Análisis de documentos maliciosos – Parte 02 – Documentos PDF

Español

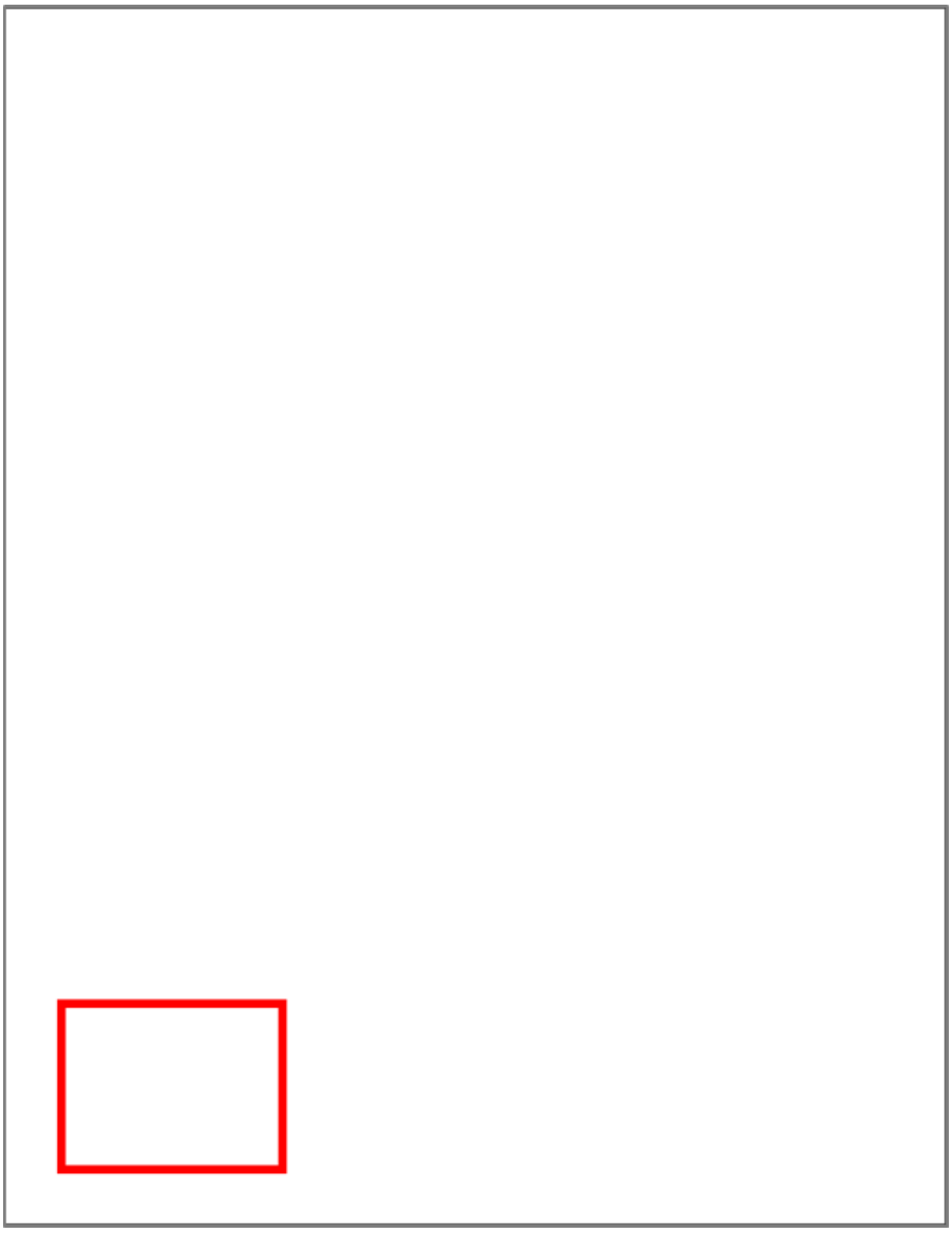
Después de la visión general de la gestión de máquinas virtuales y la instalación de Remnux como entorno para analizar artefactos sospechosos, exploraremos los archivos PDF en términos de formato y formas en que pueden ser utilizados para perjudicar a los usuarios.

¿Qué es un PDF?

Cuando abrimos un archivo PDF, utilizamos un software específico que renderiza su contenido en forma legible. Sin embargo, como muchos otros tipos de archivos, los PDF se elaboran utilizando una combinación de texto plano normal y datos binarios (para imágenes y otros elementos que puedan requerirlo), por ejemplo, el siguiente texto renderiza el archivo PDF que se muestra a continuación:

```
%PDF-1.4
1 0 obj
<<
        /Length 51
>>
stream
1 0 0 RG
5 w
36 144 m
180 144 l
180 36 l
36 36 l
s
endstream
endobj
2 0 obj
<<
        /Type /Catalog
        /Pages 3 0 R
>>
endobj
3 0 obj
<<
        /Type /Pages
        /Kids [4 0 R ]
        /Count 1
>>
endobj
4 0 obj
```

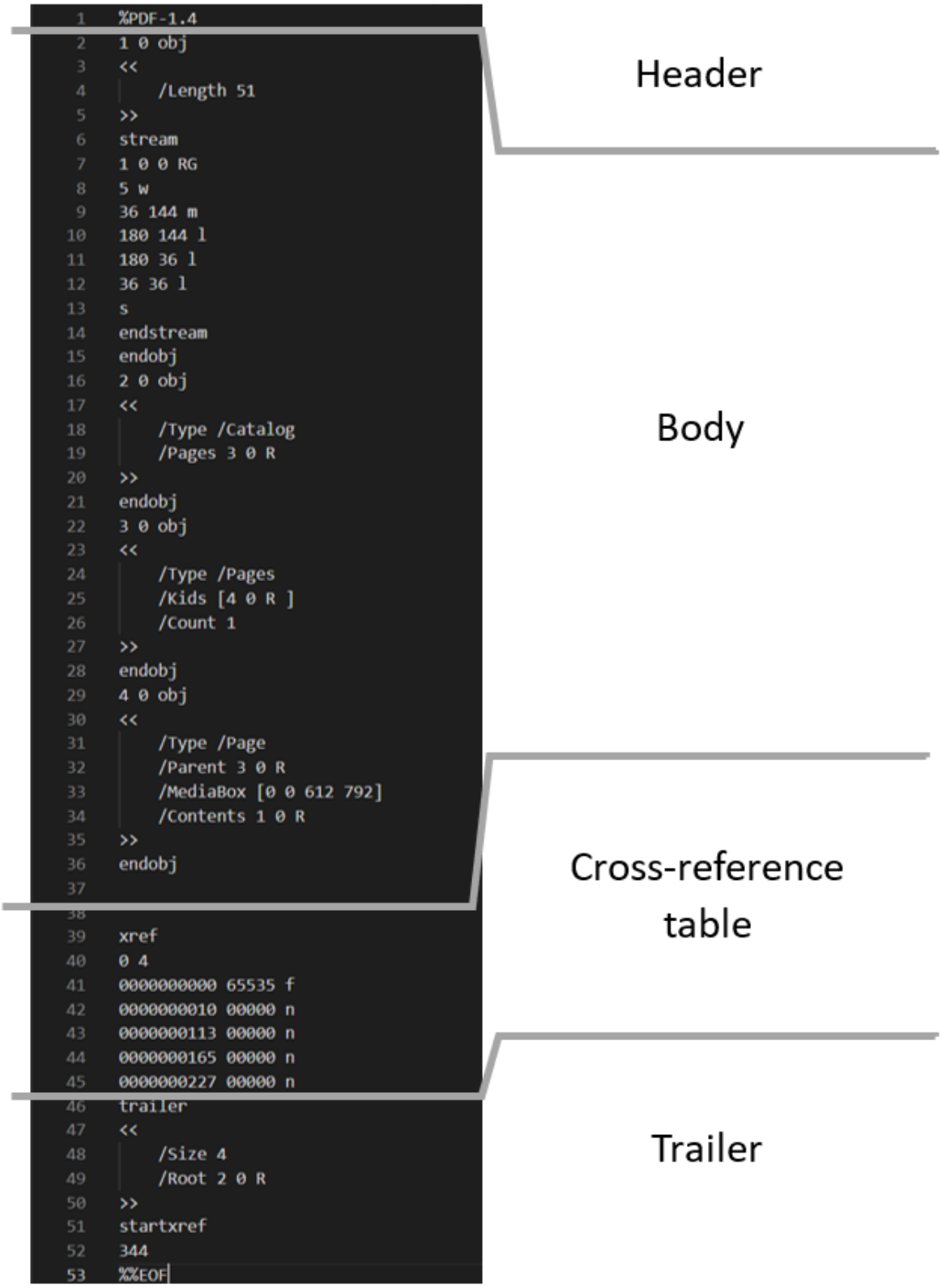
```
<<
	/Type /Page
	/Parent 3 0 R
	/MediaBox [0 0 612 792]
	/Contents 1 0 R
>>
endobj


xref
0 4
0000000000 65535 f
0000000010 00000 n
0000000113 00000 n
0000000165 00000 n
0000000227 00000 n
trailer
<<
	/Size 4
	/Root 2 0 R
>>
startxref
344
%%EOF
```



Tomado de "How to create a simple PDF file" de Callas Software

Estructura del archivo

Partiendo de este ejemplo, podemos ver la estructura estándar de cualquier archivo PDF:



Encabezado: Contiene la versión del protocolo con el que fue elaborado el archivo, para indicar al programa lector cómo leer el resto de la estructura y renderizar todos sus elementos.

Cuerpo: Aquí estarán todos los objetos que conforman el archivo PDF, páginas, imágenes, texto, fuentes, etc. Incluso código y acciones automatizadas si el archivo las tiene.

Tabla de referencias cruzadas: En ella encontraremos una lista de todos los objetos del documento para un fácil acceso y sus respectivos lugares dentro del archivo. Esto es parecido a un "índice" pero para que lo lea el software lector de PDF. Si en algún momento el lector necesita renderizar un objeto específico (por ejemplo, si nos desplazamos a una página cualquiera en un documento grande), el software lector verá qué página debería renderizar, y la buscará en esta tabla para localizar los elementos respectivos en el cuerpo para cargarlos en la pantalla.

Tráiler: Aquí encontraremos el lugar en el documento de la tabla de referencia cruzada y otra información útil, como el número de objetos de la tabla de referencias cruzadas (para verificar, por ejemplo, que el archivo no está corrupto), el objeto raíz del documento e información de encriptación si corresponde. Por diseño, los lectores de PDF empiezan a leer los documentos desde el final, donde pueden encontrar rápidamente el objeto raíz y la tabla de referencias cruzadas para comenzar a mostrar el contenido.

Comprender los objetos PDF

Por la forma en que están estructurados los PDF, la sección más interesante en la cual profundizar es el cuerpo, porque allí se encuentra todo lo que podemos ver y con lo que podemos interactuar (texto, páginas, imágenes, formularios, código, etc.). Todos estos elementos se llaman objetos, y hay una amplia variedad de ellos según las especificaciones. Por ejemplo, en el caso anterior, el objeto

```
4 0 obj
<<
        /Type /Page
        /Parent 3 0 R
        /MediaBox [0 0 612 792]
        /Contents 1 0 R
>>
Endobj
```

es una página con el id 4, tiene un elemento padre con el id 3, y contiene como hijo el elemento con el id 1 (el rectángulo rojo). Mostrar la forma exacta en que cada objeto está escrito está fuera del alcance de este material, sin embargo, hay algunos tipos de elementos que podrían ser utilizados con fines maliciosos, y queremos hablar un poco más de ellos.

/OpenAction: Este objeto hace referencia a un conjunto de acciones que se ejecutan cuando se abre el archivo PDF. Esto puede ser utilizado para lanzar una página web que apoye el contenido o con fines de seguimiento, ejecutar código JavaScript, etc. Podría utilizarse para engañar a los usuarios y hacer que ejecuten acciones adicionales que pueden ser perjudiciales o, dependiendo del entorno de la computadora, este objeto puede ejecutar malware directamente.

/AA: Este objeto también incluye una serie de acciones que se activan bajo distintas circunstancias, como visualizar una página, pasar el puntero del ratón por encima de objetos específicos, llenar campos de formularios, etc. Los riesgos asociados son similares a los del objeto /OpenAction, pero con muchos más escenarios de activación.

/JS o /Javascript: Contiene código JavaScript que será ejecutado después de que se active una acción. Este código puede incluir funciones exclusivas de los PDF.

/Launch: Intenta lanzar una aplicación externa en el dispositivo después de que se active una acción. Esto podría utilizarse, por ejemplo, para abrir otros documentos o ejecutar comandos específicos, lo que podría ser perjudicial.

/EmbeddedFile: Permite la inclusión de archivos arbitrarios, desde documentos hasta ejecutables dentro del archivo PDF. Hay antecedentes de archivos PDF benignos que contienen otros archivos dañinos incrustados, como ejecutables de malware o documentos de Microsoft Office con macros maliciosas.

/ObjStm: Contiene información arbitraria que será procesada según la forma en que sea llamado. El uso principal de este objeto es para agrupar muchos objetos y comprimirlos, obteniendo un archivo más pequeño. Sin embargo, también puede ser utilizado para comprimir código malicioso como una forma de ofuscarlo y evitar la detección por parte de los antivirus. Debido a los numerosos casos de uso de este tipo de objetos, asumir su presencia como maliciosa nos llevará a muchos falsos positivos.

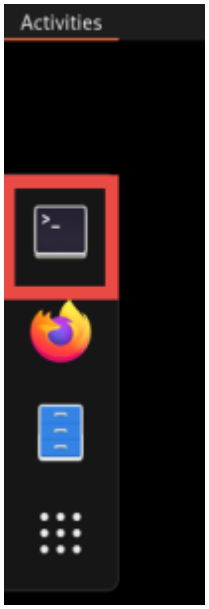
En ataques más elaborados, se puede utilizar una combinación de estos objetos. Por ejemplo, un archivo PDF puede activar una acción que abra un archivo incrustado en el mismo archivo encriptado en un /ObjStm.

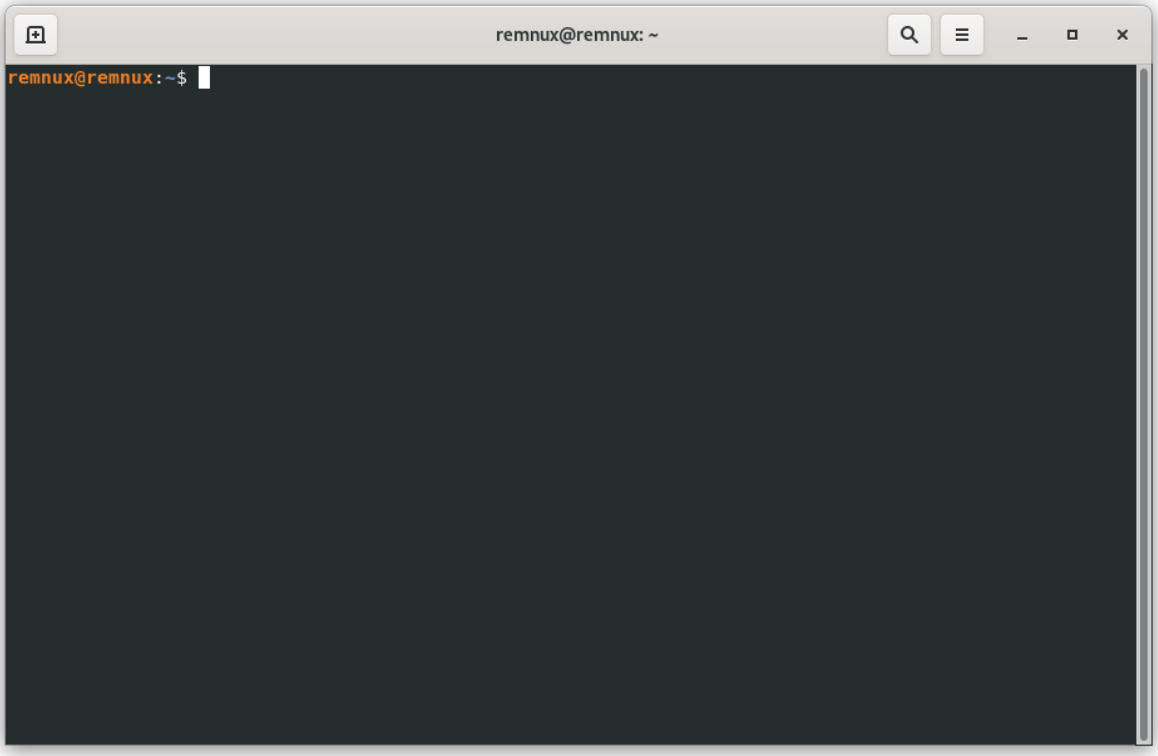
Considerando todo esto, queremos saber si un archivo tiene alguno de estos tipos de objetos como primer paso para ver si un archivo PDF es malicioso, o al menos para asegurarnos de que no lo es.

Introducción a pdfid

Ya que sabemos por dónde empezar a buscar señales de alerta en los archivos PDF que podríamos considerar sospechosos, podemos comenzar a utilizar la herramienta pdfid como primer paso para ver qué tipos de objetos están contenidos en nuestro archivo. Pdfid es parte de un conjunto de herramientas desarrolladas por Didier Stevens para agilizar algunos procesos de análisis en archivos PDF. Estas herramientas se ejecutan mediante la línea de comandos, por lo que se conocen como aplicaciones CLI (Interfaz de línea de comandos). Explicaremos cómo usarlas utilizando la máquina virtual Remnux que configuramos en la parte anterior de este curso.

Para usar pdfid, necesitamos abrir una aplicación de Terminal en nuestra máquina virtual. Cuando iniciamos nuestra VM esta ventana ya debería estar abierta, sin embargo, siempre podemos hacer clic en el menú Actividades en la esquina superior izquierda, y luego en el icono Terminal en el panel izquierdo, como se muestra en la imagen.



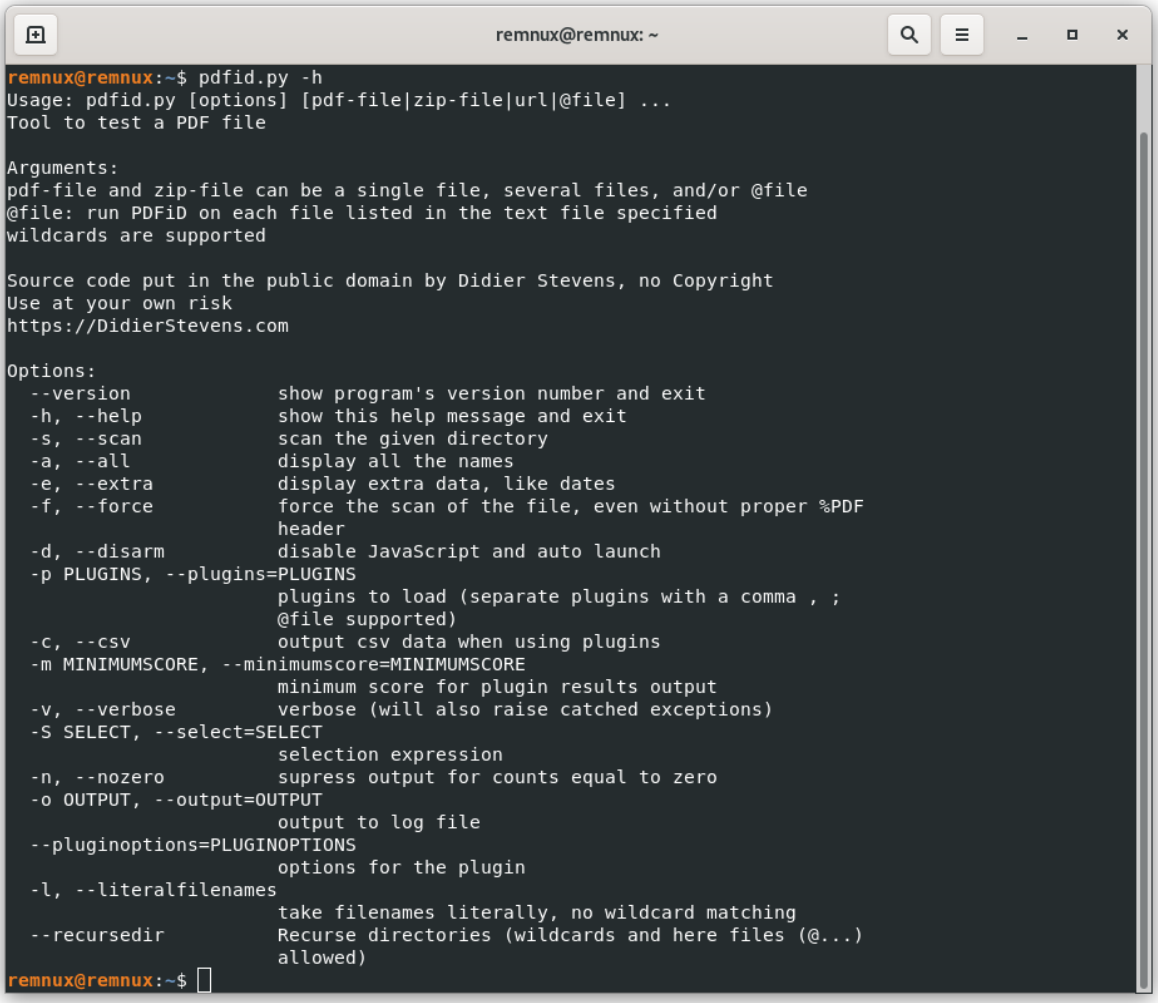


Una vez en la ventana Terminal, podemos empezar a explorar el uso del comando pdfid a través de su ayuda, sólo necesitamos escribir

```
pdfid.py -h
```

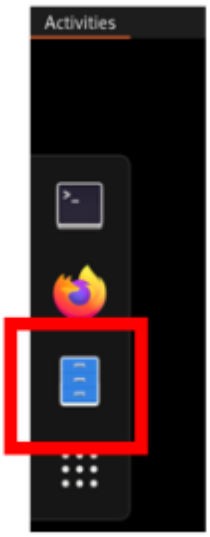
-h es para "ayuda"

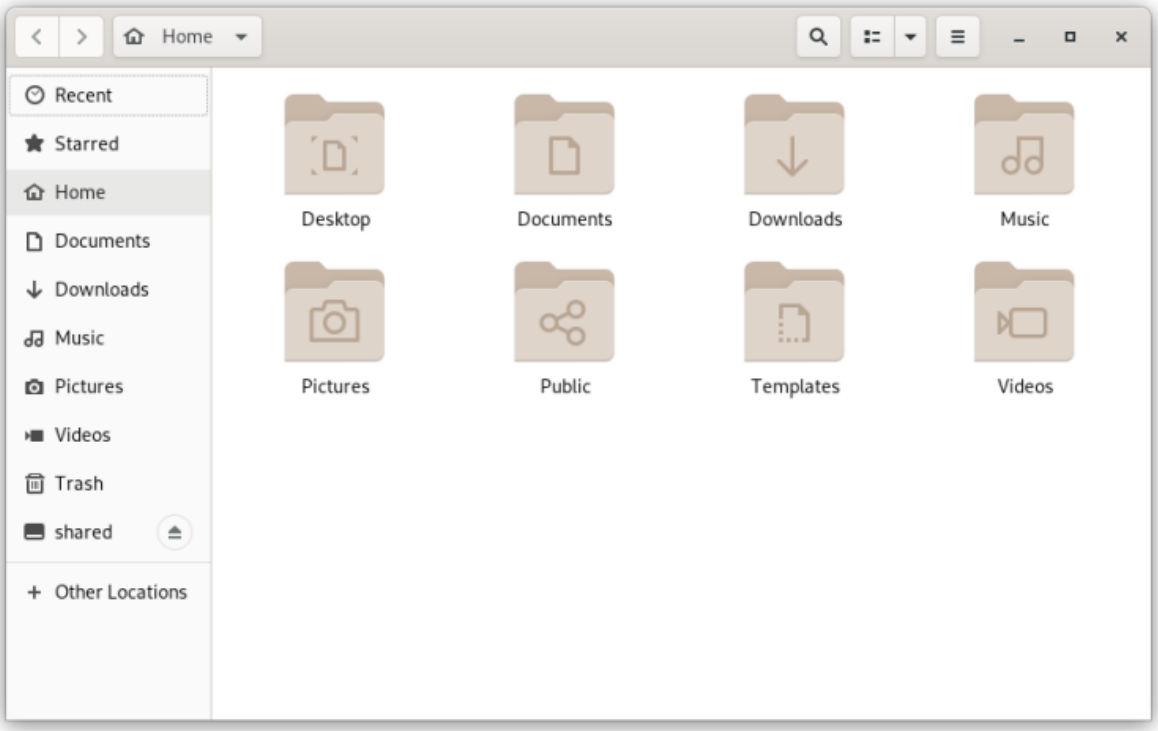
Consejo: siempre podemos utilizar la tecla Tab para autocompletar algunos comandos



Aquí podemos ver una serie de opciones que podemos emplear al usar pdfid. Esto puede parecer un reto para aquellos que son nuevos en el uso de la terminal, sin embargo, generalmente nos apegamos a unas cuantas de estas opciones, y además con un poco de práctica, el proceso se vuelve más rápido y fácil.

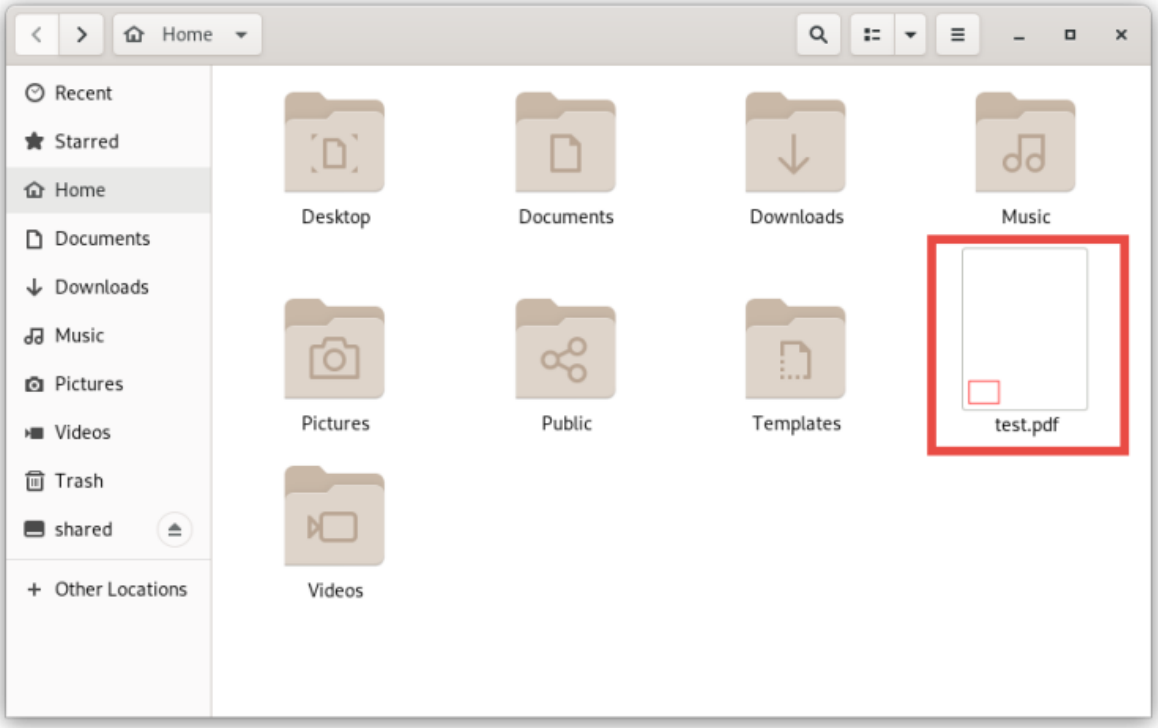
Para analizar nuestro primer archivo, necesitamos tener presente que el comando que ejecutamos en la línea de comandos se ejecuta "desde una carpeta/directorio", por lo que necesitamos saber desde dónde estamos ejecutando el comando, y dónde está ubicado el archivo que queremos analizar. Para darle un poco de contexto, cada vez que abrimos la aplicación de Terminal en Remnux, estamos abriendo una terminal en el directorio Inicio, la misma ubicación que vemos cuando abrimos la aplicación Archivos





Para facilitar las cosas por ahora, podemos colocar nuestros archivos PDF en esta carpeta, de modo que el comando de terminal se ejecute desde el mismo directorio que nuestro archivo PDF.

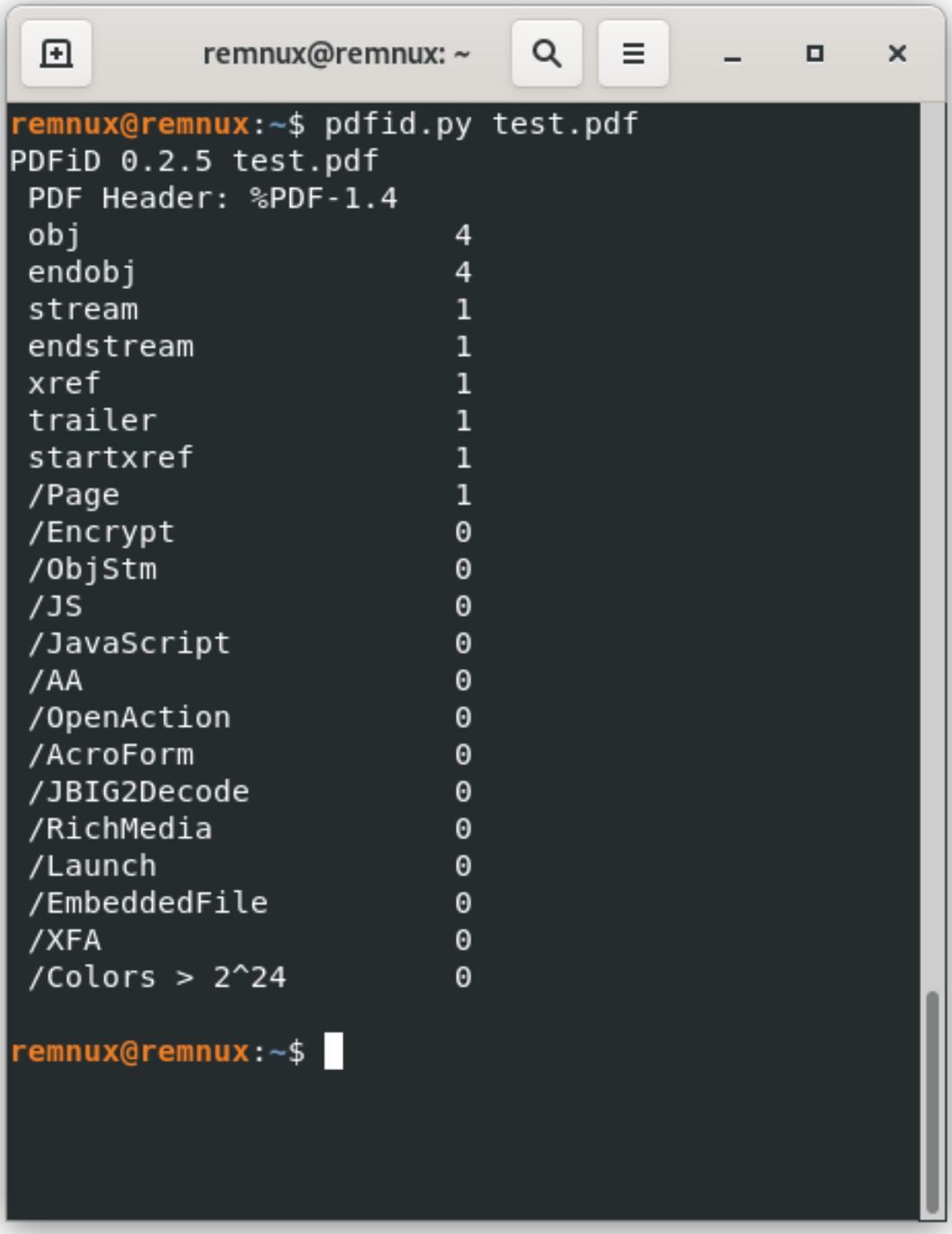
Podemos tomar nuestro archivo de ejemplo mencionado anteriormente y guardarlo como un archivo pdf con la ayuda de un editor de texto en nuestra computadora host y arrastrar y soltar el archivo en el directorio de inicio de Remnux.



Y con esto, podemos ejecutar el siguiente comando en nuestro Terminal:

```
pdfid.py test.pdf
```

Para recibir esta respuesta:



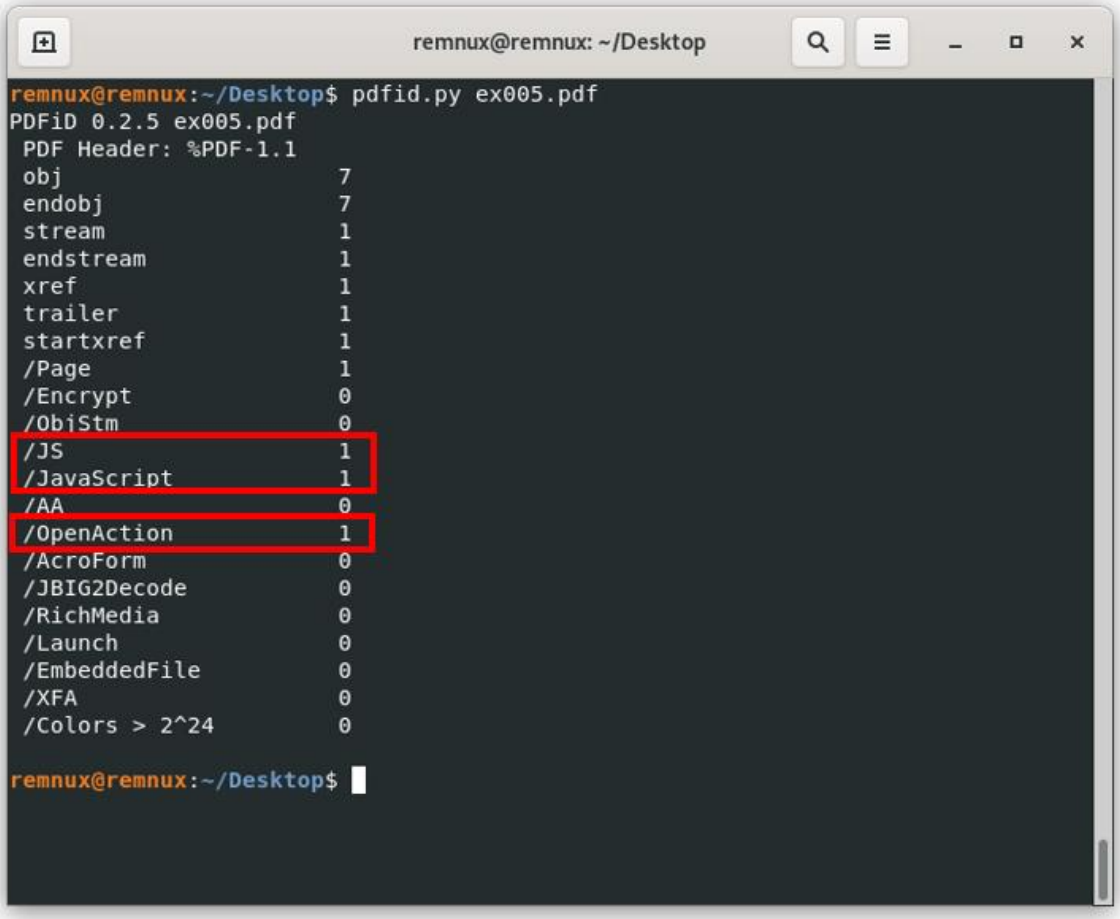
Como podemos verificar, todos los objetos vistos por pdfid coinciden con los que conocemos de la fuente del archivo PDF, y ninguno de ellos parece estar en la lista de objetos sospechosos que describimos anteriormente.

Como mencionamos antes, existen técnicas que los actores maliciosos emplean para evitar la detección fácil de ciertos tipos de objetos. Pdfid intenta mostrar incluso los objetos

ofuscados, sin embargo, en algunos casos menos comunes, puede haber objetos ocultos que requerirán un análisis más profundo para descubrirlos.

Introducción a pdf-parser, ejemplo 1

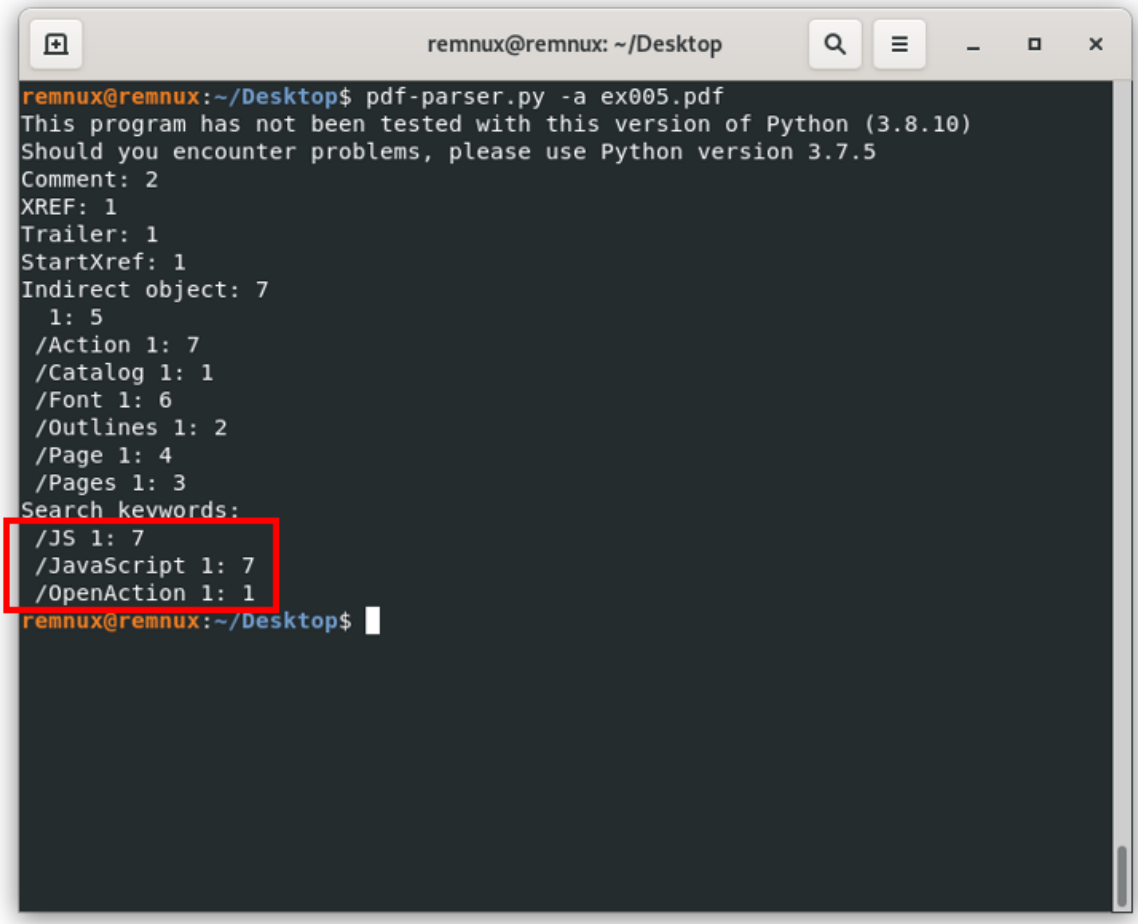
Ahora bien, ¿qué sucede cuando encontramos un archivo PDF con un objeto sospechoso? Imaginemos que tenemos un archivo ex005.pdf que nos da un resultado como este en pdfid:



A partir de aquí, y con la guía anterior de este recurso, sabemos que hay 3 objetos de los tipos /JS, /Javascript y /OpenAction que podría ser interesante revisar, especialmente porque sugieren que el archivo intenta ejecutar alguna acción al abrirlo. Aquí, podemos procesarlo con pdf-parser para saber qué tipo de objeto es cada uno y ver su contenido. Para nuestro archivo de ejemplo, ejecutaremos el siguiente comando:

```
pdf-parser.py -a ex005.pdf
```

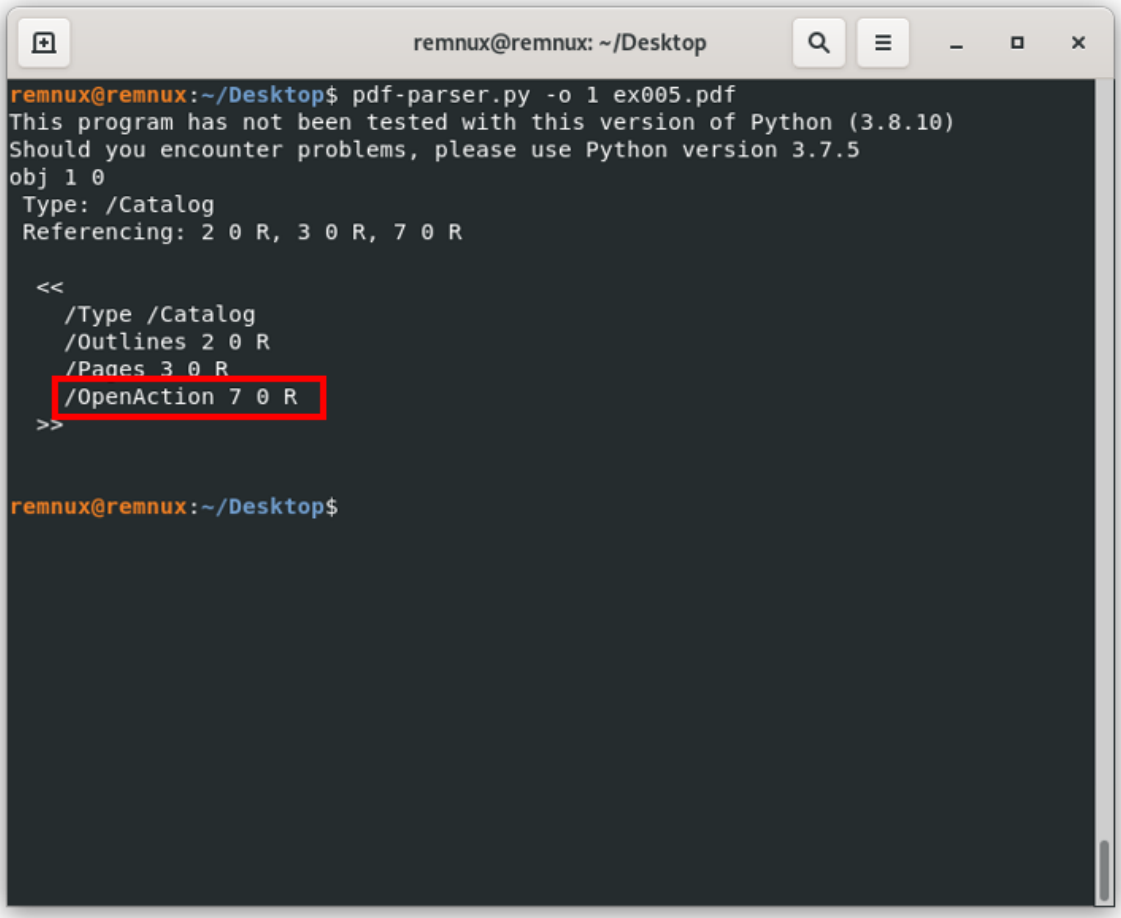
Usamos el argumento -a para ver las estadísticas del archivo. Como referencia adicional, siempre podemos usar pdf-parser.py –help para ver una lista de opciones en pantalla.



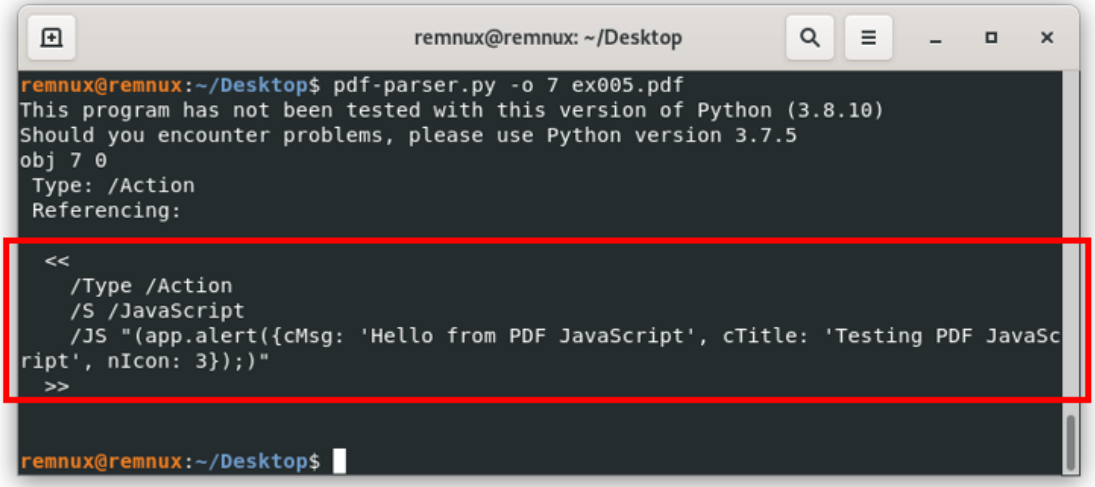
Aquí podemos confirmar que efectivamente tenemos estos tres objetos problemáticos, además de información adicional, podemos ver el id de los objetos para cada tipo. Ahora sabemos que los objetos /JS y /Javascript están en el objeto con id 7, y el objeto /OpenAction está en el objeto con id 1. A continuación, podemos ver el contenido del objeto /OpenAction para ver qué intenta hacer el documento al abrirlo. Para ello, utilizamos el comando:

```
Pdf-parser.py -o 1 ex005.pdf
```

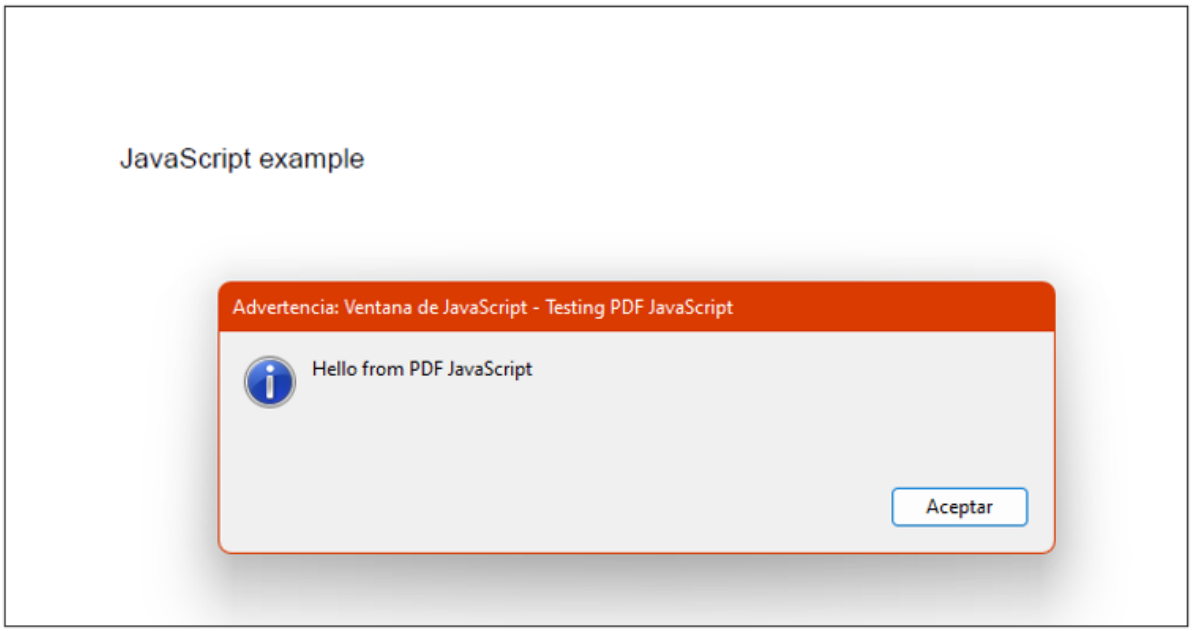
Aquí, el argumento -o se utiliza para indicarle a la herramienta el id del objeto cuyo contenido queremos ver en pantalla:



Aquí, podemos ver la línea "/OpenAction 7 0 R", lo que significa que el contenido real del objeto /OpenAction está en el objeto con id 7, y al abrir el archivo, se llamará o hará referencia a dicho objeto. Repitiendo el proceso para ver el contenido del objeto con id 7 obtenemos:

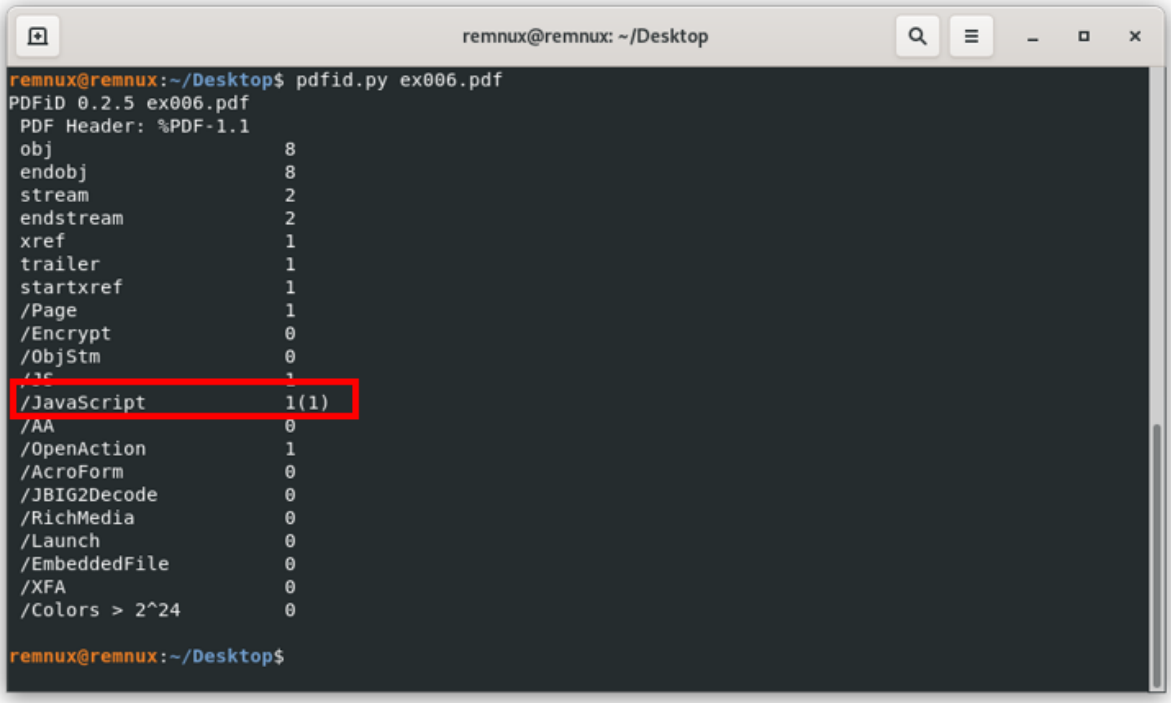


Donde podemos observar que el documento intenta mostrar una alerta o ventana emergente con el mensaje descrito en la terminal, si abrimos el archivo tendrá el siguiente aspecto:

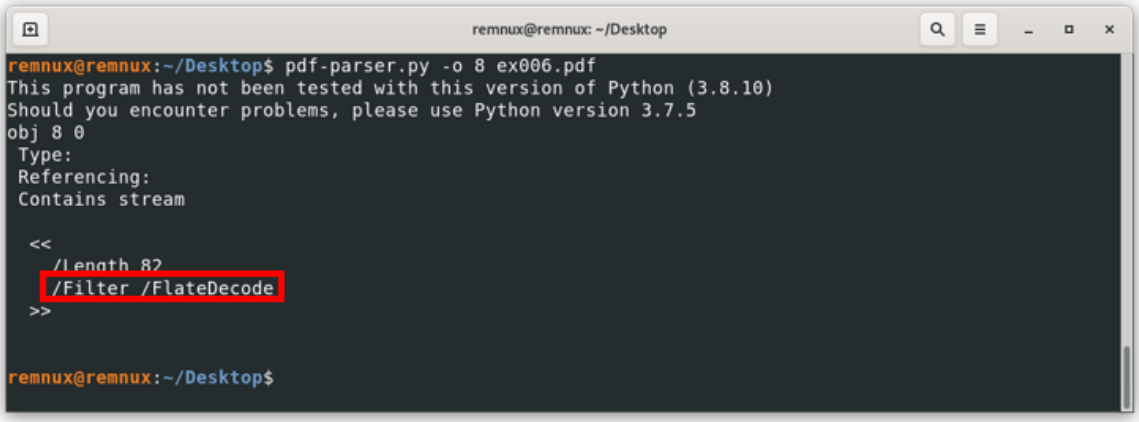


Ejemplo 2

Como mencionamos antes, puede haber archivos donde el contenido sospechoso no sea visible en texto plano. Esto podría deberse a varias razones en casos legítimos, como comprimir fragmentos largos de información para reducir el tamaño del archivo, entre otras. Sin embargo, los archivos maliciosos emplean estas técnicas con fines de ofuscación para evitar ser detectados por el software antivirus y otras soluciones de seguridad. Por ejemplo, si repetimos el flujo de trabajo anterior con el archivo ex006.pdf, veremos que la salida del comando pdfid es la siguiente:



Aquí podemos ver en la línea /JavaScript "1(1)", lo que significa que pdfid detectó un objeto de este tipo, pero ofuscado. Repitiendo el flujo de trabajo que ya conocemos, revisamos el objeto con id 8 (donde reside el código JavaScript) para ver lo siguiente:

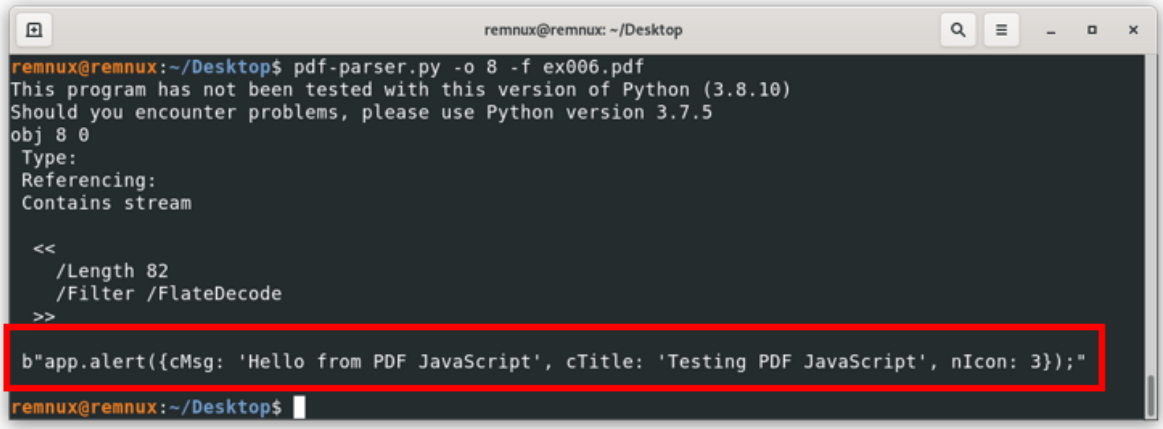


Aquí no podemos ver el código real como en el último ejemplo. En cambio, observamos entre otras cosas la línea "/Filter /FlateDecode". La opción /Filter ejecuta una operación sobre el contenido final de un stream para decodificarlo, entonces /FlateDecode indica la codificación asociada que debe considerarse al decodificar el contenido. Para comprender mejor esto, si abrimos el archivo con un editor de texto y buscamos manualmente este elemento deberíamos ver algo como esto:

```
64
65  8 0 obj
66  <<
67   /Length 82
68   /Filter /FlateDecode
69  >>
70  stream
71  x??K,(??K??I-*??N?-N?RP?H???WH+??U?pqS??J,K?N.?,(Q??QH??,?I?*
72  I-.??K??T?癮?g?`\?i
73  ??/
74  endstream
75  endobj
```

Donde el contenido dentro del cuadro rojo es el contenido real codificado. Para este caso, pdf-parser puede intentar decodificar el contenido real, para ello utilizamos el argumento -f, con lo que terminamos utilizando el comando

```
Pdf-parser.py -o 8 -f ex006.pdf
```

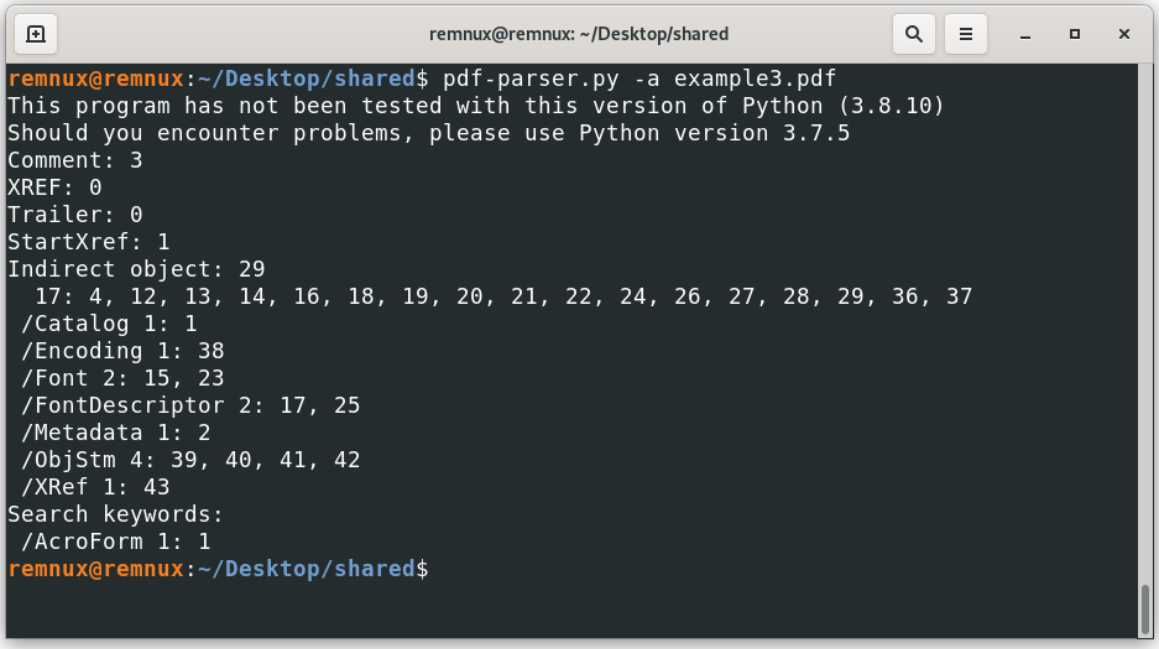


Donde podemos ver el contenido real del objeto que será renderizado por el lector de PDF.

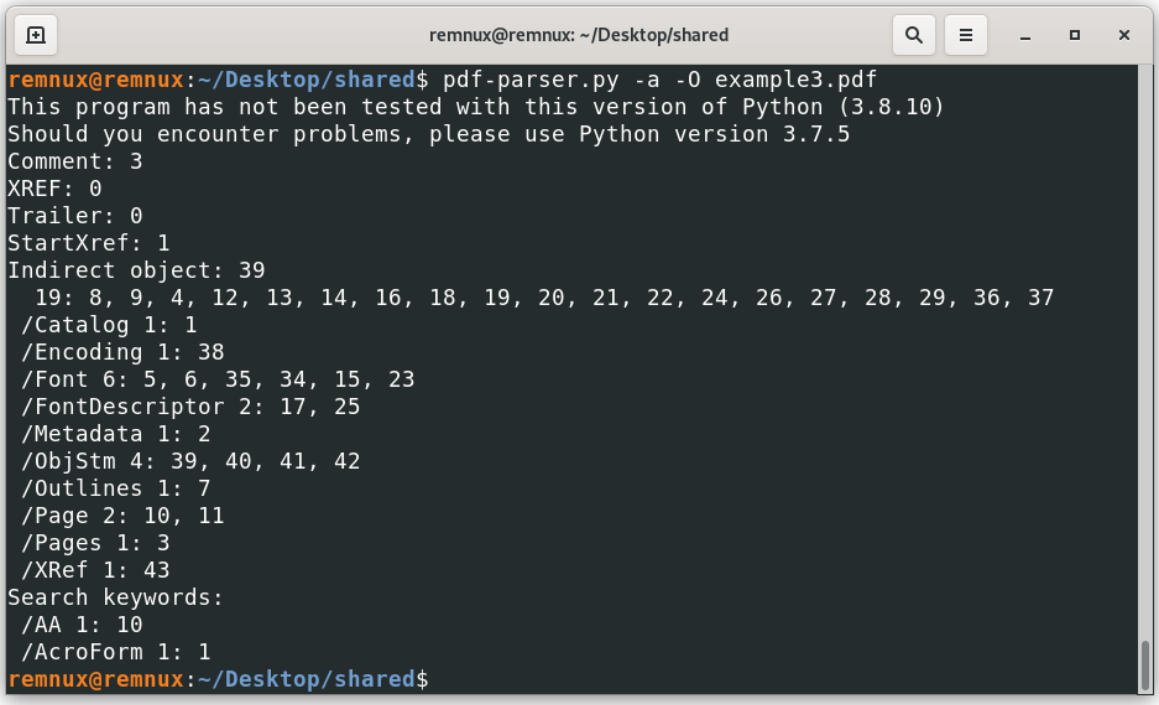
Ahora que conocemos los fundamentos de cómo revisar archivos PDF en busca de objetos sospechosos, tenemos un par de retos para usted.

Ejemplo 3

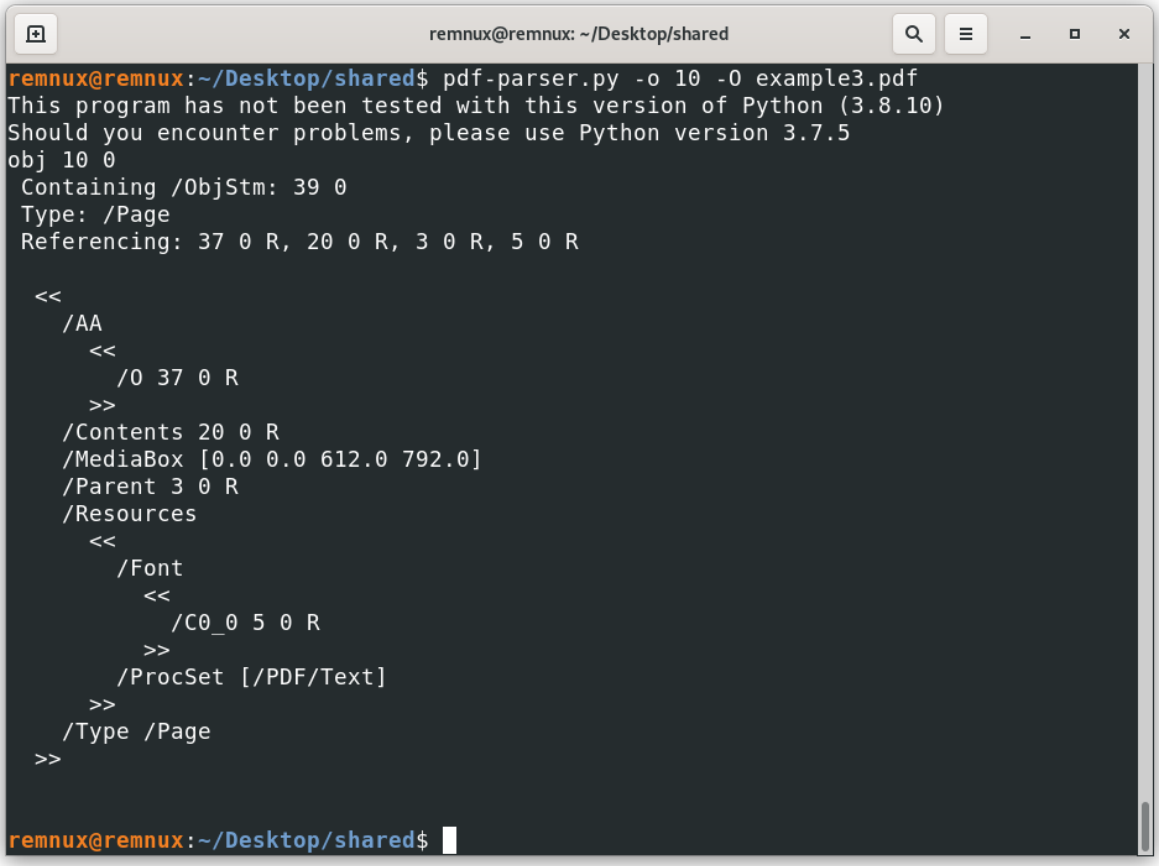
Otra práctica común del software creador de PDF es crear objetos dentro de streams que están codificados para hacer que los archivos resultantes sean más pequeños. Esto, aunque deseable en general, también crea una forma de ofuscar aún más el código malicioso. Analizando el archivo example3.pdf vemos algunos /ObjStm (Streams de objetos) que pueden contener (y de hecho contienen) otros objetos que podrían ser interesantes.



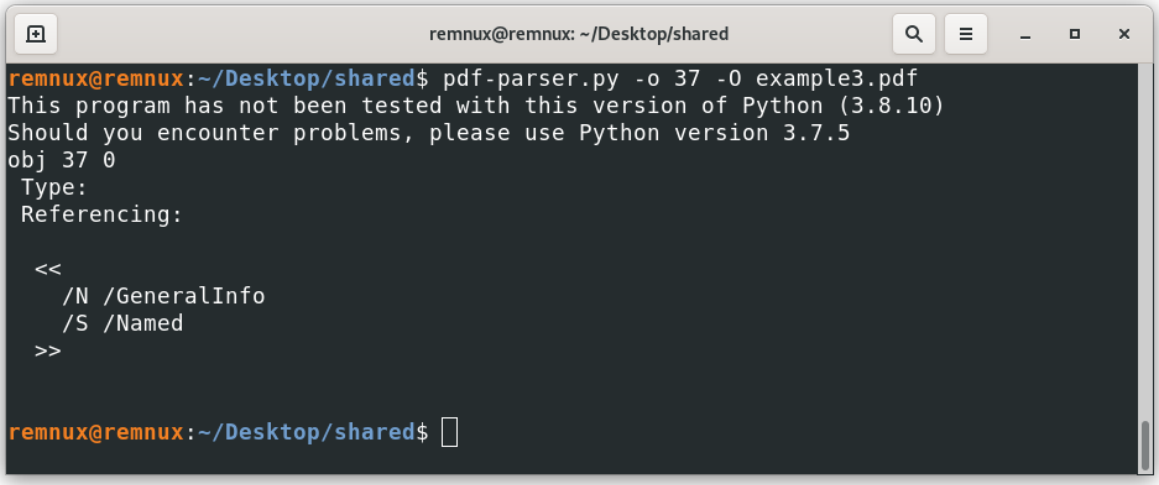
Para este tipo de escenarios, es recomendable utilizar la opción -O (como en o mayúscula) de pdf-parser. Esta opción intentará analizar cualquier stream que contenga un objeto y los tratará como objetos regulares del archivo, por ejemplo, utilizando esta opción así.



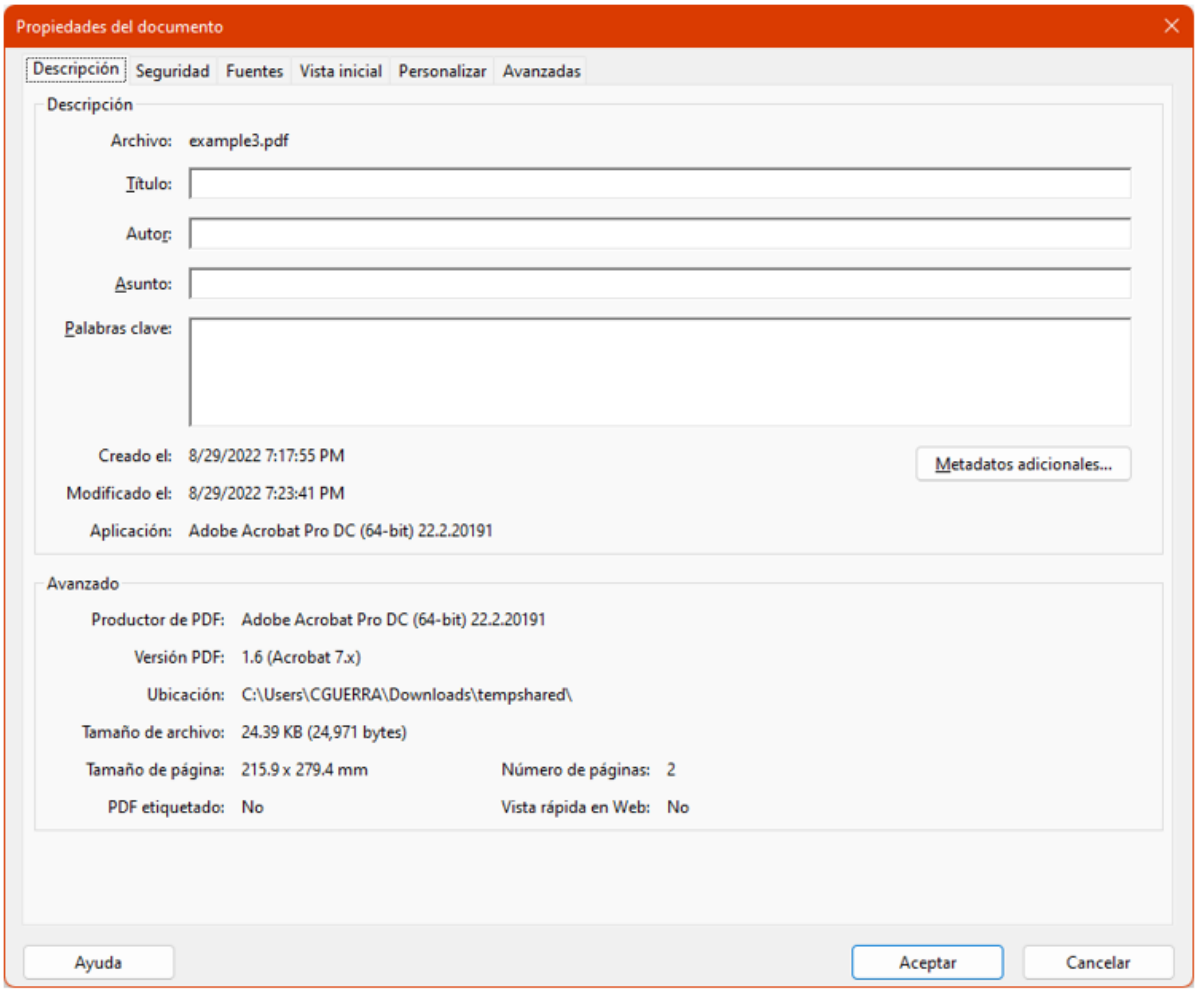
Revela que el archivo tiene "nuevos" objetos y que uno de ellos es un /AA, lo cual es interesante para buscar comportamientos maliciosos, observando el objeto respectivo tenemos:



Nos dice que la acción está vinculada a un objeto de página. En este caso, /O indica que la acción se activa cuando abrimos la página, y la acción real se almacena en el objeto 37, entonces:



Después de investigar un poco, podemos concluir que este objeto intenta abrir el cuadro de diálogo de propiedades del lector de PDF de esta manera (nada especialmente peligroso - ejemplo en una computadora configurada en español)



Conclusión: intente utilizar el parámetro -O en caso de que haya otros objetos escondidos dentro de los streams

Desafíos

Pregunta: al analizar el archivo challenge1.pdf (md5: 3b20821cb817e40e088d9583e8699938), ¿qué tipo de objeto interesante se esconde detrás de un stream?

- 1. OpenAction
- 2. AA
- 3. JS
- 4. JavaScript

Pregunta: al analizar el archivo challenge2.pdf (md5: 30373b268d516845751c10dc2b579c97), podemos ver una acción que intenta abrir una URL, ¿qué código incluye la URL como código de seguimiento? (pista: 6 números)

- Quiero ver la respuesta

Ahora que sabemos más sobre los PDF, vamos a cubrir en la siguiente parte otro tipo de archivos altamente usados como armas: los documentos de Office.