Detecting DGA malware using NetFlow

Martin Grill*†, Ivan Nikolaev*†, Veronica Valeros†, Martin Rehak*†
†Cisco Systems, Inc.
magrill@cisco.com, inikolae@cisco.com, vvaleros@cisco.com, marrehak@cisco.com
* Faculty of Electrical Engineering, Czech Technical University in Prague
grillmar@fel.cvut.cz, nikoliva@fel.cvut.cz, rehakmar@fel.cvut.cz

***Abstract*—Botnet detection systems struggle with performance and privacy issues when analyzing data from large-scale networks. Deep packet inspection, reverse engineering, clustering and other time consuming approaches are unfeasible for large-scale networks. Therefore, many researchers focus on fast and simple botnet detection methods that use as little information as possible to avoid privacy violations. We present a novel technique for detecting malware using Domain Generation Algorithms (DGA), that is able to evaluate data from large scale networks without reverse engineering a binary or performing Non-Existent Domain (NXDomain) inspection. We propose to use a statistical approach and model the ratio of DNS requests and visited IPs for every host in the local network and label the deviations from this model as DGA-performing malware. We expect the malware to try to resolve more domains during a small time interval without a corresponding amount of newly visited IPs. For this we need only the NetFlow/IPFIX statistics collected from the network of interest. These can be generated by almost any modern router. We show that by using this approach we are able to identify DGA-based malware with zero to very few false positives. Because of the simplicity of our approach we can inspect data from very large networks with minimal computational costs.**

I. Introduction

Botnets are one of the the main attack vectors on the internet today. They are the root cause of many malicious activities in computer networks, such as denial of service, spam distribution, click fraud, adware, distributed brute-forcing of remote services, identity and data theft and many more. A typical botnet consists of a number of malware-compromised machines, called bots, that are remotely controlled by a botmaster using a command and control (C&C) channel. Exploitation of a machine starts with a malware infection from a malicious web page, email attachment, etc. As soon as the malware infects a host, it usually tries to establish a connection to one or more C&C servers to download updates and retrieve commands or send private information gained from the host. There are two main types of botnet structures [9]: peer-to-peer (P2P) and centralized.

In the P2P [17], [20] structure every node can serve as C&C server distributing commands and updates in P2P manner. This makes the botnet more robust and resilient, hard to identify and to take down. This approach is less popular because it is very hard to implement and maintain. Commands take a longer time to reach all the bots because of the latency introduced by the distributed botnet topology. Finally, each newly infected host has to be provided with a list of bots to which it may connect.

The centralized structure [9] is the most popular, due its simplicity. In this scenario, the bots contact one predefined domain or IP address on which the C&C server is located. The disadvantage of this approach is that the C&C server represents a single point of failure. When taken down, the botmaster loses control over the whole botnet. Network administrators use blacklists of well-known C&C domains to block the communication at the firewall level. Furthermore, Anti-virus companies and OS vendors are working hard to take down such C&C servers and are successful doing so.

To overcome this disadvantage of the centralized structure, modern malware uses various techniques to hide its C&C server. One of these techniques is fast-flux [10], in which the C&C server is hidden behind a number of proxies that are associated with one domain name and the IP addresses are swapped in and out with extremely high frequency using domain name server (DNS) changes. This way the bots communicate with the C&C using a number of ever changing proxies.

Similarly, malware can use a *domain generation algorithm* (DGA), also referred to as *domain fluxing*. In this scenario, the malware contacts a domain that was generated using a domain generation algorithm with a specific seed in specific time intervals. Whenever the botmaster wants to send a command to his botnet, he needs to register a new domain that he generated using his own copy of DGA with the same seed as the botnet just before the botnet will try to contact it. Botmasters are trying to expose their C&C servers for the minimum amount of time. Domains are registered and DNS configurations are made just a few minutes before the infected bot is supposed to query the domain, and the C&C servers are shut down and removed immediately afterwards, so the whole process takes less than an hour. This renders the detection mechanisms that rely solely on a static domain list ineffective.

The DGA can be a simple algorithm that uses a seed and the current date and/or time to generate alphanumeric combinations for a new domain. More sophisticated DGAs (i.e. Kraken botnet [2]) can create English-language-like domains with properly matched syllables or even more advanced DGA can use combinations of English dictionary words, which makes them undetectable by the means of domain names analysis.

When such a malware is found, it has to be reverse engineered to uncover the underlying domain generation algorithm in order to block all the generated domains on a firewall or register them before the botmaster does. This task can be time-consuming and needs advanced reverse engineering skills. Furthermore, attackers can make this even more difficult by altering the technique in a way that the DGA seed is based on the responses of popular sites like *google.com*, *baidu.com*,

978-3-901882-76-0 @2015 IFIP

answers.com (Conficker-C [14]) or even trending topics on twitter (Torpig botnet [19]) that cannot be known in advance, rendering the filtering approach unusable.

In this paper we present a new privacy-preserving technique to detect hosts infected by DGA-malware in a large scale networks using only NetFlow [4], [7] information. One NetFlow is defined as an aggregation of all packets sent from one source IP and port pair to one destination IP and port pair, over the same protocol. Using only simple statistics about the network communication, we are able to detect DGA-based malware independent of the specific DGA technique used by the malware. Our approach is based on the fact that the malware will try to resolve many domains during a small time interval without a corresponding amount of newly visited IPs. Large numbers of domain trials are expected because they lower the chance of generating already existing or blocked domains. We show that using this method we are able to identify popular bots in real network traffic collected from university and large company networks.

This paper is organized as follows. In Section II we discuss the state-of-the-art detection methods of the DGA-based malware. The main idea of the detection technique is presented in Section III. Identification of requests and responses together with service detection needed for precise identification of the number of requests each host in the network is presented in Sections III-A and III-B. In Section III-C we formally define the anomaly detection algorithm. Finally, the presented method is evaluated in Section IV.

II. Related Work

The first report on DGA was made by Stone-Gross et. al. [8] in 2005. By reverse engineering a Torpig malware, they found that it periodically generated a list of domains and tried to contact them.

Antonakakis et. al., [3] presented a method that leverages the idea that bots from the same botnet will generate similar non-existent domains (NXDomain) [5]. Using a combination of clustering and supervised learning they were able to identify and classify botnets. Similarly, Guerid et. al., [16] analyzed DNS traffic to identify and cluster various botnets according to their NXDomain responses.

Zhou et. al., [23] leverages the idea that every domain name in the domain group generated by one botnet using DGA is often used for a short period of time and have similar life and query style. They clustered all requested domains according to the top level domain and IP. Then the clusters were compared to see if there are some clusters with similar lifetime span and similar visit time patterns. These clusters were then reported as malicious.

A number of researchers tried to use the fact that algorithmically generated domain names have an alphanumeric distribution different from legitimate domain names, as observed by McGrath and Gupta [13]. Yadav et. al., [21] used various metrics, such as K-L distance, Edit distance and Jaccard measure, to identify patterns inherent to algorithmically generated domains. Mae et. al., [12] employed statistical learning techniques based on lexical features (domain name length, host name, number of dots in URL, etc.). However, these techniques are ineffective against modern DGAs that use English dictionary words with slight modifications like adding suffixes -able, -hood, -ment, -ship, etc.

The aforementioned methods use information contained in the DNS resolve packets, which means that they need to have a DNS that is capable of logging all the requests or capture all the packets between the users and DNS servers. We are not aware of a method that uses only NetFlow information. Furthermore, techniques based on clustering of NXDomains suffer from a great amount of false positives generated from common typos and misconfigured applications, e.g. a common typo in google.com — let us say goooogle.com, can create a cluster of all users that made the same typo that would be labeled as DGA malware.

The previous work on service detection that we use for the request-response identification in Section III-B, was done by Berthier et. al. [6]. They used several heuristics which they combined using Bayesian inference. Some of their heuristics are correlated, like number of distinct ports, IPs and port-IP tuples. Others are based on the assumption that services use specific port numbers which is often true but is not guaranteed. Another heuristic is timestamps, which is often unreliable due to errors in NetFlow probes as described in Section III-A.

III. DGA malware detection

Our DGA detection algorithm is based on the fact that each host is expected to make a DNS query before contacting any IP, that was not previously visited by the host. Thus, we monitor the amount of DNS requests of each host in the local network together with the amount of unique IP addresses that it contacts. We then calculate the ratio $\rho(a)$ for each host a in the local network, defined as

$$\rho(a) = \frac{\delta(a)}{\pi(a) + 1}, \tag{1}$$

where $\delta(a)$ is the number of DNS requests created by the host a and $\pi(a)$ is the number of unique IP addresses contacted by the a. There is an addition of one in the denominator to avoid undefined values of $\rho(a)$ when the number of IPs contacted is zero.

It is important for the $\pi(a)$ to be the number of unique IPs that communicated with host a and a was an initiator of such communication. This allows us to detect DGA malware even if it is running on a server which we would not be able to do if we did not know the initiator of communication. Illustration of DGA-infected client and server machines is shown in Figure 1. Since the NetFlows are unidirectional, we are using request-response identification described in Section III-A to identify the initiator of each communication.

The ratio $\rho(a)$ is typically low for ordinary hosts. A high value of $\rho(a)$ is expected when a host is running DGA. We found that a high value can also indicate a server under some type of brute-force attack.[1] This means that both DGA and some brute-force activities can be detected using our approach.

[1] Servers are typically logging all suspicious activities together with the DNS resolve of the IP. An example may be an ssh server that for each unsuccessful login attempt logs DNS resolved information of the client.

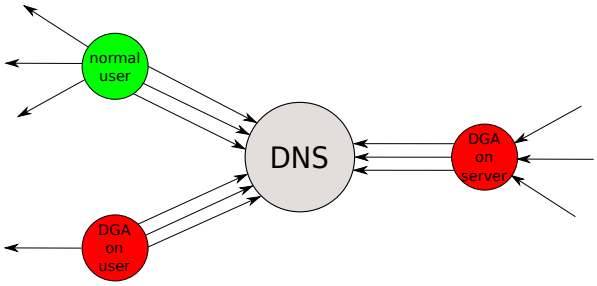


Fig. 1. Illustration of DGA behavior. A normal user has a number of DNS requests proportional to number of contacted IPs, whereas DGA on a user host has a high number of DNS queries without contacting any new IPs. Finally, a server with DGA is contacted by a high number of IPs from the network, but still the number of DNS requests it does is disproportionate to the number of IPs contacted by the server.

Figure 2 shows the histogram of $\rho(a)$ for all IPs from a part the Czech Technical University (CTU) network. As can be seen the ratio follows normal distribution with heavy tail on the right. Which supports our above mentioned assumption.

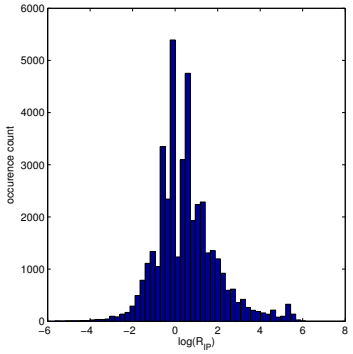


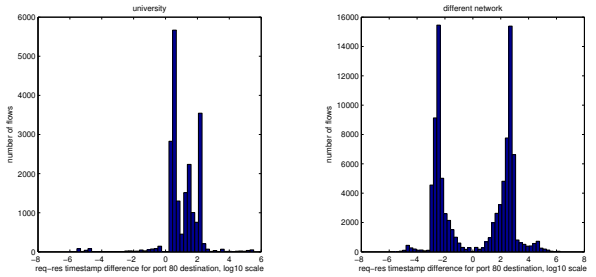
Fig. 2. Histogram of $\log \rho(a)$ of all hosts that make at least one DNS request in Czech Technical University network accumulated over several hours.

To detect the hosts infected by DGA malware we use a simple outlier detection technique, described in Section III-C, to identify hosts from the heavy tail of the distribution, that represent hosts with an unusually high ratio when compared to the majority of the hosts in the observed network and mark them as anomalous.

A. Request-response identification

In order to be able to correctly calculate the amount of IP addresses that the host contacts and differentiate them from IP addresses that the host is contacted by, we need to know which flows are requests and which are responses. Our request-response identification is done in two steps.

Since the request has to be prior to the response, we match request-response pairs based on IP-port-protocol triples and mark the NetFlow in a pair with smaller starting time as request and the other as response. The unmatched NetFlows are marked as requests without responses. This however suffers from high imprecision of timestamps on some network probes which sometimes leads to random labeling of NetFlows as requests and responses.



(a) CTU network (b) Large corporate network

Fig. 3. Timestamps difference histogram.

Different network probes have different error distributions of timestamps. Figure 3a shows the distribution of timestamp differences for all request-response pairs where requests were made to port 80 for a part of Czech Technical University network. We assume that all the communication with port 80 is communication with HTTP servers, which enables us to decide which flow is the request and which is response. There are approximately a thousand hosts in the network and all the data capture is done by a single software probe. Figure 3b shows the distribution of the same feature for a large corporate network. The network has around 50 thousand hosts. The capture of Netflows is done by several routers and hardware probes around the network. Request-response pairs with equal timestamps are not shown in the histogram.

In Figure 3a most of the timestamp differences are positive. That means the response has greater timestamp than the request, which is to be expected. In Figure 3b the distribution is nearly symmetrical around zero. That means that the request has 50 percent probability of having a smaller timestamp than corresponding response.

It is evident that in the university network, the timestamp difference is a strong feature while in the corporate network it leads to random results. This can be due to the fact that the second network is considerably larger and has more than one probe, which means that requests and responses can take different routes and potentially be captured by different probes altogether. These different probes do not necessarily have synchronized times between them. Even when running time synchronization services, such is NTP, can cause few to hundred milliseconds difference. In such environment set up, the timestamps feature is useless.

Next section introduces a service detection technique that is used to improve the above described method in the environments with high NetFlow timestamp errors. We assume that in client-service communications all flows from client to the service are requests and flows from service to client are responses. We therefore proceed to find all active services and mark NetFlows that are involved in communication with them accordingly.

B. Service Detection

The service detection algorithm is based on the median of number of peers difference. Let us define an endpoint as IP-port-protocol triple. The peer of an endpoint e is then an endpoint that communicates with e. The median number of

peer difference is used because endpoints that represent a service will typically have many peers (many clients that are connecting to that endpoint) and the client endpoints — peers of the service endpoint – will have a smaller amount of peers (clients are typically using a different port for every request, thus creating unique a endpoint for communication with every server).

The services obtained from the peers difference rule are then filtered using two other features, described further, in order to lower false positives.

1) Median of number of peers difference: First we calculate the number of peers that each endpoint communicates with. We then calculate the average difference in number of peers between an endpoint and all of its peers. More formally

$$d_e = \text{median}\{|P_e| - |P_i|\}_{i \in P_e},$$

where d_e is the median of number of peers difference for endpoint e. P_e and P_i are the sets of peers for endpoint e and i respectively.

Figure 4 shows a histogram of d_e values for CTU network over the time frame of 15 minutes. Port numbers were used for labeling the endpoints as services or clients in the histogram. Even though we used port numbers for labeling, we did not want to use them as a feature in the algorithm, because many services run on higher ports and many clients (e.g. NTP) communicate from lower ports. This is something that is also reflected in the histogram. Most endpoints with low ports and negative or zero values were either NTP, SNMP or similar which communicate on lower ports for both sides or HTTP servers with one or two clients only. The endpoints with high port number and positive value were usually legitimate services like HTTP on port 8080.

The final decision if an endpoint e is a service is made by thresholding the d_e values. Services are those endpoints whose median of number of peers difference is positive, the rest are considered clients.

2) Unsuccessful connections: The median of number of peers difference is a strong feature, however, it can lead to some false positives. One is when a scan from a fixed IP port is performed. In this scenario the endpoint performing the scan has a high number of peers and would be classified as a service. This endpoint can be differentiated from a service by the number of unsuccessful connections originated by that endpoint f_e. An ordinary service should have zero or a very low f_e, while f_e for a scan will usually be high.

3) Communication via both TCP and UDP on ports higher than 1023: A lot of P2P traffic is also caught by the service detector. This is because P2P often uses a fixed port for communication and contacts a high number of hosts therefore the number of peers for an endpoint involved in P2P can be high.

Many P2P communications can be filtered out using a simple rule — communication on a port higher than 1023 using both TCP and UDP protocols which is typical of peer to peer [11].

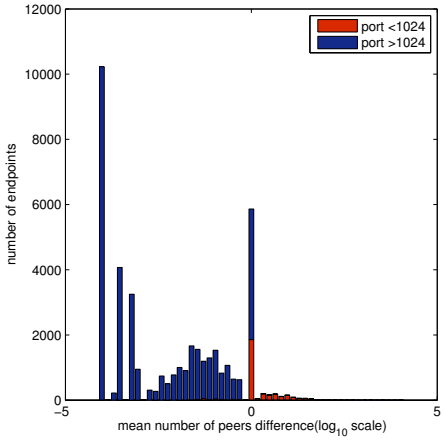


Fig. 4. Median of number of peers difference histogram, 15 minutes aggregation.

C. DNS anomaly detector

We expect the majority of the network connections to be legitimate. Only a small amount of traffic is expected to be malicious [15]. If we assume that the ratio $\rho(a)$, defined in Equation 1, of the legitimate behavior follows a normal distribution, we can label the tails that have really low probability as anomalous.

For normal distribution we have:

$$f(x) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{2}$$

where the μ and σ are the mean and standard deviation of the $\rho(a)$ values of all hosts a in the network.

Thus the probability that a point will lie in more t standard deviations from the mean is equivalent to the area under the normal probability density function in intervals $[-\infty, \mu - t\sigma]$ and $[\mu + t\sigma, \infty]$.

Since we are interested only in detection of hosts that have more DNS requests than number of visited IPs, which corresponds to higher value of $\rho(a)$, we are labeling only the right tail as anomalous. The left tail represents behaviors in which the host contacts a lot of IPs without DNS queries, such as horizontal scanning or P2P networks.

The anomaly detection algorithm first calculates the ratio $\rho(a)$ given by Equation 1 for every host on the local network. We model the normal distribution by the mean and standard deviation $\rho(a)$ of all hosts on the local network in five minute NetFlow batches. The choice of the time interval was made on the basis of the related works [18], [22] showing 5-minute intervals give good performance. It is updated every five minutes using exponential forgetting given by

$$\hat{u}_n = \alpha \hat{u}_{n-1} + (1 - \alpha) u_n,$$

where u_n is the input at step n, $\hat{u}_n$ is the updated value, $\hat{u}_{n-1}$ is the remembered value from the previous step and α is the forgetting coefficient.

TABLE I. AUCs

Day	1	2	3
AUC value	0.9733	0.9923	0.9989

Anomaly values are obtained using a fuzzy function given by

$$f(x) = \begin{cases} 0 & \text{if } x \leq \mu + t_1\sigma \\ \frac{x-(\mu+t_1\sigma)}{(t_2-t_1)\sigma} & \text{if } \mu + t_1\sigma < x < \mu + t_2\sigma \\ 1 & \text{if } x \geq \mu + t_2\sigma \end{cases},$$

where x is the value of the ratio for a given endpoint, μ and σ are the current model values and t_1 and t_2 are thresholds.

Thresholds t_1 and t_2 are important parameters. They are set manually and essentially determine the sensitivity of the detector. Based on our experiments, the thresholds were set to $t_1 = 2$ and $t_2 = 4$, selecting only the samples with probability smaller than 0.04550 and 0.00006 respectively.

D. DNS resolver detection

DNS resolver are typical false positives of the proposed DGA detector. For each DNS request they get they make several DNS requests themselves, to resolve the DNS record not present in their cache. They therefore look anomalous from the standpoint of a DGA detector. We filter out DNS resolvers by using information from service detection defined in Section III-B.

IV. Experiments

A. Controlled infection

Experiments were conducted using deliberate infection with Shiz malware [1]. We infected a clean Windows machine connected to the CTU network. The malware was run for several hours on three different days and times to capture the various states of the network resulting in three different experiments. Detection was conducted on the NetFlows generated from the traffic of the whole university network. Collected NetFlows were labeled using the source IP addresses of the infected hosts as malicious and the rest of the traffic was labeled as legitimate. So if there was DGA-performing malware already present it would lower efficacy results. The area under the curve (AUC) of the receiver operating characteristic (ROC) were calculated from the outputs of the detector for each of the conducted experiments. The results, given in Table I, show very high detection quality for each of the presented experiments that represent the network in different states.

Figure 5 shows the distribution of anomaly values for one five-minute batch in which the DNS anomaly detector identified traffic generated by Shiz. The traffic from Shiz is labeled red in the histogram, based on the IP address of the machine in which it was running. Only the outgoing traffic is labeled, as it is what we want to detect. Please note that some of the background traffic was also labeled as very anomalous. Manual inspection of those NetFlows revealed that the traffic was originating from several IP addresses that really did show some DGA-like behavior. For example, one of the cases was a mail server that performed reverse DNS resolution for every client, resulting in hundreds of DNS requests every five minutes.

TABLE II. User-server comparison.

Host type	inactive host	user	server
AUC value	0.9811	0.9179	0.9432

B. Mixed — various host types

Another set of experiments was conducted using NetFlow records of the Shiz malware [1]. The sample was run inside a virtual machine for 12 hours and packet captures (pcap) were created. NetFlows were generated from the pcaps and then mixed into background traffic from the CTU network. Three datasets were generated with the same background traffic, starting at the same time. In the first dataset the malware traffic was mixed into a previously inactive IP address, in the second dataset into a user IP address and in the third dataset into an HTTP server IP address. The AUC values are shown in Table II.

The best results are for the previously inactive IP, with the highest AUC values. This is because it does not have any other traffic except for DGA, therefore its $\rho(a)$ ratio is high. The host with the user has the lowest AUC. This is because the user is active, he contacts websites and other servers, therefore his $\rho(a)$ is lowered by the number of IPs he contacts. The HTTP server's AUC is somewhere in between. This is probably because the server also contacts some IPs, possibly for time synchronization or updates. Its $\rho(a)$ is not affected by the users of the server, because we are able to differentiate between requests and responses. A version of the DGA detector that did not make a distinction between requests and responses got 0.8229 AUC on the same dataset.

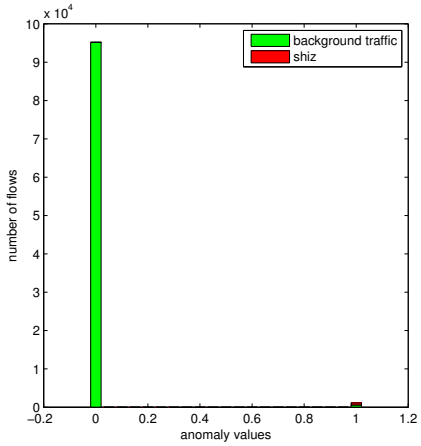


Fig. 5. Anomaly values assigned to flows from the CTU network where Shiz malware was run.

C. Mixed — various DGA malware families

In the last experiment we used ten malware samples of six different families. Again, all the malware samples were run inside a virtual machine for 12 hours and pcaps were captured. NetFlows were generated from the pcaps and then mixed into background traffic from a large size company with more than 50 000 users. The malicious traffic was mixed into the traffic of six randomly selected active user IPs and six active servers. Each row of the Table III shows the minimal

TABLE III. AUCs OF VARIOUS MALWARE SAMPLES .

Host type	Malware family	mean(AUC)	min(AUC)
User	Trojan-Generic	0.9926	0.9883
	Variant-Kazy	0.9943	0.9943
	Win32-AutoRun	0.9303	0.8650
	Win32-AutoRun	0.9259	0.7974
	Win32-AutoRun	0.9279	0.8093
	Win32-Waski-A	0.8894	0.8646
	GameOverZeus	0.9837	0.9505
	GameOverZeus	0.9907	0.9843
	GameOverZeus	0.9830	0.9684
	caphaw	0.9920	0.9824
Server	Trojan-Generic	0.9687	0.9156
	Variant-Kazy	0.9953	0.9953
	Win32-AutoRun	0.8877	0.6966
	Win32-AutoRun	0.9116	0.7703
	Win32-AutoRun	0.8977	0.7325
	Win32-Waski-A	0.8586	0.7478
	GameOverZeus	0.9720	0.9355
	GameOverZeus	0.9687	0.9175
	GameOverZeus	0.9767	0.9554
	caphaw	0.9588	0.8857

and average AUC score over all six hosts of the same type for a specific malware sample. As can be seen from the table the presented technique is able to detect various DGA malware families with high precision when running on the user machine — the minimal AUC is above 0.80 and the average AUC score above 0.89 for all the tested samples. The efficacy is slightly worse for the server hosts, where the minimal AUC score is 0.70 for the Win32-AutoRun malware sample and the average is above 0.86 for all the samples. We should also note that for our experiments we assume that the original traffic that was not altered by mixing is legitimate and does not contain any hosts infected by DGA performing malware. This is not necessarily true, manual inspection of the false positive showed many hosts performing DGA like activities. Unfortunately, due to lack of additional information, it was impossible to definitely prove or disprove an actual infection by malware.

V. CONCLUSION

In this paper we presented a method that, using only NetFlow information, can effectively detect hosts compromised with DGA-performing malware inside the monitored network. We analyze 5-minute batches of NetFlow data and model the mean and variance of the ratio between the number of DNS requests and newly visited IPs of every host in the local network. Deviations from the mean are labeled as anomalous DNS behaviors that typically corresponds to DGA-based malware. Our experiments show that the proposed solution is able to identify host infected by Shiz malware together with several suspicious hosts that were performing a lot of DNS requests. Deep inspection showed that these hosts were either services under brute-force attack or mail servers or similar, that are logging DNS information of their clients.

REFERENCES

[1] Lavasoft Alexander Adamov. Backdoor.Win32.Shiz. http://www.lavasoft.com/mylavasoft/malware-descriptions/blog/backdoorwin32shiz, 2012.

[2] P Amini and C Pierce. Kraken botnet infiltration, 2008.

[3] Manos Antonakakis, Roberto Perdisci, Yacin Nadji, Nikolaos Vasiloglou, Saeed Abu-Nimeh, Wenke Lee, and David Dagon. From throw-away traffic to bots: detecting the rise of dga-based malware. In *Proceedings of the 21st USENIX Security Symposium*, 2012.

[4] B. RFC 3954 - Cisco Systems NetFlow Services Export Version 9, October 2004.

[5] Paul Barford and Vinod Yegneswaran. An inside look at botnets. In *Malware Detection*, pages 171–191. Springer, 2007.

[6] Robin Berthier, Michel Cukier, Matti Hiltunen, Dave Kormann, Gregg Vesonder, and Dan Sheleheda. Nfsight: netflow-based network awareness tool. *Proceedings of the 24th USENIX LISA*, 2010.

[7] Benoit Claise, Brian Trammell, and Paul Aitken. Rfc 7011: Specification of the ipfix protocol for the exchange of flow information, Sep 2013.

[8] David Dagon. Botnet detection and response. In *OARC workshop*, volume 2005, 2005.

[9] David Dagon, Guofei Gu, Christopher P Lee, and Wenke Lee. A taxonomy of botnet structures. In *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, pages 325–339. IEEE, 2007.

[10] Thorsten Holz, Christian Gorecki, Konrad Rieck, and Felix C Freiling. Measuring and detecting fast-flux service networks. In *NDSS*, 2008.

[11] Thomas Karagiannis, Andre Broido, Michalis Faloutsos, et al. Transport layer identification of p2p traffic. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 121–134. ACM, 2004.

[12] Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. Beyond blacklists: learning to detect malicious web sites from suspicious urls. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1245–1254. ACM, 2009.

[13] D Kevin McGrath and Minaxi Gupta. Behind phishing: An examination of phisher modi operandi. *LEET*, 8:1–8, 2008.

[14] Phillip Porras, Hassen Saidi, and Vinod Yegneswaran. Conficker c analysis. *SRI International*, 2009.

[15] Leonid Portnoy, Eleazar Eskin, and Sal Stolfo. Intrusion detection with unlabeled data using clustering. In *In Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001*, pages 5–8, 2001.

[16] Jason Reed, Adam J Aviv, Daniel Wagner, Andreas Haeberlen, Benjamin C Pierce, and Jonathan M Smith. Differential privacy for collaborative security. In *Proceedings of the Third European Workshop on System Security*, pages 1–7. ACM, 2010.

[17] Christian Rossow, Dennis Andriesse, Tillmann Werner, Brett Stone-Gross, Daniel Plohmann, Christian J Dietrich, Herbert Bos, FKIE Fraunhofer, and Dell SecureWorks. Sok: P2pwnedmodeling and evaluating the resilience of peer-to-peer botnets.

[18] Fernando Silveira, Christophe Diot, Nina Taft, and Ramesh Govindan. Astute: detecting a different class of traffic anomalies. In Shivkumar Kalyanaraman, Venkata N. Padmanabhan, K. K. Ramakrishnan, Rajeev Shorey, and Geoffrey M. Voelker, editors, *SIGCOMM*, pages 267–278. ACM, 2010.

[19] Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna. Your botnet is my botnet: analysis of a botnet takeover. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 635–647. ACM, 2009.

[20] Sam Stover, Dave Dittrich, John Hernandez, and Sven Dietrich. Analysis of the storm and nugache trojans: P2p is here. *USENIX; login*, 32(6):18–27, 2007.

[21] Sandeep Yadav, Ashwath Kumar Krishna Reddy, AL Reddy, and Supranamaya Ranjan. Detecting algorithmically generated domain-flux attacks with dns traffic analysis. *IEEE/ACM Transactions on Networking (TON)*, 20(5):1663–1677, 2012.

[22] Y. Zhang, Z. Ge, A. Greenberg, and M. Roughan. Network anomography. In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, IMC '05, page 30, Berkeley, CA, USA, 2005. USENIX Association.

[23] Yong-lin Zhou, Qing-shan Li, Qidi Miao, and Kangbin Yim. Dga-based botnet detection using dns traffic. *Journal of Internet Services and Information Security (JISIS)*, 3(3/4):116–123.