Chapter 8

INFORMATION-ENTROPY-BASED DNS TUNNEL PREDICTION

Irvin Homem, Panagiotis Papapetrou and Spyridon Dosis

Abstract DNS tunneling techniques are often used for malicious purposes. Network security mechanisms have struggled to detect DNS tunneling. Network forensic analysis has been proposed as a solution, but it is slow, invasive and tedious as network forensic analysis tools struggle to deal with undocumented and new network tunneling techniques.

This chapter presents a method for supporting forensic analysis by automating the inference of tunneled protocols. The internal packet structure of DNS tunneling techniques is analyzed and the information entropy of various network protocols and their DNS tunneled equivalents are characterized. This provides the basis for a protocol prediction method that uses entropy distribution averaging. Experiments demonstrate that the method has a prediction accuracy of 75%. The method also preserves privacy because it only computes the information entropy and does not parse the actual tunneled content.

Keywords: Network forensics, DNS tunneling, information entropy

1. Introduction

Recent years have seen an increase in the use of DNS tunneling to stealthily perpetrate malicious activities such as exfiltrating sensitive data, hiding network attacks and orchestrating malware activities via botnet communications [23]. Several strains of malware such as the Morto worm [18] and Feederbot [9], and variants of point-of-sale malware such as BernhardPOS and FrameworkPOS [22] demonstrate the increased popularity of DNS tunneling to implement stealthy communications. The availability of DNS tunneling tools, such as NSTX, Iodine, `dnscat`, DeNiSe, OzymanDNS and Heyoka [19], have also enhanced the popularity and uptake of DNS tunneling.

G. Peterson and S. Shenoi (Eds.): Advances in Digital Forensics XIV, IFIP AICT 532, pp. 127–140, 2018.
https://doi.org/10.1007/978-3-319-99277-8_8

Preventive measures are unable to curtail these activities [22]. DNS tunneling detection mechanisms have been developed [6, 7], but they discover only 3% of attacks in sophisticated real-world cases [22]. Research has focused on improving the detection of tunneling [13, 14], but despite these efforts, network breaches that leverage tunneling are on the increase [18, 22].

Reactive security mechanisms such as network security monitoring and network forensic analysis techniques offer some promise. However, the techniques are often manual and labor intensive [8]. The techniques also require considerable expertise and are very time-consuming, taking up to seven months [17, 22]. Furthermore, available network forensic analysis tools only parse standardized network protocols; previously unseen or undocumented protocols commonly used in DNS tunneling require manual dissection [8]. New and innovative methods are sorely needed to alleviate these challenges and speed up the forensic analysis of tunneled network traffic.

The primary goals of network forensic analysis of tunneled networked traffic are to identify the carrier tunneling protocol, the internally tunneled protocol, the communicating parties, the content being tunneled and its significance. This work assumes that the identification of DNS tunneling has already been accomplished using, for example, the methods described in [6, 13]. Thus, the focus is on the next important task – the discovery of the network protocol being carried inside a DNS tunnel.

Little, if any, work has focused on the forensic analysis of DNS tunneling techniques. Also, no work has been done on identifying tunneled network protocols. To address this gap, this research has sought to develop a prediction mechanism that probabilistically identifies the network protocols carried in DNS tunnels. The hypothesis is that a network protocol exhibits a unique entropy distribution in its byte content during normal use. Furthermore, a network protocol carried in a tunneling mechanism maintains some similarity to its original byte entropy distribution. This enables the probabilistic matching of DNS tunneled traffic to a particular protocol with reasonable accuracy.

This research is limited to identifying the HTTP and FTP protocols when tunneled individually by a single DNS tunneling tool. The identification of the IP addresses of the communicating parties and the exact content transmitted in the messages of an internally-tunneled protocol are topics for future work. For simplicity, the focus is on the popular Iodine DNS tunneling tool, although the proposed approach could be applied to tools that use other DNS tunneling methods. Multilevel nesting of tunneled protocols is not considered, nor are encrypted protocols carried in tunneling protocols or tunneling protocols that use encryp-

tion, such as IPSec, SSH and SSL. The reason is that encryption makes information entropy uniform per block or per stream, rendering different portions of a message indistinguishable from each other. In any case, these are all interesting topics for future work.

The proposed method is implemented in a network protocol prediction tool. Experiments with the tool yield promising results with a prediction accuracy of 75%. Given the large volumes of network traffic captures encountered in digital investigations, the protocol prediction tool can help speed up the triage process as well as the analysis of network traffic. Specifically, a forensic analyst could identify network flows where a certain suspect protocol may be present in the network traffic, but is hidden by DNS tunneling. The tool also preserves privacy because it does not parse actual content; rather, it only computes the information entropy of a specific field in a packet.

2. Background and Related Work

Relatively few studies have sought to determine the protocols carried within a tunneling protocol. Bernaille and Teixeira [3] have classified the traffic of protocols tunneled over SSL using only the sizes of the first few packets of an SSL session. Their approach differentiates SSL traffic from normal traffic and uses a clustering mechanism based on Gaussian mixture models to distinguish between several protocols (HTTP, FTP, BitTorrent, edonkey, SMTP and POP3) tunneled over SSL. Dusi et al. [10] have employed statistical analysis of packet sizes and inter-arrival times to fingerprint normal SSH usage and when SSH is used for tunneling other protocols. Dusi et al. [11] subsequently extended their approach to distinguish HTTP tunneling traffic from normal HTTP traffic and to predict the presence of plaintext protocols such as POP3, SMTP, Chat and P2P in SSH and HTTP tunnels. Alshammari and Zincir-Heywood [2] have used Adaboost, C4.5 and genetic-programming-based classifiers to distinguish Skype and SSH traffic from other traffic, and to identify the type of application traffic (Shell, SFTP, SCP, local/remote forwarding or X11) in SSH tunnels.

Other researchers have focused on identifying proprietary protocols such as Skype and Spotify in other network traffic [1, 4, 15]. Song et al. [20] have extracted exact keystrokes from encrypted live SSH shell sessions, demonstrating the inference of content in tunnels.

Most studies on DNS tunneling have focused on detection. Born and Gustafson [6] have developed an n-gram character frequency analysis method for identifying domain names typical of DNS tunneling traffic. Xu et al. [23] have presented an anomaly detection method that contrasts

the statistical and information-theoretic properties of payload content in normal DNS traffic from those of DNS tunneled traffic. Farnham [13] discusses several DNS tunneling tools and detection heuristics, including DNS request and response sizes, domain name entropy, use of uncommon resource records, volume of DNS requests per IP address or per domain, number of sub-domains per domain and presence of large numbers of orphaned DNS requests.

Davidoff and Ham [8] have developed initial methods for manually disassembling DNS tunneling traffic to recover the internally-carried protocols and data. However, no studies have focused on the automated prediction of network protocols carried in DNS tunneling traffic for the benefit of forensic analysis, which is the principal thrust of the research described in this chapter.

3. DNS Tunnel Internals and Dataset Collection

The flexibility of the DNS protocol enables DNS tunneling tools to leverage a number of techniques [5]. Many tools append data as a sub-domain in the name field of queries, but they vary in their ease of use, throughput and invisibility to security mechanisms. For example, the `dns2tcp` tool uses TXT records, Iodine uses NULL records and DNScat uses CNAME records. Iodine and Heyoka use EDNS(0) extensions to increase throughput [5].

3.1 Tunneling with Iodine

The research described in this chapter employs the Iodine DNS (IP-over-DNS) tunneling tool due to its popularity, ease of use and availability of documentation [8]. Iodine encapsulates IPv4 packets in the payloads of DNS packets. By default, it uses NULL resource records, but it can also use other resource records such as PRIVATE, TXT, SRV, MX, CNAME and A. The query/answer name field in the resource record in use holds the encapsulated data.

Upstream data is compressed with GZIP and encoded. Encoding options include Base32, Base64 (or Base64u) and Base128. This is determined by checking for character set support at intermediate DNS servers. Downstream data is compressed with GZIP and encoded, as in the case of upstream data [12]. When encoding is applied, the downstream header is prepended with an ASCII character that signifies the encoding type that is used.

Tunneled data in the name field consists of a header and the fragment of the packet being tunneled, prepended as a sub-domain of the tunneling server. The header preceding the tunneled data contains metadata such

as the user ID, codecs in use (Base32/64/128), fragment size, fragment number, sequence number, whether compression is used and a cache miss counter [12].

3.2 DNS Tunneling Setup and Dataset Capture

No well-known DNS tunneling network traffic captures are available and it is difficult to obtain network traffic captures involving malicious DNS activity [21]. As a result, a DNS tunnel was set up using the Iodine tool to create the experimental dataset.

The DNS tunneled traffic dataset was created by simulating the use of HTTP and FTP protocols, each in its own DNS tunnel. HTTP traffic over DNS was generated by performing simple web requests to eight websites, allowing for additional requests for extra content (images, CSS, JavaScript, Ads) to continue. The FTP protocol was simulated by downloading several files that were placed on an FTP server prior to the experiments. To ensure variation, the downloaded files were of different types, including image, PDF, text, audio, video and ZIP files. Twelve files were downloaded in all and stored at different paths on the FTP server.

The dataset comprised a total of 20 DNS tunneled traffic samples. Eight were HTTP communications, one for each of the eight websites. The other twelve were FTP communications involving logins, directory traversals and individual file downloads.

The control dataset for comparison of the tunneled network protocols against their plain versions was created by simulating normal HTTP web traffic and normal FTP traffic. The HTTP traffic was generated by visiting a randomly-chosen website with substantial content to be loaded. Normal FTP traffic was generated by logging into an FTP server and sequentially downloading three files. Multiple files were chosen due to the terse nature of FTP protocol commands.

4. Protocol Feature Trends and Analysis

The proposed method for network protocol prediction identifies patterns based on features found in plain protocol traffic that can be mapped to equivalent features in DNS tunneled traffic. Several features could be chosen to characterize the differences between protocols and the similarities across the plain and tunneled protocol versions; these include byte frequencies, information entropy and packet lengths. However, the proposed method employs information entropy analysis because it is inherently tied to the actual data bytes in packets. The idea is to observe normal HTTP and FTP traffic, and to compare the traffic against their

DNS tunneled equivalents that have been fragmented, encoded and compressed in the tunneling process.

4.1 Experimenting with Information Entropy

Measurements may be made at different levels of abstraction of a network packet to characterize its features: (i) IP packet level; (ii) transport level; and (iii) application level. Differences also can be identified between protocol client requests and server responses. HTTP and FTP have relatively small vocabularies of commands and content that go into requests, but their responses could include large amounts of data with considerable variation. Therefore, the focus is only on requests whose content and variation are more predictable and likely more significant for comparisons.

Information entropy is a measure of the variations of the components that make up a message. The proposed method computes the entropy of the bytes that make up a packet layer or field value as follows:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log p(x_i) \tag{1}$$

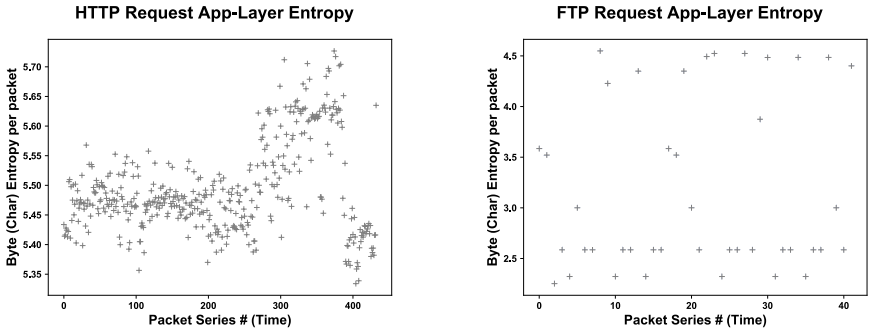
where $p(x_i)$ is the probability of a particular byte occurrence and n is the number of byte occurrences.

The hypothesis underlying the use of entropy is that the request packets in a protocol flow have a specific entropy distribution. The hypothesis was tested by creating a simple Python program using the Scapy and Matplotlib libraries to measure information entropy and plot charts for visual analysis of the distribution trends.

4.2 Comparison of Plain and Tunneled Protocols

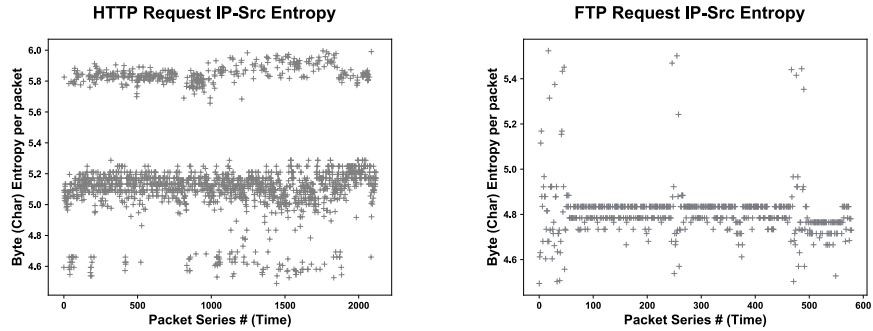
Figures 1(a) and 1(b) show the entropy distributions of application layer requests for plain HTTP and plain FTP network traffic, respectively. The traffic was filtered by taking only the packets containing application layer content and destined to port 80 and port 21 for HTTP and FTP, respectively. Thus, no transport layer features were present and no ACKs were observed because ACKs do not have any application layer content.

Figures 2(a) and 2(b) show the entropy distributions at the IP packet level for plain HTTP and FTP traffic, respectively. The traffic was filtered at a more granular level in that all the IP traffic from the client was captured. The packet entropy distributions contain packets with application data and packets with transport layer ACKs because HTTP



(a) HTTP requests. (b) FTP requests.

Figure 1. Application layer entropy of plain protocols.

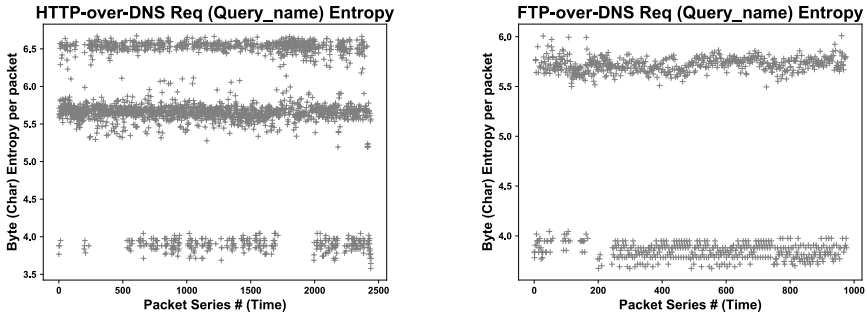


(a) HTTP source IP address traffic. (b) FTP source IP address traffic.

Figure 2. IP layer entropy of plain protocols.

and FTP both use TCP for the transport layer. The client ACKs acknowledge the receipt of prior server S-ACKs that originated from the delivery of requests to the server.

Figures 3(a) and 3(b) show the entropy distributions of HTTP-over-DNS traffic and FTP-over-DNS traffic, respectively. Specifically, the figures show the entropy of the query name field for DNS requests destined for DNS port 53. Because the filtering was based on the client side traffic destined for port 53, it includes tunneled application protocol request packets as well as those embodying transport layer properties (e.g., ACKs, sequencing and reliability information). Filtering was performed in this manner because no established methods exist for parsing and differentiating between application layer data and data from other layers of DNS tunneled traffic. Privacy was preserved because computa-



(a) HTTP-over-DNS requests. (b) FTP-over-DNS requests.

Figure 3. DNS request query name field entropy of tunneled protocols.

tions were performed over the packet content to generate metadata (i.e., entropy) without necessarily making sense of the actual content.

In the case of Base128 encoded traffic (default used by Iodine), the theoretical maximum entropy for completely random data tends towards eight bits. The distributions in Figures 1 through 3 show discernible variations between the protocols as well as between different layers of abstraction. The distributions are not uniformly distributed at the eight-bit value, demonstrating an absence of absolute randomness. The distributions also have different patterns that may indicate the presence of specific network protocols.

The HTTP application layer traffic entropy values in Figures 1(a) and 1(b) are significantly denser than those for FTP traffic. The FTP traffic has application layer packet entropy values around 4.5 and other values around 2.5 whereas the HTTP application layer traffic has most of its entropy values between 5.4 and 5.6.

In Figures 2(a) and 2(b), the density of packets appears to be less for FTP traffic than for HTTP traffic. There are three bands of clusters for HTTP traffic, with the densest band having entropy values between 4.9 and 5.3; the other two bands are around 4.6 and 5.8. FTP traffic has a single dense cluster of entropy values around the 4.8 mark. However, one could argue for the presence of two very sparse clusters around the 5.4 and 4.5 entropy values. These sparse clusters may be significant when one looks across the sequence of packets, where the hypothesized variations in entropy away from the 4.8 mark correspond to the FTP requests that initiated downloads of the three files.

Figures 3(a) and 3(b) reveal some clustering of the entropy values for HTTP-over-DNS traffic and FTP-over-DNS traffic, respectively. The HTTP-over-DNS traffic in Figure 3(a) appears to have three cluster

bands – the densest band is around 5.6 and 5.8, the next less dense band is around 6.5 and the least dense band is in the 3.8 to 4.0 entropy range. The FTP-over-DNS traffic in Figure 3(b) has two main clusters of entropy values – one band is in the 5.6 to 5.8 range and the other is in the 3.8 to 4.0 range.

Shorter message sizes and smaller alphabets produce lower entropy values [3]. In the case of tunneled traffic with a fixed Base128 encoding, two types of traffic could have lower entropy values due to their shorter packet lengths. One is the ACKs from the transport layer and the other is the short query request messages (ping traffic) sent by a DNS tunneling tool to prevent DNS servers from timing out.

It can be postulated that the bands in Figures 3(a) and 3(b) around the 4.0 mark correspond to the entropy values of the pings because they are shorter (less than 30 bytes) than those of ACKs (at least 40 bytes) when seen in a manual DNS tunneling disassembly. This is reinforced by the fact that they are less dense in the HTTP traffic because an HTTP request can spawn several other HTTP requests to retrieve more content in order to load websites properly. These HTTP requests keep the connection between the DNS tunneling client and tunneling server open, resulting in fewer pings being generated for time-out prevention.

The bands in the 5.6 to 5.8 range for tunneled HTTP traffic and tunneled FTP traffic in Figures 3(a) and 3(b), respectively, likely correspond to ACKs in the transport layer. This inference is made primarily because the HTTP-over-DNS traffic has another band with higher entropy that likely corresponds to HTTP requests, which have higher entropy values because they are the longest messages. Also, the small repetitive pattern seen in the first 200 packets in Figure 3(b) appears to correspond to the downloading of the three files, implying that the band around the 4.0 mark contains FTP request packets hidden in the same cluster as the pings. This is reinforced by the fact that FTP commands in FTP requests are terse and have a fixed structure. This would contribute to a lower entropy than for the HTTP requests, which have a larger request header set and many more fields.

The clustering of different types of packets based on the effects that their content and lengths have on entropy implies that inherent distributions exist that can help distinguish between DNS tunneled traffic that contains different protocols. The different average values of the bands suggests that the average entropy of tunneled network traffic can help distinguish between internally-tunneled protocols. The next section discusses protocol prediction experiments that explore whether these trends and mean entropy values can identify internally-tunneled protocols.

5. Protocol Prediction Experiments

The dataset containing 20 test traffic captures and two ground truth captures was used to determine the similarities between the entropy distributions (variables) of plain network traffic of a particular protocol and its equivalent DNS tunneled versions. A simple similarity metric based on the averages of the distributions (variables) was employed. This metric is referred to as "MeanDiff," which is the shortened form of "Mean Differences." It is computed as the absolute difference between the means of the two variables:

$$m(X, Y) = |\mu_X - \mu_Y| \tag{2}$$

where the variable X corresponds to the entropy values of the ground truth protocol packet capture over time, and variable Y corresponds to the entropy values of a specific tunneled test capture.

5.1 Results

A classifier tool was written to evaluate the suitability of the MeanDiff similarity metric for predicting the underlying protocol in a DNS tunneled network traffic capture. The classifier scripts are available at a GitHub repository [16].

The tool takes two ground truth traffic captures, one containing plain HTTP traffic and the other containing plain FTP traffic. It computes the entropy values of each packet in a capture stream at the IP packet level, generating two entropy distributions, one for HTTP traffic and the other for FTP traffic.

The tool then accepts a DNS tunneled traffic capture with an unknown internally-tunneled protocol. It performs random sampling by selecting a consecutive series of entropy values from the DNS tunneling capture. The random sample series length is set at 90% of the length of the ground truth capture used for comparison. One thousand samples are taken and the MeanDiff metric is calculated for each sample against the respective HTTP and FTP entropy distributions. An average of the 1,000 rounds is then taken as the MeanDiff score against the respective ground truth entropy distributions (HTTP and FTP) for a given DNS tunneled sample. This score is used as the basis for prediction, where MeanDiff is the distance metric. The ground truth protocol with the smallest MeanDiff score is deemed to be closest to the test DNS tunneled traffic sample. This ground truth protocol is the predicted internally-carried protocol.

Table 1. Sample run results.

No.	DNS Tunneled Sample	True Value	MeanDiff Prediction
1	[amazon]	HTTP	HTTP
2	[bbc]	HTTP	HTTP
3	[craigslist]	HTTP	FTP
4	[dsv.su.se]	HTTP	HTTP
5	[en.wikipedia]	HTTP	HTTP
6	[facebook]	HTTP	HTTP
7	[google]	HTTP	FTP
8	[youtube]	HTTP	FTP
9	[audio-wav]	FTP	HTTP
10	[audio-mp3]	FTP	HTTP
11	[img-jpg1]	FTP	FTP
12	[img-jpg2]	FTP	FTP
13	[img-png1]	FTP	FTP
14	[img-png2]	FTP	FTP
15	[pdf1]	FTP	FTP
16	[pdf2]	FTP	FTP
17	[txt1]	FTP	FTP
18	[txt2]	FTP	FTP
19	[video]	FTP	FTP
20	[zipfile]	FTP	FTP

5.2 Discussion

The classifier tool was applied to the dataset. Table 1 shows the actual and predicted protocols in a sample run. The MeanDiff metric yields a prediction accuracy of approximately 75%. Subsequent runs yielded 70% to 80% prediction accuracy, demonstrating the promise of the proposed approach for predicting DNS tunneled protocols. Note that five of the eight HTTP-over-DNS test samples were classified correctly, corresponding to a 62.5% recall (true positive) rate. Also, ten of the twelve FTP-over-DNS test samples were classified correctly, corresponding to an 83.3% recall.

The confusion matrix in Table 2 summarizes the classifier performance. The misclassification rate is 25%. The precision is 71.4% for HTTP and 76.9% for FTP. The false positive rate is 37.5% for the HTTP class and 16.7% for the FTP class. These results demonstrate the effectiveness of the method for predicting the underlying network protocols in DNS tunnels.

Table 2. Classifier performance confusion matrix.

		Predicted		
	N = 20	**HTTP**	**FTP**	**Total**
Actual	**HTTP**	5	3	8
	FTP	2	10	12
	Total	7	13	20

6. Conclusions

This research has taken on the challenging task of predicting the application protocols tunneled in DNS traffic. The exploration of the internal structure of DNS tunneling techniques contributed to the use of entropy distributions of packet bytes in a method for characterizing and predicting internally-tunneled protocols. Packet traces were visualized in order to identify patterns arising from various protocol packets due to their content and function. A classifier tool was developed and applied to a dataset of DNS tunneled traffic to evaluate the approach. Protocol classification based on entropy value averages yielded a prediction accuracy of 75%, indicating that the method holds promise.

DNS tunneling is increasingly leveraged in security breaches and other criminal activities. The proposed method assists forensic analysts in triaging and identifying DNS tunneling network traffic that may contain protocols of interest, enabling them to focus on specific DNS tunnel flows instead of having to analyze all the DNS tunneled traffic. The proposed method also preserves privacy because it only computes the information entropy and does not scrutinize the contents of packets. This is an important feature that enables the method to adhere to privacy laws that limit the invasive nature of forensic investigations.

This research is just the first foray into the relatively unexplored field of DNS tunneled traffic forensics. Although the 75% prediction accuracy obtained in the experiments is quite good, certain improvements can be made to improve the performance. One approach is to incorporate features in addition to information entropy in tunneled protocol classification; these features include packet lengths, inter-arrival times and character n-grams. A wider analysis of DNS tunneling techniques and candidate internally-tunneled protocols would help identify the best set of features for classification. Other statistical metrics that offer fine-

grained differentiation of protocols should also be explored. Finally, machine learning and data mining techniques should be leveraged to improve protocol classification. For example, current research is employing dynamic time warping in time series analysis for robust matching of plain protocols against their DNS tunneled variants.

References

[1] R. Alshammari and A. Zincir-Heywood, Machine learning based encrypted traffic classification: Identifying SSH and Skype, *Proceedings of the IEEE Symposium on Computational Intelligence in Security and Defense Applications*, 2009.

[2] R. Alshammari and A. Zincir-Heywood, Can encrypted traffic be identified without port numbers, IP addresses and payload inspection? *Computer Networks*, vol. 55(6), pp. 1326–1350, 2011.

[3] L. Bernaille and R. Teixeira, Early recognition of encrypted applications, *Proceedings of the Eighth International Conference on Passive and Active Network Measurement*, pp. 165–175, 2007.

[4] D. Bonfiglio, M. Mellia, M. Meo, D. Rossi and P. Tofanelli, Revealing Skype traffic: When randomness plays with you, *ACM SIGCOMM Computer Communication Review*, vol. 37(4), pp. 37–48, 2007.

[5] K. Born, PSUDP: A passive approach to network-wide covert communications, presented at *Black Hat USA*, 2010.

[6] K. Born and D. Gustafson, Detecting DNS tunnels using character frequency analysis, *Proceedings of the Ninth Annual Security Conference*, 2010.

[7] M. Crotti, M. Dusi, F. Gringoli and L. Salgarelli, Detecting HTTP tunnels with statistical mechanisms, *Proceedings of the IEEE International Conference on Communications*, pp. 6162–6168, 2007.

[8] S. Davidoff and J. Ham, *Network Forensics: Tracking Hackers through Cyberspace*, Pearson Education, Upper Saddle River, New Jersey, 2012.

[9] C. Dietrich, C. Rossow, F. Freiling, H. Bos, M. van Steen and N. Pohlmann, On botnets that use DNS for command and control, *Proceedings of the Seventh European Conference on Computer Network Defense*, pp. 9–16, 2011.

[10] M. Dusi, M. Crotti, F. Gringoli and L. Salgarelli, Detection of encrypted tunnels across network boundaries, *Proceedings of the IEEE International Conference on Communications*, pp. 1738–1744, 2008.

[11] M. Dusi, M. Crotti, F. Gringoli and L. Salgarelli, Tunnel Hunter: Detecting application-layer tunnels with statistical fingerprinting, *Computer Networks*, vol. 53(1), pp. 81–97, 2009.

[12] E. Ekman and B. Andersson, Iodine Tunneling Protocol Documentation v502 (`github.com/yarrick/iodine`), 2014.

[13] G. Farnham, Detecting DNS Tunneling, InfoSec Reading Room, SANS Institute, Bethesda, Maryland, 2013.

[14] N. Hands, B. Yang and R. Hansen, A study on botnets utilizing DNS, *Proceedings of the Fourth Annual ACM Conference on Research in Information Technology*, pp. 23–28, 2015.

[15] E. Hjelmvik and W. John, Breaking and Improving Protocol Obfuscation, Technical Report No. 2010-05, Department of Computer Science and Engineering, Chalmers University of Technology, Goteborg, Sweden, 2010.

[16] I. Homem, TunnelStatsTests (`github.com/irvinhomem/TunnelStatsTests`), 2016.

[17] Mandiant, M-Trends 2014 Annual Threat Report: Beyond the Breach, Alexandria, Virginia, 2014.

[18] OpenDNS, OpenDNS Security Talk: The Role of DNS in Botnet Command and Control, San Francisco, California, 2011.

[19] O. Santos, *Network Security with NetFlow and IPFIX: Big Data Analytics for Information Security*, Cisco Press, Indianapolis, Indiana, 2016.

[20] D. Song, D. Wagner and X. Tian, Timing analysis of keystrokes and timing attacks on SSH, *Proceedings of the Tenth USENIX Security Symposium*, article no. 25, 2001.

[21] M. Stevanovic, J. Pedersen, A. D'Alconzo, S. Ruehrup and A. Berger, On the ground truth problem of malicious DNS traffic analysis, *Computers and Security*, vol. 55, pp. 142–158, 2015.

[22] I. Valenzuela, Game Changer: Identifying and defending against data exfiltration attempts, presented at the *SANS Cyber Defense Summit*, 2015.

[23] K. Xu, P. Butler, S. Saha and D. Yao, DNS for massive-scale command and control, *IEEE Transactions on Dependable and Secure Computing*, vol. 10(3), pp. 143–153, 2013.