PDF File Scripting Attack

Lewis Koplon, Zi Deng, Emmet Gormican, Sina Malek

Emails: lewisk1899@email.arizona.edu, zkdeng@email.arizona.edu,

emmetg@email.arizona.edu, sinamalek@email.arizona.edu

ECE 509 Cyber Security: Concept, Theory, and Practice Fall ‘21

September 19, 2021

Contents

1. Problem Statement:

The portable document file (PDF) has been around since 1993, when Adobe Systems released it to the public. It is used to present “text, images, multimedia elements, web page links and more” as described in Adobe I/O website and has become standardized by the international organization of standardization (ISO): ISO 32000-1:2008. Since its inception there have been many vulnerabilities discovered and the list is continuing to expand. To try and prevent attackers from exploiting these vulnerabilities, Adobe has tried various ways to improve the security of their systems. Before going into detail about the types of attacks, first we need to understand the basic structure of a PDF. Adobe has a public application programming interface (API), where it shows the content and structure of a PDF in detail.

The article “PDF file format: Basic structure [updated 2020]” by Lukan explains that a PDF is structured as such: Header, Body, ‘xref’ Table, Trailer. The Header holds the PDF version that is currently being used for that PDF. The Body holds the PDF’s content, such as images, text, etc. The ‘xref’ Table or the ‘cross-reference’ table, has a 20 byte reference to the object(s) in the PDF, indicating whether the object(s) are free or in use. The Trailer is used to indicate to the application that is reading the PDF as to where the cross-reference table is.

In the paper “Portable Document Format (PDF) Security Analysis and Malware Threats” by Blonce et al, lists some of these functions and how they can be used to exploit a system. The functions launch, URI, and JavaScript are some examples. The launch function, as the name suggests, is used to “launch an application, open or print a document”. By using this function an attacker can eavesdrop, steal data, the possible types of attacks are numerous. The Universal Resource Indicator (URI) function allows a user to access an external hyperlink. Obviously, the risks of this function is that an attacker can call a malicious hyperlink from the Internet, the possibilities are endless. The JavaScript function allows the user to “execute JavaScript file” which, again, an attacker can use to run a malicious script. These are only some of the functions that PDF uses and an attacker can use multiple of them simultaneously.

There are many threats when using PDFs, in “Portable Document Format (PDF) Security Analysis and Malware Threats” by Blonce et al, list some known attacks performed with PDFs, when using Adobe Acrobat, not when using another application to run the PDF. The W32.Yourde virus was able to exploit a vulnerability in Adobe Acrobat version 5.0.5, essentially “writes code into the user’s Plug-ins folder” and replicates a code, the virus can be used to steal data about the user’s computer, IP address, etc. Another vulnerability is in Adobe Acrobat’s embedded mode, is the Cross Site Scripting (XSS) attack, where a malicious script is injected into the user’s computer and the potential to what the scripts can do is dependent on the attacker.

2. Approaches and Techniques:

There have been many research papers on the topic of malicious portable document file (PDF) attacks. In this project we will be exploring them to work out the method and approach to creating a malicious PDF that will execute a malicious script when it is opened. The PDF format is an incredibly commonly used file system that many previously had perceived to be safer than other file formats due to the fact that it is not an explicit executable like other computer programs. However this is a misconception, PDFs have the ability to execute JavaScript (JS) as part of its design and thus can be embedded with malicious JS that is aimed to gain access to your machine and either harm it or extract sensitive information. It is also possible to embed another file within a PDF which can then be used to result in similar harm.

The paper “Malicious PDF Documents Explained” by Ahmad explains how early PDF malwares were used in the history of PDFs. Attackers would utilize vulnerabilities in the PDF reader such as the JBIG2Decode or util.printf bugs to gain extended instruction pointer (EIP) control to control over the attacked program and execute malicious shell code. The malicious PDF when opened would execute a JS program that would either download this shellcode from HTTP website or directly embed the shellcode into the PDF and access it there. Once obtained the attackers will write the malicious shellcode to the disk then execute it using the instruction control obtained from the bug exploit. A fearsome attribute of this process is that even unskilled attackers could abuse this vulnerability by using a technique called heap-spraying. Heap-spraying involves copying the shellcode stored in the JavaScript hundreds of thousands of times into the heap memory. This makes it so when EIP points to the heap, there’s a very likely chance it will execute the malicious shellcode.

The survey paper “Detection of Malicious PDF files and directions for enhancement: A state of the art survey” by Nissim et al. explores the current state of PDF attacks and detection in the modern era. A large proportion of PDF malware attacks are executed by embedding malicious JS code into the PDF file to exploit vulnerabilities in the PDF viewer. While Adobe Acrobat continues to create new safety features such as the Protected Mode that uses a sandbox technique to create an isolated environment for Acrobat Reader in the newest X edition, many companies and individuals either do not properly update their reader or use a third party reader with less safety features. Another method of attack involves embedding a different file type inside of the PDF file and exploits the vulnerabilities of the another file type in the PDF attack. An example of this is reverse mimicry which creates a file similar to a benign PDF file but then injects it with a malicious exe payload. This method is particularly resilient to many forms of PDF malware detection as it attacks on a different axis than standard detection frameworks expect. Currently the state of the art methods for detecting malicious PDFs revolve around building a classification model to classify new and unfamiliar PDFs. A popular technique is to tokenize the JS content inside the PDF, then run the content through a machine learning classifier with labeled data to allow an algorithm to be developed to be able to learn which PDFs are malicious and which are not. An alternative approach to PDF malware detection is the utilization of meta data (data about data) of the PDF files to determine abnormalities and strike them as malicious. Common forms of this are keyword analysis, hierarchical analysis, content metadata analysis, term frequency and entropy analysis. A state of the art framework would utilize a multitude of these detection techniques to rigorously explore and validate the content of any given PDF.

3. Main Issues

The main issue our project intends to address is how memory corruption vulnerabilities can be exploited, specifically how an attacker can target you via utilizing a pdf as a means of delivering a payload in this case, shellcode. Manajiit Pal et al clearly explain the concern for these attacks, memory corruption vulnerabilities are exploits that have the ability to alter the program behaviour as well as the possibility of taking complete control over the control flow of a computer. Applications that have little memory or no type safety are especially vulnerable to these types of attacks, but our main focus will be explaining how a pdf can be used as a means of delivering a malicious shellcode to your computer using a tactic called heap spraying.

To ease the explanation of this vulnerability the use of a simple system architecture model is greatly beneficial. A computer is composed of hardware components that make up the foundational layer of itself, these components include but are not limited to the CPU, memory both random access and disk, and a network card for interfacing with your network. On top of this foundational layer, we have an operating system such as windows and linux, and system software such as drivers. Lastly, we have the application program that runs on the OS, in the space of the application there will be memory allocated for different regions, the code/text region is designated for machine instructions, the stack region is a region that stores temporary data created by functions, among other data. The area that will be receiving most of the attention however is the heap. The heap is a dynamically allocated region where data files get loaded into, as well as many forms of instantial data, it is also a crucial attack surface.

Heap spraying is a process of copying the malicious shellcode over and over again into the heap, so that the code can be advanced and ran. Furthermore, the shellcode can have a nop sled, which are numerous numbers of "no operation" instructions that increase the area the malicious code covers in the heap, prepended to it in order to increase the probability of the malicious code being executed. If the EIP lands on this NOP sled, then it will slide down to the shellcode and then the shellcode can be advanced using other exploits.

4. Technology, Tools, and Hardware

The main tools we will be using are possibly a few different Integrated Development Environments (IDEs), such as PyCharm, Visual Studios or Eclipse depending on which would be better for applying our project. The languages we are planning on using are JavaScript, to write the malicious code and possibly Python to execute those scripts. JavaScript and Python can be written in Visual Studios, or their respective IDEs. To perform these attacks we are planning on utilizing Virtual Machines (VMs), to ensure that we are not attacking actual systems and practice these attacks in a safe sandbox environment. We will not be utilizing any hardware for this project. Through our research we have decided to attempt a spear-phishing attack as our project, since it is targeting a particular individual (in this case, members of the group), to try and steal their data.

5. Experiments

Malicious PDF files can attack many different or specific targets and inflict varying levels of damaging results depending on the components. Some attacks can target the machine that opens the PDF directly and others exploit the application that opens the PDF. These applications can be Adobe Reader, Acrobat, and browsers that can open PDFs and the code contained in the PDF will exploit flaws on the application level. For our research we will be concerned with attacks that target the payload of the system. The code that is commonly used is JavaScript with Python due to its customizability and the physical structure of PDF files. While there are many types of attacks the most prevalent method to explore further today is heap spraying using NOP (no operation) sleds.

A PDF document has four main parts: header, objects body, cross reference table, and trailer. A malicious PDF document will have attacking code embedded within that structure. For a heap spray attack the source code of the exploit sprays the disordered heap of sequenced bytes to a specified location in the memory of the target. The malicious code is loaded into duplicated locations within the heap. The more locations the malicious code appears in heaps, will lead to better chances of it being executed. The NOP sled component adds the sequenced instructions directly after the malicious code block and duplicates the code for more locations to require memory. The NOP sled will search the machine's instructions of no operations to then execute the exploit.

6. Conclusion

PDF documents have been a crucial tool for everyday operations in organizations since the inception of the internet, and they will continue to be utilized for their functionality for many years to come. The ease of use for PDFs is undisputed, "in 2019 there were over 20 billion PDfs are shared on google… 100 million PDFs are saved every day, no technology is threatening to replace PDFs as of yet" (Google 2019). Every time a PDF is opened creates a potential new node for an attack surface. Malicious PDF exploits are also on the rise and threaten people's privacy and threaten other potential fallout. Attackers are continuously developing more and more advanced malicious scripts to embed in the PDF structure as tools become easier to use. Javascript and python offer attackers the tools and open source platform enable the refinement and spread of malicious information. The prevalent type of exploit for PDF attacks are using Heap Spray and NOP Sleds to execute the malicious code. For this reason we will need to further survey this type of attack by exploring the structure of the malicious code as well as the structure of PDF documents that an application would normally open. The PDF document will act as the delivery system for the malicious code. This technique of using a heap spray will be a crucial part to infect a system with that malicious code. It will do this by spraying heaps of information that all require memory. The malicious code loaded in the heap will allow the attacker to have their code run within that memory space.

This is a strong and present concern, thus further exploration and creation of existing methodologies will answer questions of potential future exploits. Being familiar with the attackers tools as well as targets will help create a framework for mitigation and protection of computer systems.

Works Cited:

Chu, Mike. “Can Pdfs Contain Viruses? 4 Critical Things to Watch out For.” *Data Overhaulers*, 31 Dec. 2020, dataoverhaulers.com/can-pdfs-contain-viruses/.

Developers, Adobe I/O — Adobe. “SDK Developer Kit: PDF Library: Adobe Document Services - Adobe Developers.” *SDK Developer Kit | PDF Library | Adobe Document Services - Adobe Developers*, 15 Sept. 2021, www.adobe.io/apis/documentcloud/dcsdk/.

Johnson, Duff. “PDF Statistics – the Universe of Electronic Documents.”

Pal, Manajit & Dey, Prashant & Dokania, Vaibhav. (2016). Memory Corruption-Basic Attacks and Counter Measures. International Journal of Engineering Science and Computing. 6. 3511.

“PDF File Format: Basic Structure [Updated 2020].” *Infosec Resources*, 27 May 2021, resources.infosecinstitute.com/topic/pdf-file-format-basic-structure/.

“Sustainability of Digital Formats: Planning for Library of Congress Collections.” *PDF_1_3, PDF Versions 1.0-1.3*, 6 Jan. 2019, www.loc.gov/preservation/digital/formats/fdd/fdd000316.shtml.

“W32.Yourde.” *Liutilities.com*, www.liutilities.com/malware/computer-virus/w32-yourde/.