

*Projeto 1 - software security takes a champion*  
*Projeto 2 - ferramentas e técnicas de cyclomatic complexity*



**Universidade do Minho**  
Escola de Engenharia

Universidade do Minho  
Mestrado Integrado em Engenharia Informática  
Engenharia de Segurança  
Projeto 3 - CMD-SOAP - Scala

Diogo Duarte  
(pg41843@alunos.uminho.pt)  
Mateus da Silva Ferreira  
(pg37159@alunos.uminho.pt)  
Ricardo Cunha Dias  
(pg39295@alunos.uminho.pt)

6 de Julho de 2020

# Conteúdo

|          |   |          |
|----------|---|----------|
| <b>1</b> | <b>Introdução</b>   | <b>2</b> |
| <b>2</b> | <b>Estratégias Adotadas</b>   | <b>2</b> |
| 2.1      | Técnicas de desenvolvimento seguro de software utilizadas . . . . .             | 2        |
| 2.2      | Testes, ferramentas e indicadores de qualidade de software utilizados . . . . . | 4        |
| 2.2.1    | Lizard . . . . .  | 4        |
| 2.2.2    | IntelliJ - Analyzing Tools . . . . .  | 5        |
| 2.3      | Dificuldades Encontradas . . . . .  | 5        |
| 2.3.1    | SonarLint . . . . .   | 5        |
| <b>3</b> | <b>Resultados obtidos</b>   | <b>6</b> |
| 3.1      | GetCertificate . . . . .  | 6        |
| 3.2      | CCMoveISign . . . . .   | 6        |
| 3.3      | ValidateOtp . . . . .   | 7        |
| 3.4      | TestAll . . . . .   | 7        |
| <b>4</b> | <b>Conclusão</b>  | <b>8</b> |

# 1. Introdução

Este relatório visa descrever o trabalho efetuado, bem como, os resultados obtidos do Projeto 3 da Unidade Curricular Engenharia de Software.

Para a realização deste projeto foi necessário basear nos no que foi aprendido nos dois trabalhos anteriores e, deste modo, foi desenvolvida uma aplicação comando linha (CLI). Esta foi elaborada em Scala e usada para testar as operações do serviço SCMD, utilizando como base a aplicação CMD-SOAP, escrita em Python e algumas classes em Java disponibilizadas nos materiais desta Unidade Curricular pelo professor.

Assim, serão apresentados neste relatório as técnicas desenvolvidas para obter software seguro, as ferramentas e indicadores de qualidade de software utilizados, bem como, os testes efetuados e, por fim, o modo de testar o código desenvolvido.

## 2. Estratégias Adotadas

### 2.1 Técnicas de desenvolvimento seguro de software utilizadas

A fim de desenvolver código eficiente, de fácil deteção de erros e com um baixo risco de falhas, procurou-se seguir as *guidelines* disponíveis na documentação da linguagem relativas às práticas de Code Standard, como por exemplo: identação de 2 espaços, a utilização de *Camel case* para a criação de nomes de variáveis, classes e funções e o uso de *type inference* para definir os tipos de dados.

Além disto, na tentativa de construir uma aplicação segura, foram aplicadas algumas técnicas de desenvolvimento seguro de software, como a **validação de *inputs***, onde todos os inputs são recebidos como *String* e verificados os seus tamanhos e assegurados que os dados obrigatórios foram inseridos. Na imagem seguinte temos um exemplo de validação de inputs, onde é verificado o tamanho do *pin* e se foram introduzidos apenas números.

\* Não é válida o num telefone  
 \* Não é válida PIN no CC Move!Sign

```

411     println("\nIntroduza o pin:")
412     val pin = scala.io.StdIn.readLine()
413     if (!(pin forall Character.isDigit) && pin.length > 8)
414         die(msg = "Pin not valid", args = "Pin not valid")
  
```

\* Não é válida OTP  
 \* Programa estorna se OTP com letras

Figura 2.1: Exemplo de validação de *input*

A *PartialFunction* **pf** verifica se os parâmetros inseridos estão dentro dos padrões aceites, além de encerrar o programa caso tenha sido colocada alguma uma opção não validada. Como podemos ver na figura seguinte, a função *cmd* valida os tamanhos dos argumentos inseridos.

```

525 val pf: PartialFunction[List[String], List[String]] = {
526   case "GetCertificate" :: tail => getcert = true; tail
527   case "CCMove!Sign" :: tail => ccmove! = true; tail
528   case "ValidateOtp" :: tail => validotp = true; tail
529   case "TestAll" :: tail => testAll = true; tail
530   case "-h" :: tail => help = true; tail
531   case "-u" :: (arg: String) :: tail => userID = arg; tail
532   case "-a" :: (arg: String) :: tail => applicationID = arg; tail
533   case "-s" :: (arg: String) :: tail => pin = arg; tail
534   case "-p" :: (arg: String) :: tail => processID = arg; tail
535   case "-o" :: (arg: String) :: tail => otp = arg; tail
536   case "-d" :: (arg: String) :: tail => docName = arg; tail
537   case "-H" :: (arg: String) :: tail => docHash = arg; tail
538   case unknown(bad) :: tail => die(msg = "unknown argument " + bad + "\n" + usage, args = s"Invalid parameters: $bad")
539 }
  
```

Figura 2.2: Função **pf**

\* Não é válida processID  
 \* Programa estorna se processID incorreto

Outro mecanismo utilizado para se fazer cumprir os requisitos de segurança foi a criação de um sistema para guardar informações sobre a utilização da aplicação em *logs* e garantir a audibilidade da aplicação. A figura abaixo exemplifica um *log* gerado.

```

Jul 03, 2020 11:49:13 P.M. Main$ cmd
INFO: Get Certificate: userID: +351917806614, applicationID: b826359c-06f8-425e-8ec3-50a97a418916
Jul 03, 2020 11:49:14 P.M. Main$CMDgetcertificate sendSOAPrequest
SEVERE: Error java.io.IOException: Server returned HTTP response code: 500 for URL: https://cmd.autenticacao.gov.pt/Ama.Authentication.Frontend/CCMove!DigitalSignature.svc
Jul 03, 2020 11:51:19 P.M. Main$ cmd
INFO: Get Certificate: userID: +351917806614, applicationID: b826359c-06f8-425e-8ec3-50a97a418916
Jul 03, 2020 11:51:19 P.M. Main$CMDgetcertificate sendSOAPrequest
SEVERE: Error java.io.IOException: Server returned HTTP response code: 500 for URL: https://cmd.autenticacao.gov.pt/Ama.Authentication.Frontend/CCMove!DigitalSignature.svc
Jul 03, 2020 11:52:11 P.M. Main$ cmd
INFO: Get Certificate: userID: +351917806614, applicationID: b826359c-06f8-425e-8ec3-50a97a418916
Jul 03, 2020 11:52:12 P.M. Main$CMDgetcertificate sendSOAPrequest
SEVERE: Error java.io.IOException: Server returned HTTP response code: 500 for URL: https://cmd.autenticacao.gov.pt/Ama.Authentication.Frontend/CCMove!DigitalSignature.svc
Jul 03, 2020 11:52:12 P.M. Main$ die
SEVERE: Certificates not found
Jul 03, 2020 11:56:02 P.M. Main$ cmd
INFO: Get Certificate: userID: +351917806614, applicationID: b826359c-06f8-425e-8ec3-50a97a418916
Jul 03, 2020 11:56:03 P.M. Main$CMDgetcertificate sendSOAPrequest
SEVERE: Error java.io.IOException: Server returned HTTP response code: 500 for URL: https://cmd.autenticacao.gov.pt/Ama.Authentication.Frontend/CCMove!DigitalSignature.svc
Jul 03, 2020 11:56:03 P.M. Main$ die
SEVERE: Certificates not found
Jul 03, 2020 11:59:23 P.M. Main$ cmd
INFO: Get Certificate: userID: +351917806614, applicationID: b826359c-06f8-425e-8ec3-50a97a418916
Jul 03, 2020 11:59:24 P.M. Main$CMDgetcertificate sendSOAPrequest
SEVERE: Error java.io.IOException: Server returned HTTP response code: 500 for URL: https://cmd.autenticacao.gov.pt/Ama.Authentication.Frontend/CCMove!DigitalSignature.svc
Jul 03, 2020 11:59:24 P.M. Main$ die
SEVERE: Certificates not found
Jul 04, 2020 12:01:16 A.M. Main$ die
SEVERE: Invalid parameters: -h
Jul 04, 2020 12:02:25 A.M. Main$ cmd
INFO: CCMove!Sign: userID: +351917806614, applicationID: b826359c-06f8-425e-8ec3-50a97a418916, docName: test.scala
Jul 04, 2020 12:02:25 A.M. Main$CMDccmove!sign sendSOAPrequest
SEVERE: Error java.io.IOException: Server returned HTTP response code: 500 for URL: https://cmd.autenticacao.gov.pt/Ama.Authentication.Frontend/CCMove!DigitalSignature.svc
Jul 04, 2020 12:04:31 A.M. Main$ cmd
INFO: CCMove!Sign: userID: +351917806614, applicationID: b826359c-06f8-425e-8ec3-50a97a418916, docName: test.scala
Jul 04, 2020 12:04:32 A.M. Main$CMDccmove!sign sendSOAPrequest
SEVERE: Error java.io.IOException: Server returned HTTP response code: 500 for URL: https://cmd.autenticacao.gov.pt/Ama.Authentication.Frontend/CCMove!DigitalSignature.svc
Jul 04, 2020 12:04:32 A.M. Main$ die
SEVERE: Error running CCMove!Sign service
Jul 04, 2020 12:05:08 A.M. Main$ die
  
```

Figura 2.3: *Log* gerado pela aplicação

\* Programa estorna no testAll, após OTP, se PIN errado

## 2.2 Testes, ferramentas e indicadores de qualidade de software utilizados

### 2.2.1 Lizard

Apesar de ser um programa relativamente simples, optamos por utilizar ferramentas de teste, para tentar garantir um nível de complexidade aceitável na aplicação e minimizar as chances de se haver brechas de segurança.

O Lizard é uma ferramenta *open-source* escrita em Python, para análise de complexidade ciclomática que suporta uma extensa lista de linguagens de programação. Além da complexidade ciclomática chamada pela ferramenta de CCN (*Cyclomatic Complexity Number*), o Lizard também calcula o número de linhas do código sem comentários (nloc), como também a contagem de *tokens* e parâmetros das funções.

No princípio, ao testar o código com o Lizard, foi obtido, na função **cmd**, uma complexidade ciclomática maior que o limite aceitável pela ferramenta. Sendo assim, dividimos esta em diferentes funções e ao correr novamente a ferramenta através do comando: "lizard CMDSOAP.scala" obteve-se o seguinte resultado:

```
$lizard CMDSOAP.scala
=====
NLOC    CCN    token  PARAM  length  location
-----
    23     4    158     1     28  parseResponse@41-68@CMDSOAP.scala
     5     1     24     1      5  parseResponse@192-196@CMDSOAP.scala
     6     1     28     1      7  parseResponse@215-221@CMDSOAP.scala
     6     3     26     2      7  checkTestAll@268-274@CMDSOAP.scala
     6     6     48     5      7  checkGetCert@274-280@CMDSOAP.scala
     6     7     52     6      7  checkCCMove@280-286@CMDSOAP.scala
     6     7     54     6      7  checkValidOTP@286-292@CMDSOAP.scala
    16     4     92     2     21  pemFile@292-312@CMDSOAP.scala
    19     5    124     1     23  getCertChain@312-334@CMDSOAP.scala
     6     1     49     2      7  getCn@334-340@CMDSOAP.scala
    56     9    396     2     79  testAll@340-418@CMDSOAP.scala
     8     1     64     3     12  verify@418-429@CMDSOAP.scala
     8     1    109     2     10  hashPrefix@429-438@CMDSOAP.scala
     5     1     28     1      7  hash@438-444@CMDSOAP.scala
    47    11    378     1     67  cmd@485-551@CMDSOAP.scala
     5     4     71     3      6  parseArgs@553-558@CMDSOAP.scala
     6     1     33     2      7  die@558-564@CMDSOAP.scala
1 file analyzed.
=====
NLOC    Avg.NLOC  AvgCCN  Avg.token  function_cnt  file
-----
    434      13.8    3.9    102.0         17  CMDSOAP.scala
=====
No thresholds exceeded (cyclomatic_complexity > 15 or nloc > 1000000 or length > 1000 or parameter_count > 100)
=====
Total nloc  Avg.NLOC  AvgCCN  Avg.token  Fun Cnt  Warning cnt  Fun Rt  nloc Rt
-----
    434      13.8    3.9    102.0      17         0     0.00  0.00
```

Figura 2.4: Cálculos do Lizard para o código da aplicação

Na figura anterior mostra que o código possui uma complexidade ciclomática média de 3.9, possui também uma média de 13.8 número de linhas por função, além de 17 funções. Demonstrando que o código, após as correções e segundo o relatório da ferramenta, apresenta-se dentro dos limites aceitáveis de complexidade.

## 2.2.2 IntelliJ - Analyzing Tools

Além da ferramenta Lizard, também foi utilizado um *plugin* da IDE IntelliJ para inspecionar o código do programa a procura de erros e *warnings*, onde foram detectadas algumas variáveis que foram declaradas, porém não utilizadas, que foram corrigidas pela própria IDE após a análise.

## 2.3 Dificuldades Encontradas

Por se tratar de uma linguagem que era a primeira vez que todos os elementos do grupo trabalhavam com esta, sentimos algumas dificuldades em respeitar alguns coding standards. Assim, não conseguimos dividir as várias funções por diferentes classes e em diferentes ficheiros, pois obtínhamos erros constantes com os imports que não conseguimos resolver.

Outra dificuldade encontrada ocorreu na verificação da assinatura devolvida pelo servidor SOAP utilizando uma biblioteca JAVA que implementa o RSA. Deste modo, considerámos que não conseguimos testar o resultado final do projeto devido a erros relacionados com as versões dos standards de criptografia.

### 2.3.1 SonarLint

Por fim, utilizamos plugin SonarLint que analisa o código a procura de erros e possíveis falhas de segurança. Instalamos esta ferramenta na IDE IntelliJ e foi nos retornado um relatório onde mostra que o código da aplicação não possui erros, como mostra a figura em baixo.

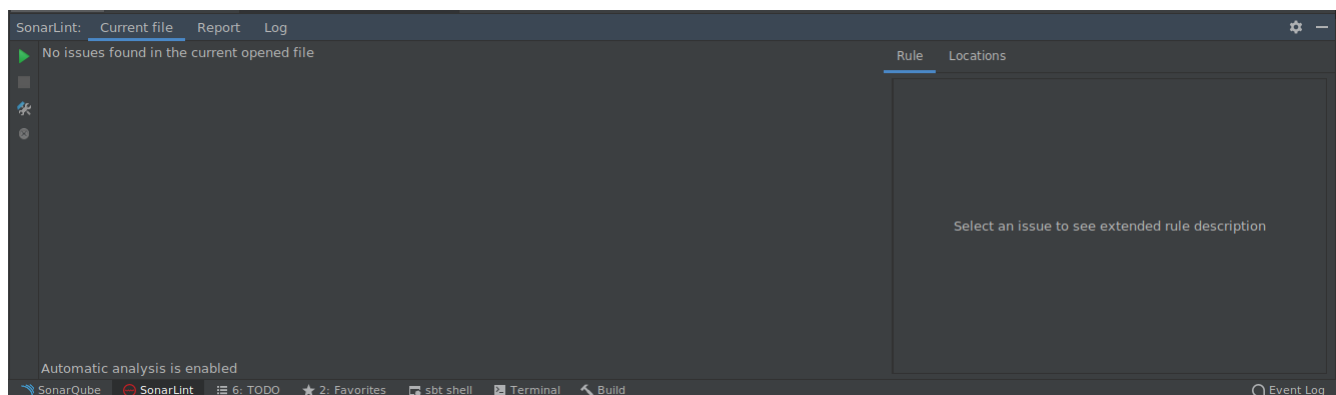


Figura 2.5: Relatório do SonarLint

## 3. Resultados obtidos

### 3.1 GetCertificate

O método GetCertificate, tem como objectivo devolver a cadeia de certificados do utilizador do serviço de Chave Móvel Digital, funciona através do comando:

```
scala CMDSOAP.scala GetCertificate -u +351NNNNNNNNNN
```

Onde o programa envia o *request* para o servidor, onde o *userId* corresponde ao parâmetro inserido após a opção -u”, sendo neste caso, o número do telemóvel do utilizador, mas poderia ser o endereço de email. Apesar do *applicationId* já ter sido implementado ”hard-coded” no programa, pode-se também, por opção, ser inserido um novo através da opção -a”, estas opções também se encontram disponíveis nas outras funções do programa.

Caso o *userId* seja validado, o servidor responde com os certificados pedidos e o programa guarda-os num ficheiro em formato “.pem”, além de imprimi-los na consola.

### 3.2 CCMovelSign

O método CCMovelSign, permite assinar um documento, através da chave móvel digital de um utilizador, e opera através do seguinte comando:

```
scala CMDSOAP.scala CCMovelSign -u +351NNNNNNNNNN -s pin -d docName
```

No qual a aplicação faz um *request* ao servidor enviando o *userId* do utilizador, juntamente ao seu código pin, o nome do documento a ser assinado e o *hash* do documento. Além do *applicationId*, nesta função também pode ser inserido por opção, o *hash* do documento. Caso o *hash* não seja inserido, o programa calcula-o e envia ao servidor.

Feito isto, o servidor retorna para a aplicação o *processId* e envia um sms para o telemóvel do utilizador um código OTP (*One Time Password*). Estes parâmetros serão utilizados pela função ValidateOTP, que devolve a assinatura do documento caso o código OTP seja validado.

### 3.3 ValidateOtp

O método ValidateOtp valida um código OTP através do seguinte comando:

```
scala CMDSOAP.scala ValidateOtp -o OTP -p processID
```

Caso o OTP seja válido, o servidor devolve a assinatura referente ao *processId* inserido.

### 3.4 TestAll

O método TestAll recebe como parâmetros o *userId* e o nome do ficheiro a ser assinado e realiza um teste de todas as funcionalidades da aplicação, além de conferir se a assinatura devolvida pelo servidor é válida.

```
scala CMDSOAP.scala TestAll -u +351NNNNNNNNNN -d assinar.txt
```

```
+++ Test All inicializado +++
```

```
0% ... Leitura de argumentos da linha de comando ficheiro: 'assinar.txt', user: +351NNNNNNNNNN
10% ... A contactar servidor SOAP CMD para operação GetCertificate
20% ... Certificado emitido para 'Nome do Utilizador' pela Entidade de Certificação 'EC de CH
30% ... Leitura do ficheiro 'assinar.txt'
40% ... Geração de hash do ficheiro 'assinar.txt'
50% ... Hash gerada: MDEwDQYJYIZIAWUDBAIBBQAEIO0wxEKY/BwUmvvOyJlvuSQnrkHkZJuTTKSVmRt4UrhV
Introduza o pin:
****
60% ... A contactar servidor SOAP CMD para operação CCMovelSign
70% ... ProcessID devolvido pela operação CCMovelSign: 5ae324d8-88c9-4b6a-8179-c9f5987d9c21
80% ... A iniciar operação ValidateOtp
```

Introduza o OTP recebido no seu dispositivo:

NNNNNN

```
90% ... A contactar servidor SOAP CMD para operação ValidateOtp
100% ... Assinatura devolvida pela operação ValidateOtp: VQ/K052x+T8xkXSrZjxLR3gU5U0Ku9E9EZvV
110% ... A validar assinatura ...
Assinatura verificada? true
```

?  
Em 1.3 referem que não  
conseguem verificar



# Faltou indicar que se deve compilar primeiro o código com scalac,  
seguido de scala -classpath . Main

## 4. Conclusão

Após a concluirmos este projeto, somos capazes de fazer uma retrospectiva do trabalho efetuado, bem como, da importância do conteúdo leccionado nesta Unidade Curricular, e entendemos a criticidade de aplicar metodologias de segurança nos projectos, de modo a não só obter um produto mais seguro, mas sobretudo para alcançar um produto de valor acrescentado.

A concessão de software seguro, por parte de algumas empresas, ainda é visto como uma etapa final do desenvolvimento de produtos e isto implica um aumento de esforços, custo e tempo e não temos total garantia que o software estará protegido. Deste modo, percebemos que a elaboração de software seguro é um processo que deve ser implementado desde o início do projeto (fase de requisitos do sistema) e necessita de ser acompanhado até à fase final. Assim, este processo é atualmente considerado uma estratégia para maximizar o retorno sobre o investimento. Para ser possível este desenvolvimento é necessário adotar métodos que permitem garantir qualidade durante todas as etapas de desenvolvimento.

Concluindo, considerámos que atingimos os objetivos propostos para o Projeto 3, apesar de não conseguirmos verificar a assinatura gerada e devolvida pelo servidor utilizando o RSA.