Analyse de documents malveillants (partie 2) : documents PDF

Français

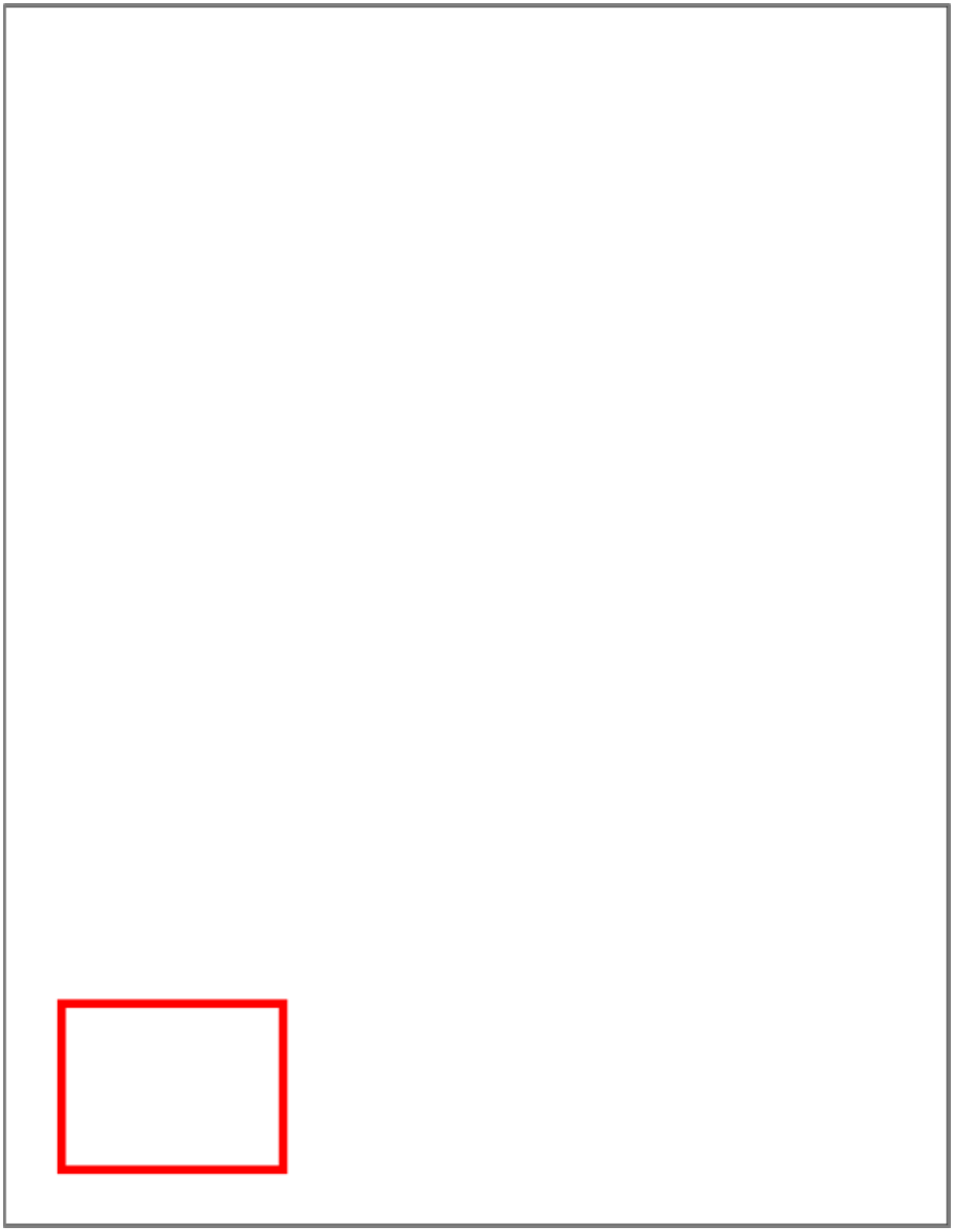
Après avoir passé en revue la gestion des machines virtuelles et l'installation de Remnux comme environnement d'analyse des artefacts suspects, nous allons explorer les fichiers PDF concernant leur format et les façons dont ils peuvent être utilisés pour nuire aux utilisateurs.

Qu'est-ce qu'un fichier PDF ?

Lorsque nous ouvrons un fichier PDF, nous utilisons un logiciel spécifique qui renvoie son contenu de manière lisible. Cependant, comme beaucoup d'autres types de fichiers, les fichiers PDF sont construits en utilisant une combinaison de texte brut ordinaire et de données binaires (pour les images et autres éléments qui pourraient le nécessiter). Par exemple, le texte suivant renvoie le fichier PDF affiché ensuite :

```
%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
```

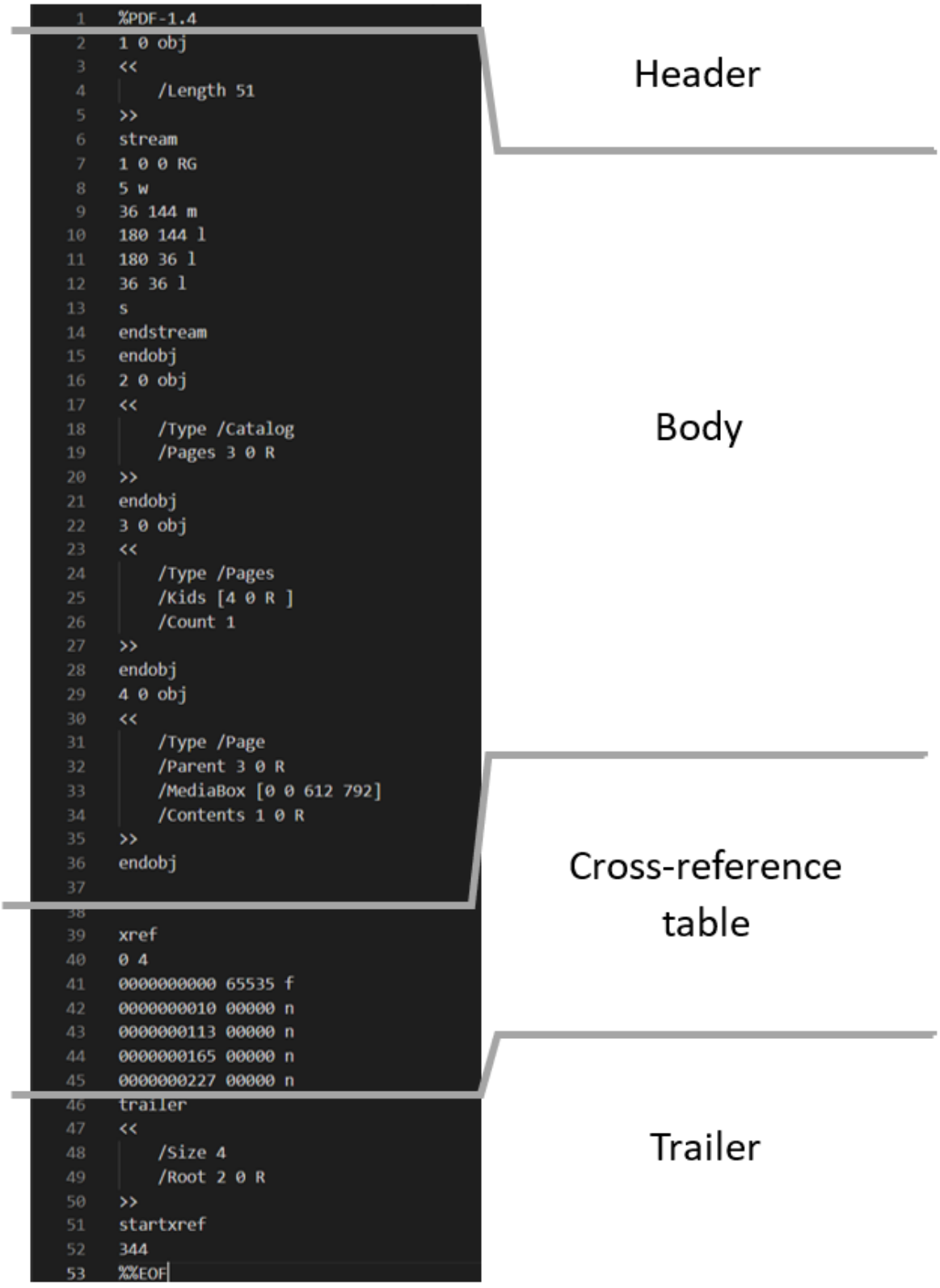
```
<<
	/Type /Page
	/Parent 3 0 R
	/MediaBox [0 0 612 792]
	/Contents 1 0 R
>>
endobj


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
```



De « Comment créer un simple fichier PDF » de Callas Software

Structure du fichier

Cet exemple nous permet de voir la structure standard d'un fichier PDF :



En-tête : il contient la version du protocole avec laquelle le fichier a été construit, pour indiquer au programme de lecture comment lire le reste de la structure et renvoyer correctement tous ses éléments.

Corps : c'est ici que seront affichés tous les objets composant le fichier PDF, pages, images, texte, polices, etc. Même le code et les actions automatisées si le fichier en possède.

Table des références croisées : nous trouvons ici une liste de tous les objets du document pour permettre un accès facile et leurs emplacements respectifs dans le fichier. Cette table est similaire à une table des matières, mais à l'usage des logiciels de lecture des fichiers PDF. Si à un moment donné le lecteur doit renvoyer un objet spécifique (par exemple, si nous faisons défiler une page aléatoire sur un grand document), le logiciel de lecture verra quelle page il doit renvoyer et la recherchera dans cette table pour localiser les éléments respectifs dans le corps afin de les charger sur l'écran.

Section finale : c'est l'endroit du document où se trouvent la table des références croisées et d'autres informations utiles, comme le nombre d'objets dans la table des références croisées (pour vérifier que le fichier n'est pas corrompu, par exemple), l'objet racine du document et les informations de chiffrement, le cas échéant. Par conception, les lecteurs de fichiers PDF commencent à lire les documents à partir de la fin, où ils peuvent trouver rapidement l'emplacement de l'objet racine et le tableau des références croisées pour commencer le rendu du contenu.

Comprendre les objets PDF

Étant donné la structure des PDF, la section la plus intéressante à approfondir est le corps, car il est possible d'y trouver tout ce qui est visible et interactif (texte, pages, images, formulaires, code, etc.). Tous ces éléments sont appelés objets et il y en a un large éventail selon les spécifications, par exemple, dans l'exemple ci-dessus, l'objet

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

est une page ayant l'id 4, il possède un élément parent ayant l'id 3 et il contient comme enfant l'élément ayant l'id 1 (le rectangle rouge). L'explication de la façon exacte dont chaque objet est écrit dépasse le cadre de ce contenu. Cependant, il convient de souligner que certains types d'éléments peuvent être utilisés à des fins malveillantes, et nous voulons les aborder sommairement.

/OpenAction : cet objet fait référence à un ensemble d'actions à exécuter lors de l'ouverture du fichier PDF, ceci peut être utilisé pour lancer un site Web afin d'en soutenir le contenu ou à des fins de suivi, exécuter du code JavaScript, etc. Cela peut être utilisé pour inciter les utilisateurs à exécuter des actions supplémentaires qui peuvent être nuisibles, ou selon l'environnement informatique, cet objet peut exécuter directement un logiciel malveillant.

/AA : cet objet comprend également une série d'actions qui sont déclenchées dans des circonstances variables, comme la visualisation d'une page, le pointeur de la souris sur des objets spécifiques, le remplissage de champs de formulaire, etc. Les risques associés sont semblables à l'objet /OpenAction, mais avec beaucoup plus de scénarios déclencheurs.

/JS ou /Javascript : contient du code JavaScript à exécuter après le déclenchement d'une action, ce code peut inclure des fonctions exclusives aux PDF.

/Launch : tente de lancer une application externe sur l'appareil après le déclenchement d'une action. Cela peut être utilisé, par exemple, pour ouvrir d'autres documents ou exécuter des commandes spécifiques, ce qui pourrait s'avérer nuisible.

/EmbeddedFile : permet l'inclusion de fichiers arbitraires, des documents aux fichiers exécutables, à l'intérieur du fichier PDF. Il existe des antécédents de fichiers PDF bénins contenant d'autres fichiers nuisibles intégrés, comme les exécutables de logiciels malveillants ou les documents Microsoft Office contenant des macros malveillantes.

/ObjStm : contient des informations arbitraires qui seront traitées selon la façon dont il est appelé. Cet objet est principalement utilisé pour regrouper de nombreux objets et les compresser afin de réduire la taille du fichier. Cependant, cela peut être utilisé pour compresser le code malveillant comme un moyen de l'obscurcir et d'éviter sa détection par les logiciels antivirus. Compte tenu des nombreux cas d'utilisation différents pour ce type d'objet, supposer que sa présence est malveillante peut aboutir à de nombreuses fausses alertes.

Dans les attaques plus élaborées, une combinaison de ces objets peut être utilisée, par exemple, un fichier PDF peut déclencher une action qui ouvre un fichier intégré dans le même fichier chiffré dans un objet /ObjStm.

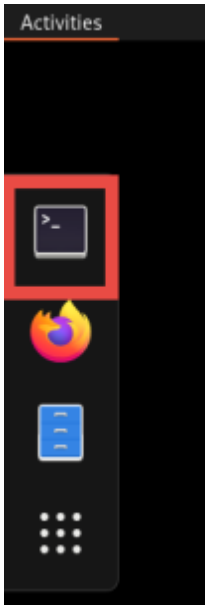
En considérant tous ces éléments, nous vérifierons si un fichier PDF comprend ces types d'objets comme une première étape pour déterminer s'il est malveillant, ou au moins pour s'assurer qu'il ne l'est pas.

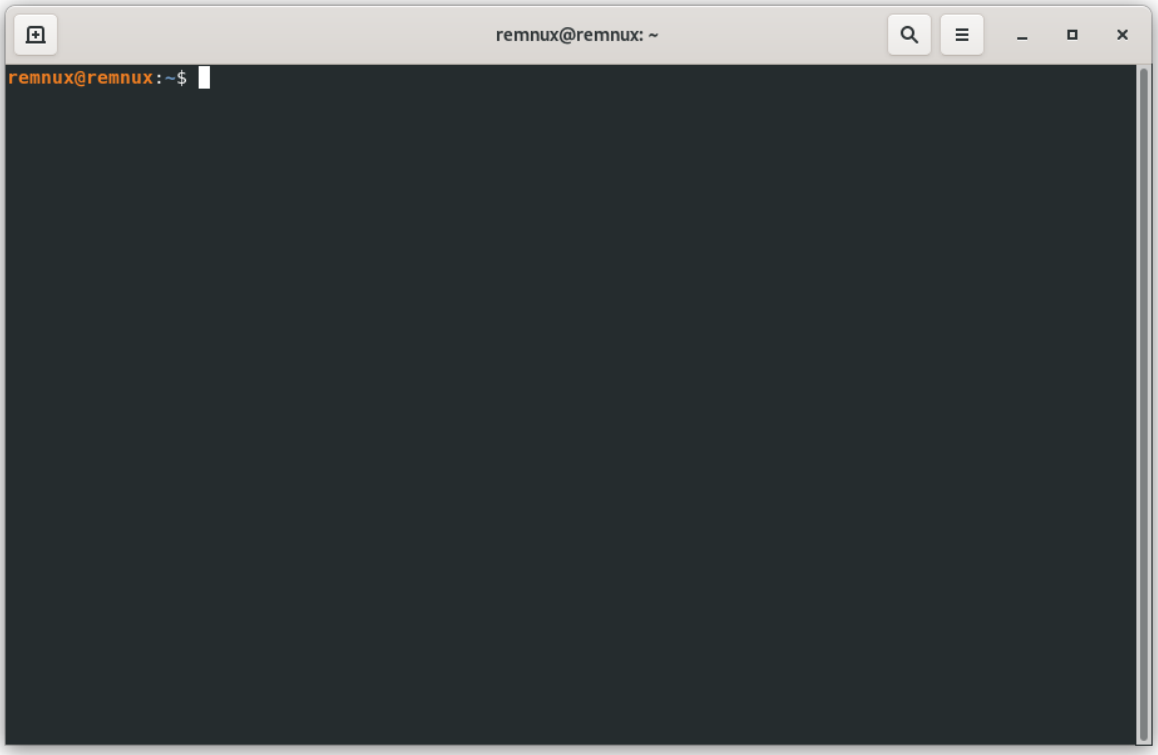
Découvrir pdfid

Maintenant que nous savons où chercher les drapeaux rouges dans les fichiers PDF que nous pourrions considérer comme étant suspects, nous pouvons commencer à utiliser l'outil pdfid comme une première étape pour voir les types d'objets qui sont contenus dans notre fichier. Pdfid fait partie d'une suite d'outils développés par Didier Stevens pour rationaliser certains processus d'analyse des fichiers PDF. Ces outils sont exécutés en ligne de commande et sont donc connus comme des applications CLI (Command Line Interface). Nous allons expliquer comment les utiliser par le biais de la machine virtuelle Remnux que nous avons configurée dans la partie précédente de ce cours.

Pour utiliser pdfid, nous avons besoin d'ouvrir une application Terminal dans notre machine virtuelle. Cette fenêtre devrait déjà être ouverte lorsque nous démarrons notre VM.

Cependant, nous pouvons toujours cliquer sur le menu Activités dans le coin supérieur gauche, puis sur l'icône du terminal dans le panneau de gauche, comme indiqué sur l'image.



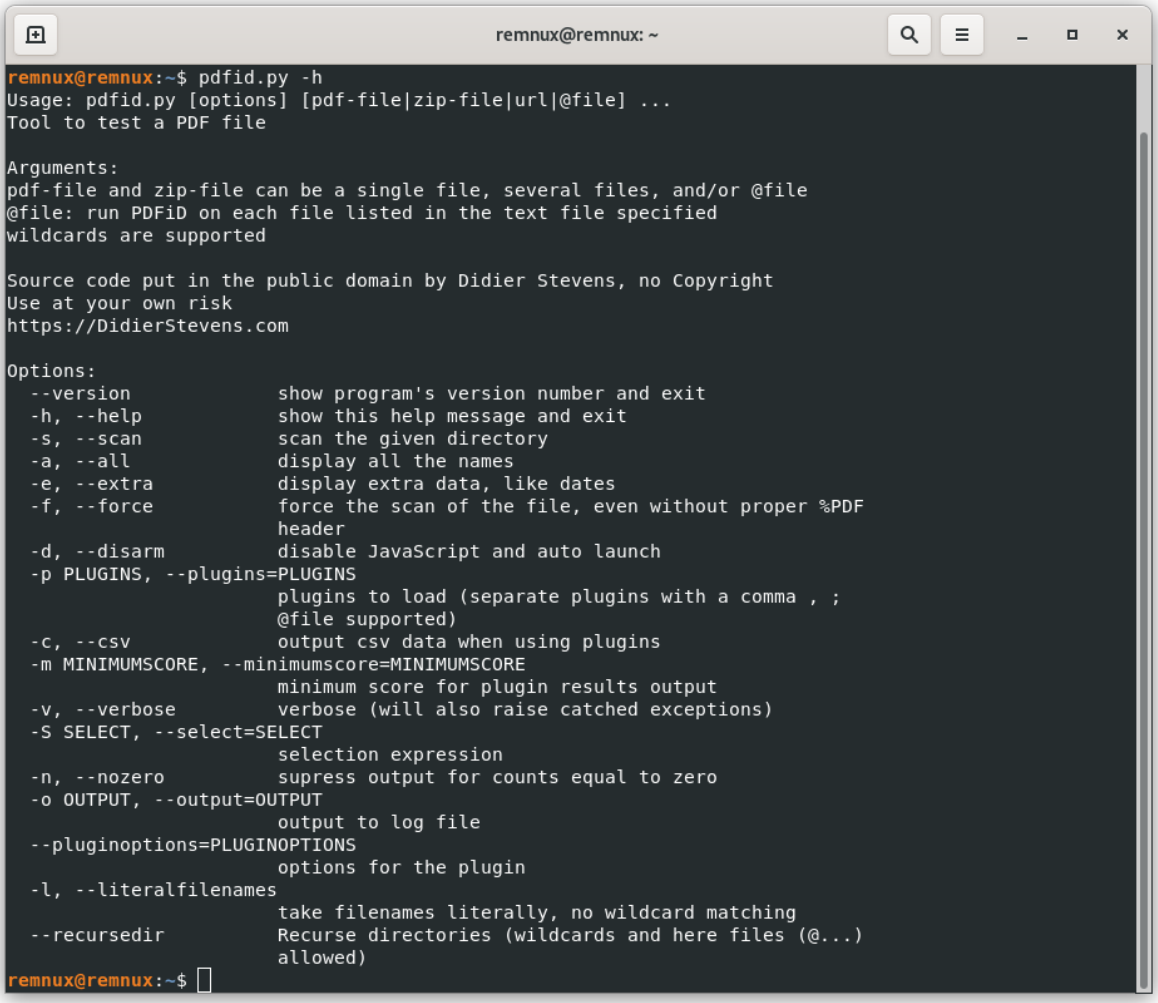


Une fois dans la fenêtre du Terminal, nous pouvons commencer à explorer l'utilisation de la commande pdfid dans sa section d'aide. Pour cela, il suffit de saisir :

```
pdfid.py -h
```

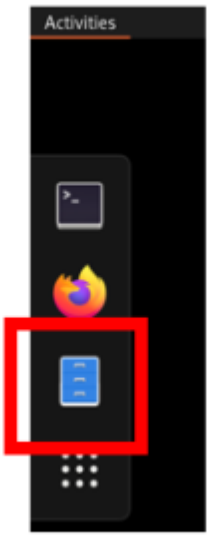
-h signifiant « help » (aide)

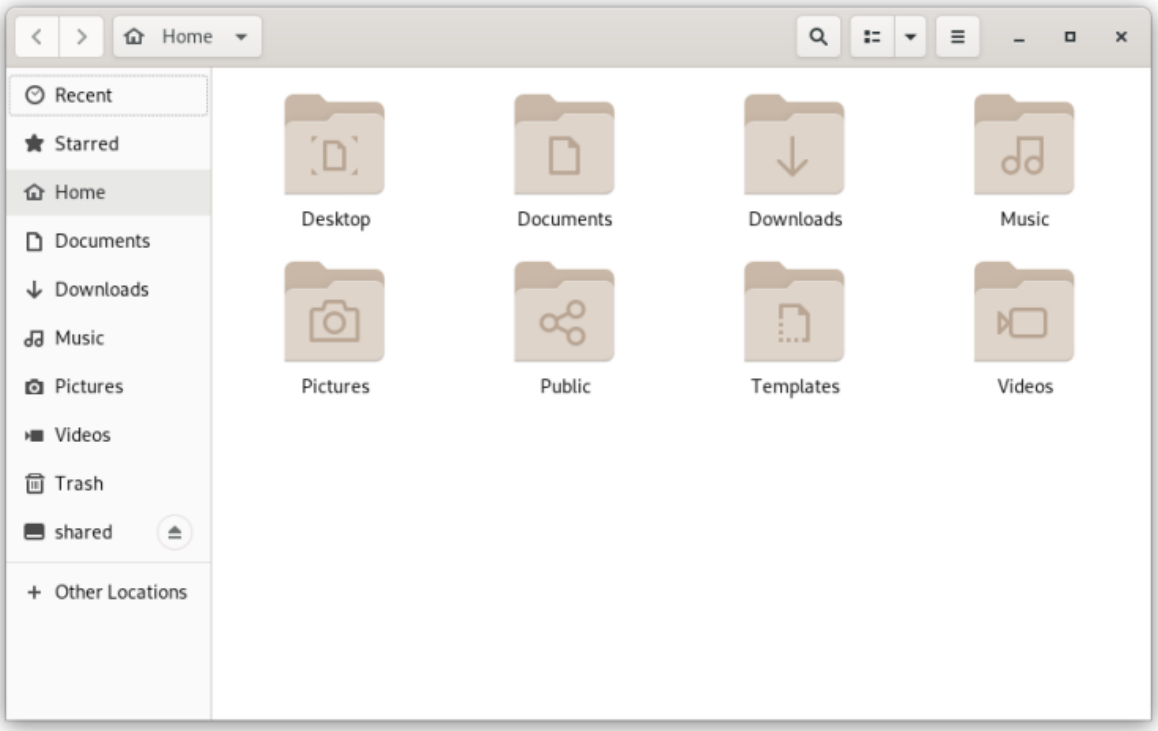
Conseil : nous pouvons toujours utiliser la touche Tab pour remplir automatiquement certaines commandes :



Ici, nous pouvons voir un certain nombre d'options que nous pouvons employer lors de l'utilisation de pdfid. Cela peut sembler difficile pour ceux qui n'ont pas l'habitude d'utiliser le terminal, mais nous n'utilisons généralement que quelques-unes de ces options, et avec peu de pratique, le processus devient plus rapide et plus facile.

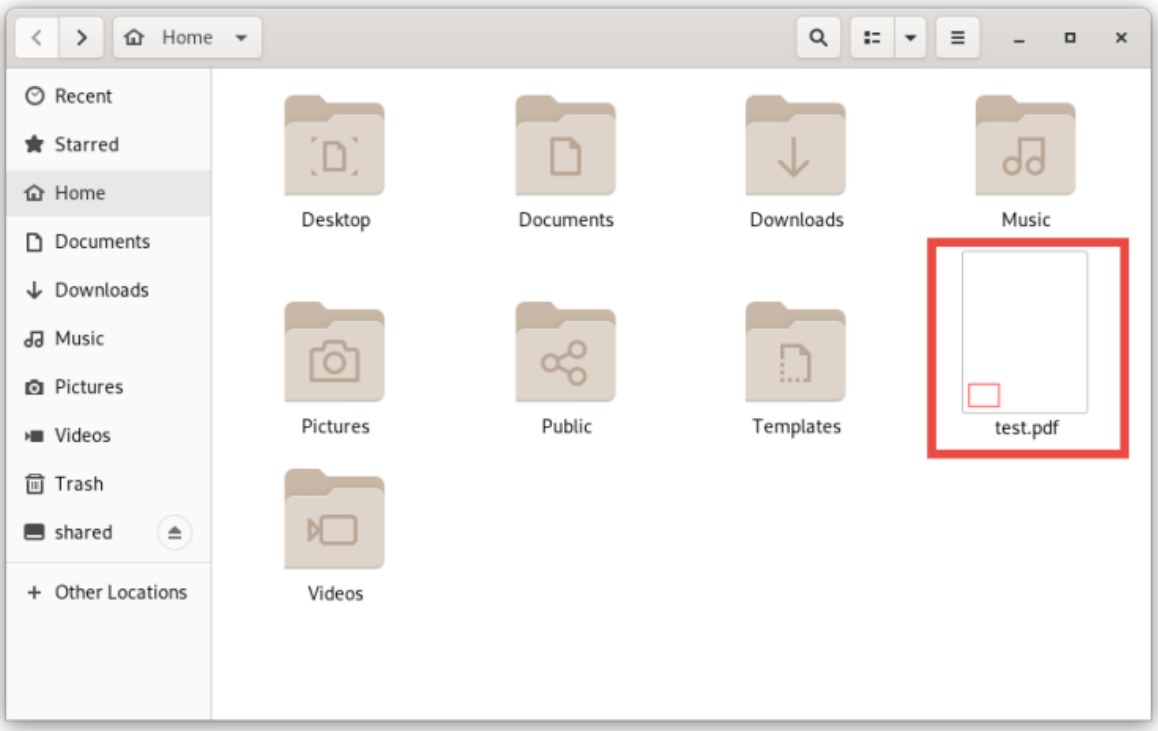
Pour analyser notre premier fichier, nous devons garder à l'esprit que la commande que nous exécutons dans la ligne de commande est en cours d'exécution « à partir d'un dossier/répertoire ». Nous avons donc besoin de savoir où est exécutée la commande et où se trouve le fichier que nous voulons analyser. Pour vous donner un peu de contexte, chaque fois que nous ouvrons l'application Terminal dans Remnux, nous ouvrons un terminal dans le répertoire Home, le même emplacement que nous voyons lorsque nous ouvrons l'application Files :





Pour faciliter les choses, nous pouvons mettre nos fichiers PDF dans ce dossier, de sorte que la commande Terminal est exécutée à partir du même répertoire que celui de notre fichier PDF.

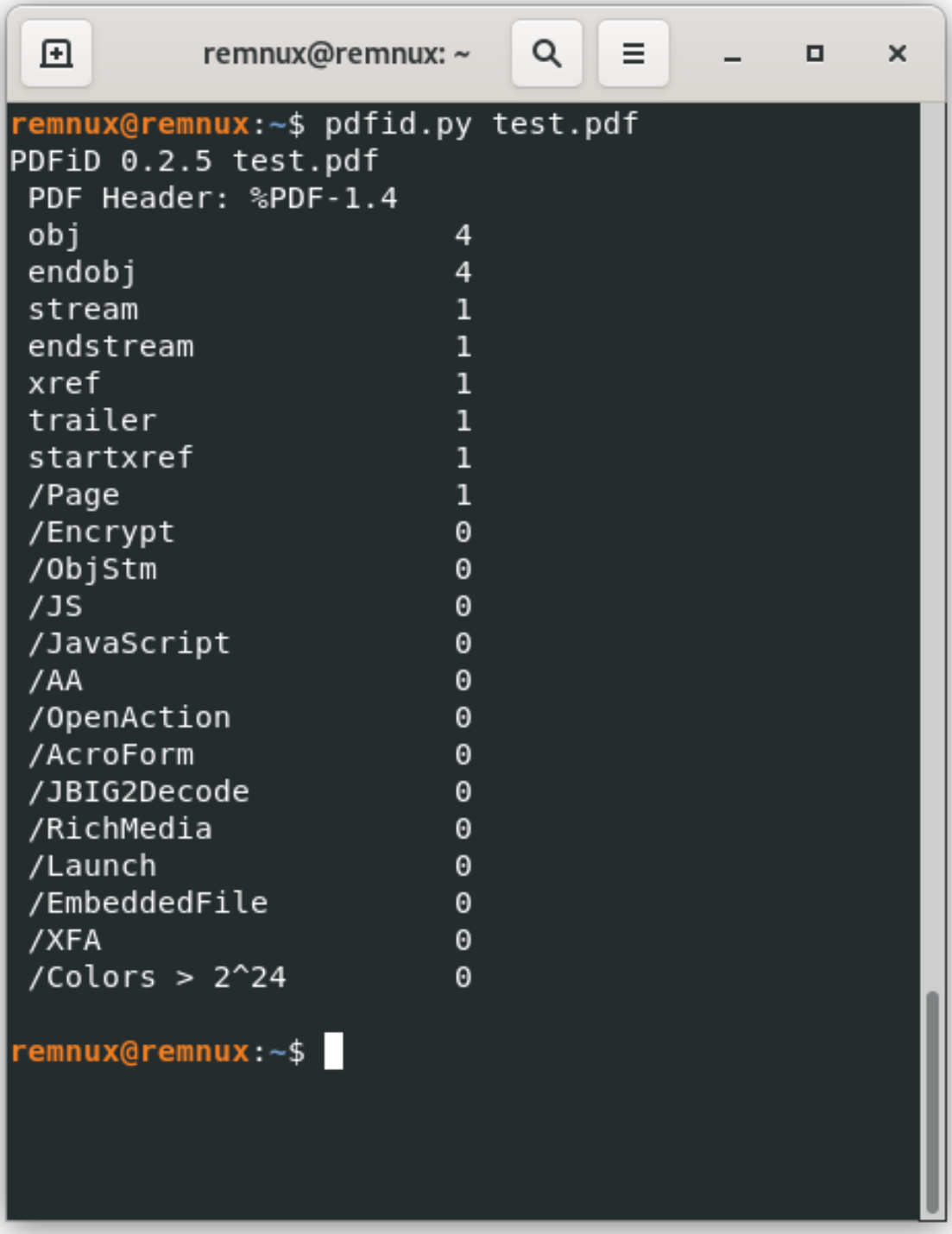
Nous pouvons prendre notre fichier d'exemple ci-dessus et l'enregistrer en tant que fichier PDF à l'aide d'un éditeur de texte sur notre ordinateur hôte et faire glisser le fichier dans le répertoire home de Remnux.



Nous pouvons ensuite exécuter la commande suivante dans notre Terminal :

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

Pour recevoir cette réponse :



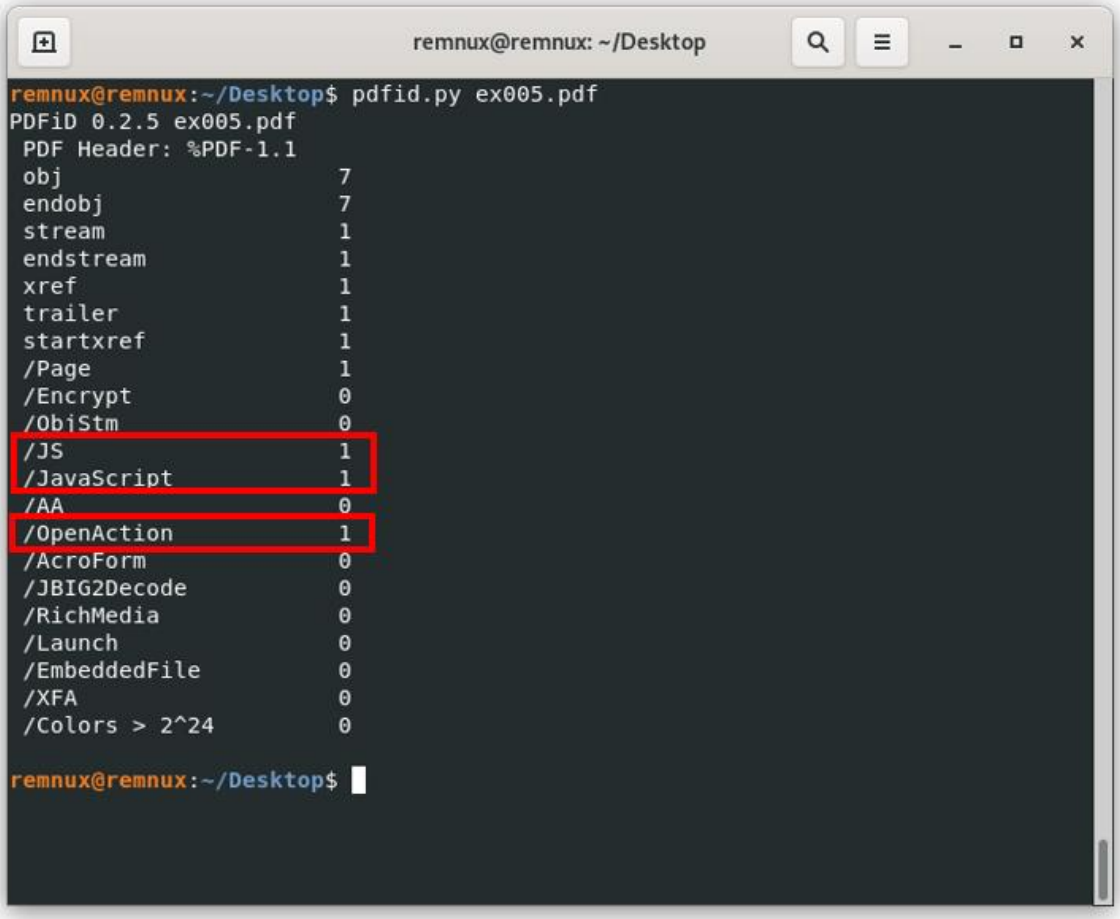
Comme nous pouvons le vérifier, tous les objets vus par pdfid correspondent à ceux que nous connaissons de la source du fichier PDF, et aucun d'entre eux ne semble être dans la liste des objets suspects que nous avons décrits ci-dessus.

Comme nous l'avons dit précédemment, les acteurs malveillants utilisent certaines techniques pour éviter la détection de types spécifiques d'objets et l'outil pdfid tente de

montrer même les objets masqués. Cependant, dans certains cas moins courants, certains objets cachés peuvent nécessiter une analyse un peu plus approfondie pour être découverts.

Saisissez pdf-parser, exemple 1

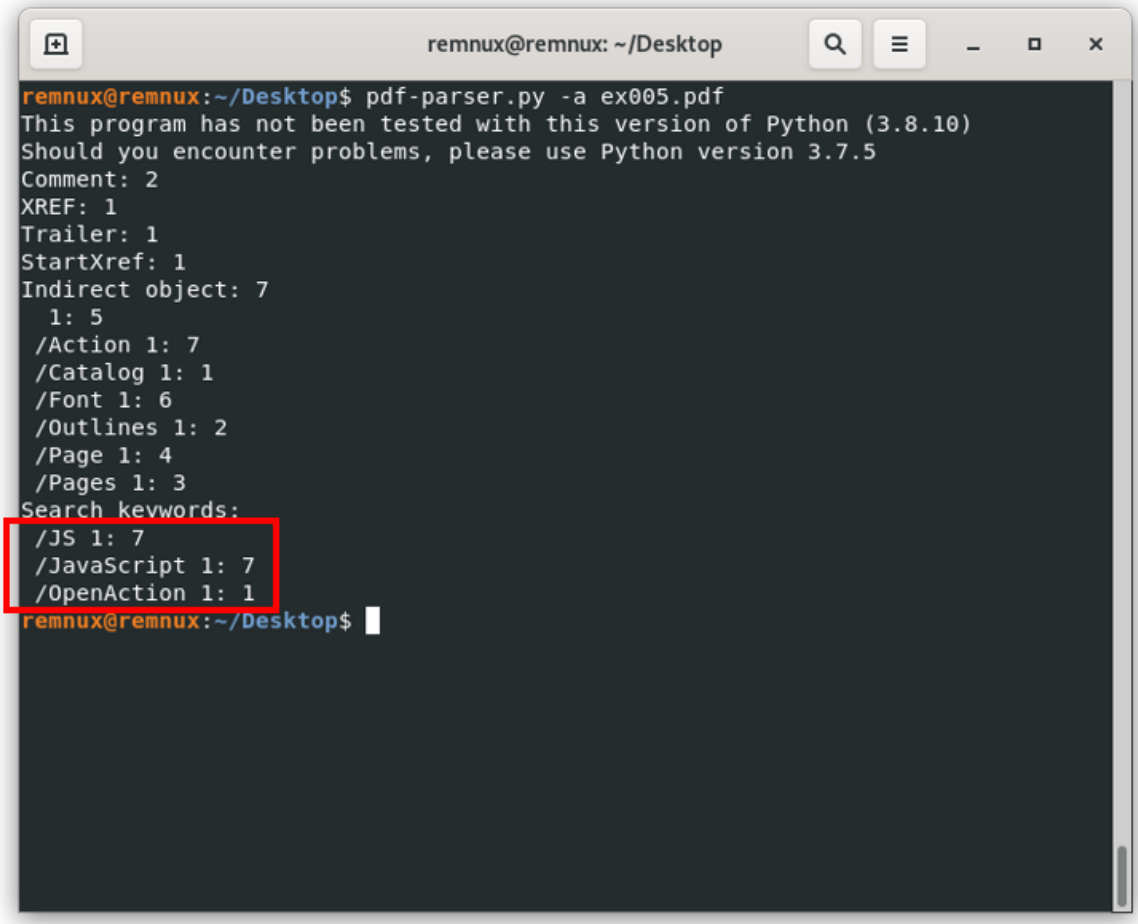
Que se passe-t-il lorsque nous trouvons un fichier PDF avec un objet suspect ? Imaginez que nous ayons un fichier ex005.pdf qui nous donne ce résultat sur pdfid :



À ce stade, compte tenu des indications fournies ci-dessus dans cette ressource, nous savons que les 3 objets de types /JS, /Javascript et /OpenAction peuvent être intéressants à examiner, particulièrement parce qu'ils suggèrent que le fichier tente d'exécuter une action quand nous ouvrons le fichier. Ici, nous pouvons l'exécuter avec pdf-parser pour déterminer quel type d'objet correspondant à chaque objet et voir le contenu de ces objets. Pour notre exemple de fichier, nous allons exécuter la commande suivante :

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

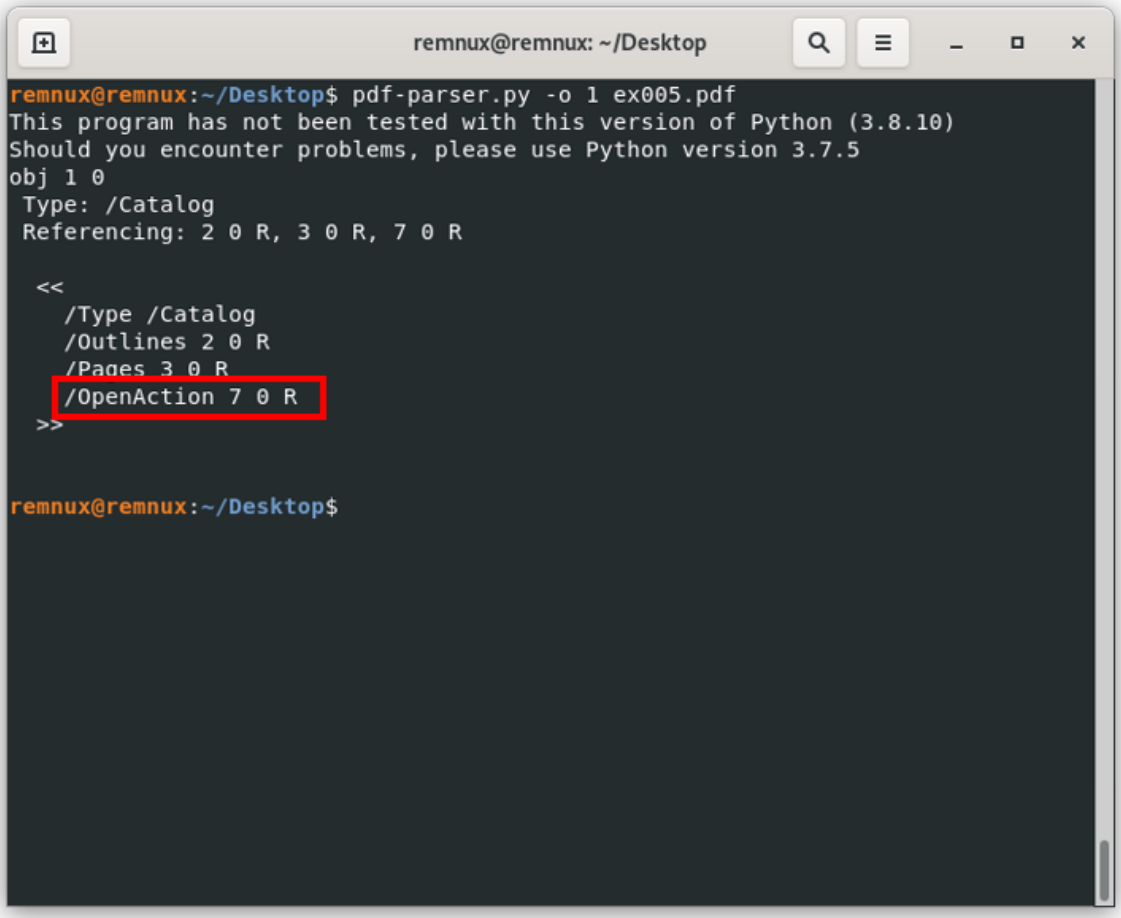
Nous utilisons l'argument `-a` pour voir les statistiques du fichier. Pour obtenir une meilleure référence, nous pouvons toujours utiliser `pdf-parser.py -help` afin d'afficher une liste d'options à l'écran.



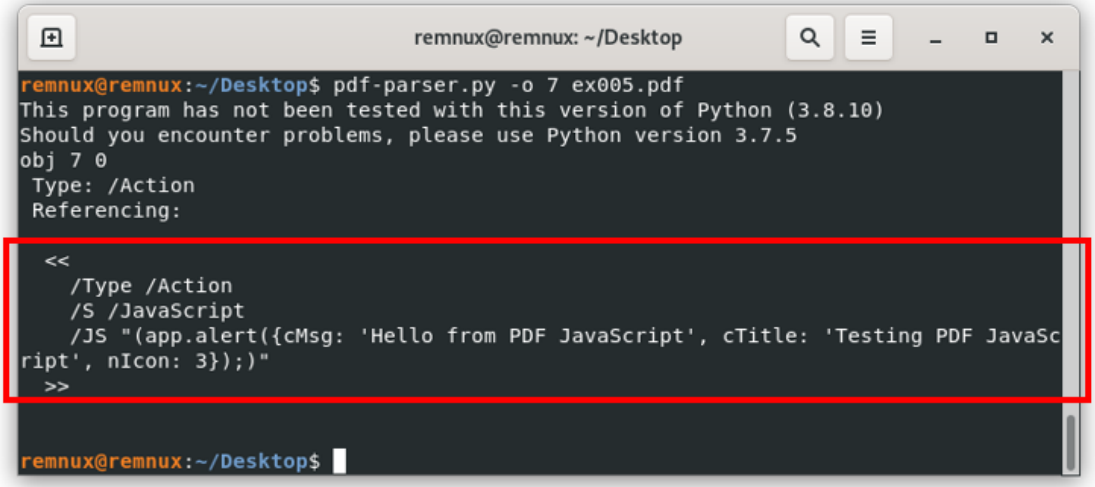
Ici, nous pouvons voir que nous avons effectivement ces trois objets problématiques, mais aussi quelques informations supplémentaires. Nous pouvons voir les id des objets pour chaque type d'objet. Nous savons maintenant que les objets /JS et /Javascript sont compris dans l'objet ayant l'id 7, et /OpenAction est contenu dans l'objet ayant l'id 1. Ensuite, nous pouvons voir le contenu de l'objet /OpenAction pour voir ce que le document essaie de faire en l'ouvrant. Pour cela, nous utilisons la commande :

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

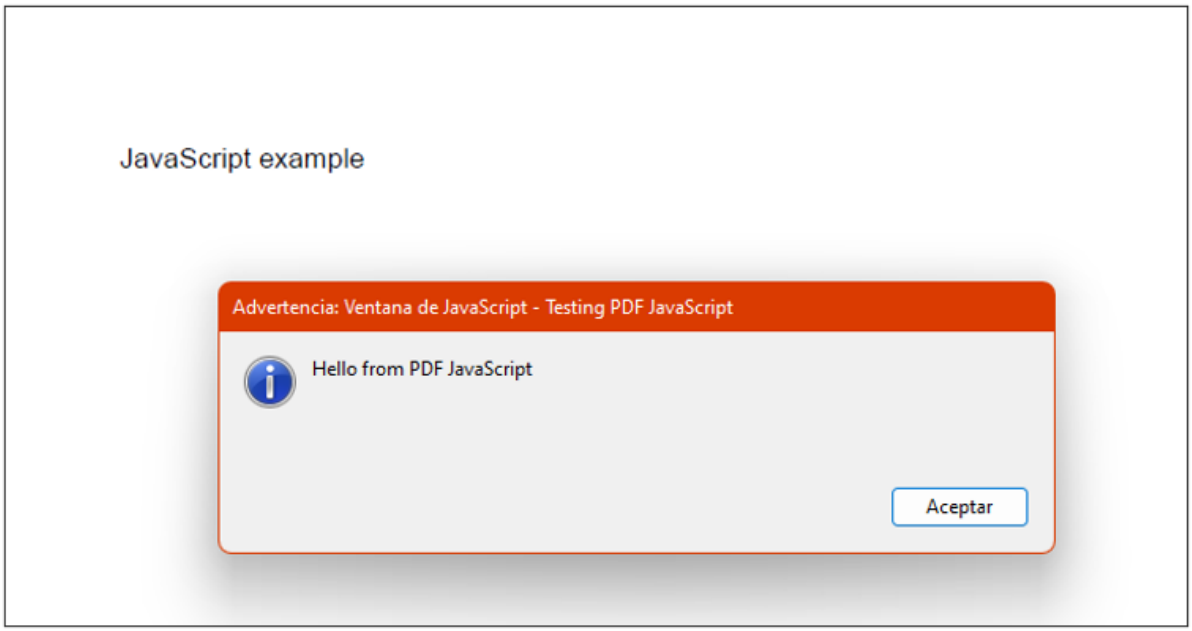
Ici l'argument -o est utilisé pour donner à l'outil l'id de l'objet dont nous souhaitons voir le contenu à l'écran :



Ici, nous pouvons voir la ligne « /OpenAction 7 0 R » qui signifie que le contenu réel de l'objet /OpenAction est contenu dans l'objet ayant l'id 7, et que lorsque nous ouvrirons le fichier, nous appellerons ou référencerons cet objet. En répétant le processus pour voir le contenu de l'objet ayant l'id 7, nous obtenons :

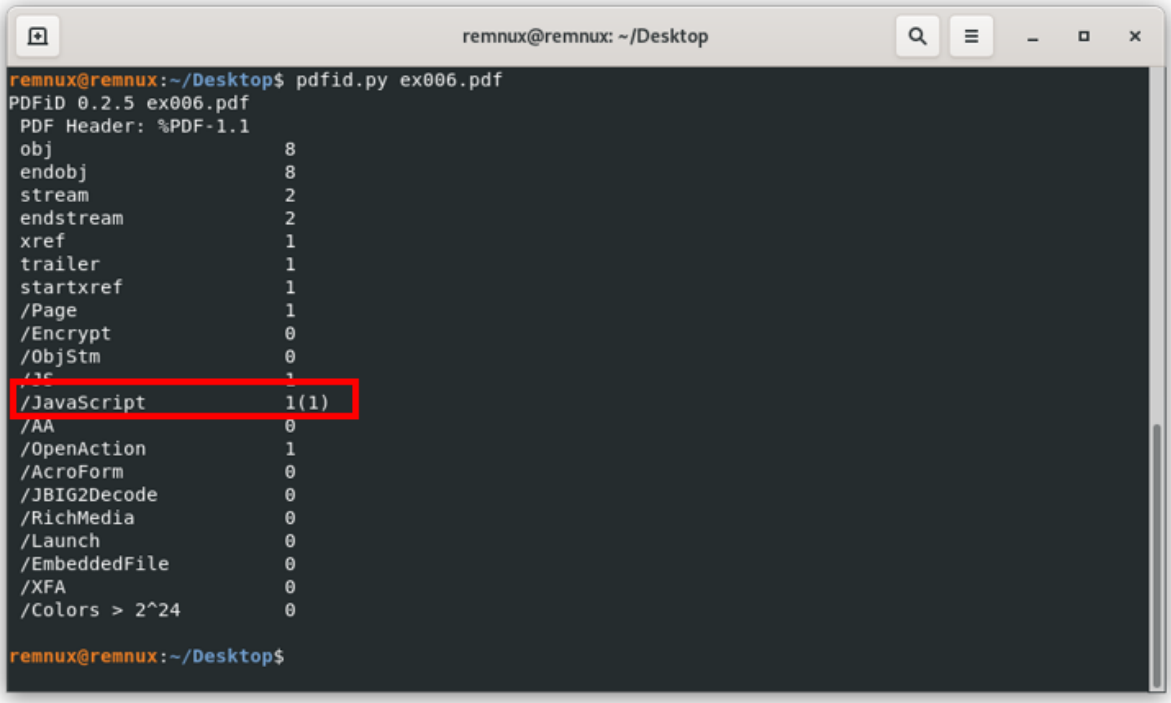


Lorsque nous voyons que le document essaie d'afficher une alerte ou une fenêtre contextuelle avec le message décrit dans le terminal, si nous ouvrons le fichier, il ressemblera à ceci :

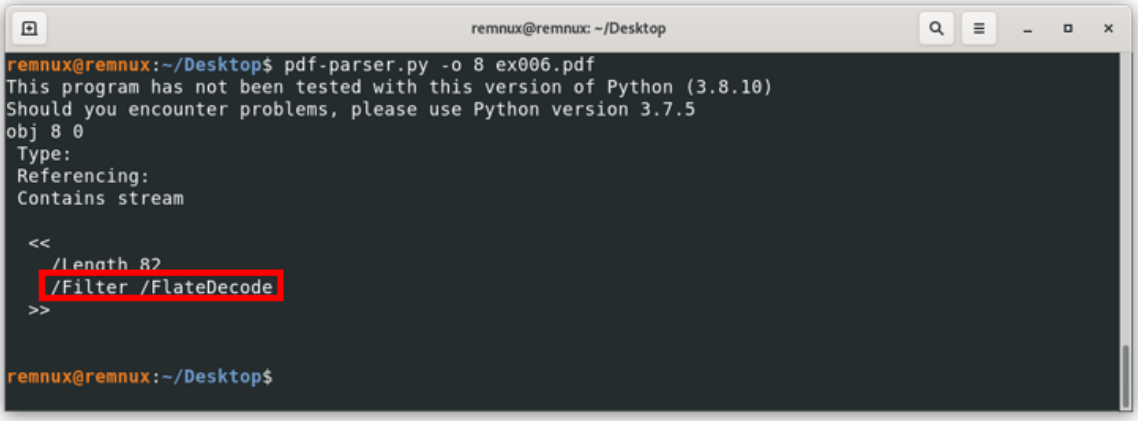


Exemple 2

Comme nous l'avons mentionné précédemment, il peut y avoir des fichiers dans lesquels le contenu suspect n'est pas visible en texte brut, et il peut y avoir un certain nombre de raisons pour le faire dans certains cas légitimes, comme la compression de longs morceaux d'information pour réduire la taille du fichier, entre autres. Cependant, les fichiers malveillants utilisent ces techniques à des fins d'obscurcissement pour aider à éviter la détection par un logiciel antivirus et d'autres solutions de sécurité. Par exemple, si nous répétons le flux de travail précédent dans le fichier ex006.pdf, nous verrons que le résultat de la commande pdfid est la suivante :



Ici, nous pouvons voir dans la ligne /JavaScript « 1(1) », cela signifie que pdfid a détecté un objet de ce type, bien qu'il soit masqué. En répétant le même processus que l'on connaît déjà, nous passons en revue l'objet ayant l'id 8 (où se trouve le code JavaScript) pour voir ce qui suit :

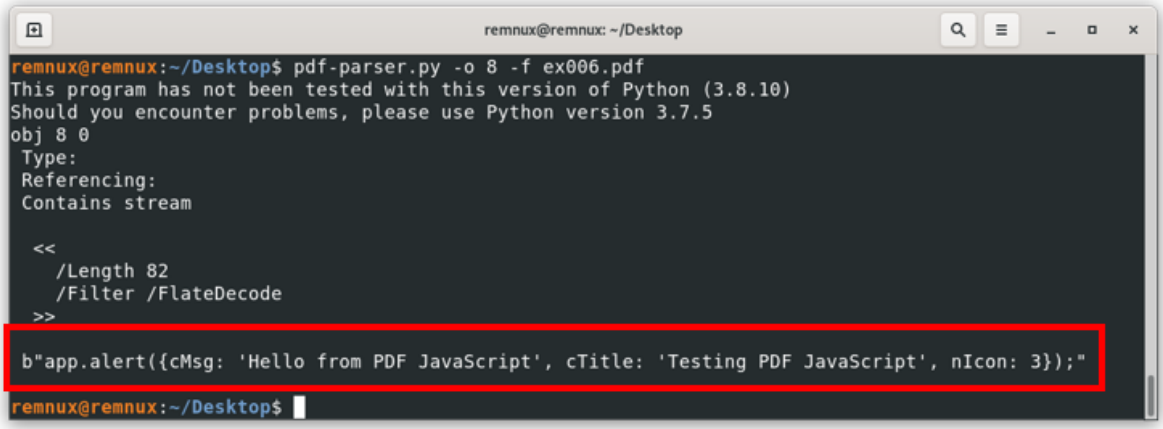


Ici, nous ne voyons pas le code réel comme dans le dernier exemple, au lieu de cela, nous pouvons voir, entre autres, la ligne « /Filter /FlateDecode ». L'option /Filter exécute une opération sur le contenu final d'un flux pour le décoder, puis /FlateDecode indique le codage associé qui doit être pris en compte lors du décodage du contenu. Pour le voir d'une autre façon, si nous ouvrons le fichier avec un éditeur de texte et que nous recherchons manuellement cet élément, nous devrions voir quelque chose comme ceci :

```
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
```

Où le contenu à l'intérieur du carré rouge est le contenu encodé réel. Dans ce cas, pdf-parser peut essayer de décoder le contenu réel. Pour cela nous utilisons l'argument -f, et nous finissons par utiliser la commande :

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

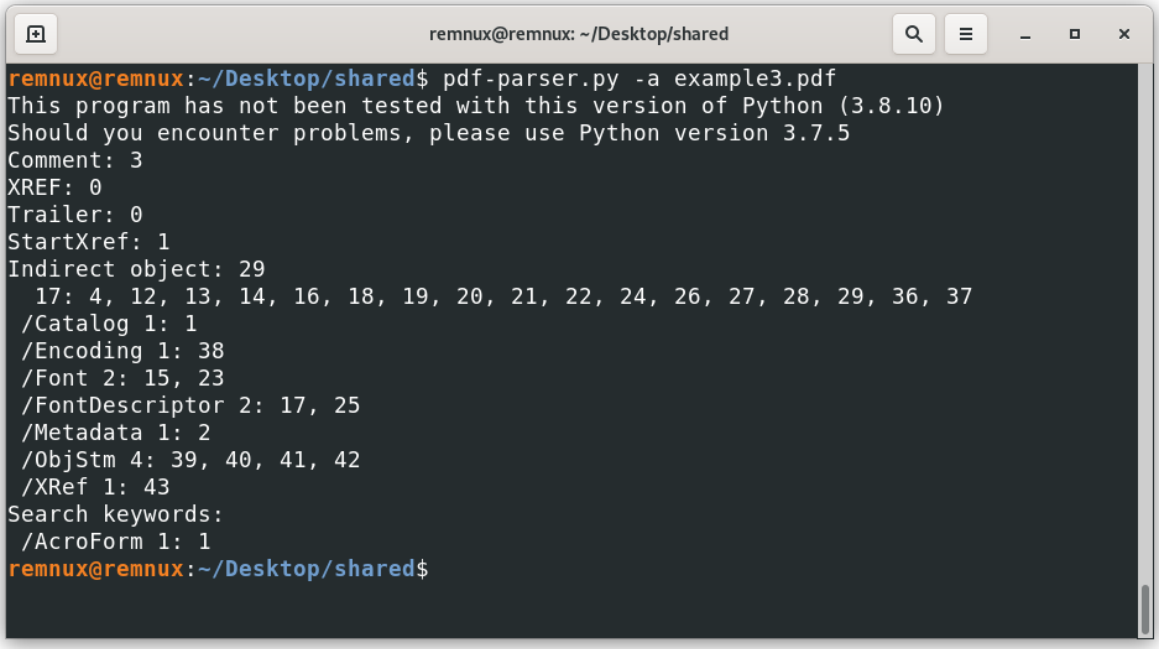


Où nous pouvons voir le contenu réel de l'objet à rendre par le lecteur PDF.

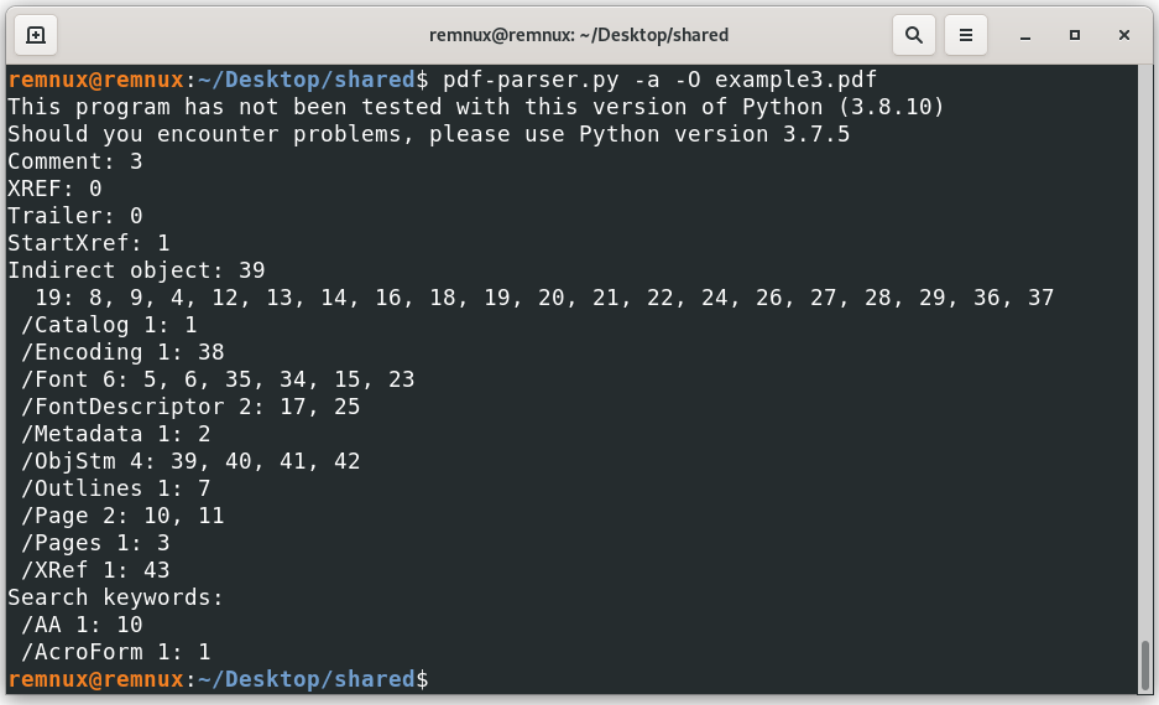
Maintenant que nous connaissons les bases de la façon de consulter des fichiers PDF pour rechercher des objets suspects, nous pouvons relever quelques défis.

Exemple 3

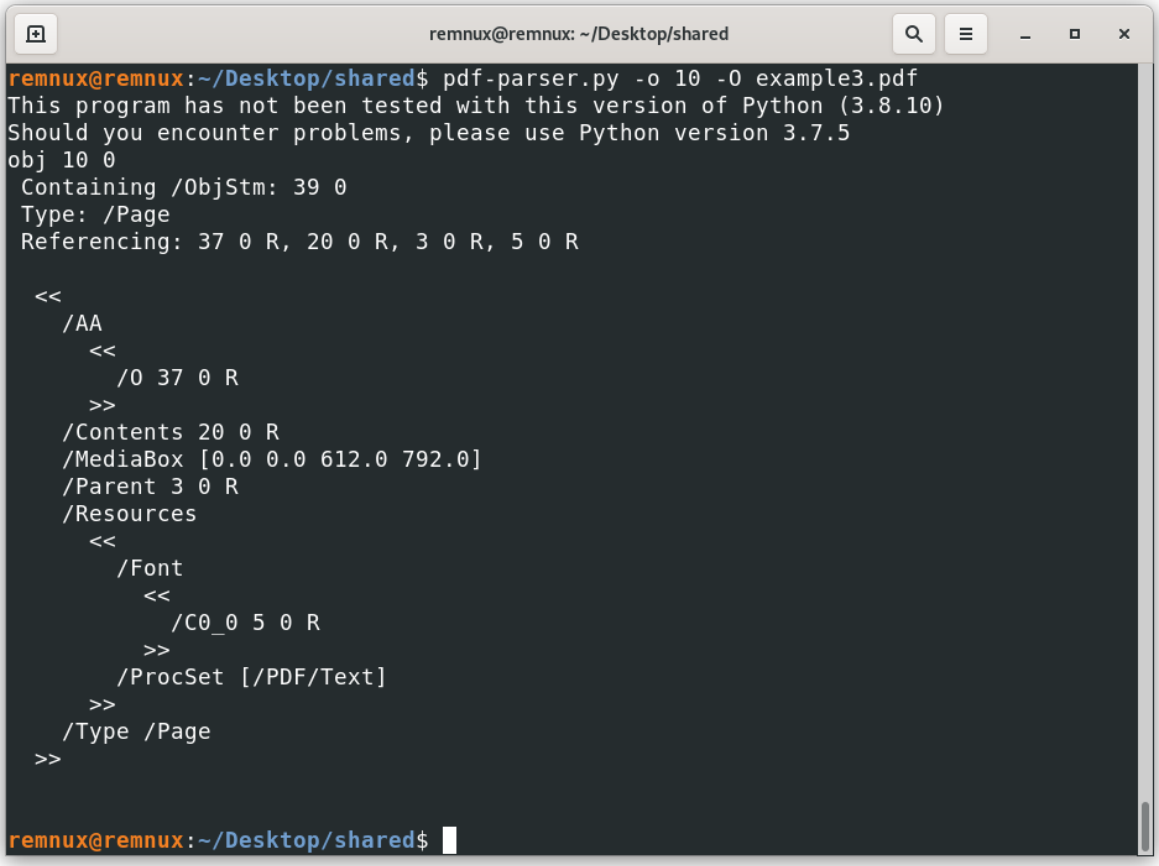
Une autre chose que les logiciels de création de PDF font habituellement pour créer de nouveaux fichiers est de créer des objets à l'intérieur des flux qui sont codés pour rendre les fichiers résultants plus petits, ce qui est souhaitable en général, mais crée également un moyen d'obscurcir davantage le code malveillant. En analysant le fichier example3.pdf, nous voyons que certains /ObjStm (Object Streams) peuvent contenir (et contiennent effectivement) d'autres objets qui pourraient être intéressants.



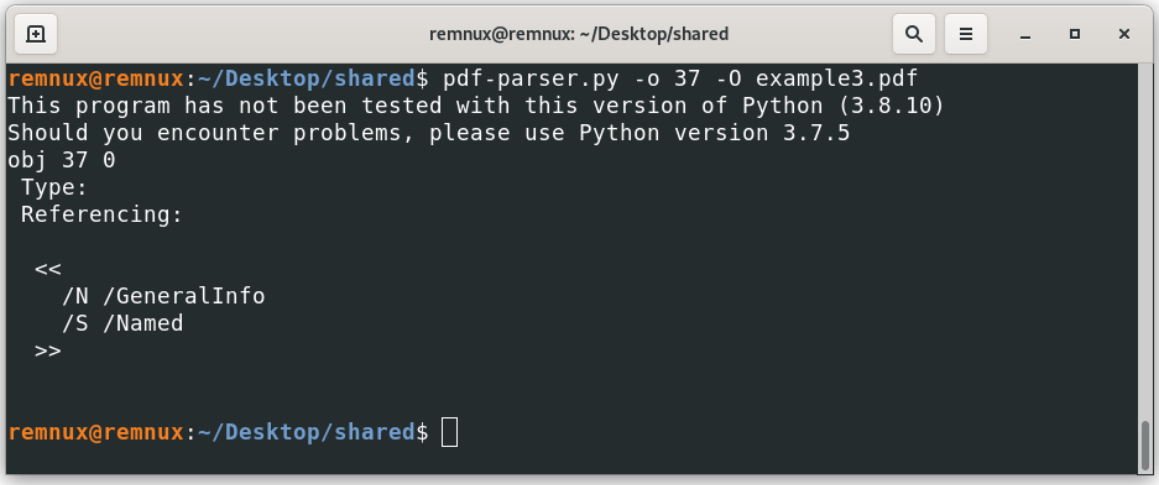
Dans ce genre de situations, il est conseillé d'utiliser l'option -O (la lettre o majuscule) de pdf-parser, cette option tentera d'analyser les flux contenant un objet et les traitera comme des objets réguliers du fichier, par exemple, en utilisant cette option comme ceci.



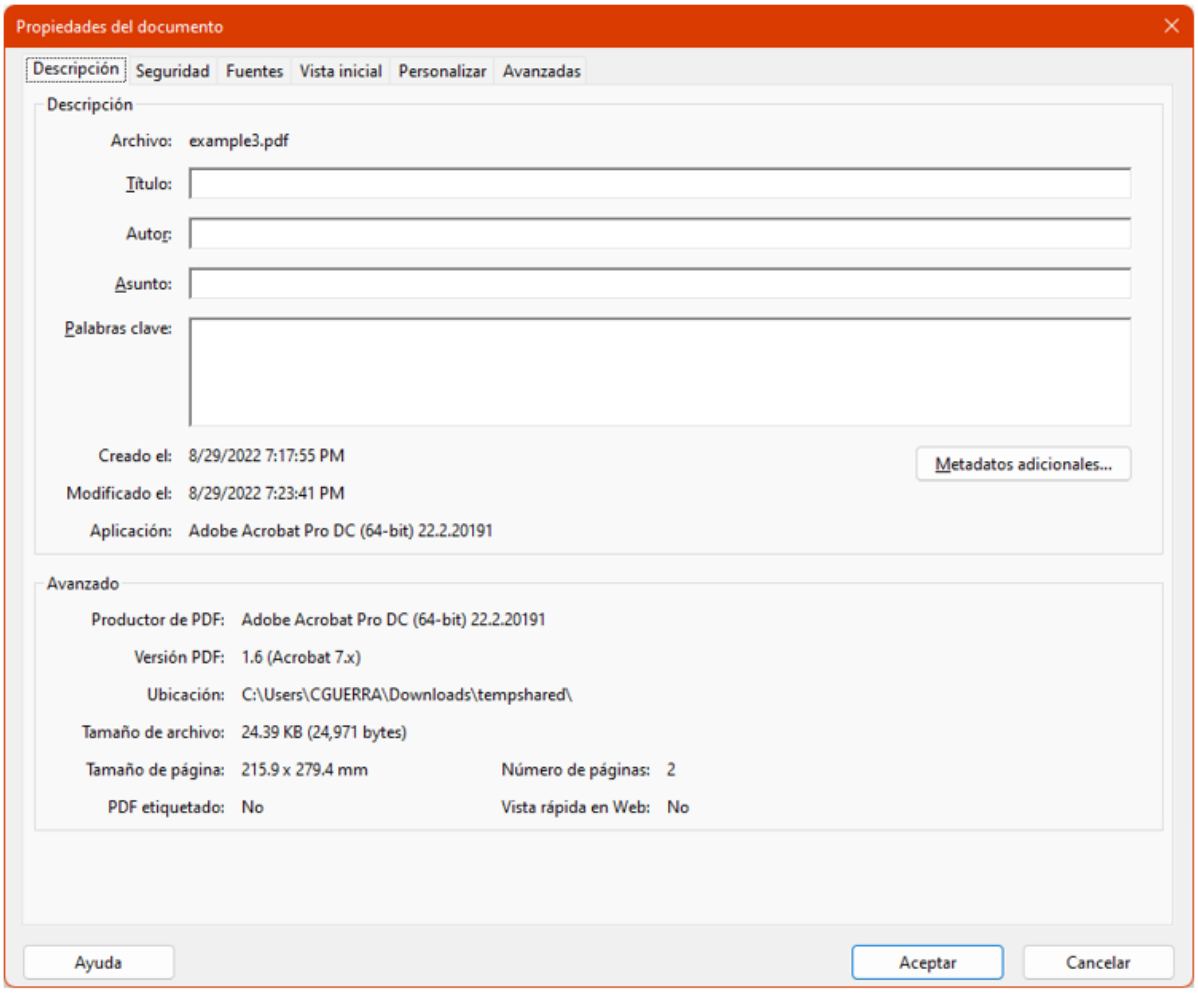
Cela révèle que le fichier contient des objets « nouveaux » et qu'un de ces objets est un /AA, ce qui est intéressant pour rechercher les comportements malveillants, en regardant l'objet respectif que nous avons :



Cela nous indique que l'action est liée à un objet page, dans ce cas le /O indique que l'action est déclenchée lorsque nous ouvrons la page, et l'action réelle est stockée dans l'objet 37, alors :



Après quelques recherches, nous pouvons conclure que cet objet tente d'ouvrir la boîte de dialogue des propriétés du lecteur PDF comme ceci (rien de particulièrement dangereux. Exemple dans un ordinateur configuré en espagnol)



Conclusion : essayez d'utiliser le paramètre -O au cas où il y aurait d'autres objets cachés dans les flux

Défis

Question : en analysant le fichier challenge1.pdf (md5: 3b20821cb817e40e088d9583e8699938), quel genre d'objet intéressant se cache derrière un flux ?

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

Question : en analysant le fichier challenge2.pdf (md5: 30373b268d516845751c10dc2b579c97), nous pouvons voir une action qui tente d'ouvrir une URL, quel code l'URL inclut-elle comme code de suivi ? (Indice : 6 chiffres)

- Je veux voir la réponse

Maintenant que nous en savons plus sur les fichiers PDF, nous allons aborder dans la partie suivante un autre type de fichiers fortement utilisé à des fins malveillantes : les documents Office.