JavaScript Static Analysis with IronWASP

Nullcon
Goa 2012

Lavakumar Kuppan
Twitter: @lavakumark
e-Mail: lava@ironwasp.org
http://ironwasp.org

About

- Penetration Tester
  - 5+ years of experience
- Security Researcher
  - Flash 0-day
  - WAF bypass 0-day using HPP
  - Multiple HTML5 based attack techniques
  - 5th best Web Application Hacking Technique of 2010
  - Attack and Defense Labs – http://andlabs.org
  - HTML5 Security Resources Repository – http://html5security.org

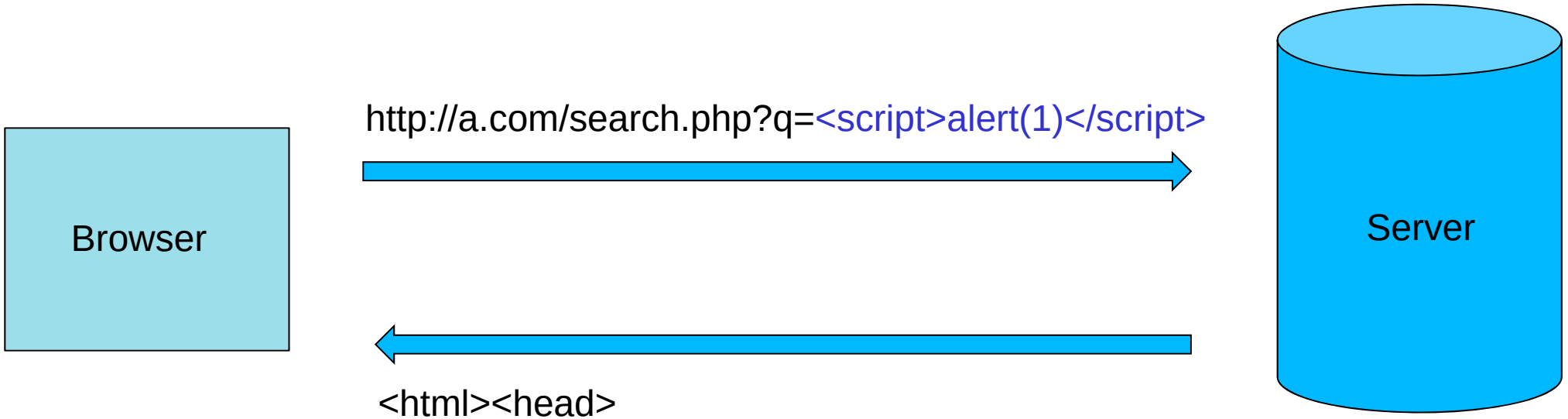
About

- Developer
  - IronWASP (C# + Python + Ruby)
  - Ravan (PHP + JavaScript)
  - JS-Recon (JavaScript)
  - Shell of the Future (C# + JavaScript)
  - Imposter (C# + JavaScript)

- Speaker
  - BlackHat
  - OWASP AppSec Asia
  - NullCon
  - SecurityByte
  - ClubHack

Cross-site Scripting??

Server-side Vulnerability



Not Exactly there is also
“DOM based XSS”

[DEMO]

DOM XSS Source & Sink

Source:

DOM Properties that can be influenced by an attacker

Sink:

DOM Properties, JavaScript functions and other client-side entities that can lead to or influence client-side code execution

Source Types

- Location based
- Client-side Storage based
- Navigation based
- Cross-domain

Location based Source

- location
- location.hash
- location.href
- location.pathname
- location.search
- document.URL
- document.baseURI
- document.documentURI
- document. URLUnencoded

Client-side Storage Based

- document.cookie
- sessionStorage*
- localStorage*
- Web SQL Database*
- Indexed DB*

*** HTML5**

Navigation Based

- window.name
- document.referrer
- history (HTML5)

Cross-domain

- postMessage*
- XHR call responses from 3rd party JavaScript API
- JSON calls backs from 3rd party JavaScript API

***HTML5**

Sink Types

- Execution based
- Url Based
- HTML Based
- Others

Execution Based

- eval()
- Function()
- setTimeout()
- setInterval()
- execScript() (IE Only)
- crypto.generateCRMFRequest() (FF Only)

Url Based

- location
- location.assign()
- location.replace()
- location.href
- location.protocol*
- location.search*
- location.hostname*
- location.pathname*

***Indirect impact**

HTML Based

- document.write()
- document.writeln()
- HTML Elements
- HTML Element Attributes
  - ‘src’
  - onclick, onload, onerror etc
  - Form action
  - href

Others

- XHR Calls
  - open()
  - send()
  - setRequestHeader()
- postMessage
- Client-side Storage
- JavaScript variables

JavaScript Static Analysis using IronWASP

[ONLY DEMOS FROM THIS POINT]

DOM XSS Vulnerable Code Example - 1

Source Code

```
<script>
   var l = location.hash.slice(1);
   eval(l);
</script>
```

IronWASP Trace

```
1  var l = location.hash.slice(1);
2  eval(l);
```

DOM XSS Vulnerable Code Example - 2

Source Code

```
<script>
    var a = "a.b.c.d";
    arr = a.split(".");
    var l = location.hash.slice(1);
    c = "xxx" + arr[1];
    d = l.indexOf("/");
    f = l.substring(d);
    s = eval;
    Add(c, arr);
    s(l);
 </script>
```

IronWASP Trace

```
1    var a = "a.b.c.d";
2    arr = a.split(".");
3    var l = location.hash.slice(1);
4    c = "xxx" + arr[1];
5    d = l.indexOf("/");
6    f = l.substring(d);
7    s = eval;
8    Add(c, arr);
9    s(l);
```

DOM XSS Vulnerable Code Example - 3

Source Code

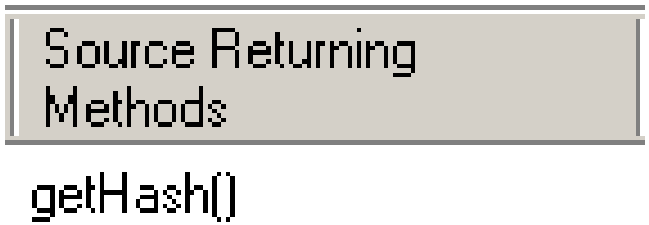
```
<script>
    function getHash()
    {
        var l = location.hash.slice(1);
        return l;
    }
    var h = getHash();
    eval(h);
</script>
```

IronWASP Trace

```
1  function getHash() {
2      var l = location.hash.slice(1);
3      return l;
4  }
5  var h = getHash();
6  eval(h);
```

Update Taint Config

- The function ‘getHash’ returns a DOM XSS Source.
- Let’s update the ‘Taint Config’ with that:



- Let’s redo the trace now.

IronWASP Trace

```
1  function getHash() {
2      var l = location.hash.slice(1);
3      return l;
4  }
5  var h = getHash();
6  eval(h);
```

DOM XSS Vulnerable Code Example - 4

Source Code

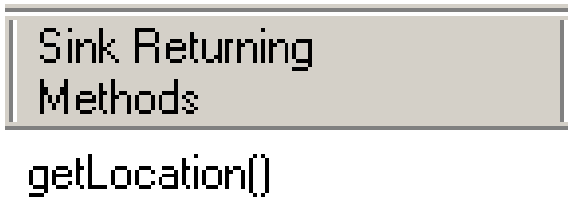
```
<script>
    function getLocation()
   {
         var l = location;
         return l;
   }
   var loc = getLocation();
   loc = name;
</script>
```

IronWASP Trace

```
1   function getLocation() {
2       var l = location;
3       return l;
4   }
5   var loc = getLocation();
6   loc = name;
```

Update Taint Config

- The function ‘getLocation’ returns a DOM XSS Sink.
- Let’s update the ‘Taint Config’ with that:



- Let’s redo the trace now.

IronWASP Trace

```
1  function getLocation() {
2      var l = location;
3      return l;
4  }
5  var loc = getLocation();
6  loc = name;
```

DOM XSS Vulnerable Code Example - 5

Source Code

```
<script>
    function doEval(text)
   {
         eval(text);
   }
   var h = location.hash.slice(1);
   doEval(h);
</script>
```

IronWASP Trace

```
1    function doEval(text) {
2        eval(text);
3    }
4    var h = location.hash.slice(1);
5    doEval(h);
```

Update Taint Config

- The function ‘doEval’ assigns its argument to a DOM XSS Sink.
- Let’s update the ‘Taint Config’ with that:

- Let’s redo the trace now.

IronWASP Trace

```
1    function doEval(text) {
2        eval(text);
3    }
4    var h = location.hash.slice(1);
5    doEval(h);
```

DOM XSS Vulnerable Code Example - 6

Source Code

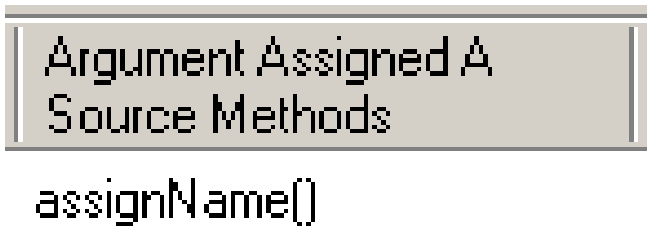
```
<script>
   function assignName(property)
   {
         var n = window.name;
         property = n;
   }
   var l = location;
   assignName(l);
</script>
```

IronWASP Trace

```
1  function assignName(property) {
2      var n = window.name;
3      property = n;
4  }
5  var l = location;
6  assignName(l);
```

Update Taint Config

- The function ‘assignName’ assigns a DOM XSS Source to its argument.
- Let’s update the ‘Taint Config’ with that:



- Let’s redo the trace now.

IronWASP Trace

```
1  function assignName(property) {
2      var n = window.name;
3      property = n;
4  }
5  var l = location;
6  assignName(l);
```

DOM XSS Vulnerable Code Example - 7

Source Code

```
<script src="sourceret.js"></script>
    function getHash() {
     var l = location.hash.slice(1);
     return l;
     }
<script src="sinkass.js"></script>
    function doEval(text) {
     eval(text);
    }
<script>
     var h = getHash();
     doEval(h);
</script>
```

IronWASP Trace

```
1        var h = getHash();
2        doEval(h);
```

- We did not the analyze the JavaScript that was loaded from the ‘sourceret.js’ and ‘sinkass.js’ files.
- We can get the list of all external scripts referenced for all pages in a site by analyzing the requests and responses captured in the logs.
- This can be done with a simple script

The simple Python script:

```
sessions = Session.FromProxyLog()
for sess in sessions:
    if sess.Response != None:
        if sess.Response.IsHtml:
            script_files = sess.Response.Html.GetValues("script", "src")
            print sess.Request.Url
            for sf in script_files:
                print "\t - " + sf
```

Update Taint Config

- The function ‘getHash’ from ‘sourceret.js’ returns a DOM XSS Source.
- The function ‘doEval’ from ‘sinkass.js’ assigns its argument to a DOM XSS Sink.
- Let’s update the ‘Taint Config’ with that:

Source Returning Methods
getHash()

Argument Assigned To Sink Methods
doEval()

- Let’s redo the trace now.

IronWASP Trace

References

- IronWASP http://ironwasp.org
- DOM XSS Wiki http://code.google.com/p/domxsswiki