Networks and Systems Security
Week 05
Web Security

Aims of the Seminar

Welcome to the Web Security workshop. In this workshop, we'll dive into the essential principles Web application vulnerability scanning. Web application vulnerability scanning is the process of using automated tools to identify security weaknesses in web applications. These scanners work by sending a variety of requests to a web application and analysing the responses to find potential vulnerabilities. This is a crucial part of a web security audit, as it helps to identify and fix security holes before they can be exploited by attackers.

Feel free to discuss your work with peers, or with any member of the teaching staff.

Reminder

We encourage you to discuss the content of the workshop with the delivery team and any findings you gather from the session.

Workshops are not isolated, if you have questions from previous weeks, or lecture content, please come and talk to us.

Exercises herein represent an example of what to do; feel free to expand upon this.

Helpful Resources

Wapiti Docs: https://wapiti-scanner.github.io/

OWASP Juice Shop: https://owasp.org/www-project-juice-shop/

Google Gruyere: https://google-gruyere.appspot.com/

Full Wapiti GitHub: https://github.com/wapiti-scanner/wapiti

Setting up

```
Install Wapiti

Run this cell to install Wapiti in the notebook environment. The PyPI package
name used by Wapiti is `wapiti3`.
```

Install the python libraries library

```
!pip install --upgrade pip
!pip install wapiti3

# Verify installation
!wapiti --version
```

```
**Note:** On some systems (Windows), running Wapiti may require WSL or an
environment with Python 3.12+. If `wapiti` is not found after installation,
check your PATH or run within the virtual environment.
```

⚠ Ethical Reminder ⚠

⚠ Never scan a website without explicit permission. ⚠

The apps below are designed for security testing—they are legal, safe, and maintained for education.
Do not scan any other site during this workshop.

Wapiti

Wapiti is a command-line web application vulnerability scanner written in Python. It performs black-box testing, meaning it doesn't require access to the source code. Instead, it crawls web applications to find URLs and injects payloads to test for vulnerabilities such as:

- Cross-Site Scripting (XSS)
- SQL Injection
- Command Injection
- File Inclusion
- Insecure File Uploads
- Server-Side Request Forgery (SSRF)

Target:

Google Gruyere (http://google-gruyere.appspot.com)

Important: For Google Gruyere, you must first start your own instance at https://google-gruyere.appspot.com.

Click **Create Instance** and note your unique ID (e.g., 789012). Your URL becomes:

https://google-gruyere.appspot.com/789012/

Other Potential targets:

Platform	URL	Purpose
OWASP Juice Shop	`https://juice-shop.herokuapp.com`	Modern vulnerable app (XSS, SQLi, etc.)
Google Gruyere	`https://google-gruyere.appspot.com/123456/` *(replace `123456` with your instance ID)*	Classic Google training app
PentesterLab	`https://54.80.249.209/` (public demo)	Simple vulnerable endpoints
Hack The Box Academy	Use labs from [HTB Academy](https://academy.hackthebox.com/) (requires free account)	Guided vulnerable scenarios

Script example:

```
import os
from IPython.display import display, HTML
import glob
import pathlib

# Base 'out' name you passed to Wapiti (-o). Update if you used a different
name.
out_name = "juice_wapiti_report.html"

def find_html_report(out):
    # If out is a file and ends with .html, use it
    if os.path.isfile(out) and out.lower().endswith('.html'):
        return os.path.abspath(out)
    # If out is a directory, search for the newest .html inside it
    if os.path.isdir(out):
        matches = glob.glob(os.path.join(out, "*.html"))
        if matches:
            # choose the most recent file
            matches.sort(key=os.path.getmtime, reverse=True)
            return os.path.abspath(matches[0])
    # If out doesn't exist as provided, search for any html file pattern that
looks like wapiti output
    candidates = glob.glob(out + "*/*.html") + glob.glob(out + "*.html")
    if candidates:
        candidates.sort(key=os.path.getmtime, reverse=True)
        return os.path.abspath(candidates[0])
    return None

report_path = find_html_report(out_name)

if report_path:
    print("Report file found:", report_path)
    try:
        with open(report_path, 'r', encoding='utf-8') as f:
            html = f.read()
        display(HTML(html))
    except Exception as e:
        print("Could not render HTML inline (error):", e)
        print("Open the file in your browser instead:", report_path)
else:
    print("No HTML report found. Check that the scan produced the report and
that 'out_name' matches the -o argument passed to Wapiti.")
```

Happy scanning!