Botnet detection using software defined networking

Article · June 2015

DOI: 10.1109/ICT.2015.7124686

Udaya Tupakula
Macquarie University
76 PUBLICATIONS **774** CITATIONS
SEE PROFILE

Vijay Varadharajan
Macquarie University
480 PUBLICATIONS **5,246** CITATIONS
SEE PROFILE

Botnet Detection using Software Defined Networking

Udaya Wijesinghe Udaya Tupakula Vijay Varadharajan
Advanced Cyber Security Research Centre
Macquarie University
Sydney, Australia
udaya.wijesinghe@students.mq.edu.au; {udaya.tupakula; vijay.varadharajan}@mq.edu.au

Abstract—**Software Defined Networking (SDN) is considered as a new approach promising simplified network management by providing a programmable interface. The idea of SDN is based on the separation of control plane from the data plane in networking devices. This is achieved by having the network intelligence centralised in what is called as SDN controller. In this paper we propose techniques for botnet detection in networks using SDN. The SDN controller makes use of generic templates for capturing the traffic flow information from the OpenFlow switches and makes use of this information for detecting bots. We will show that our model can detect a range of bots including IRC, HTTP and peer-to-peer bots.**

Keywords—SDN; Botnet; IPFIX

I. Introduction

Software Defined Networking (SDN) [1] is based on the idea of having directly programmable networks. SDN is considered as the next generation of networking technology. The idea of SDN is based upon the separation of control plane from networking devices to better optimize each. The control of the whole network becomes under the responsibility of the controller. The controller manages the flow-entries in the flow tables of the SDN switches such as OpenFlow switches through a southbound API using controller's global view of the network. This management process might be done both reactively (in response to packets) and proactively through secure channels. There are many SDN controllers available at the moment depending on the programming language used in each of them and purpose of the work. For example, NOX is based on C++ and Python programming languages; POX and Ryu, which are based on Python; and Beacon and Floodlight are based on Java.

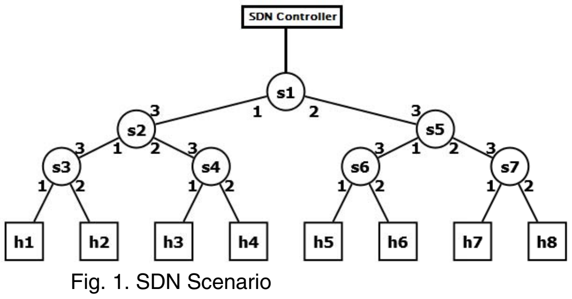
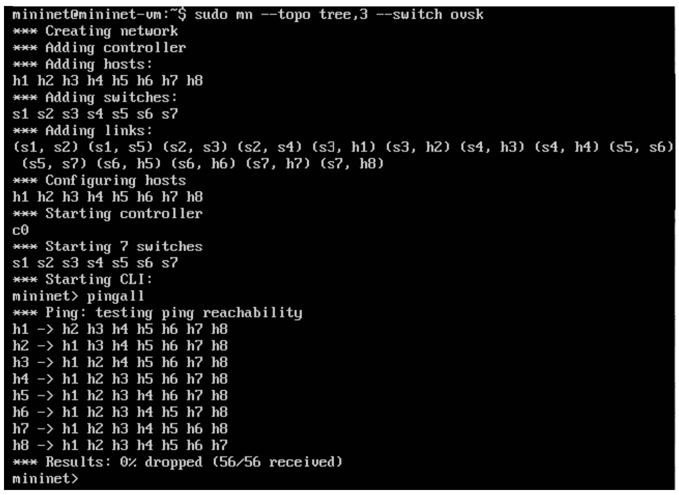


Fig. 1. SDN Scenario

OpenFlow protocol is commonly used in SDN and OpenFlow switch is used to represent functionality from layer 2 to Layer 7 in the OSI model (ex. Load balancing, WAN acceleration and virtual security appliance). An OpenFlow compatible forwarding device in data plane consists of three parts: i) A flow table, with an action linked with each flow entry, to tell the switch how to process the flow; ii) A secure channel that joins the switch to a remote control process (called the controller), allowing instructions and packets to be sent between a controller and the switch using OpenFlow protocol; iii) the OpenFlow Protocol, which provides an open and standard way for a controller to interconnect with a switch. An entry in the Flow-Table has three fields: i) A packet header that defines the flow; ii) the action, which defines how the packets should be handled; iii) counters, which keep track of the amount of packets per flow, and the time since the last packet matched the flow (to help with the removal of idle flows).



```
mininet@mininet-vm:~$ sudo mn --topo tree,3 --switch ovsk
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7
*** Adding links:
(s1, s2) (s1, s5) (s2, s3) (s2, s4) (s3, h1) (s3, h2) (s4, h3) (s4, h4) (s5, s6)
 (s5, s7) (s6, h5) (s6, h6) (s7, h7) (s7, h8)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controller
c0
*** Starting 7 switches
s1 s2 s3 s4 s5 s6 s7
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8
h2 -> h1 h3 h4 h5 h6 h7 h8
h3 -> h1 h2 h4 h5 h6 h7 h8
h4 -> h1 h2 h3 h5 h6 h7 h8
h5 -> h1 h2 h3 h4 h6 h7 h8
h6 -> h1 h2 h3 h4 h5 h7 h8
h7 -> h1 h2 h3 h4 h5 h6 h8
h8 -> h1 h2 h3 h4 h5 h6 h7
*** Results: 0% dropped (56/56 received)
mininet>
```

Fig. 2. Flow establsihement between the hosts

In traditional networks, when the network devices receive the traffic from a host, they make a decision based on the routing information available in each device (pre-configured or captured from dynamic routing protocols) and does not have a centralised control of the network. In SDN, when the switches receive a new flow from a host, they simply forward the traffic to the controller and rely on it to establish the communication path. The centralised controller makes a decision on how to forward the traffic to the destination and configures corresponding flow rules on all the switches between the source and destination. Any further packets related to the same flow are directly forwarded by using flow rules of the switches. Figure 1 shows a sample SDN topology generated in Mininet emulator, which makes use of POX controller. Figure 2 shows successful establishment of end-to-end communication flows between the hosts. However the attackers can leverage established flows to generate attacks in such networks. For example, the compromised hosts can be used to generate attacks after the controller establishes the flows. Hence there is need to deal with such security incidents.

The paper is organised as follows. Section II presents the attacker model and overview of the operation of our model. Section III presents detail discussion on the architecture components that are used for bot detection. Section IV presents the implementation and Section V considers some of the related work. Section VI concludes.

II. Our Approach

A. Attacker Model

We consider a generic SDN scenario with several hosts that are connected using OpenFlow switches and the controller is used for establishing end-to-end flows between the hosts. In Figure 1, the switch that is connected to the host receives the flows initiated by the hosts. Since the switch does not know how to handle the packet, it reports the traffic to the controller. The controller will make a decision on how to forward the traffic to the destination and sets up the flow rules in the switches for forwarding the traffic. For example, the controller needs to configure flow table in s3, s2, s1, s5 and s7 to enable communication between h1 and h8. Any further packets from the host will be directly forwarded by the switches to the destination according to the flow rules configured by the controller.

Malware is extremely serious pressure to modern data networks including SDN. There are various types of malware. Among the malware, botnets are a clever piece of software, which carry out sophisticated synchronized activities while being resilient. The botnet is a group of hijacked computers, which are employed under command and control mechanism administered by a botmaster. There is need for techniques to detect botnets and isolate the compromised hosts in SDN. In this paper we consider the case where the end hosts are compromised by the botnet and used for generation of malicious traffic. For example, the hosts that are compromised by the botnet can generate its malicious traffic (C&C traffic or attack traffic) after the controller establishes the flows. Note that the compromised hosts can also directly flood the controller but switches can be configured to drop all the traffic from hosts that is destined to the controller. Hence the main focus of our work is to deal with the bot attacks in data plane.

B. Operation Overview

The SDN controller is critical component which is used for securing and managing devices in the SDN network. In our model, the controller configures the OpenFlow switches to capture the traffic flow records according to the IPFIX template and makes use of this information to detect the bots. Figure 3 illustrates the high-level architecture of our model. Generic templates, Flow collector, Multistage filtering, Botnet detection engine and Attack prevention are some of the important components in our model. Generic templates are used for capturing flow informatin using IPFIX. Flow collector is used for storing and organising the data captured by the generic templates. Filtering is used to reduce dataset by eliminating unwanted data that is not related to botnet. The botnet detection engine correlates flow information using machine learning techniques to find the pattern and detect bots. The attack prevention component is used for isolating the hosts that are infected with bot. We will discuss each of these component in detail in Section III.

We use offline analysis for capturing the behaviour of different bot families. We make use of virtualisation techniques to generate traffic related to the bot. For example, we create a test bed with several virtual machines and infect some of the virtual machines with the bot and monitor the behaviour of the infected machine. Then we use machine-learning techniques for analysing the behaviour of the bot and identify the specific features that can be used to detect the bots. We repeat the process for different bot families and identify the specific features that can be used for detecting the bots. We have used different machine learning techniques for analysing the communication of different bot families and made some interesting observations.

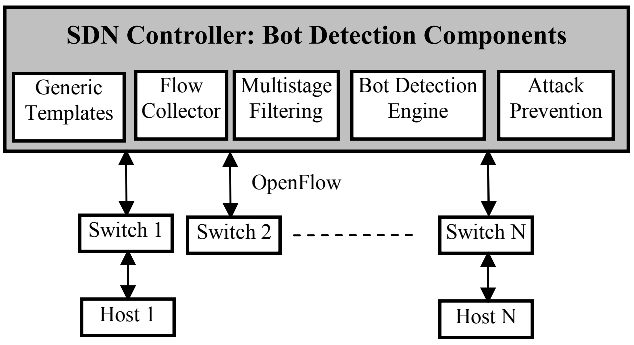


Fig. 3. Bot Detection using SDN

As a pre-programmed system, botnet runs automatic algorithms throughout their life cycle in order to conduct different activities. Even though botnet designers make a significant effort to randomize their activities, our analysis confirms that there are still patterns to identify them.

- Some of the bot families such as IRC-based, HTTP-based and DNS-based have specific characteristics when they react to the C&C instructions and how bots react to those instructions
- Bots such as SpyEye and some versions of Zeus have a unique flow behaviour that can be used as signature for detecting the bots.
- The communication in peer-to-peer bots such as, queries for update and instruction create many small uniformed packets than other legitimate P2P communications. Also the initial exchanges of packets have a fixed format and can be easily differentiated from legitimate traffic.

After analysing a range of bots, we develop a generic template by combining all the unique features of network flows for detecting different bot families. For example, features such as source address and destination address are common features for detecting any bot. Hence our generic template includes all the features for detecting different bot families. Also note, if we are interested in detecting specific bots, we only need to use the specific features to detect that bot. There is no need to develop a generic template for such cases. Now let us consider how the generic template can be used to detect different types of bots.

We make use of the IPFIX standard for defining the generic template. In the current state of art, different vendors have proprietary protocols such as Cisco-NetFlow, Juniper-j-flow or cflowd, Huawei-Netstream for capturing the flow traffic. Hence by defining the generic template using IPFIX, it can be applied on the SDN switches that belong to different vendors.

The generic templates are applied on the OpenFlow switches. These devices analyse the traffic flowing through them and report the traffic flow information to the flow collector. All packets with the uniform source/destination IP address and ports, protocol, interface and class of service are categorized into a flow. Subsequently number of packets and number of bytes are matched. This approach of fingerprinting of a flow is scalable as a large amount of flow information is summarized into a database. This flow information is extremely valuable for analysing malicious behaviour of high speed, high traffic volume IP networks. As we are only summarizing flow details it is relatively very small compared to actual size of data streams.

Multistage filtering is used to further minimise the captured dataset by eliminating redundancies and unwanted traffic that is not related to botnet. For example, core OpenFlow switches can report the network flow information of communication between internal hosts and external hosts. Gateway OpenFlow switches can report network flow information of external hosts. Hence we only store the internal host communication information reported by the core switches and the external host communication reported by the gateway switches. Botnet Detection Engine is used for analysing the filtered traffic and detecting bots. Since generic templates are used for capturing the flow information, the captured data includes data for detecting different bots. Hence different bots can be detected by analysing subset of the features captured in the network traffic flows. The machine learning techniques that were initially used for identifying the specific features in C&C communication, for each bot are also used for detecting the attacks from the flow dataset. If the bot detection engine identifies any of the hosts to be infected with bot then Attack Prevention component will apply an access control filter and configure the OpenFlow switches to isolate the infected host from communicating with other hosts in the network.

III. Architecture Components

In this Section we will describe the components of our architecture.

A. Generic Templates

Templates are used for capturing network traffic flows from different devices for detecting the bots. Flows are defined as group of IP packets, which share common, attributes such as source IP, destination IP, source port, destination port, protocols type, class of service and interface of the network element. Based on these attributes each packet going through a network element can be classified into unique flows. For example, these attributes enable us to determine if the received packet belongs to an existing flow or a new flow.

Existing IP flow standards have several limitations since they use proprietary protocols, have fixed format for flow classification and are not compatible with products from different vendors. Hence the related techniques such as Botfinder, Disclosure, BotSniffer, and BotTrack which make use of these tools for flow analysis, are not efficient. Also these techniques are not designed to be used in SDN.

The fixed nature of capturing IP flows limited the capability of security analysis while avoiding important features from IP flow data, For example, conventional fixed IP flows only report 20 attributes: Source IP address, Destination IP address, IP address of the next hop router, SNMP index of the input interface, SNMP index of the output interface, packet in the flow, total number of L3 bytes in the flow's packets, system up time at start of the flow, system uptime at the last packet of the flow received, layer 4 source port number, layer 4 destination port number, cumulative OR of TCP flag, IP protocol, IP type of service, AS number of the source, AS number of the destination, source address prefix mask bits, destination address prefix mask bits, and two unused attributes (pad 1 and pad 2) .

The fixed attributes consume 48 bytes for each of the flow record. From our analysis, we identified that most of the attributes that are used in the fixed flow templates for classification of the flows are not useful for detecting the bots. Furthermore, there is no way to include new attributes that can be used for efficient classification of the flows for bot detection. Hence the main disadvantages of using fixed IP flow is that it narrows down the security analysis that can be performed on the captured traffic and it also results in unnecessarily increasing the size of data set.

Recently, IETF introduced open standard for flexible IP flows named as IPFIX. This IP fixed abstracts most of the features from NetFlow version 9 to define standard for IP flows. However unlike Netflow version 9, IPFIX added variable length fields. Furthermore, IPFIX standard allow users to select wider range of attributes from more than 400 different data points (RFC7012, 2013), Hence IPFIX enables the users to define the templates with any of the attributes for traffic classification.

Our system makes use of IPFIX for developing customized templates for detecting botnets. There are two main benefits with the customized IP flow templates used in our model. First, customized templates facilitate to reduce size of the data set by removing unwanted attributes from fixed data set. This is a very important benefit of our proposed system, as this will significantly reduce the size of data sets for analysis and save space and computing power. Secondly, proposed templates provide more room for botnet detection since new templates come up with additional important attributes in addition to traditional network flow data set. For example, customized templates allow us to develop new data templates based on botnet families, adding new features from IP header, adding new features from pay load (DNS attributes), adding new statistical attributes (average pay load size) in order to detect botnets. Further, this will facilitates to detect DGA and fast flux attacks by introducing DNS attributes in to network flow data set.

Recall that we have used machine-learning techniques for identifying the attributes that are efficient for detecting the bots. We have used these attributes for developing generic templates by using only the attributes that are efficient for detecting different bot families. Once customized templates have been developed, it is applied to different network devices. Since we make use of IPFIX for developing the template, it can be used on SDN switches that belong to different vendors. Now the network devices classify the flows based on the attributes specified in the template and send this information to the flow collector.

B. Flow Collector

The flow collector is used to collect the flow records from different devices. As flexible IP flow comes with variable size filed set and data set, flow collector enables storing according to the requirement of flow templates. The network devices send their data along with data templates to the flow collector. The flow collector software facilitates the security administrators to define storage template to match with the flow templates for storing the data. Then collector builds a raw dataset based on information from templates, by storing received data in correct order. Data is arranged into directories based on type of traffic, flow direction, year, month, date and time. Flow collector also provides additional features such as compression of the data to save the storage space and ability to perform search on the stored data. Another important feature of flow collector is to facilitate usage of various analytical tools on the stored data to eliminate unwanted traffic and detect botnets. This provides more room to use different filtering and analytical tools to access data and make data available for various analysis methods.

C. Multistage Filtering

The main purpose of the multistage filtering is to reduce data set since generic templates can also capture the flows that are not related to botnet. Hence before forwarding the collected flows to the next stage for further processing and detection of attacks, we reduce the dataset by removing unnecessary data stored in the flow collector. This will make data set more manageable and save computing power in next steps while providing capability to deal with high volume data networks. Similar to Strayer et al. [2] we use a five step filtering mechanism for botnet detection system. However there are also some differences in the filtering used in our model. Below we summarise how our model differs to Strayer.

Firstly, as Strayer mainly forcuced on IRC based botnets, his method only uses TCP flows for the botnet detection and filters out all other traffic. Our model targets a range of bot families. Hence we make use of TCP or UDP flows based on the behavior of botnet families. For example, some bots manipulate legitimate DNS traffic to tunnel C&C communication. In that case we filter out TCP flows and keep UDP flows which are suitable for DNS based botnet families.

Secondly, Strayer uses a filter to remove the nuisance port-scanning chaff in order to reduce dataset. The main idea behind this is if a flow contains only SYN or RST flag that means TCP flow is not established and un-established flows are no value for C&C communication. However, un-established flows represent some botnet activity such as DDoS or random packet throwing to mislead detection tools. So instead of filtering out un-established flow we categorize flows with only SYN or RST flags in order to detect botnets. Further, flexible IP flows allow us to customize flow template to collect flow data based on TCP flags while fixed flows only allow us to collect cumulative TCP flags.

Thirdly, Strayer introduced a filter to remove high volume data flows as C&C communication is not supposed to generate high bitrate data flows. Also software updates and peer-to-peer data transfer generate high volume data flows. Hence we use similar filter to distinct legitimate peer-to-peer traffic from P2P or hybrid botnets' C&C channels.

Fourthly, Strayer uses filter to remove flows with large IP packets specifically packets size above 300 bytes. This filter can only be used for detecting IRC botnets. However some botnets use large data packets to send stolen data to its C&C server. Hence we consider all the packets for our analysis. However, if specific pattern exists for some of the bots we modify this filter in order to identify such a communication by selecting correct size of the packets. Further, usage of IPFIX in our model allows us to get payload and header size of the packets separately. We use this attribute for fine-tuning of the filter.

Finally, Strayer used filter to remove flows with 2 or less packets or very brief time windows to remove port-scanning flows. In our model we make use of this filter to identify vulnerability or open port scanning by the botnets to find new victims.

Another important feature in our model is to define additional filters for C&C traffic behaviours of different botnet families. Such filtering enables to filter flows related to particular botnet family and facilitate further analysis effectively as it will significantly reduce data set.

D. Botnet Detection Engine

Once data set has been selected for comprehensive analysis by series of filters described in previous stage, data set is fed to botnet detection engine. First step of the engine is to classify or cluster the flows in data set in order to identify similarities of the flows. The machine learning techniques, which are initially used to identify the specific features for each bot, are used in this stage to detect the specific bots. There are three main behavioural patterns; bot behaviour, botnet behaviour and temporal behaviour are used in our system.

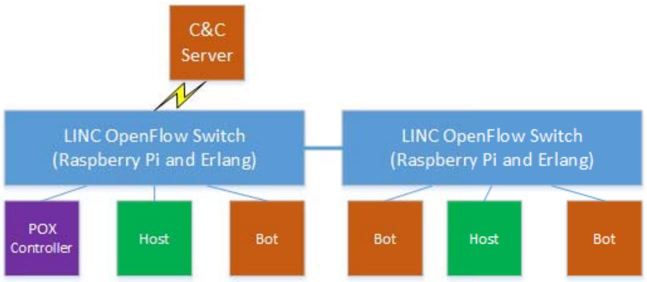
In the bot behaviour, we analyse flows generated from one bot or single machine to identify its C&C communication or attack graphs. In botnet behaviour, we analyse flows generated by group of bots or machines, in order to detect botnet activities. Further, in above method we horizontally analyse flows generated in a network to find suspicious pattern related to botnet communication. Finally, in temporal or vertical method, we analyse flows generated by bots or botnets over the period of time in order to detect patterns. Once we identify, similarities or correlations, then we compare them with the patterns we already identified through machine learning. In Section IV we will present several observations that were used to detect compromise of hosts with different bot families.

E. Attack Prevention

After the bot detection engine identifies the hosts that are compromised by the attacker, there is need to isolate the compromised host from the network. Attack Prevention component is used for isolating the hosts that are infected with bot. It generates an access control filter and dynamically configures the access control policies in the OpenFlow switches to block all the traffic from the compromised host. Hence our model can detect the compromised host and isolate them from generating attacks in networks.

IV. IMPLEMENTATION

Our approach consists of two main steps. Firstly, we make an effort to identify best suitable template in order to collect flow data set. As there is no publicly available IPFIX dataset that can use for our study, we have to setup our own controlled lab environment to get a flexible IP flow dataset that includes malicious traffic sample. Figure 4 illustrate the design of the lab, which has been used to analyse different botnet samples and collect IPFIX data sets from SDN switches. We have build OpenFlow switches using Raspberry Pi B+ boards. Erlang LINC OpenFlow switch is installed on Raspberian Linux distribution. LINC is a virtualised OpenFlow switch that supports up to OpenFlow v1.4. Then we used 7 virtual machines run on Proxmox VE KVM. One VM build with POX SDN controller, one VM as C&C server and other 5 VMs are Windows XP machines. To get mix of benign and bot traffic sample we have infected 3 VMs with botnets.

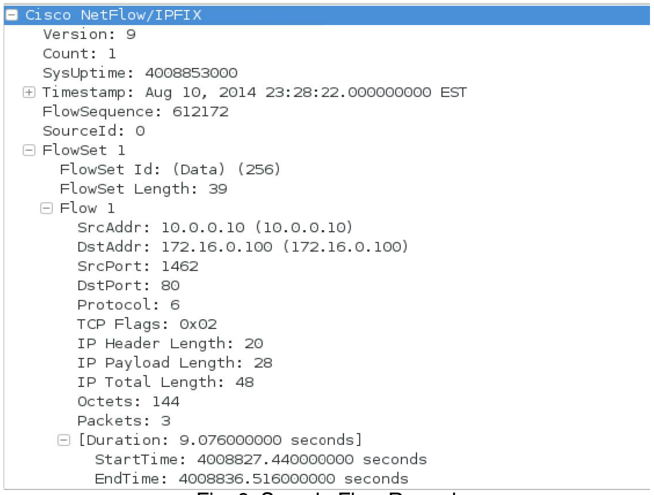

C&C Server
LINC OpenFlow Switch (Raspberry Pi and Erlang)
LINC OpenFlow Switch (Raspberry Pi and Erlang)
POX Controller
Host
Bot
Bot
Host
Bot
Fig. 4. Lab Environment

In such a case we obtain two separate datasets, dataset from analysis of different botnets in our lab environment and dataset from University network over a period of two days (about 5 GB), which contains everyday usage traffic patterns. We merged both the datasets using TCP replay tool to produce test dataset. We have used 10-fold cross-validation technique for training and testing of our model and achieved high detection accuracy.

Figure 5 shows the generic IPFIX template that is used for capturing the traffic flow information for bot detection and Figure 6 shows the sample flow record. The traffic flows are clustered in order to identify similarities of the flows. We have used several machine learning techniques such as Bayesian Network, Neural Network, Support vector Machine, Gaussian and Nearest Neighbour classifier for identifying the best attributes for each bot since none of the machine learning technique was found to be effective for detecting all the bot families. For example, Decision Tree classifier is found to be effective for detecting peer-to-peer botnet by analysing the flow intervals. SVM and Bayesian networks were found to be effective for detecting C&C communication of the centralized botnets such as HTTP and IRC botnets.

```
Template ID    : 256
Source ID      : 0
Record Size    : 35
Template layout
_____________________________________________________________________________
|                 Field                         |  Type | Offset |  Size  |
-----------------------------------------------------------------------------
| ipv4 source address                           |     8 |      0 |      4 |
| ipv4 destination address                      |    12 |      4 |      4 |
| transport source-port                         |     7 |      8 |      2 |
| transport destination-port                    |    11 |     10 |      2 |
| ip protocol                                   |     4 |     12 |      1 |
| transport tcp flags                           |     6 |     13 |      1 |
| ip length header                              |   189 |     14 |      1 |
| ip length payload                             |   204 |     15 |      2 |
| ip length total                               |   224 |     17 |      2 |
| counter bytes                                 |     1 |     19 |      4 |
| counter packets                               |     2 |     23 |      4 |
| timestamp sys-uptime first                    |    22 |     27 |      4 |
| timestamp sys-uptime last                     |    21 |     31 |      4 |
-----------------------------------------------------------------------------
```
Fig. 5. IPFIX Genaric Template

There are three main behavioural patterns; bot behaviour, botnet behaviour and temporal behaviour are used in our system.


```
⊟ Cisco NetFlow/IPFIX
    Version: 9
    Count: 1
    SysUptime: 4008853000
  ⊞ Timestamp: Aug 10, 2014 23:28:22.000000000 EST
    FlowSequence: 612172
    SourceId: 0
  ⊟ FlowSet 1
      FlowSet Id: (Data) (256)
      FlowSet Length: 39
    ⊟ Flow 1
        SrcAddr: 10.0.0.10 (10.0.0.10)
        DstAddr: 172.16.0.100 (172.16.0.100)
        SrcPort: 1462
        DstPort: 80
        Protocol: 6
        TCP Flags: 0x02
        IP Header Length: 20
        IP Payload Length: 28
        IP Total Length: 48
        Octets: 144
        Packets: 3
      ⊟ [Duration: 9.076000000 seconds]
          StartTime: 4008827.440000000 seconds
          EndTime: 4008836.516000000 seconds
```
Fig. 6. Sample Flow Record

We have analysed a range of bot families such as IRC, HTTP, peer-to-peer and hybrid bots and made several observations that enable for the detection of bots. For example, in case of C&C bots our analysis was mainly focused on detecting the bots during different phases in botnet life cycle; initial infection, secondary infection, connection, malicious command and control, and update and maintenance. In the initial infection phase attacker exploits vulnerabilities on victims and gains basic control over victims. Then secondary infection phase is used to further download and install malicious script and binaries to get full control of the victim. Once secondary infection is complete, bots make connection to its C&C server in order to become a member of botnet. Then bot will receive command and control from C&C server to conduct malicious coordinated activities. Finally bots update its binaries to get more functionality or evade detection. Below we summarise some of the important observations from that analysis of range of bot families:

In case of direct C&C related bots such as IRC and HTTP every bot needs to find its own C&C controller to be a member of centralized botnet. In case of peer-to-peer and

Hybrid bots such as Kademila, Chord and GameOver (Zeus v3) each bots need to find its servant bot or proxy bot to get C&C instruction and become a member of P2P botnet. Our analysis confirms that these botnets are using hard coded static IP lists or/and DNS service to locate its C&C server, from which it has to receive the control commands and updates. This generates a pattern, which is used for the detection of bots. Conversely, to evade from discovery and close it down, the C&C servers' use distinctive methods like Domain flux (Domain Generation Algorithm - DGA), and IP flux to alter their DNS name or the IP addresses connected with FQDN. However, this makes the botnets to connect to different C&C servers instead of connecting to a specific one. As a result, the bots perform a large number of DNS lookups and scan a large volume of addresses to find the C&C server. These patterns are used in our model to track the botnet.

The second observation related to C&C bots is that the bots need to perform frequent communication with the C&C server. This is essential to keep control and update the botnet by the bot master. Once C&C server has been discovered, the bots take updates and commands from the botmaster about what type of action has to be executed. The action can be whatsoever from sending spam emails to intensifying a DDoS attack. Another important massage type is keep alive packets that are sent from the bot to the C&C server. Moreover, some bots (such as Zeus and SpyEye) report back to C&C server with the information they steal from compromised computers. Even though botnet designers are working hard to randomize those communications to evade detection there are some vertical or/and horizontal correlation on their communication. For example, some of the Zeus bots (v1.3) send updates at fixed interval of 20 minutes.

All the botnets desire to spread over its network and recruit more bots into its botnet, which ultimately benefits to surge the strength of a botnet and consequently that of the attack carried out too. Hence the bots scan for other machines in its network for vulnerabilities. If a vulnerable machine is found, they will run exploits to compromise the machine. When scanning the network for possible machine to infect, bots generate a burst of small packets. So this activity makes a sudden increase in the number of packets without a major increase in the traffic volume that could be used to detect bots.

The bot detection engine also looks for DDoS activities such as, outbound TCP SYN packets having an invalid source IP address. The reason for these large number of TCP SYN packets could mean that some of the internal hosts in the network is part of a botnet and are participating in a DDoS attack. Some botnets such as (Grum, Bobax, Cutwail and Donbot) generate email spamming. Email spamming involves sending enormous amount of spam emails advertising fake products intended at financial gains. When the hosts in a network are part of a botnet involved with spamming, they send huge number of emails to the outside world and mostly using some external email server. So, unusual SMTP activity from the network to the outside is another significant network activity that is used for tracking the bots.

From the above discussion, it is clear that our model can detect a range of botnet families using various activities throughout their lifecycle including C&C interactions, recruiting new bot members and synchronized attacks. Furthermore, the modular design of the architecture components enable distributed implementation and is compatible with different SDN controllers.

V. Related Work

In this Section we will consider some of the recent work in SDN security. In [3] the authors highlight that the requirement to build secure and dependable SDNs by design. Then they have identified several threat vectors that potentially abuses on vulnerabilities of SDN. Finally they draft some of the security measures to deal with the attacks in SDN. Porras et al. [4] discussed the security challenges and the need for security enforcement kernel at the controller. Then it proposed role-based enforcement of security policies and conflict resolution for NOX controller. Li et al. [5] proposed to use multiple controllers to deal with the case of failure or compromise of single controller and more than one controller manages each switch. To protect SDN from such kind of threats to the control plane, we need to deal with the challenge of letting the controllers continue operate correctly, even if some of them exhibit arbitrary, possibly malicious behavior. The paper assumes that majority of the controllers are not compromised by the attacker.

The main focus of our work is making use of SDN for detecting a range of botnet families and isolation of compromised hosts from the network. Furthermore, our model makes use of IPFIX template, which is developed with attributes that are efficient for bot detection.

VI. Conclusion

In this paper we have proposed techniques for botnet detection using SDN. We have shown that our model can be used to detect different types of bots such as IRC, HTTP and peer to peer in SDN. Furthermore the SDN controller can dynamically isolate the compromised host from SDN.

References

[1] OpenNetworkingFoundation, "Software-Defined Networking: The New Norm for Networks", Available at: https://www.opennetworking.org/images/stories/downloads/sdnresources/white-papers/wp-sdn-newnorm.pdf [Accessed 12 Nov. 2014].

[2] Strayer, WT, Lapsely, D, Walsh, R & Livadas, C., "Botnet Detection Based on Network Behavior", Botnet Detection, Advances in Information Security Volume 36, 2008, pp 1-24, Springer.

[3] Kreutz, D., Ramos, F. and Verissimo, P. "Towards secure and dependable software-defined networks" Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking. New York, NY, USA: ACM, pp.Pages 55-60, 2013.

[4] Porras, P., Shin, S., Yegneswaran, V., Fong, M., Tyson, M. and Gu, G. "A Security Enforcement Kernel for OpenFlow Networks" Hot topics in software defined networks. New York: ACM, pp.121-126, 2012.

[5] He Li, Peng Li, and Song Guo, Shui Yu, "Byzantine-Resilient Secure Software-Defined Networks with Multiple Controllers" IEEE International Conference on Communications (ICC), Sydney, Australia, June 2014.