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

Curtis Carmony, Mu Zhang, Xunchao Hu,
Abhishek Vasisht Bhaskar, and Heng Yin

Department of EECS, Syracuse University

L.C.Smith College of Engineering and Computer Science

Malicious PDF Detection is important!

- 129 Adobe Reader CVE's in 2015
    - Up from 44 in 2014
- Existing detection techniques have limitations
- Malicious PDF detection is difficult:
    - The PDF format is very complex and evolving
    - Adobe Reader with often process PDFs deviating from the specification in an attempt to “just work”

Existing Malicious PDF Detection Methods

Technique	Detectors	Detection Capability	Parser Requirement	Evasion Techniques
Signature-based	AV Scanners Shafiq et al.	Varies	Low - Medium	Malware Polymorphism
Metadata & Structure -based	PDF Malware Slayer PDFrate Šrndić and Laskov	Medium	Medium	Mimicry Attack Reverse Mimicry Attack
JavaScript-based	Liu et al. MDScan PJScan	Varies	High	

Parsing Matters

- We need to actually look for malicious content
- JavaScript based detection methods are most likely to detect modern threats, but have highest parser requirements
- Successful malicious PDF detection depends on accurate and reliable parsing

Hypotheses

- Significant parsing discrepancies between detectors and Adobe Reader likely exist
- By improving the parser and removing these discrepancies existing detection methods can be improved

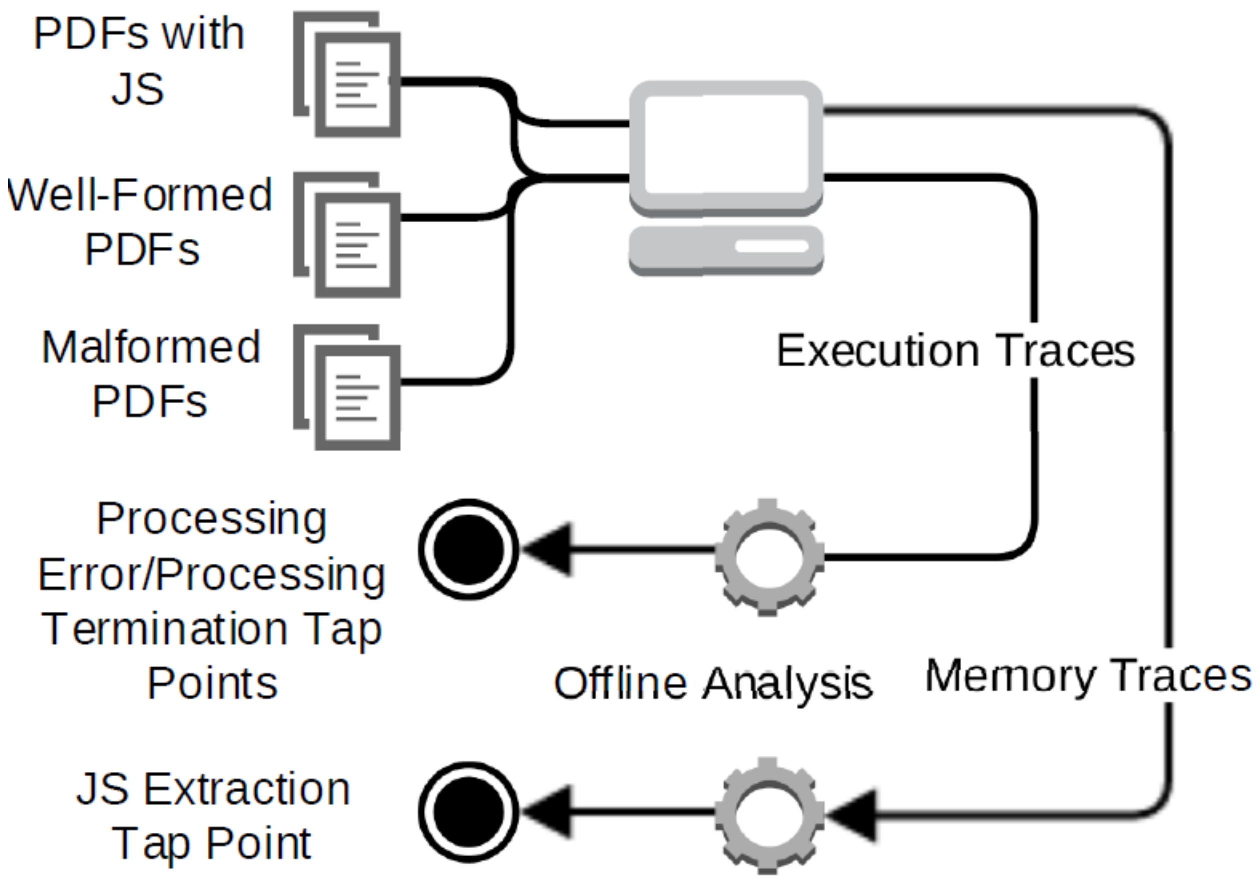
The Reference Extractor

- To evaluate our hypotheses we need to know:
  - Which files Adobe Reader will actually open and those which it will not
  - Precisely the JS Adobe Reader executes
- We can modify Adobe Reader to produce this information – “reference extractor”
- Each reference extractor is specific to a version of Adobe Reader
  - We need a technique which is robust and repeatable
  - Mostly-automatic/low level of manual effort

Development of the Reference Extractor

- Identify “tap points” – locations in Adobe Reader binary where we can extract information:
  - processing termination – indicates Adobe Reader has finished initial processing of file
  - processing error – indicates Adobe Reader has encountered an error during initial processing
  - JavaScript extraction – yields a reference to all executed JavaScript

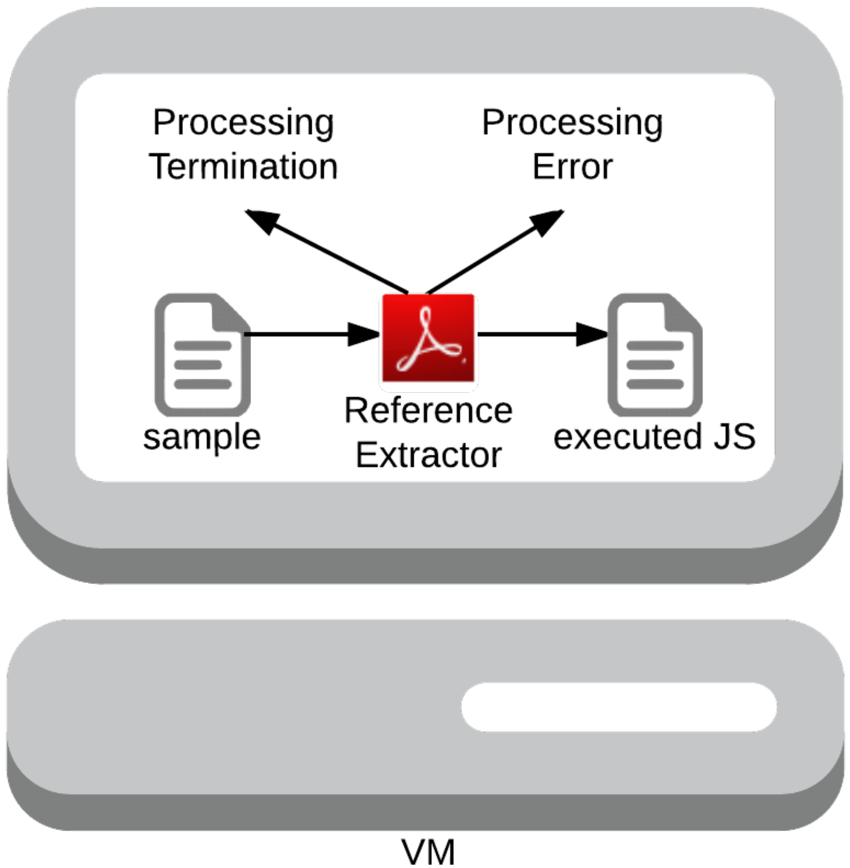
Development of the Reference Extractor



Tap Point Identification

- Processing Error/Processing Termination tap points:
  - Compare execution traces to identify basic-blocks executed precisely when the conditions for each tap point are met
- JavaScript extraction tap point:
  - Group memory accesses into contiguous memory operations
  - Look for JavaScript which we know was executed
  - Based on existing technique (Dolan-Gavitt et al. ’13)
- Full details are in paper

Reference Extractor Deployment



Data Set

- Collected 163,306 PDF’s from VT, no restrictions
- Ran them through two reference extractors and four open source tools
- 5,267 were identified as containing JavaScript by any single tool
- 1,453 of the samples we consider malicious with 15 or more VT detections

Differential Analysis Results

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

Differential Analysis Results

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

Differential Analysis Results

	Version 9.5.0				
	Reference Extractor	libpfjs	jsunpack-n	Origami	PDFiD
Total	4397	4625	5053	4508	4398
Matches	-	3940	4247	3863	3721
Invalid (ben./mal.)	-	7(7/0)	26(10/16)	23(0/23)	-
Zero (ben./mal.)	-	450(20/430)	124(113/11)	511(76/435)	676(253/423)
Inconclusive	-	356	500	318	677

	Version 11.0.08				
	Reference Extractor	libpfjs	jsunpack-n	Origami	PDFiD
Total	4704	4625	5053	4508	4398
Matches	-	4269	4537	4167	3904
Invalid (ben./mal.)	-	0(0/0)	16(0/16)	23(0/23)	-
Zero (ben./mal.)	-	435(6/429)	151(140/11)	514(80/434)	800(377/423)
Inconclusive	-	356	500	318	494

Differential Analysis Results

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

Differential Analysis Results

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

Differential Analysis Results

	Version 9.5.0				
	Reference Extractor	libpfjs	jsunpack-n	Origami	PDFiD
Total	4397	4625	5053	4508	4398
Matches	-	3940	4247	3863	3721
Invalid (ben./mal.)	-	7(7/0)	26(10/16)	23(0/23)	-
Zero (ben./mal.)	-	450(20/430)	124(113/11)	511(76/435)	676(253/423)
Inconclusive	-	356	500	318	677

	Version 11.0.08				
	Reference Extractor	libpfjs	jsunpack-n	Origami	PDFiD
Total	4704	4625	5053	4508	4398
Matches	-	4269	4537	4167	3904
Invalid (ben./mal.)	-	0(0/0)	16(0/16)	23(0/23)	-
Zero (ben./mal.)	-	435(6/429)	151(140/11)	514(80/434)	800(377/423)
Inconclusive	-	356	500	318	494

Differential Analysis Results

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

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	×	×	✓

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	×	×	✓

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	✗	✗	✓

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	✗	✗	✓

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	×	×	✓

Bugs – Comment Injection

```
trailer << /Root 1 0 R /Size 8 >>
```

(a) Original Trailer

```
trailer << /Root %!@#!@#
1 0 R /Size 8 >>
```

(b) Trailer With Injected Comment

Omission - PDF Encryption

- PDF specification allows for encryption with blank password
- Most parsers struggle with encryption
- Adobe creates and opens PDFs using “un-published” R6 security handler from PDF 2.0 spec
- Only one tool we evaluated (Origami) supports this handler

Ambiguities – Document Recovery

- What should happen when document is malformed?
- Specification states that every PDF must contain cross-reference table listing objects and their locations in the file
- Adobe Reader and other applications will attempt to open files without this table by scanning the file for objects
- Specification makes no mention of this recovery or how it should be performed

Parser Confusion Attacks

- We call the exploitation of these discrepancies to evade detection *parser confusion attacks*
- Combine obfuscations to exploit multiple discrepancies
- Increase reliance on parser by maximizing the amount of malicious content which is encoded

Attack Construction

```
3 0 obj
<< /JS 6 0 R /S /JavaScript /Type /Action >>
endobj
...
6 0 obj
<< /Length 3907 >>
stream
function heapSpray(str, str_addr, r_addr) {
...
}
endstream
endobj
```

(a) Malicious JavaScript and reference are unobfuscated.

Attack Construction

```
3 0 obj
<< /JS 6 0 R /S /JavaScript /Type /Action
>>
endobj
...
6 0 obj
<< /Length 1552 /Filter /FlateDecode>>
stream
<encoded JavaScript>
endstream
endobj
```

(b) By applying stream filter, the malicious JavaScript is encoded but the reference is unobfuscated. Detectors which cannot decode the stream are only aware of the existence of JavaScript

Attack Construction

```
2 0 obj
<< /Type /ObjStm /Length 1696 /Filter
/FlateDecode /N 4 /First 20 >>
stream
<encoded objects>
endstream
endobj
```

(c) By placing objects in streams and then encoding them, the malicious JavaScript and references are obfuscated. No trace of the malicious JavaScript is left for detectors which cannot decode the stream.

Attack vs VT

Obfuscation	Detection Ratio	Origami	libpdfjs	PDFiD	jsunpack-n
None	30/55	✓	✓	✓	✓
Flate Compression, objects streams	24/56	✓	✓	×	✓
Flate Compression, R5 security handler	19/56	✓	×	✓	×
Flate Compression, R5 security handler, objects streams	14/54	✓	×	×	×
Flate Compression, R6 security handler	4/57	✓	×	✓	×
Flate Compression, R6 security handler, object streams	0/56	✓	×	×	×
Flate Compression, R6 security handler, objects streams, comment in trailer	0/57	×	×	×	×
JS encoded as UTF-16BE in hex string	23/55	✓	✓	✓	✓
JS encoded as UTF-16BE in hex string. Flate compression, object streams	10/55	✓	✓	×	×
JS encoded as UTF-16BE in hex string, Flate Compression, R5 security handler, objects streams, comment in trailer	1/57	×	×	×	×

Attack vs VT

Obfuscation	Detection Ratio	Origami	libpdfjs	PDFiD	jsunpack-n
None	30/55	✓	✓	✓	✓
Flate Compression, objects streams	24/56	✓	✓	×	✓
Flate Compression, R5 security handler	19/56	✓	×	✓	×
Flate Compression, R5 security handler, objects streams	14/54	✓	×	×	×
Flate Compression, R6 security handler	4/57	✓	×	✓	×
Flate Compression, R6 security handler, object streams	0/56	✓	×	×	×
Flate Compression, R6 security handler, objects streams, comment in trailer	0/57	×	×	×	×
JS encoded as UTF-16BE in hex string	23/55	✓	✓	✓	✓
JS encoded as UTF-16BE in hex string. Flate compression, object streams	10/55	✓	✓	×	×
JS encoded as UTF-16BE in hex string, Flate Compression, R5 security handler, objects streams, comment in trailer	1/57	×	×	×	×

Attack vs VT

Obfuscation	Detection Ratio	Origami	libpdfjs	PDFiD	jsunpack-n
None	30/55	✓	✓	✓	✓
Flate Compression, objects streams	24/56	✓	✓	✗	✓
Flate Compression, R5 security handler	19/56	✓	✗	✓	✗
Flate Compression, R5 security handler, objects streams	14/54	✓	✗	✗	✗
Flate Compression, R6 security handler	4/57	✓	✗	✓	✗
Flate Compression, R6 security handler, object streams	0/56	✓	✗	✗	✗
Flate Compression, R6 security handler, objects streams, comment in trailer	0/57	✗	✗	✗	✗
JS encoded as UTF-16BE in hex string	23/55	✓	✓	✓	✓
JS encoded as UTF-16BE in hex string. Flate compression, object streams	10/55	✓	✓	✗	✗
JS encoded as UTF-16BE in hex string, Flate Compression, R5 security handler, objects streams, comment in trailer	1/57	✗	✗	✗	✗

Attack vs VT

Obfuscation	Detection Ratio	Origami	libpdfjs	PDFiD	jsunpack-n
None	30/55	✓	✓	✓	✓
Flate Compression, objects streams	24/56	✓	✓	✗	✓
Flate Compression, R5 security handler	19/56	✓	✗	✓	✗
Flate Compression, R5 security handler, objects streams	14/54	✓	✗	✗	✗
Flate Compression, R6 security handler	4/57	✓	✗	✓	✗
Flate Compression, R6 security handler, object streams	0/56	✓	✗	✗	✗
Flate Compression, R6 security handler, objects streams, comment in trailer	0/57	✗	✗	✗	✗
JS encoded as UTF-16BE in hex string	23/55	✓	✓	✓	✓
JS encoded as UTF-16BE in hex string. Flate compression, object streams	10/55	✓	✓	✗	✗
JS encoded as UTF-16BE in hex string, Flate Compression, R5 security handler, objects streams, comment in trailer	1/57	✗	✗	✗	✗

Attack vs VT

Obfuscation	Detection Ratio	Origami	libpdfjs	PDFiD	jsunpack-n
None	30/55	✓	✓	✓	✓
Flate Compression, objects streams	24/56	✓	✓	✗	✓
Flate Compression, R5 security handler	19/56	✓	✗	✓	✗
Flate Compression, R5 security handler, objects streams	14/54	✓	✗	✗	✗
Flate Compression, R6 security handler	4/57	✓	✗	✓	✗
Flate Compression, R6 security handler, object streams	0/56	✓	✗	✗	✗
Flate Compression, R6 security handler, objects streams, comment in trailer	0/57	✗	✗	✗	✗
JS encoded as UTF-16BE in hex string	23/55	✓	✓	✓	✓
JS encoded as UTF-16BE in hex string. Flate compression, object streams	10/55	✓	✓	✗	✗
JS encoded as UTF-16BE in hex string, Flate Compression, R5 security handler, objects streams, comment in trailer	1/57	✗	✗	✗	✗

Attack vs PDFRate

Obfuscation	Contagio Malware Dump	George Mason University	PDFrate Community
None	86.40%	89.60%	91.00%
Malware w/parser confusion attack only	70.00%	65.80%	82.20%
Benign root file	0.70%	13.90%	13.50%
Root file w/parser confusion + reverse mimicry attacks	7.80%	2.30%	11.00%

Attack vs PDFRate

Obfuscation	Contagio Malware Dump	George Mason University	PDFrate Community
None	86.40%	89.60%	91.00%
Malware w/parser confusion attack only	70.00%	65.80%	82.20%
Benign root file	0.70%	13.90%	13.50%
Root file w/parser confusion + reverse mimicry attacks	7.80%	2.30%	11.00%

Attack vs PDFRate

Obfuscation	Contagio Malware Dump	George Mason University	PDFrate Community
None	86.40%	89.60%	91.00%
Malware w/parser confusion attack only	70.00%	65.80%	82.20%
Benign root file	0.70%	13.90%	13.50%
Root file w/parser confusion + reverse mimicry attacks	7.80%	2.30%	11.00%

Attack vs PDFRate

Obfuscation	Contagio Malware Dump	George Mason University	PDFrate Community
None	86.40%	89.60%	91.00%
Malware w/parser confusion attack only	70.00%	65.80%	82.20%
Benign root file	0.70%	13.90%	13.50%
Root file w/parser confusion + reverse mimicry attacks	7.80%	2.30%	11.00%

Attack vs PDFRate

Obfuscation	Contagio Malware Dump	George Mason University	PDFrate Community
None	86.40%	89.60%	91.00%
Malware w/parser confusion attack only	70.00%	65.80%	82.20%
Benign root file	0.70%	13.90%	13.50%
Root file w/parser confusion + reverse mimicry attacks	7.80%	2.30%	11.00%

Detection Improvement

Tool	TP	FP
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)

- PJScan can only produce 1021 extractions, compared to 1429 and 1013 for the 9.5.0 and 11.0.08 extractors
- Reference extractors can filter out malformed files, reduces noise
- Number of files given to each tool shown in parenthesis

Overhead

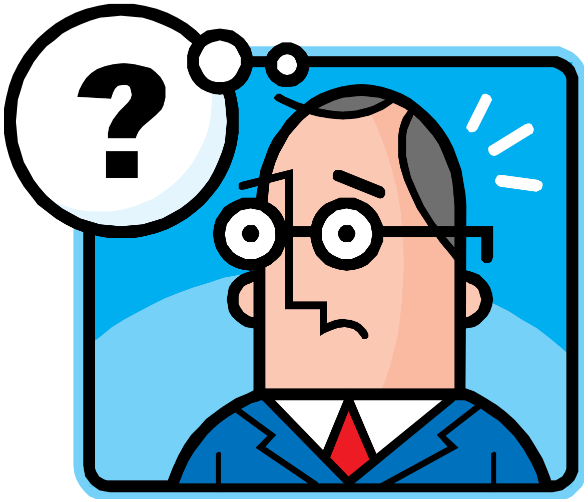
Tool	Avg. Runtime (s)
libpdfjs	0.05
jsunpack-n	0.78
Origami	1.86
Reference Extractor	3.93

- The reference extractor is slower than other tools, mostly due to VM snapshot resets
- Could be mitigated in a real world application with better virtualization technologies, the use of RAM disks and multiple VM's

Conclusion:

- Malicious PDF detection isn't solved
- Parser discrepancies are prevalent in existing detection methods
- Using parser confusion attacks we can evade all existing detection methods we evaluated
- The reference extractor mitigates these parser discrepancies and could be using in a real world application

Questions?



Signature Based

- Look for known malicious files, families, exploits, etc.
- Trivial to evade with polymorphism
- Can't detect novel threats
- Parsers are used to decode content before applying signatures

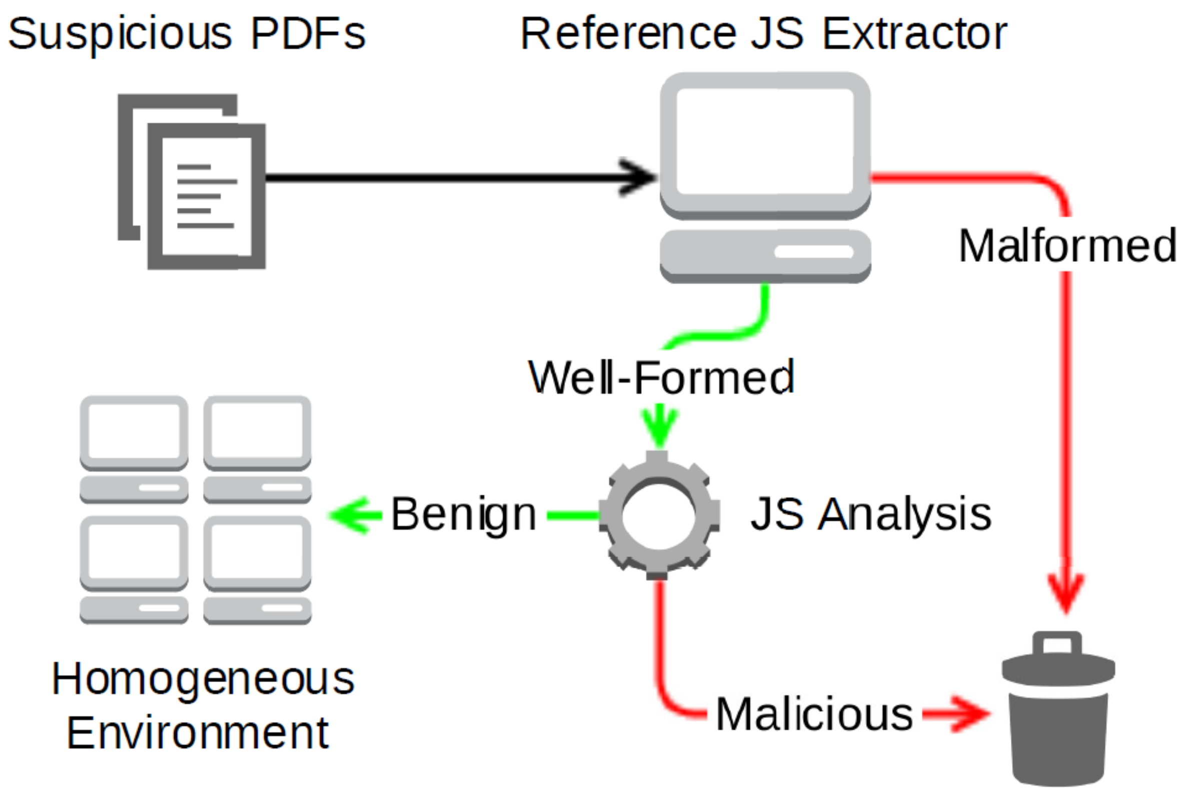
Metadata/Structure Based

- Use a PDFs metadata or structural features with a machine-learning classifier (Maiorca et al. ‘12), (Smutz and Stavrou ‘12), (Šrndić and Laskov ‘13)
- Susceptible to mimicry and reverse mimicry attacks (Šrndić and Laskov 2014), (Maiorca et al. 2013)
- Based only on similarities between malicious/benign sets
- Parsers are used to extract feature sets

JavaScript Based

- Extract/instrument and analyze embedded JavaScript (Liu et al. ‘14), (Tzermias et al. ‘11), (Laskov and Šrndić ‘11), (Lu et al. ‘13)
- Can only detect malicious PDFs which use JavaScript (almost all modern attacks)
- Parsers are used to extract/identify JavaScript
- We think the best option for detecting modern and advanced attacks

Deployment


Suspicious PDFs
Reference JS Extractor
Malformed
Well-Formed
Benign
JS Analysis
Homogeneous
Environment
Malicious