# Advanced Persistent Threat:

# Malicious Code Hidden in PDF Documents

Ken Chang  Dr. Ying-Dar Lin
National Chiao Tung University
Email: gakaza3333@gmail.com; ydlin@cs.nctu.edu.tw
September 18, 2014

## Abstract

Advanced Persistent Threat (APT) in recent years has become a very popular choice to steal information of specific targets using the vulnerabilities on the targets' machine. APT involves a set of complex phases, which are difficult to detect and often initiated with spear phishing in the early stage of invasion. To help defend against APT, it is important to study the malformed Portable Document Format (PDF) documents used in spear phishing e-mails, and also study the common JavaScript obfuscation and non-JavaScript based obfuscation used to hide shellcode in the malformed PDF documents.

With XecScan, PDF Examiner, and two analysis tools PDF Stream Dumper and Cerbero's Profiler, we scanned 150 samples from three different Common Vulnerabilities and Exposures (CVE); CVE-2011-2462 and CVE-2013-0640 samples are all detected to be malicious, and five out of fourty-nine CVE-2010-0188 samples could not be detected by XecScan and PDF Examiner. Investigating these fourty-nine samples with PDF Stream Dumper and Cerbero's Profiler, one could extract the shellcode hidden in the embedded TIFF image from most of these samples. In the CVE-2010-0188 case study, URL scanning tools urlQuery and VirusTotal can track the website pointed by the URL in malformed documents, and the tools can determine if there is a malware or malicious script implemented in the website.

Keywords: APT, Malformed Document, PDF, Common Vulnerabilities and Exposures (CVE), Embedded TIFF Image

## 1. Introduction

### What is Advanced Persistent Threat?

Advanced Persistent Threat (APT) is a set of complex, stealthy and continuous processes that aim to monitor, administrate, and steal specific targets in a long term while being undetected. Three major components or parts of APT are: advanced, persistent, and threat. The advanced component suggests that the attacker uses cyber-intrusion methods, administrative techniques, and custom exploits or tools when invading the target's network and system. The persistent component is the part where the attacker plans and decide a long-term objective and work to stay undetected while monitoring the target and

extracting the desired information. Last but not least, the threat component indicates that the attacker is organized, funded and motivated.

Other than the three major components of APT, the following are the outline or the phases of APT attacking processes: targeting, access or compromise, reconnaissance, lateral movement, data collection and exfiltration, and administration [1]. Targeting is the phase where the attacker collects information about the target using social engineering technique while testing different methods to grant access. In the access and compromise phase, the attacker then will gain access and study the most effective methods to exploit the system. After compromising the system, reconnaissance phase will take place as the attacker explores the network structure such as domain services, domain controllers, and administrative rights in order to access other systems and application. Once the attacker discovered the method of traversing to other systems that are more suitable or potential, the attacker will carry on the lateral movement to other targets. Data collection and exfiltration then will be performed to steal and extract valuable information. Last but not least, the attacker will continue to administrate and maintain access over time, which is one of the main goal of APT.

**APT vs. Traditional Attack**

Without the target taking the bait, neither APT nor traditional attack are able to setup their plot to steal information. The initiation done by APT is very different from initiation of traditional malwares. Traditional attack targets anyone by spreading itself throughout the network to infiltrate as many as possible. As a result, analyst often receive many samples to study and build a solid defense against such attacks.

On the other hand, APT attackers favourite in spear phishing to initiate, which is to pinpoint an attack against some subset of people by baiting them to execute malicious attachment or URL in e-mail or website. One of the advantages of using spear phishing is to greatly reduce the sample number if the malicious content is ever discovered, making the analysts much more difficult to justify their solution to this attack due to small sample size. When sending an e-mail that includes malicious attachment or URL, attacker often disguise as a person or organization that is trusted by the target. For example, the attacker will send an e-mail to a company including a resume stored in Portable Document Format (PDF) as an attachment; the receiver must open the resume using a PDF reader tool if he or she wants to hire the e-mail sender. Upon opening the resume, the receiver is infected. Nowdays these malformed document comes in many different file formats using different techniques to hide and execute the malware, and they are difficult to detect by traditional defense system.

Since spear phishing is very effective on baiting the target to open malicious content, it is important to prevent such attack by studying the malformed document, especially the most commonly used file format, PDF. These malformed PDF documents can achieve the task listed in the Table 1.

Table 1. APT vs Traditional Attack [2]

| | APT | Traditional Attack |
|---|---|---|
| Specific Target | Yes | No |
| Personal Info | Yes | No |
| Account Password | Yes | Yes |
| Malware Document | Yes | No |
| Zero-day Malware | Yes | No |

In the next section, the common PDF exploits and vulnerabilities will be discussed, giving the readers some basic concepts of these exploits. By understanding these common exploits and vulnerabilities, one can identify the malicious code implanted in malformed PDF documents. In the method section, the detection method for analyzing a malicious PDF will be introduced. Furthermore, the tools for analyzing and detecting malicious documents will be presented in the experiment section, where malformed PDF documents will be the main focus in the experiment. To be more specific, CVE-2010-0188 samples will be tested due to the fact that it once was one of the most common used exploits and also it is difficult to detect. We will examine some existing analysis tools and common exploits related to the malicious content to help protect, understand, prevent, and build awareness against common exploits or zero-day attack.

# 2. Background

### PDF Format

Portable Document Format (PDF) is a file format that represents a document. The advantages of PDF includes: compact, securable, quick and easy to create. As a result, PDF is now one of the most popular document format to read with. PDF is designed and stored in the basic structure shown in Figure 1.

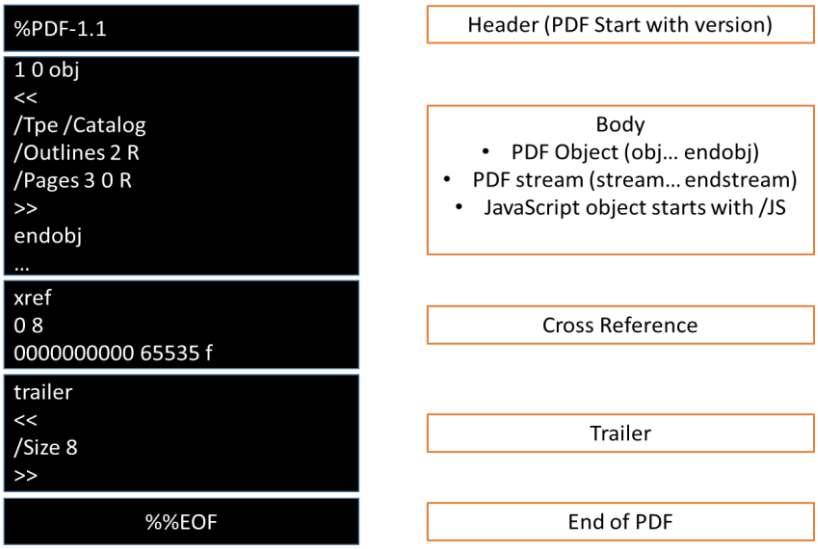



Figure 1. PDF structure [3]

In the basic PDF structure, the header section marks the start of the document. The body section is commonly filled with JavaScript objects or PDF streams that records the content of the PDF document shown to the reader. Yet the body section is also the section where most attacker implant their malicious code or malicious content. The cross reference section will referenced all objects in the PDF, and it also supports the objects being stored in random order. The trailer section acts as a dictionary, recording the necessary information to read the document. Finally, the EOF will mark the end of a PDF document.

Since PDF allows macro and executable code, a successful APT initiation is to implant a malware to install a certain type of Remote Access Tools (RAT) so the attacker could gain access to the target's system and proceed to the next phase of APT. The malicious code often practices the technique of heap spray, which will overtake a program by take advantage of that the program's return address. The malformed document that contains this malware or malicious code would try not trigger the defense system on the target's machine once the target open this document and automatically executes the malware or malicious code in the background. In addition, malformed document often contains JavaScript obfuscation and non-JavaScript based obfuscation, which are the two most common techniques practiced in malformed PDF document to bypass the defense system. Many of the exploits use these two techniques to hide the malicious code hidden within the PDF document. As shown in Table 2, if the creator desire a zero-day attack, the malformed document will pretend to be harmless by avoiding JavaScript and unusually encoding so the malformed document cannot be detected by signature based defense. On the other hand, malformed documents that are created based on existing exploits often try to use confusing or obfuscating encoding techniques to hide the malicious code so the defense mechanism cannot recognize and defend against it.

Table 2. Goals of PDF malware obfuscation [4]

| Goals | Pretend to be harmless | Be confusing |
|---|---|---|
| Artifice | Be well-formed<br>Avoid JavaScript based exploits<br>Avoid unusual encodings | Use obfuscation<br>Break the tools |

**PDF Exploits: JavaScript Obfuscation**

There are many ways to obfuscate so the PDF software or defense system could not recognize or locate the hidden malicious code. In this section, some of the most common JavaScript obfuscation techniques will be introduced, and they are: Name obfuscation, Eval or evaluate function, JavaScript Splitting, and Callee-trick [4] [5].

Name obfuscation perhaps is the simplest PDF trick to hide the actual content. The goal is to hide original meaning by renaming all variables and functions. Very simple it may sound, yet renaming hundreds of variables and functions is considering annoying at the time when most malformed PDF documents have to rely on human analyst to study and decode. Of course, today there are tools like PDF Stream Dumper that can quickly solve this problem, so name obfuscation is no longer an issue.

However, there are many malformed PDF documents still use naming obfuscation to greatly reduce the readability of the code to confuse and slow down the analyst.

Eval function or evaluate function is one of the function that allows the PDF creator to execute or evaluate a string without the need to reference any object. If the string is an expression or statement, the evaluate function will evaluate the string with respect to that expression or statement. However, most attacker often used the evaluate function to hide and execute encrypted shellcode. One of the known exploit executes the following string to exploit vulnerability of the program:

```
eval("this.media.newPlayer(null)")
```

JavaScript Splitting, or concatenation, is another feature that can obfuscate the code. JavaScript splitting allows user to split strings into parts over several PDF objects and execute them consecutively, even in random orders. Tracking between objects in random order greatly increases the difficulty of revealing the malicious code's original content. Together, evaluate function and JavaScript splitting, or concatenation, make the code even more obfuscated. Using the example code for evaluate function, the following example will show how to use JavaScript splitting and evaluate function together:

```
eval("thi" + "s.me"+ "dia.n" + "ewPl" + "a" + "yer" + "(null)")
```

Another very common encoding or encryption technique is called the callee-trick. The idea of callee-trick is to have functions to access its own resource and use this resource as a key to decrypt code or data. This technique is effective to obfuscate when it is used with multiple layers of encoding or encryption, which will be mention in the next section.

**PDF Exploits: Non-JavaScript Based Obfuscation**

In this section, techniques that are not based on JavaScript will be introduced. Most of them are still very commonly used today even for the latest malformed PDF documents. These are also the techniques that explains why having support in macros or execute files is not a good idea since the attacker will exploit vulnerabilities in each of macros or execute file.

Flash stream and flash object are commonly used in PDF. However, attacker had found a way to execute ActionScript using flash stream to heap spraying. Fortunately, the PDF tools with the latest patch installed will show warning to the user that flash object is present; the user can also disable the flash content all the time. As shown in Figure 2, an embedded flash stream will begin with "FWS".

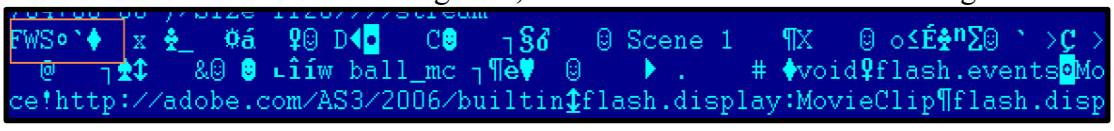



Figure 2. Embedded flash stream example [6]

Embedded Tagged Image File Format (TIFF) image is another vulnerability used by the attacker to trigger a download of a malware from a webpage. The attacker will compress the malformed TIFF image with shellcode in XML Forms Architecture (XFA) stream and AcroForm. Once the PDF malformed has been opened, the vulnerability of the TIFF image will be exploited, and the shellcode

will perform a heap spray, downloading the malware from a webpage. An example of embedded TIFF image hidden in XML data is shown in Figure 3.

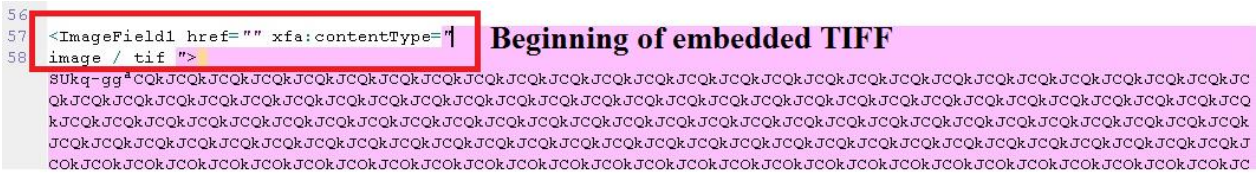



Figure 3. Embedded TIFF image example

Multiple layers of encoding or encryption are common techniques used to hide the shellcode. The idea is to encrypt or encode in many layers to hide the original content, making it difficult to analyze. In this case, these techniques are used to hide all or parts of the malicious code, avoid being detected by the defense system. As demonstrate in Figure 4, one may encode a piece of the code by filling in extra alphabet letter or number and then perform JavaScript escaping to become a piece of meaningless code or data block for the machine or analyst to read. To further complicate the situation, attacker could combine this technique with callee-trick to protect the malicious content from being discovered. Once the PDF reader opens the document and use the key or script to decode this "meaningless" data block, this meaningless data block become JavaScript or shellcode that exploit all kinds of vulnerabilities.

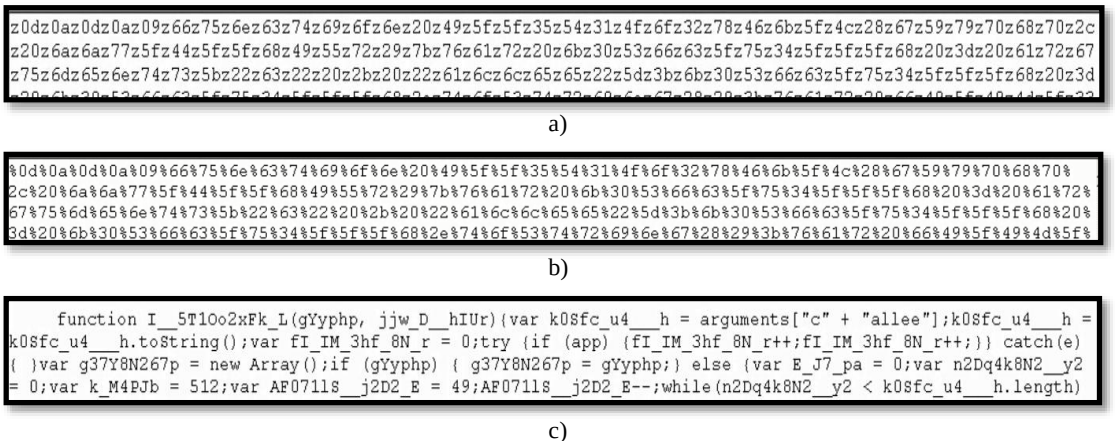



Figure 4. a) Encoded data block. b) The code after filtering the character 'z'. c) The shell script after JavaScript escaping. [7]

# 3. Method

To be acquainted with variety of exploits used in malformed PDF document, analyzing with several tools to identify these exploits or vulnerabilities is strongly advised to be efficient and stay on task. If one only wishes to determine whether a PDF document is malicious or not, uploading the PDF document and scanning with online scanning tools like XecScan [8] and PDF Examiner [9] can help determine most of the malformed PDF documents. While XecScan is more accurate in determining exploits, PDF examiner extracts the malicious content and categorized these contents for the user to study. To further study the malformed PDF document, one can manually analyze the documents with

analysis tools like Cerbero's Profiler [10] or PDF Stream Dumper [11]. Cerbero's Profiler provides quick automatic analysis which helps the user to identify the malicious content. On the other hand, PDF Stream Dumper is a combination of tools including sandbox tools, JavaScript editor, and signature detection system with graphical interfaces to help the user run commands to reveal the hidden content within the malicious document. Using Profiler will be much easier since the tool provides almost instantaneous detection of the sample; the samples will immediately be marked malicious or suspicious once it is loaded and ready for examination. In contrast, PDF Stream Dumper usually could not detect the malicious content immediately. The user often needs to de-obfuscate using the tools provided or manually de-obfuscate so the PDF Stream Dumper could identify any threat. Though PDF Stream Dumper is strongly advised when analyzing the samples that heavily based on JavaScript due to its better designed JavaScript editor, which allows the user to run sandbox tools on all or certain section of the script. However, the human analyst must understand and know JavaScript obfuscation or non-JavaScript based obfuscation before begin investigating and exploring with the analysis tools.

# 4. Experimental Study

The goal of the experiment is to study the CVE-2010-0188 samples and its structure using both online scanning tools and analysis tools. Also, it is important to inspect the features and flaws in all tools so an analyst can track or analyze the samples and lower the false negative rate when scanning a sample. Each samples will be analyzed until the hidden URL has been revealed along with its destination IP. In most cases, the shellcode hidden in embedded TIFF image will include an URL that is used to download malware from a webpage or host server.

**Experiment Setup**

The environment to test the malformed PDF samples is VMware's virtual machine running Windows 7, 64-bit. The samples used in the experiment are fourty-nine CVE-2010-0188 samples. The CVE-2010-0188 is a high severed vulnerability in Adobe Reader and Acrobat 8.x before 8.2.1 and 9.x before 9.3.1 allowing attacker to crash the application and execute arbitrary code once the malformed PDF document is opened [12].

The reason of studying this particular case is because it was once the second most commonly used vulnerabilities according to malware tracker's "pdf current threats" report in July 2014 [13]. Hence, the result from XecScan and PDF examiner in Figure 5 also shows that CVE-2010-0188 samples have a much higher false negative rate compared with the test result of twenty-five CVE-2011-2462 samples and twenty-one CVE-2013-0640 samples, which all are detected to be malicious or suspicious by XecScan and PDF Examiner. Another reason why CVE-2010-0188 is a case worth studying is because of the PDF examiner's unusual behavior, which is not providing any feedback or extraction of the malicious code. Also, it is worth noting that XecScan has a much lower false negative rate compared

with PDF Examiner, and there are only five out of fourty-nine samples marked safe by both XecScan and PDF Examiner

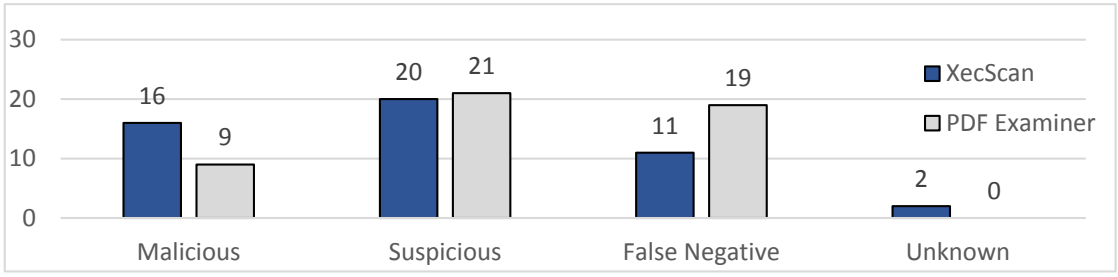



Figure 5. Result of 49 CVE-2010-0188 samples

**Online Scanner: XecScan and PDF Examiner**

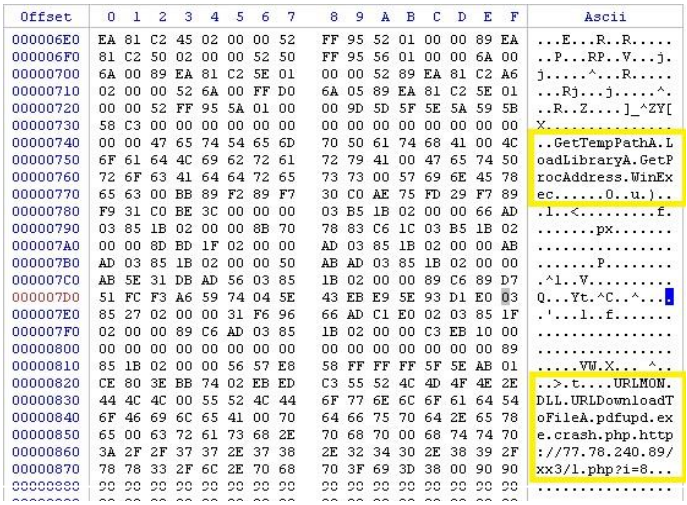
After recording the result from both of these online scanner, the samples that received completely the opposite result or the same result between XecScan and PDF examiner will be further analyzed using PDF Stream Dumper and Cerbero's Profiler. Comparing between the same results or completely the opposite results will help the analyst understand the differences or similarities between the two scanners. The samples that are marked suspicious by both XecScan and PDF Examiner will not be reviewed because PDF Examiner is not sensitive to miss used JavaScript, it also failed extract the malicious code to the user whether the sample is marked suspicious or malicious. On the other hand, XecScan only provides useful feedback to those who understands the setting of XecScan's yara rules and snort rules, so the samples that are marked suspicious by XecScan wouldn't help the analyst understand the code hidden in the document any better.

One might discover a certain pattern or encoding technique used in these samples that can help improve the accuracy of both XecScan and PDF Examiner. Although XecScan has a higher detection rate as a whole, there are a few samples that XecScan could not pick up but PDF Examiner can. As a result, only five samples that are marked safe by both XecScan and PDF Examiner.

**Analysis Tools: PDF Stream Dumper and Cerbero's Profiler**

Studying only five samples that are marked safe by XecScan and PDF Examiner is definitely not enough. One must obtain the result from XecScan and PDF Examiner then investigate further with PDF Stream Dumper and Cerbero's Profiler. Out of the samples that have opposite or same result from the two scanners, an analyst should be able to extract the URL in the embedded TIFF image; this is generally the key to identify whether this file is malicious or not. Profiler usually detects the malicious code right away and is able to identify the threat, though there are cases when Profiler fails; like PDF Examiner, Profiler will mark the document suspicious if the content is not a threat. Yet, to extract the URL manually, one must look into the XML data of the sample and copy the base64 code for conversion, which both Profiler and PDF Stream Dumper supports. In general, the shellcode converted from base64 will reveal the hidden URL in the middle section; many of the other section are filled with the different characters, including '.' or '?' to buffer overflow the heap memory. As shown in Figure 6, one can locate the embedded TIFF image using PDF Stream Dumper and covert to this two pieces of shellcode that include a URL and some instruction sets accessing the library and the process's

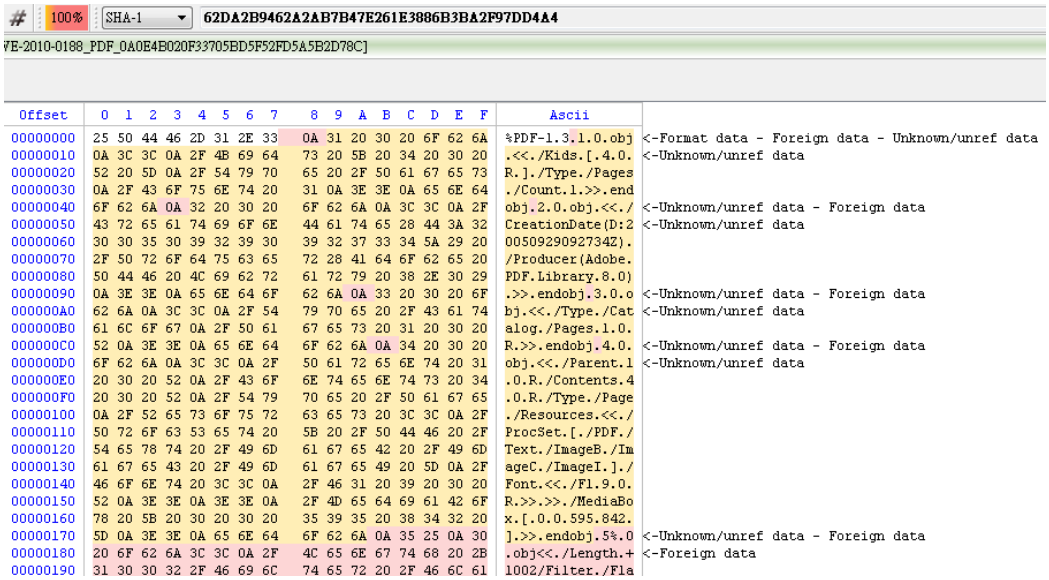
address; there are a very few cases that requires the analyst to JavaScript escaping this piece of code after converting from base64.



| Offset | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F | Ascii |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 000006E0 | EA | 81 | C2 | 45 | 02 | 00 | 00 | 52 | FF | 95 | 52 | 01 | 00 | 00 | 89 | EA | ...E...R..R..... |
| 000006F0 | 81 | C2 | 50 | 02 | 00 | 00 | 52 | 50 | FF | 95 | 56 | 01 | 00 | 00 | 6A | 00 | ..P...RP..V...j. |
| 00000700 | 6A | 00 | 89 | EA | 81 | C2 | 5E | 01 | 00 | 00 | 52 | 89 | EA | 81 | C2 | A6 | j.....^...R..... |
| 00000710 | 02 | 00 | 00 | 52 | 6A | 00 | FF | D0 | 6A | 05 | 89 | EA | 81 | C2 | 5E | 01 | ...Rj...j.....^. |
| 00000720 | 00 | 00 | 52 | FF | 95 | 5A | 01 | 00 | 00 | 9D | 5D | 5F | 5E | 5A | 59 | 5B | ..R..Z....]_^ZY[ |
| 00000730 | 58 | C3 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | X............... |
| 00000740 | 00 | 00 | 47 | 65 | 74 | 54 | 65 | 6D | 70 | 50 | 61 | 74 | 68 | 41 | 00 | 4C | ..GetTempPathA.L |
| 00000750 | 6F | 61 | 64 | 4C | 69 | 62 | 72 | 61 | 72 | 79 | 41 | 00 | 47 | 65 | 74 | 50 | oadLibraryA.GetP |
| 00000760 | 72 | 6F | 63 | 41 | 64 | 64 | 72 | 65 | 73 | 73 | 00 | 57 | 69 | 6E | 45 | 78 | rocAddress.WinEx |
| 00000770 | 65 | 63 | 00 | BB | 89 | F2 | 89 | F7 | 30 | C0 | AE | 75 | FD | 29 | F7 | 89 | ec......0..u.).. |
| 00000780 | F9 | 31 | C0 | BE | 3C | 00 | 00 | 00 | 03 | B5 | 1B | 02 | 00 | 00 | 66 | AD | .1..<.........f. |
| 00000790 | 03 | 85 | 1B | 02 | 00 | 00 | 8B | 70 | 78 | 83 | C6 | 1C | 03 | B5 | 1B | 02 | .......px....... |
| 000007A0 | 00 | 00 | 8D | BD | 1F | 02 | 00 | 00 | AD | 03 | 85 | 1B | 02 | 00 | 00 | AB | ................ |
| 000007B0 | AD | 03 | 85 | 1B | 02 | 00 | 00 | 50 | AB | AD | 03 | 85 | 1B | 02 | 00 | 00 | .......P........ |
| 000007C0 | AB | 5E | 31 | DB | AD | 56 | 03 | 85 | 1B | 02 | 00 | 00 | 89 | C6 | 89 | D7 | .^1..V.......... |
| 000007D0 | 51 | FC | F3 | A6 | 59 | 74 | 04 | 5E | 43 | EB | E9 | 5E | 93 | D1 | E0 | 03 | Q...Yt.^C..^.... |
| 000007E0 | 85 | 27 | 02 | 00 | 00 | 31 | F6 | 96 | 66 | AD | C1 | E0 | 02 | 03 | 85 | 1F | .'...1..f....... |
| 000007F0 | 02 | 00 | 00 | 89 | C6 | AD | 03 | 85 | 1B | 02 | 00 | 00 | C3 | EB | 10 | 00 | ................ |
| 00000800 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 89 | ................ |
| 00000810 | 85 | 1B | 02 | 00 | 00 | 56 | 57 | E8 | 58 | FF | FF | FF | 5F | 5E | AB | 01 | .....VW.X... ^.. |
| 00000820 | CE | 80 | 3E | BB | 74 | 02 | EB | ED | C3 | 55 | 52 | 4C | 4D | 4F | 4E | 2E | ..>.t....URLMON. |
| 00000830 | 44 | 4C | 4C | 00 | 55 | 52 | 4C | 44 | 6F | 77 | 6E | 6C | 6F | 61 | 64 | 54 | DLL.URLDownloadT |
| 00000840 | 6F | 46 | 69 | 6C | 65 | 41 | 00 | 70 | 64 | 66 | 75 | 70 | 64 | 2E | 65 | 78 | oFileA.pdfupd.ex |
| 00000850 | 65 | 00 | 63 | 72 | 61 | 73 | 68 | 2E | 70 | 68 | 70 | 00 | 68 | 74 | 74 | 70 | e.crash.php.http |
| 00000860 | 3A | 2F | 2F | 37 | 37 | 2E | 37 | 38 | 2E | 32 | 34 | 30 | 2E | 38 | 39 | 2F | ://77.78.240.89/ |
| 00000870 | 78 | 78 | 33 | 2F | 6C | 2E | 70 | 68 | 70 | 3F | 69 | 3D | 38 | 00 | 90 | 90 | xx3/l.php?i=8... |
| 00000880 | 90 | 90 | 90 | 90 | 90 | 90 | 90 | 90 | 90 | 90 | 90 | 90 | 90 | 90 | 90 | 90 | ................ |

Figure 6. Shellcode hidden in embedded TIFF image located XML data

Keep in mind that APT attacker will create exploit for particular targets using specific program exploits. Most of the samples includes XML data compressed in XFA and AcroForm to store the embedded TIFF image. JavaScript stream and object stream are the other two structures commonly exist in the samples. However, there is one sample that is completely different from the others, which only contains /Action or /Launch script. Also, there are two other particular samples using the same encoding or encryption technique that Profiler and PDF Examiner could not recognize; both samples contains large block of unreferenced data stored in the object streams shown in Figure 7, so it is possible that this kind of sample includes the use of the callee-trick instead of the normal embedded TIFF vulnerability; in this case, the beginner could rely the severe rate bar that locates at the tool bar section of the Cerbero's Profiler, displaying the sample is 100% malicious.



100% SHA-1 62DA2B9462A2AB7B47E261E3886B3BA2F97DD4A4

7E-2010-0188_PDF_0A0E4B020F33705BD5F52FD5A5B2D78C]

| Offset | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F | Ascii | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00000000 | 25 | 50 | 44 | 46 | 2D | 31 | 2E | 33 | 0A | 31 | 20 | 30 | 20 | 6F | 62 | 6A | %PDF-1.3.1.0.obj | <-Format data - Foreign data - Unknown/unref data |
| 00000010 | 0A | 3C | 3C | 0A | 2F | 4B | 69 | 64 | 73 | 20 | 5B | 20 | 34 | 20 | 30 | 20 | .<<./Kids.[.4.0. | <-Unknown/unref data |
| 00000020 | 52 | 20 | 5D | 0A | 2F | 54 | 79 | 70 | 65 | 20 | 2F | 50 | 61 | 67 | 65 | 73 | R.]./Type./Pages | |
| 00000030 | 0A | 2F | 43 | 6F | 75 | 6E | 74 | 20 | 31 | 0A | 3E | 3E | 0A | 65 | 6E | 64 | ./Count.1.>>.end | |
| 00000040 | 6F | 62 | 6A | 0A | 32 | 20 | 30 | 20 | 6F | 62 | 6A | 0A | 3C | 3C | 0A | 2F | obj.2.0.obj.<<./ | <-Unknown/unref data - Foreign data |
| 00000050 | 43 | 72 | 65 | 61 | 74 | 69 | 6F | 6E | 44 | 61 | 74 | 65 | 28 | 44 | 3A | 32 | CreationDate(D:2 | <-Unknown/unref data |
| 00000060 | 30 | 30 | 35 | 30 | 39 | 32 | 39 | 30 | 39 | 32 | 37 | 33 | 34 | 5A | 29 | 20 | 00509290927342). | |
| 00000070 | 2F | 50 | 72 | 6F | 64 | 75 | 63 | 65 | 72 | 28 | 41 | 64 | 6F | 62 | 65 | 20 | /Producer(Adobe. | |
| 00000080 | 50 | 44 | 46 | 20 | 4C | 69 | 62 | 72 | 61 | 72 | 79 | 20 | 38 | 2E | 30 | 29 | PDF.Library.8.0) | |
| 00000090 | 0A | 3E | 3E | 0A | 65 | 6E | 64 | 6F | 62 | 6A | 0A | 33 | 20 | 30 | 20 | 6F | .>>.endobj.3.0.o | <-Unknown/unref data - Foreign data |
| 000000A0 | 62 | 6A | 0A | 3C | 3C | 0A | 2F | 54 | 79 | 70 | 65 | 20 | 2F | 43 | 61 | 74 | bj.<<./Type./Cat | <-Unknown/unref data |
| 000000B0 | 61 | 6C | 6F | 67 | 0A | 2F | 50 | 61 | 67 | 65 | 73 | 20 | 31 | 20 | 30 | 20 | alog./Pages.1.0. | |
| 000000C0 | 52 | 0A | 3E | 3E | 0A | 65 | 6E | 64 | 6F | 62 | 6A | 0A | 34 | 20 | 30 | 20 | R.>>.endobj.4.0. | <-Unknown/unref data - Foreign data |
| 000000D0 | 6F | 62 | 6A | 0A | 3C | 3C | 0A | 2F | 50 | 61 | 72 | 65 | 6E | 74 | 20 | 31 | obj.<<./Parent.1 | <-Unknown/unref data |
| 000000E0 | 20 | 30 | 20 | 52 | 0A | 2F | 43 | 6F | 6E | 74 | 65 | 6E | 74 | 73 | 20 | 34 | .0.R./Contents.4 | |
| 000000F0 | 20 | 30 | 20 | 52 | 0A | 2F | 54 | 79 | 70 | 65 | 20 | 2F | 50 | 61 | 67 | 65 | .0.R./Type./Page | |
| 00000100 | 0A | 2F | 52 | 65 | 73 | 6F | 75 | 72 | 63 | 65 | 73 | 20 | 3C | 3C | 0A | 2F | ./Resources.<<./ | |
| 00000110 | 50 | 72 | 6F | 63 | 53 | 65 | 74 | 20 | 5B | 20 | 2F | 50 | 44 | 46 | 20 | 2F | ProcSet.[./PDF./ | |
| 00000120 | 54 | 65 | 78 | 74 | 20 | 2F | 49 | 6D | 61 | 67 | 65 | 42 | 20 | 2F | 49 | 6D | Text./ImageB./Im | |
| 00000130 | 61 | 67 | 65 | 43 | 20 | 2F | 49 | 6D | 61 | 67 | 65 | 49 | 20 | 5D | 0A | 2F | ageC./ImageI.]./ | |
| 00000140 | 46 | 6F | 6E | 74 | 20 | 3C | 3C | 0A | 2F | 46 | 31 | 20 | 39 | 20 | 30 | 20 | Font.<<./F1.9.0. | |
| 00000150 | 52 | 0A | 3E | 3E | 0A | 3E | 3E | 0A | 2F | 4D | 65 | 64 | 69 | 61 | 42 | 6F | R.>>.>>./MediaBo | |
| 00000160 | 78 | 20 | 5B | 20 | 30 | 20 | 30 | 20 | 35 | 39 | 35 | 20 | 38 | 34 | 32 | 20 | x.[.0.0.595.842. | |
| 00000170 | 5D | 0A | 3E | 3E | 0A | 65 | 6E | 64 | 6F | 62 | 6A | 0A | 35 | 25 | 0A | 30 | ].>>.endobj.5%.0 | <-Unknown/unref data - Foreign data |
| 00000180 | 20 | 6F | 62 | 6A | 3C | 3C | 0A | 2F | 4C | 65 | 6E | 67 | 74 | 68 | 20 | 2B | .obj<<./Length.+ | <-Foreign data |
| 00000190 | 31 | 30 | 30 | 32 | 2F | 46 | 69 | 6C | 74 | 65 | 72 | 20 | 2F | 46 | 6C | 61 | 1002/Filter./Fla | |

Figure 7. Profiler detects unreferenced data block to be 100% malicious

**Destination IP Tracking: VirusTotal [14], Site Safety Center [15], urlQuery [16]**

Although the analysis tools and online scanners are generally accurate about the CVE-2010-0188 sample being malicious or not, the URL extracted from embedded TIFF image can be further studied or investigated by uploading them to VirusTotal, Trend Micro's Site Safety Center, and urlQuery. The samples using the same URL or same destination IP are considered to be created by the same attacker or organization. Unfortunately, there are no clear indication on how the samples with the same attacker leads to failed detection from XecScan or PDF Examiner. Out of fourty-nine CVE-2010-0188 samples, 3.26 anti-viruses are able to detect a suspicious or malicious URL from each case. Kaspersky, Websense ThreatSeeker, and BitDefender are the three with the highest detection rate.

After uploading to three different URL scanners, VirusTotal will send the samples to common anti-viruses available on the internet and summarize the results. On the other hand, Site Safety Center will simply check whether the URL and the destination IP is dangerous or not using its own signature database. Unfortunately, Site Safety Center's result usually does not agree with the results of urlQuery and VirustTotal, which the two often have similar results. urlQuery is the most useful one out of the three; urlQuery not only provides severity checking of the destination IP using its sandbox, but it also reports other connected webpages in hierarchy structure and the script used in the destination IP.

VirusTotal detects most of the destination IP from the samples to be malicious. There are exceptions, but these exceptions usually happens because either website of the destination IP is removed or the websites has been cleaned. Most existing malicious website will initiate one attack from one IP address, except for one particular case. XecScan marked this sample to be malicious while PDF scanner said the sample is safe. The destination IP extracted from the sample is 50.87.141.179, which initiates two attack from the same IP address then another attack from a different IP address within two seconds as shown in Figure 8.

| | Timestamp | Severity | Source IP | Destination IP | Alert |
|---|---|---|---|---|---|
| Snort with Sourcefire VRT | 10:41:34 | 1 | urlQuery Client | 50.87.141.179 (U.S.) | EXPLOIT-KIT Bleeding Life exploit kit module call |
| | 10:41:36 | 1 | urlQuery Client | 50.87.141.179 (U.S.) | EXPLOIT-KIT Bleeding Life exploit kit module call |
| | 10:41:43 | 1 | urlQuery Client | 50.87.141.179 (U.S.) | EXPLOIT-KIT Bleeding Life exploit kit module call |
| Suricata with Emerging Threats Pro | Timestamp | Severity | Source IP | Destination IP | Alert |
| | 10:41:36 | 1 | urlQuery Client | 50.87.141.179 (U.S.) | EXPLOIT-KIT Bleeding Life exploit kit module call |
| | 10:41:38 | 1 | urlQuery Client | 50.87.141.179 (U.S.) | EXPLOIT-KIT Bleeding Life exploit kit module call |

Figure 8. urlQuery's intrusion detection example [16]

In the end, U.S. and Ukraine are the top two countries that these malicious websites are located. One is safe to say that the samples that uses the same destination IP to download the malware are written from the same author or organization, and studying the categorized code with respect to the author would help identifying similar coding styles and implementation techniques among other samples.

# 5. Conclusion and Future Work

Even though there are many tools to analyze a malformed PDF document, a solid method to defend against APT attack and spear phishing will be scanning the attachment using online scanner tools like Xecure or PDF Examiner, which are both accurate in most cases. To further study the samples, analyzing with PDF Stream Dumper or Profiler is highly recommended in order to investigate the malicious code hidden within. For the CVE-2010-0188 case study, urlQuery and VirusTotal is also recommended for identifying the source of attack so the human analyst can study specific samples written by the same author or organization. From the CVE-2010-0188 case study, we can conclude the following:

1. Most embedded TIFF image can be converted from base64 and using JavaScript escaping after extracting from the compressed XML data; except there are a very few cases where the encryption method is unknown or the XML data is missing.
2. Shellcode extracted from embedded TIFF image contains suspicious URL and instruction sets that downloads malware from the suspicious URL. This encoding and obfuscation techniques are categorized as a non-JavaScript based obfuscation.
3. XecScan generally has a higher detection rate than PDF Examiner while PDF Examiner provides more information for later examination; though sometimes it is completely the other way around with some samples.
4. Out of the 58 anti-viruses provided by VirustTotal testing on all extracted URL, 3.26 anti-viruses on an average are able to identify the malicious URL extracted from the samples; the top three products that have the highest detection rate are Kaspersky, Websense ThreatSeeker, and BitDefender.
5. Tracking the destination IP using the URL obtained from shellcode would not help improving the detection accuracy of XecScan and PDF Examiner, but it will help categorizing the attackers with similar coding style, encoding or encryption pattern.

Although we are certain that malformed PDF document with malicious embedded TIFF images and its URL is considered to be malicious, extracting the malicious code from the embedded TIFF image and debug it using Profiler and the debugging tool OllyDbg [17] is a more advanced method to study the shellcode. By using the OllyDbg, additional study the assembly code will help understand the coding style and instructions of the attacker. One may also choose to run PDF Stream Dumper's sandbox tool scLog or scDbg for testing the samples. Hopefully by categorizing and understanding

these samples and concepts will help more people understand the threat and build awareness against malformed PDF documents, fighting and preventing from APT and spear phishing attack.

# References


[1] **Stuart McClure, Jeol Scambray, George Kurtz, (2012), "Hacking Exposed 7", McGraw-Hill**

[2] **廖峰澤, 林盈達, (2013-09-30), "APT 逆向與正向工程技術", 國立交通大學資訊工程系**

[3] **Mahmud Ab Rahman, (2010), "Getting Owned By Malicious PDF – Analysis", SANS Institute**

[4] **Sebastian Porst, (2010), "How to really obfuscate your PDF malware", ReCon, http://recon.cx/2010/slides/recon_2010_sebastian_porst.pdf**

[5] **ange.alb...@gmail.com, (2014-05-19), "PDFTricks", corkami, https://code.google.com/p/corkami/wiki/PDFTricks#JavaScript_-based_encodings**

[6] **Karthik Selvaraj, Nino Fred Gutierrez, (2010-3-29), "The Rise of PDF Malware", Symantec: Security Respond**

[7] **dzzie2, *Pdf Exploit Analysis with PDFStreamDumper - Encrypted Script*, http://youtu.be/UWeAom4La6g**

[8] **Xecure Lab Co., Copyright 2012, "XecScan Rapid APT Identification Service", http://scan.xecure-lab.com/**

[9] **Malware Tracker limited., Copyright 2009-2014, "pdf examiner", https://www.malwaretracker.com/pdf.php**

[10] **Cerbero GmbH., Copyright 2014, "Profiler", http://cerbero.io/profiler/**

[11] **David Zimmer, (2010-07-21), "PDF Stream Dumper", http://sandsprite.com/blogs/index.php?uid=7&pid=57**

[12] **Daniel Pistelli, (2013-05-21) , "CVE-2010-0188: PDF/Form/TIFF", http://cerbero-blog.com/?p=1076**

[13] **Malware Tracker, (2013-06-16), "PDF Current Threats", http://www.malwaretracker.com/pdfthreat.php, viewed on (2014-07-04)**

[14] **VirusTotal, https://www.virustotal.com/en/**

[15] **Trend Micro, Copyright 2014, "Site Safety Center", http://global.sitesafety.trendmicro.com/**

[16] **urlQurey.net, Copyright 2011, "urlQuery", http://www.urlquery.net/**

[17] **OllyDbg, (2014-05-02), http://www.ollydbg.de/**

[18] **Common Vulnerabilities and Exposures (CVE ), Copyright 1999-2014, https://cve.mitre.org/**

[19] **"Advance Persistent Threat", Wikipedia, (2014-7-11), http://en.wikipedia.org/wiki/Advanced_persistent_threat**

[20] **decalage, (2013-11-15), "PDF Security Issues", http://www.decalage.info/file_formats_security/pdf**

[21] **Teddy Reed V, (2010-08-20), "Analyzing CVE-2010-0188 exploits: The Legend of Pat Casey (Part 1)", http://prosauce.org/blog/2010/8/20/analyzing-cve-2010-0188exploits-the-legend-of-pat-casey-par.html**

[22] **Teddy Reed V, (2010-10-02), "Analyzing CVE-2010-0188 exploits: Context aware malware (Part 2)", http://prosauce.org/blog/2010/10/2/analyzing-cve-2010-0188-exploits-context-aware-malware-part.html**

[23] **Mila, *Contagio malware dump*, http://contagiodump.blogspot.e/**

[24] ***bugix - security research*, "CVE-2010-0188 Adobe Working Exploit", http://bugix-security.blogspot.tw/2010/03/adobe-pdf-libtiff-working-exploitcve.html**

[25] **sketchymoose, "PDF Analysis using PDF Stream Dumper", (2011-12-21), http://www.securitytube.net/video/2602**

[26] **Prateek Dewan, Anand Kashyap, Ponnurangam Kumaraguru, (2014-6-14), "Analyzing Social and Stylometric Features to Identify Spear Phishing Emails"**

[27] **dzzie2, *PdfStreamDumper_trainer*, http://sandsprite.com/CodeStuff/PdfStreamDumper_trainer.wmv**