UNIVERSIDADE DO MINHO

DEPARTAMENTO DE INFORMÁTICA

ENGENHARIA DE SEGURANÇA

Projeto 3

Reverse engineer* da aplicação *CMD-SOAP

Linguagem - *PHP*

Grupo 4

Autores:

Joel Gama (A82202)

Tiago Pinheiro (A82491)

6 de Julho de 2020

Conteúdo

Introdução

O presente relatório surge no âmbito da Unidade Curricular de Engenharia de Segurança integrada no perfil de Criptografia e Segurança da Informação.

Este trabalho é o último trabalho prático da UC. Este trabalho consiste em fazer *reverse engineer* da aplicação **CMD-SOAP**, uma aplicação comando linha (CLI) que permite testar as operações do serviço SCMD (Signature CMD). O objetivo era todos os grupos fazerem os trabalhos em linguagens de programação diferentes, como tal, nós ficamos com a linguagem *php*.

Para além de desenvolver o código era necessário ter certos cuidados, que foram estudados no projeto 1 e projeto 2, realizados anteriormente. Assim como utilizar as ferramentas de *coding standards* vistas no projeto 2.

Código

2.1 cmd_config.php

O ficheiro $cmd\_config.php$ é onde é feita a configuração do *id* da aplicação e a definição da função que o retorna. Desta forma fica mais simples fazer a configuração e obtenção do *id*.

2.2 cmd_soap_msg.php

O ficheiro $cmd\_soap\_msg.php$ é onde são processados os pedidos do cliente.

São definidas as funções $getWsdl$ para obter o *link* utilizado posteriormente para serem feitos os pedidos e a função $getClient$ retorna um cliente inicializado com o argumento anterior. A função $hashPrefix$ recebendo o tipo de *hash* que se pretende adiciona um prefixo à *hash* passada como argumento.

Em seguida vem a função $getCertificate$ que retorna o certificado associado ao utilizador. A $ccmovelsign$ assina o documento passado e a $ccmovelmultiplesign$ assina vários documentos. Por fim, a função $validateOtp$ faz a validação da *OTP* fornecida.

2.3 test_cmd_wsdl.php

O ficheiro de `test\_cmd\_wsdl.php` tem as funções para lidar com os argumentos recebidos e chamar as funções adequadas, depois de tratar do *input*. E também tem a função para testar todas as funcionalidades do programa - `testAll`.

Na parte dos argumentos, existem três funções:

- main - Verifica quantos argumentos recebeu. Se receber menos do que dois (o ficheiro conta como um), retorna uma mensagem de ajuda. Se receber dois, passa à *handleSingle*. Caso contrário, passa à *handleAll*.
- handleSingle - destinada aos comandos *help* e *version*, se não for nenhum do dois, retorna uma mensagem de ajuda.
- handleAll - destinada aos comandos *gc*, *ms*, *mms*, *otp* e *otp*, ou equivalentes. Obriga os argumentos a virem na mesma ordem que foi explicada nos comandos de *help*, verifica

o *input* e se não tiver os argumentos certos (nem a mais nem a menos) retorna uma mensagem de erro. Caso todo corra bem, chama a função necessário com os respetivos parâmetros.

2.4 helpers.php

O ficheiro `helpers.php` contém apenas funções auxiliares.

As funções `defaultHelp`, `helpHelp`, `gcHelp`, `msHelp`, `mmsHelp`, `otpHelp` e a `otpHelp` são as funções que escrevem no ecrã as mensagem de ajuda (opções `-h` ou `--help`).

Ainda neste ficheiro estão presentes as funções que verificam e validam o *input* dado:

- itsUser - Verifica se o número enviado pelo utilizador está no formato correto (+XXX NNNNNNNNN) e é um número (todos os caracteres são dígitos com exceção do primeiro e do quinto que são + e *whitespace*, respetivamente).
- itsPin - Verifica se o *input* é um *pin* (todos os caracteres são dígitos).
- itsOtp - Verifica se o *input* é um *otp* (6 dígitos).
- itsProcessId - Verifica se o *input* tem um formato de *processId* -> xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx, onde `x` tem que uma letra minúscula ou um dígito.

Por último, ainda existe a função `stringSplit` que dada uma *string*, que contém três certificados em formato *PEM*, produz um *array* de tamanho três. Cada entrada do *array* vai ser uma *string*, um certificado em formato *PEM*.

Técnicas de desenvolvimento seguro de software

3.1 Validação de *Input*

Neste programa, uma das técnicas aplicadas para garantir o desenvolvimento seguro de software foi a validação do *input*. Para um *input* ser válido ele apenas pode conter certos tipos de carateres, ter um determinado tamanho e respeitar o formato pedido. Isto aplica-se em várias situações, tanto o número de argumentos passados e a sua ordem como os argumentos propriamente ditos.

3.2 *Naming conventions* e Indentação

Neste trabalho foram utilizadas algumas técnicas de *coding standards*, por exemplo, as mencionadas acima - *naming conventions* e indentação.

No nome das variáveis e das funções foram respeitados os *names default* da linguagem e, se um *code sniffer* da linguagem, o programa é exímio no que toca à indentação.

3.3 *Coding standards* não respeitados

Neste programa, não foram respeitados certos *coding standards*. Por exemplo, existem funções com um elevado número de linhas, devido ao pouco à vontade com a linguagem. Por outro lado, o código não se encontra documento, como houve falta de tempo, escolheu-se não documentar. E, por último, em muitos casos apenas foram usadas mensagens de erro e não retorno de valores de erro.

Ferramentas e indicadores de qualidade de software utilizados

4.1 *PHP Code Sniffer*

O *PHP Code Sniffer* é uma *script* que faz *"sniffs"* em *PHP, JavaScript* e *CSS*. Ele deteta violações nos *coding standards* definidos. É considerada uma ferramenta essencial no desenvolvimento pois assegura que o código se mantém limpo e consistente. Para além de de que ajuda a prevenir erros sintáticos.

4.1.1 Correção do ficheiro *cmd_config.php*

No ficheiro $cmd\_config.php$ foram encontrados inicialmente 5 erros, porém depois das alterações feitas ficaram apenas dois erros por falta de documentação.

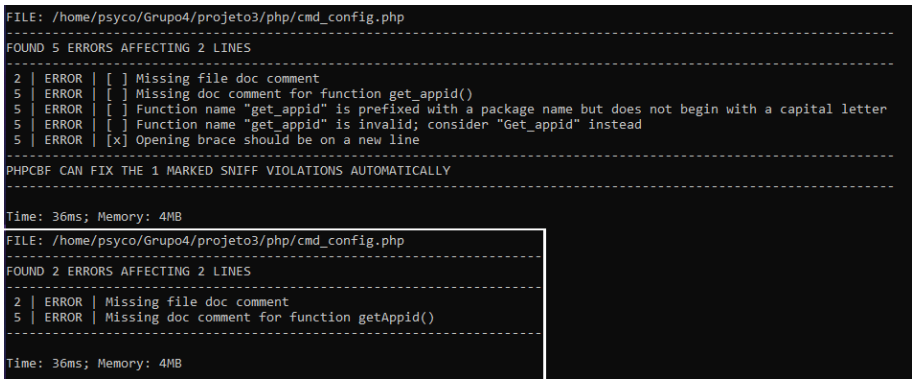

Figura 4.1: Ficheiro $cmd\_config.php$ antes e depois das correções.

4.1.2 Correção do ficheiro *cmd_soap_msg.php*

No ficheiro $cmd\_config.php$ foram encontrados inicialmente 25 erros. Depois de ser feita a correção por parte do grupo ficaram apenas 8 erros. Destes 8 erros 6 eram de falta de documentação e 2 por mais de 85 caracteres numa linha, pois correspondem às linhas dos *links* do *WSDL*.

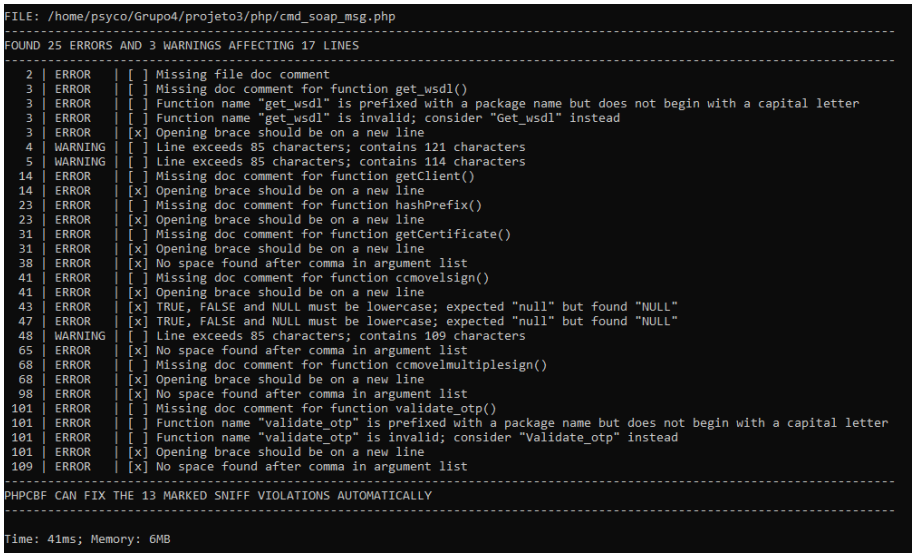

Figura 4.2: Ficheiro $cmd\_soap\_msg.php$ antes das correções.

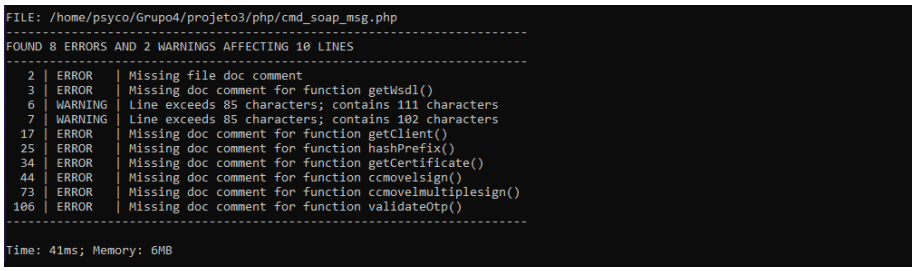

Figura 4.3: Ficheiro $cmd\_soap\_msg.php$ depois das correções.

4.1.3 Correção do ficheiro *test_cmd_wsdl.php*

No ficheiro $test\_cmd\_wsdl.php$ foi encontrados cerca de 400 erros, porém não foi registado o valor exato. Este ficheiro foi o mais complexo de ser tratado. Tendo inicialmente cerca de 1200 linhas de código foi refatorizado, obtendo um valor de quase 700 linhas e apenas 8 erros.

Figura 4.4: Ficheiro $test\_cmd\_wsdl.php$ depois das correções.

4.1.4 Correção do ficheiro *helpers.php*

No ficheiro $helpers.php$ foram encontrados 39 erros inicialmente. Seguindo o mesmo processo que foi utilizado nos outros ficheiros e depois de serem corrigidos vários dos erros ficaram apenas 13 erros por falta de documentação.

```
FILE: /home/psyco/Grupo4/projeto3/php/helpers.php
----------------------------------------------------------------------------------------------------------------------
FOUND 39 ERRORS AND 13 WARNINGS AFFECTING 27 LINES
----------------------------------------------------------------------------------------------------------------------
   2 | ERROR   | [ ] Missing file doc comment
   3 | ERROR   | [ ] Missing doc comment for function its_user()
   3 | ERROR   | [ ] Function name "its_user" is prefixed with a package name but does not begin with a capital letter
   3 | ERROR   | [ ] Function name "its_user" is invalid; consider "Its_user" instead
   9 | WARNING | [ ] Line exceeds 85 characters; contains 96 characters
  10 | ERROR   | [x] Multi-line IF statement not indented correctly; expected 8 spaces but found 14
  10 | ERROR   | [x] Each line in a multi-line IF statement must begin with a boolean operator
  10 | WARNING | [ ] Line exceeds 85 characters; contains 96 characters
  11 | ERROR   | [x] Multi-line IF statement not indented correctly; expected 8 spaces but found 16
  11 | ERROR   | [x] Each line in a multi-line IF statement must begin with a boolean operator
  12 | ERROR   | [x] Multi-line IF statement not indented correctly; expected 8 spaces but found 16
  12 | ERROR   | [x] Each line in a multi-line IF statement must begin with a boolean operator
  12 | ERROR   | [x] Closing parenthesis of a multi-line IF statement must be on a new line
  19 | ERROR   | [ ] Missing doc comment for function its_pin()
  19 | ERROR   | [ ] Function name "its_pin" is prefixed with a package name but does not begin with a capital letter
  19 | ERROR   | [ ] Function name "its_pin" is invalid; consider "Its_pin" instead
  23 | ERROR   | [ ] Expected "for (...) {\n"; found "for(...) {\n"
  32 | ERROR   | [ ] Missing doc comment for function its_otp()
  32 | ERROR   | [ ] Function name "its_otp" is prefixed with a package name but does not begin with a capital letter
  32 | ERROR   | [ ] Function name "its_otp" is invalid; consider "Its_otp" instead
  49 | ERROR   | [ ] Missing doc comment for function its_processId()
  49 | ERROR   | [ ] Function name "its_processId" is prefixed with a package name but does not begin with a capital letter
  49 | ERROR   | [ ] Function name "its_processId" is invalid; consider "Its_processId" instead
  77 | ERROR   | [ ] Missing doc comment for function string_split()
  77 | ERROR   | [ ] Function name "string_split" is prefixed with a package name but does not begin with a capital letter
  77 | ERROR   | [ ] Function name "string_split" is invalid; consider "String_split" instead
  79 | WARNING | [ ] Line exceeds 85 characters; contains 102 characters
  90 | ERROR   | [ ] Missing doc comment for function gc_help()
  90 | ERROR   | [ ] Function name "gc_help" is prefixed with a package name but does not begin with a capital letter
  90 | ERROR   | [ ] Function name "gc_help" is invalid; consider "Gc_help" instead
  92 | WARNING | [ ] Line exceeds 85 characters; contains 91 characters
 101 | WARNING | [ ] Line exceeds 85 characters; contains 122 characters
 104 | ERROR   | [ ] Missing doc comment for function ms_help()
 104 | ERROR   | [ ] Function name "ms_help" is prefixed with a package name but does not begin with a capital letter
 104 | ERROR   | [ ] Function name "ms_help" is invalid; consider "Ms_help" instead
 106 | WARNING | [ ] Line exceeds 85 characters; contains 88 characters
 116 | WARNING | [ ] Line exceeds 85 characters; contains 122 characters
 119 | ERROR   | [ ] Missing doc comment for function mms_help()
 119 | ERROR   | [ ] Function name "mms_help" is prefixed with a package name but does not begin with a capital letter
 119 | ERROR   | [ ] Function name "mms_help" is invalid; consider "Mms_help" instead
 122 | WARNING | [ ] Line exceeds 85 characters; contains 91 characters
 132 | WARNING | [ ] Line exceeds 85 characters; contains 122 characters
 135 | ERROR   | [ ] Missing doc comment for function otp_help()
 135 | ERROR   | [ ] Function name "otp_help" is prefixed with a package name but does not begin with a capital letter
 135 | ERROR   | [ ] Function name "otp_help" is invalid; consider "Otp_help" instead
 137 | WARNING | [ ] Line exceeds 85 characters; contains 88 characters
 143 | WARNING | [ ] Line exceeds 85 characters; contains 124 characters
 147 | WARNING | [ ] Line exceeds 85 characters; contains 122 characters
 150 | ERROR   | [ ] Missing doc comment for function test_help()
 150 | ERROR   | [ ] Function name "test_help" is prefixed with a package name but does not begin with a capital letter
 150 | ERROR   | [ ] Function name "test_help" is invalid; consider "Test_help" instead
 163 | WARNING | [ ] Line exceeds 85 characters; contains 122 characters
----------------------------------------------------------------------------------------------------------------------
PHPCBF CAN FIX THE 7 MARKED SNIFF VIOLATIONS AUTOMATICALLY
----------------------------------------------------------------------------------------------------------------------

Time: 42ms; Memory: 6MB
```

Figura 4.5: Ficheiro $helpers.php$ antes das correções.

```
FILE: /home/psyco/Grupo4/projeto3/php/helpers.php
----------------------------------------------------------------------
FOUND 13 ERRORS AFFECTING 13 LINES
----------------------------------------------------------------------
   2 | ERROR | Missing file doc comment
   3 | ERROR | Missing doc comment for function itsUser()
  21 | ERROR | Missing doc comment for function itsPin()
  34 | ERROR | Missing doc comment for function itsOtp()
  51 | ERROR | Missing doc comment for function itsProcessId()
  79 | ERROR | Missing doc comment for function stringSplit()
  94 | ERROR | Missing doc comment for function defaultHelp()
 101 | ERROR | Missing doc comment for function helpHelp()
 124 | ERROR | Missing doc comment for function gcHelp()
 140 | ERROR | Missing doc comment for function msHelp()
 157 | ERROR | Missing doc comment for function mmsHelp()
 175 | ERROR | Missing doc comment for function otpHelp()
 193 | ERROR | Missing doc comment for function testHelp()
----------------------------------------------------------------------

Time: 45ms; Memory: 6MB
```

Figura 4.6: Ficheiro $helpers.php$ depois das correções.

4.2 *VS Code PHP Code Checker*

Durante o processo de desenvolvimento do código foi utilizada uma extensão para o *Visual Studio Code* que fazia a verificação em tempo real do código que estava a ser escrito. Dessa forma conseguimos poupar vário tempo de *debug* pois encontrar os erros à medida que acontecem é mais rápido do que ter de compilar o ficheiro.

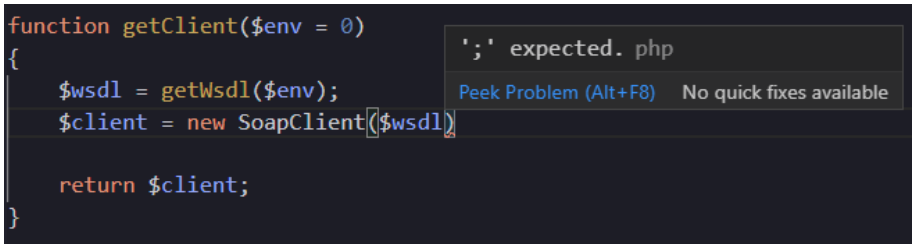

Figura 4.7: Exemplo de erro no *VS Code*.

Testes realizados

Para obter os melhores resultados nos testes é sempre recomendado fazer testes gerados automaticamente. Porém pela falta de tempo apenas foi possível fazer testes manuais.

Nas figuras seguintes estão representados dois testes para mostrar geração de certificados e fazer assinatura com parâmetros inválidos, respetivamente.

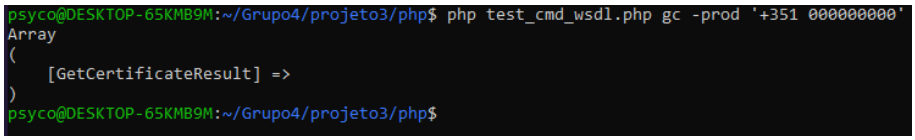

Figura 5.1: Teste de geração de certificado.

Figura 5.2: Teste de assinatura.

Apesar de apenas termos estes dois exemplos, foram testados todas as opções do programa. Apenas uma das opções (*otp*) não estava a funcionar corretamente e, consequente, a opção *test* também não funcionava totalmente.

Conclusão

Este trabalho foi interessante continuar a exploração do tema que envolve os certificados, o cartão do cidadão e a assinatura digital, tema este que foi abordado na cadeira de Tecnologia Criptográfica, no perfil de Criptografia e Segurança da Informação. E, por outro lado, foi possível continuar a exploração dos temas abordados neste semestre.

A principal dificuldade enfrentada neste trabalho foi a linguagem, uma vez que, foi a primeira vez que o grupo programou em *php*.

No final, o grupo ficou satisfeito com o resultado obtido, apesar da função `validateOtp` não retornar nada quando obtém sucesso, um erro que o grupo não consigo solucionar.