuímos todas as bibliotecas necessárias, foram de seguida desenvolvidos as classes e métodos.

GETCERT_CMD

Aqui foi desenvolvido o método **getCertificate** que recebe como entrada a applicationID ou identificador da aplicação e o UserID ou identificação do utilizador, faz request Soap para o url

designado – neste caso ao serviço de autenticação da AMA. Após realizar parseResponse devolve os certificados referentes aos argumentos inseridos.

```
public string[] getCertificate(string ApplicationId, string UserId)
{
    string encodestring = Base64.encode(ApplicationId.representation);
    //string encodestring = Base64.encode("b826359c-06f8-425e-8ec3-50a97a418916".representation);

    //string soapreq = sendSOAPrequest(encodestring, "+351 935423080");
    string soapreq;

    soapreq = sendSOAPrequest(encodestring, UserId);

    string[]  certs = parseResponse(soapreq);
    return certs;
}
```

CCMOVELSIGN

Aqui foi desenvolvido o método **ccMovelSign** que recebe como entrada o identificador da aplicação, a identificação do utilizador, o nome do documento inserido como também a sua hash e ainda o pin do utilizador. E devolve um array ou vetor com os valores do processo.

```
public string ccMovelSign(string applicationId,string docName, string docHash, string pin, string userId)
{
    string encodestring = Base64.encode(applicationId.representation);
    string response = sendSOAPrequest(encodestring,docName, docHash,pin, userId);
    string processID = parseResponse(response);
    return processID;
}
```

VALIDATEOTP

Aqui foi desenvolvido o método **ValidateOtp** que recebe o código recebido pelo SMS enviado para o número do utilizador, o identificador do Processo e da aplicação. E devolve a assinatura, o código e por último a mensagem.

```
public string[] ValidateOtp(string otp,string processId, string applicationId){
    string encodestring = Base64.encode(applicationId.representation);

    string response = sendSoapRequest(otp,processId,encodestring);
    string[] results = parseResponse(response);
    return  results;
}
```

TÉCNICAS DESENVOLVIMENTO SEGURO

Depois de desenvolvido a base e toda a estrutura do programa tentamos implementar algumas das técnicas que aprendemos durante este semestre.

Portanto, de modo a proteger o nosso código de eventuais problemas de segurança ou de adulteração, sendo que este fator não seja muito relevante destacar neste programa, mas foi armazenado todo o código num repositório online, com base no princípio de menor privilégio, assim, apenas os elementos do grupo teriam acesso e capacidade de adulteração. Ainda podíamos assinar ou utilizar *hash* criptográficas para atestar a integridade do mesmo.

Depois foi levado em conta certos elementos:

- Os requisitos de segurança e a informação de risco que foi acordada no início do projeto;
- Verificar o estado do software de terceiros estava em conformidade, principalmente algumas bibliotecas que foram descontinuadas nesta linguagem.
- O desenvolvimento de Código seguro, seguindo algumas das práticas ensinadas durante este semestre.
- Testar o Programa com diversos inputs entre outros aspetos, de modo a encontrar alguma falha.
- Rever o Código feito.

VALIDAÇÃO DE INPUTS

Mas principalmente foi tido em atenção os inputs fornecidos pelo utilizador, limitando-o apenas a inputs corretos, como por exemplo: O pin inserido pelo utilizador apenas poderá ter 4 dígitos, ou o UserID terá de ser obrigatoriamente um número de telemóvel. Caso tal não se verifique é então interrompido o processo em que o utilizador pretende realizar, esta validação de inputs não foi completamente implementada.

```
string getPin(string str_Pin){
    if(str_Pin.length == 4){
        writeln("Pin Valido!");
        string PIN = str_Pin;
        return PIN;
    }
    else{return msgError();}
}
```

LIMPAR BUFFERS – X509 E NO OTP

```
if(otpresult[1] != "200")
{
    writeln("Erro " ~otpresult[1] ~ ". " ~  otpresult[2]);
    X509_free(certuser);
    X509_free(certca);
    X509_free(certroot);
    exit(0);
}
```

VERIFICAÇÃO DE OUTPUTS

Além disto também foram verificados alguns processos que eventualmente falhavam a meio, e era necessário ocorrer exceções ou interromper o fluxo do programa, usando *try catch*. Também foi tida em atenção os outputs dos métodos, como o tipo da função entre outros aspetos.

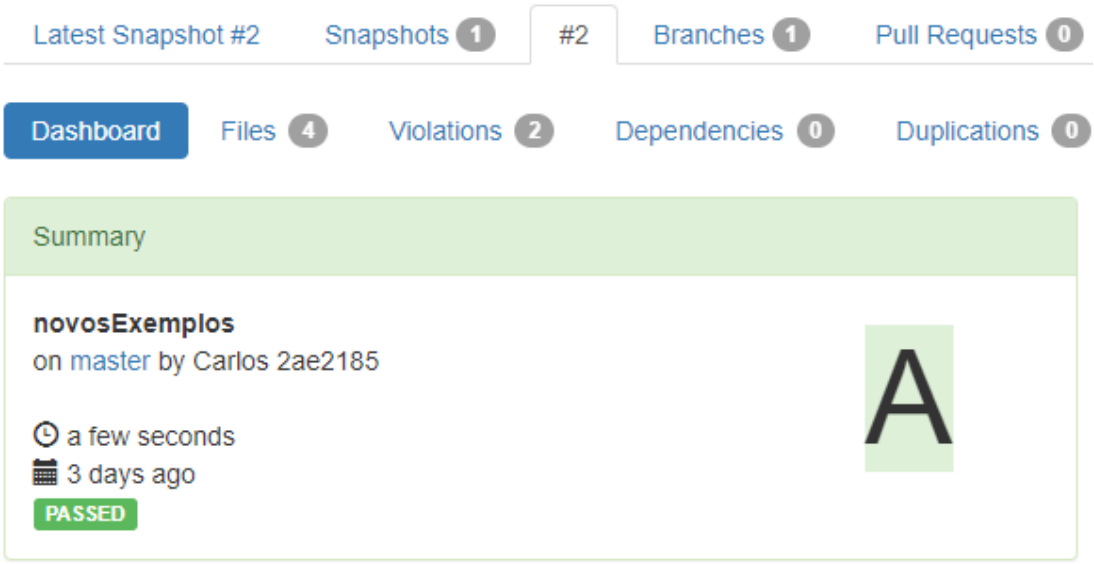
CERTIFICADOS

Algo que foi notado na geração dos certificados, foi o fato de por vezes estes virem divididos, recebendo metades de certificados, como não conseguimos descobrir a razão para tal acontecimento, fomos obrigados a fazer o seguinte:

```
//Exceçao no caso do certificado
if (tempcerts.length < 2)
{
    writeln("Impossível obter certificado");
    exit(0);
}
```

FERRAMENTAS E INDICADORES DE QUALIDADE DE CÓDIGO

Para verificar e ter uma noção de qual o estado do nosso código, recorremos a algumas ferramentas que já tínhamos utilizado no trabalho prático anterior, mas visto que a linguagem de programação D não possui muito suporte (pelo que podemos concluir no desenrolar deste projeto) portanto não foi possível encontrar praticamente nenhuma ferramenta de análise que nos analisasse o código e nos desse um relatório completo de eventuais erros ou qualidade de código. Dito isto, ainda realizamos alguns scans com algumas ferramentas que possuem suporte para linguagens semelhantes a esta, como C ou C++, mas foi verificado em todos os *reports* fornecidos pelo Kritika.io que estes não ofereciam com precisão os diversos elementos, sendo que os *reports* resultaram todos numa pontuação A. Contudo, como o Kritika.io não suporta esta linguagem estes relatórios foram descartados, pois apenas tinha como base a sintaxe de C e C++. Foi ainda utilizado uma ferramenta que já vinha incorporado nas extensões da linguagem – **DScanner** – mas este não nos oferecia grande informação, apenas servia como um auxiliar no compilador. Abaixo é possível verificar um dos relatórios resultantes do Kritika.io.



MANUAL DE UTILIZAÇÃO

1º - $ dub build --compiler=ldc2

$ ./engsegd -h

```
cmdsa@DESKTOP-6PV81R9:/mnt/c/Users/carlo/Desktop/ENGSEGDiff$ ./engsegd -h
As opções de cada operação podem ser visualizadas através da
 execução de ./engsegd <oper> -h, em que <oper> é uma das seguintes operações:

    'GetCertificate' ou 'gc'
        testa o comando SOAP GetCertificate do SCMD
    'CCMovelSign' ou 'ms'
        testa o comando SOAP CCMovelSign do SCMD
    'CCMovelMultipleSign' ou 'mms'
        testa o comando SOAP CCMovelMultipleSign do SCMD
    'ValidateOtp' ou 'otp'
        testa o comando SOAP ValidateOtp do SCMD
    'TestAll' ou 'test'
        testa automaticamente a sequência de comandos GetCertificate, CCMovelSign e ValidateOt
 verificando no final a assinatura, baseado na assinatura recebida, na hash gerada e na chave
```

Menu de ajuda – Exibe todos os métodos disponíveis

$ ./engsegd <oper> -h

```
cmdsa@DESKTOP-6PV81R9:/mnt/c/Users/carlo/Desktop/ENGSEGDiff$ ./engsegd otp -h
testa o comando SOAP ValidateOtp do SCMD
Exemplo de utilização:
./engsegd otp '+351 999999999' notas.txt 1111
```

Menu de ajuda – Exibe exemplo de utilização de cada método

$ ./engsegd gc “+xxx xxxxxxxxx”

```
cmdsa@DESKTOP-6PV81R9:/mnt/c/Users/carlo/Desktop/ENGSEGDiff$ ./engsegd gc "+351 935423080"
-----BEGIN CERTIFICATE-----
MIIJhjCCB26gAwIBAgIIZP3bhOiSnu0wDQYJKoZIhvcNAQELBQAwgekxCzAJBgNVBAYTAlBUMUIw
QAYDVQQKDDl1BTUEgLSBBR8OKTkNJQSBQQOVJBIEEgTU9ERVJOSVpBw4fDg08gOURNSU5JU1RSQVRJ
```

Obter Certificados

$ ./engsegd gc “+xxx xxxxxxxxx” file xxxx

```
cmdsa@DESKTOP-6PV81R9:/mnt/c/Users/carlo/Desktop/ENGSEGDiff$ ./engsegd ms "+351 935423080" notas.txt 1111
ProcessID devolvido pela operação CCMovelSign: 24862db5-64c9-4b0e-84f8-85ddc64f2175
```

Pedido de assinatura

$ ./engsegd otp “+xxx xxxxxxxxx” file xxxx

```
cmdsa@DESKTOP-6PV81R9:/mnt/c/Users/carlo/Desktop/ENGSEGDiff$ ./engsegd otp "+351 93542308
ProcessID devolvido pela operação CCMovelSign: 8e133d42-f160-44c9-8ecb-67a95b4c648e
```

Validação do código OTP

$ ./engsegd test “+xxx xxxxxxxxx” file xxxx

```
10% ... A contactar servidor SOAP CMD para operação GetCertificate
20% ... Certificado emitido para BRUNO MIGUEL GOMES RODRIGUES pela Entidade de Certificação EC de Chave Móvel Digital de Assinatura Dig
 00001 na hierarquia do Cartão de Cidadão 004
30% ... Leitura do ficheiro notas.txt
40% ... Geração de hash do ficheiro notas.txt
 50% ... Hash gerada (em base64): KPMO9Sio+XlyQ8J4RY8HjTYhLKMQg1dB/jg+TWaYR4c=
60% ... A contactar servidor SOAP CMD para operação CCMovelSign
70% ... ProcessID devolvido pela operação CCMovelSign: d62a2fac-d26b-4121-b3ed-24a6369a8997
80% ... A iniciar operação ValidateOtp
Introduza o OTP recebido no seu dispositivo:
861675
90% ... A contactar servidor SOAP CMD para operação ValidateOtp
100% ... Assinatura (em base 64) devolvida pela operação ValidateOtp: p19Pyowzl0qqO09xldXXFhdzvTOYtbUmQvUZH3EgCyRO9Gir8AGHUbw42x4UNkDuR
pjMmEHE88UXHwkqZgEVRILBSj1uf15MSEZSPyi6FF3Bf+k4BEKSnc1SaPOTFCZ7FoZVfmrGYlGECEtFMjLmvh7R8NtJ1pBOSJVaFDHReveFp3D4LCe+pN4+E5T2Oz5lj5AJhrCO
8HKB2NkfVlZRDCtIt13kKMA3D1r9Hg5Kq3Y7JaAuwA9dGOUzKvgKs45pNmpCqz9BO7jQ==
110% ... A validar assinatura ...
```

Testar todas as Operações

CONCLUSÃO

Com este trabalho podemos verificar que a transcrição ou tradução de um programa numa certa linguagem para outra, pode não ser tarefa fácil. Ainda que tenhamos conseguido implementar praticamente todos os métodos, apenas não conseguimos corrigir os problemas no método de verificação. Porém, num panorama geral, consideramos que boa parte dos objetivos pretendidos com este projeto foram alcançados, frisando o facto que nunca tivemos qualquer contacto com a linguagem D, ainda que esta possua diversas vantagens referente a C++, mas devido à pouca documentação, a falta de bibliotecas e exemplos, tornou-se mais "desafiante" realizar o *reverse engineering* deste programa se comparado com outras linguagens. Além disto, pudemos colocar algumas práticas de desenvolvimento de software seguro que fomos aprendendo durante o respetivo semestre.

1 BIBLIOGRAFIA

(s.d.). Obtido de Kritika: kritika.io

cr0hn. (27 de Junho de 2019). Obtido de https://github.com/cr0hn/vulnerable-node

dlang. (s.d.). Obtido de https://dlang.org/phobos/index.html

Gov PT. (s.d.). Obtido de https://www.autenticacao.gov.pt/web/guest/a-chave-movel-digital

HackerPilot. (s.d.). *dlang-community*. Obtido de D-Scanner: https://github.com/dlang-community/D-Scanner

OpenSSL. (s.d.). Obtido de https://www.openssl.org/

Wikipedia. (s.d.). Obtido de https://pt.wikipedia.org/wiki/SOAP