X is not enough!
Grab the PDF by the tail!

Jindrich Kubec

Jiri Sejtko

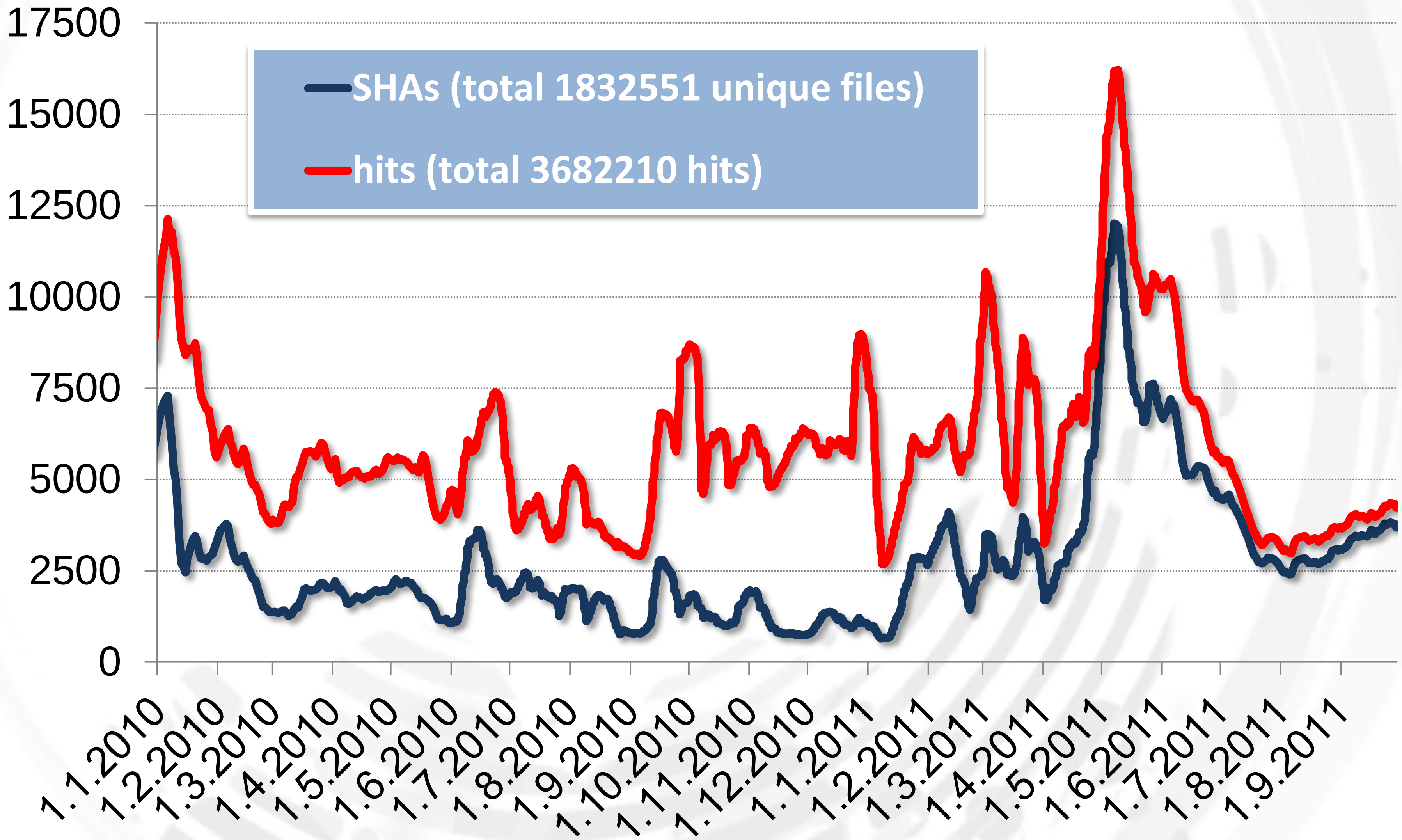
Agenda

- Introduction
- File format heuristics
- JavaScript heuristics
- Conclusion

Quick introduction to PDF

- Portable Document Format
  - Extremely liberal file format!!!
- Powerful scripting language
- Adobe Reader
  - 82% market share
  - Multiple vulnerabilities
  - Ideal channel for spreading?

Prevalence

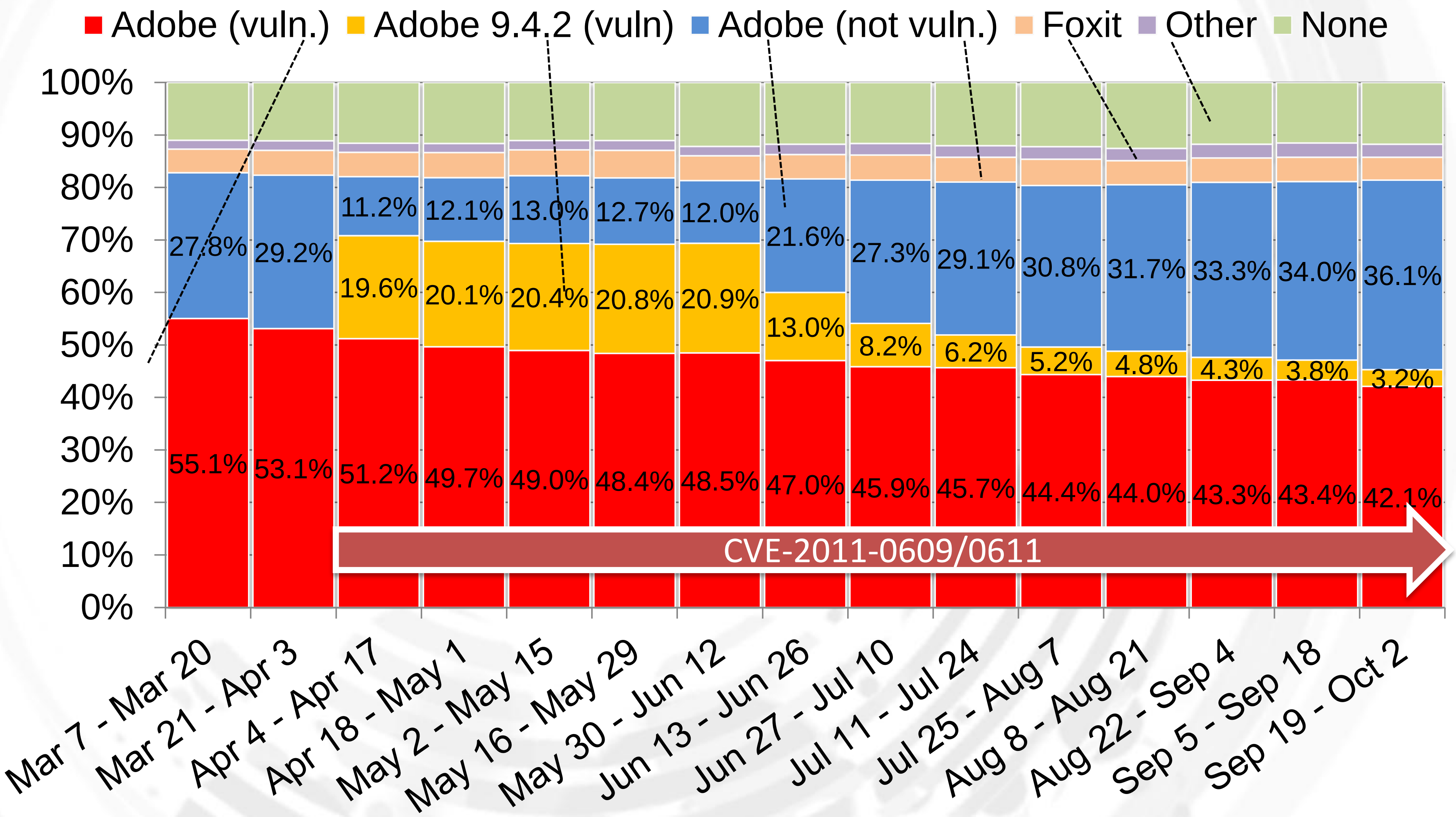

17500
15000
12500
10000
7500
5000
2500
0
SHAs (total 1832551 unique files)
hits (total 3682210 hits)
1.1.2010
1.2.2010
1.3.2010
1.4.2010
1.5.2010
1.6.2010
1.7.2010
1.8.2010
1.9.2010
1.10.2010
1.11.2010
1.12.2010
1.1.2011
1.2.2011
1.3.2011
1.4.2011
1.5.2011
1.6.2011
1.7.2011
1.8.2011
1.9.2011

Adobe Reader X

- 18. November 2010
- Sandboxing
  - Escape presented on BlackHat 2011 http://bit.ly/nP9Fw9 (already patched)
  - Solution to the buggy reader?
  - Could be for those who have ‘X’

- About 42% of all users run old & buggy versions (more than 50% of Adobe users)!
  - Let’s see ->

PDF Market share



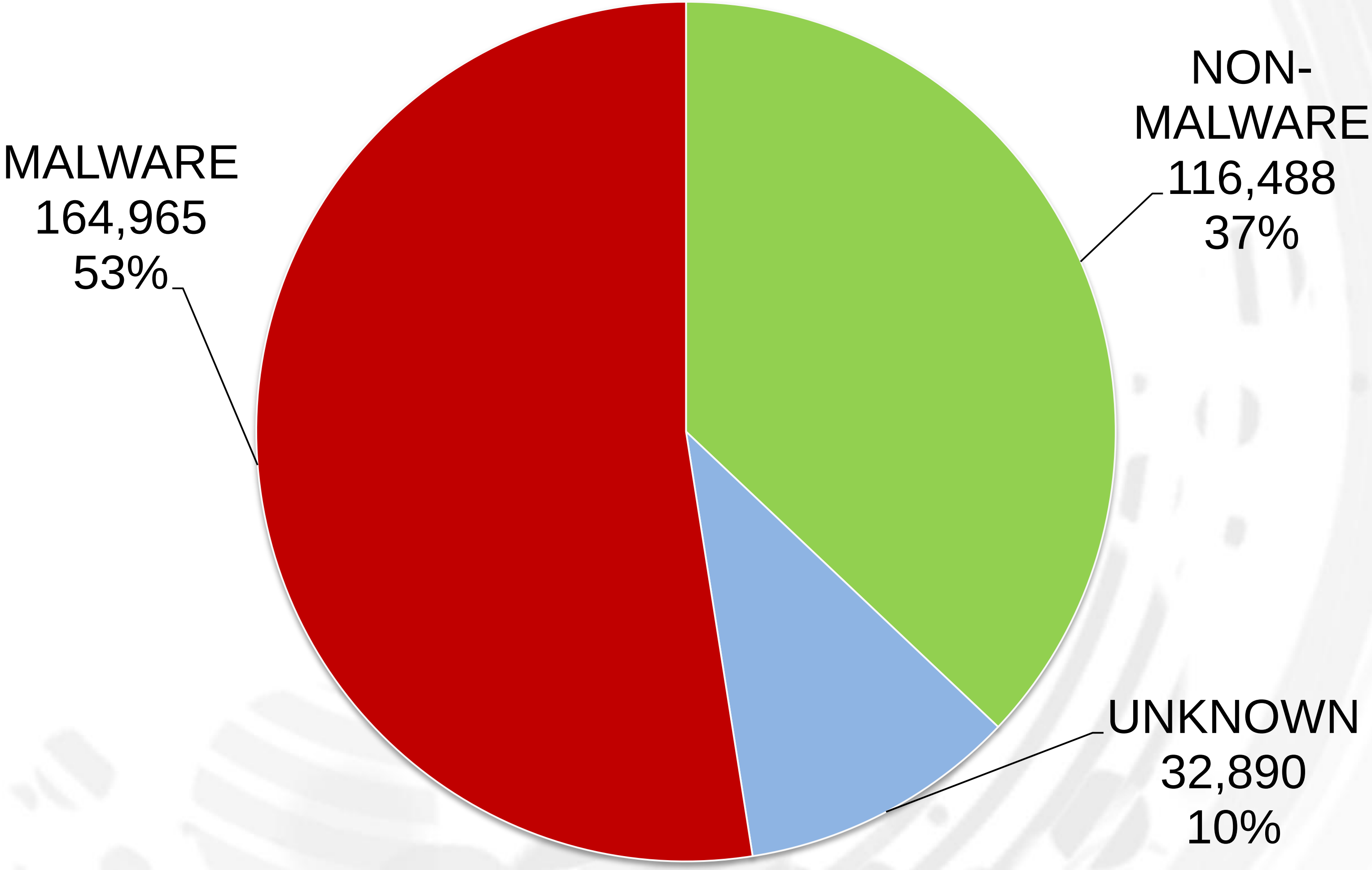
Format-based heuristics

- Challenge is to highlight obscure issues
  - But what’s really strange?
- Weak PDF specification
  - Manipulation is possible even if PDF specification says “impossible”
  - Undocumented features
- Rich PDF specification
  - Many different ways to obtain the same result

How do we parse a PDF?

- Parse in the same way as does the Reader
  - These attempts are not always successful!
- Accept only valid PDF files
  - ‘Repair’ reimplementation
- Ability to discover and highlight obscure issues
  - Collect metainfo while parsing

Our dataset



Our dataset #2

	NON-MALWARE	UNKNOWN	MALWARE
TOTAL	**116,488**	**32,890**	**164,965**
HDR SHIFT	16 (0.01%)	169 (0.51%)	1,913 (1.16%)
HDR WRONG	0 (0.00%)	70 (0.21%)	5,837 (3.54%)
1 PAGE	35,294 (30.30%)	11,246 (34.19%)	130,108 (78.87%)
1 PG. NO CONT.	245 (0.21%)	465 (1.41%)	54,376 (32.96%)
BROKEN XREF	178 (0.15%)	2,575 (7.83%)	146,501 (88.81%)
BIG DATA	1,037 (0.89%)	1,502 (4.57%)	40,156 (24.34%)

/XFA

- XFA = XML Adobe Forms
- CVE-2010-0188 (TIF)

> The value of this entry must be either a stream representing the entire contents of the XML Data Package or an array of text string and stream pairs representing the individual packets comprising the XML Data Package.
>
> PDF reference, version 1.7 – 8.6.1, page 673

/XFA Array

- MUST be an array of pairs (spec.)
  - String representing packet name
  - Stream representing packet data

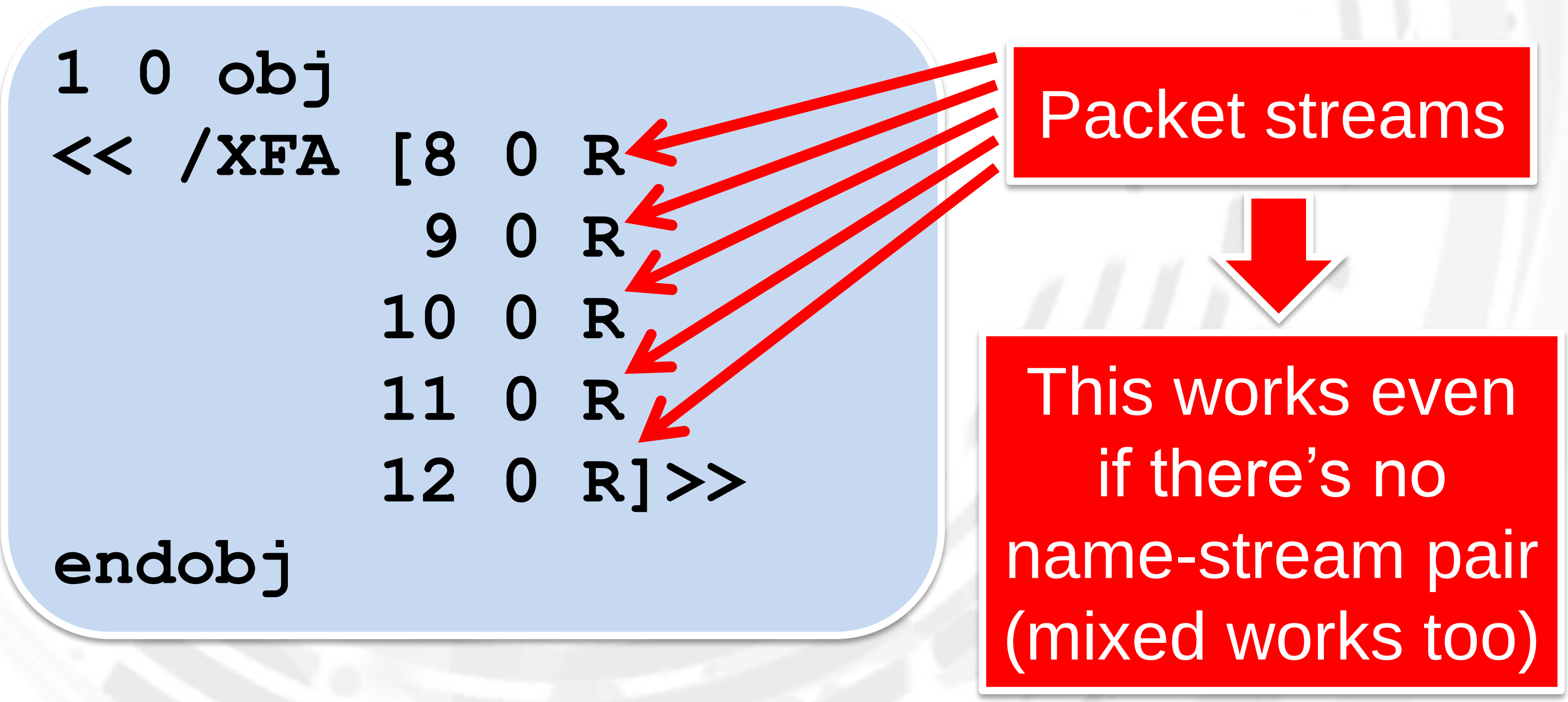
```
1 0 obj
<< /XFA [(xdp:xdp) 10 0 R
         (template) 11 0 R
         (config) 12 0 R
         (/xdp:xdp) 13 0 R
]>>
endobj
```

Packet name

Packet stream

/XFA Array #2

- MUST be an array of pairs (spec.)
  - O’rly? What about this?



```
1 0 obj
<< /XFA [8 0 R
          9 0 R
         10 0 R
         11 0 R
         12 0 R]>>
endobj
```

Packet streams

This works even if there’s no name-stream pair (mixed works too)

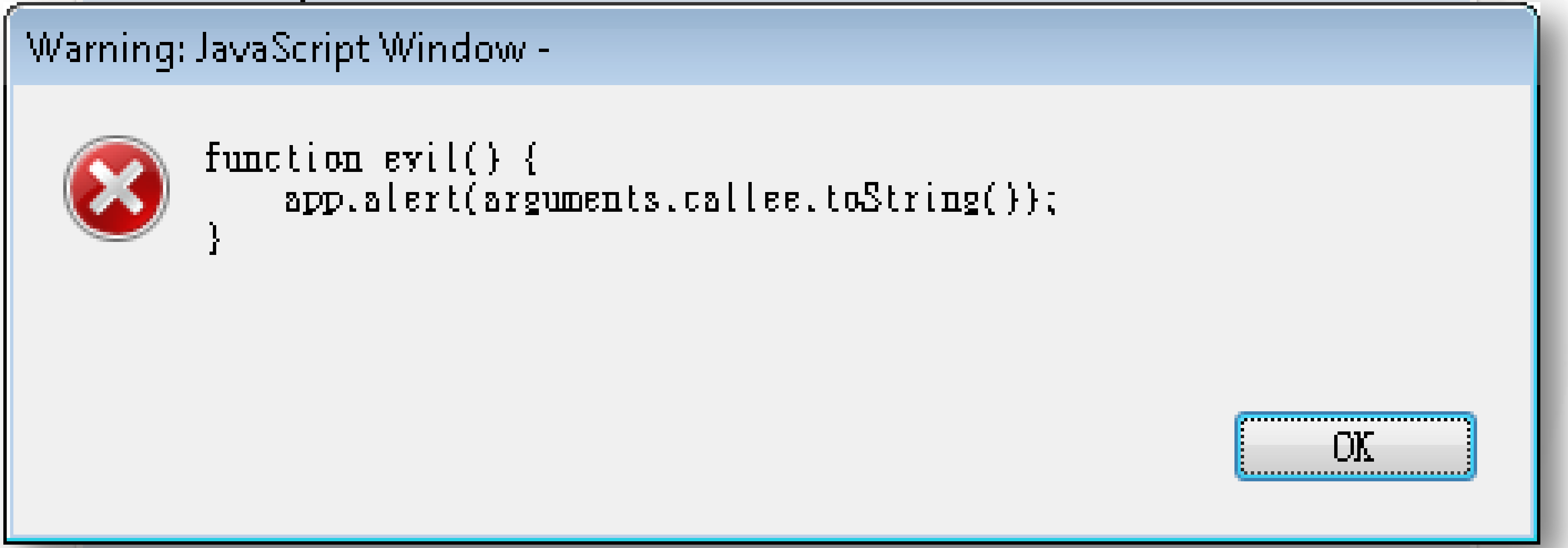
/XFA packet

- packets allow the split of logical form
- template, data, config, etc…

> Each packet represents a complete XML element, with the exception of the first and last packet, which specify the beginning and end tags for the xdp:xdp element.
>
> PDF reference, version 1.7 – 8.6.7, page 772

/XFA packet #2

```
<event activity="initialize” …>
 <script …>
```



Warning: JavaScript Window -

```
function evil() {
    app.alert(arguments.callee.toString());
}
```

```
 </script>
</event>
```

/XFA – results

- Notice result for dataset limited to PDF files containing XFA

	NON-MALWARE	UNKNOWN	MALWARE
TOTAL / XFA	**116,488 / 2,787**	**32,890 / 3,128**	**164,965 / 66,816**
no NAME	0 (0.00% / 0.00%)	8 (0.02%/0.26%)	24,311 (14.74%/36.38%)
split SCRIPT	0 (0.00% / 0.00%)	10 (0.03%/0.32%)	13,657 (8.28%/20.44%)

/Filter – stream filters

- Indicate how streams are encoded
- Evaluate only on valuable objects!
- Multiple different filters
  - /ASCIIHexDecode /FlateDecode …
- Filter repetition
  - /ASCIIHexDecode … /ASCIIHexDecode
- Unexpected filters
  - Text under JBig2Decode, JPXDecode, …

/Filter – different / duplicate

	NON-MALWARE	UNKNOWN	MALWARE
TOTAL	**116,488**	**32,890**	**164,965**
1 filter	17,298 (14.85%)	6,207 (18.87%)	78,904 (47.83%)
2 filters	0 (0.00%)	73 (0.22%)	11,615 (7.04%)
3 filters	0 (0.00%)	9 (0.03%)	979 (0.59%)
4 filters	0 (0.00%)	1 (0.00%)	175 (0.11%)
5 filters	0 (0.00%)	11 (0.03%)	405 (0.25%)

- Maximum of 5 filters encountered ITW
  - – More than 20 work! We tested them.

Dupl. filters	0 (0.00%)	5 (0.02%)	116 (0.07%)

/Filter – JBIG2Decode

- Pure image encoding algorithm
  - Monochrome (1 bit per pixel)
  - Both lossy and lossless (text encode)

> Also note that JBIG2Decode and JPXDecode are not listed in Table 4.44 because those filters can be applied only to image XObjects.
>
> PDF reference, version 1.7 – 4.8.6, page 353

/Filter – JBIG2Decode #2

- Would you expect it in text streams?
  - – We didn’t, due to the specifications!

```
200 0 obj <<
/XFA [(template) 201 0 R (dataset) 301 0 R]
>> endobj
...
201 0 obj <<
/Length 3125
/Filter [ /FlateDecode /JBIG2Decode ]
>>
stream ...
```

/Filter – Unexpected filters | parameters

- Unexpected on non-image data
  - JS, XFA, font
- Any data might be declared as an image
- Encoding needs to be lossless
  - RunLengthDecode, CCITTFaxDecode, JBig2Decode
- Parameter /Predictor in LZW and Flate
  - for TIFF and PNG images

/Filter – overview

	NON-MALWARE	UNKNOWN	MALWARE
TOTAL	**116,488**	**32,890**	**164,965**
PLAINTEXT	17,298 (14.85%)	6,221 (18.91%)	88,048 (53.37%)
DEFLATE	13,227 (11.35%)	2,748 (8.36%)	71,052 (43.07%)
ASCII85	0 (0.00%)	133 (0.40%)	15,140 (9.18%)
ASCIIHEX	0 (0.00%)	33 (0.10%)	3,138 (1.90%)
RLE	0 (0.00%)	19 (0.06%)	724 (0.44%)
LZW	0 (0.00%)	14 (0.04%)	624 (0.38%)
PREDICTOR	0 (0.00%)	0 (0.00%)	491 (0.30%)
CCITTFAX	0 (0.00%)	0 (0.00%)	92 (0.06%)
JBIG2	0 (0.00%)	0 (0.00%)	45 (0.03%)
UNHANDLED	0 (0.00%)	0 (0.00%)	0 (0.00%)

Format-based heuristics - review

- Many strange attributes
  - – Allows well-balanced heuristics

	NON-MALWARE	UNKNOWN	MALWARE
TOTAL	**116,488**	**32,890**	**164,965**
“Normal”	116,251 (99.80%)	32,043 (97.42%)	51,232 (31.06%)
Sth. “abnormal”	237 (0.20%)	847 (2.58%)	113,733 (68.94%)

PDF JavaScript

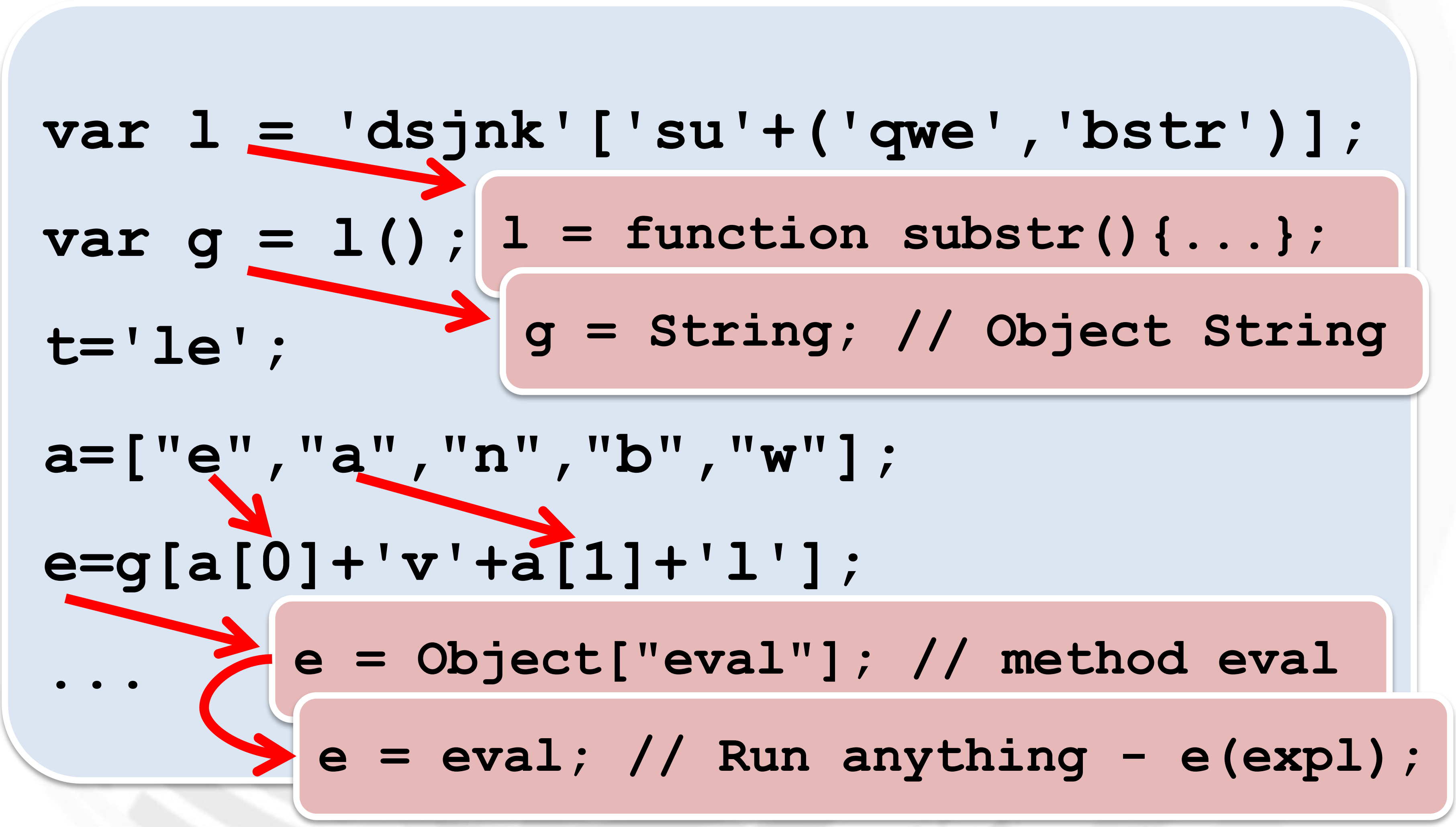
- The engine of malicious PDFs
- Powerful control
- Various usage
  - Main exploitation (printd, getIcon, ….)
  - Heap spraying
  - Obfuscation
- Our nightmare!

Our dataset – XFA/JavaScript

- Limit samples in all categories
  - Including XFA or JS

	NON-MALWARE	UNKNOWN	MALWARE
TOTAL	**116,488**	**32,890**	**164,965**
JS	30,504 (26.19%)	8,754 (26.62%)	102,271 (62.00%)
XFA	2,787 (2.39%)	3,128 (9.51%)	66,816 (40.50%)
JS or XFA	30,525 (26.20%)	9,049 (27.51%)	163,130 (98.89%)

PDF JS use in malware



```
var l = 'dsjnk'['su'+('qwe','bstr')];
var g = l();
t='le';
a=["e","a","n","b","w"];
e=g[a[0]+'v'+a[1]+'l'];
...
```

```
l = function substr(){...};
```

```
g = String; // Object String
```

```
e = Object["eval"]; // method eval
```

```
e = eval; // Run anything - e(expl);
```

PDF JavaScript – Light Side

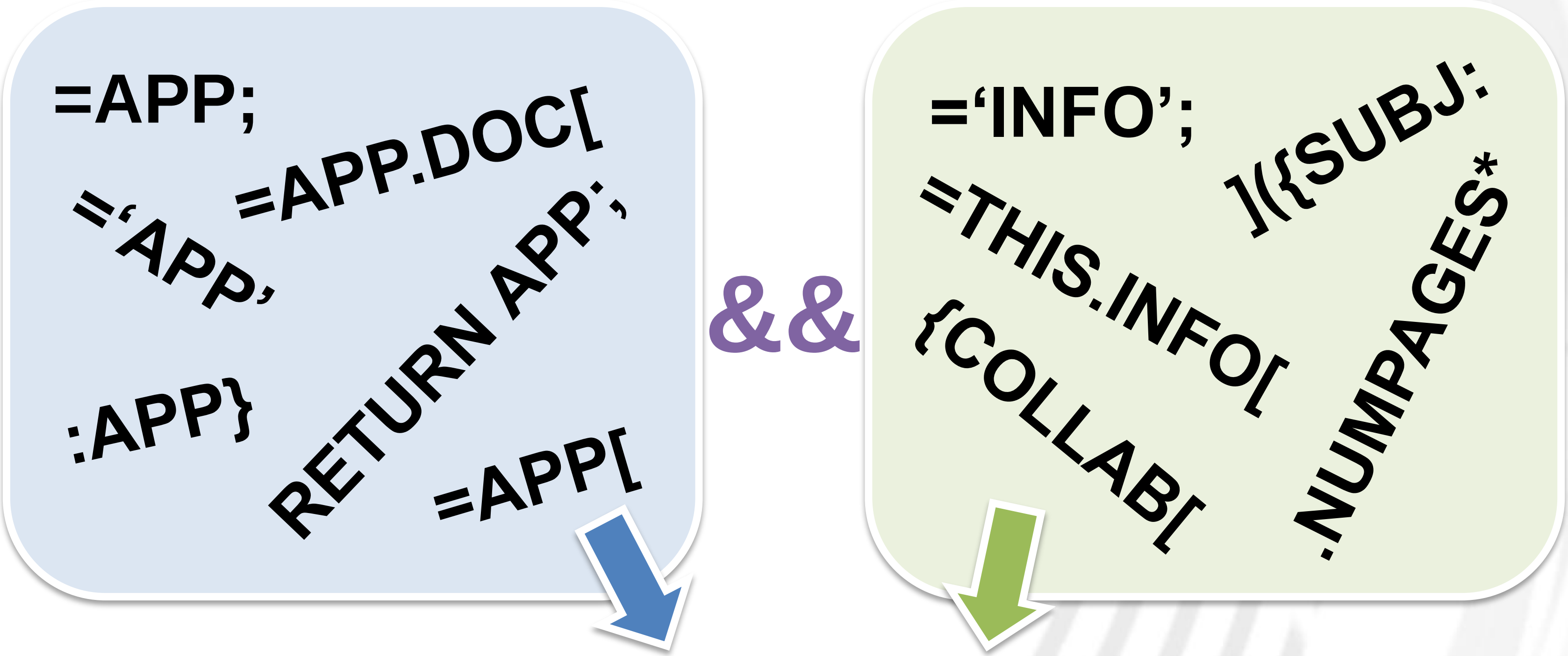
- Non-malware JS in PDF is conservative
  - Usually clean & readable code
  - Low use of public obfuscators
  - Nearly no custom obfuscations

- Allows us to be strict!
  - Penalize everything abnormal

PDF JS – Group rules

- Groups targeting abnormality
  - Minimal script patterns (strings, regexp, ...)
  - Group made of many patterns (based on similarity)
  - Sum ‘chunk hits’ inside groups
- Simple detection rules between groups
  - **A** [&& **B** [&& **C** […]]] **(rule: A && B)**
  - Negation of group **(rule: A && !B)**
  - <,<=,==,>=,> **(rule: A && !B && sum C > 4)**

PDF JS – Group rules #2



=APP;

=APP.DOC[

=‘APP’

RETURN APP;

:APP}

=APP[

&&

=‘INFO’;

]({SUBJ:

=THIS.INFO[

{COLLAB[

.NUMPAGES*

SET	SAMPLES	group APP	group DOM	APP && DOM
NON-MAL	116,488	1 (0.00%)	869 (0.75%)	0 (0.00%)
UNKNOWN	32,890	80 (0.24%)	540 (1.64%)	0 (0.00%)
MALWARE	164,965	28,685 (17.39%)	34,184 (20.72%)	9,456 (5.73%)

PDF JS – Group rules #3

- 88 groups
- 137 detection rules (13 submission)
- 2,700 patterns
- >1,800,000 combinations

SET	ALL / JS or XFA	DETECTION
NON-MALWARE	116,488 / 30,525	0 (0.00% / 0.00%)
UNKNOWN	32,890 / 9,049	0 (0.00% / 0.00%)
MALWARE	164,965 / 163,130	147,941 (89.68% / 90.69%)

- The rest is detected using other methods

QA against QA – quick review

- Most Malware authors have QA
  - Processes to avoid AV detections (at least)
- Avoiding basic detections is easy
  - Services similar to virustotal.com,…
- How about detections they don't see!?!
  - Based on additional triggers (source, structure, data flow, …)
  - Avoiding is not so easy ;-)

Conclusion

- PDF format is very complex, easy to misuse
- Reader is very widespread and not upgraded consistently
- Structural heuristics work, but only on a limited subset of files
- JavaScript heuristics rules have almost complete coverage
- But the bad guys aren’t sleeping, so combining techniques is the way to go.

Thank you, now wake up!

- Any Questions?

- Contacts
  - – Jindrich Kubec (kubec@avast.com)
  - – Jiri Sejtko (sejtko@avast.com)
  - – http://blog.avast.com