intro to web part 1

Intro to Web - Part 1 CTF Writeup

This is a writeup for the "Intro to Web - Part 1" challenge from GPN CTF, which featured 5 stages with different web vulnerabilities. Here's how I solved each stage:

Challenge Overview

The challenge consisted of a Flask note-taking application with multiple vulnerability stages:

- **Stage 1**: File Path Traversal
- **Stage 2**: XSS to steal admin note content
- **Stage 3**: XSS to steal FLAG cookie
- **Stage 4**: Admin privilege escalation to access development routes
- **Stage 5**: Pickle deserialization RCE

Stage 1: File Path Traversal - `GPNCTF{jUS7_L34k_Al1_7he_tHIn6S}`

Vulnerability Analysis

The `read_file` function in main.py had a flawed regex validation:

```
def read_file(path):
    # Prevent reading files outside the allowed directory (.img/).
    if not re.search(r'^\.\w+.*$', str(os.path.relpath(path))):
        return ""
    try:
        with open(path, 'rb') as f:
            content = f.read()
            base64_content = base64.b64encode(content).decode('utf-8')
            return base64_content
    except Exception:
        return ""
```

The Exploit

The key insight was that `os.path.relpath()` normalizes the path **before** the regex check. This means:

1. Input: `.img/../.env`

2. After `os.path.relpath()`: `.env`
3. Regex check: Does `.env` match `^\.\w+.*$`? **YES!**

Attack Steps

1. **Created a note** with malicious `image_path` parameter
2. **Used payload**: `.img/../.env` (URL encoded as `.img%2F..%2F.env`)
3. **The path traversal worked** because the normalized path `.env` still matched the regex
4. **Extracted the flag** from the base64-encoded image data in the HTML source

The `.env` file contained:

```
FLASK_APP_SECRET_KEY=c266d5581033896f391126416198ac5c026c24b1d4c38c8c9a3a8
5511e1ce423271390188304e6b3e772373afb43af033c

FLAG_STAGE_1=GPNCTF{jUS7_L34k_Al1_7he_tHIn6S}
```

Key Insights

Stage 1 Success Factors

1. **Path Normalization Bypass**: Understanding that `os.path.relpath()` processes the path before regex validation
2. **Regex Pattern Analysis**: The pattern `^\.\w+.*$` allowed paths like `.env` after normalization
3. **Base64 Decoding**: Recognizing that the file contents were base64-encoded in the image source

Authentication Vulnerability

The login function had a critical flaw - it didn't actually verify passwords:

python

`session['user'] = { 'username': username, 'role': 'admin' if hashlib.sha512(password.encode()).hexdigest() == ADMIN_PASSWORD_HASH else 'user' }`

This meant ANY username with ANY password would create a session, with role assignment based solely on password hash matching.

Lessons Learned

1. **Path Traversal**: Always validate file paths after normalization, not before

2. **Regex Validation**: Be careful with regex patterns that can be bypassed through path manipulation
3. **Authentication**: Proper password verification is crucial - don't just check hashes for role assignment

Tools Used

- **Burp Suite**: For intercepting and modifying HTTP requests
- **curl**: For automated requests and cookie handling

This challenge demonstrated a progression from simple file traversal to more complex web application vulnerabilities, making it an excellent introduction to web security testing techniques.

Final Flag for Stage 1: `GPNCTF{jUS7_L34k_Al1_7he_tHIn6S}`

Note: While I successfully completed Stage 1 and discovered the admin password, the remaining stages (2-5) involving XSS, privilege escalation, and pickle deserialization were not completed during this session due to the complexity of bypassing the HTML encoding protections for the XSS-based attacks.