



UNIVERSITY OF AMSTERDAM

SECURING AN OUTSOURCED NETWORK: DETECTING AND PREVENTING MALWARE INFECTIONS

Dennis Cortjens
dennis.cortjens@os3.nl

Tarik El Yassem
tarik.elyassem@os3.nl

Abstract

Drive-by download infections pose a big threat against companies and organisations that don't have control over their end systems. This research tries to construct a system for detecting and preventing these infections by purely looking at the HTTP traffic on the network that is associated with drive-by downloads. This research resulted in a scalable HTTP based proxy solution with a scoring system for malicious HTTP traffic. Basic testing shows that this is a feasible approach, although more time is needed for implementing additional checks and researching a more balanced scoring system. The end result of the research is a proof of concept that consists of an open-source platform which provides a foundation for future research and development.

Acknowledgement

We would like to thank the T-Mobile security team for their hospitality, pleasant and thought provoking working environment. Especially we want to extend our gratitude to Ton van Ginkel and Ewout Meij for their trust and support.

Contents

1	Introduction	1
1.1	Problem	1
1.2	Scope	1
2	Background	2
2.1	Malware	2
2.2	Drive-by downloads	2
2.2.1	Workings	2
2.2.2	Research	3
3	Preparation	4
3.1	Base environment	4
3.1.1	Server	5
3.1.2	Clients	8
3.2	Previous research analysis	8
3.2.1	Geolocation score	10
3.2.2	Hostname score	11

3.2.3	Domain score	12
3.2.4	User-Agent score	14
3.2.5	POST score	15
3.2.6	Content-Type score	15
3.3	Infection history analysis	16
4	Implementation	17
4.1	A brief introduction to ICAP	17
4.2	c-icap	17
4.3	Implementing the inktvipAM checks	17
4.3.1	Hostname check	18
4.3.2	User-Agent check	19
4.3.3	POST check	20
4.3.4	Content-Type check	21
4.3.5	Domain check	22
4.3.6	Geolocation check	23
5	Test	25
5.1	facebook.com: 69.171.242.53	26
5.2	nu.nl	27
5.3	piratebay.org: 194.71.107.15	29
5.4	137.254.16.66	30
6	Conclusion	32
6.1	Achievements	32
6.2	Future research	33
6.2.1	Adding checks	33
6.2.2	Balancing scoring system	33
6.2.3	Testing live	33
6.3	General conclusion	34

1 Introduction

In the 90s outsourcing IT became increasingly popular [1]. Nowadays this is still widely done, mostly by outsourcing to subsidiary companies or by moving to the cloud. Outsourcing the IT service management of office automation has some benefits, like having one structural solution across the entire enterprise which eventually results in cost savings. There is however a down side to this kind of outsourcing. It has made client security increasingly difficult to manage for IT security departments. Outsourced IT service management may comply to security standards, but often there is a mismatch between the security standards of the service provider and the IT security department (client). Besides that the IT requirements change quickly due to technical and business evolution, while service level agreements (SLA) remain static over time. In many cases the SLA is based on business oriented continuity and risk management, which makes most software updates/upgrades a possible risk for the IT infrastructure uptime. This may result in situations where clients run old and insecure configurations.

Another upcoming trend that causes a similar IT security challenge, is the 'bring your own device' concept. Users may bring and use their own devices to connect to the business IT infrastructure and start using them for work purposes. In this concept, the user is responsible for maintaining the device and its security. This is also a possible threat to the IT infrastructure of the business and a concern for IT security departments.

1.1 Problem

In outsourced IT service management and the 'bring your own device' concept there is a high risk of getting infected by malware. Old and insecure configuration and personal devices of employees without proper security software (updates) are vulnerable to exploits which can lead to malware infections. These infections can be caused by various threats such as drive-by downloads and rogue applications that are installed by users. This is hard to prevent in cases where the service provider has the control over the clients and does not provide a service to prevent these malware infections. The IT security departments want to prevent these infections, because critical company data can be exfiltrated or a company computer can be used for illegal or rogue activities. This could be done by breaking the end-to-end connectivity between clients and the internet. The life cycle of this problem is showed in figure 1.



Figure 1: Malware infection triangle

1.2 Scope

The main research question in this study is:

How could malware infection attempts be detected and prevented from within the IT infrastructure of a business that has outsourced IT service management or that allows 'bring your own device'?

This question is researched with the following subquestions:

1. What kind of attacks cause the largest percentage of infections?
2. What are the characteristics of these attacks?

3. Can these characteristics be used to prevent these attacks?
4. Can these attacks be prevented from a network perspective?
5. What are the best methods to prevent these attacks?
6. What kind of solutions are already available and is it possible to combine these?

We shall focus on analyzing HTTP traffic on port 80. Our aim is to come up with a practical solution and deliver a proof of concept to provide a platform for future research. The detection must be conducted on a network level without involving end points. Finally, we strive to implement our system with open source software.

2 Background

2.1 Malware

Malware, short for malicious software, is software designed to disrupt computer operation, gather sensitive information or gain unauthorized access to computer systems. It is a general term used to describe any kind of software or code specifically designed to exploit a computer, or the data it contains, without consent of the user. The expression is used by computer professionals to mean a variety of forms of malicious software. Malware has come into existence when the first computer virus, Brain [2], was released in 1970. Since then, the use of malware has increased in popularity, mostly amongst computer criminals. In recent years it is also used by various government institutions such as law enforcement and intelligence agencies. The main goal of computer criminals is to use malware in order to gain wealth. There are various ways that these criminals utilize malware in order to make a profit. The most common way is to infect systems with so called bots, that are controlled by the criminal through a botnet. These bots may perform a variety of functions. For example, sending SPAM email messages, catching keystrokes in order to obtain passwords or manipulating online banking transactions. The authorities mainly use malware to eavesdrop on citizens [3], for espionage on countries or companies, or to clean infected systems [4]. Although technically possible, no information is available on mass infections by government sanctioned malware. Most infections are caused by cyber criminals, who seek to infect as many systems as possible in order to maximize the profit.

A malware infection process can occur in multiple stages, all of which have been researched extensively. Common infection vectors are drive-by downloads, SPAM, trojanised software, worms, infected data carriers (such as USB drives) and targeted attacks. Malware is usually part of an attack in one way or another. It is usually spread and managed by botnets [5].

2.2 Drive-by downloads

We suspect that drive-by downloads are one of the biggest causes of malware infections. The reason we suspect this is because of the following factors:

- web traffic is popular
- web traffic is not much filtered
- web servers are common
- there are many vulnerabilities in web related software

Furthermore, web programming languages allow much flexibility and provide power while still providing access to a multitude of platforms. This allows an attacker to easily infiltrate legitimate websites, create attack tools, attack systems and hide their tracks. A single website intrusion may allow a huge number of end user systems to be attacked. There have been many incidents where intrusions into popular websites have lead to mass compromise of end users [6] [7] [8] [9] [10]. It's a particularly potent attack, because it uses popular, mainstream, trusted sites to attack its many visitors.

2.2.1 Workings

A drive-by download attack consists of several stages in order to exploit and infect systems. The first stage consists of an attacker finding and exploiting a vulnerability in a service that provides web content directly

or indirectly to a given website. For example, an attacker might exploit a vulnerability in the web server or gain access to an advertisement service that places advertisements on third-party websites. The main aim of this stage is that the attacker will become able to inject code (1) that will instruct the victim's web client program to make a connection to another server, one that is controlled by the attacker. The second stage occurs when a user visits the compromised website (2). The malicious script inserted by the attacker will cause a redirection (3) to a system that hosts an exploit to attack the victim's web client. This web client software might be a browser, a plugin or any other web client application. In most cases a multitude of exploits is served to increase the range of vulnerable client software. After a client has been successfully exploited, arbitrary code can be executed on the client. This code usually instructs a client system to execute the next stage of the infection process: downloading malware (4). In this stage the client is infected with malware. Usually the malware is hosted on another system instead of the system that hosts the exploits. After infection there might be further downloads of malware, but formally the drive-by download stages have been completed. Each stage might consist of multiple layers of redirection and obfuscated web content in order to make detection and prevention more difficult. The stages of a basic drive-by download scenario are displayed in figure 2.



Figure 2: Drive-by download stages

2.2.2 Research

Drive-by download detection has been previously researched, but mostly from a client perspective. Wang Tao, Yu Shunzheng and Xie Bailin have researched a system to detect drive-by downloads by analyzing client-side behavior [11]. Konrad Rieck, Tammo Krueger and Andreas Dewald also described a system that can detect drive-by downloads by emulating a client and analyzing it [12]. Aikaterinaki Niki described drive-by downloads and suggested detection mechanisms based on analyzing Domain Name Server (DNS) and webserver relationships and combining static heuristics with client honeypots [13]. Julia Narvaez, Barbara Endicott-Popovsky,

Christian Seifert, Chiraag Aval and Deborah A. Frincke researched drive-by downloads by utilizing a client-side honeypot and analyzing it's data [14]. Marco Cova, Christopher Kruegel, and Giovanni Vigna researched the detection and analysis of malicious JavaScript code in order to identity drive-by downloads [15]. Van Lam Le, Ian Welch, Xiaoying Gao, Peter Komisarczuk described ways to detect malicious pages by analysing certain aspects of the webpage content itself [16].

Kinkhorst and Van Kleij have researched detection of drive-by downloads by measuring HTTP traffic patterns and meta data. They have identified a number of distinct properties that can be used to identify drive-by download infections [17]. These methods can be applied from a network perspective and do not require the use of client-side data analysis. Malicious infrastructures have been and continue to be extensively researched. Application of this research has lead to the availability of much information on these malicious systems that can be used to prevent infections. We believe that researching, improving and applying the approach of Kinkhorst and Van Kleij together with implementing blacklist and anti-virus services based on information from public sources will significantly decrease infections and exfiltration.

3 Preparation

3.1 Base environment

The base environment was set up in a virtual environment with VMware vSphere ESXi 5. This is a server operating system in which multiple virtual networks and virtual computers (servers and clients) can be installed and configured. To simulate a real business network infrastructure, two networks had to be created. One as a border network to the internet, called Virtual Network OUT, and the other as an internal business network, called Virtual Network IN. Within these networks a proxy server was set up with two network interface cards (NICs). One (eth0) to connect to the OUT network and the other (eth1) to connect to the IN network, making the proxy server a router for connecting both networks with each other. To simulate a real outsourced client computer, an default image of the business was used to configure three clients in the network. This is schematically shown in figure 3.

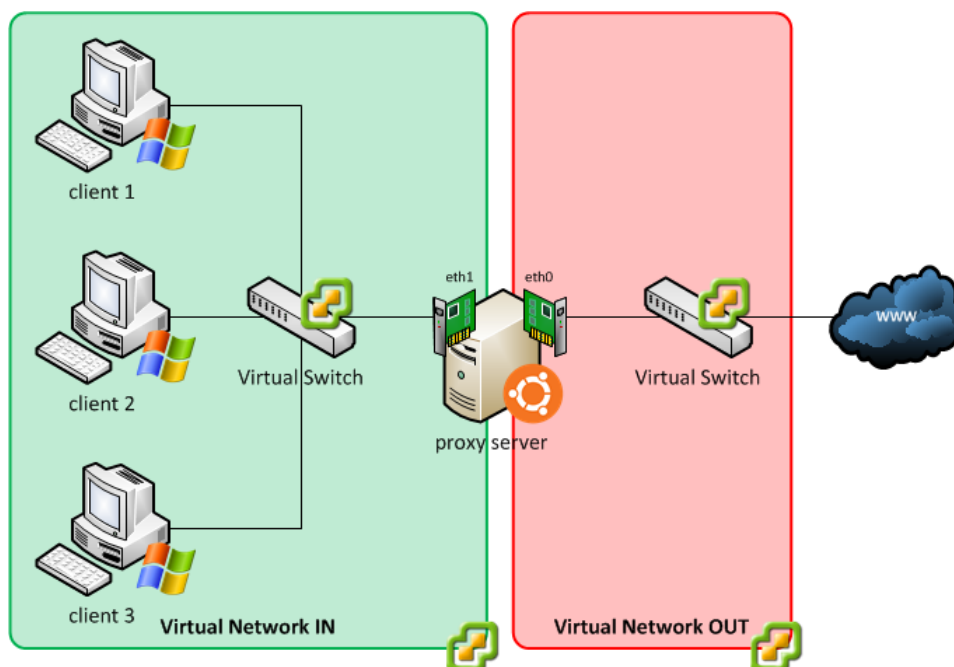


Figure 3: Network topology in VMware vSphere ESXi

3.1.1 Server

The proxy server was set up with the Ubuntu Server 11.10 operating system. The server was configured with the needed open-source applications and configurations. The name of and cooperation between these applications is schematically shown in figure 4 and are explained in the following paragraphs.

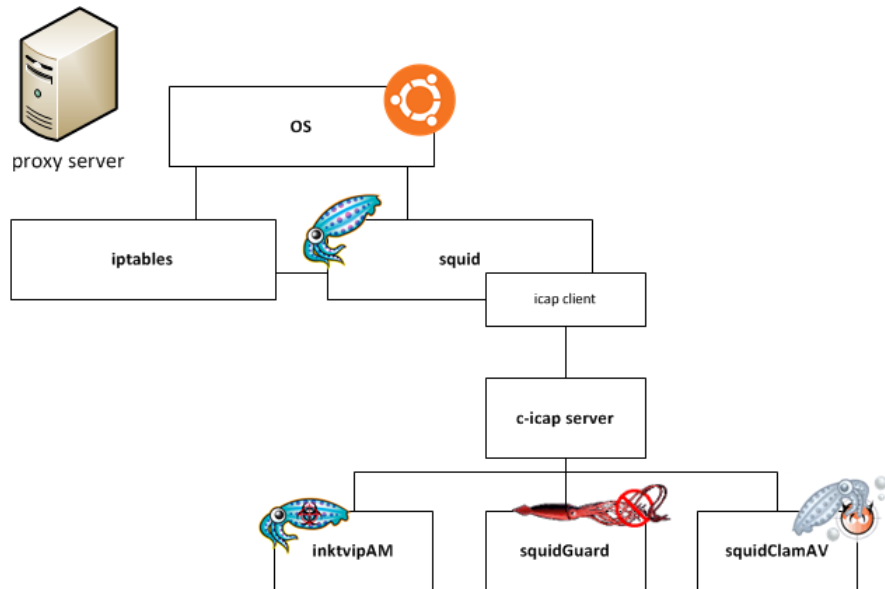


Figure 4: Software topology on proxy server

Router In order to be able to forward web traffic to the proxy server and to act as a router, a number of settings in the `/etc/sysctl.conf` file were enabled. In order to reduce complexity, IPv6 was kept out of scope. The configurations therefore only apply to IPv4. The relevant `sysctl.conf` configuration options are shown in figure 5.

```
# Uncomment the next two lines to enable Spoof protection (reverse-path filter)
# Turn on Source Address Verification in all interfaces to
# prevent some spoofing attacks
net.ipv4.conf.default.rp_filter=1
net.ipv4.conf.all.rp_filter=1

# Uncomment the next line to enable TCP/IP SYN cookies
# See http://lwn.net/Articles/277146/
# Note: This may impact IPv6 TCP sessions too
net.ipv4.tcp_syncookies=1

# Uncomment the next line to enable packet forwarding for IPv4
net.ipv4.ip_forward=1

# Uncomment the next line to enable packet forwarding for IPv6
# Enabling this option disables Stateless Address Autoconfiguration
# based on Router Advertisements for this host
#net.ipv6.conf.all.forwarding=1
```

Figure 5: sysctl configuration

Firewall An iptables firewall was configured to route the traffic from the IN network to the OUT network with Network Address Translation (NAT). The HTTP traffic on port 80 is forwarded to port 3128. This is the default port on which the proxy server is listening. The iptables configuration is shown in figure 6.

```
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT
iptables -t nat -A PREROUTING -i eth1 -p tcp --dport 80 -j REDIRECT --to-port 3128
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
iptables -A INPUT -i eth0 -p tcp --dport 22 -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp -m multiport --dports 80,443 -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A INPUT -i eth0 -p tcp -m multiport --sports 80,443 -m state --state ESTABLISHED -j ACCEPT
```

Figure 6: iptables configuration

DHCP A DHCP server was installed and configured to automatically assign an IP address to all clients on the internal network (IN). As DHCP server, the ISC DHCP server was chosen. All clients receive an internal IP address according to RFC1918 and a router setting that points to the server. The router option was used for the clients to send all outgoing traffic to the server by default. The relevant dhcpd configuration is shown in figure 7.

```
# Configuration for the proxy server
subnet 192.168.3.0 netmask 255.255.255.0 {
range 192.168.3.2 192.168.3.254;
option domain-name "proxy";
option domain-name-servers 145.100.96.11, 145.100.96.22;
option routers 192.168.3.1;
}
```

Figure 7: dhcpd configuration

Note: the domain name servers are those of the University of Amsterdams OS3 network and should be replaced by own domain name servers.

Proxy server The squid proxy server was used as the HTTP proxy. squid is a caching proxy for web traffic. It can reduce bandwidth and improve response times by caching and reusing frequently-requested web pages [18]. squid is used by many companies including Internet Service Providers (ISPs). squid offers many features, one of which is transparent proxying. A transparent proxy will act as a proxy for a given client without any need for client side configurations. squid can be configured to proxy Secure Sockets Layer (SSL) traffic. This would be achieved by acquiring a certificate and configuring it on the server. The Certificate Authority (CA) that issued the certificate must be configured on all clients. For our research, SSL was not specifically relevant and therefore SSL was kept out of scope. The relevant squid configuration for a basic transparent proxy server is shown in figure 8.

```
# This allows web traffic from the internal network
acl localnet src 192.168.0.0/16 # RFC1918 possible internal network

# Squid normally listens to port 3128
http_port 192.168.3.1:3128 transparent
```

Figure 8: squid configuration

Note: the squid caching option has to be disabled, otherwise websites are loaded from the cache and aren't checked.

ICAP server The Internet Content Adaption Protocol (ICAP) is used to process and adapt the HTTP request and response headers. It is documented in RFC3057 [19] and will be explained in 4.1 A brief introduction to ICAP. There are ICAP servers written in different languages, like Python, C and Java. All of them have the same features, but not all of them are well maintained. The Python version is from 2002, a last

commit has been made to its repository in 2010. Attempts to get this ICAP server functioning with a basic HTTP response modification function failed. However the C version, called `c-icap`, is very well maintained and the last source is from 2011. `c-icap` offers a way to build custom modules in C.

squidguard To block and/or redirect known malicious URL squidguard can be used. squidguard is an open source URL redirector. It can block and/or redirect specific URLs according to blacklists. There are many different blacklists available, both free and commercial. Most blacklists are arranged by topics such as advertising, gambling, porn or malicious content. squidguard can be configured to handle specific blacklists. Squid offers a redirector feature which allows URLs to be redirected to another service. This can be enabled by including the squid configuration option specified in figure 9.

```
# Redirect url's to squidguard
url_rewrite_program /usr/bin/squidGuard -c /etc/squid3/squidGuard.conf
```

Figure 9: squid redirection option with squidguard

squidclamav Binaries that are downloaded can be scanned for virus signatures by the ClamAV open source virus scanner. It is possible to use ClamAV in conjunction with squid, so it can scan binaries downloaded via HTTP. For this, the squid redirector can't be used since squid only allows one redirector. However, squid offers another option to allow an external service to process HTTP traffic through ICAP. squid supports ICAP since version 3.0 and is RFC compliant since version 3.1. An ICAP client is shipped with the squid since version 3.0. The squidclamav application combines ClamAV with an ICAP service to interface with squid and process HTTP traffic. HTTP traffic can also be redirected by squidclamav and squidguard. This allows a multitude of different setups by combining redirection from squid with ICAP connections and redirection by squidclamav or squidguard. These different setups are beyond the scope of this project, but may be an interesting research subject. To scan HTTP traffic with squidclamav through the ICAP service, squid can be configured as illustrated by figure 10.

```
icap_enable on

icap_service service_req reqmod_precache bypass=0 icap://127.0.0.1:1344/squidclamav
adaptation_access service_req allow all

icap_send_client_ip on

icap_service service_resp respmod_precache bypass=0 icap://127.0.0.1:1344/squidclamav
adaptation_access service_resp allow all

logformat icap_squid %t1 %a %>p %<A %la %lp %<la %<lp %tr %dt
icap_log /var/log/squid3/icap.log icap_squid
```

Figure 10: squidclamav as ICAP service in squid configuration

Note: the configuration parameters are slightly different between squid version 3.0 and 3.1. The above parameters are applicable to version 3.1. ICAP servers can send log information to squid which can write them to a specified log file. The log format can be configured to specific needs. ICAP services can be configured to be bypassed by the proxy in certain cases, a setting not recommended in the case of security applications such as virus scanning.

inktvipAM The sections 3.1.1 squidguard and 3.1.1 squidclamav described a way to prevent known malicious domains to be accessed and known malicious binaries to be downloaded. However, this is not sufficient protection against drive-by downloads. One of the main aspects of drive-by downloads is that they use compromised legitimate websites for at least a part of the process. Furthermore, with drive-by download infections the malware is most likely unknown to anti-virus engines, because new malware specimens are likely to be generated for a specific drive-by download attack. In order to successfully protect against drive-by downloads, additional measures are necessary. These measures can be implemented as an ICAP service in much of the same way as squidclamav and squidguard. To accomplish this, we introduced the inktvipAM

module. It's name was derived from a Dutch commercial of a telecommunications company, illustrating a way of dealing with communication errors due to modern technology. The Dutch word for a squid is *inktvip*, but misspelled due to automated spell checking of a mobile phone it is showed as *inktvip*. AM stands for anti-malware. We thought it is an appropriate name.

3.1.2 Clients

The clients have a default image from the outsourcing IT company and run Microsoft Windows XP with Service Pack 3, including Microsoft Internet Explorer 6.0. They have some basic software installed, like Adobe Reader X, Microsoft Office 2003 and the Java Runtime Environment 1.6.20. A user has no administrative rights and can't install or update anything.

3.2 Previous research analysis

Previous research by Kinkhorst and Van Kleij identified the following distinct properties that can be used to identify drive-by infections:

- TCP port number
- Geographical location
- Hostname
- User-Agent
- POST request
- URI
- Content-Type
- Redirection

Not all of these properties are easy to implement or a good way of detecting drive-by downloads. The TCP port number is such a property. Kinkhorst and Van Kleij discovered that 40% of their infected data had at least one connection to port 8080 and that 8% had at least one connection to port 443. These numbers can be doubted, because the data set for infected website was very small and because these days it's very easy and cheap to temporary register a domain. So it isn't necessarily to hide the port 80 traffic from the administrator anymore. This property is however beyond the scope of this project, because there was a focus on HTTP traffic on port 80. However, TCP port number information is only available on the transport layer of the Open Systems Interconnection (OSI) model and isn't present at the application layer anymore, where the HTTP header can be found. A detailed view of the OSI model is shown in figure 11.

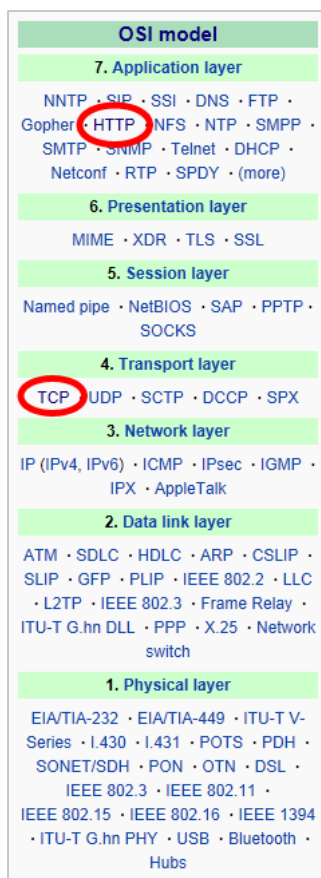


Figure 11: OSI model (wikipedia)

The redirection property is also one which is hard to implement. The HTTP request header has an `Referer` field, but sites can be redirected in many ways. So this field isn't always filled with data. To be able to capture all these redirections, the capturing of sessions is needed. When there is a lot of traffic from different clients, it is hard to distinguish which HTTP request belong to a certain session. Kinkhorst and Van Kleij discovered that infected websites have more redirections than clean websites. De Kok and Bakker have done research in this field and came up with a proof of concept [20]. This approach isn't applicable for this project, because it ends with the the same problem as with the TCP port number property. The information needed to distinguish sessions is available at the transport layer and not at the higher application layer.

A good malware scoring system which will classify every HTTP request, doesn't need to identify redirections and sessions. If the system is able to classify every request, it will find malicious traffic when redirected from a site and doesn't need to know its former request(s). This can be achieved with the right checks and balancing the scoring system. However, identifying redirections could result in an extra score point and thus a higher total score which eventually can result in blocking a request.

The Uniform Resource Identifier (URI) property isn't a good way of detecting drive-by downloads. The URI is specified as the path name that is referenced in requests. Kinkhorst and Van Kleij mentioned substrings, especially the end of the path names which should identify that something is wrong and could be an infected websites. These findings aren't founded by any other research and therefore are not used in this project. There aren't any specifications for naming conventions in an URI, so it's hard to filter traffic upon them.

The other properties were all used to create a malware scoring system and for this system a main flow was created in which the data is classified. Every property has its own checks which will calculate a score. All the scores are summed and will result in a total score. This total score is then used to classify the HTTP request. A request with a score higher than y will be classified as malicious traffic and will be blocked. A score between x and y will be classified as suspected traffic and will be logged. All the scores below x are classified as unsuspected and corresponding traffic will be allowed. The malware scoring system is illustrated in a flowchart in figure 12.

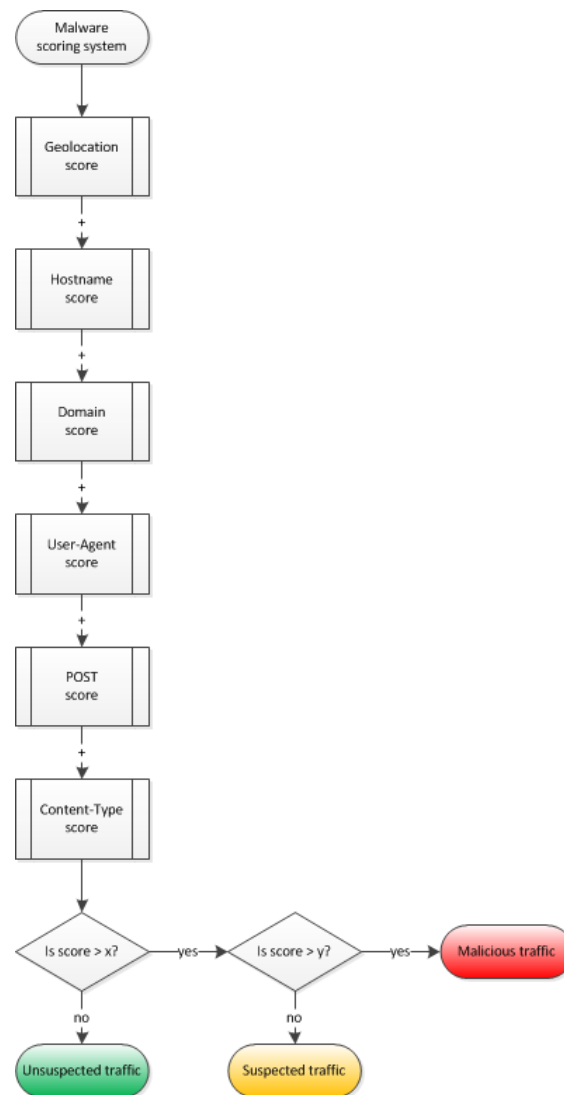


Figure 12: Malware scoring system flowchart

3.2.1 Geolocation score

The geolocation score is able to locate the country in which the website is hosted based upon the IP address. Together with a top 10 list of malware hosting countries this can result in a score for well-known malware hosting countries. Kinkhorst and Van Kleij mentioned in their research three countries that came up with all their infected websites. These countries were China, Russia and Ukraine. However a foundation for the countries was not given. To find a top 10 list of malware hosting countries, research was done by looking for official reports from the year 2011 by well-known anti-virus companies. This resulted in three reports from Sophos [21], Kaspersky [22] and BitDefender [23]. These reports had some differences in listing countries,

but by calculating an average for each country and combining them, a list was created. This resulted in a founded top 10 list of malware hosting countries. This list is shown in table 1.

Rank	Country	Code	Percentage
1	United States	US	24.69
2	Russia	RU	15.69
3	China	CN	13.81
4	Brazil	BR	8.40
5	Netherlands	NL	7.82
6	Germany	DE	7.43
7	France	FR	6.15
8	United Kingdom	UK	4.86
9	Spain	ES	4.11
10	Ukraine	UA	3.29

Table 1: Malware hosting countries top 10

Besides a check whether or not a country is listed within the top 10, another check involving the Referer field is conducted. This field consists of the address of the previous web page from which a link to the currently requested page was followed. This field can eventually result in an extra score point and blocking the request. The flowchart of the geolocation score is illustrated in figure 13.

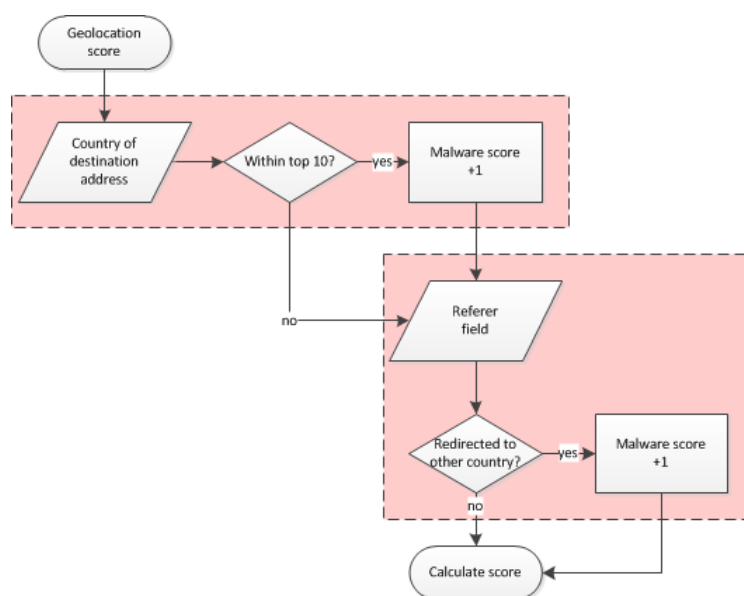


Figure 13: Geolocation scoring flowchart

Note: checks within the red borders are not implemented in the concept.

3.2.2 Hostname score

The hostname score is responsible for determining whether or not the Host field in the HTTP header is empty. This field is the domain name of the server. It is a mandatory field according to the HTTP/1.1 specification (RFC2616) [24]. So leaving this field empty, points to being a bad implemented and maybe malicious site. Kinkhorst and Van Kleij also did some research on this field by looking at two label domains, multi label domains, IP addresses and the .cn domains. They discovered a notable higher number of two and multi label domain with their infected websites and a significantly higher number for IP addresses. These finding are definitely true and can be founded by analysing the data at the Malware Domain List website [25]. However

they can't really distinguish clean and infected websites anymore with the coming of short URLs (TinyURLs). The .cn domain is another matter. In none of the reports from last year by anti-virus companies the .cn is mentioned as the top listed and high risk malware domain. Therefore the hostname score is only determined by the check whether or not the Host field is empty as illustrated in figure 14. Additional checking on domain characteristics is conducted by the domain score which is mentioned in 3.2.3 Domain score.

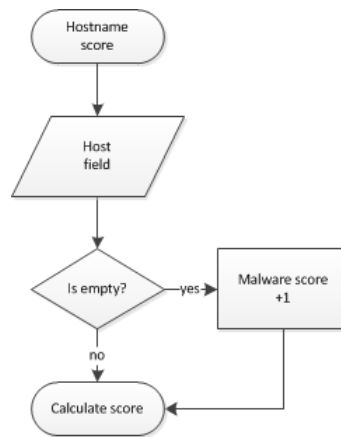


Figure 14: Hostname scoring flowchart

3.2.3 Domain score

In addition to the hostname score, the domain score was created. The domain is extracted from the URL in the HTTP header and not from the Host field, because this field can be left empty and the URL can't. The domain score check consists of five checks. The first one is derived from the research by Kinkhorst and Van Kleij and is introduced in 3.2.2 Hostname score. It checks whether or not the domain is an IP address. The second one checks for a country code top-level domain (ccTLD) and determines whether or not the ccTLD and geolocation of the domain are the same. This check uses the result from the geolocation score which is mentioned in 3.2.1 Geolocation score. The check is introduced, because clean websites with a ccTLD are normally hosted in the corresponding country. A difference could suggest a possible malicious site. The third one checks the registration date of the domain. Malicious sites are oftenly hosted in the free trial period of a hosting provider which differs from a week to a month. A website with a registration date younger a month or even a week could also be a possible malicious site. Together with the other checks this could eventually result in blocking the request. The last two checks are character related. The first (fourth check) checks a dictionary for known words. This is a good way of checking for a domain registrations with company, person, etc. related names. It can also be configured with a certain language or multiple languages. The other (fifth check) checks the alphanumeric ratio of characters to determine the randomness of the domain name. Both are founded on research by Kinkhorst and Van Kleij, who discovered auto-generated domain names within their infected site data set. It is also founded by the Malware Domain List website, where a lot of random domain names are discovered. The flowchart for the domain score is illustrated in figure 15.

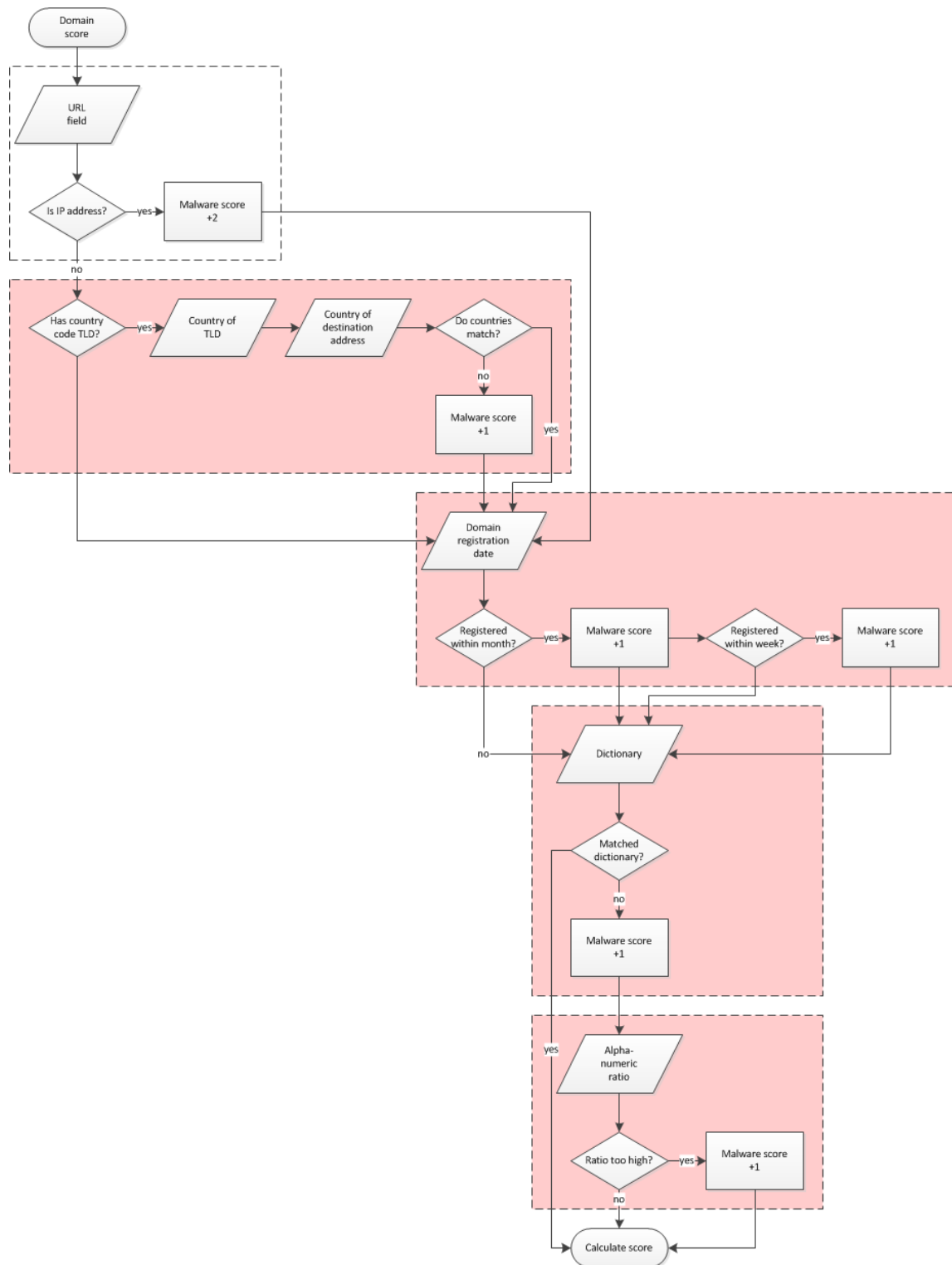


Figure 15: Domain scoring flowchart

Note: checks within the red borders are not implemented in the concept.

3.2.4 User-Agent score

The User-Agent field is an important field in the HTTP header. Although it can't be used to prevent malware infections, it can block traffic from an infected client. Kinkhorst and Van Kleij mentioned that malware programs often use other information within the User-Agent field than the default browsers do. They even found cases of the field being empty. So this is a good way of preventing rogue activities of an infected system, when the website that hosts the malware isn't blocked the first time. The User-Agent score therefore performs four checks which should be able to block the traffic. The first check is simple and only checks for an empty User-Agent field. The second check determines the User-Agent field consists of a known blacklisted User-Agent. These blacklisted User-Agents can be found on the internet. The last two checks perform whitelist checks. The User-Agent string format is specified in the HTTP/1.1 specification. It consists of a render engine part and a browser part. The first one (fourth check) determines the User-Agent field consists of a known (whitelisted) render engine which is normally formatted Mozilla/x.0. The second one (fifth check) determines the field consists of a known (whitelisted) browser which can be found for every browser version on the ZYTRAX website [26]. These blacklists and whitelists are definable. They can be filled with specific data, like with only Microsoft/Windows Internet Explorer User-Agents to be sure no one within the network are able to use other browsers. The checks are illustrated in a flowchart in figure 16.

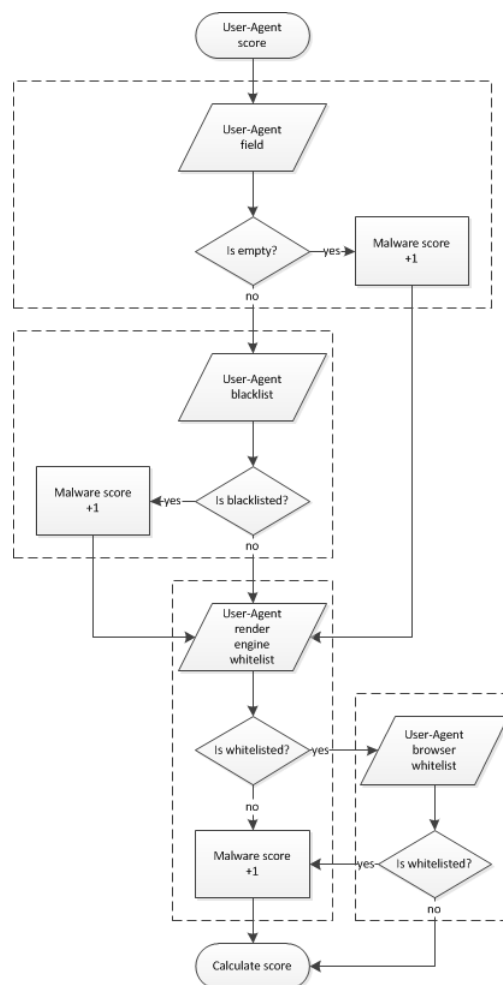


Figure 16: User-Agent scoring flowchart

3.2.5 POST score

The POST score is responsible for comparing the HTTP body length. The HTTP request header has a `Content-Length` field and the length in this field should be equal to the actual body length. Kinkhorst and Van Kleij discovered that in their clean sessions the `Content-Length` field and body length were always equal. In their infected sessions 56% of the requests had an unequal length. This suggests that HTTP requests with an unequal length could be an infected website and together with the other checks this could eventually result in blocking the request. The flowchart for this check is illustrated in figure 17.

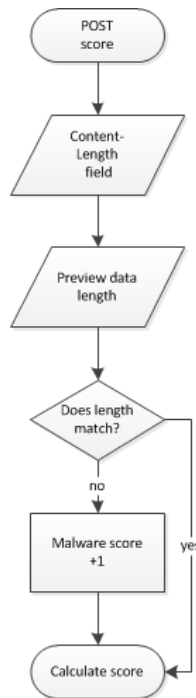


Figure 17: POST scoring flowchart

3.2.6 Content-Type score

The `Content-Type` field is an interesting field within the HTTP header. This field shows the object type of the request, like a text page (`text/html`), a picture (`image/png`) or a Java page (`application/java`). These object types have been researched by Kinkhorst and Van Kleij. They discovered a higher number of HTTP requests with the `application/octet-stream` object type within their infected sessions. This could be doubted, because this object type is often used to download legitimate files. This is also definable and can be filled with specific data, like not allowing any file downloads within the network. The `application/java` is however used often to infect a computer with malware. This is founded by Malware Domain List website and the infection history analysis which is mentioned in 3.3 Infection history analysis. These findings resulted in checking for the `application/octet-stream` and `application/java` object type within the `Content-Type` score which is illustrated in 18.

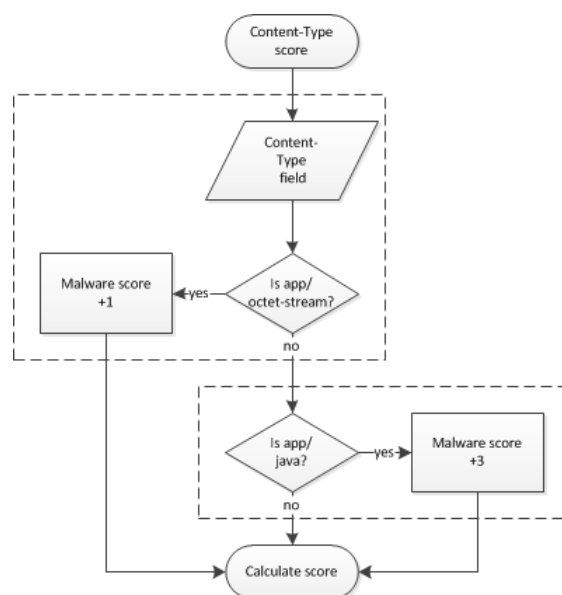


Figure 18: Content-Type scoring flowchart

3.3 Infection history analysis

To verify the relevancy of the checks suggested by Kinkhorst and Van Kleij, we manually analyzed infection data. The data we used were analysis reports of actual malware infections within a business IT infrastructure. The reports were drafted by security specialists and contained, amongst others, the HTTP traffic before, during and briefly after an infection occurred. Because this data is highly sensitive, we will refrain from reporting the outcome in general terms. Not all checks were possible by hand due to various reasons. One of the reasons was that the infection data did not contain some information such as the body size of a post request. Furthermore, some checks require to be ran on live sites. However, most of the infection data was not recent enough for the sites to be available in a compromised state or available at all. The number of unique infections was relevant enough to gain useful information on the effectiveness of the checks that we were able to verify manually. For each infection, each check was applied and if applicable we would mark the given check. After analysing all infections, it became clear that some checks were more effective than others. This information allowed us to prioritize the order in which we would develop the checks. The most telling aspects of drive-by download traffic patterns analysed using this method were:

- URL contains string that is not to be found in a dictionary (semi-random sequence of characters)
- specific TLDs used, either specific ccTLDs (.in, .cn and .ru) or non country code (.info)
- URL is an IP address

The letter to number ratio check suggested by Kinkhorst and Van Kleij also scored high on relevancy, except that for many URLs the specific ratio mentioned was not applicable. Many of the auto-generated URLs had a slightly different construction of letter/number combinations.

From the infection data, we also found some other aspects that signified traffic from or to a malicious source. We found the following aspects in a significant number of infection reports:

- encrypted body in HTTP request or response
- GET requests with a very large URL
- long filenames in downloaded content (verrylongstringofcharactersthatseemtooonforever.zip)
- downloads of files with a suffix like .jar, .class or similar Java related
- downloads of files with a suffix like .org.class, .net.class or similar
- Content-Types of application/java

4 Implementation

4.1 A brief introduction to ICAP

The ICAP protocol allows for redirection of HTTP traffic in order to be read and or modified by an external service. A proxy server that supports ICAP can encapsulate all or specific HTTP request and response packets within an ICAP message to be sent to a specified external service. The ICAP server is the receiver of the ICAP message, sent by the ICAP client. In our implementation, the ICAP client resides on the proxy and sends the ICAP messages to the ICAP server for processing. This external service can access the given HTTP request or response data and modify both header and body. In ICAP terminology the access or modification of HTTP request or response messages is called `reqmod` or `respmod` respectively. Adaptation is a term used in ICAP to refer to the processing or modification of HTTP messages. `Reqmod` messages are used to either modify the content of a request message or to send back an HTTP response to the ICAP client. For example, to display an error message. A typical application where `reqmod` is used is a URL blacklist ICAP service such as `squidguard`. The HTTP request is processed and the requested url is scrutinized. If the URL is on the blacklist, an error message is sent back to the ICAP client, for forwarding to the originating end point. `Respmod` returns either a modified HTTP response or an error message. `Respmod` is used for anti-virus engines, to automatically translate pages or to remove certain content such as flash objects. ICAP responses must start with an ICAP status line that is similar to that of HTTP. It also uses status codes in this line, but the specific codes are different from HTTP status codes. ICAP requests from client to server can include a preview. The preview is a feature of ICAP that allows an ICAP server to receive a small portion of a transaction in order to decide early on if it wants to process the remaining part of a message or not. This feature is useful for many different services, anti-virus or content scanning to name a few. It's main function is to increase performance since decisions of the service might be possible on a fraction of the HTTP packet. This brief introduction to ICAP should be sufficient for understanding the implementation of our checks. Refer to RFC3057 for the full ICAP protocol specification [19].

4.2 c-icap

After failing to successfully run an ICAP service with the python based ICAP server, we switched to `c-icap`. `c-icap` is an open source implementation of an ICAP server and is actively maintained. C-ICAP is used by `squidclamav`. Many commercial anti-virus and content scanning products also use `c-icap`. Next to the `c-icap` server package, a modules package is also available that contains two ICAP service implementations. The first service is `srv_clamav`, a service that uses ClamAV for virus scanning. The second service is a basic URL blacklist implementation called `srv_url_check`. The ICAP server package contains an echo service, a basic implementation to demonstrate an ICAP service. It only echoes back the original HTTP request. We used this echo service as a starting template to implement our checks. To use this echo service, the `c-icap` server package must first be built using the accompanying `configure` and `make` files. We also implemented a redirection to a 403 error page to be used when a page is blocked. Next, the ICAP service must be added to the `squid` configuration, much like the configuration for `squidclamav`.

4.3 Implementing the inktvipAM checks

We used the echo service distributed with the `c-icap` server package as the starting point for our implementation. However, we also used the `srv_clamav` and `srv_url_check` as guidelines to implement blocking and redirection. The first goal was to find the line of code where the incoming HTTP request was processed. We found this to be in the `echo_check_preview_handler` function. As described in the brief ICAP introduction, this function deals with the handling of preview data. The `c-icap` specification describes certain response codes that can be used in order to control the flow of preview data handling. We use these status codes to either let a request through, respond to it with a modified HTTP response that contains a warning or block it completely. As was shown in the flowchart that specifies the checks, this decision is made on the sum of each individual check score. The code snippet in figure 19 illustrates the code for adding up all scores to calculate the final score. Then, the score is classified as either malicious, suspected or unsuspected and consequently the relevant status code is returned.

Listing 1: snippet from `srv_inktvipam.c`

```
1  /* sum all the scores */
2  int final_score = 0;
3  final_score = geo_score + hostname_score + domain_score + user_agent_score + post_score + content_type_score;
4
5  /* classify the score */
6  if (final_score >= 5) {
7      ci_debug_printf(1, "DEBUG malicious traffic! Blocking traffic! TOTAL MALWARE SCORE: %d\n", final_score);
8      generate_redirect_page(req);
9      return CI_MOD_DONE;
10 }
11 else if (final_score >= 3) {
12     ci_debug_printf(1, "DEBUG suspected traffic! Allowing, but logging traffic! TOTAL MALWARE SCORE: %d\n",
13         final_score);
14     return CI_MOD_ALLOW204;
15 }
16 else {
17     ci_debug_printf(1, "DEBUG unsuspected traffic! Allowing traffic! TOTAL MALWARE SCORE: %d\n", final_score);
18     return CI_MOD_ALLOW204;
19 }
```

Figure 19: Score calculation, classification and decision

We shall now describe the implementation of each individual check and include code snippets to illustrate interesting pieces. As a refresher, we implemented the following checks:

- Hostname
- User-Agent
- POST
- Content-Type
- Domain
- Geographical location

4.3.1 Hostname check

Implementing the hostname check was rather trivial once we found out there was a function in the `c-icap` API that could retrieve fields from the HTTP request header. We already found the original request data variable and only needed to call the right function in order to get the Host field value. The actual check itself is merely checking if the field is empty or not.

Listing 2: snippet from `srv_inktvipam.c`

```
1  /* Hostname check > hostname_check.c */
2  char *hostname = ci_http_request_get_header(req, "Host");
3  int hostname_score = 0;
4  hostname_score = hostname_check(hostname);
```

Figure 20: Retrieving the hostname field through the `c-icap` API function

Listing 3: hostname_check.c

```
1 int hostname_check(char *hostname);
2
3 int hostname_check(char *hostname) {
4     int h_score = 0;
5     ci_debug_printf(1, "DEBUG HTTP Hostname: %s\n", hostname);
6     if (hostname != NULL) {
7         ci_debug_printf(1, "DEBUG HTTP Content-Length is unsuspected! MALWARE SCORE: %d\n", h_score);
8     }
9     else {
10        h_score += 1;
11        ci_debug_printf(1, "DEBUG HTTP Content-Length is empty! MALWARE SCORE: %d\n", h_score);
12    }
13    return h_score;
14 }
```

Figure 21: Implementation of the hostname check

4.3.2 User-Agent check

Another field we could easily access was the User-Agent field. The same method was applied as in getting the hostname value. However, implementing the blacklist and whitelist was a bit more work. String parsing in C is a tedious task compared to modern day scripting languages such as Python.

Listing 4: snippet from srv_inktvipam.c

```
1 /* User-Agent check > user_agent_check.c */
2 char *user_agent = ci_http_request_get_header(req, "User-Agent");
3 int user_agent_score = 0;
4 user_agent_score = user_agent_check(user_agent);
```

Figure 22: Retrieving the User-Agent field through the c-icap API function

Listing 5: user_agent_check.c

```

1 int user_agent_check(char *user_agent);
2 FILE *ua_blacklist;
3 char blacklisted_ua[100];
4 FILE *ua_whitelist_re;
5 FILE *ua_whitelist_b;
6 char whitelisted_re[100];
7 char whitelisted_b[100];
8 int ua_whitelist_score = 0;
9
10 int user_agent_check(char *user_agent) {
11     int ua_score = 0;
12     if (user_agent != NULL) {
13         ci_debug_printf(1, "DEBUG HTTP User-Agent is: %s\n", user_agent);
14         ua_blacklist = fopen("user_agent_blacklist.txt", "r");
15         while (fgets(blacklisted_ua, sizeof blacklisted_ua, ua_blacklist) != NULL) {
16             blacklisted_ua[strlen(blacklisted_ua)-1] = '\0';
17
18             /* check the User-Agent against the blacklist database */
19             if (strstr(user_agent, blacklisted_ua) != NULL) {
20                 ua_score += 1;
21                 ci_debug_printf(1, "DEBUG HTTP User-Agent '%s' is blacklisted! MALWARE SCORE: %d\n",
22                                 blacklisted_ua, ua_score);
23             }
24         }
25         ua_whitelist_re = fopen("user_agent_whitelist_render_engines.txt", "r");
26         while (fgets(whitelisted_re, sizeof whitelisted_re, ua_whitelist_re) != NULL) {
27             whitelisted_re[strlen(whitelisted_re)-1] = '\0';
28
29             /* check the User-Agent against the whitelist render engine database */
30             if (strstr(user_agent, whitelisted_re) != NULL) {
31                 ua_whitelist_score += 1;
32                 ci_debug_printf(1, "DEBUG HTTP User-Agent render engine '%s' is whitelisted!
33                                 MALWARE Score: %d\n", whitelisted_re, ua_score);
34                 ua_whitelist_b = fopen("user_agent_whitelist_browsers.txt", "r");
35                 while (fgets(whitelisted_b, sizeof whitelisted_b, ua_whitelist_b) != NULL) {
36                     whitelisted_b[strlen(whitelisted_b)-1] = '\0';
37
38                     /* check the User-Agent against the whitelist browser database */
39                     if (strstr(user_agent, whitelisted_b) != NULL) {
40                         ua_whitelist_score += 1;
41                         ci_debug_printf(1, "DEBUG HTTP User-Agent browser '%s' is whitelisted!
42                                 MALWARE SCORE: %d\n", whitelisted_b, ua_score);
43                     }
44                 }
45             }
46         }
47         if (ua_whitelist_score == 2 && ua_score == 0) {
48             ci_debug_printf(1, "scores are whitelist=%d and blacklist=%d", ua_whitelist_score, ua_score);
49             ci_debug_printf(1, "DEBUG HTTP User-Agent is unsuspected! MALWARE SCORE: %d\n", ua_score);
50         }
51         else {
52             ua_score += 1;
53             ci_debug_printf(1, "DEBUG HTTP User-Agent is suspected! MALWARE SCORE: %d\n", ua_score);
54         }
55     }
56     else {
57         ua_score += 1;
58         ci_debug_printf(1, "DEBUG HTTP User-Agent is empty! MALWARE SCORE: %d\n", ua_score);
59     }
60     return ua_score;
61 }

```

Figure 23: Implementation of the User-Agent check

4.3.3 POST check

To perform the POST check, two values need to be compared: the length of the HTTP body and the value of the Content-Length field. The c-icap API offers the `ci_http_content_length` function, which will return the value of the Content-Length field from the request data. The HTTP body size is already available in the echo script that is included with the c-icap package. These values were retrieved and passed to the POST check function which can be found in the code listing below. If both values don't match, something is out of order and a score is assigned.

Listing 6: snippet from `srv_inktvipam.c`

```

1  /* POST check > post_check.c */
2  int content_length = content_len;
3  int body_size = preview_data_len;
4  int post_score = 0;
5  post_score = post_check(content_length, body_size);

```

Figure 24: Retrieving the Content-Length field through the c-icap API function

Listing 7: `post_check.c`

```

1  int post_check(int content_length, int body_size);
2
3  int post_check(int content_length, int body_size) {
4      int p_score = 0;
5      if (content_length != NULL) {
6          ci_debug_printf(1, "DEBUG HTTP Content-Length is: %d\n", content_length);
7          ci_debug_printf(1, "DEBUG HTTP body size is: %d\n", body_size);
8
9          /* check the Content-Length against the length of the body */
10         if (content_length != body_size) {
11             p_score += 1;
12             ci_debug_printf(1, "DEBUG HTTP Content-Length does not match body size!
13                             MALWARE SCORE: %d\n", p_score);
14         }
15         else {
16             ci_debug_printf(1, "DEBUG HTTP Content-Length is unsuspected!
17                             MALWARE SCORE: %d\n", p_score);
18         }
19     }
20     else {
21         p_score += 1;
22         ci_debug_printf(1, "DEBUG HTTP Content-Length is empty!
23                             MALWARE SCORE: %d\n", p_score);
24     }
25     return p_score;
26 }

```

Figure 25: Implementation of the POST check

4.3.4 Content-Type check

We found a function we could use to extract the Content-Type field from the request data. Since our main goal was to implement a proof of concept we only applied checks for two specific Content-Types. More checks could easily be added. Please note that we apply a different score for Java. Java is especially relevant in drive-by downloads since many exploits target Java. Also we wanted to demonstrate that it is easy to modify the simple scoring mechanism.

Listing 8: snippet from `srv_inktvipam.c`

```

1  /* Content-Type check > content_type_check.c */
2  char *content_type = http_content_type(req);
3  int content_type_score = 0;
4  content_type_score = content_type_check(content_type);

```

Figure 26: Retrieving the Content-Type field through the c-icap API function

Listing 9: content_type_check.c

```

1 #include "simple_api.h"
2
3 int content_type_score(char *content_type);
4
5 int content_type_score(char *content_type) {
6     int ct_score = 0;
7
8     if (content_type != NULL) {
9         ci_debug_printf(1, "DEBUG HTTP Content-Type is: %s\n", content_type);
10
11         /* check the Content-Type 'octec-stream' and 'java-vm' */
12         if (strcmp(content_type,"application/octet-stream") == 0) {
13             ct_score += 1;
14             ci_debug_printf(1, "DEBUG HTTP Content-Type is 'application/octet-stream'!
15                             MALWARE SCORE: %d\n", ct_score);
16         }
17         else if (strcmp(content_type,"application/java-vm") == 0) {
18             ct_score += 3;
19             ci_debug_printf(1, "DEBUG HTTP Content-Type is 'application/java-vm'!
20                             MALWARE SCORE: %d\n", ct_score);
21         }
22         else {
23             ci_debug_printf(1, "DEBUG HTTP Content-Type is unsuspected!
24                             MALWARE SCORE: %d\n", ct_score);
25         }
26     }
27     else {
28         ct_score += 1;
29         ci_debug_printf(1, "DEBUG HTTP Content-Type is empty!
30                             MALWARE SCORE: %d\n", ct_score);
31     }
32 }
33
34 return ct_score;
35 }
36 }

```

Figure 27: Implementation of the Content-Type check

4.3.5 Domain check

In order to perform the domain check, the URL must first be retrieved by using a function from the `c-icap` API. Next, several string operations need to be performed in order to get the domain. The following code snippet shows the necessary steps to get the domain from the URL:

Listing 10: snippet from srv_inktvipam.c

```

1 /* Domain check > domain_check.c */
2 /* get the URL from the HTTP request header */
3 char *http_url[8192];
4 ci_http_request_url(req, http_url, 8192);
5
6 /* get the length of the URL */
7 int http_url_len = strlen(http_url);
8
9 /* strip 'http://' from the URL */
10 char *url_http = substr(http_url,7,http_url_len);
11
12 /* get the first position with '/' */
13 int *url_ptr = strchr(url_http,"/");
14
15 /* strip '/' and everything behind it, leaving the domain */
16 char *domain = substr(url_http,0,url_ptr);
17 int domain_score = 0;
18 domain_score = domain_check(domain);

```

Figure 28: srv_inktvipam.c snippet

The first check we perform on the domain, is to verify if it is not an IP address. If it is we assign a score. Because we have a full domain, further string operations are to be performed to get the TLD. Once the TLD

is stripped, a comparison is made with a blacklist. If the domain occurs in the blacklist, a score is assigned.

Listing 11: domain_check.c

```

1
2 int domain_check(char *url);
3
4 int domain_check(char *url) {
5     int d_score = 0;
6     FILE *f;
7     char blacklistedTld[4];
8
9
10    char *substr(const char* str, size_t begin, size_t len) {
11        if (strlen(str) == 0 || strlen(str) < begin) {
12            return "ERROR: string length = 0 or string length < starting point";
13        }
14        return strdup(str + begin, len);
15    }
16
17    if (url != NULL) {
18        ci_debug_printf(1, "DEBUG HTTP URL is: %s\n", url);
19        /* determine the URL is a valid IP address */
20        if ((isValidIpAddress(url)) == 1) {
21            d_score += 1;
22            ci_debug_printf(1, "DEBUG HTTP URL is an IP address! MALWARE SCORE: %d\n", d_score);
23        }
24        else {
25            ci_debug_printf(1, "DEBUG HTTP URL is not an IP address! MALWARE SCORE: %d\n", d_score);
26        }
27
28        /*not a valid IP, so must be url. strip until we have tld.*/
29        /* get the last position with '.' */
30        int *tld_ptr = strchr(url, ".");
31
32        /* strip '.' and everything before it, leaving the tld */
33        char *tld = substr(url, tld_ptr, strlen(url) - 1);
34
35        ci_debug_printf(1, "DEBUG HTTP TLD is: %s\n", tld);
36
37        f = fopen("blacklisted_tlds.txt", "r");
38        while (fscanf(f, "%s", blacklistedTld) != EOF) {
39            if (strcmp(blacklistedTld, tld) == 0) {
40                ci_debug_printf(1, "tld %s matches blacklisted tld! MALWARE SCORE: %d\n", tld, d_score);
41                d_score += 1;
42            }
43            else {
44                ci_debug_printf(1, "tld %s does not match blacklisted tld!\n", tld);
45            }
46        }
47    }
48    else {
49        d_score += 1;
50        ci_debug_printf(1, "DEBUG HTTP URL is empty! MALWARE SCORE: %d\n", d_score);
51    }
52    return d_score;
53 }
54

```

Figure 29: Unfinished implementation of the domain check

Unfortunately, the domain check was not finished in time to be part of the testing.

4.3.6 Geolocation check

To perform GeolP lookups the the GPL licensed Maxmind API was used. A source file from the Maxmind package was adapted to perform the GeolP lookup, resulting in the following code:

Listing 12: geolocation_check.c

```

1  /*
2  *
3  * Modified version of the test file provided by MaxMind LLC
4  *
5  * This library is free software; you can redistribute it and/or
6  * modify it under the terms of the GNU Lesser General Public
7  * License as published by the Free Software Foundation; either
8  *
9  * This library is distributed in the hope that it will be useful,
10 * but WITHOUT ANY WARRANTY; without even the implied warranty of
11 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
12 * Lesser General Public License for more details.
13 *
14 * You should have received a copy of the GNU Lesser General Public
15 * License along with this library; if not, write to the Free Software
16 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
17 */
18
19 //make sure GeoIP is installed on the system
20
21 #include "GeoIP.h"
22
23 int geolocation_check (char *ipAddress) {
24     fprintf(stderr, "in geolocation_check\n");
25     int geo_score;
26     FILE *f;
27     FILE *g;
28     //char ipAddress[30];
29     const char * returnedCountry;
30     char blacklistedCountry[3];
31
32     GeoIP * gi;
33     fprintf(stderr, "IP adress for GEOIP is%s\n",ipAddress);
34     int i;
35     for (i = 0; i < 2; ++i) {
36         if (0 == i) {
37             /* Read from filesystem, check for updated file */
38             gi = GeoIP_open("./GeoIP-1.4.8/data/GeoIP.dat", GEOIP_STANDARD | GEOIP_CHECK_CACHE);
39         } else {
40             /* Read from memory, faster but takes up more memory */
41             gi = GeoIP_open("./GeoIP-1.4.8/data/GeoIP.dat", GEOIP_MEMORY_CACHE);
42         }
43
44         if (gi == NULL) {
45             fprintf(stderr, "Error opening database\n");
46             exit(1);
47         }
48
49         /* make sure GeoIP deals with invalid query gracefully */
50         returnedCountry = GeoIP_country_code_by_addr(gi,NULL);
51         if (returnedCountry != NULL) {
52             fprintf(stderr,"Invalid Query test failed, got non NULL, expected NULL\n");
53         }
54
55         returnedCountry = GeoIP_country_code_by_name(gi,NULL);
56         if (returnedCountry != NULL) {
57             fprintf(stderr,"Invalid Query test failed, got non NULL, expected NULL\n");
58         }
59
60         f = fopen("geolocation_ip.txt","r");
61         while (fscanf(f, "%s", ipAddress) != EOF) {
62             returnedCountry = GeoIP_country_code_by_addr(gi,ipAddress);
63             fprintf(stderr, "Geo returnedCountry is: %s\n", returnedCountry);
64             if (returnedCountry == NULL) {
65                 fprintf(stderr, "GeoIP lookup of %s failed\n",ipAddress);
66             }
67             fprintf(stderr,"GeoIP lookup of %s results in %s\n",ipAddress, returnedCountry);
68             g = fopen("blacklisted_countries.txt","r");
69             while (fscanf(g, "%s", blacklistedCountry) != EOF) {
70                 if (strcmp(blacklistedCountry, returnedCountry) == 0) {
71                     fprintf(stderr,"GeoIP lookup of %s is %s matches blacklisted country\n",
72                             ipAddress,returnedCountry);
73                     geo_score = 10;
74                 }
75             }
76         }
77         fclose(f);
78         fclose(g);
79         GeoIP_delete(gi);
80     }
81     return geo_score;
82 }

```

Figure 30: Implementation of the geolocation check

Unfortunately, we did not succeed in compiling our `inktvipAM` module such that the Maxmind API was linked within the deadline of the implementation.

5 Test

The code written for the separate scores and its checks was first tested by adding known values to the code and checking the code step by step for working correctly and calculating the right scores at every point. This is also known as white box testing and was done during the coding process. Eventually the implemented scoring system was tested by using real website data and checking the results, also known as black box testing. For these tests 10 of the top websites in the Netherlands were chosen. These websites were randomly chosen from the first 35 of the Alexa top 500 of the Netherlands [27]. This resulted in the following 10 websites:

1. `google.nl`
2. `google.com`
3. `facebook.com`
4. `wikipedia.org`
5. `nu.nl`
6. `ing.nl`
7. `t.co`
8. `tweakers.net`
9. `piratebay.org`
10. `powned.tv`

Additionally two IP addresses were added from `facebook.com` and `piratebay.org` to test how the scoring system would react to IP addresses. A third URL with an IP address, related to Java, was added to test how the scoring system would react to an IP address and Java application. This combination should result in a blocked (part of the) website, because it exceeds the score for allowed traffic.

The black box tests were conducted by using one of the clients within the virtual environment and checking its result on the proxy server. The site was entered in the Microsoft Internet Explorer 6.0 browser on this client and requested. On the server the score of the request(s) was recorded and reported. A website with multiple requests is reported in one row, meaning a score on one of the checks is treated as a result for the main site. After each test the client was reset to the default snapshot. The results of the website are presented in table 2.

URL	H o s t n a m e	D o m a i n	U s e r - A g e n t	P O S T	C o n t e n t - T y p e	Allow	Block
google.nl	0	0	0	0	0	X	
google.com	0	0	0	0	0	X	
facebook.com	0	0	0	1	0	X	
wikipedia.org	0	0	0	0	0	X	
nu.nl	0	0	0	1	0	X	
ing.nl	0	0	0	0	0	X	
t.co	0	0	0	0	0	X	
tweakers.net	0	0	0	0	0	X	
piratebay.org	0	0	0	1	0	X	
powned.tv	0	0	0	0	0	X	
69.171.242.53	0	0	0	1	0	X	
194.71.107.15	0	2	0	0	0	X	
137.254.16.66/nl/download/installed.jsp?detect=jre&try=1	0	2	0	1	3		X
TOTAL	0	4	0	4	3	12	1

Table 2: Test results

In 12 of the 13 cases the website was allowed and only ones a (part of the) site was blocked. This was as predicted. The interesting part of these 12 cases was that in six of them the website got some kind of score, meaning there were some aspects of the site that points to a bad implementation of the specification or a malicious site. These six cases are mentioned in the following subsections.

5.1 facebook.com: 69.171.242.53

The facebook.com website is loaded correctly as shown in figure 31. It came with a lot of requests, including some ad and statistics related requests. All of these requests score 0 points with the checks, except the first one for the main facebook.com HTML page. This request scored 1 point on the POST check, because the Content-Length is empty as shown in figure 32. This points to a bad implemented main HTML page in which the Content-Length field is not set in the HTTP header. This result is repeated when requesting facebook.com with its IP address. The request directly forwards to facebook.com. This is probably done by the Domain Name Server (DNS). The IP address doesn't even show up as a request, so the result is exactly the same without scoring any point for being an IP address. It does however score 1 point on the POST check. A score of 1 point isn't enough for blocking the request, but it certainly is something to keep in mind. This concludes that a clean website can have implementation errors which can lead to false positives. In this case the traffic is allowed because they score only 1 point.

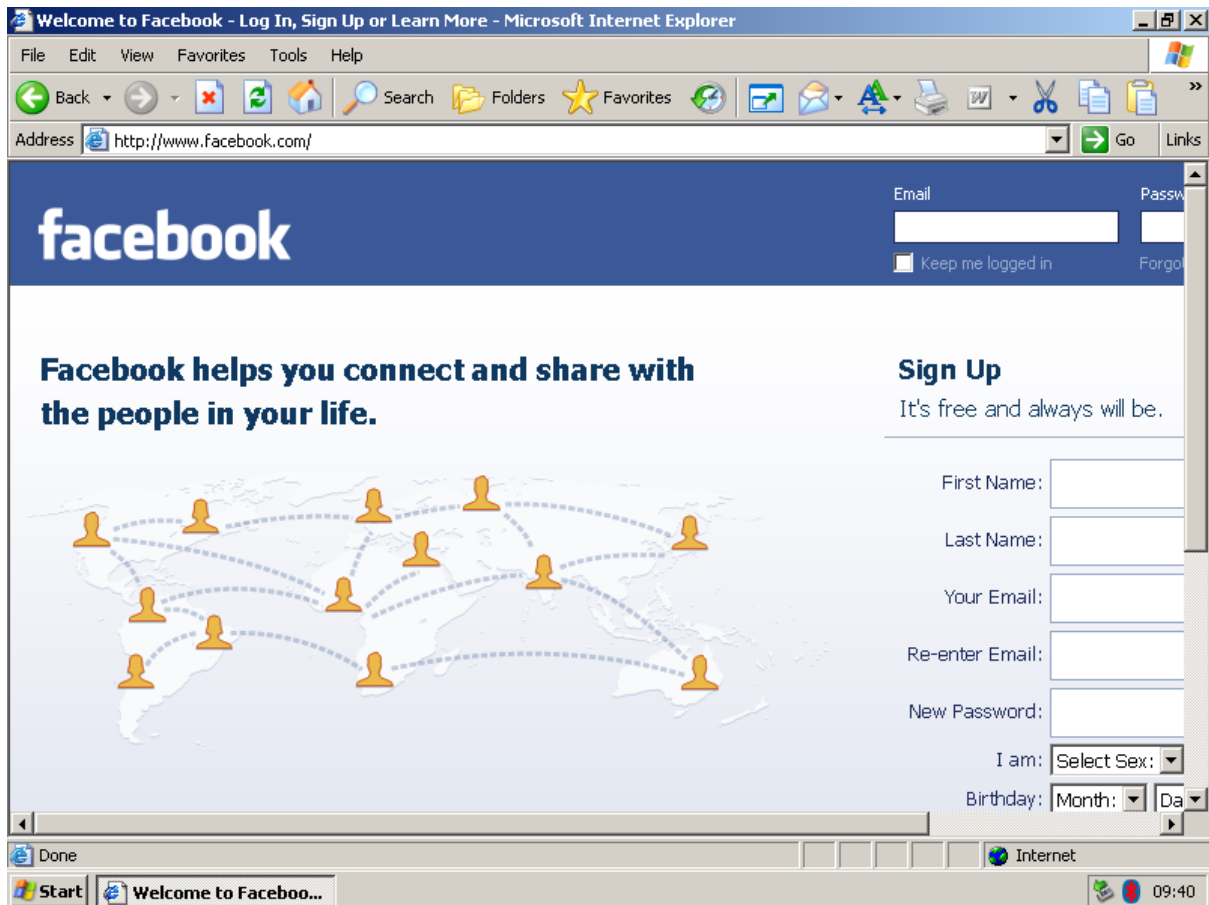


Figure 31: facebook.com client screenshot

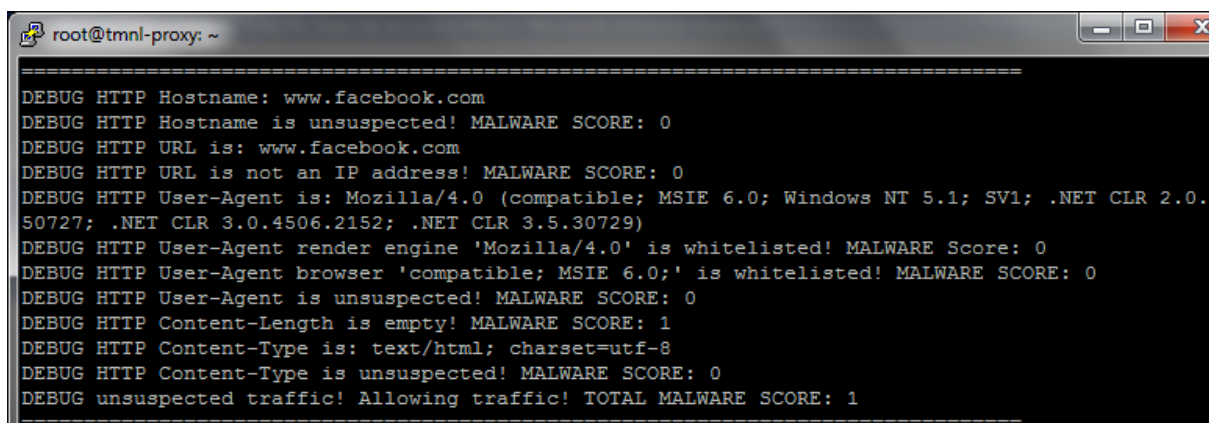


Figure 32: facebook.com server screenshot

5.2 nu.nl

The nu.nl website isn't loaded correctly as shown in figure 33. The reason for this remains unclear. It also came with a lot of requests, but all of them are allowed. The incorrect loading of this site is probably the

effect of using an old browser (Microsoft Internet Explorer 6.0) with a modern website. Again all of the requests score 0 points with the checks, except for one ad related request. This request also scored 1 point on the POST check, because the Content-Length is empty as shown in figure 34. The conclusion is the same as mentioned in 5.1 facebook.com: 69.171.242.53 and because of this test is passed for this website. The nu.nl

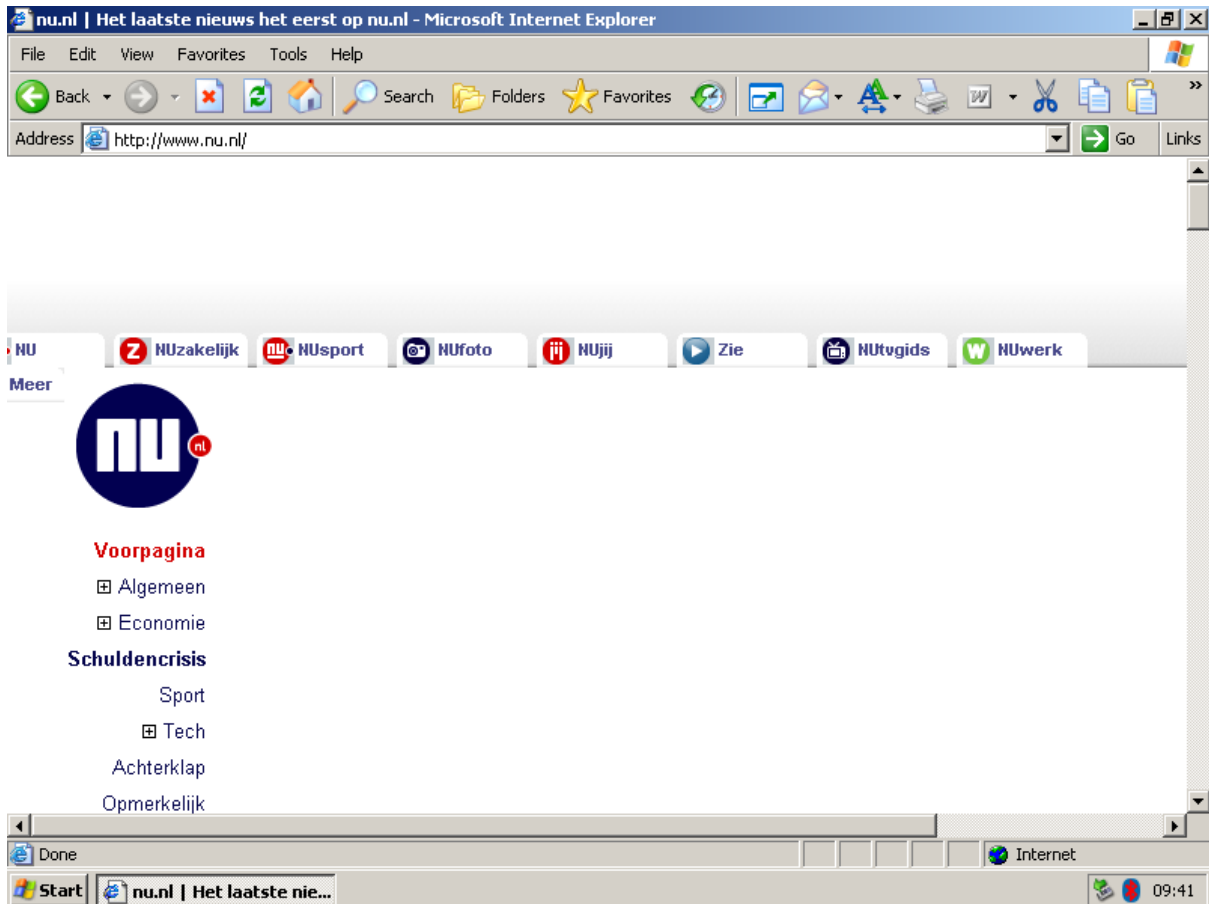


Figure 33: nu.nl client screenshot

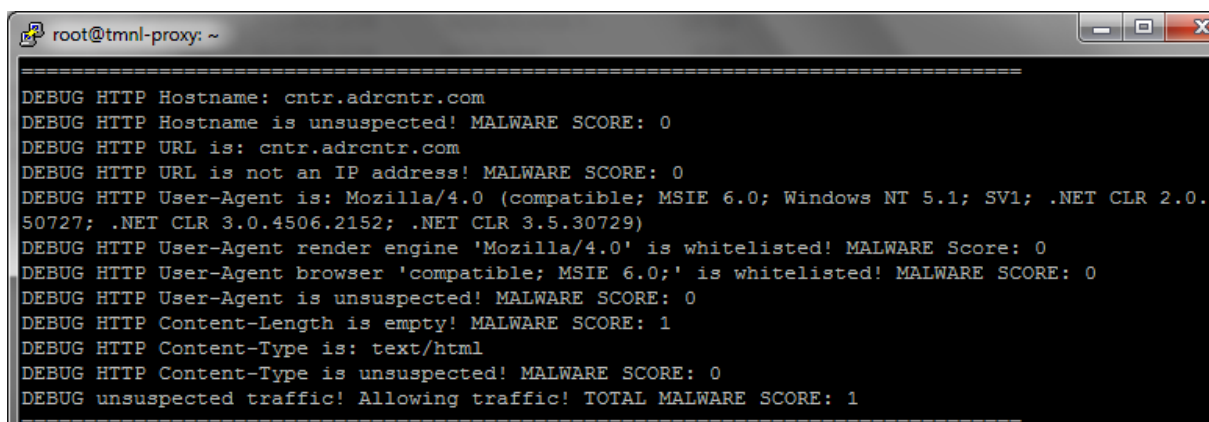


Figure 34: nu.nl server screenshot

5.3 piratebay.org: 194.71.107.15

The `piratebay.org` website is loaded correctly as shown in figure 35. A big difference with the `facebook.com` and `nu.nl` was that it didn't come with a lot of requests. There are just two requests as shown in 36. The `piratebay.org` website is requested, but after this request another request is done for `thepiratebay.se`, making it a redirection. The `Referer` is not used in any of the currently implemented checks, so it's not founded by real data. However the follow up behavior of the requests clearly shows a redirection pattern and this is founded by the browsers address bar with `thepiratebay.se` in it (shown in figure 35). Another remarkable discovery which supports the hypothesis of not needing to know which requests belong to a session (stated in 3.2 Previous research), is shown by the `thepiratebay.se` request. This request scores 1 point on the POST check, because the `Content-Length` is empty (shown in figure 36. The conclusion about the scoring is the same as mentioned in 5.1 `facebook.com`: 69.171.242.53 and 5.2 `nu.nl`, but shows that a redirection is scored in the same way as the former request. This means that a malicious redirection will be blocked if it scores enough points on the checks and that a redirection check could be a good addition to the scoring system, but isn't really necessary to be able to block malicious websites.

When requesting `piratebay.org` with its IP address the same thing is done as requesting `piratebay.org`. The first request is the typed URL in the address bar of the browser which in this case is `194.71.107.15`. This request gets 2 points for being an IP address by the domain score as shown in figure 37, because it is not a usual URL to browse to. The request is followed up by the `thepiratebay.se` request, resulting in the same findings as the first test. Both tests are passed although they respectively get 1 point and 2 points. The tests even supported an earlier hypothesis about being able to identify which individual requests belong to a given session.

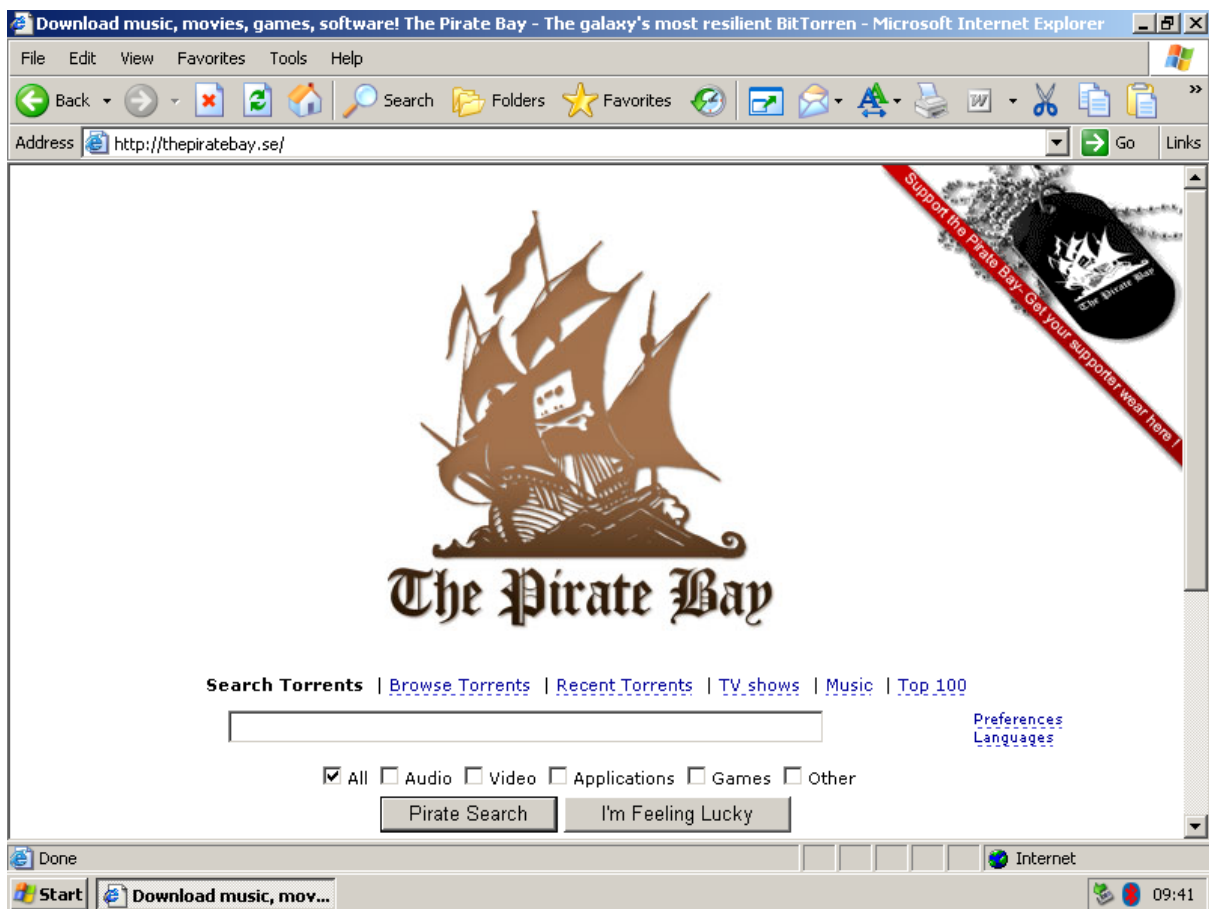


Figure 35: piratebay.org client screenshot

```

root@tmnl-proxy: ~
=====
DEBUG HTTP Hostname: www.piratebay.org
DEBUG HTTP Hostname is unsuspected! MALWARE SCORE: 0
DEBUG HTTP URL is: www.piratebay.org
DEBUG HTTP URL is not an IP address! MALWARE SCORE: 0
DEBUG HTTP User-Agent is: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)
DEBUG HTTP User-Agent render engine 'Mozilla/4.0' is whitelisted! MALWARE Score: 0
DEBUG HTTP User-Agent browser 'compatible; MSIE 6.0;' is whitelisted! MALWARE SCORE: 0
DEBUG HTTP User-Agent is unsuspected! MALWARE SCORE: 0
DEBUG HTTP Content-Length is: 26
DEBUG HTTP body size is: 26
DEBUG HTTP Content-Length is unsuspected! MALWARE SCORE: 0
DEBUG HTTP Content-Type is: text/html
DEBUG HTTP Content-Type is unsuspected! MALWARE SCORE: 0
DEBUG unsuspected traffic! Allowing traffic! TOTAL MALWARE SCORE: 0
=====
DEBUG HTTP Hostname: thepiratebay.se
DEBUG HTTP Hostname is unsuspected! MALWARE SCORE: 0
DEBUG HTTP URL is: thepiratebay.se
DEBUG HTTP URL is not an IP address! MALWARE SCORE: 0
DEBUG HTTP User-Agent is: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)
DEBUG HTTP User-Agent render engine 'Mozilla/4.0' is whitelisted! MALWARE Score: 0
DEBUG HTTP User-Agent browser 'compatible; MSIE 6.0;' is whitelisted! MALWARE SCORE: 0
DEBUG HTTP User-Agent is unsuspected! MALWARE SCORE: 0
DEBUG HTTP Content-Length is empty! MALWARE SCORE: 1
DEBUG HTTP Content-Type is: text/html;charset=UTF-8
DEBUG HTTP Content-Type is unsuspected! MALWARE SCORE: 0
DEBUG unsuspected traffic! Allowing traffic! TOTAL MALWARE SCORE: 1
=====

```

Figure 36: piratebay.org server screenshot

```

root@tmnl-proxy: ~
=====
DEBUG HTTP Hostname: 194.71.107.15
DEBUG HTTP Hostname is unsuspected! MALWARE SCORE: 0
DEBUG HTTP URL is: 194.71.107.15
DEBUG HTTP URL is an IP address! MALWARE SCORE: 2
DEBUG HTTP User-Agent is: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)
DEBUG HTTP User-Agent render engine 'Mozilla/4.0' is whitelisted! MALWARE Score: 0
DEBUG HTTP User-Agent browser 'compatible; MSIE 6.0;' is whitelisted! MALWARE SCORE: 0
DEBUG HTTP User-Agent is unsuspected! MALWARE SCORE: 0
DEBUG HTTP Content-Length is: 26
DEBUG HTTP body size is: 26
DEBUG HTTP Content-Length is unsuspected! MALWARE SCORE: 0
DEBUG HTTP Content-Type is: text/html
DEBUG HTTP Content-Type is unsuspected! MALWARE SCORE: 0
DEBUG unsuspected traffic! Allowing traffic! TOTAL MALWARE SCORE: 2
=====

```

Figure 37: 194.71.107.15 server screenshot

5.4 137.254.16.66

The 137.254.16.66/nl/download/installed.jsp?detect=jre&try=1 URL isn't loaded correctly as shown in figure 38 and in this case that's a good thing. A part of this website is blocked, because the request is to an IP address and contains Java related content. The website came with a lot of requests which all score at

least 2 points for being an IP address. A score of 2 points for being an IP address isn't enough for blocking the request. However one of the requests contains the content-type for having Java related content, so this request scores 3 point at the content-type check as shown in figure 39. Eventually resulting in a total score of 5 points which is the minimum score for blocking a request. This shows that the concept scoring system will work for blocking traffic. One of the requests for this website also scored 1 point by the POST check for having an empty Content-Length field as shown in figure 40. Together with the 2 points for being an IP address, this resulted in a total score of 3 points. This score isn't enough for blocking the request, but this does point to suspected traffic which you do want to log for auditing purposes.

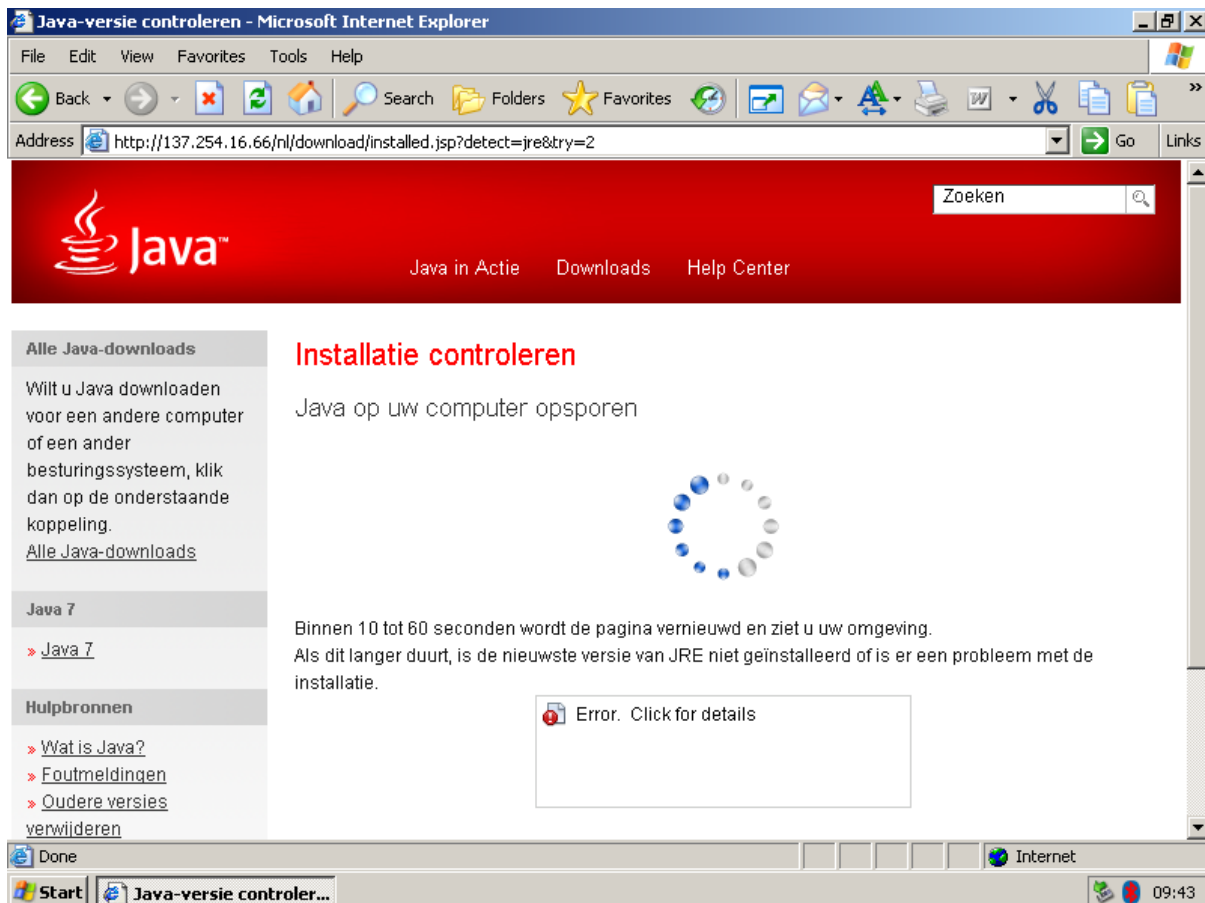
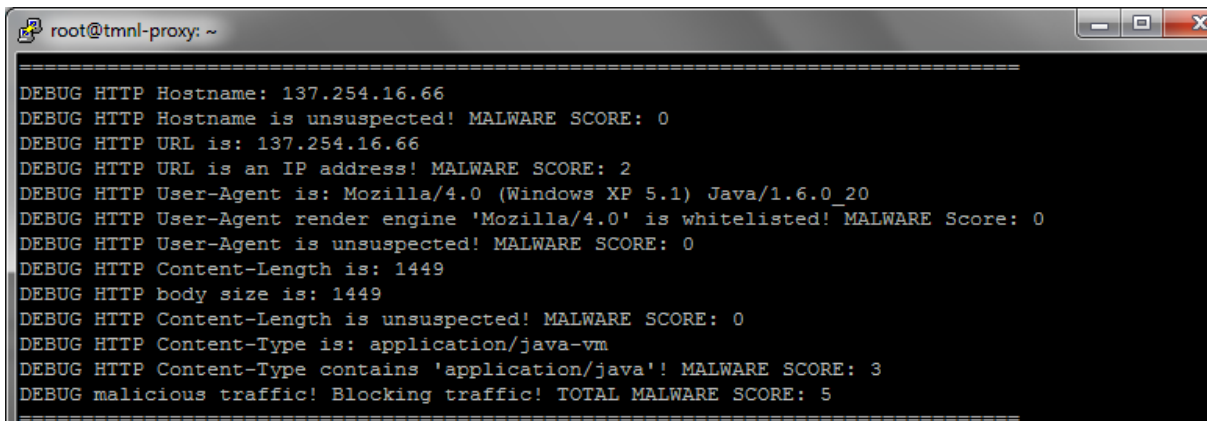
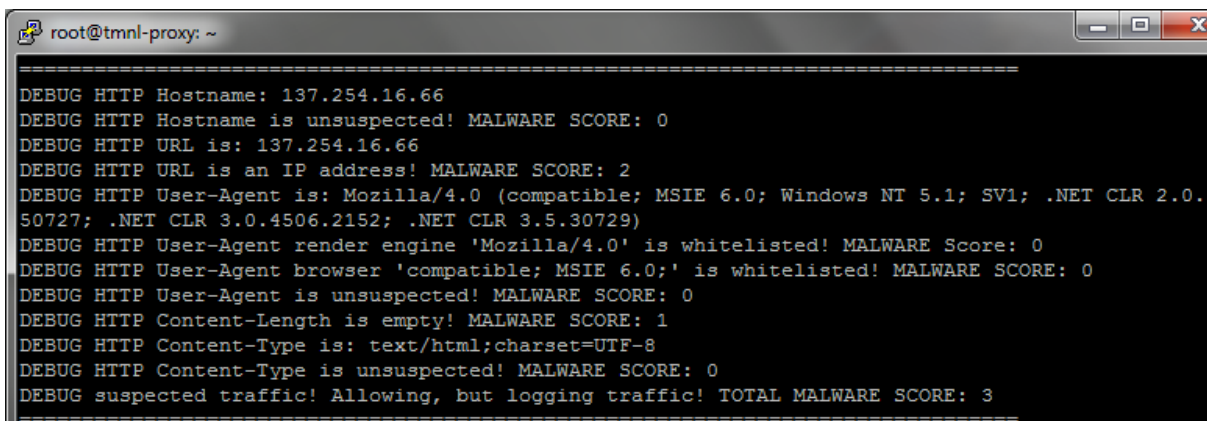


Figure 38: 137.254.16.66 client screenshot



```
root@tmnl-proxy: ~
=====
DEBUG HTTP Hostname: 137.254.16.66
DEBUG HTTP Hostname is unsuspected! MALWARE SCORE: 0
DEBUG HTTP URL is: 137.254.16.66
DEBUG HTTP URL is an IP address! MALWARE SCORE: 2
DEBUG HTTP User-Agent is: Mozilla/4.0 (Windows XP 5.1) Java/1.6.0_20
DEBUG HTTP User-Agent render engine 'Mozilla/4.0' is whitelisted! MALWARE Score: 0
DEBUG HTTP User-Agent is unsuspected! MALWARE SCORE: 0
DEBUG HTTP Content-Length is: 1449
DEBUG HTTP body size is: 1449
DEBUG HTTP Content-Length is unsuspected! MALWARE SCORE: 0
DEBUG HTTP Content-Type is: application/java-vm
DEBUG HTTP Content-Type contains 'application/java!' MALWARE SCORE: 3
DEBUG malicious traffic! Blocking traffic! TOTAL MALWARE SCORE: 5
=====
```

Figure 39: 137.254.16.66 server screenshot (5 points)



```
root@tmnl-proxy: ~
=====
DEBUG HTTP Hostname: 137.254.16.66
DEBUG HTTP Hostname is unsuspected! MALWARE SCORE: 0
DEBUG HTTP URL is: 137.254.16.66
DEBUG HTTP URL is an IP address! MALWARE SCORE: 2
DEBUG HTTP User-Agent is: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)
DEBUG HTTP User-Agent render engine 'Mozilla/4.0' is whitelisted! MALWARE Score: 0
DEBUG HTTP User-Agent browser 'compatible; MSIE 6.0;' is whitelisted! MALWARE SCORE: 0
DEBUG HTTP User-Agent is unsuspected! MALWARE SCORE: 0
DEBUG HTTP Content-Length is empty! MALWARE SCORE: 1
DEBUG HTTP Content-Type is: text/html;charset=UTF-8
DEBUG HTTP Content-Type is unsuspected! MALWARE SCORE: 0
DEBUG suspected traffic! Allowing, but logging traffic! TOTAL MALWARE SCORE: 3
=====
```

Figure 40: 137.254.16.66 server screenshot (3 points)

6 Conclusion

This chapter concludes our research report. We will briefly summarize the achievements, point out subjects for further research and finalize with the general conclusion.

6.1 Achievements

With this research project some important things are achieved:

improved former methods/checks This research improved the methods and checks discovered by Kinkhorst and Van Kleij with other research done in this field and with self-discovered new checks.

implemented working concept This research translated a theory about a malware scoring system to a working concept which is able to score traffic based on the information in the HTTP header.

showed a scalable possible solution for enterprises The implementation showed some good insight in the scalability of the concept with the use of freely available open source applications and tools.

showed that session information isn't really needed This research showed that being able to identify individual requests to certain sessions is not a prerequisite to being able to identify malicious HTTP traffic.

created an open platform for future research The implemented concept offers a good starting point and open platform to conduct future research on, which could eventually lead to a fully working malware scoring system.

6.2 Future research

This report offers a starting point for future research as mentioned in 6.1 Achievements. While performing research on this project, a couple of very important aspects were discovered that would help improve the concept scoring system in being able to identify malicious traffic in a better way.

6.2.1 Adding checks

In 3.2 Previous research analysis, the geolocation and domain score are showed with checks that aren't implemented in the concept. These could be described as follows:

Geolocation score

- geographical IP lookup
- geographical referer IP lookup

Domain score

- geographical and ccTLD match
- domain registration date lookup
- dictionary lookup
- number/letter ratio calculation

In addition to these checks section 3.3 Infection history analysis showed some aspects that could distinguish malicious traffic. These could be described as followed:

- encrypted body check
- large URL check
- long file name check
- Java file check
- domain TLD checks

All these checks would be a great addition to the concept and will enhance the scoring system. Especially the checks already defined in the flowcharts will increase the effectiveness of the detection system. Besides the suggested checks, other checks could be added that will help improve the detection system.

6.2.2 Balancing scoring system

Adding more checks will result in the need for a good balanced scoring system. This should be done by finding a mathematical balance between the checks and scores. The balance should be generic and applicable for different scenarios. The balance must be tested on real malware infection data and on a large scale. For drive-by downloads, testing is somewhat problematic since drive-by infections are very fluid in their nature. Building a corpus of drive-by data is therefore a difficult task, made even more problematic due to the sensitivity of the information. We advise researchers to make packet captures when they find a live drive-by attack, in order to record and preserve the traffic it generates.

6.2.3 Testing live

The additional checks and the balancing of the scoring system will eventually result in a complete proof of concept, one that should be able to run in a live business environment. Testing the scoring system in a live environment will show its potential and shortcomings for being an applicable malware prevention and detection system.

6.3 General conclusion

The main research question of this study was:

How could malware infection attempts be detected and prevented from within the IT infrastructure of the business that has outsourced IT service management or that allows 'bring your own device'?

Our research shows that malware infection attempts, specifically drive-by downloads, can be detected and prevented without having control of end points. We have described an approach of using a transparent web proxy that interfaces to specialised detection modules and have shown it having potential. The scoring system we implemented is too unsophisticated and needs further research more development and serious testing in order to be of use for detecting malicious web traffic in enterprise environments. Our research lays the foundation for this and we believe further efforts along this path are worthwhile.

References

- [1] Michael J. Earl, *The Risks of Outsourcing IT*, 1996.
- [2] F-Secure - Searching for the first PC virus in Pakistan, <http://campaigns.f-secure.com/brain/>.
- [3] CCC - Chaos Computer Club analyzes government malware, <http://ccc.de/en/updates/2011/staatstrojaner/>.
- [4] Daily Yomiuri Online - Govt working on defensive cyberweapon / Virus can trace, disable sources of cyber-attacks, <http://www.yomiuri.co.jp/dy/national/T120102002799.htm>.
- [5] ENISA - Botnets: Measurement, Detection, Disinfection and Defence, <http://www.enisa.europa.eu/act/res/botnets/botnets-measurement-detection-disinfection-and-defence/>.
- [6] Krebs on Security - Is That a Virus in Your Shopping Cart?, <http://krebsonsecurity.com/2011/08/is-that-a-virus-in-your-shopping-cart/>.
- [7] Krebs on Security - MySQL.com Sold for \$3k, Serves Malware, <http://krebsonsecurity.com/2011/09/mysql-com-sold-for-3k-serves-malware/>.
- [8] The Register - Mass compromise powers massive drive-by download attack, http://www.theregister.co.uk/2008/03/13/mass_compromise/.
- [9] The Register - Hacked BBC streaming websites serve up malware, http://www.theregister.co.uk/2011/02/15/bbc_driveby_download/.
- [10] ZDNet - 55,000 Web sites hacked to serve up malware cocktail, <http://www.zdnet.com/blog/security/55000-web-sites-hacked-to-serve-up-malware-cocktail/4091/>.
- [11] Wang Tao, Yu Shunzheng and Xie Bailin, *A Novel Framework for Learning to Detect Malicious Web Pages*, 2010.
- [12] Konrad Rieck, Tammo Krueger and Andreas Dewald, *Cujo: Efficient Detection and Prevention of Drive-by-Download Attacks*, 2010.
- [13] Aikaterinaki Niki, *Drive-by Download Attacks: Effects and Detection Methods*, 2009.
- [14] Julia Narvaez, Barbara Endicott-Popovsky, Christian Seifert, Chiraag Aval and Deborah A. Frincke, *Drive-by-Downloads*, 2010.
- [15] Marco Cova, Christopher Kruegel and Giovanni Vigna, *Detection and Analysis of Drive-by-Download Attacks and Malicious JavaScript Code*, 2010.
- [16] Van Lam Le, Ian Welch, Xiaoying Gao and Peter Komisarczuk, *Identification of Potential Malicious Web Pages*, 2011.

- [17] Thijs Kinkhorst and Michael van Kleij, *Detecting the ghost in the browser: Real time detection of drive-by infections*, 2009.
- [18] squid: Optimising Web Delivery, <http://www.squid-cache.org>.
- [19] RFC 3507: Internet Content Adaptation Protocol (ICAP), <http://tools.ietf.org/html/rfc3507/>.
- [20] Kevin de Kok and Marcus Bakker, *HTTP Session Identification*, 2010.
- [21] Sophos, *Security threat report 2011*, 2011.
- [22] Yury Namestnikov, *IT Threat Evolution: Q3 2011*, http://www.securelist.com/en/analysis/204792201/IT_Threat_Evolution_Q3_2011/, 2011.
- [23] Bogdan Botezatu, *H1 2011 E-Threat Landscape Report*, 2011.
- [24] RFC 2616: Hypertext Transfer Protocol - HTTP/1.1, <http://tools.ietf.org/html/rfc2616/>.
- [25] MDL: Malware Domain List, <http://www.malwaredomainlist.com>.
- [26] ZYTRAX - Browser ID Strings (a.k.a. User Agent ID), http://www.zytrax.com/tech/web/browser_ids.htm.
- [27] Alexa - Top Sites in Netherlands, <http://www.alexa.com/topsites/countries/NL/>.

List of Figures

1	Malware infection triangle	1
2	Drive-by download stages	3
3	Network topology in VMware vSphere ESXi	4
4	Software topology on proxy server	5
5	sysctl configuration	5
6	iptables configuration	6
7	dhcpcd configuration	6
8	squid configuration	6
9	squid redirection option with squidguard	7
10	squidclamav as ICAP service in squid configuration	7
11	OSI model (wikipedia)	9
12	Malware scoring system flowchart	10
13	Geolocation scoring flowchart	11
14	Hostname scoring flowchart	12
15	Domain scoring flowchart	13
16	User-Agent scoring flowchart	14
17	POST scoring flowchart	15
18	Content-Type scoring flowchart	16
19	Score calculation, classification and decision	18
20	Retrieving the hostname field through the c-icap API function	18
21	Implementation of the hostname check	19
22	Retrieving the User-Agent field through the c-icap API function	19
23	Implementation of the User-Agent check	20
24	Retrieving the Content-Length field through the c-icap API function	21
25	Implementation of the POST check	21
26	Retrieving the Content-Type field through the c-icap API function	21
27	Implementation of the Content-Type check	22
28	srv_inktvipam.c snippet	22
29	Unfinished implementation of the domain check	23
30	Implementation of the geolocation check	24

31	facebook.com client screenshot	27
32	facebook.com server screenshot	27
33	nu.nl client screenshot	28
34	nu.nl server screenshot	28
35	piratebay.org client screenshot	29
36	piratebay.org server screenshot	30
37	194.71.107.15 server screenshot	30
38	137.254.16.66 client screenshot	31
39	137.254.16.66 server screenshot (5 points)	32
40	137.254.16.66 server screenshot (3 points)	32

List of Tables

1	Malware hosting countries top 10	11
2	Test results	26