FORMAÇÃO AVANÇADA
EM CIBERSEGURANÇA

CNCS
Centro Nacional de Cibersegurança
PORTUGAL

CYBER ACADEMY

C-ACADEMY FORMAÇÃO AVANÇADA EM CIBERSEGURANÇA

CNCS Centro Nacional de Cibersegurança PORTUGAL

NÍVEL A

Princípios Técnicos em Cibersegurança

35 horas / José Simão / Março 2023

C-ACADEMY FORMAÇÃO AVANÇADA EM CIBERSEGURANÇA

CNCS Centro Nacional de Cibersegurança PORTUGAL

Laboratório 2 – Sockets, GitHub e Introdução à criptografia

Laboratório 2

Introdução

Este segundo laboratório tem os seguintes objetivos:

a) Analisar programas que usam sockets para comunicação cliente/servidor;
b) Realizar as operações essenciais na plataforma GitHub para partilha de código;
c) Aplicar proteções criptográficas para proteger e desproteger ficheiros, via ferramenta OpenSSL e, opcionalmente, através de bilioteca criptográfica em Python.

Em anexo ao laboratório existem ficheiros Python com código base para as questões 2 e 5.

Entrega

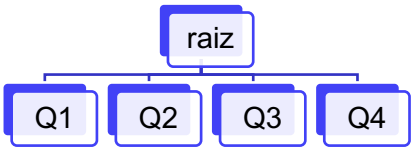
- (ZIP) Código fonte
- (PDF) Documento síntese do laboratório com o endereço do repositório do grupo, ecrãs e respostas solicitadas

1. Aplicação para lista de tarefas com arquitetura cliente/servidor

a) Considere os ficheiros `client_todo.py` e `server_todo.py` em anexo.
b) Abra duas linhas de comando do sistema (através do Visual Code usando a aplicação nativa do sistema operativo, `cmd.exe` no caso do Windows, `terminal` no caso macOS), ambas na diretoria onde tem o código fonte.
c) Execute o cliente e o servidor, indicando no código do servidor a porta que pretende usar. Registe no documento de sintése uma captura de ecrã onde seja possível ver os dois programas a executar.
d) *(extra)* Acrescente ao sistema um comando para o cliente saber o número de itens totais da lista que ainda não estão completos, sem o cliente necessitar de obter a listagem dos itens individuais.

2. GitHub

Nsta alínea e nas próximas use uma estrutura de directorias com o nome da alínea, como sugerido na figura:



a) Cada elemento do grupo cria uma conta na plataforma GitHub (ou garante o acesso através de uma conta que já tenha criado anteriormente);
b) Um dos elementos do grupo cria um projeto GitHub e dá acesso a esse projeto a todos os elementos do grupo e ao formador (utilizador: jsimaoisel);
c) Todos os elementos instalam a aplicação GitHub Desktop (https://desktop.github.com);
d) Um dos elementos do grupo adiciona o código fonte da alínea 1 no repositório criado;
e) Os restantes elmentos fazem *pull* do respositório;
f) Todos os elementos acrescentam à raiz dos seus repositórios locais um ficheiro de texto cujo conteúdo é o nome do formando;
g) Todos os elementos do grupo fazem push para o repositório e verificam a correta sincronização do estado global do respositório.

3. Criptografia com OpenSSL

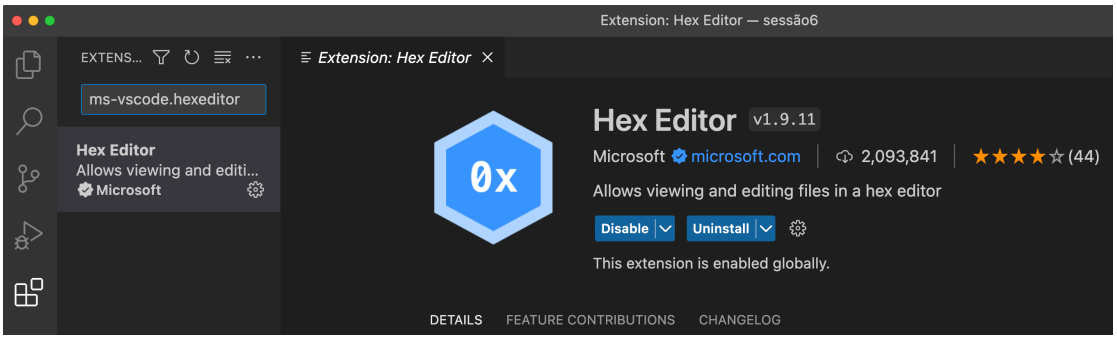
A biblioteca OpenSSL é uma biblioteca com código fonte aberto, escrita maioritariamente na linguagem C, e usada em muitas aplicações comerciais para realizar diferentes operações criptográficas (https://www.openssl.org). Existem duas formas de usar esta biblioteca: atráves do código fonte ou através de ferramenta de linha comandos. Neste laboratório iremos usar o segundo caso.

Na generalidade dos sistemas operativos Linux e macOS a ferramenta OpenSSL já está instalada e disponível na linha de comandos:

```
$ openssl version↵
LibreSSL 3.3.6
```

Nos sistemas Windows, há duas hipóteses: ou se compila o código C, ou se confia em algum binário que já tenha sido compilado por alguém. Vamos usar a segunda hipótese. Faça *download* do executável mais recente atráves deste link https://indy.fulgan.com/SSL/, cuja referência está presente nas páginas de documentação do OpenSSL (https://wiki.openssl.org/index.php/Binaries). Coloque o executável alcancável na *path* do sistema operativo (se necessário consulte as instruções para o efeito no laboratório 1).

Recomenda-se também a instalação da extensão "ms-vscode.hexeditor" no editor *visual code* para facilitar a visualização e moficação de ficheiros em modo binário.



a) Cálculo de *hash*

A ferramenta OpensSSL calcula valores de hash com base no conteúdo de um ficheiro:

```
$ openssl dgst -sha512 <file>
SHA512(...)=....abd3244f....
```

Documentação sobre o documento `dgst`: https://www.openssl.org/docs/man1.1.1/man1/openssl-dgst.html

i. Calcule o *hash* SHA512 do ficheiro com o código fonte da biblioteca Apache Commons, obtido através deste link: https://dlcdn.apache.org//commons/crypto/source/commons-crypto-1.2.0-src.zip;
ii. Confirme que o valor de *hash* é igual ao anunciado no site: https://downloads.apache.org/commons/crypto/source/commons-crypto-1.2.0-src.zip.sha512;
iii. Modifique apenas 1 byte no ficheiro e volte a gerar o hash, comparando com o anterior;
iv. No relatório síntese coloque os ecrãs da geração do hash, da modificação do ficheiro e da segunda geraão do hash.

b) Cálculo de *Message Authentication Code* (MAC)

A ferramenta OpenSSL calcula o MAC do conteúdo de um ficheiro e tem como parâmetro uma chave (neste caso uma palavra-passe):

```
$ openssl dgst -sha512 -hmac "cncs!!2023" x.zip
HMAC-SHA512(...)=...abc524a...
```

i. Calcule o MAC do PDF deste enunciado usando uma chave escolhida pelo grupo;
ii. No relatório síntese coloque os ecrãs da geração do MAC.

c) Cifra e decifra simétrica

A ferramenta OpenSSL realiza a cifra e decifra simétrica de ficheiros, podendo usar diferentes algoritmos e modos de operação. O comandos é sempre o `enc` mas a opção `-d` realiza a decifra. Por exemplo, os seguintes comandos cifram e decifram o ficheiro `x.pdf` usando a primitiva AES e o modo de operação CBC:

Cifra

```
$ openssl enc -aes-256-cbc -in x.pdf -out x.enc
enter aes-256-cbc encryption password:...
```

Decifra

```
$ openssl enc -aes-256-cbc -d -in x.enc -out x_decrypted.pdf
enter aes-256-cbc decryption password:...
```

Documentação sobre o documento `enc`: https://www.openssl.org/docs/man1.1.1/man1/enc.html

i. Um elemento do grupo cifra o ficheiro PDF do enunciado usando uma chave combinada com todos os elementos;
ii. O elemento que cifrou o ficheiro adiciona o ficheiro cifrado ao repositório;
iii. Os restantes elementos obtém o ficheiro e confirmam que conseguem realizar a decifra abrindo corretamente o PDF nos seus sistemas.
iv. No documento de síntese escreva a chave usada para cifrar e decifra o ficheiro.

d) Modos de operação

Pretende-se realizar a experiência abordada na sessão teórica, onde o mesmo ficheiro com uma imagem é cifrado duas vezes com o mesmo algoritmo, mas com dois modos de operação diferentes.

Nesta experiência é preciso copiar bytes de ficheiros em posições específicas. Nos sistemas Linux e macOS, os comandos `head`, `tail` e `cat` resolvem essa questão. Consideremos o exemplo seguinte:

```
$ head -c 54 p1.bmp  > header
$ tail -c +55 p2.bmp > body
$ cat header body > new.bmp
```

Esta sequência de comandos começa por copiar do ficheiro `p1.bmp` os primeiros 54 *bytes* (dimensão do cabeçalho de imagens no formato BMP) para o ficheiro `header`. Depois, copia para o ficheiro `body` todos os *bytes* a partir da posição 55 do ficheiro `p2.bmp`, até ao fim do ficheiro. Finalmente, junta no ficheiro `new.bmp` os bytes de ambos os ficheiros `header` e `body`.

O ficheiro Python `copy-file-bytes.py` em anexo tem duas funções que fazem o equivalente a estes comandos. A função `copy_binary_file` copia um intervalo de *bytes* do ficheiro origem para o ficheiro destino, e a função `join_binary_file` junta o conteúdo de dois ficheiros num só.

i. Considere a imagem em anexo `c-academy.bmp` e realize a cifra com AES e modo de operação ECB, de maneira a cifrar apenas o corpo da mensagem, mantendo a informação original do cabeçalho no ficheiro produzido;
ii. Repita o processo mas agora com o modo de operação CBC;
iii. Adicione os dois ficheiros resultantes da experiência no repositório GitHub;
iv. Explique no documento síntese a diferença de resultados entre os dois ficheiros cifrados.

e) Criptografia assimétrica

Pretende-se nesta alínea realizar uma experiência com criptografia assimétrica, nomeadamente, gerar um par de chaves assimétricas, obter a assinatura de um ficheiro e verificar a assinatura

Para gerar um par de chaves RSA com dimensão 2048 bits, armazenar a chave pública num certificado e o material privado no ficheiro `key.pem`, use o comando:

```
$ openssl req -newkey rsa:2048 -nodes -keyout key.pem -x509 -days 365 -out cert.pem
```

Para este comando guardar a chave pública no certificado serão feitas algumas perguntas para o openssl recolher os dados do *distinguished name* a colocar no certificado. Responda com informação fictícia ou deixe em branco.

O comando seguinte mostra na consola o conteúdo do certificado:

```
$ openssl x509 -in cert.pem -text -noout
```

O comando seguinte gera uma assinatura para o ficheiro `x.pdf`. Note que o comando não incorpora os *bytes* da assinatura no PDF, apenas a escreve no ficheiro `signature.bin`:

```
$ openssl dgst -sha256 -sign key.pem -out signature.bin x.pdf
```

O comando seguinte obtém do certificado a chave pública e verifica a assinatura gerada anteriormente:

```
$ openssl x509 -in cert.pem -pubkey -noout > pubkey.pem
$ openssl dgst -sha256 -verify pubkey.pem -signature signature.bin x.pdf
```

i. Adapte os comandos anteriores para assinar e verificar o PDF do enunciado;
ii. Modifique 1 byte do ficheiro PDF original usando o editor visual code. Adicione ao relatório de síntese o ecrã capturado com o processo de edição;
iii. Repita o processo de verificação e verifique que existe falha. Inclua no documento síntese a captura de ecrã da verificação com sucesso e com falha.

4. (extra) Criptografia em Python

Considere o ficheiro `fernet-sym-cypher.py` em anexo, cujo programa gera uma chave simétrica, realizando depois a cifra e a decifra uma string. O programa usa a biblioteca `cryptography`, em particular o módulo simplificado para cifra simétrica autenticada (`Fernet`). A biblioteca é composta por este módulo e por um conjunto de serviços de criptografia onde é necessário fazer mais algumas parametrizações para um correto uso da biblioteca.

i. Intale a biblioteca `cryptography` usando a aplicaçãp pip: https://cryptography.io/en/latest/;
ii. Usando o módulo Fernet, realize um programa que gera e grava em ficheiro uma nova chave para usar neste módulo;
iii. Considere o ficheiro `server_todo_fernet.py` fornecido em anexo. O programa corresponde a uma versão modificada do servidor de tarefas, onde cada tarefa é guardada cifrada no ficheiro `todo_list.fernet`, cada vez que há uma alteração à lista. O programa tem um erro – quando o ficheiro está a ser carregado para memória na fase inicial da aplicação (caso o ficheiro exista), a informação sobre a tarefa estar completa não está a ser considerada. Corrija o erro;
iv. Teste o cliente anterior e o novo servidor, observando que o ficheiro é sempre gravado com os itens cifrados. Coloque ecrãs no documento síntese que demonstrem os testes;
v. Publique o código no repositório.