Hardware Isolation Technique for IRC-Based Botnets Detection

Festus Hategekimana*, Adil Tbatou†, Christophe Bobda*, Charles Kamhoua§, and Kevin Kwiat§
* *CSCE Department, University of Arkansas Fayetteville, Arkansas, 72701*
Email: fhategek@uark.edu, cbobda@uark.edu
†*ESIREM-DIJON, Dijon, France*
Email:adiltbatou@gmail.com
§*Air Force Research Laboratory, Cyber Assurance Branch, Rome, New York, 13441,*
Email: charles.kamhoua.1@us.af.mil, kevin.kwiat@us.af.mil

***Abstract*—Botnets are widely considered one of the most dangerous threats on the internet due to their modular and adaptive nature which makes them difficult to defend against. In contrast to previous generations of malicious codes, botnets have a command and control (C^2) infrastucture which allows them to be remotely controlled by their masters. A command and control infrastructure based on Internet Relay Chat protocol (IRC-based C^2) is one of the most popular C^2 infrastructures botnet creators use to deploy their botnets' malwares (IRC botnets). In this paper, we propose a novel approach to detect and eliminate IRC botnets. Our approach consists of inserting a reconfigurable hardware isolation layer between the network link and the target. Our reconfigurable hardware is an FPGA System-on-Chip (FPGA SoC) that uses both anomaly-based detection and signature-based detection approaches to identify IRC botnets. Since, unlike other viruses, to be able to freely communicate with their masters, botnets' primary objective is to disable any protection mechanism (firewalls, antivirus applications) found on the target machine; our hardware-based isolation infrastructure presents an improvement over existing software-based solutions. We evaluated our architecture codenamed BotPGA using real-world IRC botnets' non-encrypted network traces. The results show that BotPGA can detect real-world non-encrypted malicious IRC traffic and botnets with high accuracy.**

***Keywords*-Field Programmable Gate Arrays (FPGAs), Botnets, Internet Relay Chat (IRC), Signature-based Detection, Anomaly-based Detection.**

I. Introduction

Annual economic loss attributed to malicious software activities is estimated at more than US $10 Billion [1]. Among malicious techniques and tools, botnets are by far one of the most dangerous species due to their modular and adaptive nature. A botnet (or, network of zombies) is a network of compromised computers (bots) remotely controlled and used to perform: large-scale distributed denial-of-service (DDoS) attacks; send spams; trojans and phishing emails; and stealing of private information [2]. Unlike previous generations of malicious code,a bot's infectious malware is a malware of malwares [3]. Each infectious malware is made of several malicious modules. One module exploits the vulnerabilities it finds on the target machine to gain control. It then downloads another module to protect the old module by disabling antivirus applications and firewalls; a third module may begin scanning for other vulnerable targets on the network. Some botnets also contain a backdoor module, a DDoS capable module, and many more (ex. RBot botnet) [3].

Botnets possess the characteristic of a command and control mechanism (C^2) which allows the ¨botmaster¨ (who controls the botnet) to interact (issue commands and receiving responses) with the bots. The commonly used protocols which allow this C^2 mechanism are HTTP, P2P, and IRC. Recently, C^2s based on IRC (Internet Relay Chat) protocol have emerged as the most adopted C^2 mechanism by botnet creators because it has proven to be highly successful [4]. Therefore, understanding IRC botnets is essential to develop effective countermeasures against botnets.

IRC traffic botnets are difficult to detect because they tend to follow normal IRC protocol usage and there may be very few bots in the monitored traffic [4]. However, we observe that IRC bots fail particularly at demonstrating the normal behavior expected from humans during an IRC chat session. For example, some bots are known to join IRC channels multiple times frequently more than the messages they exchange over the same channel [3]. Wei Lu et al, [5] studied the response time of bots and found that bots' response time is faster compared to what you would expect from a human. These observations and others have led to development of many solutions to combat botnets attacks.

Current techniques for detecting botnets examine traffic content for public internet communication services like IRC [6], monitor DNS for strange usage [7], or set up honeypots to capture live bots [8]. More advanced network analytic approaches have been used to examine network data such as bandwidth, packet timing, and burst duration to find common communication patterns that would suggest the activity of a botnet [9]. The main problem with existing techniques is their software implementation. Botnets' primary objective is to disable any antivirus applications and firewalls running on the target machine to protect new bots. Consequently, software-based malware detection solutions that rely on network traffic data analyses become useless when the target has been hijacked. In addition, traffic analysis is computationally expensive with software implementations. It requires lots of resources (Snort or Suricata high cpu usage for example) which degrades the overall system performance. There are few existing hardware based Network Intrusion Detection Systems (NIDS) solutions such as, Ciscos 4200 series [10], and Proventias GX5108C-V2 [11] for example, but they are very expensive and are usually not optimized for botnets.

In this paper, we propose a novel approach to deal with the IRC botnets threats. Our approach consists of inserting a reconfigurable hardware isolation layer between the network link and the target. Our hardware is an FPGA System-on-Chip (FPGA SoC) which performs real-time packet processing to identify malicious network traffic. Concretely, our system performs both signature-based and anomaly-based detection techniques to analyze non-encrypted Internet Relay Chat (IRC) traffic packets for botnets. We developed a deep packet inspection engine (DPI engine, perhaps the most important part of our work, that detects IRC packets and filters them for preliminary diagnostic work. Features of interests are extracted from the packets received and stored into memory where they are read and processed by our binary classification engine.

The contribution of our work is the development of a reconfigurable hardware isolation layer which uses high-speed pattern matching and machine learning algorithms to implement signature-based detection and anomaly-based detection to detect malicious IRC traffic and IRC botnets. Preliminary results show it has high detection accuracy. We leave the problem of detecting HTTP, P2P botnets, DGAs-based, and Tor-based botnets in our isolation layer as our future work. The rest of this paper is organized as follows: In section II, we provide a background on IRC botnets and motivation of our approach; in section III, we describe the architecture of BotPGA and describe its detection algorithms; implementation details and tests results are presented in IV; and section V concludes the paper and presents the areas of future work.

II. IRC BOTNET PHENOMENON

A. Understanding IRC Protocol

IRC protocol, standardized by the Internet Engineering Task Force in 1993 [12], is based on a client-server model where the IRC server provides the point to which the client must connect to in order to be able to talk with others. For user a to be able to communicate, his IRC client connects to an IRC server through a certain port (usually 6667 but any port can be used) [12]. The user chooses a nickname and then joins one or more channels. Once connected, the user can start sending and receiving messages from other users.

IRC packets are regular TCP packets with IRC messages and commands as packet payload. An IRC message always begins with a command. The most important commands are JOIN, PING, PONG, and PRVMSG. The JOIN command is used by an IRC client to log onto a channel. A channel is an ASCII string and it is characterized by its name (must begin with &, +, #, or !) and its current members. It is important to mention that all channel members receive messages addressed to the channel (botmasters explore this property). To avoid idling too long, the IRC server sends PING messages to IRC clients. Typically PING are sent periodically every 30 sec. PONG are returned to the server to indicate that the client is still active and does not want to be logged out. A PRVMSG command is used to send private message that contains the channel name and data (this can be a message instructing bots to scan ports, update binary, etc.) The IRC protocol uses a space character as a parameter separator and a comma as a list-items separator. In addition, all IRC messages end with CR-LF (carriage return line feed) [3].

B. IRC Botnet

The IRC protocol is very popular with botnet creators since it is a free, public internet communication service and it does not require registering accounts or handles and it is very easy to set up your own channel. The most important module of an IRC botnet is the one that makes an IRC client capable of understanding IRC commands [3]. Thus, a target machine does not have to have an IRC client installed on to be targeted by an IRC botnet. The malware itself has its own IRC client and therefore, can easily interact with the botmaster to coordinate their attacks. Mining IRC traffic for botnets, begins with checking whether or not the IRC protocol commands are being used properly. For instance, botmasters (usually IRC clients) sometimes send PING messages to their bots (IRC clients) to check if they are still part of the network. This is unusual use of the PING command and should raise suspicions on this channel. Gu-Hsin et al, did extensive work on this [6].

III. SYSTEM OVERVIEW

Our architecture performs two complementary traffic detection analyses. First, it implements signature-based detection by using a high speed pattern matching algorithm to mine non-encrypted IRC traffic for known virus signatures and possible TCP/IRC protocol rules violations. In this phase, we are capable of identifying malicious packets and eliminating them as long as the received packet possesses any of the signatures we are monitoring. Using partial reconfiguration techniques, the signatures pool can be updated anytime new signatures are available. Of course, the drawback here is that the model does not perform well against unknown signatures, but this is common for any signature-based malware detection system. This is where our second detection function comes into play. Important features about TCP/IRC packets that seem to conform to IRC protocol rules are sent to a binary classification engine for further processing. This is important because some botnets have been found to conform to protocol rules, but fail to demonstrate the type of behavior exhibited by humans in a normal IRC chat session [5]. Specifically, we check whether there are channel members who joined

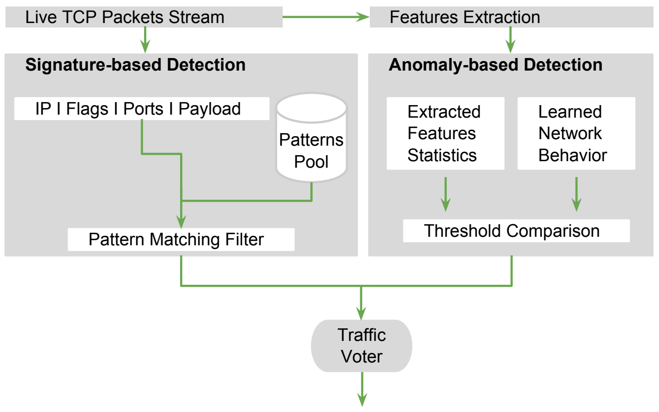

Figure 1. System Architecture Overview.

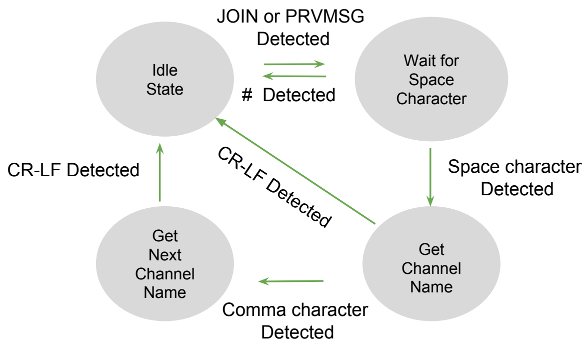

Figure 4. IRC Traffic Features Extraction Automaton.

the channel, but are not engaging in any back-and-forth communication. Our binary classification engine uses a logistic regression model to estimate the probability that a channel houses a botnet given the total number of JOIN commands observed on the channel compared to the total number of PRVMSG commands exchanged over the same channel. In this phase, we first take input training data to build a normal network behavior model offline (in our case, real-world IRC traffic channels total counts of JOIN vs. total counts of PRVMSG and whether the channel was found malicious or not). Flags are raised when any channel traffic deviates from the learned model. Figure 2 gives a high level functional representation of the architecture.

A. *Signature-based detection: Pattern Matching Algorithm*

A pattern matching problem can be defined as finding a pattern P of length M in a text T of length N where M<N. Pattern matching algorithms are classified depending on their direction of search (forward matching vs. backward matching) and the number of patterns they can detect within text T (single pattern detection vs. multiple pattern detection). Forward matching consists in moving P against T from left to right whereas backward matching does the opposite [13]. In our case, we used a multiple pattern detecting forward-matching algorithm. Figure 3 illustrates the architecture of our pattern matching module.

The preprocessing module maps incoming 8-bit characters into 5-bit characters to take care of case insensitivity; a comparator module decodes incoming ASCII characters and send them to the automaton. The automaton moves from one state to another according to previously received characters and the currently received character using a common prefix search method. A simple example will illustrate the idea: Suppose there is a file f which contains 4 words {party, paris, beef, before} we want to detect in a TCP packet payload. Our algorithm will check for a common prefix and then start building up patterns using a forward-matching approach as shown in Figure 4.

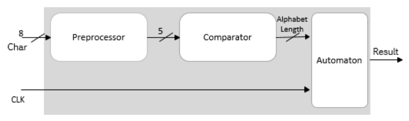

Figure 2. Pattern Matching Module.

This common prefix search method is very efficient because it does not build an automaton for each pattern we are detecting; this dramatically reduce the number of flip flops needed for implementation. As a result, our design can be ported to small FPGAs. The automaton understands ASCII characters and is case insensitive. This makes it ideal choice for malware pattern detection since they can contain numbers, letters, punctuation characters, etc.

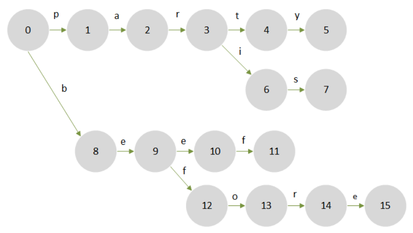

Figure 3. Common Prefix Search Method Example.

1) *IRC botnets detection:* To detect IRC botnets, our deep packet inspection (DPI) engine, using our pattern matching algorithm described above, checks TCP packet payloads for known malicious patterns, strange flag combinations, or virus attack signatures. Every time a malicious packet is detected, it is eliminated immediately. For IRC protocol-conforming packets which seem to be non-malicious, IRC traffic properties like the channels' names and their IP members, number of JOIN, PRVMSG, PING, and PONG exchanged on each channel and its IP members are extracted and sent to the memory (where they are read by our classification engine) and are updated every time a new IRC packet arrives. Figure 5 illustrates the process.

B. *Binary classification*

Differentiating a non-malicious traffic from a malicious one by looking at IRC messages exchanged over an IRC channel is a binary classification problem since there are only 2 outcomes to this scenario. We used logistic regression as our binary classifier to estimate the probability a particular IRC traffic is malicious.

In logistic regression, as explained in details by [14] and [15], there is a hypothesis function:

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}} = P(y=1 \mid x) \tag{1}$$

Where the hypothesis is the probability that the IRC traffic is malicious given the values of features in x. The cost function J(θ) is defined as:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}[-y^i \log(h_\theta(x^i)) - (1-y^i)\log(1-h_\theta(x^i))] \tag{2}$$

and the update rule for Newton's method is

$$\theta^{(t+1)} = \theta^{(t)} - H^{-1}\nabla\theta J \tag{3}$$

the gradient,

$$\nabla\theta J = \frac{1}{m}\sum_{i=1}^{m}[((h_\theta(x^i)) - (y^i))x^i] \tag{4}$$

and Hessian

$$H = \frac{1}{m}\sum_{i=1}^{m}[h_\theta(x^i)(1-h_\theta(x^i))x^i(x^i)^T] \tag{5}$$

Our task is to find the values of θ where the hypothesis function converges using equation 3, 4, and 5 . This problem involves performing multiple iterations updating the values of θ to find the global minimum [15]. To increase the efficiency of the implementation, the training part was done offline. When the final values of θ are found, they are used to update the hypothesis function to perform the prediction. Figure 6 illustrates the process

In the offline training phase, for each iteration i, input data x are taken in and the hypothesis function is computed. Then, we compute the gradient and the Hessian to get the updated values of θ. We then calculate the cost function to test the convergence of the hypothesis. Next, we calculate the decision boundary which gives us an approximation on where our threshold is going to be. After sufficient number of iterations , we test the new hypothesis function with known IRC traffic botnets features (total counts of JOIN vs. total counts of PRVMSG in our case) and check how well the model performs. Ideally our training set is a snapshot of typical IRC traffic over a certain channel. To increase the detection rate, we mix these data with recorded IRC traffic botnets data.

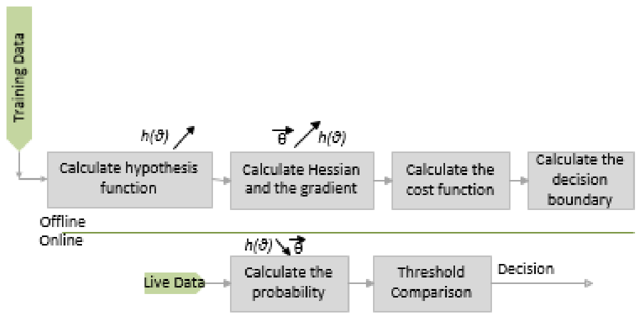
Training Data
h(θ)
θ h(θ)
Calculate hypothesis function
Calculate Hessian and the gradient
Calculate the cost function
Calculate the decision boundary
Offline
Online
h(θ) θ
Live Data
Calculate the probability
Threshold Comparison
Decision

Figure 5. Logistic Regression for IRC Traffic Analysis

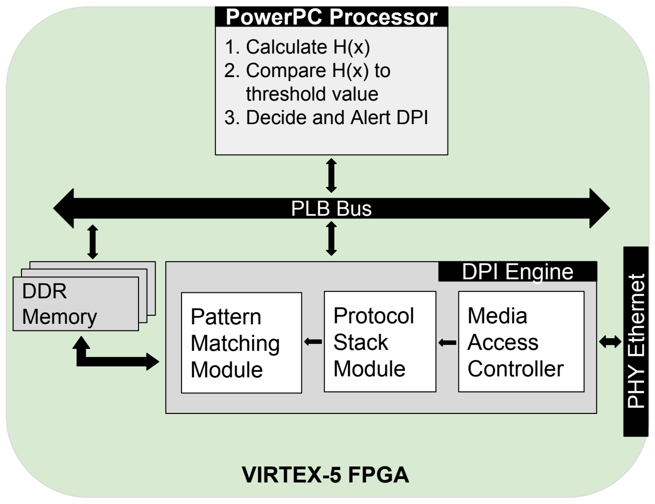

Figure 6. FPGA SoC Implementation: Hardware/Software Partitions.

IV. Implementation

We developed an FPGA-based SoC to act as an isolation layer between the network traffic and the target. We partitioned the signature-based detection portion of our design to be implemented in hardware and the anomaly-based detection portion in software. A Xilinx ML507 FPGA evaluation board featuring a VIRTEX 5 FPGA, 256MB DDR2 memory, Ethernet interface among others [16] was used to implement the architecture. The system was synthesized using Xilinx design tools XPS and EDK (Version 14.5). Figure 7 illustrates the architecture's implementation.

A. *SoC Partitions*

In the hardware portion, we developed a DPI engine to perform the signature-based detection. As shown in the figure 7, DPI engine implements a network layer (ARP, IP, ICMP) and transport layer (TCP) of the OSI model on top of the Xilinx xps_ethernet_lite IP core which only implements the two bottom layers. [17]. The DPI engine monitors, in real-time, the network flow by reading every TCP packet header and payload and looks for suspicious IRC packets. The module checks for well-known patterns of attacks and

Table I
VIRTEX 5 RESOURCE UTILIZATION

Resources	**Utilization Percentile**
Registers	11864 out of 44800 (26%)
LUTs	19276 out of 44800 (43%)
I/O Pins	144 out of 640 (22%)

Table II
SIGNATURE-BASED DETECTION PERFORMANCE

Attack Type	**Description**	**Detection Results**
Virus signature	EICAR.txt virus signature	Detected
Land attack	source IP = destination IP	Detected
TCP violation	source IP address equal to 127.0.0.1 ; SYN and FIN flags both set	Detected

eliminates every packet with these signatures. In addition, it records important information about IRC traffic such as the channel name, IP members, number of exchanged IRC messages, etc. Using a multi-port memory (MPMC) controller module, the DPI engine writes TCP/IRC traffic information of interests in the memory where they are read by the binary classifier. Table I shows FPGA resources utilization.

Notice that the amount of resource needed by our architecture is negligible on the Virtex 5. New families of FPGAs like Zynq can offer even a much smaller design, thus the cost effectiveness of our solution. For the software portion, we wrote a script to train our model offline and used Xilinx SDK to write a C module that extracts our classification engine input data from the memory and implements our classification algorithm.

B. Test Results

1) Signature-based detection: To test the performance of our system, we emulated a network connection between a network simulator and the FPGA. For the network simulator, we used both *Nmap* and *Netcat* tools. *Nmap* provides a flexible way to customize packets payloads and send them at any rate you want (network load customization) [18]. This is important since we want to check whether our system is able to capture different types of attacks (DoS attack, land attack, virus packets, etc.) in real time. We used *Netcat* to establish a TCP connection between the machine running Nmap and the Xilinx ML507 evaluation board hosting BotPGA. Using *Nmap*, we then sent the following packets: EICAR (European Institute for Computer Anti-Virus Research) virus packets [19], packets with TCP protocol violation (loopback address packets with SYN and FIN flags set for example [20]), and land attack packets. As Table II indicates, our DPI engine detected all three packets categories.

2) Anomaly-based detection: Recall that our classifier first takes in training data to build the knowledge of a traffic profile. Then, it notifies us every time it detects any IRC

Table III
BINARY CLASSIFIER PREDICTION ON TRAFFIC RECORD GENERATED BETWEEN IRC CLIENT AND IRC SERVER (TRAFFIC WAS CAPTURED BY 3RD PARTY TOOL "OURMON" AT PORTLAND STATE UNIVERSITY. THE TOOL AND DATA ARE DISCUSSED IN [3])

Channels	**JOINs Count**	**PRVMSGs Count**	**Classifier Prediction**	**Actual Traffic Nature**
# Rubyonrails	325	2165	Non-Malicious (P = 0.996)	Non-Malicious
# f	153248	28531	Malicious (P=1)	Malicious
# .i-exp	0	1	Non-Malicious (P = 0.339)	Non-Malicious

traffic that is outside the norm. The challenge here is to define normal behavior for IRC traffic. Keep in mind that botmasters do their best to hide their real intentions when communicating with their IRC bots. A typical testing scenario involves using our classifier with real-world IRC traffic mixed with botnet traffic. We used Portland State University's network IRC traffic data captured by a tool called *Ourmon* as our training data. The data and a detailed description of *Ourmon* tool can be found in [3]. Figure 10 illustrates the convergence of the cost function after the training phase.

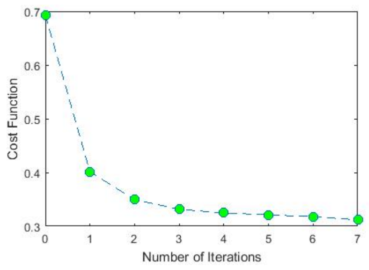

Figure 7. Cost Function Plot.

From Figure 9, you can see that Newton's Method converged by around 7 iterations. The cost function changes by 0.0058 between 7th and 8th iteration. Ideally, you want this number to be 0 for a perfect convergence. But since the 'level' of convergence the cost function is able to achieve depends on the nature of the training data set, it is possible for the cost function to not converge to 0 . After testing the cost function convergence, we proceeded to test the model with separate Portland State University's network IRC traffic data. This data was also captured by *Ourmon* tool and can be found in [3]. As shown in Table III, our classifier engine was able to correctly identify two channels that were found in [3] to house a botnet; but it also suffered from a false alarm raised by a normal activity (false positives). Obviously, our testing input vector is very small and can not be used to predict how the model will behave when you increase the size of the input vector. It is very difficult to find IRC traffic data that features the number of IRC commands sent over IRC channels. A lot of network traffic data we were able to find on the web didn't have the features we were looking for. So, part of our current ongoing efforts is to increase the size of input vector and expand the number of features we can extract from IRC traffic and integrate them into our training data set.

V. Conclusion

To conclude, we presented an FPGA-based hardware-software architecture to detect IRC botnets and malicious TCP/IRC non-encrypted traffic. To achieve higher detection rates, our architecture uses signature-based detection and anomaly-based detection at the same time. For signature-based detection, it uses a forward pattern-matching algorithm to detect known malicious signatures, strange flag combinations, and other known virus patterns achieving a perfect detection rate and employing small FPGA resources. For anomaly-based detection, it implements a binary classification model that learns normal behavior of IRC traffic and raises a flag every time it sees an IRC traffic outside the norm. Our binary classifier performs well on heavy IRC traces, but it still needs improvement. Our architecture presents a distinct advantage because it introduces a reconfigurable hardware isolation layer between the network link and the target. It eliminates botnets greatest strength which is to seize and control the target machine including onboard antivirus applications. Our ongoing efforts focus primarily on improving the detection rate of our binary classification model. We are looking to expand the number of features we can extract from IRC traffic and integrate them into our training data set since this number correlates directly with detection rate. We are also expanding our architecture to accommodate HTTP and P2P protocols to capture the entire current ecosystem of botnet attacks.

References

[1] G. Hogben, D. Plohmann, E. Gerhards-Padilla, and F. Leder, "Botnets: Measurement, detection, disinfection and defence." Tech. Rep. ENISA, March 07 2011.

[2] A. Stevenson, "Botnets infecting 18 systems per second, warns fbi," 2014, [Online; accessed 2-March-2015]. [Online]. Available: http://www.v3.co.uk/v3-uk/news/2355596/botnets-infecting-18-systems-per-second-warns-fbi

[3] B. Schiller, Craig and Jim, *Botnets: The Killer Web Applications*. Syngress Publishing, 2007.

[4] G. Gu, J. Zhang, and W. Lee, "BotSniffer: Detecting botnet command and control channels in network traffic," in *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08)*, February 2008.

[5] W. Lu, M. Tavallaee, and A. A. Ghorbani, "Automatic discovery of botnet communities on large-scale communication networks," in *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, ser. ASIACCS '09, 2009, pp. 1–10.

[6] L. Gu-Shin, C. Chia-Mei, T. Ray-Yu, L. Chi-Sung, and F. Christos.:, "Botnet detection by abnormal irc traffic," *In proceedings of the Fourth Joint Workshop on Information Security*, 2009.

[7] H. Choi, H. Lee, H. Lee, and H. Kim, "Botnet detection by monitoring group activities in dns traffic," in *Computer and Information Technology, 2007. CIT 2007. 7th IEEE International Conference on*, Oct 2007, pp. 715–720.

[8] W. Lee, C. Wang, and D. Dagon, *Botnet Detection: Countering the Largest Security Threat (Advances in Information Security)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.

[9] A. Karasaridis, B. Rexroad, and D. Hoeflin, "Wide-scale botnet detection and characterization," in *Proceedings of the First Conference on First Workshop on Hot Topics in Understanding Botnets*, ser. HotBots'07, 2007, pp. 7–7.

[10] Cisco, "Cisco ips 4200 series," 2012. [Online]. Available: http://www.cisco.com/c/en/us/products/security/ips-4200-series-sensors/index.html

[11] ProventiaWorks, "Ibm proventia network intrusion prevention system (ips) gx5108," 2012. [Online]. Available: http://www.proventiaworks.com/GX5108.asp

[12] C. Kalt, "Internet relay chat: Client protocol," April, 2000. [Online]. Available: https://tools.ietf.org/html/rfc2810

[13] C. Bobda and T. Lehmann, "Efficient building of word recognizer in fpgas for term-document matrices construction," in *Field-Programmable Logic and Applications: The Roadmap to Reconfigurable Computing*, ser. Lecture Notes in Computer Science, R. Hartenstein and H. Grnbacher, Eds., 2000, vol. 1896.

[14] D. J. Hand, P. Smyth, and H. Mannila, *Principles of Data Mining*. Cambridge, MA, USA: MIT Press, 2001.

[15] A. Ng, "Machine learning : Openclassroom," 2010-2012. [Online]. Available: http://openclassroom.stanford.edu

[16] Xilinx, "Ml507 evaluation platform," 2012. [Online]. Available: http://www.xilinx.com/products/boards/ml507/docs.htm

[17] ——, "Xps ethernet lite ip core documentation," 2015. [Online]. Available: http://www.xilinx.com/products/intellectual-property/xps_ethernetlite.html

[18] G. Lyon, "Nmap - free security scanner for network exploration," 2014. [Online]. Available: http://nmap.org/

[19] EICAR, "Eicar intended use," 2012. [Online]. Available: http://www.eicar.org/86-0-Intended-use.html

[20] A. Das, D. Nguyen, J. Zambreno, G. Memik, and A. Choudhary, "An fpga-based network intrusion detection architecture," *Information Forensics and Security, IEEE Transactions on*, vol. 3, no. 1, pp. 118–132, March 2008.