PhoneyPDF
A virtual PDF analysis framework

Kiran A. Bandla

Feb 2014

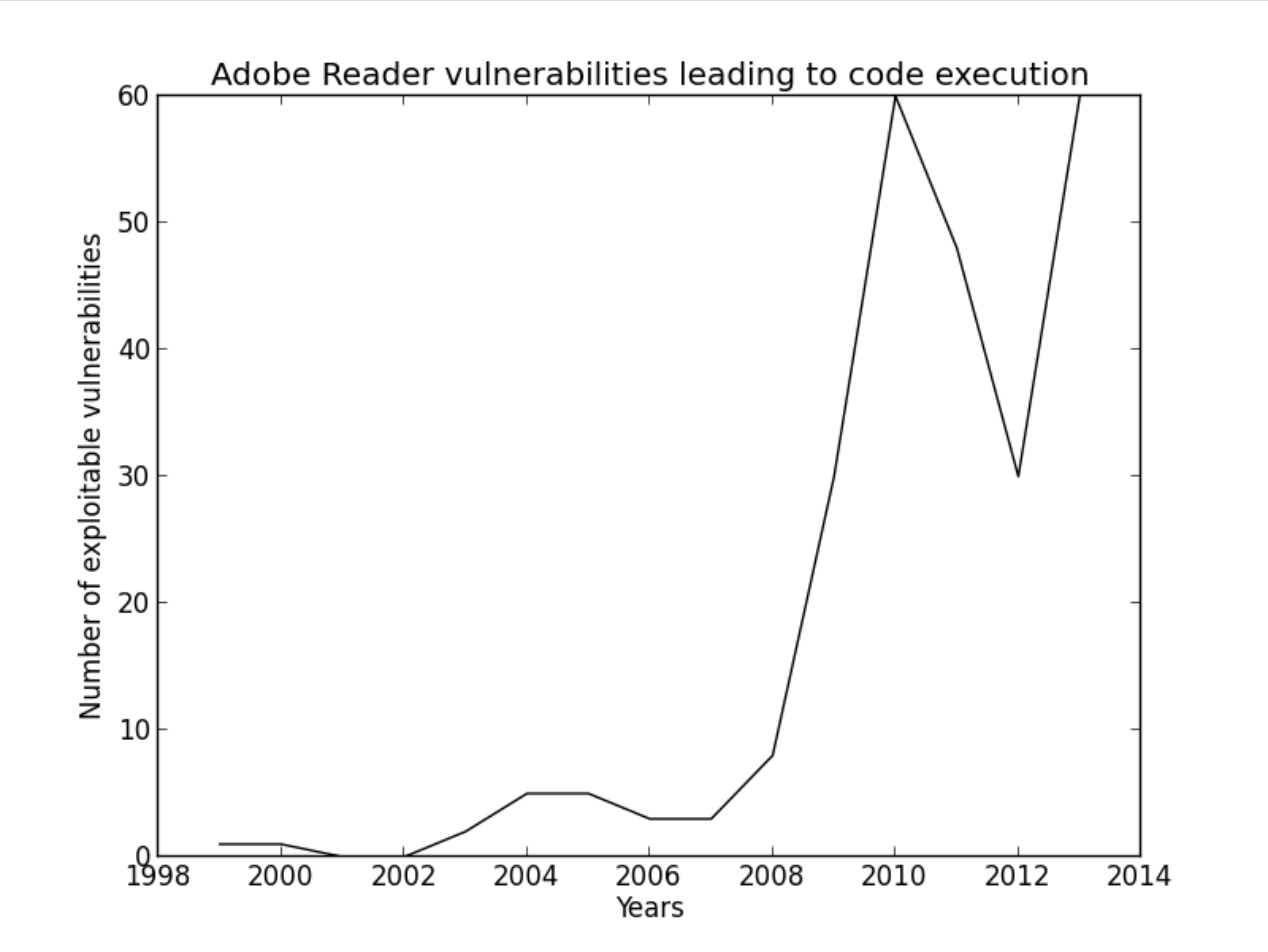
Lightning version

- Virtual PDF honeyclient framework
- Its `dynamic`
- Python + OS X|Linux
- JavaScript [PyV8]
- Adobe DOM, Adobe XFA [lxml]
- Yara signatures
- ML Ready
- Standalone tool or Library

Adobe Reader

- Version 9
- Specification
- Based on SpiderMonkey
  - Used to fingerprint
- Implementation differences across versions

Code Exec Vulns in Reader

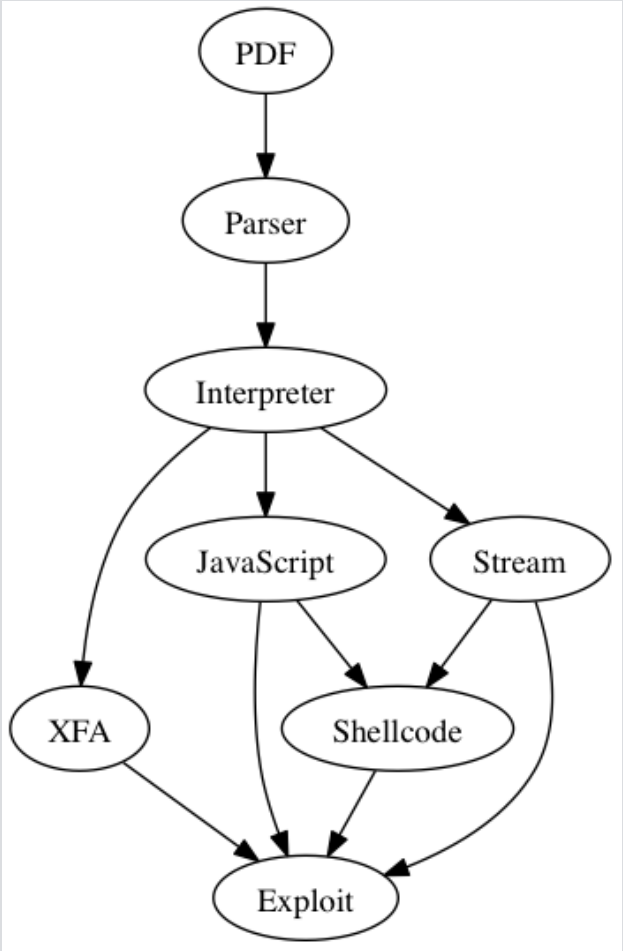


PhoneyPDF

Previous Work

- Mostly static
  - pdf-parser
  - peepdf
  - pdf x-ray, etc
- Lectures
  - omg-wtf-pdf
  - Malicious PDF analysis, etc
- A few published papers
- PhoneyPDF is `dynamic`

Design



PhoneyPDF

The Parser

- Based on Didier's pdf-parser
- Character-based state machine
- Heavily modified, improved
- Pythonic representation
  - id, md5, stream, dict, etc
- De-obfuscation
- Filters
- Features extraction for Machine Learning

Parsing is not fun

PhoneyPDF

Example #1: Raw to Python

```
trailer
<</Size 22 /Root 23 0 R>>
```

```
>> dir(pdfObj)
['content', 'dict', 'id', 'type']

>> pdfObj
<PDFElement id:None type:PDF_ELEMENT_TRAILER>

>> pdfObj.dict
{'/Size': 22.0, '/Root': <PDFIndObjRef id=23 gen=0>}
```

PhoneyPDF

Example #2

RAW dict Object

```
<<
    /DA (/CourierStd 10 Tf 0 g)
    /F 4
    /FT/Btn
    /Ff 65536
    /MK <</TP 1>>
    /P 1 0 R
    /Parent 19 0 R
    /Rect [12.5 5.1 18.3 7.7]
    /Subtype /Widget
    /T (favwwbw[0])
    /TU (favwwbw)
    /Type/Annot
>>
```

Pythonic equivalent

```
{
 '/DA'   : '/CourierStd 10 Tf 0 g',
 '/F'    : 4.0,
 '/FT'   : '/Btn',
 '/Ff'   : 65536.0,
 '/MK'   : {'/TP': 1.0},
 '/P'    : <PDFIndObjRef id=1 gen=0>,
 '/Parent': <PDFIndObjRef id=19 gen=0>,
 '/Rect': [12.5, 5.1, 18.3, 7.7],
 '/Subtype': '/Widget',
 '/T'    : 'favwwbw[0]',
 '/TU'   : 'favwwbw',
 '/Type': '/Annot'
}
```

PhoneyPDF

The Analysis Engine

- Consumes parsed objects
- Analysis logic
- 'Renders' the PDF
  - Walk objects
  - JavaScript, Adobe DOM Emulation
  - Adobe XFA
- Yara Signatures
- Revisit this in a few slides

JavaScript

- PyV8
- One context
- Debug mode
- Adobe DOM Emulation!
- Heapspray detection
- Yara

JavaScript #2

- Bootstrap
  - SpiderMonkey hacks
  - `this` object
  - Ex: Object.prototype.eval = this.eval;
  - return -1 for some functions on undefined objects
- Hooks
  - eval, timeout functions, etc
- Emulate Adobe DOM

Adobe DOM Emulation

- Console object for logging
- Collab,FullScreen,Thermometer,util,app,doc..
- Object hierarchy; initialization order
  - app object needs FullScreen, Thermometer, etc
- Generic handlers – log
- Custom handlers – detection
  - spell.customDictionaryOpen : CVE-2009-1493
- Argument introspection
- PDF to JS bridge

Adobe XML Forms Architecture / XFA

- 2 method for forms in PDF
  - AcroForms
  - Adobe XML Forms Architecture (XFA) forms
- lxml
- JavaScript in XFA objects
- XML to XFA DOM
- JavaScript bridge

‘Render’ the PDF

- Based on the PDF Specification document
- Stay close to Adobe’s implementation
- Start emulation
  - Create JavaScript context
  - Walk PDF objects, avoiding cycles
  - Dynamic handlers based on object type
  - Monitored fields from XFA and JavaScript

Handlers

- 6 types; based on object type
  - Comment, XREF, Trailer, StartXREF, Indirect, Malformed
- Walk object list based on object's `/Type`
- About 35 types; extensible
- Object relations

/Action type handler

```
# Specific handlers for various types
if pdfElement.dict[KEY_TYPE] == KEY_ACTION:
    #/Action object
    keys = pdfElement.dict.keys()
    if KEY_S in keys:
        if pdfElement.dict[KEY_S] == KEY_JAVASCRIPT:
            js = None
            # Handle JS here
            # This has to be either 1) text string or 2) Stream
            if hasattr(pdfElement, 'stream'):
                # Stream!
                #TODO: JS is served. Pick it up
                js = pdfElement.stream
            elif isinstance( pdfElement.dict[KEY_JS], str):
                js = pdfElement.dict[KEY_JS]
                #TODO: JS is served. Pick it up
            elif isinstance( pdfElement.dict[KEY_JS], PDFIndObjRef ):
                js = self.__getIndirectObjectJavaScript( pdfElement.dict[KEY_JS] )
                #TODO: JS is served. Pick it up
            self.logger.debug("[%s ID:%s] Executing Javascript" %(pdfElement.dict[KEY_TYPE], pdfE
            if js:
                self.featureCollection.incFeatureValue("F_C_JS_SCRIPTS_FOUND")
                self.__executeJavaScript( js )
        elif pdfElement.dict[KEY_S] == KEY_LAUNCH:
```

PhoneyPDF

A few /Types

KEY_ROOT = '/Root'
KEY_CATALOG = '/Catalog'
KEY_ACTION = '/Action'
KEY_OPENACTION = '/OpenAction'
KEY_AA = '/AA'
KEY_JAVASCRIPT = '/JavaScript'
KEY_RENDITION = '/Rendition'
KEY_OUTLINES = '/Outlines'
KEY_JS = '/JS'
KEY_PAGES = '/Pages'
KEY_PAGE = '/Page'
KEY_METADATA = '/Metadata'
KEY_SUBTYPE = '/Subtype'

PhoneyPDF

CVE-2010-0188

Adobe Reader TIFF vulnerability

- Most popular in Exploit Kits, still
- Complex flow to trigger the vulnerability
- PhoneyPDF detects it; extracts payload
- Exploit Kits using this vulnerability
  - Blackhole
  - Whitehole
  - Fiesta/Neisploit
  - Cool Styxy
  - Many more!

Open Source

- github.com/verisign/phoneypdf
- 3-BSD
- Dependencies
  - PyV8
    - V8
  - lxml
  - Yara
- Selective loading

Questions