# STATIC MALWARE ANALYSIS

## Pre-Lab

Technology advancement is rapidly growing as new innovations are developed and released every year. This improves our daily lives as our activities are automated and made easier. As we continue to enjoy the good side of this wave, it is vital to be aware of the other side where cybercriminals continue to thrive at the same rate as technological growth. Malicious software well known as malware is becoming a widespread threat that can cause damage to any small or large networks. Malware is defined as unwanted malicious software that is intended to harm the user or the target system. Malware comes in different types such as trojans, viruses, backdoors et cetera (Manuel, Theodoor, Engin, & Christopher, 2012).

In this paper we delve into static malware analysis where we detail a case study of an infected computer showing the paramount steps to be followed. Malware analysis is the process of understanding the behavior and purpose of suspicious software. This comprises of the breakdown of its core components and source code, investigating its character and functionality to trace the origin and impact and therefore come up with mitigation and prevention plans for future occurrence (Chiradeep, 2021).

Static malware analysis examines the malicious program without executing the code as we will be doing in the steps below. We received a call from one of our clients from Help company where their computer was infected by malware.

## Initialization

We received a call from Help company where one of their computers was infected. During the call we documented and advised them to isolate the computer from the network to prevent further infection to other devices. This stage also included the backing up of crucial information to an external storage device. After isolation and backing up we obtain the malware sample that we will be examining.

## File identification

During this step we identified the type of file e.g. executable or script that could have been infected with malicious content. Since the computer was running a windows operating system, we used the windows portable executable file identifier which is used to provide information about the executable file in equation. We found an executable file called clean.exe which was superbly detailed by the PE tool in the headers and sections (Debarshi, 2023).

## Hashing

We ran the clean.exe file into virustotal tool https://www.virustotal.com/gui/home/upload to check the hashes of the file. This tool is used for analyzing suspicious files, domains, internet protocols (IP) to detect for malware (VTDOC, n.d.). These hashes are paramount to uniquely identify the file and compare it against other databases which provide more details on the executable file at hand.

## Metadata Etraction

We then extracted the metadata of the file to gain more insights and details such as the timestamp, creation date etc. Using the PEview tool we checked the import table, export tables and other relevant headers (Joshua, 2014).

## String Analysis

Using strings as a command line tool we extracted human readable strings from the file which revealed URL, IP addresses and other content of the malware.

## Disassembly

This phase includes the breakdown of the malware to understand how the malware works without executing it. We focused on file system interaction using IDA pro tool which is a well-known disassembler and debugger that provides powerful analysis capabilities (Philip, 2022).

## Threat Intelligence

After gathering the artifacts of the malware using the tools above, we cross referenced by comparing them to other known platforms such as virustotal to detect whether the malware was discovered and recorded.

## Indicator of Compromise

After the analysis we generated IOCs to show that the computer was compromised using the hashes, Ip addresses, domain names and URLs associated with the malware (Agofee, 2022).

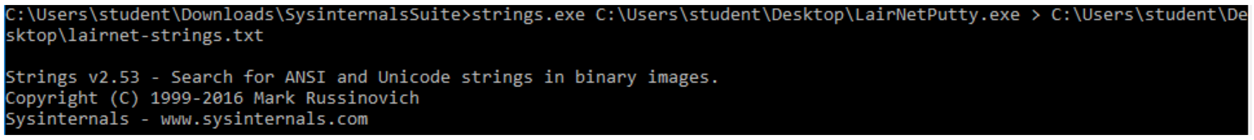
## Document your Findings.

We prepared a detailed report touching on the behavior of the malware based on analysis and the recommendations such as installing anti-viruses and patching the computer up to date.

## Background

### 1.Analysis of Suspicious Executable

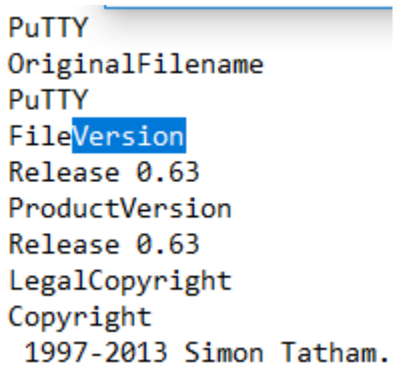
We will be using strings, which as mentioned above is a command line tool that we will run one of the executable files in our lab called LairNetPutty.exe. We will be saving the output in a document file we created as shown in the screenshot below.



```
C:\Users\student\Downloads\SysinternalsSuite>strings.exe C:\Users\student\Desktop\LairNetPutty.exe > C:\Users\student\De
sktop\lairnet-strings.txt

Strings v2.53 - Search for ANSI and Unicode strings in binary images.
Copyright (C) 1999-2016 Mark Russinovich
Sysinternals - www.sysinternals.com
```

*Figure 1/strings.*

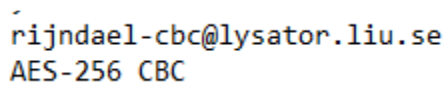
The release version of putty was 0.63 as shown in the screenshot below.



```
PuTTY
OriginalFilename
PuTTY
FileVersion
Release 0.63
ProductVersion
Release 0.63
LegalCopyright
Copyright
 1997-2013 Simon Tatham.
```

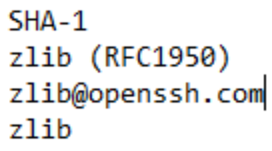
*Figure 2/Version.*

The email addresses contained in relation to an encryption cipher are as follows.



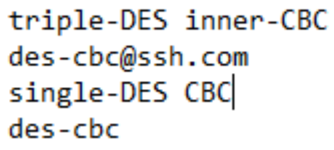
```
rijndael-cbc@lysator.liu.se
AES-256 CBC
```

*Figure 3/Email1.*



```
SHA-1
zlib (RFC1950)
zlib@openssh.com
zlib
```

*Figure 4/Email2.*

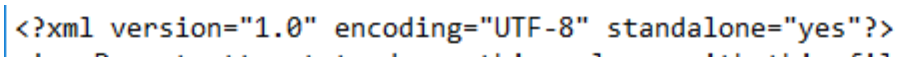


```
triple-DES inner-CBC
des-cbc@ssh.com
single-DES CBC
des-cbc
```

*Figure 5/Email3.*

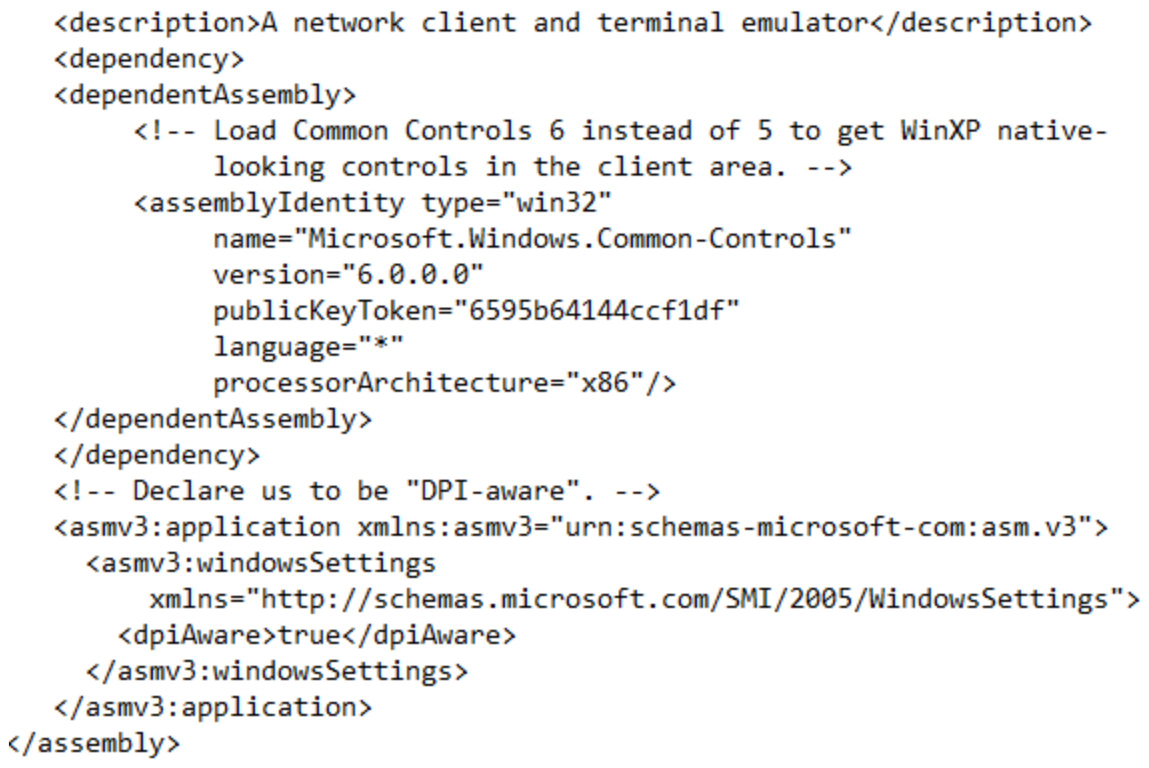
The programming language that this executable was based on were: </assembly>, xml.

Xml also known as extensible markup language is a flexible way used to create common information formats and share them on the world wide web (freecode, 2020).



```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

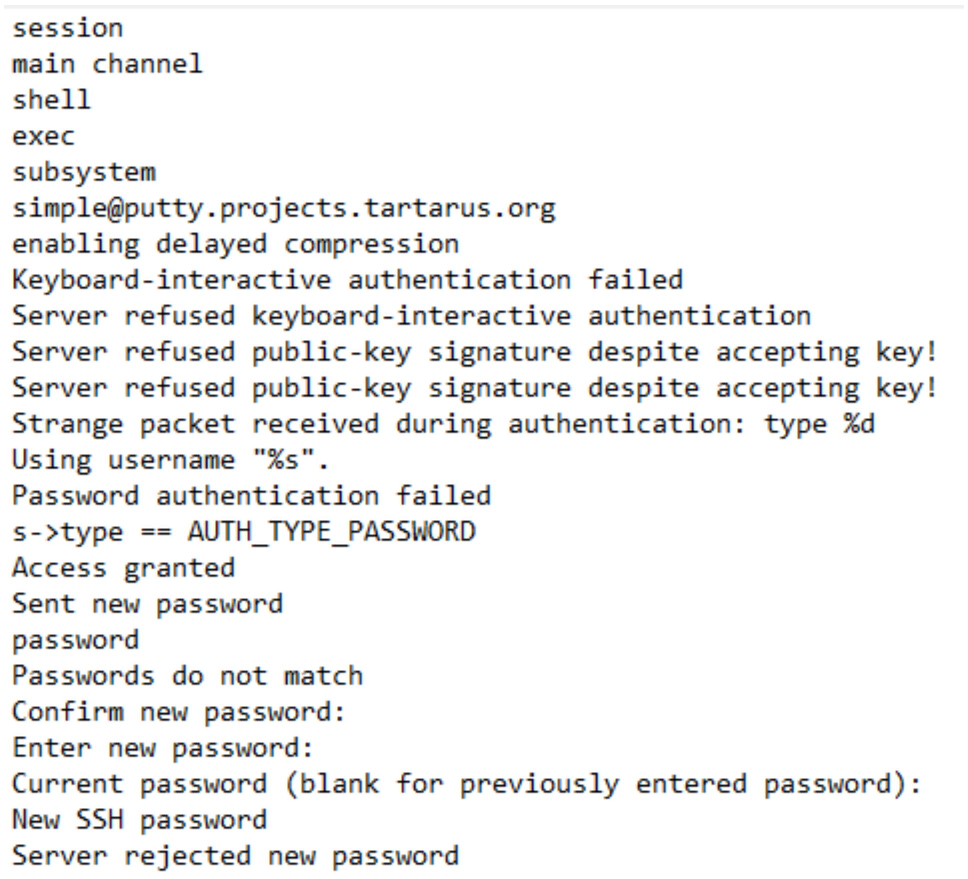
*Figure 6/xml.*



```
   <description>A network client and terminal emulator</description>
   <dependency>
   <dependentAssembly>
        <!-- Load Common Controls 6 instead of 5 to get WinXP native-
             looking controls in the client area. -->
        <assemblyIdentity type="win32"
             name="Microsoft.Windows.Common-Controls"
             version="6.0.0.0"
             publicKeyToken="6595b64144ccf1df"
             language="*"
             processorArchitecture="x86"/>
   </dependentAssembly>
   </dependency>
   <!-- Declare us to be "DPI-aware". -->
   <asmv3:application xmlns:asmv3="urn:schemas-microsoft-com:asm.v3">
     <asmv3:windowsSettings
         xmlns="http://schemas.microsoft.com/SMI/2005/WindowsSettings">
       <dpiAware>true</dpiAware>
     </asmv3:windowsSettings>
   </asmv3:application>
</assembly>
```

*Figure 7/assembly.*

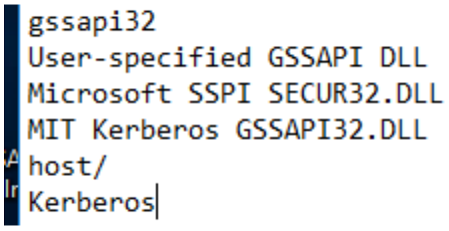
The function exec as shown in the figure below tries to start a remote connection which fails and in turn tries to set up new passwords.



```
session
main channel
shell
exec
subsystem
simple@putty.projects.tartarus.org
enabling delayed compression
Keyboard-interactive authentication failed
Server refused keyboard-interactive authentication
Server refused public-key signature despite accepting key!
Server refused public-key signature despite accepting key!
Strange packet received during authentication: type %d
Using username "%s".
Password authentication failed
s->type == AUTH_TYPE_PASSWORD
Access granted
Sent new password
password
Passwords do not match
Confirm new password:
Enter new password:
Current password (blank for previously entered password):
New SSH password
Server rejected new password
```

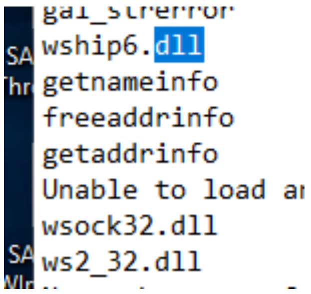
*Figure 8/execution.*

The windows Dynamic Link Libraries that were referenced in our string output were as shown in the screenshots below:



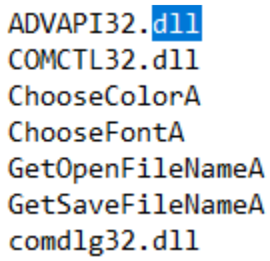
```
gssapi32
User-specified GSSAPI DLL
Microsoft SSPI SECUR32.DLL
MIT Kerberos GSSAPI32.DLL
host/
Kerberos
```

*Figure 9/dll.*



```
gai_strerror
wship6.dll
getnameinfo
freeaddrinfo
getaddrinfo
Unable to load ar
wsock32.dll
ws2_32.dll
```

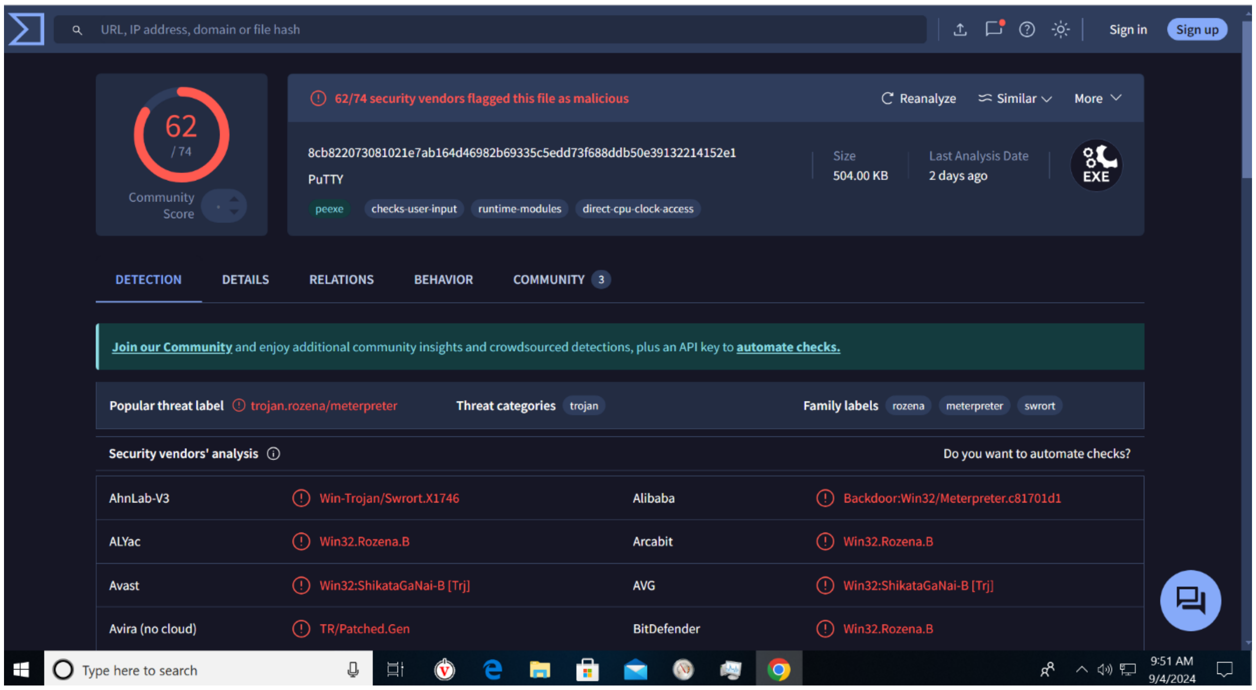
*Figure 10/dll1.*



```
ADVAPI32.dll
COMCTL32.dll
ChooseColorA
ChooseFontA
GetOpenFileNameA
GetSaveFileNameA
comdlg32.dll
```

*Figure 11/dll2.*

We uploaded the executable file in the virustotal tool that we described above and gave the following insights.



62/74 security vendors flagged this file as malicious

8cb822073081021e7ab164d46982b69335c5edd73f688ddb50e39132214152e1
PuTTY
peexe checks-user-input runtime-modules direct-cpu-clock-access

Size: 504.00 KB
Last Analysis Date: 2 days ago

Join our Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

Popular threat label: trojan.rozena/meterpreter
Threat categories: trojan
Family labels: rozena meterpreter swrort

Security vendors' analysis
Do you want to automate checks?

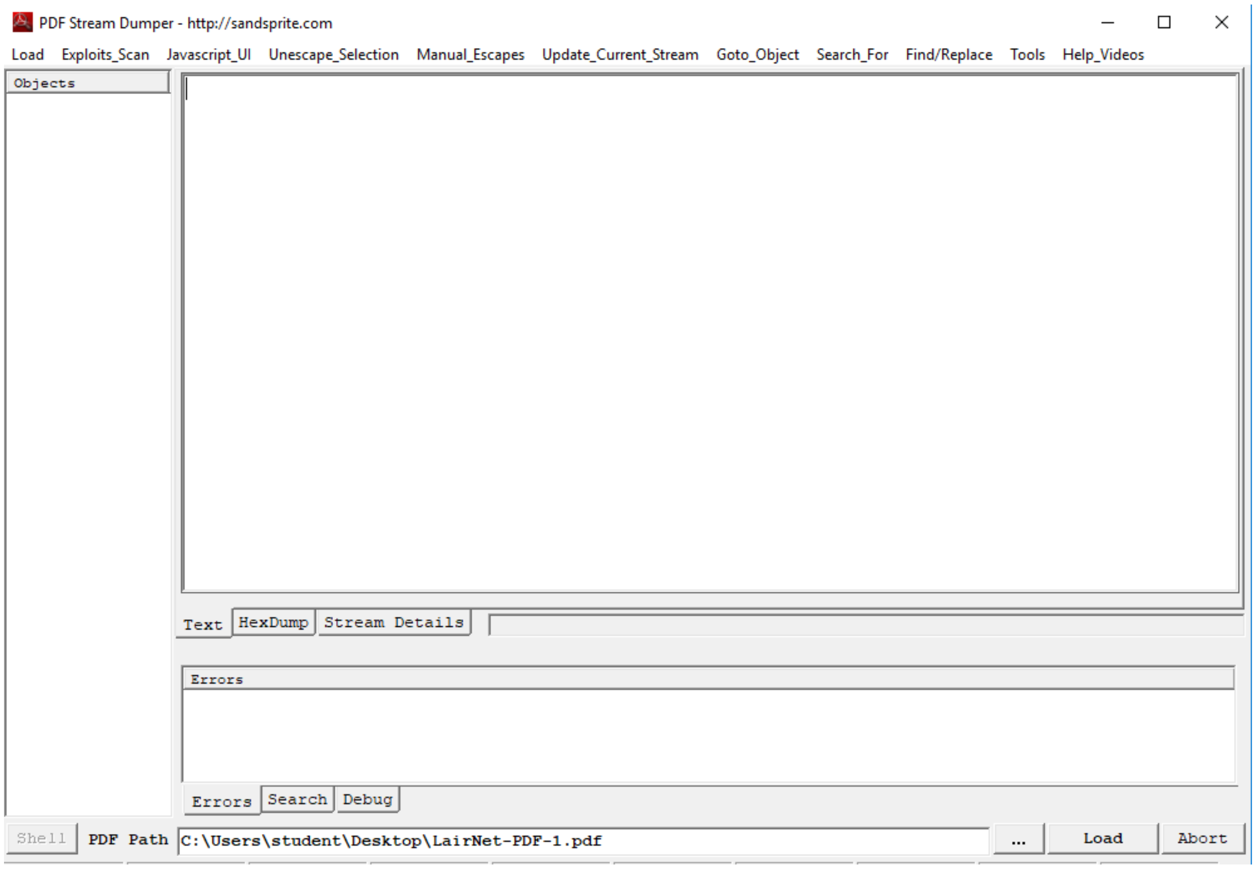
| | | | |
|---|---|---|---|
| AhnLab-V3 | Win-Trojan/Swrort.X1746 | Alibaba | Backdoor:Win32/Meterpreter.c81701d1 |
| ALYac | Win32.Rozena.B | Arcabit | Win32.Rozena.B |
| Avast | Win32:ShikataGaNai-B [Trj] | AVG | Win32:ShikataGaNai-B [Trj] |
| Avira (no cloud) | TR/Patched.Gen | BitDefender | Win32.Rozena.B |

*Figure 12/virustotal.*

The executable file shows that it is malicious from the above results which shows a high number of the contributing community flagging it. The popular threat is trojan. rozena/meterpreter. To shed light on this rozena can is defined as an executable file that masquerades as a Microsoft word file. It is popularly known for creating a reverse TCP connection to remote servers. This is like leaving your phone unlocked to strangers allowing them to access all the information. Meterpreter is defined as a malicious program that allows adversaries to remotely control compromised computers. It can run commands, executable files, send and receive files et cetera. With the relation to the purpose of putty this shows that the executable file is disguised to serve the purpose but with a hidden sinister intention (Tomas, 2021).

## 2.Analysis of Suspicious PDFs

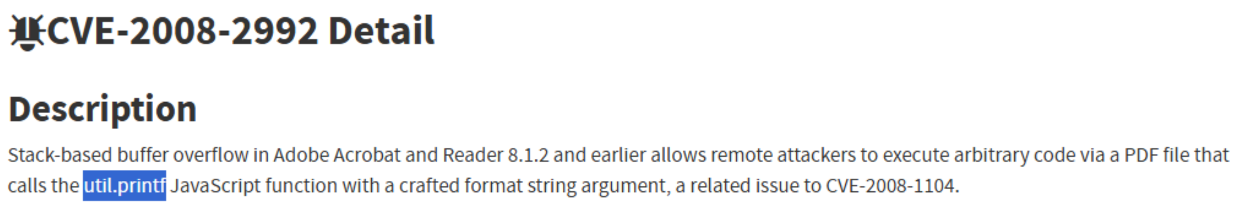
We will be running PDFStreamDumper to examine our questionable files in our system. This is a tool that we will use to analyze suspicious PDF files as we detail every bit. The user interface is shown below.



C:\Users\student\Desktop\LairNet-PDF-1.pdf

*Figure 13/pdf streamer.*

We load pdf files by clicking on the load button and navigated to our suspicious files where after loading we got the screen below. We will be using the exploits scan button which checks for any exploits within the pdf files. Our first document returned a Common vulnerabilities and exposures Exploit CVE-2008-2992 Date:11.4.08 v8.1.2 as shown in the figure below. According to the National vulnerability database the mentioned cve is an exploit that allows remote attackers to remotely execute arbitrary code via a pdf that calls the util. printf JavaScript function.

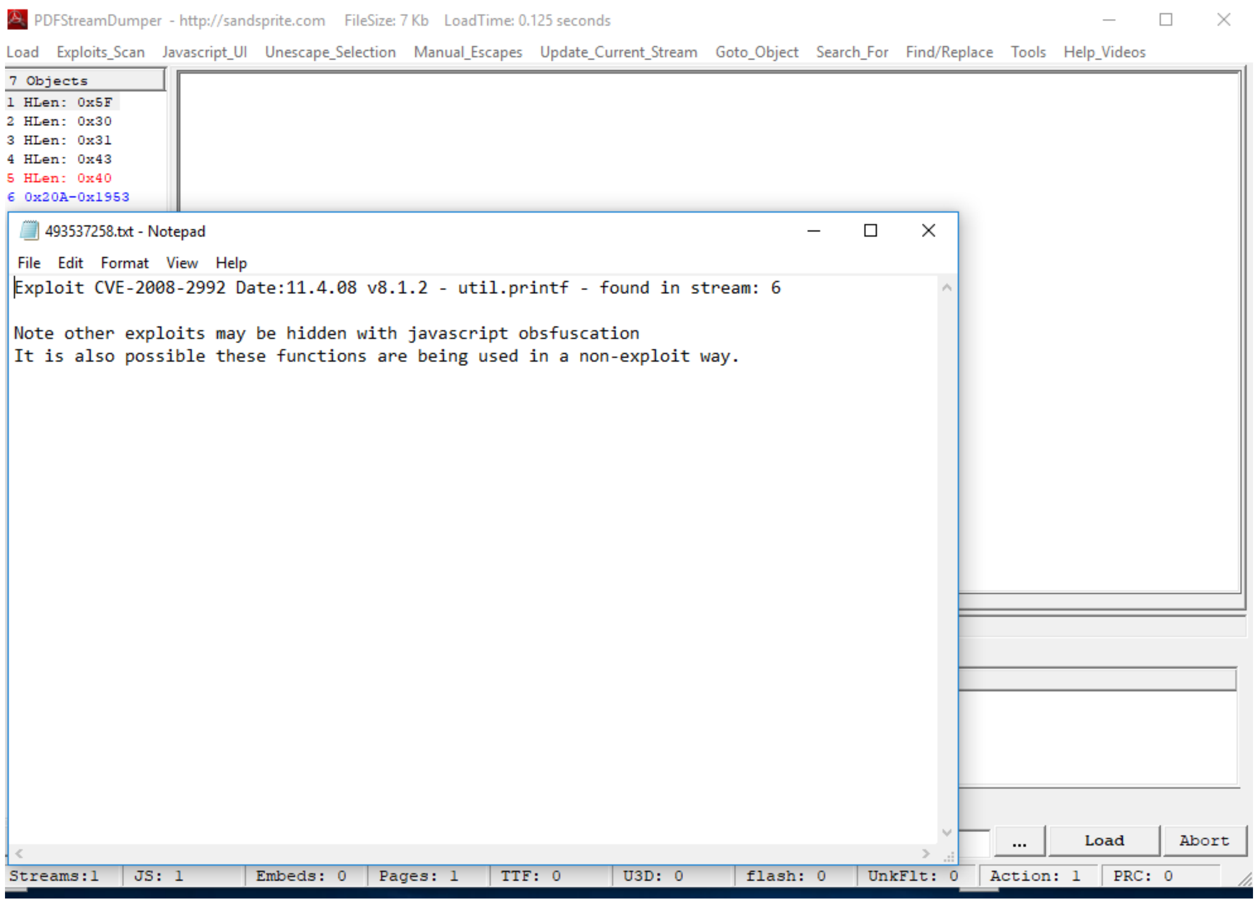


CVE-2008-2992 Detail

Description

Stack-based buffer overflow in Adobe Acrobat and Reader 8.1.2 and earlier allows remote attackers to execute arbitrary code via a PDF file that calls the util.printf JavaScript function with a crafted format string argument, a related issue to CVE-2008-1104.

*Figure 14/cve.*


PDFStreamDumper - http://sandsprite.com FileSize: 7 Kb LoadTime: 0.125 seconds
Load Exploits_Scan Javascript_UI Unescape_Selection Manual_Escapes Update_Current_Stream Goto_Object Search_For Find/Replace Tools Help_Videos
493537258.txt - Notepad
File Edit Format View Help
... Load Abort

```
7 Objects
1 HLen: 0x5F
2 HLen: 0x30
3 HLen: 0x31
4 HLen: 0x43
5 HLen: 0x40
6 0x20A-0x1953
```

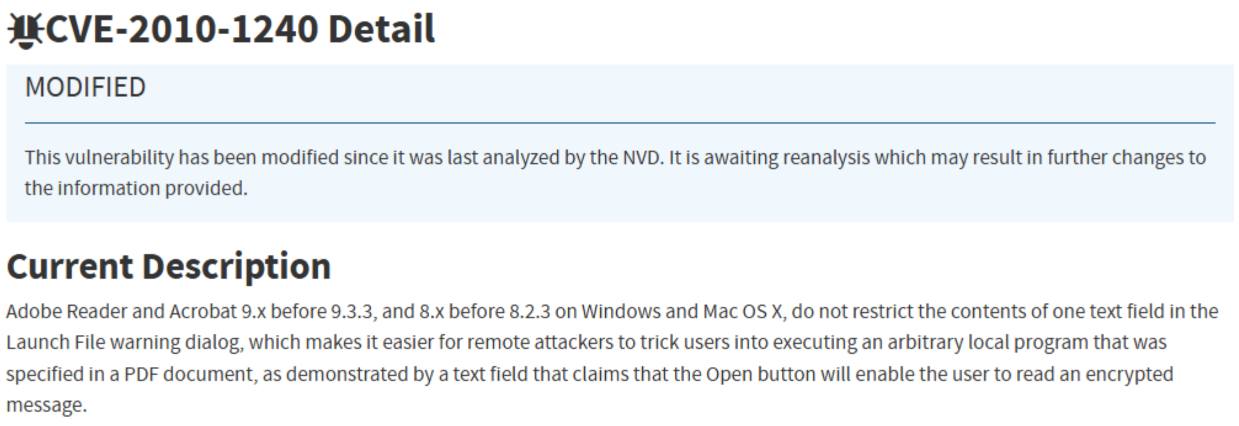
```
Exploit CVE-2008-2992 Date:11.4.08 v8.1.2 - util.printf - found in stream: 6

Note other exploits may be hidden with javascript obsfuscation
It is also possible these functions are being used in a non-exploit way.
```

```
Streams:1 | JS: 1 | Embeds: 0 | Pages: 1 | TTF: 0 | U3D: 0 | flash: 0 | UnkFlt: 0 | Action: 1 | PRC: 0
```

*Figure 15/document1.*

The above show LairNet-PDF-1 contains a malicious cve in stream 6 as indicated above.

The second document LairNet-PDF-2 returned the following details showing that it contained a Launch Action - possible CVE-2010-1240 Date:6.29.10 v9.3.2.



CVE-2010-1240 Detail

MODIFIED

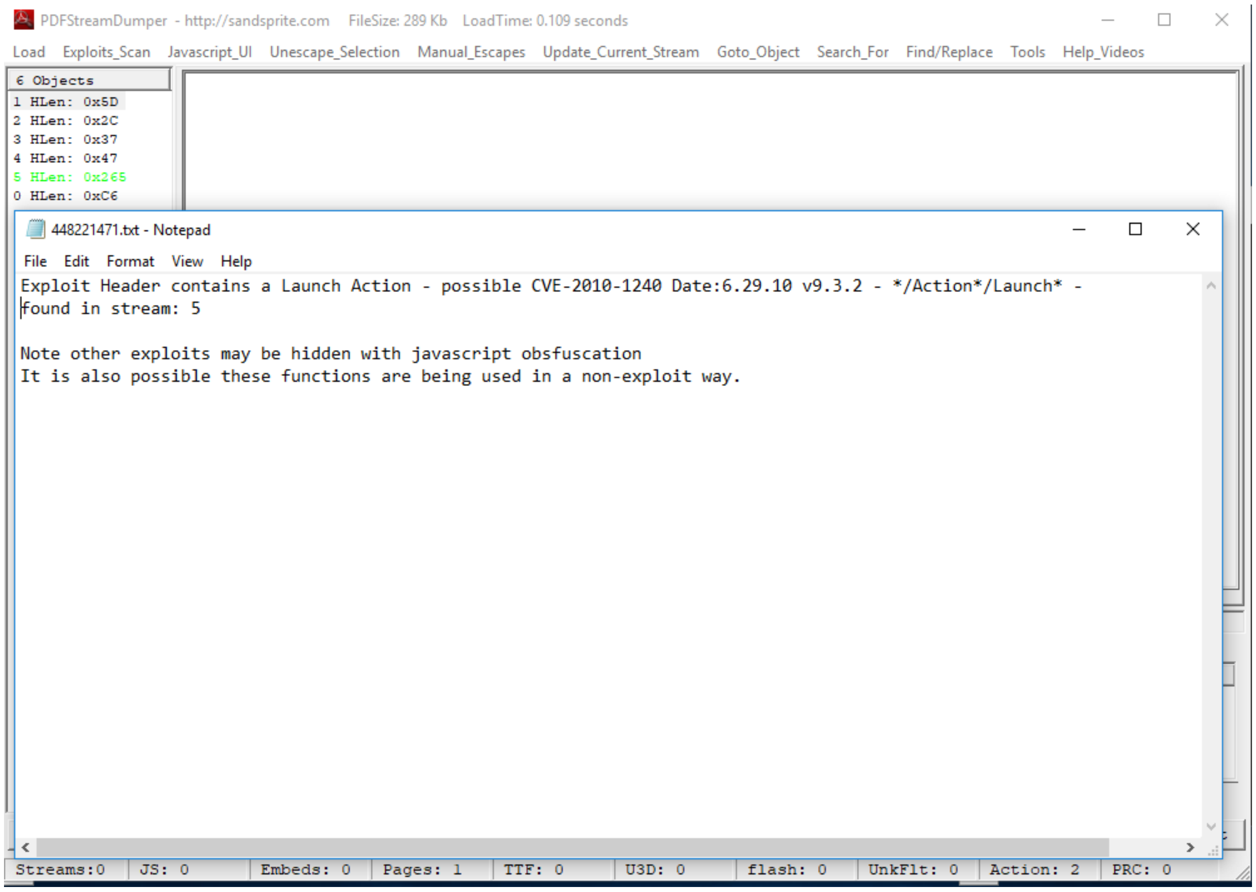
This vulnerability has been modified since it was last analyzed by the NVD. It is awaiting reanalysis which may result in further changes to the information provided.

Current Description

Adobe Reader and Acrobat 9.x before 9.3.3, and 8.x before 8.2.3 on Windows and Mac OS X, do not restrict the contents of one text field in the Launch File warning dialog, which makes it easier for remote attackers to trick users into executing an arbitrary local program that was specified in a PDF document, as demonstrated by a text field that claims that the Open button will enable the user to read an encrypted message.

*Figure 16/CVE2.*

The NVD (National vulnerability database) returned the above screenshot showing the exploit would trick a user into executing a program that is disguised as a different action.


PDFStreamDumper - http://sandsprite.com FileSize: 289 Kb LoadTime: 0.109 seconds
Load Exploits_Scan Javascript_UI Unescape_Selection Manual_Escapes Update_Current_Stream Goto_Object Search_For Find/Replace Tools Help_Videos
6 Objects
1 HLen: 0x5D
2 HLen: 0x2C
3 HLen: 0x37
4 HLen: 0x47
5 HLen: 0x265
0 HLen: 0xC6
448221471.txt - Notepad
File Edit Format View Help

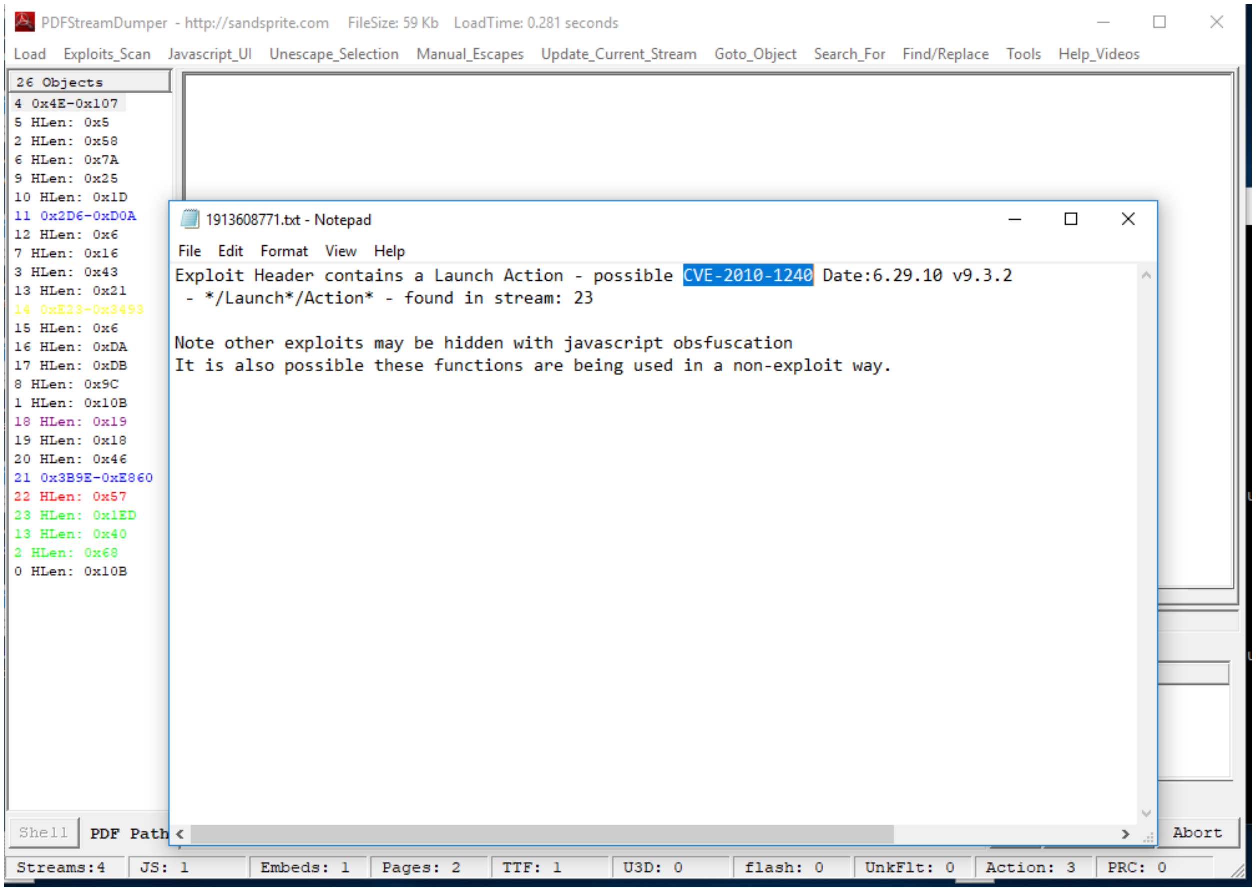
```
Exploit Header contains a Launch Action - possible CVE-2010-1240 Date:6.29.10 v9.3.2 - */Action*/Launch* -
found in stream: 5

Note other exploits may be hidden with javascript obsfuscation
It is also possible these functions are being used in a non-exploit way.
```

```
Streams:0 | JS: 0 | Embeds: 0 | Pages: 1 | TTF: 0 | U3D: 0 | flash: 0 | UnkFlt: 0 | Action: 2 | PRC: 0
```

*Figure 17/Document2.*

This indicates that the exploit was found in stream 5.

The LairNet-PDF-3 document returned Exploit Header contains a Launch Action - possible CVE-2010-1240 Date:6.29.10 v9.3.2. Which was the same vulnerability as the previous one and returned stream 23.


PDFStreamDumper - http://sandsprite.com FileSize: 59 Kb LoadTime: 0.281 seconds
Load Exploits_Scan Javascript_UI Unescape_Selection Manual_Escapes Update_Current_Stream Goto_Object Search_For Find/Replace Tools Help_Videos
1913608771.txt - Notepad
File Edit Format View Help
Shell PDF Path
Abort

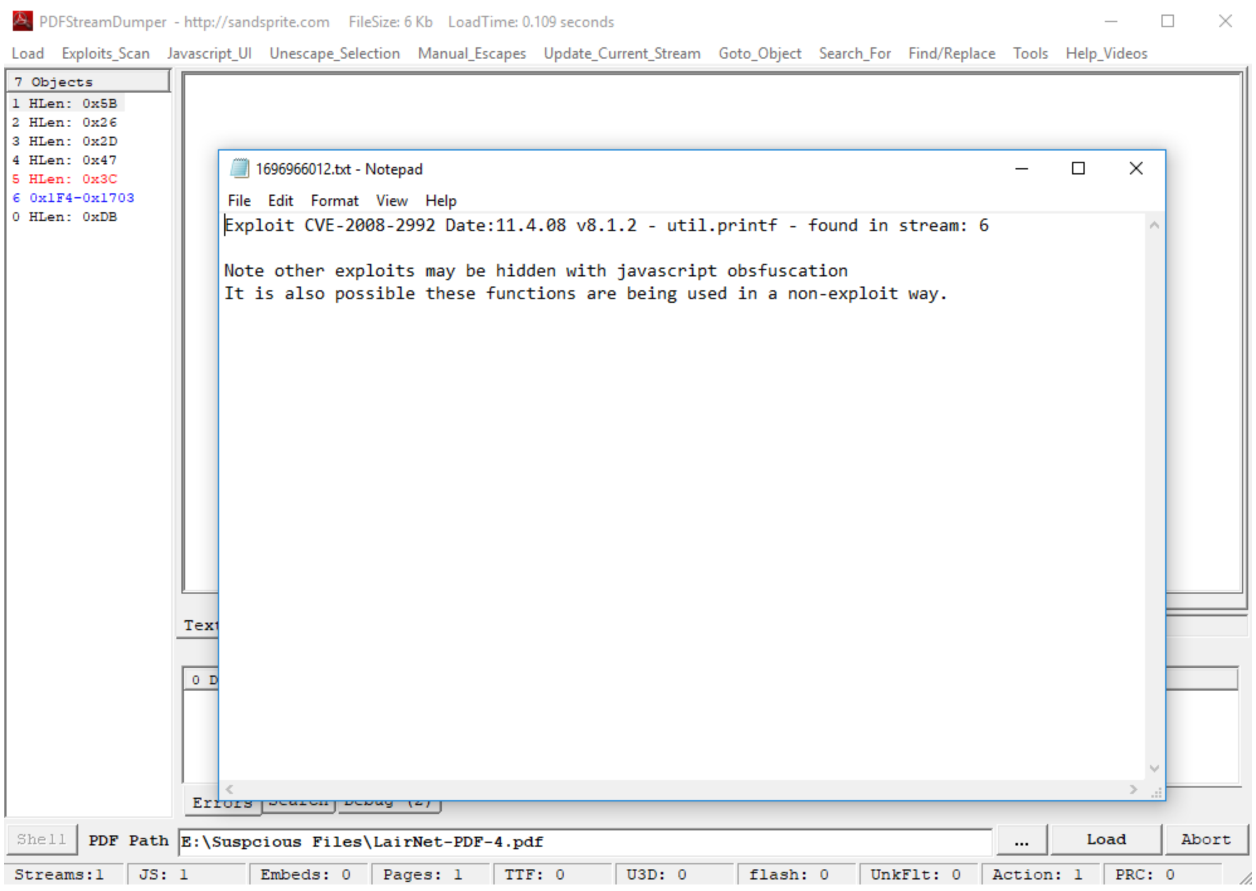
```
26 Objects
4 0x4E-0x107
5 HLen: 0x5
2 HLen: 0x58
6 HLen: 0x7A
9 HLen: 0x25
10 HLen: 0x1D
11 0x2D6-0xD0A
12 HLen: 0x6
7 HLen: 0x16
3 HLen: 0x43
13 HLen: 0x21
14 0xE33-0x3493
15 HLen: 0x6
16 HLen: 0xDA
17 HLen: 0xDB
8 HLen: 0x9C
1 HLen: 0x10B
18 HLen: 0x19
19 HLen: 0x18
20 HLen: 0x46
21 0x3B9E-0xE860
22 HLen: 0x57
23 HLen: 0x1ED
13 HLen: 0x40
2 HLen: 0x68
0 HLen: 0x10B
```

```
Exploit Header contains a Launch Action - possible CVE-2010-1240 Date:6.29.10 v9.3.2
 - */Launch*/Action* - found in stream: 23

Note other exploits may be hidden with javascript obsfuscation
It is also possible these functions are being used in a non-exploit way.
```

```
Streams:4 | JS: 1 | Embeds: 1 | Pages: 2 | TTF: 1 | U3D: 0 | flash: 0 | UnkFlt: 0 | Action: 3 | PRC: 0
```

*Figure 18/Document 3*

The LairNet-PDF-4 document returned an exploit in steam 6 as should below.


PDFStreamDumper - http://sandsprite.com FileSize: 6 Kb LoadTime: 0.109 seconds
Load Exploits_Scan Javascript_UI Unescape_Selection Manual_Escapes Update_Current_Stream Goto_Object Search_For Find/Replace Tools Help_Videos
1696966012.txt - Notepad
File Edit Format View Help
Shell PDF Path
Load Abort

```
7 Objects
1 HLen: 0x5B
2 HLen: 0x26
3 HLen: 0x2D
4 HLen: 0x47
5 HLen: 0x3C
6 0x1F4-0x1703
0 HLen: 0xDB
```

```
Exploit CVE-2008-2992 Date:11.4.08 v8.1.2 - util.printf - found in stream: 6

Note other exploits may be hidden with javascript obsfuscation
It is also possible these functions are being used in a non-exploit way.
```

```
E:\Suspcious Files\LairNet-PDF-4.pdf
Streams:1 JS: 1 Embeds: 0 Pages: 1 TTF: 0 U3D: 0 flash: 0 UnkFlt: 0 Action: 1 PRC: 0
```

*Figure 19/Document4.*

This was the same exploit as the first document which was Exploit CVE-2008-2992 Date:11.4.08 v8.1.2 - util. printf.

There was a consistent return of exploits in the four documents which were recorded on the National Vulnerability Database NVD - Home (nist.gov) with the CVE-2008-2992 and CVE-2010-1240.These identifiers are used to uniquely identify common vulnerabilities and exposures which are populated with the year it was discovered. This is critical for organizations as it enables them to track vulnerabilities and hence patch their systems against these known vulnerabilities.

## References

Agofee. (2022, October 5). *Indicators of Compromise (IOC) Security*. Retrieved from crowdstrike: https://www.crowdstrike.com/en-us/cybersecurity-101/threat-intelligence/indicators-of-compromise-ioc/

Chiradeep, B. (2021, August 19). *what-is-malware-analysis-definition-types-stages-best-practices*. Retrieved from Spiceworks: https://www.spiceworks.com/it-security/data-security/articles/what-is-malware-analysis-definition-types-stages-best-practices/#:~:text=Here%E2%80%99s%20a%20more%20in-depth%20insight%20into%20the%20malware,investigation.%20...%206%20Step%206%3A%20Document%

Debarshi, D. (2023, March 24). *What Is the Windows Portable Executable File Format?* Retrieved from makeuseof: https://www.makeuseof.com/windows-portable-executable-file-format-guide/

freecode. (2020, February 1). *An Introduction to Extensible Markup Language (XML)*. Retrieved from freecode camp: https://www.freecodecamp.org/news/an-introduction-to-extensible-markup-language-xml/

Joshua, C. (2014, March 28). *Threat Intelligence*. Retrieved from Threatdown: https://www.threatdown.com/blog/five-pe-analysis-tools-worth-looking-at/

Manuel, E., Theodoor, S., Engin, K., & Christopher, K. (2012). A survey on automated dynamic malware-analysis techniques and tools. *ACM computing surveys (CSUR)*, 42.

Philip, T. (2022, April 26). *Annotating Malware Disassembly Functions Using Neural Machine Translation*. Retrieved from Google Cloud: https://cloud.google.com/blog/topics/threat-intelligence/annotating-malware-disassembly-functions

Tomas, M. (2021, september 15). *How to remove Meterpreter*. Retrieved from pcrisk: https://www.pcrisk.com/removal-guides/17035-meterpreter-trojan

*VTDOC*. (n.d.). Retrieved from VirusTotal Intelligence Introduction: https://docs.virustotal.com/docs/virustotal-intelligence-introduction