Universidade de Aveiro

Departamento de Electrónica, Telecomunicações e Informática

Análise e Exploração de Vulnerabilidades - 2022-2023

Teste 1

Data: 11 de novembro de 2022

Duração: 90 minutos

Nome: _____

MEC: _____

---

1) [6 pts] Considere o excerto de código da página seguinte:

(a) [2 pts] Enumere e descreva as vulnerabilidades existentes, explique genericamente como podem ser exploradas e descreva qual o impacto mais direto da sua exploração.

(b) [2 pts] Para **duas** das vulnerabilidades, identifique os dados de entrada que permitem a sua exploração.

(c) [2 pts] Para **uma** das vulnerabilidades, explique as alterações necessárias no código para a mitigar. Use código real ou pseudocódigo, desde que com o detalhe suficiente para compreender as ações a tomar.

2) [2 pts] Considerando o conceito de vulnerabilidade e a sua gestão:

(a) [1 pt] Em que consiste o CVSS, qual a sua utilidade e que limitações poderão existir no seu uso para a gestão de aplicações com exposições públicas diferenciadas? Justifique.

(b) [1 pt] Descreva o ciclo de vida da gestão de vulnerabilidades numa entidade.

3) [2 pts] Considerando a família de vulnerabilidades por Injeção:

(a) [1 pt] Em que consiste o conceito de Living of The Land, porque existe, como pode facilitar um ataque deste tipo, e que mitigações devem ser implementadas?

(b) [1 pt] Em que medida o mecanismo CORS e o uso do cabeçalho `Access-Control-Allow-Origin` podem evitar alguns destes ataques?

```python
### Python Flask Health Application providing blood test results
################################################
@app.errorhandler(404)
def page_not_found(e):
    template = f'Oops! Not found: {unquote(request.url)}' # Get REQUEST.URL to help users debug issues
    return render_template_string(template), 404          # Use unquote to show the actual text
                                                          # as it will help debugging

################################################
@app.route('/upload', methods=['POST'])
def put_result():
  if not can_access('put_result'):                        # Check if session exists and user is admin
    abort(401)

  file = request.files['file']                            # Get the filename provided
  file.save(os.path.join(LAB_RES_DIR, file.name))         # Save results
  os.system(f'/usr/local/bin/new_result.sh {file.name}')  # Notify owner that result is available
  return render_template('upload.html')                   # Present information to user

################################################
@app.route('/api/results', methods=['GET'])               # Allows getting a lab result
def get_results(result_id):
  if not can_access('get_result', result_id):             # Only admins or result owners can get the result
    abort(401)                                            # Otherwise abort with HTTP 401

  pname = os.path.join(LAB_RES_DIR, result_id)            # Build path name
  if os.path.exists(pname):                               # Check that it exists
      return send_file(pname, mimetype='application/pdf') # Senf file to user directly as it is quicker
  return render_template('error.html', msg=f'Lab results {pname} not found') # Error. Should not happen

################################################
@app.route('/login', methods=['GET'])
def login(user=None, password=None, domain='health1.com'):# Authenticate users to a given clinic
  cnx = db.connect(user='rwuser', password='qwerty1234',  # Connect to database using ip
                   host=f'db.{domain}', database='users')

  cursor = cnx.cursor()                                   # Create cursor and run query to find products
  cursor.execute(f'SELECT * FROM users WHERE user="%{user}%"')
  cookie = json.dumps(dict(id=-1))
  if cursor.rowcount == 0:                                # Check if user exists. Helping users
    msg = f'Unknown user: {user}'                         # that do not know username and/or password
  else:
    cursor.execute(f'SELECT * FROM users WHERE user="%{user}%" and pass="%{password}%"') # Auth
    if cursor.rowcount == 0:                              # Bad password
      msg = 'Invalid password'
    else:
      row = cursor.fetchall()                             # Get user data
      msg = f'{username} access granted'
      session.append(dict(user=user, isadmin=bool(row[3])))  # Store user in session list for quicker authz
      cookie = json.dumps(dict(id=len(session)))             # Build cookie

  resp = make_response(render_template('login.html', msg=unquote(msg))) # Build answer with message
  resp.set_cookie('auth', b2a_base64(hashlib.md5(cookie)))              # encrypt cookie

  return resp;
```