Peer-to-Peer Botnet Detection Using NetFlow

Master Thesis

Connor Dillon
System and Network Engineering
University of Amsterdam

July 11, 2014

Abstract

Traditional botnets use a centralized communications architecture where all the bots connect to Command and Control (C&C) servers. These servers are the weak point of the botnet, as they are easy targets for take down and monitoring. Peer-to-peer (p2p) botnets have a distributed architecture, which make them more resilient. This research aims at the detection of individual p2p bots within a network perimeter. This is done by looking at the communications with their p2p overlay network. The NetFlow protocol is used to gain insight in all traffic within the network. A detection algorithm is proposed that can detect p2p malware in live NetFlow data. The algorithm is based on characteristics that separate malicious from benign p2p traffic, such as: traffic volume, packet symmetry and traffic patterns. These characteristics were identified by analyzing the behavior of different benign p2p applications and the Zeus p2p malware.

Contents

1 Introduction

Traditionally botnet administrators manage their bots through central Command and Control (C&C) servers. A central architecture has a number of disadvantages. The C&C servers can easily be identified and there are organizations that actively keep track of C&C servers on the Internet[1]. C&C servers of large botnets are often taken down by the police in order to dismantle the botnet. In response to this, some botnets now use a peer-to-peer (p2p) architecture for their communications. They use an overlay network to distribute commands and updates so there is no need for a central server, this makes them more robust[14]. Detecting p2p malware can be difficult as their behavior is similar to benign p2p software[15]. Detection becomes even harder when the communication is encrypted. In that case only meta-data such as IP address, port numbers and packets sizes can be used in detection algorithms[9].

This research focuses on the detection of advanced malware that use p2p communications. A detection algorithm is proposed that runs against traffic meta-data collected through the NetFlow protocol. Designing an effective detection algorithm with a low false positive rate is the main goal of this research. To achieve this goal, different benign p2p applications as well as p2p malware were analyzed to find differences in behavior that are observable through traffic flows. These key differences are the base for the detection algorithm.

1.1 Research question

Can p2p bots be detected effectively by analyzing traffic flow data?

1.2 Related research

There are a number of proposed systems that use NetFlow data to find p2p bots, such as BotGrep[11], BotTrack[8] and BotSuer[9]. BotGrep implements a detection algorithm that is based on the communication graph of botnet overlay networks. BotTrack creates a dependency graph between hosts in a network, it then runs the PageRank algorithm on this graph to find hosts communicate with each other, this identifies p2p networks. It then use information from a honeypot to find malicious p2p networks. BotSuer aims at detecting p2p bots based on their behavior, using NetFlow data from within a company network. They use machine learning techniques to differentiate between benign and malicious p2p traffic.

Yen and Reiter have researched the key differences in behavior between benign and malicious p2p software. Their dataset of malicious traffic consists of 24 hours of traffic from 13 Storm bots and 82 Nugache bots. They analyzed differences in traffic volume (avg. bytes/flow), peer churn (peers joining/leaving the network) and Human-driven vs. Machine-driven behavior.

This research is similar to that of Yen and Reiter, as determining the differences between benign and malicious p2p traffic is crucial for a good true/false positive rate. However, during this research a broader scope of p2p applications is analyzed as well as a different malware type. Also, this research focuses on working with live NetFlow data, instead of historical data. The results of this research are compared to those of previous research in section 4.5

2 Background

2.1 NetFlow and IPFIX

NetFlow is a protocol that allows routers and switches to provide a summary of the traffic that passes through the device. Originally it was a Cisco proprietary protocol, but many vendors implemented the protocol. Some under a slightly different name such as Jflow for Juniper and Rflow for Ericsson. Since NetFlow v9, the protocol is specified in an RFC, however, this is not an official IETF standard[3]. The IETF has standardized the successor of NetFlow as the IP Flow Information Export (IPFIX) protocol[5]. IPFIX was used for this research.

A flow is a summary of traffic sent between two endpoints. Information such as the source and destination ip/port and the number of packets and bytes are included in the flow. NetFlow v9 and IPFIX allow for the definition of a template in which different attributes can be selected to be included in the flow. Table 1 shows the template that was used for this research.

IPFIX field	Description
sourceIPv4Address	source IP
destinationIPv4Address	destination IP
sourceTransportPort	source port
destinationTransportPort	destination port
octetDeltaCount	up bytes
postOctetDeltaCount	down bytes
packetDeltaCount	up packets
postPacketDeltaCount	down packets

Table 1: IPFIX template

The router or switch keeps track of all flows in memory and updates them as packets pass through the device. A flow is exported after a timeout or when a TCP session is closed. It is then sent to an external device called a NetFlow collector for further processing. Configurable timeouts for flows are listed below, default values were used for this research.

- Inactive Timeout: after x seconds without new traffic (default 15 secs)
- Active Timeout: after x seconds, regardless of activity (default 60 secs)

The main benefit of IPFIX for this research is the built in support for bidirectional flows[4]. NetFlow exports unidirectional flows, this means that request and response will be separated. Many NetFlow tools automatically stitch the flows together based on source and destination ip + port. When using IPFIX, this is not needed.

NetFlow and IPFIX can utilize a lot of CPU power on network equipment that have not implemented the protocol in hardware. That is why it's possible to enable a sampling rate in which only 1 in every x packets is logged. For this research it is assumed that sampling is not enabled, which is acceptable for high end equipment that have implemented the protocol in hardware.

2.2 Zeus p2p malware

For this research the Zeus p2p malware (a.k.a. GameOver Zeus) was used for analysis. At the time, it was possible to obtain active samples of this malware via public sandboxes. The protocol that the Zeus p2p malware uses was analyzed by Andriesse and Bos[2]. The architecture of the botnet is shown in Figure 1.

The C&C layer of the botnet was taken down and its administrator was arrested on June 2nd 2014[12]. However, the p2p layer remains active and there is an ongoing effort to clean infected computers. Because of this, the malware is still relevant for this research.

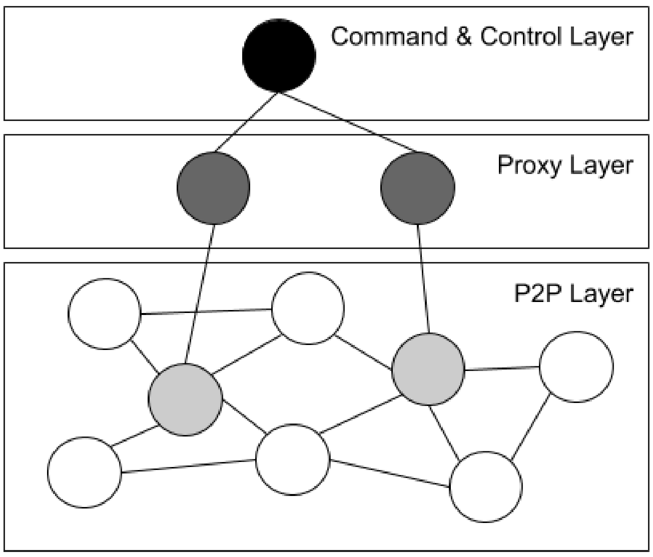

Figure 1: Zeus p2p Architecture

3 Data set

3.1 Benign traffic

Benign traffic was generated in a lab, specifically for this research. The data set includes web traffic generated by crawlers that simulate human-like browsing behavior. Web traffic also includes DNS traffic and data streams from YouTube and Internet radio. The data set also includes a lot of p2p traffic generated by different file sharing applications on different p2p networks. Table 2 gives an overview of the p2p applications. Because the data was generated in a lab, it was possible to categorize the traffic according to application.

Application	Protocol	Number of flows
Ares Galaxy	Ares	510
Bearshare	Gnutella	49026
eMule	eDonkey, Kademlia	3817
FrostWire	Bittorrent	78739
Imesh	IM2Net	26123
Shareaza	Mutiple protocols	30567
uTorrent	uTorrent transport protocol	238545

Table 2: Benign p2p data set

3.2 Malicious traffic

Three different binaries of the Zeus p2p malware were obtained via the public sandbox malwr.com. Each of these were installed on a Windows 7 machine in a controlled environment. All traffic generated by the infected machines was stored in PCAP format. The lab environment is behind a router that uses Network Address Translation (NAT). The Zeus p2p protocol doesn't add peers that are behind NAT[2]. So the monitored machines did not receive incoming requests from other peers. However, peers do respond to requests originating from peers behind a NAT. Table 3 gives an overview of the the malicious data set.

MD5 of Zeus binary	Capture time (mins)	Number of flows
fe529753c8ba280246afa574150046a8	120	153
b2ccfc9c9b1302ee46e277b92662414b	720	1900
db323438017bf1cc92a58a64474d923d	100	72

Table 3: Malicious p2p data set

4 Findings

This section shows the results of the analysis of the data set. All traffic was grouped by source IP + port and cut in to two hour chunks. The source IP + port combination was chosen because, UDP p2p applications initiate connections from a single source port, unlike TCP, which chooses a random source port. Thus this grouping covers all traffic from a p2p application. A chunk size of two hours was chosen, because the Zeus malware generates relatively little traffic. It must thus be monitored for a longer time before patterns and characteristics become visible. More than two hours would unnecessarily increase the amount of data that needs to be processed when working with flow data in real time.

4.1 P2P filter

To reduce the scope of traffic and the chance of false positives, the p2p traffic was first filtered out from other traffic. P2p applications can be identified by a high number of failed connections. The failed connections are the result of peers that are unreachable, either because they are no longer online or are behind a firewall/NAT. Sources with failed connections to different destinations are filtered out as p2p.

4.2 Traffic volume

In general, the main difference between Zeus and benign p2p applications is that Zeus generates very little traffic. Benign p2p applications are generally used for file sharing and thus exchange a huge amount of data, especially with multi-media files. In contrast Zeus only needs to maintain a list of active peers and occasionally download software updates or malicious payload and upload harvested information. This makes looking at traffic volume interesting for detection. However, when only looking at the total amount of traffic over a certain amount of time, it's likely that false positives would be triggered when a benign p2p application downloads a small file. Therefore it's better to look at statistics such as the average bytes per flow and the average bytes per packet. These details could be characteristics of the protocol and could thus be used for detection. Figure 2 shows the average bytes per flow for each application in the data set. The graph shows that the results for Zeus are within the same range as the FrostWire application. It is clear that relying on traffic volume alone for detection would result in false positives.

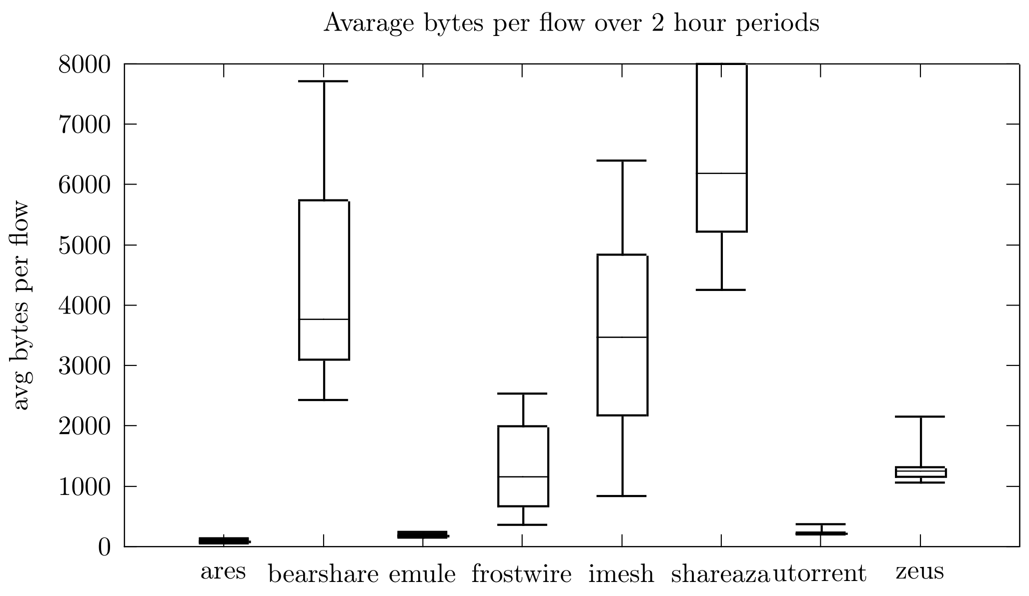

Figure 2: Traffic volume

4.3 Packet symmetry

Packet symmetry is the relation between the outgoing and incoming packets. Analyzing packet symmetry has shown to be effective for detection high volume DDoS attacks, as these attacks generate a lot of incoming traffic without outgoing traffic[7]. But it can also be useful for detecting malicious traffic in general[10]. The packet ratio is calculated by dividing the outgoing packets by the incoming packets. Protocols that rely on TCP for transmitting packets, have a packet ratio close to 1, as packets need to be acknowledged. Packet ratios for UDP protocols can be more varying, because UDP does not provide an acknowledgement mechanism. However, most benign protocols that use UDP implement their own mechanism for acknowledgements to ensure reliability.

Figure 3 shows the measured packet ratio for each application in the data set. Most of the applications have a ratio that is between 1.4 and 1.8. A high ratio for p2p applications can be explained by the high number of failed connections, as these result in outgoing packets without incoming packets. Zeus has a relatively low amount of failed connections which results in a lower packet ratio. Zeus also does not implement an acknowledgement mechanism. So when it requests data from other peers, it receives a large amount of packets with only a single outgoing request packet. This greatly reduces the packet ratio and separates it from benign p2p applications. The packet ratio should be balanced out by requests from other peers, because those will result in more outgoing packets. But because bidirectional flows are used, it should be clear which side initiated the flow. This allows one to focus only on flows initiated from within the network, to expose Zeus' low packet ratio. Unfortunately this was not verified during this research, because the lab environment was behind NAT and there were no incoming requests.

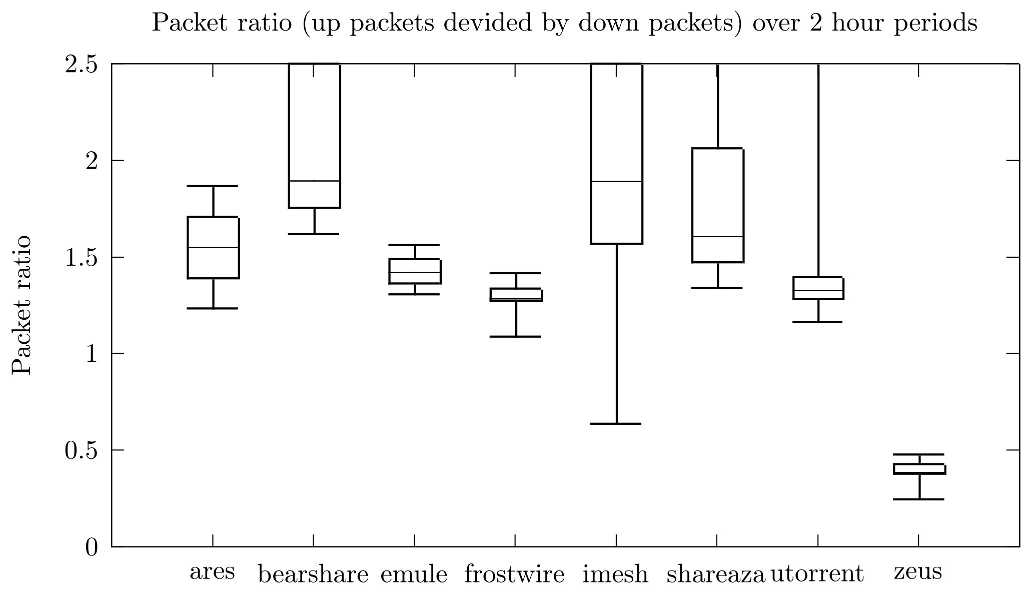

Figure 3: Packet symmetry

4.4 Traffic pattern

The Zeus malware has a control loop that periodically wakes up the bot to contact its peers and query them for their current configuration. Previous research has shown that the loop repeats every 30 minutes[2]. However, the malware that was observed during this research repeats itself every 20 minutes. It is possible that newer versions of the malware have a shorter cycle. In between each cycle there is no communication, this results in a clearly visible traffic pattern. Figure 4 shows the outgoing measured over five minute periods. Note that activities may vary in duration, but there is always 20 minutes between the last packet of a cycle and the first packet of the next cycle. Benign p2p applications do not show such a traffic pattern. When uploading or downloading files, the traffic appears random. Some benign p2p applications do periodically send keep-alive messages to other peers, even when not down/uploading files. However, these are generally sent much more frequent[13]. Thus Zeus' traffic pattern makes it stand out from other p2p applications.

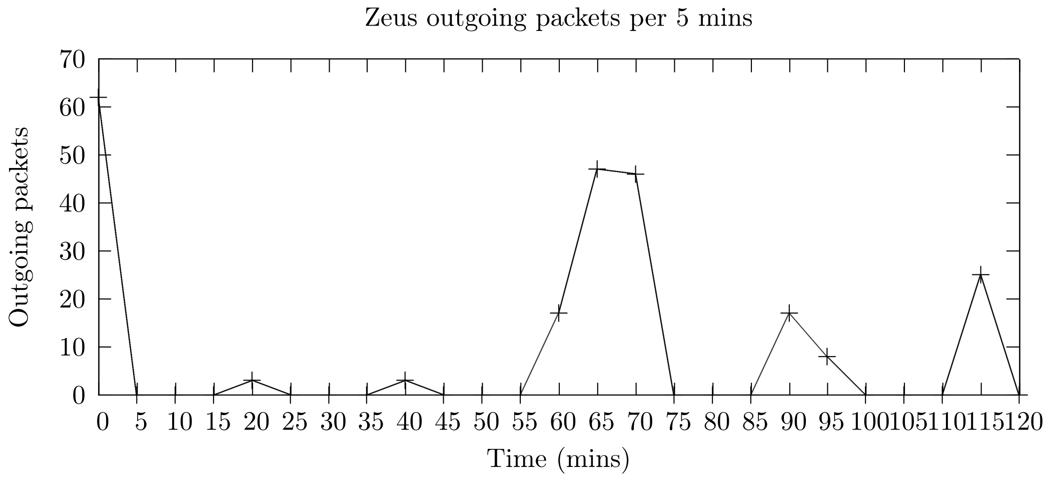

Figure 4: Traffic pattern

4.5 Compared to previous research

During this research, only the Zeus p2p malware was analyzed. However, there are many other malware that have a p2p architecture. In this section, the findings of this research are compared to that of previous research that have analyzed other p2p malware. As mentioned in section 1.2, Yen and Reiter analyzed traffic volume, peer churn and machine-like behavior of the Nugache and Storm malware. While their approach in general was different, the results are similar to that of this research. They also separate p2p traffic from non p2p traffic based on a high failed connection rate. Detection based on traffic volume (bytes per flow) also results in a high false positive rate. They analyzed the number of peers joining and leaving the overlay network (peer churn). Storm and Nugache have a relatively low peer churn, also resulting in a lower number of failed connections. While peer churn was not specifically a focus during this research, similar behavior was also observed for Zeus. Finally Storm and Nugache were analyzed for machine-driven traffic patterns. It turns out that the malware traffic patterns are less random than those of benign applications, that are operated by humans. This is also the case for Zeus, which has a very predictable traffic pattern.

Kheir and Wolley used machine learning techniques to filter malicious from benign p2p traffic. They obtained samples of seven different p2p malware types. Their first step was also to filter out p2p from non p2p traffic. They use different methods to do so, including looking at DNS requests. P2p applications generally don't need DNS, because they use the overlay network for finding addresses, thus a lack of DNS requests can be used for identifying p2p applications. They define three categories of training features or their learning algorithm: time-based, space-based and flow size-based features. Time-based feature include traffic patterns comparable to what was observed during this research. It turns out that all of the malware types they analyzed had a control loop that repeats itself every x minutes. Space-based features show how peers interact with each other. Malware generally contacts less peers than benign applications. Flow size-based features include statistics such as the average bytes per flow. While their approach is very different than that of this research, there are some similarities in what characteristics are used for detection, such as traffic patterns and traffic volume.

5 Proof of concept

5.1 Detection algorithm

Based on the results of the analysis of the data set the following detection algorithm is proposed. The details are explained section 5.2.

1. Group flows by source IP + port
2. Filter p2p traffic: sources with more than 4 failed connections to different hosts are considered p2p
3. Detect zeus based on either:
   a) Packet ratio: the sum of up packets divided by the sum of down packets is less than 0.4
   b) Traffic pattern: there are 3 periods of more than 5 minutes apart, with standard deviation of less than 150

5.2 Implementation

The proposed detection algorithm was implemented in a custom NetFlow collector written in Python. The NetFlow collector deals with live data and keeps a table of flows, grouped by source IP + port, in memory. Flows are expired after a period of two hours to free memory. When a new flow is sent to the controller, it's added to the set of matching flows based on source IP + port, the set is then passed to the p2p filter. If a set is marked as p2p, it's passed to the Zeus detection functions. The details of these functions is shown below. The full implementation is available on GitHub[6].

p2p filter

Get the set of unique destination IPs that have a failed flow. A failed flow is a flow with 1 or more outgoing packets and 0 incoming packets. If the set contains more than 3 IPs, the source is considered p2p.

```
def p2p_detect(flows):
    unreachables = set(
        flow.dst_ip
        for flow in flows
        if flow.up_packets > 0 and flow.down_packets == 0
    )

    if len(unreachables) > 3:
        return True
```

Zeus packet symmetry detect

Calculate packet ratio by dividing the sum of all outgoing packets by the sum of all incoming packets. If the ratio is less than 0.4, the source is considered Zeus.

```
def zeus_ratio_detect(flows):
    up = sum(flow.up_packets for flow in flows)
    down = sum(flow.down_packets for flow in flows)

    if up / down < 0.4:
        return True
```

Zeus traffic pattern detect

Get the set of timestamps. For each timestamp, if the difference with the previous timestamp is more than 300 seconds, add the difference to the set of intervals. If the set of intervals has more than 3 entries and the standard deviation of the set is less than 150, the source is considered Zeus.

```
def zeus_pattern_detect(flows):
    timestamps = list(flow.timestamp for flow in flows)
    intervals = list()

    previous_timestamp = timestamps[0]

    for timestamp in timestamps:
        if timestamp - previous_timestamp > 300:
            intervals.append(timestamp - previous_timestamp)

        previous_timestamp = timestamp

    if len(intervals) > 3:
        if stdev(intervals) < 150:
            return True
```

5.3 Results

All the traffic in the data set was replayed and the exported flows were sent to the implemented NetFlow collector in real time. The result was a 100% true positive rate and a 0% false positive rate. Of course, because of the limited data set, no statements can be made about what the results would be with real user data. The detection algorithm thus needs to be tested with more and different data, as explained in Section 7.

6 Conclusion

This research proposes a solution for detecting p2p malware in live NetFlow data, with a fully behavioral approach. It is shown that the Zeus p2p malware behaves differently then benign p2p applications, and this difference is visible in traffic flows. The different behavior can be explained by the different purposes that benign and malicious p2p applications serve. Benign applications are generally used for transferring large files over the Internet. A file transfer is initiated by a human operating the application. These factors cause traffic generated by such an application to appear random. In contrast the Zeus p2p malware only transfers small amounts of data and does so at a repeated interval. This results in an identifiable traffic pattern, which is used for detection. Zeus implements a custom protocol for p2p communications. This protocol has identifiable characteristics, such as that it does not implement an acknowledgement mechanism. This is observable when analyzing packet symmetry and makes it stand out from benign p2p applications.

Because detection is based on behavior, malware could adapt its behavior to more closely resemble that of benign p2p applications. Their control loop could be modified to initiate communications at random intervals. This would complicate identifying traffic patterns. Should detection based on behavior prove to be a real threat to botnets, it's likely that they will indeed adapt their communications patterns. However, fully imitating benign p2p applications also means transferring large amounts of data. This at least makes it easier to spot, it will just be harder to differentiate between benign and malicious p2p traffic. It remains to be seen how well behavioral detection works against malware that purposely tries to avoid it.

7 Future work

All data used for this research was generated specifically for that purpose. This allowed for easy identification of individual p2p applications. However, it's not the best representation of real world data. Therefore, it would be very useful to test the detection algorithm against real wold data. Both benign for false positives and malicious for true positives. It would also be interesting to test with data generated by malware other than Zeus, to see if the same techniques can be applied for detecting them as well. Also, the lab environment used for this research was behind NAT, which means that there were no incoming requests. It would be interesting to see how the malware behaves when publicly accessible.

References

[1] abuse.ch. Zeus tracker. https://zeustracker.abuse.ch/.

[2] Dennis Andriesse and Herbert Bos. An analysis of the zeus peer-to-peer protocol. Technical Report IR-CS-74, VU University Amsterdam, April 2014.

[3] B. Claise. Cisco systems netflow services export version 9, October 2004. RFC 3954.

[4] B. Claise and E. Boschi. Bidirectional flow export using ip flow information export (ipfix), January 2008. RFC 5103.

[5] B. Claise, B. Trammell, and P. Aitken. Specification of the ip flow information export (ipfix) protocol for the exchange of ip traffic flow information, September 2013. RFC 5101.

[6] Connor Dillon. Peer-to-peer malware detection using netflow. https://github.com/ConnorDillon/p2pmalwaredetect.

[7] Connor Dillon and Mike Berkelaar. Openflow (d)dos mitigation, 2014. http://rp.delaat.net/2013-2014/p42/report.pdf.

[8] Jerome Francois, Shaonan Wang, Radu State, and Thomas Engel. Bottrack: Tracking botnets using netflow and pagerank. In *NETWORKING 2011*, volume 6640 of *Lecture Notes in Computer Science*, pages 1–14. Springer Berlin Heidelberg, 2011.

[9] Nizar Kheir and Chirine Wolley. Botsuer: Suing stealthy p2p bots in network traffic through netflow analysis. In *Cryptology and Network Security*, volume 8257 of *Lecture Notes in Computer Science*, pages 162–178. Springer International Publishing, 2013.

[10] Christian Kreibich, Andrew Warfield, Jon Crowcroft, Steven Hand, and Ian Pratt. Using packet symmetry to curtail malicious traffic. *USENIX Security '10*, 2005.

[11] Shishir Nagaraja, Prateek Mittal, Chi-Yao Hong, Matthew Caesar, and Nikita Borisov. Botgrep: Finding p2p bots with structured graph analysis. *USENIX Security '10*, 2010.

[12] U.S. Department of Justice. U.s. leads multi-national action against gameover zeus botnet and cryptolocker ransomware, charges botnet administrator, June 2014. `http://www.fbi.gov/news/pressrel/press-releases/`.

[13] Richard Price, Peter TiÅĹo, and Georgios Theodoropoulos. Still alive: Extending keep-alive intervals in p2p overlay networks. *Mobile Networks and Applications*, 17(3):378–394, 2012.

[14] E. Van Ruitenbeek and W.H. Sanders. Modeling peer-to-peer botnets. In *Quantitative Evaluation of Systems, 2008. QEST '08. Fifth International Conference on*, pages 307–316, 09 2008.

[15] Ting-Fang Yen and Michael K. Reiter. Are your hosts trading or plotting? telling p2p file-sharing and bots apart. In *Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference*, pages 241–252, 06 2010.