

Assignment 1: Vulnerable app

Análise e Exploração de Vulnerabilidades

Curso: Mestrado em Cibersegurança

Grupo: David Morais, nmec 93147
Gerson Marques, nmec 105428

Preparação: Aveiro, 21 de Fevereiro de 2022

Circulação: Docentes e discentes de AEV.

Introdução

Serve o presente relatório para documentar e explicar as decisões tomadas na implementação de uma aplicação vulnerável criada no âmbito do primeiro projeto da cadeira de Análise e Exploração de Vulnerabilidade, e vão ser abordadas as vulnerabilidades presentes na mesma, como estas podem ser atacadas e, por fim, como podem ser mitigadas.

A aplicação desenvolvida foi implementada usando Python e a Flask framework, usando uma base de dados MySQL para persistência e sendo deployed em docker containers, sendo que ferramentas extras foram usadas para a análise e exploração das vulnerabilidades presentes e estas vão ser detalhadas na secção seguinte, cada uma na vulnerabilidade a que lhe diz respeito. O objetivo da aplicação é uma loja de móveis em segunda mão, onde um utilizador pode criar uma conta para poder comprar e vender produtos.

É de fazer notar que todas as imagens presentes neste relatório encontram-se também no diretório */analysis* do deliverable e, por forma a simplificar o processo de desenvolvimento, foram abstraídos de detalhes desnecessários como a finalização de compras (assim como o shopping cart associado) e páginas meramente informativas como por exemplo contactos ou informações sobre o objetivo do website.

Vulnerable app

Esta secção foca-se na apresentação das vulnerabilidades presentes na aplicação desenvolvidas, como um potencial atacante pode estar ciente delas e como o mesmo pode usar este conhecimento (e que ferramentas pode usar) para as explorar, causar danos ou extrair informação da mesma. As vulnerabilidades presentes são:

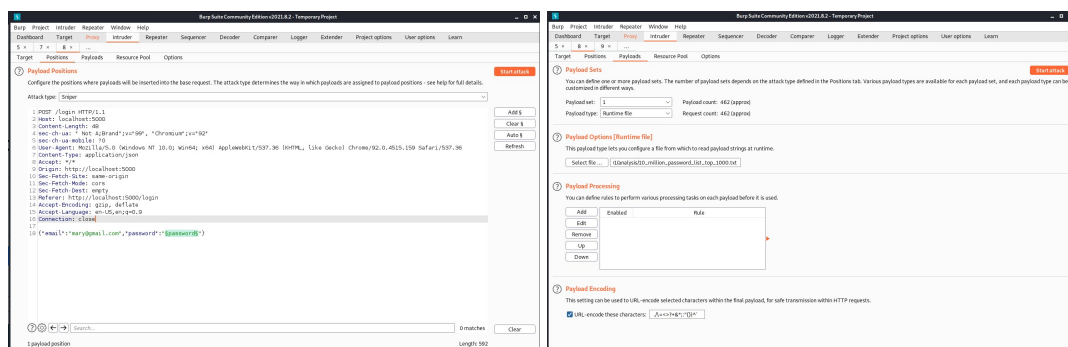
- *CWE-521*: Weak Password Requirements
- *CWE-287*: Improper Authentication
- *CWE-89*: SQL Injection
- *CWE-22*: Path Transversal
- *CWE-434*: Unrestricted Upload of File with Dangerous Type
- *CWE-79*: Cross-site Scripting

Todas estas vulnerabilidades encontram-se presentes no Mitre top 25 e podem ser exploradas de forma individual, mas em alguns casos específicos umas podem ser usadas para despoletar ou facilitar o processo de outras.

Weak Password Requirements (CWE-521)

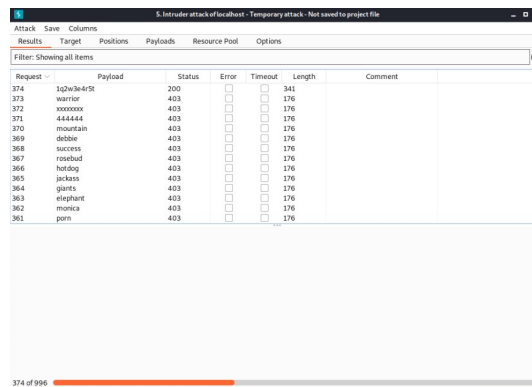
Esta vulnerabilidade é em referência a, ao se fazer o registo de uma conta no site, não ser requerido que o utilizador tenha uma password forte, sendo possível criar contas com passwords com baixa entropia como, por exemplo, só uma letra. Desta forma, um atacante consegue muito facilmente, através de bruteforce ou ataques de dicionários, obter as passwords de alguns utilizadores (pelo menos aqueles que não tem as boas práticas necessárias para criar passwords fortes de livre vontade). Isto pode ser detectado por um atacante, através do simples facto de que, no formulário de registo de conta, não ser levantado nenhum erro quando a password escolhida é fraca.

Usando o burp suite e uma lista das 1000 passwords mais comuns, um atacante consegue fixar o email de um utilizador (que conseguimos enumerar manualmente, visto o login dar demasiada informação de erro no caso de uma autenticação falhada, ou através de SQL Injection, abordado em mais detalhe numa subsecção seguinte) e facilmente obter as passwords do utilizador.



Dos três utilizadores carregados no init da base de dados, isto só funciona com um porque, como foi dito, esta abordagem baseia-se unicamente na falta de boas práticas de um utilizador ao criar as suas credenciais. Posto isto, à 374ª tentativa, o status code associado ao payload '1q2w3e4r5t' é 200, podendo concluir-se que esta é a password do utilizador cujo mail é 'mary@gmail.com'

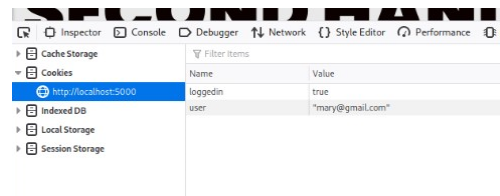
Assignment 1: Vulnerable app



Request	Payload	Status	Error	Timeout	Length	Comment
374	sqw3e3e4t	200			341	
373	warrior	403			176	
372	xxxxxxx	403			176	
371	aaaaaaa	403			176	
370	mountain	403			176	
369	debbie	403			176	
368	success	403			176	
367	rosebud	403			176	
366	hotdog	403			176	
365	jackson	403			176	
364	giants	403			176	
363	elephant	403			176	
362	monica	403			176	
361	porn	403			176	

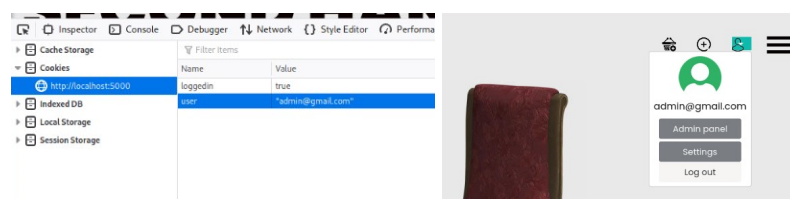
Improper Authentication (CWE-287)

Esta vulnerabilidade permite a um atacante fazer o bypass do processo de autenticação usando os cookies guardados para este propósito, modificando-os e enviando-os ao servidor, uma vez que estes não são verificados nem validados no server-side, resultando em improper authentication. Isto pode ser detectável por um atacante visto que, ao fazer um login numa conta que tenha criado, são gerados dois cookies - um “loggedin” que, como o nome indica, verifica se o utilizador está autenticado, e um “user” que identifica o utilizador.



Name	Value
loggedin	true
user	"mary@gmail.com"

Desta maneira, se o utilizador alterar os cookies à mão por forma a este conter um user diferente, por exemplo admin (*admin@gmail.com*, que pode ser enumerado das mesmas maneiras mencionadas acima), isto leva a querer que seja autenticado como administrador do sistema e tenha acesso a funções com privilégios mais altos.



SQL Injection (CWE-89)

Uma das vulnerabilidades mais comuns, em que os inputs do utilizador não são sanitizados ou verificados, ou são feitos de forma imprópria. Esta pode ser explorada através da search bar nas página de produtos, visto que um atacante pode presumir que, muito provavelmente, a query à base de dados está a ser feita na forma de:

SELECT [atributos] FROM [tabela de produtos] WHERE [name attribute] LIKE '%[q]%';

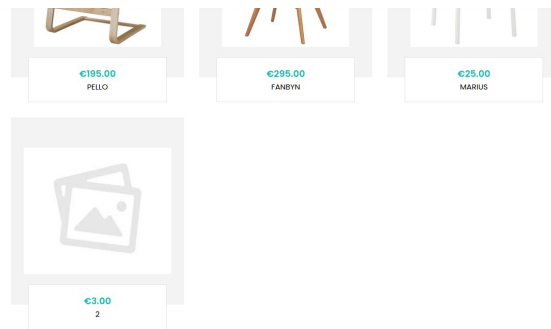
No entanto, se um payload básico tal como

'order by 1; --

for usado nesse campo, não são retornados nenhuns resultados, podendo então o atacante assumir (e corretamente, visto que é isto que está a acontecer) que os caracteres - estão a ser bloqueados, sendo que não pode comentar o resto da query através de -- na sintaxe MySQL. Então uma tentativa para fazer o bypass disto seria criar um payload que faça a UNION com uma que continue a query predefinida, isto é, por exemplo:

'union select 1,2,3,4,5,6,7,8,9,10,11,12 where 1 like '

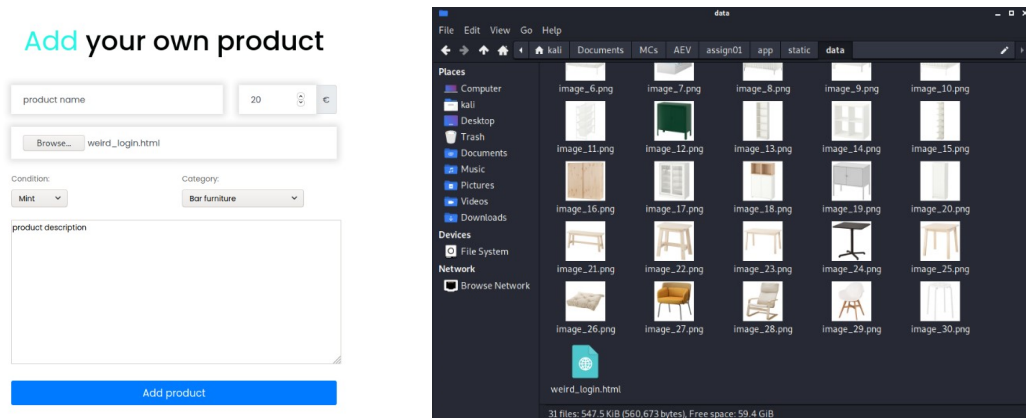
Neste caso, no último item, teríamos a segunda e terceira coluna a ser mostrada ao atacante, o que com mais payloads diferentes mas com a mesma estrutura permite obter informação como nomes de tabelas, enumerar utilizadores, hashes de passwords, etc.



Unrestricted Upload of File with Dangerous Type (CWE-434)

Quando um utilizador se encontra autenticado, existe a possibilidade de adicionar o seu próprio produto ao catálogo da loja online, e isto requer que um form seja preenchido, no qual é pedido uma imagem para ser associada ao produto. No entanto, não existe qualquer tipo de verificação ao tipo de ficheiro que pode ser uploaded e, devido a isto, pode ser feito o upload de qualquer ficheiro.

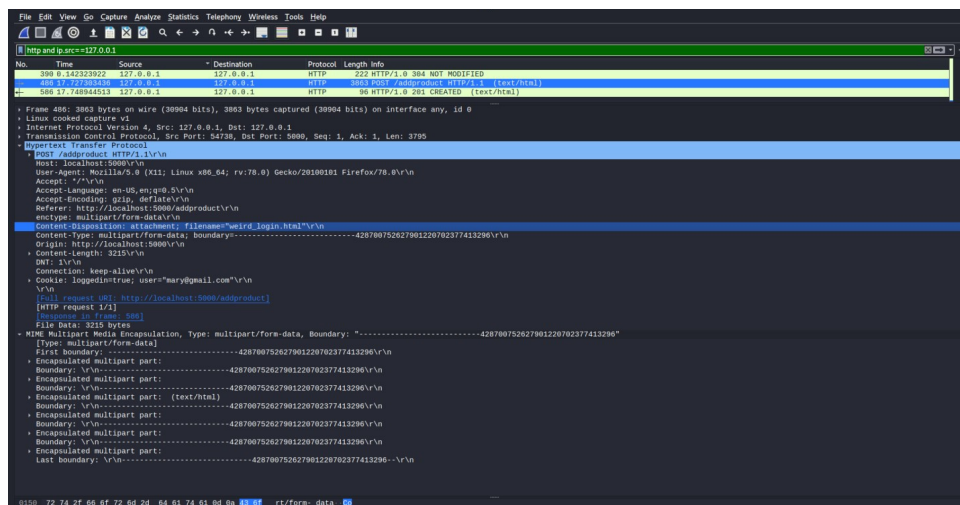
Um atacante pode usar isto a seu favor ao, por exemplo, fazer o upload de script ou de ficheiros html, que em conjunto com path transversal (explicado na subsecção seguinte) lhe permite mudar o html que é mostrado aos utilizadores numa página em específico. Neste caso foi feito o upload de um ficheiro weird_login.html, sem qualquer problema e este pode ser usado a seguir para enviar os dados de um login para um servidor (criado através de netcat, por exemplo) por forma a que credenciais possam ser roubadas.



Path Transversal (CWE-22)

Como foi mencionado na subseção anterior, ao fazer o upload de um ficheiro, é possível fazer path transversal por forma a que o ficheiro, em vez de ser armazenado no diretório em que é suposto, ser guardado noutra, uma vez que o nome completo do ficheiro (presente na header do request) não é validado nem verificado, e este path transversal pode ser feito através da presença de `../` no mesmo.

O atacante consegue ter noção disto através da captação de um request através do Wireshark ao adicionar o seu próprio produto, onde na header (*Content-Disposition: filename*) se encontra o nome do ficheiro a ser guardado, e no corpo os dados do formulário preenchido assim como os dados presentes no ficheiro.



Assumindo que a webapp tem a estrutura genérica de uma aplicação em Flask, tal que:

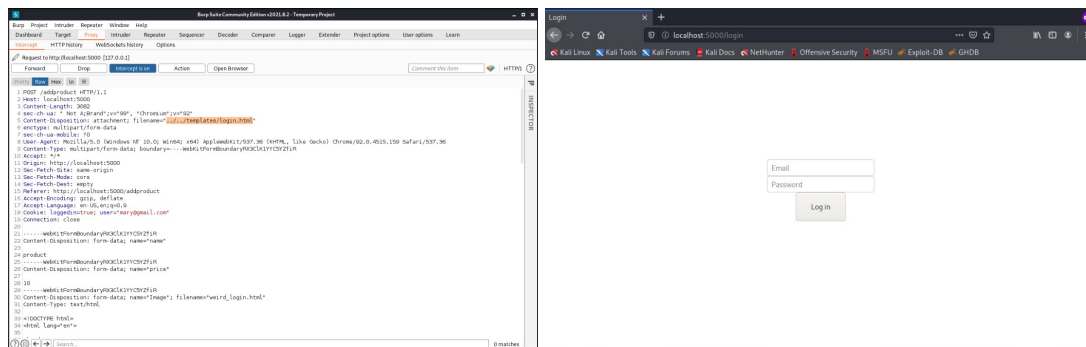
```

app/
├── templates/
│   └── login.html (most likely)
├── static/
│   ├── css/
│   ├── js/
│   └── [dir for user data]/
└── [main file].py

```

então, se alterar a header para o filename ser `../templates/login.html` (que neste caso foi feito usando burp suite) conseguimos fazer um overwrite do template usado para a página que é mostrada no login. Um exemplo de um possível ataque seria usar o ficheiro `weird_login.html` (mencionado acima), que contém um formulário de autenticação, mas quando é submetido, em vez de ser enviada uma request ao servidor, é enviado para um servidor maligno, expondo as credenciais ao atacante.

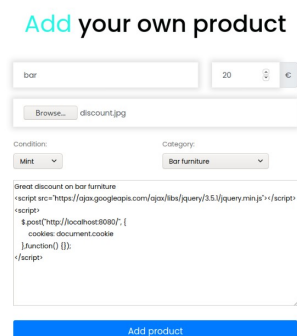
Assignment 1: Vulnerable app



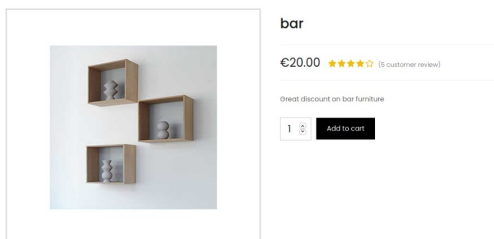
Este método também pode ser usado, não só para credenciais do website, mas possivelmente para informação de cartão de crédito ou credenciais de outras plataformas, sob o pretexto de ser preciso confirmar/validar a conta ou identidade de um utilizador.

Cross-site Scripting (CWE-79)

Por último, quando se adiciona um produto ao catálogo do website, o campo da descrição do mesmo é vulnerável a stored Cross-site Scripting (XSS). Desta forma, um atacante consegue injetar código ao fazer a inscrição de um produto e, cada vez que um utilizador abrir a página de detalhes desse mesmo produto, o código vai ser executado.



Um possível ataque seria injetar código javascript que, quando a página é carregada, faz um POST request cujo corpo contém todos os cookies do utilizador. Isto pode ser feito com o payload presente em *storedXSS_payload.js* e criando um servidor no port 8080 através de netcat, por exemplo. Desta forma, cada vez que um utilizador visse aquela mobília de bar em específico, um request ia ser feito, dando discluse a todos os seus cookies, quer referentes a esta aplicação quer a outras plataformas. Este tipo de ataques pode também ser efetuado para correr outro tipos de código maligno em client-side assim como roubar credenciais.

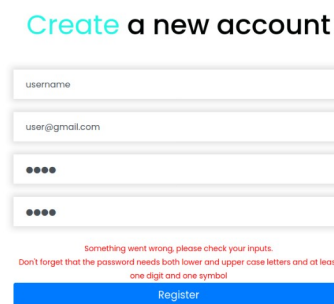


Fixes implementados

Esta seção foca-se na correção e mitigação das vulnerabilidades apresentadas anteriormente, assim como pedaços de código alterados ou resultados, uma vez que as devidas proteções estão no sítio. A maior parte das vulnerabilidades foi corrigida de forma direta, visto serem, no fundo, erros simples de faltas de verificação que programadores inexperientes ou menos focados nos aspectos da segurança geral do sistema poderiam ter feito.

Weak Password Requirements (CWE-521)

Por forma a resolver este problema com eficácia, o mais simples a fazer é, no servidor, antes de criar um utilizador, garantir que a password escolhida tem uma alta entropia. Isto é feito através do seu tamanho (tem que ter mais que 8 caracteres), assim como a presença de caracteres alfanuméricos, maiúsculos e minúsculos, e pelo menos um símbolo. Se a password que o utilizador escolheu não está em conformidade com estes padrões, uma mensagem de erro é levantada na interface de registo. Além disso, para que a enumeração de e-mails através deste form não seja possível, foram aplicadas mensagens de erro mais genéricas.



Improper Authentication (CWE-287)

Por forma a que a autenticação através de cookies não esteja sujeita a ser posta em causa caso o utilizador os altere, sempre que um cookie é gerado, este é assinado usando *SHA-3* (esta foi uma escolha consciente, visto que as famílias *SHA-1* e *SHA-2* podem ser sujeitas a ataques de *Length Extension*). Assim, um cookie é criado no formato *isLoggedIn={},user={},signature={}*, e cada vez que é necessário utilizar este cookie, é feito o hash dos dados e comparado com a signature presente no cookie. Desta forma, o ataque descrito em cima torna-se invalidado e não é possível fazer user impersonation através de modificação manual de cookies.

```
def __generate_cookie(email):
    hash = hashlib.sha3_256()
    cookie = f'loggedin=true,user={email}'
    hash.update(bytes(config['secret'], 'utf-8'))
    hash.update(bytes(cookie, 'utf-8'))

    return f'{cookie},signature={hash.hexdigest()}'

def __verify_cookie(cookie):
    loggedin, user, signature = cookie.split(',')
    hash = hashlib.sha3_256()
    hash.update(bytes(config['secret'], 'utf-8'))
    hash.update(bytes(f'loggedin={loggedin},user={user}', 'utf-8'))
    signature_to_verify = f'signature={hash.hexdigest()}'

    return compare_digest(signature, signature_to_verify)
```


SQL Injection (CWE-89)

Para mitigar SQL Injection existe a necessidade de sanitizar e validar os inputs vindos do utilizador, no entanto isto é um processo extenso em que se deve ter atenção a detalhes como encodings diferentes de URL e todos os possíveis cenários. No entanto, uma maneira mais simples de mitigar esta vulnerabilidade seria fazer as queries à base de dados de forma parametrizada, fazendo a clara distinção do que faz parte da query e o que são variáveis que vêm do input do utilizador. Desta maneira o método *search_products()* foi substituído para refletir isto mesmo.

```
def search_products(mysql, q):
    if not mysql:
        return []

    try:
        cursor = mysql.get_db().cursor()
        query = "SELECT * from (PRODUCT LEFT JOIN CATEGORY ON PRODUCT.category_id = CATEGORY.id) LEFT JOIN " + \
            "furniture.CONDITION ON PRODUCT.condition_id = furniture.CONDITION.id where PRODUCT.name LIKE '%s%';"
        cursor.execute(query, (q))
        _products = cursor.fetchall()
        cursor.close()
        return [Product(_p) for _p in _products]
    except Exception as err:
        print(err)
        return []
```

Com esta nova query, se um atacante tentar usar payloads como os previstos na secção acima, vai ser levantada uma exceção relativa à sintaxe do MySQL ao executar a query, e o utilizador vai receber uma lista vazia de produtos.

Unrestricted Upload of File with Dangerous Type (CWE-434)

Para resolver esta vulnerabilidade, foi primeiro alterado o HTML por forma a refletir um input de ficheiros que apenas permite imagens através do atributo *accept* da tag *<input>*. Para além disso, no servidor, foi feito um check adicional caso o pedido não utilize a interface gráfica, através de um set predefinido de extensões possíveis que o ficheiro pode ter.

```
if _filename == '':
    return invalid_resp

file_ext = os.path.splitext(_filename)[1]
if file_ext not in app.config['UPLOAD_EXTENSIONS']:
    return invalid_resp
```

Path Transversal (CWE-22)

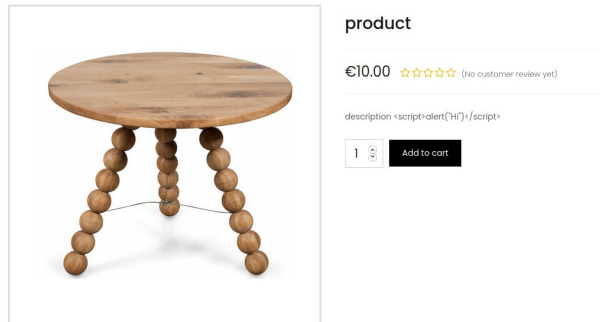
Adicionando à limpeza do nome do ficheiro na header feita acima, poderia também ser adicionado a impossibilidade de este conter caracteres como / ou \ para evitar situações de path transversal indesejadas. No entanto, o que foi feito foi gerar um filename para cada imagem de um produto que precisa de ser gravada através de um *uuid*, que permite a geração de filenames sem que haja colisões devido à probabilidade elevada de serem gerados dois iguais.

```
_stream = request.files.get("Image")

with open(FILE_DIR + uuid.uuid4().hex + "." + file_ext, "wb") as f:
    f.write(_stream.read())
```

Cross-site Scripting (CWE-79)

Por último, para mitigar o XSS nas descrições de produtos, foi adicionado o escape de instruções HTML e foi tido mais cuidado na maneira como os templates das páginas estavam a ser gerados. Desta forma, mesmo que um atacante insira código javascript na descrição de um produto, este vai ser rendered pelo browser como texto e não vai ser feito o parse para instruções de HTML ou javascript.



Conclusão

Foi implementado um site com algumas das vulnerabilidades mais comuns que, como foi dito na introdução deste relatório, muitas vezes resultam de falta de boas práticas de código seguro, falta de experiência na área ou puro desconhecimento que uma implementação específica de uma funcionalidade possa permitir tantos tipos de ataques. No entanto, considera-se que eles foram explorados e mitigados com sucesso, apesar de em geral terem sido mitigação bastante simples de implementar.

É também de mencionar que todos os problemas referidos e analisados neste documento foram os que o grupo tinha noção de que existiam no decorrer do desenvolvimento do projeto, sendo que provavelmente outras tantas vulnerabilidades ainda existam, mesmo na versão fixada da aplicação web.

Referências

- [1] berandal666, 10 million password list: top 10000, Github repository (online),
<https://github.com/berandal666/Passwords>
- [2] Website template, <https://html.design/download/niture-free-furniture-html-template/>