Detecting Malicious JavaScript in PDFs

Using Conservative Abstract Interpretation

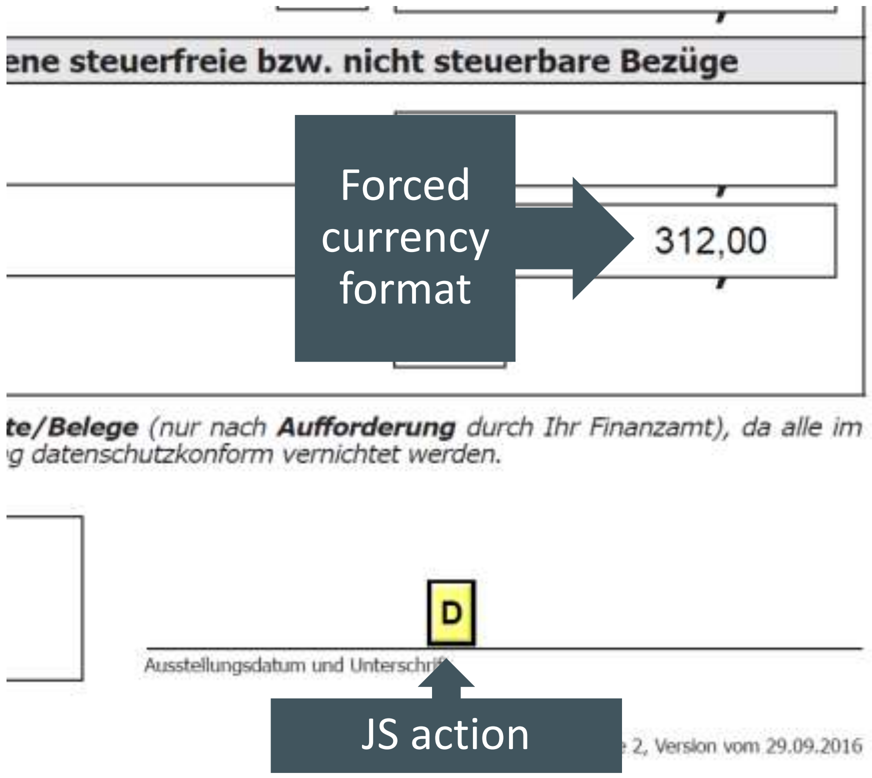
Alexander Jordan
Oracle Labs Australia

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Motivation

"What is JavaScript doing in my PDFs?"


ene steuerfreie bzw. nicht steuerbare Bezüge

312,00

te/Belege (nur nach **Aufforderung** durch Ihr Finanzamt), da alle im
g datenschutzkonform vernichtet werden.

Ausstellungsdatum und Unterschri

2, Version vom 29.09.2016

- Form validation/completion
- Automation, utility functions
  - email
  - printing
- And more…
  - control embedded media
  - connect to a SQL database

Exploit Examples

CVE Id	Vulnerable Target Application	Exploit
2003-0284	Adobe Acrobat 5	Unchecked access to file-system
2007-5659	Adobe Reader, Acrobat 8.1.1	Buffer overflow (arbitrary code execution)
2008-0667	Adobe Reader, Acrobat 8.1.1	Denial-of-service (printing API)
2008-0726	Adobe Reader, Acrobat < 8.1.2	Integer overflow (arbitrary code exection)
2008-1104	Foxit Reader < 2.3	Buffer overflow (arbitrary code execution)

Current Defenses

Static

- Malware detection based on
  - signature
  - patterns
  - structure
  - machine learning

Dynamic

- Malware detection during sandboxed execution
- Runtime protection (requires modified reader application)

Current Defenses

Static

- Malware detection based on
  - signature
  - patterns
  - structure
  - machine learning

Defeated by obfuscation!

Dynamic

- Malware detection during sandboxed execution
- Runtime protection (requires modified reader application)

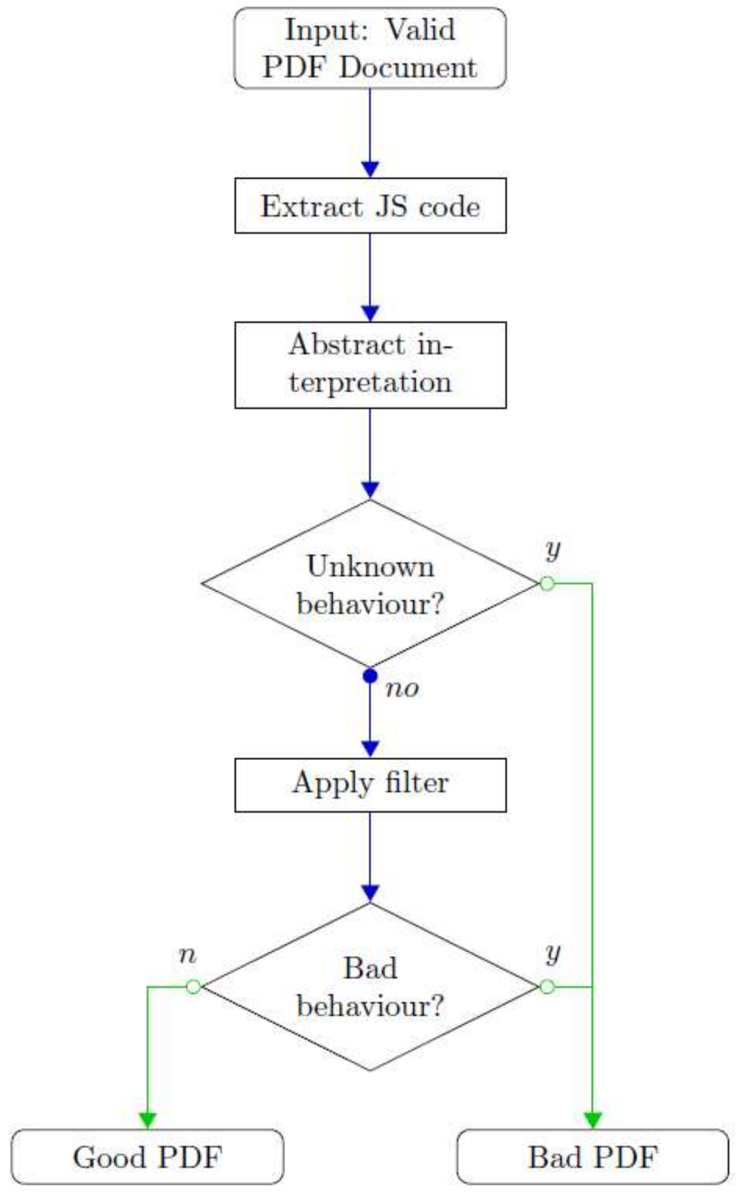
How to execute all paths?

Best defense?

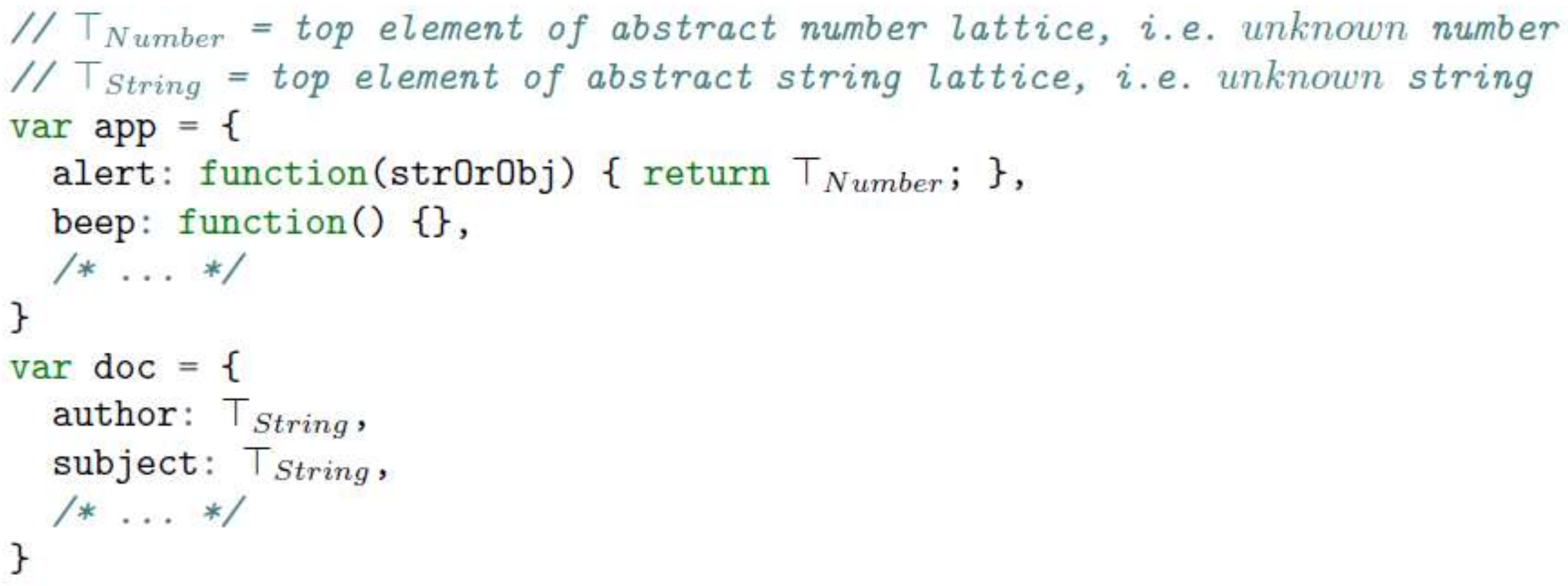
Strip all JavaScript from PDF documents.

Detection based on *conservative* abstract interpretation

- The idea
  - extracted JS code is analyzed with a model of Acrobat's JS API
- *Conservative?*
  - the model is conservative
  - unknown behavior is handled conservatively



PDF-JS Model



```
// $\top_{Number}$ = top element of abstract number lattice, i.e. unknown number
// $\top_{String}$ = top element of abstract string lattice, i.e. unknown string
var app = {
  alert: function(strOrObj) { return $\top_{Number}$; },
  beep: function() {},
  /* ... */
}
var doc = {
  author: $\top_{String}$,
  subject: $\top_{String}$,
  /* ... */
}
```

Prototype Implementation

Abstract Interpretation based on *SAFE*[1]

+

Initial PDF-JS Model (120 LoC)

1) https://github.com/sukyoung/safe

Results

Oracle-Internal Benchmark Set

Malicious

- 1323 documents
- All rejected as malicious

Non-Malicious

- 60 documents
- 22 rejected as malicious (*false positives)*
- But all 22 use unsafe JavaScript APIs

ORACLE

Current Status

- ✓ Patent filed
- ❏ Paper to be submitted in July

Takeaway

Why does abstract interpretation work in this context?

1. Non-malicious JavaScript in PDF is *easy* to analyze.
2. If something is *too hard*, it is okay to reject it.
3. We can gradually refine the PDF-JS model as long as it remains conservative.

Safe Harbor Statement

Integrated Cloud

Applications & Platform Services