Peer Based Tracking using Multi-Tuple Indexing for Network Traffic Analysis and Malware Detection

Matthew Hagan, BooJoong Kang, Kieran McLaughlin, Sakir Sezer,
Centre for Secure Information Systems (CSIT), Queens University Belfast
Belfast, United Kingdom
m.hagan, b.kang, k.mclaughlin, s.sezer@qub.ac.uk

***Abstract*—Traditional firewalls, Intrusion Detection Systems(IDS) and network analytics tools extensively use the 'flow' connection concept, consisting of five 'tuples' of source and destination IP, ports and protocol type, for classification and management of network activities. By analysing flows, information can be obtained from TCP/IP fields and packet content to give an understanding of what is being transferred within a single connection. As networks have evolved to incorporate more connections and greater bandwidth, particularly from "always on" IoT devices and video and data streaming, so too have malicious network threats, whose communication methods have increased in sophistication. As a result, the concept of the 5 tuple flow in isolation is unable to detect such threats and malicious behaviours. This is due to factors such as the length of time and data required to understand the network traffic behaviour, which cannot be accomplished by observing a single connection.**

To alleviate this issue, this paper proposes the use of additional, two tuple and single tuple flow types to associate multiple 5 tuple communications, with generated metadata used to profile individual connnection behaviour. This proposed approach enables advanced linking of different connections and behaviours, developing a clearer picture as to what network activities have been taking place over a prolonged period of time.

To demonstrate the capability of this approach, an expert system rule set has been developed to detect the presence of a multi-peered ZeuS botnet, which communicates by making multiple connections with multiple hosts, thus undetectable to standard IDS systems observing 5 tuple flow types in isolation. Finally, as the solution is rule based, this implementation operates in realtime and does not require post-processing and analytics of other research solutions. This paper aims to demonstrate possible applications for next generation firewalls and methods to acquire additional information from network traffic.

***Index Terms*—5-tuple flow tables, Zeus botnet, Network Behavioural detection, Next generation firewall**

I. Introduction

Within established network traffic accounting, analysis tools and intrusion detection systems(IDSs), the 5 tuple classification, consisting of source and destination IP addresses, ports and protocol, is extensively used for defining and isolating connections. Accounting methods such as Netflow [1] make use of the 5 tuples and 2 additional minor parameters (TOS and interface). The Linux connection tracking mechanism is an example of how the 5 tuples can be used to allow inbound traffic from host originated connections, but prevent new connections on the same ports and IP to the host. Modern IDS systems such as SNORT make use of the 5-tuple method to account for connections.

With the advent of greater bandwidth availability, always-on IoT devices and increased data transmission from mobile and standard computing devices, the ability to account for traffic has become more difficult, requiring analysis over longer periods of time. Network based malicious threats have become ever more sophisticated with detection having shifted from port number to application layer *Deep packet inspection* (DPI). Malware has been increasing their sophistication to disguise their activities, using obfuscation, encryption and peer to peer communication, meaning that more advanced detection methods are necessary.

This paper proposes an enhancement to IDS methods used to detect malware by introducing and simultaneously using 5, 2 and single tuple flows to associate different connections to the same or different hosts, where solely under 5 tuple flows, they could not be associated. By generating additional metadata based flow behaviour, better informed decisions can be made as to the purpose of greater quantities of traffic.

This contribution will study existing research that goes beyond observing 5 tuple based traits, using methods such as machine learning and other statistical measurements, to derive information that would have been inaccessible within a standard IDS. This work will then demonstrate a proposed approach which maintains the features and real time nature of existing IDS systems, but adds ability to extend them to detect new features spanning multiple connections within an expert system ruleset approach. The aim of this work is to indicate possible directions for rule based applications that provide realtime functionality to the user.

II. Related-work

The research community has been looking at a number of novel network based detection and analysis methods. These methods also go beyond use of the 5 tuple connection classification in order to gain additional behavioural information.

Narang *et al.* [2] studied conversational based data between IP addresses only, thus using a 2-tuple approach rather than 5-tuple. By studying conversations between IP addresses over longer periods of time, they found it was possible to perform statistical analysis of conversations based on interarrival times, packet size and duration of conversation prior to specified timeout. This work was demonstrated successfully against peer to peer bots. This approach does not consider other aspects such as packet content, nor empty packets. Being

978-1-5386-7493-2/18/$31.00

a machine learning based approach, this method required extensive amounts of training data.

Azab *et al.* [3] studied the application of machine learning to ZeuS botnet detection. Using their optimised framework and training with data from ZeuS v1, they were able to achieve a 0.667 recall result using their classifier and 0.556 recall using a standard machine learning framework against other ZeuS variants.

Zhuang *et al.* [4] further studied Peer to peer botnets, observing community based behaviour and isolating communications with bot masters within a peered connection environment. Botnet based activity was located by analysing for community activities between different hosts, before analysing flow statistical features, destination IP diversity, mutual peer contact and community structure. This research required availability of multiple connected hosts within a network.

Hjelmvik *et al.* [5] introduced the concept of protocol fingerprinting by observing statistically measurable attributes within the connection dialogue. Rather than looking at the packet payload using signatures, extracted attributes such as packet length can be statistically analysed to gather information about the protocol being used, where DPI cannot, due to obfuscation. This work aimed to assist in locating peer to peer file sharing protocols such as bittorrent, as well as tunnelling applications, which may make use of obfuscation to obscure themselves.

Herrmann *et al.* [6] performed work into website analysis, finding that by using statistical analysis of packet sizes within a connection dialogue, it was possible to determine specific websites being visited through encrypted VPNs and anonymising tools such as Tor.

Gu *et al.* [7] created the 'Bothunter' project with support from the US Army Research Office. This application observes various network activities to observe suspicious events such as inbound port scanning, exploit usage, malware downloading, outbound command and control server connections and outbound attacks. Bothunter builds a trail of evidence by logging and correlating these suspicious events. While this application considers detection of command and control activities, this feature may not be reliable to indicate an infection alone. Should bothunter be deployed with a host already infected, or if infected via a separate source, such as a USB pen drive, bothunter may fail to recognise botnet activity, as it may not be able to build a trail of suspicious evidence. This issue is further compounded by the fact that the application is closed source. It is therefore unclear precisely what detection methods are in use.

A similar application, created by Bothunters developers, is 'BotSniffer' [8]. This application performs group analysis of network traffic, to attempt to ascertain common events, such as command and control activity or highly similar network traffic activity. This research takes advantage of the likely situation that multiple hosts may exhibit highly similar network activity if infected by the same malware type. Detection is achieved by formulating a command and control "response crowd", consisting of multiple similar communications to a host, before determining whether the activities are malicious. However, a substantial issue with this application is that it requires multiple infected hosts. A network with a single infected host may therefore not flag a detection within this application.

Other studies have focused on protocols which may be used for malicious command and control communications within botnets. Strayer *et al.* [9] observed possibilities in isolating malicious IRC communications through classification using machine learning, followed by topological analysis to identify clusters of flows which can be linked to determine a common botnet controller. This was performed with the "Kaiten" IRC bot variant. While the researchers were successful in isolating malicious traffic from the majority of available traffic, this method could not uniquely detect malicious activity, thus manual analysis was still required. Furthermore, IRC traffic may more likely be filtered due to being an uncommon protocol with higher likelihood of abuse. Instead, malware may make use of more common protocols such at HTTP.

III. Proposed Approach

This section will detail the approach used to create the multiple indexes based on 5, 2 and single tuple connections. This will be followed by an example to highlight a relevant example which utilises the proposed multiple indexing feature, by using the ZeuS botnet malware. The ZeuS botnet has been used as the investigation tool and use case due to availability of the source code and ease to configure an isolated botnet for test purposes, as well as its implemented IDS evading features.

1) 5, 2 and single Tuple Implementation: The multi-tuple implementation is implemented through monitoring of network traffic utilising the *PCAP library*. A hashtable method is used maintain the information. When a packet is detected, firstly, the source and destination IP addresses, source and destination ports and protocol type will be hashed together and stored with a 5 tuple index reference generated. A separate 2 tuple index will be held, this time only for the hash of the source and destination IP pair. Finally, each IP address will be hashed individually and stored in the single tuple index table. For 5 and 2 tuple flows, the flow direction will be established to prevent indexing of packets travelling the opposite direction to the first packet received. Table I demonstrates the multiple tuple indexes created from a series of example connections.

The proposed approach involves implementing multiple indexes based on the following connection parameters, or tuples. This proposed approach allows for the following associations.

- *5-tuple*: The standard connection consisting of IP pair, source and destination ports and protocol type. Multiple packets from the same connection will be associated
- *2-tuple*: These indexes can associate different 5-tuple connections.
- *1-tuple*: Two exist for each packet. These can associate multiple 5-tuple and 2-tuple connections between different peers with same source or destination IP.

2) Metadata: Standard IDS features like regular expression based packet content analysis were implemented in the test application as rules. Upon detection of a rule, the application can log information against any of the 5 tuple, 2 tuple and IP

TABLE I
REPRESENTATION OF GENERATED 5, 2 AND SINGLE TUPLE INDEXES BASED ON CONNECTIONS MADE

SRC IP	DST IP	SRC Port	DST Port	5-T ID	2-T ID	1-T IDs
192.168.1.67	192.168.1.1	49152	80	1	1	1,2
192.168.1.1	192.168.1.67	80	49152	1	1	2,1
192.168.1.67	192.168.1.1	49153	80	2	1	1,2
192.168.1.1	192.168.1.67	80	49153	2	1	2,1
192.168.1.67	8.8.8.8	49154	53	3	2	1,3
8.8.8.8	192.168.1.67	53	49154	3	2	3,1
192.168.1.67	192.168.1.2	49156	80	4	3	1,4
192.168.1.2	192.168.1.67	80	49156	4	3	4,1
192.168.1.67	192.168.1.1	49157	80	5	1	1,2
192.168.1.1	192.168.1.67	49152	137	6	1	2,1

address indexes. For example, this may be the occurrence of a rule particular, or any other packet information, such as packet size, time, or application layer data. Finally, the application, when instructed, can check generated meta-data for a pattern. For example, rule matches on packets of a particular size, at regular intervals are relevant for detecting ZeuS traffic.

A. ZeuS Botnet Connection Features

The ZeuS malware client makes use of the HTTP protocol to communicate with the server. Network based malware, such as ZeuS, will often use known network protocols, like HTTP, to ease configuration of the server and lower the chance of the connection being blocked by a firewall. Within the HTTP connection, other techniques may be employed, such as obfuscating the data inside the payload, to thwart signature based DPI rules. In isolation, detection of obfuscated or encrypted data within an HTTP packet is insufficient to identify of a threat. The ZeuS client performs beacon communications based on a number of configurable settings within the malware builder application. These are:

1) "timer_config" Beaconing: The configuration file beaconing pattern consists of the client making a periodic HTTP GET request to the server, which hosts the configuration file. The client HTTP header closely resembles that of the Internet Explorer browser, with specific traits which are common to all tested Zeus variants. These include;

- The acceptance of any file type in the response (Accept: */*)
- Internet Explorer mimicking (User-Agent: Mozilla/4.0 (compatible; MSIE))
- Instruction not to cache the response (Cache-Control or Pragma: no-cache)

The HTTP response header to this request will contain 'Content-Type: application/octet-stream'. By default, this communication will occur on an hourly basis, but can be modified as an integer within the configuration, in minutes.

2) "timer_stats" Beaconing: A second form of beaconing uses HTTP POST requests to report information about the infected host back to the command and control server. The HTTP header of the POST request exhibits the same header traits as the 'timer_config' GET header. Secondary to this,

```
GET /ughjvh/config1.bin HTTP/1.1
Accept: */*
Connection: Close
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/4.0;
SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0)
Host: cont24x7host.org
Cache-Control: no-cache
```

Fig. 1. HTTP header from a typical "timer_config" GET request

the header definition "Content-Type: text/html" is present, but the HTTP payload consists of obfuscated or encrypted data. Furthermore, no encoding setting in present within the header, which would explain the presence of scrambled data. This anomaly may be detected by a regular expression through the lack of parentheses within the HTTP data payload. Within the 'timer_stats' beaconing pattern, the size of each POST request to the Zeus command and control server does not change.

```
POST /ughjvh/gate.php HTTP/1.1
Accept: */*
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/4.0;
SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0)
Host: cont24x7host.org
Content-Length: 376
Connection: Keep-Alive
Cache-Control: no-cache

.~).5/.....?.s....".Q1Q
p.F..m3...&........[.a.9......8.
[K..#. ./.dr~.........v...s.e>W,."..-....PZ../..RD.iA.....D...g.>...t..).2.......".uM.!
+.).URNF..bbi.JJ.bH.
..5.....Hie.....Hw....V...6.5+.x.....=p..+.
[..Dt.).I....<....;.........N.......2...)...'..+.$........yG.....9..nM..]Pzc.'..=&._o..9
$Z..2P.9.......r...g.5.l<h*.=.......7...........XD4~\.EH...0r.7P..{n.......U&...
```

Fig. 2. Timer_stats POST request header and encrypted payload from Zeus version 2.

```
HTTP/1.1
200 OK
Date: Tue, 06 Jun 2017 17:42:10 GMT
Server: Apache/2.2.24 (Unix) mod_ssl/2.2.24 OpenSSL/0.9.8e-fips-rhel5
mod_auth_passthrough/2.1 mod_bwlimited/1.4 FrontPage/5.0.2.2635 mod_jk/1.2.35
mod_fcgid/2.3.6
X-Powered-By: PHP/5.3.23
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html

40
.w..q.w.b...N.....<.}.|'].k..J........h.{.;..V
.Xm.WvN.P...z....
0
```

Fig. 3. Timer_stats response. Note the "Content-Type" and payload data mismatch.

B. Building Expert System Ruleset

The detection process for mentioned beacon patterns requires single traits to first be isolated, before their logged against the index in the form of a timestamp, associated to the 2-tuple flow. Regular expression is used to detect the malware traits described within the 5-tuple flow. A timestamp to the associated 2-tuple flow and system will check if a time gap exists, consistent to ZeuS's default for the specified option or for equidistant separation between three most recent instances. Should either of these results be positive, the trait will be logged against the 2-tuple index as being potentially ZeuS traffic. Finally, the 2-tuple index will be logged against the single tuple index of the source IP which originated the traffic. Should both elements of ZeuS connectivity, timer_config and

timer_stats, be detected with the same or another host, from that same source IP, it is considered, for the purpose of this expert system, that a positive ZeuS detection has been made. Algorithms 1 and 2 detail a sample of the method used to append data to the index upon detection of a suspicious trait.

Algorithm 1 Timer_config isolation and logging
Input: in
Output: out
Initialisation :
1: $regex\_step1$ = [Accept: */*]&[User-Agent: Mozilla/4.0 (compatible; MSIE]&[Cache-Control|Pragma: no-cache]
Incoming packet
2: **while** $recv\_packet$ **do**
3: **if** packet($regex\_step1$) = true **then**
4: logappend($2T\_index$->$timer\_config\_pkt\_timestamp$)
5: **end if**
6: **end while**
7: **goto** timer_config_patterncheck[$2T\_index$]

Algorithm 2 Timer_stats detection
Input: in
Output: out
Initialisation :
1: $regex\_step1$ = [Accept: */*]&[User-Agent: Mozilla/4.0 (compatible; MSIE]&[Cache-Control|Pragma: no-cache]
2: $regex\_step2$ = [Content-Type: text/html]&![Content-Encoding:]&![.*<.*>.*</.*>]
Incoming packet
3: **while** $recv\_packet$ **do**
4: **if** packet($regex\_step1$) = true **then**
5: log($5T\_index$->$zeus\_timer\_stats\_step1$)
6: **break**
7: **end if**
8: **if** packet($regex\_step2$) = true **and** ($5T\_index$->$zeus\_timer\_stats\_step1$) **then**
9: logappend($2T\_index$->$timer\_stats\_pkt\_timestamp$)
10: **end if**
11: **end while**
12: **goto** timer_stats_patterncheck[$2T\_index$]

In order to detect each of the beaconing patterns, a number of factors must be considered, concerning the connectivity of the infected host within network and the traffic passing through the network;

- A single beaconing pattern consists of short TCP connections between 2 IP addresses over a prolonged period of time.
- 'timer_stats' and 'timer_config' beacons may be to different servers from the infected host.
- Legitimate HTTP traffic may exhibit traits similar to Zeus
- In a working environment, a high volume of network communications may be present

Time	Source	Destination	Protocol	Length	Info
2435.733	192.168.1.67	[redacted]	HTTP	205	GET /ztest/cfg.bin HTTP/1.1
2435.736	[redacted]	192.168.1.67	HTTP/DL	809	Connect
2436.367	192.168.1.67	[redacted]	HTTP	506	POST /ztest/gate.php HTTP/1.1
2436.377	[redacted]	192.168.1.67	HTTP	346	HTTP/1.1 200 OK (text/html)
3636.408	192.168.1.67	[redacted]	HTTP	506	POST /ztest/gate.php HTTP/1.1
3636.419	[redacted]	192.168.1.67	HTTP	347	HTTP/1.1 200 OK (text/html)
4836.461	192.168.1.67	[redacted]	HTTP	506	POST /ztest/gate.php HTTP/1.1
4836.471	[redacted]	192.168.1.67	HTTP	347	HTTP/1.1 200 OK (text/html)
6035.805	192.168.1.67	[redacted]	HTTP	205	GET /ztest/cfg.bin HTTP/1.1
6035.808	[redacted]	192.168.1.67	HTTP/DL	809	Connect
6036.488	192.168.1.67	[redacted]	HTTP	506	POST /ztest/gate.php HTTP/1.1
6036.498	[redacted]	192.168.1.67	HTTP	346	HTTP/1.1 200 OK (text/html)
7236.537	192.168.1.67	[redacted]	HTTP	506	POST /ztest/gate.php HTTP/1.1
7236.549	[redacted]	192.168.1.67	HTTP	347	HTTP/1.1 200 OK (text/html)
8436.576	192.168.1.67	[redacted]	HTTP	506	POST /ztest/gate.php HTTP/1.1
8436.587	[redacted]	192.168.1.67	HTTP	347	HTTP/1.1 200 OK (text/html)

Fig. 4. The beaconing pattern is shown with default settings "timer_config 60" and "timer_stats 20"

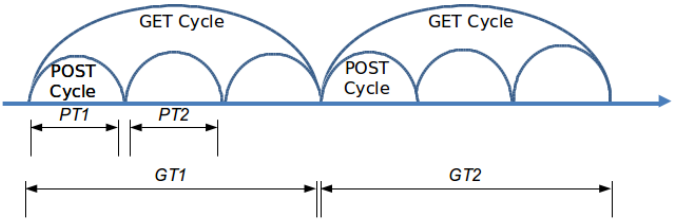

Fig. 5. A beaconing pattern has been detected when time deltas GT1 and GT2, between three suspicious GET request and responses, are equal, in combination with three suspicious POST requests and responses, detected with equidistant time deltas PT1 and PT2.

The traits which have been used to determine the presence of Zeus traffic have been defined through manual analysis of the network traffic created by multiple Zeus clients. The detection process uses regular expression to identify traits within the packet payload. Other information including 5-tuple connection information, time, IP address pairing and packet length are also necessary to locate traits within the traffic.

C. Implementation and Testing

To demonstrate the effectiveness of this proposed approach, the ZeuS botnet, version 2.0.8.9, configured to use multiple hosts, was used. The ITACA platform[10], modified to include 2-tuple and 1-tuple indexes along with the 5-tuple index, was used for the implementation.

The proposed multi-tuple indexing was used to maintain connection information. Network malware such as Zeus performs a large number TCP connections over an indefinite time between the client and server. These will appear as separate 5-tuple connections, each of which can trigger detection of the 'timer_config' or 'timer_stats', it is the temporal pattern between separate TCP connections that can trigger a beacon detection. For each beacon trait detection, the timestamp will be recorded. Equidistant time deltas between these trait will yield a positive beacon detection. The detection can take place independent of quantity of other connections, as they will be logged with a different index. This indexing method therefore allows scaling of the detection application to detect traits within network environments where many connections may

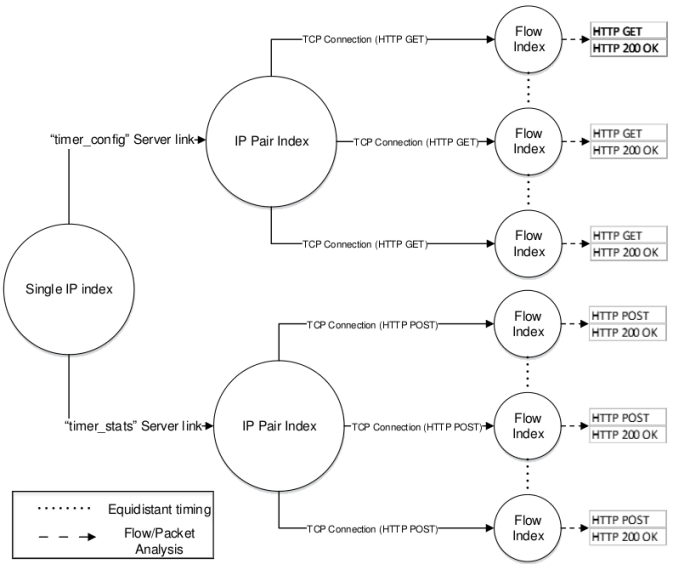

Fig. 6. 5-tuple connections are grouped using 2-tuple and single tuple indexes. Temporal analysis determines whether the activity corresponds to Zeus options before a decision is made on whether a single IP address is exhibiting Zeus client traffic.

be occurring, such as within an a corporate network gateway or ISP environment.

The experimental application design relies on measurable attributes contained within the packet data. Furthermore, the application itself must analyse packet tuple information to create three separate indexes, those being each individual IP address, each IP address pairing and each 5 tuple flow. This process is performed in conjunction with the isolation of Zeus HTTP header traits within the traffic. The Zeus beaconing detection method will make use of the packet content and index information provided and log any traits found. For each of the beaconing patterns, information about potential Zeus activity will be logged within the application's memory space. Only when all attributes of the trait detection type are equal will a second occurrence and its time be logged. For example, if a Zeus trait occurs on an IP pair index with a different POST packet size to a previous occurrence, this will be considered a separate Zeus trait. Such a phenomenon may indicate a change in Zeus' client configuration, multiple instances of the Zeus client running on a host with the same server specified, or two infected hosts who may be sharing a public IP address through the use of *Network Address Translation* (NAT). When three identical traits occur, temporal analysis can determine whether the connections are beacons.

IV. Conclusion and Future Development

This paper has investigated the concept of generating multiple tuple indexes from network traffic. By analysing traffic behaviours within these indexes, it is made possible to establish associations between disparate network connections, which would not be possible with a traditional IDS. This contribution has applied this concept, by creating a ruleset which can, with the generated behavioural metadata, infer the presence of malicious software within the network traffic, which would otherwise not be detectable.

This concept has been successfully implemented to detect a multi-peered ZeuS botnet. This has been performed by creating a expert system ruleset of packet content filters and temporal pattern analysis of detectable "atomic events", which are too minor in isolation, but when associated through multiple tuple connections become a strong indicator of malware presence in the network. The expert system described achieved successful detection of ZeuS botnet, independent of options specified and regardless of network size and quantity of traffic present.

The next stage of this research is to refine the algorithm based methodology to allow definition of "expert system" rulesets by the user. This may involve creation of a detection language to locate features, generate metadata to log traffic characteristics across multiple connections, to eventually detect complex network traffic events. Applications for this expert system detection method are not confined to malware threats, but it can be used for analysing many kinds of complex network behaviours, such as those mentioned in the *Related work* section, along with browsing, device usage analytics and fraud detection. The proposed method also provides a platform for generating additional data for use in other form of analysis including machine learning.

References

[1] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," *SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 4, pp. 323–336, Aug. 2002.

[2] P. Narang, S. Ray, C. Hota, and V. Venkatakrishnan, "Peershark: Detecting peer-to-peer botnets by tracking conversations," in *2014 IEEE Security and Privacy Workshops*, May 2014, pp. 108–115.

[3] A. Azab, M. Alazab, and M. Aiash, "Machine learning based botnet identification traffic," in *2016 IEEE Trustcom/BigDataSE/ISPA*, Aug 2016, pp. 1788–1794.

[4] D. Zhuang and J. M. Chang, "Peerhunter: Detecting peer-to-peer botnets through community behavior analysis," *CoRR*, vol. abs/1709.06440, 2017. [Online]. Available: http://arxiv.org/abs/1709.06440

[5] E. Hjelmvik and W. John, "Breaking and Improving Protocol Obfuscation," Chalmers University of Technology, Tech. Rep., 2010.

[6] D. Herrmann, R. Wendolsky, and H. Federrath, "Website fingerprinting: Attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier," in *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, ser. CCSW '09. New York, NY, USA: ACM, 2009, pp. 31–42. [Online]. Available: http://doi.acm.org/10.1145/1655008.1655013

[7] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee, "Bothunter: Detecting malware infection through ids-driven dialog correlation," in *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, ser. SS'07. Berkeley, CA, USA: USENIX Association, 2007, pp. 12:1–12:16. [Online]. Available: http://dl.acm.org/citation.cfm?id=1362903.1362915

[8] G. Gu, J. Zhang, and W. Lee, "Botsniffer: Detecting botnet command and control channels in network traffic," in *Proceedings of the Network and Distributed System Security Symposium, NDSS 2008, San Diego, California, USA, 10th February - 13th February 2008*, 2008.

[9] W. T. Strayer, R. Walsh, C. Livadas, and D. Lapsley, "Detecting botnets with tight command and control," in *Proceedings. 2006 31st IEEE Conference on Local Computer Networks*, Nov 2006, pp. 195–202.

[10] J. Hurley, A. Munoz, and S. Sezer, "Itaca: Flexible, scalable network analysis," in *2012 IEEE International Conference on Communications (ICC)*, June 2012, pp. 1069–1073.