Global Information Assurance Certification Paper

Client Fingerprinting via Analysis of Browser Scripting Environment

GIAC (GWAPT) Gold Certification

Author: Mark Fioravanti, mark.fioravanti.ii@gmail.com
Advisor: Aman Hardikar

Accepted: N/A, DRAFT

Abstract

An essential part of any Web Application Penetration Test that includes the exploitation of clients is the ability to accurately fingerprint the end point. There exists the ability to determine potentially unique characteristics of the client through the innate scripting functions provided within each major browser. These characteristics range from identifying the browser to the operating system (O/S). The level of detail that can be obtained based on identified characteristics can range from simply identifying the browser family to identifying the specific browser version, O/S version and in some cases the processor architecture. With the use of JavaScript, VBScript and Jscript, a fairly accurate fingerprint can be constructed of the client system. Despite various browsers including the ability to spoof User Agents and some plug-ins providing the ability to spoof various components of the Document Object Model (DOM), there are still a number of ways that a web application penetration tester can fingerprint the system. This paper details which fingerprints can be collected and analyzed, as well as using specific fingerprints to aid in the identification of specific clients.

1. Introduction

During a Web Application Penetration Test, it is important to test the security of the clients that are interacting with the application. Although not all Web Application Penetration Testing engagements include this activity, when it is performed it is essential to properly identify the client that is being exploited. Beyond simply identifying the browser, it is also important to identify the operating system (O/S) before attempting to manipulate or exploit the client. An accurate assessment of the characteristics of the client allows for the execution of optimized scripts and/or executing a few exploits instead of executing all of the available exploits and hoping the client does not notice or crash.

There are a few websites and projects which attempt to document the functionality and behaviors of the various browsers. W3schools (w3schools.com) provides an online tutorial for learning to write JavaScript code, and it provides information when specific JavaScript functions have been implemented in each of the major browsers. The Browser Security Handbook (Zalewski, 2009) defines a number of test cases which can be used to identify specific families of browsers such as determining if the browser is Microsoft Internet Explorer or Mozilla Firefox, but being able to distinguish between Firefox 3.6.4 or 3.6.8. Another project is the docType project (Google) which is attempts to enumerate the various objects and properties available within each of the browsers. A significant amount of information is available but it is similar to the Browser Security Handbook in that it only allows families of browsers to be identified.

Browser fingerprinting techniques commonly use the User Agent string to determine the client that is interacting with a web site, but all of the major browsers offer various methods for changing this variable. Some browsers allow the User Agent to be configured via the registry or via a configuration option while some browsers have plug-ins which allows a large number of environment variables to be manipulated. The User Agent string is only one of a large number of environment objects and methods which can be used to determine the type of browser.

Beyond using the User Agent, there are a number of other properties available for fingerprinting the client. Each browser provides its own scripting environment and based on how the scripting environment interacts with scripts, the properties that are made available to scripts, and the specific values that are provided it is possible to get an accurate assessment of the browser and operating system.

2. Browser Scripting Environments

All major browsers support at least one scripting language. JavaScript is the scripting language that is typically supported, although some browsers may partially or fully support other languages such as JScript and Visual Basic Script (VBScript). Support for other languages within a browser is handled by the installation or inclusion of browser plug-ins, such as Java and ActionScript.

2.1. JavaScript

JavaScript is an implementation of the European Computer Manufacturers Association (ECMA) Specification 262. JavaScript was originally developed under the name of Mocha or LiveScript for the Netscape browser in 1995 (Netscape, 1995). The current version of JavaScript is 1.9 (ECMAScript version 5). However, most browsers do not support version 1.9 of JavaScript. They typically support at least version 1.3 (ECMAScript version 2).

2.1.1. Script Versions

JavaScript allows different versions of the scripting language to be executed depending upon the version of JavaScript (JavaScript Kit). Originally this allowed custom scripts to be executed based on the version of JavaScript, but since RFC4929 (Hoehrmann, 2006) this has been made obsolete. Most browsers still support the controlled execution of JavaScript versions despite the changes recommended by the RFC.

Script Tag	Browser Execution
`<script></script>`	The browser executes the script with its default scripting engine and the default language.
`<script language="javascript"></script>`	The browser engine that is used to

`<script language="jscript"></script>` `<script language="vbscript"></script>`	execute the script is controlled by the language tag. JavaScript, Microsoft JScript or Microsoft VBScript will be executed.
`<script language="javascript1.1"></script>` `<script language="javascript1.2"></script>`	The browser engine will only execute the script if it supports that specific version of the JavaScript language.

Table 2.1.1-1: Using Script Tags to Specify Different Languages in HTML

2.1.2. Scripting Environment

The scripting environment includes a number of objects available for determining the nature of the browser that the script is executing within. The Document Object Model (DOM) which allows scripts to interact with the HTML that is being presented to the client via the 'document' object. The World Wide Web Consortium (W3C) maintains the DOM standard (W3C, 2005). A number of non-standard objects exist that allow scripts to interact with various aspects of the scripting environment, such as the window object. Within the 'window' object there are a number of objects like history, location, navigator and screen which can provide additional information about the client and none of which are based on a formal standard (w3schools). Basic information about the functions and properties available within a browser can be obtained at either the w3schools tutorial site or the docType project (Google).

2.2. Microsoft Visual Basic Script (VBScript)

Microsoft VBScript is another scripting language that is implemented in only a few browsers and is modeled on the Visual Basic language (also by Microsoft). Microsoft VBScript was originally released in 1996, and like Microsoft JScript the current version of the scripting engine is v5.8. Unlike JavaScript and JScript, VBScript is not implemented to be compliant with ECMA-262 specification (Microsoft).

2.3. Microsoft JScript

Microsoft JScript is Microsoft's implementation of the ECMA-262 specification, and the current version of the scripting engine is version 5.8. When JavaScript is being executed within the Microsoft Internet Explorer browser, it is actually being executed with the JScript engine (Microsoft). By using the language attributes of the HTML script

tags, it is possible to strictly invoke the JScript functionality. There are a number of ECMA-262 compliant and non-compliant features of the JScript language (Microsoft).

3. Collection

A number of different browsers and O/S combinations are possible and in order to obtain sufficient coverage of relevant browser and O/S combinations, current market share was used to identify the most common browsers and the most common O/Ss on the Internet. Beyond identifying common desktop solutions, it is important to include mobile devices as they are becoming increasingly common on the Internet (and in general).

3.1. Browsers

When determining which browsers to fingerprint, browsers were selected based on two criteria; 1) overall browser market share (GlobalStats, 2010, NetMarketShare, 2010, and w3schools, 2010), and 2) overall browser share for an O/S family. It was decided that the browsers listed in the following table would be used for collecting fingerprints.

Browser	Source	Version(s)
Microsoft Internet Explorer	http://www.microsoft.com	6.0, 7.0, 8.0, and 9.0 (Preview and Beta)
Mozilla Firefox	http://www.mozilla.com	2.0.0.x, 3.0.x, 3.5.x, 3.6.x, 4.0 (Beta)
Apple Safari	http://www.apple.com	4.0.x, and 5.0.x
Google Chrome	http://chrome.google.com	2.x, 3.x, 4.x, 5.x, 6.x, 7.x
ASA Software Opera	http://www.opera.com	10.x
KDE's Konqueror	Included within each GNU/Linux and BSD distribution	3.5.x, 4.x

Table 3.2-1: Fingerprinted Web Browsers

3.2. Operating Systems

When determining which O/Ss to host the browsers on for the purposes of fingerprinting, they were selected based on three criteria; 1) overall O/S market share, 2) general O/S availability/patch accessibility, and 3) ease of install.

O/S	Family	Variants
Microsoft Windows	XP Professional	SP1, SP2, SP3
	Vista Ultimate	No SP, SP1, SP2
	7 Ultimate	No SP
Mac OS X	Snow Leopard	10.6.2-4
	iOS	iPhone, iPod Touch, iPad
GNU/Linux	Fedora	11, 12, 13, 14 (Alpha)
	Ubuntu	8.04 LTS, 9.04, 9.10, 10.04 LTS, 10.10 (Beta)
	CentOS	5.4, 5.5
	Sabayon	5.1, 5.2
	OpenSUSE	11.1, 11.2, 11.3
	Mandriva	2010
BSD	FreeBSD	PC-BSD

Table 3.2-1: Fingerprinted Operating Systems

In addition to collecting variants of the O/S versions, variant processor architectures were collected; 32-bit (x86) and 64-bit (x86-64) architectures. Not all freely available variants of the O/S were available in 64-bit versions.

3.3. Mobile Devices

Mobile devices are becoming more common and identifying and correctly interacting with these types of devices will become more important as time passes. A number of mobile devices were included in the sample of browsers and O/Ss.

Device	Browser	Variants
Android	Default Android Browser	Android 2.1, Android 2.2
	ASA Software Opera Mini	5.1

BlackBerry	Default Blackberry Browser	9630
Apple iPhone, iPod Touch, iPad	Safari 4.0.x	iPhone 3G 3.x, 4.x iPod Touch 3.x, 4.x iPad 3.2.x
	ASA Software Opera Mini	5.1
Nokia Internet Tablet/Phone (N900, N810, N800, N770)	Maemo Browser (MicroB)	Maemo 5
	ASA Software Opera Mobile 10.1	Mobi 4

Table 3.2-1: Fingerprinted Operating Systems

3.4. Collection Page

A simple HTML page can be used to collect fingerprint information about a browser. The collection page was hosted on an Internet accessible site to allow browsers and mobile devices to access the page regardless of their network accessibility. The collection page is designed such that multiple pages work to collect different aspects of the browser's scripting environment. The collection page is designed with the following sections;

- **Variable Initialization** – various script variables are defined and initialized. Variables are defined to allow information from VBScript and JScript sections to be accessed by JavaScript.
- **JavaScript Version Detection** – individual scripts are executed to collect information about the versions of JavaScript that the browser supports.
- **JScript Detection** – a versioning script is executed to determine the version of the script engine and build number of the Microsoft JScript engine.
- **VBScript Detection** – a versioning script is executed to determine the version of the script engine and build number of the Microsoft VBScript engine.

- **Browser Objects and DOM Collection** – multiple scripts are executed to test if individual browser objects (navigator and window) and/or DOM properties exist. If they exist, their values are recorded. When testing for specific objects, the object is first tested to see if it exists within the environment. If the object does not exist, a value of 'undefined' is returned for a value. If the object does exist, it is checked to see if it has a value assigned to it, if it does not 'dne' is returned (e.g. the value is an empty string so it does not exist), otherwise the value of the property is recorded.
- **Variable Consolidation** – the results of the various script tests are consolidated into a single string for either display or submission.
- **Collection/Submission** – the resulting string is displayed in a textarea so that can be inspected prior to submission.

When collecting some of the navigator properties it was noticed that some browsers would crash during the collection of some navigator properties. The HTML collection page was modified such that the document, navigator, and window properties were collected via individual scripts to prevent a single script error from preventing the collection of other information. The following HTML page was used to collect information about each browser.

```
<html><head><title>Collect</title>
<script type="text/javascript">
<!--
  var ver_JS = 1.0;
  var env_MJS = 'disabled'; var ver_MJS = '0.0'; var bld_MJS = -1; var
typ_MJS = 'none';
  var env_VBS = 'disabled'; var ver_VBS = '0.0'; var bld_VBS = -1; var
typ_VBS = 'none';
  var __n = navigator; var __d = document; var __w = window;
//-->
</script>
<script language="Javascript1.1">ver_JS = 1.1;</script>
<script language="Javascript1.2">ver_JS = 1.2;</script>
<script language="Javascript1.3">ver_JS = 1.3;</script>
<script language="Javascript1.4">ver_JS = 1.4;</script>
<script language="Javascript1.5">ver_JS = 1.5;</script>
<script language="Javascript1.6">ver_JS = 1.6;</script>
<script language="Javascript1.7">ver_JS = 1.7;</script>
<script language="Javascript1.8">ver_JS = 1.8;var strAnTest = '  TXT
';if (strAnTest.trim) {ver_JS = '1.8.1';}</script>
<script language="Javascript1.9">ver_JS = 1.9;</script>
<script language="Javascript2.0">ver_JS = 2.0;</script>
<script type="text/jscript" language="JScript">
```

```
<!--
  env_MJS = 'enabled';
  ver_MJS = ScriptEngineMajorVersion() + '.' +
ScriptEngineMinorVersion();
  bld_MJS = ScriptEngineBuildVersion();
  typ_MJS = ScriptEngine();
//-->
</script>
<script type="text/vbscript" language="vbscript">
<!--
  env_VBS = "enabled"
  ver_VBS = ScriptEngineMajorVersion & "." & ScriptEngineMinorVersion
  bld_VBS = ScriptEngineBuildVersion
  typ_VBS = ScriptEngine
//-->
</script>
<script language="javascript" type="text/javascript">
  var n_userAgent = 'exception';
  n_userAgent = ((__n.userAgent) ? __n.userAgent : ((__n.userAgent ==
undefined) ? 'undefined' : 'dne'))
</script>
<script language="javascript" type="text/javascript">
  var n_appName = 'exception';
  n_appName = ((__n.appName) ? __n.appName : ((__n.appName ==
undefined) ? 'undefined' : 'dne'))
</script>
<script language="javascript" type="text/javascript">
  var n_appCodeName = 'exception';
  n_appCodeName = ((__n.appCodeName) ? __n.appCodeName :
((__n.appCodeName == undefined) ? 'undefined' : 'dne'))
</script>
<script language="javascript" type="text/javascript">
  var n_appVersion = 'exception';
  n_appVersion = ((__n.appVersion) ? __n.appVersion : ((__n.appVersion
== undefined) ? 'undefined' : 'dne'))
</script>
<script language="javascript" type="text/javascript">
  var n_appMinorVersion = 'exception';
  n_appMinorVersion = ((__n.appMinorVersion) ? __n.appMinorVersion :
((__n.appMinorVersion == undefined) ? 'undefined' : 'dne'))
</script>
<script language="javascript" type="text/javascript">
  var n_browserLanguage = 'exception';
  n_browserLanguage = ((__n.browserLanguage) ? __n.browserLanguage :
((__n.browserLanguage == undefined) ? 'undefined' : 'dne'))
</script>
<script language="javascript" type="text/javascript">
  var n_cpuClass = 'exception';
  n_cpuClass = ((__n.cpuClass) ? __n.cpuClass : ((__n.cpuClass ==
undefined) ? 'undefined' : 'dne'))
</script>
<script language="javascript" type="text/javascript">
  var n_systemLanguage = 'exception';
  n_systemLanguage = ((__n.systemLanguage) ? __n.systemLanguage :
((__n.systemLanguage == undefined) ? 'undefined' : 'dne'))
</script>
<script language="javascript" type="text/javascript">
```

```
  var n_language = 'exception';
  n_language = ((__n.language) ? __n.language : ((__n.language ==
undefined) ? 'undefined' : 'dne'))
</script>
<script language="javascript" type="text/javascript">
  var n_buildID = 'exception';
  n_buildID = ((__n.buildID) ? __n.buildID : ((__n.buildID ==
undefined) ? 'undefined' : 'dne'))
</script>
<script language="javascript" type="text/javascript">
  var n_oscpu = 'exception';
  n_oscpu = ((__n.oscpu) ? __n.oscpu : ((__n.oscpu == undefined) ?
'undefined' : 'dne'))
</script>
<script language="javascript" type="text/javascript">
  var n_platform = 'exception';
  n_platform = ((__n.platform) ? __n.platform : ((__n.platform ==
undefined) ? 'undefined' : 'dne'))
</script>
<script language="javascript" type="text/javascript">
  var n_product = 'exception';
  n_product = ((__n.product) ? __n.product : ((__n.product ==
undefined) ? 'undefined' : 'dne'))
</script>
<script language="javascript" type="text/javascript">
  var n_productSub = 'exception';
  n_productSub = ((__n.productSub) ? __n.productSub : ((__n.productSub
== undefined) ? 'undefined' : 'dne'))
</script>
<script language="javascript" type="text/javascript">
  var n_userLanguage = 'exception';
  n_userLanguage = ((__n.userLanguage) ? __n.userLanguage :
((__n.userLanguage == undefined) ? 'undefined' : 'dne'))
</script>
<script language="javascript" type="text/javascript">
  var n_userProfile = 'exception';
  n_userProfile = ((__n.userProfile) ? 'present' : ((__n.userProfile ==
undefined) ? 'undefined' : 'dne'))
</script>
<script language="javascript" type="text/javascript">
  var n_vendor = 'exception';
  n_vendor = ((__n.vendor) ? __n.vendor : ((__n.vendor == undefined) ?
'undefined' : 'dne'))
</script>
<script language="javascript" type="text/javascript">
  var n_vendorSub = 'exception';
  n_vendorSub = ((__n.vendorSub) ? __n.vendorSub : ((__n.vendorSub ==
undefined) ? 'undefined' : 'dne'))
</script>
<script language="javascript" type="text/javascript">
  var strEnv = "Javascript/" + ver_JS;

    if (env_MJS == 'enabled') {
      strEnv += " JScript/" + ver_MJS + ((bld_MJS != -1) ? (" JScript/"
+ bld_MJS) : "" );
    }
```

```
      if (env_VBS == 'enabled') {
        strEnv += " VBScript/" + ver_VBS + ((bld_VBS != -1) ? ("
VBScript/" + bld_VBS) : "");
      }

      var strVar = "";

      if (__w.opera) {
        strVar += ((__w.opera.version()) ? (" Opera/" +
__w.opera.version()): "");
        strVar += ((__w.opera.buildNumber()) ? (" Opera/" +
__w.opera.buildNumber()): "");
      }

      strVar += ' ' + ((__d.all) ? "" : "!" ) + 'd.all';
      strVar += ' ' + ((__d.childNodes) ? "" : "!" ) + 'd.childNodes';
      strVar += ' ' + ((__d.compatMode) ? "" : "!" ) + 'd.compatMode';
      strVar += ' ' + ((__d.documentMode) ? "" : "!" ) +
'd.documentMode';
      strVar += ' ' + ((__d.getElementById) ? "" : "!" ) +
'd.getElementById';
      strVar += ' ' + ((__d.getElementsByClassName) ? "" : "!" ) +
'd.getElementsByClassName';
      strVar += ' ' + ((__n.savePreferences) ? "" : "!" ) +
'n.savePreferences';
      strVar += ' ' + ((__w.XMLHttpRequest) ? "" : "!" ) +
'w.XMLHttpRequest';
      strVar += ' ' + ((__w.globalStorage) ? "" : "!" ) +
'w.globalStorage';
//      strVar += ' ' + '#w.globalStorage';
      strVar += ' ' + ((__w.postMessage) ? "" : "!" ) +
'w.postMessage';

      strText  = "navigator.userAgent: " + n_userAgent + "\n";
      strText += "navigator.appName: " + n_appName + "\n";
      strText += "navigator.appCodeName: " + n_appCodeName + "\n";
      strText += "navigator.appVersion: " + n_appVersion + "\n";
      strText += "navigator.appMinorVersion: " + n_appMinorVersion +
"\n";
      strText += "navigator.browserLanguage: " + n_browserLanguage +
"\n";
      strText += "navigator.cpuClass: " + n_cpuClass + "\n";
      strText += "navigator.systemLanguage: " + n_systemLanguage +
"\n";
      strText += "navigator.language: " + n_language + "\n";
      strText += "navigator.buildID: " + n_buildID + "\n";
      strText += "navigator.oscpu: " + n_oscpu + "\n";
      strText += "navigator.platform: " + n_platform + "\n";
      strText += "navigator.product: " + n_product + "\n";
      strText += "navigator.productSub: " + n_productSub + "\n";
      strText += "navigator.userLanguage: " + n_userLanguage + "\n";
      strText += "navigator.userProfile: " + n_userProfile + "\n";
      strText += "navigator.vendor: " + n_vendor + "\n";
      strText += "navigator.vendorSub: " + n_vendorSub + "\n";
      strText += "custom.scripting: " + strEnv + "\n";
      strText += "custom.property: " + strVar + "\n";
</script>
```

```
</head><body>
<script type="text/javascript">
<!--
  document.write('<textarea cols="100" rows="25">\n');
  document.write(strText);
  document.write('</textarea>\n');
//-->
</script>
</body></html>
```

Figure 3.3-1: Browser Collection Page

The page will contain the following properties when displayed.

```
navigator.userAgent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; .NET CLR
2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729; .NET4.0C)
navigator.appName: Microsoft Internet Explorer
navigator.appCodeName: Mozilla
navigator.appVersion: 4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; .NET CLR
2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729; .NET4.0C)
navigator.appMinorVersion: 0
navigator.browserLanguage: en-us
navigator.cpuClass: x86
navigator.systemLanguage: en-us
navigator.language: undefined
navigator.buildID: undefined
navigator.oscpu: undefined
navigator.platform: Win32
navigator.product: undefined
navigator.productSub: undefined
navigator.userLanguage: en-us
navigator.userProfile: undefined
navigator.vendor: undefined
navigator.vendorSub: undefined
custom.order:
custom.scripting: Javascript/1.3 JScript/5.8 JScript/22960 VBScript/5.8 VBScript/23000
custom.property:  d.all d.childNodes d.compatMode d.documentMode d.getElementById !
d.getElementsByClassName !n.savePreferences w.XMLHttpRequest !w.globalStorage w.postMessage
```

Figure 3.3-2: Results of Viewing the Collection Page (Internet Explorer 8.0)

4. Analysis

O/Ss were loaded into virtual machines, each O/S had a number of different browsers loaded, and the HTML collection page was opened in each browser. The information collected was recorded into a plain text file and then the results were organized by directory (Blackberry, Droid, IE, Firefox, Chrome, Safari, Opera, and Konqueror).

The following script was used for analysis.

```
#!/bin/bash

if [ -d Analysis ]; then
  rm -rf Analysis
fi
mkdir Analysis
```

```
for browser in *
do
  if [ -d $browser ]; then
    if [ "$browser" != "Analysis" ]; then
      mkdir Analysis/$browser

      label=`echo $browser | tr '[A-Z]' '[a-z]'`
      count=`ls $browser/needlebeard-*.txt | wc -l`

      echo ""
      echo "$browser (${count})"

      if [ "$count" -gt "0" ]; then
        for y in appName appCodeName appVersion appMinorVersion
browserLanguage buildID cpuClass language oscpu platform product
productSub systemLanguage userAgent userLanguage userProfile vendor
vendorSub
        do
          grep -i "navigator.$y:" ${browser}/*.txt | perl -pi -e
's/\r\n?/\n/g' | sed -e "s/navigator\.$y: //" | sed -e
"s/$browser\/needlebeard-//" | sed -e "s/\.txt:/ /" | sed -e "s/-/ /" |
sort | uniq | sed '/^$/d' > Analysis/$browser/$label.navigator.$y.index

          grep -ih "navigator.$y:" ${browser}/*.txt | perl -pi -e
's/\r\n?/\n/g' | sed -e "s/^navigator\.$y: //" | sort | uniq | sed
'/^$/d' > Analysis/$browser/$label.navigator.$y.values
          z=`wc -l Analysis/$browser/$label.navigator.$y.values | awk
'{print $1}'`
          echo "[*] navigator.$y (${z})"

          while read i;
          do
            echo "$i" >> Analysis/$browser/$label.navigator.$y.map
            grep -lw "navigator.$y: $i" ${browser}/*.txt | sed -e
"s/^$browser\/needlebeard-/ /" | sed -e "s/-/ /" | sed -e "s/\.txt$//"
| sort | uniq | sed '/^$/d' >>
Analysis/$browser/$label.navigator.$y.map
          done < Analysis/$browser/$label.navigator.$y.values
        done

        for y in scripting property
        do
          grep -i "custom.$y:" ${browser}/*.txt | perl -pi -e
's/\r\n?/\n/g' | sed -e "s/custom\.$y: //" | sed -e
"s/$browser\/needlebeard-//" | sed -e "s/\.txt:/ /" | sed -e "s/-/ /" |
sort | uniq | sed '/^$/d' > Analysis/$browser/$label.custom.$y.index

          grep -ih "custom.$y:" ${browser}/*.txt | perl -pi -e
's/\r\n?/\n/g' | sed -e "s/^custom\.$y: //" | sort | uniq | sed '/^$/d'
> Analysis/$browser/$label.custom.$y.values
          z=`wc -l Analysis/$browser/$label.custom.$y.values | awk
'{print $1}'`
          echo "[*] custom.$y (${z})"

          while read i;
          do
```

```
            echo "$i" >> Analysis/$browser/$label.custom.$y.map
            grep -lw "custom.$y: $i" ${browser}/*.txt | sed -e
"s/^$browser\/needlebeard-/ /" | sed -e "s/-/ /" | sed -e "s/\.txt$//"
| sort | uniq | sed '/^$/d' >> Analysis/$browser/$label.custom.$y.map
          done < Analysis/$browser/$label.custom.$y.values
        done
      fi
    fi
  fi
done
```

Figure 4-1: Collection Analysis Script

When the collection analysis script is executed, it will output information similar to the following to the screen to provide information about the processing being performed.

```
Blackberry (2)
[*] navigator.appName (1)
[*] navigator.appCodeName (1)
[*] navigator.appVersion (2)
[*] navigator.appMinorVersion (1)
[*] navigator.browserLanguage (1)
[*] navigator.buildID (1)
[*] navigator.cpuClass (1)
[*] navigator.language (1)
[*] navigator.oscpu (1)
[*] navigator.platform (1)
[*] navigator.product (1)
[*] navigator.productSub (1)
[*] navigator.systemLanguage (1)
[*] navigator.userAgent (2)
[*] navigator.userLanguage (1)
[*] navigator.userProfile (1)
[*] navigator.vendor (1)
[*] navigator.vendorSub (1)
[*] custom.scripting (1)
[*] custom.property (1)

Chrome (320)
[*] navigator.appName (1)
[*] navigator.appCodeName (1)
[*] navigator.appVersion (136)
[*] navigator.appMinorVersion (1)
[*] navigator.browserLanguage (1)
[*] navigator.buildID (1)
[*] navigator.cpuClass (1)
[*] navigator.language (1)
[*] navigator.oscpu (1)
[*] navigator.platform (3)
[*] navigator.product (1)
[*] navigator.productSub (1)
[*] navigator.systemLanguage (1)
[*] navigator.userAgent (136)
[*] navigator.userLanguage (1)
```

```
[*] navigator.userProfile (1)
[*] navigator.vendor (1)
[*] navigator.vendorSub (1)
[*] custom.scripting (1)
[*] custom.property (2)

Droid (2)
[*] navigator.appName (1)
[*] navigator.appCodeName (1)
[*] navigator.appVersion (2)
[*] navigator.appMinorVersion (1)
[*] navigator.browserLanguage (1)
[*] navigator.buildID (1)
[*] navigator.cpuClass (1)
[*] navigator.language (1)
[*] navigator.oscpu (1)
[*] navigator.platform (1)
[*] navigator.product (1)
[*] navigator.productSub (1)
[*] navigator.systemLanguage (1)
[*] navigator.userAgent (2)
[*] navigator.userLanguage (1)
[*] navigator.userProfile (1)
[*] navigator.vendor (2)
[*] navigator.vendorSub (1)
[*] custom.scripting (1)
[*] custom.property (1)

Firefox (521)
[*] navigator.appName (1)
[*] navigator.appCodeName (1)
[*] navigator.appVersion (18)
[*] navigator.appMinorVersion (1)
[*] navigator.browserLanguage (1)
[*] navigator.buildID (339)
[*] navigator.cpuClass (1)
[*] navigator.language (13)
[*] navigator.oscpu (11)
[*] navigator.platform (8)
[*] navigator.product (1)
[*] navigator.productSub (184)
[*] navigator.systemLanguage (1)
[*] navigator.userAgent (471)
[*] navigator.userLanguage (1)
[*] navigator.userProfile (1)
[*] navigator.vendor (10)
[*] navigator.vendorSub (57)
[*] custom.scripting (3)
[*] custom.property (4)
```

Figure 4-2: Collection Analysis Script

At the end of the script execution two files were produced, 1) a file which contains all of the different values which were seen for that field (given a .values

extension), and a file which contains all of the different variants organized by value (given a .map extension).

5. Results

By analyzing the output of the analysis scripts, a number of fingerprints have been identified which allow the identification of various browsers, even despite some basic attempts at changing the browser's identity. Some browsers can only be identified by their families, while others provide enough information to uniquely identify the browser, the browser's version, the O/S, the O/S's version and the processor architecture.

5.1. Microsoft Internet Explorer

```
navigator.userAgent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1;
Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR
3.0.30729; Media Center PC 6.0)
navigator.appName: Microsoft Internet Explorer
navigator.appCodeName: Mozilla
navigator.appVersion: 4.0 (compatible; MSIE 8.0; Windows NT 6.1;
Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR
3.0.30729; Media Center PC 6.0)
navigator.appMinorVersion: 0
navigator.browserLanguage: en-us
navigator.cpuClass: x86
navigator.systemLanguage: en-us
navigator.language: undefined
navigator.buildID: undefined
navigator.oscpu: undefined
navigator.platform: Win32
navigator.product: undefined
navigator.productSub: undefined
navigator.userLanguage: en-us
navigator.userProfile: undefined
navigator.vendor: undefined
navigator.vendorSub: undefined
custom.scripting: Javascript/1.3 JScript/5.8 JScript/16385 VBScript/5.8
VBScript/16385
custom.property: d.all d.childNodes d.compatMode d.documentMode
d.getElementById !d.getElementsByClassName !n.savePreferences
w.XMLHttpRequest !w.globalStorage w.postMessage
```

Figure 5.1-1: Internet Explorer 8.0 on Windows 7 Ultimate, x86 Processor

Microsoft Internet Explorer contains a number of features that allows it to be uniquely distinguished from other browsers. By searching for either 'MSIE' or 'Trident/' in the navigator.userAgent property Internet Explorer can be identified. The 'Trident/' token is only present in Internet Explorer 7 or higher. Unlike all of the other browsers

that had signatures collected, it is the only one that supports JavaScript, JScript and VBScript. Only JavaScript versions up to version 1.3 are supported by Internet Explorer.

Unlike JavaScript, the version of JScript and VBScript cannot be specified as part of the script tag. The Microsoft JScript and VBScript environments provide the ScriptEngineMajorVerion, ScriptEngineMinorVersion and ScriptEngineBuildVersion properties, which can be used to obtain detailed information about the script engine. The version of the script engine will be either be 5.6, 5.7 or 5.8 for Internet Explorer 6.0, either 5.7 or 5.8 for Internet Explorer 7.0, 5.8 for Internet Explorer 8.0, and 9.0 for Internet Explorer 9.0 preview and beta. The build version of the script can be used to identify specific O/S variants. As an example, version 16385 is Windows 7, while 6000 is Windows Vista. This functionality was previously described at BlackHat USA 2009 presentation by a developer of the Metasploit Framework (Lee, 2009).

The navigator.browserLanguage, navigator.systemLanguage, and navigator.userLanguage provide information about the language being used by the browser and O/S. The property returns a 2 character lower case language code, followed by a country code with a hyphen (-) for a separator. Internet Explorer 6, 7, 8 and 9 Beta all return lower case country codes, while the Internet Explorer 9.0 Previews return an upper case country code.

The navigator.appMinorVersion property returns a value of '0', with two exceptions for Internet Explorer 6.0 and Internet Explorer 9.0 beta. Internet Explorer 6.0 returns the service pack that is installed on the O/S, and Internet Explorer 9.0 beta returns the value of 'beta'. In addition to the navigator.appMinorVersion, navigator.userProfile can also be used to identify Internet Explorer 6.0. The navigator.userProfile Internet Explorer 6.0 is not undefined.

Unlike most other browsers which return 'Netscape' in the navigator.appName property, Internet Explorer returns 'Microsoft Internet Explorer'. All version of Internet Explorer that was tested returned this value.

Information about the processor architecture is contained within the navigator.cpuClass and navigator.platform properties. 32-bit (x86) and 64-bit (x86-64) architectures will return 'x86' for navigator.cpuClass and 'Win32' for navigator.platform.

If the system is a 64-bit system, the 'WOW64' token will be added to the navigator.userAgent property.

Internet Explorer has some ability to change its navigator.userAgent by editing the windows registry (Microsoft). The HKEY_LOCAL_MACHINE\ Software\Microsoft\Windows\CurrentVersion\Internet Settings\5.0\User Agent may contain the 'Version' key of type REG_SZ. This key can be changed to modify the navigator.userAgent, but it only replaces the MSIE token with the value entered. There are a number of other tokens that can be added or modified to change individual aspects of the user agent, but these values only change individual tokens and not the behavior of the browser.

5.2. Mozilla Firefox

```
navigator.userAgent: Mozilla/5.0 (X11; U; Linux i686; en-US;
rv:1.9.2.7) Gecko/20100720 Fedora/3.6.7-1.fc13 Firefox/3.6.7
navigator.appName: Netscape
navigator.appCodeName: Mozilla
navigator.appVersion: 5.0 (X11; en-US)
navigator.appMinorVersion: undefined
navigator.browserLanguage: undefined
navigator.cpuClass: undefined
navigator.systemLanguage: undefined
navigator.language: en-US
navigator.buildID: 20100720105013
navigator.oscpu: Linux i686
navigator.platform: Linux i686
navigator.product: Gecko
navigator.productSub: 20100720
navigator.userLanguage: undefined
navigator.userProfile: undefined
navigator.vendor: Fedora
navigator.vendorSub: 3.6.7-1.fc13
custom.scripting: Javascript/1.8.1
custom.property: !d.all d.childNodes d.compatMode !d.documentMode
d.getElementById d.getElementsByClassName !n.savePreferences
w.XMLHttpRequest w.globalStorage w.postMessage
```

Figure 5.2-1: Mozilla Firefox 3.6.7 on Fedora 13, x86-64 processor.

Mozilla Firefox browser contains a number of characteristics which allow it to easily be identified, ranging from simply searching the navigator.userAgent for 'Firefox/' to more subtle information returned from browser specific properties. It does not provide any support for Microsoft JScript or VBScript, but Firefox 2.0.0.x will process JavaScript

up to version 1.7, Firefox 3.0.x will process up to JavaScript version 1.8, while Firefox 3.5 and higher will support up to JavaScript version 1.8.1.

Mozilla Firefox has the navigator.language property which provides information about the browser's language. The format of this field is always a 2 character lower case language code, and if a country is specified, it will be in uppercase separated by a hyphen (-) character.

Mozilla Firefox provides some basic information about the O/S through the navigator.oscpu, navigator.platform, navigator.vendor and navigator.vendorSub properties. The navigator.oscpu properties provides more detailed information about the O/S that the navigator.platform. As an example, Firefox running on Windows XP will return 'Windows NT 5.1' for navigator.oscpu and 'Win32' for navigator.platform. Some GNU/Linux distributions have modified navigator.vendor and navigator.vendorSub to return specific information about the browser and O/S; the default binaries from the Mozilla Foundation's website return an empty string (instead of being an undefined property). As an example, the version of Firefox provided by the Fedora Project will return 'Fedora' for navigator.vendor and '3.6.2-1.fc13' for navigator.vendorSub while the version of Firefox provided by Ubuntu 10.04 will return 'Ubuntu' for navigator.vendor and '10.04' for navigator.vendorSub.

Mozilla Firefox browser has the potential to provide extensive information about the version of the browser based upon the navigator.buildID and navigator.productSub. The navigator.buildID represents when the date/time of browser is complied, with Firefox 2.0.0.x using the YYYYMMDDHH format, and Firefox 3+ using the YYYYMMDDHHMMSS format. It appears that Firefox is built from source sequentially so the build times between the three major O/S families can be identified. Also, if a GNU/Linux distribution takes the source code and complies it, the time at which the source is compiled is unlikely to match that of the Mozilla foundation (and will vary between the various GNU/Linux distributions) allowing for the distribution to be identified. As the distributions compile different versions for the 32-bit and 64-bit versions of the distribution, there can be differences which allow the architecture to be identified. The navigator.productSub property is similar to the navigator.buildID

property in that it has date/time characteristics which are helpful in identifying the browser. Firefox 2.0.0.x uses the YYYYMMDDHH format, while Firefox 3+ uses the YYYYMMDD format. Although the resolution on the navigator.productSub is not very granular, it may not allow individual O/S variants to be identified it will most likely allow the version of Firefox to be identified.

There are two basic ways in which to customize Firefox to pretend to be a different browser; setting the 'general.useragent.extra.firefox' property within about:config or using a browser extension like User Agent Switcher.

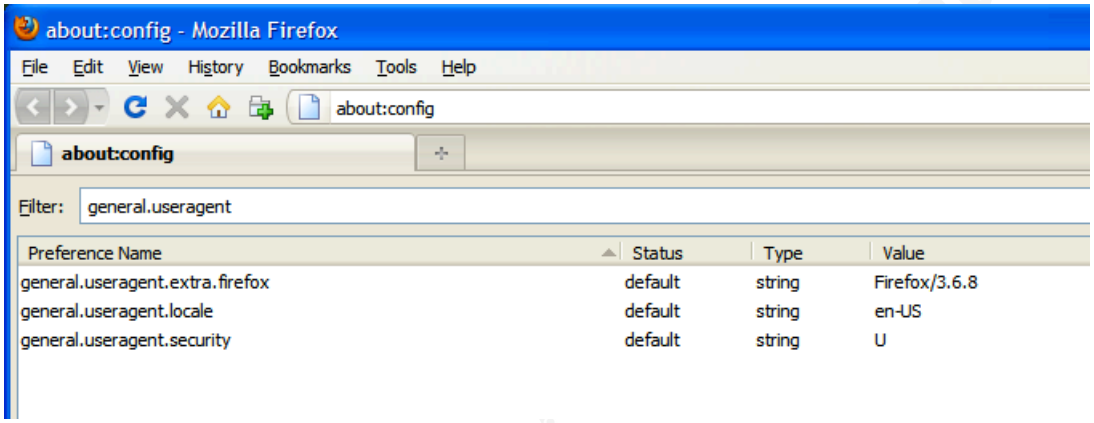

Figure 5.2-2: Built-In Firefox Spoofing

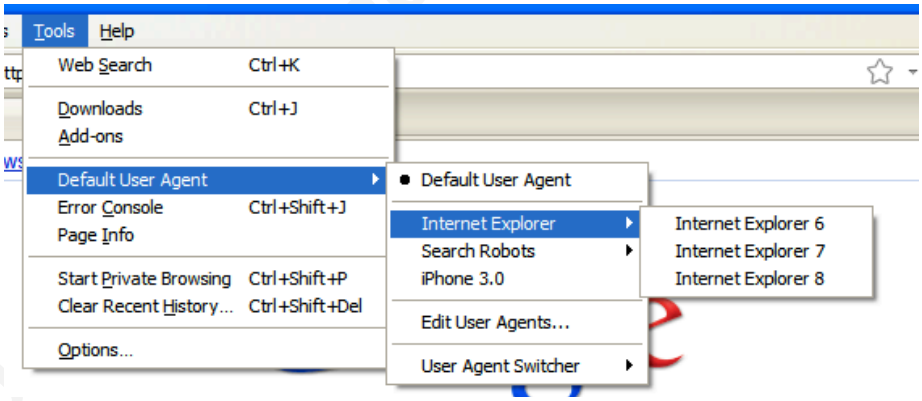

Figure 5.2-3: Spoofing Browsers with User Agent Switcher

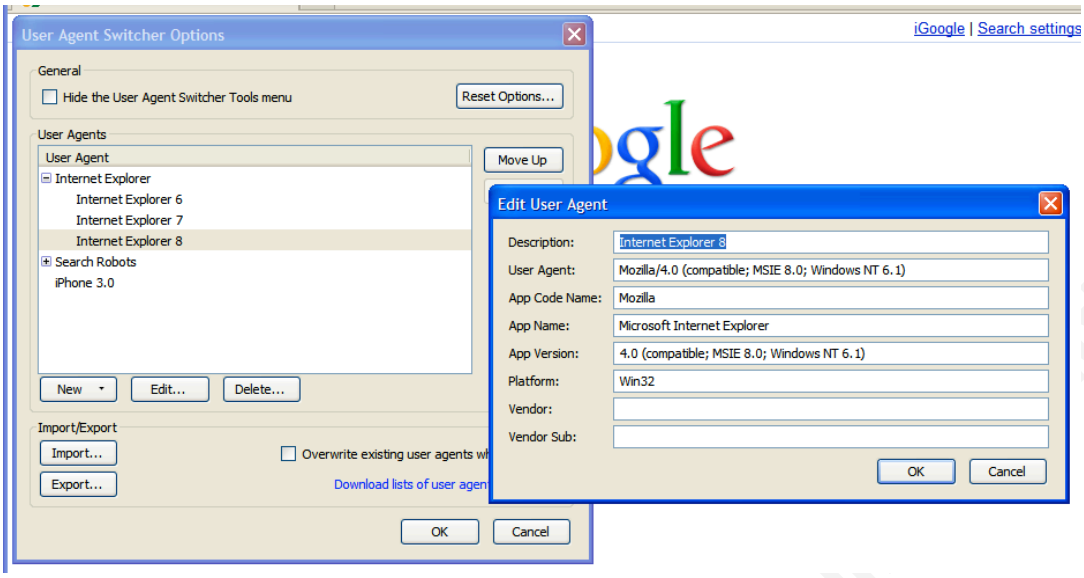

Figure 5.2-4: Editing Specific Browser Properties with User Agent Switcher

Unfortunately neither of these methods changes the versions of JavaScript that the browser will execute, nor will they provide the browser with the ability to process Jscript or VBScript. The first method only allows the Firefox/Version portion of the navigator.userAgent property to be modified; it does not change any other properties. The second method, using the User Agent Switcher extension, allows a number of properties to be modified but it does not include the functionality to change the navigator.buildID or navigator.productSub which can still be used to potentially identify the browser version, O/S, and processor architecture.

5.3. Apple Safari

```
navigator.userAgent: Mozilla/5.0 (iPhone; U; CPU iPhone OS 4_0_1 like
Mac OS X; en-us) AppleWebKit/532.9 (KHTML, like Gecko) Version/4.0.5
Mobile/8A306 Safari/6531.22.7
navigator.appName: Netscape
navigator.appCodeName: Mozilla
navigator.appVersion: 5.0 (iPhone; U; CPU iPhone OS 4_0_1 like Mac OS
X; en-us) AppleWebKit/532.9 (KHTML, like Gecko) Version/4.0.5
Mobile/8A306 Safari/6531.22.7
navigator.appMinorVersion: undefined
navigator.browserLanguage: undefined
navigator.cpuClass: undefined
navigator.systemLanguage: undefined
navigator.language: en-us
navigator.buildID: undefined
navigator.oscpu: undefined
navigator.platform: iPhone
navigator.product: Gecko
navigator.productSub: 20030107
```

```
navigator.userLanguage: undefined
navigator.userProfile: undefined
navigator.vendor: Apple Computer, Inc.
navigator.vendorSub: dne
custom.scripting: Javascript/1.7 JScript/0.0
custom.property: !d.all d.childNodes d.compatMode !d.documentMode
d.getElementById d.getElementsByClassName !n.savePreferences
w.XMLHttpRequest !w.globalStorage w.postMessage
```

Figure 5.3-1: Apple Safari 4.0.5 on iPhone 3G with iOS 4.0.1

The Apple Safari browser provides a number of properties which allow it to be identified, but this identification can be limited depending upon how the properties are modified. As Safari and Google Chrome are based upon WebKit for rendering the pages, the differences between the Safari and Google Chrome is reduced as compared to the differences between Safari and other browsers such as Microsoft Internet Explorer or Mozilla Firefox. Safari will execute scripts with a version up to and including JavaScript version 1.7. Also Safari will execute Microsoft JScript as though it was JavaScript until it encounters a Microsoft specific function.

Specific information about the O/S is included in navigator.appVersion and navigator.userAgent. These two JavaScript properties will contain the 'Safari/', 'AppleWebKit/' and 'Version/' tokens each of which is followed by a version identifier. If Safari is running on an Apple device such as an iPad, iPhone, or an iPod Touch it will also contain a 'Mobile/' token which contains a device identifier.

Safari has the navigator.language property which provides information about the browser's language. The format of this field is always a 2 character lower case language code, and if a country is specified with will be in separated by a hyphen (-) character. If the country code is uppercase, then the browser is running on Microsoft Windows, otherwise if the country code's characters are all lower case it is running on an Apple platform (Mac OS X, iPhone, iPad, etc).

The navigator.platform provides additional information about the device. It will return values of iPad (e.g. an iPad), iPhone (e.g. an iPhone, any variant), iPod (e.g. iPod Touch), MacIntel (e.g. Apple Mac OS X with an Intel processor) or Win32 (e.g. Microsoft Windows O/S).

The navigator.productSub and navigator.vendor JavaScript properties are populated with WebKit specific values. The navigator.productSub contains the date '20030107', while navigator.vendor contains 'Apple Computer, Inc.'

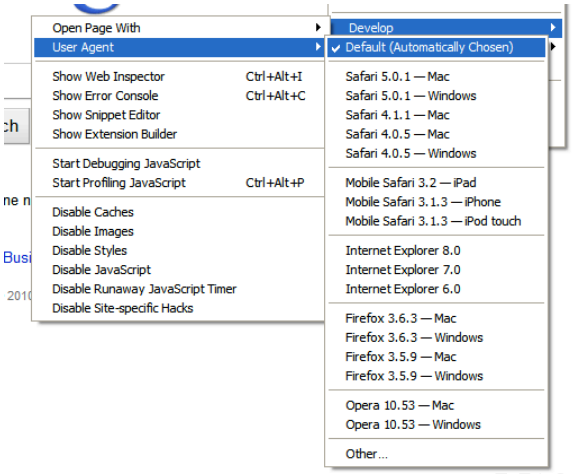

Figure 5.3-2: Built-In Apple Safari Browser Spoofing

Safari contains developer tools which allow the browser to alter its navigator.userAgent and navigator.appVersion properties to be that of another browser/O/S combination (Apple, 2010). These changes do not affect the other object properties like version of JavaScript processed, available DOM objects or the navigator objects like navigator.platform or navigator.vendorSub. The modifications to masquerade as a version of Microsoft Internet Explorer are also limited in that there are no .Net framework identifiers supplied and most versions of Microsoft Windows include at least one version of the .Net framework.

5.4. Google Chrome

```
navigator.userAgent: Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US)
AppleWebKit/534.3 (KHTML, like Gecko) Chrome/6.0.472.0 Safari/534.3
navigator.appName: Netscape
navigator.appCodeName: Mozilla
navigator.appVersion: 5.0 (Windows; U; Windows NT 6.1; en-US)
AppleWebKit/534.3 (KHTML, like Gecko) Chrome/6.0.472.0 Safari/534.3
navigator.appMinorVersion: undefined
navigator.browserLanguage: undefined
navigator.cpuClass: undefined
navigator.systemLanguage: undefined
navigator.language: en-US
navigator.buildID: undefined
navigator.oscpu: undefined
navigator.platform: Win32
navigator.product: Gecko
```

```
navigator.productSub: 20030107
navigator.userLanguage: undefined
navigator.userProfile: undefined
navigator.vendor: Google Inc.
navigator.vendorSub: dne
custom.scripting: Javascript/1.7 JScript/0.0
custom.property: !d.all d.childNodes d.compatMode !d.documentMode
d.getElementById d.getElementsByClassName !n.savePreferences
w.XMLHttpRequest !w.globalStorage w.postMessage
```

Figure 5.4-1: Google Chrome 6.0.472.0 on Windows 7 Ultimate, x86 Processor

Google Chrome is similar to Apple Safari in the types and values of the browser properties that are available. Both browsers make use of the WebKit libraries which contributes to these similarities but there are some differences. Google Chrome will execute scripts with a version up to and including JavaScript version 1.7. Also Google Chrome will execute Microsoft JScript as though it was JavaScript until it encounters a Microsoft specific function.

Specific information about the O/S is included in the values of the navigator.appVersion and navigator.userAgent properties. These two JavaScript properties will contain the 'Safari/' and 'AppleWebKit/' tokens each of which is followed by a version identifier, similar to Safari but unlike Safari the 'Version/' is absent and a 'Chrome/' token is included. Some Linux distributions will include other tokens, like Sabayon which includes a 'Sabayon' token or Ubuntu which includes 'Ubuntu/[Version]' token in these two properties.

Google Chrome utilizes the navigator.language property to provide information about the browser's language. The format of this field is always a 2 character lower case language code, and if a country is specified, it will be separated by a hyphen (-) character with an uppercase country code.

The navigator.platform provides additional information about the O/S. It will return values of Linux i686 (e.g. GNU/Linux O/S), Win32 (e.g. Microsoft Windows O/S) or MacIntel (e.g. Apple OS X O/S). Unlike other browsers, if the processor architecture is x86-64, the navigator property will still return the 32-bit identifiers (e.g. Linux i686 and Win32).

The navigator.productSub and navigator.vendor JavaScript properties are populated with values similar to that of Apple Safari. The navigator.productSub contains

the date '20030107' like Safari, but navigator.vendor property's value instead returns 'Google, Inc.'

Google Chrome also includes some Chrome specific objects; like window.google, window.chrome, and window.chromium. The window.chrome object only exists for versions of Google Chrome 3+.

Google Chrome provides some built in functionality for attempting to masquerade as a different browser (Google, 2008). If Google Chrome is started from the command line, with the --user-agent="Desired User Agent" the browser will use that User Agent string for the duration of the session. Once the browser is restarted, the User Agent will revert to the real value. These changes do not affect the other object properties like version of JavaScript processed, available DOM objects, the navigator objects like navigator.platform or navigator.vendorSub, or the Google Chrome specific browser objects (window.google, window.chrome, and/or window.chromium).

5.5. ASA Software Opera

```
navigator.userAgent: Opera/9.80 (Macintosh; Intel Mac OS X; U; en)
Presto/2.6.30 Version/10.61
navigator.appName: Opera
navigator.appCodeName: Mozilla
navigator.appVersion: 9.80 (Macintosh; Intel Mac OS X; U; en)
navigator.appMinorVersion: dne
navigator.browserLanguage: en
navigator.cpuClass: undefined
navigator.systemLanguage: undefined
navigator.language: en
navigator.buildID: undefined
navigator.oscpu: undefined
navigator.platform: MacIntel
navigator.product: undefined
navigator.productSub: undefined
navigator.userLanguage: en
navigator.userProfile: undefined
navigator.vendor: undefined
navigator.vendorSub: undefined
custom.scripting: Javascript/2 JScript/0.0
custom.property: Opera/10.61 Opera/8429 !d.all d.childNodes
d.compatMode !d.documentMode d.getElementById d.getElementsByClassName
!n.savePreferences w.XMLHttpRequest !w.globalStorage w.postMessage
```

Figure 5.5-1: ASA Opera 10.61 on Apple Mac OS X 10.6.4

ASA Software's Opera Browser also provides a number of functions and includes a number of different behaviors to allow it to be uniquely identified. The Opera Browser

will process JavaScript and like Safari and Google Chrome it will attempt to process Microsoft JScript with its JavaScript processor until a Microsoft Internet Explorer unique function is encountered. Opera also is compliant with RFC4329 which obsoletes the JavaScript version identifier associated with the script tag, so it will process all JavaScript found on the page regardless of the JavaScript version associated with the script.

The Opera Browser also returns fairly unique value for the navigator.appName property, 'Opera'. The navigator.browserLanguage, navigator.language, and navigator.userLanguage provide information about the language is being used by the browser and O/S. The property returns a 2 character lower case language code with no associated country codes.

Besides navigator.appVersion and navigator.userAgent, Opera only provides information in the navigator.platform property to return information about the O/S. Typically values returned by the navigator.platform property include: FreeBSD, Linux, MacIntel, and Win32. Specific information about the O/S and/or processor architecture is only available from within the navigator.appVersion and navigator.userAgent properties. As an example, both 32-bit and 64-bit Linux O/Ss will have "X11; Linux i686;" or "X11; Linux x86_64;" tokens in the navigator.appVersion and navigator.userAgent properties but navigator.platform will only contain "Linux". This feature also helps to distinguish it from other browsers, in that other browsers will commonly return something similar to "Linux i686" or "Linux amd" (e.g. Linux with a Processor Architecture identifier).

Opera also provides the window.opera object which scripts can access additional information and make use of specific functionality within the browser. This functionality was previously described at BlackHat USA 2009 presentation by a developer of the Metasploit Framework (Lee, 2009). Specifically the window.opera.buildNumber() and window.opera.version() functions return specific information about the browser. The window.opera.version() function returns the version of opera that is being used (i.e. "10.10" or "10.61"), while the window.opera.buildNumber() function returns the build number of the browser (i.e. "6386", "8402", or "3445" for Opera 10.61). These values can be used to identify specific O/Ss. For example, Opera 10.61 with a Microsoft

Windows O/S will have a build number of 3445, Opera 10.61 with Apple Mac OS X will have a build number of 6386, while Opera 10.61 with a Linux O/S will have build number of 6386. The differences in the build numbers are no longer being supported by ASA Software, and all O/S variants will now utilize the same build number for a given version of Opera (2010, Aleksandersen).

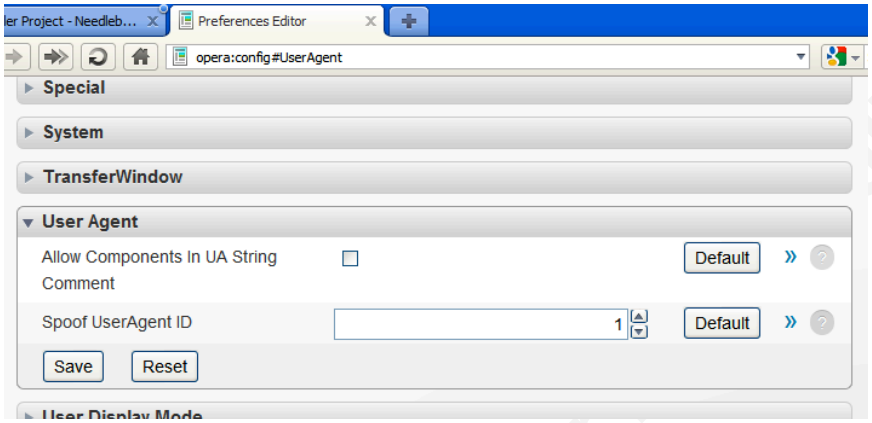

Figure 5.5-2: Built-In Opera Browser Spoofing

Opera provides built-in functionality to allow it to masquerade as other browsers. By default a user can change to browser to appear to be Microsoft Internet Explorer or Mozilla Firefox. This functionality can be accessed by entering about:config in the URL. When the masquerading feature is enabled the navigator.appName, navigator.appVersion and navigator.userAgent properties are updated to match that of the desired browser. Depending upon how the masquerading function is configured, Opera may or may not include an Opera version number at the end of the navigator.appVersion and navigator.userAgent properties. Beyond updating the values of those fields, no other changes to the browsers functionality are changed. It still allows the properties of the window.opera object to be accessed and it still processes all version of JavaScript and attempts to process Microsoft JScript.

5.6. Konqueror

```
navigator.userAgent: Mozilla/5.0 (compatible; Konqueror/4.4; Linux)
KHTML/4.4.1 (like Gecko)
navigator.appName: Netscape
navigator.appCodeName: Mozilla
navigator.appVersion: 5.0 (compatible; Konqueror/4.4; Linux)
KHTML/4.4.1 (like Gecko)
navigator.appMinorVersion: undefined
navigator.browserLanguage: en_US
```

```
navigator.cpuClass: x86_64
navigator.systemLanguage: undefined
navigator.language: en_US
navigator.buildID: undefined
navigator.oscpu: undefined
navigator.platform: Linux x86_64
navigator.product: Gecko
navigator.productSub: 20030107
navigator.userLanguage: en_US
navigator.userProfile: undefined
navigator.vendor: KDE
navigator.vendorSub: dne
custom.scripting: Javascript/1.5 JScript/0.0
custom.property: !d.all d.childNodes d.compatMode !d.documentMode
d.getElementById d.getElementsByClassName !n.savePreferences
w.XMLHttpRequest !w.globalStorage !w.postMessage
```

Figure 5.6-1: Konqueror 4.4.1 on Ubuntu 10.04, x86-64 processor

The K Desktop Environment (KDE) Konqueror browser is available for Linux and UNIX O/Ss as part of the KDE desktop. Konqueror will execute scripts with a version up to and including JavaScript version 1.5. Also Konqueror will execute Microsoft JScript as though it was JavaScript until it encounters a Microsoft specific function. Konqueror does not display a large amount of variance between the sub versions of 3.x.x or 4.x.x, but there are a number of differences between version 3.x and 4.x of the browser.

The navigator.appName property contains the values of either 'Konqueror' or 'Netscape'. 'Konqueror' is returned for 3.x, while 'Netscape' is returned for 4.x.

The navigator.browserLanguage, navigator.language, and navigator.userLanguage provide information about the language is being used by the browser and O/S. The format of this field is always a 2 character lower case language code, and an uppercase country code. Unlike other browsers, the language code and country code are separated with an underscore (_) character.

The navigator.cpuClass and navigator.platform provide some information about the O/S. Konqueror on PC-BSD will return 'i386' or 'amd64' depending on the processor architecture, while Konqueror on GNU/Linux will return either 'i686' or 'x86_64' depending on the processor architecture. The returned values of the navigator.platform property are similar, but they allow the general version of Konqueror to be determined on GNU/Linux. Konqueror on PC-BSD will return 'FreeBSD i386' or

'FreeBSD amd64' depending upon the processor architecture. Konqueror on GNU/Linux will return one of four different values (e.g. 'Linux i686', 'Linux i686 X11', 'Linux x86_64', 'Linux x86_64 X11') depending upon the processor architecture and Konqueror version. Konqueror 3.x will include 'X11' after the processor architecture.

The navigator.product and navigator.productSub contain information will allow Konqueror 3.x browsers to be distinguished from Konqueror 4.x browsers. In general Konqueror 4.x provides values more similar to those of other browsers. Konqueror 3.x will contain values of 'Konqueror/khtml' for navigator.product and '20040107' for navigator.productSub, while Konqueror 4.x will contain values of 'Gecko' for navigator.product and '20030107' for navigator.productSub.

The navigator.vendor property contains returns the value of 'KDE' while will allow it to easily distinguish it from other browsers. The possible states of the navigator.vendorSub property provide another way in which Konqueror 3.x can be distinguished from Konqueror 4.x. In Konqueror 3.x, navigator.vendorSub is an undefined function, while in Konqueror 4.x the property is defined but returns and empty string.

5.7. Android

```
navigator.userAgent: Mozilla/5.0 (Linux; U; Android 2.2; en-us; Sprint
APA9292KT Build/FRF91) AppleWebKit/533.1 (KHTML, like Gecko)
Version/4.0 Mobile Safari/533.1
navigator.appName: Netscape
navigator.appCodeName: Mozilla
navigator.appVersion: 5.0 (Linux; U; Android 2.2; en-us; Sprint
APA9292KT Build/FRF91) AppleWebKit/533.1 (KHTML, like Gecko)
Version/4.0 Mobile Safari/533.1
navigator.appMinorVersion: undefined
navigator.browserLanguage: undefined
navigator.cpuClass: undefined
navigator.systemLanguage: undefined
navigator.language: en
navigator.buildID: undefined
navigator.oscpu: undefined
navigator.platform: Linux armv7l
navigator.product: Gecko
navigator.productSub: 20030107
navigator.userLanguage: undefined
navigator.userProfile: undefined
navigator.vendor: Google Inc.
navigator.vendorSub: dne
custom.scripting: Javascript/1.7 JScript/0.0
```

```
custom.property: !d.all d.childNodes d.compatMode !d.documentMode
d.getElementById d.getElementsByClassName !n.savePreferences
w.XMLHttpRequest !w.globalStorage w.postMessage
```

Figure 5.7-1: Android 2.2 HTC Evo (Sprint)

The embedded Android O/S includes a browser to allow users to navigate the Internet, and this browser is very similar to the Google Chrome browser which it is based upon. Android browser will execute scripts with a version up to and including JavaScript version 1.7. Also the Android browser will execute Microsoft JScript as though it was JavaScript until it encounters a Microsoft specific function.

The navigator.appVersion and navigator.userAgent provide some information about the device. These two properties include tokens to specify that it is an Android device along with the version (e.g. 'Android 2.1-update1'), a token to specify that it is a mobile device (e.g. 'Mobile') and a browser version token (e.g. 'Version/').

The navigator.language property provides information about the language that the device is configured to support. The value of the property is the two character lower case language code, but unlike Google Chrome the value does not include a country identifier. The navigator.language property does not match the language token listed in the navigator.appVersion and navigator.userAgent property. On Android embedded devices, the language token in the navigator.appVersion and navigator.userAgent properties contain the language code and the country code, while navigator.language only contains the language code.

The navigator.platform property contains the value of 'Linux armv71' which is the processor of the device. The desktop version of Google Chrome will return 'Linux i686' for a processor type.

The navigator.vendor property returns a value of 'Google Inc.' for the desktop versions of Google Chrome, but the browser of the embedded Android devices displays some variance in the values that are returned. Android devices will return values of either 'Google Inc.' or 'Apple Computer, Inc.' for navigator.vendor.

Unlike the desktop version of Google Chrome, Android devices do not include native functionality to modify their User Agent, so it is more difficult for these devices to masquerade as other browsers and/or O/Ss.

5.8. Blackberry

```
navigator.userAgent: BlackBerry9630/5.0.0.732 Profile/MIDP-2.1
Configuration/CLDC-1.1 VendorID/105
navigator.appName: Netscape
navigator.appCodeName: Mozilla
navigator.appVersion: 5.0.0.732 Profile/MIDP-2.1 Configuration/CLDC-1.1
VendorID/105
navigator.appMinorVersion: undefined
navigator.browserLanguage: undefined
navigator.cpuClass: undefined
navigator.systemLanguage: undefined
navigator.language: en
navigator.buildID: Today's
navigator.oscpu: undefined
navigator.platform: BlackBerry
navigator.product: Gecko
navigator.productSub: undefined
navigator.userLanguage: undefined
navigator.userProfile: undefined
navigator.vendor: undefined
navigator.vendorSub: undefined
custom.scripting: Javascript/1.3
custom.property: !d.all d.childNodes d.compatMode !d.documentMode
d.getElementById !d.getElementsByClassName !n.savePreferences
w.XMLHttpRequest !w.globalStorage !w.postMessage
```

Figure 5.8-1: BlackBerry Browser on BlackBerry 9630 (Verizon)

The BlackBerry browser can easily be identified by the navigator.userAgent but beyond the navigator.userAgent there are a number of other properties, which can be used to uniquely identify the browser. Blackberry browser will execute scripts with a version up to and including JavaScript version 1.3, and unlike some of the other browsers that were fingerprinted only JavaScript is executed and neither Microsoft VBScript or Microsoft JScript were executed.

A number of navigator properties which other browsers populated, only a few items were defined within the Blackberry scripting environment. The navigator.language property contains the device's language listed as a 2 character lowercase language code. The navigator.language property does not return a country code within the language field, only a language code. Unlike most of the other browsers, the navigator.buildID is defined, but it returns the unique value of "Today's", and the navigator.platform value is also unique in that it returns the value of "BlackBerry".

Similar to the other mobile device browsers that were fingerprinted, the Blackberry browser does not contain functionality to allow it to masquerade as another

browser. As a significant number of navigator properties are undefined, it will be difficult for other browsers to masquerade as a Blackberry device due to the presence of their defined defaults.

5.9. ASA Software OperaMini

```
navigator.userAgent: Opera/9.80 (iPhone; Opera Mini/5.0.019802/19.892;
U; en) Presto/2.5.2525
navigator.appName: Opera
navigator.appCodeName: Mozilla
navigator.appVersion: 9.80 (iPhone; Opera Mini/5.0.019802/19.892; U;
en)
navigator.appMinorVersion: dne
navigator.browserLanguage: en
navigator.cpuClass: undefined
navigator.systemLanguage: undefined
navigator.language: en
navigator.buildID: undefined
navigator.oscpu: undefined
navigator.platform: Pike v7.6 release 92
navigator.product: undefined
navigator.productSub: undefined
navigator.userLanguage: en
navigator.userProfile: undefined
navigator.vendor: undefined
navigator.vendorSub: undefined
custom.scripting: Javascript/2 JScript/0.0
custom.property: Opera/10.00 Opera/892 !d.all d.childNodes d.compatMode
!d.documentMode d.getElementById d.getElementsByClassName
!n.savePreferences w.XMLHttpRequest !w.globalStorage w.postMessage
```

Figure 5.9-1: ASA Software OperaMini on iPhone 3G with iOS4.0.2

The ASA Software OperaMini Browser is similar to the Opera Browser except that it is designed for mobile platforms. OperaMini is very similar to the Opera browser, but there are a small number of notable exceptions. The version that is listed in the navigator.appVersion and navigator.userAgent is different from the value that is returned by the window.opera.version() function. The OperaMini's navigator.platform is different from the desktop version of the Opera browser, as it returns a value of 'Pike v7.6 release 92'. The OperaMini browser does not provide support to masquerade as a different browser.

5.10. ASA Software Opera Mobile

```
navigator.userAgent: Opera/9.80 (Linux armv7l; Maemo; Opera Mobi/4; U;
en-GB) Presto/2.5.28 Version/10.1
navigator.appName: Opera
navigator.appCodeName: Mozilla
```

```
navigator.appVersion: 9.80 (Linux armv7l; Maemo; Opera Mobi/4; U; en-
GB)
navigator.appMinorVersion: dne
navigator.browserLanguage: en-GB
navigator.cpuClass: undefined
navigator.systemLanguage: undefined
navigator.language: en-GB
navigator.buildID: undefined
navigator.oscpu: undefined
navigator.platform: Linux
navigator.product: undefined
navigator.productSub: undefined
navigator.userLanguage: en-GB
navigator.userProfile: undefined
navigator.vendor: undefined
navigator.vendorSub: undefined
custom.scripting: Javascript/2 JScript/0.0
custom.property: Opera/10.1 Opera/4 !d.all d.childNodes d.compatMode
!d.documentMode d.getElementById d.getElementsByClassName
!n.savePreferences w.XMLHttpRequest !w.globalStorage w.postMessage
```

Figure 5.10-1: ASA Software Opera Mobi 4 Nokia N900 with Maemo 5

The ASA Software Opera Mobile Browser is similar to the desktop version of the Opera Browser except that it is designed for mobile platforms. Like the OperaMini, the Opera Mobile browser is very similar to the desktop Opera browser. Again the window.opera.buildNumber() function provides a different build number from those that are seen with the desktop and the mini versions of the browser. Information contained within the navigator.appVersion and navigator.userAgent properties has tokens which allow the identification of the Mobile version of the browser (e.g. 'Maemo' and 'Opera Mobi 4' tokens). Unlike the other versions of the Opera browser, the navigator.browserLanguage, navigator.language and navigator.userLanguage contain the language code and the country code of the device. The language code is in lower case, while the country code is in upper case. The two fields are separated by a hyphen (-).

5.11. Maemo Browser (MicroB)

```
navigator.userAgent: Mozilla/5.0 (X11; U; Linux armv7l; en-GB;
rv:1.9.2b6pre) Gecko/20100318 Firefox/3.5 Maemo Browser 1.7.4.8 RX-51
N900
navigator.appName: Netscape
navigator.appCodeName: Mozilla
navigator.appVersion: 5.0 (X11; en-GB)
navigator.appMinorVersion: undefined
navigator.browserLanguage: undefined
navigator.cpuClass: undefined
navigator.systemLanguage: undefined
navigator.language: en-GB
```

```
navigator.buildID: 20091023080530
navigator.oscpu: Linux armv7l
navigator.platform: Linux armv7l
navigator.product: Gecko
navigator.productSub: 20100318
navigator.userLanguage: undefined
navigator.userProfile: undefined
navigator.vendor: Firefox/3.5 Maemo Browser 1.7.4.8 RX-51 N900
navigator.vendorSub: dne
custom.scripting: Javascript/1.8.1
custom.property: !d.all d.childNodes d.compatMode !d.documentMode
d.getElementById d.getElementsByClassName !n.savePreferences
w.XMLHttpRequest w.globalStorage w.postMessage
```

Figure 5.11-1: Maemo Browser on Nokia N900 with Maemo 5

The Maemo Browser (MicroB) is similar to the desktop version of the Mozilla Firefox Browser except that it is designed for mobile platforms. Although the Maemo Browser is very similar to the Firefox browser, there are a small number of notable exceptions. The version that is listed in the navigator.vendor and navigator.userAgent are different from the values that are used on the desktop version. The navigator.vendor contains information about the device such as the model (e.g. 'RX-51 N900') and browser version (e.g. Maemo Browser 1.7.4.8). The navigator.oscpu and navigator.platform properties return values which are characteristics of an ARM processor (e.g. these properties return the value of 'Linux armv71'). Similar to the navigator.buildID of the desktop version of Mozilla Firefox, the value returned by navigator.buildID contains enough resolution to uniquely identify the browser as compared to other Mozilla Firefox browsers.

6. Application

The ability to accurately fingerprint a browser and/or determine the underlying O/S of a system can be integrated into the tools of a Web Application Penetration Tester. Two Free and Open Source Software (FOSS) projects that can benefit from the ability to accurately identify browsers are the Browser Exploitation Framework (BeEF) (Alcorn, 2010) and the Metasploit Framework. BeEF currently relies upon the navigator.userAgent property to determine the O/S of the Browser Zombie. The Metasploit Framework's Browser Autopwn component has been expanded to include a number of the items previously referenced for Microsoft Internet Explorer and ASA

Software's Opera browser (Lee, 2009). Browser Autopwn can be and has been expanded as a result of this fingerprinting project to make use of the Google Chrome and Mozilla Firefox identification techniques previously described.

7. Conclusion

As a result of the collection and analysis of a number of different browser and O/Ss, it was determined that a browser and/or operating system could fairly easily be identified by using the scripting techniques identified in this paper.

Determining the version of the browser can be difficult, as usually it is possible to overwrite the information providing specific details about the browser. In the case of Microsoft Internet Explorer, Mozilla Firefox, and ASA Software's Opera browser sufficient information is available to usually determine the specific browser version being used, even if a level of masquerading is being performed. A combination of the major, minor and build versions of the script engine allow the version of Internet Explorer to be identified, Mozilla Firefox version's can be determined by the navigator.buildID, while ASA Software's Opera browser can be identified by either the opera.buildNumber() or opera.version() functions.

Determining the O/S that the browser is operating within can be a little more difficult, but it is usually possible to at least determine the O/S family that is hosting the browser. Some browsers only operate within specific environments, such as Microsoft Internet Explorer, Apple's Safari Browser or KDE's Konqueror. All of the browsers provided enough information to determine the O/S, while other provided enough information to determine the O/S family. ASA Software's Opera browser provides the opera.buildNumber() function by which the O/S family can be determined. Some exposed more information which allowed specific O/S variants and distributions to be identified or even specific processor architectures. At a minimum Mozilla Firefox allows the O/S family to be identified, while in most cases the navigator.buildID allows the O/S, O/S distribution and even the processor architecture to be determined.

During a Web Application Penetration test which includes client side testing, it is important to correctly identify the browser and O/S of the client before attempting to

exploit it. Incorrect identification can result in a failed exploit that can crash a browser or even the operating system, potentially resulting in the lost data. By correctly identifying a client, exploits can be more carefully selected to reduce the likelihood of collateral damage and increase the likelihood of a successfully compromising a client and possibly successfully manipulating the client to perform the desired actions of the web application penetration tester.

8. References

Alcorn, W. (2010, February 25). *BindShell.Net: browser exploitation framework.* Retrieved from http://www.bindshell.net/tools/beef/

Aleksandersen, D. (2010, August 25). *Bug fixing wednesday on a unified build number.* Retrieved from http://my.opera.com/desktopteam/blog/b9034

Apple Computer, Inc. (2010, June 21). *Safari user guide for developers: prototyping your website.* Retrieved from http://developer.apple.com/safari/library/documentation/appleapplications/conceptual/safari_developer_guide/PrototypingYourWebsite/PrototypingYourWebsite.html

European Computer Manufacturers Association. (1999, December) *Standard ECMA-262: ECMAScript language specification, 3rd Edition.* Retrieved from http://www.ecma-international.org/publications/files/ECMA-ST-ARCH/ECMA-262,%203rd%20edition,%20December%201999.pdf

European Computer Manufacturers Association. (2009, December) *Standard ECMA-262: ECMAScript language specification, 5th Edition.* Retrieved from http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf

GlobalStats. (2010, August). *StatsCounter: top 5 browsers from august 2009 through august 2010.* Retrieved from http://gs.statcounter.com/

Google. (2008, December 30). *Google Chrome: help forum: built-in user agent switcher?* Retrieved from http://www.google.com/support/forum/p/Chrome/thread?tid=64e4e45037f55919&hl=en

Google. *Google DocType: documenting the open web.* Retrieved from http://code.google.com/doctype/

Hoehrmann, B. (2006, April). *Scripting media types.* Retrieved from http://www.rfc-editor.org/rfc/rfc4329.txt

JavaScript Kit. *The language & type attributes of JavaScript: variations of language attribute.* Retrieved from http://www.javascriptkit.com/javatutors/languageattri3.shtml

Lee, J. (2009, August) *Using Guided Missiles in Drive-by's: Automatic browser fingerprinting and exploitation with Metasploit.* Retrieved from http://www.blackhat.com/presentations/bh-usa-09/EGYPT/BHUSA09-Egypt-GuidedMissiles-SLIDES.pdf

Microsoft. *JScript (windows script technologies).* Retrieved from http://msdn.microsoft.com/en-us/library/hbxc2t98(VS.85).aspx

Microsoft. (2009, March). *Version information (windows scripting - jscript).* Retrieved from http://msdn.microsoft.com/en-us/library/s4esdbwz(v=VS.85).aspx

Microsoft. (2008, August). *Microsoft JScript features - ECMA (windows scripting - JScript).* Retrieved from http://msdn.microsoft.com/en-us/library/d33e43t3(v=VS.85).aspx

Microsoft. *Microsoft JScript features - Non-ECMA (windows scripting - JScript).* Retrieved from http://msdn.microsoft.com/en-us/library/4tc5a343(v=VS.85).aspx

Microsoft. *VBScript*. Retrieved from http://msdn.microsoft.com/en-us/library/t0aew7h6.aspx

Microsoft. (2009, March). *VBScript version information.* Retrieved from http://msdn.microsoft.com/en-us/library/4y5y7bh5.aspx

Microsoft. *Understanding user-agent strings.* Retrieved from http://msdn.microsoft.com/en-us/library/ms537503%28VS.85%29.aspx

NetMarketShare. (2010, August). Market share for browsers, operating systems, and search engines. Retrieved from http://marketshare.hitslink.com/report.aspx?qprid=0

Netscape Communications Corporation, Sun Microsystems, Inc. (1995, December 4). *Netscape and Sun announce JavaScript, the open, cross-platform object scripting*

language for enterprise networks and the Internet. Retrieved from http://web.archive.org/web/20070916144913/http://wp.netscape.com/newsref/pr/newsrelease67.html

World Wide Web Consortium. (2005, Janruary 19). *Document object model.* Retrieved from http://www.w3.org/DOM/

W3schools. *HTML <script> type attribute.* Retrieved from http://www.w3schools.com/TAGS/att_script_type.asp

W3schools. *The history object.* Retrieved from http://www.w3schools.com/jsref/obj_history.asp

W3schools. *The location object.* Retrieved from http://www.w3schools.com/jsref/obj_location.asp

W3schools. *The navigator object.* Retrieved from http://www.w3schools.com/jsref/obj_navigator.asp

W3schools. *The screen object.* Retrieved from http://www.w3schools.com/jsref/obj_screen.asp

W3schools. (2010, July). *Browser statistics.* Retrieved from http://www.w3schools.com/browsers/browsers_stats.asp

Zalewski, M. (2009, December). *Browser security handbook.* Retrieved from http://code.google.com/p/browsersec/

Last Updated: July 21st, 2017

Upcoming Training

SANSFIRE 2017	Washington, DC	Jul 22, 2017 - Jul 29, 2017	Live Event
SANS Boston 2017	Boston, MA	Aug 07, 2017 - Aug 12, 2017	Live Event
SANS Prague 2017	Prague, Czech Republic	Aug 07, 2017 - Aug 12, 2017	Live Event
Mentor Session - SEC542	Des Moines, IA	Aug 14, 2017 - Sep 13, 2017	Mentor
SANS Virginia Beach 2017	Virginia Beach, VA	Aug 21, 2017 - Sep 01, 2017	Live Event
SANS Network Security 2017	Las Vegas, NV	Sep 10, 2017 - Sep 17, 2017	Live Event
Community SANS Toronto SEC542	Toronto, ON	Sep 11, 2017 - Sep 16, 2017	Community SANS
SANS Dublin 2017	Dublin, Ireland	Sep 11, 2017 - Sep 16, 2017	Live Event
SANS London September 2017	London, United Kingdom	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS Oslo Autumn 2017	Oslo, Norway	Oct 02, 2017 - Oct 07, 2017	Live Event
SANS vLive - SEC542: Web App Penetration Testing and Ethical Hacking	SEC542 - 201710,	Oct 03, 2017 - Nov 09, 2017	vLive
SANS Tysons Corner Fall 2017	McLean, VA	Oct 14, 2017 - Oct 21, 2017	Live Event
SANS Tokyo Autumn 2017	Tokyo, Japan	Oct 16, 2017 - Oct 28, 2017	Live Event
Community SANS Minneapolis SEC542	Minneapolis, MN	Oct 16, 2017 - Oct 21, 2017	Community SANS
Community SANS New York SEC542	New York, NY	Oct 16, 2017 - Oct 21, 2017	Community SANS
SANS Berlin 2017	Berlin, Germany	Oct 23, 2017 - Oct 28, 2017	Live Event
Community SANS Tampa SEC542	Tampa, FL	Nov 13, 2017 - Nov 18, 2017	Community SANS
Community SANS Austin SEC542	Austin, TX	Nov 13, 2017 - Nov 18, 2017	Community SANS
SANS London November 2017	London, United Kingdom	Nov 27, 2017 - Dec 02, 2017	Live Event
Community SANS Ottawa SEC542	Ottawa, ON	Nov 27, 2017 - Dec 02, 2017	Community SANS
SANS San Francisco Winter 2017	San Francisco, CA	Nov 27, 2017 - Dec 02, 2017	Live Event
Community SANS Marina Del Rey SEC542	Marina Del Rey, CA	Nov 27, 2017 - Dec 02, 2017	Community SANS
SANS Frankfurt 2017	Frankfurt, Germany	Dec 11, 2017 - Dec 16, 2017	Live Event
SANS Cyber Defense Initiative 2017	Washington, DC	Dec 12, 2017 - Dec 19, 2017	Live Event
SANS Cyber Defense Initiative 2017 - SEC542: Web App Penetration Testing and Ethical Hacking	Washington, DC	Dec 14, 2017 - Dec 19, 2017	vLive
SANS Security East 2018	New Orleans, LA	Jan 08, 2018 - Jan 13, 2018	Live Event
SANS Miami 2018	Miami, FL	Jan 29, 2018 - Feb 03, 2018	Live Event
SANS OnDemand	Online	Anytime	Self Paced
SANS SelfStudy	Books & MP3s Only	Anytime	Self Paced