Use of NetFlow/IPFIX Botnet Detection Tools to Determine Placement for Autonomous VMs

Razvan-Ioan Dinita, razvan.dinita@anglia.ac.uk
Adrian Winckles, adrian.winckles@anglia.ac.uk
George Wilson, george.wilson@anglia.ac.uk
Anglia Ruskin University, Cambridge, UK

ABSTRACT

This paper describes a novel method of autonomously detecting malicious Botnet behaviour within a Cloud datacentre, while at the same time managing Virtual Machine (VM) placement in accordance to its findings, and it presents its implementation with the Scala programming language.

A key feature of this method, using output from NetFlow/IPFIX, both of which are capable of producing detailed network traffic logs, is its capability of detecting unusual Client behaviour through the analysis of individual data packet information. It has been implemented as a module of an Autonomous Management Distributed System (AMDS) presented in [Dinita, R. I. et al., 2013], giving it direct access to all the VMs and Hypervisors on the Cloud network.

Another key feature is that it can have an immediate and effective impact on network security in a Botnet attack context by issuing lockout commands to every networked VM through the AMDS. It possesses the ability to intelligently control VMWare vSphere local instances based on analysis of collected data and predefined parameters. vSphere in turn, once it receives commands from the AMDS, proceeds to issue instructions to multiple locally monitored ESXi severs in order to ensure continuous security.

A proof of concept has been developed and is currently running successfully on the authors' test bed.

Keywords: botnet, software, security, detection, autonomous

1. INTRODUCTION

This paper sets out to describe one potential design and implementation of a Botnet Detection software module. This module is aimed at analysing portions of live network traffic in an attempt to detect unauthorised and malicious activity within a Cloud Computing infrastructure. The work presented is building upon previous research undertaken by the authors into autonomous datacentre management and supervision software by bolting the aforementioned botnet module onto the Autonomous Management Distributed System (AMDS) [1].

The botnet detection algorithm used is based on the access pattern-based detection method [2], which looks at client behaviour to differentiate a legitimate client from an infected one. This is based on the assumption that a compromised client will, in most cases, operate in the same manner as the bot that originally infected it. All network data is compared against existing, legitimate client profiles, continuously refined over time, in an attempt to spot and block the compromised ones.

It also makes use of an anomaly-based heuristic algorithm based on the work carried out by [3]. The algorithm comprises of two main detection components: an Internet Relay Chat (IRC) mesh, and a Transmission Control Protocol (TCP) scan detection heuristic. The author has made use only of the TCP heuristic component and adapted it as necessary into the AMDS botnet detection module.

1.1. Botnets Background

The term *bot* is short for *robot*. Criminals distribute malicious software (also known as malware) that can turn your computer into a bot (also known as a zombie). When this occurs, your computer can perform automated tasks over the Internet, without you knowing it. Bots are typically used to infect large numbers of computers. These computers form a network, or a botnet, and are used to send out spam email messages, spread viruses, attack computers and servers, and commit other kinds of crime and fraud.

Peer-2-Peer bots are split into two categories: Masters and Slaves. Each of these uses secure channels to pass information between one another containing a command and details on the originating control module. This allows the Bot Masters to issue commands to the Bot Slaves. Bot masters will use root-kits and anti-VM static- DLL/binary code [4] along with secure channels in order to avoid detection. Once the control channel is established, it is used for communications that employ two different channel operation architectures, the *centralised architecture* and the *decentralised architecture*.

The *centralised architecture* has been used in the past by IRC-based (Internet Relay Chat, communication protocol) botnets, which used IRC servers to issue commands to all malware-infected machines. This mode has many different variations [5] e.g. the final destination could contain a text document with a list of static IP addresses and lists of URLs, so that flexible IP addresses can utilised. The *decentralised architecture*, on the other hand, is a newer communication architecture that enables infected hosts to exchange information via distributed networks such as Peer-2-Peer. This method may reduce the rate of firewall and antivirus detection [5].

Regardless of the type of architecture, there are two types of command and control channels, the *Persistent Channel* and the *Periodic Channel*. The *Persistent Channel* maintains a direct connection with the Bot Master. This connection type is normally employed by IRC bots, however, it is increasingly becoming obsolete due to periodic channels. The *Periodic Channel*, on the other hand, connects to the Bot Master periodically in order to avoid detection. Typically, the destination has had no prior communication with the host. This is normally used to throw network security devices and probes off track.

1.2. AMDS / Botnet Detection Module Network Logical Layout

The Botnet Detection module is a critical part of the AMDS [1] in the context of malicious behaviour detection. It is charged with storing and analysing output network data flows, as produced by the NetFlow/IPFIX setup, in an attempt to internally construct and continuously iterate on a generic botnet detection model. It hopes to achieves this through always going back and forth between new, unidentified flows and past, already analysed flows and comparing specific elements from each with aim to brand the new flows as either healthy of infected.

As it can be seen in *Figure 1* below, the AMDS is deployed on a Virtual Machine Instance (VMI) on the authors' Cloud Test Bed, which has been created based on a VM Template through the VMWare vSphere Client and connected to the internal Cloud Network.

At a logical overview level, the Cloud Infrastructure is composed of a Cisco ASA Router (Adaptive Security Appliance), three VLANs (Virtual Local Area Network), the vSphere Client, the ESXi Server, a group of ILOs (Integrated Lights Out), and the AMDS. Each of these components is underpinned by a series of physical network cables, switches, and routers that facilitate interconnectivity between them. The ESXi Server is comprised of multiple independent VMs interconnected by the three VLANs.

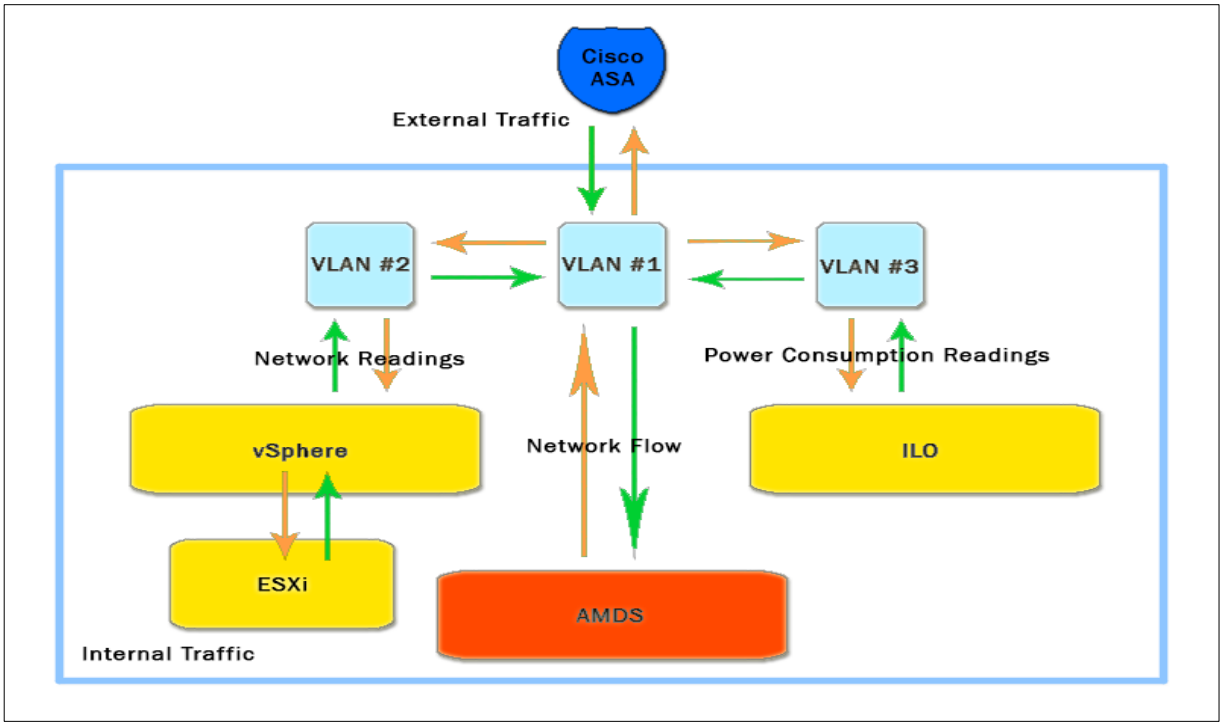

Figure 1 – AMDS/Botnet Module Network Logical Layout

The network operational flow is expressed through two different arrow colours in *Figure 1*, both relevant to the AMDS: *GREEN* reflects network traffic flowing towards the AMDS, while *ORANGE* reflects network traffic flowing from the AMDS towards all other infrastructure components.

Green traffic is composed of data the AMDS requires when performing the infrastructure logical analysis from the points of view of processor loads, power consumption, and network flow, from the following sources:

I. Processor load data is retrieved from the vSphere Client;

II. Power consumption data is retrieved from the ILO group; and
III. Network flow data is obtained from the various switches and routers spread across the infrastructure.

Orange traffic is composed of commands the AMDS issues post-analysis. This includes:
I. VM moving commands to the ESXi Servers through vSphere;
II. Physical server shut-down / start-up commands to various ESXi Servers through vSphere;
III. Network flow restriction requests to the Cisco ASA router; and
IV. Load balancing requests across the infrastructure to other AMDS running instances.

Even though all infrastructure components are located within the same physical network, the VLANs allow splitting it up into smaller logical groups which can be more easily maintained and controlled. Each VLAN is created and maintained by the vSphere Client and only has direct access to the logical component in its immediate vicinity. This allows for the formation of highly compartmented and self-contained/-managed logical groups.

A direct consequence of the information presented above, the Botnet Detection module, which is an inherent part of the AMDS, has direct access to all VLANs. This makes it capable of interfering with regular network data flow based on its post-analysis results. This gives it the power to control every aspect of a physical infrastructure through controlling its logical groups. Once AMDS collects several hours worth of data, it is then capable of issuing commands to vSphere, which in turn will forward these, as needed, to other Components under its control.

2. NETFLOW / IPFIX BACKGROUND

2.1. NetFlow Based Traffic Monitoring

NetFlow has been developed in-house initially to provide improved packet switching capabilities to some of their network devices in 1990. It then has steadily grown into a complex network operational analysis[1] tool.

NetFlow is capable of providing maximum network awareness, and, if used as part of a complex cloud infrastructure, it is able of giving insight into the different types of data packets flowing through the network at any given time. It makes use of 5 to 7 IP packet attributes to generate traffic flow reports, which can be later on analysed by system administrators.

NetFlow utilises the following information in its traffic flow caches: packet size, IP addresses and ports for both packet source and destination, class of service, device interface, and protocol type. In addition, it also records flow timestamps, next hop IP address, subnet mask, and TCP flags. All of these flow parameters aim to provide a holistic network view used in many different scenarios, of which the one of interest is detecting malicious behaviour in a cloud network infrastructure.

As it can be seen in *Figure 2*, upon enabling a device to utilise NetFlow, it then proceeds with recording all traffic coming into the network. These recordings, after undergoing NetFlow processing, are stored in a database entitled NetFlow Cache in the form of a table, each row containing one data flow cache. These flows have a shorter or longer lifetime as determined by the flow timestamp, depending on whether traffic has stopped flowing, a FIN signal has been received indicating the end of the flow, or they have exceeded their pre-set lifetime.

The *NetFlow Enabled Device* (*Figure 2*) is, in this case, the NetFlow Exporter. Its primary concern is capturing traffic data and transforming it into flows. This data is then transferred over to a *NetFlow*

[1] http://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.html

Collector Database Flow Cache (*Figure 3*) where it is analysed and converted into meaningful reports that present an overview of the processed network data traffic.

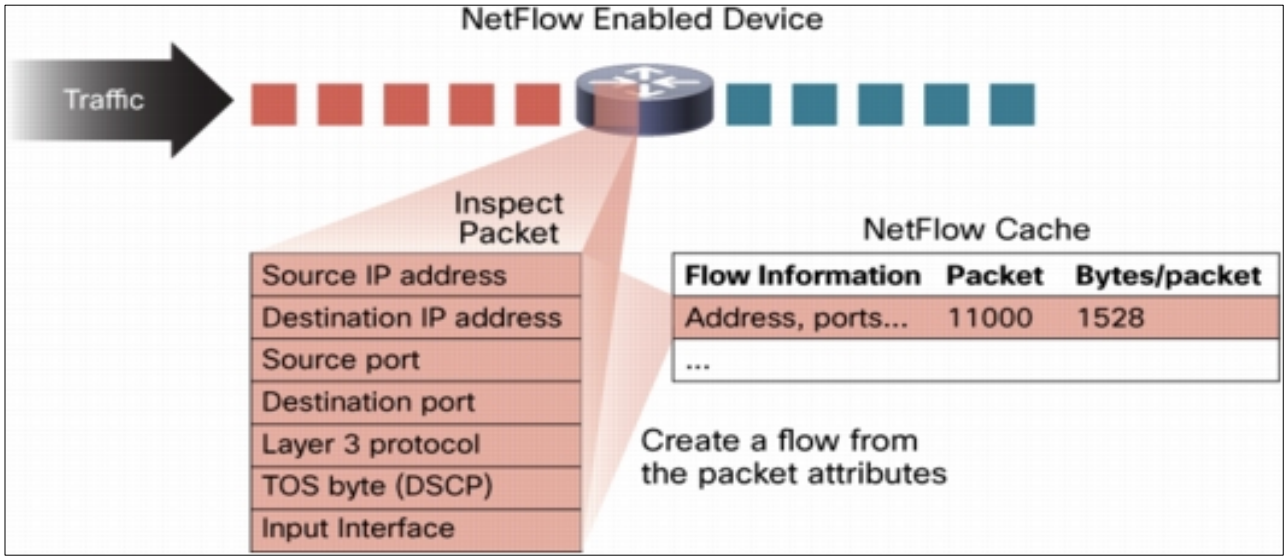

Figure 2 - NetFlow Flow Cache Generation[2]

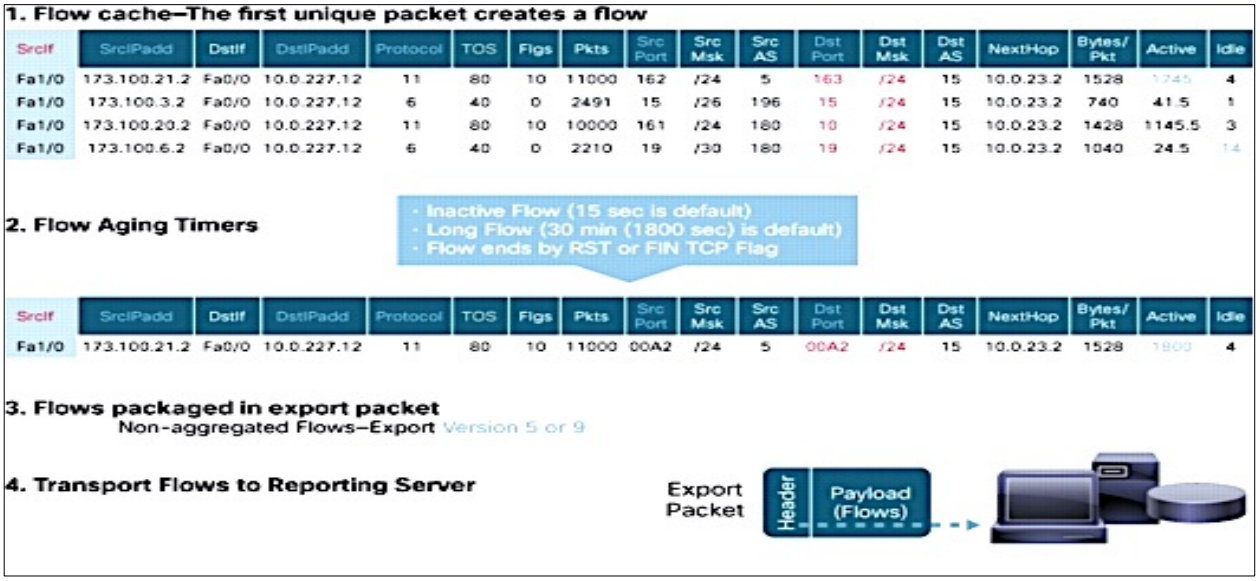

Figure 3 - Example NetFlow Cache[3]

[2] http://www.cisco.com/c/dam/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.doc/_jcr_content/renditions/prod_white_paper0900aecd80406232-1.jpg

[3] http://www.cisco.com/c/dam/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.doc/_jcr_content/renditions/prod_white_paper0900aecd80406232-2.jpg

As it can be seen in *Figure 3*, a typical NetFlow Cache database table contains many different columns generated based on parameters both taken from the packet itself and created by NetFlow in its flow generation. It keeps track of both new and old flows by separating them into two different tables: one for active flows, another for ended flows (either because they had exceeded their pre-set lifetime or the traffic had stopped).

2.2. IPFIX Based Traffic Monitoring

On the other hand, the IP Flow Information Export (IPFIX) format is an Internet Engineering Task Force (IETF) protocol designed to be a universally accepted standard for capturing traffic flow through network devices. IPFIX is responsible for defining the format of captured flows as well as detailing the flow method of transfer between the exporter and collector.

IPFIX, similarly to NetFlow, is capable of breaking down data packets within the observed network traffic according to their attributes. As such, the flow metering process offers several configuration possibilities, which can be used to narrow down the packets included in the flow. There can be defined different sampling and filtering functions, each of which is dealing with a certain aspect of data selection.

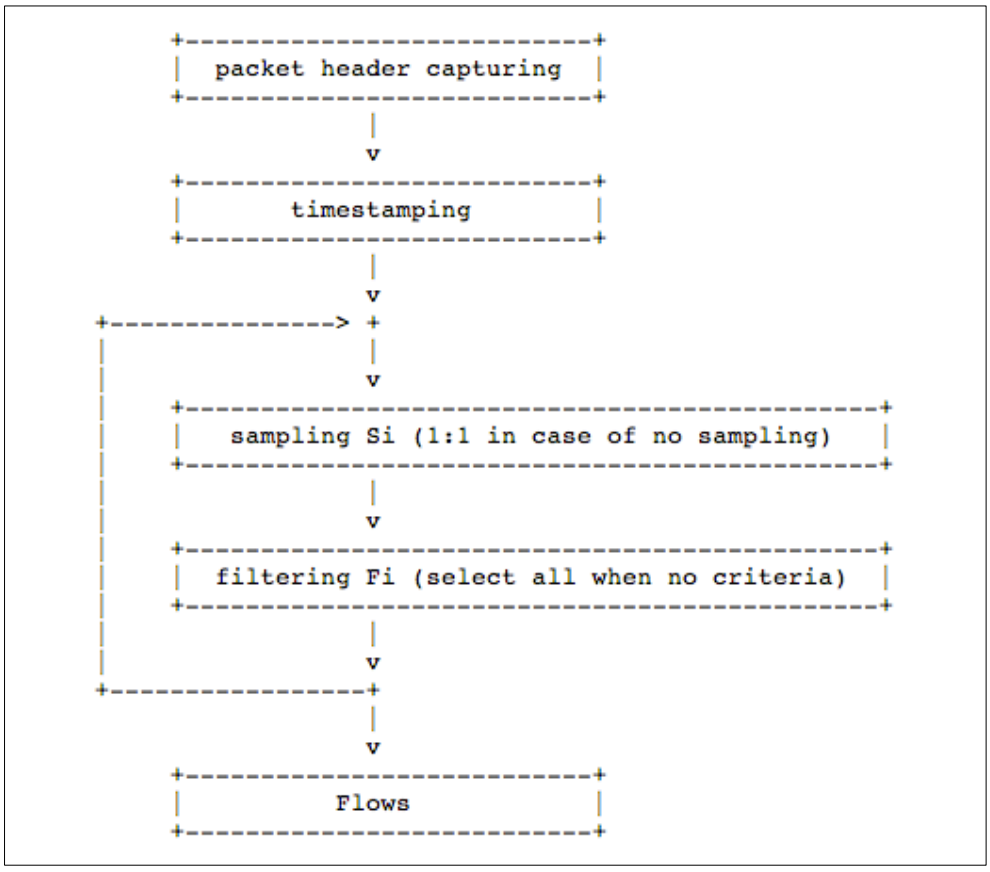

Figure 4 - IPFIX Packet Selection Criteria[4]

As it can be seen in *Figure 4*, the packet selection process happens immediately after a packet has had its header (addressing and destination information) examined and a timestamp assigned.

[4] http://www.rfc-editor.org/rfc/rfc5470.txt

First, the sampling functions are applied that determine whether the packet needs to be included in the flow generation process through straightforward selection preferences. A sample function's duty might be to only select every 100th packet. If there is no sampling function defined, then all packets are included.

Next, the filtering functions are applied that determine whether a packet needs to be included based on selection patterns applied to its attributes. For example, a packet could only be included in the flow generation process if the its associated protocol is TCP and destination port is less than 1024. If no filtering functions are defined, then all packets are included.

3. BOTNET DETECTION MODULE DESIGN

As it can be seen in *Figure 5* below, the Botnet Detection software module design specifies the logical position of its five major components and the information flow between them, illustrated by the numbered yellow arrows. The data flow direction is given by the yellow arrows, while the sequence in which it takes place is indicated by the numbers 1 to 7, 1 being the data entry point and 7 being the data exit point, both of which being facilitated by the AMDS Connection module. For more information on the Connection module's design, please see [1].

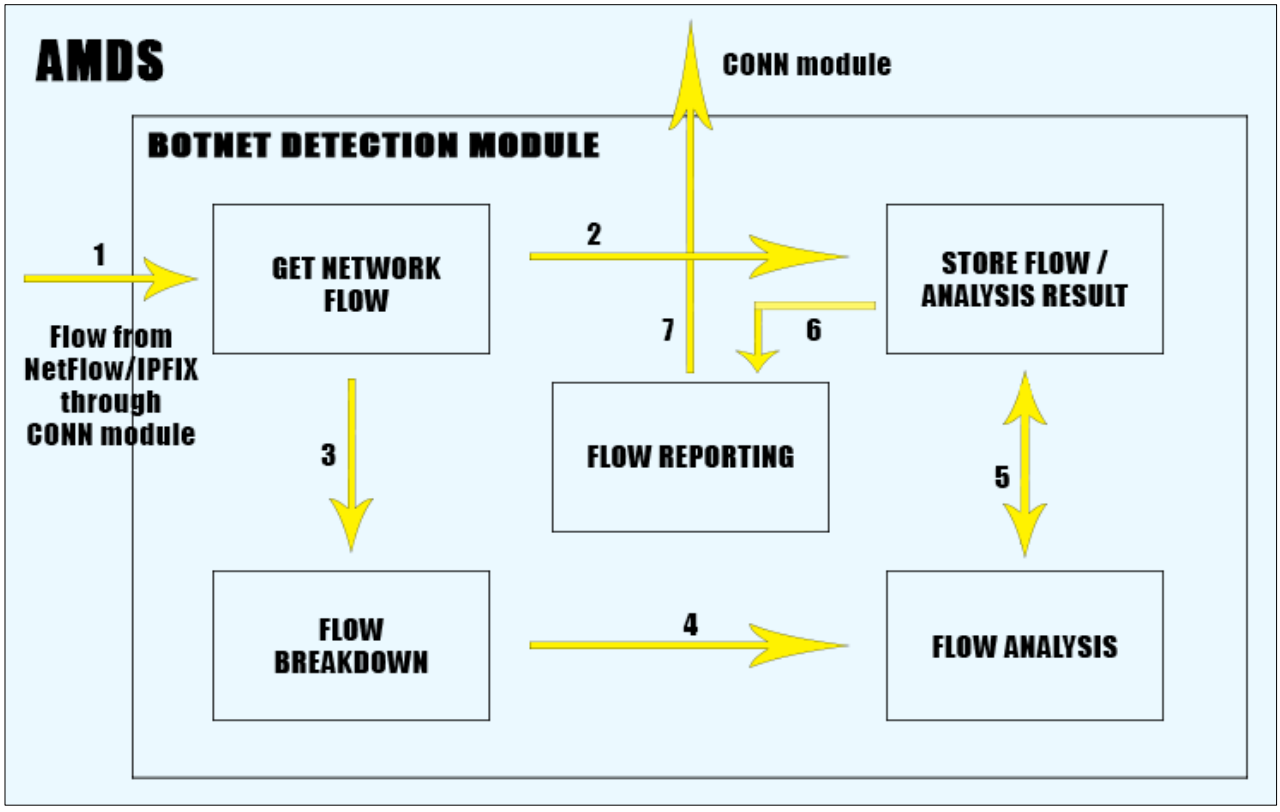

Figure 5 - Botnet Detection Module Design

This new module's major components are all designed to each fulfil a specific function, thus feeding into the AMDS modular design concept. They will now be described at an overview level, as follows:

I. *Get Network Flow*. This component, as its label suggests, is responsible for requesting and accepting network data flows from the NetFlow/IPFIX setup through the AMDS Connection

module. It first initiates the request, which is then forwarded by the Conn module using an appropriate communication interface, and it then awaits a response in the form of a Network Flow. Once it receives this information, it passes it on to both the storage and breakdown components.

II. *Store Flow / Analysis Result*. This part of the Botnet module is responsible with storing, as a long-term solution, raw Network Flows as forwarded by the previously discussed component and analysis results as transmitted by the *Flow Analysis* component, described below. It uses an internal storage system rather than relying on the other AMDS modules for security considerations. Another responsibility it carries refers to passing on the analysis results to the *Flow Reporting* component.

III. *Flow Breakdown*. This element is charged with extracting key bits of information from the raw network flows, as required by the detection algorithm, and forwarding them to the *Flow Analysis* component. Generally, it looks for packet size, IP addresses and ports for both packet source and destination, class of service, device interface, and protocol type.

IV. *Flow Analysis*. In this part is where the bulk of the module functionality exists. It is the embodiment of the detection algorithm and, as such, is charged with utilising the information extracted from the raw network flows. The main purpose of this component is to attempt to detect malicious botnet activities by comparing current findings with past findings. This is achieved through querying the *Store Flow / Analysis Result* component for old records of previously analysed data and comparing healthy flows with current, yet unidentified flows. In the event of an inconclusive analysis result, it flags the current flow as such. Finally, it forwards its findings to the storage component.

V. *Flow Reporting*. This element's main responsibility is to retrieve analysis results and forward them to the User Interface AMDS module for review purposes. It also is charged with forwarding analysis statistics based on past results, which helps provide an overview of this module's activities.

4. EXPERIMENTAL SETUP

The scope of this experiment is limited to AMDS network sampling through the use of NetFlow and IPFIX open source software by the Botnet Detection Module. It has several key points it is achieving:

I. Sample 10% of all network data flow using IPFIX / NetFlow.

II. Sort collected samples into logical groups based on parameters such as data packet Size, Source, Destination, and Commands.

III. Test AMDS' ability to query stored data packets and analyse their respective groups in an attempt to discover unusual network behaviour.

IV. Aid with constructing an operational model based on packet analysis results capable of detecting suspicious client behaviour.

The experiment has employed the following parameters:

A. 1Gbps network connection speed between the clients and the AMDS, which allows a maximum of 1 billion bytes per second.

B. Data packet sample size was set at 10% of all traffic at the point of collection.

C. Average data packet size ranged between 500 and 1000 bytes of data.

D. Infected (Botnet) packets have been used randomly starting with Sample #500.

In *Figure 6* below, a new logical node has been created and inserted in-between existing logical network nodes (VLAN #1-3, and the ASA). This allows for potentially all data packets, the ones coming into the network as well as the ones going outside the network, to be stored in a local database and inspected at a very low level. Every packet has the option of being grouped up with other similar packets for easier comparison.

The IPFIX / NetFlow logical node is capable, through outside (other locally networked devices) interference, to sample random data packets passing through them. Although they are capable of sampling

100% of the data, this is not recommended as it would slow down network flow as well as increase power consumption on the node they reside. For these reasons only 10% of all data is sampled, stored, and grouped up in the local IPFIX / NetFlow database.

The local data packet database has the ability to be queried for small portions of data at a time for easier analysis. Also, the AMDS has the capability of achieving this task through one of its built for purpose modules. Once retrieved, the AMDS creates a graph of data packets by comparing their Size, Source, Destination, and Commands they carry. The more regular data it analyses, the higher the chance of it detecting unusual behaviour on the network, and the lower number of false positives.

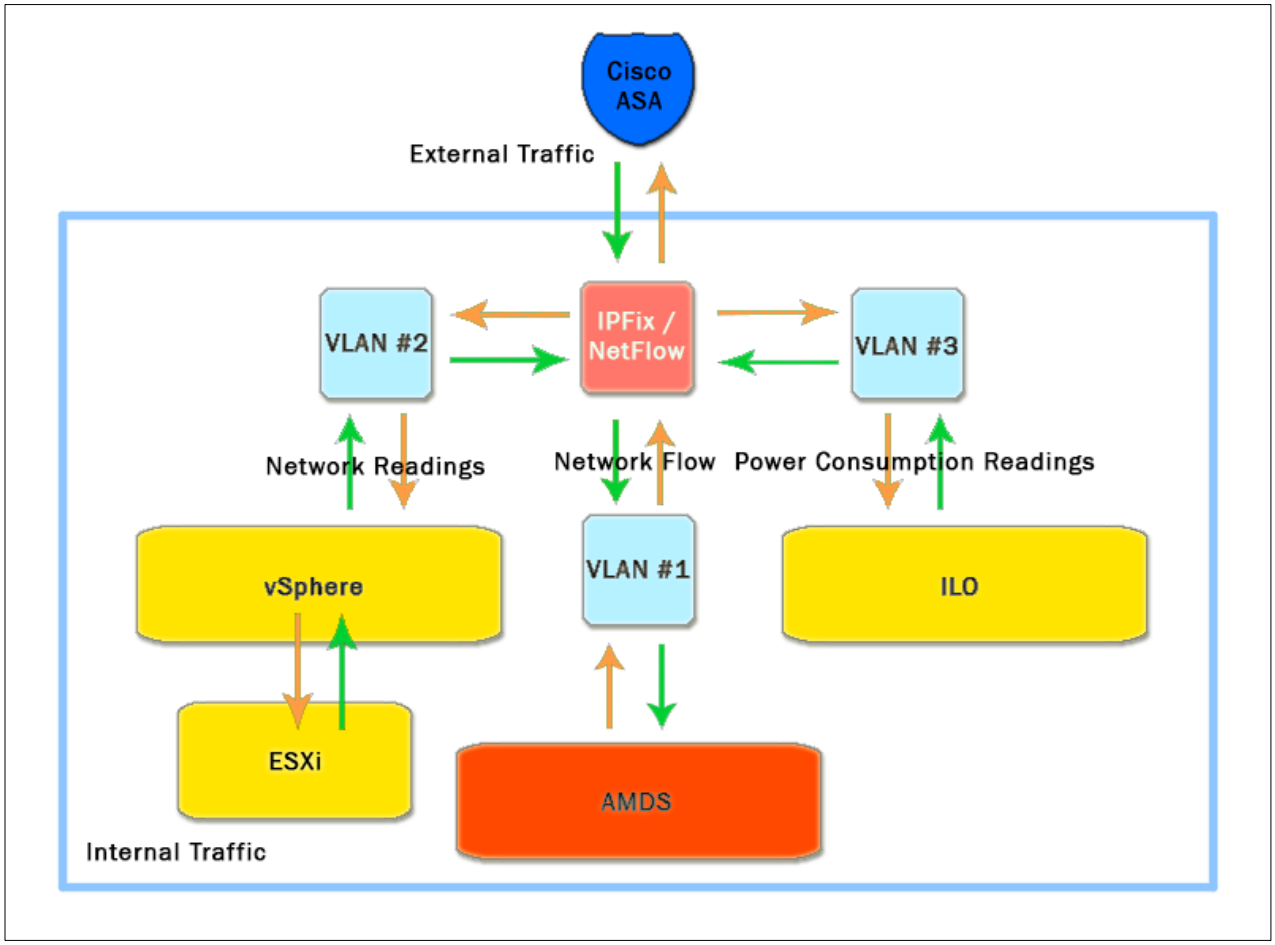

Figure 6 - AMDS Traffic Sampling Network Layout

5. EXPERIMENTAL RESULTS

The experiment has run over an extensive period of time and it has produced a great deal of log data. This data, along with some of the experiment parameters, can be seen in *Table 1* below.

For the first half of the experiment, as it can be seen in *Table 1*, regular packets with an average size of 500 bytes have been filtered through the AMDS Botnet Detection Module. This has been used as a training mechanism for the heuristic algorithm, so it would later on have a healthy packet model to compare infected packets against.

As it can be seen in *Figure 7*, using the 1Gbps network link has evaluated into approximately 2 million data packets, of which 200 thousand have been sampled at each of the 500 initial readings for analysis. Having an increased average packet size has impacted the samples containing infected Botnet packets in addition to the regular packets by having a reduced size. These samples were made up of between 100 thousand and 133 thousand packets on average.

For the second half of the experiment, a random percentage of Botnet generated data packets have been introduced instead of the regular data packets used in the first half of the experiment. This had a direct impact on the data packet size as this has increased the average packet size of the samples by 50%, from 500 to 750 bytes each. The Botnet packets have been created by combining botnet and client commands in one single packet, resulting in a data packet with an average size of 1000 bytes.

Sample #	# of Packets (1000s)	Avg. Packet Size (B)	# Infected (1000s)	# Detected (1000s)	Detection Rate (%)
50	200	500	0	0	0
250	200	500	0	0	0
500	133	750	18	5	28
1000	100	1000	28	12	43

Table 1 - Packet Analysis Results

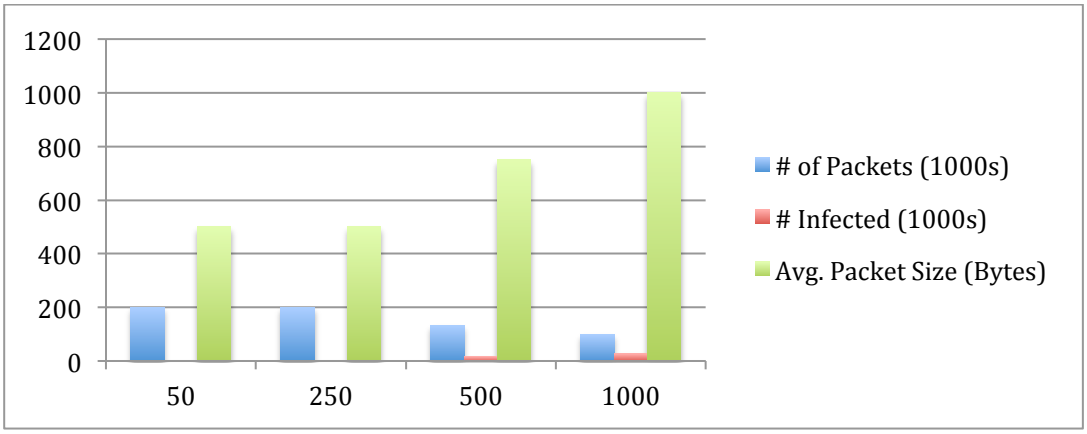

Figure 7 - Packet Distribution per 10% Sample

As it can be seen in *Figure 8*, the AMDS, through the use of its Botnet Detection heuristic algorithm, has managed to detect approximately 28% of all infected packets at the start of the Botnet attack. This detection rate has steadily increased up until the end of the experiment to approximately 42% of all infected packets.

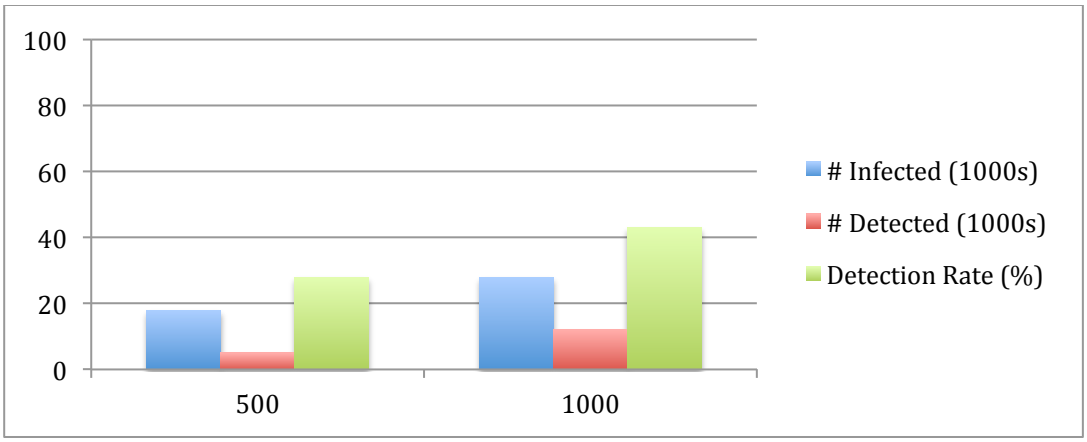

Figure 8 - Botnet Packet Detection Rate

6. CONCLUSION

The implications of the results shown in *Figure 8* go beyond just detecting a potential botnet attack in a datacentre. Although this fact alone give these results meaning, it also allows for an abstract software model to be defined, which can then be implemented using different programming languages in a multitude of different computing environments.

The main value of this experiment is apparent and will become even more so in time, when after several more data sets have been analysed, a comprehensive detection model will be created and continuously refined, which can used to test all future packet samples in an attempt to discover new, zero-day malicious activities.

The results presented above give a clear indication of the potential of the AMDS having its Botnet Detection Module activated. Applying the heuristic algorithm to more and more data packet samples allows the AMDS Botnet Detection module better understand what Botnet data packets look like, and detect more similar packets or even unknown Botnet packet type in the future.

REFERENCES

[1] Dinita, R. I., Wilson, G., Winckles, A., Cirstea, M., Rowsell, T. (2013). A Novel Autonomous Management Distributed System for Cloud Computing Environments. In *Industrial Electronics Conference (IECON), 2013 39th Annual Conference of*. IEEE.

[2] Zeidanloo, H. R., Shooshtari, M. J. Z., Amoli, P. V., Safari, M., & Zamani, M. (2010, July). A taxonomy of botnet detection techniques. In *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on* (Vol. 2, pp. 158-162). IEEE.

[3] Binkley, J. R., & Singh, S. (2006, July). An algorithm for anomaly-based botnet detection. In *Proceedings of USENIX Steps to Reducing Unwanted Traffic on the Internet Workshop (SRUTI)* (pp. 43-48).

[4] Jeong, H. C., Im, C. T., & Oh, J. H. (2010). *U.S. Patent Application 12/942,700*.

[5] Liu, L., Chen, S., Yan, G., & Zhang, Z. (2008). Bottracer: Execution-based bot-like malware detection. In *Information Security* (pp. 97-113). Springer Berlin Heidelberg.