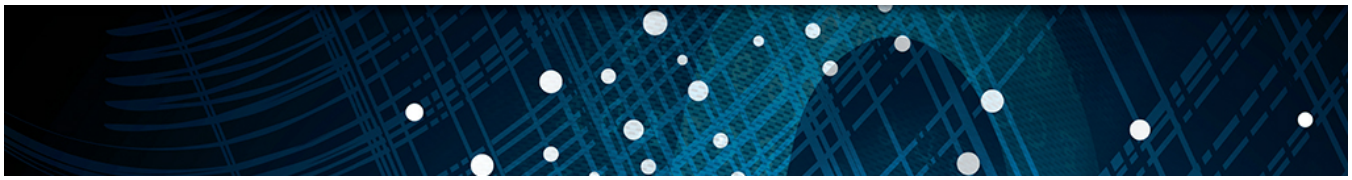
Characterizing Anomalies in Malware-Generated HTTP Traffic

hindawi.com/journals/scn/2020/8848863/



Research Article | Open Access

Volume 2020 |Article ID 8848863 | https://doi.org/10.1155/2020/8848863
Piotr Białczak, Wojciech Mazurczyk, "Characterizing Anomalies in Malware-Generated HTTP Traffic", *Security and Communication Networks*, vol. 2020, Article ID 8848863, 26 pages, 2020. https://doi.org/10.1155/2020/8848863

Piotr Białczak

[1] and Wojciech Mazurczyk[2]
[1]CERT Polska/Research and Academic Computer Network (NASK), Kolska 12, Warsaw 01-045, Poland

[2]Warsaw University of Technology, Nowowiejska 15/19, Warsaw 00-665, Poland

Academic Editor: Clemente Galdi

Received14 Apr 2020

Revised18 Jun 2020

Accepted07 Aug 2020

Published01 Sep 2020

Abstract

Currently, we are witnessing a significant rise in various types of malware, which has an impact not only on companies, institutions, and individuals, but also on entire countries and societies. Malicious software developers try to devise increasingly sophisticated ways to perform nefarious actions. In consequence, the security community is under pressure to develop more effective defensive solutions and to continuously improve them. To accomplish this, the defenders must understand and be able to recognize the threat when it appears. That is why, in this paper, a large dataset of recent real-life malware samples was used to identify anomalies in the HTTP traffic produced by the malicious software. The authors analyzed malware-generated HTTP requests, as well as benign traffic of the popular web browsers, using 3 groups of features related to the structure of requests, header field values, and payload characteristics. It was observed that certain attributes of the HTTP traffic can serve as an indicator of malicious actions, including lack of some popular HTTP headers and their values or usage of the protocol features in an uncommon way. The findings of this paper can be conveniently incorporated into the existing detection systems and network traffic forensic tools, making it easier to spot and eliminate potential threats.

1. Introduction

In the present-day Internet, one of the most commonly used protocols is the Hypertext Transfer Protocol (HTTP) [1, 2]. Its utilization is widespread as it is an essential component of web browsing. It also serves as a “backbone” of many services, even those standardized with other network protocols like e-mail and instant messaging. However, HTTP protocol prevalence is steadily decreasing in favor of TLS, HTTP/2, and FB-ZERO protocols, according to [1]. Deployment of an HTTP server is easy even for those who are not tech-savvy users, with many tutorials available in national languages. It is also often provided as a service by webhosting companies. On top of this, there is a lack of monitoring or blocking in many networks and easily achievable blending with legitimate network traffic. It is not surprising that malware developers use HTTP as a primary protocol to enable malicious communication. For example, according to Miller and Smith [3], HTTP is the most popular protocol used in C&C traffic, surpassing HTTPS. All this led the authors of this paper to focus on analyzing the HTTP protocol solely.

HTTP is used by malware for various purposes, for example, for connecting with the Command and Control (C&C) server to register/download commands, checking the external IP address of the infected host, and downloading additional modules. It is also used to perform DDoS (Distributed Denial of Service) attacks or create revenue by clicking on referral links. Such communication is masked by benign HTTP traffic which can be vastly different, depending on the application and its usage purpose. It must be noted that the HTTP protocol can be used by applications other than web browsers, for example, updaters, operating system mechanisms, application shops, and messengers. The main difference between the network traffic of such applications and the network traffic of web browsers lays in the characteristic of used addresses. The latter traffic can be potentially directed to any address, while in the former, the addresses are constant: they are either a set of domain names or an IPs range. For example, addresses of servers used by Windows telemetry services or Windows update mechanisms are widely known and are listed in many manuals focusing on blocking these services with network firewalls [4, 5] or dedicated tools such as WindowsSpyBlocker (https://github.com/crazy-max/WindowsSpyBlocker/). Network traffic of these applications can be easily identified using, for example, publicly available address lists or a short analysis of the traffic in the network proxy log. Considering the above, the authors decided to focus only on the web browser traffic as the other popular HTTP-based applications are relatively easy to be identified and filtered out from the network traffic.

The analysis of HTTP traffic characteristics presented in the current malware behavior research [6–9] suggests that some malware families’ HTTP requests differ from those generated by benign applications. This is especially visible when compared to the network traffic of applications operated by humans, e.g., web browsers. However, to the authors’ best knowledge, there is no extensive study which systematically identifies and analyzes dissimilarities between the malicious (malware) and benign (web browsers) HTTP traffic.

To fill this gap, the authors have thoroughly analyzed HTTP requests of both malware and browser traffic (using recent traffic sources), in order to establish their distinctive features. The research has focused solely on the Microsoft Windows operating systems family, as it is still the most frequently attacked platform—in 2018, more than half of the newly developed malware targeted these systems [10]. The conducted investigation explores a set of features and was created based on the authors’ own experience with real malware samples’ analyses and previous research work in this area (see Section 2). The chosen features reflect the structure of requests, values of different HTTP protocol fields, and the analysis of payload data. The main objective is to identify which parts of HTTP requests are different in malware and the browser network traffic and which can be identified as general features for distinguishing between these two types. The features and their values deviating from standards defined by network traffic originating from browsers

will be defined as anomalies. Some of the analyzed features can be seen as anomalies because they do not conform to standards or registered values, and they are present in both malware and browser network traffic. In such cases, the frequency of such occurrences will be quantified.

The main motivation behind this work is to provide other researchers with a list of identified anomalies of malware HTTP traffic. Such a list can be used directly by analysts when analyzing network traffic (e.g., during digital investigation) but also as an entry point for the design of malware detection systems. Availability of a well-described set of network anomalies can also help in developing other monitoring systems, for example, malware fingerprinting solutions. Therefore, the authors believe that this work will help fighting malicious software.

Considering the above, the main contributions of this paper are(i)Conducting a survey of HTTP requests' features previously used to detect malware in the existing research and performing an academic verification of usefulness of features proposed by previous nonacademic work(ii)Proposing an improved set of HTTP requests' features, including original ones, which can be potentially utilized for malware identification(iii)Identifying and analyzing malware HTTP requests' distinctive features and performing analysis of their influence on each other(iv)Providing a list of malware HTTP requests' distinctive features, along with practical usage scenarios

The contributions of this paper in a summarized and concise form are presented in Sections 6.1 and 6.2.

Due to a substantial number of performed analyses, not all of them were described in this paper, in order to maintain its clarity. Included are only those results which can help distinguish between malware and HTTP network traffic.

The rest of the paper is structured as follows. Section 2 describes the existing work related to the HTTP-based anomaly detection. Section 3 explains the fundamentals of the HTTP protocol. In Section 4, an experimental methodology used in this paper is outlined in detail. Section 5 presents obtained experimental results. Section 6 investigates how our discoveries can be applied in practice to the existing detection solutions. In Section 7, several limitations of this work are discussed. Finally, Section 8 concludes this paper and outlines future work.

2. Related Work

This section reviews the existing works which are most closely related to the research conducted in this paper. To start with, academic research papers exploring the behavior of malware are described, as they can be directly compared with the below work. Several nonacademic sources are also investigated; they show or use features for identification of the malware HTTP requests.

Rossow et al. presented in [11] the results of analysis of malware network traffic. They analyzed more than 100,000 samples, from which about 43.8% performed network activity. The authors provide observations about DNS and HTTP traffic, but only the latter will be summarized here. Analysis of the HTTP requests revealed that 89.5% of samples sent *GET* and 56.3% sent *POST* requests. Furthermore, 144 unique header names were observed. 98.6% of samples specified the *User-Agent* header; however, only 31% of samples included correct values. Additionally, 50.6% of samples changed this value during execution. 44.3% of samples included the *Accept-Language* header; however, 24.1% of them did not respect the operating system language locale.

In [12], Nelson presented a framework for analyzing and visualizing malware network traffic. The framework expands on the Sandnet framework [11] and its analyses. It provides a means for execution of malware samples and capturing their network traffic; it also provides analysis and clustering of protocols and visualization of the obtained results. To evaluate the framework, an analysis was conducted, providing manual inspection of 5 malware families and semiautomatic inspection of the whole dataset of 16,967 pcap files. In the latter part, the author analyzed network protocol breakdown and characteristics of DNS and HTTP protocols. Analysis of 118,035 HTTP requests included in the dataset was performed on multiple features, such as the request method, URI, or popular header values. The results reveal that in 86.7% of the captured files, *GET* requests were present. Also, in 86.8% of these files, *POST* requests were present. 24 unique header names were observed, as well as 33 unique *User-Agent* header values. The author also performed an analysis of *Accept-Language* and *Content-Type* headers, stating that their values can be used to identify malicious network traffic.

Calzarossa and Massari in [13] presented an evaluation of headers' usage in HTTP traffic. The authors monitored network traffic generated towards web servers at their university, focusing mainly on capturing HTTP requests. The analyzed a dataset which consisted of 315,000 requests, sent by about 6100 clients. The results indicated interesting characteristics of HTTP traffic. About 4% of requests were sent using HTTP/1.0, and the number of header fields was distributed between 0 and 14 (with mean 6.34). The number of unique header names was about 60, but the number of occurrences was different. *Host* and *User-Agent* headers were the most popular ones and appeared in more than 99% of requests, followed by *Connection*, *Accept*, and *From*. The authors also analyzed headers' usage patterns, i.e., popularity of headers' sets among requests. The 10 most popular patterns occurred in 81% of requests, and about two-thirds of requests shared one pattern. The authors also observed that the number of headers and usage patterns differed between the browsers and web robots, thus allowing to distinguish them easily.

As already mentioned, some nonacademic sources related to this research are presented below.

In a presentation “HTTP Header Hunter–Looking for Malicious Behavior into Your HTTP Header Traffic,” Montoro presents the scoring system for the HTTP request headers [14]. The system inspects HTTP requests’ features, whitelists and blacklists of the *User-Agent* header values, or top-level domains and the third-party data sources such as geoIP. The analyzed HTTP requests’ features include presence of common headers (for example, *Cookie*, *Accept-Encoding*, and *Connection*), number of header fields in a request, protocol version, *User-Agent* header values’ size, type of files being requested in URI, and presence of the *Host* header in HTTP/1.0 requests. He also proposed the usage of headers’ ordering, response headers, and parameter names; however, this was not implemented in his work. The author’s analysis showed that the malware sometimes does not include *User-Agent* or its value length is usually shorter than 90 bytes. Also, malware tends to send 1–3 headers in requests, and nonmalicious applications usually send more than 9 headers. The presented system adds a score to the features to provide information about maliciousness of requests, and it was tested on 6127 data streams. The resulting detection rate of 89.1% and a false-positive rate of 9.15% have been achieved.

Cuckoo malware sandbox system (https://cuckoosandbox.org/) provides community modules which analyze HTTP protocol traffic. The *network_cnc_http* module [15] provides information about “suspicious features which may be indicative of malware-related traffic.” It analyzes the lack of the *Referer* header in the POST request, the lack of the *User-Agent* header in the *POST* and *GET* requests, the presence of HTTP 1.0 version requests, and the presence of the IP address in the *Host* header. The *multiple_useragent* module [16] verifies whether multiple *User-Agent* header values are used.

Lewis presented a paper about HTTP headers’ heuristics for malware detection [17]. The author proposes utilization of some particular anomalies to help in the malware recognition. These include observing the *User-Agent* string for values which are nonstandard and different from the usual for the particular network, typographic errors in headers’ names and values (additional whitespaces and misspellings of header names), and complexity of the requested resource, e.g., the length of the requested URL.

It must be noted that a large portion of academic research papers focus on describing malware detection systems using the HTTP protocol. A selected representation has been described below.

Mizuno et al. presented in [18] the malware detection system called BotDetector, which uses HTTP requests’ header patterns. The system creates HTTP templates based on header fields; it does not focus on chosen fields, but on all of them. Each field is split into words, which are then evaluated using conditional probability of their appearance in a particular position of the header field. After performing calculations, header fields are clustered using the DBSCAN algorithm, thus producing the HTTP request template.

Li et al. presented a framework for detection and classification of network traffic of malicious Android application [19]. It is based on analysis of the HTTP protocol and organized into 3 components: training module, clustering module, and malware classification and detection module. Training module uses 5 features for model building. It includes values of the headers: *Host*, *Referer*, *User-Agent*, and *Content-Type* and the value of the request URI. This module uses the scoring mechanism which incorporates the header value occurrence frequency and its previous presence in the database.

Kheir in [20] introduced the malware taxonomy based on the *User-Agent* header values, which is used for detecting anomalous values proposed by the author. The *User-Agent* header values are clustered in a two-step process. During the first phase, they are clustered based on high-level features like length of the string and different character type frequencies. In the second step, the values are fine-grained and clustered based on the similarity of value parts. Finally, clusters are tokenized to produce HTTP signatures, used for detection purposes.

Li et al. in [21] presented the detection system for malware traffic. The system uses HTTP requests’ features such as character distribution and the length of the URL, values of *Content-Type* and *User-Agent* headers, and ordering of the header in request.

In [22], the authors presented the malware detection system utilizing analysis of multiple HTTP requests to create behavior models. Statistical models were created for coarse-grained clustering, based on multiple requests of malware using, for example, the average length of the payload, response, or URI. For fine-grained clustering, they used the request method and lexical features of the URI.

To the authors’ best knowledge, the papers presented above have the following disadvantages when compared with the research presented in this paper:(i)The number of malware families, samples, or analyzed HTTP requests is smaller than that in this analysis(ii)The scope and the number of analyzed features are limited(iii)The existing work focuses on presentation of the detection system, without thoroughly (or only to the limited extent) exploring feature analysis(iv)The identification of anomalies is not proved within the existing analyses(v)Features of HTTP requests identified by nonacademic sources were not verified academically

Considering the above, the below work aims at filling these gaps by analyzing more extensively the dataset and providing systematic analysis of a large number of HTTP requests’ features.

Some academic sources provide different approaches to the problem of malware and browser distinctiveness or to a broader problem of detection of malicious behavior. Two examples are presented below, along with discussion about connection to this paper.

Mimura and Tanaka in [23] presented a generic attack detection method based on proxy server logs and URL. The method is independent of attack methods and does not require designing features for classifiers. The authors used the paragraph vector algorithm to capture the context between multiple lines of proxy logs and produce vectors for three classifiers: support vector machine,

random forests, and multilayer perceptron. The experimental results proved that the method can detect unseen drive-by-download attacks and C&C traffic in proxy server logs. Mimura et al. in their paper focused on detection of malware behavior, while in this paper, authors emphasize on the search of particular features, which distinguish malware and browser traffic. Moreover, the method used by Mimura et al. does not require designing of features and can be seen as independent of particular features, whereas in this paper, the authors provided a static list of features, which are analyzed.

Nia et al. in [24] presented a detection method of new generations of cyber threats using the pattern-based random walk. The authors proposed to use a limited method of random walk called the self-avoiding walk, in order to create a behavioral graph based on network traffic. The method uses the ordered triple of *time*, *size*, and *direction*, created for packets in analyzed network flows. The authors created a database of behavioral graphs for known threats. If the analyzed packet set has a similar graph created by the self-avoiding walk algorithm, then it is detected as malicious. The authors reported a true detection rate of 95% for malicious traffic. Nia et al. focused on detection of threats, while authors of this paper emphasize on identification of features distinguishing malware and browser traffic. Moreover, the method by Nia et al. works on flow-level features and is independent of higher network-level protocols, while authors of this paper focused on specific parts of the HTTP protocol, which is an application-level protocol.

3. HTTP Protocol Basics

HTTP protocol in version 1.1 was originally defined in RFC 2616 [25] in June 1999. The RFC has been obsoleted by RFCs 7230–7235 [26–31]. Two earlier versions of the protocol exists: 0.9 and 1.0, where the latter one was defined in RFC 1945 [32]. However, only 1.0 version should still be supported.

The HTTP protocol is based on the client-server architecture, where the client sends a request and the server replies to this request with a response. Request methods defined by RFC 7231 [27] are presented below with short descriptions:(i)*GET*: the primary method for resource retrieval; it usually does not carry payload data, but this is not forbidden(ii)*HEAD*: this method is similar to *GET*, but the server must not send any data in the response body (except for the header section)(iii)*POST*: the method of signalling to the server request for processing data enclosed in the payload(iv)*PUT*: this method is used to create or replace the state of the target resource with the state enclosed in the message payload(v)*DELETE*: according to RFC 7231, this method "requests that the origin server removes the association between the target resource and its current functionality"(vi)*CONNECT*: this method is used to signal proxy request for creation of a connection with a destination server provided in the request(vii)*OPTIONS*: this method is used for discovering information about the communication options available for the requested resource(viii)*TRACE*: this method is used to request the server to resend the request back to the client; it must not contain payload data

An example of an HTTP *GET* request is presented in Figure 1.

```
GET / HTTP/1.1
Host: cert.pl
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64;
            rv:63.0) Gecko/20100101 Firefox/63.0
Accept: text/html,application/xhtml+xml,
        application/xml;q=0.9,*/*;q=0.8
Accept-Language: pl,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

Figure 1

Listing of an exemplary HTTP request (some of the lines were wrapped to fit the table).

The first line in the request in Figure 1 indicates the request method, "GET," requested Uniform Resource Identifier (URI)—"/," and the protocol statement with protocol version—"HTTP/1.1." At the end, the Carriage Return Line Feed (CRLF) is added. Further lines contain header fields, each with a field name and a field value, separated by a colon ":". Field names are case insensitive, and field values consist of printable US-ASCII characters. In practice, the majority of the HTTP client implementations follow such behavior and use printable US-ASCII characters in the header fields. Additionally, RFC 7230 obsoleted the usage of non-US-ASCII characters in these fields.

According to RFC 7230, there cannot be any space or horizontal tabulator between the field name and the colon. There can however be any number of such characters between the colon and the field value, as well as between the field value and the end of the header field. Usually, there is only one space before the field value and no whitespace characters at the end of the field. Additionally, the header field order does not have any special meaning.

According to Section 5 of RFC 7231, the reason behind the usage of header fields is "[...] to provide more information about the request context, make the request conditional based on the target resource state, suggest preferred formats for the response, supply authentication credentials, or modify the expected request processing." Moreover, the header fields are "ought to be registered with IANA" (according to Section 3.2.1. of RFC 7230). The registry can be accessed at https://www.iana.org/assignments/message-headers/message-headers.xhtml.

Despite a rather extensive number of registered header field names (or shortly, headers), some of them are more popular than others. Many of these fields are included in this analysis. They are listed below:(i)*Host*—carries information about the host, port, and target URI; this header field must be present in all requests of the HTTP protocol version 1.1(ii)*Accept*—specifies response media types that are acceptable by the client(iii)*Accept-Language*—characterizes the set of natural languages preferred by the client in the response(iv)*Accept-Encoding*—depicts acceptable content codings(v)*User-Agent*—indicates which application is the source of the request(vi)*Connection*—specifies control options of the connection desired by the client(vii)*Referer*—points to the source URI from which requested URI originates

Once the set of header fields is in place, an additional CRLF tag is inserted. At this point, a message body can be added. The message body may end with the CRLF.

The message body can contain encoded data if the original payload is compressed. The popular methods used for this purpose are *deflate* and *gzip*. Additionally, data can be divided and encoded with chunked transfer coding. In such a way, parts of the data are sent using chunk-size information.

RFC 7230 also defines pipelining mechanism. When using this communication mode, the client can send multiple requests without waiting for corresponding responses.

4. HTTP Traffic Analysis Overview and Experimental Methodology

4.1. HTTP Traffic Analysis Overview

An overview of the malware HTTP requests' analysis workflow is presented in Figure 2. The process of analysis begins with choosing pcap files which contain HTTP request traffic from datasets. The files are filtered with the tshark (https://www.wireshark.org/docs/man-pages/tshark.html) filter so that only the TCP protocol segments containing HTTP requests that are not OCSP (Online Certificate Status Protocol) requests remain. Files with HTTP requests are then fed to an IDS system. In the proposed approach, Snort IDS with ET Pro rules (https://www.proofpoint.com/us/threat-insight/et-pro-ruleset) and Snort registered rules (https://www.snort.org/downloads/#rule-downloads) are used to check for alert logs triggered by the network traffic within the pcap files. Labeling of HTTP requests begins with semimanual check of the generated alerts. They are reviewed in order to filter out those which do not present HTTP traffic, or without information about maliciousness, or alerting about nonmalicious applications or services, for example, Tor traffic. The top 10 most common Snort IDs (SIDs) with alert messages after semimanual filtering are presented in Table 1. Alerts for Trojan Dridex and ransomware Locky are frequent in the dataset, and this impact will be discussed in Section 4.2. In the next step of request labeling, every unique SID is labeled manually with the malware family name, depending on the name provided by the alert message. If no family name is present, it is labeled as *No-name*. If different variants of family names are present (occurring when various vendors/malware analysts provide different names), they are normalized to one name. In the final step of HTTP request labeling, every request is assigned with a set of Snort IDs alerted for a particular request. The assignment is done automatically, on the basis of correlation of tuple: *source IP address, source port number, destination IP address, destination port number,* and *timestamp* between the tshark output for every request and the corresponding tuple in the IDS alert set. Please note that the *timestamp* values are transformed and normalized, in order to prevent any time deviations, if the request timestamp reported by tshark is different than that reported by the IDS. If the tuples are the same, the request is assigned with a particular SID of the alert, along with the malware family name. In the case of multiple SIDs assigned to one request, but with different family names, the request is analyzed manually to provide the final label.

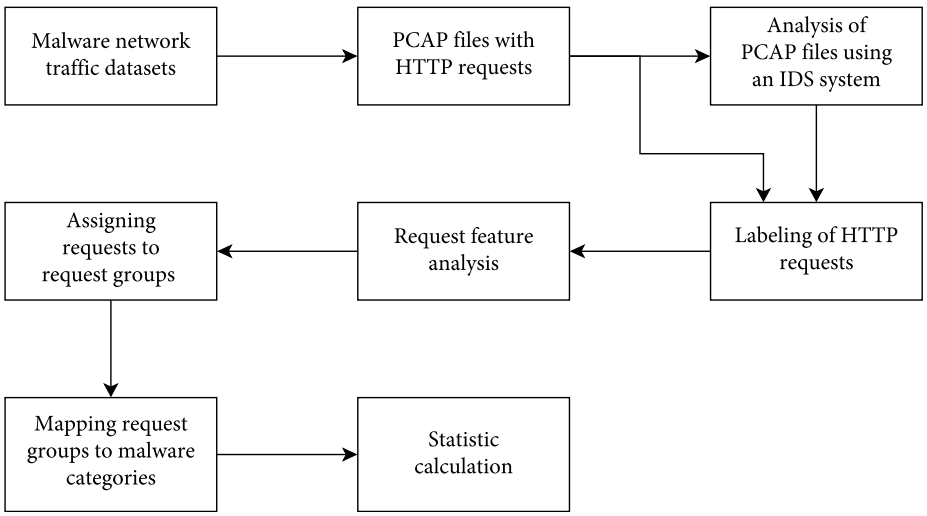


Figure 2

An overview of the malware HTTP requests' analysis workflow used in the paper.

Snort ID	Alert message
1:43685:1	MALWARE-OTHER Win.Trojan.Nemucod variant outbound connection
1:2023577:1	ET TROJAN Locky CnC Checkin HTTP Pattern
1:32678:2	MALWARE-CNC Win.Trojan.Dridex variant outbound connection
1:33145:2	MALWARE-CNC Win.Trojan.Dridex initial outbound connection
1:2019478:1	ET TROJAN Dridex POST Checkin
1:2023551:1	ET TROJAN Locky CnC checkin Nov 21
1:2023552:1	ET TROJAN Locky CnC checkin Nov 21 M2
1:2807610:2	ETPRO TROJAN DirtJumper DDoS (INBOUND)
1:2016879:2	ET POLICY Unsupported/Fake Windows NT Version 5.0
1:2821731:3	ETPRO CURRENT_EVENTS MalDoc Request for Payload Aug 17, 2016

Table 1

The top 10 most common Snort IDs and alert messages observed in the analyzed traffic.

HTTP requests labeled as malicious are evaluated using feature analyzers. The feature list is static and is discussed in Section 4.3. The analyzers utilize popular tools to perform the actual analysis of the features. Feature extraction and analysis process can be divided into three steps: (i) analysis of basic features, (ii) analysis of complex features, and (iii) analysis of payload features.

Basic feature extraction and analysis covers all features which can be analyzed by their direct value extracted from the HTTP request, for example, version of the protocol, the type of the request method, or the popular header values. For the extraction process, tshark is used to provide values of particular fields, supported by the tool, for example, the *http.request.method* for extraction of the request method.

Extraction and analysis of complex features covers the process of obtaining values for features which are not direct values of request fields but involves further analysis of such fields. These features include, for example, verifying the presence of unusual whitespace characters or non-US-ASCII characters in the header values. The process is performed by using the Scapy Python library (https://scapy.net/) for analysis of pcap files and extraction of HTTP requests. The actual analysis of data provided by Scapy is continued using Python scripts, depending on the particular feature.

The final step of feature extraction and analysis is performed for requests which have payload. Firstly, tshark Lua scripts are used for extraction of the payload data. Then, a Perl script is used to detect the presence of the non-US-ASCII characters in the extracted payload. Finally, the payload entropy is calculated using *ent*—a pseudorandom number sequence test program (http://www.fourmilab.ch/random/).

After analyzing request features, the requests are assigned to request groups and malware categories in order to prepare data for statistic calculation. This part of the process is extensively described in Section 4.2.

It must be noted that the benign browser-based HTTP traffic was directly analyzed using the same set of analyzers as described above. Additionally, it was also fed into the IDS system in search for any traces of malicious traffic. The results did not show any significant alerts.

4.2. HTTP Traffic Statistic Calculation

The traffic statistics for the malicious dataset rely on grouping analyzed requests into different sets of requests, called request groups. Requests form a request group when they trigger the same alerts at the IDS. Exemplary relations between inspected HTTP requests and request groups are presented in Figure 3.

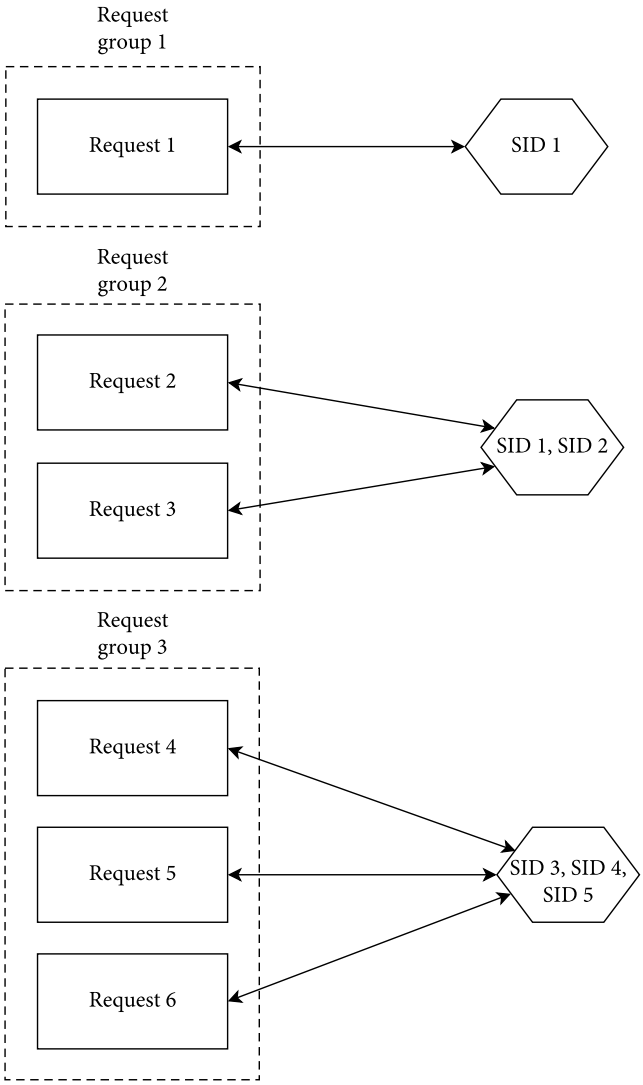


Figure 3

An example of assigning HTTP requests into request groups based on the Snort ID rules reported by this IDS.

The mechanism of assigning HTTP requests to particular request groups is as follows. For all HTTP requests reported as malicious, Snort IDs (SIDs) indicated for this particular request are analyzed, and a corresponding vector of SIDs is created (this part of the procedure is presented in Section 4.1). All HTTP requests with the identical SID set (i.e., the same vector) are put into the same request group. As presented in Figure 3, SIDs can overlap between the request groups (see, e.g., SID 1); however, only unique SID vectors are treated as distinct. Therefore, {SID 1} and {SID 1, SID 2} groups are treated as different groups, as well as distinct from the {SID 1, SID 2, SID 3} request group. The motivation behind it is that the HTTP requests triggering similar but a bit different set of IDS rules are also a little different from each other.

If not specified otherwise, statistics of the HTTP features are calculated based on the request groups and not based on single requests. An example of statistic calculation is presented in Figure 4. When considering how often request methods are prevalent, it must be verified in how many of the request groups a particular method is present. Every request in a request group is checked; if in all of them the method is present, the request group is treated as one entity from the statistical point of view. In the example presented in Figure 4, such a situation occurs for request groups 1 and 3 with the *GET* method and for the request group 4 with the *POST* method. In the request group 2, both *GET* and *POST* methods are present; thus, such a request group is reported as having multiple values. Depending on the feature, such cases are rather infrequent. Final results for the abovementioned example indicate that the *GET* method was present in 50% of request groups, *POST* in 25%, and multiple values were present in 25% of request groups.

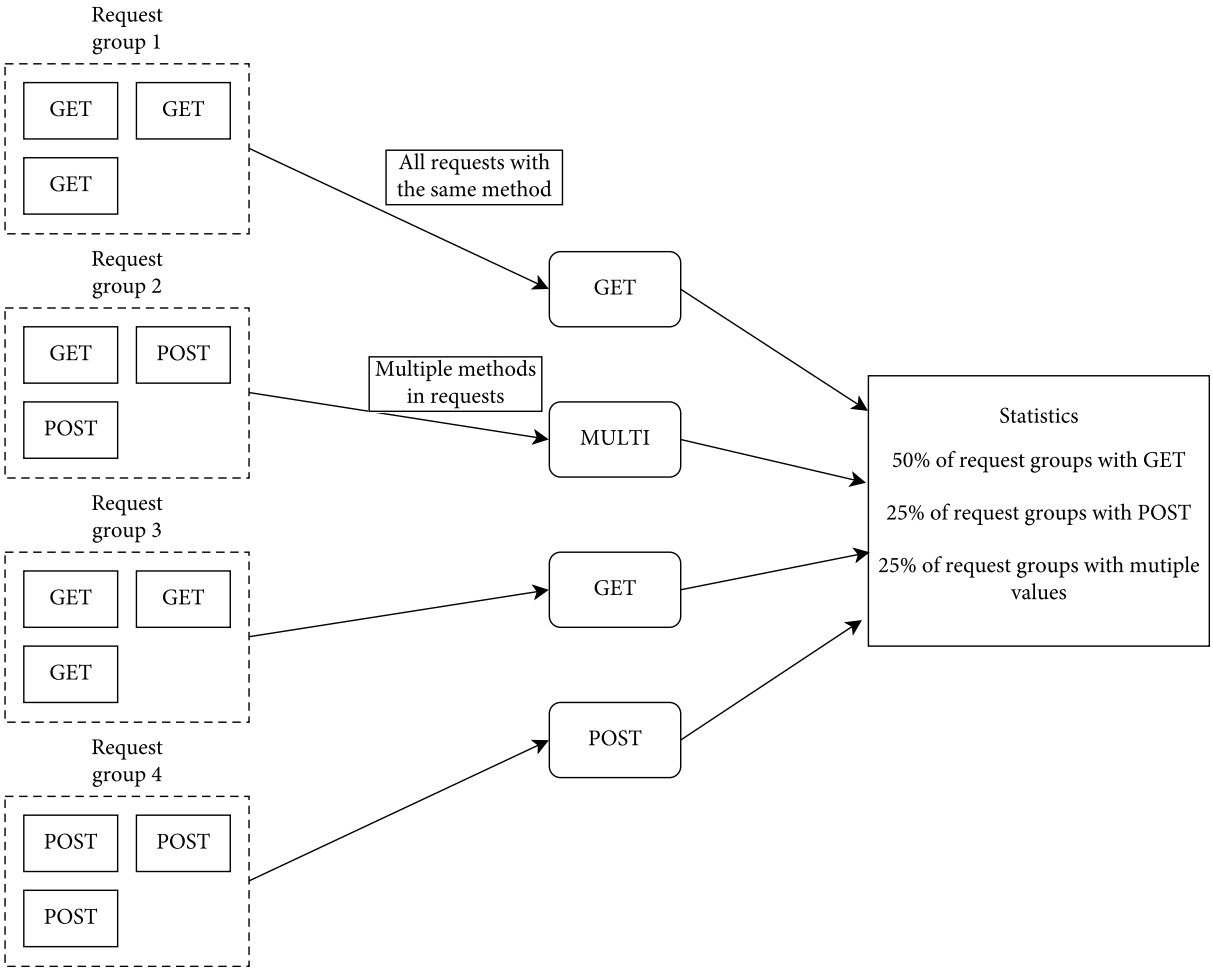


Figure 4

An example presenting statistic calculation methodology used for result analysis.

It is worth noting that the statistics of the benign dataset are calculated directly on requests—in this case, the requests are not grouped as they do not trigger IDS alerts; thus, it is impossible to group them.

Request groups are further divided depending on the type of malware they represent. The main idea behind such a presentation of results is to provide potential insights into characteristics of various malware categories, which can often demonstrate different operational behavior.

As in the example introduced earlier, statistics of the *GET* request occurrences are calculated using request groups of a particular malware category. The algorithm is the same as the one described before; i.e., every request in a request group is checked, and if all of them are *GET* requests, the request group of the category is treated as one unit for the statistics. When in 2 out of the 4 request groups of the category a certain feature is present, the results will show its 50% occurrence in this category.

The name of the related malware was obtained from the IDS alerts and was used for classification purposes. The request groups were divided into 20 categories, presented in Table 2, along with the number of request groups in each category. These categories were based on information provided by the IDS rule comments, information from malware dissection articles from the Internet, and from own experience. Request groups were labeled semimanually.

Category	Description	Number of request groups
Downloader	Downloading other malware	134
Banker	Banking Trojan	125
Trojan	Trojan malware	117
Ransomware	Crypting files and demanding ransom	85
Stealer	Stealing users' information	45
PUA/Adware	Potentially unwanted applications or adware	30
IP check	Checking IP address or connectivity	28
UA problem	Problem with *User-Agent* header value	26
DDoS	DDoS attack malware	24
Spambot	Sending spam e-mails	20
Malicious download	Downloading other malware	20
Miner	Cryptocurrency mining	18
Maldoc	Downloading other malware	16
Clicker	Ad and link clicking	13
Downloader/JS	Downloading other malware	12
Backdoor	Backdoor Trojan	11
RAT	Remote access Trojan	9
Bruteforce	Bruteforcing, e.g., login panels	9
Other	Other malware	8
Keylogger	User key stroke logging	6

Table 2

Malware categories used to organize the obtained experimental results.

Many of the malware categories mentioned in Table 2 are self-explanatory, e.g., *Banker*, *Spambot*, *Ransomware*, or *RAT*. However, some other classes need to be explained in more detail. The *IP check* category groups requests which were sent to the IP address identification services. In that way, malware typically checks the external IP address of the infected machine or whether there is an Internet connection available. The *UA problem* category contains only requests, which were alerted by the IDS as a problematic *User-Agent* header value but without information about the malware family. The *Downloader* type groups all malware families which are used to download other malware. This class is different from *Downloader/JS*, where in the latter one, the actual code is a JavaScript, while in the former one, it is a binary file (EXE file). Similar to these categories is *Maldoc*, where additional malware is downloaded using malicious documents, for example, Microsoft Word or Excel macros. When the IDS labeled a request as a malware download, but no information about the malware family name was provided, the request was treated as the *Malicious download* type. Request groups of Trojan malware which cannot be assigned to any other specialized category (such as Stealer, Banker, or Clicker) were treated as the catch-all *Trojan* kind. Finally, the group request which could not be incorporated into any other category is ascribed to the *Other* class.

The reason behind such a request arrangement (both in request groups and in malware categories) is to limit the effect of inequality of the number of requests between malware families. For example, the *Locky* ransomware family is represented in our dataset by one of the biggest number of requests (ca. 180,000) which constitutes almost 30% of all requests. Without the proposed categorization, such requests would have a tremendous impact on the presented results.

It must also be noted that the quantity bias was not fully eliminated. Even after introduction of request groups, families with a large number of requests can still have a higher number of request groups. The approach taken in this paper regarding malware categories can limit this impact, but it cannot eliminate it completely. The authors of this paper believe that it is a trade-off between bias of the dataset and identification of potential anomalies in the broader datasets.

4.3. Analyzed HTTP Traffic Features

The analyzed HTTP requests' features for the purpose of this paper were assigned into 3 categories, with each of them representing different aspects of the request characteristics: (i) HTTP request structure, (ii) header field values, and (iii) HTTP request payload feature groups. Such an approach was based on the authors' malware behavior analysis experience and previous works of other researchers, as discussed in Section 2. Table 3 outlines the relation between the research source and the corresponding feature category. The description of the proposed feature groups is presented below, along with information, in which features were proposed by the authors of this paper, based on their experience. Such features in this paper are marked with "".

Feature group	Research sources
HTTP request structure	Montero [14], Cuckoo, Calzarossa et al. [13], Rossow et al. [11], Nelson [12], Li et al. [21]
Header field values	Montero [14], Lewis [17], Mizuno et al. [18], Calzarossa et al. [13], Li et al. [19], Kheir [20], Rossow et al. [11], Nelson [12], Li et al. [21], Perdisci et al. [22]
HTTP request payload	Perdisci et al. [22]

Table 3

The relation between the research source and the analyzed HTTP request feature groups.

4.3.1. HTTP Request Structure Features

The analysis of the HTTP request structure involves checking the form of the request, i.e., occurring headers, protocol control information, structure of the fields, and, as an extension, TCP protocol destination port of the request. The features are presented in Table 4.

Feature name
HTTP protocol version
Request method
Repetitions of the header (two header fields with the same name)
Lack of colon in the header field
Number of headers in the request
Frequency of the headers' occurrence
Misspellings of the header names
Presence of request pipelining
TCP destination port in the request

Features proposed by the authors of this paper are marked with (an asterisk).

Table 4

List of HTTP requests' structure features.

Repetition of some headers is a known method for HTTP requests' smuggling through network devices such as firewalls and web proxy servers (cf. [33]). In this research, it is utilized to identify errors of malware developers, such as unskillful change in the header value.

4.3.2. Header Field Value Features

Header field values were examined in order to verify whether any of them are invalid or significantly different from others. Also, the presence of some additional or unusual whitespace characters was verified as well as the presence of non-US-ASCII characters (from this point onward, non-US-ASCII and non-ASCII will be used interchangeably). Evaluation of the *User-Agent* header was performed to obtain a list of names which malware presents itself to the server. During analysis of the *Host* header value, the type of the value was determined; it was verified whether it was an IP address, domain name, or some other value. The feature list is presented in Table 5.

Feature name
First character of the header field is a whitespace
Whitespace before CRLF tag
Space before colon, semicolon, or comma
New line character other than CRLF
Double space
Nonstandard whitespace characters in the header field
Non-ASCII value in the header
Accept-Language header value
Accept-Encoding header value
Connection header value
Host header value
User-Agent header value

Features proposed by the authors of this paper are marked with (an asterisk).

Table 5

List of header field value features.

The *Host* header value was analyzed for the value types as presented in Table 6.

Host header value type
IP address
Domain name
IP address with the port number
Domain name with the port number
Error in the domain
Other value

Table 6

List of *Host* header value types.

Host header value is frequently analyzed in the existing works (for example, Li et al. [19]). In this research, it was inspected to establish its value type, including error values. As such, according to the best of the authors' knowledge, it is the first attempt to provide such information in a general manner.

4.3.3. HTTP Request Payload Features

Analysis of the payload data includes features as presented in Table 7.

Feature name
Payload data length
Payload entropy value
Presence of non-ASCII characters in the payload
Presence of non-POST requests with the payload
Presence of *Referer* header in the POST request

Features proposed by the authors of this paper are marked with (an asterisk).

Table 7

List of HTTP request payload features.

Its evaluation was performed on the data after decoding/decompression and dechunking and not on the data as seen on the wire. The reason behind it was to analyze the final payload, excluding the influence of compression and chunking mechanisms on the analyzed payload data. Please note that in the "presence of non-ASCII characters in the payload" feature, "non-ASCII" is meant as "non-US-ASCII," and it will be used in the shorter form in this text.

4.4. Data Sources

Two data sources were used in the conducted investigations. As the sources of malicious HTTP traffic, pcap files from CERT Polska's sandbox systems and Malware Capture Facility Project (MCFP) (https://www.stratosphereips.org/datasets-malware) were used. Basic information about these two sources is presented in Table 8.

Feature	CERT.pl	MCFP	Sum
No. of pcaps in repository	36,268	117	36,385
No. of pcaps with HTTP network traffic	26,042	91	26,133
No. of pcaps with HTTP network traffic containing requests alerted by IDS	22,630	67	22,697
No. of reported IDS alerts	2,133,682	425,441	2,559,123
No. of reported IDS alerts assigned to requests	405,116	238,805	643,921
No. of unique alerted IDS rules	578	139	642

Table 8

Basic information about malicious pcap repositories.

PCAP files from CERT Polska's sandbox environment were generated by a Windows-based malware, analyzed in 2016–2018. The malware samples originate from automatic systems and incident reports. The former represents systems which collect samples from CERT Polska's internal malware hunting systems and publicly available sources provided by various entities, including Shadowserver (https://www.shadowserver.org/) or Abuse.ch (https://abuse.ch/). The incident reports which provided malware samples were reported mainly by Polish citizens (CERT Polska acts as a Polish national CSIRT) and also by researchers and other entities outside of Poland. All malware samples were acquired during the period of 2016–2018 and represent malware encountered in the wild. The malware analysis system consisted of Windows 7 virtual machines orchestrated by the modified Cuckoo Sandbox system. Main modifications were introduced into hardening the system against anti-VM and anti-analysis techniques and into process monitoring services. MCFP repository is maintained at the Czech Technical University in Prague and consists of pcap files from the long-term Windows malware observations. Both repositories represent popular malware families.

For the legitimate browser traffic, the authors decided to generate it on their own. Various web browsers under control of different versions of the Windows OS were used, as depicted in Table 9. This table also contains a number of analyzed requests for each web browser. The browsers were instrumented using the Selenium automation toolset (https://www.seleniumhq.org/) to visit websites from the list of 500 most popular websites worldwide. The list was created using the Alexa top 1 million websites worldwide (http://s3.amazonaws.com/alexa-static/top-1m.csv.zip). The websites were accessed between 9 and 15 February 2017 and between 13 and 18 October 2017, depending on the browser.

Browser name	Operating system	Abbreviation	Number of requests
Microsoft Edge	Windows 10	Edge Win10	17,912
Google Chrome	Windows 7	Chrome Win7	30,621
Mozilla Firefox (Adobe Flash Player installed)	Windows 7	Firefox-FP Win7	18,705
Mozilla Firefox	Windows 7	Firefox Win7	28,178
Microsoft Internet Explorer 11	Windows 7	IE11 Win7	30,799
Google Chrome	Windows 8.1	Chrome Win8.1	23,967
Mozilla Firefox	Windows 8.1	Firefox Win8.1	18,153
Microsoft Internet Explorer 11	Windows 8.1	IE11 Win8.1	20,248

Note. Abbreviations introduced here are used in the paper to refer to the specific environments.

Table 9

Networking environments in which HTTP traffic was analyzed.

Table 10 presents the top 5 malware families in the categories grouped by the number of request groups. It should be noted that 18.67% of request groups were sent by an unknown malware. In the table, such request groups are marked as *No-name*.

Family name	Number of request groups

Backdoor	
Htbot	3
GrayBird	2
Dimnie	2
Zeprox	1
Votwup.D	1
Mokes	1

Banker	
Ursnif	27
Dreambot	24
Chthonic	12
Emotet	11
Kronos	10

Bruteforce	
No-name	6
Pifagor	2

Clicker	
KOVTER	6
Zeroaccess	4
Sefnit	2
Miuref/Boaxxe	1

DDoS	
DirtJumper	17
MegalodonHTTP	4
Madness	2
MedusaHTTP	1

Downloader	
Pony	21
Nemucod	19
SmokeLoader	17
Locky	12

Zbot	11
Downloader/JS	
No-name	8
Cryxos	4
IP check	
No-name	28
Keylogger	
AgentTesla	3
Keybase	2
KeyLogger.acqh	1
Maldoc	
No-name	16
Malicious download	
No-name	20
Miner	
No-name	11
Adylkuzz	4
1ms0rry	2
Smominru	1
Other	
FakeAlert.jh	3
Ratankba	1
Psiphon	1
No-name	1
DustySky	1
PUA/Adware	

Wizzcaster	3
InstallCapital	3
BubbleDock	3
Sureseeker	2
OfferCast	2

Ransomware	
Locky	38
AlphaCrypt	8
PadCrypt	4
Sage	3
Fatboy	3

RAT	
Quasar	2
XPCSpyPro	1
TViewer	1
Teamspy	1
ShinoBot	1

Spambot	
Kelihos.F	8
Necurs	5
XnxxAgent	3
Sality	3
Tofsee	1

Stealer	
AZORult	11
Loki	10
FormBook	6
WernikStealer	2
Hawkeye	2

Trojan	
Zbot	29

No-name	16
Andromeda	12
Graftor	7
Betabot	6

UA problem	
No-name	26

Table 10

Top 5 malware families in categories grouped by the number of request groups.

5. Experimental Results

In this section, experimental results of analysis of features presented before in Section 4.3 are outlined. The presentation of the results uses the categorization introduced there.

5.1. HTTP Request Structure Features

Some of the analyzed features did not show any results in the malware and browser HTTP traffic. These are the repetitions of the header (two header fields with the same name) and presence of pipelined requests; i.e., multiple requests are sent without waiting for their corresponding responses. The lack of colon in the header field was not observed in malware traffic and was present in only 4 requests of Internet Explorer 11 on Windows 7 and Windows 8.1 (two in each browser), i.e., in about 0.01% of requests. Also, analysis of HTTP request methods showed that it cannot be directly used to distinguish between malware and browser traffic. It is however indicated that browsers mostly sent the *GET* request, while a significant portion of malware requests are *POST* requests.

5.1.1. HTTP Version

The results of the analysis of the HTTP protocol version in malware traffic are presented in Figure 5. The majority of the analyzed malware families grouped in categories usually used version 1.0 of HTTP. The highest level of occurrence of the version 1.1 (i.e., 42.22% of requests groups) has the malware in the *Stealer* category, while the other categories have a lower number of request groups for this version of the protocol. Banker, Downloader, and Trojan have about 15% of request groups with version 1.1. RAT and PUA/Adware about 10%. There are 6 categories with nonzero levels up to 10% of requests groups. Finally, 6 categories of malware do not have any requests with version 1.1.

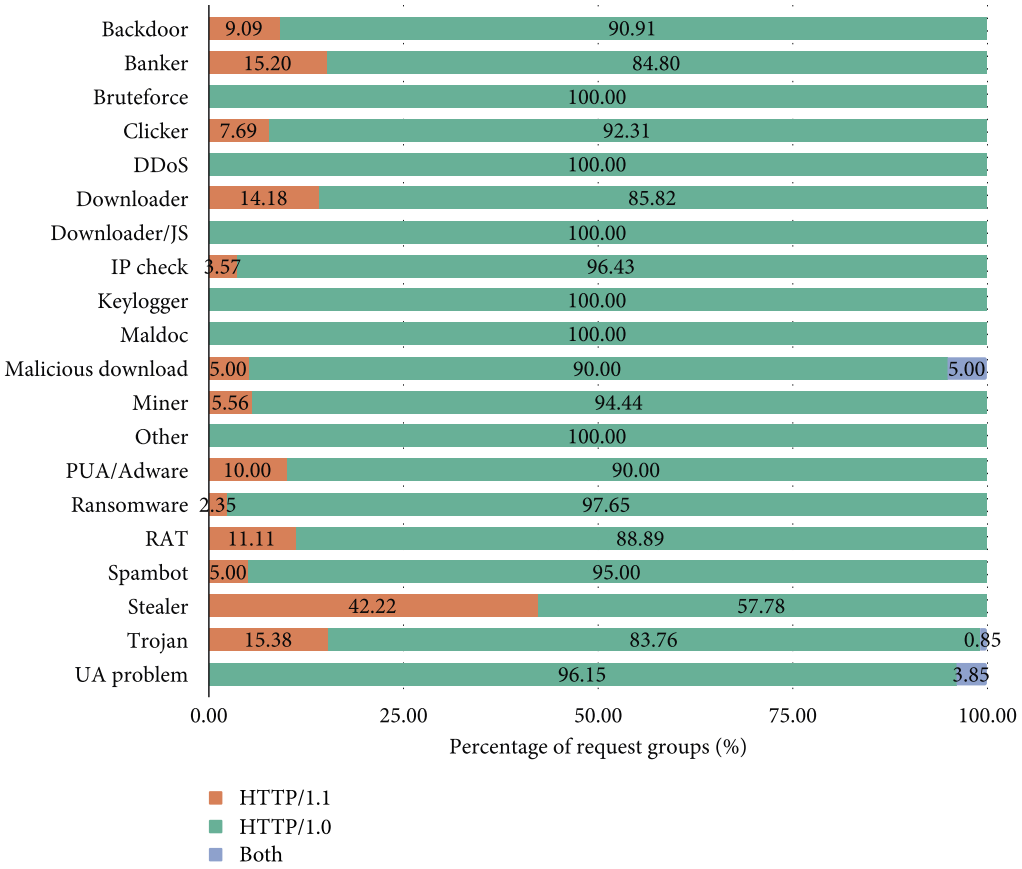


Figure 5

HTTP protocol version (malicious traffic).

The same analysis related to the benign browser traffic showed that only in Internet Explorer-based traffic running under Windows 7 and 8.1 OSs, HTTP requests with version 1.0 occurred (0.01% and 0.08%, respectively). The comparison of the results for malicious and benign traffic shows significant difference in protocol version usage. Therefore, in the authors' opinion, this feature is a good candidate in selection of features distinguishing malware from browsers.

5.1.2. Number of Header Fields

Analysis of the number of headers in the request groups shows that the number of headers varies between 1 and 11. The results are presented in a graphical form in Figure 6. Their analysis shows that in most categories, there were less than 8 headers and up to 6 headers in categories such as *Clicker*, *DDoS*, *Maldoc*, *Miner*, *PUA/Adware*, or *Spambot*. In many categories, 5 headers in a request is a dominant value.

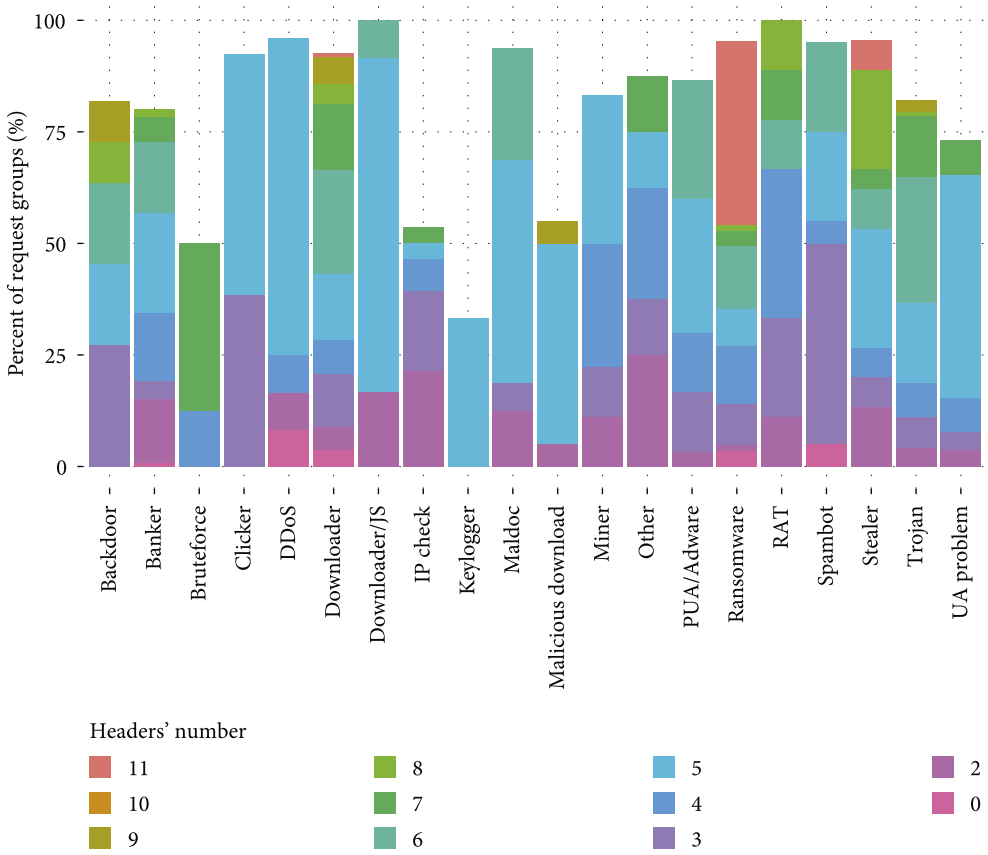


Figure 6

The number of headers in a request (malicious traffic).

Most categories have request groups with multiple values of headers' number. These include 4 categories with more than 40% of request groups (*Bruteforce, IP check, Keylogger,* and *Malicious download*). All of them were analyzed further, and results indicate similar header number ranges as in the single header number value request groups.

The number of headers in browser requests is in the range from 0 to 24 headers. However, for every browser, the number of headers was in the range between 0 and 11 in at least about 99% of requests. Results for this range are presented in Figure 7 where percentage results of the number of headers in the browser traffic requests are illustrated. The most common number of headers is 7 and 8.

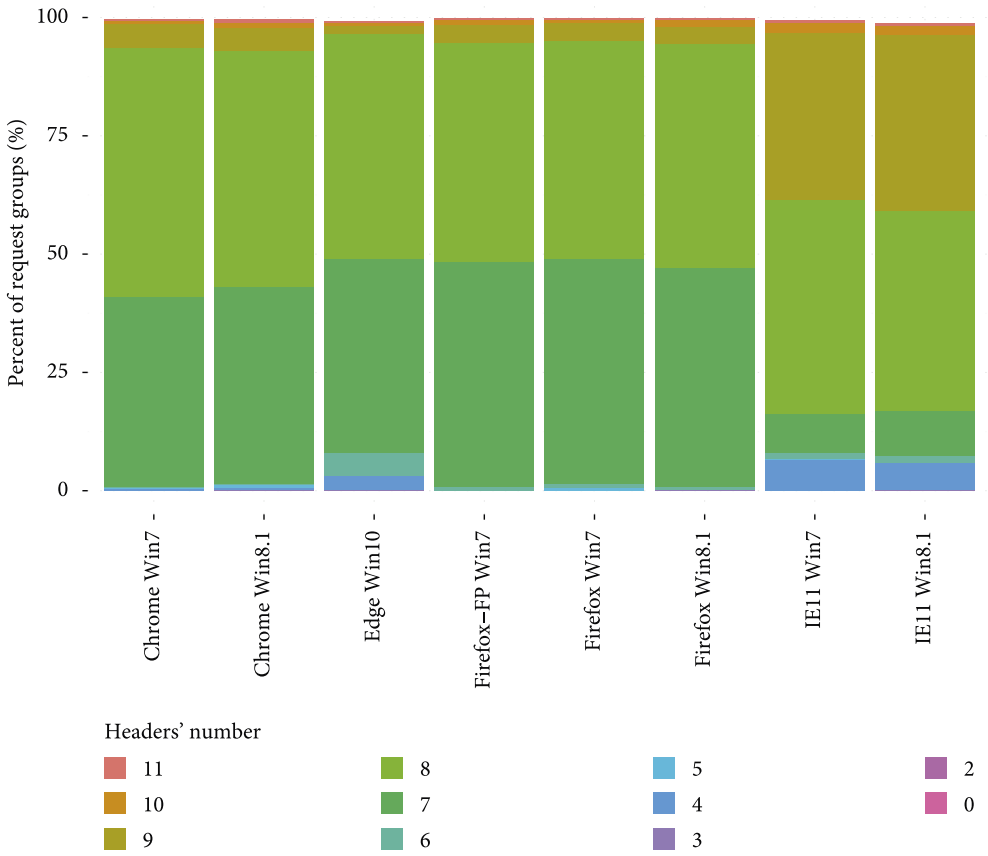


Figure 7

The number of headers in a request (benign traffic).

However, the ranges of the number of headers for malware and browser traffic overlap, and their distributions are different. As already mentioned, in the benign traffic, most of the values are close to two maxima (7 or 8 headers in a request). For malicious traffic, the majority of requests has up to 6 headers. From this perspective, the number of headers in the request can be perceived as a useful feature to distinguish malware and browser HTTP traffic.

5.1.3. Header Occurrence

The top 10 headers sorted by their average frequency of occurrences in the benign traffic are presented in Table 11. The first 7 headers occurred in all browsers in at least about 90% of the requests. Some of the well-known headers did not occur in the top 10, for example, *Origin* (3.97% of all requests in all browsers), *Content-Type* (1.21% of all requests), *Cache-Control* (1.18% of all requests), or *Content-Length* (0.95% of all requests). Nonstandard headers, which begin with prefix *X*, were also observed. Some of them are relatively known, e.g., *x-flash-version*, but others are server platform specific, e.g., *X-TeaLeaf-Browser-Res*. One of the headers observed in the benign dataset was particularly interesting. The "_" (an underscore) header was present only in two requests sent to *unid.go.com* on Windows 7 OS by Google Chrome and Microsoft Internet Explorer 11. This network traffic is associated with the content delivery networks (CDN) operations. Additionally, some of the well-known headers were observed written in varying cases, for example, *Authorization* and *authorization*, *Content-Type*, *Content-type*, and *content-type*, and *Accept* and *accept*. Generally, lower case versions were less frequent.

Browser	Header name-percentage of requests									
“Host”	“User-Agent”	“Connection”	“Accept”	“Accept-Encoding”	“Accept-Language”	“Referer”	“Cookie”	“DNT”	“Upgrade-Insecure-Requests”	
Edge Win10	100.00	99.89	100.00	99.86	96.80	95.46	91.05	48.64	1.07	0.00
Chrome Win7	100.00	100.00	100.00	99.79	99.62	99.16	95.62	55.19	0.00	6.12
Firefox-FP Win7	100.00	100.00	100.00	100.00	99.96	99.84	93.97	48.30	0.00	7.35
Firefox Win7	100.00	100.00	100.00	99.99	99.92	98.12	93.86	48.05	0.00	5.64
IE11 Win7	99.99	99.99	99.99	99.98	92.90	91.98	88.33	43.88	78.48	0.00
Chrome Win8.1	100.00	100.00	99.78	99.46	99.39	97.78	93.64	52.02	0.00	6.81
Firefox Win8.1	100.00	100.00	99.82	99.83	99.78	99.77	94.01	49.62	0.00	7.31
IE11 Win8.1	99.99	99.99	99.74	99.70	93.91	92.98	88.65	44.02	80.26	0.00
Average	100.00	99.99	99.93	99.83	97.71	96.77	92.41	48.83	21.54	4.13

Table 11

Top 10 headers in a request (benign traffic).

Table 12 summarizes the presence of particular headers in the requests of malware traffic. The values present the top 10 headers regarding the percentage of all malware categories where the header appeared. Percentages were counted in the request groups in which the header was present in all requests.

Category	Percentage of requests in category									
"Host"	"User-Agent"	"Connection"	"Accept"	"Accept-Encoding"	"Cache-Control"	"Content-Length"	"Content-Type"	"Accept-Language"	"Cookie"	
Backdoor	100.00	100.00	45.45	45.45	27.27	54.55	45.45	54.55	27.27	9.09
Banker	100.00	73.60	75.20	13.60	1.60	77.60	60.80	19.20	4.80	4.80
Bruteforce	100.00	100.00	87.50	100.00	12.50	0.00	87.50	87.50	0.00	25.00
Clicker	100.00	100.00	30.77	7.69	7.69	46.15	61.54	46.15	0.00	0.00
DDoS	100.00	83.33	87.50	0.00	0.00	8.33	4.17	4.17	0.00	0.00
Downloader	100.00	90.30	75.37	55.22	32.84	44.03	36.57	30.60	20.15	0.75
Downloader/JS	100.00	83.33	100.00	83.33	83.33	0.00	0.00	0.00	8.33	0.00
IP check	100.00	67.86	75.00	32.14	28.57	14.29	0.00	0.00	28.57	7.14
Keylogger	100.00	66.67	0.00	0.00	0.00	16.67	66.67	66.67	0.00	0.00
Maldoc	100.00	100.00	81.25	81.25	81.25	6.25	0.00	0.00	25.00	0.00
Malicious download	100.00	75.00	80.00	60.00	40.00	35.00	10.00	10.00	5.00	0.00
Miner	100.00	77.78	50.00	11.11	27.78	0.00	38.89	38.89	0.00	0.00
Other	100.00	75.00	62.50	12.50	12.50	12.50	12.50	25.00	12.50	12.50
PUA/Adware	100.00	80.00	66.67	23.33	13.33	43.33	40.00	33.33	0.00	3.33
Ransomware	100.00	80.00	71.76	62.35	47.06	70.59	77.65	71.76	42.35	1.18
RAT	100.00	88.89	55.56	22.22	22.22	33.33	44.44	44.44	0.00	11.11
Spambot	100.00	70.00	45.00	5.00	5.00	40.00	75.00	35.00	5.00	0.00
Stealer	100.00	57.78	86.67	35.56	13.33	11.11	66.67	68.89	26.67	0.00
Trojan	100.00	90.60	74.36	47.86	12.82	53.85	52.14	35.90	9.40	2.56
UA problem	96.15	92.31	80.77	11.54	15.38	19.23	11.54	11.54	0.00	7.69

Note. The header was present in all requests of a particular request group in the malware category.

Table 12

Top 10 headers present in requests (malicious traffic) sorted by % of all categories where they appeared.

Analysis of unique header names found in malicious traffic shows that besides well-known headers, their versions written in lowercase were also present, for example, *accept-Language* or *Content-type*. Some of the header names cannot be found in any official documentation, for example, *Filename*, *Idle-time*, *Content-Key*, or *Server-Key*. One header *user-* looks like it was created to mimic the *User-Agent* header, but for some reason it remained unfinished.

It must also be noted that no misspellings of header names were found in the observed malicious and benign traffic. The *user-* cannot be categorized as misspelling, but in some way it proves the observation that sometimes malware developers do make errors.

In both HTTP traffic datasets, the header names with alternative case spellings were observed, for example, *Content-Type* and *Content-type*. RFC 7230 does not prohibit such a usage, stating that header names are case insensitive. However, the observed traffic demonstrates that upper-cased first characters are more popular, regardless of the type of the traffic dataset (benign/malicious). The occurrence of the lower-cased version of the header names is low in both malware and browser traffic and therefore is not distinctive enough to show the general difference between the malicious and benign traffic. Nevertheless, it can be more useful for distinction from the perspective of individual malware families.

Based on the presented analysis, it can be concluded that the presence of some particular headers can be used as a feature for distinction between malicious and benign traffic. The list of such headers should include those indicated as the most popular ones in the browser traffic: *Connection*, *Accept*, *Accept-Encoding*, *Accept-Language*, and *Referer*. These headers appear in the analyzed malware traffic less frequently.

Some previous works have been already performed when it comes to the usage of the header order in malware detection or browser fingerprinting, for example, in the p0f tool (http://lcamtuf.coredump.cx/p0f3/). The idea behind it was to check the order in which the headers occur in the HTTP request and to identify the application which sends it. The authors of this paper believe that this problem has not been fully analyzed, and more research is needed.

5.1.4. Destination Port

Destination ports of requests in malicious traffic were also investigated. Unsurprisingly, most of the requests were sent on port 80. 7 malware categories sent requests to other ports, but even in these situations, at least 88% of the request groups used port 80. Two categories (*Banker* and *Ransomware*) sent requests on port 443, which is a registered port for the HTTPS protocol. This behavior can be seen as anomalous regardless of the application type which sent such requests.

Destination ports of requests in benign traffic were also analyzed. It occurred that every browser sent over 99.8% of requests to port 80. However, some other ports were also discovered, e.g., 443, 8080, 880, and 8050.

When comparing traffic results for malware and browsers, one can see that both categories send requests mainly on port 80. Other destination ports occur, but they are not so frequent. The main difference between the browser and the malware HTTP traffic is that the malware uses ports of higher numbers, for example, higher than 10,000 in the *Downloader* category and higher than 40,000 in the *Ransomware* category. This difference can be seen only for single-request groups, and it cannot be confirmed as regular. Thus, the analysis of the utilized destination ports cannot be conclusive to distinguish between the malicious and benign traffic.

Nevertheless, usage of some ports for HTTP traffic is improper, for example, port 443 which is registered by IANA for the HTTPS protocol. Such situation can be anomalous on its own, regardless of the type of network traffic, and is usually alerted by the network monitoring systems.

5.2. Header Field Value Features

In this section, different features of the header field values were investigated. The conducted experiments revealed that some of the features initially selected to inspect (see Section 4) were not present at all in the analyzed HTTP requests. This includes the following features: (i) the presence of the space at the beginning of the header field, (ii) the occurrence of space before colon, and (iii) the appearance of the space before semicolon. Considering the above, the obtained results for these 3 features are omitted. Some other features were observed in the analyzed traffic. They did not however give any significant results that can be utilized for distinguishing between malware and browser traffic because they were rare. Whitespace character before CRLF tag was not present in the browser traffic, but it was present in network traffic of 5 malware categories (of which 2 categories were exceeding 10% of request groups). The next feature is the presence of a space character appearing before a comma in the header field. Its analysis revealed that it was absent in the browser traffic and present only in about 2.35% of request groups of the *Ransomware* category. The new line character other than CRLF was not present in malware traffic, but it was observed in the browser traffic of Internet Explorer 11 on Windows 7 and 8.1 in 2 and 13 requests, respectively. Both values are below 0.1% of all requests for both browsers. The next feature which gave limited results is the presence of a double space in the header field. It was not observed in the malicious traffic; with regards to the browser traffic, only Google Chrome (both on Windows 7 and 8.1) did not send requests with double space in the header field. Mozilla Firefox on Windows 7 presented the highest percentage of such requests (0.21%), but overall, the percentages are low. Finally, nonstandard whitespace characters were not observed in browser requests, and in malicious traffic, it was present only in 1.18% of *Ransomware* request groups. Additionally, analysis of values of *Accept-Language*, *Accept-Encoding*, and *Connection* headers did not indicate any distinguishing features between browser and malware traffic. Thus, the numerical results will be omitted for brevity of the text.

5.2.1. Non-ASCII Characters in the Header Field

Furthermore, the presence of the non-ASCII characters in header fields in the malicious traffic was analyzed. This feature was observed in two malware categories: *Backdoor* (3.57% of request groups) and *Ransomware* (1.18% of request groups). Additionally, in 3 request groups of the *Ransomware* category (additional 3.53%), the feature was present irregularly.

Non-ASCII character in the header field in benign HTTP traffic was observed only for 3 browsers: Chrome, Firefox, and Internet Explorer running on Windows 7 OS. However, it was only one request per browser, which is less than 0.01% of all requests in the respective sets. It was caused by the presence of Polish characters.

It must also be noted that the presence of the non-ASCII character in the header field can be treated as an anomaly itself. It however occurs sporadically in both malicious and benign traffic. It can be considered as an indicator of anomalous traffic, but in the presented form, it cannot be used as a general rule to distinguish malware and browser HTTP traffic.

5.2.2. Host Header

The obtained *Host* header values in the malicious traffic are presented in Table 13. The domain is present as the main value in most of the malware categories. However, for some categories, also the IP address value is noticeable. These include *Ransomware*, *Spambot*, *Clicker*, and *Miner* categories. Some categories also have request groups with multitype values, for example, *Maldoc* has the highest percentage among all categories, i.e., 31.25%.

Category	Domain	IP	IP + port	Domain + port	Error in domain	Other	Multi
% of all	% of all	% of all	% of all	% of all	% of all	% of all	
Backdoor	72.73	9.09	0.00	0.00	0.00	0.00	18.18
Banker	70.40	9.60	8.00	0.00	6.40	0.00	5.60
Bruteforce	87.50	0.00	0.00	0.00	0.00	0.00	12.50
Clicker	69.23	23.08	0.00	0.00	0.00	0.00	7.69
DDoS	100.00	0.00	0.00	0.00	0.00	0.00	0.00
Downloader	76.87	6.72	2.24	0.75	2.24	0.75	10.45
Downloader/JS	100.00	0.00	0.00	0.00	0.00	0.00	0.00
IP check	100.00	0.00	0.00	0.00	0.00	0.00	0.00
Keylogger	66.67	16.67	0.00	0.00	0.00	0.00	16.67
Maldoc	68.75	0.00	0.00	0.00	0.00	0.00	31.25
Malicious download	80.00	10.00	0.00	0.00	0.00	0.00	10.00
Miner	72.22	22.22	0.00	5.56	0.00	0.00	0.00
Other	75.00	0.00	25.00	0.00	0.00	0.00	0.00
PUA/Adware	100.00	0.00	0.00	0.00	0.00	0.00	0.00
Ransomware	40.00	52.94	1.18	0.00	0.00	0.00	5.88
RAT	77.78	11.11	0.00	0.00	11.11	0.00	0.00
Spambot	40.00	60.00	0.00	0.00	0.00	0.00	0.00
Stealer	68.89	13.33	4.44	0.00	0.00	0.00	13.33
Trojan	65.81	17.09	3.42	0.85	0.00	0.00	12.82
UA problem	80.77	11.54	0.00	0.00	0.00	0.00	7.69

Table 13

Host header values (malicious traffic).

The analysis of actual values in the requests with value types defined as *Error in domain* and *Other* was also performed. The results indicate that the domain names contained suffixes such as *.bit*, *.xn–p1ai*, additional "." characters (*.com.*), or were malformed, for example, *7M5 us* or *5t9AR us*. Also, the malformed IP address and the port pair were identified (*5.141.22.43:13404*).

The results of the feature analysis for the benign browser traffic show that in the majority of requests, the domain is present in the *Host* header value, regardless of the browser and OS used. In all browsers, such a value was present in at least 99.8% of requests. Other value types include IP address (maximum value of 0.07% for Chrome browser on Windows 7), domain and port (maximum value of 0.1% in case of Firefox browser on Windows 8.1), and IP address and port. However, the latter ones are negligible for all browsers.

The comparison of results of value types in the *Host* header shows that values other than the domain name are more frequently spotted in the malware traffic. This means that this feature can be used in some cases to discern malware and browser traffic. However, the feature is strongly related with the infrastructure used by cybercriminals. Intuition and malware analysis experience suggest that attackers do use some nonstandard addresses for C&C servers. The results show that it does not happen as often as it could be expected. This research does not analyze the purpose of sending particular requests; nevertheless, some of them are sent by malware to benign addresses, for example, as a connectivity check. This could impact the obtained results.

5.2.3. User-Agent Header

The analysis of the *User-Agent* header strings was performed for both traffic types. Some typical values observed in the browser traffic are presented in Table 14.

Browser	User-Agent value
Edge Win10	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.116, Safari/537.36 Edge/15.15063
Chrome Win7	Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924.87 Safari/537.36
Firefox-FP Win7	Mozilla/5.0 (Windows NT 6.1; rv:51.0) Gecko/20100101 Firefox/51.0
Firefox Win7	Mozilla/5.0 (Windows NT 6.1; rv:51.0) Gecko/20100101 Firefox/51.0
IE11 Win7	Mozilla/5.0 (Windows NT 6.1; Trident/7.0; rv:11.0) like Gecko
Chrome Win8.1	Mozilla/5.0 (Windows NT 6.3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36
Firefox Win8.1	Mozilla/5.0 (Windows NT 6.3; rv:56.0) Gecko/20100101 Firefox/56.0
IE11 Win8.1	Mozilla/5.0 (Windows NT 6.3; Trident/7.0; rv:11.0) like Gecko

Table 14

The standard values of the *User-Agent* header (benign traffic).

Table 15 presents the distribution of the *User-Agent* value types in the browser traffic. The values were analyzed based on their similarity to standard values (presented in Table 14) or lack of them in the *User-Agent* header. All browsers used standard values in at least 91% of the requests. Internet Explorer on Windows 7 and Windows 8.1 OSs in about 1% of the requests used values similar to the standard ones but slightly expanded in some fields, e.g., *Mozilla/4.0* (*compatible; MSIE 7.0; Windows NT 6.1; Trident/7.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C*).

Browser	Main UA	Similar to main UA	No UA	Other
% of all	% of all	% of all	% of all	
Edge Win10	96.81	0.00	0.11	3.08
Chrome Win7	99.19	0.00	0.00	0.81
Firefox-FP Win7	99.84	0.00	0.00	0.16
Firefox Win7	98.12	0.00	0.00	1.88
IE11 Win7	91.53	1.11	0.01	7.35
Chrome Win8.1	97.91	0.00	0.00	2.09
Firefox Win8.1	99.77	0.00	0.00	0.23
IE11 Win8.1	91.99	1.63	0.01	6.37

Table 15

The distribution of the *User-Agent* header value types (benign traffic).

Also, these two browsers experienced a higher percentage of values not similar to the standard User-Agent strings (7.35% and 6.37%, respectively). Some significant results were also noted for the Microsoft Edge, Mozilla Firefox on Windows 7, and Google Chrome on Windows 8.1. For all Microsoft browsers (Internet Explorer 11 on Windows 7 and 8.1 and Edge) the main part of the requests with nonstandard values consists of the request sent by modules responsible for downloading certificate revocation lists [34]—*Microsoft-CryptoAPI*. This mechanism is utilized by browsers to download the current sets of revocated X.509 certificates used in the HTTPS protocol communication. Other *User-Agent* header values were also present, for example, *Microsoft BITS* or *Microsoft-WNS* which can be used by OS mechanisms like Windows Update or Windows Push Notification Services from Windows 8 onward. The usage of the *Microsoft BITS User-Agent* found in the Chrome browser HTTP traffic (both for Windows 7 and 8.1) can be attributed to the update mechanism of this browser.

Four browsers requests without the *User-Agent* string were present. In overall, this was applied to less than 0.1%. These requests were sent by system mechanisms or generated along with the web page activity.

In the end, the authors decided to leave such requests in the dataset as it is not known with certainty which requests were sent by the browsers and which were not. The value of the *User-Agent* header could be misleading, or assumption about the system service could be erroneous. Additionally, the number of nonbrowser requests is not high and thus should not introduce much noise into the results.

For the *User-Agent* header, the analysis performed on the malicious HTTP traffic uncovered 6218 unique values of the *User-Agent* header. These values were further analyzed in order to establish well-known browser results as malware developers typically try to mimic the behavior of the benign software. The results of this analysis are presented in Figure 8, and they are grouped by the popular browser names and lack of the *User-Agent* header or the nonstandard value. If the requests carry many different values from any of these classification groups, they are classified into the *Misc* group.

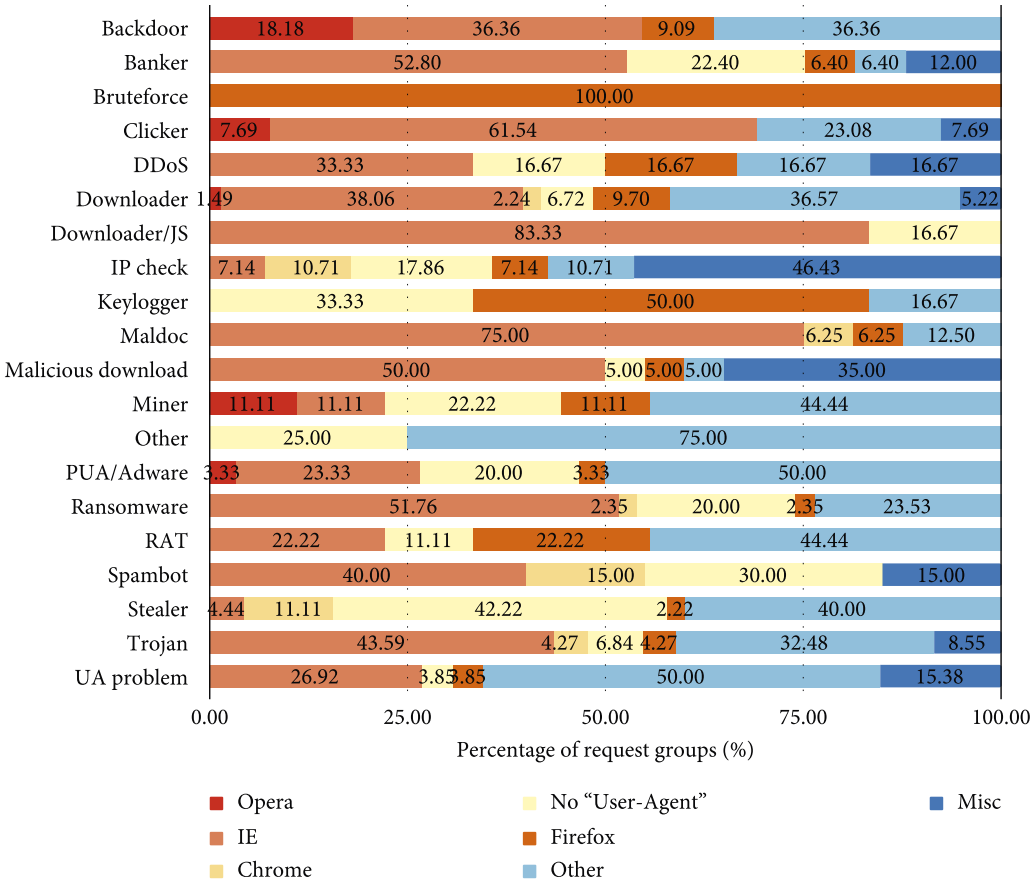


Figure 8

Browser string as reported in the *User-Agent* header (malicious traffic).

From the 4 browsers indicated in Figure 8, most of the requests include Internet Explorer or the Firefox *User-Agent* string. Also, only 4 categories (*Backdoor, Bruteforce, Downloader/JS,* and *Spambot*) do not have requests without the *User-Agent* header. Some categories have a high percentage of the *User-Agent* values other than those 4 standard ones, e.g., *Miner, Other, PUA/Adware, RAT*, or *Stealer*.

Nonstandard *User-Agent* header values discovered in the malicious HTTP traffic present values of the software modules/libraries, such as *LuaSocket 2.0.2*, *AutoIt*, or *python-requests/2.12.4*. Some others include meaningless values, e.g., *W1pbbA((*, *EMSFRTCBVD*. Others upload some system information, presented below with regular expression or obfuscated for privacy purposes: *regexp: ^\{[A-Z0-9]{8}-[A-Z0-9]{4}-[A-Z0-9]{4}-[A-Z0-9]{4}-[A-Z0-9]{10} \}$*, *C:Users[user's name]AppDataRoaming v2o5g0Ie5itemp.zip*. Finally, some of them present directly their name and purpose, e.g., *TrickLoader* or *Botnet by Danij*.

It must also be noted that additional examination should be applied to the requests without the *User-Agent* header. They are hard to be found in the browser HTTP traffic, but they are present in the majority of malware traffic—from 3.85% of the request groups of the *UA problem* category up to 42.22% for the *Stealer* category. The authors believe that the lack of the *User-Agent* header can be used to distinguish between the malicious and benign HTTP traffic.

5.3. HTTP Request Payload Features

The analysis of the payload data length did not give any significant results in distinction between malware and benign traffic. However, the authors of this paper believe it should be analyzed with a more specialized approach than used in the study, giving more specific results.

5.3.1. Payload Entropy

In Table 16, the results of analysis of the request payload entropy for malicious traffic are presented. As was the case with the payload length, the statistics are based on the values of the payload entropy inside malware categories, but the values are not organized into request groups. On the other hand, the investigation of the same feature in the browser traffic is presented in Table 17.

Category	Median	Mean	1st quartile	3rd quartile	Min value	Max value
Backdoor	5.86	5.85	5.86	5.86	4.08	5.96
Banker	5.42	6.11	5.42	7.35	1.00	7.86
Bruteforce	4.27	4.32	4.27	4.35	4.24	7.63
Clicker	5.91	5.88	5.89	5.92	4.29	5.96
DDoS	4.31	4.31	4.31	4.31	4.31	4.31
Downloader	6.15	6.41	5.06	7.76	3.88	8.00
Keylogger	4.96	5.05	4.96	5.14	1.82	7.63
Miner	5.52	5.47	5.51	5.53	3.60	5.99
PUA/Adware	4.21	4.45	4.21	4.21	4.21	7.95
Ransomware	4.44	4.48	4.39	4.48	3.51	7.53
RAT	4.99	4.97	4.78	5.00	4.45	5.89
Spambot	7.10	6.84	6.77	7.16	3.85	8.00
Stealer	6.00	5.24	4.29	6.03	4.11	6.68
Trojan	5.81	5.39	4.36	5.82	1.00	7.99
UA problem	4.98	5.04	4.87	5.13	4.65	5.63

Note. The statistics were counted using all requests in the particular malware category, without being organized into request groups.

Table 16

The payload entropy statistics for HTTP requests (malicious traffic) in bits.

Browser	Median	Mean	1st quartile	3rd quartile	Min value	Max value
Edge Win10	4.84	4.74	4.62	4.99	2.73	5.72
Chrome Win7	4.81	4.71	4.42	5.10	1.00	5.93
Firefox-FP Win7	4.72	4.30	3.24	4.92	1.00	5.76
Firefox Win7	4.82	4.72	4.43	5.16	1.00	5.75
IE11 Win7	4.80	4.72	4.43	5.09	2.50	5.94
Chrome Win8.1	4.74	4.39	3.24	4.96	1.00	6.13
Firefox Win8.1	4.83	4.70	4.53	5.02	1.00	5.78
IE11 Win8.1	4.82	4.76	4.59	4.98	2.50	5.72

Table 17

The payload entropy statistics for HTTP requests (benign traffic) in bits.

The comparison of the obtained results demonstrates that many malware families achieved higher levels of the payload entropy. When comparing the median, mean, quartile, and maximum values, 11 out of 15 malware categories have higher mean and median values of the payload entropy than in the benign HTTP requests. The maximum value of the browser traffic payload entropy is 6.13 bits (in Chrome browser on Windows 8.1), whereas in the malicious traffic in 4 categories (i.e., *Downloader, PUA/Adware, Spambot*, and *Trojan*), the value achieved almost the highest possible value of 8 bits. The minimum values of entropy of 1.0 bits in both malware and browser traffic are caused by the requests with a very small payload size (1-2 bytes).

Figure 9 allows visual comparison of malware categories and browser traffic, and it presents the boxplot diagram of the corresponding payload entropy. Categories such as *Bruteforce, DDoS, Keylogger, RAT*, and *UA problem* almost overlap with the benign dataset entropy range when analyzing their interquartile range. For *Ransomware* and *PUA/Adware* categories, the median value is slightly smaller than for the benign traffic. Both categories however have many outliers. *Backdoor, Clicker*, and *Miner* categories have median values higher than those for the browser traffic. They also have outlying values in the interquartile range of benign traffic. *Stealer* and *Trojan* categories have higher median values than browser HTTP traffic, but still, the interquartile range partially overlaps with those of browser traffic. Categories such as *Banker, Downloader*, and *Spambot* visually achieve different distribution of values than other categories, and their median values are higher than those of browser traffic.

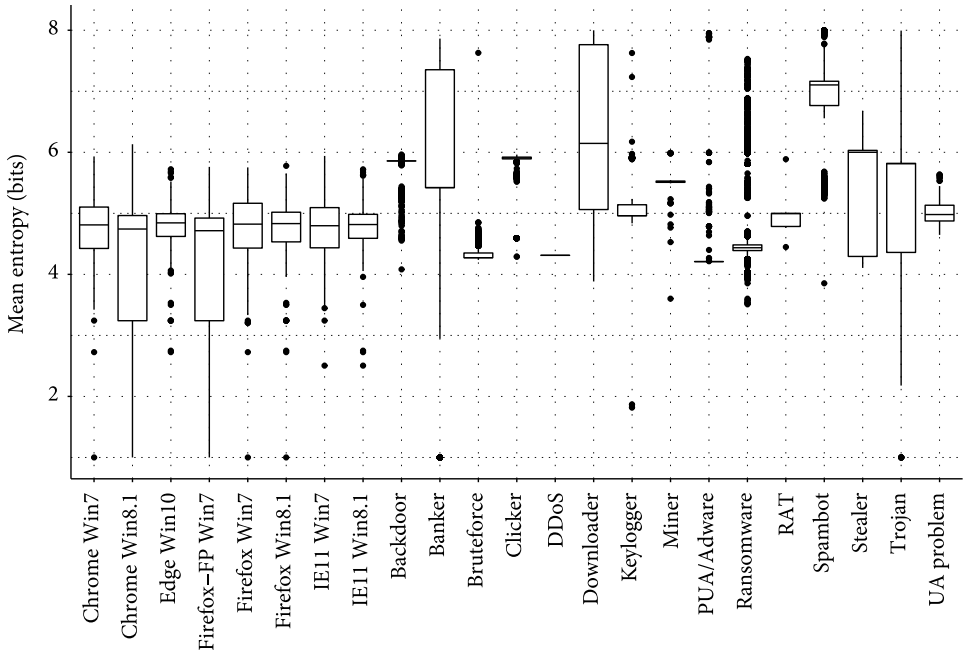


Figure 9

Boxplot diagrams of the payload entropy of malicious and benign HTTP traffic.

Overall, the obtained results prove that for many malware categories, the payload entropy is higher than for the browser HTTP traffic. From this perspective, the value of the payload entropy can be used as a feature to differentiate between the malicious and benign traffic. It should also be noted that the above conclusions are made in a general manner and that some particular malware families can exhibit different behaviors.

5.3.2. Non-ASCII Characters in Payload

Figure 10 shows the presence of non-ASCII characters in the payload of malicious HTTP traffic. For the *Banker, Downloader*, and *Spambot* categories, more than 50% of request groups contained non-ASCII characters in the payload data. Additionally, non-ASCII characters were present also in the *Bruteforce, Keylogger, PUA/Adware, Ransomware*, and *Trojan* categories, but in less than 40% of request groups.

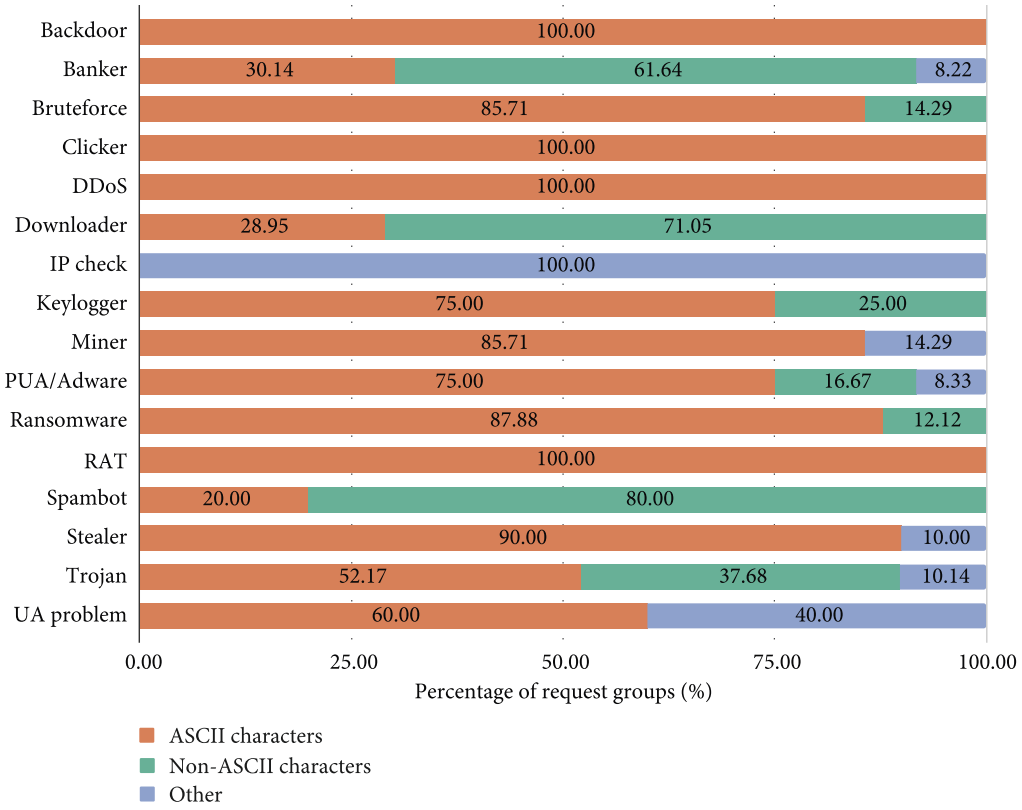


Figure 10

Non-ASCII characters in the payload (malicious traffic).

It must be noted that non-ASCII characters are rarely seen in the browser benign HTTP request payloads, but they are present in the traffic of all browsers. Only for the Firefox and Chrome browsers on Windows 7 OS, the numbers are higher than 1% (3.40% and 1.26%, respectively). After performing manual analysis of these requests, it turned out that these were either a part of a JSON with Chinese UTF encoded characters or a part of data sent to URL: http://sqm.microsoft.com/sqm/vstudio/sqmserver.dll.

To summarize, this feature should be considered useful for identifying malicious HTTP traffic.

5.3.3. Methods of Requests with Payload

Request methods with payload data in the malware traffic were also analyzed. The majority of malware categories used *POST* request to send data. However, 8 malware categories used requests other than *POST*. In case of *Clicker* and *Spambot*, *GET* requests contained payload in 25.00% and 46.67% of request groups, respectively. In the remaining 6 categories, mixed values were present—*GET* or *POST* requests could both be present in request groups. They were also mixed with requests without payload.

Analogous analysis has been performed on the benign browser traffic. It turned out that all requests used *POST* methods.

Therefore, the comparison of the results for both types of traffic leads to the conclusion that in some cases, the feature can be used to distinguish between these two traffic types. Note that RFC 7231 does not prohibit sending payload data in the *GET* requests; however, as it can be seen from the obtained results, browsers usually perform such operation using the *POST* method.

5.3.4. Presence of Referer Header in POST Requests

Finally, the presence of the *Referer* header in *POST* requests of malicious HTTP traffic has been investigated.

The obtained results show that most malware categories sent *POST* requests without the *Referer* header. However, in the *Ransomware* category, this header was present in almost 55% of request groups. Only in 2 categories (i.e., *Stealer* and *Bruteforce*) *Referer* was spotted in more than 10% request of groups (27.78% and 14.29%, respectively).

POST requests constitute a small fraction of all browser requests in the analyzed traffic, i.e., less than 1.5% depending on the browser. For 2 browsers, the *Referer* header is present in every *POST* request (Internet Explorer 11 and Firefox with Flash Player, both on Windows 7). In the other case, only at most in 3.17% of all *POST* requests, this header is not present.

Based on the comparison of the obtained results for malware and browser traffic, it can be concluded that the lack of presence of the *Referer* header in *POST* requests can be a promising feature to distinguish malicious and benign HTTP traffic.

5.4. Comparison of Results with the Related Work

In this section, the obtained results are compared with those reported in the previously discussed papers (see Section 2). In the remainder of this section, only sources which explored features of HTTP network traffic are taken into account, as these can be directly compared to this work.

In two sources (Rossow et al. in [11] and Nelson in [12]), a number of unique header names in malware traffic was observed. In the first source, it was 144 headers, and in the second one, it was 24, whereas in this analysis, it was 42. Also, as reported by Rossow et al. [11], in 98.6% of samples the *User-Agent* header was present. The below paper revealed that depending on the malware category, it was at least 57.78%, but in many cases, the percentage was closer to 80%. Rossow et al. observed the *Accept-Language* header in 44.3% of samples—in this analysis, this header was hardly present in requests at all. Nelson in [12] also analyzed the *Accept-Language* header along with *Content-Type*, considering values of these headers as helpful for identifying malware. In this paper, their values were not analyzed extensively, but it was observed that their presence (or lack) can be used as a distinctive feature to spot malware traffic.

Calzarossa et al. in [13] analyzed benign HTTP network traffic. The authors observed HTTP/1.0 version of protocol in 4% of requests, whereas in the below research, it was not observed at all or it was present in less than 0.1% of requests. Also, they reported about 60 unique header names with their number between 0 and 14 in request. In the below analysis, it was about 90 unique header names, with 0 to 24 headers in a single request. The most popular headers were similar in both analyses, apart the *From* header which was not present in the below dataset. It can be explained with the nature of analyzed network traffic. This header is popular in requests sent by robot HTTP clients, and such traffic was present in the discussed analysis. However, it was not present in the traffic analyzed here.

In Section 2, nonacademic sources were also reviewed. Montoro in his presentation [14] presented a set of request features, which can be used for malware detection. Some of them were also identified here as distinctive for malware network traffic. These include lack of popular requests, protocol version, and the number of headers in request. The author observed that malware sometimes does not include the *User-Agent* header; this was also observed in the below research. Additionally, he also identified that malware sometimes sends less than 4 headers, while benign applications send usually more than 9. In the below analysis, a similar dichotomy was observed; i.e., malware tends to send less headers in request than browsers. However, the numbers were different, as malware tends to send at most 6 headers and browsers between 7 and 9.

Analysis included in community modules of the Cuckoo sandbox system (https://cuckoosandbox.org/) was also explored in this work. The features of HTTP traffic analyzed by the *network_cnc_http* module were also identified here, proving their usefulness. Lack of the *Referer* header in malware in *POST* requests was observed as well as the lack of the *User-Agent* header in *POST* and *GET* requests. Also, it was identified that HTTP/1.0 version of the protocol is often seen in malware requests. Regarding the module's feature of the IP address in the *Host* header value, it was observed that values other than the domain name are hardly seen (less than 1% of requests) in the in the browser traffic. The IP address is not as a popular *Host* value as the domain in the malicious dataset; however, it is still more frequent than in the browser traffic. As the value of the *Host* header depends on the infrastructure of the attacker, it can be used as a potential indicator in malicious traffic identification.

Lewis presented in [17] observations about HTTP headers sent by malware. The below analysis has confirmed the findings of the author that malware sometimes uses nonstandard values of *User-Agent*. However, the below experiments did not find frequent typographic errors in the header names and values. For example, the only features regarding the whitespace character which give any results were when the space was present before CRLF (present in the network traffic of 5 malware categories, only 2 categories exceeding 10%) and when the space was present before a comma (2.35% of request groups in the *Ransomware* category). Additionally, the double space was not present in the malware traffic, but it was observed in the browser traffic (below 0.1% of requests).

The differences in results between the below analysis and the reviewed papers can be explained as follows. Firstly, the datasets analyzed in all sources and in this analysis are not uniform; that is, the uniformity of represented malware families and samples is not guaranteed. Secondly, the reviewed work is older than this analysis, and some malware families' behavior could already change, for example, to be able to further avoid detection. This shows that the analysis of malware network behavior should be performed regularly, especially when observed against the behavior of benign software. In this case, having a continuous monitoring can significantly increase the chance of detection of evolving threats.

6. Application of the Conducted Research

Until this point, HTTP protocol requests were analyzed in order to identify features, which could be helpful in distinguishing between malicious (malware) and benign (browser) HTTP traffic. In this section, the obtained results will be summarized to provide more practical and operational information and insights.

6.1. Practical Observations

In the previous analyses, the number of features indicated significant differences between the malware and browser traffic. These features are summarized in Table 18. Features marked with were proposed by the authors at the beginning of this paper as worth of analysis. Please note that the results for the destination port other than 80 are limited; however, they are analyzed against values of the *Host* header.

Name of the feature
HTTP/1.0 version of protocol
0–3 headers
High entropy of the payload
Lack of the *User-Agent* header
Nonstandard value of the *User-Agent* header
Non-ASCII characters in payload
Presence of POST request without the *Referer* header
Presence of GET request with payload
Host header value other than domain
Destination port other than 80
Lack of any of *Accept*, *Accept-Encoding*, *Accept-Language*, *Referer*, *Connection* headers

Features marked with (an asterisk) were proposed originally by the authors at the beginning of this paper.

Table 18

Features indicating significant differences between malware and browser traffic.

The values for the number of headers were chosen according to analyses conducted in Section 5.1.2 which showed that only in a small number of browser requests, less than 4 headers were present. Also, the analyses indicated that the boundary of 6 headers in a request can be chosen as a distinctive value for the majority of requests between malicious and benign HTTP traffic.

High payload entropy is defined as greater than 6.13 bits. This specific value was chosen as the maximum of the entropy value observed in the browser traffic (Section 5.3.1). Such a definition can also be supported from the practical perspective as in the popular network tool CyberChef (https://gchq.github.io/CyberChef/#recipe=Entropy()) in which authors state that English texts' entropy value lies usually between 3.50 and 5 bits. Also, an analysis of the Zeus botnet by Al-Bataineh and White [35] showed that the payload entropy was higher than 6.5 bits which is similar to the value proposed in this paper.

The features presented in Table 18 were further analyzed to determine how often their pairs co-occur. The results of this analysis provide some practical observations. They can be used as indicators in the manual malware analysis or treated as an entry point for further analyses. The most important observations from co-occurrence of features analysis in the malicious HTTP requests are that in the significant number of requests:(i)A low number of headers occurs with the lack of the *User-Agent* header(ii)Requests with the high entropy payload do not have a domain in the *Host* header value or the requests use the *POST* method without the *Referer* header(iii)When the request is sent to port other than 80, the *User-Agent* header value is different from the standard ones(iv)When the *GET* request has payload, it also has a low number of headers or the entropy of payload is high or the payload contains non-ASCII characters(v)The *POST* requests without the *Referer* header also have non-ASCII characters in payload(vi)Requests without the *Accept* header also lack *Accept-Encoding* or *Accept-Language* headers(vii)Requests without the *Connection* header also lack *Accept*, *Accept-Encoding*, or *Accept-Language* headers(viii)When the request is sent to the port other than 80, the *Host* header value is not a domain(ix)With *1.0* version of the protocol, *POST* requests do not contain the *Referer* header

6.2. Practical Usage Scenarios

HTTP request features presented in previous sections can be practically applied in multiple scenarios. The main ones are outlined below.

Firstly, some of the features, especially those presented in Section 6.1, can be used directly to identify suspicious requests. The term suspicious is used intentionally because the presence or lack of some of these features cannot be treated as an unambiguous indicator of the request maliciousness. Nevertheless, multiple usage scenarios can be presented. One of them is a manual inspection of the HTTP traffic during malware sample analysis. Also, the observations presented in this paper can be incorporated as rules in network security monitoring systems. Examples of such rules are presented in Figures 11 and 12. Such rules were used with success in the malware analysis laboratory of CERT Polska.

```
alert http any any -> any any \
(msg:"Suspicious POST request"; \
flow:established,to_server; content:"POST"; http_method; \
http_protocol; content:"HTTP/1.0"; \
http_header_names; content:!"Referer"; \
classtype:trojan-activity; sid:9000004;)
```

Figure 11

An example of the Suricata IDS rule based on presented observations. The rule detects *POST* requests in 1.0 version of the protocol without the *Referer* header.

```
@load policy/protocols/http/header-names.zeek

module HTTP;

export {
  redef enum Notice::Type += {
    Suspicious_HTTP_Request,
  };
}

event http_reply(c: connection, version: string,
                 code: count, reason: string){
  if ( |c$http$client_header_names| < 4 ){
    if (! c$http?$user_agent){
      NOTICE([$note=HTTP::Suspicious_HTTP_Request,
              $msg=fmt("Suspicious HTTP request"),
              $conn=c,
              $identifier=cat(c$id$resp_h)]);
    }
  }
}
```

Figure 12

An example of the Zeek network security monitoring analysis module based on presented observations. The rule detects requests with the number of headers less than 4 and without the *User-Agent* header.

Secondly, all presented features can be used to create an application fingerprinting system. Such a system can create a unique identifier by extracting and investigating particular features of the HTTP traffic. The identifier can be attributed to the particular application and afterwards used as a pattern to recognize such application's network traffic. Fingerprinting systems are used for some protocols, for example, for the TLS with the JA3 system (https://github.com/salesforce/ja3) or HTTP with p0f (http://lcamtuf.coredump.cx/p0f3/). The latter is not actively developed; however, it can be used as an inspiration. The HTTP request fingerprinting system can be used to identify particular malware families but potentially can also help to reveal information about the nature or the purpose of the inspected requests. For example, it can provide information whether request was a C&C server beacon or a connectivity check. Also, in the strictly controlled environments, a list of allowed application fingerprints can be used, and if a fingerprint previously not encountered is detected, the system can raise an alarm. Observations presented in this paper were used to create a prototype of the HTTP analysis and fingerprinting module for the Long-Term Sandboxing subsystem in the Horizon 2020 SISSDEN Project [36]. The system helped in the observation of malware behavior, for example, by providing a means for identification of malware operations such as connecting to the C&C server or connectivity checks.

Thirdly, the presented analyses identified HTTP request features which allow to distinguish malicious and benign HTTP traffic. Such features can be conveniently used to create a malware detection system which utilizes them to provide information about maliciousness of the HTTP requests. As a result, information on whether infected hosts are present in the monitored network can be provided.

7. Limitations of the Work

Even after carefully designed and performed analyses, some limitations of this research were identified. They are discussed below.

First of all, the presented generalized observations can be applied only to the analyzed malware samples. Other malware families or even other samples of the presented families can behave in a different way. The authors believe, however, that most of the identified HTTP request features can be generalized to other malicious software representatives as many of them capture differences inherent to

the general malware behavior.

Secondly, every work aimed at describing general behavior directly depends on the quality of analyzed data, especially in the representation and identification of actual malware behavior. Much effort was put in providing high-quality and relevant data. Nevertheless, the quality of the malware execution process in the sandbox systems cannot be guaranteed, as these were not designed and maintained by the authors of this paper. It must be noted that some malware families are able to detect that they are being analyzed in a virtual environment, what triggers their different behavior or even termination of operations [37, 38]. As such, it could alter the analysis results. Yet, checking the presence of such antianalysis techniques could be a broad task and hard to perform using only pcap files without the knowledge of the machine-level behavior. In such a situation, it was assumed that the network traffic alerted by IDS will represent actual behavior and that the lack of the traffic will result in termination of analysis of a particular malware sample.

Thirdly, the analysis environment has an impact on the obtained results. The authors of this paper used industry-proven IDS rule sets as the ground truth to conduct the process of malware request detection and identification of their family names. The authors believe that it is a high-quality source of information, but as with every detection system, it is possible that some HTTP requests were detected mistakenly or were not detected at all. The detection of such cases would have required additional detection systems, which would have, in turn, introduced additional complexity of the system and would be partially in conflict with the rule sets' licenses.

Fourthly, malware request grouping and categorization could potentially introduce bias of data. As discussed in Section 4.2, the authors introduced such an approach to limit the impact of different sizes of malicious request sets. As an alternative approach, reducing the number of requests for some malware families was considered. However, the authors did not want to miss too much information related to the malicious software behavior within omitted requests. Also, as the reduction of a number of requests was feasible, addition of requests to some families to equate the numbers was not. Thus, such a reduction would also introduce data bias. From this perspective, the authors believe that their approach was more adequate, despite some data bias residues.

Fifthly, identified features are based on observed behavior differences of malware and browser network traffic. In a situation when malware is equipped with mimicking mechanisms, that is, its behavior is deliberately changed to imitate a browser, the features will deteriorate with regard to distinction of malicious and benign traffic, depending on the changed feature. The identified features have different levels of technical difficulty to introduce mimicking behavior, some of them deeply connected with the C&C protocol design, as with the payload entropy higher than 6 bits. Changing such behavior would require, for example, rejection of payload obfuscation or using other protocol for data exchange. The authors of this paper believe that in the majority of examples, simple mimicry techniques would be used, such as the changing value of the *User-Agent* header or adding the additional header, which still could not counter all identified features. In the authors' opinion, in an extreme scenario of nearly perfect imitation of browser traffic, malware would still manifest some features of network traffic which would differentiate it from the web browser, as these two types of software are designed to perform different tasks. However, such scenarios are out of scope of this study.

Sixthly, in this paper the authors focused on analysis of differences between malware and browser traffic from the perspective of network monitoring in the sandbox environment. This perspective can be changed to a centralized one using logs from an actual proxy server as the data source, where data would contain more diverse sets of HTTP clients. Such a change could have impact on results of the analysis.

Finally, malware can use the HTTPS protocol for communication; however, this study did not analyze such cases. As mentioned in the introduction, the HTTP protocol is more popular than HTTPS when used by malware, thus the authors have focused solely on it. Nevertheless, if HTTPS malware traffic is decrypted to the HTTP protocol (for example, in a sandbox environment), it can be analyzed using the identified features. It must be noted that the presented analysis did not utilize such decrypted network traffic; however, the authors believe it should be similar to nonencrypted traffic. In a scenario when HTTPS traffic cannot be decrypted, the below findings cannot be applied, and analysts should refer to techniques based on this protocol.

8. Conclusion and Future Work

This paper focuses on presenting extensive and systematic analyses of the HTTP requests for malware- and browser-generated traffic. Its main aim was to establish the most promising distinctive features which can be used to identify malicious requests. Several features have been designed based on the previous works and own experience from malware behavior analysis. Datasets of malware and browser network traffic were analyzed using these features to identify which can be utilized to distinguish between malicious and benign HTTP traffic. The obtained results indicate which features can be generally used to spot anomalies understood as a deviation from the normal behavior. It was identified that these features include HTTP/1.0 protocol version, number of headers smaller than 4, the lack of *Accept*, *Accept-Encoding*, *Accept-Language*, *Connection*, and to some extent *Referer* headers, the payload entropy higher than 6 bits, the occurrence of non-ASCII characters in the payload, and the presence of the *Referer* header in the *POST* requests.

A special category of features are those connected to the *Host* and *User-Agent* headers. Because of their purpose, their values are changed frequently. *Host* header values other than domain names, such as IP addresses, are more often experienced in the malicious than in the benign HTTP traffic. However, these values are strictly connected to the network infrastructure used by criminals and as such should be used in a controlled manner. In case of the *User-Agent* header, its value presents an even more complicated matter. Certainly, the lack of this header should be treated as an anomaly, but an analysis of its values is a more demanding task. The results

in this paper indicate that many malware categories use well-known values, similar to those sent by the browsers. Nevertheless, a significant number of malicious software families use values which were not recognized as popular. Interestingly, many malware categories use predominantly one *User-Agent* header value.

Other analyzed features did not yield any significant results; i.e., these features were not observed in the traffic at all or were too scarce to be treated as deviations from the typical browser traffic. Some of these features could be seen as anomalies on their own, even without comparing them with those from the browser traffic because they break RFCs or standardization. Good examples are the lack of a colon in the header field, misspellings of the header name, requests sent to the TCP protocol port other than registered for HTTP, not ASCII printable character in the header value, a new line character other than CRLF, repetition of headers (applicable to the majority of them), or nonstandard whitespace characters in the header field (other than space or horizontal tabulator).

Results presented in this paper showed that nonacademic sources reviewed at the beginning of the analysis provide features which are helpful for distinction between malware and browser traffic. Only the category of typographic errors, presented by Lewis in [17], did not yield any results, as these errors were very rare or nonexistent in the analyzed datasets.

Some of the features need additional investigations. These include payload data length or the value of some popular headers such as *Accept*, *Accept-Encoding*, *Language*, and *Connection*. Analysis of their values could be correlated with the inspection of the *User-Agent* value in such way that the occurrence of the particular value in 1 header should imply a defined value in the other.

Additional issue is caused by the *GET* requests with payload data. They were not present in the analyzed browser dataset, but they were rarely seen in the malware dataset. RFC 7230 does not prohibit sending such requests, but experience tells to monitor them. Despite low-level occurrence of these requests, the authors consider them as an anomaly.

In order to search for suspicious requests, features and anomalies identified in the course of this analysis can be directly applied to the existing network monitoring systems, such as IDSs or malware sandboxes. Also, with the use of the presented results, it would be feasible to create a malware detection system. Such a system could detect new malware samples in which presented anomalies appear. Finally, this work can be utilized to create a fingerprinting system which can be used as an identification mechanism of similar malware requests or as a source to create a whitelist of known applications in the network. The authors plan to explore these directions in their future work.

Data Availability

A part of the pcap files used in this study origins from the Malware Capture Facility Project (MCFP) and is publicly available at https://www.stratosphereips.org/datasets-malware. PCAP files from CERT Polska's sandbox system and web browser traffic have not been made publicly available because of commercial confidentiality and privacy reasons.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this article.

Acknowledgments

This research was partially supported by the EU Horizon 2020 program towards the Internet of Radio-Light project (H2020-ICT 761992).

References

1. M. Trevisan, D. Giordano, I. Drago, M. Mellia, and M. Munafo, "Five years at the edge: watching Internet from the ISP network," in *Proceedings of the 14th International Conference on emerging Networking Experiments and Technologies*, pp. 1–12, ACM, Heraklion, Greece, December 2018. View at: Google Scholar
2. P. Richter, N. Chatzis, G. Smaragdakis, A. Feldmann, and W. Willinger, "Distilling the Internet's application mix from packet-sampled traffic," in *Passive and Active Measurement*, J. Mirkovic and Y. Liu, Eds., pp. 179–192, Springer International Publishing, Cham, Switzerland, 2015. View at: Google Scholar
3. S. Miller and P. Smith, *Rise of Legitimate Services for Backdoor Command and Control*, Anomali, Tech. Rep., 2017, https://www.anomali.com/files/anomali-labs-reports/legit-services.pdf.
4. S. Tkachenko, *Stop Windows 10 Spying on You Using Just Windows Firewall*, 2015, https://winaero.com/blog/stop-windows-10-spying-on-you-using-just-windows-firewall/.
5. Block Windows Update with Firewall, 2018, https://www.reddit.com/r/MoneroMining/comments/8l5wpt/block_windows_update_with_firewall/.
6. B. Duncan, *2019-03-15-Malspam Pushes Lokibot*, 2019, http://malware-traffic-analysis.net/2019/03/15/index2.html.
7. P. Srokosz, *Analysis of Emotet V4*, 2017, https://www.cert.pl/en/news/single/analysis-of-emotet-v4/.
8. E. Brumaghin and H. Unterbrink, *Picking Apart Remcos Botnet-In-A-Box*, 2018, https://blog.talosintelligence.com/2018/08/picking-apart-remcos.html.
9. R. Joven, *New Stealth Worker Campaign Creates a Multi-Platform Army of Brute Forcers*, 2019, https://www.fortinet.com/blog/threat-research/new-stealth-worker-campaign-creates-a-multi-platform-army-of-bru.html.

10. A.-T. GmbH, *AV-TEST Security Report 2018/2019*, AV-TEST Institute, Tech. Rep., 2019, https://www.av-test.org/fileadmin/pdf/security_report/AV-TEST_Security_Report_2018-2019.pdf.
11. C. Rossow, C. J. Dietrich, H. Bos et al., "Sandnet: network traffic analysis of malicious software," in *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, pp. 78–88, ACM, New York, NY, USA, 2011. View at: Google Scholar
12. A. Nelson, "Sandnet++-a framework for analysing and visualising network traffic from malware," *Information Security Group*, Royal Holloway University of London, Tech. Rep., 2016. View at: Google Scholar
13. M. C. Calzarossa and L. Massari, "Analysis of header usage patterns of HTTP request messages,," in *Proceedings of the 2014 IEEE International Conference on High Performance Computing and Communications, 2014 IEEE 6th International Symposium on Cyberspace Safety and Security, 2014 IEEE 11th International Conference on Embedded Software and Syst (HPCC, CSS, ICESS)*, pp. 847–853, IEEE, Paris, France, 2014. View at: Google Scholar
14. R. Montoro, "HTTP Header Hunter-Looking for malicious behavior into your HTTP header traffic," 2011, http://2011.video.sector.ca/video/39786962. View at: Google Scholar
15. "Cuckoo sandbox network CnC HTTP community module," 2019, https://github.com/cuckoosandbox/community/blob/master/modules/signatures/network/network_cnc_http.py. View at: Google Scholar
16. "Cuckoo sandbox multiple user-agents community module," https://github.com/cuckoosandbox/community/blob/master/modules/signatures/windows/multiple_ua.py. View at: Google Scholar
17. T. Lewis, "HTTP header heuristics for malware detection," Tech. Rep., SANS Institute InfoSec Reading Room, 2013, Tech. Rep. View at: Google Scholar
18. S. Mizuno, M. Hatada, T. Mori, and S. Goto, "Botdetector: a robust and scalable approach toward detecting malware-infected devices," in *Proceedings of the 2017 IEEE Int. Conference on Communications (ICC)*, pp. 1–7, IEEE, Paris, France, May 2017. View at: Google Scholar
19. Z. Li, L. Sun, Q. Yan, W. Srisa-an, and Z. Chen, "Droidclassifier: efficient adaptive mining of application-layer header for classifying android malware," in *Proceedings of the International Conference on Security and Privacy in Communication Systems*, pp. 597–616, Springer, Guangzhou, China, October 2016. View at: Google Scholar
20. N. Kheir, "Behavioral classification and detection of malware through HTTP user agent anomalies," *Journal of Information Security and Applications*, vol. 18, no. 1, pp. 2–13, 2013. View at: Publisher Site | Google Scholar
21. K. Li, R. Chen, L. Gu, C. Liu, and J. Yin, "A method based on statistical characteristics for detection malware requests in network traffic," in *Proceedings of the 2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)*, pp. 527–532, IEEE, Guangdong, China, June 2018. View at: Google Scholar
22. R. Perdisci, W. Lee, and N. Feamster, "Behavioral clustering of HTTP-based malware and signature generation using malicious network traces," in *Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation*, p. 14, San Jose, CA, USA, 2010. View at: Google Scholar
23. M. Mimura and H. Tanaka, "Leaving all proxy server logs to paragraph vector," *Journal of Information Processing*, vol. 26, pp. 804–812, 2018. View at: Publisher Site | Google Scholar
24. M. A. Nia, R. E. Atani, B. Fabian, and E. Babulak, "On detecting unidentified network traffic using pattern-based random walk," *Security and Communication Networks*, vol. 9, no. 16, pp. 3509–3526, 2016. View at: Publisher Site | Google Scholar
25. R. Fielding, J. Gettys, J. Mogul et al., *RFC 2616: Hypertext Transfer Protocol–HTTP/1.1*, 1999.
26. R. Fielding and J. Reschke, *RFC 7230: Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*, 2014.
27. RFC 7231: Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content, 2014.
28. RFC 7232: Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests, 2014.
29. R. Fielding, Y. Lafon, and J. Reschke, *RFC 7233: Hypertext Transfer Protocol (HTTP/1.1): Range Requests*, 2014.
30. R. Fielding, M. Nottingham, and J. Reschke, *RFC 7234: Hypertext Transfer Protocol (HTTP/1.1): Caching*, 2018.
31. R. Fielding and J. Reschke, *RFC 7235: Hypertext Transfer Protocol (HTTP/1.1): Authentication*, 2014.
32. T. Berners-Lee, R. Fielding, and H. Frystyk, *RFC 1945: Hypertext Transfer Protocol–HTTP/1.0*, 1996.
33. C. Linhart, A. Klein, R. Heled, and S. Orrin, *HTTP Request Smuggling*, Watchfire Corporation, Tech. Rep., 2005.
34. D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, *RFC 5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, 2008.
35. R. R. Scherberger, H. Kaess, and S. Brückner, "Studies on the action of an anticholinergic agent in combination with a tranquilizer on gastric juice secretion in man," in *Proceedings of the 2012 7th International Conference on Malicious and Unwanted Software*, pp. 1460–1463, IEEE, Washington, DC, USA, October 2012. View at: Google Scholar
36. SISSDEN, *Secure Information Sharing Sensor Delivery Event Network (SISSDEN). Deliverable D5.3: Final Data Analysis Results*, 2019, https://sissden.eu/download/SISSDEN-D5.3-Final_Data_Analysis_Results.pdf.
37. R. R. Branco, G. N. Barbosa, and P. D. Neto, *Scientific but not Academical Overview of Malware Anti-Debugging, Anti-Disassembly and Anti-VM Technologies*, 2012.
38. P. Chen, C. Huygens, L. Desmet, and W. Joosen, "Advanced or not? a comparative study of the use of anti-debugging and anti-VM techniques in generic and targeted malware," in *Proceedings of the IFIP International Information Security and Privacy Conference*, pp. 323–336, Springer, Ghent, Belgium, May 2016. View at: Google Scholar

Copyright