JavaScript
Ein Crashkurs

Mario Heiderich

Worum geht's?

- Ein wenig Geschichte
- JavaScript und das DOM
- Datentypen in JavaScript
- Versteckte Properties
- Wunderliche Methoden
- Tricks
- Obfuscated Code

Es war einmal...

- JavaScript – erstmals implementiert um 1995
- Entwickelt von Netscape
  - LiveScript
- Später kopiert von Microsoft
  - JScript
- Standardisiert vom ECMA
- Aktuelle Revisionen sind 1.6 – 1.8.1
- Konstanter Wandel

JavaScript und das DOM

- Das DOM (Document Object Model)
- Repräsentation der Seiteneigenschaften und Methoden
- Objekt-hierarchische Struktur
- Oft als reine Repräsentation des Dokumenten-Markups missverstanden
- Das DOM ist jedoch weit mehr

Klebstoff

- Das DOM kann als Klebstoff zwischen Markup und JavaScript verstanden werden
- Methoden zum Lesen, Durchreisen und Verändern der Markup-Struktur
- Einige standardisiert – einige proprietär oder vorimplementiert
- Selbiges gilt für die Properties
- Einige sind public – andere nur für das Chrome sichtbar

Das Chrome?

- Als Chrome bezeichen wir die Browserumgebung
- Firefox Extensions nutzen chrome:// URIs
- Das Chrome stellt verschiedenste Methoden und Eigenschaften bereit
- Unter anderem Ressourcen wie view-source: URIs, moz-icon:, resource:, res://, etc.
- Jeder Browser hat sein Chrome – mancheiner heisst sogar so...

Privilegien

- Das Chrome darf auf die History zugreifen – höher gelegene Schichten nicht
- Mit Gecko-basierten Browsern kann man per Chrome auch beliebig Dateien lesen und schreiben
- Änderungen an der Browserkonfiguration
- JavaScript im Kontext einer Website ist eingeschränkt
- Daten schreiben nur über DOM Storage Methoden und LSOs

Kleines Beispiel

```
<script>
function runFile(f) {
     var file = Components.classes["@mozilla.org/file/local;1"]
          .createInstance(Components.interfaces.nsILocalFile);
      file.initWithPath(f);
     file.launch();
}
runFile('c:\\WINDOWS\\system32\\sndrec32.exe');

function writeFile(filename, ext) {

     var data =
atob('TVqQAAMAAAAEAAAA//8AALgAAAAAAAAA...AYQBuAHMAbABhAHQAaQBvAG4AAAAAAAAkEsAQAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=');

     var file = Components.classes["@mozilla.org/file/local;1"]
          .createInstance(Components.interfaces.nsILocalFile);
     var stream = Components.classes["@mozilla.org/network/file-output-stream;1"]
            .createInstance(Components.interfaces.nsIFileOutputStream);

     file.initWithPath(filename + '.' + ext);
     file.create(Components.interfaces.nsIFile.NORMAL_FILE_TYPE, 0777);

     stream.init(file, 0x02 | 0x08 | 0x20, 0777, null);
     stream.write(data, data.length);
     stream.close();

     file.launch();
}
writeFile('c:\\test', 'exe');
</script>
```

Ein Blick ins DOM

- FF und Firebug helfen weiter
- Firebug hat erweiterte Rechte zum Lesen geschützter Properties
- Ausgabe von `console.dir(window)`
- Viele interessante Properties nicht im Kontext der Webseite nutzbar
  - `Components nsXPCComponents`
  - `controllers XULControllers`
  - `history[1 … n]`

Zurück zum JavaScript

- JavaScript läuft nicht nur im Browser
- Verschiedene Runtimes stehen zur Verfügung
  - SpiderMonkey
  - ActionMonkey
  - Tamarin
  - SJS
  - Etc.
- DOM Emulatoren können genutzt werden
- Soft-DOMS oder fake DOMs

Datentypen

- Alles ist ein Objekt
- Aber kein Object – Object ist auch ein Objekt
- Auch Integers sind Objekte und haben Kindeigenschaften
- Manche sind unsichbar
  - `1['constructor'] //Number()`
  - `'1'.constructor //String()`
- Objekt-Typen sind ebenfalls Properties
- Die wiederum Objekte sind.. Rekursion!

Methoden

- Keine Funktionen in JavaScript
- Nur Methoden – selbst wenn die Methoden *Function* heissen
- Und auch Methoden sind wieder Objekte
  - `typeof alert // function`
  - `alert.constructor // Function()`
- Und können auf verschiedene Weisen gerufen werden
- `Object.method(pm)` - or `Object['method'](pm)`
- `['alert(1)',1].sort(eval)`

Moment!

- Jedes Objekt ist auch ein Array?
- Prinzipiell schon
- Und über die Array-Indizes lässt sich noch mehr entdecken
  - `'123456'[1] //2`
  - `'123456'[-1] //6`

Noch nicht verwirrt?

- Einige Objekte können über ihren Konstruktor instanziiert werden
- Zum Beispiel
  - `new Array()`
  - `new RegExp()`
  - `new Date()`
  - `new Math()`
  - etc.

Andere wiederum...

- ...akzeptieren Abkürzungen und Shorthands
- So zum Beispiel
  - `'' // === new String()`
  - `[] // === new Array()`
  - `{} // === new Object()`
  - `/./ //=== new RegExp()`
- Interessant wirds num beim Up- und Downcast
  - `''+{} // “[object Object]”`

Operatoren und Objekte

- Einige Beispiele
  - `{} // create a new object`
  - `{}+'' // transform it to be a string`
  - `~({}+'') // becaomes an integer again`
- Leere Arrays und anderes
  - `1+2+3+-+-4 // 10`
  - `1+2+3+[]+4 // “64”`
  - `1+2+~new Image()+3+4 // 9`
- JavaScript und Obfuscation = Pech und Schwefel

Morphender Code

- Nicht sonderlich angenehmes Beispiel für AVs
- JavaScript generiert morphendes JavaScript
- Kaum zu entschlüsseln - oder?

```
<script>
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)+'))')

//http://pastebin.com/f3546211c
</script>
```

Doch

- Dank `toSource()`
- Ein kleines Beispiel:
  - https://www.2checkout.com/static/checkout/javas

Mehr verrücktes JavaScript

- JavaScript Bild-Caching – mit Code Execution
  - `Option().innerHTML='<img src=x onerror=alert(1)>'`
- Mehr versteckte Properties
  - `Date[-5] // “Date”`
- Leaking DOM Objects
  - `(1,[].reverse)() // window object`
  - `[].sort.call() // window object`

Maxium Obfuscation

- Kürzester no-alnum alert(1)
- No-alnum?
  - `([Ç,µ,,É,,Á,È,ª,,,Ó]=[$=!''']+!$+{},[]`
    `[ª+Ó+µ+Ç])()[Á+È+É+µ+Ç](+$)`
- Bislang kürzester window-getter
  - `[ö='__'][õ=/^/[Ô=[Ó=![Ò={}+ö]+Ò][Õ=!!Ò+ö]`
    `+Ò,Ò[Ø=-~[Ö=-~Ò],ø=Ø*Ø+Ö]+Ò[Ö]+Ô[Ö]`
    `+Ó[ò=Ö+Ø]+Õ[ó=-!Ö]+Õ[Ö]+Ô[ó]+Ò[ø]+Õ[ó]`
    `+Ò[Ö]+Õ[Ö]]+ö,ö+õ[ø*ò-Ö]+Ó[Ö]+Õ[Ö]+Õ[ò]`
    `+Ô[Ö]+Õ[ó]+ö]`

Zurück zur Normalität

- Methoden aufrufen
- Objekt-Methoden aufrufen
  - `Object.method(pm1, pm2)`
  - `Object['method'](pm1, pm2)`
  - `window.alert(1)`
  - `String.fromCharCode(34)`
- Methoden im Scope aufrufen
  - `window.alert(1) === alert(1)`

Call und Apply

- Manche Methoden können indirekt gerufen werden
- Über andere Methoden
- Entweder als Callback oder via `call()` oder `apply()`
  - `[1].map(alert)`
  - `[1,2].sort.call([3,4])`
  - `[1,2].reverse.apply([3,4])`

Traversal

- Object Traversal in JavaScript
- Punkt-Operator oder Array Notation
  - `window.document.cookie`
  - `alert(document.cookie)`
  - `alert(document['cookie'])`
- Scope Chains
  - `with(window)alert(document.cookie)`
  - `with(document)write(cookie)`

Wichtige DOM Objekte

- Neben window gibt es noch viel weiteres von Interesse
  - document
  - links
  - forms
  - frames
  - etc.
- Firebug hilft mehr zu entdecken

DOM Objekte überschreiben

- Die meisten publiken Methoden und Objekte können einfach überschreiben werden
- `alert=void` – best XSS protection :)
- Doch auch hier lauern Fallen
- `crypto.alert(1)`
- Und warum nicht gleich ein neues DOM bauen wenn das alte präpariert ist?
- `watch(), __defineSetter()__` etc.

Nun zu etwas ganz anderem

- Ein harmlos aussehendes Objekt
- It's just RegExp, mom!
- Dunkle Geheimnisse
- Global information leakage
- Ich weiss was Du letzten Sommer *gematcht* hast!

Kleine Demo

- RegExp Zeitmaschine

```
'foobar'.match(/foo/)


...
a=1;
b=2;
alert('hello')



console.dir(RegExp)
```

Closures

- Globaler Scope versus Function Scope
  - `a = 1`
  - `var a = 1`
  - `(function(){ var a=1})()`
  - Scope bleeding

Prototypen

- Klingen kompliziert
- Sind aber ganz einfach zu verstehen
- Bestehende Objekte zur Laufzeit erweitern
  - `String.prototype.a = function(b) {alert(b)}`
  - `c = New String()`
  - `c.a(1)`

Kurze Zusammenfassung

- JavaScript ist leicht zu erlernen
- Aber enorm schwer im Ganzen zu begreifen
- Vergleich Schatztruhe
- Als Web-Hacker muss man JavaScript atmen
- Die Sprache ist verrückt, kaputt und voller Überraschungen

Pause