Web2Doc2 — Writeup

Flag

```
ENO{weasy_pr1nt_can_h4v3_f1l3s_1n_PDF_att4chments!}
```

Resumen

El servicio es el mismo que en Web2Doc v1 (convertidor URL → PDF en Flask), pero **sin** el endpoint `/admin/flag`. El objetivo sigue siendo leer el contenido de `/flag.txt` en el servidor.

- El backend usa **WeasyPrint** para generar el PDF (se ve en el metadata del PDF: `Producer: WeasyPrint 68.1`).
- No podemos pasar `file:///flag.txt` como URL directamente: el backend hace un fetch de la URL (p. ej. con `requests`), que falla o bloquea el protocolo `file://`.
- La idea es que el servidor **sí** cargue una URL pública que devuelve **HTML**. Ese HTML puede contener referencias a recursos que WeasyPrint resuelve **en el servidor** (en el mismo host donde corre el convertidor). Ahí entra el abuso de **adjuntos PDF**.

Vulnerabilidad: adjuntos con `file://` en WeasyPrint

En HTML, WeasyPrint soporta **adjuntos** al PDF mediante:

```
<link rel="attachment" href="URL" title="descripción">
```

Si `href` es una URL que el **url_fetcher** de WeasyPrint puede resolver, el contenido se incrusta como archivo adjunto en el PDF generado. En el reto, al procesar HTML obtenido desde una URL pública (p. ej. Litterbox), el motor sigue resolviendo `file://` para esos adjuntos en el contexto del servidor, por lo que puede leerse `/flag.txt` y terminar embebido en el PDF.

(CVE-2024-28184 afectaba a WeasyPrint 61.0–61.1 y permitía adjuntos desde archivos/URLs arbitrarios incluso con `url_fetcher` restringido; en el reto, el comportamiento observado es que un adjunto con `file:///flag.txt` sigue siendo resuelto y embebido.)

Pasos de la explotación

1. **HTML malicioso**
   Crear una página que declare un adjunto apuntando al archivo de la flag:

```
<link rel="attachment" href="file:///flag.txt" title="flag">
```

2. **Alojar el HTML en una URL pública**
   Subir ese HTML a un servicio accesible desde el servidor del CTF (p. ej. **Litterbox**, Gist + raw.githack, etc.) y obtener una URL (p. ej. `https://litter.catbox.moe/xxxxx.html`).
3. **Convertir a PDF en el reto**
   En la web del reto, resolver el captcha y enviar en `/convert` la URL del paso 2. El servidor:
   - Descarga ese HTML.
   - Se lo pasa a WeasyPrint.
   - WeasyPrint genera el PDF y, al procesar el `<link rel="attachment" href="file:///flag.txt">`, lee `/flag.txt` en el servidor y lo incluye como archivo embebido en el PDF.
4. **Extraer la flag del PDF**
   Descargar el PDF devuelto y extraer el contenido del adjunto (p. ej. descomprimiendo los streams que corresponden al `EmbeddedFile` y buscando la cadena `ENO{...}`).

Automatización

El script `solve.py`:

- Genera el HTML con `<link rel="attachment" href="file:///flag.txt">`.
- Lo sube a Litterbox y obtiene la URL.
- Obtiene el captcha de la página del reto.
- Llama a `/convert` con esa URL y el captcha.
- Lee el PDF devuelto, localiza el stream del adjunto (FlateDecode) y extrae la flag.

Ejecución: `pip install requests && python3 solve.py`.

Resumen técnico

Concepto	Detalle
Objetivo	Leer `/flag.txt` en el servidor (sin endpoint `/admin/flag`).
Vector	HTML público con `<link rel="attachment" href="file:///flag.txt">`.
Motor	WeasyPrint (PDF); el backend obtiene el HTML por HTTP.
Abuso	WeasyPrint resuelve `file://` para adjuntos en el contexto del servidor.
Exfiltración	Contenido de `/flag.txt` embebido en el PDF; se extrae del stream.