Sandnet: Network Traffic Analysis of Malicious Software

Christian Rossow[1,2], Christian J. Dietrich[1,3], Herbert Bos[2], Lorenzo Cavallaro[2],
Maarten van Steen[2], Felix C. Freiling[3], Norbert Pohlmann[1]

[1] Institute for Internet Security, University of Applied Sciences Gelsenkirchen, Germany
[2] Department of Computer Science, VU University Amsterdam, The Netherlands
[3] Department of Computer Science, University of Erlangen, Germany

ABSTRACT

Dynamic analysis of malware is widely used to obtain a better understanding of unknown software. While existing systems mainly focus on host-level activities of malware and limit the analysis period to a few minutes, we concentrate on the network behavior of malware over longer periods. We provide a comprehensive overview of typical malware network behavior by discussing the results that we obtained during the analysis of more than 100,000 malware samples. The resulting network behavior was dissected in our new analysis environment called *Sandnet* that complements existing systems by focusing on network traffic analysis. Our in-depth analysis of the two protocols that are most popular among malware authors, DNS and HTTP, helps to understand and characterize the usage of these prevalent protocols.

1. INTRODUCTION

Dynamic analysis, i.e. runtime monitoring, has proven to be a well-established and effective tool to understand the workings of yet unknown software [8, 14, 17]. Understanding the behavior of malicious software may not only provide insights about actions of malicious intents, upcoming techniques, and underground economy trends, but it also gives the opportunity to develop novel countermeasures specifically built on top of that understanding. Current analysis systems have specialized in monitoring system-level activities, e.g., manipulation of Windows registry keys and accesses to the file system, but little effort has generally been devoted to understanding the network behavior exposed by malware. In fact, similarly to system-level activities, network-level activities also show very distinct behaviors that can back up the insights provided by system-level analyses. Second, the very same network behaviors can uniquely provide further specific understanding necessary to develop novel approaches to collect, classify and eventually mitigate malicious software. Driven by this observation we focus our research on dissecting, analyzing, and understanding the behavior of malicious software as observed at the network level.

As we will show later, the observed malware behavior highly depends on the duration of the dynamic analysis. Current systems try to analyze as many malware samples as possible in a given period of time. This results in very short analysis periods, usually lasting only a few minutes, which makes it difficult to observe malicious network behavior that goes beyond the bootstrapping process. From a network behavior point of view, however, the post-bootstrap behavior is often more interesting than what happens in the first few minutes. A thorough analysis is key to understanding the highly dynamic workings of malware, which is frequently observed to be modular and often undergoes behavior updates in a pay-for-service model.

In this paper we present an in-depth analysis of malware network behavior that we gathered with a new system called *Sandnet* during the last 12 months. Sandnet [3] is an analysis environment for malware that complements existing systems by a highly detailed analysis of malicious network traffic. With Sandnet, we try to address two major limitations we see in publicly available dynamic analysis systems: a short analysis period and the lack of detailed network-behavior analysis. While existing systems have usually spent only a couple of minutes to run a malware sample, we ran each sample for at least one hour. In addition, using the data collected through Sandnet, we provide a comprehensive overview of network activities of current malware. We first present a general overview of network protocols used by malware, showing that DNS and notably HTTP are prevalent protocols used by the majority of malware samples. We will then provide an in-depth analysis of DNS and HTTP usage in malware. The results of our analysis [3] can be used to spawn new research such as clustering malware based on network-level features or network-level malware detection.

The main contributions of this work are:

- We have in operation a new data collection and analysis environment called Sandnet that will be up for the long run and that we will continuously use to gather information on malware network behavior.

- We give an overview of the network activities of more than 100,000 malware samples and compare the results with data from previous efforts.

- An in-depth analysis of DNS and HTTP traffic provides details on typical protocol-specific usage behaviors of malware, e.g. DNS fast-flux or click fraud.

BADGERS 2011 Workshop on Building Analysis Datasets and Gathering Experience Returns for Security, Salzburg, 2011

This paper is structured as follows. In Section 2, we will give an overview of Sandnet. Section 3 describes the dataset our analysis is based on. We will then provide a general malware network traffic overview in Section 4. In Section 5, we will provide a deep analysis on the usage of the DNS protocol by malware. Section 6 describes the usage of the HTTP protocol by malware. We will discuss related work in Section 7 and show future work in Section 8.

2. SYSTEM OVERVIEW

In Sandnet, malware is analyzed in execution environments known as *sandpuppets* consisting of (virtualized) hardware and a software stack. Currently, we use VMs with Windows XP SP3 based on VirtualBox as sandpuppets. The machines are infected immediately after booting and gracefully shut down after a configurable time interval, which is typically one hour. Each sandpuppet is configured to have a local IPv4 address and a NATed Internet connection. A local DNS resolver is preconfigured.

The *sandherder* is a Linux system hosting the sandpuppet virtual machines. Besides virtualization, the sandherder also records, controls and transparently proxies network traffic to the Internet. We limit the potential damage of running malware samples by transparently redirecting certain traffic (e.g. spam, infections) to local sinkholes or honeypots. In addition, we limit the number of concurrent connections as well as the network bandwidth and packet rate per sandpuppet to mitigate DoS activities. Internet connectivity parameters such as bandwidth and packet rate must be shared fairly among all sandpuppets in order to avoid inter-execution artifacts. The current Sandnet setup comprises five bot sandherders with four sandpuppets each, resulting in twenty sandpuppets dedicated to malware analysis. Herders and sandpuppets can easily be added due to a flexible and distributed design.

After executing a malware binary, we dissect the recorded network traffic for further analysis. A *flow-extractor* converts raw .pcap-files into UDP/TCP *flows*. A flow is a network stream identified by the usual 5-tuple (layer 4 protocol, source IP addr., destination IP addr., source port, destination port). For TCP, a flow corresponds to a reassembled TCP connection. For UDP, a flow is considered to be a stream of packets terminated by an inactivity period of 5 minutes. Our experience shows that this timeout length is a reasonable mechanism to compensate the lack of UDP flow termination frames. Additionally, we use payload-based protocol detection in order to determine the application-level protocol of a flow. We define a flow to be *empty*, if no UDP/TCP payload is transmitted in this flow.

Automated execution of malicious software raises some ethical concerns. Given unrestricted network connectivity, malware could potentially harm others on the Internet. Possible attack scenarios are, but not limited to, Denial-of-Service attacks, spam or infection of other hosts. We tried to find the right balance between ethical concerns when designing Sandnet and restrict the Internet connectivity. Technically, we integrated certain honeywall techniques. The harm of DoS attacks is limited by network level rate-limiting, spam is transparently redirected to local mail servers and protocols known to be used for infection are redirected to local honeypots. Sandnet is closely monitored during execution. Admittedly, it is technically impossible to completely prevent all possible attacks. However, we are convinced that within the bounds of possibility we implemented a huge part of mitigation techniques and that the value of Sandnet strongly outweighs the reasonably limited attack potential.

3. DATASET

In order to study malicious network traffic, we analyzed malware samples that were provided to a great degree by partner research institutions. For each sample we acquire A/V scan results from VirusTotal [1]. 85% of the samples that we executed had at least one scan result indicating malware (see Figure 1). In order to avoid accidental benign samples we collated our set of samples with a list of known software applications using Shadowserver's bintest [4]. We randomly chose the samples from a broad distribution of all malware families. We tried to mitigate side-effects of polymorphism by extracting the family name of a given malware sample's A/V labels and limit the number of analyses per malware family.

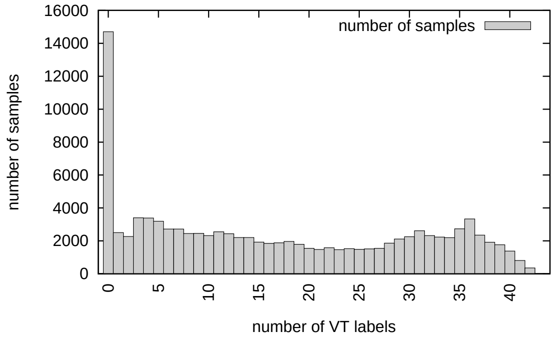

Figure 1: Histogram of VirusTotal Labels per Sample

For our analysis we defined the following set of samples. We analyzed a total of 104,345 distinct samples (in terms of MD5 hashes) over a timespan of one year. Samples were executed with regard to their age. On average, the samples were executed about 7.8 days after submission. We gradually increased the time between sample acquisition and execution from 6 hours to 150 days in order to evaluate whether the execution age significantly influences malware activity. Some statistics on the database of our data set is provided in annex F. The total analysis time of all samples in this data set sums up to an analysis period of 12 years.

4. NETWORK STATISTICS OVERVIEW

Of the 104,345 samples, the subset S_{Net} of 45,651 (43.8%) samples exhibited some kind of network activity. The network traffic caused by these samples sums up to more than 70 million flows and a volume of 207 GB. It remains an open issue to understand why a majority of the samples did not show any network activity. We suspect that most of the inactive samples a) are invalid PE files, b) operate on a local system only (e.g. disk encryption), c) are active only if there is user-activity or d) detected that they are being analyzed and stopped working.

Protocol inspection reveals that a typical sample in S_{Net} uses DNS (92.3%) and HTTP (58.6%). IRC is still quite popular: 8% of the samples exposed IRC. Interestingly, SMTP only occurred in 3.8% of the samples in S_{Net}. A complete list of the ISO/OSI layer-7 protocol distribution can be found in annex A. As DNS and HTTP are by far the most widely used protocols in Sandnet traffic, we will inspect these in more detail in Table 1. Table 1 also compares our protocol statistics with data based on Anubis provided by Bayer et al. [7] in 2009. Interestingly, when comparing the results, the samples we analyzed showed increased usage of all protocols. However, the ranking and the proportion of the protocols remain similar. We suspect this increase is a) due to a relatively long execution of malware samples and b) caused by a growing usage of different application-level protocols by malware.

Protocol	Reference	Sandnet
DNS	44.5	92.3
HTTP	37.6	58.6
IRC	2.3	8.0
SMTP	1.6	3.8

Table 1: Sandnet: Layer-7 protocol distribution compared with [7] (% of S_{Net})

30.1% of the flows were empty (no payload was transmitted). All these flows are presumable scanning attempts. Already 90% of the empty flows targeted NetBIOS/SMB services. The remaining empty flows are normally distributed over lots of different ports.

Of the remaining flows with payload (69.9%), for 22.8% no well-known protocol could be determined. Over 60% of these flows account for NetBIOS or SMB-related communication (mostly scanning) according to the destination port. Again, the remaining flows with failed protocol detection are normally distributed across many destination ports.

Payload-based protocol detection is a big advantage if protocols are used over other than their well-known ports. We found that 12.8% of S_{Net} use protocols over other than the well-known ports. We speculate that in these cases malware tries to communicate via ports opened in the firewall, independent from the actual communication protocol. For instance, we regularly found IRC bots connecting to IRC servers listening on TCP port 80. Thus, non-standard port usage might serve as a malware classification or detection feature. The top 3 affected protocols are listed in Table 2.

Protocol	S_{Net} Samples (%)	Distinct Ports
HTTP	8.17	303
IRC	7.13	174
Flash	0.91	9

Table 2: Top 3 protocols over non-standard ports

As additional analysis, we found out that a longer analysis period is indeed helpful for a better understanding of malware behavior. To judge on this, we performed three measurements each after an analysis period of 5 minutes and after 1 hour. First, we found out that only 23.6% of the communication endpoints that we have seen samples connecting to were contacted in the first 5 minutes of analysis. We then calculated that only a minor fraction (6.1%) of all flows started within the first 5 minutes. Lastly, we found that 4.8% of the samples started using a new protocol after 5 minutes that they have not used in the first minutes.

5. DNS

DNS is by far the most prevalent layer-7 protocol in Sandnet network traffic and gives an interesting insight into malware activity. The subset of samples using DNS is denoted by S_{DNS}.

5.1 DNS Resolution

Although all sandpuppets have their Windows stub resolver point to a working DNS resolver, we observed malware that used a different resolver or even carried its own iterative resolver. We developed the following heuristic in order to detect executions that carry an iterative resolver. An execution is considered as carrying an iterative resolver if there is an incoming DNS response from a server other than the preconfigured DNS resolver with a referral concerning a TLD (a resource record of type NS in the authority section) and the Recursion Available flag set to 0. We cross checked the resulting executions whether at least one of the DNS root servers had been contacted via DNS.

We can only speculate on the reasons why the preconfigured local DNS resolver is avoided. Using one's own resolver clearly has advantages. Resolution of certain domains might be blocked at the preconfigured resolvers in some environments (e.g. corporate ones). Additionally, using one's own resolver avoids leaving traces in logs or caches of the preconfigured resolver. If the Windows stub resolver is configured to use one's own resolver, local queries can be modified at will. This could be used for phishing attacks (redirect to a proxy) or to prevent A/V software from updating. Furthermore, preconfigured resolvers might be rate-limited.

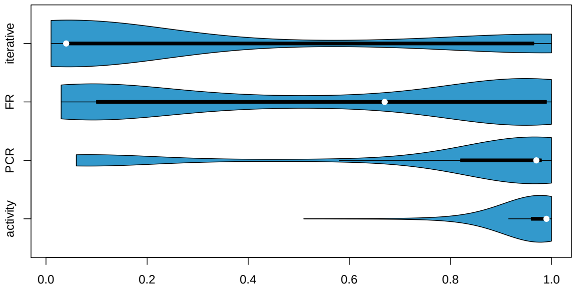

Figure 2: Violin plot of DNS activity end distribution

We found that 99% of the samples in S_{DNS} use the preconfigured resolver. Given this high ratio, a DNS resolver indeed turns out to be an interesting source for network-based malware detection - much more suitable than we had expected beforehand. We leave it up to future work to look into malware detection methods based on DNS resolver logs. 3% of S_{DNS} perform recursive DNS resolution with other resolvers than the preconfigured one (termed foreign resolvers in the following). Only 2% of S_{DNS} expose iterative DNS resolution. Note that the sets are not disjunct, as an execution may exhibit multiple resolution methods or resolvers. We speculate that this is due to the fact that malware occasionally downloads and executes multiple binaries, each of which might have different resolution methods. The foreign resolvers used include Google's Public DNS (used by 0.38%) as well as OpenDNS (0.25%). However, there is a large number of foreign resolvers that are used less frequently. One resolver that was located in China got our attention because queries for well-known popular domains such as facebook.com and youtube.com resolved into arbitrary IP addresses with no recognizable relation to the domain. We consider this to be an artifact of the so-called Great Firewall of China [10]. In total 932 of 5092 (18.3%) distinct DNS servers were used recursively at least once and thus can be regarded as publicly available recursive DNS resolvers.

Furthermore, we looked into the activity distribution of the different resolution methods (see Figure 2). The preconfigured resolver (PCR) was typically used throughout the whole analysis period. The end of the usage of foreign resolvers (FR) is wide-spread over time, leaning toward the end of the analysis. Interestingly, iterative resolution appears to end much sooner compared to the other resolution methods.

5.2 DNS TTL Analysis

The Time To Live parameter was of special interest to us, as it could be an indicator of fast flux usage. Fast flux is used as a means to provide flexibility among the C&C infrastructure of bots [12].

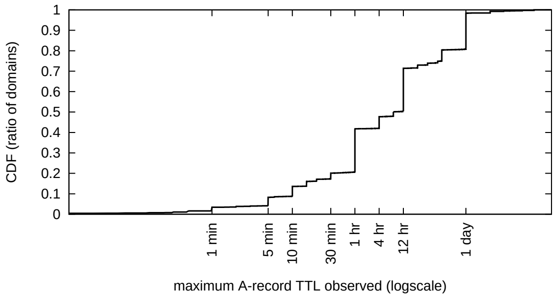

Figure 3: CDF of DNS TTL per domains

Figure 3 shows that 10% of all domains have a maximum TTL of 5 minutes or below. As spotted elsewhere [12], we expected domains with a small TTL and a large set of distinct answer records to be fast-flux candidates. However, when inspected manually, we found many domains of content distribution networks and large web sites. Using small TTLs seems to have become common among web hosters. As a result, the distinction between malicious fast-flux networks and legitimate hosting services becomes much more difficult. Interestingly, we also found a couple of responses with a TTL of zero that looked themselves like C&C communication. These responses were characterized by very long domain names as hex-strings. The TTL of zero prevents caching of these responses, effectively causing the resolver to always fetch the newest response from the authoritative DNS server. All in all, DNS suits well as a low-profile, low-bandwidth C&C channel in heavily firewalled environments, e.g. for targeted attacks.

5.3 DNS Message Error Rate

In order to measure DNS transaction failure, we defined the *DNS request error rate* as the number of DNS requests that were not successfully resolved over the total number of DNS requests. When aggregating the DNS message error rate per sample, we realized that for 10.1% of the samples in S_{Net} all of their DNS resolution attempts fail. However, the majority of the samples in S_{Net} (60.3%) have all DNS queries successfully resolved. The complete CDF is provided in Figure 4.

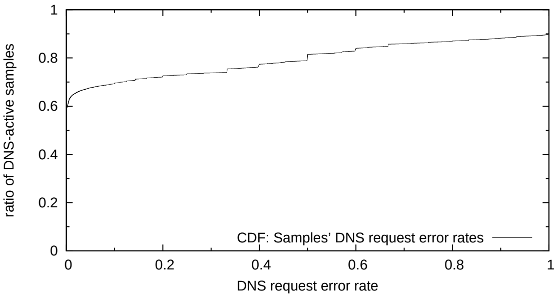

Figure 4: CDF of DNS message error rate

5.4 Resource Record Type Distribution

Figure 5 shows the distribution of the Resource Record types of the query section. Obviously, A records dominate DNS queries in Sandnet traffic, followed by queries for MX records. All samples in S_{DNS} have queried for an A record at least once. The high prevalence of A records is expected as A records are used to translate domain names into IP addresses. Furthermore, 2.3% of the samples in S_{DNS} queried blacklists. MX records have been queried by far less samples (8%). Interestingly, when comparing the MX query rate with SMTP activity, we have seen both: samples that performed MX lookups but had no SMTP activity and samples that successfully used SMTP but showed no MX queries at all. We assume that in the latter case, the required information on the MX destinations is provided via other means, e.g. C&C.

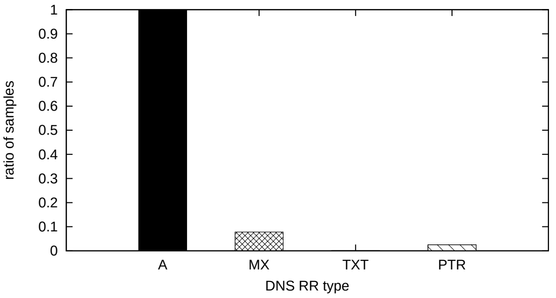

Figure 5: Resource Record distribution among samples

5.5 Resolution for Other Protocols

DNS, though itself a layer-7 protocol, plays a special role as it provides resolution service to all other layer-7 protocols. We analyzed how malware uses DNS before connecting to certain destinations. 23% of the samples in S_{DNS} show at least one flow without prior DNS resolution of the destination (DNS flows and scans excluded). In such a case either the destination's IP address is known (e.g. hard-coded in the binary) or resolution takes place via some other mechanism than DNS. A table providing flow destination DNS resolution by protocol can be found in annex I. Furthermore, 2.3% of the samples in S_{DNS} queried blacklists (26% of these also sent spam).

6. HTTP

HTTP traffic sums up to 88 GB inbound and 21 GB outbound, which makes HTTP by far the most prevalent protocol in Sandnet measured by traffic. The subset of samples using HTTP is denoted by S_{HTTP}. Given the high detail of the OpenDPI protocol classification, additional protocols that are carried in HTTP traffic are treated separately and thus contribute additional traffic: The Macromedia Flash protocol sums up to an additional 32 GB, video streams like MPEG and Apple Quicktime sum up to an additional 9 GB. We observed that the protocols carried in HTTP are usually caused by embedded objects included in websites that are visited by samples.

The immense potential abuse of HTTP-driven services motivated us to perform an in-depth analysis of typical malware HTTP traffic. Not only botnets started using HTTP as C&C structures. To name but a few, click fraud (i.e. the abuse of advertising services), mail address harvesting, drive-by downloads and DoS attacks on web servers are malicious activities of a wide range of malware authors. Of all samples with network activity (S_{Net}), the majority of 58.6% exposed HTTP activity. This section provides details to which extent, how, and why malware typically utilizes the HTTP protocol.

6.1 HTTP Requests

The analyzed samples typically act as HTTP clients and contact HTTP servers, mainly because the Sandnet communication is behind a NAT firewall. As a consequence, we can assume that virtually all recorded HTTP requests were made by malware. Figure 6 gives a general overview of how many HTTP requests malware typically made during the analysis period. The number of requests gives us a lead for which role malware has. Whereas one would expect a tremendous amount of requests during click fraud campaigns or DoS activities, malware update functionality and C&C channels potentially need little HTTP activity only. Interestingly, only 65% of the samples in S_{HTTP} made more than 5 HTTP requests. 16.3% of the samples in S_{HTTP} made only one HTTP request and then stopped their HTTP activity, although 70% of these samples continued with other network activity. We manually checked a fraction of these cases and found that many samples use HTTP to load second-stage binaries and continue with non-HTTP based damage functionality. The samples that completely ceased communicating after their initial HTTP flow presumably either failed to update themselves or waited for user-input triggers.

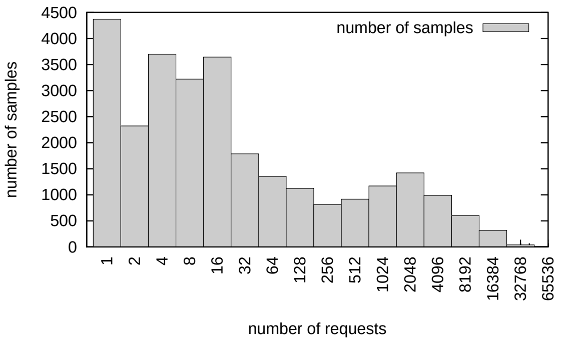

Figure 6: Histogram of HTTP Request Distribution

The GET request method was used by 89.5% of the samples in S_{HTTP}. We observed that 72% of the samples in S_{HTTP} additionally included GET parameters. Analysing just the fraction of GET requests with parameters, GET requests have on average 4.3 GET parameters. The average size of GET parameter were 12 characters for the key and 33.3 characters for the value. Although other means (such as steganography) allow to pass data to the sever, GET parameters seem to remain a popular method. On average, we have observed 1966 GET requests per sample with at least one request parameter. Interestingly, the number of unique GET parameter keys used by a sample is significantly lower than the total number of GET parameters per sample. This trend is particularly strong for samples with many parametrized GET requests and indicates that parameter keys are reused for follow-up requests. On average, the ratio between the number of distinct GET parameter keys and the total number of GET parameters is merely 1:16. We plan to further analyze the vast use of GET parameters, as started in [13], in the future.

The POST request method was used by 56.3% of the samples in S_{HTTP}. The average body size of POST requests is 739 bytes. We manually inspected a randomly chosen fraction of POST bodies to find out for what purpose malware uses POST requests. A large fraction of the inspected POST requests was used within C&C communication with a botnet server. We commonly observed that data passed to the server was base64-encoded and usually additionally obfuscated/encrypted. In addition, we frequently saw POST requests directed to search engines.

42% of the samples in S_{HTTP} used both POST and GET requests. Only 0.9% of the samples in S_{HTTP} showed HEAD requests at all. All other HTTP methods were used by less than 0.1% of the samples in S_{HTTP} and seem insignificant.

6.2 HTTP Request Headers

Annex C gives a comprehensive list of the 30 most popular HTTP request headers as observed in Sandnet. These HTTP headers include common headers usually used by benign web browsers. In total, we have observed 144 unique HTTP request headers. At a closer look at these, we identified a significant amount of misspelled or non-standard headers (excluding all extension headers, i.e. those starting with *'X-'*). Manual inspection shows that the fewer a specific header is used (in terms of samples), the more suspicious it is. Merely 5.7% of all samples in S_{HTTP} sent an HTTP request without any header at all. As a consequence, we see a need to further analyze specific request headers that we consider interesting.

6.2.1 User-Agent

In an ideal world, the HTTP *User-Agent* header specifies which exact web browser (including its version number) is requesting web content. However, the client and thus also malware samples can potentially forge the User-Agent header to be less suspicious. Annex B gives a detailed list of the 30 most popular raw User-Agent strings observed in Sandnet. Most samples (98.6% of S_{HTTP}) specified a User-Agent header at least once.

In an approach to get an overview of *actual* user agents we developed heuristics to filter the User-Agent list. First, we observed that 29.9% of the samples in S_{HTTP} specified wrong operating systems or Windows versions in their forged

HTTP User-Agent headers. Next, we identified that at least 13.4% of the samples in S_{HTTP} claim to use non-existing browser versions (e.g. *wget 3.0*, *Mozilla 6.0*). In addition, we saw that 37.8% of the samples in S_{HTTP} specified malformed or very short and to us unknown User-Agent values. In total, 67.5% of the samples in S_{HTTP} transmitted at least once a suspicious User-Agent string. Over the whole analysis period, only 31% of the samples in S_{HTTP} specified apparently correct User-Agent strings.

This result suggests that most samples have their own HTTP components that are bad in forging real web browsers. Interestingly, about half (50.6%) of the samples in S_{HTTP} change or alternate the User-Agent header during their analysis period. We hypothesize that this is due to the modular architecture of malware, where the modules have inconsistent User-Agent strings. Furthermore, based on this observation, we suspect that malware adapts the User-Agent header (and possibly other headers) depending on the target website.

6.2.2 Localization Headers

HTTP requests typically include headers that tell the server which languages and character sets the client understands (*Accept-Language* and *Accept-Charset*). We inspected these two localization headers and compared it with the locale setting (German) of the sandpuppets. While the Accept-Charset header was used by 0.35% of the samples in S_{HTTP}, the Accept-Language values are more interesting to analyze: In total, 44.3% of the samples in S_{HTTP} included Accept-Language as an HTTP request header. Of these samples, 24.1% did not respect the locale setting and specified a non-German language. Chinese (zh) and English (en) are the foreign languages specified most frequently, followed by Russian (ru). We speculate that in these cases malware authors forge HTTP headers either as observed at their own local systems or with respect to the target website. This would depict yet another indicator that malware carries its own (possibly self-made) HTTP implementation. Another reason could be that malware authors explicitly specify foreign languages to hoax web servers.

6.3 HTTP Responses

In Sandnet, all HTTP responses observed originated from HTTP servers on the Internet that were contacted by a sample. Therefore, the following analysis is not an analysis of the samples themselves, but may give indications to which type of servers malware communicates.

We observed that 97.8% of the HTTP requests were answered with an HTTP response. We define the *HTTP error rate* as the ratio between failed responses (HTTP status codes 4XX and 5XX) and all responses. Figure 7 shows a distribution of the sample-wise HTTP error rate. Only a small fraction (less than 10%) of samples virtually always get non-successful status-codes and apparently completely fail to retrieve the requested web content. Most samples have a relatively small error-ratio, indicating the web sites requested by the samples are still in place. We will give an overview of the requested servers in Section 6.6.

6.4 HTTP Response Headers

As opposed to HTTP request headers, response headers are set by servers and are not chosen by the malware samples. Analyzing the headers helps us to understand which servers are contacted by malware samples and gives information about the type of the retrieved content.

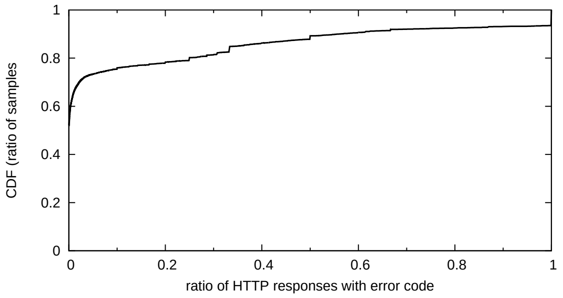

Figure 7: Distribution of HTTP error rates among samples

6.4.1 Content-Type

The Content-Type header shows which type of web content was retrieved by the samples. Figure 8 shows that most samples at least retrieve web sites with Content-Type *text/**. By far the most popular content-type of textual responses is *text/html*. However, only about half of all samples retrieved rich documents with Content-Type set to *images/** (48%) or *application/** (59.4%). 23.9% of the HTTP active samples with more than a single request got textual responses only. We see two reasons for such presumably light HTTP clients: First, spidering web sites without loading images is much more efficient. Second, we hypothesize that a considerable number of samples lacks a full-blown HTTP implementation that can recursively fetch objects embedded in web sites.

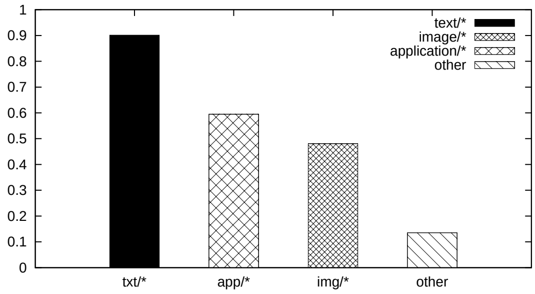

Figure 8: Ratio of samples using given Content-Type

6.4.2 Server

The *Server* HTTP response header indicates which type of web server is responding to the malware's HTTP request. Note that the content of this header can again be forged. Moreover, the majority of contacted web servers is presumably benign. However, when manually inspecting the HTTP Server response header, we spotted servers that presented suspicious banner strings. Annex E summarizes the list of the 30 most popular server types observed in Sandnet.

6.5 HTTP Responses with PE Binaries

After compromising a system with minimized exploits, attackers usually load so-called second-stage binaries. These binaries carry the actual malware functionality rather than just the exploit with minimized shell-code. In Sandnet, we usually analyze second-stage binaries instead of shell-code binaries. Yet, malware authors - as we will show - frequently load new portable executable (PE) binaries that expand or update the functionality of a malware sample. We assume this is due to a modular structure of the typical malware.

We extracted all binaries downloaded via HTTP by searching for the typical PE bytes in the body of HTTP responses. This straight-forward extraction of PE binaries already discovered that 16.7% of the samples in S_{Net} loaded additional PE files. To our surprise, we observed that 19% of these samples load binaries for multiple times - occasionally even more than 100 times. We verified that the five binaries downloaded most often were not corrupt and lack reasonable explanations why the binaries were downloaded that often. In total, we detected 42,295 PE headers, resulting in 17,676 unique PE files. The maximum size of a downloaded binary was 978 kB, the average size is 144 kB.

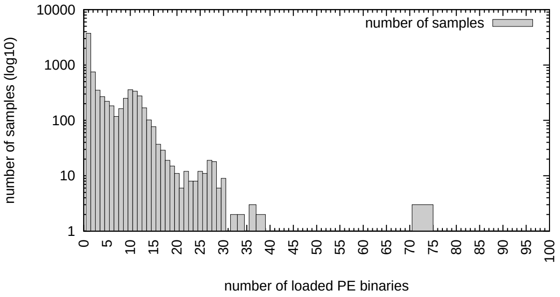

Figure 9: Distribution of # of PE binaries loaded

Figure 9 shows that most of the samples load more than a single PE binary. For readability of the graph we precluded 21 samples that loaded more than 100 and up to 1080 unique PE binaries. Annex D summarizes the Content-Type values of all HTTP responses that contain PE binaries. Most samples retrieve reasonable Content-Type values from the server. However, a significant number of servers tries to camouflage PE binary downloads as text, HTML, JavaScript or image files.

6.6 HTTP Servers

When recalling that HTTP is a protocol used by malware authors excessively, we see a need in analyzing which particular HTTP servers are visited by malware. We created a list of the 50 most popular domains ordered by the number of different samples visiting it in annex G. Obviously, many HTTP requests were put to presumably benign web sites. The next sections should briefly discuss why malware contacts these services.

6.6.1 Ad Services

We identified a significant number of ad service networks in the list of popular domains. Of the Top 50 domains in annex G, we manually identified 40 domains that are related to ads. Thousands of different malware samples use these services. A possible reason for this is that ads are included in virtually every web site and crawlers also follow the ads. However, after manually exploring the HTTP traffic of particular samples we assume that the reason for the popularity of ad services is vicious: click fraud. We leave it up to future work to analyze and mitigate the abuse of ad services by malware samples in greater detail.

6.6.2 Public Web APIs

Similarly to its popularity among benign users, Yahoo's and particularly Google's public Web APIs are present in Sandnet traffic, too. We suspect there are two reasons behind the popularity of these or similar services. First, some of these services are ubiquitous on the Internet. For example, a wide variety of web sites for example include Google Analytics to record statistics on the visitor behavior. Each time a sample visits such a web site and follows the embedded links, it will contact Google. As most of such services are open to anyone, we also suspect malicious usage of Google's and Yahoo's services by malware samples to be a reason for their popularity. A typical scenario that we observed was the abuse of search engines as a kind of C&C engine. In this case the malware searched for specific keywords and fetched the web sites suggested from the search results. Moreover, we have observed malware using the search engines to harvest new e-mail addresses for spamming campaigns. In general, benign human interaction with these services is particularly hard to be distinguished from abuse, especially from the client-perspective. We assume this is one of the main reasons malware authors use these HTTP-based services.

6.6.3 PE File Hosters

Based on the set of PE files that malware samples downloaded, we analyzed the file hosting servers. Annex H lists the most popular of all 1823 PE file hosters that we identified. 42.3% of the samples that downloaded PE files contacted the PE host directly without prior DNS resolution. This proves that still a significant number of malware samples include hard-coded IP addresses to download binaries. We further observed that a significant fraction of the URIs requested from file servers are non-static, although frequently only the parameters change. This observation may be important for blacklists trying to block entire URIs instead of IP addresses or domains.

6.6.4 HTTP C&C Servers

HTTP based botnets such as e.g. Torpig [15] switched from the classical IRC protocol to using HTTP. While manually inspecting Sandnet HTTP traffic, we occasionally encounter C&C traffic. What we see most is that samples include infection status information in their GET request parameters. Whereas some samples include clear-text status information, we and others [16] have observed many samples started encoding and encrypting the data exchanged with the server. However, we found it difficult to automatically spot C&C servers without knowing the command syntax of specific botnets. The big difference to IRC is that HTTP is a prevalent protocol on clean, non-infected systems and is thus harder to spot in the volume of HTTP data. Encouraged by the results reported in [9, 13], we believe that clustering the network behaviors of malware may help us in spotting generic C&C communication channels.

7. RELATED WORK

The malware phenomenon has been considerably studied over the last years by researchers and security practitioners. The community has proposed numerous techniques to collect [5], analyze [2, 8, 9, 11, 13, 17], or detect malware [9, 13].

For instance, Perdisci et al. [13] present an interesting system to cluster network-level behavior of malware by focusing on similarities among malicious HTTP traffic traces.

Similarly, Cavallaro et al. [9] present cluster-based analyses aimed at inferring interesting payload-agnostic network behaviors of malicious software. While Sandnet is currently limited to analyzing a large corpus of network protocols, it is clear how the adoption of similar cluster-level analyses can provide better understandings of the network behaviors of unknown software.

Anubis [7, 8] and CWSandbox [17] are probably the closest work related to our research. Although they both provide interesting—but basic—network statistics, their main goal is to provide insights about the host behaviors of unknown—potentially malicious—software. In this context, Sandnet complements Anubis and CWSandbox important results by providing an in-depth analysis of the network behaviors of the analyzed samples. As described elsewhere, this is only the first step toward a comprehensive understanding of the network activities perpetrated by such software. More analyses are currently being examined (e.g., [9, 13]) and are planned to extend Sandnet as part of our future research.

8. CONCLUSION AND FUTURE WORK

In this work, we presented a comprehensive overview of network traffic as observed by typical malware samples. The data was derived by analyzing more than 100k malware samples in Sandnet. Our in-depth analysis of DNS and HTTP traffic has shown novel malware trends and led to numerous inspirations to combat malware. The provided data is not only of great value because it is that detailed. It also perfectly complements related work that is either outdated, analyzes particular malware families only, or focuses mainly on the host behavior of malware. To share these insights with the research community, Sandnet is accessible via `http://www.if-is.net/sandnet/`.

We are currently expanding Sandnet to mitigate some of its current limitations and to perform a number of more detailed and sophisticated analyses, which will provide more insights into the behaviors of the network activities perpetrated by unknown, potentially malicious, software. For instance, clustering the network behavior of malware may automatically filter out uninteresting actions while unveiling the core patterns that represent the most interesting behavior of the malware [9, 13]. Furthermore, cross-correlation of network- and host-level clusters [6] may disclose interesting relationships among malware families. We also plan to assign public IP addresses to sandpuppets and to compare the malware behavior with a restricted, NATed network breakout. Similarly, we plan to integrate the analysis of system-level activities to Sandnet, such as linking process information to network activity. Including observations on how processes react to varying network input could further help to identify C&C channels. In addition, we strive to a more accurate view on the analysis data, particularly to distinguish benign from malicious communication endpoints.

Another direction our research may suggest is tailored toward performing a more detailed analysis of ad service abuse, especially click fraud. We plan on exploring click fraud detection mechanisms derived from the web site request behavior of malware observed in Sandnet. Possibly, we will expand this idea by also inspecting the abuse of public web services (e.g. the Google API).

We believe the data collected and analyzed by Sandnet represents a first step toward a comprehensive characterization of the network behaviors of malware. Driven by recent results, we thus hope our ongoing research to be of a great value to researchers and practitioners to help them acquiring a more detailed understanding of such behaviors. Not only this enables the development of more effective countermeasures and mitigation techniques, but it may also help to understand the social trends and facts of the underground malware economy.

9. ACKNOWLEDGEMENTS

Malware samples are to a great degree provided by partner research institutions. We thankfully acknowledge their generosity to let us use the samples.

10. REFERENCES

[1] Hispasec Sistemas - VirusTotal. `http://www.virustotal.com/`.

[2] Norman Sandbox. `http://www.norman.com/security_center/security_tools/`.

[3] Sandnet. `http://www.if-is.net/sandnet/`.

[4] The Shadowserver Foundation - bin-test. `http://bin-test.shadowserver.org/`.

[5] P. Baecher, M. Koetter, M. Dornseif, and F. Freiling. The Nepenthes Platform: An Efficient Approach to Collect Malware. In *RAID 2006*.

[6] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Kruegel, E. Kirda, and S. Barbara. Scalable, Behavior-Based Malware Clustering. In *NDSS 2009*.

[7] U. Bayer, I. Habibi, D. Balzarotti, E. Kirda, and C. Kruegel. A View on Current Malware Behaviors. In *USENIX LEET 2009*.

[8] U. Bayer, C. Kruegel, and E. Kirda. TTAnalyze: A Tool for Analyzing Malware. In *EICAR 2006*.

[9] L. Cavallaro, C. Kruegel, and G. Vigna. Mining the Network Behavior of Bots, Technical Report, 2009.

[10] R. Clayton, S. J. Murdoch, and R. N. M. Watson. Ignoring the Great Firewall of China. In *Privacy Enhancing Technologies*, pages 20–35, 2006.

[11] J. Goebel and T. Holz. Rishi: Identify Bot Contaminated Hosts by IRC Nickname Evaluation. In *USENIX HotBots '07*.

[12] T. Holz, C. Gorecki, K. Rieck, and F. C. Freiling. Measuring and detecting fast-flux service networks. In *NDSS*, 2008.

[13] R. Perdisci, W. Lee, and N. Feamster. Behavioral Clustering of HTTP-Based Malware and Signature Generation Using Malicious Network Traces. In *NSDI 2010*.

[14] K. Rieck, P. Trinius, C. Willems, and T. Holz. Automatic Analysis of Malware Behavior using Machine Learning. In *Journal of Computer Security 2011*.

[15] B. Stone-Gross, M. Cova, B. Gilbert, L. Cavallaro, M. Szydlowski, C. Kruegel, G. Vigna, and R. Kemmerer. Your Botnet is My Botnet: Analysis of a Botnet Takeover. In *CCS 2009*, Chicago, IL, November 2009.

[16] M. van den Berg. A Taste of HTTP Botnets, 2008.

[17] C. Willems, T. Holz, and F. Freiling. Toward Automated Dynamic Malware Analysis Using CWSandbox. In *IEEE Security and Privacy Magazine*, volume 5, pages 32–39, March 2007.

APPENDIX

A. GENERAL TRAFFIC OVERVIEW

L7 Protocol	Samples	Flows	Bytes	Out Bytes	In Bytes	Destinations	Dst Domains
DNS	42143	11845193	3730 MB	1355 MB	2375 MB	241126	14732
HTTP	26738	13492189	110 GB	21 GB	88 GB	36921	55032
Unknown	18349	32265514	24 GB	14 GB	10 GB	9145625	86523
Flash	5881	299986	32 GB	692 MB	31 GB	2955	2205
SSL	5104	79344	1884 MB	139 MB	1745 MB	2278	1622
SMB	4275	8602414	6116 MB	4210 MB	1906 MB	7253975	10
IRC	3657	169833	70 MB	15 MB	55 MB	564	554
SMTP	1715	3155014	20 GB	19 GB	1124 MB	282401	118959
MPEG	1162	2200	220 MB	1050 kB	219 MB	58	44
SSDP	885	1861	3651 kB	3651 kB	0 bytes	2	0
Quicktime	389	1222	8315 MB	1518 kB	8313 MB	62	41
FTP	243	7523	3144 kB	860 kB	2285 kB	159	121
NetBIOS	184	134600	54 MB	36 MB	18 MB	108909	0
TDS	163	1086	31 MB	1044 kB	30 MB	44	36
NTP	102	2950	266 kB	156 kB	109 kB	13	5
STUN	68	276	71 kB	54 kB	18 kB	19	8
TFTP	48	12492	626 MB	5165 kB	621 MB	19	0
PPLIVE	37	1481	85 MB	9042 kB	76 MB	1321	0
Gnutella	32	20545	181 MB	102 MB	79 MB	15640	0
DDL	28	277	29 MB	140 kB	29 MB	52	35
Bittorrent	26	1180	147 MB	5090 kB	142 MB	588	32
Mysql	21	33	38 kB	4288 bytes	34 kB	12	7

B. HTTP USER AGENTS

User Agent	Requests	Samples
Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Trident/4.0; .NET CLR 2.0.507	17193201	11168
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)	861353	5628
Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; .NET CLR 2.0.507	1937020	5376
Microsoft-CryptoAPI/5.131.2600.5512	17581	3485
Mozilla/6.0 (Windows; wget 3.0)	12851	3242
Download	5022	2042
Mozilla/4.0 (compatible; MSIE 8.0.6001.18702; Windows NT 5.1.2600)	23022	1802
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)	12569	1546
ClickAdsByIE 0.7.3	34615	1208
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)	69078	992
XML	3403	891
Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; SV1)	1714	849
PinballCorp-BSAI/VER_STR_COMMA	3454	771
Mozilla/3.0 (compatible; Indy Library)	71971	761
Microsoft Internet Explorer	8652	750
gbot/2.3	22791	694
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET CLR 1.1.4322)	23772	608
_	5327	589
NSISDL/1.2 (Mozilla)	692	535
Microsoft-ATL-Native/9.00	3827	524
Mozilla/5.0 (Windows; U; Windows NT 5.1; de; rv:1.9.2.6) Gecko/20100625 Firefox/	31078	514
Mozilla/4.0 (compatible)	6004	487
Mozilla/4.0 (compatible; MSIE 8.0; 10.1.53.64; Windows NT 5.1)	884	426
NSIS_Inetc (Mozilla)	515	403
wget 3.0	3917	339
Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR 1.1.4322)	946	311
opera	946	300
Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2.3) Gecko/20100401 Firef	6764	300

C. HTTP REQUEST HEADERS

HTTP Header	Samples	HTTP Requests
Host	27771	21054208
User-Agent	26359	20923840
Connection	21205	20570434
Cache-Control	18529	1346260
Accept	18483	20554040
Content-Length	14811	977547
Accept-Encoding	14406	19424065
Content-Type	14135	1033111
Accept-Language	11382	18319897
Referer	10079	18311670
Cookie	10075	10939127
If-Modified-Since	5462	3044837
If-None-Match	4696	1005364
x-flash-version	4386	464334
Pragma	4290	73427
x-requested-with	2079	14329
Range	1597	15451
If-Range	1006	3882
Unless-Modified-Since	962	3868
Accept-Charset	922	69908
X-Agent	658	36302
Keep-Alive	642	87149
X-Moz	511	517
Content-length	438	1494
x-prototype-version	408	1831
http	287	13984
UA-CPU	208	12899
x-svn-rev	111	351
x-type	84	167
Content-type	81	4876

D. CONTENT-TYPE OF PE DOWNLOADS

Content-Type	# Binaries	# Samples
application/octet-stream	6468	5908
text/plain	356	1716
application/x-msdownload	732	1082
application/x-msdos-program	550	786
image/gif	177	402
image/jpeg	390	365
text/plain; charset=UTF-8	166	344
text/html	776	326
application/x-javascript	190	78
image/png	68	55

E. HTTP SERVER TYPES

Server	Ratio (%)	Servers
Apache	68.4	326237
Microsoft-IIS	49.4	102652
nginx	40.9	108104
Golfe	21.4	20534
lighttpd	21.4	32934
YTS	20.0	28320
sffe	19.4	15128
GFE	18.3	21089
Apache-Coyote	17.6	41875
QS	15.4	6906
PWS	14.7	16297
DCLK-AdSvr	13.9	6782
cafe	13.7	11399
AmazonS3	13.7	17203
ADITIONSERVER 1.0	10.9	6092
AkamaiGHost	10.3	3520
Cookie Matcher	10.1	4011
gws	9.5	6075
VM_BANNERSERVER 1.0	9.4	2620
CS	9.1	3987
Adtech Adserver	8.9	4842
CacheFlyServe v26b	8.0	2242
RSI	7.8	2196
yesup httpd 89	7.8	2160
yesup httpd 103	7.7	2151
Resin	7.7	4375
ECS (fra	6.7	7615
Oversee Turing v1.0.0	6.1	2579
JBird	6.0	1987
TRP Apache-Coyote	5.7	1966

F. DATABASE STATISTICS

Attribute	Value
Distinct samples	104,345
Total traffic	207 GB
Outbound traffic	61 GB
Inbound traffic	146 GB
Number of Flows	70,106,728

G. HTTP SERVERS

HTTP domain	# Samples
www.google-analytics.com	5286
ad.yieldmanager.com	5046
cookex.amp.yahoo.com	4716
content.yieldmanager.com	4655
ak1.abmr.net	4288
pixel.quantserve.com	4050
content.yieldmanager.edgesuite.net	4009
edge.quantserve.com	3957
ad.doubleclick.net	3677
ad.harrenmedianetwork.com	3470
ad.103092804.com	3458
s0.2mdn.net	3370
ib.adnxs.com	3280
pixer.meaningtool.com	3219
ad-emea.doubleclick.net	2972
www.google.com	2940
ad.harrenmedia.com	2920
www.mupimg.de	2823
imagesrv.adition.com	2770
www.mupads.de	2759
view.atdmt.com	2754
ad.xtendmedia.com	2726
cm.g.doubleclick.net	2669
googleads.g.doubleclick.net	2657
fpdownload2.macromedia.com	2619
www.myroitracking.com	2573
serw.clicksor.com	2489
ad.adition.net	2468
ads.clicksor.com	2466
ad.tlvmedia.com	2449
ad.adserverplus.com	2414
b.scorecardresearch.com	2376
pub.clicksor.net	2375
ajax.googleapis.com	2335
img.billiger.de	2308
tags.bluekai.com	2308
adx.adnxs.com	2287
adfarm1.adition.com	2286
admax.quisma.com	2201
pagead2.googlesyndication.com	2199
a.collective-media.net	2115
ads.revsci.net	2099
pix04.revsci.net	2065
js.revsci.net	2050
ad.globe7.com	2040
staging.pixer.meaningtool.com	2030
ad.reduxmedia.com	2028
suresafe1.adsovo.com	2012
adserver.adtech.de	2010
crl.verisign.com	1999

H. PE FILE HOSTERS

PE File Server	#S	#B
64.79.86.26	775	1340
66.96.221.102	681	1063
ku1.installstorm.com	487	944
origin-ics.hotbar.com	483	483
64.191.44.9	480	727
img.ub8.net	458	460
pic.iwillhavesexygirls.com	437	478
64.120.232.147	431	747
origin-ics.clickpotato.tv	390	390
p2pshares.org	389	391
sky.installstorm.com	363	363
208.43.146.98	331	531
file0129.iwillhavesexygirls.com	323	853
173.45.70.226	315	437
173.45.70.227	313	444
dl.ghura.pl	302	302
122.224.6.48	300	553

#S = number of samples contacting file hoster
#B = number of binaries downloaded

I. DNS RESOLUTION BY PROTOCOL

Protocol	Samples (%)
NetBIOS	0.00
MSN	0.00
SMB	0.00
SIP	0.00
DHCP	0.00
TFTP	0.00
Gnutella	0.00
STUN	0.00
SSDP	0.00
mDNS	0.00
Bittorrent	19.23
TDS	39.88
SMTP	41.05
Unknown	46.84
FTP	47.74
DDL	53.57
Oscar	57.89
Mysql	66.67
HTTP	67.72
IRC	80.45
NTP	82.35
POP	83.33
Flash	83.63
SSL	87.93
Quicktime	93.57
MPEG	96.04