# Detecting the ghost in the browser: Real time detection of drive-by infections

*Thijs Kinkhorst <thijs.kinkhorst@os3.nl>*
*Michael van Kleij <michael.vankleij@os3.nl>*
*Master Education System and Network Engineering*
*Universiteit van Amsterdam*

**Abstract**

Drive-by infections exploit common browser (plugin) vulnerabilities to infect client machines. Exploits are often loaded via compromised legitimate websites. This research tries to construct a methodology of detecting these infections purely by looking at the HTTP network traffic that this generates. We identify a number of salient characteristics and define a ruleset based scoring framework to determine whether an attack has taken place. Validation shows that this is a feasible approach, although more time is needed to create a balanced scoring ruleset.

## 1 Introduction

Compromised hosts are big business on the Internet. An individual zombie may be worth just a few cents, but selling them in a package of thousands starts to make serious money. This is reportedly a multi-million dollar industry.[1] Targeting these hosts directly from the outside, as traditional malware infections tend to do, has its limitations. Users nowadays are more likely to be protected by a firewall or cannot be probed directly from the Internet due to network address translation (NAT). The answer of the malware industry is to attack the host from the inside.

A trending vector to mount such an attack is through exploiting vulnerabilities in the user's web browser or one of the common plugins thereof, like Flash, Acrobat Reader or Quicktime. A malicious piece of scripting code is placed inside an otherwise legitimate website. Once the user visits this website, the code tries a number of known exploits. This kind of attack is called a *drive-by infection* because the user is infected merely by visiting a web page. Such attacks are often successful because browser plugins are not usually part of patch regimes. When an attacker successfully modifies one of the big websites on the Internet the number of infections within one hour can be overwhelming. An example of a partially successful hack on a large website is the inclusion of a malicious

iframe on the website of MSN Canada on 10 June 2009.[2] Fortunately visitors of the website were redirected to another website before the malicious code could be executed. Would this not have been the case, thousands of computers could have been infected before the iframe was discovered, as MSN Canada reportedly belongs to the 300 most visited websites on the Internet.[3]

Several methods of detecting infections have been researched, for example so-called client-side honeypots.[4] Our approach is different in that we will attempt to identify a drive-by infection purely from the network traffic generated by the host.

Our approach is based on the IDS service that Fox-IT offers.[5] This IDS will passively sniff all network traffic on a selected LAN. Their desire is to be able to detect anomalies in the user's web browsing traffic, so a drive-by infection can be identified. We want to avoid relying on a signature based approach, since attacks change often and a more general approach can detect new vulnerabilities as well.

## Research scope

The central question of this research is hence:

> *Can drive-by infections be discerned from legitimate sessions purely by measuring changes in* HTTP *traffic patterns and meta data?*

Meta data includes but is not limited to timing of requests and responses, geographical location of the requested resource, domain- and filenames, mime-types, redirection and headers. We will focus on externally observable, HTTP-protocol traffic patterns only.

This implies that we will not study the actual content of the HTTP protocol packets, for example to do signature matching. The behaviour of the malware after it has completed infection of a host, for example sending out spam email, is explicitly out of the scope of this research, as are user-assisted infections like a popup that advises to install a "free video player".

## This report

The remainder of this report is structured as follows. In chapter 2 we will further define the term "drive-by infections" and describe common characteristics. Chapters 3 and 4 describe how we gathered and analysed HTTP traffic data of drive-by infections. From that data we try to distill patterns to use in detection in chapter 5, the quality of which we validate in chapter 6. We finish with a conclusion and a view to future work.

The work for this research has been carried out in the context of the master education System and Network Engineering, Universiteit van Amsterdam and was supervised by Bart Roos and Sander Peters of Fox-IT, Delft.

# 2 Characteristics of drive-by-infections

## 2.1 Defining the term

Web-based infections can be assisted in various forms by the user of the target machine. Malware can piggyback on more-or-less legitimate software installation like a filesharing program, or pose as useful software that the user would need, like a video player or codec. These attacks require that the user, although mostly ignorant of the consequences, acknowledges the download or installation. Often social engineering is used to convince the user to take the action.[6, 7] In this research we focus on the most dangerous form: where infection occurs without user interaction and hence probably completely unnoticed. We speak of a drive-by infection if software is automatically downloaded or installed without user knowledge or consent, as a consequence of visiting a web site.[8]

## 2.2 Malware distribution networks

Provos et al.[6, 8], Wang et al.[9] and Moshchuk et al.[10] all describe the infrastructure of malware distribution networks. Common characteristic in all these infrastructures is the differentiation between landing nodes and distribution sites. Figure 1 depicts such a distribution network. A user on the computer browses to the infected website, on which an iframe (a hidden part of a web page that can reference another page) redirects the browser to a landing node. Depending on the preferences of the attacker, the browser can be redirected to an ad serving company or to further redirection steps ('JavaScript' in the image). In the end the browser is given a file that contains an exploit for the installed browser or plugins. Common exploits are buffer overflows in Apple Quicktime or Adobe Acrobat Reader. This buffer overflow allows for executing the attacker's code, which usually results in additional software to be downloaded from the distribution site after which the downloaded software is executed. At this point the machine is under control of the attacker and can be used for its designated malicious purpose.

In order to make the infection successful, the potential target needs to visit the infected website. To achieve maximum infection it's therefore most interesting for an attacker to use existing, legitimate websites for this. The attacker inserts his small piece of malicious code into this website in such a way that it goes as unnoticed as possible. The following vectors are used to insert this code[6, 11]:

- Web server or server scripts vulnerability;
- Advertisement networks, especially via a number of sub-syndication steps;
- Third party website plugins (widgets);
- User contributed content (comments, forum posts) not properly sanitised;
- Guessed FTP credentials, or stolen via other malware.

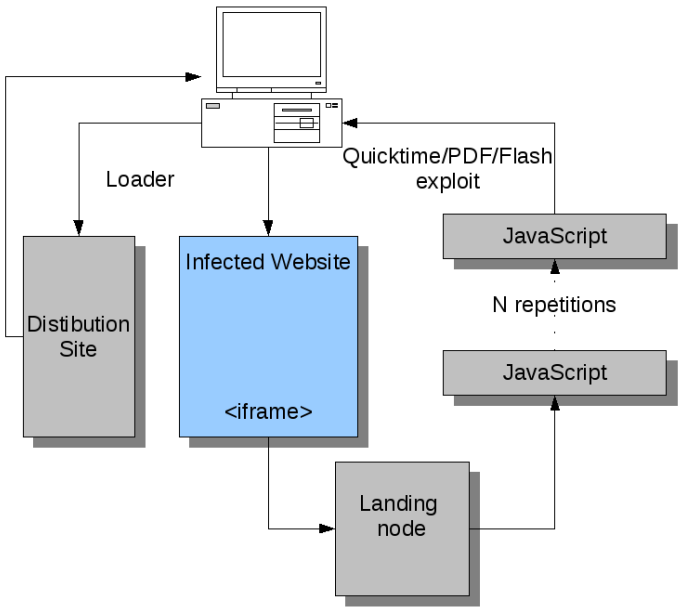


Fig. 1: Distribution network

Drive-by infections use a pull based mechanism to spread malware. To be able to do this there must be locations from which malware can be 'pulled'. Often these are compromised websites, or websites that rent out advertisement space. On these, often trusted, websites an iframe is inserted. These iframes point to other websites. In many cases these are so called redirection nodes. According to both Provos and Wang a browser visits multiple redirection nodes before arriving at a landing site; the malware is then downloaded from these landing sites. These redirection steps are included to bypass detection methods.

## 2.3 Known drive-by infection patterns

In previous research some drive-by infection patterns were already discovered. We can use these patterns in our analysis of the data. The following patterns were discovered by Provos et al.:[6]

- Distribution of landing and distribution sites over the IPv4 address space.
- Number of downloads averages around 8 per malware infection.

According to Provos et al. most of the distribution sites are located in the 58.* – 61.* and 209.* – 221.* IP address ranges. Another characteristic of malware they found was the number of file downloads within the session.

McGrath et al.[12] describe some other interesting patterns. Although their study focuses on phishing websites, we believe that some of the patterns found

in their research could apply to malware distribution sites as well. According to them the following characteristics are significant:

- More than 75% of the found URLs contain subdomains.
- Very few legitimate websites have URLs longer than 75 characters while a very large part of the phishing URLs contain between 75 and 150 characters.
- Phishing domains tend to be shorter than regular domains.
- Relative letter frequency of phishing domains is 'off'.

We will use these characteristics as starting points to see whether they also apply to drive-by infections, or whether additional or different measures are needed.

The existing research cited above employs detection on the client machine, for example by monitoring file system or process list changes, or using anti-virus scanners. Our approach, which studies the effects on network traffic observable from outside the client, has seen no precedent as far as we know.

## 2.4 Quantity and volatility

Provos describes that the system they run to detect infected websites finds between ten and thirty thousand malicious URLs each day.[6] Working for Google, they estimate that the total number of currently infected websites runs in the millions, and that over one percent of Google search queries returns an infected web site; a similar number was found by Wang for Microsoft's search engine.[9]

Despite being relatively high in numbers, the set of affected websites is very volatile. Attackers mostly target legitimate websites, which will be taken down or cleaned as soon as the compromise is detected. Because they are not online for longer periods of time, analysing these attacks can be hard. To add to that, the malware distribution networks treat subsequent requests from the same host differently, for example by serving legitimate ads. Using a client like `wget` may even get your IP blacklisted.

## 2.5 A Case Study: themoomusic.com

As an illustration of how an infection works in practice, we'll present a real life case of an infected website: themoomusic.com.[1] This is the legitimate web presence of the band "the moo" from the Czech republic, and for the casual visitor it indeed looks that way: your typical band website with history, discography and guestbook.

Looking under the hood, near the top of the page's HTML source code, we find that the following snippet has been added after the `<body>` tag:

[1] We fabricated this name because the page has been cleaned in the meantime.

```
<iframe src="http://globalnameshop.cn:8080/index.php" width=153
height=102 style="visibility: hidden"></iframe>
```

This `iframe` is an inline frame inside the webpage that can reference another URL as a source. Its width and height are dummy values, since the CSS style code sets the visibility to hidden, meaning that it will not be rendered in the browser. The net effect is that a remote resource is fetched by the browser without displaying the results – a genuine covert operation. In this case the browser connects to globalnameshop.cn on port 8080 and requests index.php.

The rogue website globalnameshop.cn sends back some obfuscated JavaScript that in turn instructs the browser to download a number of files on the same server:

- `/load.php?id=0` which contains binary executable code named `load.exe`;
- `/cache/readme.pdf`, a Portable Document Format file; and
- `/cache/flash.swf`, a Shockwave Flash applet.

This turns out to be the "Gumblar" trojan.[11] Each of these files is loaded with exploit code for weaknesses in the browser (plugins) or operating system. Only one has to succeed for the infection to work, which it has. The CPU load increases, Acrobat Reader is started and unknown processes are spotted in the process list.

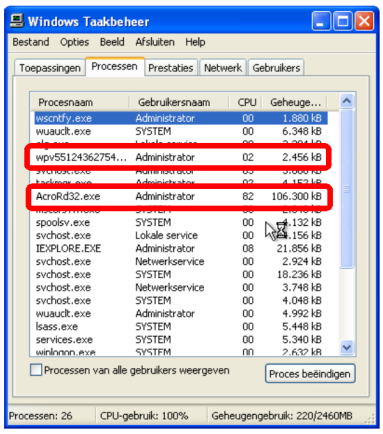


Fig. 2: Rogue processes appear after an infection.

We now also see requests appearing to a new host, only known by its IP address, where the following URLs are fetched:

```
/new/controller.php?action=bot&entity_list=&uid=1&first=1&guid=
    3759229163&rnd=981633
/new/controller.php?action=report&guid=0&rnd=123&uid=1&entity=
    1241530597:unique_start;1243627542:unique_start
```

This is probably the newly installed malware calling home, because the requests are different than the ones done from the browser: they lack common header fields like User-Agent and Referer, which are present in all regular requests. Probably the malware has a built-in HTTP client that doesn't emit these headers. The web server sends (near) empty responses on this.

The machine is now ready to perform its nefarious tasks, but we prevented trouble by shutting it down: our research only concerns the infection process itself.

# 3 Lab setup and data collection

In order to determine a detection method we need data to work on. This data will be collected by visiting known malware-distributing sites in a lab environment, where we can store and subsequently process all network traffic.

## 3.1 Data source

We acquired URLs contaminated with a drive-by download at the web sites `http://www.malwaredomainlist.com` and `http://www.malwareurl.com`. As their names suggest, they collect URLs that are known to distribute malware, intended for blocking or research purposes. The list contains direct links to the malware, but also infected websites that refer to it. We also used the similar `http://hosts-file.net` and picked up some infected websites through security blogs.

In a first iteration, we visited a number of these websites in our controlled environment, which is explained in more detail below, and stored the network traffic of those cases where infection occurred. These sites are different in nature: from truly legitimate websites that have been compromised, via more 'grey' websites to websites that have been set up with the express purpose to lure unsuspecting visitors in and infect them.

To get more comparison material we also opted to collect "before and after" data from commonly visited websites. We first visit the selected website as-is and store the network traffic. Next we inject an iframe directing to malware into a copy of the page, and run the test again. For this we used a number of sites appearing in the top twenty websites in popularity in The Netherlands over 2008:[13]

| | |
|---|---|
| `www.google.nl` | `www.buienradar.nl` |
| `www.bing.com`[2] | `nl.wikipedia.org` |
| `www.hyves.nl` | `www.detelefoongids.nl` |
| `www.marktplaats.nl` | `www.startpagina.nl` |

Tab. 1: Websites on which an infection was simulated.

We first captured the traffic generated by visiting each site at its original location, i.e. typing the web address into the location bar. Then we did the same but now for a mirror of only the start page of the website on our own server, directing all links in it back to their original location (using the `<base href>` tag). All pages have been mirrored on 9 June 2009. We compare the output of these two to ensure that the mirroring process itself didn't introduce artifacts into the traffic data. Then we modified the mirrored start page just to insert a malicious `iframe` or `script` tag which we obtained through the sources mentioned above. The traffic of this session, where we actually infect the virtual machine, is also stored and used in our analysis.

[2] The original website listed in the Multiscope Top 20 is live.com, which currently redirects to bing.com.

## 3.2 Lab setup

To make the experiments reproducible, we start our OS for every link visited in a known-clean state. Because of this we chose to use virtualisation for our client machines. Besides the easy rollback, it also allows us more flexibility with respect to capturing the network traffic and to prevent malware leakage from an infected machine. We rolled back to the clean state after each URL we visited, regardless of whether it resulted in an actual infection.

As virtualisation platform we use the Xen hypervisor on an Ubuntu 8.04 server, with hardware virtual machine (HVM) to avoid anti-virtual-machine behaviour in the malware. As guest OS we use an unpatched version of Windows XP SP2 with Internet Explorer 6. An incompletely patched version of Windows and IE6 were chosen because of the greater likelihood of successful infections while still resembling an installation that's frequently found in the wild. We also installed the plugin versions listed in the table below; these seem to be commonly installed in the real world.[14]

| Plugin | Version |
|---|---|
| Microsoft .NET Framework | 2.0 |
| Adobe Acrobat Reader | 7.0.5 |
| Apple Quicktime | 7.0.3 |
| Adobe Flash Player | 7.0.19 |
| Microsoft Java Virtual Machine | 5.0.3802 |

Fig. 3: Browser plugin versions used

Because we run a not completely patched version of Windows XP we are at risk of other infections than the one we're interested in, for example more classic exploits that attack the system from the outside. To mitigate this risk, we have set the virtual machine up behind a NAT environment to prevent direct access to our virtual machines from the world, and thus only allowing pull based mechanisms like drive-by infections. The used network configuration is depicted in image 4.

We are not interested in the actions of the machine after infection, only in the infection process itself. To prevent mayhem instigated by the infected machine carrying out its payload, we shut it down shortly after the compromise has taken place. Further we've blocked outgoing port 25 (SMTP) as we believe that this will not interfere with our capture of related traffic (we focus on HTTP traffic) but may prevent accidental spam getting sent out.

With this setup we process known malware sites one by one following the capture protocol detailed below. This results in one capture file that contains the network traffic generated by visiting exactly one URL.

## 3.3 Capture protocol

We need to capture data in an uniform and reproducible way. Therefore we have specified a protocol, which specifies every single step that is performed for

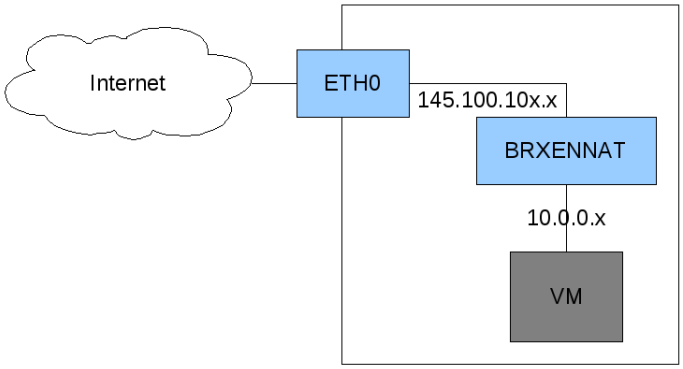


Fig. 4: The configuration of our test network

one capture. The protocol is as follows:

1. Start virtual machine.
   `xm create windowsxp.hvm`
2. Start capture.
   `tcpdump -i brxennat -s0 -S -U -l -w <filename>`
   filename format: yyyymmddhhmmdomainname
   - -i interface
   - -s0 full capture
   - -S absolute ordering
   - -U packet buffering
   - -l stdout linebuffered
   - -w write to file
3. Open browser and surf to the specific website.
4. Wait for 2.00 minutes.
5. Close browser and wait 2.00 minutes again.
6. Shutdown virtual machine.
7. Stop capture.
8. Commit capture.
   `svn add <filename>`
   `svn commit`
9. Restore virtual machine to clean state.
   Change external IP address.
   `restore-clean-state`

The virtual machine is created from a known-clean disk image. We then start the capture and instruct it to keep all traffic; in the analysis phase we can decide on further filtering. We're now ready to browse to the specific URL under investigation, so it is typed into the location bar of the browser. We keep it open for exactly two minutes: literature shows that some known attacks have timers but these never exceed two minutes so this is enough at least for currently known attacks to be launched.[9, 10] We also close the browser afterwards (and wait for the same time) as some JavaScript reportedly only starts after the user navigates away from the infected page.[10]

After the infection we stop the virtual machine to prevent any installed malware from causing unintended abuse. The captured data file is immediately committed into a dedicated Subversion repository; this ensures us that it's kept safely and is not inadvertently changed afterwards. Then we rewind the virtual machine to the clean state to be ready for the next capture, and we rotate the IP address. This is done because reportedly some attacks are non-functional on any second and further requests from the same host.[15] As the behaviour of the host after infection is not within the scope of this research, we do not keep snapshots of each infected host.

## 3.4 Composition of the data set

Testing a total of 64 URLs with the protocol above resulted in 64 different capture files. Inspection of the system during capture and the resulting capture files told us that out of these, 39 were clean. The other 25 experienced an infection (see table 2). Most notable is the so-called Gumblar trojan, which is very present in the dataset. This coincides with the reported surge in Gumblar infections reported in early June 2009.[16] The other infections are attributable to the very recent Nine-Ball attack[15] and an assortment of other attack strategies.

| | | |
|---|---|---|
| Trojan Gumblar | 11 | |
| Trojan Nine-Ball | 3 | |
| Trojan Other | 11 | |
| Infected | | 25 |
| Clean | | 39 |
| **Total** | | **64** |

Tab. 2: Distribution of different malware types and clean sessions over the collected dataset.

## 3.5 Composition of the validation set

To verify our analysis and our detection methods we created a second dataset: completely separate from the original set above and collected at a somewhat later date. Of this dataset 20 captures were clean and 15 were infected. In table 3

the composition of this set is displayed. The number of Gumblar infections has declined a little and the other trojans are somewhat more prevalent.

This dataset has been kept completely separate from the main set and will be used exclusively for validation, not for designing the detection method.

| | | |
|---|---|---|
| Trojan Gumblar | 5 | |
| Trojan Nine-Ball | 2 | |
| Trojan Other | 8 | |
| Infected | | 15 |
| Clean | | 20 |
| **Total** | | **35** |

Tab. 3: Distribution of different malware types and clean sessions over the validation dataset.

# 4 Analysis of patterns

In this section we discuss the various patterns we have found in our dataset. We first analysed the dataset manually to select strategies with a high potential, which we've further analysed and will describe in this chapter.

In the drive-by infections industry trends are just as common as in other industries. The current trend, the Gumblar infection, is a prominent part of our dataset and thus unavoidably influences our patterns. Nonetheless we think we've extracted parameters general enough, with help of the rest of our dataset, to be of use beyond identification of Gumblar alone.

The patterns we describe here are converted into a more concrete approach to detection in chapter 5. In chapter 6 we will test this approach with a separate dataset.

## 4.1 TCP Port numbers

We have looked at the commonly used ports for HTTP traffic. These are port 80 for normal HTTP traffic and port 443 for HTTPS traffic. Port 8080 is defined as the alternative HTTP port. For each website we kept a statistic specifying the number of requests to each port. Of the websites we found clean of malware *none* used port 8080, and only a few used port 443. However on 40% of the contaminated websites references to port 8080 were found. In table 4 the exact numbers can be found. Of the 39 clean websites we tested all had at least one connection on port 80. Of the 25 infected websites we analysed 10 websites had at least one connection to port 8080.

| | Clean | | Infected | |
|---|---|---|---|---|
| | # of sess. | % of sess. | # of sess. | % of sess. |
| port 80 | 39 | 100% | 25 | 100% |
| port 8080 | 0 | 0% | 10 | 40% |
| port 443 | 3 | 8% | 2 | 8% |

Tab. 4: Number and share of captured sessions having at least one request to the given port number.

A reason for the high prevalence of non-standard port numbers with malware is that it's often distributed from compromised webservers. If the attacker would occupy port 80, this has a high chance of being noticed by the server administrator; listening on port 8080 does not have that risk, while at the same time port 8080 is still common enough to be allowed by firewalls. Hence, port numbers can be used as an indicator whether traffic might be malicious. Even though not all infected websites use alternative ports we have seen that none of the clean websites use these ports. Seeing another port than 80 or 443 might be suspicious.

We also calculated the average number of requests per port. These are displayed in table 5.

| | Clean session | Infected session |
|---|---|---|
| port 80 | 44 | 73 / 47[3] |
| port 8080 | 0 | 2 |
| port 443 | 1 | 27 |

Tab. 5: Average number of requests on a port per captured session.

The average number of requests on port 80 does not differ enough to be able to recognise malicious traffic. The number of requests on port 443 does differ a lot. One of the possible causes for this was that we did not capture specific HTTPS pages. In normal browsing behaviour HTTPS might be more common as people use services like online banking. We believe that the usage of port 443 in the case of the infected websites is due to the downloaded and installed malware on the client's computer. These malware programs might use HTTPS to be able to inconspicuously download other programs and instructions.

## 4.2 Geographical locations

As we found many references to websites with Chinese domain names we thought it might be interesting to use GeoIP – a mapping of IP addresses to countries – to locate the IP addresses of the various hosts we encountered. To do this we used the GeoIP database available in Ubuntu Linux[4]. We found that the clean websites have an average of 2.36 different countries per session and infected websites have an average of 3.64 countries per session. Of the 39 tested clean sites 21 had only one country to which requests were made. Of the other 18 sessions, 7 had two different countries and 11 more than that. Of these 18 sessions 6 had less than 10% (or one request) to a foreign country. We assume these to be advertisements or counters. However, we discovered that every clean website that was not hosted in China, Russia or Ukraine did not have any references to those countries while all infected websites have. So if the beginning of a session does not start within any of these countries the chance is small that any of the requests will go to these countries. We can define this as follows:

$(x \notin \beta) \cap (Y \subset \beta) \Rightarrow suspicious$
where:
$\beta$ is the set of 'bad' countries,
$x$ is the first request in a session,
$Y$ is the set of consecutive requests in a session.

This requires the construction of a set of bad countries. Furthermore it requires to be able to recognise sessions as such, to be able to find the first request of such a session.

[3] One of the infected websites generated 627 requests on port 80. When we remove this outlier from the average the number of requests becomes 47

[4] libgeoip1, version 1.4.4.dfsg-3

## 4.3 Hostnames

An interesting subject are the DNS hostnames used to download content from. We define this as the value of the `Hostname` field in a HTTP request. We examined several aspects of this field; the first is the number of labels in this hostname. For example, the hostname `www.example.com` has three labels while `example.com` has only two. This header may also contain an IP address, in case the original URL was of the form `http://10.11.12.13/`. These are considered separately.

While varying numbers of labels are prevalent in our data, there's a notably higher number of two-label domains in infected websites, as can be seen in table 6. This lists the number of websites that caused at least one request to a two-label or a multi-label (three or more labels) domain name. Also the presence of requests to a host identified by a bare IP address is significantly higher for infected sites.

Both the two-label hostname as the IP address sporadically appear in the traffic to clean websites as well, albeit significantly less often than for malicious sites. This means we can use both as a measure that indicates a higher likelihood for a website to be infected but not as conclusive evidence.

Another aspect that we researched is that malware often employs domain names from the Chinese `.cn` top level domain (like in the example in section 2.5). This doesn't mean that they are also referring to websites in China: a lookup of such a hostname may turn up a number of different IP addresses spread around the world with a short time-to-live field. In other words, the IPs belong to machines currently under control of the miscreant but can be swapped quickly when needed. As can be seen in table 6, there's a clear difference between clean and infected websites in this regard.

| | Clean | | Infected | |
|---|---|---|---|---|
| | # of sess. | % of sess. | # of sess. | % of sess. |
| two-label domain | 7 | 18% | 25 | 100% |
| multi-label domain | 39 | 100% | 25 | 100% |
| IP address | 3 | 8% | 17 | 68% |
| .cn domain | 1 | 3% | 17 | 68% |

Tab. 6: Number of sessions containing at least one Hostname field of the given type.

In three cases we also spotted up to ten different domains of the following forms being used by malware.

```
mbp27hv405c.com  xnc73ti051o.com  nes40ky627f.com
rhv73nc051i.com  fuj51bp728v.com
```

They look auto-generated. This could be considered a pattern but as it's only present in a small number of infected sites, the sensitivity of this pattern is low.

## 4.4 User Agents

We found that when a computer was infected with malware it started to download other programs and instructions. Most of these downloads were executed over normal HTTP connections. However, the malware programs used other header information than the default browser. In some cases we even saw a completely empty HTTP header except for the mandatory `Host` header. However the easiest way to recognise infections was the `User-Agent` header, which provides an identification string of the web browser of the user placing the requests. This is displayed in table 7.

| | Clean sess. | Infected sess. |
|---|---|---|
| Total number of different User-Agent headers found | 2 | 9 |
| Average number of unique User-Agent headers per session | 1.0 | 2.5 |
| Average number of requests with non-original User-Agent per session | 0.2 | 6.1 |

Tab. 7: Presence of User-Agent headers: the total number of different strings found in our set, the average number of different strings per session and the average number of requests in a session having a User-Agent different from the user's browser.

The presence of multiple User Agents when browsing to a legitimate website is clearly less common than for an infected one. We did find a legitimate website introducing a new User Agent: this was because of a Java plugin on the website that downloaded some extra modules. Another User Agent we found was Adobe's updater. This User Agent is recognisable by its name. However, all other User Agents we found were variants on normal User Agents that you would normally not see in a regular browsing session. The complete absence of a User Agent header was even the most common deviation. We also found that the legitimate User Agent was always the first User Agent found within a browsing session. It was also the most common User Agent in all sessions, this might be used to identify the legitimate User Agent. The absence of a User Agent header field is a very strong indicator of malicious activities. A deviation of the User Agent is an indication of malicious activity, but not decisive proof.

## 4.5 Invalid POST Requests

Most of the requests are of type GET, but sometimes a few POSTs are made, both on clean and infected websites. Malware on infected sites sometimes uses POST to send identification data back to a control server. It turns out that some of the POST requests that malware sends have a `Content-Length` header that specifies an invalid value: the actual number of bytes in the request body is different. This violates the HTTP specification.[17]

This implementation quirk is often found in our collection of infected sessions, but never turns up on clean ones. Since our capture protocol doesn't provide for

interacting with websites, and POST is more common on interaction, we also did some separate capturing of POST requests to a number of legitimate websites with different browsers to generate enough comparison material.

| | Clean | | Infected | |
|---|---|---|---|---|
| | # of sess. | % of sess. | # of sess. | % of sess. |
| Invalid POST | 0 | 0% | 14 | 56% |

Tab. 8: Sessions having at least one POST request not conforming to RFC2616.

Interesting about this item is that we did not find any case of a legitimate HTTP client behaving in this way. That gives such requests a high predictive value of a drive-by infection.

## 4.6 Request URIs

In HTTP, the request URI is the 'path name' that is referenced in requests. Table 9 shows a number of recurring URIs that we encountered in sessions browsing to infected websites. As can be seen, these URIs are unique to the group of infected sites and are never present in clean ones. They form a good indicator that something is wrong. However, the other half of infected websites employed a variety of other URIs that only appeared once or twice each.

Note that the URI components referenced in this table are substrings, not anchored on the left or right.

| | Clean | | Infected | |
|---|---|---|---|---|
| | # of sess. | % of sess. | # of sess. | % of sess. |
| `/landig.php?id=4` | 0 | 0% | 10 | 40% |
| `/load.php?id=0` | 0 | 0% | 11 | 44% |
| `/in.cgi?` | 0 | 0% | 7 | 28% |
| `/cache/flash.swf` | 0 | 0% | 10 | 40% |
| `/cache/readme.pdf` | 0 | 0% | 10 | 40% |
| `/controller.php?action=` | 0 | 0% | 12 | 48% |
| `/receiver/online`[5] | 0 | 0% | 12 | 48% |

Tab. 9: Number of sessions that place at least one request to suspicious request URIs

## 4.7 Content Types

We expect that the common MIME content types in HTTP traffic should primarily be text/html and images. Some JavaScript, CSS and Flash might also be found depending on the website. The different content types that were found in our dataset are shown in table 10. One of the content types that caught the eye was

[5] This URI is only used in requests using the POST method.

application/octet-stream. This content type occurred only once in the sessions browsing to a clean website, but 69 times when browsing to infected sites. The application/octet-stream content type is used to define content which is binary and doesn't have a specific task within a session like flash or pdf. Downloads fall in this category. It is not uncommon to see downloads within clean sessions. However, according to Niels Provos the average number of downloads within an infected session is 8. We discovered an average of 2.76 (only counting application/octet-stream as an identifier of a download). We do believe this to be more than in clean sessions, as we only discovered 1 application/octet-stream content type within the clean sessions.

There's also a noticeable difference for application/pdf. However, this statistic may be skewed due to our capture method: we captured only front pages, and this is indeed hardly ever a PDF. We are not using this statistic because we expect that in real life more legitimate application/pdf content types will show up.

| Content-Type | Clean sessions | | Infected sessions | |
|---|---|---|---|---|
| | # | % | # | % |
| image/jpeg | 442 | 31.2% | 152 | 14.3% |
| image/gif | 437 | 30.8% | 220 | 20.6% |
| text/html | 225 | 15.9% | 296 | 27.8% |
| application/javascript | 123 | 8.7% | 150 | 14.1% |
| image/png | 106 | 7.4% | 63 | 5.9% |
| text/css | 40 | 2.8% | 38 | 3.6% |
| application/x-shockwave-flash | 17 | 1.2% | 16 | 1.5% |
| application/x-www-form-urlencoded | 10 | 0.7% | 21 | 2.0% |
| text/plain | 10 | 0.7% | 5 | 0.5% |
| application/octet-stream | 1 | 0.1% | 69 | 6.5% |
| application/pdf | 0 | 0.0% | 22 | 2.1% |
| multipart/form-data | 0 | 0.0% | 5 | 0.5% |
| *other* | 6 | 0.4% | 9 | 0.9% |
| total content types | 1417 | 100% | 1066 | 100% |

Tab. 10: Total number of different content types encountered across the dataset.

## 4.8 Redirection

From literature we know that landing sites commonly use a number of redirection steps before we arrive at the real exploit (see section 2.2). In our investigation it turned out that the pattern of 'formal' redirects by using the Location header from the HTTP specification[17] – status codes 301 Moved Permanently and 302 Found – is not significantly different between our clean and infected datasets.

The landing sites do use redirection, but employ JavaScript or the loading of a new page within the initial page to do that instead. Given that we only consider the HTTP headers and not the request body, our way of detecting this is limited

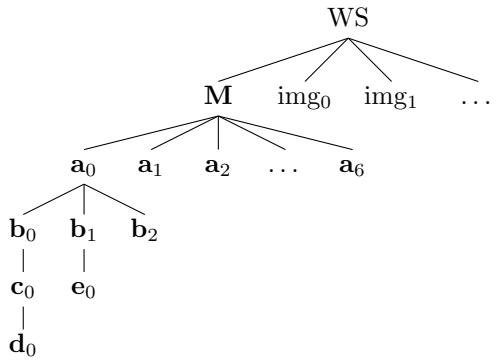


Fig. 5: Redirection structure of an infected website. WS is the original web page and M the malware landing page. Illegitimate content is in bold face.

to following the chain of `Referer` headers: a request to a redirected-to page will usually have the URL of the redirecting page in that field. Following this chain leads to a tree of objects that have been loaded via their parent, rooted in the initial web page (under the assumption that there are no redirect loops). Figure 5 shows what this tree looks like for one of our captures of an infected website. The website will load its images directly, but also loads the landing page. That in turn loads a set of pages, some of which load yet other pages, etc.

We measured the height of the redirection tree for each of our captures. When a request doesn't contain a `Referer` header we ignore it for this purpose, as there is no good way we can relate it to other requests just basing ourselves on HTTP metadata. The results are found in table 11: in legitimate websites most content is loaded directly, with sometimes a little indirection. For infected sites the amount of indirection is significantly higher.

| Tree height | clean sessions | infected sessions |
|---|---|---|
| Maximum | 4 | 6 |
| Minimum | 0 | 1 |
| Mode | 1 | 3 |
| Average | 1.5 | 2.9 |

Tab. 11: Redirection tree heights, aggregated over the sessions.

## 4.9 Rejected characteristics

### 4.9.1 Timing of requests and responses

When looking at the timings of requests and responses it is possible to see anomalies in the infected sessions. Of these infected sessions a few responses take more than one second and in some cases up to ten seconds. However, we found that on average only one response per session had an excessive response time. We do not believe that just one request is too fragile to be a feasible

characteristic for identifying malicious traffic. A clean website could also have a problem at a given point in time in which responses are delayed for several seconds. Because of this the timing characteristic has been rejected.

### 4.9.2 404 status messages

When looking at the gathered dataset it appeared that 404 status codes were more common in infected sessions. After research we found out that these status codes are distributed equally across our dataset, so this characteristic has been rejected as an identifier.

### 4.9.3 TCP retransmissions

We assumed due to the lower quality of hosting for malicious content that the number of TCP retransmissions would be significantly higher in infected sessions than it would be in clean sessions. After research we discovered this not to be true for this dataset.

# 5 Description of detection methods

To be able to detect malicious traffic on the network the previously identified characteristics will be used. We have seen that none of the characteristics uniquely defines malicious traffic, but a combination of various characteristics will increase the likelihood. We propose a ruleset based scoring mechanism like the one that is used in for example SpamAssassin.[18] In such a system, rules are applied to the input under examination. A match of the rule results in a specific score associated with that rule. If the sum of the scores passes a predetermined threshold, the input under consideration is flagged.

This method allows to combine individually inconclusive information into a high-confidence verdict. Furthermore it also aids flexibility and expandability of the method, as new rules can be added or scores adjusted for changing circumstances.

In order to create a scoring mechanism we must decide on the amount of points required before traffic is considered malicious. At the moment it is not possible for us to make a balanced scoring algorithm as this requires more research. Malwareland has shown to be sensitive to trends, and the current trends we captured for this research may prove to not be robust enough for the next trend. The mentioned flexibility however allows for adjustments in the future while keeping the general method.

## 5.1 Session reconstruction

Many of our rules depend on the ability to detect sessions and keep them apart from other sessions. We define a session as all related HTTP requests and responses when visiting a single website. As a study of how to detect sessions within network traffic is out of scope we will only offer our ideas on possibilities to do this. This is not trivial, because HTTP is a sessionless protocol; session reconstruction will be heuristical by nature and no method might be foolproof.

Some ideas for sessionising have been explored by Spiliopoulou et al.[19], although they are using web server log files as input. In our case, linking Referer and Location headers together, combining it with possible session cookies seems plausible. (Semi-)permanent cookies exist, but this problem can be tackled by defining an upper bound on the duration of any single session. It has to be noted that some requests we've seen were not part of any session as the malware itself sends out requests to various servers. These requests should be captured and analysed regardless of session information as these requests have a high value in detecting malware.

Further refinement of the demarcation between different sessions is a research subject on its own and out of scope for this report. For the remainder of this chapter we assume an existing division into individual sessions, which is the case for our dataset.

## 5.2 Detection & Scoring

In chapter 4 we presented various parameters useful to finding malicious traffic. We'll summarise these characteristics here and discuss the various detection possibilities they offer.

- TCP port numbers
- Geographical locations
- Hostnames
- User Agents
- Invalid POST requests
- Request URIs
- Content Types
- Redirection chains

For our scoring mechanism we have chosen a threshold of 5.0 points before a session is marked as malicious. To be able to reach this threshold at least a combination of characteristics must be discovered within a session; there is not one characteristic that will mark a session as malicious on its own. For that reason we will assign scores to individual rules at a maximum of 2.0 points. The score can be lower if our data gives a reason to be more conservative. Because some rules are additive, in our algorithm we also cap the contribution of a single rule to 4.0 points, ensuring again that no single rule can flag a session on its own. It must be noted that the specific scoring values are preliminary; more data is needed to properly assess the optimal scores.

The ruleset is described using pseudo code, one rule in each subsection. These rules need to be matched against a captured session; for each match the total score is incremented by the score that the rule returns. The algorithm takes a single capture, as a list of requests, as input and boils down to the following.

**Algorithm 1** Framework algorithm for evaluating the ruleset.

```
score ← 0

firstrequest ← front( capture )
for all rule ∈ ruleset do
  score ← score + min ( rule(capture, firstrequest), 4.0 )
end for

return score ≥ 5.0
```

### 5.2.1 TCP port numbers

We found that in our dataset none of the 39 tested clean websites used port 8080. This could be used as an indicator of malicious traffic. Especially since we found that in all sessions containing malicious traffic the session started on port 80 like clean sessions. However somewhere during the session port 8080 is introduced. This is strange behaviour. There is a possibility that clean websites use other ports than port 80, so we have to look at the first used port in a session. If this is port 80 it would be suspicious to find other HTTP ports during the session. We have to exclude port 443 from this as it is common to switch to secured websites within a session, for example on a webshop. For this we could define that if a session begins on port 80 it would normally continue on port 80 or port 443, but not on other ports. We also assume that when a session begins on an alternative port it would continue using this port or switch to port 80 or port 443.

The score is set to 2.0, because our data set gave a high specificity for this aspect: no clean websites were found that satisfy the rule. So we can define the following rule:

**Rule 1** Detecting 'bad' ports

```
function Rule (capture, firstreq) : s
for all request ∈ capture do
  if request.port ∉ {80,443} ∧ request.port ≠ firstreq.port then
    return 2.0
  end if
end for
return 0
end function
```

### 5.2.2 Geographical locations

The geographical location is a difficult characteristic to use as an identifier of suspicious behaviour. We can only detect 'odd' behaviour which is not necessarily malicious. The location of a website is indicative for the other countries which are listed. It is common for an Europe based website to have references within European countries and the United States. However, it is odd for an European based website to have references to China or Russia.

This is assigned the score of 2.0 if at least one request of the capture was in one of these countries while the original request was not placed there. Because we did not find clean captures that satisfied this the score is set high. We can define rule 2.

**Rule 2** Detecting 'bad' countries

**function** Rule (*capture*, *firstreq*) : $s$
**for all** *request* $\in$ *capture* **do**
  **if** *request.country* $\in \beta \wedge$ *request.country* $\neq$ *firstreq.country* **then**
    **return** 2.0
  **end if**
**end for**
**return** 0
**end function**

Where $\beta$ is the set of 'suspicious' countries. Based on our measurements we initially define this set as Russia, China and Ukraine.

### 5.2.3 Hostnames

During our analysis we found that in all sessions in which we browsed to a infected site at least one two-label hostname was present. Furthermore in 68% of the cases a hostname consisting of a bare IP address or a `.cn` domain was present. This contrasts to the 18%, 8% and 3% respectively of the clean sessions. This information can be used as an indicator for possible malicious activities, however, because of the occasional occurrence in clean sessions it is important to choose a relatively low score so not to generate false positives. `.cn` receives the maximum score because we did not measure any legitimate non-`.cn` websites referencing such a domain.

We define the following rules, where $\simeq$ symbolises the well-known Perl Compatible Regular Expression matching.

**Rule 3** Detecting 'bad' hostnames: two-label domains

**function** Rule (*capture*, *firstreq*) : $s$
**for all** *request* $\in$ *capture* **do**
  **if** *request.hostname* $\simeq \Re_0$ **then**
    **return** 1.5
  **end if**
**end for**
**return** 0
**end function**

**Rule 4** Detecting 'bad' hostnames: IP address

**function** Rule (*capture*, *firstreq*) : $s$
**for all** *request* $\in$ *capture* **do**
  **if** *request.hostname* $\simeq \Re_1$ **then**
    **return** 1.0
  **end if**
**end for**
**return** 0
**end function**

**Rule 5** Detecting 'bad' hostnames: .cn

**function** Rule (*capture*, *firstreq*) : $s$
**for all** *request* $\in$ *capture* **do**
  **if** *request.hostname* $\simeq \Re_2$ **then**
    **return** 2.0
  **end if**
**end for**
**return** 0
**end function**

Where we define $\Re_n$ in terms of PCRE:
$\Re_0$ is `/^[-a-zA-Z0-9]+\.[-a-zA-Z0-9]+$/`
$\Re_1$ is `/^([0-9]+\.){3}[0-9]+$/`
$\Re_2$ is `/\.cn$/`

### 5.2.4 User Agents

We found a very good indicator of malicious activity in the user agent as it should seldom change within a session. Within all clean sessions we found two different user agent strings of which one was Acrobat's updater. Within malicious sessions we saw an average of 2.4 different user agent strings within each session. This is a clear indication of malicious activities. We also found that the first user agent within a session is always the correct user agent. Here we choose to assign a relatively low score but increment the resulting score any time a user agent is encountered that wasn't in the first request. We chose the 0.4 based on the averages encountered in our dataset: to reach a target of 2.0 points, with a measured average of 6.1 user agents we need 0.4 points per user agent.

**Rule 6** Detecting 'bad' user agents

**function** Rule (*capture*, *firstreq*) : $s$
$s \leftarrow 0$
**for all** *request* $\in$ *capture* **do**
  **if** *request.useragent* $\notin \alpha \wedge$
  *request.useragent* $\neq$ *firstreq.useragent* **then**
    $s \leftarrow s + 0.4$
  **end if**
**end for**
**return** $s$
**end function**

Where: $\alpha$ is the set of whitelisted user agent strings, e.g. Adobe updater.

### 5.2.5 Invalid POST requests

Normal browsers generate correct POST requests. We found that some of the malware programs had a small HTTP client built in. This client was used to download programs and instructions, but also to report to its control unit. One of the used mechanisms for this are POST messages. Some of these POST messages violated the HTTP specifications: the content-length header did not match the real content length. This never happens in legitimate traffic. Therefore we can use this as a highly predictive characteristic, yielding 2.0 points if the session has at least one of these requests.

**Rule 7** Detecting malformed POST messages

```
function Rule (capture, firstreq) : s
for all request ∈ capture do
  if request.method = post ∧ request.content_length ≠ request.content_bytes
  then
    return 2.0
  end if
end for
return 0
```

### 5.2.6 Request URIs

We found some telling URIs in our dataset. While these URIs might be specific to one attack they can help identify malicious traffic. It would mean that a set of known malicious URIs has to be maintained. Each occurrence of a malicious URI within a session can be counted as a malicious sign, so each occurrence increments the score, but only once for each unique URI. Each such occurrence gets 1.0 point. We can define this as follows:

**Rule 8** Detecting listed URIs

```
function Rule (capture, firstreq) : s
s ← 0
for all uri ∈ β do
  s_u ← 0
  for all request ∈ capture do
    if request.uri ≃ uri ∧s_u = 0 then
      s_u ← 1.0
    end if
  end for
  s ← s_u
end for
return s
end function
```

Where: $\beta$ is the set of known malicious URIs, as listed in table 9 on page 17.

### 5.2.7 Content Types

Some content types are more common when traffic is malicious. For example application/octet-stream is more common in malicious traffic because it often downloads programs. Each download adds to the score. However, because it is possible for a legitimate session to have these content types scoring has to be kept on a low level. To detect suspicious content types the following definition can be used.

**Rule 9** Detecting octet-stream content type.

```
function Rule (capture, firstreq) : s
s ← 0
for all request ∈ capture do
  if request.contenttype = application/octet-stream then
    s ← s + 0.2
  end if
end for
return s
end function
```

### 5.2.8 Redirection

Relatively long chains of indirection have been identified as an indicator for malware. The rule builds a redirection tree by following Referer header chains, as described in section 4.8. The height of this tree is used as a scoring metric. Because our average captures have a modal height of 1, we subtract 2 from the height: being above that certainly tends to infection. However we bound the score on 2.0 so a legitimate site with a outlier very high redirection tree doesn't immediately raise all flags. This means that the resulting score will either be 0 when all trees are below height 3, 1.0 when a tree of height 3 is present, and 2.0 when even higher trees are found.

**Rule 10** Analysing redirection trees

```
function Rule (capture, firstreq) : s
T ← BuildRedirectionTree ( capture )
return min ( max ( 0, height(T) − 2), 2.0 )
end function
```

## 5.3 Considerations

As stated previously the composition of this ruleset is based on data collection in one month time. How future-proof will this be? It's even not unthinkable that malware authors will try to improve their software to explicitly evade known detection rules.

A key aspect of a ruleset based approach is the flexibility in adding new rules, augmenting existing ones and finetuning their associated scores. There's no

doubt that such a ruleset will have to be kept up to date with emerging malware trends. The SpamAssassin case however has shown that if you keep rulesets up to date to changing environments, the concept keeps being a viable approach to detection of new abuse. Its open source nature illustrates that even if the miscreants have full access to the ruleset, this still works.

Of specific interest is the possibility of malware starting to make use of encrypted HTTPS traffic instead of plain HTTP. The encryption would obscure a number of factors we currently base detection on, like HTTP headers. However, this may not be trivial to implement without creating unexpected popups for the user in the browser, and to cope well with switching between compromised hosts around the globe quickly. Additionally such requests may provide detection leads by virtue of their specific HTTPS usage. In any case our method will be more robust against such a (currently theoretical) move than the currently prevailing content inspection (signature-based detection), since our method can already incorporate information that will be detectable regardless of encryption, like TCP information or DNS information.

# 6 Validation of the detection methods

To validate the detection method we defined in chapter 5 we collected a second dataset, completely different from the dataset we used to construct our method. This dataset is described in section 3.5. In this chapter we describe the results of applying the previously defined method and its scores on this new dataset. Tables 13 and 14 show the scores for each rule and the total score per session, for the infected and clean datasets respectively.

The table for the infected dataset shows that 14 out of 15 sites indeed reach the set threshold of 5.0 points and are therefore flagged as malicious. In other words, just one session was not flagged which should have been. This is a sensitivity or true-positive rate of 93%.

For the clean dataset, all scores are below 5.0 so none are flagged as malicious. This means that the specificity is 100%, or a false-positive rate of 0%.

| | true | false |
|---|---|---|
| positive | 93% | 0% |
| negative | 100% | 7% |

Tab. 12: Summary of validation results.

| rule | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | $\sum$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| session 1 | 2.0 | 2.0 | | 1.0 | | 0.4 | 2.0 | 2.0 | 0.4 | | 9.8 |
| session 2 | 2.0 | 2.0 | 1.5 | 1.0 | 2.0 | 0.8 | 2.0 | 3.0 | 0.4 | | 14.7 |
| session 3 | | | 1.5 | 1.0 | | 4.0 | | 1.0 | 1.2 | | 8.7 |
| session 4 | | | 1.5 | | 2.0 | 0.4 | | | 1.6 | 2.0 | 7.5 |
| session 5 | | | 1.5 | | 2.0 | 0.8 | | | 1.8 | 2.0 | 8.1 |
| session 6 | | | 1.5 | | 2.0 | 0.8 | | | 0.4 | 2.0 | 6.7 |
| session 7 | 2.0 | 2.0 | | 1.0 | | | | 1.0 | 0.4 | | 6.4 |
| session 8 | | | 1.5 | | 2.0 | 0.4 | | | 2.2 | 2.0 | 8.1 |
| session 9 | 2.0 | 2.0 | | 1.0 | | | 2.0 | 2.0 | 0.4 | | 9.4 |
| session 10 | 2.0 | 2.0 | | 1.0 | | | 2.0 | 3.0 | 0.4 | | 10.4 |
| session 11 | | | 1.5 | | 2.0 | 0.8 | | | 1.0 | 2.0 | 7.3 |
| session 12 | | | 1.5 | | 2.0 | 0.8 | | | 2.0 | 2.0 | 8.3 |
| session 13 | | 2.0 | 1.5 | 1.0 | 2.0 | 1.2 | | 1.0 | 0.2 | | 8.9 |
| session 14 | | | 1.5 | | | | | | 0.2 | | **1.7** |
| session 15 | | 2.0 | 1.5 | | 2.0 | 1.6 | | 1.0 | | 1.0 | 9.1 |

Tab. 13: Scoring per rule of the sessions in infected dataset.

| rule | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | $\sum$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| session 1 | | | 1.5 | | | | | | | | 1.5 |
| session 2 | | | | | | | | | | | 0.0 |
| session 3 | | | 1.5 | | | | | | | | 1.5 |
| session 4 | | | | | | | | | | | 0.0 |
| session 5 | | | | | | | | | | | 0.0 |
| session 6 | | | | | | | | | | | 0.0 |
| session 7 | | | | | | | | | | | 0.0 |
| session 8 | | | 1.5 | | | | | | | | 1.5 |
| session 9 | | | | | | | | | | | 0.0 |
| session 10 | | | | | | | | | | | 0.0 |
| session 11 | | | | | | | | | | | 0.0 |
| session 12 | | | | | | | | | | | 0.0 |
| session 13 | | | | | | | | | | | 0.0 |
| session 14 | | | | | 2.0 | | | | | | 2.0 |
| session 15 | | | | | | | | | | | 0.0 |
| session 16 | | | | | | | | | | | 0.0 |
| session 17 | | | | | | | | | | | 0.0 |
| session 18 | | | | | | | | | | 1.0 | 1.0 |
| session 19 | | | | | | | | | | | 0.0 |
| session 20 | | | 1.5 | | | | | | | | 1.5 |

Tab. 14: Scoring per rule of the sessions in the clean dataset.

# 7 Conclusion

As drive-by infections become more and more mainstream we set out to answer the following research question:

*Can drive-by infections be discerned from legitimate sessions purely by measuring changes in* HTTP *traffic patterns and meta data?*

Our view is positive: yes, this is possible. We've identified a number of distinct properties that can be used to identify drive-by infections, many of which have a high predictive value. The validation confirms that the number of false positives and negatives is low: no false alarms were raised. Only a small number of infections were not detected; however we believe that combining our method with existing methods provides the opportunity to improve the chance of this last part to be caught after all.

Still, the scoring proposed within this paper needs to be balanced and the tests augmented in the future to cope with new developments.

## 7.1 Future work

During our research we encountered several issues that lead to interesting possibilities for future work.

*Expanding the set of characteristics.* We discovered that the malware world is susceptible to trends. During our research the Gumblar attack was very popular. Nearing the end of our research a new attack was discovered, called Nine-Ball. Further research may discover even newer characteristics. Another interesting aspect is to venture into a more widely defined set of malware by including user-assisted infections such as trojans disguised as fake antivirus programs.

*Session identification.* As we mentioned, our approach assumes that individual browsing sessions can be identified. There is no clear-cut solution for this available. We have identified some possibilities, but having a good method to identify separate browsing sessions within a stream of HTTP traffic would be very useful.

*Detecting fast-flux networks.* We encountered domains that resolve to a number of IP addresses with a short DNS TTL. These are so-called fast-flux domains: a common tool in malicious networks. Based purely on our HTTP analysis we can not identify those, but analysing DNS traffic combined with active tests may be an interesting addition to the method. The paper by Holz et al.[20] is a good starting point for that.

*Balancing the scoring.* Due to our limited time and relatively small dataset the chosen scores may not be balanced correctly. It might be possible that some characteristics are too heavily scored while others are not scored heavily enough. These scores can be better balanced when a larger dataset and 'real life' results are available.

## 7.2 Discussion

It must be noted that our data set has necessarily been gathered over a short time span. This can influence the results, and refinement of the method in the future is needed to cope with newly developed attacks. Also our data is gathered in a lab setup – inherently a simplified model of reality. Applying the method to a real-world traffic capture scenario may reveal areas that need improvement.

It remains to be seen how robust a rule-based attack is against malware developers coding specifically around it. We are under the impression that a number of characteristics are currently detectable because it's the easiest way to solve a given problem for the malware author, but with a little effort could be made to look more like legitimate traffic. This attitude may change and malware developers may even specifically work around known rulesets. Past experience with SpamAssassin has shown that even over years their general ruleset approach, albeit with constant improvement, is still feasible.

## References

[1] R. Thomas and J. Martin, *The underground economy: priceless*. Usenix ;login, 2006.

[2] Websense, *Canadian MSN Site Sympatico Compromised*. `http://securitylabs.websense.com/content/Alerts/3416.aspx` (published 10 June 2009, accessed 11 June 2009)

[3] Alexa Internet, Inc., *msn.ca – Traffic Details*. `http://www.alexa.com/siteinfo/msn.ca` (accessed 29 June 2009)

[4] M. Polychronakis, P. Mavrommatis, and N. Provos, *Ghost turns zombie: exploring the life cycle of web-based malware*. Proceedings of the 1st USENIX Workshop on Large-scale Exploits and Emergent Threats (LEET), 2008.

[5] Fox-IT, *Productsheet Managed Security Monitoring*. `https://www.fox-it.com/uploads/pdf/managed_security_monitoring_uk_copy1.pdf` (accessed 4 June 2009)

[6] N. Provos, P. Mavrommatis, M.A. Rajab, and F. Monrose, *All your iframes point to us*. Proceedings of the 17th USENIX Security Symposium, 2008, pp. 1–15.

[7] P. Tsiavos, E.A. Whitley and I. Hossein, *An exploration of the emergence, development and evolution of regulatory characteristics of information systems*. Proceedings of the 23rd International Conference on Information Systems, 2002, pp. 813–816.

[8] N. Provos, D. McNamee, P. Mavrommatis, K. Wang, and N. Modadugu, *The ghost in the browser: analysis of web-based malware*. Proceedings of the first workshop on hot topics in Botnets, USENIX Association, 2007, p. 4.

[9] Y.M. Wang, D. Beck, X. Jiang, R. Roussev, C. Verbowski, S. Chen, and S. King, *Automated web patrol with strider honeymonkeys*. Proceedings of the 2006 Network and Distributed System Security Symposium, 2006, pp. 35–49.

[10] A. Moshchuk, T. Bragin, S.D. Gribble, and H.M. Levy, *A Crawler-based Study of Spyware on the Web*. Proceedings of the 2006 Network and Distributed System Security Symposium, 2006, pp. 17–33.

[11] IBM Internet Security Systems, *Gumblar*. `http://www.iss.net/threats/gumblar.html` (published 19 May 2009, accessed 12 June 2009)

[12] D.K. McGrath and M. Gupta, *Behind phishing: an examination of phisher modi operandi*. Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats: USENIX Association, 2008, pp. 1-8.

[13] Multiscope, *Top 20 sites van 2009*. `http://www.multiscope.nl/top-20-sites-van-2008.html` (published 11 December 2008, accessed 9 June 2009)

[14] Adobe Systems Incorporated, *Adobe plug-in technology study.* `http://www.adobe.com/products/player_census` (accessed 5 June 2009)

[15] Websense Inc., *Nine-Ball Mass Injection – Details.* `http://securitylabs.websense.com/content/Blogs/3422.aspx` (published 22 June 2009, accessed 23 June 2009)

[16] N. Provos, *Top 10 Malware Sites.* `http://googleonlinesecurity.blogspot.com/2009/06/top-10-malware-sites.html` (published 3 June 2009, accessed 8 June 2009)

[17] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Mashinter, P. Leach and T. Berners-Lee, *Hypertext Transfer Protocol – HTTP/1.1.* Request for Comments 2616, The Internet Society, 1999.

[18] The Apache Foundation, *SpamAssassin.* `http://spamassassin.apache.org/` (accessed 24 June 2009)

[19] M. Spiliopoulou, B. Mobasher, B. Berendt, and M. Nakagawa, *A framework for the evaluation of session reconstruction heuristics in web-usage analysis.* INFORMS Journal on Computing, vol. 15, 2003, pp. 171–190.

[20] T. Holz, C. Gorecki, K. Rieck, and F.C. Freiling, *Measuring and Detecting Fast-Flux Service Networks.* Proceedings of the Network & Distributed System Security Symposium, 2008.