

Introduction to JavaScript

Vitaly Shmatikov

What's a Scripting Language?

- ◆ Language used to write programs that compute inputs to another language processor
 - One language embedded in another
 - Embedded JavaScript computes HTML input to the browser
 - Shell scripts compute commands executed by the shell
- ◆ Common characteristics of scripting languages
 - String processing – since commands often strings
 - Simple program structure, define things “on the fly”
 - Flexibility preferred over efficiency, safety
 - Is lack of safety a good thing? (Example: JavaScript used for Web applications...)

Why JavaScript?

◆ “Active” web pages

◆ Web 2.0

- AJAX, huge number of Web-based applications

◆ Some interesting and unusual features

- First-class functions - interesting
- Objects without classes - slightly unusual
- Powerful modification capabilities - very unusual
 - Add new method to object, redefine prototype, ...

◆ Many security and correctness issues

◆ “The world’s most misunderstood prog. language”

JavaScript History

- ◆ Developed by Brendan Eich at Netscape
 - Scripting language for Navigator 2
- ◆ Later standardized for browser compatibility
 - ECMAScript Edition 3 (aka JavaScript 1.5)
- ◆ Related to Java in name only
 - “JavaScript is to Java as carpet is to car”
 - Name was part of a marketing deal
- ◆ Various implementations available
 - SpiderMonkey C implementation (from Mozilla)
 - Rhino Java implementation (also from Mozilla)

Motivation for JavaScript



◆ Netscape, 1995

- > 90% browser market share
 - “I hacked the JS prototype in ~1 week in May and it showed! Mistakes were frozen early. Rest of year spent embedding in browser”
-- Brendan Eich, ICFP talk, 2006

◆ Design goals

- Make it easy to copy/paste snippets of code
- Tolerate “minor” errors (missing semicolons)
- Simplified onclick, onmousedown, etc., event handling
- Pick a few hard-working, powerful primitives
 - First-class functions, objects everywhere, prototype-based
- Leave all else out!

Common Uses of JavaScript

- ◆ Form validation
- ◆ Page embellishments and special effects
- ◆ Navigation systems
- ◆ Basic math calculations
- ◆ Dynamic content manipulation
- ◆ Sample applications
 - Dashboard widgets in Mac OS X, Google Maps, Philips universal remotes, Writely word processor, hundreds of others...

Example 1: Add Two Numbers

```
<html>
  ...
  <p> ... </p>
  <script>
    var num1, num2, sum
    num1 = prompt("Enter first number")
    num2 = prompt("Enter second number")
    sum = parseInt(num1) + parseInt(num2)
    alert("Sum = " + sum)
  </script>
  ...
</html>
```

Example 2: Browser Events

```
<script type="text/JavaScript">
  function whichButton(event) {
    if (event.button==1) {
      alert("You clicked the left mouse button!") }
    else {
      alert("You clicked the right mouse button!")
    }
  }
</script>
...
<body onmousedown="whichButton(event)">
...
</body>
```

Mouse event causes
page-defined function
to be called

Other events: onLoad, onMouseMove, onKeyPress, onUnload

Example 3: Page Manipulation

◆ Some possibilities

- `createElement(elementName)`
- `createTextNode(text)`
- `appendChild(newChild)`
- `removeChild(node)`

◆ Example: add a new list item

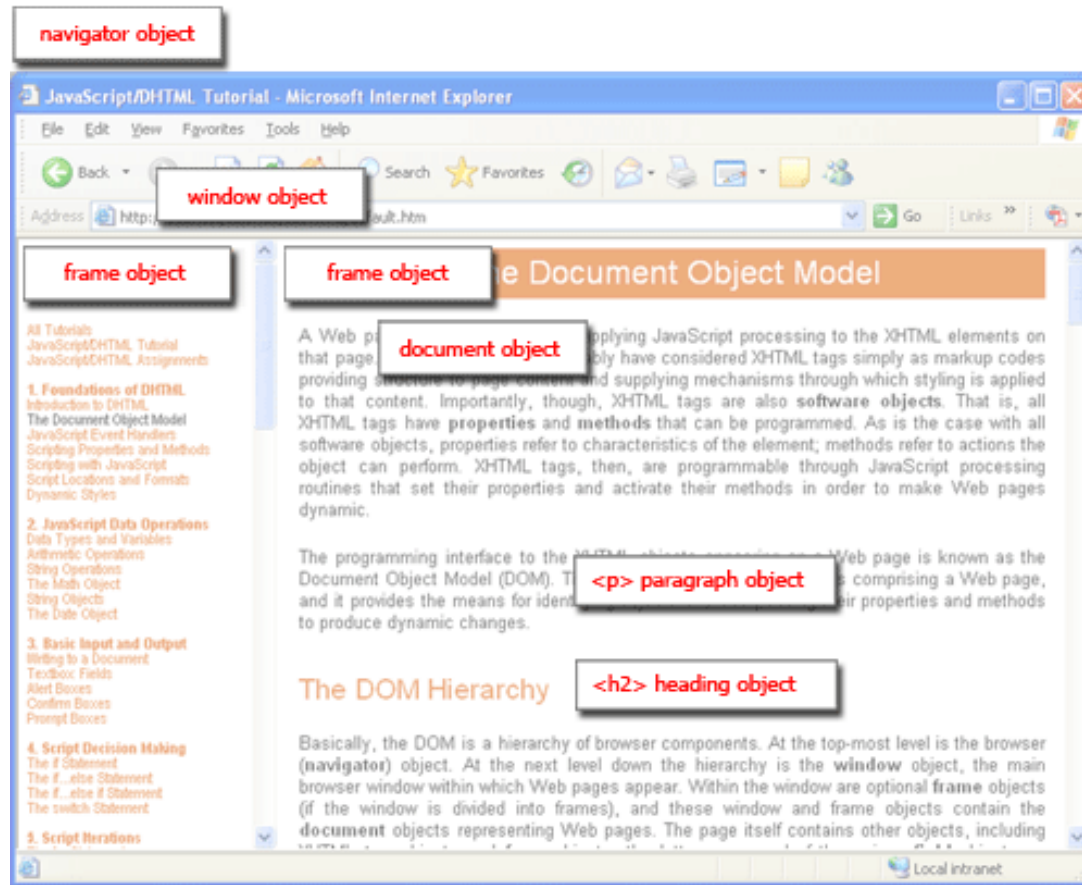
```
var list = document.getElementById('t1')
var newItem = document.createElement('li')
var newText = document.createTextNode(text)
list.appendChild(newItem)
newItem.appendChild(newText)
```

This uses the browser Document Object Model (DOM). We will focus on JavaScript as a language, not its use in the browser

Document Object Model (DOM)

- ◆ HTML page is structured data
- ◆ DOM provides representation of this hierarchy
- ◆ Examples
 - **Properties:** `document.alinkColor`, `document.URL`, `document.forms[]`, `document.links[]`, `document.anchors[]`, ...
 - **Methods:** `document.write(document.referrer)`
 - These change the content of the page!
- ◆ Also Browser Object Model (BOM)
 - `Window`, `Document`, `Frames[]`, `History`, `Location`, `Navigator` (type and version of browser)

Browser and Document Structure



W3C standard differs from models supported in existing browsers

Reading Properties with JavaScript

Sample script

1. `document.getElementById('t1').nodeName`
2. `document.getElementById('t1').nodeValue`
3. `document.getElementById('t1').firstChild.nodeName`
4. `document.getElementById('t1').firstChild.firstChild.nodeName`
5. `document.getElementById('t1').firstChild.firstChild.nodeValue`

- Example 1 returns "ul"
- Example 2 returns "null"
- Example 3 returns "li"
- Example 4 returns "text"
 - A text node below the "li" which holds the actual text data as its value
- Example 5 returns " Item 1 "

Sample HTML

```
<ul id="t1">  
<li> Item 1 </li>  
</ul>
```

Language Basics

- ◆ JavaScript is case sensitive
 - `onClick`, `ONCLICK`, ... are HTML, thus not case-sensitive
- ◆ Statements terminated by returns or semi-colons
 - `x = x+1;` same as `x = x+1`
- ◆ “Blocks” of statements enclosed in `{ ... }`
- ◆ Variables
 - Define using the `var` statement
 - Define implicitly by its first use, which must be an assignment
 - Implicit defn has global scope, even if occurs in nested scope!

JavaScript Blocks

- ◆ Use { } for grouping; **not a separate scope**

```
js> var x=3;
```

```
js> x
```

```
3
```

```
js> {var x=4; x}
```

```
4
```

```
js> x
```

```
4
```

- ◆ Not blocks in the sense of other languages

JavaScript Primitive Datatypes

- ◆ Boolean: true and false
- ◆ Number: 64-bit floating point
 - Similar to Java double and Double
 - No integer type
 - Special values NaN (not a number) and Infinity
- ◆ String: sequence of zero or more Unicode chars
 - No separate character type (just strings of length 1)
 - Literal strings using ' or " characters (must match)
- ◆ Special objects: null and undefined

Objects

- ◆ An object is a collection of named properties
- ◆ Think of it as an associative array or hash table
 - Set of name:value pairs
 - `objBob = {name: "Bob", grade: 'A', level: 3};`
 - Play a role similar to lists in Lisp / Scheme
- ◆ New members can be added at any time
 - `objBob.fullname = 'Robert';`
- ◆ Can have methods
- ◆ Can refer to `this`

Functions

- ◆ Functions are objects with method called “()”
 - A property of an object may be a function (=method)
 - function `max(x,y) { if (x>y) return x; else return y; }`;
 - `max.description = “return the maximum of two arguments”;`
 - Local declarations may appear in function body
- ◆ Call can supply any number of arguments
 - `functionname.length` : # of arguments in definition
 - `functionname.arguments.length` : # arguments in call
 - Basic types are passed by value, objects by reference
- ◆ “Anonymous” functions
 - `(function (x,y) {return x+y}) (2,3);`

Examples of Functions

◆ Curried functions

- `function CurriedAdd(x) { return function(y){ return x+y} };`
- `g = CurriedAdd(2);`
- `g(3)`

◆ Variable number of arguments

- `function sumAll() {
 var total=0;
 for (var i=0; i< sumAll.arguments.length; i++)
 total+=sumAll.arguments[i];
 return(total); }`
- `sumAll(3,5,3,5,3,2,6)`

Anonymous Functions

- ◆ Anonymous functions very useful for callbacks
 - `setTimeout(function() { alert("done"); }, 10000)`
 - Evaluation of `alert("done")` delayed until function call
- ◆ Simulate blocks by function definition and call
 - `var u = { a:1, b:2 }`
 - `var v = { a:3, b:4 }`
 - `(function (x,y) {
 var tempA = x.a; var tempB =x.b; // local variables
 x.a=y.a; x.b=y.b;
 y.a=tempA; y.b=tempB
}) (u,v) // Works because objs are passed by ref`

Basic Object Features

◆ Use a function to construct an object

- ```
function car(make, model, year) {
 this.make = make;
 this.model = model;
 this.year = year; }

```

## ◆ Objects have prototypes, can be changed

- ```
var c = new car("Ford", "Taurus", 1988);
```
- ```
car.prototype.print = function () {
 return this.year + " " + this.make + " " + this.model;}
```
- ```
c.print();
```

JavaScript in Web Pages

◆ Embedded in HTML page as `<script>` element

- JavaScript written directly inside `<script>` element
 - `<script> alert("Hello World!") </script>`
- Linked file as `src` attribute of the `<script>` element
`<script type="text/JavaScript" src="functions.js"></script>`

◆ Event handler attribute

``

◆ Pseudo-URL referenced by a link

`Click me`

We are looking at JavaScript as a language; ignore BOM, DOM, AJAX

Language Features in This Class

- ◆ Stack memory management
 - Parameters, local variables in activation records
- ◆ Garbage collection
- ◆ Closures
 - Function together with environment (global variables)
- ◆ Exceptions
- ◆ Object features
 - Dynamic lookup, encapsulation, subtyping, inheritance
- ◆ Concurrency

Stack Memory Management

◆ Local variables in activation record of function

```
function f(x) {  
    var y = 3;  
    function g(z) { return y+z; };  
    return g(x);  
}
```

```
var x= 1; var y =2;
```

```
f(x) + y;
```

Garbage Collection

- ◆ Automatic reclamation of unused memory
- ◆ Navigator 2: per-page memory management
 - Reclaim memory when browser changes page
- ◆ Navigator 3: reference counting
 - Each memory region has associated count
 - Count modified when pointers are changed
 - Reclaim memory when count reaches zero
- ◆ Navigator 4: mark-and-sweep, or equivalent
 - Garbage collector marks reachable memory
 - Sweep and reclaim unreachable memory

Closures

◆ Return a function from function call

- `function f(x) {
 var y = x;
 return function (z){y += z; return y;}
}`
- `var h = f(5);`
- `h(3);`

◆ Can use this idea to define objects with “private” fields (subtle)

- See <http://www.crockford.com/JavaScript/private.html>

Exceptions

◆ Throw an expression of any type

```
throw "Error2";
```

```
throw 42;
```

```
throw {toString: function() { return "I'm an object!"; } };
```

◆ Catch

```
try {
```

```
  } catch (e if e == "FirstException") {      // do something
```

```
  } catch (e if e == "SecondException") {    // do something else
```

```
  } catch (e){                               // executed if no match above
```

```
}
```

Reference: <http://developer.mozilla.org/en/docs/>

Core_JavaScript_1.5_Guide :Exception_Handling_Statements

Object features

◆ Dynamic lookup

- Method depends on run-time value of object

◆ Encapsulation

- Object contains private data, public operations

◆ Subtyping

- Object of one type can be used in place of another

◆ Inheritance

- Use implementation of one kind of object to implement another kind of object

Concurrency

- ◆ JavaScript itself is single-threaded
 - How can we tell if a language provides concurrency?
- ◆ AJAX provides a form of concurrency
 - Create XMLHttpRequest object, set callback function
 - Call request method, which continues asynchronously
 - Reply from remote site executes callback function
 - Event waits in event queue...
 - Closures important for proper execution of callbacks
- ◆ Another form of concurrency
 - Use setTimeout to do cooperative multi-tasking

JavaScript eval

- ◆ Evaluate string as code (seen this before?)
 - The `eval` function evaluates a string of JavaScript code, in scope of the calling code
 - `var code = "var a = 1";`
 - `eval(code); // a is now '1'`
 - `var obj = new Object();`
 - `obj.eval(code); // obj.a is now 1`
 - Common use: efficiently deserialize a complicated data structure received over network via XMLHttpRequest
- ◆ What does it cost to have `eval` in the language?
 - Can you do this in C? What would it take to implement?

Unusual Features of JavaScript

- ◆ Eval, run-time type checking functions
- ◆ Support for pattern matching (reg. expressions)
- ◆ Can add methods to an object
- ◆ Can delete methods of an object
 - `myobj.a = 5; myobj.b = 12; delete myobj.a;`
- ◆ Iterate over methods of an object
 - `for (variable in object) { statements }`
- ◆ With statement (“considered harmful” – why?)
 - `with (object) { statements }`