Content Disarm and Reconstruction

WHITE PAPER

Suite 10146, 8 The Green, Dover, DE 19901, United States
Voice: +1(228)641 1388
Email: sales@cysecuritycorp.com

Table of Contents

Introduction

CySecurity Corp is a US HQ security SaaS company that helps CISOs of BFSI/organizations to ensure the safety of their infrastructure.

This whitepaper describes the importance of Content Disarm and Reconstruction and gives a quick glimpse of the CDR technology. The objective of this paper is to provide insight on how weaponization of PDFs/doc/docx is a new threat that security experts work tirelessly to identify, research, and develop new ways to guard, using the CDR technology.

Abbreviation

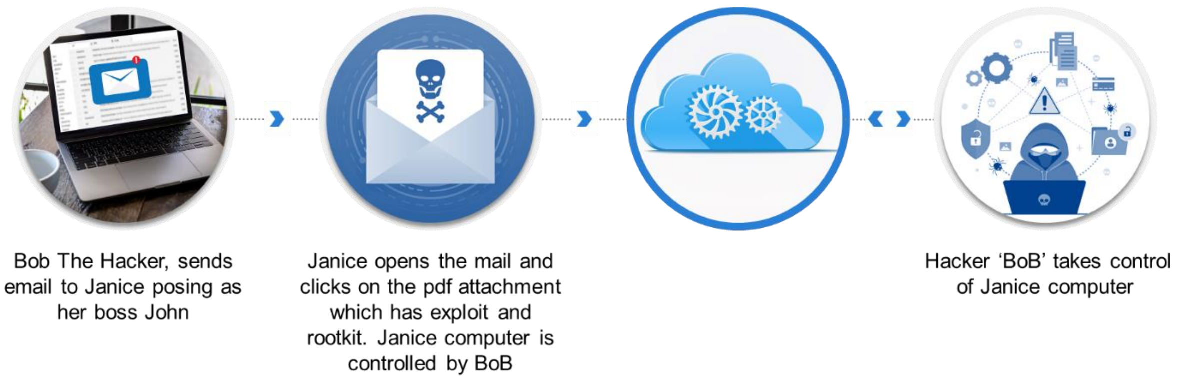
Sl. No.	Acronyms	Full Form
1.	API	Application Programming Interface
2	CDR	Content Disarm and Reconstruction
3	PDF	Portable Document Format
4	DOC	Document
5	KYC	Know Your Customer

Market Trends & Challenges

Cybercriminals find creative new ways of misdirection and obfuscation. Hackers have recently been using PDF/DOC/DOCX/IMAGE files in new and very lethal ways. Additionally, Ransomware groups have been responsible for infecting hundreds of servers with malware to gain corporate data or digital damage systems, essentially spreading misery to individuals and hospitals, businesses, government agencies and more all over the world. Weaponization of PDF/DOC/DOCX is one such new threat.

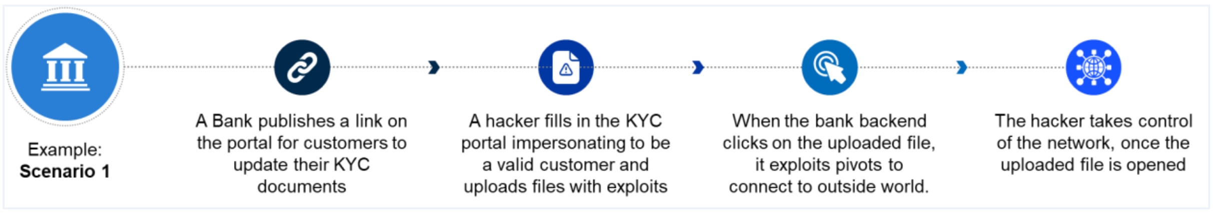
Sharing below some examples of how exploits are being implemented by hackers:

PDF/DOC exploits:

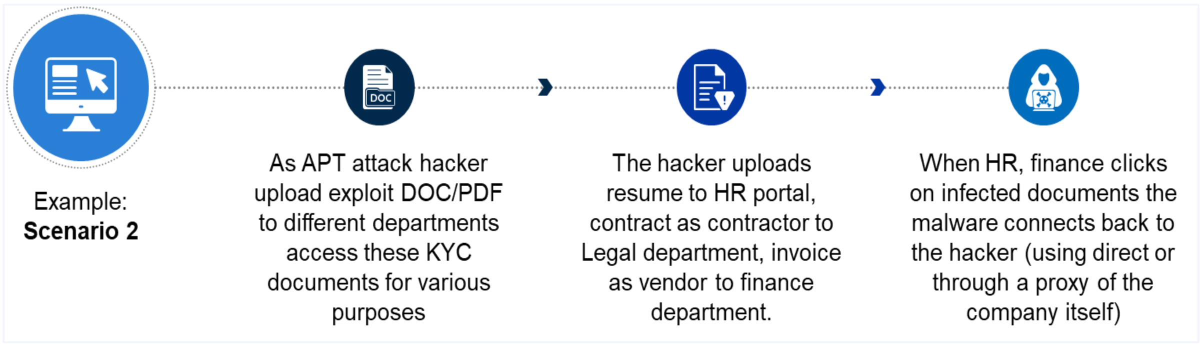

Bob The Hacker, sends email to Janice posing as her boss John
Janice opens the mail and clicks on the pdf attachment which has exploit and rootkit. Janice computer is controlled by BoB
Hacker 'BoB' takes control of Janice computer

The above example shows how Janice opens an email from a hacker who poses himself as her boss. Janice assumes the mail to be from her boss and clicks on the PDF attachment which has an exploit. The moment Janice clicked on the attachment in the mail, the hacker took control of Janice's computer through cloud. This is a highly likely scenario in any organisation/industry.

File Upload


Example:
Scenario 1
A Bank publishes a link on the portal for customers to update their KYC documents
A hacker fills in the KYC portal impersonating to be a valid customer and uploads files with exploits
When the bank backend clicks on the uploaded file, it exploits pivots to connect to outside world.
The hacker takes control of the network, once the uploaded file is opened

In the above-mentioned scenario, with KYC being the prime requirement in any bank, when a Bank publishes a link on their portal for customers to update their KYC documents, a hacker, impersonating to be a valid customer fills the KYC form and uploads files with exploits. The hacker takes control once the Bank, assuming the user to be a valid customer clicks on the uploaded file.


Example:
Scenario 2
DOC
As APT attack hacker upload exploit DOC/PDF to different departments access these KYC documents for various purposes
The hacker uploads resume to HR portal, contract as contractor to Legal department, invoice as vendor to finance department.
When HR, finance clicks on infected documents the malware connects back to the hacker (using direct or through a proxy of the company itself)

In the above scenario, an APT attack hacker, in order to get access to various departments of an organization, will upload exploit DOC/PDF. For example, the hacker might upload resume to HR portal, a contract to the Legal department in the guise of a contractor, an invoice to the finance department as a vendor. When any of these departments' representatives click on the infected attachment, the malware immediately connects back to the hacker. The hacker get access to the organization network.

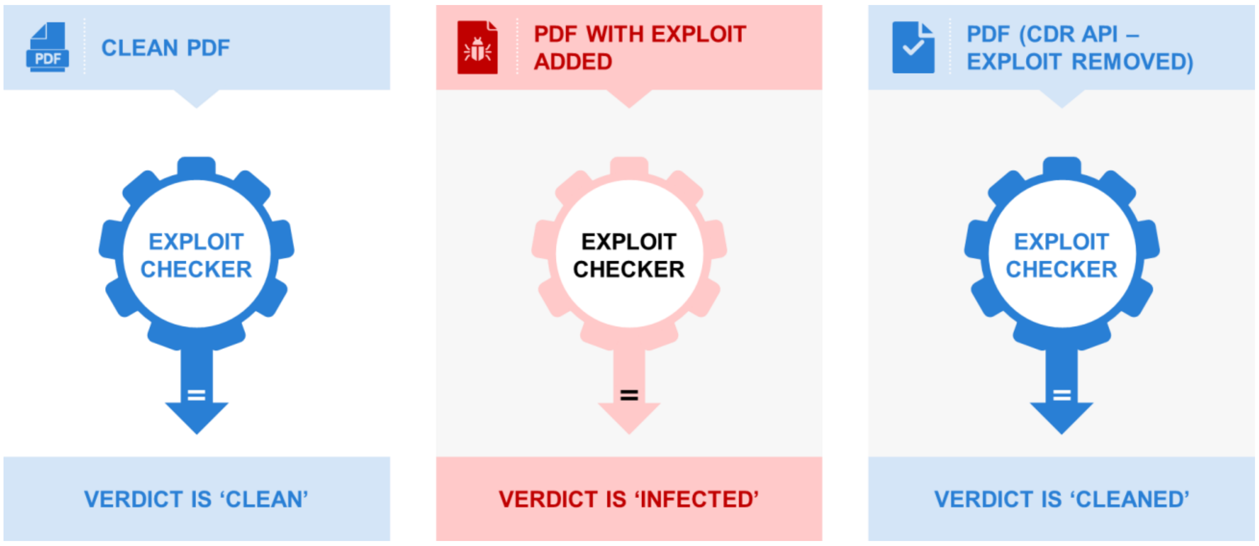
Content Disarm and Reconstruction (CDR) – The Solution

To help organisations from situations mentioned above, CDR (Content Disarm and Reconstruction) also known as Threat Extraction, proactively protects against known and unknown threats contained in documents/image/pdf by removing malware, exploits. The WS7 API is written using the CDR method. WS7PDF is primarily for pdf file. In the process of checking for malicious files/exploits, the pdf document is disarmed and reconstructed. This enables CDR to offer true zero-day prevention, while delivering files to users/customers quickly.

WS7IMG is primarily for image files with image extensions such as TIFF, JPG, PNG, GIF, BMP etc. A unique technology is used to disable the malicious stegano malware which could be hidden inside the image. The disarmed image is optionally given as a JPG file.

Shown below are two examples for easy understanding:

Example 1: PDF Exploit

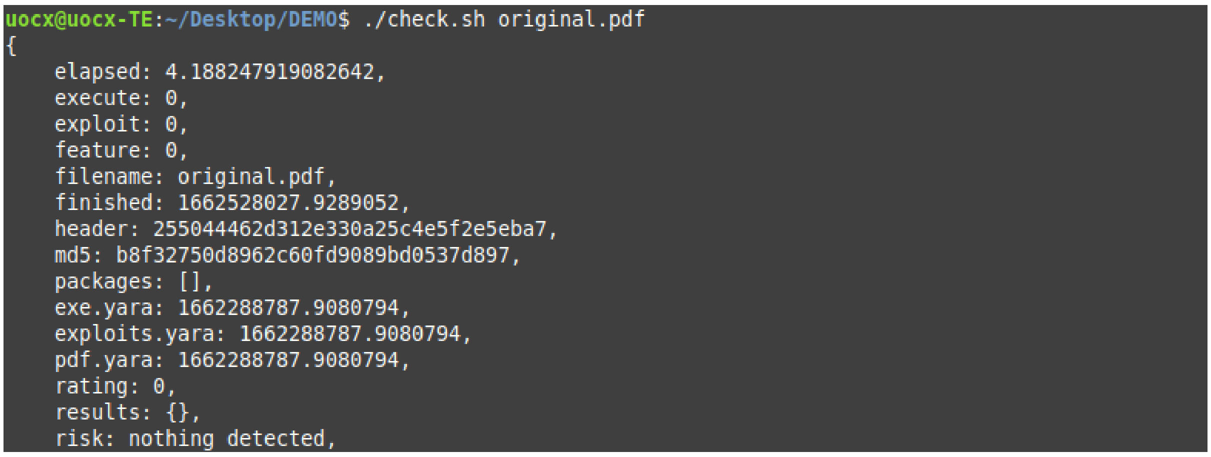


In the above example, there are 3 scenarios that display what the verdict of the exploit checker is when a clean PDF is passed, when a PDF with exploit added is passed and the verdict when the PDF goes through the CDR API. In the first scenario above, the verdict shows as 'CLEAN' for a PDF with no exploit. Similarly, when an exploit was added to a PDF, the verdict shows as 'INFECTED' by the exploit checker. Finally, when the PDF file is run through CDR engine, the exploit is removed and verdict shows as 'CLEANED'.

The below screenshots show the backend query that the CDR engine runs for different scenarios.

1) **When we analyse a clean file (fig 1 – original.pdf), the output shows as 'Nothing detected':**

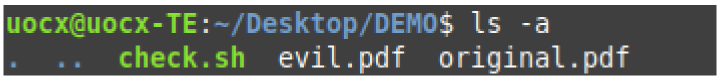
Fig: 1



```
uocx@uocx-TE:~/Desktop/DEMO$ ./check.sh original.pdf
{
    elapsed: 4.188247919082642,
    execute: 0,
    exploit: 0,
    feature: 0,
    filename: original.pdf,
    finished: 1662528027.9289052,
    header: 255044462d312e330a25c4e5f2e5eba7,
    md5: b8f32750d8962c60fd9089bd0537d897,
    packages: [],
    exe.yara: 1662288787.9080794,
    exploits.yara: 1662288787.9080794,
    pdf.yara: 1662288787.9080794,
    rating: 0,
    results: {},
    risk: nothing detected,
```

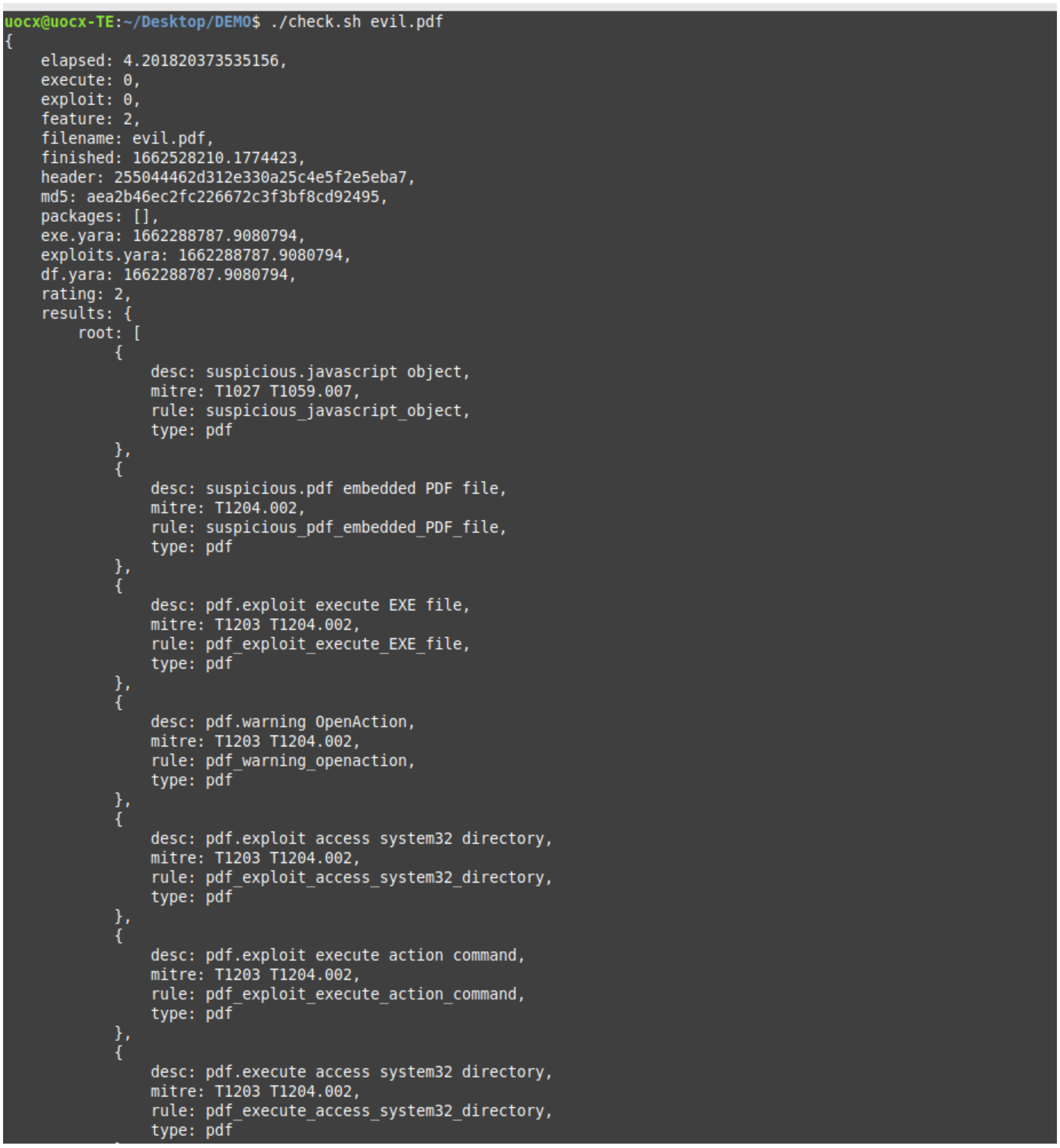
2) **We analyse evil.pdf (fig 2), and when an exploit is detected, the exploit checker shows the number of malicious content present (Fig 3) and gives a comment as 'high risk active content' (Fig 4):**

Fig: 2



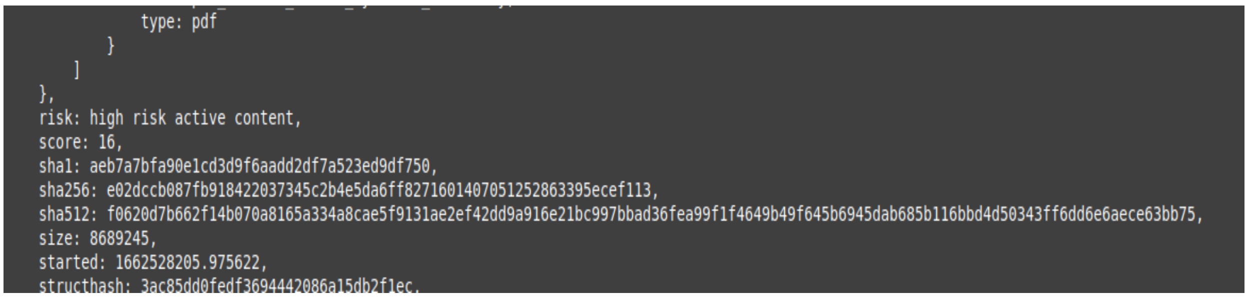
```
uocx@uocx-TE:~/Desktop/DEMO$ ls -a
.  ..  check.sh  evil.pdf  original.pdf
```

Fig 3



```
uocx@uocx-TE:~/Desktop/DEMO$ ./check.sh evil.pdf
{
    elapsed: 4.201820373535156,
    execute: 0,
    exploit: 0,
    feature: 2,
    filename: evil.pdf,
    finished: 1662528210.1774423,
    header: 255044462d312e330a25c4e5f2e5eba7,
    md5: aea2b46ec2fc226672c3f3bf8cd92495,
    packages: [],
    exe.yara: 1662288787.9080794,
    exploits.yara: 1662288787.9080794,
    df.yara: 1662288787.9080794,
    rating: 2,
    results: {
        root: [
            {
                desc: suspicious.javascript object,
                mitre: T1027 T1059.007,
                rule: suspicious_javascript_object,
                type: pdf
            },
            {
                desc: suspicious.pdf embedded PDF file,
                mitre: T1204.002,
                rule: suspicious_pdf_embedded_PDF_file,
                type: pdf
            },
            {
                desc: pdf.exploit execute EXE file,
                mitre: T1203 T1204.002,
                rule: pdf_exploit_execute_EXE_file,
                type: pdf
            },
            {
                desc: pdf.warning OpenAction,
                mitre: T1203 T1204.002,
                rule: pdf_warning_openaction,
                type: pdf
            },
            {
                desc: pdf.exploit access system32 directory,
                mitre: T1203 T1204.002,
                rule: pdf_exploit_access_system32_directory,
                type: pdf
            },
            {
                desc: pdf.exploit execute action command,
                mitre: T1203 T1204.002,
                rule: pdf_exploit_execute_action_command,
                type: pdf
            },
            {
                desc: pdf.execute access system32 directory,
                mitre: T1203 T1204.002,
                rule: pdf_execute_access_system32_directory,
                type: pdf
```

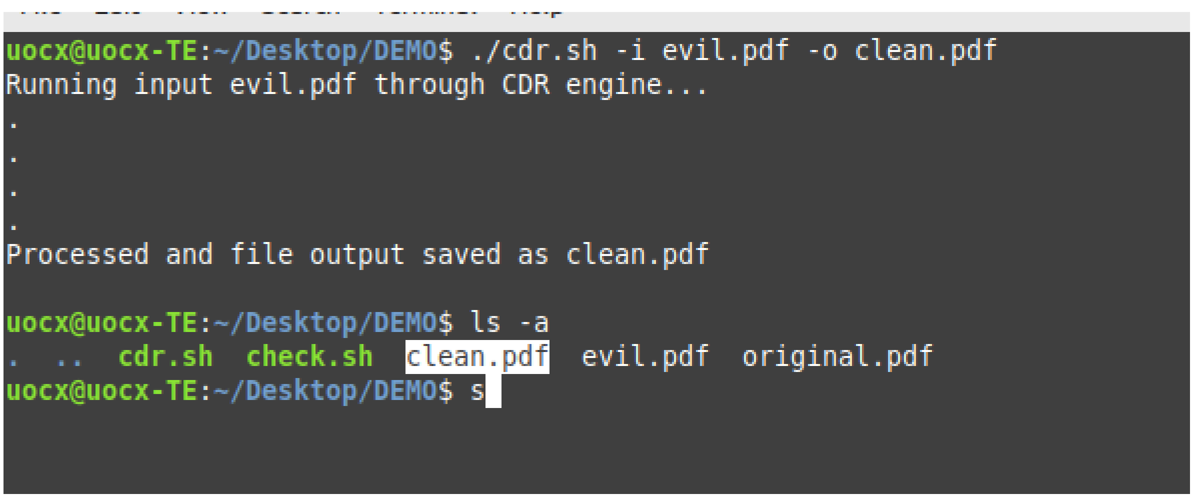
Fig 4



```
            type: pdf
        }
    ]
},
risk: high risk active content,
score: 16,
sha1: aeb7a7bfa90e1cd3d9f6aadd2df7a523ed9df750,
sha256: e02dccb087fb918422037345c2b4e5da6ff8271601407051252863395ecef113,
sha512: f0620d7b662f14b070a8165a334a8cae5f9131ae2ef42dd9a916e21bc997bbad36fea99f1f4649b49f645b6945dab685b116bbd4d50343ff6dd6e6aece63bb75,
size: 8689245,
started: 1662528205.975622,
structhash: 3ac85dd0fedf3694442086a15db2f1ec,
```

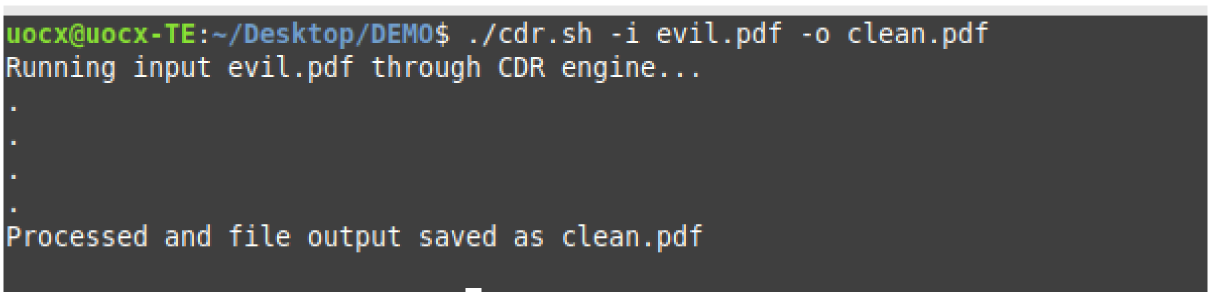
3) **When Exploit Checker runs the CDR engine (Fig 5), removes malicious content, disassembles the file (fig 6) and shares a clean pdf as an output. Fig 7 shows the final result as clean pdf with no exploits detected:**

Fig 5



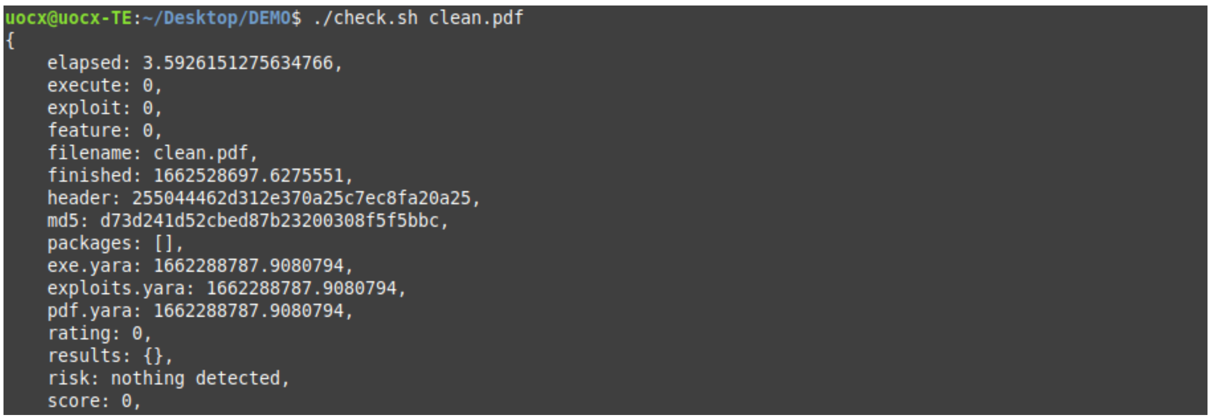
```
uocx@uocx-TE:~/Desktop/DEMO$ ./cdr.sh -i evil.pdf -o clean.pdf
Running input evil.pdf through CDR engine...
.
.
.
.
Processed and file output saved as clean.pdf

uocx@uocx-TE:~/Desktop/DEMO$ ls -a
.  ..  cdr.sh  check.sh  clean.pdf  evil.pdf  original.pdf
uocx@uocx-TE:~/Desktop/DEMO$ s
```

Fig 6



```
uocx@uocx-TE:~/Desktop/DEMO$ ./cdr.sh -i evil.pdf -o clean.pdf
Running input evil.pdf through CDR engine...
.
.
.
.
Processed and file output saved as clean.pdf
```

Fig 7

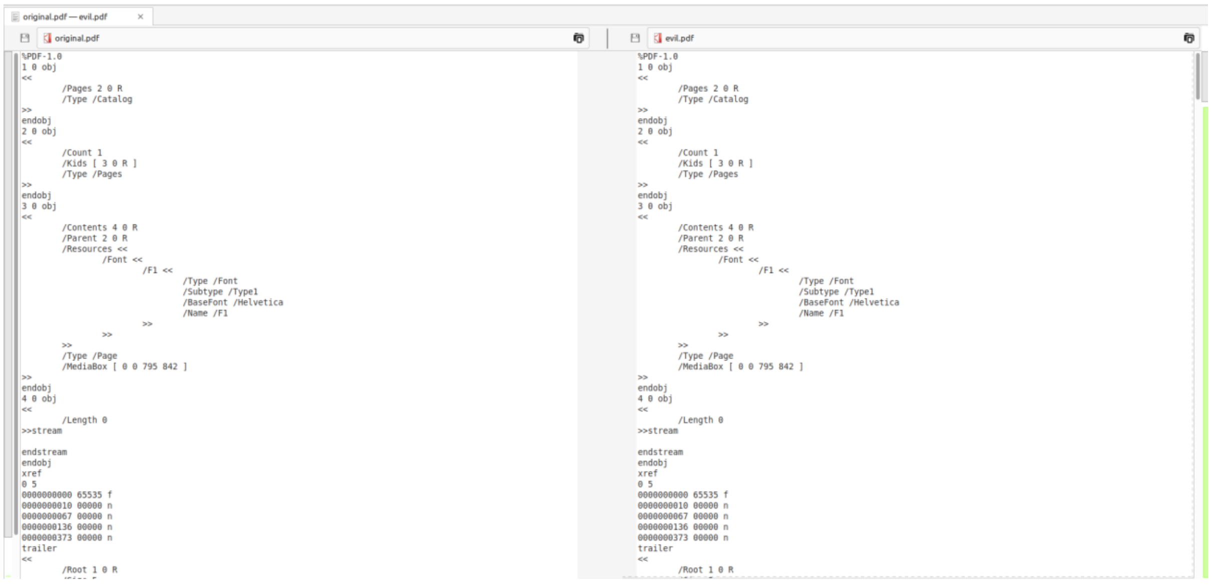


```
uocx@uocx-TE:~/Desktop/DEMO$ ./check.sh clean.pdf
{
    elapsed: 3.5926151275634766,
    execute: 0,
    exploit: 0,
    feature: 0,
    filename: clean.pdf,
    finished: 1662528697.6275551,
    header: 255044462d312e370a25c7ec8fa20a25,
    md5: d73d241d52cbed87b23200308f5f5bbc,
    packages: [],
    exe.yara: 1662288787.9080794,
    exploits.yara: 1662288787.9080794,
    pdf.yara: 1662288787.9080794,
    rating: 0,
    results: {},
    risk: nothing detected,
    score: 0,
```

4) Comparison of Clean file Vs File with Exploits

Please note, the below screenshot shows the original pdf on the left side and the file with exploits (evil.pdf) on the right side. On the original pdf, there is no scroll bar as the file is only that long as visible. However, on the evil.pdf, with exploits in the file, the scroll for the entire file is longer than the original.

Fig 8

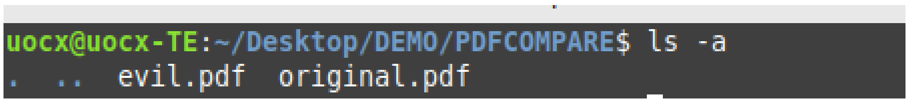


```
%PDF-1.0
1 0 obj
<<
        /Pages 2 0 R
        /Type /Catalog
>>
endobj
2 0 obj
<<
        /Count 1
        /Kids [ 3 0 R ]
        /Type /Pages
>>
endobj
3 0 obj
<<
        /Contents 4 0 R
        /Parent 2 0 R
        /Resources <<
                /Font <<
                        /F1 <<
                                /Type /Font
                                /Subtype /Type1
                                /BaseFont /Helvetica
                                /Name /F1
                        >>
                >>
        >>
        /Type /Page
        /MediaBox [ 0 0 795 842 ]
>>
endobj
4 0 obj
<<
        /Length 0
>>stream

endstream
endobj
xref
0 5
0000000000 65535 f
0000000010 00000 n
0000000067 00000 n
0000000136 00000 n
0000000373 00000 n
trailer
<<
        /Root 1 0 R
```

```
%PDF-1.0
1 0 obj
<<
        /Pages 2 0 R
        /Type /Catalog
>>
endobj
2 0 obj
<<
        /Count 1
        /Kids [ 3 0 R ]
        /Type /Pages
>>
endobj
3 0 obj
<<
        /Contents 4 0 R
        /Parent 2 0 R
        /Resources <<
                /Font <<
                        /F1 <<
                                /Type /Font
                                /Subtype /Type1
                                /BaseFont /Helvetica
                                /Name /F1
                        >>
                >>
        >>
        /Type /Page
        /MediaBox [ 0 0 795 842 ]
>>
endobj
4 0 obj
<<
        /Length 0
>>stream

endstream
endobj
xref
0 5
0000000000 65535 f
0000000010 00000 n
0000000067 00000 n
0000000136 00000 n
0000000373 00000 n
trailer
<<
        /Root 1 0 R
```

When we analyze evil.pdf (fig 9), shows the below message

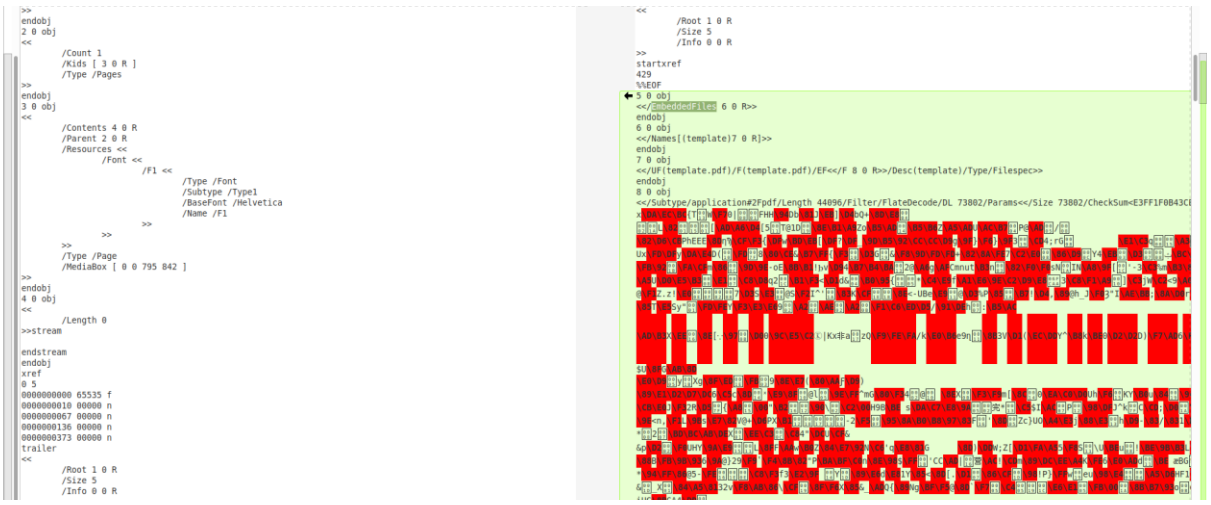
Fig 9



```
uocx@uocx-TE:~/Desktop/DEMO/PDFCOMPARE$ ls -a
.  ..  evil.pdf  original.pdf
```

The image on the right side highlighted in red shows the embedded content with exploits in a binary data format (fig 10 & 11)

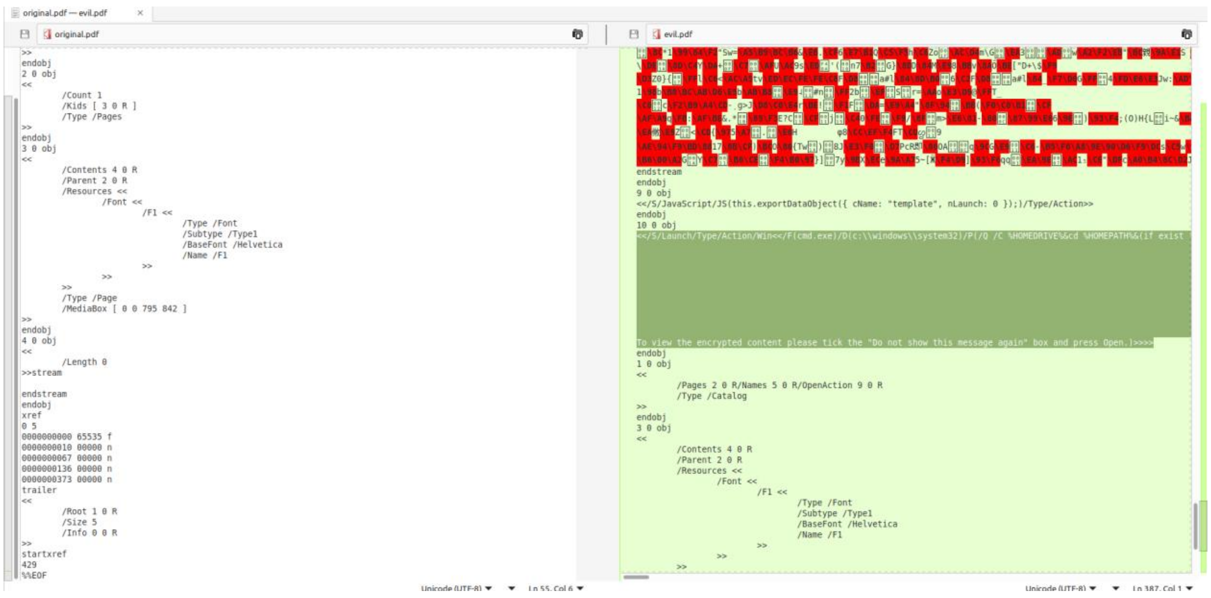
Fig 10



```
>>
endobj
2 0 obj
<<
        /Count 1
        /Kids [ 3 0 R ]
        /Type /Pages
>>
endobj
3 0 obj
<<
        /Contents 4 0 R
        /Parent 2 0 R
        /Resources <<
                /Font <<
                        /F1 <<
                                /Type /Font
                                /Subtype /Type1
                                /BaseFont /Helvetica
                                /Name /F1
                        >>
                >>
        >>
        /Type /Page
        /MediaBox [ 0 0 795 842 ]
>>
endobj
4 0 obj
<<
        /Length 0
>>stream

endstream
endobj
xref
0 5
0000000000 65535 f
0000000010 00000 n
0000000067 00000 n
0000000136 00000 n
0000000373 00000 n
trailer
<<
        /Root 1 0 R
        /Size 5
        /Info 0 0 R
```

```
<<
        /Root 1 0 R
        /Size 5
        /Info 0 0 R
>>
startxref
429
%%EOF
5 0 obj
<</EmbeddedFiles 6 0 R>>
endobj
6 0 obj
<</Names[(template)7 0 R]>>
endobj
7 0 obj
<</UF(template.pdf)/F(template.pdf)/EF<</F 8 0 R>>/Desc(template)/Type/Filespec>>
endobj
8 0 obj
<</Subtype/application#2Fpdf/Length 44096/Filter/FlateDecode/DL 73802/Params<</Size 73802/CheckSum<E3FF1F0B43CE
[illegible]
```

Fig 11

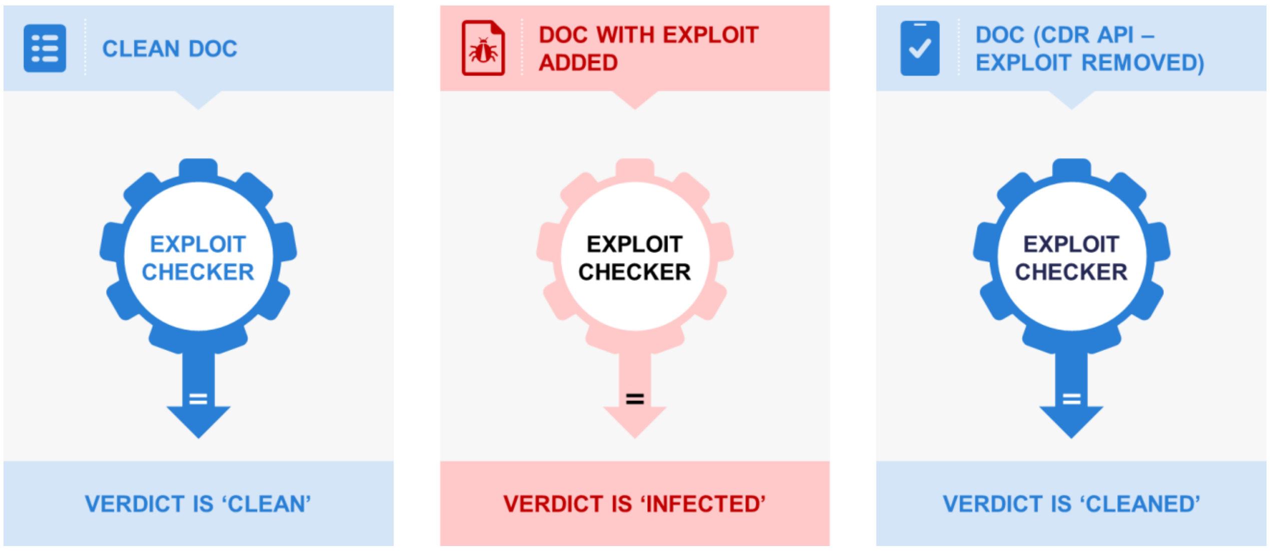


```
>>
endobj
2 0 obj
<<
        /Count 1
        /Kids [ 3 0 R ]
        /Type /Pages
>>
endobj
3 0 obj
<<
        /Contents 4 0 R
        /Parent 2 0 R
        /Resources <<
                /Font <<
                        /F1 <<
                                /Type /Font
                                /Subtype /Type1
                                /BaseFont /Helvetica
                                /Name /F1
                        >>
                >>
        >>
        /Type /Page
        /MediaBox [ 0 0 795 842 ]
>>
endobj
4 0 obj
<<
        /Length 0
>>stream

endstream
endobj
xref
0 5
0000000000 65535 f
0000000010 00000 n
0000000067 00000 n
0000000136 00000 n
0000000373 00000 n
trailer
<<
        /Root 1 0 R
        /Size 5
        /Info 0 0 R
>>
startxref
429
%%EOF
```

```
[illegible]
endstream
endobj
9 0 obj
<</S/JavaScript/JS(this.exportDataObject({ cName: "template", nLaunch: 0 });)/Type/Action>>
endobj
10 0 obj
<</S/Launch/Type/Action/Win<</F(cmd.exe)/D(c:\\windows\\system32)/P(/Q /C %HOMEDRIVE%&cd %HOMEPATH%&(if exist

To view the encrypted content please tick the "Do not show this message again" box and press Open.)>>>>
endobj
1 0 obj
<<
        /Pages 2 0 R/Names 5 0 R/OpenAction 9 0 R
        /Type /Catalog
>>
endobj
3 0 obj
<<
        /Contents 4 0 R
        /Parent 2 0 R
        /Resources <<
                /Font <<
                        /F1 <<
                                /Type /Font
                                /Subtype /Type1
                                /BaseFont /Helvetica
                                /Name /F1
                        >>
                >>
        >>
```

Example 1 and 2 shows how the CDR engine works to disassemble the file and reconstructs a clean pdf without exploits

Example 2: DOC/DOCX Exploit

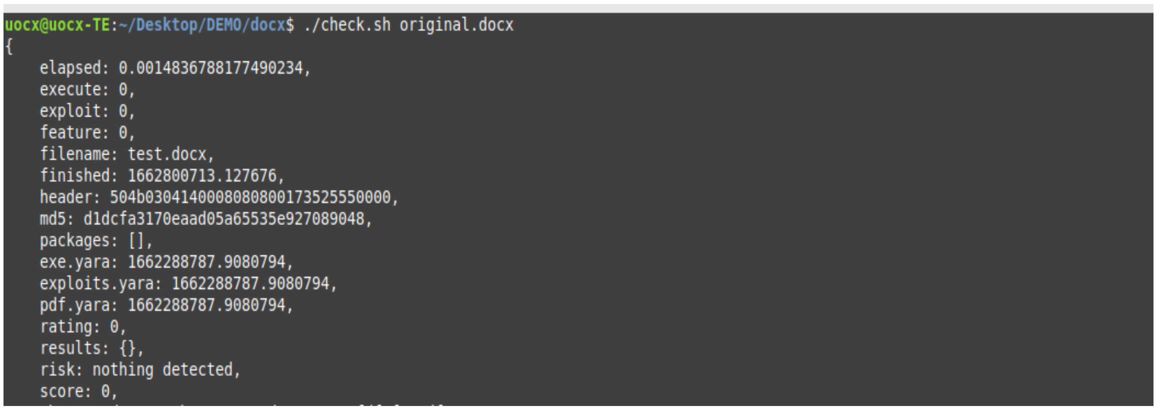


Similar to the previous example, in the current one, there are 3 scenarios that display what the verdict of the exploit checker is when a clean DOC/DOCX file is passed through the exploit checker. In the first scenario above, the verdict shows as 'CLEAN' for a DOC file with no exploit. Likewise, when an exploit was added to a DOC, the verdict shows as 'INFECTED' by the exploit checker. Finally, when the DOC file is run through CDR, the exploit is removed and verdict shows as 'CLEANED' by the exploit checker.

The below screenshots show the backend query that the CDR engine runs for different scenarios for a DOC file:

1) **When we analyse a clean file (fig 12 – original.docx), the output shows as 'Nothing detected':**

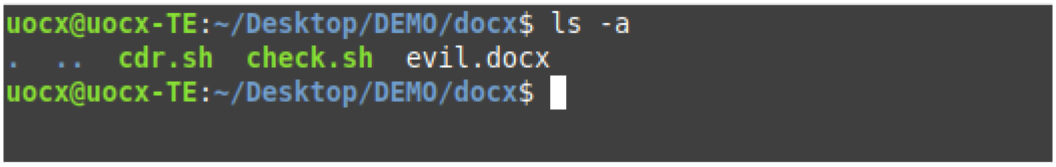
Fig 12



```
uocx@uocx-TE:~/Desktop/DEMO/docx$ ./check.sh original.docx
{
    elapsed: 0.0014836788177490234,
    execute: 0,
    exploit: 0,
    feature: 0,
    filename: test.docx,
    finished: 1662800713.127676,
    header: 504b0304140008080800173525550000,
    md5: d1dcfa3170eaad05a65535e927089048,
    packages: [],
    exe.yara: 1662288787.9080794,
    exploits.yara: 1662288787.9080794,
    pdf.yara: 1662288787.9080794,
    rating: 0,
    results: {},
    risk: nothing detected,
    score: 0,
```

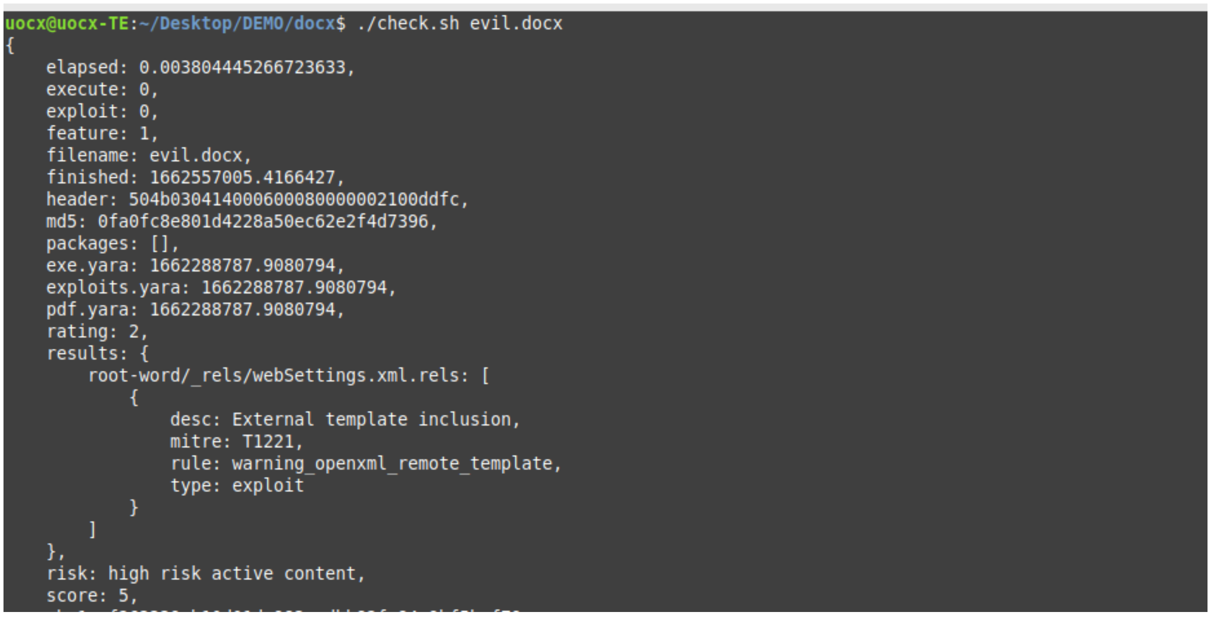
2) **We analyse evil.docx (fig 13). When an exploit is detected in a doc file, the exploit checker displays the number of malicious content present with the risk comment showing as ‘high risk active content’ (fig 14):**

Fig 13



```
uocx@uocx-TE:~/Desktop/DEMO/docx$ ls -a
.  ..  cdr.sh  check.sh  evil.docx
uocx@uocx-TE:~/Desktop/DEMO/docx$ 
```

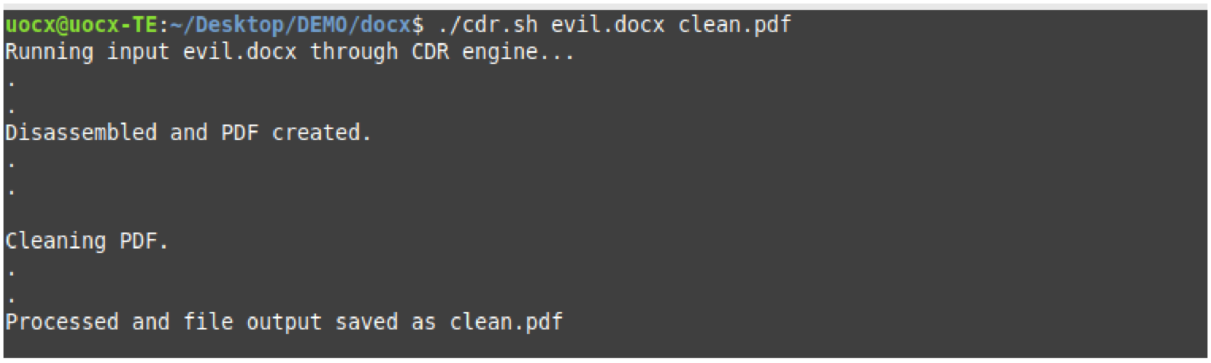
Fig 14



```
uocx@uocx-TE:~/Desktop/DEMO/docx$ ./check.sh evil.docx
{
    elapsed: 0.003804445266723633,
    execute: 0,
    exploit: 0,
    feature: 1,
    filename: evil.docx,
    finished: 1662557005.4166427,
    header: 504b030414000600080000002100ddfc,
    md5: 0fa0fc8e801d4228a50ec62e2f4d7396,
    packages: [],
    exe.yara: 1662288787.9080794,
    exploits.yara: 1662288787.9080794,
    pdf.yara: 1662288787.9080794,
    rating: 2,
    results: {
        root-word/_rels/webSettings.xml.rels: [
            {
                desc: External template inclusion,
                mitre: T1221,
                rule: warning_openxml_remote_template,
                type: exploit
            }
        ]
    },
    risk: high risk active content,
    score: 5,
```

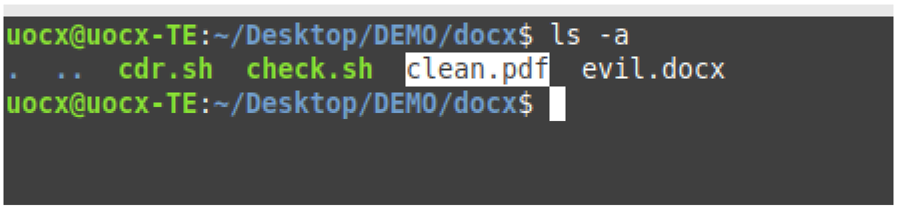
3) **When an infected file is run through the CDR engine (fig 15), the file is disassembled and a clean PDF is created as an output (fig 16). The final output shows that there is ‘nothing detected’ with a score of 0 (fig 17)**

Fig 15



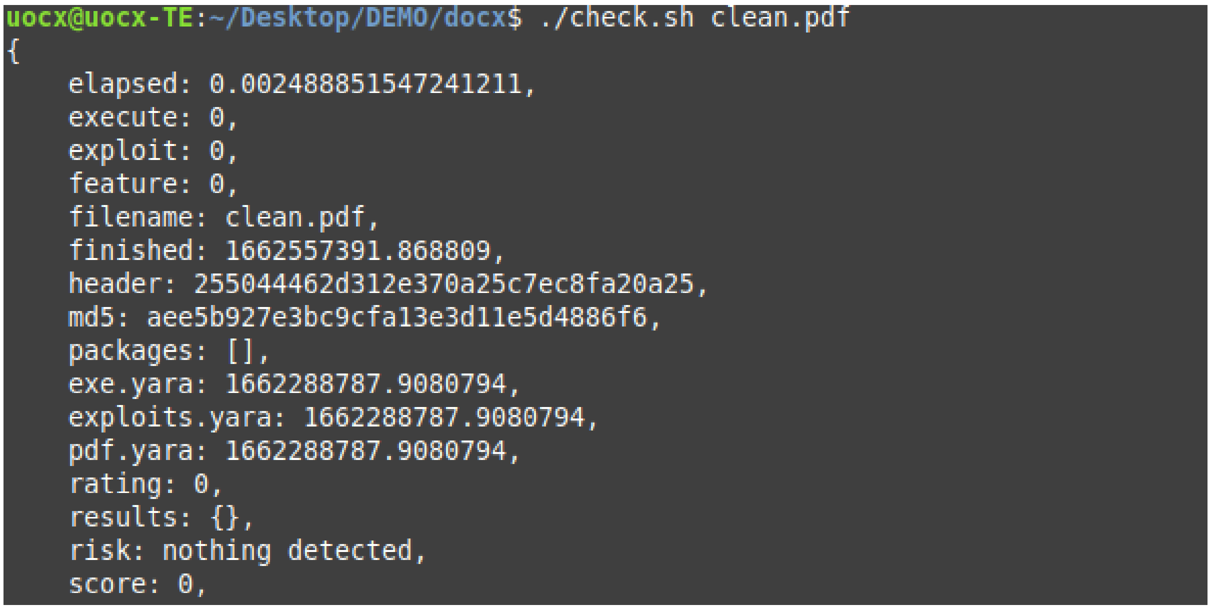
```
uocx@uocx-TE:~/Desktop/DEMO/docx$ ./cdr.sh evil.docx clean.pdf
Running input evil.docx through CDR engine...
.
.
Disassembled and PDF created.
.
.

Cleaning PDF.
.
.
Processed and file output saved as clean.pdf
```

Fig 16



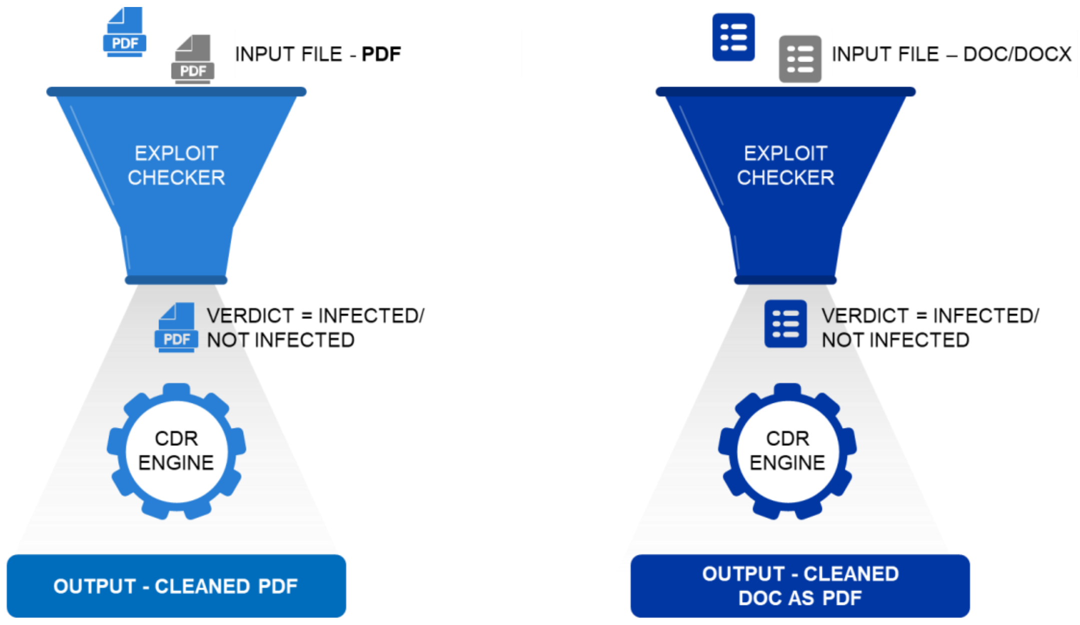
```
uocx@uocx-TE:~/Desktop/DEMO/docx$ ls -a
.  ..  cdr.sh  check.sh  clean.pdf  evil.docx
uocx@uocx-TE:~/Desktop/DEMO/docx$ 
```

Fig 17



```
uocx@uocx-TE:~/Desktop/DEMO/docx$ ./check.sh clean.pdf
{
    elapsed: 0.002488851547241211,
    execute: 0,
    exploit: 0,
    feature: 0,
    filename: clean.pdf,
    finished: 1662557391.868809,
    header: 255044462d312e370a25c7ec8fa20a25,
    md5: aee5b927e3bc9cfa13e3d11e5d4886f6,
    packages: [],
    exe.yara: 1662288787.9080794,
    exploits.yara: 1662288787.9080794,
    pdf.yara: 1662288787.9080794,
    rating: 0,
    results: {},
    risk: nothing detected,
    score: 0,
```

CDR API – The Technology



To summarize, when a customer subscribes to CDR API, any attachment that comes from external source in the form of PDF/DOC/DOCX file, the exploit checker processes the input file (PDF/DOC/DOCX) through the CDR engine and shares a cleaned file (PDF/DOC/DOCX) as an output.

Conclusion

The next time someone sends you an email with a PDF/doc/docx attachment, with the CDR subscription, one need not worry before clicking to open the file. CDR ensures to remove the exploits, disarm the file and reconstruct a clean file. This solution is highly recommended for all web applications, mobile applications and O365.

We will be happy to show a technical demo of the above solution, on request.