Extract Me if You Can: Abusing PDF Parsers in Malware Detectors

APR 25TH, 2016

论文下载

这篇文章发表在NDSS 16上，作者是Syracuse University的Curtis Carmony。这篇文章里，作者针对PDF的malware，对Adobe Reader做了个javascript提取工具，来自动化地提取恶意PDF里的javascript。然后作者分析了现有PDF病毒分析工具存在的缺陷，并总结了一些PDF javascript的混淆方法，来绕过PDF病毒分析工具和杀软。最后，作者提出了一些应对的措施，来加强PDF病毒的检测。

背景

Signature-based malicious PDF detectors

基于特征值和哈希来判断PDF是否是恶意的。

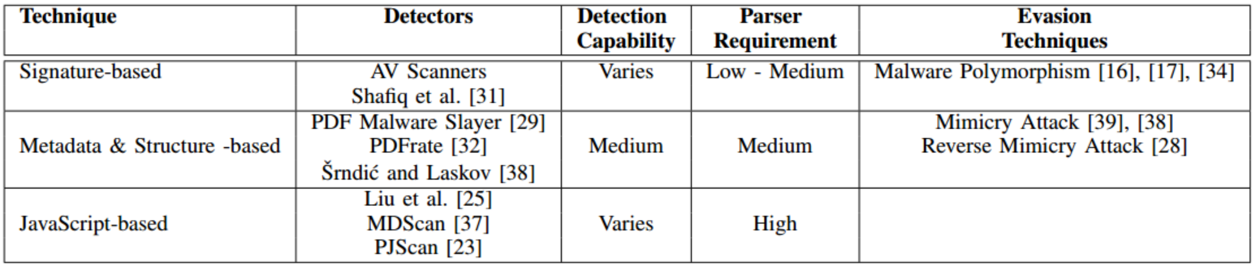
Metadata and Structural Features Based Detection

- 基于对PDF文件结构的分析，来判断PDF是否是恶意的。
- PDF Malware Slayer和PDFrate都使用了机器学习的方法，Random Forest，对PDF文件结构进行分类学习。攻击者可以伪造PDF的文件结构来bypass这类检测器。

JavaScript Based Detection

- 基于对PDF文件里的javascript代码，进行分析，来判断PDF是否是恶意的。
- MDScan会解析并提取出PDF里面的javascript代码，然后在一个改过的js引擎里执行，并进行检测。
- PJScan用机器学习的方法，One-Class SVM，来判断js代码是否是恶意的。
- MPScan直接hook了Adobe Reader的js引擎，在执行过程中判断是否执行的恶意行为。

TABLE I: Existing PDF Classifiers



Technique	Detectors	Detection Capability	Parser Requirement	Evasion Techniques
Signature-based	AV Scanners Shafiq et al. [31]	Varies	Low - Medium	Malware Polymorphism [16], [17], [34]
Metadata & Structure -based	PDF Malware Slayer [29] PDFrate [32] Šrndić and Laskov [38]	Medium	Medium	Mimicry Attack [39], [38] Reverse Mimicry Attack [28]
JavaScript-based	Liu et al. [25] MDScan [37] PJScan [23]	Varies	High	

Reference Javascript Extractor

作者认为，只有Javascript-based检测器比较有前途，然而，现在的这类检测器的实现都存在许多问题。作者认为，怎么提取出PDF文件里面的javascript代码是个很重要的问题。

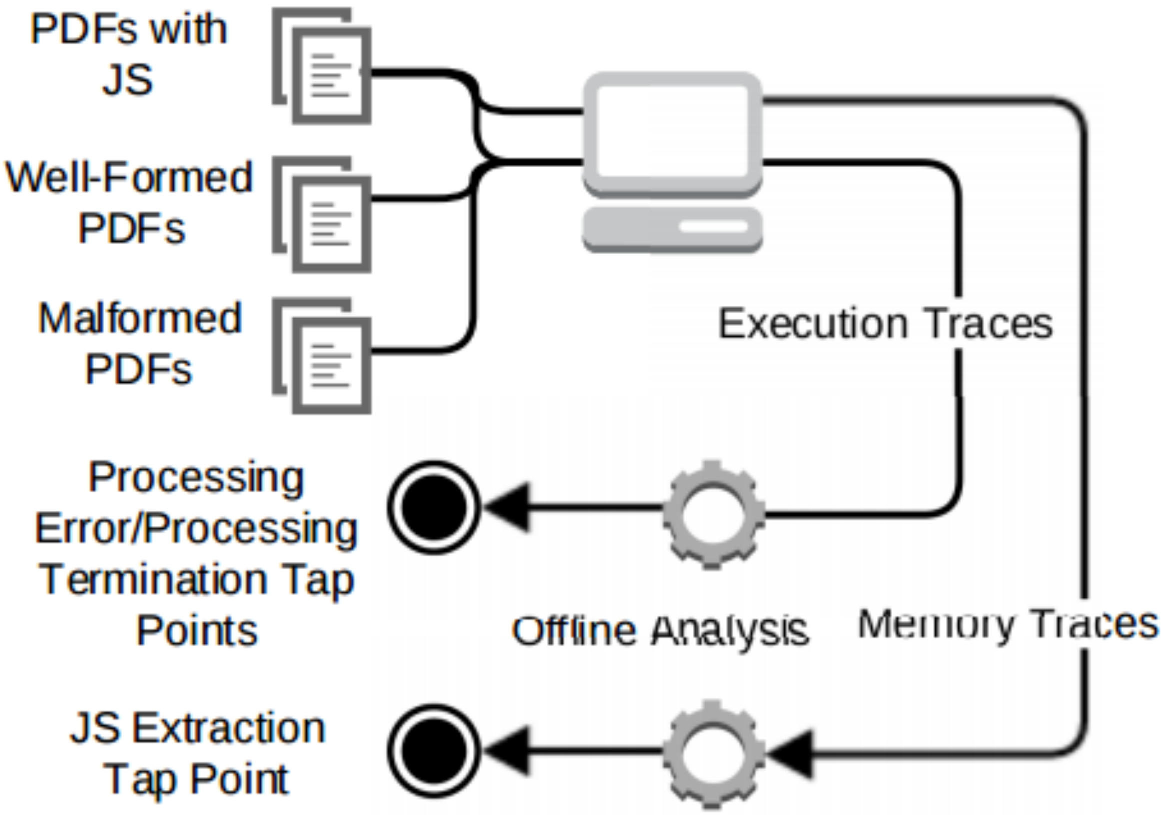
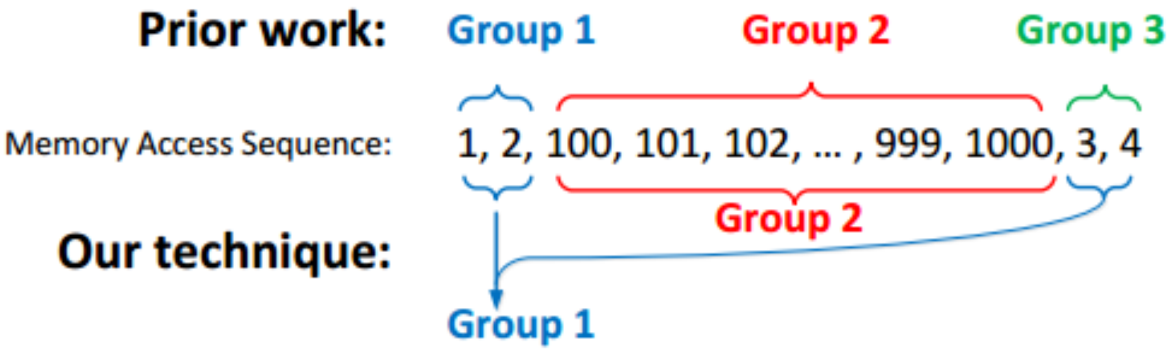


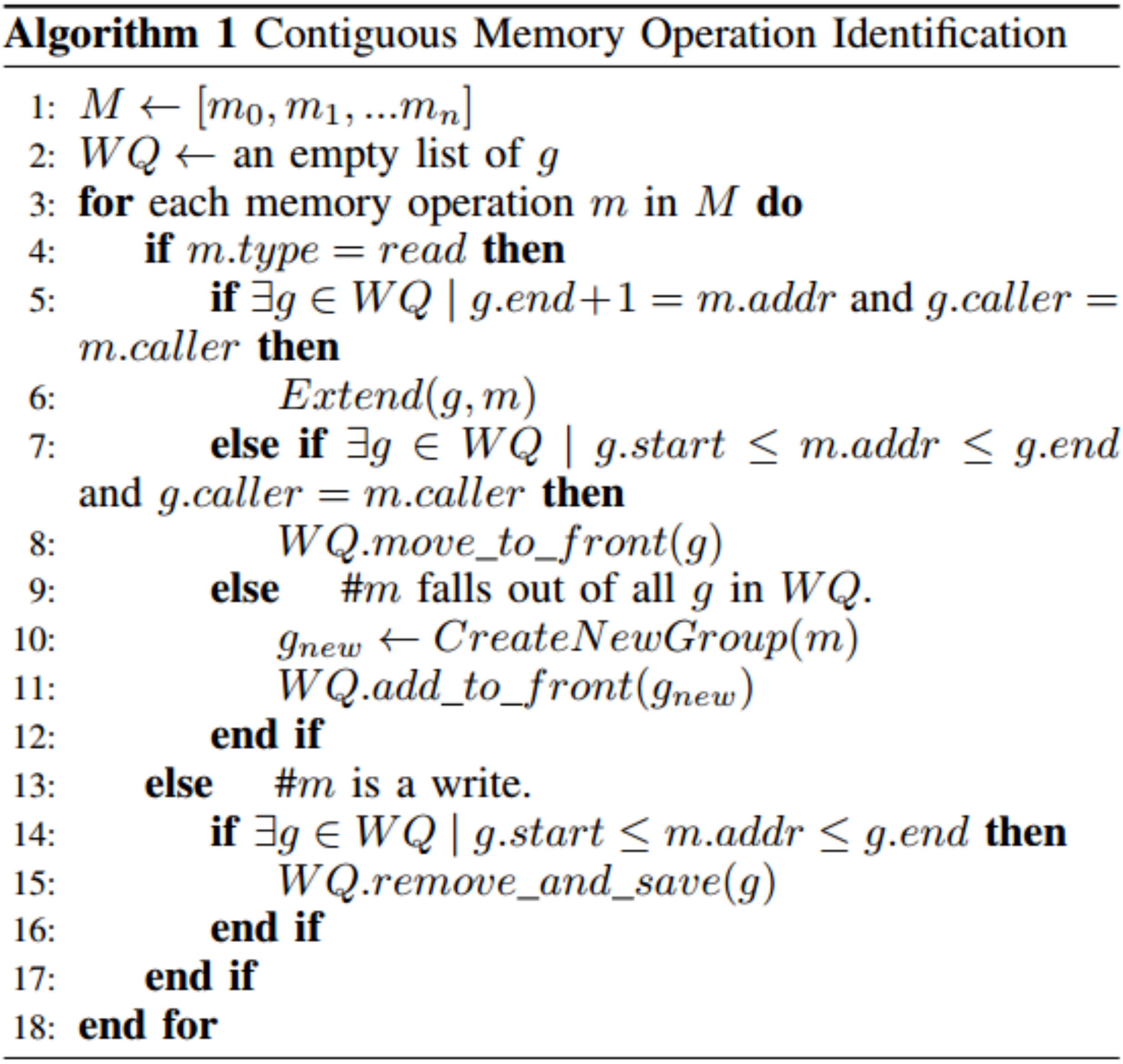
Fig. 1: Tap Point Identification.

Reference Extractor是基于TZB实现的一个工具，TZB（Tappan Zee (North) Bridge: Mining Memory Accesses for Introspection）是一个基于Panda的污点分析工具。
作者记录下Adobe Reader解析PDF文件的指令序列，然后进行分析。作者使用了三种输入:有JS的PDF，正常的PDF，Malformed PDF。分别记录下解析它们的指令序列。然后线下分析，找出三个Tap Point：JS提取完成的点，Adobe Reader进程正常结束的点，进程异常退出的点。
对第一个Tap Point，作者对每条内存读写指令记录下：

$$m = (caller, program\_counter, type, data, addr)$$

然后把内存地址连续的操作归类。





Algorithm 1 Contiguous Memory Operation Identification

```
$M \leftarrow [m_0, m_1, ...m_n]$
$WQ \leftarrow$ an empty list of $g$
for each memory operation $m$ in $M$ do
    if $m.type = read$ then
        if $\exists g \in WQ \mid g.end+1 = m.addr$ and $g.caller = m.caller$ then
            $Extend(g, m)$
        else if $\exists g \in WQ \mid g.start \leq m.addr \leq g.end$ and $g.caller = m.caller$ then
            $WQ.move\_to\_front(g)$
        else    #$m$ falls out of all $g$ in $WQ$.
            $g_{new} \leftarrow CreateNewGroup(m)$
            $WQ.add\_to\_front(g_{new})$
        end if
    else    #$m$ is a write.
        if $\exists g \in WQ \mid g.start \leq m.addr \leq g.end$ then
            $WQ.remove\_and\_save(g)$
        end if
    end if
end for
```

然后，匹配每组内存中的数据，是否包含javascript的关键字。如果存在，就说明提取出了javascript，把调用这些指令的函数设成Tap Point。

对后两类Tap Point，作者分别记录下ETJS，ETWF，ETMF，分别表示有JS的PDF，正常的PDF，Malformed PDF对应的指令集合。

正常退出的Tap Point指：

1. 在ETWF里，the basic block is always executed once and only once
2. 不在ETMF里
3. 在ETJS里，只出现在Javascript Tap Point之后

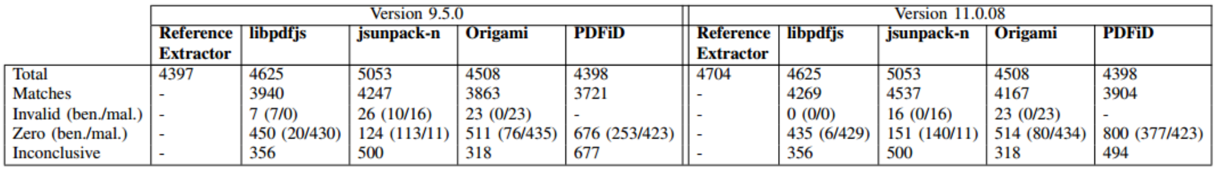
异常退出的Tap Point指：

1. 在ETMF里，the basic block is always executed once and only once
2. 不再ETWF里

后续执行时，遇到Tap Point时，作者用Microsoft Detours library来做一系列操作。

Evaluation

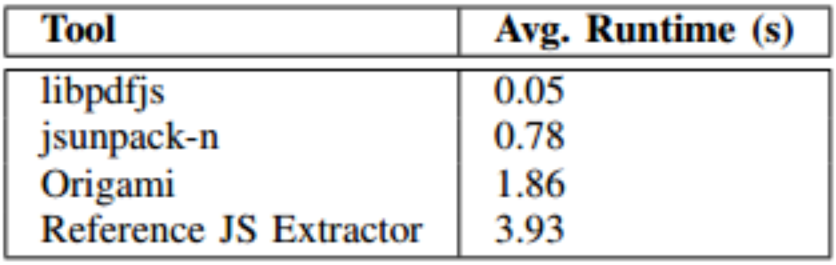
TABLE II: JavaScript Extractions



	Version 9.5.0					Version 11.0.08				
	Reference Extractor	libpdfjs	jsunpack-n	Origami	PDFiD	Reference Extractor	libpdfjs	jsunpack-n	Origami	PDFiD
Total	4397	4625	5053	4508	4398	4704	4625	5053	4508	4398
Matches	-	3940	4247	3863	3721	-	4269	4537	4167	3904
Invalid (ben./mal.)	-	7 (7/0)	26 (10/16)	23 (0/23)	-	-	0 (0/0)	16 (0/16)	23 (0/23)	-
Zero (ben./mal.)	-	450 (20/430)	124 (113/11)	511 (76/435)	676 (253/423)	-	435 (6/429)	151 (140/11)	514 (80/434)	800 (377/423)
Inconclusive	-	356	500	318	677	-	356	500	318	494

Reference Extractor是作者的提取工具。Matches指的是其他工具提取的javascript和作者的一样；Invalid指的是提取出来的javascript不一样；Zero指的是作者的工具提取出了javascript，但别的工具没有提取出来；Inconclusive指的是其他工具提取出来了但作者的工具没有提取出来。

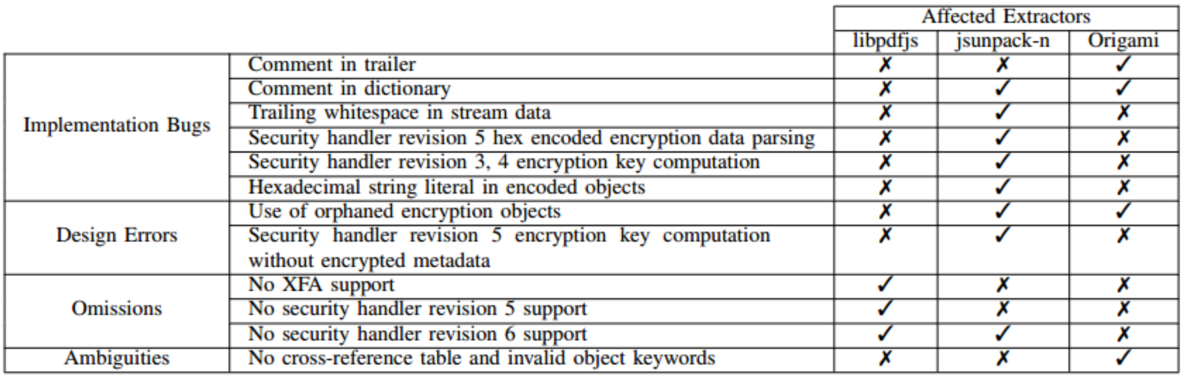
TABLE VIII: Average Runtime



Tool	Avg. Runtime (s)
libpdfjs	0.05
jsunpack-n	0.78
Origami	1.86
Reference JS Extractor	3.93

开源PDF分析工具存在的问题

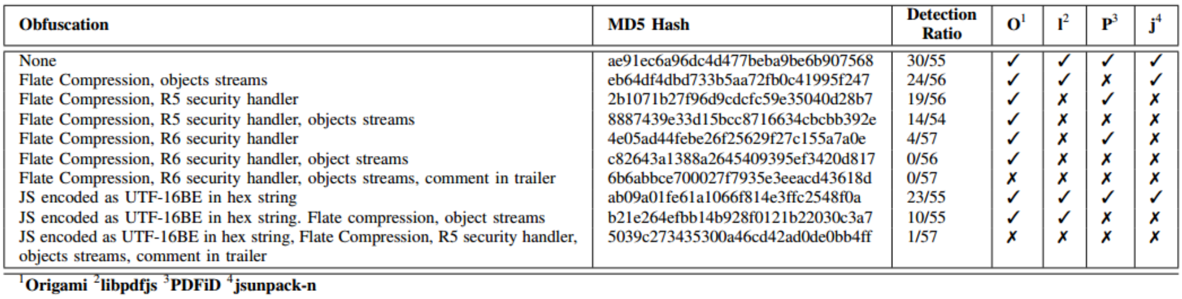
TABLE IV: Failings and Limitations



		Affected Extractors		
		libpdfjs	jsunpack-n	Origami
Implementation Bugs	Comment in trailer	✗	✗	✓
	Comment in dictionary	✗	✓	✓
	Trailing whitespace in stream data	✗	✓	✗
	Security handler revision 5 hex encoded encryption data parsing	✗	✓	✗
	Security handler revision 3, 4 encryption key computation	✗	✓	✗
	Hexadecimal string literal in encoded objects	✗	✓	✗
Design Errors	Use of orphaned encryption objects	✗	✓	✓
	Security handler revision 5 encryption key computation without encrypted metadata	✗	✓	✗
Omissions	No XFA support	✓	✗	✗
	No security handler revision 5 support	✓	✗	✗
	No security handler revision 6 support	✓	✓	✗
Ambiguities	No cross-reference table and invalid object keywords	✗	✗	✓

作者提出的PDF混淆方法和测试

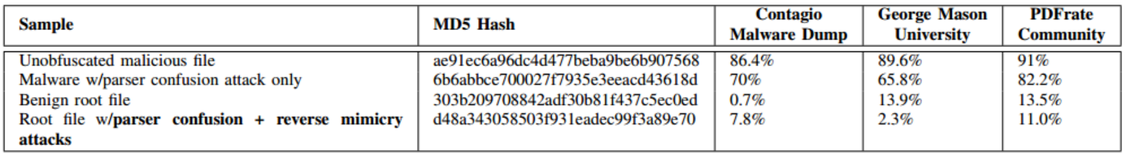
TABLE V: Parser Confusion Attacks on Commercial Detectors and JS Extractors



Obfuscation	**MD5 Hash**	**Detection Ratio**	**O**[1]	**l**[2]	**P**[3]	**j**[4]
None	ae91ec6a96dc4d477beba9be6b907568	30/55	✓	✓	✓	✓
Flate Compression, objects streams	eb64df4dbd733b5aa72fb0c41995f247	24/56	✓	✓	✗	✓
Flate Compression, R5 security handler	2b1071b27f96d9cdcfc59e35040d28b7	19/56	✓	✗	✓	✗
Flate Compression, R5 security handler, objects streams	8887439e33d15bcc8716634cbcbb392e	14/54	✓	✗	✗	✗
Flate Compression, R6 security handler	4e05ad44febe26f25629f27c155a7a0e	4/57	✓	✗	✓	✗
Flate Compression, R6 security handler, object streams	c82643a1388a2645409395ef3420d817	0/56	✓	✗	✗	✗
Flate Compression, R6 security handler, objects streams, comment in trailer	6b6abbce700027f7935e3eeacd43618d	0/57	✗	✗	✗	✗
JS encoded as UTF-16BE in hex string	ab09a01fe61a1066f814e3ffc2548f0a	23/55	✓	✓	✓	✓
JS encoded as UTF-16BE in hex string. Flate compression, object streams	b21e264efbb14b928f0121b22030c3a7	10/55	✓	✓	✗	✗
JS encoded as UTF-16BE in hex string, Flate Compression, R5 security handler, objects streams, comment in trailer	5039c273435300a46cd42ad0de0bb4ff	1/57	✗	✗	✗	✗

[1]**Origami** [2]**libpdfjs** [3]**PDFiD** [4]**jsunpack-n**

TABLE VI: PDFrate Evasion



Sample	**MD5 Hash**	**Contagio Malware Dump**	**George Mason University**	**PDFrate Community**
Unobfuscated malicious file	ae91ec6a96dc4d477beba9be6b907568	86.4%	89.6%	91%
Malware w/parser confusion attack only	6b6abbce700027f7935e3eeacd43618d	70%	65.8%	82.2%
Benign root file	303b209708842adf30b81f437c5ec0ed	0.7%	13.9%	13.5%
Root file w/**parser confusion + reverse mimicry attacks**	d48a343058503f931eadec99f3a89e70	7.8%	2.3%	11.0%

Reference Extractor配合PJScan的测试

TABLE VII: PJScan Performance

Tool	**True Positive**	**False Positive**
Original PJScan	68.34% (1453)	0.18% (3814)
PJScan & Adobe Reader 9.5.0	96.04% (1441)	0.32% (3521)
PJScan & Adobe Reader 11.0.08	94.02% (1021)	0.20% (3677)