# Análise do software - Capítulo V2

Para efetuarmos a análise ao nosso software escolhemos o capítulo 2 (V2) de entre os propostos.

## V2.1 - Password Security

Pontos **incluídos** no nosso software:

- **2.1.1** a **2.1.4**:
  - Estes tópicos referem os requisitos para a criação de uma password segura, na nossa opinião, e foi implementado aquando da **criação de um sujeito**.
- **2.1.7** (*);
- **2.1.9** (**);
- **2.1.12** (***).

- Evidências:

```
password = getpass.getpass("Enter password for the new organization creator: ")
if not password:
    print("Error: Password cannot be empty.")
    return 1
if len(password) > 128:
    print("Error: Password cannot be longer than 128 characters.")
    return 1
if len(password) < 12 or not re.search(r"[A-Z]", password) or not re.search(r"[a-z]", password) or not re.search(r"[0-9]", password) or not re.search(r"[!@#$%^&*(),.?\":{}|<>]", password):
    print("Error: Password must be at least 12 characters long and include uppercase, lowercase, digit, and special character.")
    return 1
```

O que <u>não</u> foi <u>implementado</u>:

- **2.1.5** a **2.1.6**:
  - Não foi implementado, mas para o fazer podíamos criar um outro comando "*./change_password <session_file>*", por exemplo.
- **2.1.8**:
  - Não foi implementado devido à falta de UI (User Interface) que facilitava a implementação do mesmo.
- **2.1.10**:
  - Não verificamos a existência de ciclos na nossa Password;
  - Para a implementação bastaria apenas a criação de uma função que ficaria encarregue de fazer essa verificação.
- (***) **2.1.12**.
  - Como nós apenas temos como interface um terminal (CLI) não permite que visualize a password.
  - Para implementar isso nós teríamos uma UI (User Interface) que permitisse o utilizador de escolher se quer ver por um curto período de tempo a password escrita.

**Pontos a referir**:

- (*) **2.1.7**:
    - Nós não utilizamos API's externas porque não se aplica à escala do nosso software;
    - Por outro lado, todas as nossas password têm, pelo menos, 12 caracteres de entre os quais letras maiusculas, números e caracteres especiais;
    - De referir ainda que o top 10.000 passwords mais comuns no mundo têm todas um número de caracteres inferior a 12.
- (**) **2.1.9**:
    - A nossa password possui um conjunto de requisitos, que a tornam uma password forte, de entre as quais a necessidade de caracteres especiais.

---

## V2.2 - General Authenticator Security

Pontos **incluídos** no nosso software:

- **2.2.1** (*)
- **2.2.4:**
    - Nós utilizamos a *Public Key* do repositório para encriptação das mensagens e temos um *session file* que contém o id da sessão atual;
    - O que podia ser melhorado:
        - O *session file* não é encriptado quando é passado.
        - Apenas é verificado com recurso a uma função auxiliar.

- **2.2.6:**
    - A nosso software é resistente a ataques *replay devido à* utilização de um *timestamp* único que será verificado se está dentro de um intervalo para o pedido ser válido;

- Evidências:

```
encrypted_data = {"encrypted_data": encrypt_data(data, REPO_PUB_KEY)}
```

```
encrypted_data = request.json.get("encrypted_data")
if not encrypted_data:
    return handle_error("Missing encrypted data", 400)

data = decrypt_data(encrypted_data, current_app.config["PRIVATE_KEY"])
```

O que não foi implementado:

- **2.2.2 - 2.2.3**:
  - Na nossa aplicação não temos a necessidade de utilizar SMS ou email para autenticação para a escala da nossa aplicação.
- **2.2.5**:
  - Não temos autenticadores externos - não faz sentido tendo em conta a escala da nossa aplicação.
- **2.2.7**:
  - Não temos uma OTP, mas sim um timestamp;
  - Não integramos dispositivos FIDO.

**Pontos a referir**:

- (*) **2.2.1** - Caso o utilizador erre uma vez a password, o servidor encerra.

---

## V2.3 - Authenticator Lifecycle

O que não foi implementado:

- **2.3.1**:
  - Não temos passwords geradas automaticamente e de forma aleatória;
  - Começámos por fazer assim, mas como não era possível alterar a password deixámos de ter passwords geradas pela aplicação.

- **2.3.2 - 2.3.3**:
  - Como não utilizamos autenticadores externos não são aplicáveis estes pontos.

---

# V2.4 - Credential Storage

Pontos **incluídos** no nosso software:

- **2.4.1**:
  - A password é guardada encriptada (hashed);
- **2.4.2**:
  - A key_size tem 32 bytes - 256 bits;
- **2.4.3**:
  - Utilizamos na função importada *generate_password_hash*.

- Evidências:

```python
# Generate a random AES key
def generate_aes_key(key_size=32):  # 32 bytes = 256 bits
    return os.urandom(key_size)


# Encrypt the file using AES (CBC mode with PKCS7 padding)
def encrypt_file(file_path, aes_key):
    iv = os.urandom(16)  # Initialization vector
    cipher = Cipher(algorithms.AES(aes_key), modes.CBC(iv), backend=default_backend())
    padder = sym_padding.PKCS7(algorithms.AES.block_size).padder()
    encryptor = cipher.encryptor()

    encrypted_file_path = f"{file_path}.enc"
    with open(file_path, "rb") as infile, open(encrypted_file_path, "wb") as outfile:
        outfile.write(iv)  # Escreve o IV no início do arquivo
        while chunk := infile.read(4096):
            padded_data = padder.update(chunk)
            outfile.write(encryptor.update(padded_data))
        outfile.write(encryptor.update(padder.finalize()) + encryptor.finalize())

    return encrypted_file_path, iv


# Encrypt the AES key using a public RSA key
def encrypt_aes_key(aes_key, public_key):
    encrypted_key = public_key.encrypt(
        aes_key,
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None,
        ),
    )
    return encrypted_key
```

O que <u>não</u> foi <u>implementado</u>:

- **2.4.4**:
  - Não utilizamos *bcrypt*, mas utilizamos equivalente (**PBKDF2**).
- **2.4.5**:
  - Não utilizamos claramente o identificado neste ponto, porém usamos outra alternativa:
    - Utilizamos a *generate_password_hash*, que internamente usa **PBKDF2** para a key e aplica salt para proteger a hash das keys.

## V2.5 - Credential Recovery

Pontos **incluídos** no nosso software:

- **2.5.2**:
  - Não *secret questions* porque não temos recovery.
- **2.5.3**:
  - Não temos default para o nome do sujeito;
  - Podemos dar o nome de “admin” a um sujeito, mas só se for nossa intenção, nunca é de default.

O que não foi implementado:

- **2.5.1**:
  - Não temos nada relacionado com recuperação de credenciais (*).
- **2.5.4 - 2.5.7**:
  - Não temos nada relacionado com recuperação de credenciais (*).

(*) **Como faríamos:**
Para implementar o **Credential Recovery**, podemos adotar métodos como recuperação por email já que todos os utilizadores tem todos apenas um email. O fluxo envolve desenvolvimento modular, garantindo segurança com criptografia e prevenção contra ataques. A implementação pode ser feita em fases. Testes e monitoramento contínuos assegurarão confiabilidade e usabilidade.

## V2.6 - Look-up Secret Verifier

O que não foi implementado:

- **2.6.1-2.6.3**:
    - Não temos secrets (*).

(*) **Como faríamos:**
Para implementar o **Look-up Secret Verifier**, podemos usar métodos como tokens secretos compartilhados, tabelas de verificação e autenticação multifatores. O processo pode incluir desenvolvimento modular, criptografia robusta, e integração com sistemas existentes. A implementação em fases prioriza os métodos mais essenciais, seguida de testes de segurança e monitoramento. Essa abordagem garante um sistema escalável, seguro e em conformidade com requisitos de privacidade.

---

## V2.7 - Out of Band Verifier

O que não foi implementado:

- 2.7.1 - 2.7.6:
    - Não temos autenticadores externos na nossa aplicação, visto que não nos faz sentido implementarmos.
    - Não achámos necessário a segurança de 2 fatores.

---

## V2.8 - One Time Verifier

O que não foi implementado:

- **2.8.1** - **2.8.6**:
  - Não temos OTPs, temos é um timestamp para os pedidos.
- **2.8.7**:
  - Como referido anteriormente, não temos autenticações externas.

---

## V2.9 - Cryptographic Verifier

Pontos **incluídos** no nosso software:

- **2.9.3**:
  - Utilizamos algoritmos criptográficos:
    - AES;
    - HMAC;
    - RSA;
    - SHA-256.

O que não foi implementado:

- **2.9.1**:
  - Guardamos as chaves localmente na base de dados, não utilizamos TPM nem HSM.
- **2.9.2**:
  - Não usámos nonces, mas sim timestamps.

---

# V2.10 - Service Authentication

Pontos **incluídos** no nosso software:

- **2.10.1**:
  - Utilizamos criptografia assimétrica, utilizamos timestamps e tempo de expiração para sessões.
  - Melhorar:
    - Utilização de tokens temporários (JWT com curta validade).
- **2.10.2**:
  - Implementado em parte:
    - Usamos hashes para as passwords;
    - Novos subjects têm de ser completamente configurados.
- **2.10.3**:
  - As nossas passwords são armazenadas com um secure hash;
  - *PBKDF2*, através do hash iterativo vai dificultar ataques offline.

- Evidências:

```
encrypted_data = request.json.get("encrypted_data")
if not encrypted_data:
    return handle_error("Missing encrypted data", 400)

data = decrypt_data(encrypted_data, current_app.config["PRIVATE_KEY"])

timestamp = data.get("timestamp")
if not timestamp or not validate_timestamp(timestamp):
    return handle_error("Invalid or missing timestamp", 400)
```

O que não foi implementado:

- **2.10.4**:
  - Não utilizamos API keys, à exceção da chave privada do repositório.

---

Trabalho realizado por:

- Abel Teixeira, Nº 113655;
- Filipe Sousa, Nº 114196;
- Tiago Albuquerque, Nº 112901.